# AI, Deep Learning Fundamentals and Applications

Dr. Ahmad Aljaafreh,
Professor, Intelligent Systems
Tafila Technical University

# OUTLINE

## Artificial Intelligence (AI)

- A brief introduction

## Deep Learning

- Introduction, what and why
- Applications
- Deep learning success

## Neural Networks

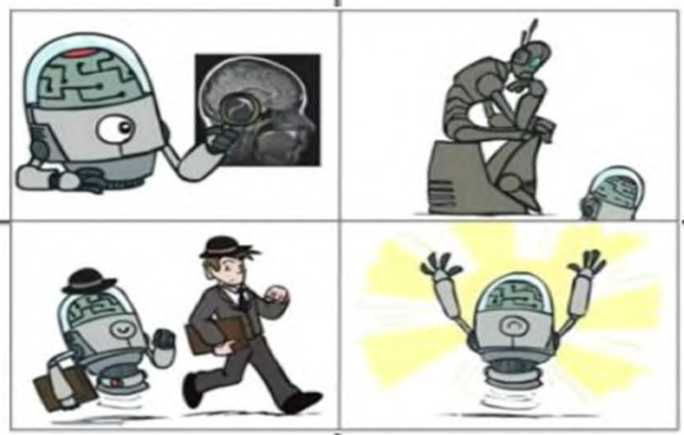
- Perceptron
- Neural network models
- CNN, RNN, Attention
- Transformers

# Artificial Intelligence (AI)

A brief introduction

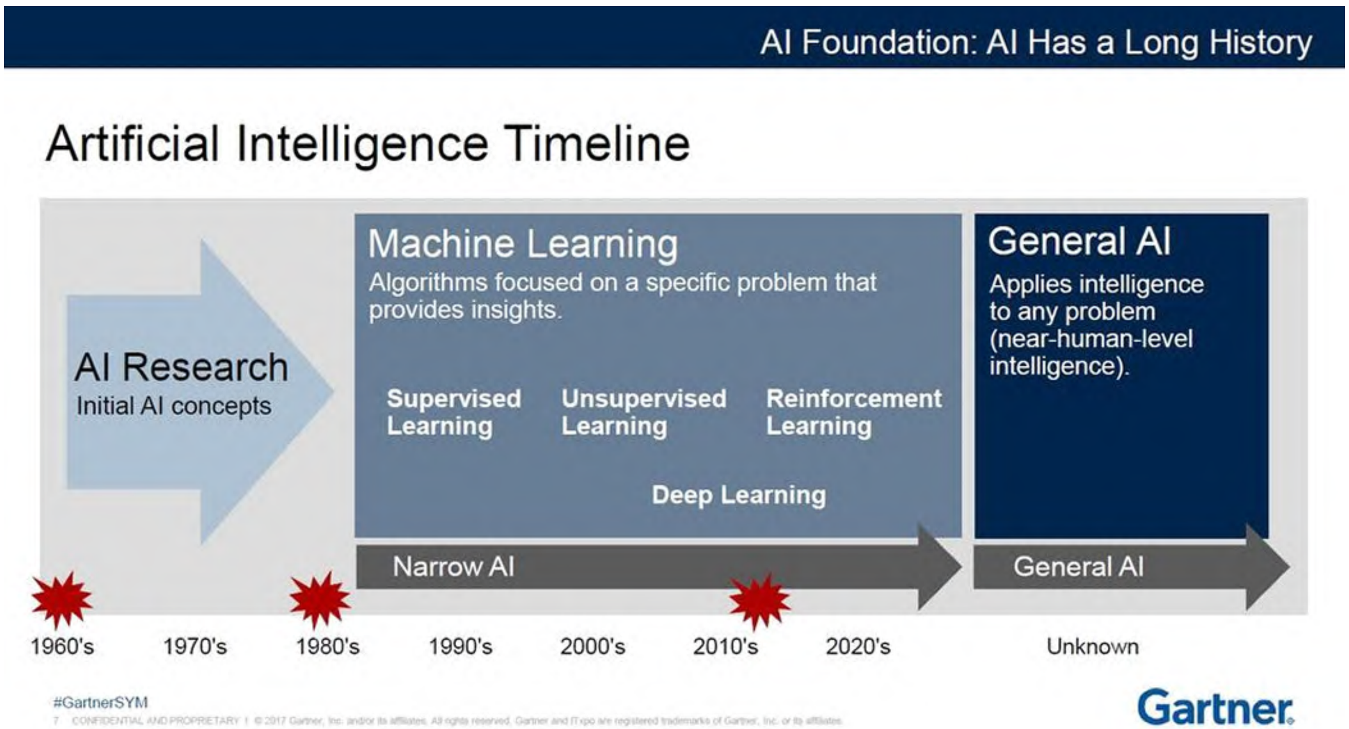
# WHAT IS A.I.?

| | |
|---|---|
| Think Humanly | Think Rationally |
| Act Humanly | Act Rationally |



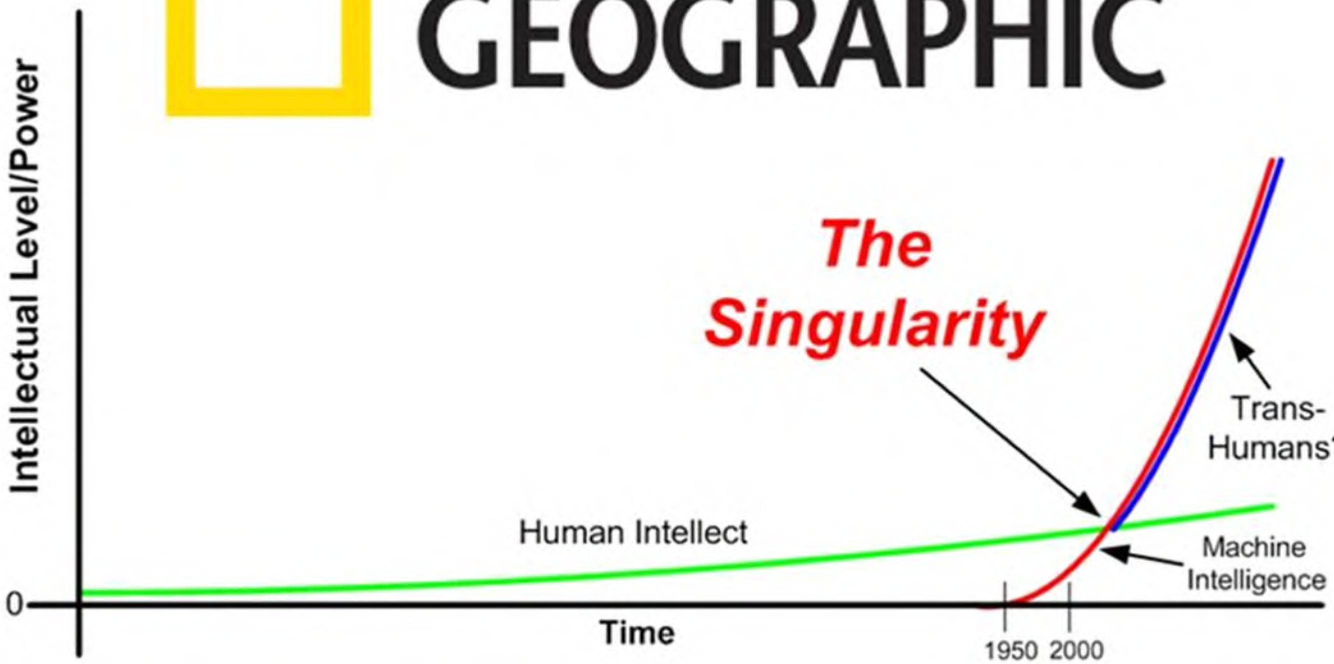

"AI IS THE NEW ELECTRICITY."
ANDREW NG

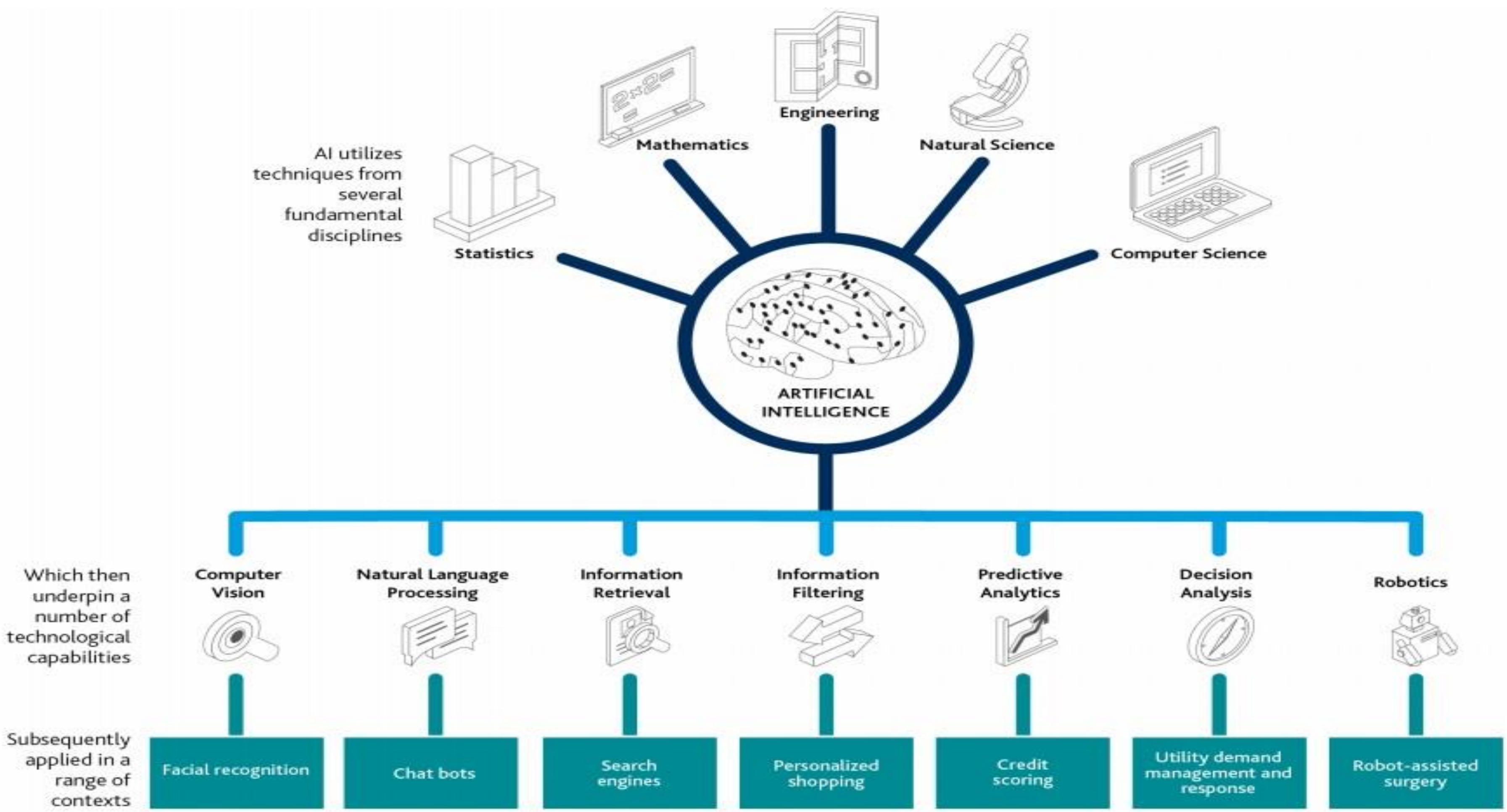
# History

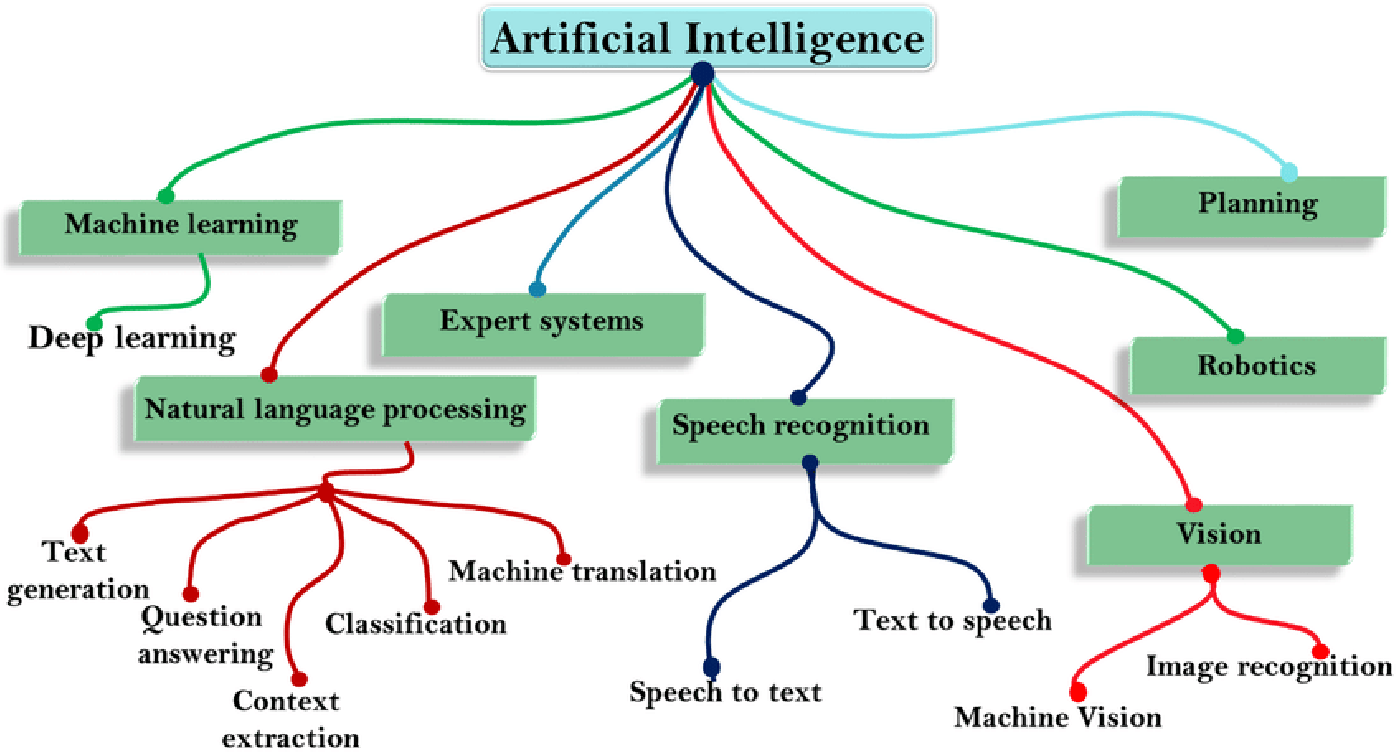


AI Foundation: AI Has a Long History
Artificial Intelligence Timeline
AI Research
Initial AI concepts
Machine Learning
Algorithms focused on a specific problem that provides insights.
Supervised Learning
Unsupervised Learning
Reinforcement Learning
Deep Learning
General AI
Applies intelligence to any problem (near-human-level intelligence).
Narrow AI
General AI
1960's
1970's
1980's
1990's
2000's
2010's
2020's
Unknown
#GartnerSYM
Gartner.

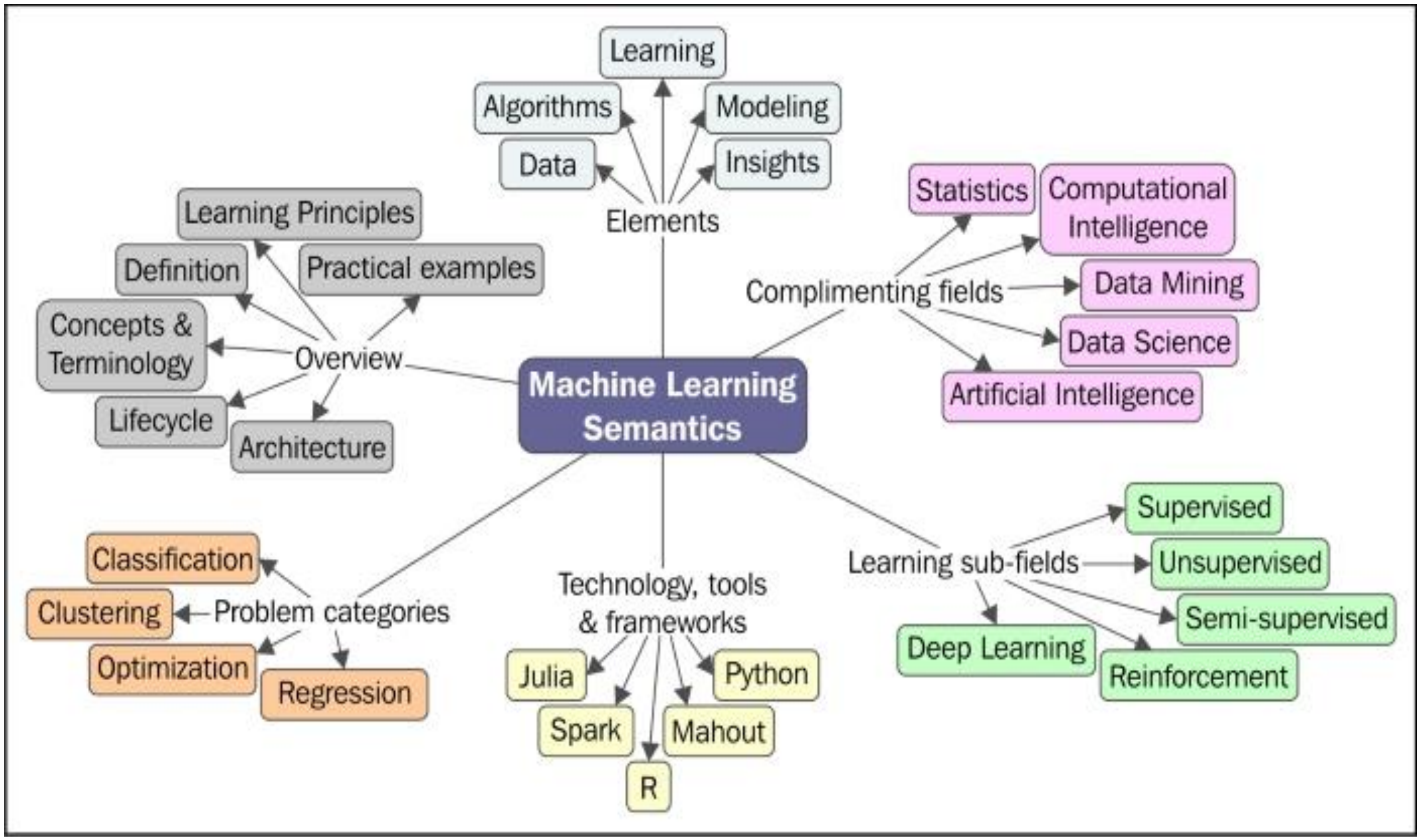
NATIONAL GEOGRAPHIC



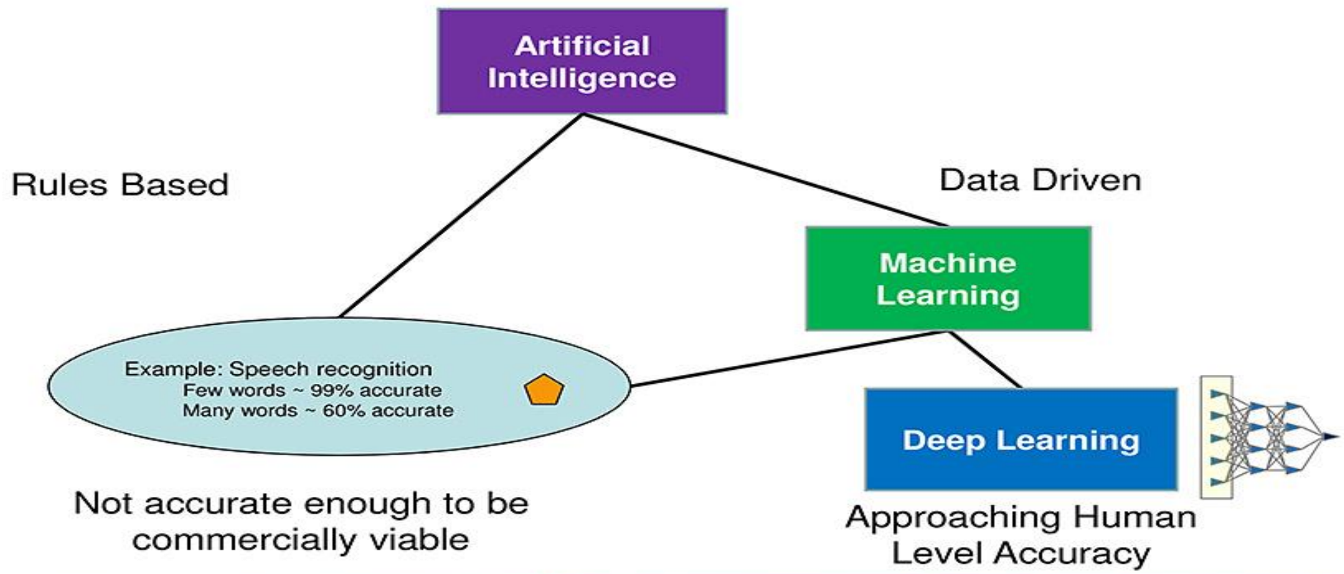

AI utilizes techniques from several fundamental disciplines
Statistics
Mathematics
Engineering
Natural Science
Computer Science
ARTIFICIAL INTELLIGENCE
Which then underpin a number of technological capabilities
Computer Vision
Natural Language Processing
Information Retrieval
Information Filtering
Predictive Analytics
Decision Analysis
Robotics
Subsequently applied in a range of contexts
Facial recognition
Chat bots
Search engines
Personalized shopping
Credit scoring
Utility demand management and response
Robot-assisted surgery

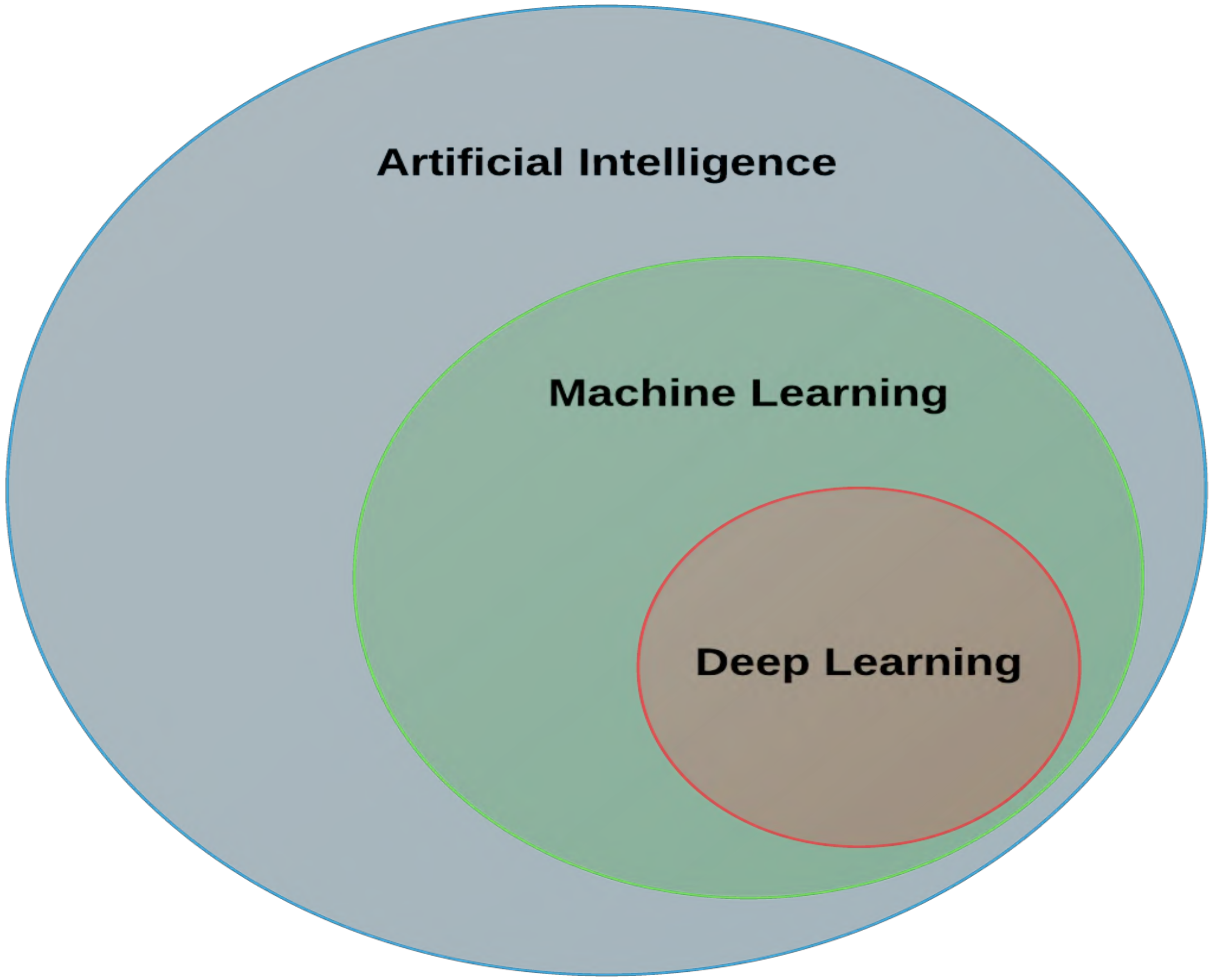


Artificial Intelligence
Machine learning
Deep learning
Expert systems
Natural language processing
Speech recognition
Planning
Robotics
Vision
Text generation
Question answering
Context extraction
Classification
Machine translation
Speech to text
Text to speech
Machine Vision
Image recognition


Machine Learning Semantics
Elements
Learning
Algorithms
Modeling
Data
Insights
Complimenting fields
Statistics
Computational Intelligence
Data Mining
Data Science
Artificial Intelligence
Overview
Learning Principles
Definition
Practical examples
Concepts & Terminology
Lifecycle
Architecture
Problem categories
Classification
Clustering
Optimization
Regression
Technology, tools & frameworks
Julia
Python
Spark
Mahout
R
Learning sub-fields
Supervised
Unsupervised
Semi-supervised
Deep Learning
Reinforcement

# Deep Learning Has Changed AI




Artificial Intelligence
Machine Learning
Deep Learning


# Deep Learning is A Revolution in Artificial Intelligence

front-page article at the New York *Times*

MIT Technology Review

# 10 BREAKTHROUGH TECHNOLOGIES 2013

Intr

**Deep Learning**

With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart. →

**Temporary Social Media**

Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous. →

**Prenatal DNA Sequencing**

Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child? →

**Add Ma**

Ske prin wor mar the tech jet

**Memory Implants**

A maverick neuroscientist believes he has deciphered the code

**Smart Watches**

**Ultra-Efficient Solar Power**

Doubling the efficiency of a solar

**Big Pho**

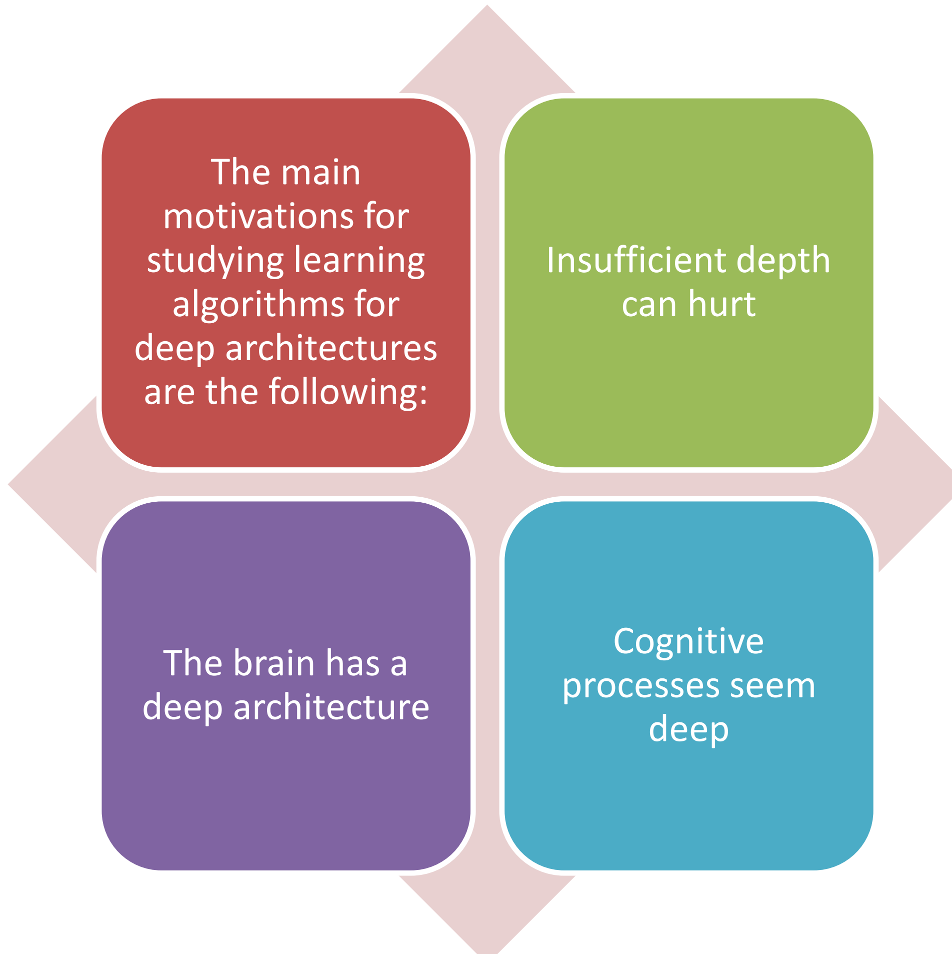
Coll ana fron

# Deep Learning

Deep Learning is a new area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals

Deep Learning is about learning multiple levels of representation and abstraction that help to make sense of data such as images, sound, and text.

# Motivations for Deep Architectures



# Why Now ?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

⋮

1986 Backpropagation
- Multi-Layer Perceptron

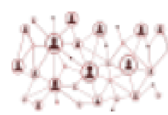
1995 Deep Convolutional NN
- Digit Recognition

⋮

Neural Networks date back decades, so why the resurgence?

## 1. Big Data
- Larger Datasets
- Easier Collection & Storage

IMAGENET

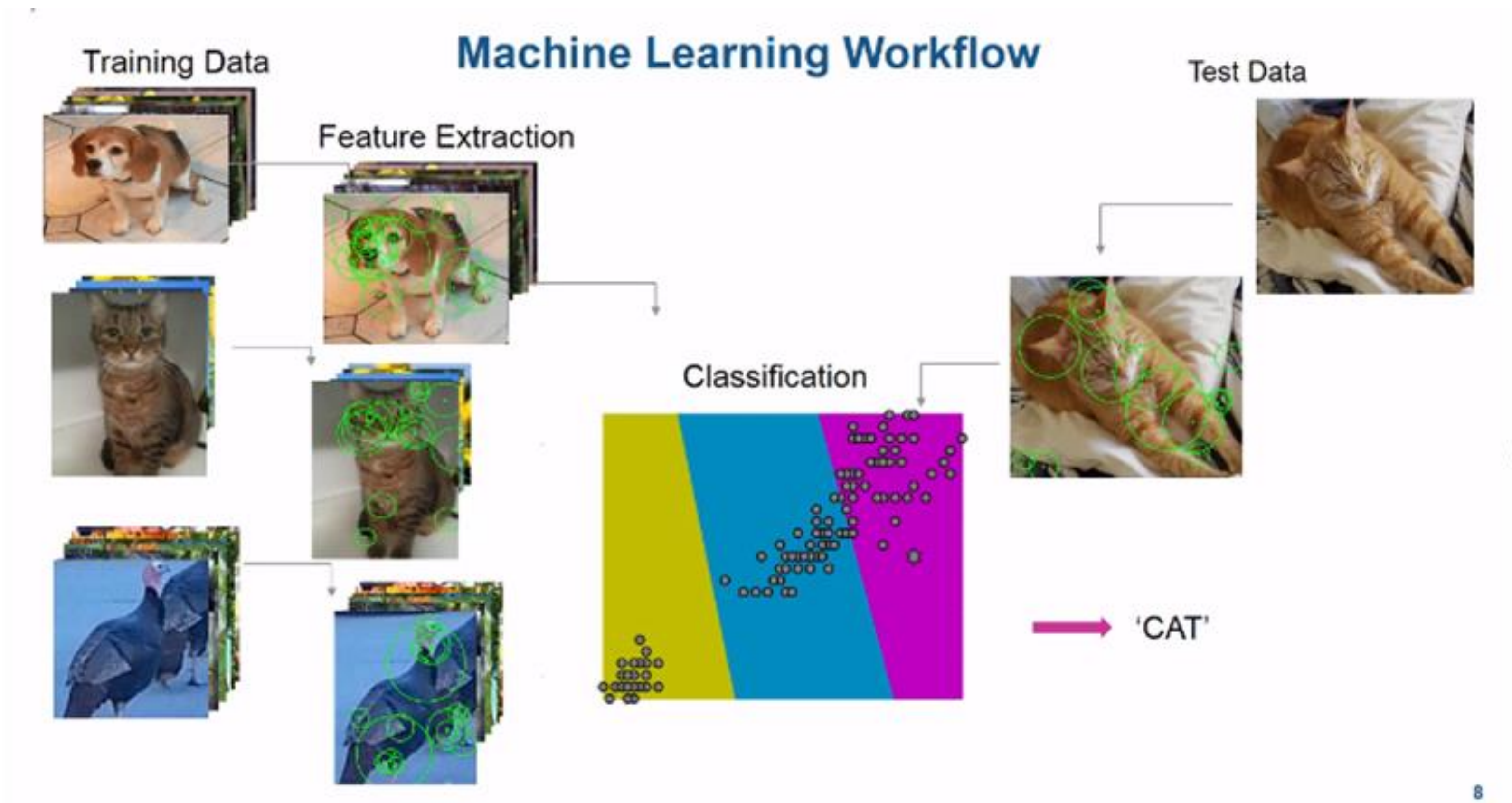


## 2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

## 3. Software
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

# Until Now…



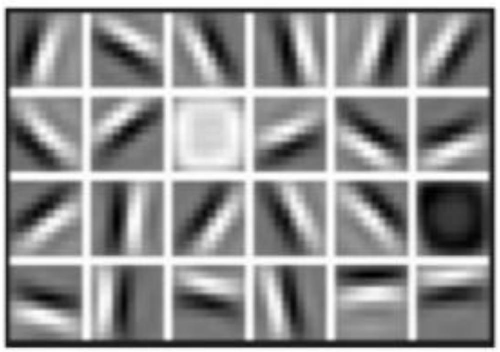
Machine Learning Workflow
Training Data
Feature Extraction
Classification
Test Data
'CAT'
8

# Deep Learning = Learning Hierarchical Representations

Hand engineered features are time consuming, brittle and not scalable in practice

Can we learn the **underlying features** directly from data?

**Low Level Features**



Lines & Edges

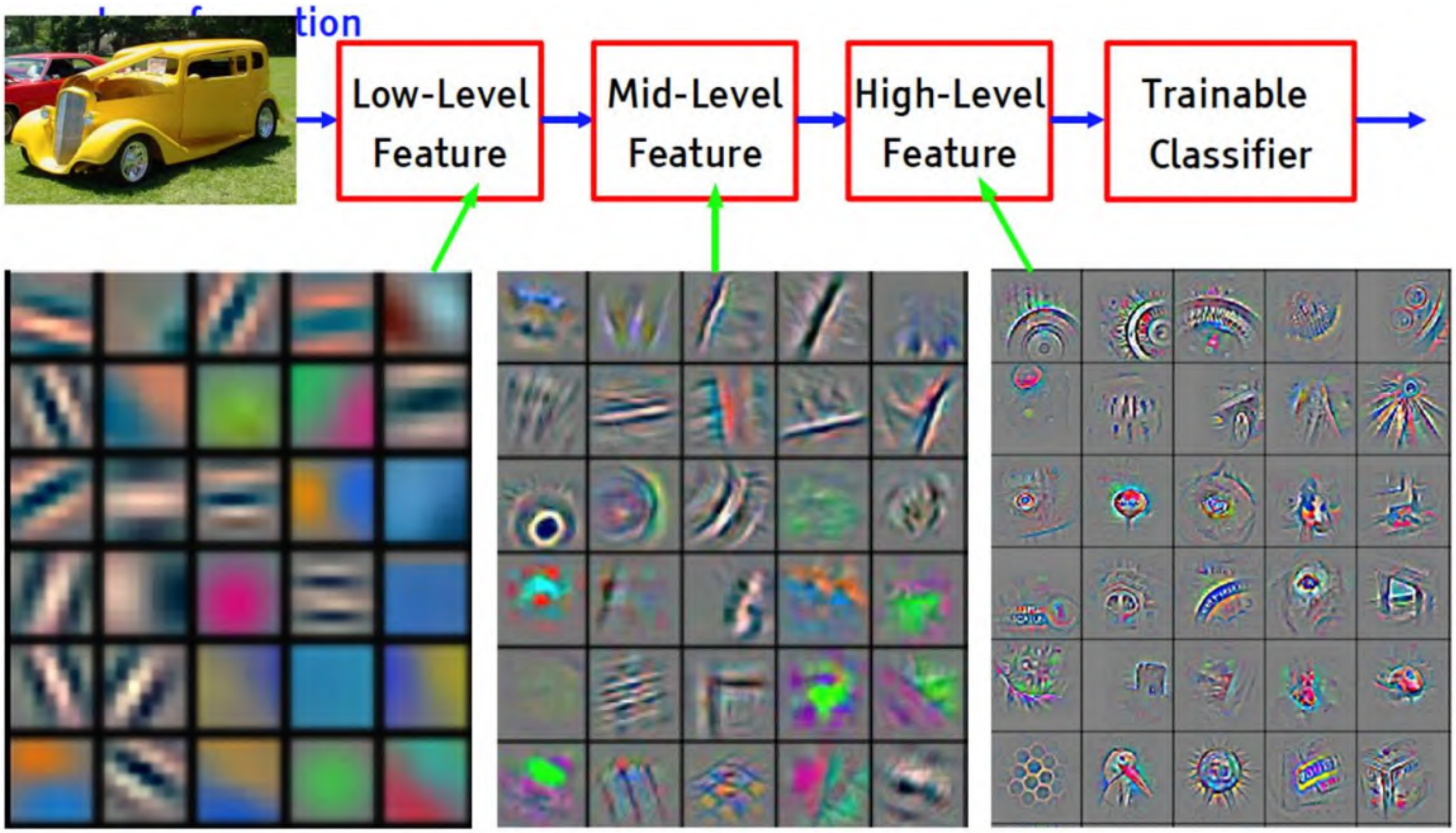
**Mid Level Features**

Eyes & Nose & Ears

**High Level Features**

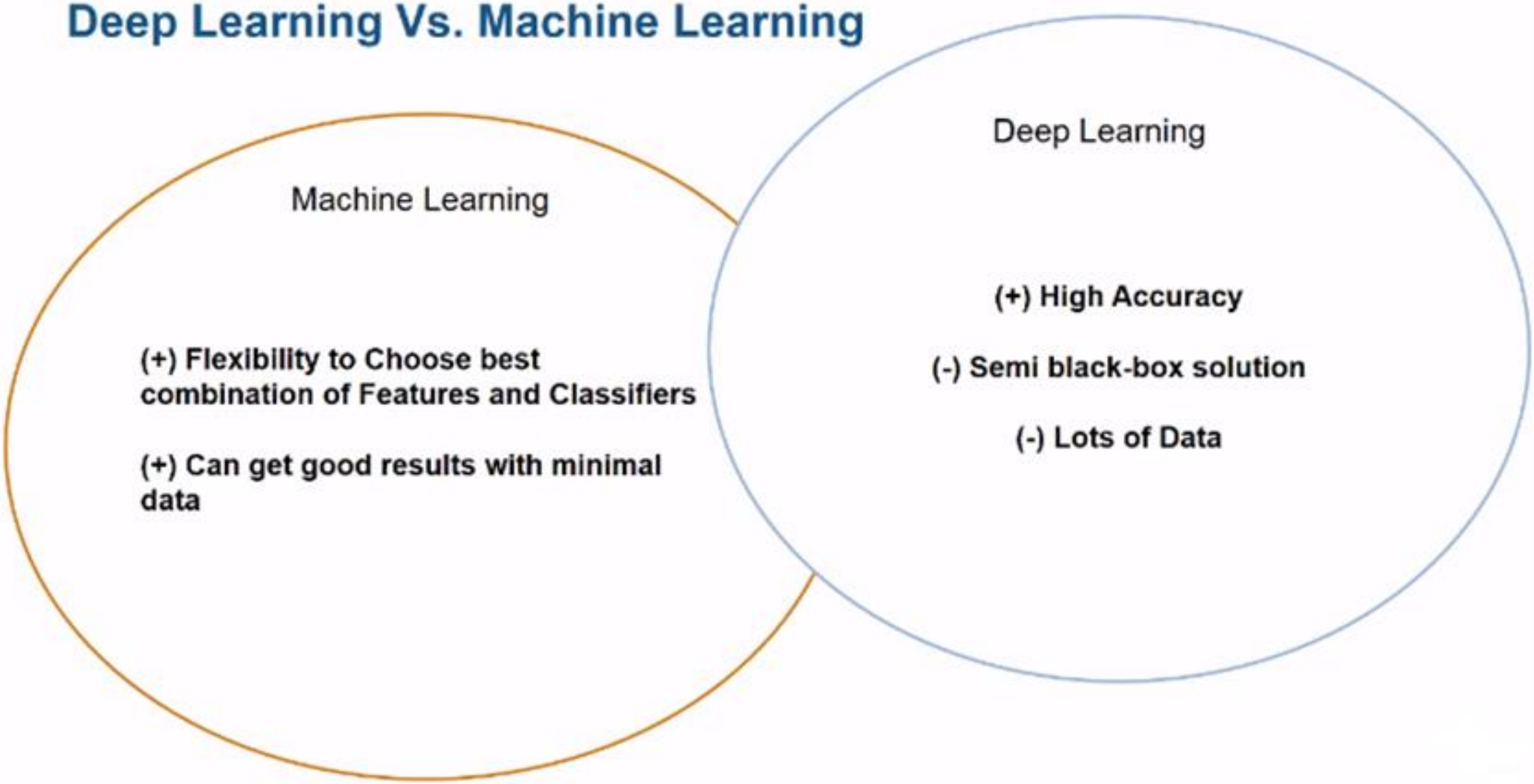
Facial Structure

# Deep Learning = Learning Hierarchical Representations

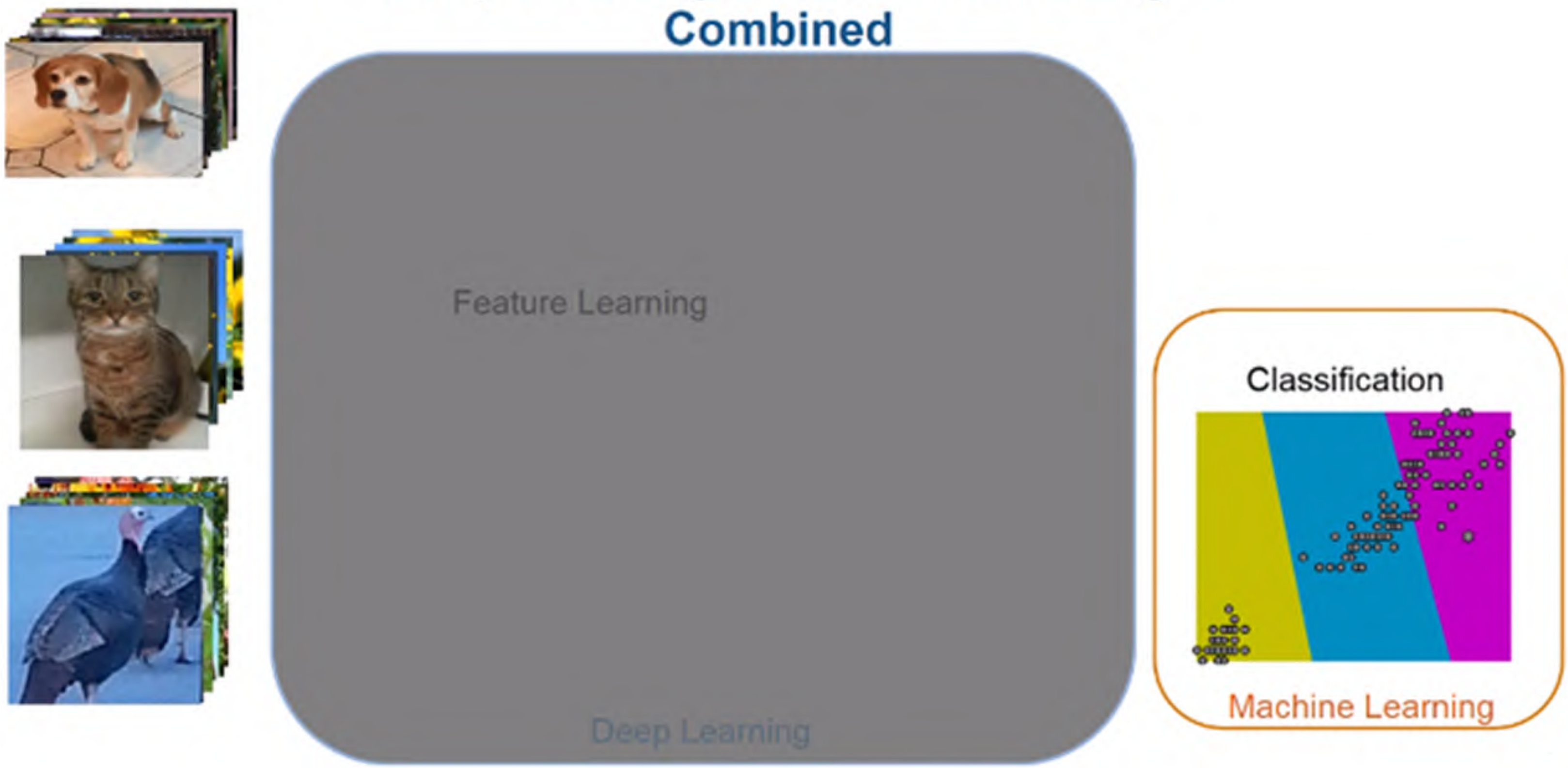



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

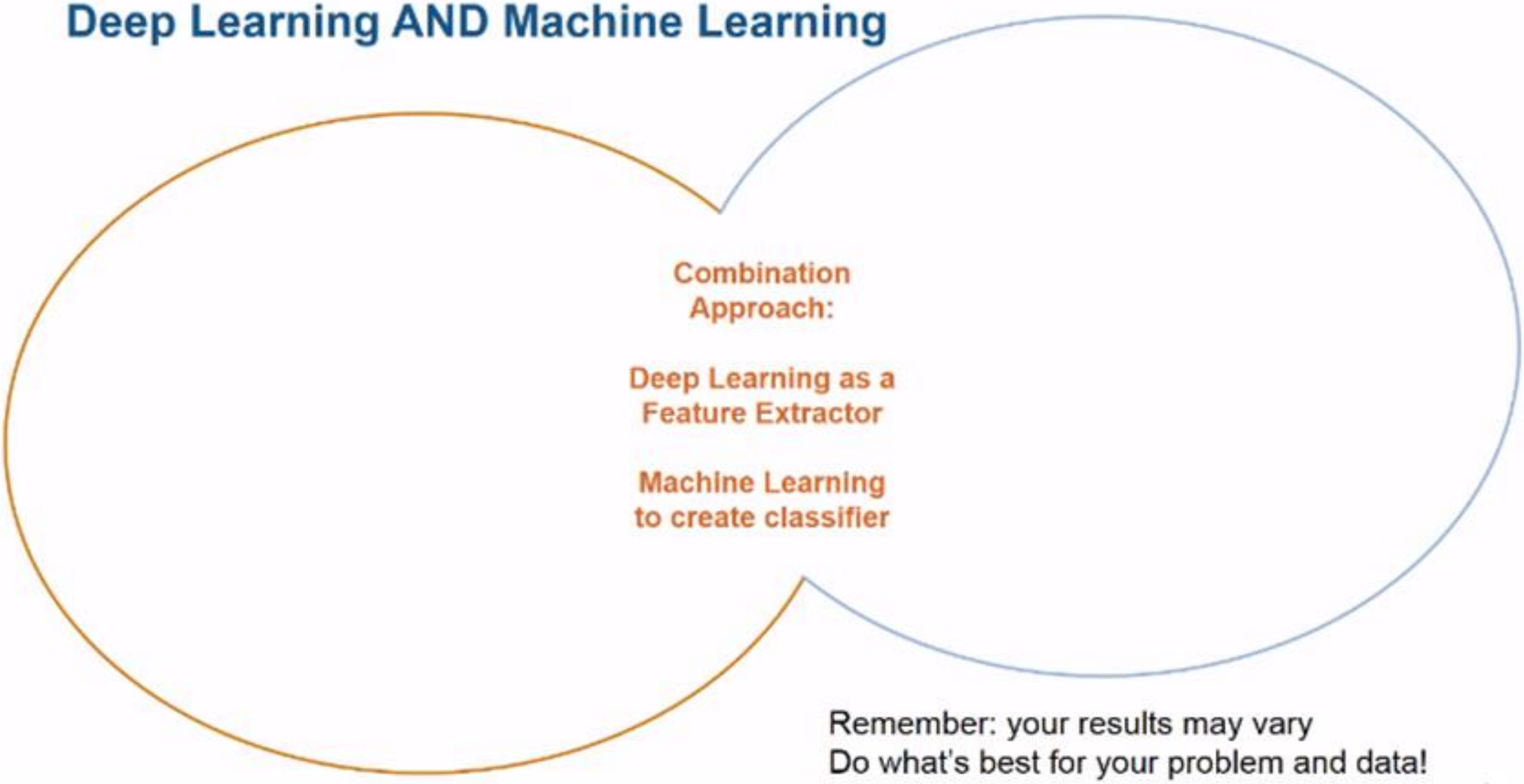
# Deep Learning Vs. Machine Learning



## Deep Learning + Machine Learning Combined



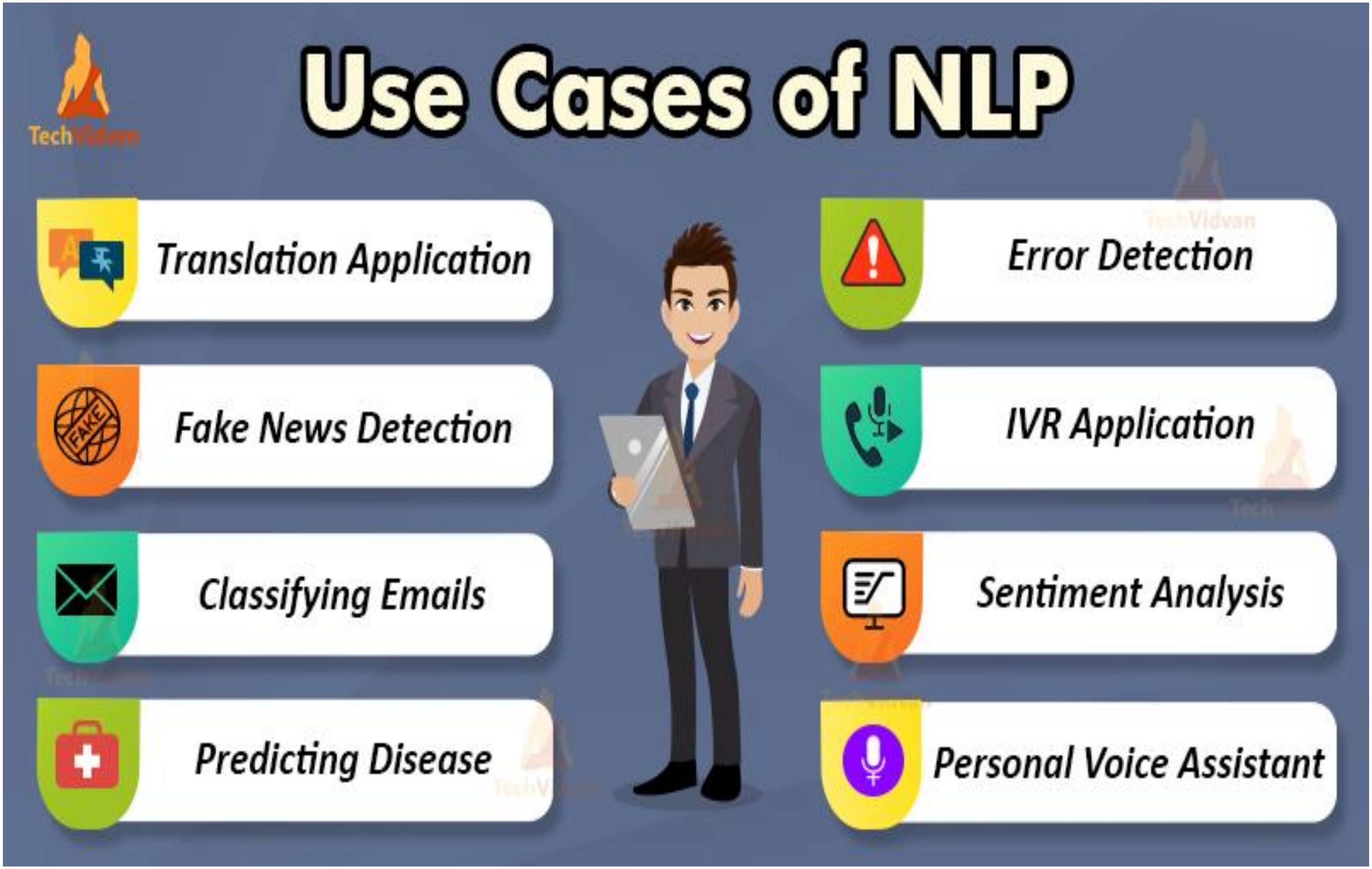
## Deep Learning AND Machine Learning



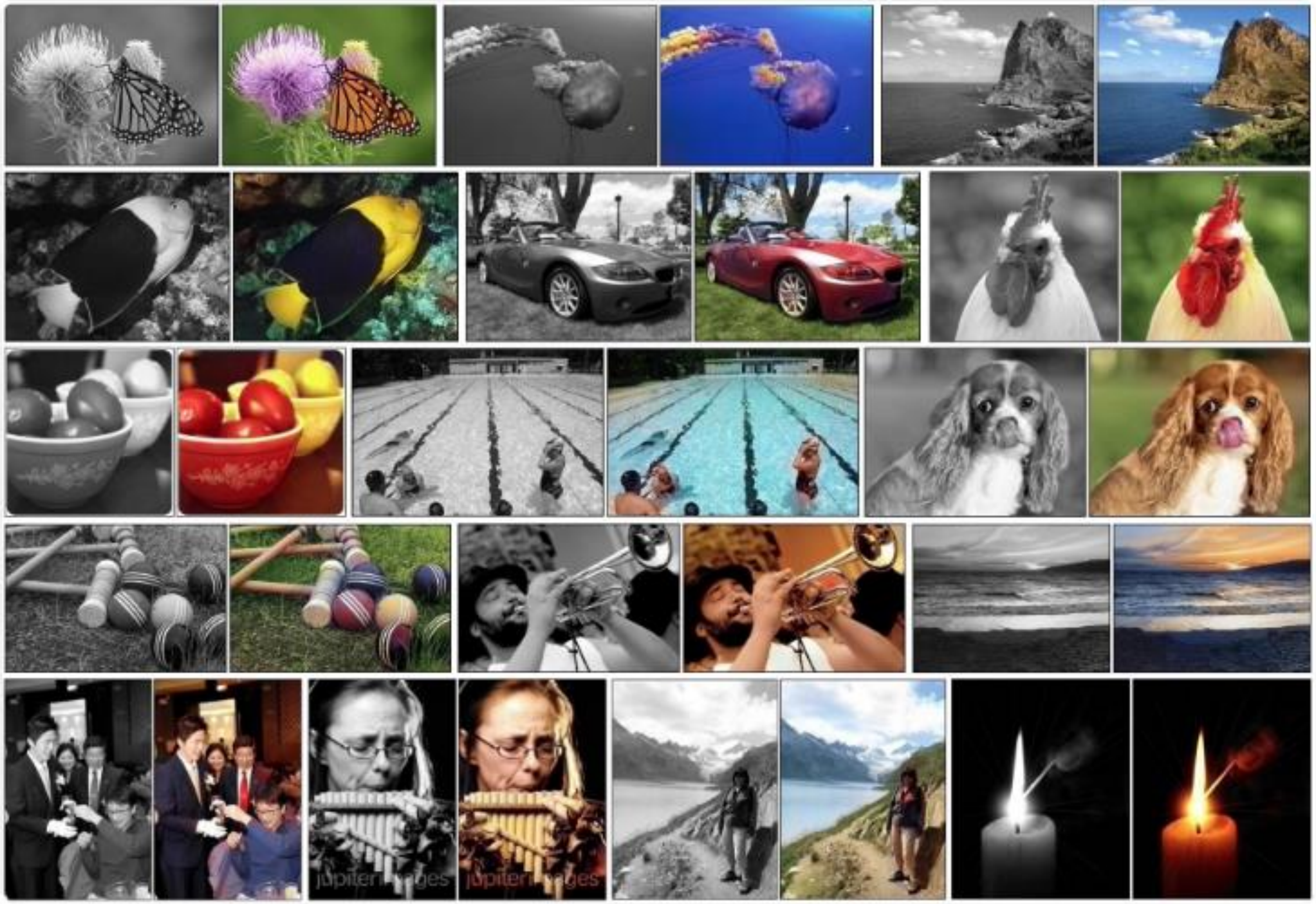

Remember: your results may vary
Do what's best for your problem and data!

# Applications of Deep Learning

http://machinelearningmastery.com/inspirational-applications-deep-learning/


TechVidvan
Use Cases of NLP
Translation Application
Fake News Detection
Classifying Emails
Predicting Disease
Error Detection
IVR Application
Sentiment Analysis
Personal Voice Assistant

# Automatic Colorization of Black and White Images



Colorful Image Colorization, Zhang et. al. 2016 https://arxiv.org/pdf/1603.08511.pdf

# Automatically Adding Sounds To Silent Movies

The system is trained using 1000 examples of video with sound of a drum stick striking different surfaces and creating different sounds. A deep learning model associates the video frames with a database of pre-rerecorded sounds in order to select a sound to play that best matches what is happening in the scene.

Demo : https://www.youtube.com/watch?v=0FW99AQmMc8

**Visually Indicated Sounds, Owens et. al. 2015, https://arxiv.org/abs/1512.08512**

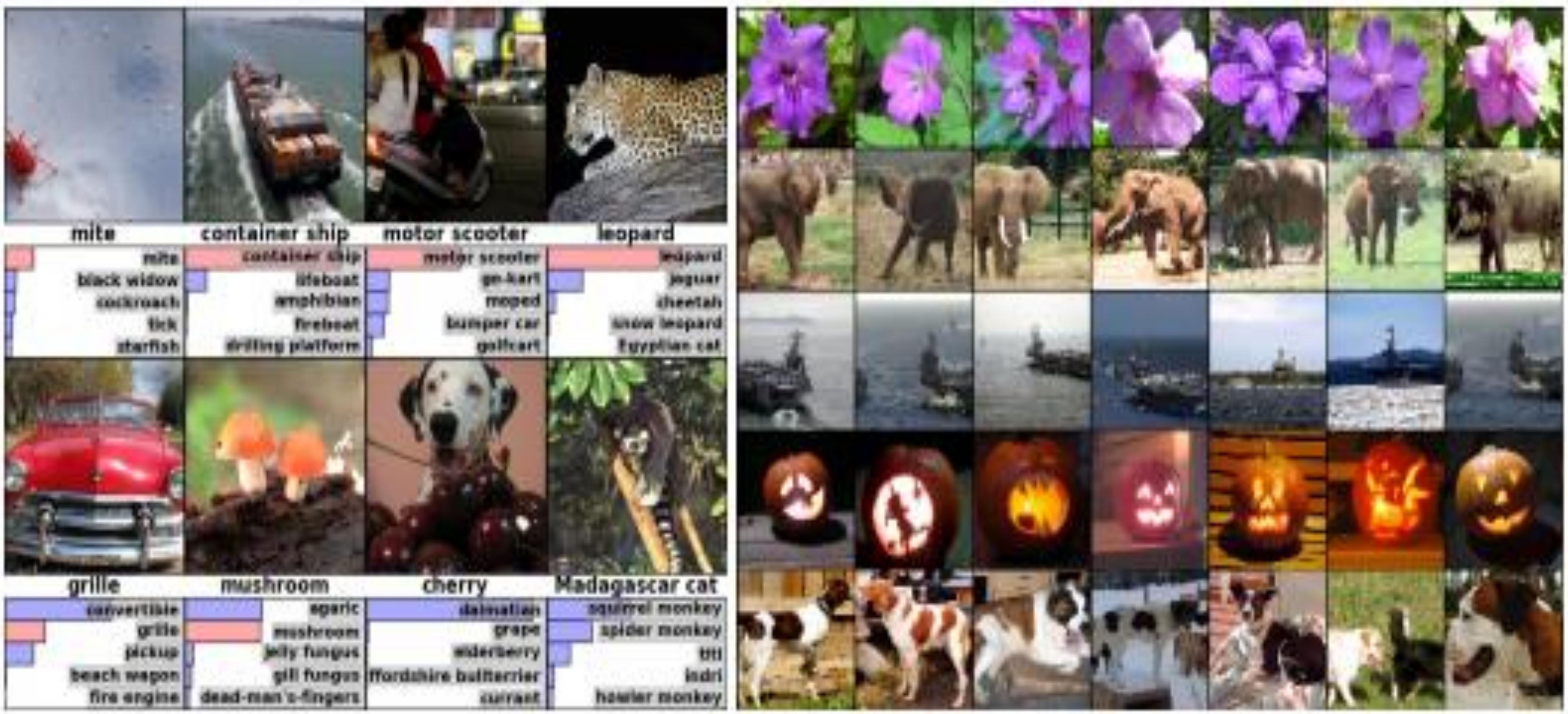
# Automatic Machine Translation

Given words, phrase or sentence in one language, automatically translate it into another language. Automatic machine translation has been around for a long time, but deep learning is achieving top results. in two specific areas:

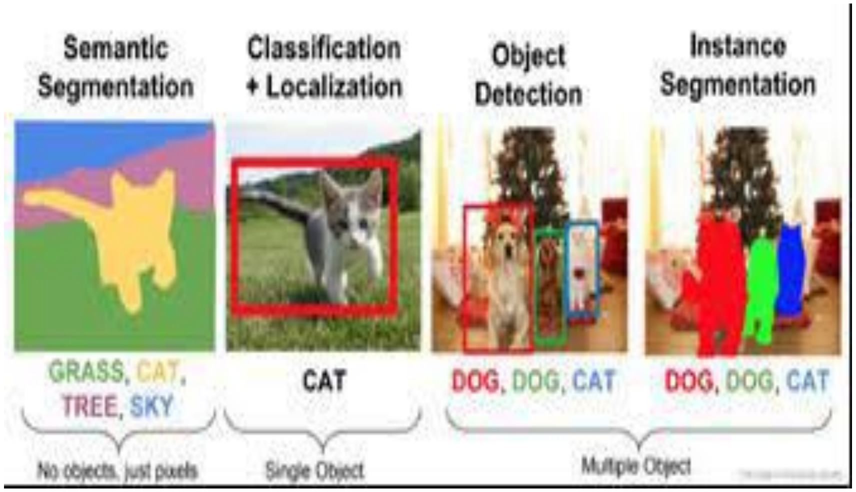
Automatic Translation of Text.
Automatic Translation of Images.

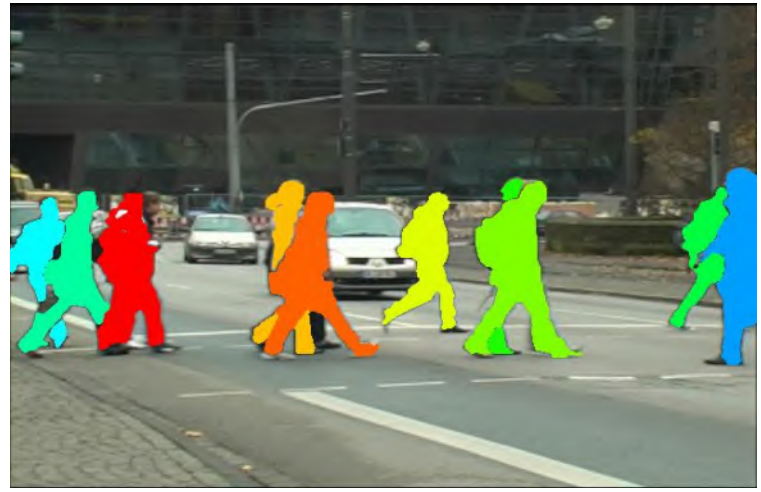
Deep Learning Network in Machine Translation, Zhang et.al. 2015,

# Object Classification and Detection in Photographs




Deep Neural Networks for Object Detection, Szegedy et.al. 2013


Semantic Segmentation
Classification + Localization
Object Detection
Instance Segmentation
GRASS, CAT, TREE, SKY
CAT
DOG, DOG, CAT
DOG, DOG, CAT
No objects, just pixels
Single Object
Multiple Object

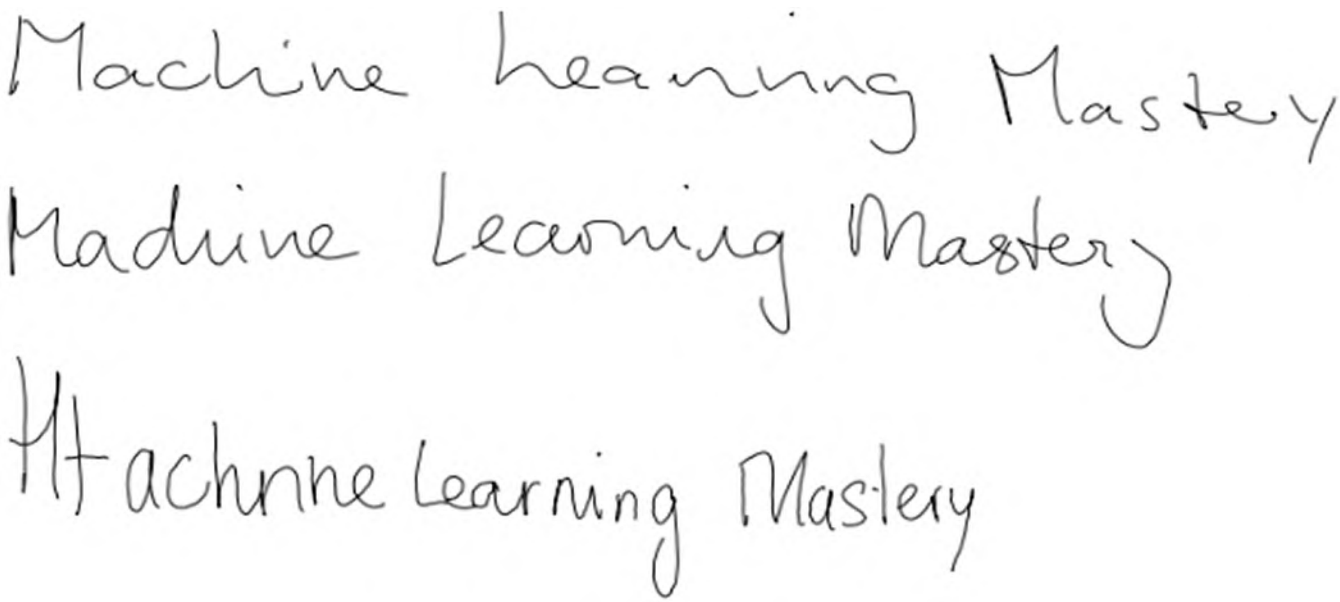



Multiple Object Tracking

# Automatic Handwriting Generation

Different styles can be learned and then mimicked

Demo : http .html




Generative sequences with recurring neural network, Graves 2014
https://arxiv.org/pdf/1308.0850v5.pdf

# Automatic Text Generation

➢This is an interesting task, where a corpus of text is learned and from this model new text is generated, word-by-word or character-by-character. The model is capable of learning how to spell, punctuate, form sentences and even capture the style of the text in the corpus.

➢Large recurrent neural networks are used to learn the relationship between items in the sequences of input strings and then generate text.

```
PANDARUS:
Alas, I think he shall be come approached and the day
When little srain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
```

Automatic Text Generation Example of Shakespeare
Example taken from Andrej Karpathy blog post

Generating Sequences With Recurrent NN, Graves, 2014 https://arxiv.org/pdf/1308.0850v5.pdf

# Automatic Image Caption Generation

"man in black shirt is playing guitar."

"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"girl in pink dress is jumping in air."

"black and white dog jumps over bar."

"young girl in pink shirt is swinging on swing."

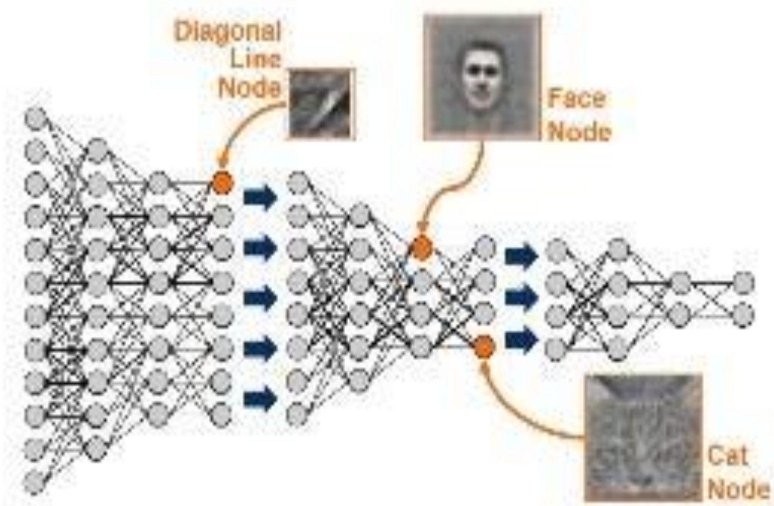
Explain Images with Multimodal Recurrent Neural Networks, Mao et.al, 2014
Sequence to Sequence, Subhashini et.al, 2015

# Deep Learning

All purpose machine learning

Using Neural Networks:
- Using large amounts of data
- Learning very complex problems
- Automatically learning features

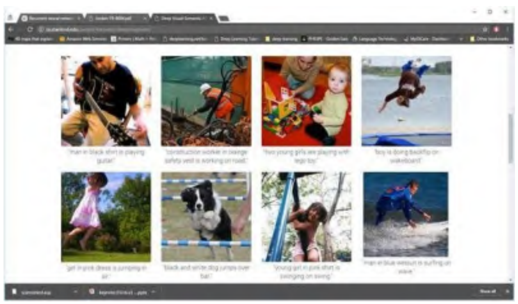
A new era of machine learning



# Neural Networks:

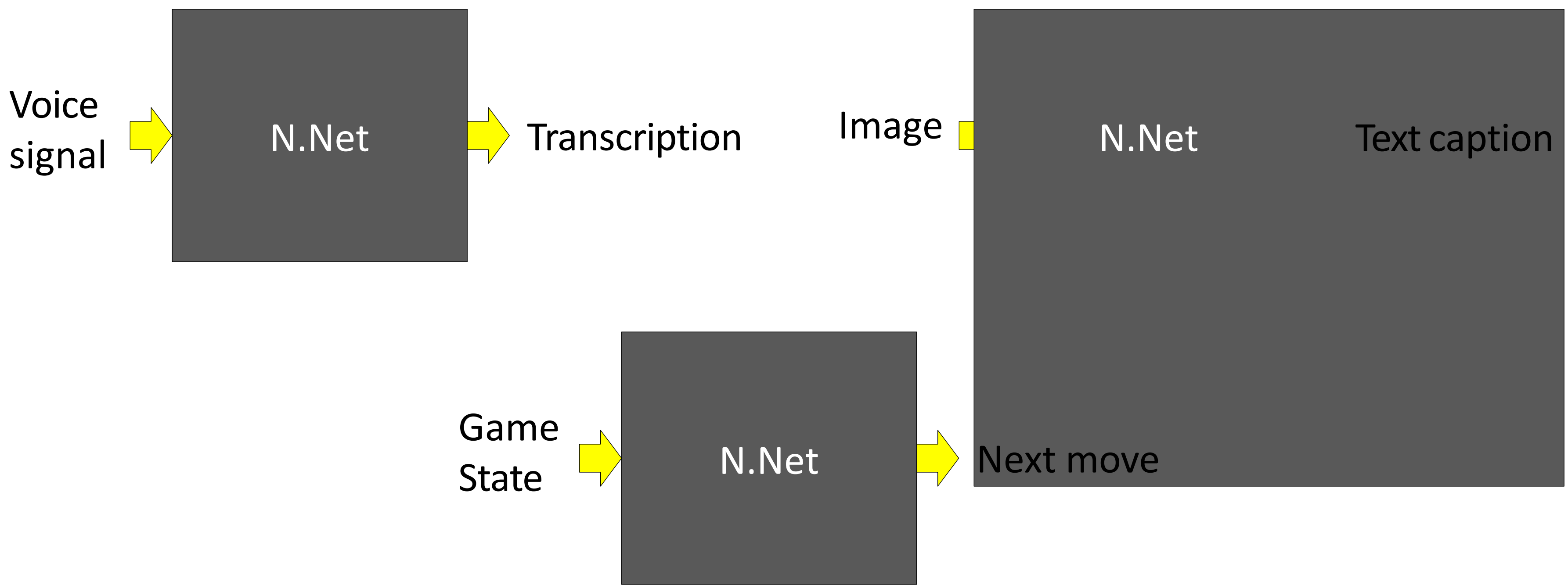
# Neural networks have taken over AI



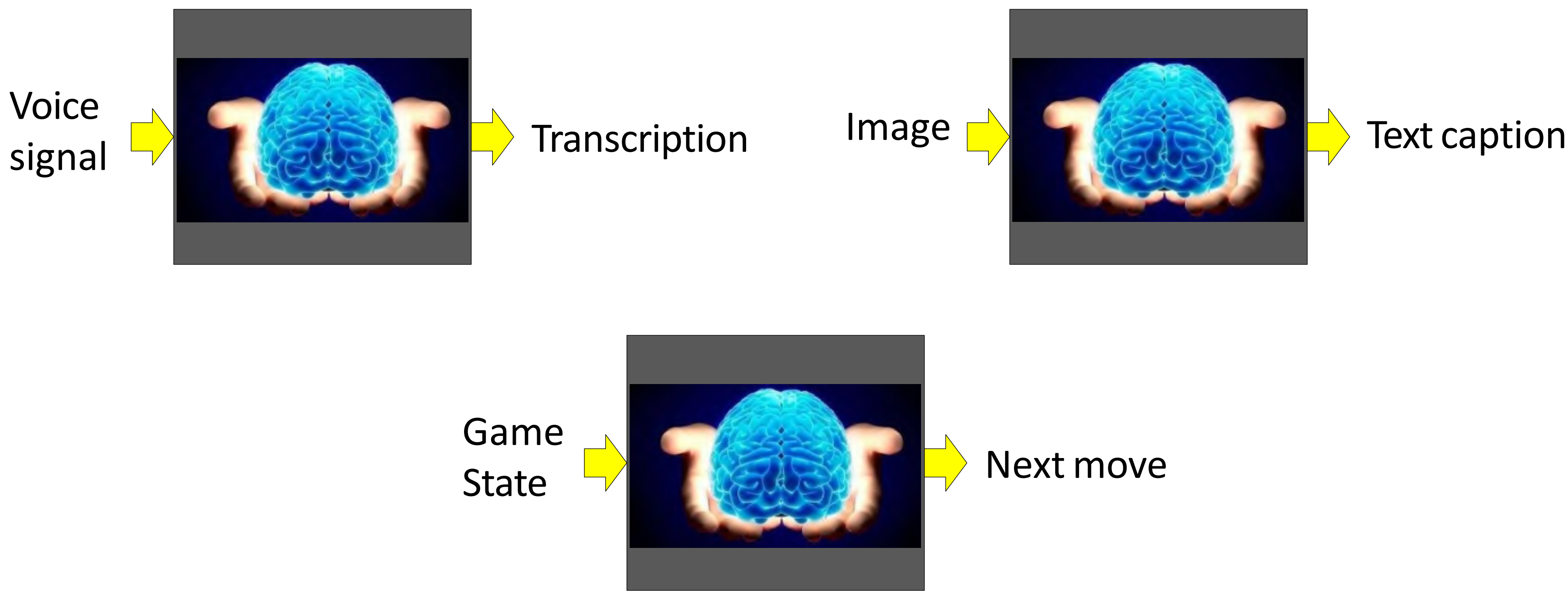

- Tasks that are made possible by NNs, aka deep learning
  - Tasks that were once assumed to be purely in the human domain of expertise

# So what are neural networks??




- What are these boxes?
  - Functions that take an input and produce an output
  - What's in these functions?

# The human perspective

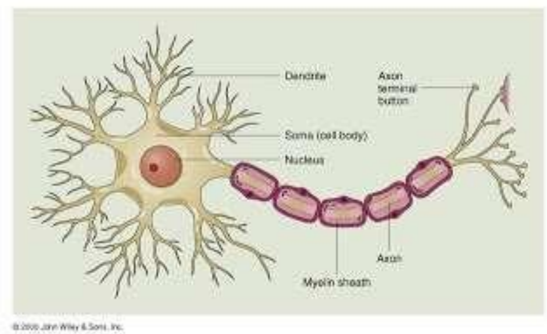



- In a human, those functions are computed by the brain…

# Recap : NNets and the brain

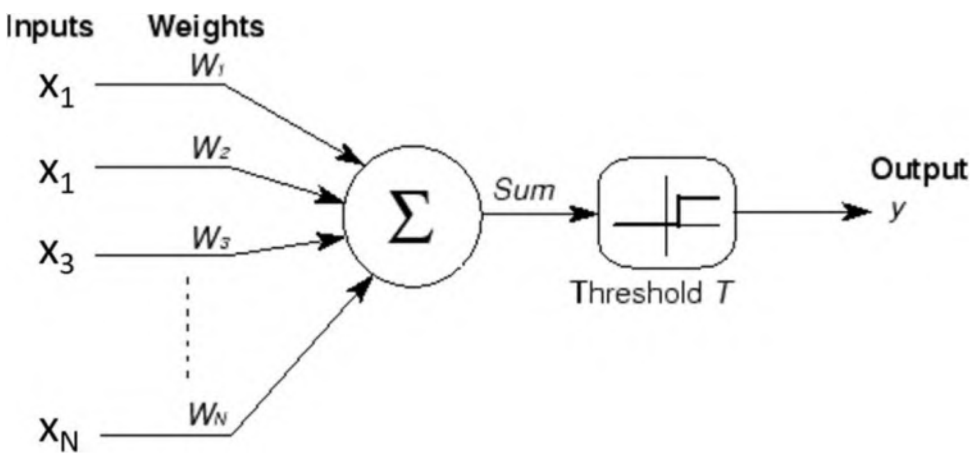
- In their basic form, NNets mimic the networked structure in the brain

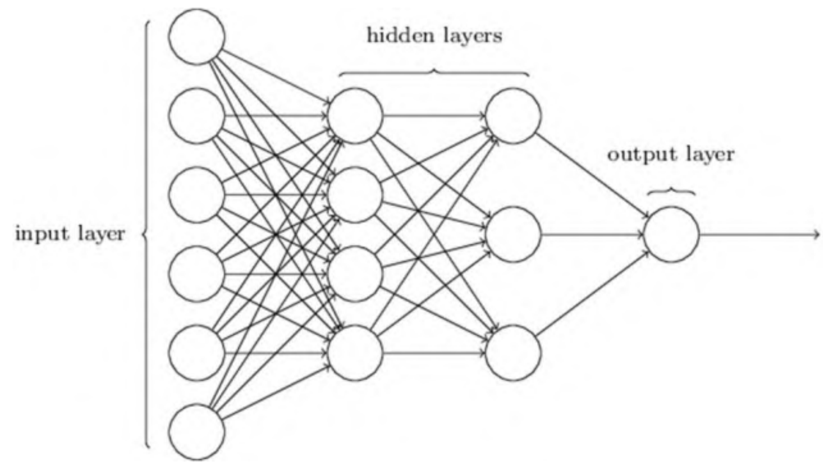
# Recap : The brain

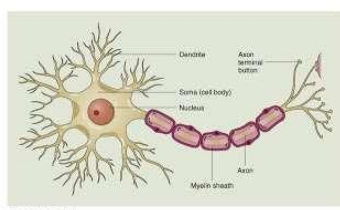



- The Brain is composed of networks of neurons

# Recap : Nnets and the brain

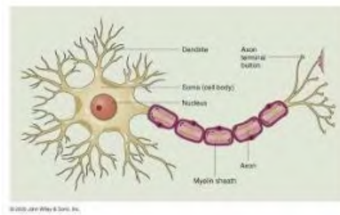


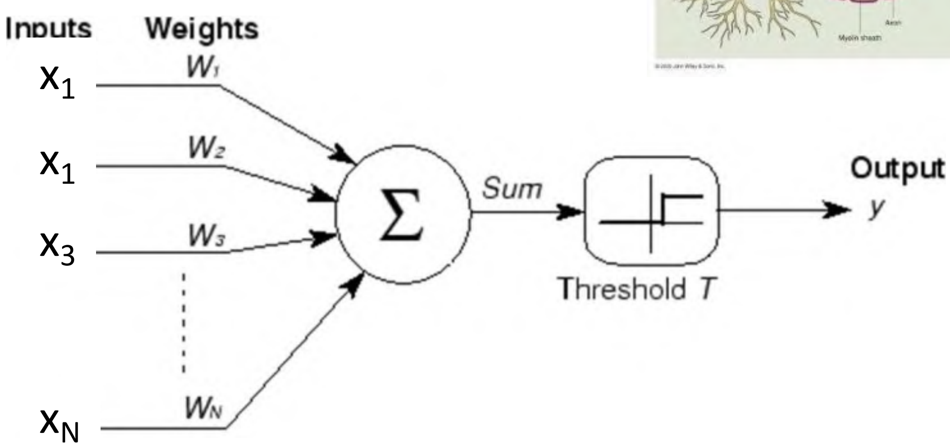



- Neural nets are composed of networks of computational models of neurons called perceptrons

# Recap: the perceptron

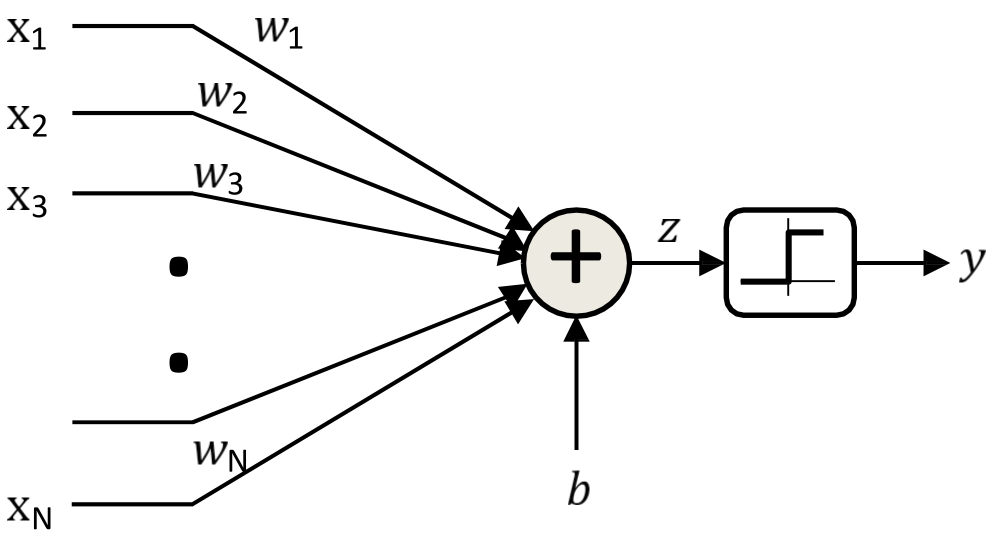





$$y = \begin{cases} 1 \; if \; \sum_{\mathrm{i}} w_{\mathrm{i}} \mathrm{x}_{\mathrm{i}} \geq T \\ 0 \; else \end{cases}$$

- A threshold unit
  - "Fires" if the weighted sum of inputs exceeds a threshold
  - Electrical engineers will call this a ***threshold gate***
    - A basic unit of Boolean circuits

# A better figure

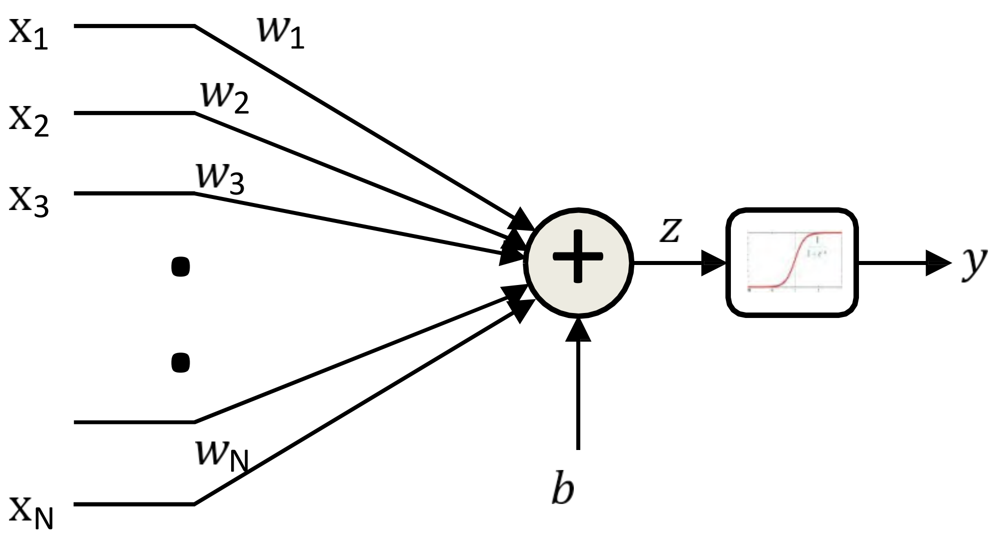



$$z = \sum_i w_i x_i + b$$

$$y = \begin{cases} 1 \; if \; z \geq 0 \\ 0 \; else \end{cases}$$

- A threshold unit
  - "Fires" if the affine function of inputs is positive
    - The bias is the negative of the threshold T in the previous slide

# The "soft" perceptron (logistic)

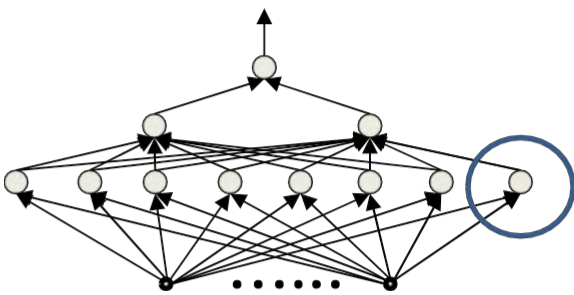


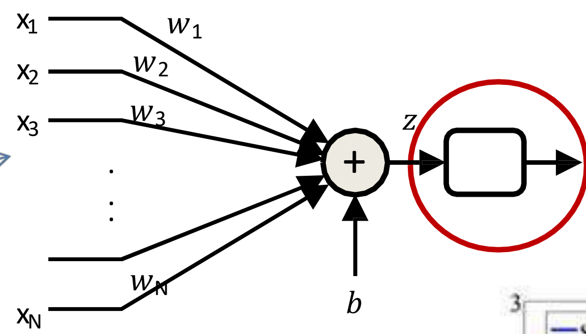

$$z = \sum_i \mathrm{w}_i \mathrm{x}_i + b$$

$$y = \frac{1}{1 + exp(-z)}$$

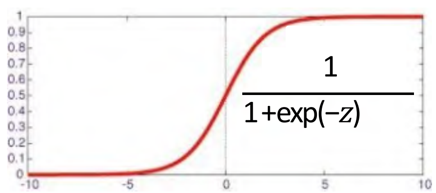
- A "squashing" function instead of a threshold at the output
  - The sigmoid "activation" replaces the threshold
    - **Activation:** The function that acts on the weighted combination of inputs (and threshold)

# Other “activations”

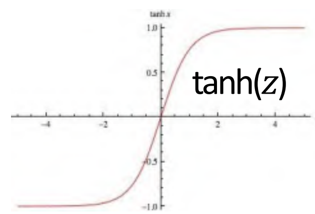


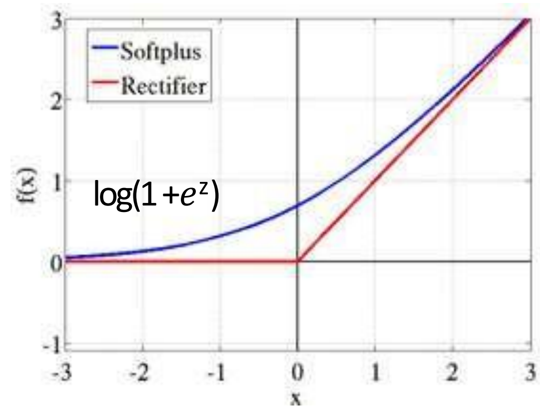



sigmoid



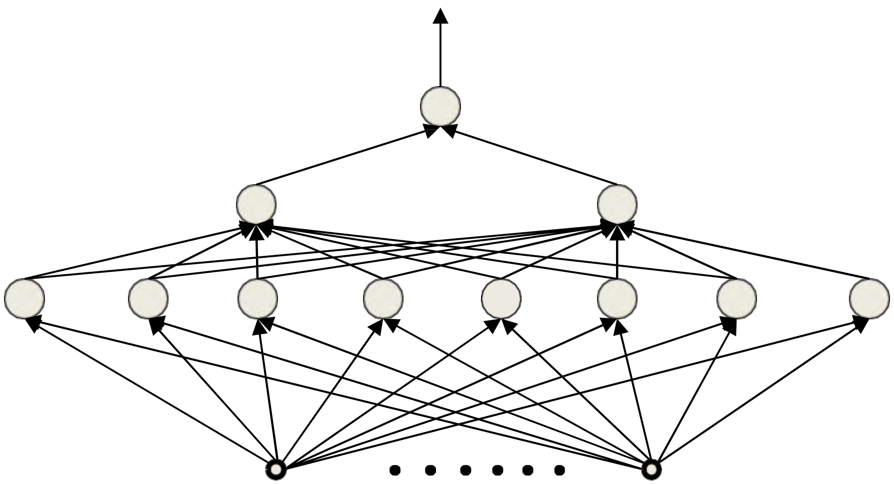

tanh

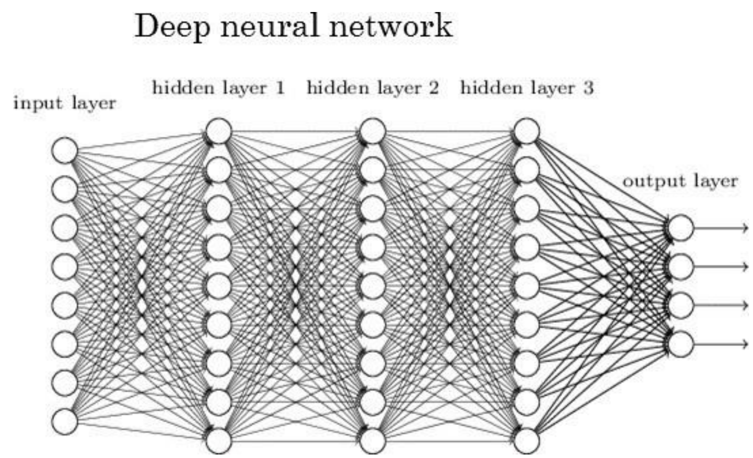



- Does not always have to be a squashing function
  - We will hear more about activations later
- We will continue to assume a “threshold” activation in this lecture

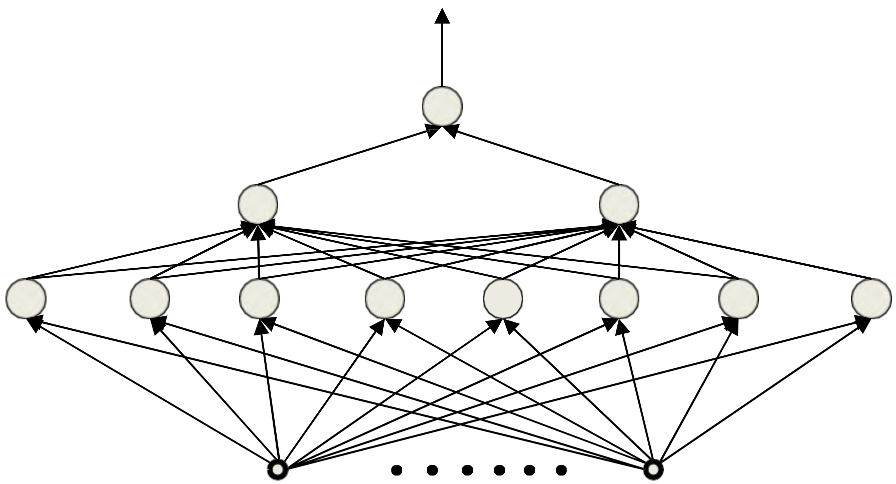
# The *multi-layer* perceptron

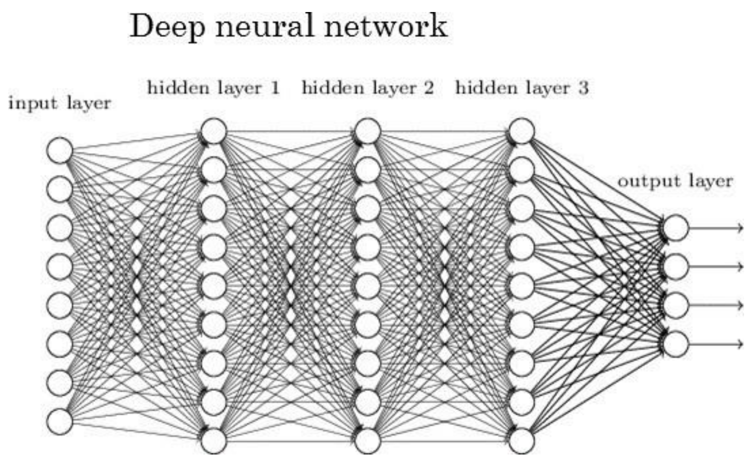





- A network of perceptrons
  - Perceptrons "feed" other perceptrons
  - We give you the "formal" definition of a layer later

# Defining “depth”

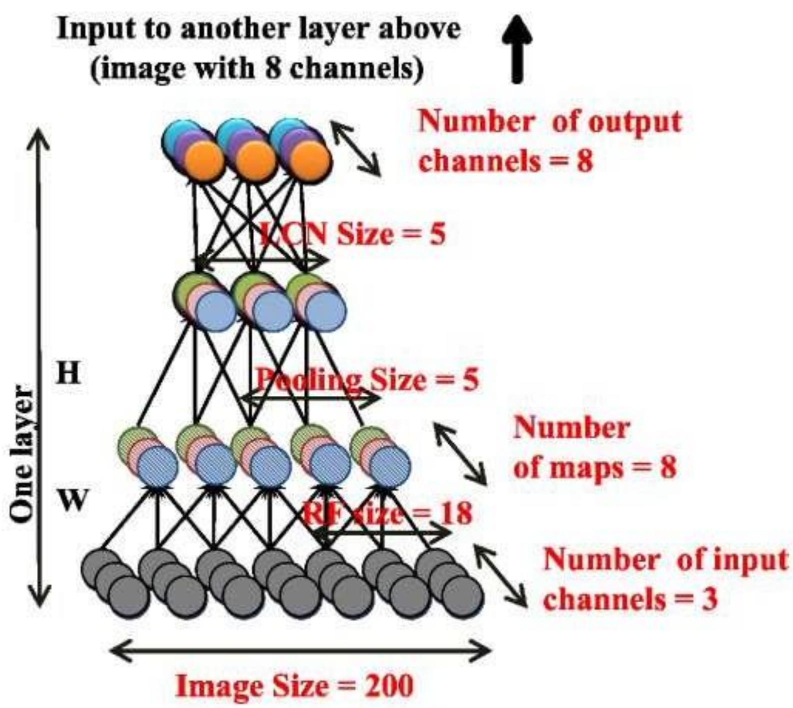


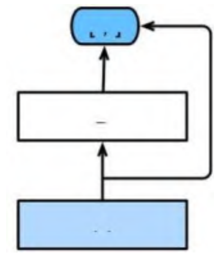


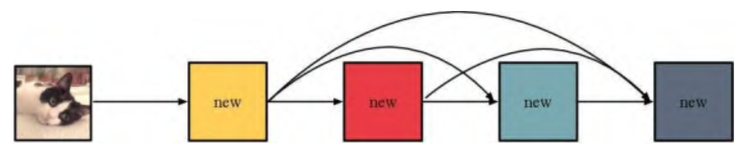

- What is a “deep” network

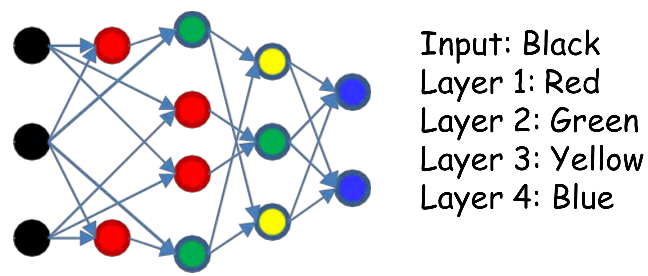
# Deep Structures

- *Layered* deep structure
  - *The input is the “source”,*
  - *The output nodes are “sinks”*



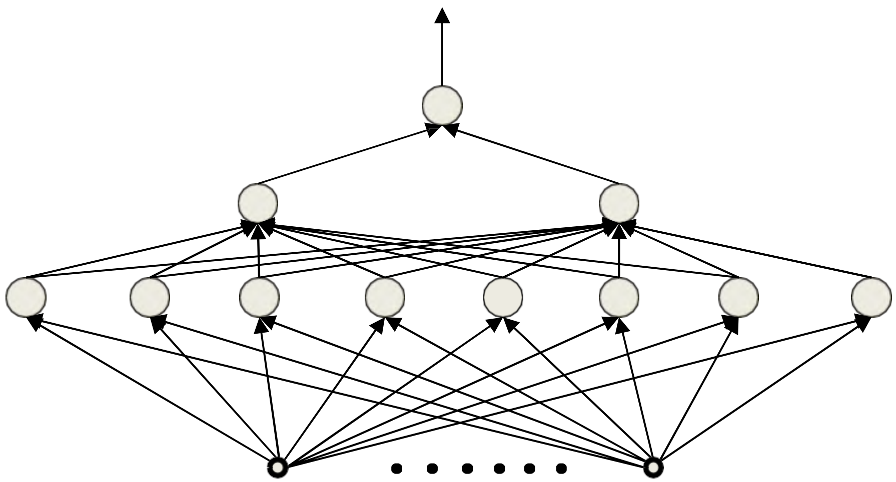



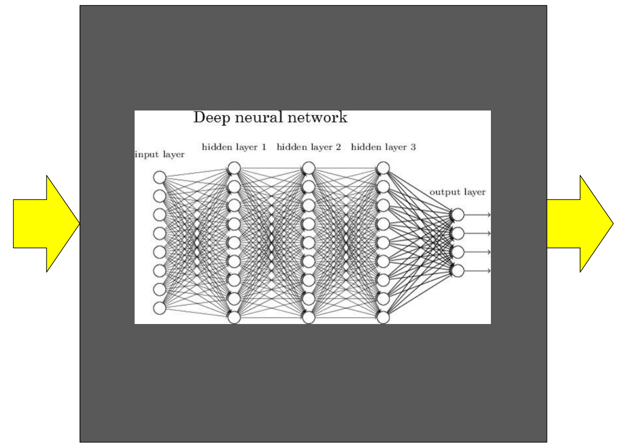



- “Deep” ☐ Depth greater than 2
- “Depth” of a layer – the depth of the neurons in the layer w.r.t. input

# The multi-layer perceptron

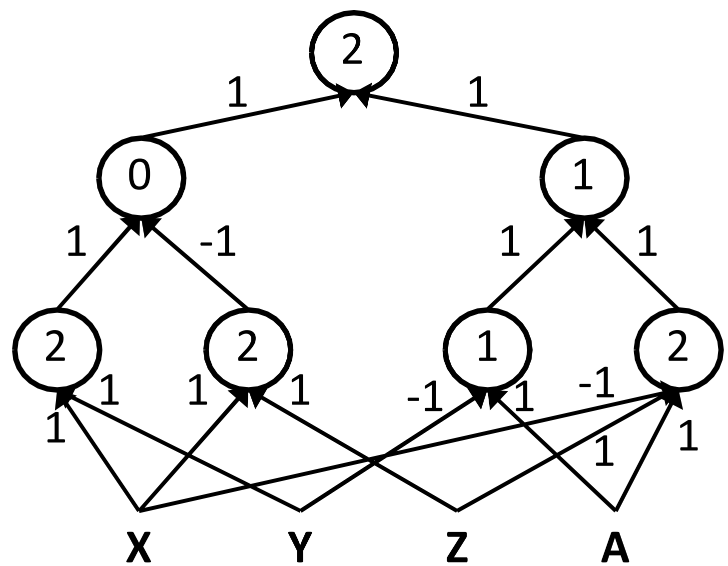


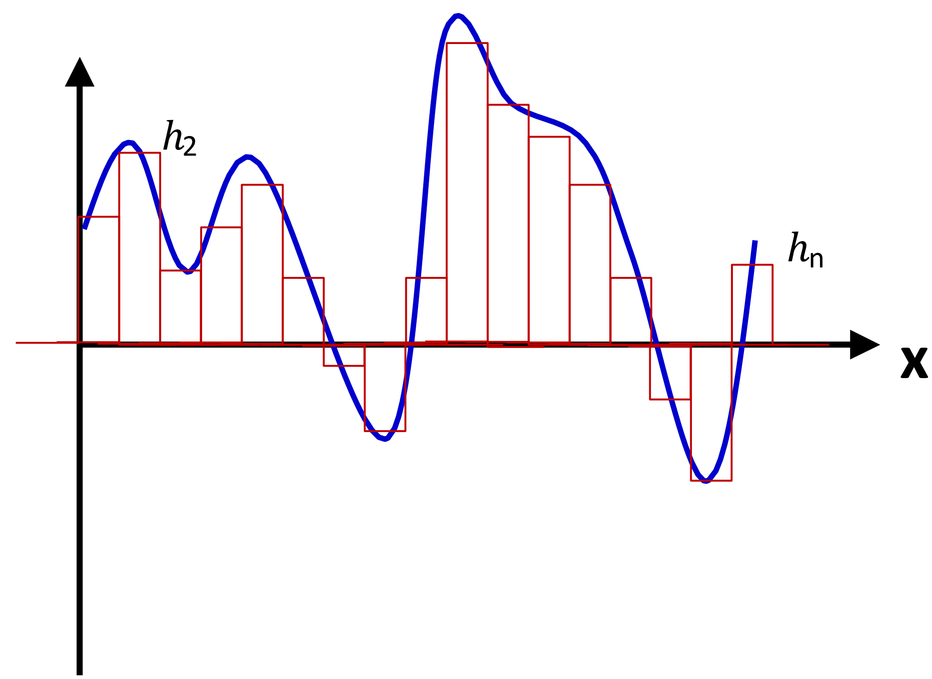



- Inputs are real or Boolean stimuli
- Outputs are real or Boolean values
  - Can have multiple outputs for a single input
- **What can this network compute?**
  - **What kinds of input/output relationships can it model?**

# MLPs approximate functions

$$((A\&\bar{X}\&Z)|(A\&\bar{Y}))\&((X\ \&\ Y)|\overline{(X\&Z)})$$



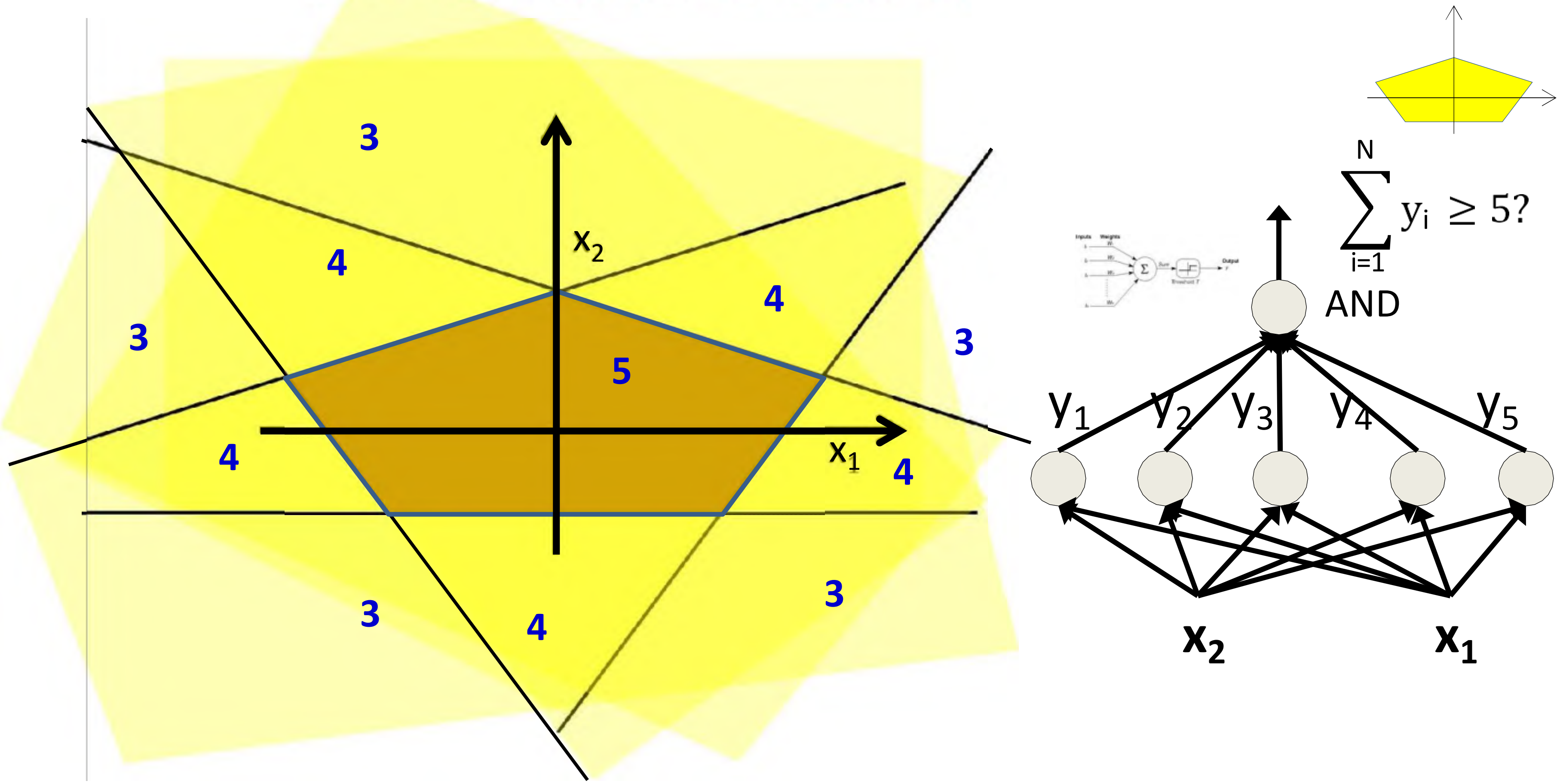




- MLPs can compose Boolean functions
- MLPs can compose real-valued functions
- What are the limitations?

# Today

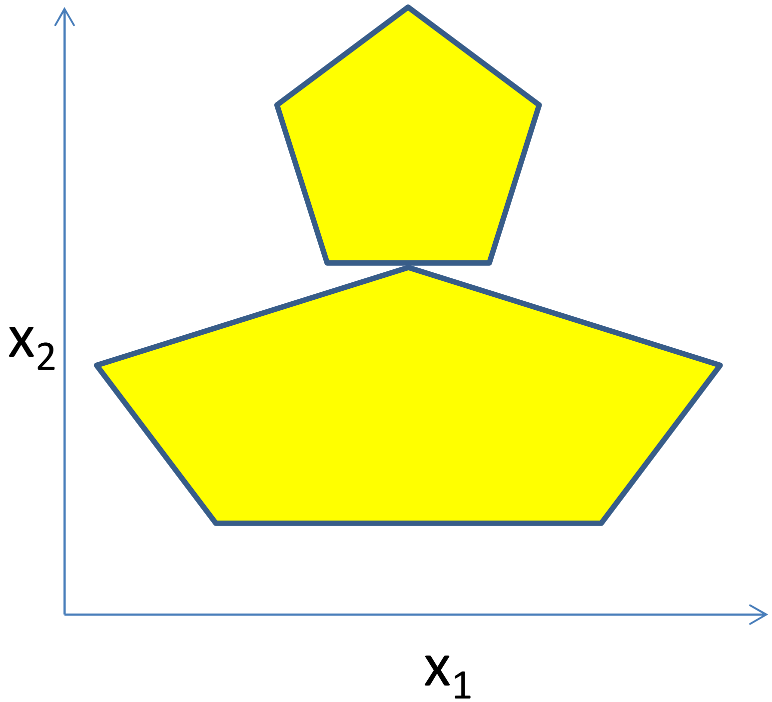
- Multi-layer Perceptrons as universal Boolean functions
  - The need for depth
- MLPs as universal classifiers
  - The need for depth
- MLPs as universal approximators

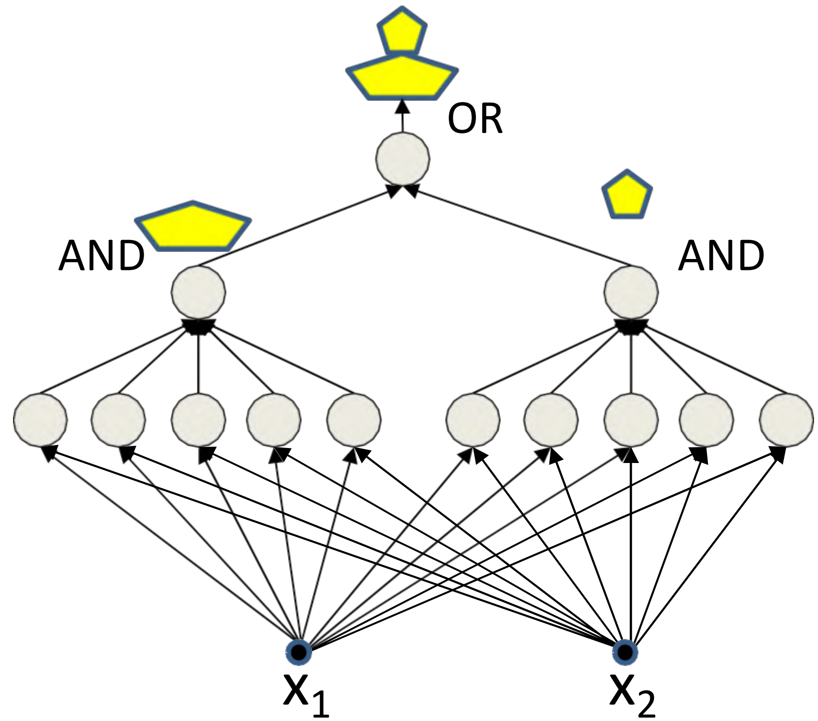
# Booleans over the reals




- The network must fire if the input is in the coloured area
  - The AND compares the sum of the hidden outputs to 5
    - NB: What would the pattern be if it compared it to 4?

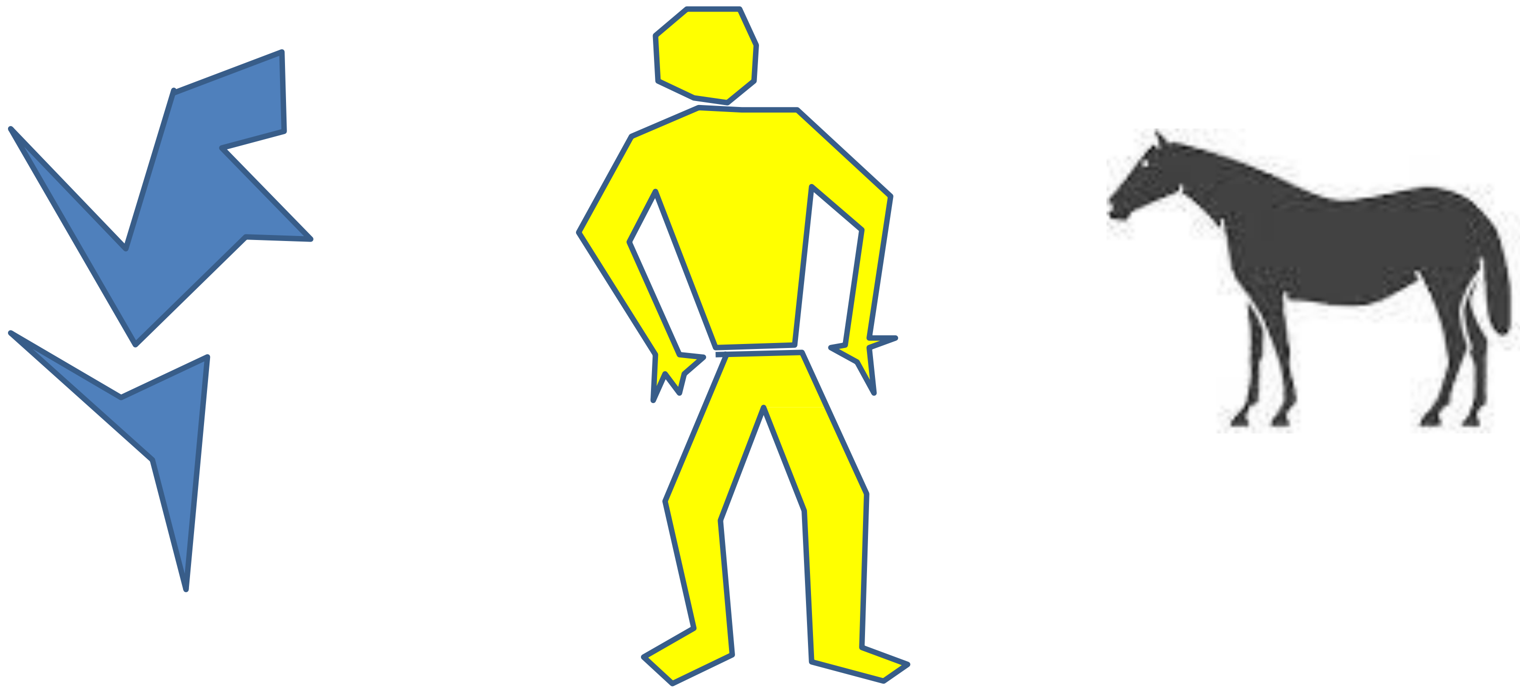
# More complex decision boundaries



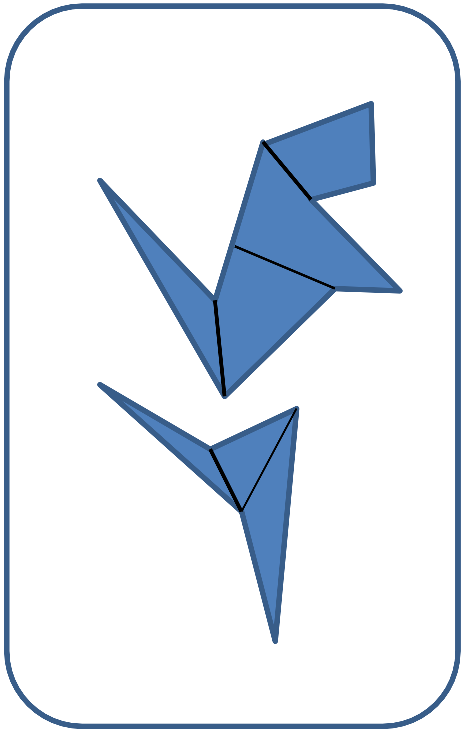

- Network to fire if the input is in the yellow area
  - "OR" two polygons
  - A third layer is required

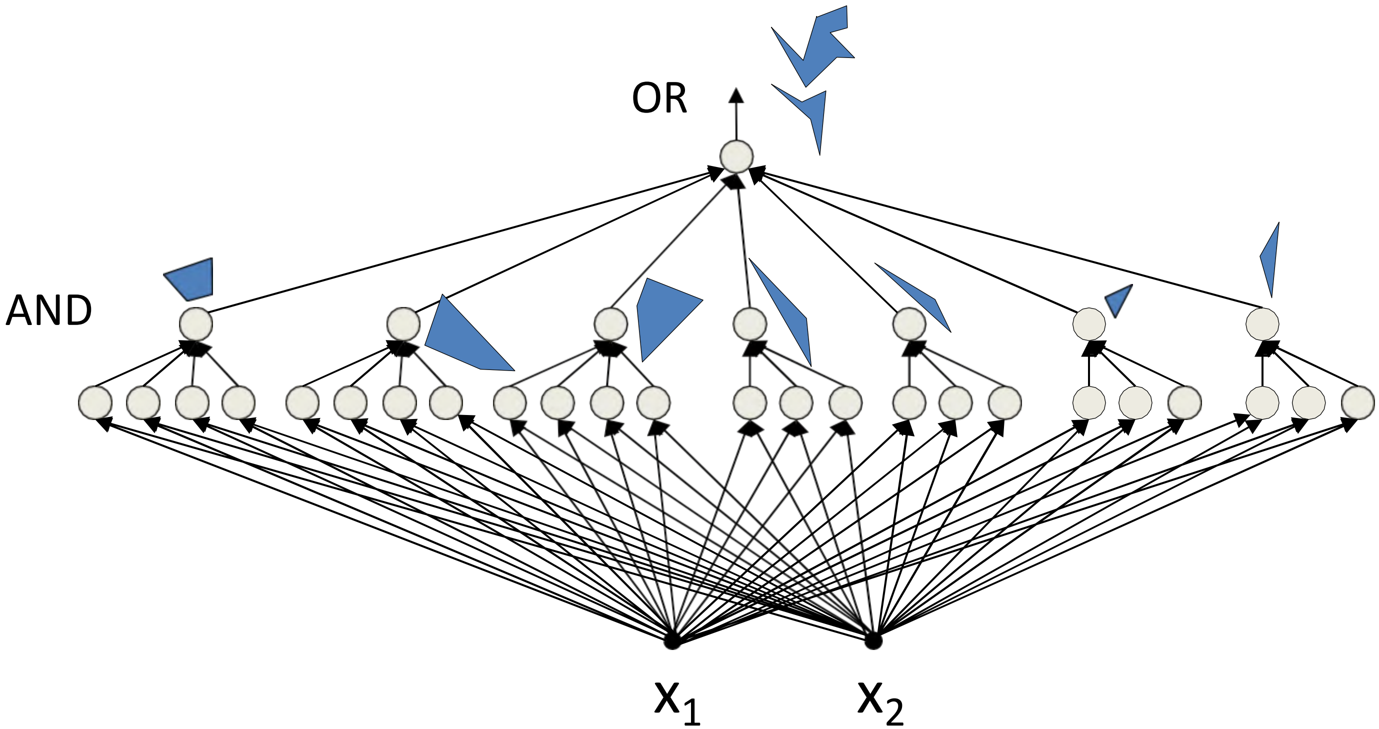
# Complex decision boundaries



- Can compose *arbitrarily* complex decision boundaries

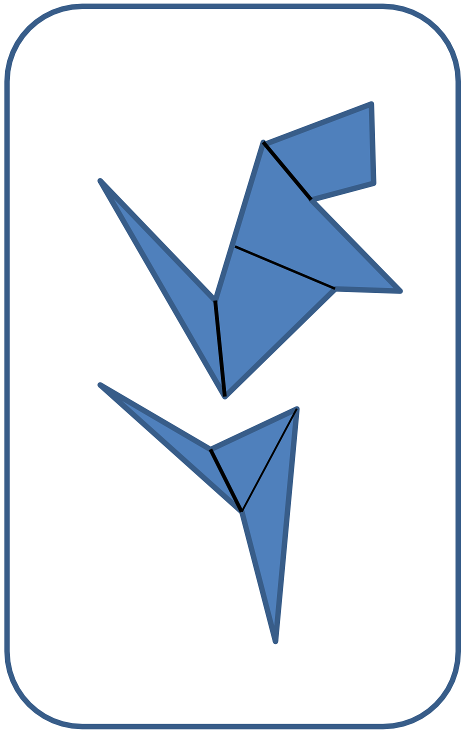
# Complex decision boundaries

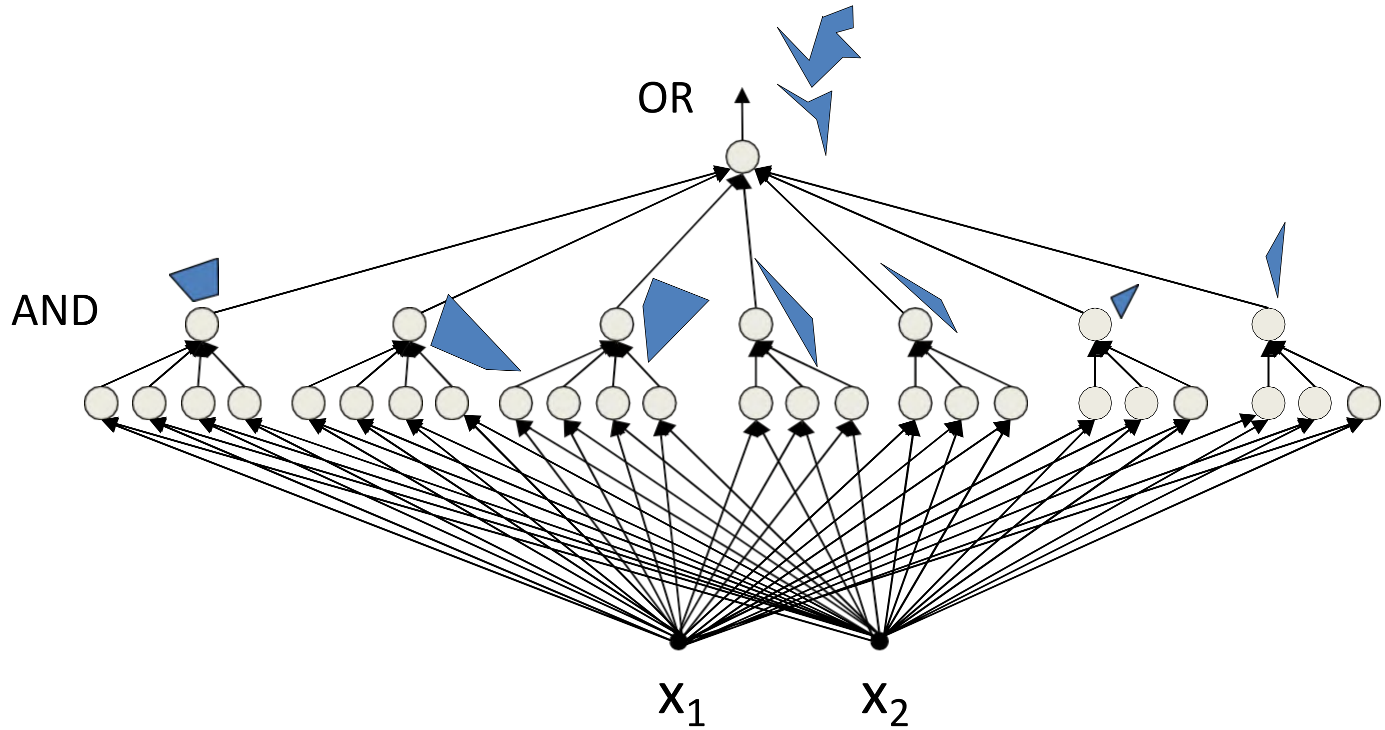





- Can compose *arbitrarily* complex decision boundaries

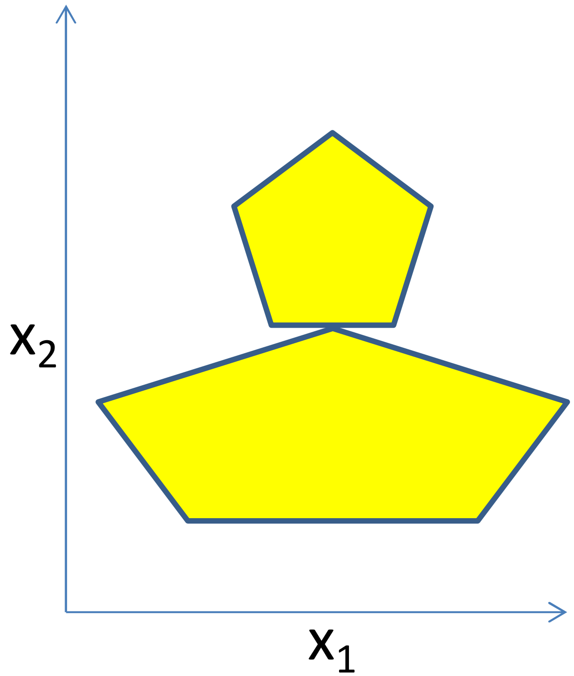
# Complex decision boundaries

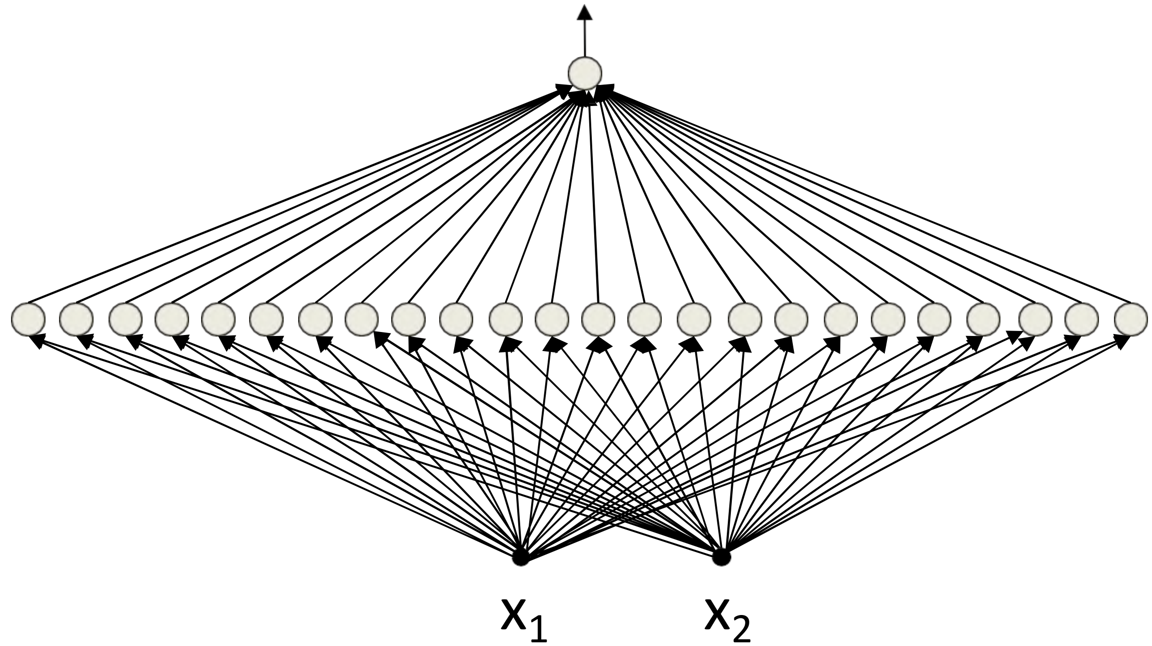





- Can compose *arbitrarily* complex decision boundaries
  - With *only one hidden layer!*
  - **How?**

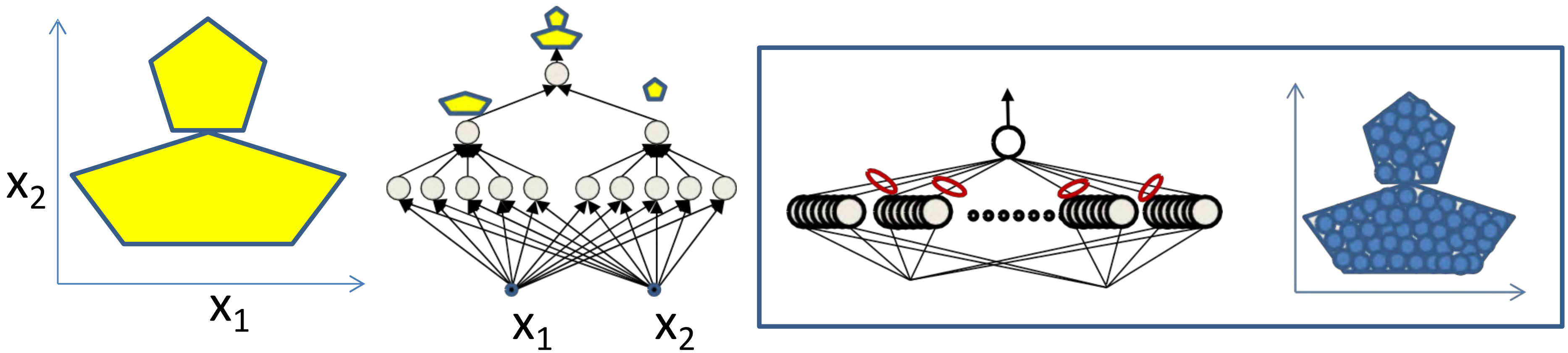
# Exercise: compose this with one hidden layer




- How would you compose the decision boundary to the left with only *one* hidden layer?

# Depth and the universal classifier

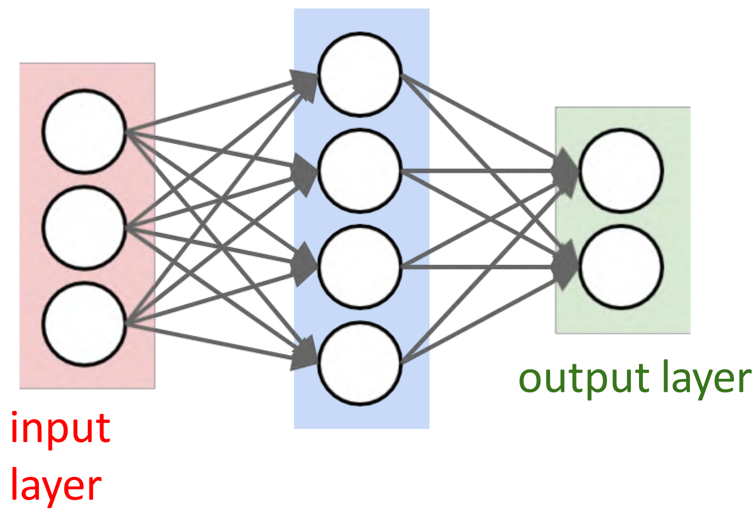



- Deeper networks can require far fewer neurons

# Deep Neural Networks
# Scanning for patterns
# (aka convolutional networks)

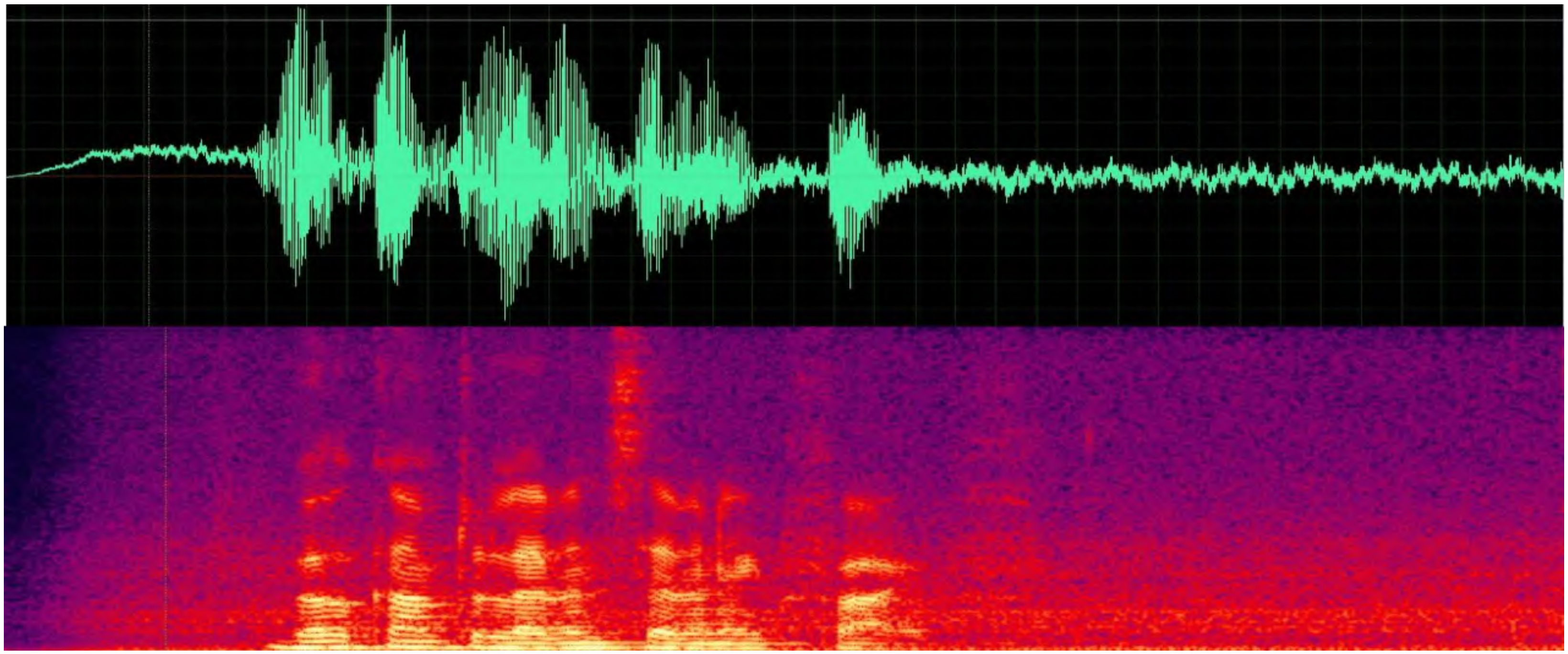
# The model so far

Or, more generally
a vector input



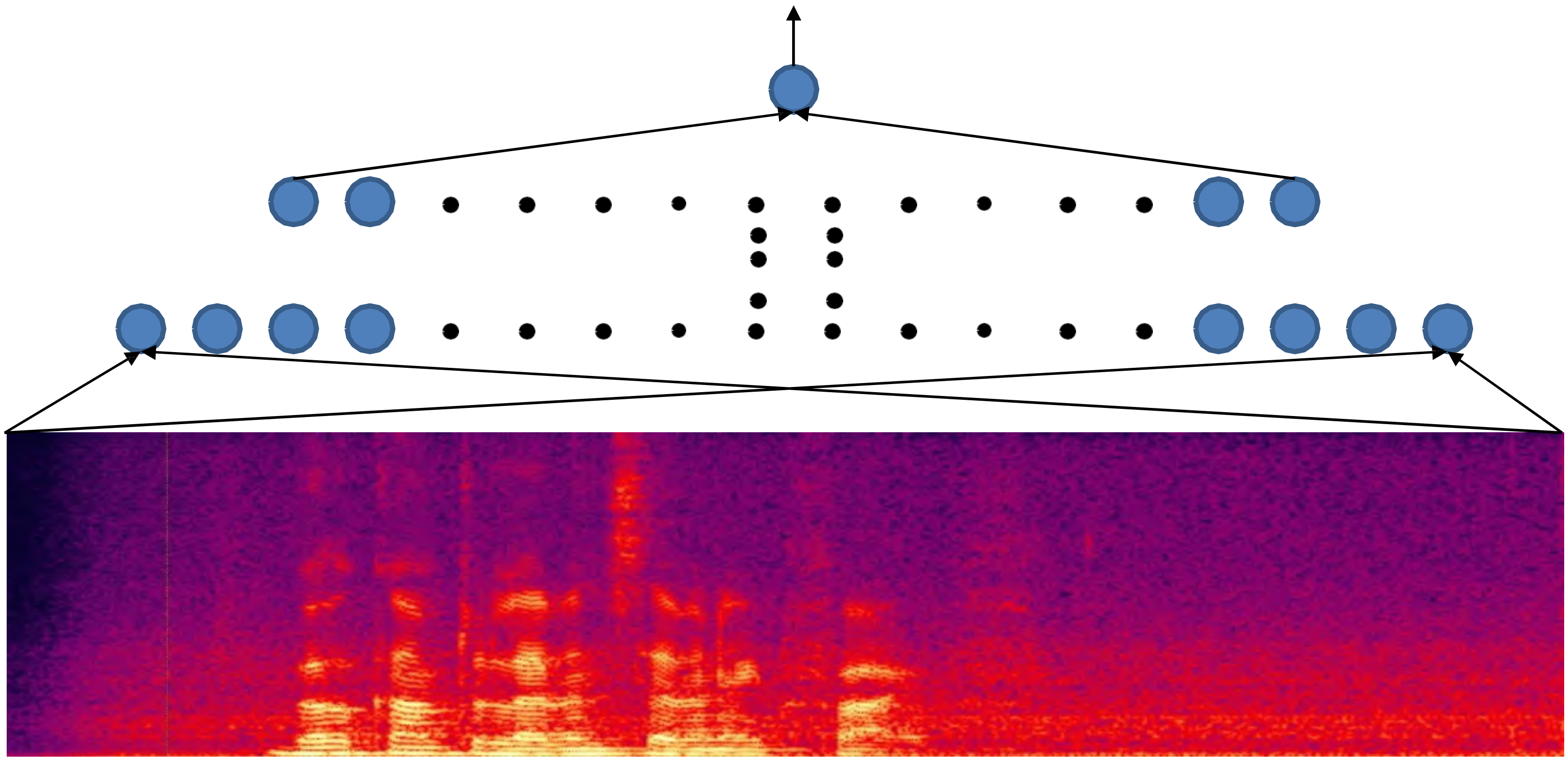

- Can recognize patterns in data
  - E.g. digits
  - Or any other vector data

# A new problem



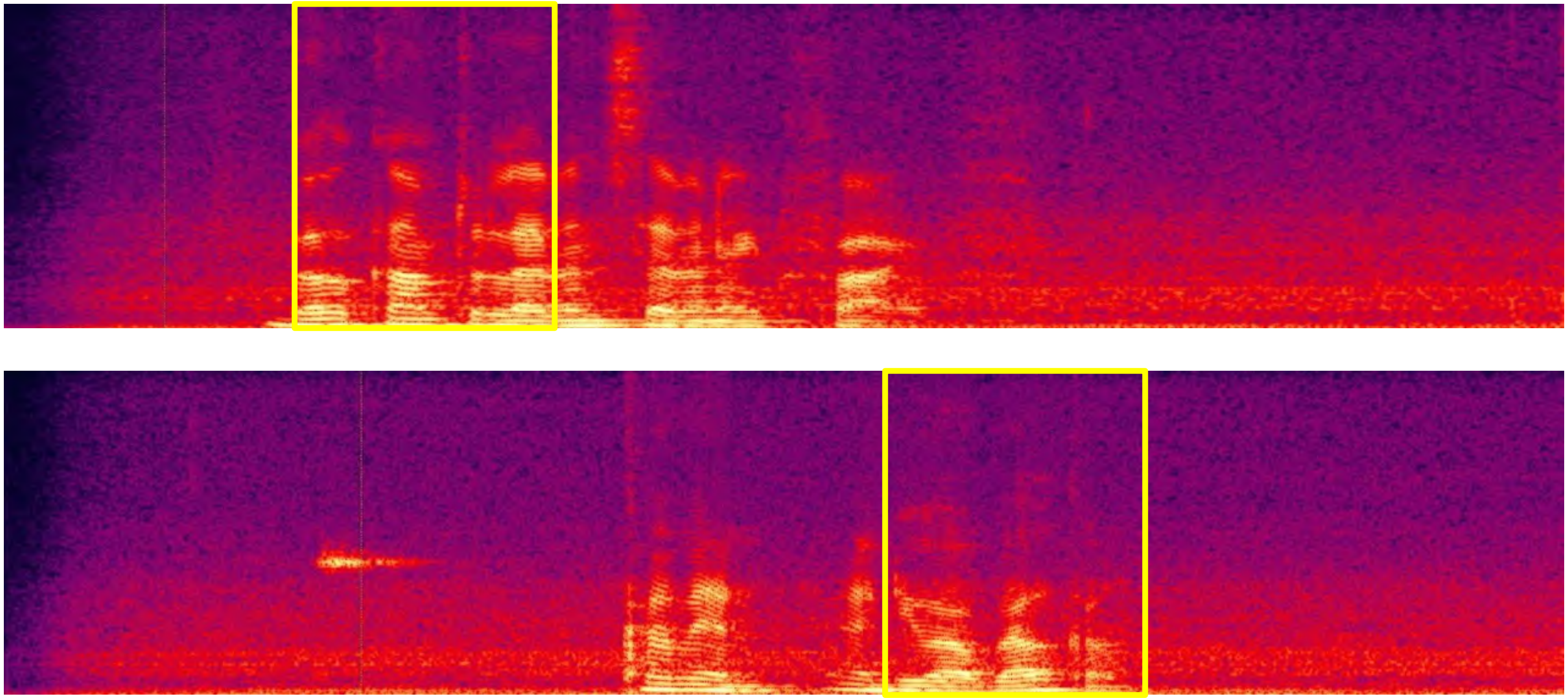
- Does this signal contain the word “Welcome”?
- Compose an MLP for this problem.
  - Assuming all recordings are exactly the same length..

# Finding a Welcome



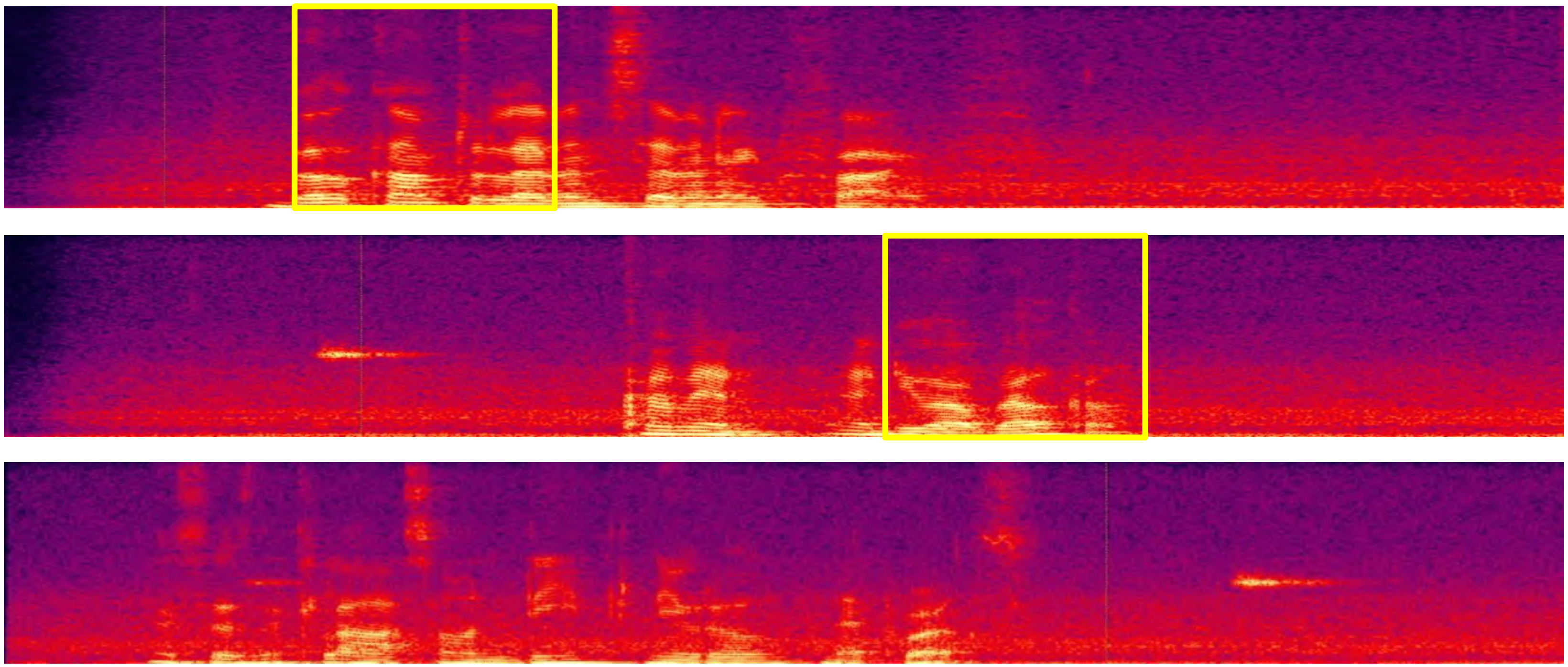
- Trivial solution: Train an MLP for the entire recording

# Finding a Welcome



- Problem with trivial solution: Network that finds a "welcome" in the top recording will not find it in the lower one
  - Unless trained with both
  - Will require a very large network and a large amount of training data to cover every case

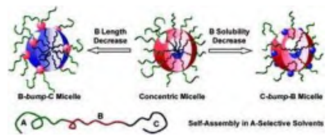
# Finding a Welcome

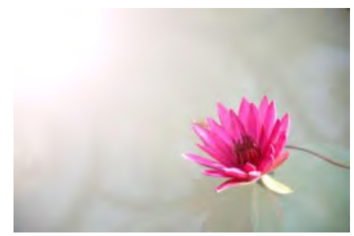


- Need a *simple* network that will fire regardless of the location of "Welcome"
  – and not fire when there is none

# Flowers

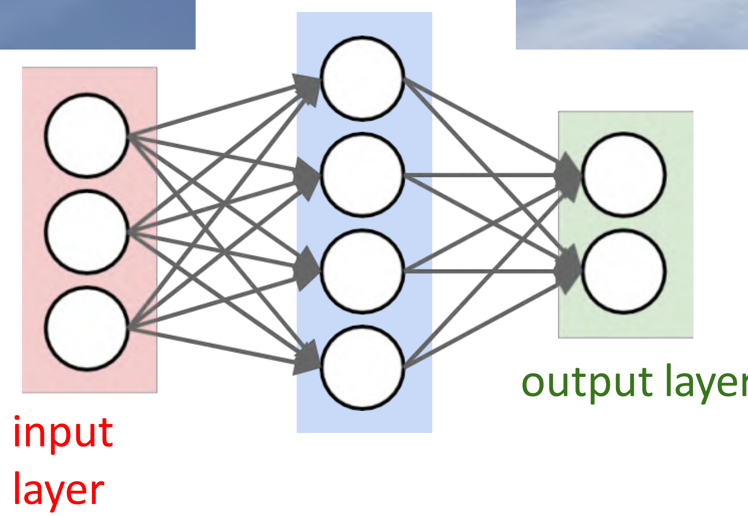


- Is there a flower in any of these images

# A problem

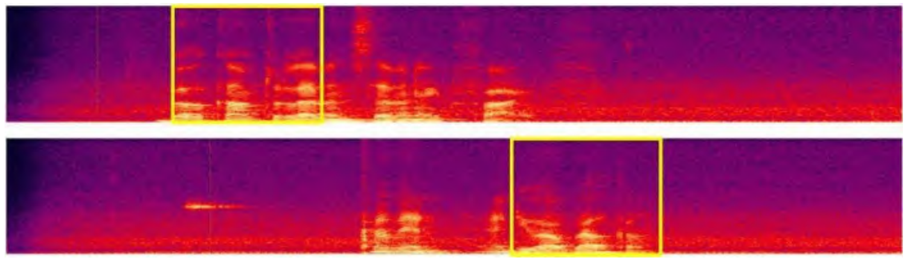



- Will an MLP that recognizes the left image as a flower also recognize the one on the right as a flower?

# A problem

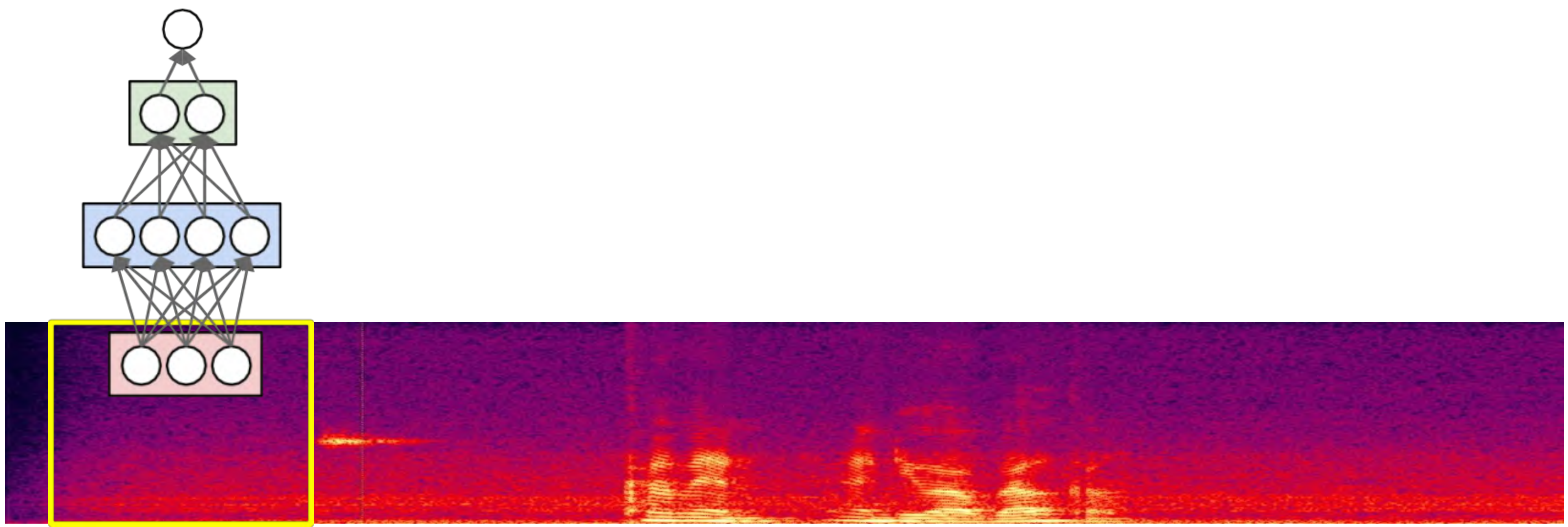
- Need a network that will “fire” regardless of the precise location of the target object

# The need for *shift invariance*



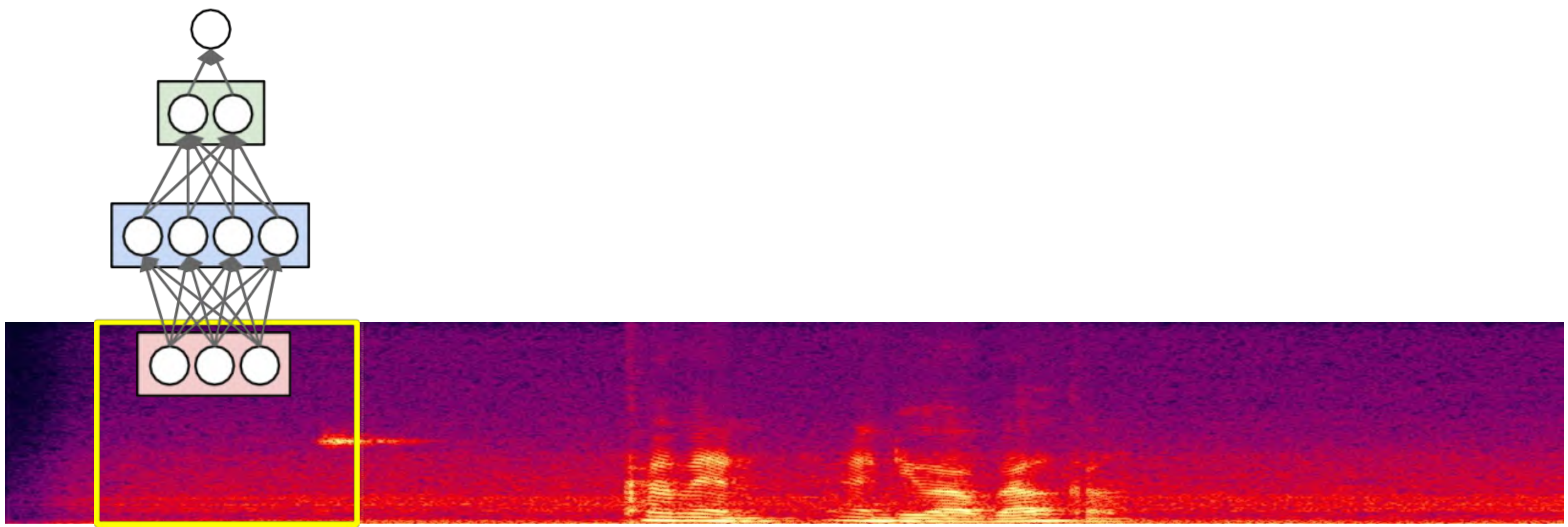
- In many problems the *location* of a pattern is not important
  - Only the presence of the pattern
- Conventional MLPs are sensitive to the location of the pattern
  - Moving it by one component results in an entirely different input that the MLP won't recognize
- Requirement: Network must be *shift invariant*

# Solution: Scan



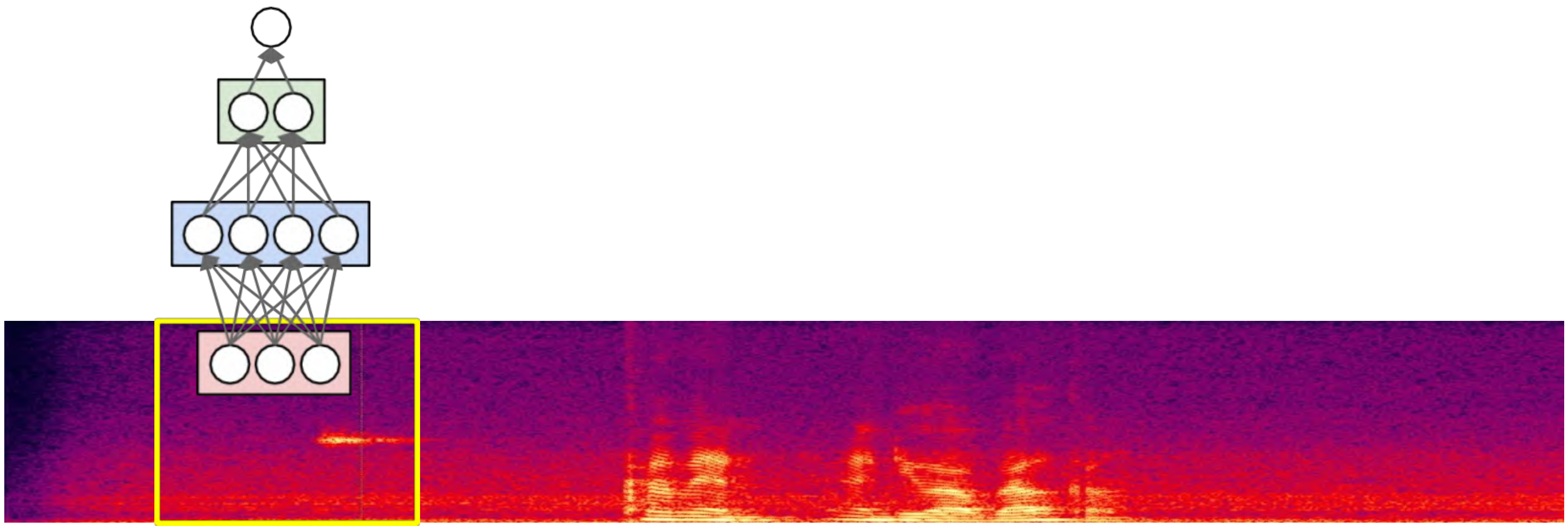
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

# Solution: Scan



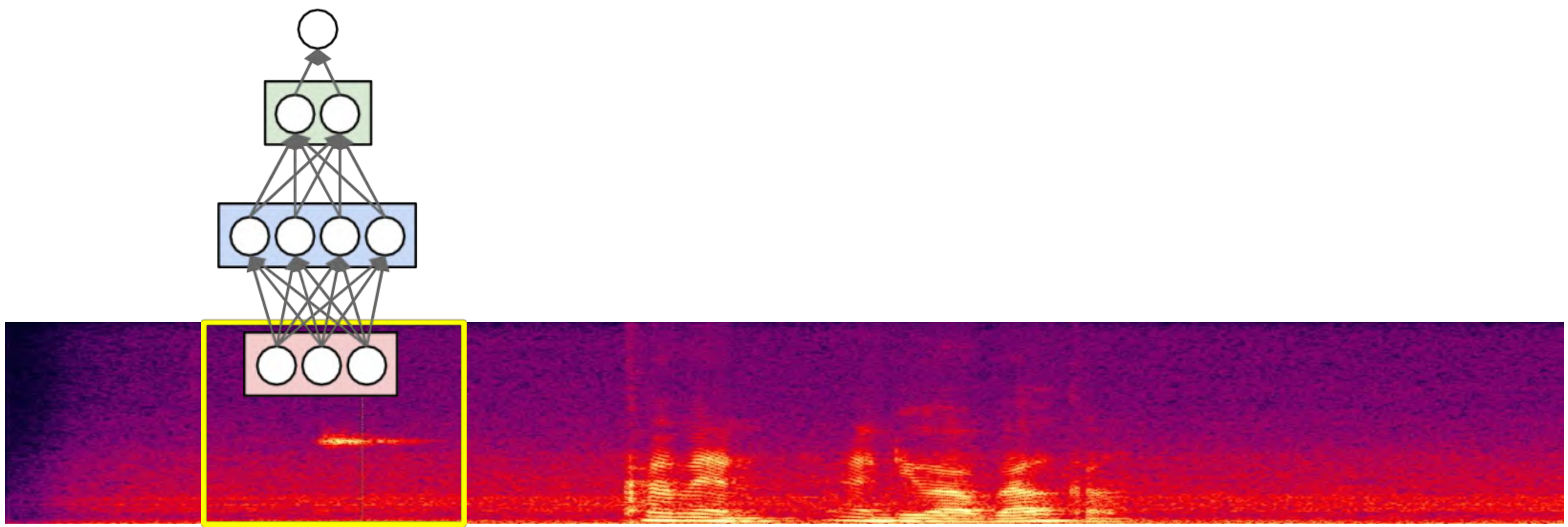
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

# Solution: Scan



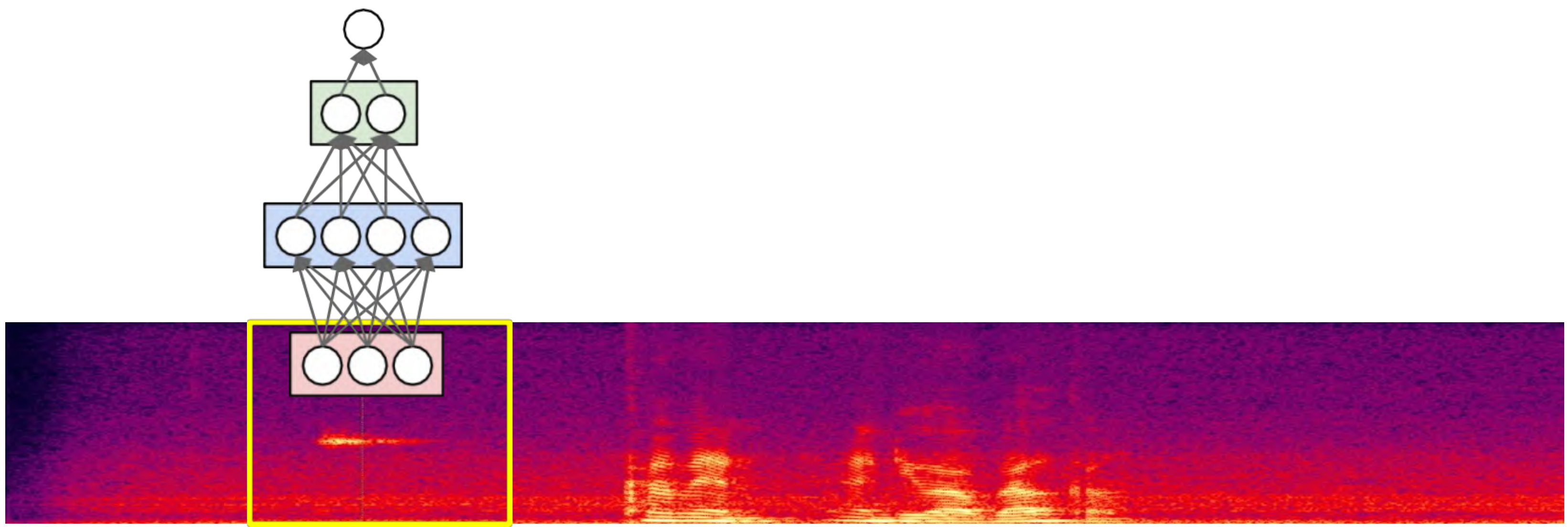
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

# Solution: Scan



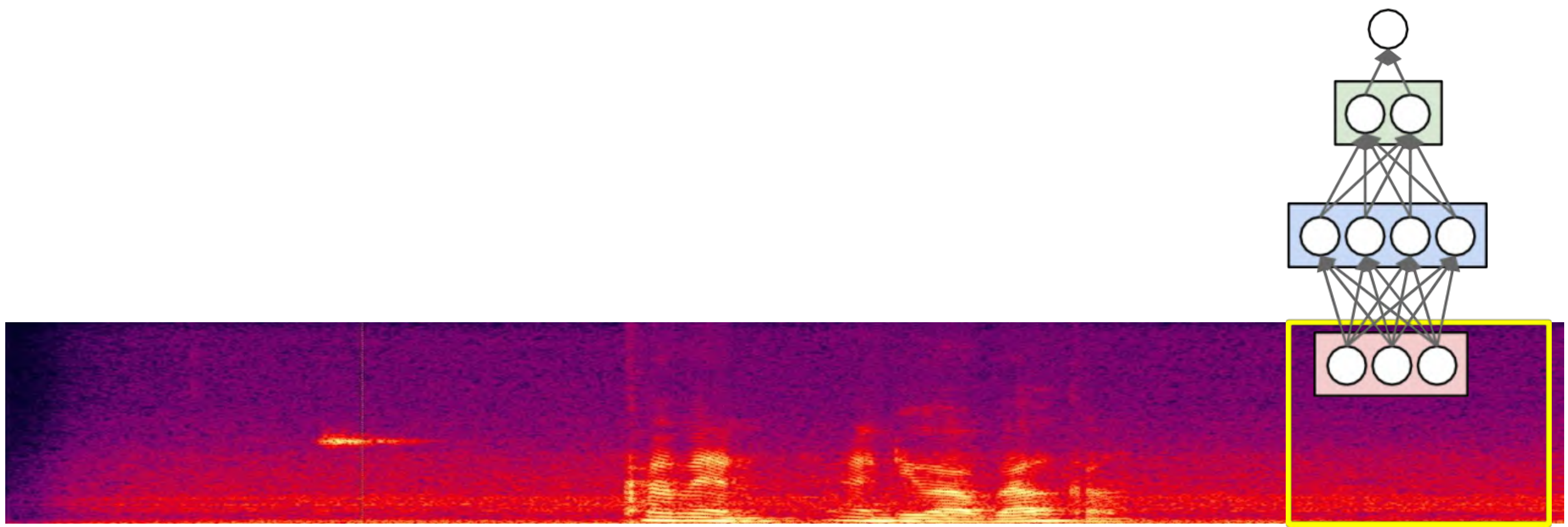
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

# Solution: Scan



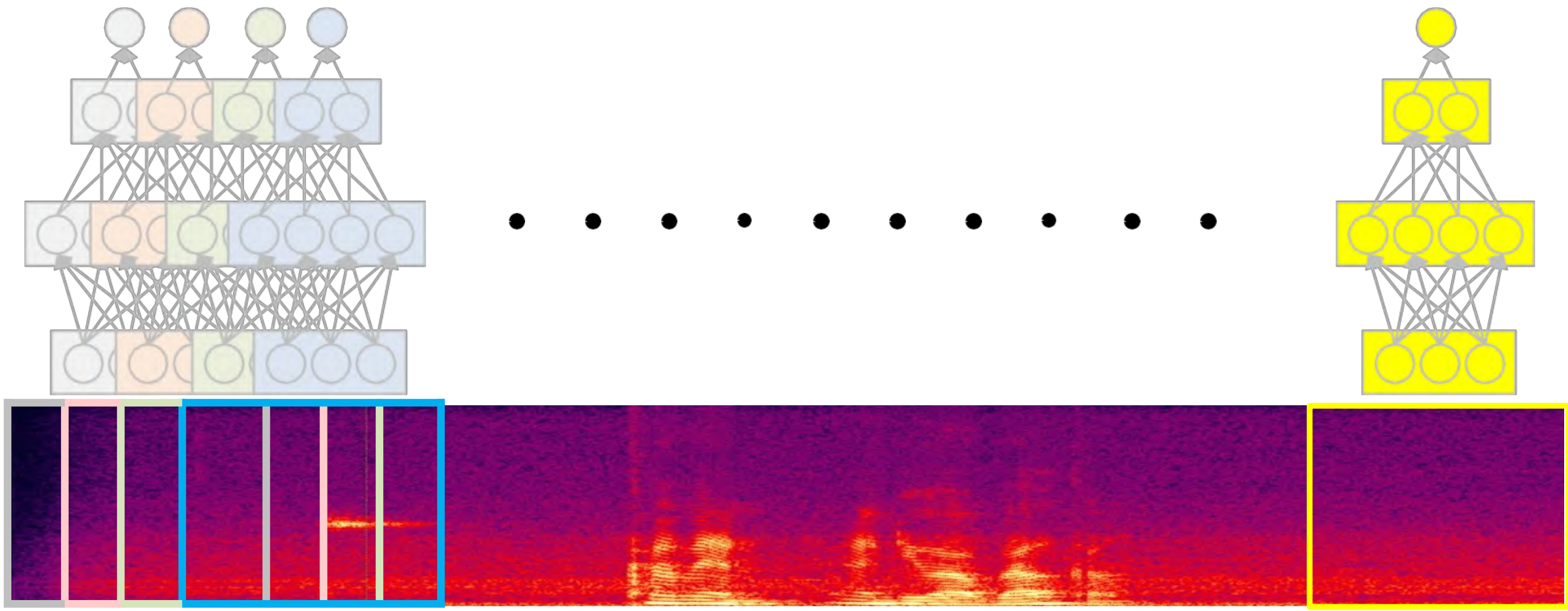
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

# Solution: Scan



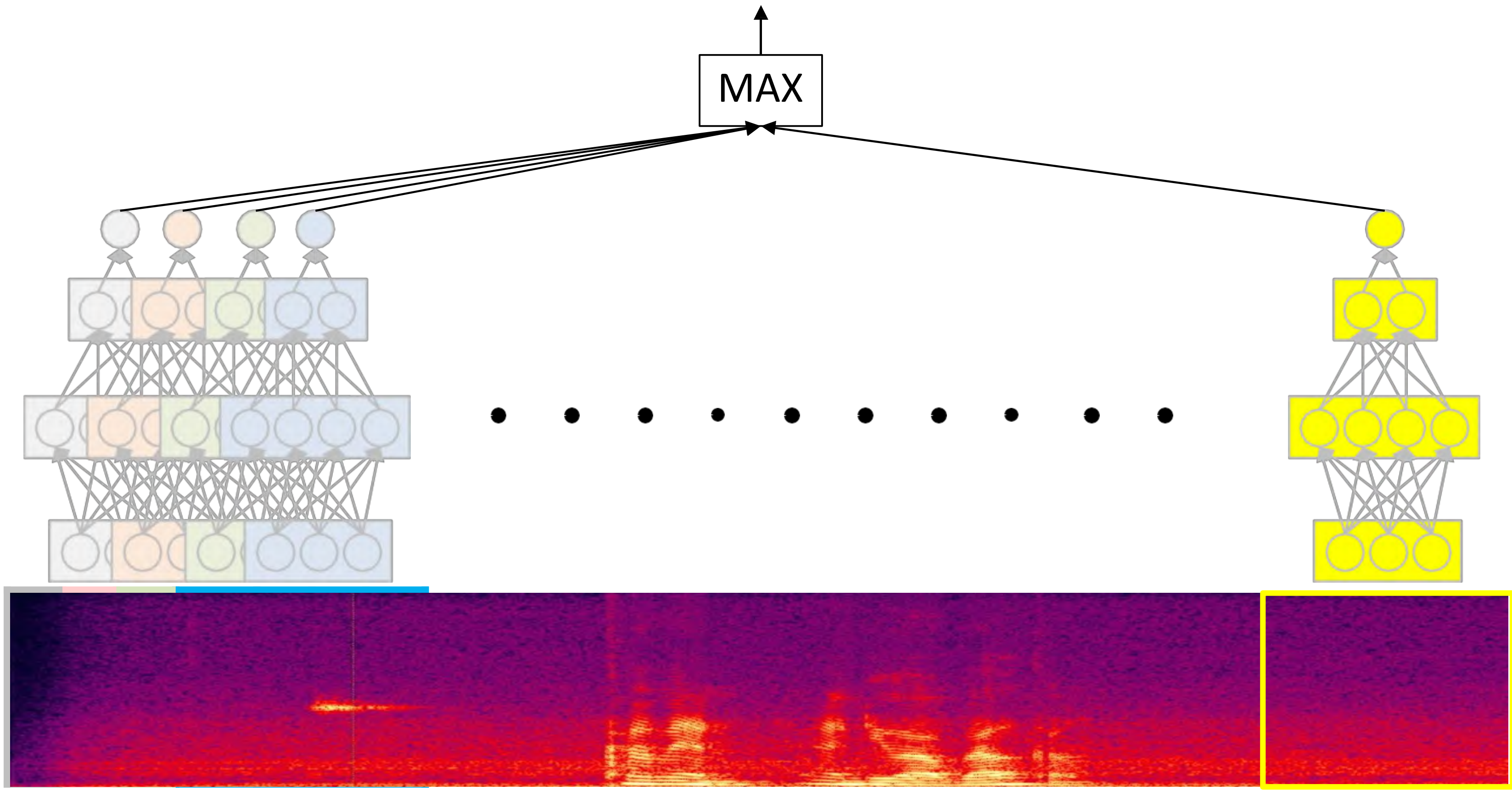
- Scan for the target word
  - The spectral time-frequency components in a "window" are input to a "welcome-detector" MLP

# Solution: Scan



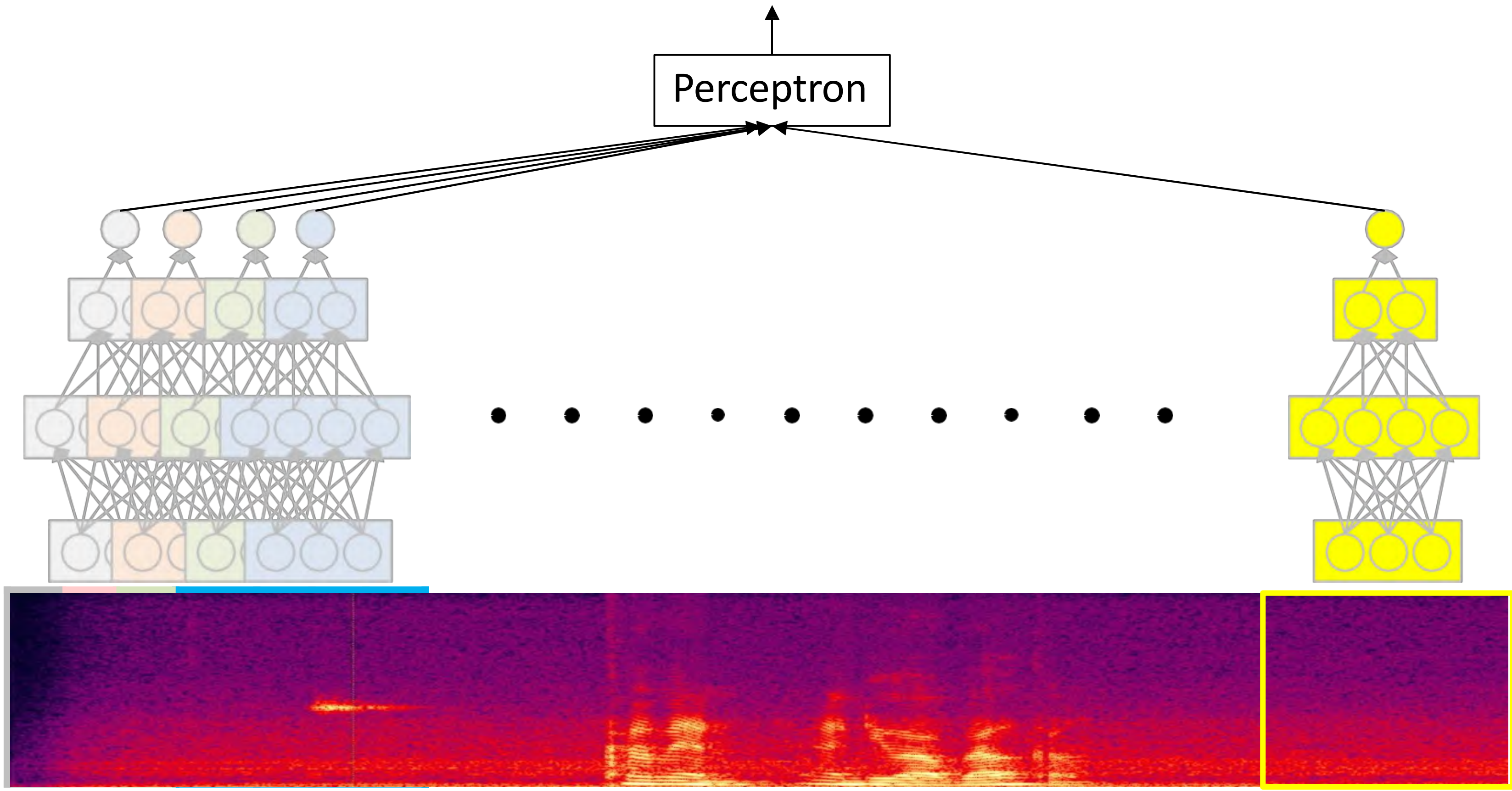
- "Does welcome occur in this recording?"
  - We have classified many "windows" individually
  - "Welcome" may have occurred in any of them

# Solution: Scan



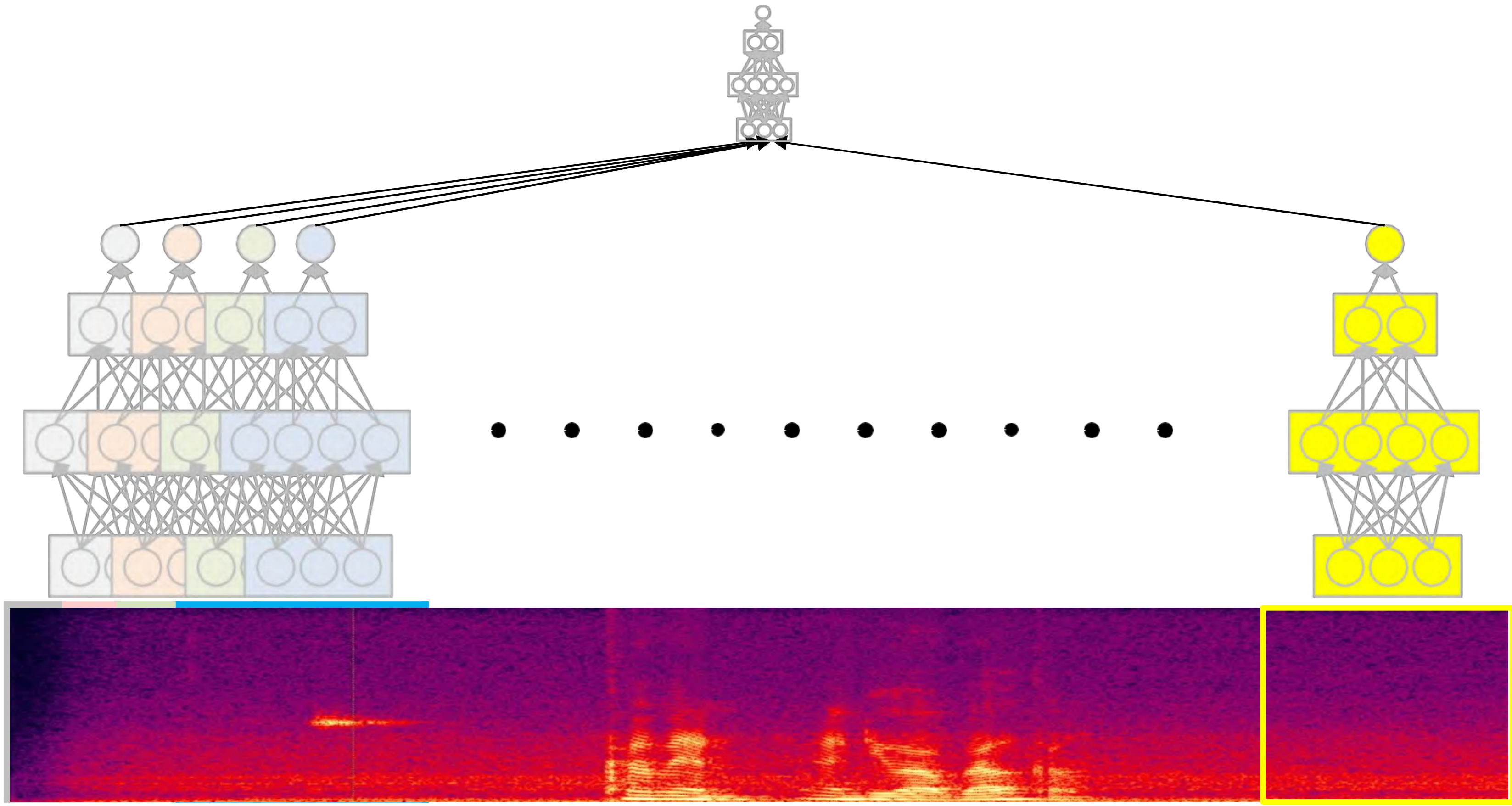

- "Does welcome occur in this recording?"
  - Maximum of all the outputs (Equivalent of Boolean OR)

# Solution: Scan



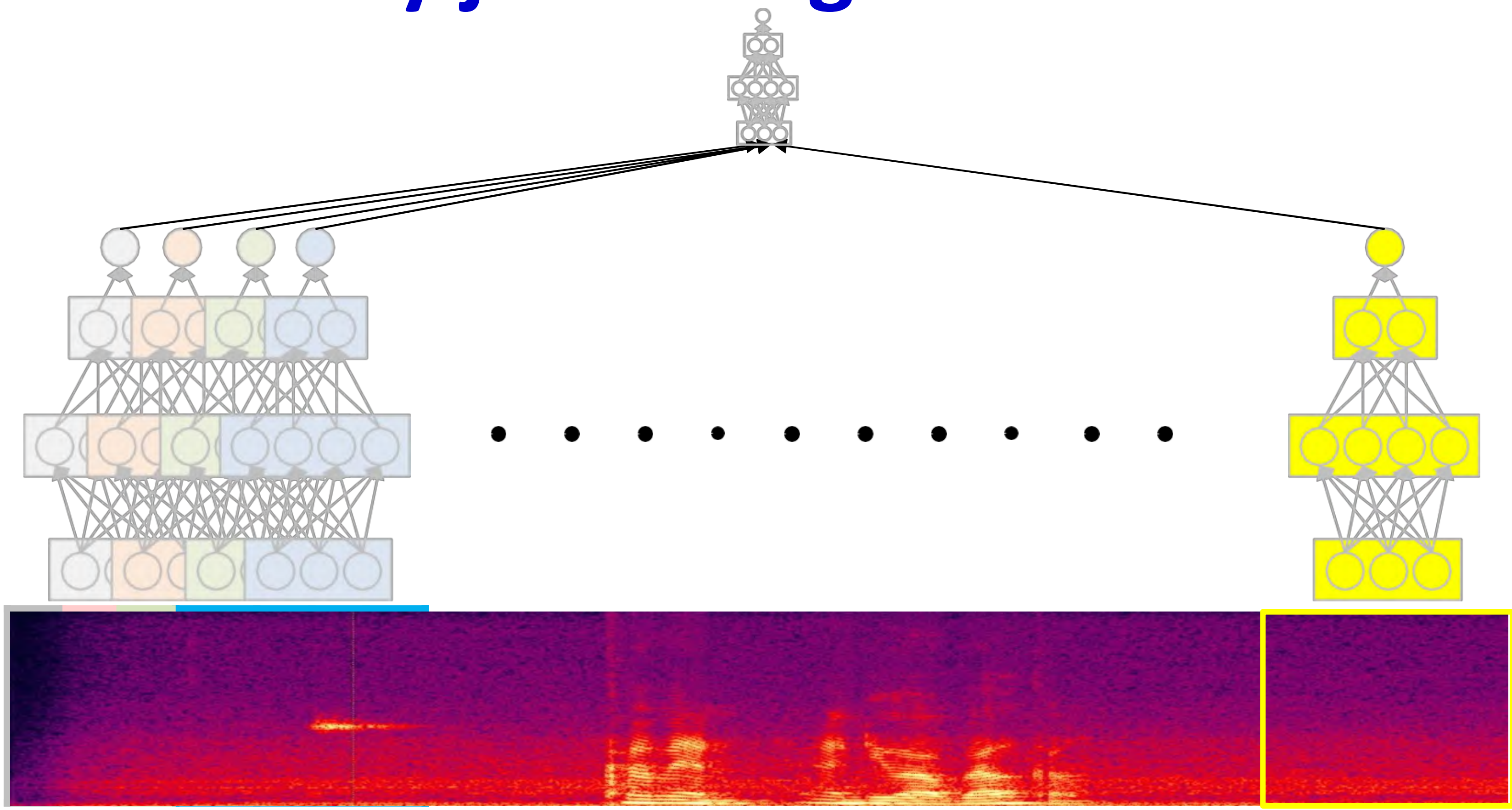

- "Does welcome occur in this recording?"
  - Maximum of all the outputs (Equivalent of Boolean OR)
  - Or a proper softmax/logistic
    - Finding a welcome in adjacent windows makes it more likely that we didn't find noise

# Solution: Scan



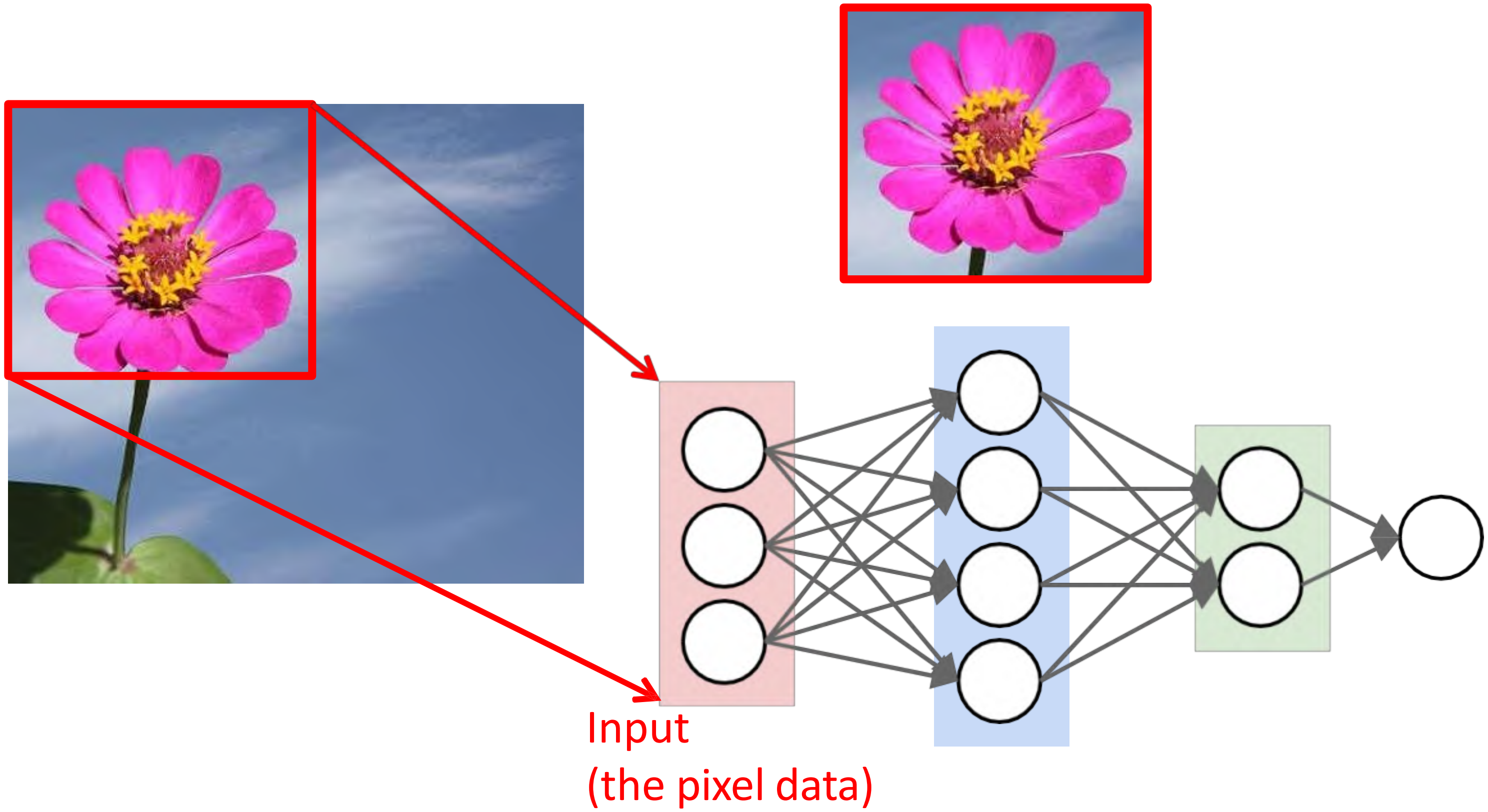
- "Does welcome occur in this recording?"
  – Maximum of all the outputs (Equivalent of Boolean OR)
  – Or a proper softmax/logistic
    - Adjacent windows can combine their evidence
  – Or even an MLP

# Its actually just one giant network



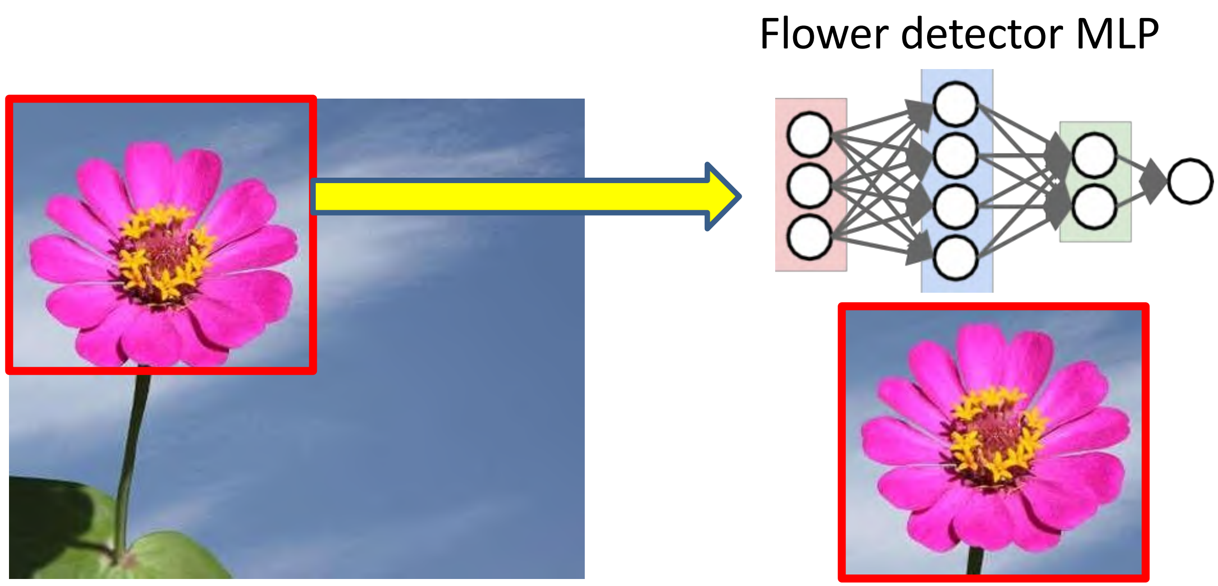
- The entire operation can be viewed as one giant network
  – With many subnetworks, one per window
  – Restriction: All subnets are identical
- The network is *shift-invariant!*

# The 2-d analogue: Does this picture have a flower?



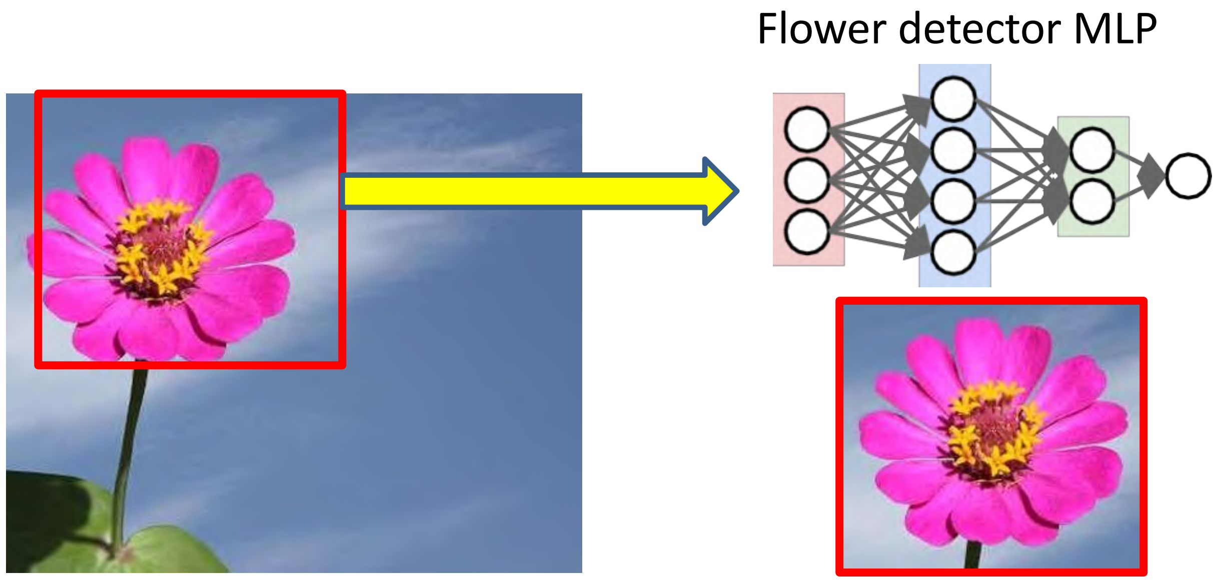

- *Scan* for the desired object
  - "Look" for the target object at each position

# Solution: Scan



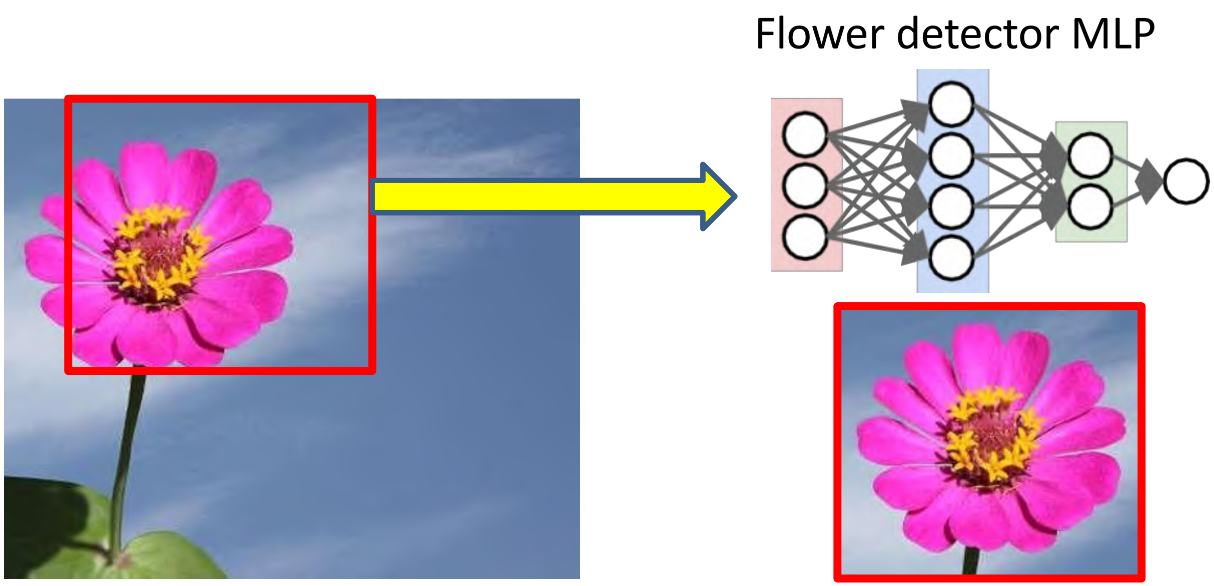

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



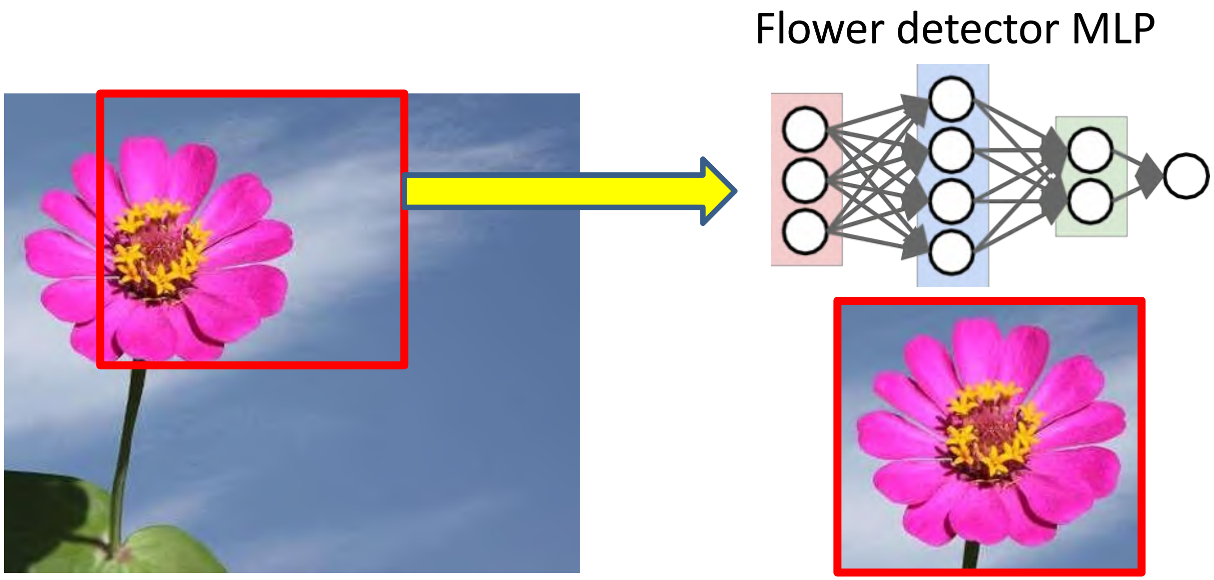

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



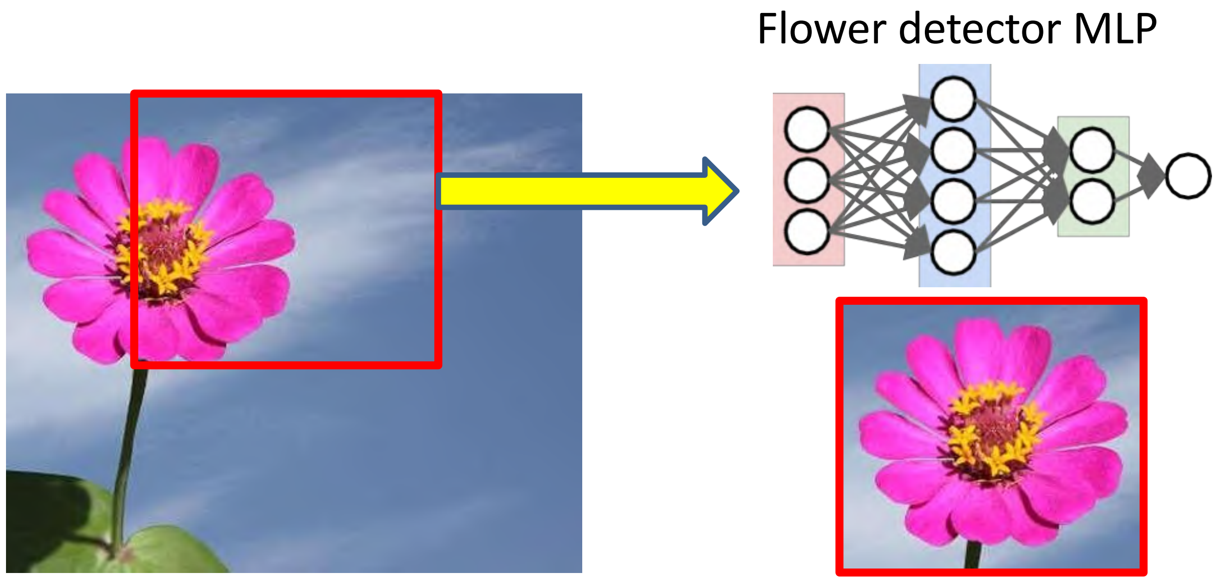

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



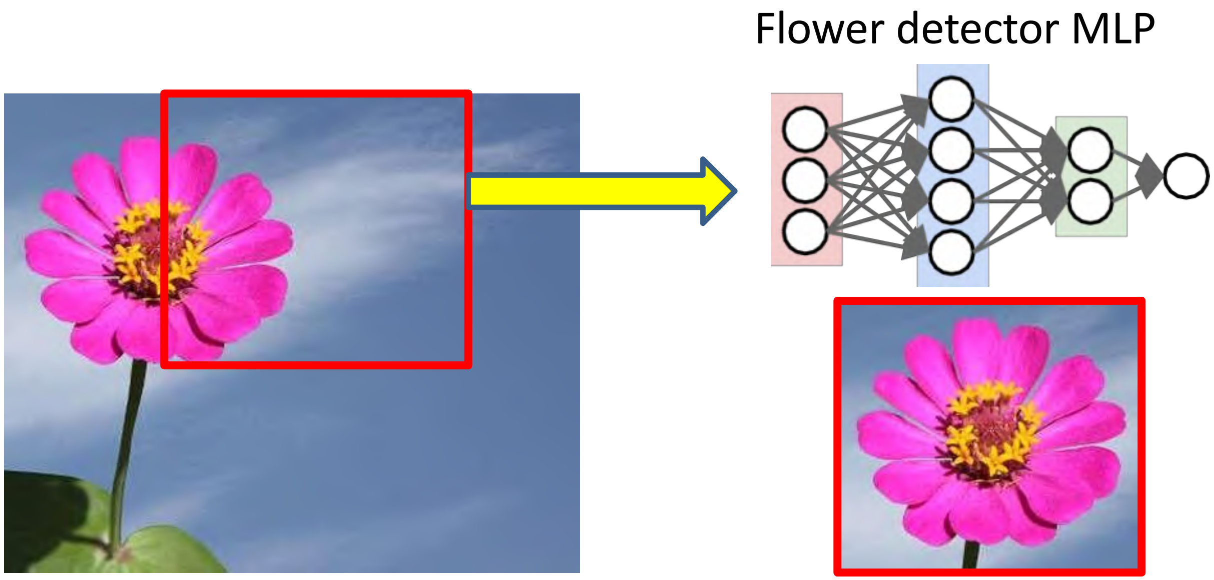

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



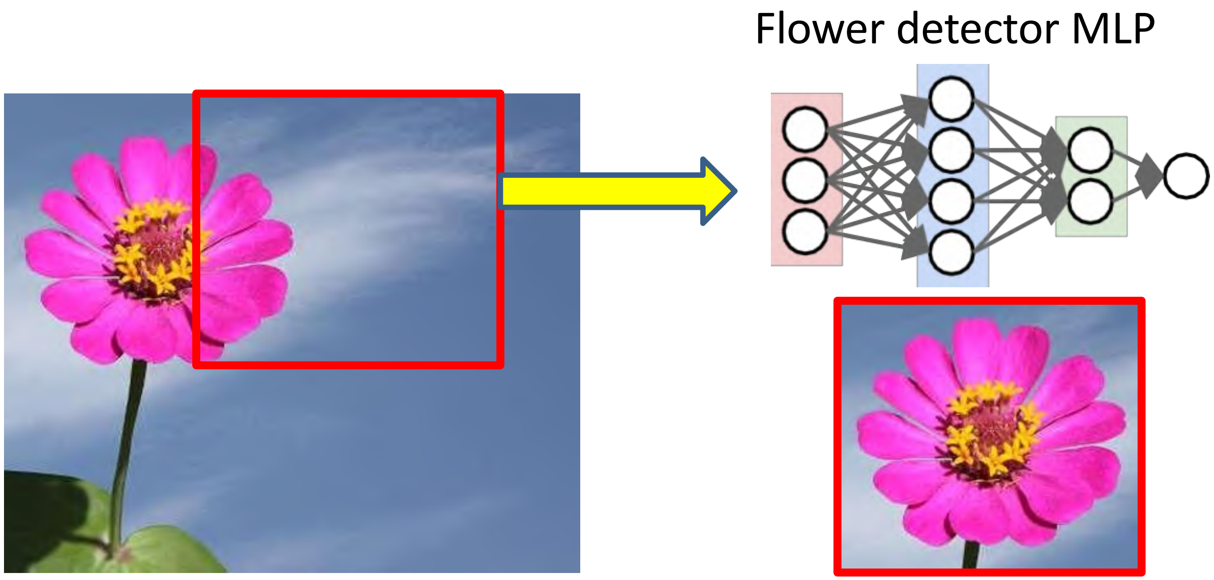

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



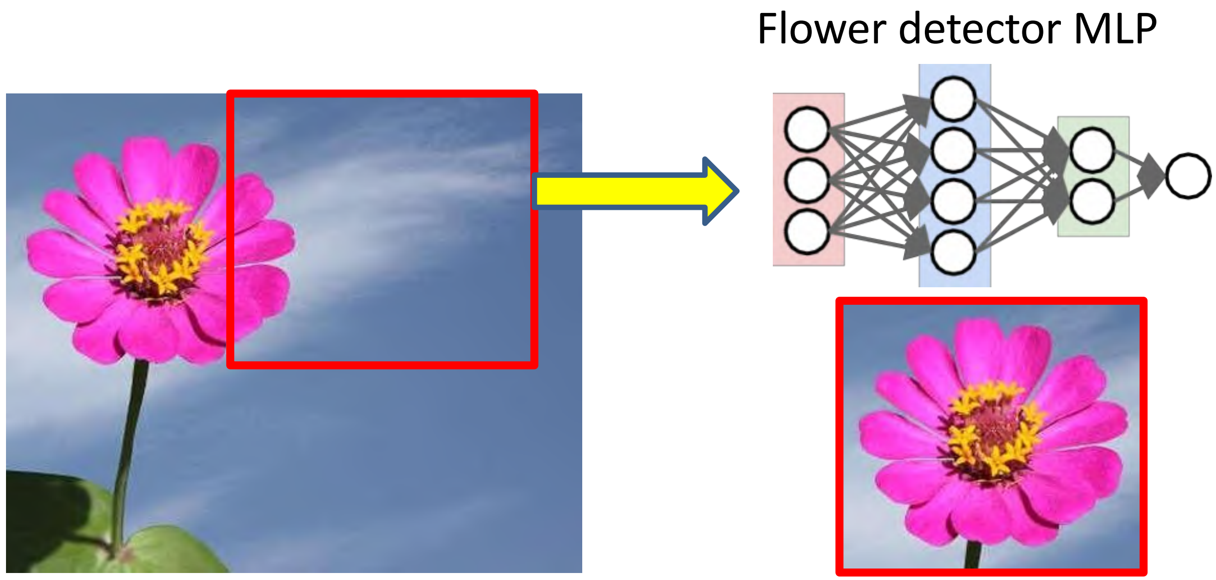

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



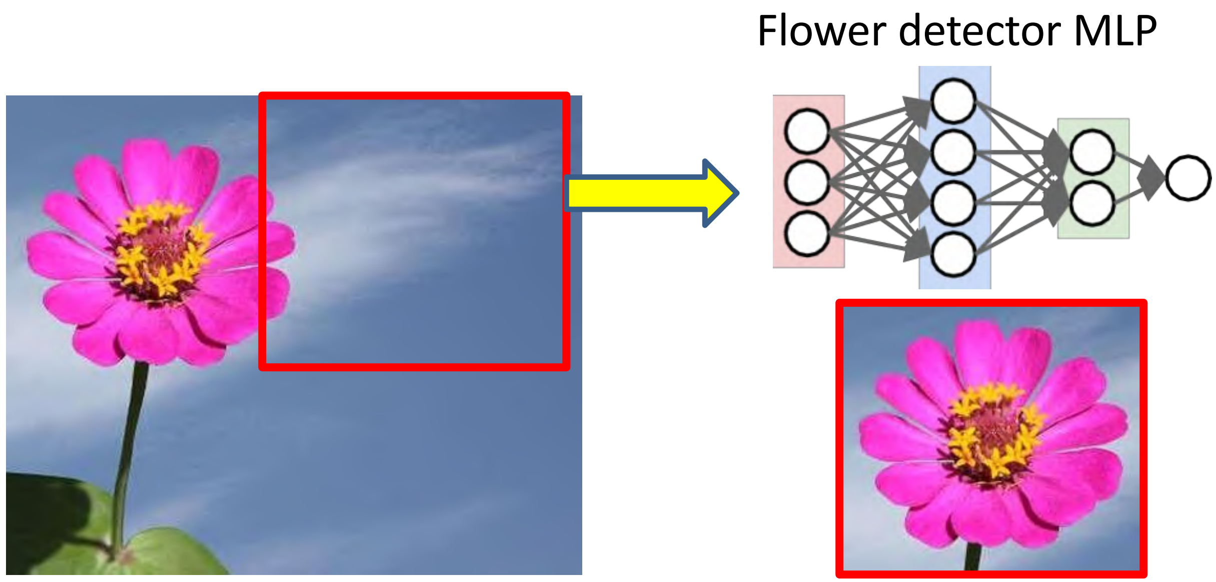

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



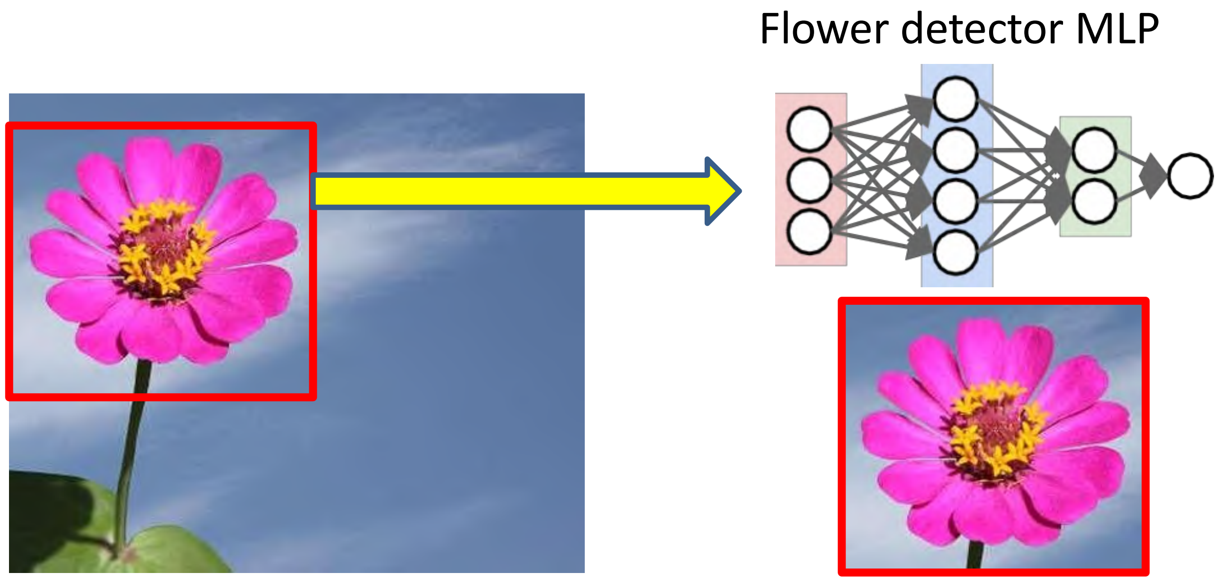

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



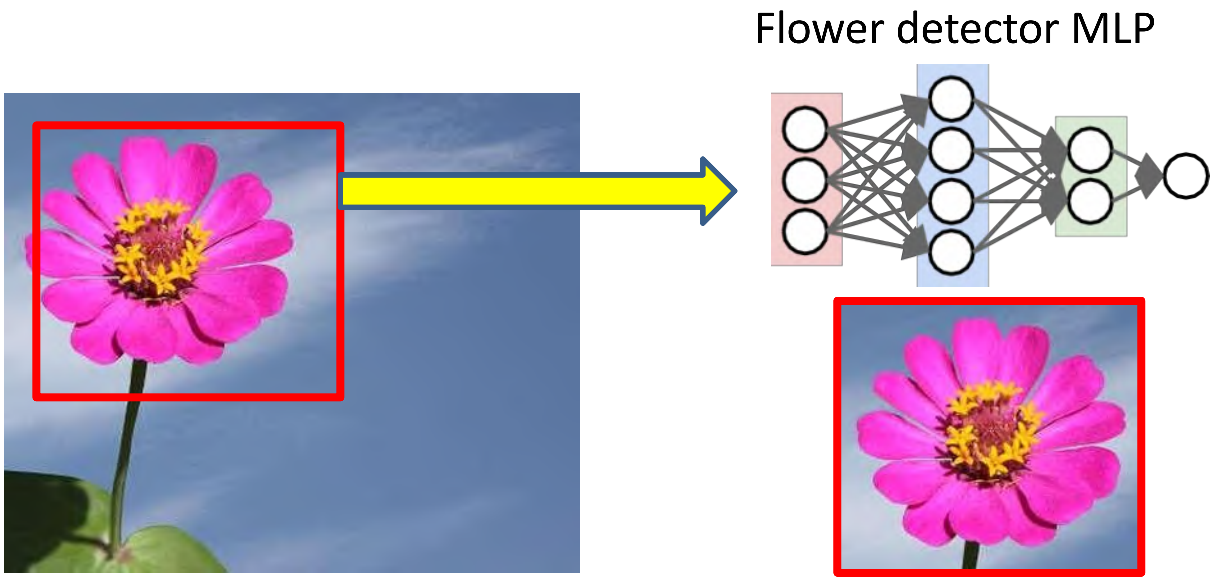

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



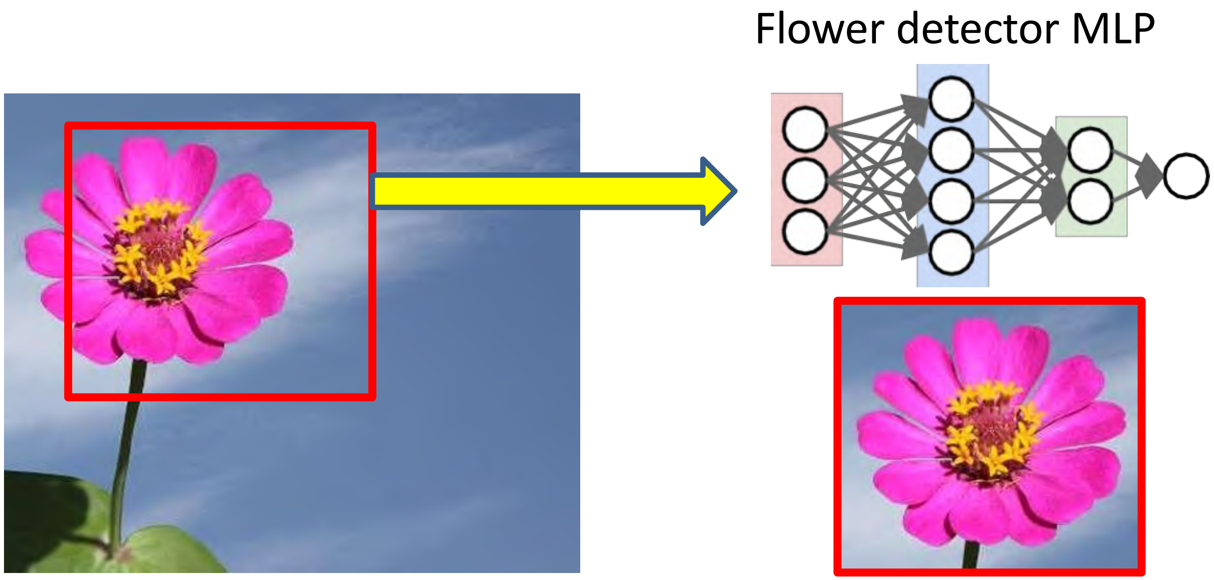

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



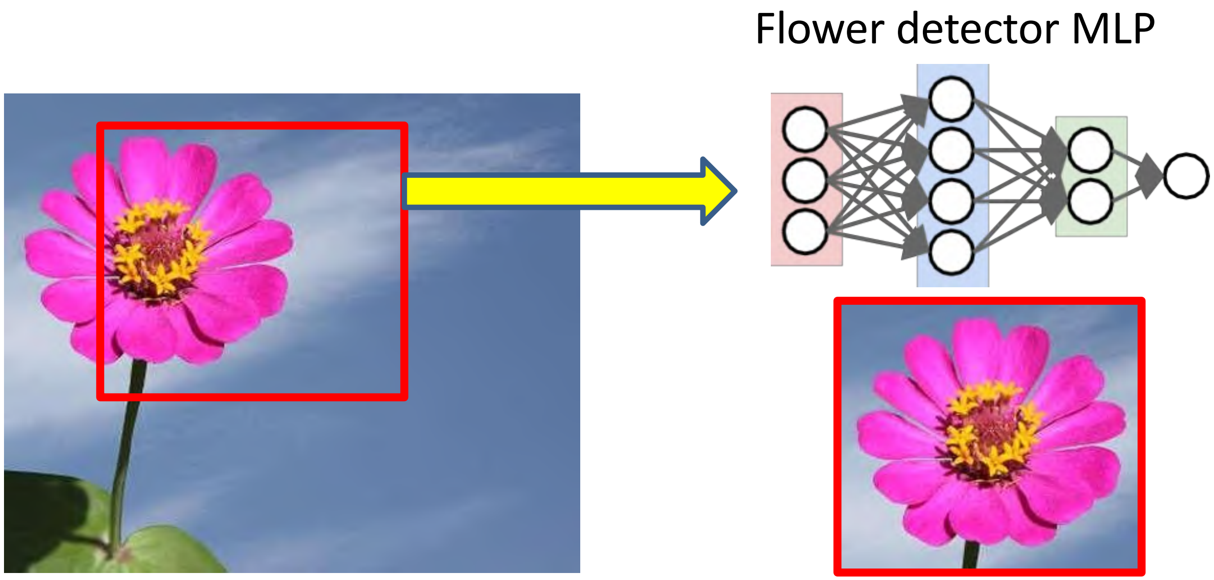

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



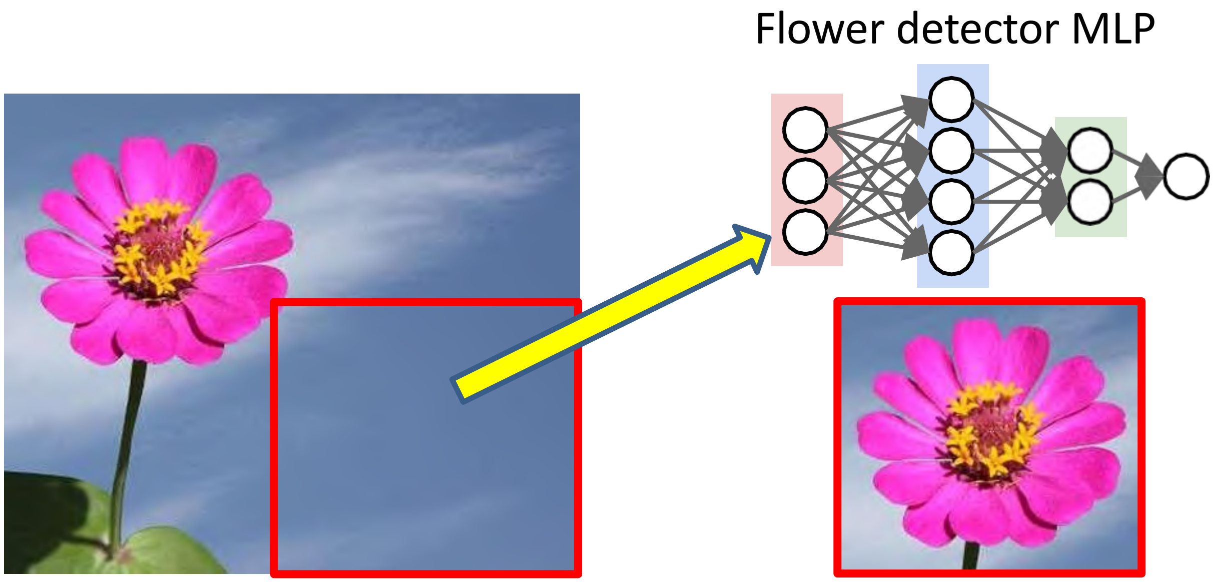

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



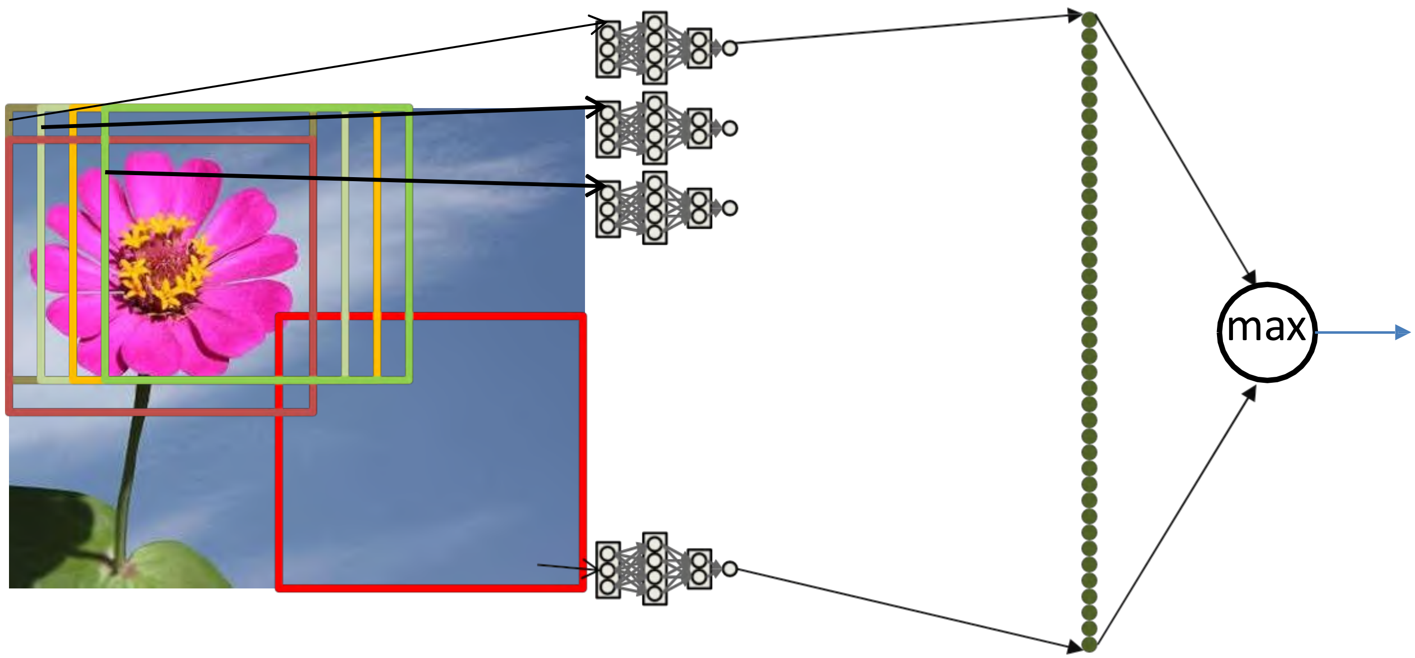

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Solution: Scan



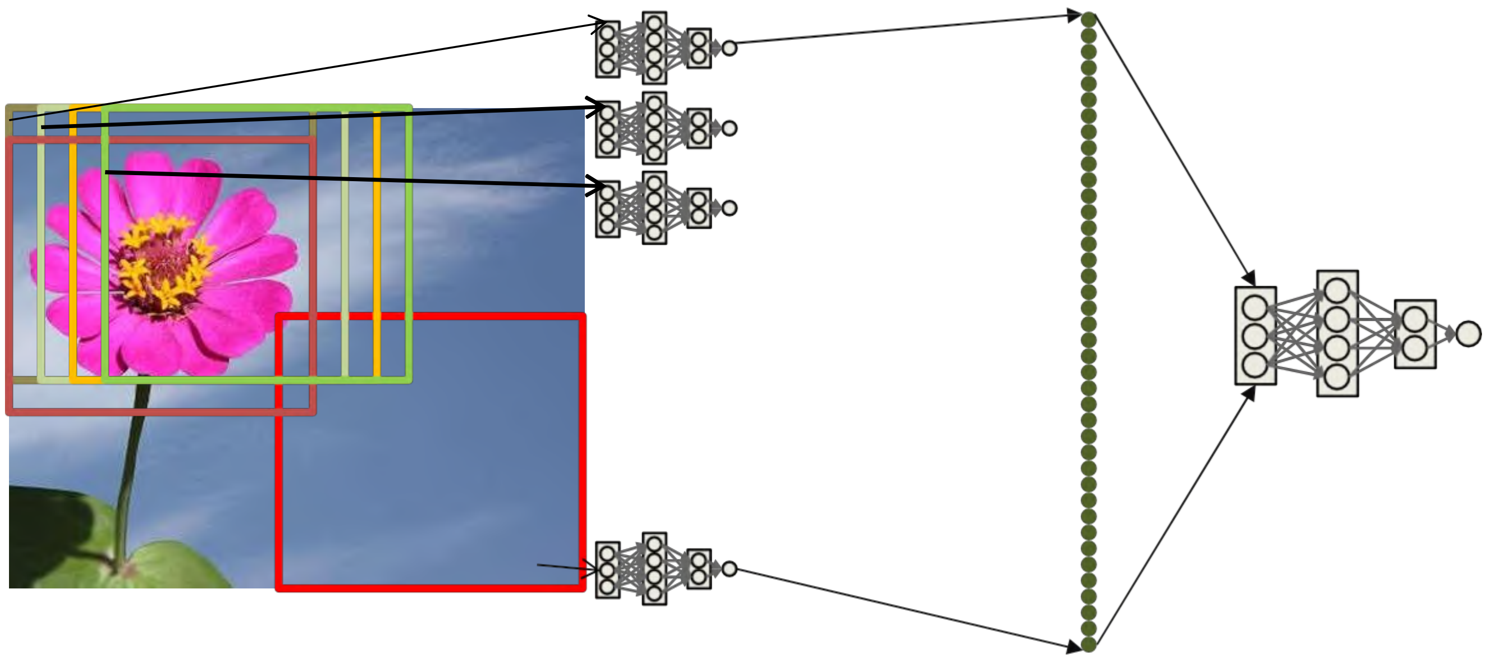

- *Scan* for the desired object
- At each location, the entire region is sent through the MLP

# Scanning the picture to find a flower



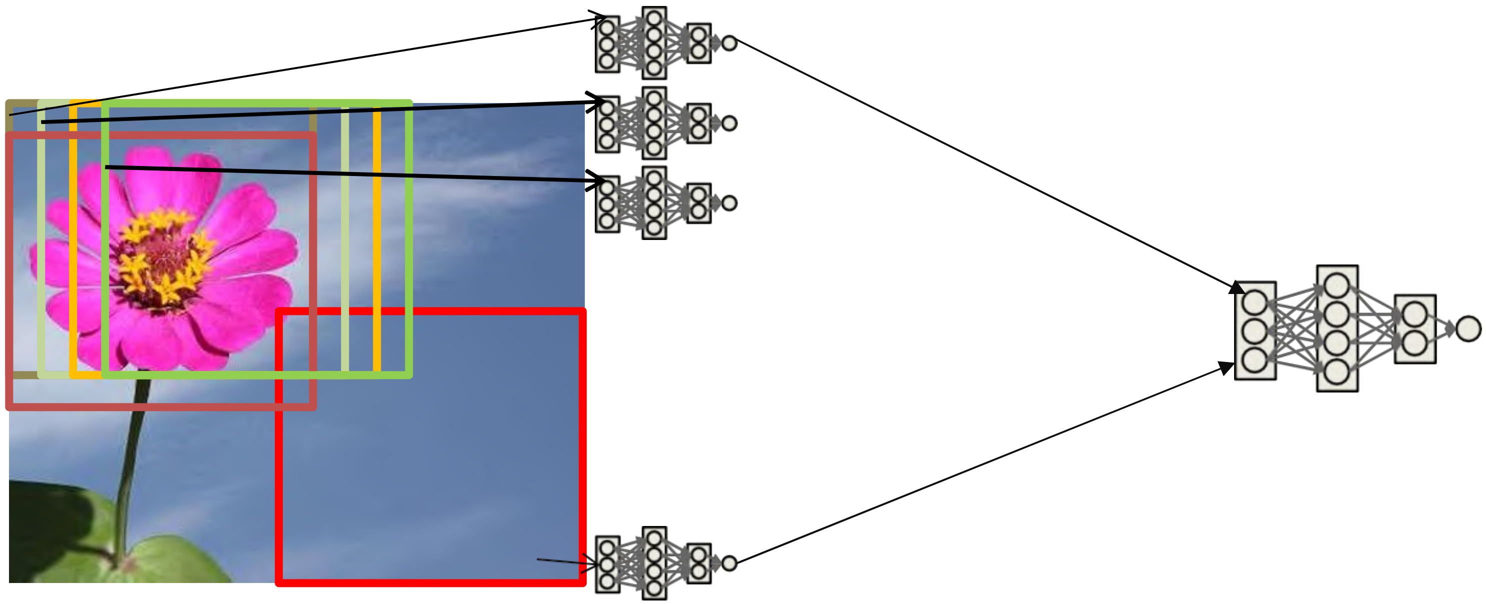

- Determine if any of the locations had a flower
  - We get one classification output per scanned location
    - Each dot in the right represents the output of the MLP when it classifies one location in the input figure
      - The score output by the MLP
  - Look at the maximum value
    - If the picture has a flower, the location with the flower will result in high output value

# Scanning the picture to find a flower



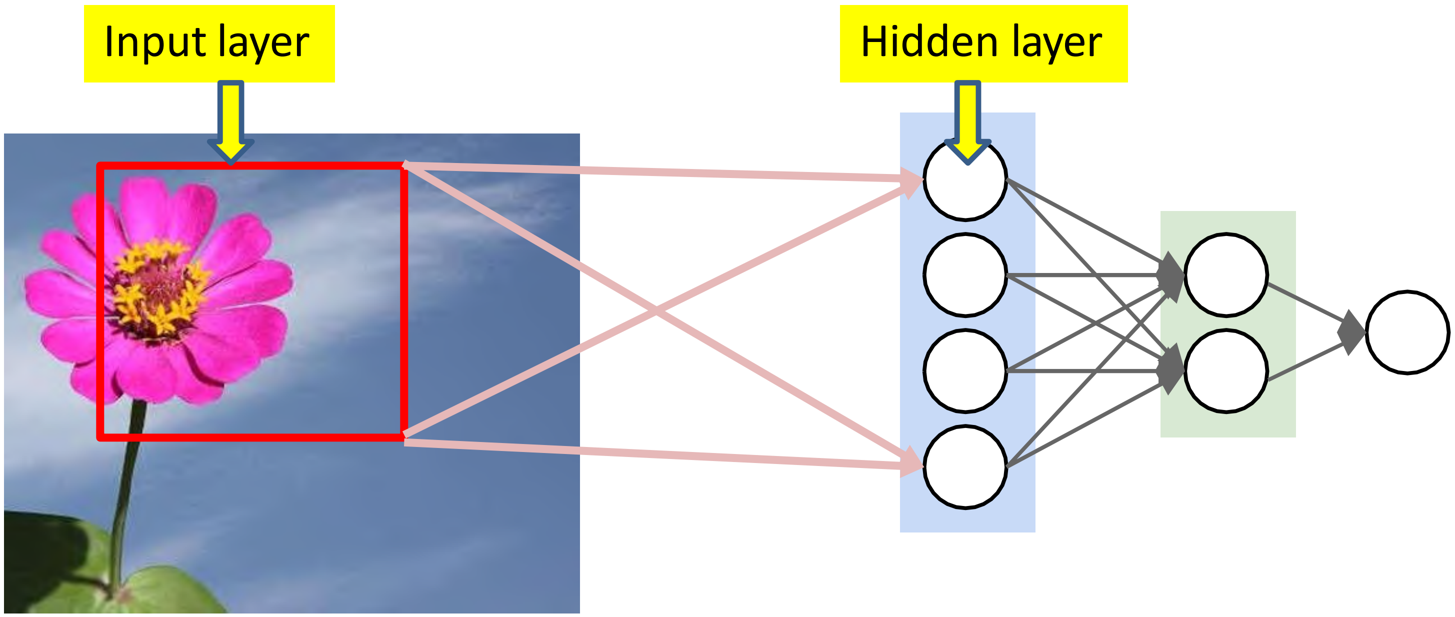
- Determine if any of the locations had a flower
  - Each dot in the right represents the output of the MLP when it classifies one location in the input figure
    - The score output by the MLP
  - Look at the maximum value
  - Or pass it through a softmax or even an MLP

# Its just a giant network with common subnets



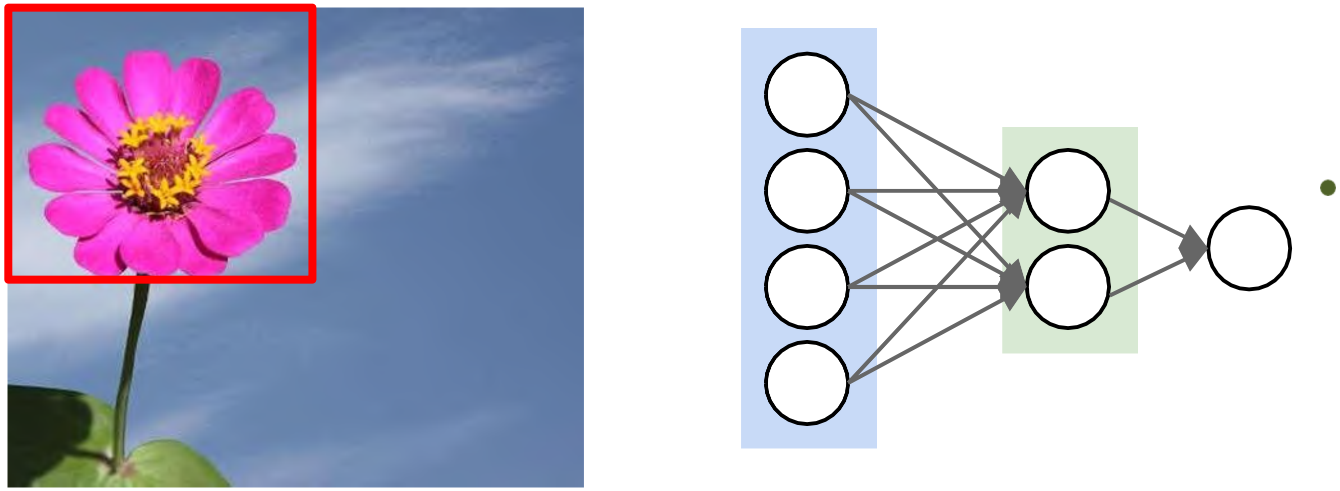
- The entire operation can be viewed as a single giant network
  - Composed of many "subnets" (one per window)
  - With one key feature: all subnets are identical
- The network is *shift invariant*.

# Scanning: A closer look



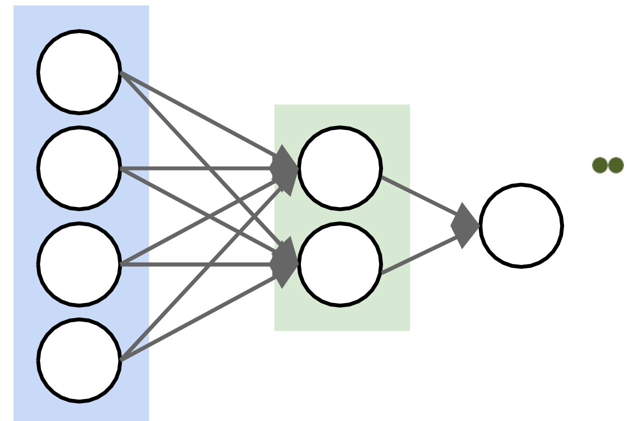

- The "input layer" is just the pixels in the image connecting to the hidden layer

# Scanning: A closer look



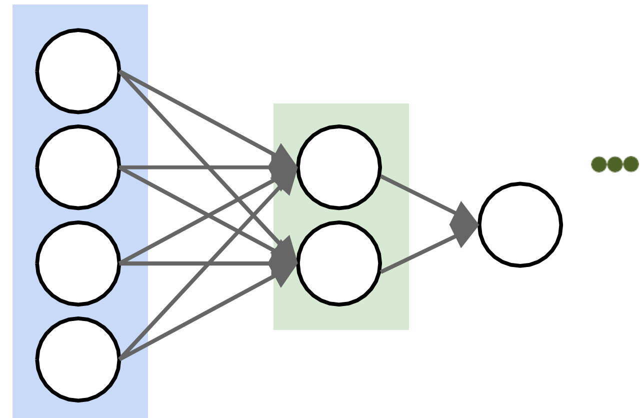
- Scanning: Analyze windows of pixels starting from top left, until the bottom right of the image
  - Produce an output for every window analyzed
  - Pass collection of outputs through a softmax

# Scanning: A closer look



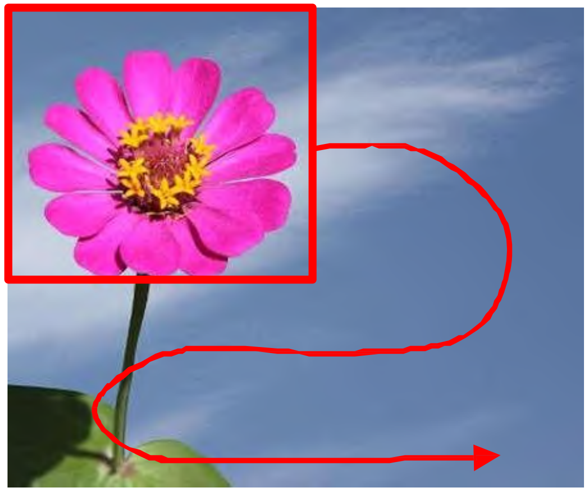
- Scanning:  Analyze windows of pixels starting from top left, until the bottom right of the image
  - Produce an output for every window analyzed
  - Pass collection of outputs through a softmax

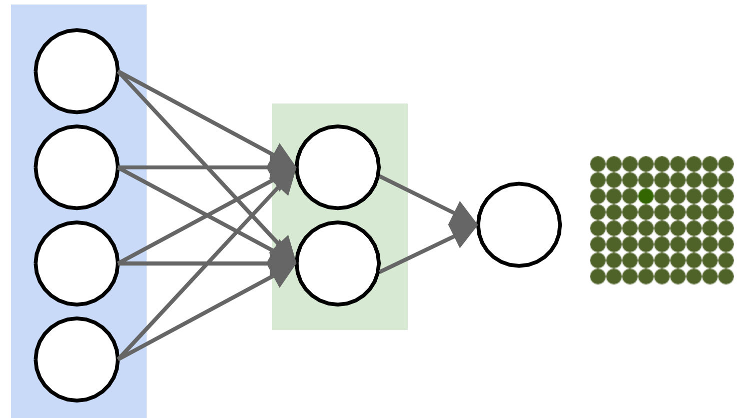
# Scanning: A closer look



- Scanning: Analyze windows of pixels starting from top left, until the bottom right of the image
  - Produce an output for every window analyzed
  - Pass collection of outputs through a softmax

# Scanning: A closer look





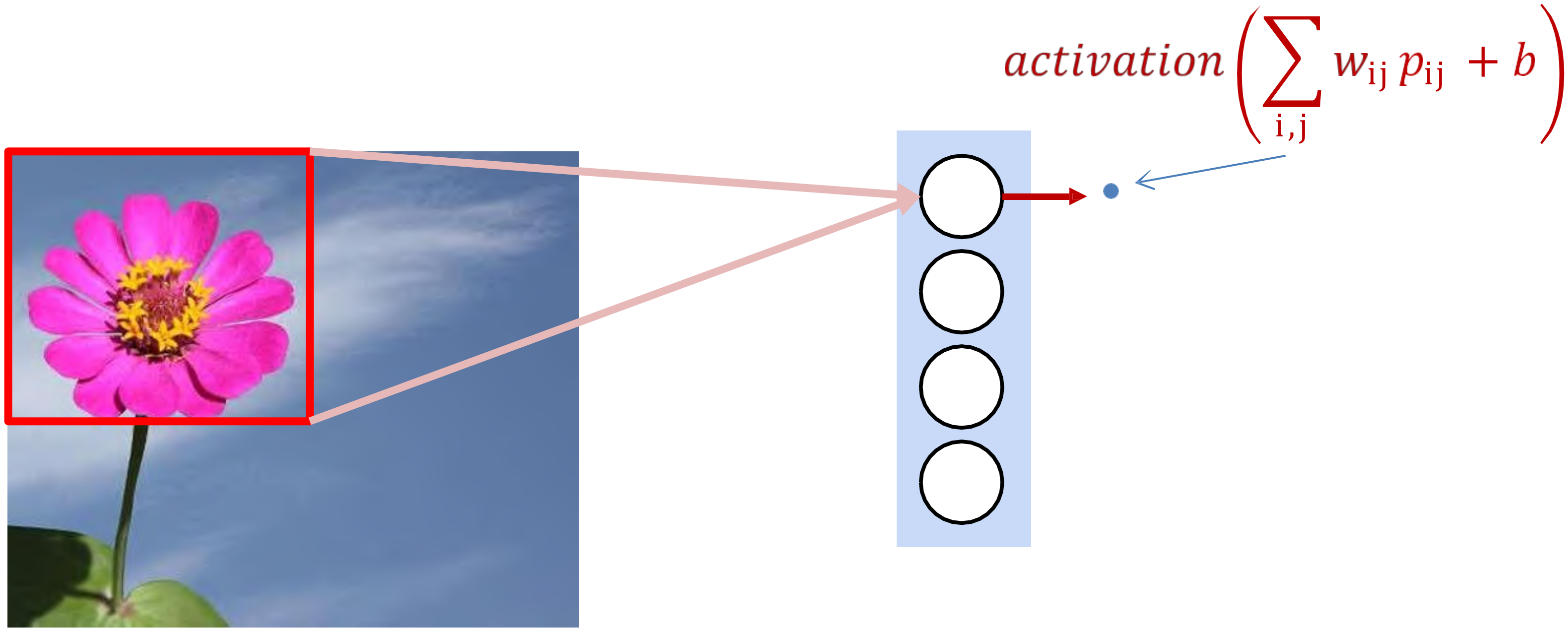
- Scanning: Analyze windows of pixels starting from top left, until the bottom right of the image
  - Produce an output for every window analyzed
  - Pass collection of outputs through a softmax

# Scanning: A closer look

$$activation\left(\sum_{i,j} w_{ij}\, p_{ij} + b\right)$$

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region

# Scanning: A closer look




- Consider a single neuron in the first layer
  – At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  – "Scanning" the image with just the neuron
  – We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region
- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - “Scanning” the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

- Consider a single neuron in the first layer
  – At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  – "Scanning" the image with just the neuron
  – We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

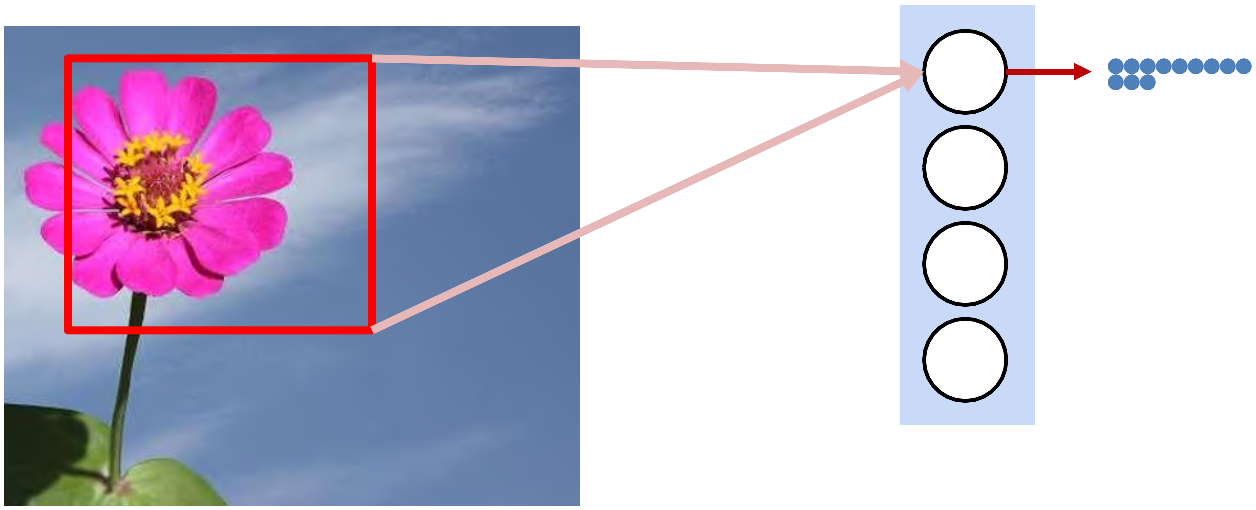
- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a “window” of the picture at that location, as part of the classification for *that* region
- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - “Scanning” the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

- Consider a single neuron in the first layer
  – At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  – "Scanning" the image with just the neuron
  – We could arrange the outputs in correspondence to the original picture

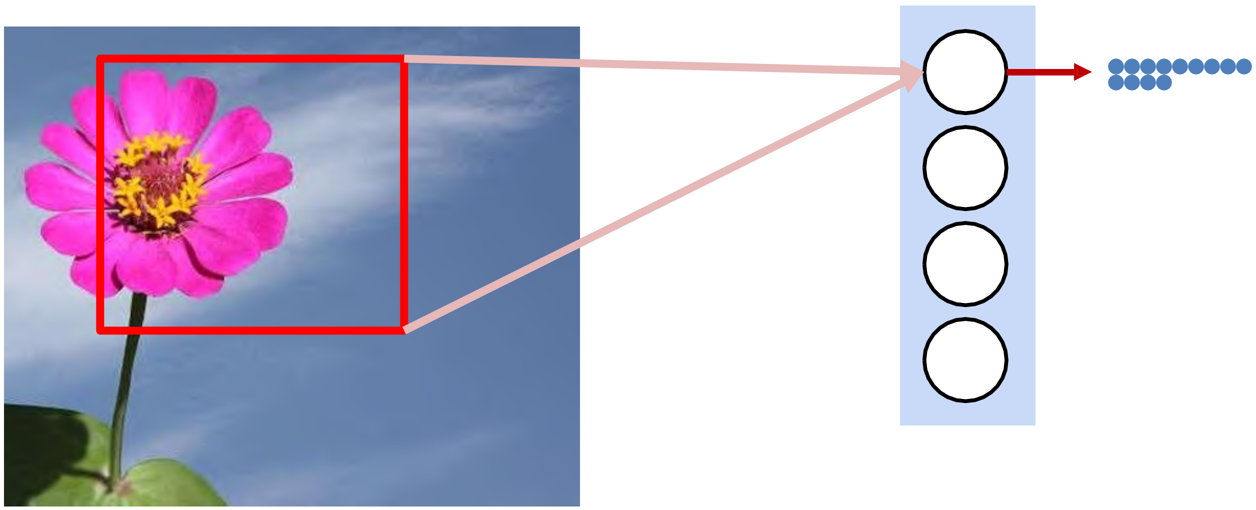
# Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look



- Consider a single neuron in the first layer
  - At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  - "Scanning" the image with just the neuron
  - We could arrange the outputs in correspondence to the original picture

# Scanning: A closer look

- Consider a single neuron in the first layer
  – At each position of the box, the neuron is evaluating a "window" of the picture at that location, as part of the classification for *that* region

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons
  – "Scanning" the image with just the neuron
  – We could arrange the outputs in correspondence to the original picture

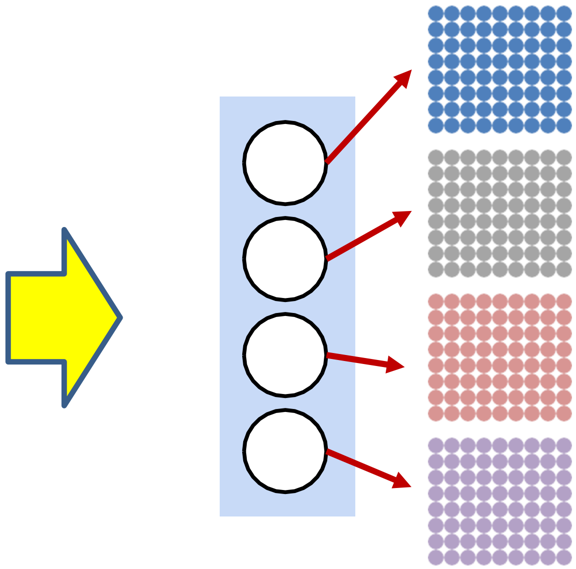
BR2

# Scanning: A closer look

- Let us compute the output of the first neuron for *all* the windows in the picture before computing the rest of the neurons

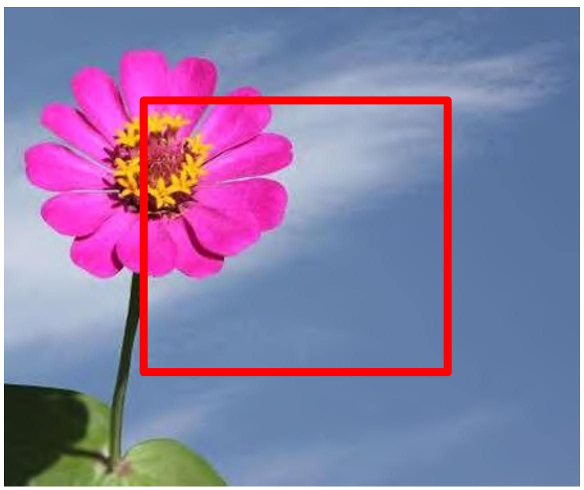
- Eventually, we can arrange the outputs from the response at the scanned positions into a rectangle that's proportional in size to the original picture

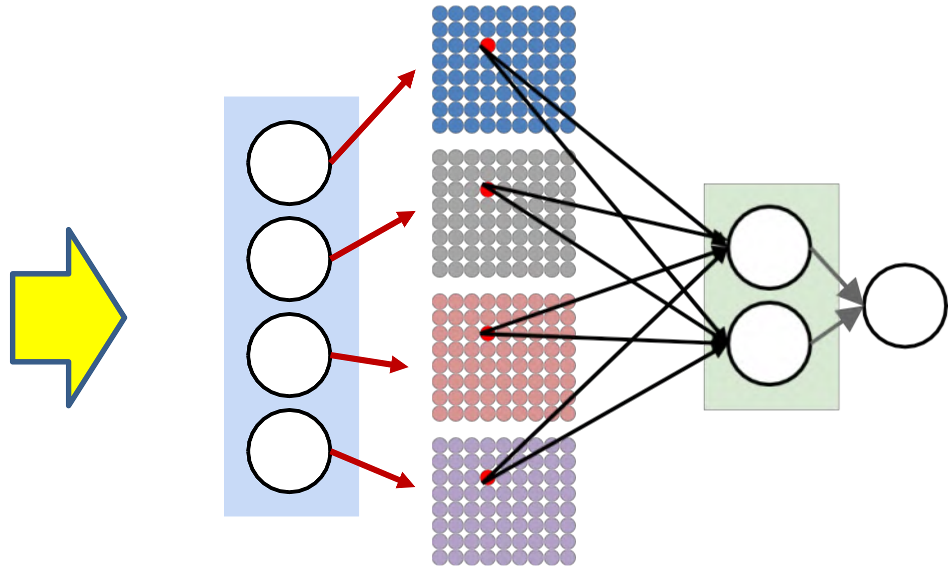
# Scanning: A closer look



- We can repeat the process for each of the first-layer neurons
  - “Scan” the input with the neuron
  - Arrange the neuron’s outputs from the scanned positions according to their positions in the original image

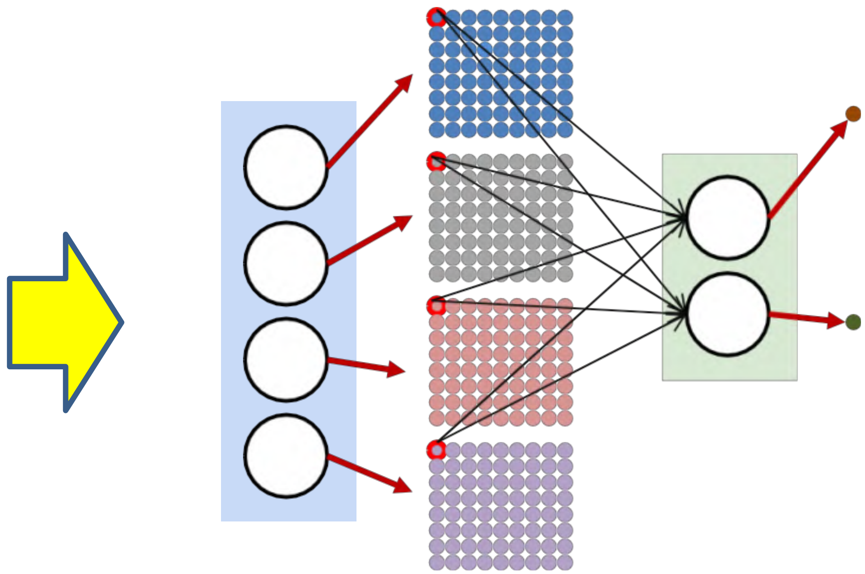
# Scanning: A closer look





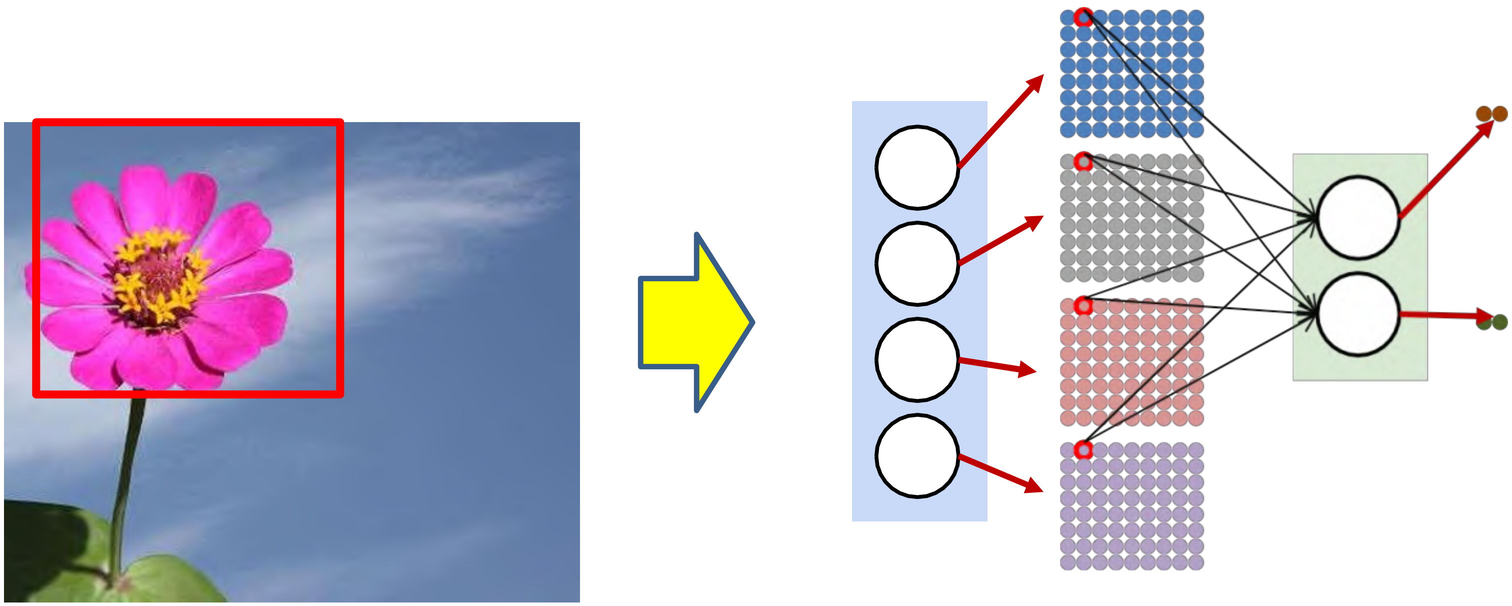
- To classify a specific “window” in the image, we send the first level activations from the positions corresponding to that position to the next layer

# Scanning: A closer look



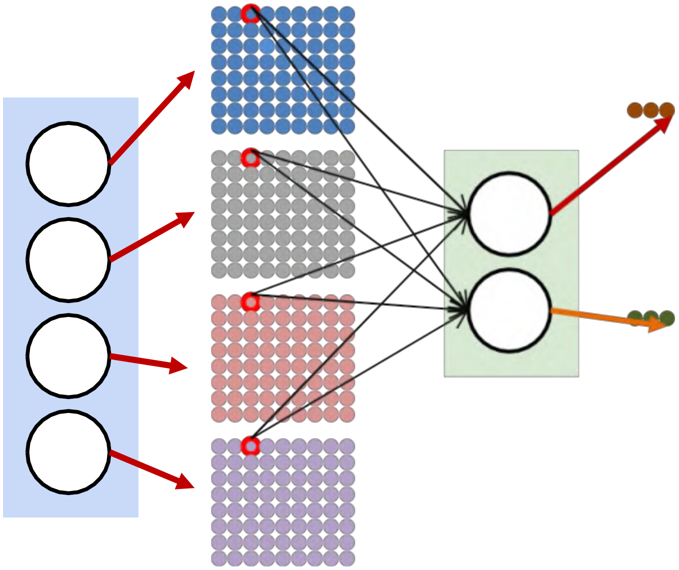
- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

# Scanning: A closer look



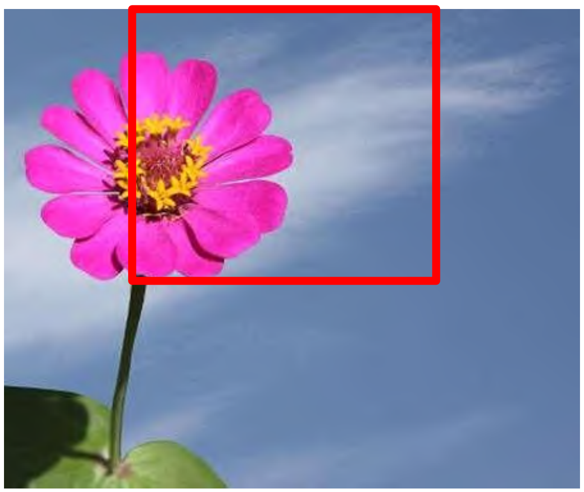
- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

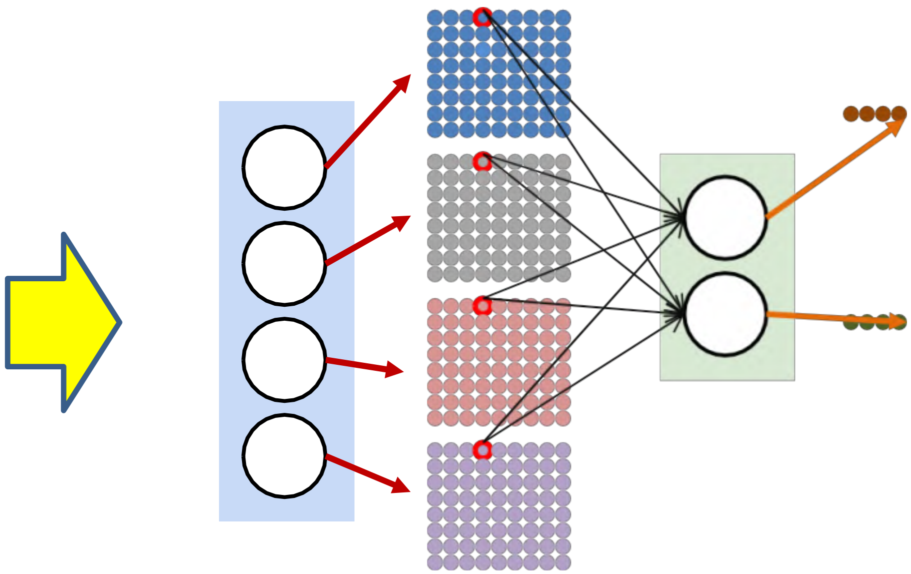
# Scanning: A closer look



- We can recurse the logic
  - The second level neurons too can “**scan**” the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* “maps”
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

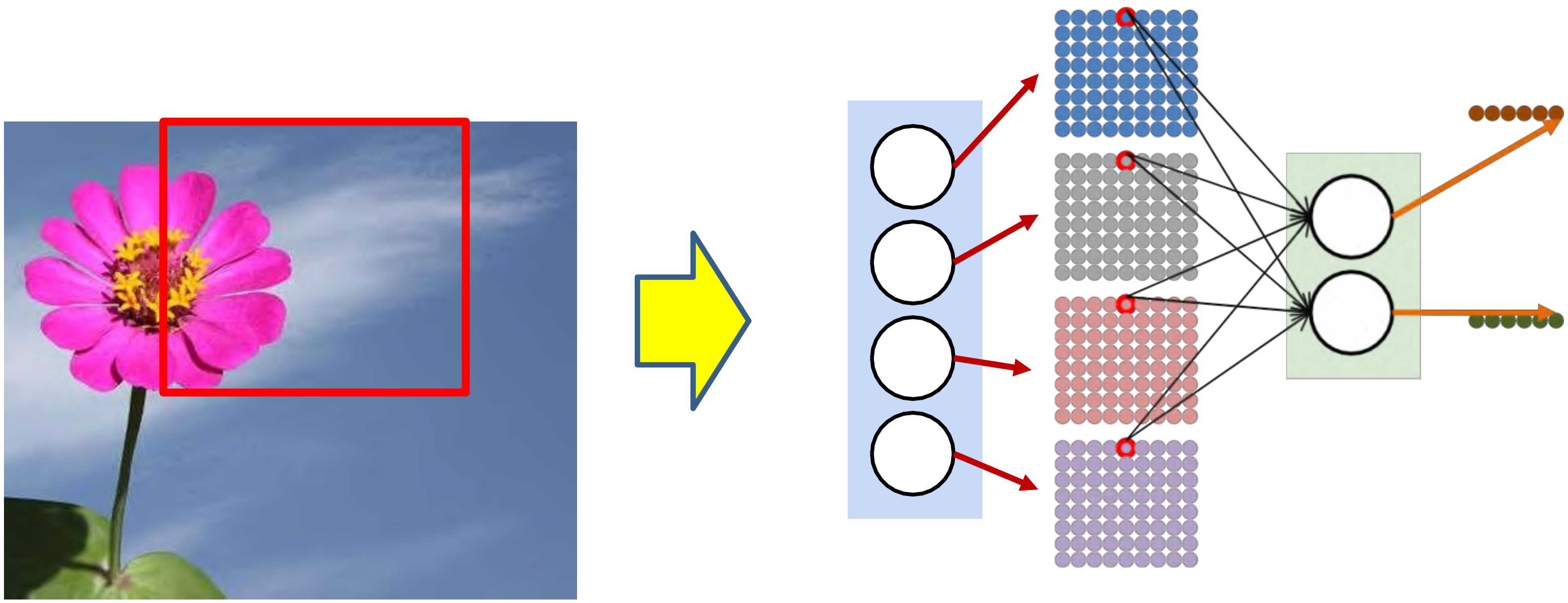
# Scanning: A closer look





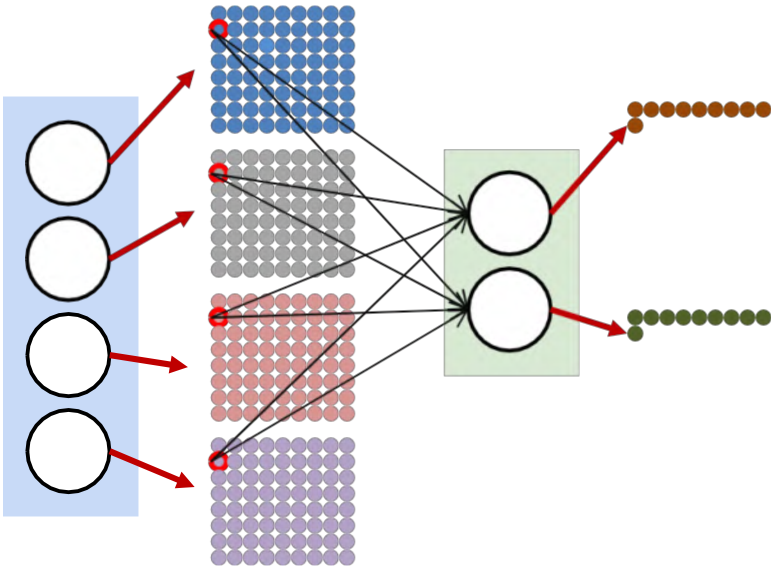
- We can recurse the logic
  - The second level neurons too can “**scan**” the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* “maps”
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

# Scanning: A closer look



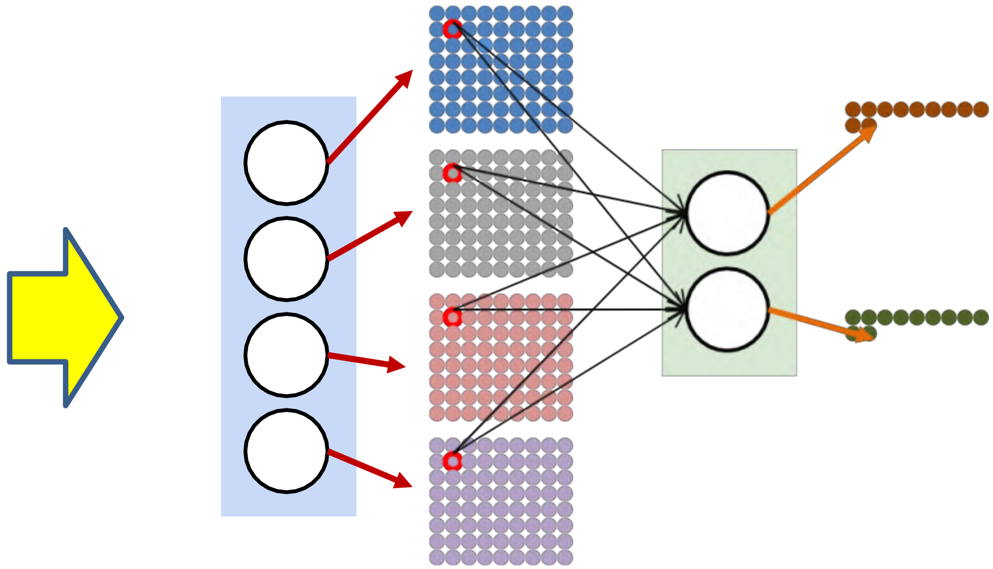
- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

# Scanning: A closer look



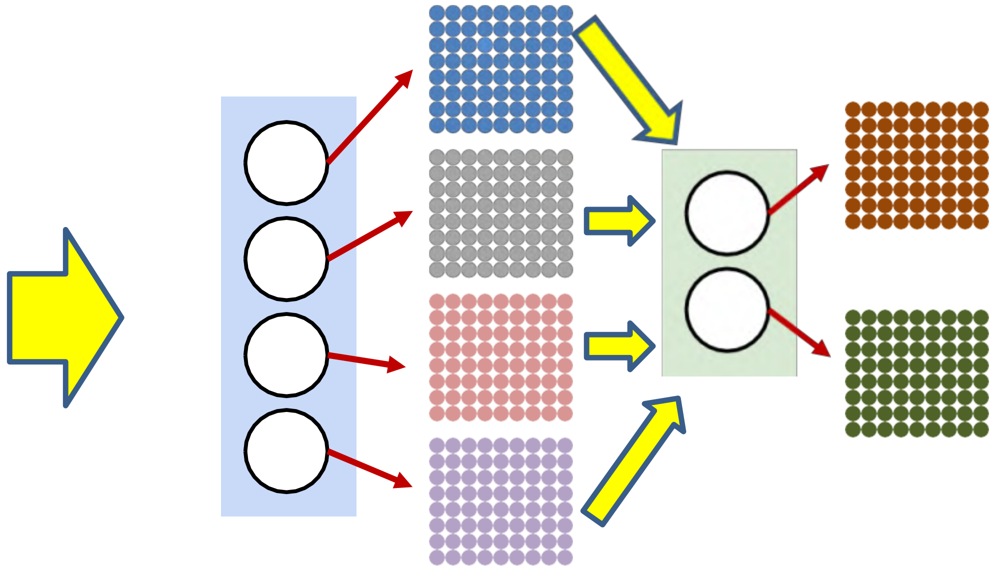
- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

# Scanning: A closer look



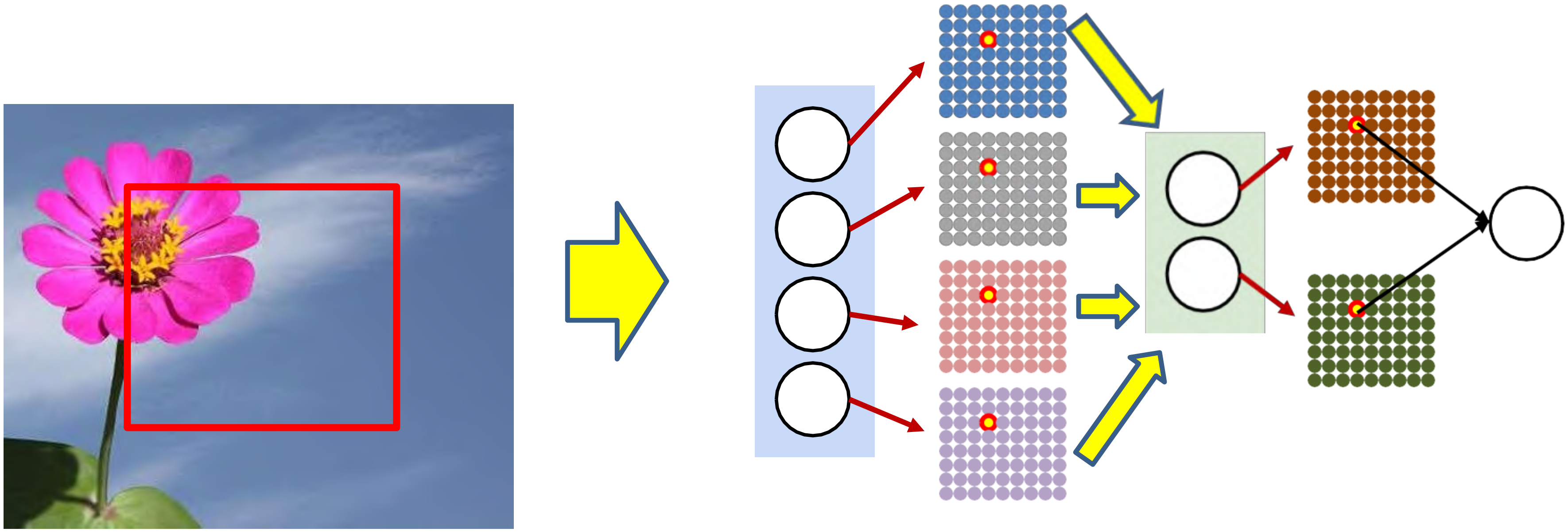
- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

# Scanning: A closer look



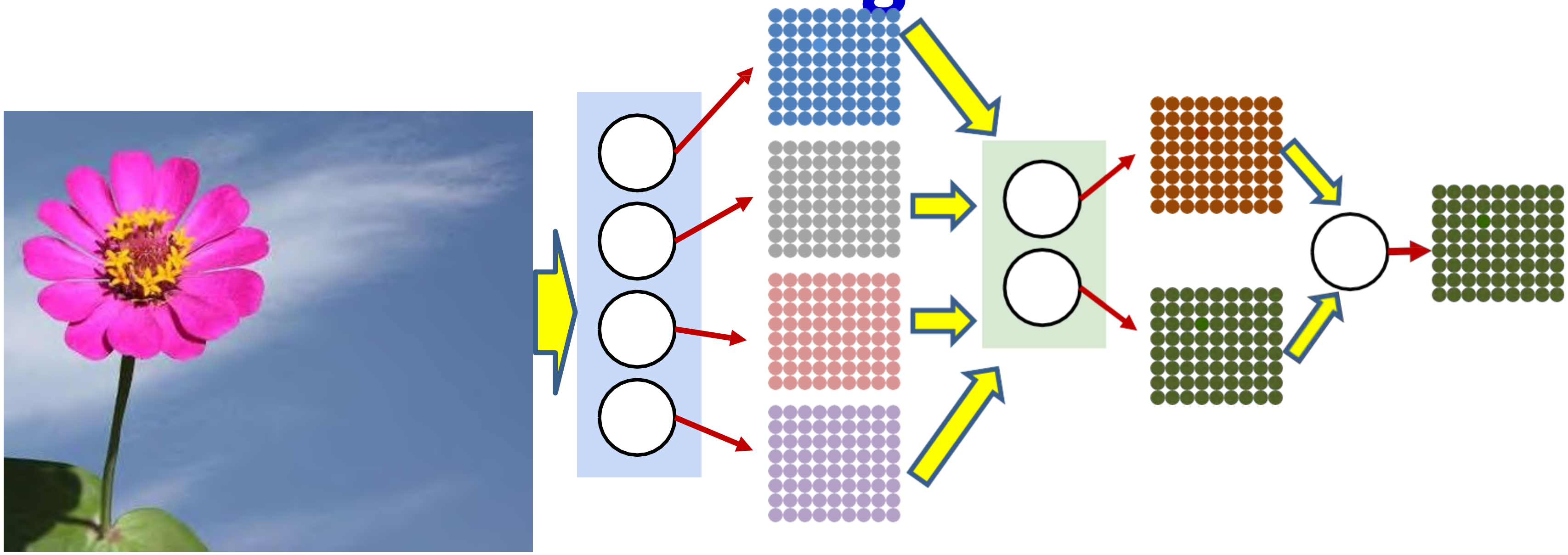
- We can recurse the logic
  - The second level neurons too can "**scan**" the rectangular outputs of the first-level neurons before computing subsequent layers
  - (Un)like the first level, they must jointly scan *multiple* "maps"
    - Each location in the output of the second level neuron considers the corresponding locations from the output maps of all the first-level neurons

# Scanning: A closer look



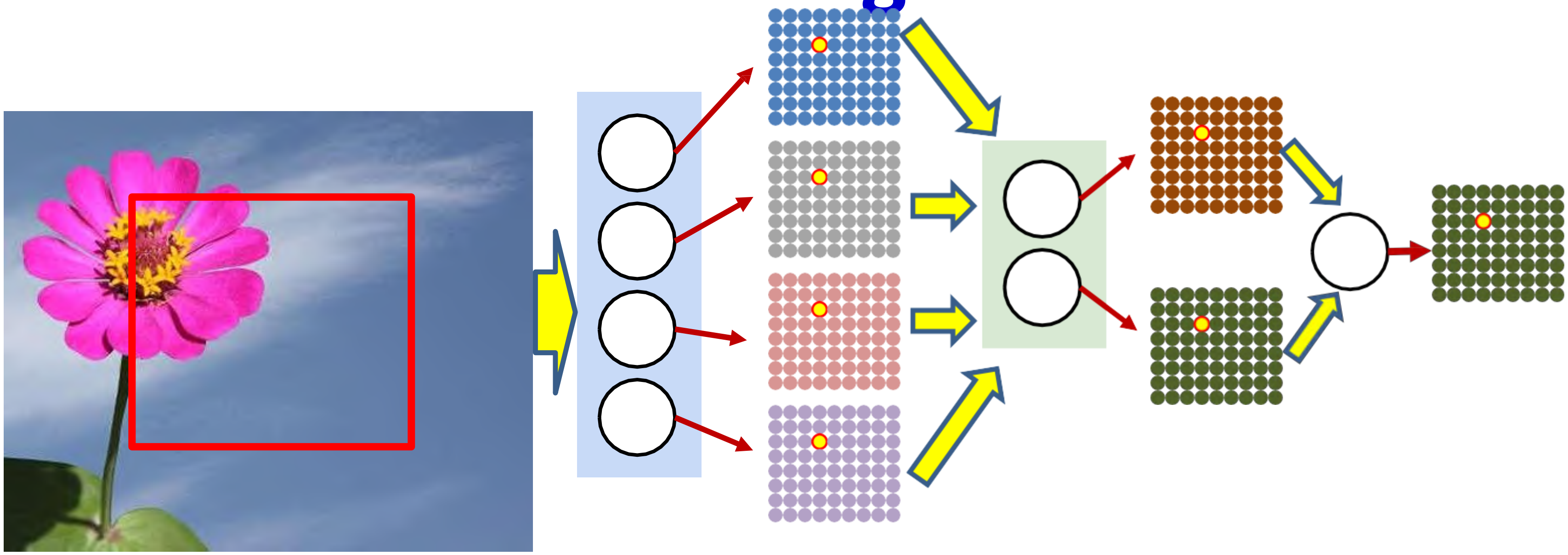
- To detect a picture ***at any location*** in the original image, the output layer must consider the corresponding outputs of the last hidden layer

# Detecting a picture anywhere in the image?



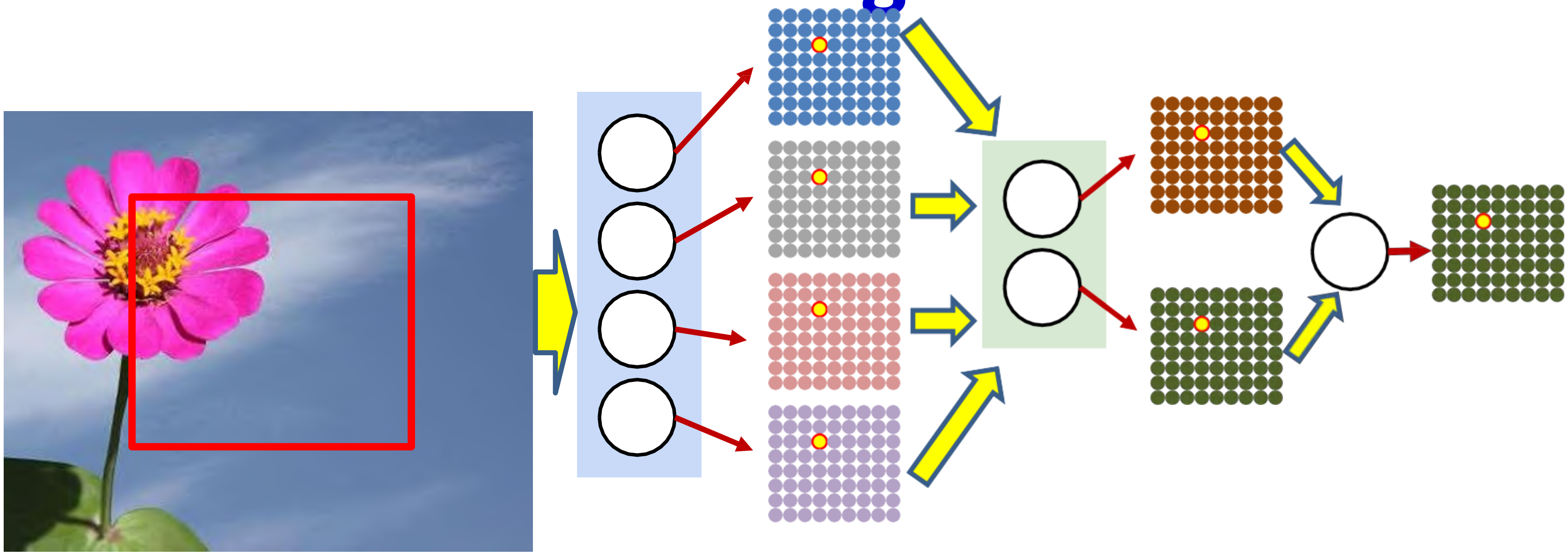
- Recursing the logic, we can create a map for the neurons in the next layer as well
  - The map is a flower detector for each location of the original image

# Detecting a picture anywhere in the image?



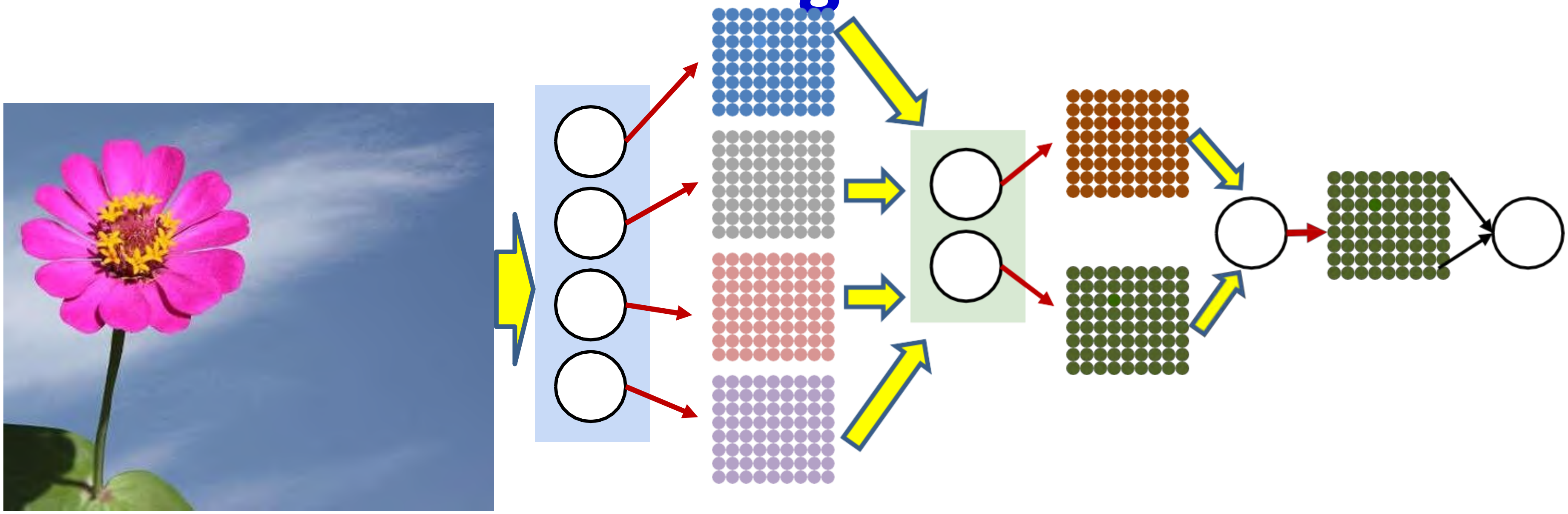
- To detect a picture ***at any location*** in the original image, only need to consider the corresponding location of the output map

# Detecting a picture anywhere in the image?



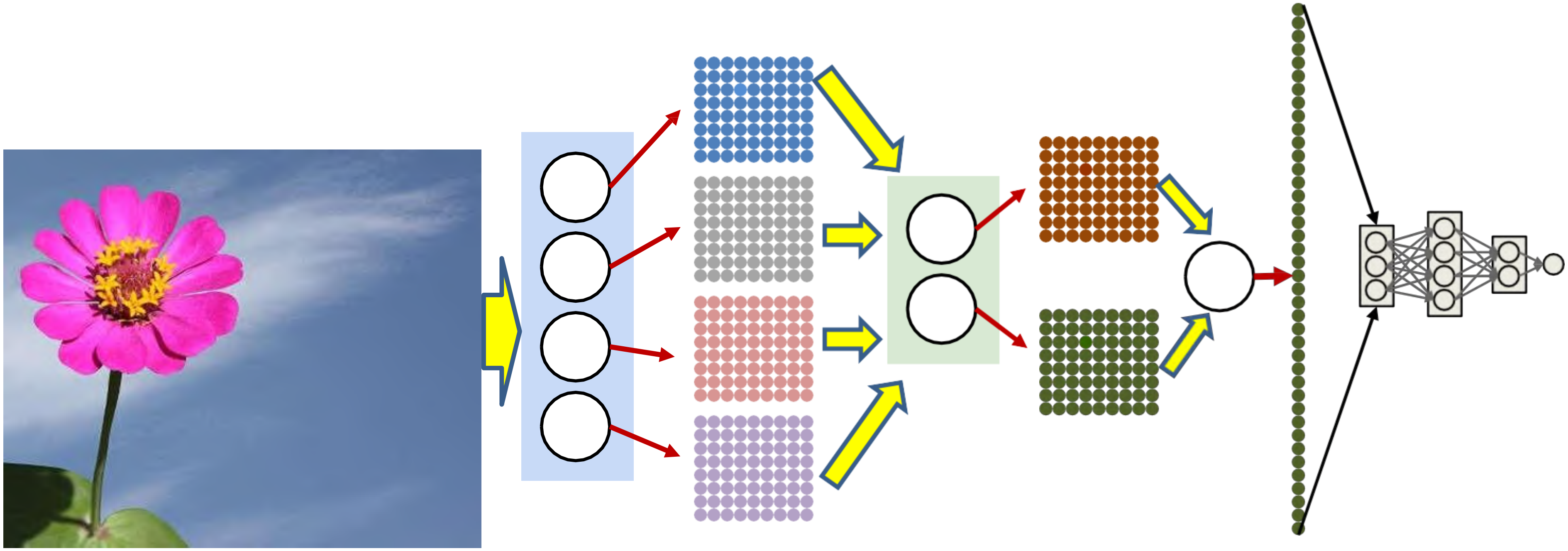
- To detect a picture ***at any location*** in the original image, only need to consider the corresponding location of the output map
- Actual problem? Is there a flower in the image
  - Not "detect the location of a flower"

# Detecting a picture anywhere in the image?



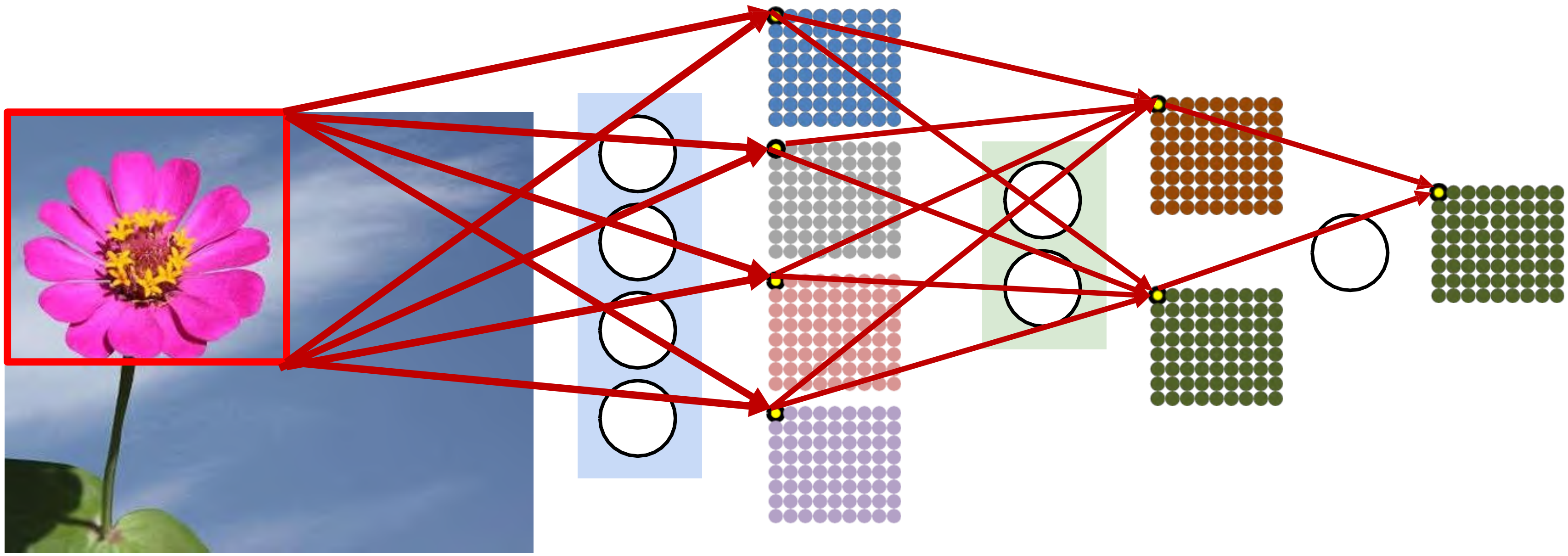
- Is there a flower in the picture?
- The entire output map can be sent into a final "max" to detect a flower in the full picture
  - Or a softmax, or a full MLP…

# Detecting a picture in the image



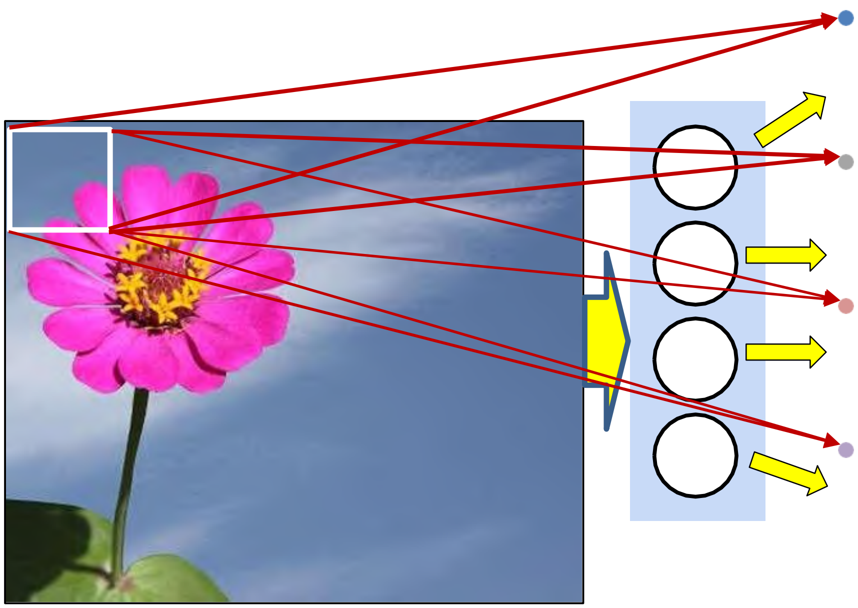
- Redrawing the final layer
  - "Flatten" the output of the neurons into a single block, since the arrangement is no longer important
  - Pass that through a max/softmax/MLP

# The behavior of the layers



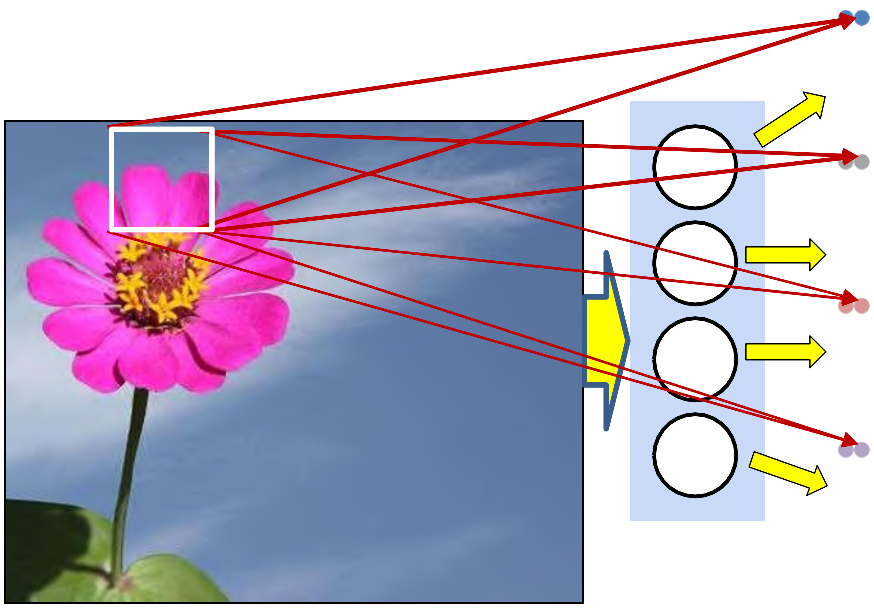
- The first layer neurons “look” at the entire “window” to extract window-level features
  - Subsequent layers only perform classification over these window-level features
- **The first layer neurons is responsible for evaluating the entire *window* of pixels**
  - **Subsequent layers only look at a *single* pixel in their input maps**

# Distributing the scan



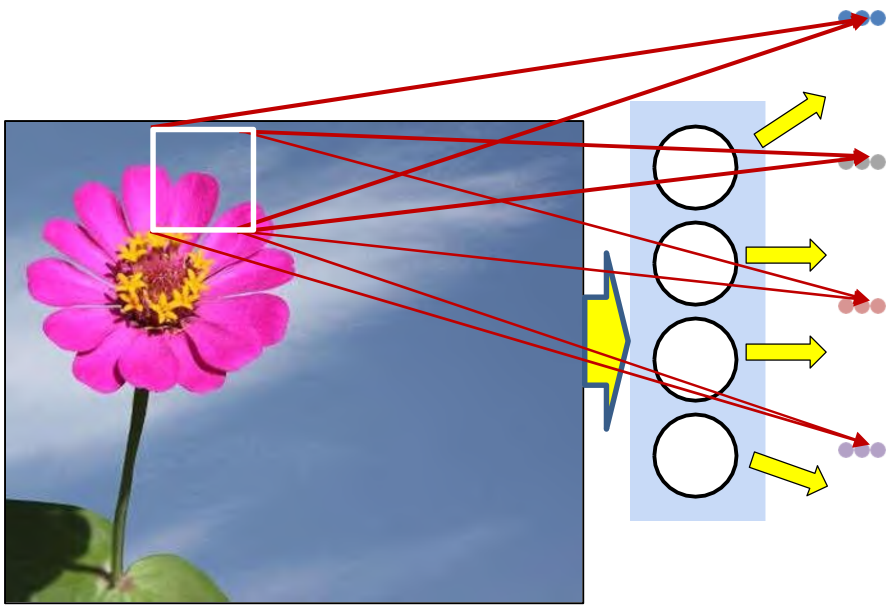
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



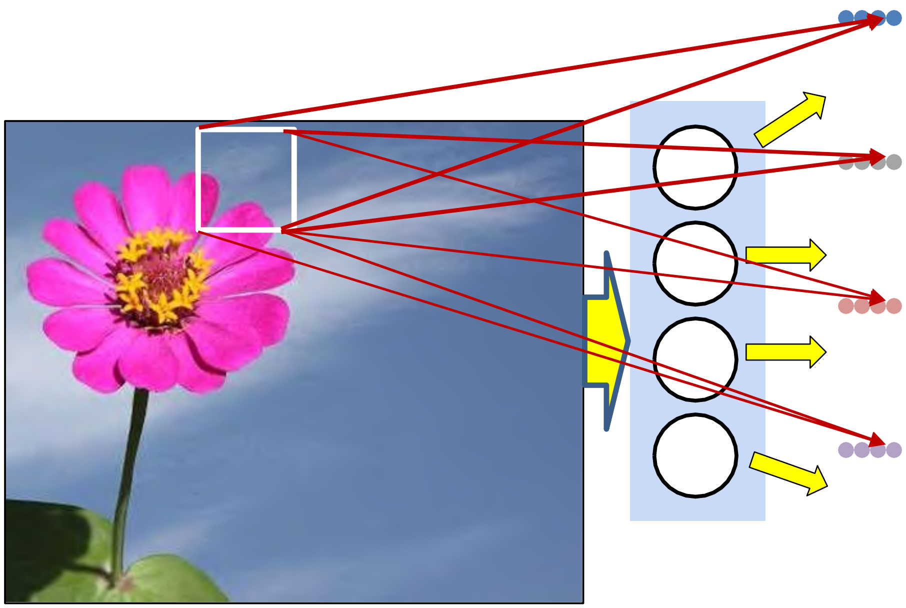
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



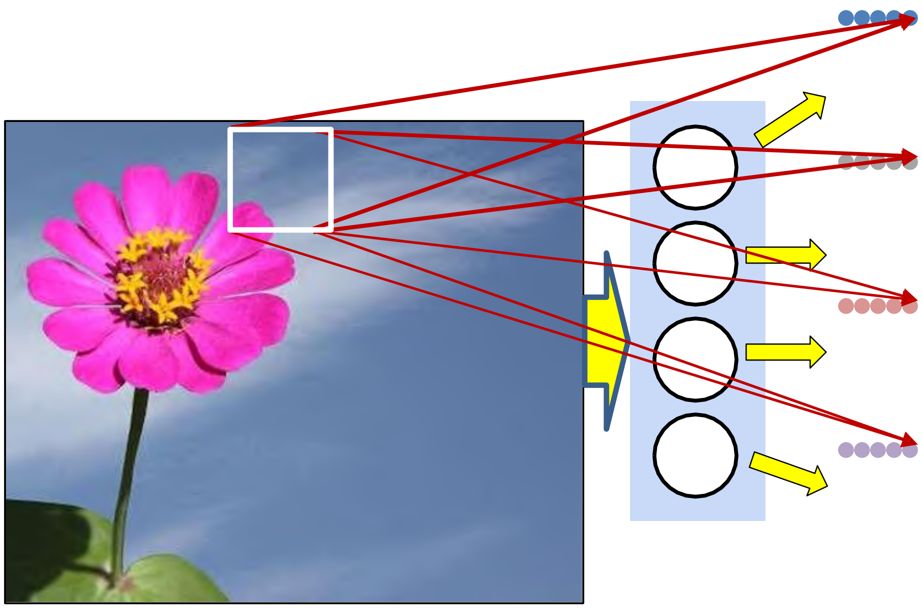
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



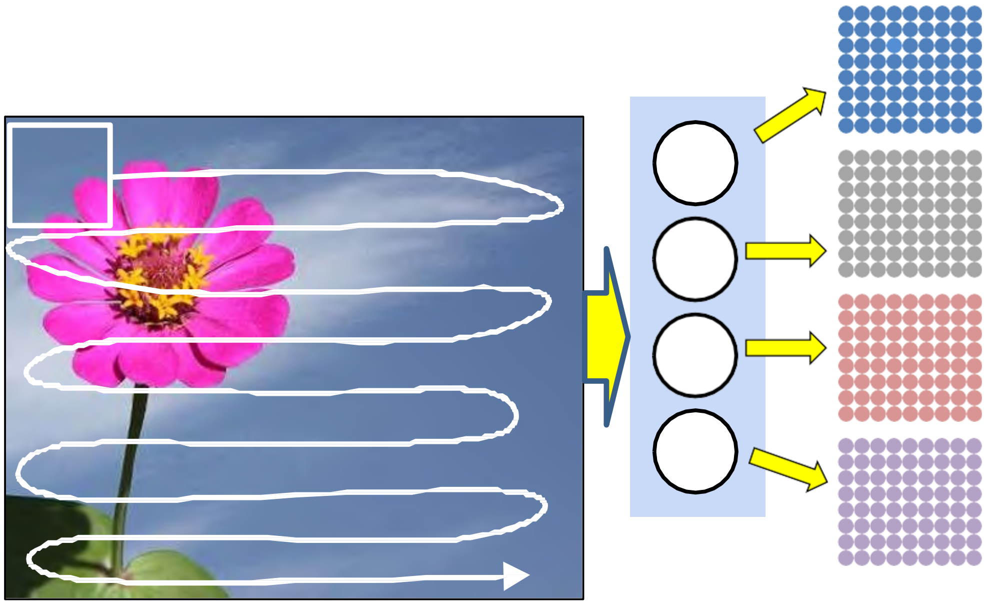
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



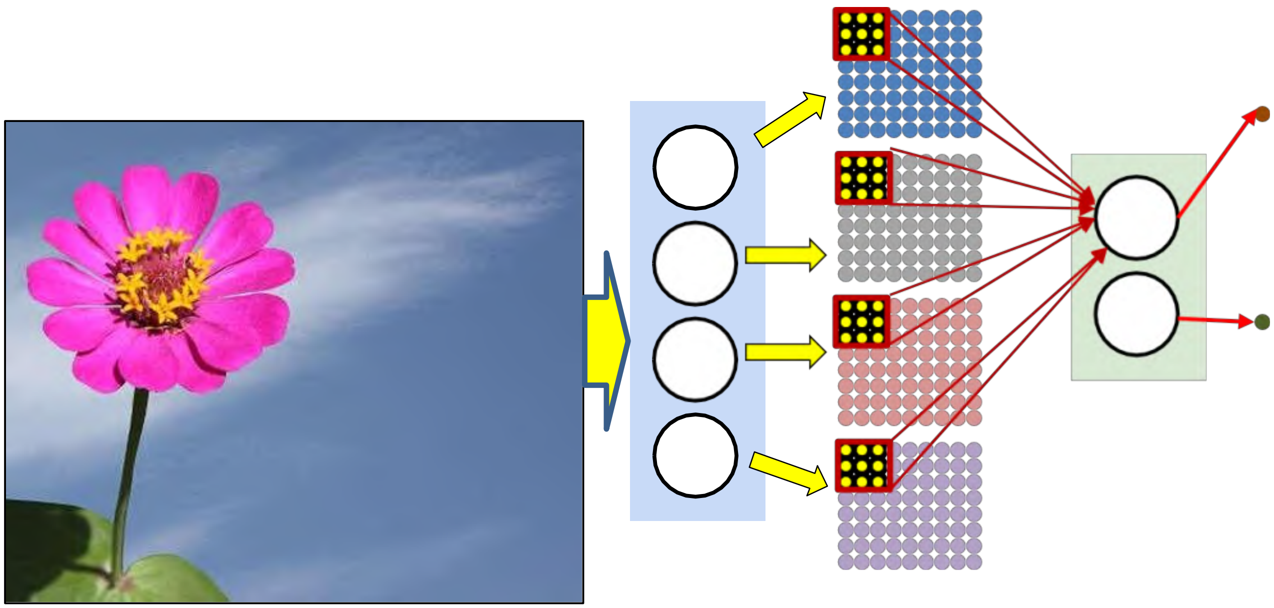
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



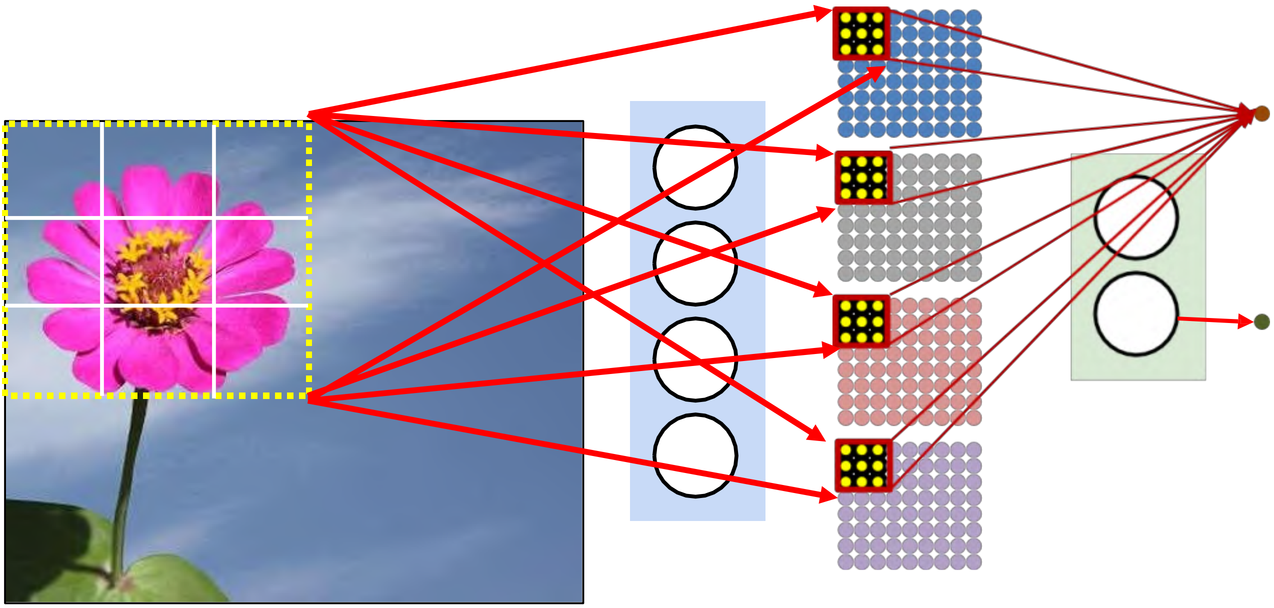
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels

# Distributing the scan



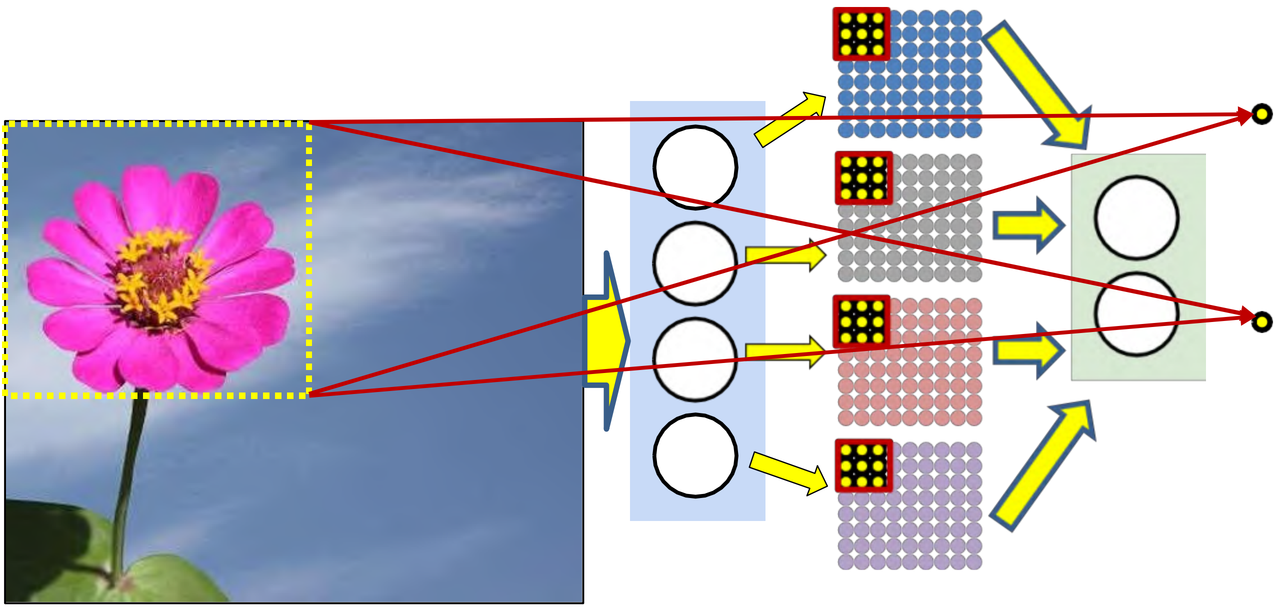
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates blocks of outputs from the first layer

# Distributing the scan



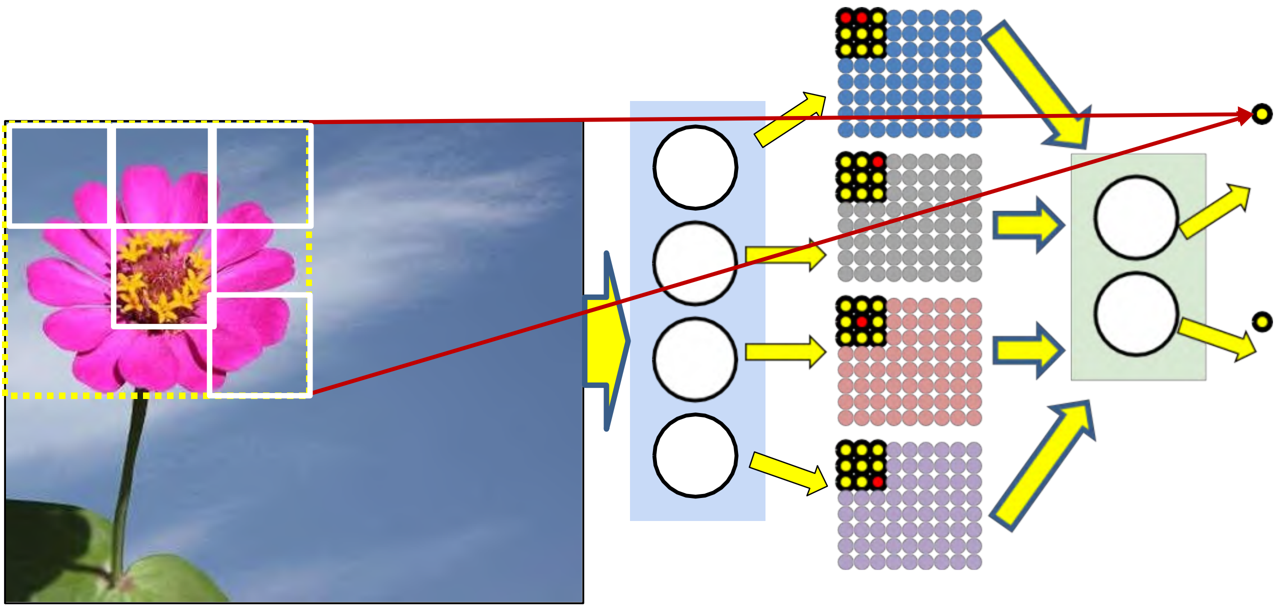
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  - The first layer evaluates smaller blocks of pixels
  - The next layer evaluates the windows of outputs from the first layer
  - This effectively evaluates the larger window of the original image

# Distributing the scan



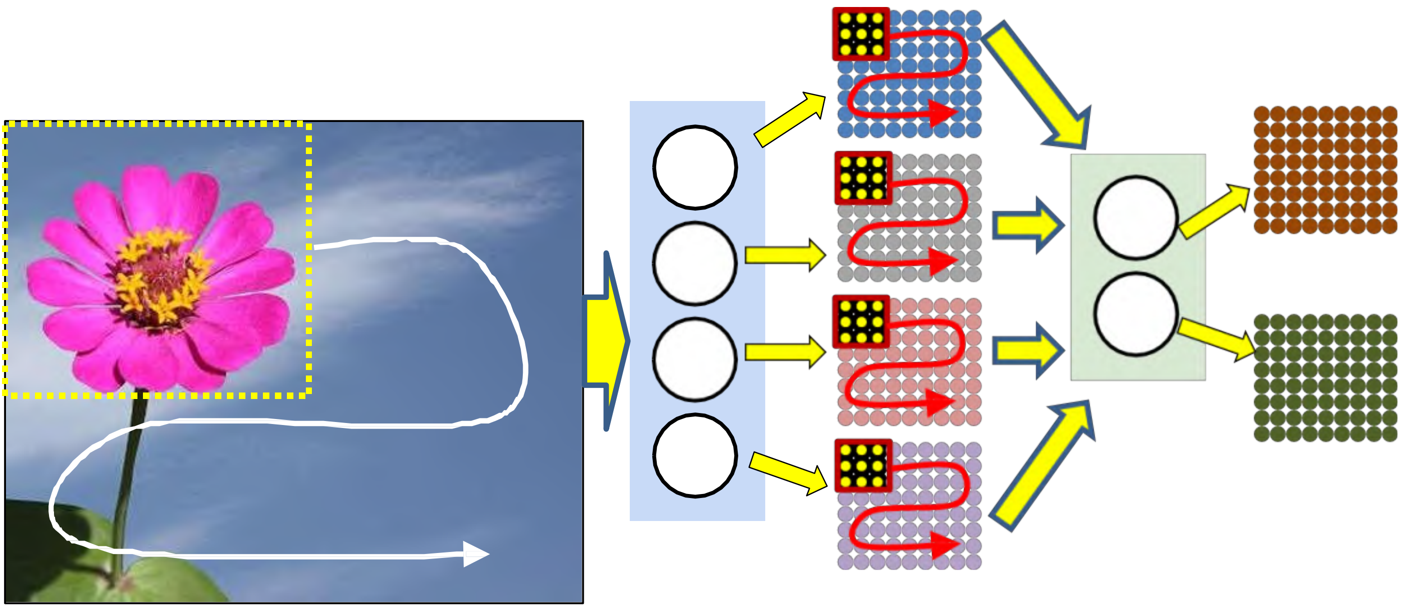
- We can distribute the pattern matching over two layers and still achieve the same block analysis at the second layer
  – The first layer evaluates smaller blocks of pixels
  – The next layer evaluates windows of outputs from the first layer
  – This effectively evaluates the larger window of the original image

# Distributing the scan



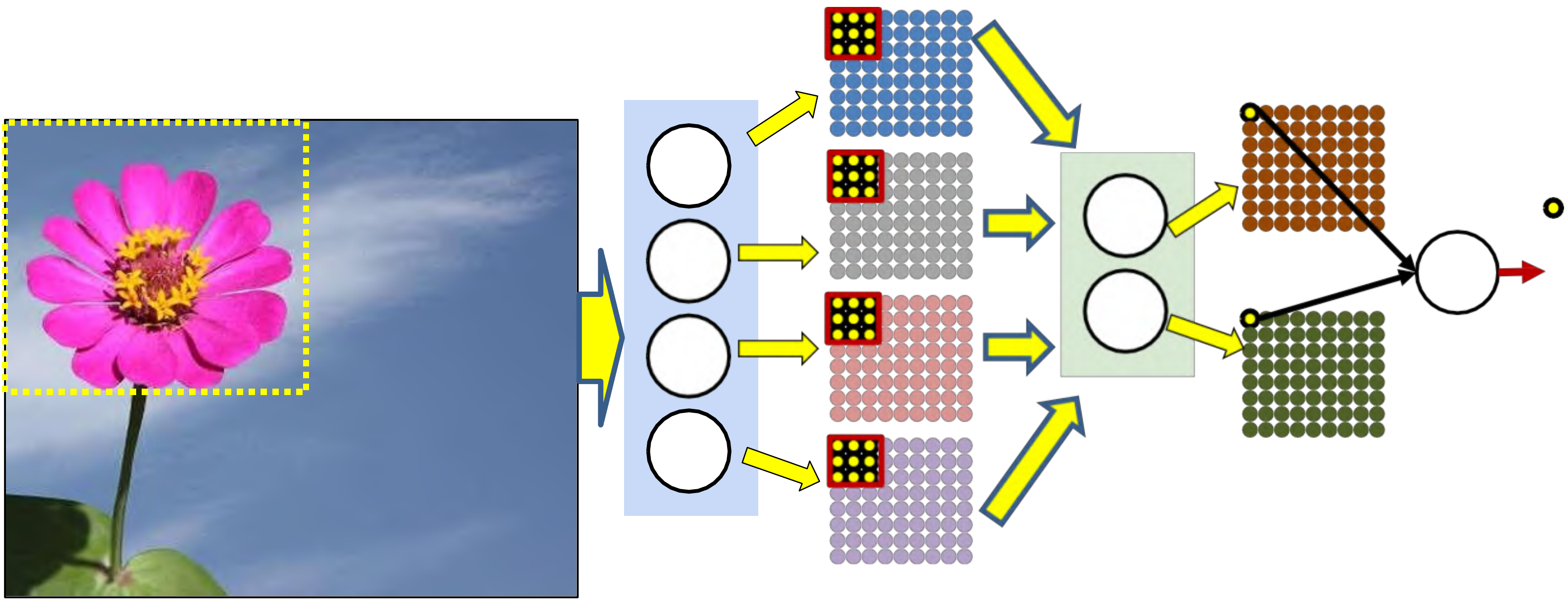
- The window has been *distributed* over two layers
- The higher layer implicitly learns the *arrangement* of sub patterns that represents the larger pattern (the flower in this case)

# Distributing the scan



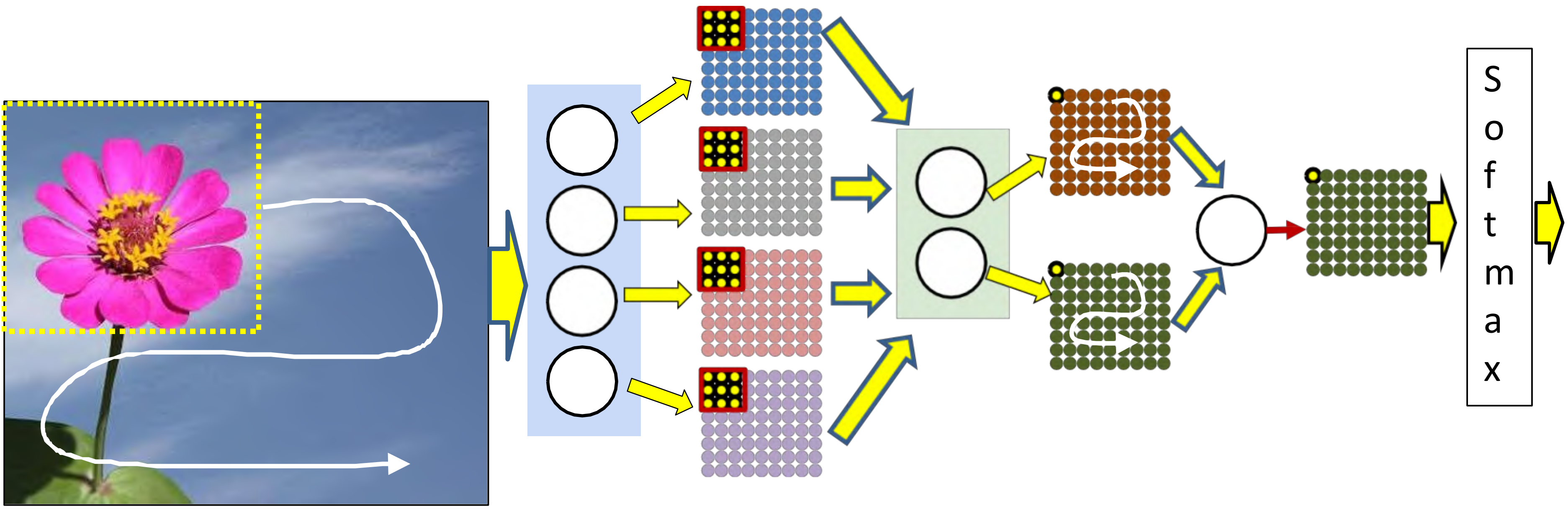
- If second layer neurons scan the maps output by first-layer neurons, they effectively scan the input with the full-sized window
  - *Jointly* scan all the first-layer maps
  - Each output of the second-layer neuron represents the output for *one* full-sized input window

# Distributing the scan



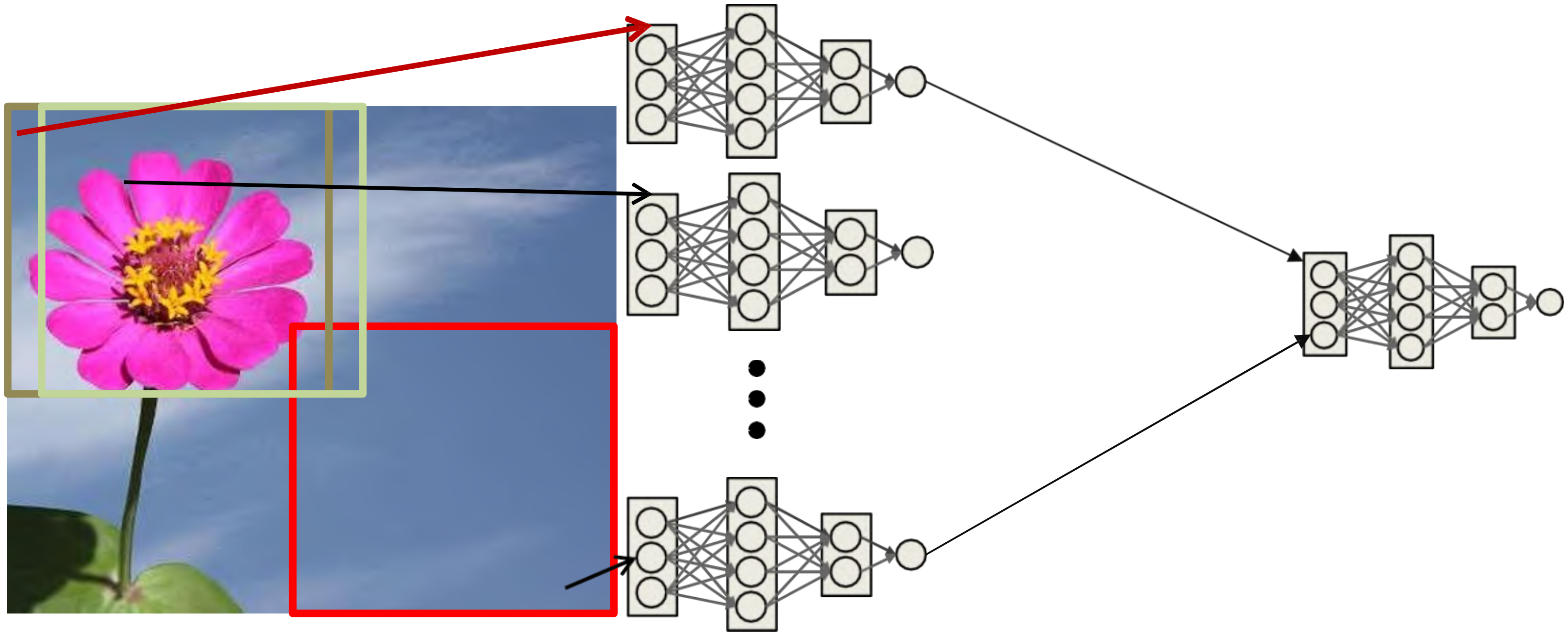
- If second layer neurons (jointly) scan the maps output by first-layer neurons, they effectively scan the input with the full-sized window
  - Each output of the second-layer neuron represents the output for *one* full-sized input window
- To compute the MLP output for a window of input, the output neuron only needs to consider the corresponding outputs of second-layer maps

# Distributing the scan



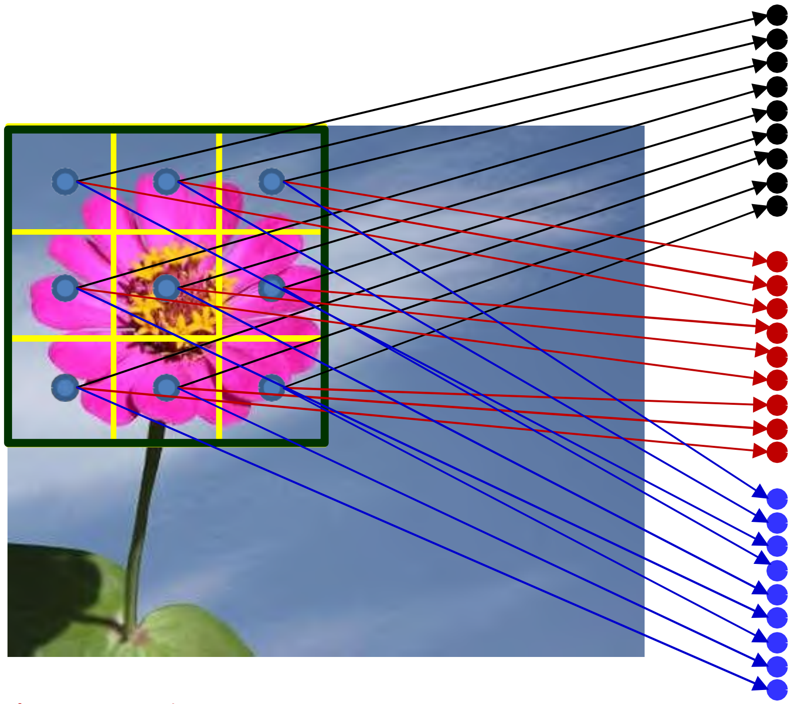

- If second layer neurons (jointly) scan the maps output by first-layer neurons, they effectively scan the input with the full-sized window
  - Each output of the second-layer neuron represents the output for *one* full-sized input window
- To compute the MLP output for a window of input, the output neuron only needs to consider the corresponding outputs of second-layer maps
- The output neuron can compute its outputs for every window in the input from the values of the second layer maps (and send it to a subsequent softmax)

# This is *still* just scanning with a shared parameter network



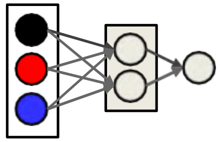
- With a minor modification…

# This is *still* just scanning with a shared parameter network



Each arrow represents an entire set of weights over the smaller cell

The pattern of weights going out of any cell is identical to that from any other cell.

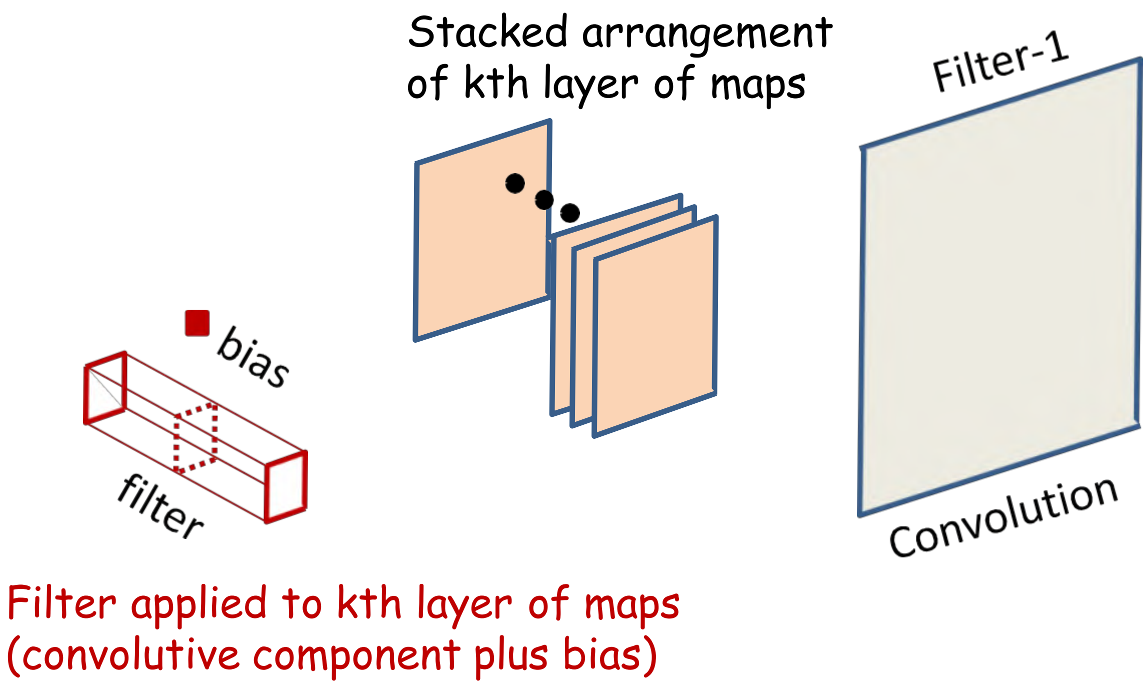


Colors indicate neurons with shared parameters

Layer 1

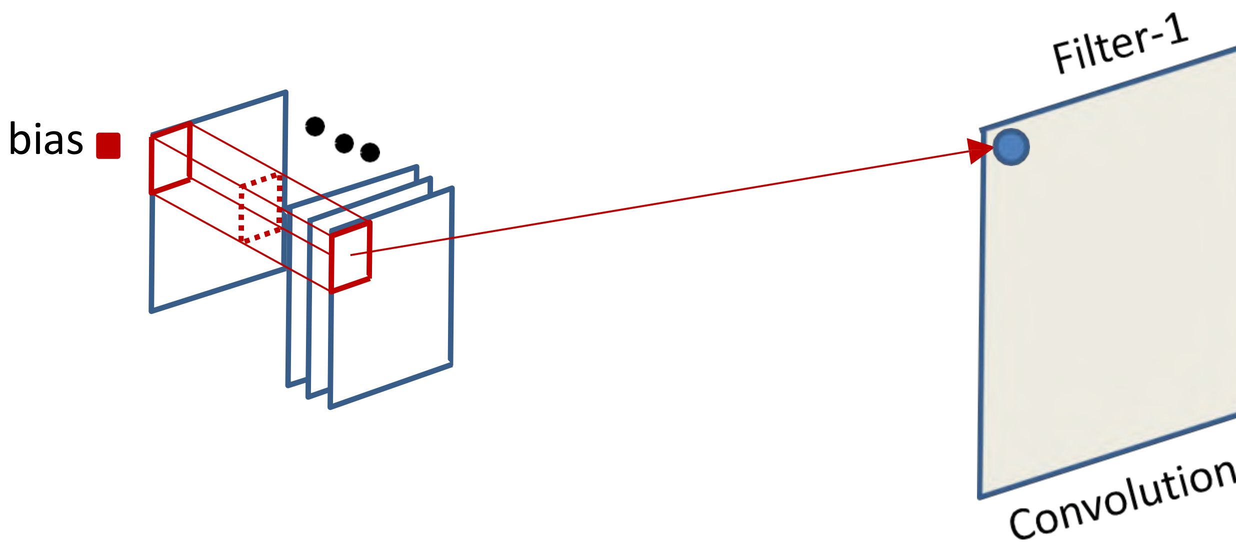
- The network that analyzes individual blocks is now itself a shared parameter network..

# A different view




- ..A *stacked* arrangement of planes
- We can view the joint processing of the various maps as processing the stack using a three-dimensional filter

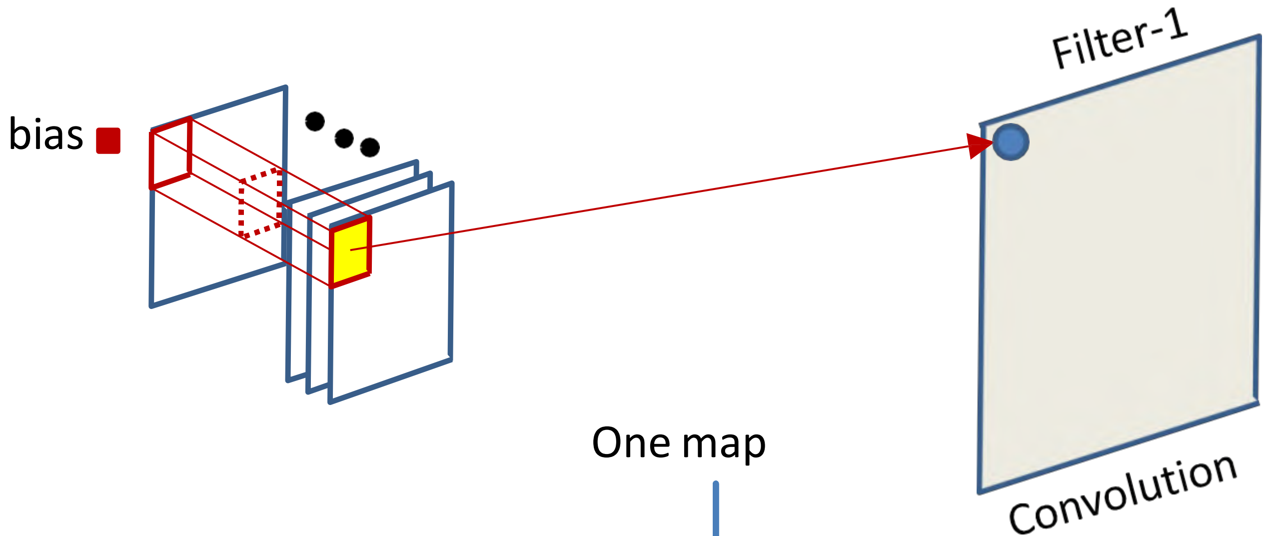
# The "cube" view of input maps




$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l)Y(p,i+l-1,j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

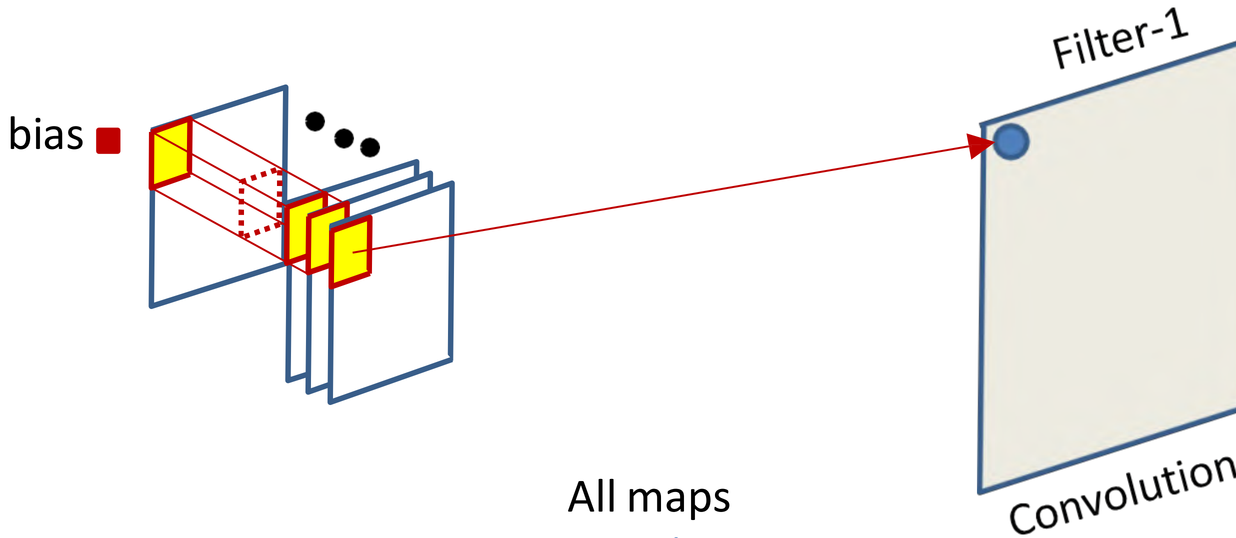
# The "cube" view of input maps




$$z(s,i,j) = \underbrace{\sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1)}_{\text{One map}} + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

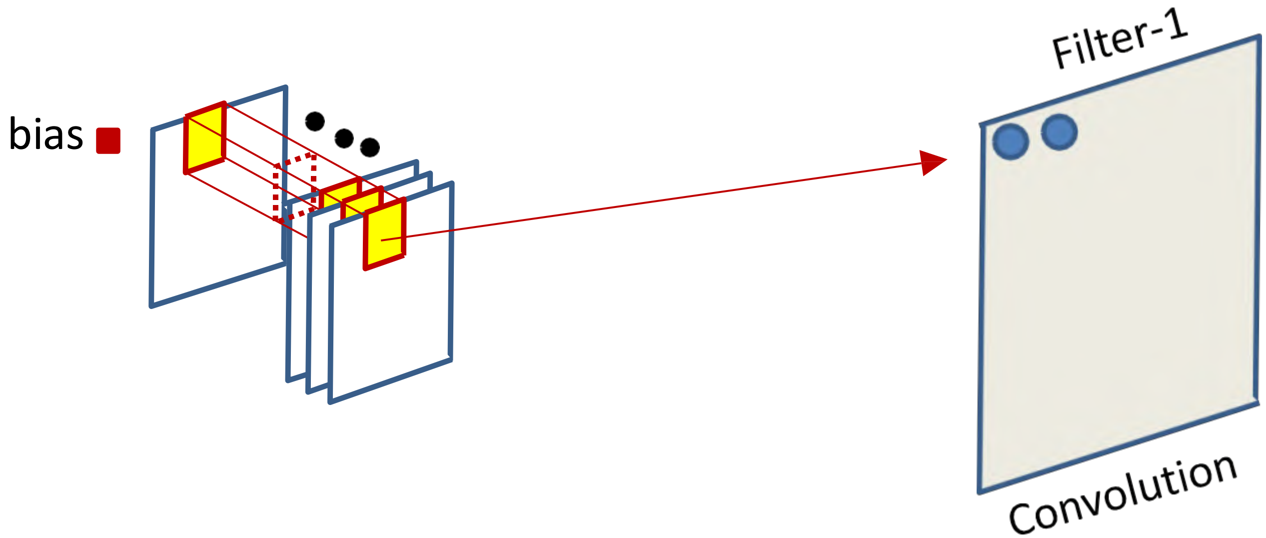
# The "cube" view of input maps




$$z(s,i,j) = \overbrace{\sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)}^{\text{All maps}}$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

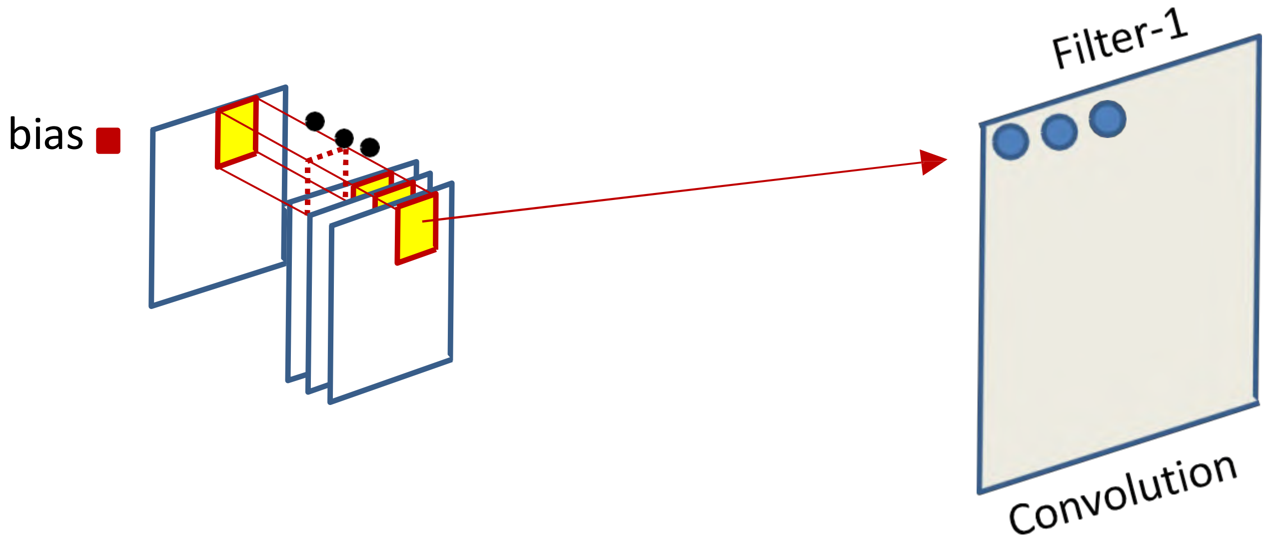
# The "cube" view of input maps




$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l)Y(p,i+l-1,j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

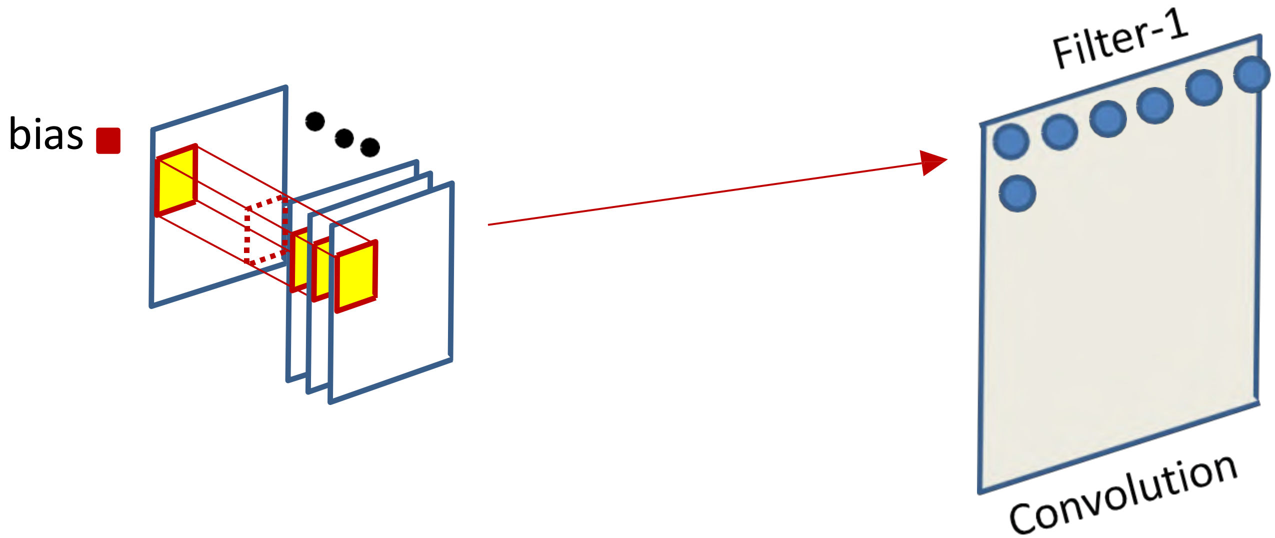
# The "cube" view of input maps




$$z(s,i,j) = \sum_{p}\sum_{k=1}^{L}\sum_{l=1}^{L} w(s,p,k,l)Y(p,i+l-1,j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

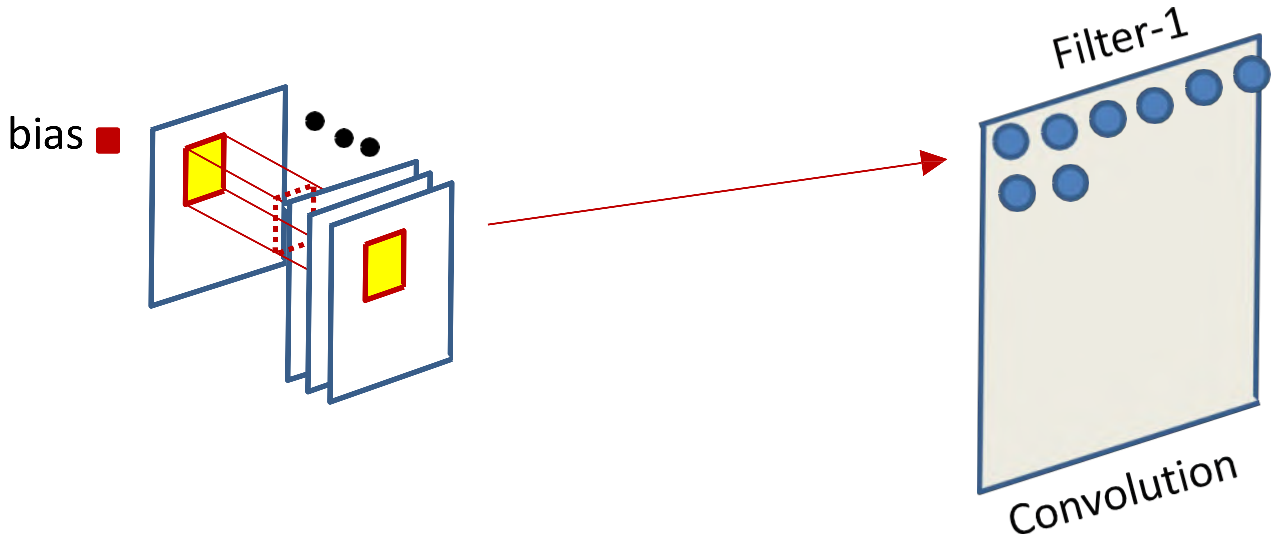
# The "cube" view of input maps




$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

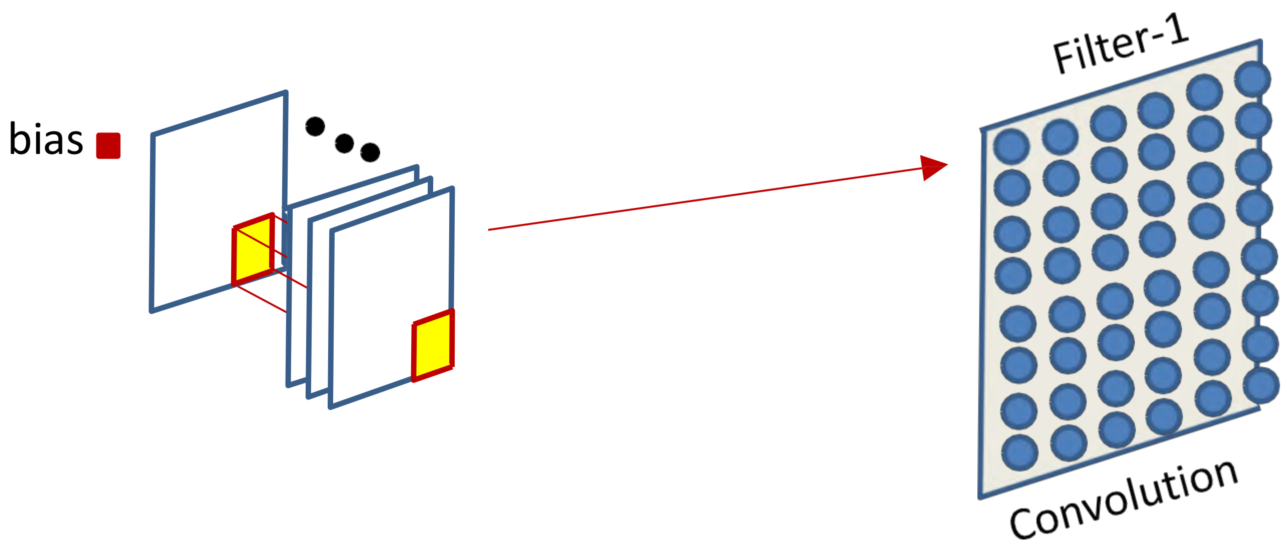
# The "cube" view of input maps




$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

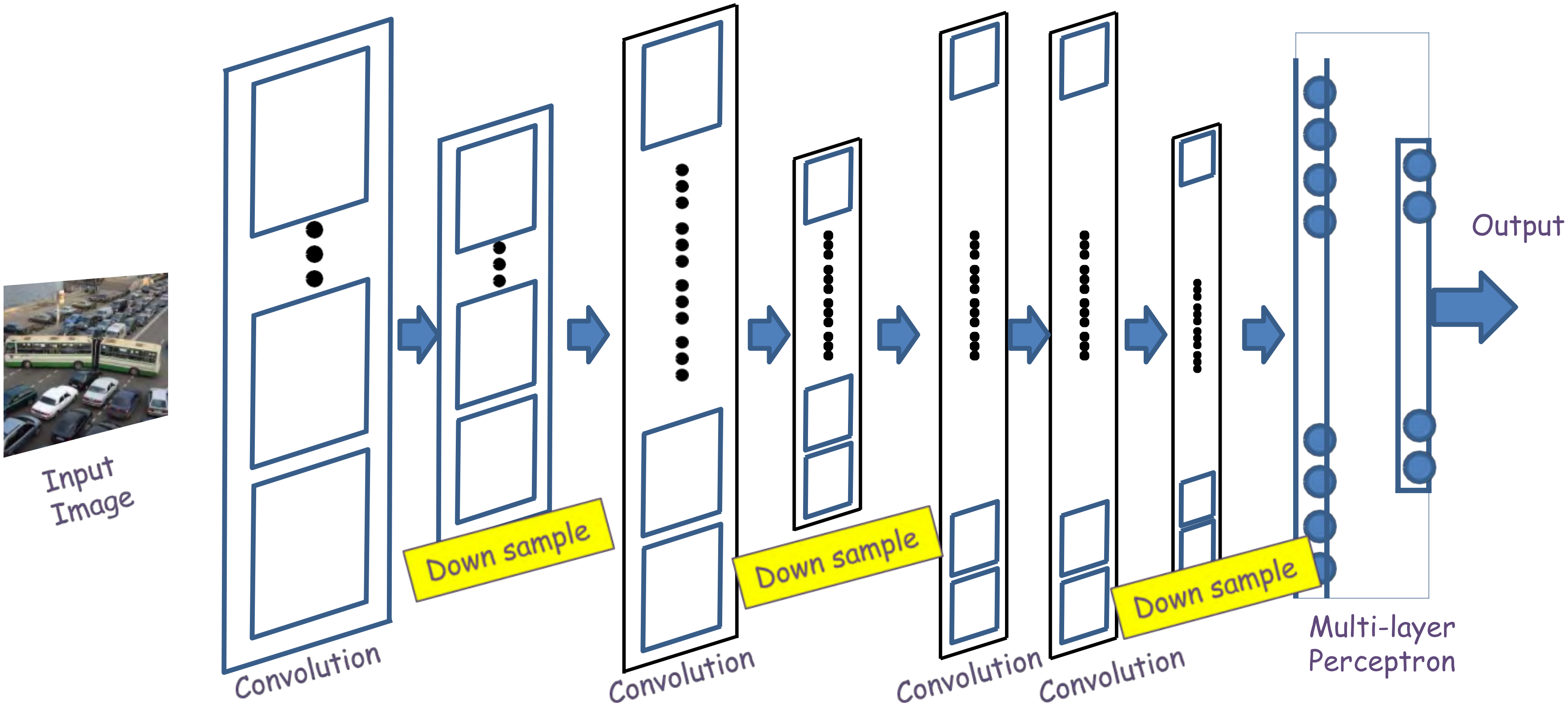
# The “cube” view of input maps




$$z(s,i,j) = \sum_{p} \sum_{k=1}^{L} \sum_{l=1}^{L} w(s,p,k,l) Y(p, i+l-1, j+k-1) + b(s)$$

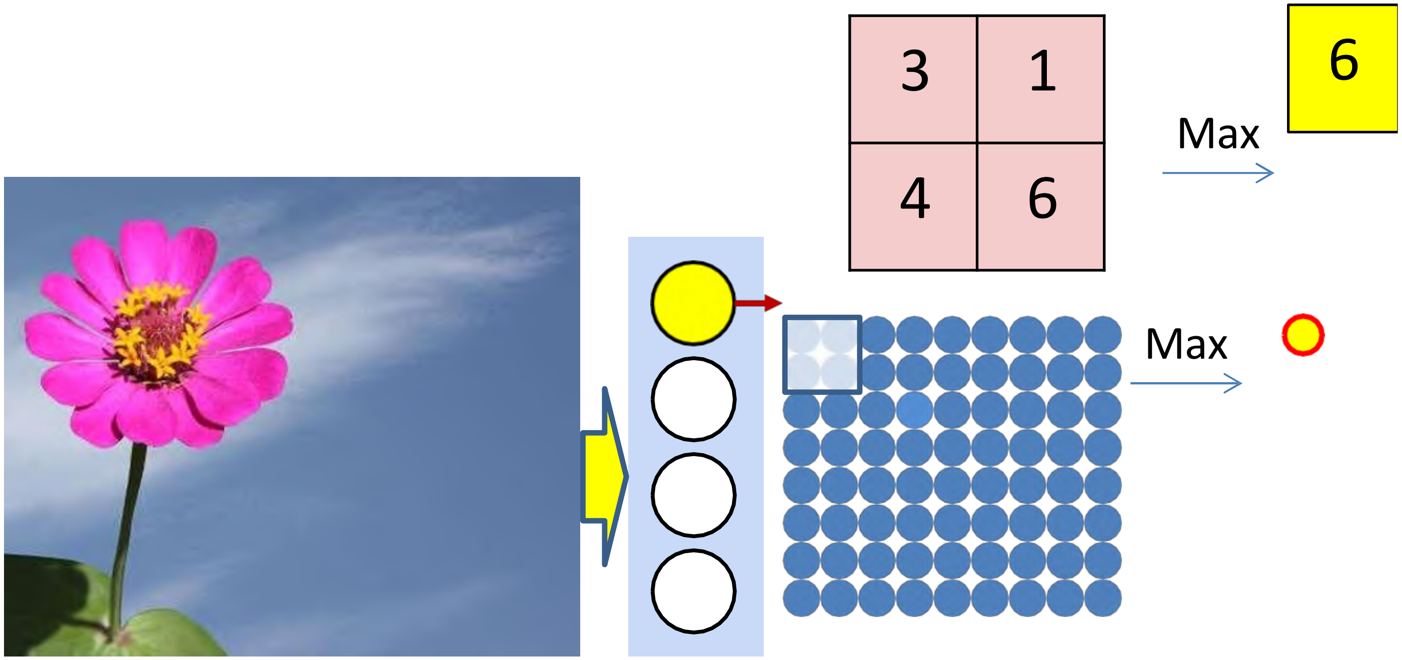
- The computation of the convolutional map at any location *sums* the convolutional outputs *at all planes*

# The other component
# Downsampling/Pooling



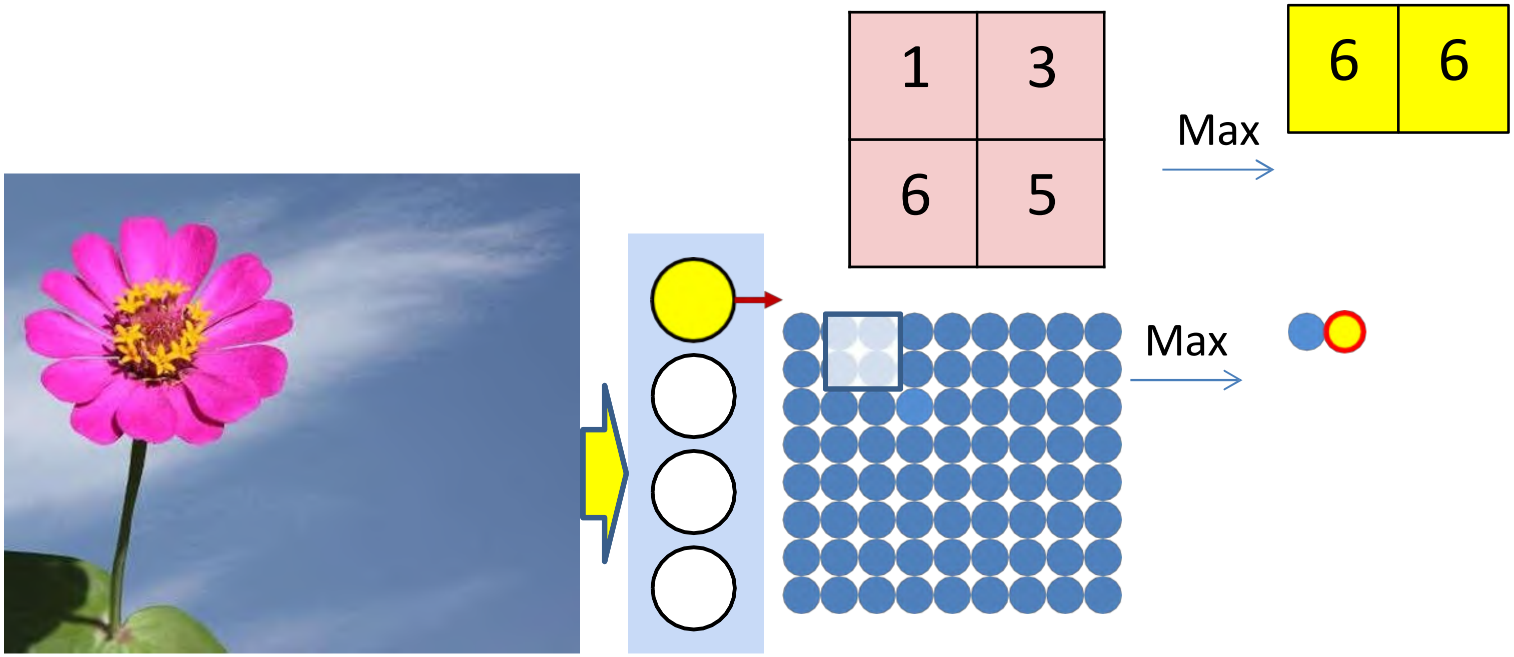

- Convolution (and activation) layers are followed intermittently by "downsampling with pooling" layers
  - Typically (but not always) "max" pooling
  - Often, they alternate with convolution, though this is not necessary

# Max pooling



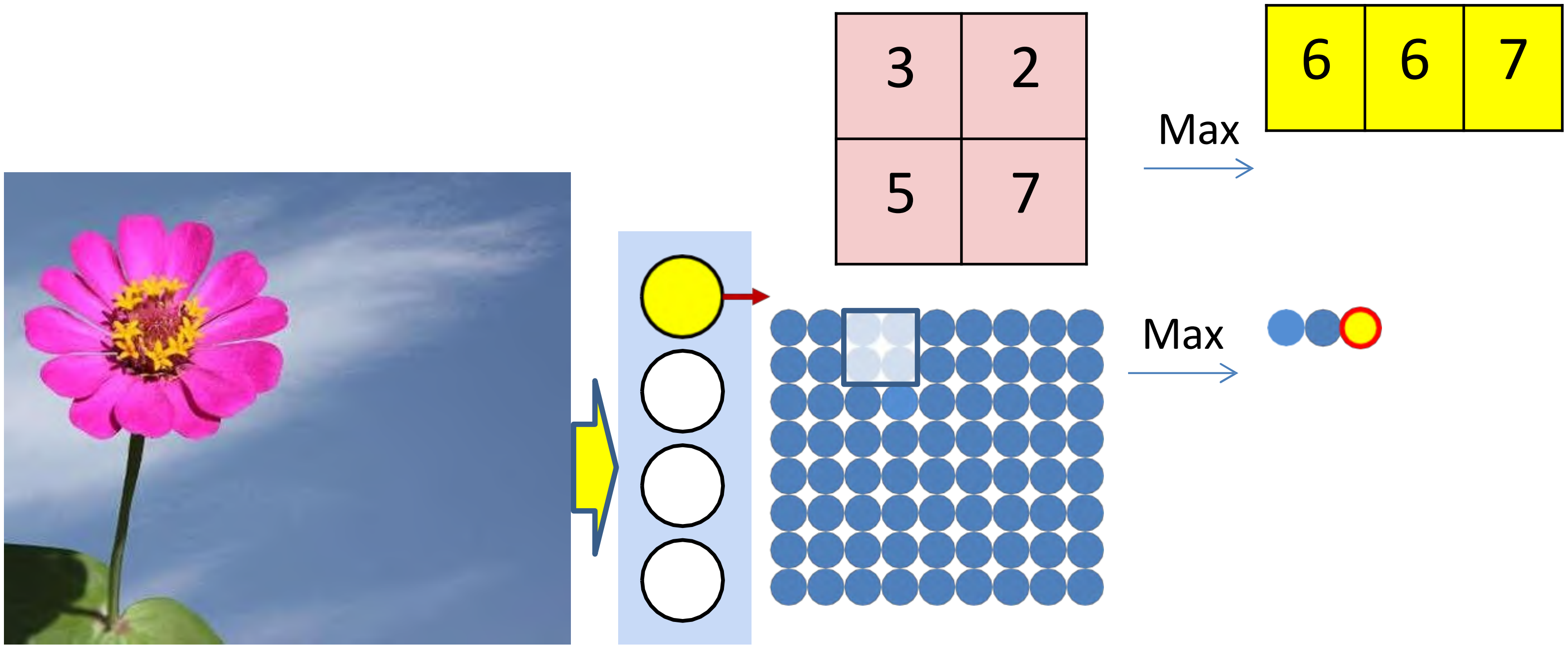

- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

# Recall: Max pooling



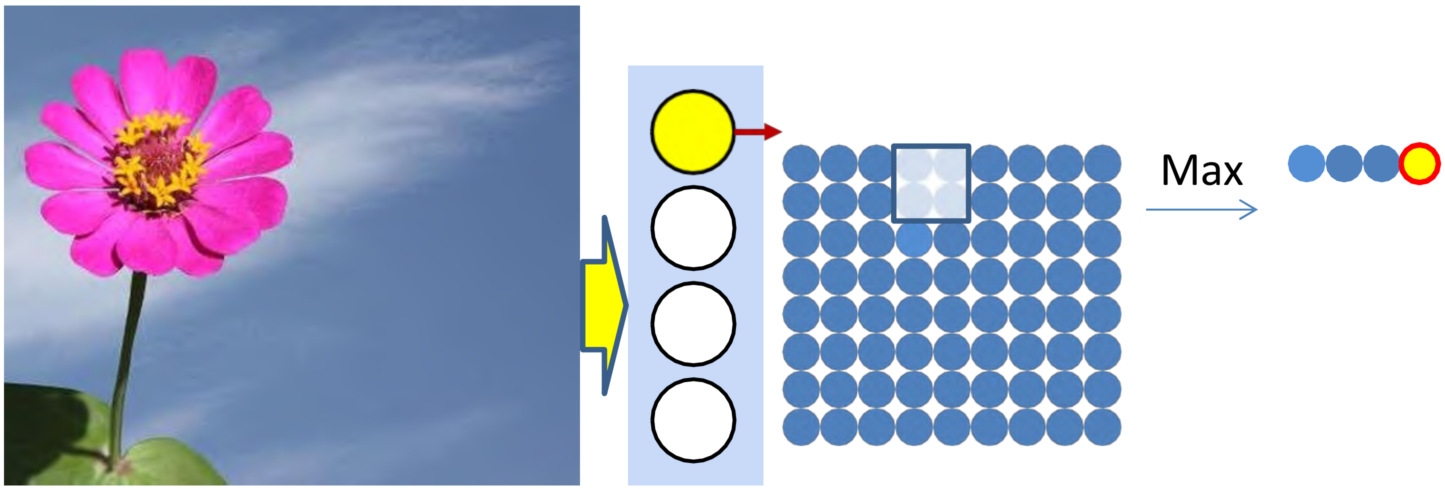

- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

# Recall: Max pooling



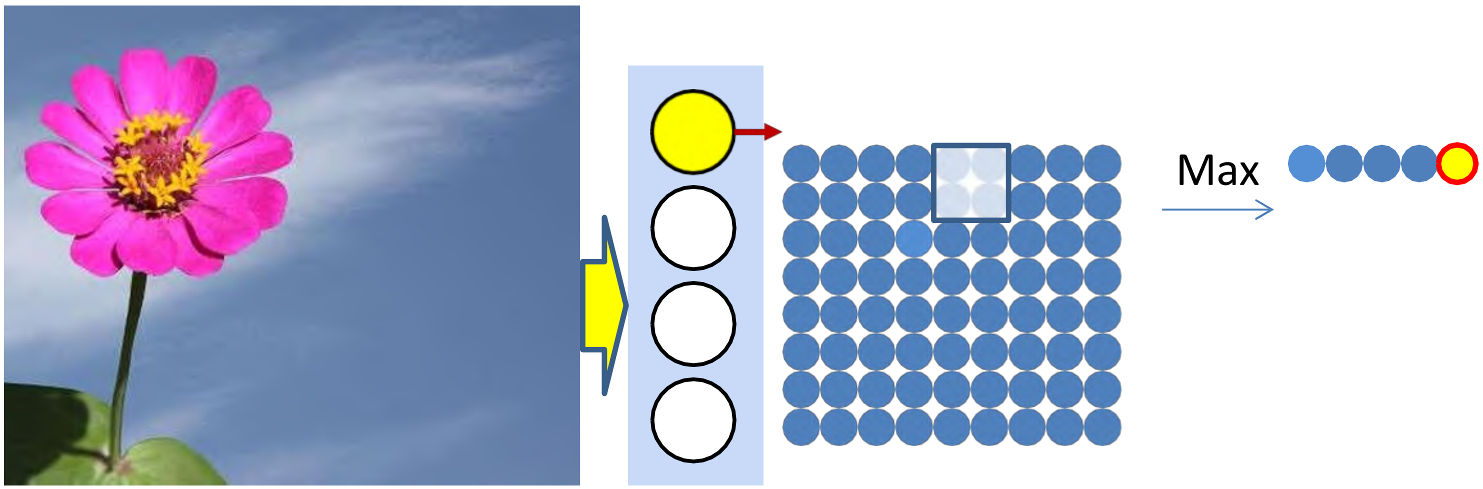

- Max pooling selects the largest from a pool of elements
- Pooling is performed by "scanning" the input

# Recall: Max pooling



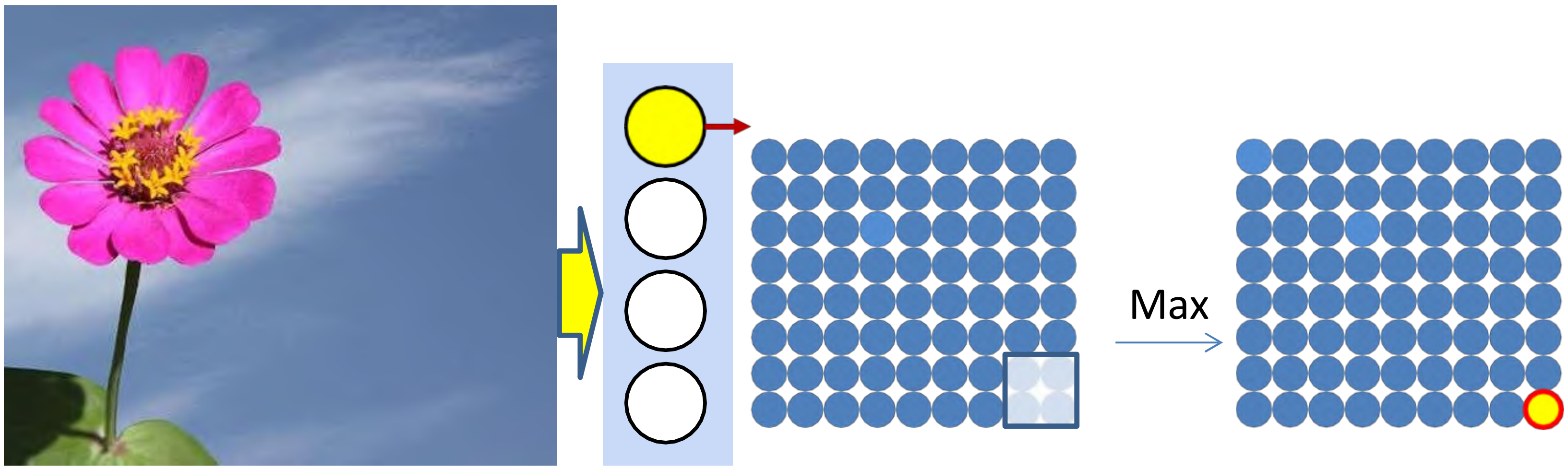

- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

# Recall: Max pooling



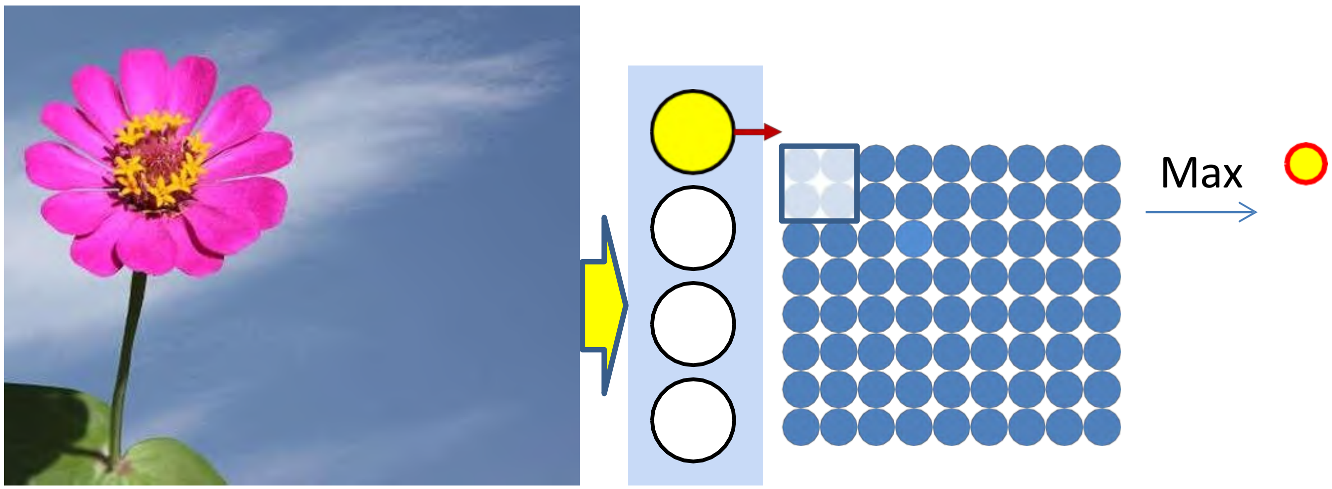

- Max pooling selects the largest from a pool of elements
- Pooling is performed by “scanning” the input

# Recall: Max pooling



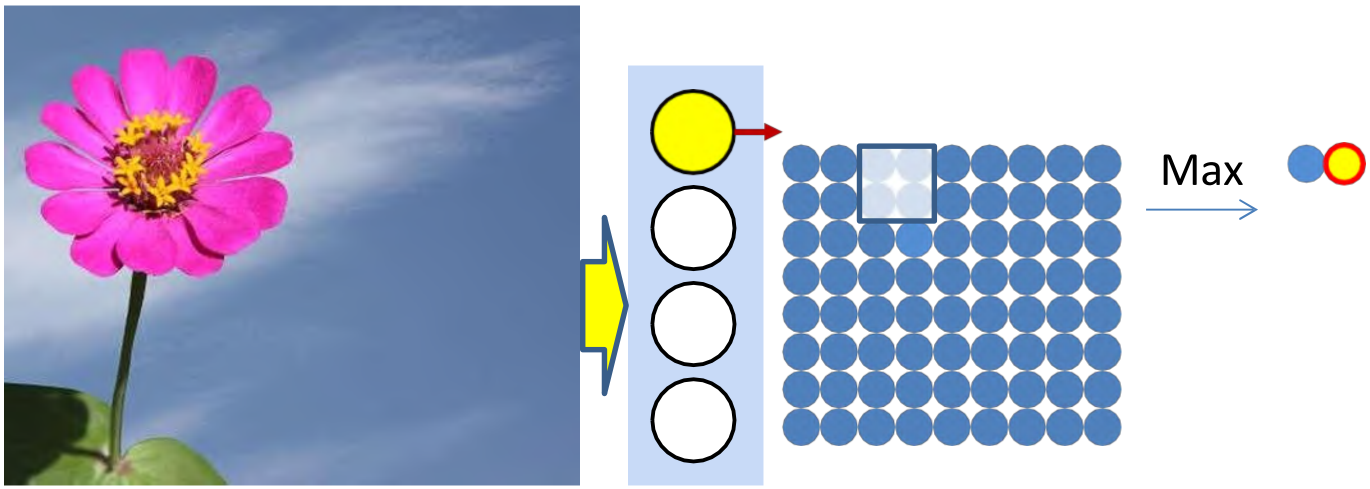

- Max pooling scans with a stride of 1 confer jitter-robustness, but do not constitute downsampling
- Downsampling requires a stride greater than 1

# Downsampling requires Stride>1



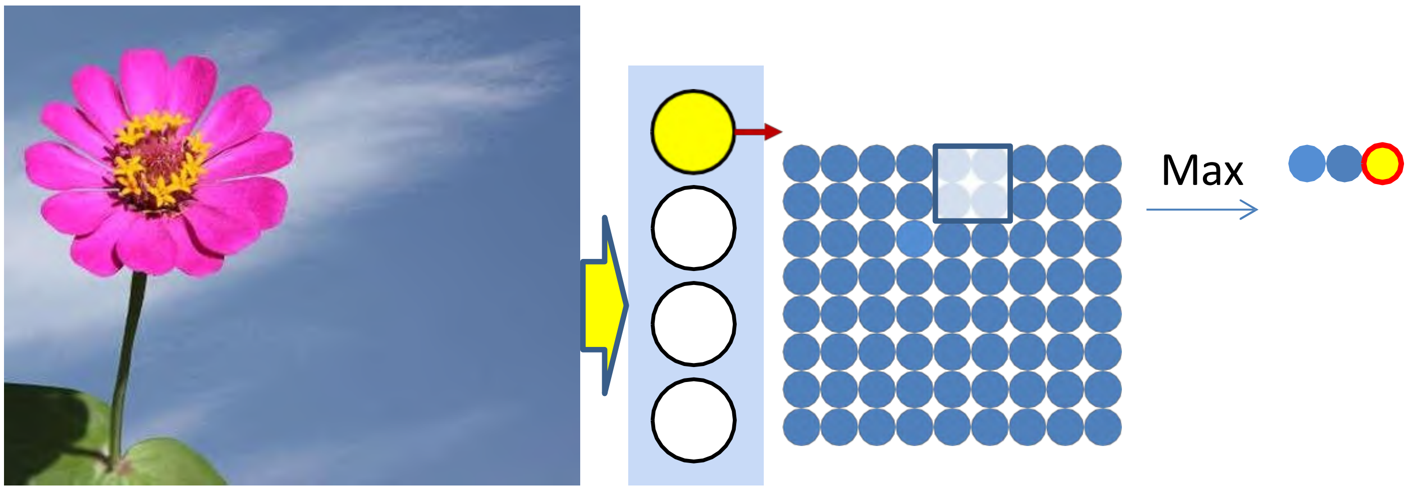

- The "max pooling" operation with "stride" greater than 1 results in an output smaller than the input
  - One output per stride
  - The output is "downsampled"

# Downsampling requires Stride>1



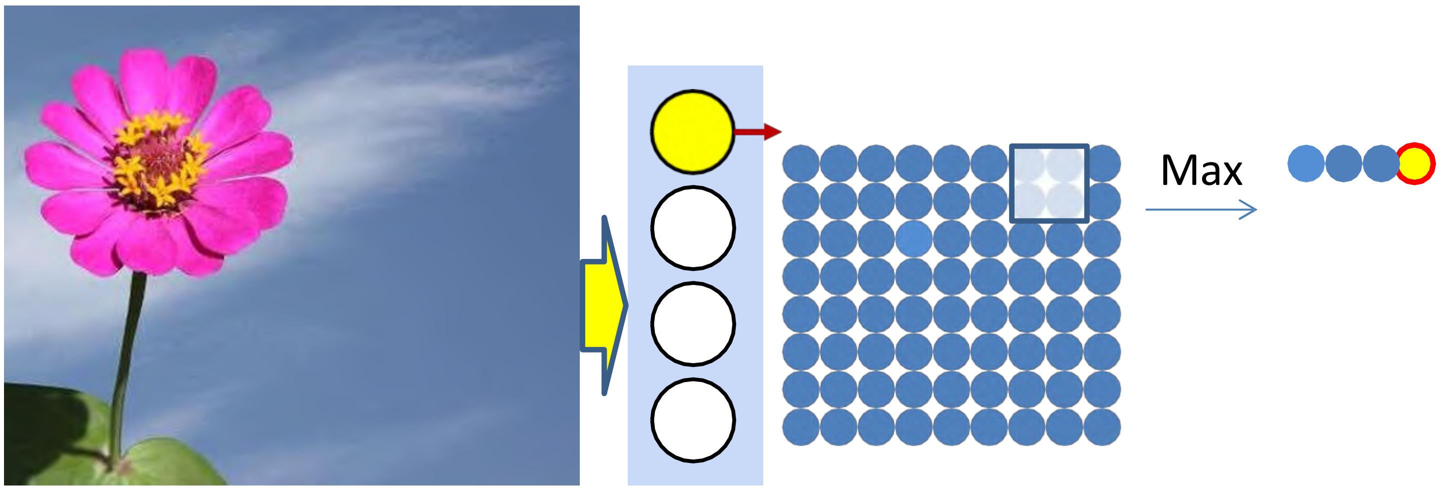

- The "max pooling" operation with "stride" greater than 1 results in an output smaller than the input
  - One output per stride
  - The output is "downsampled"

# Downsampling requires Stride>1



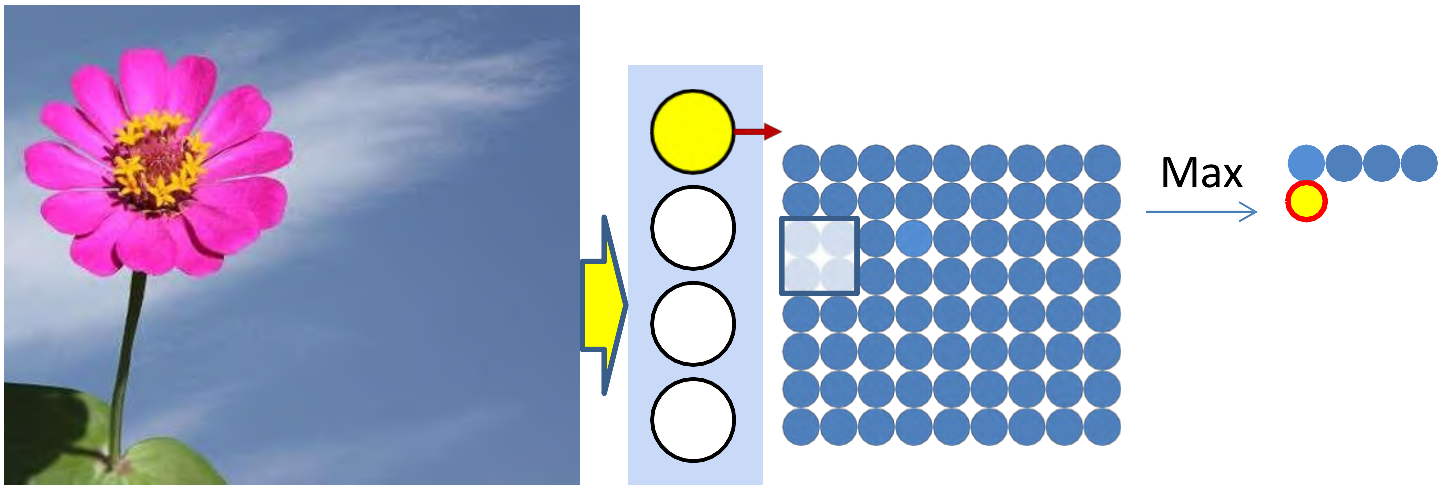

- The “max pooling” operation with “stride” greater than 1 results in an output smaller than the input
  - One output per stride
  - The output is “downsampled”

# Downsampling requires Stride>1



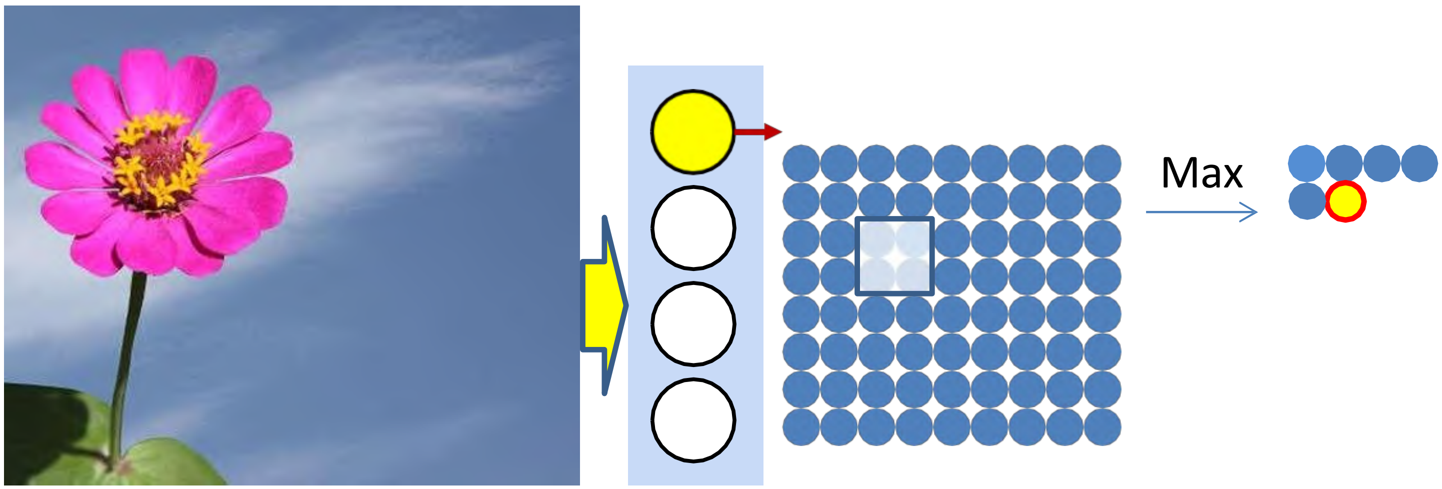

- The “max pooling” operation with “stride” greater than 1 results in an output smaller than the input
  - One output per stride
  - The output is “downsampled”

# Downsampling requires Stride>1



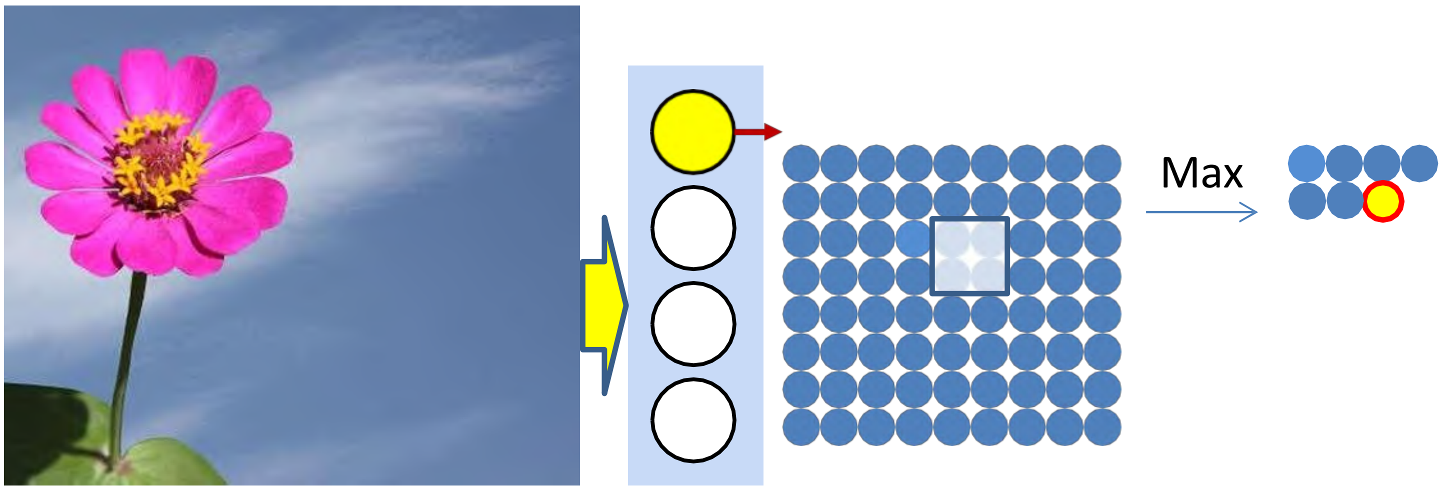

- The "max pooling" operation with "stride" greater than 1 results in an output smaller than the input
  - One output per stride
  - The output is "downsampled"

# Downsampling requires Stride>1



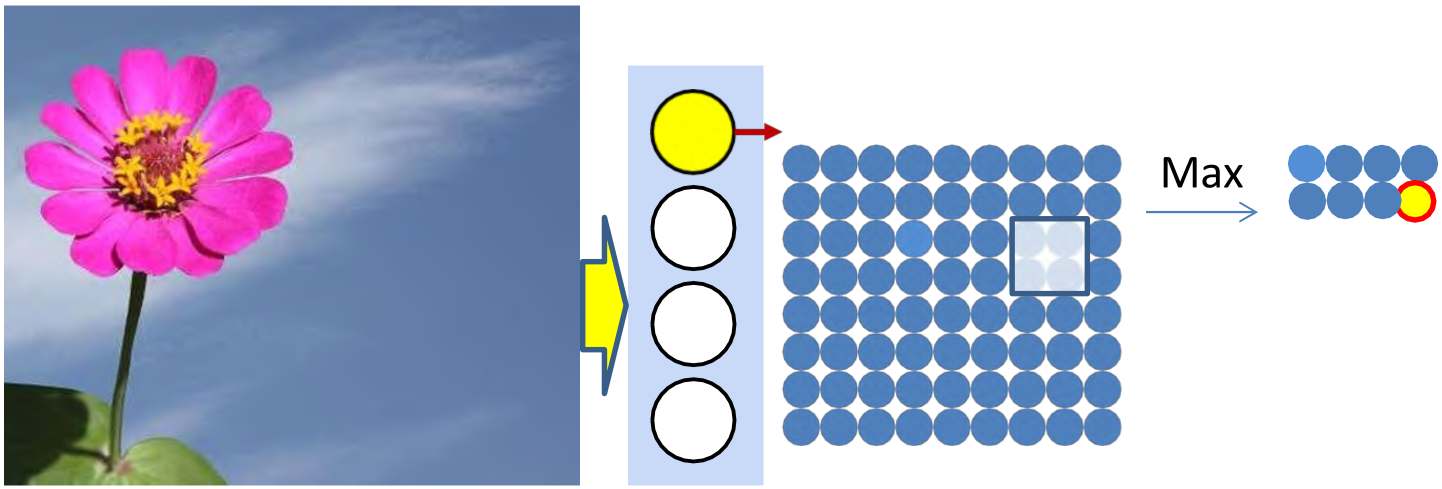

- The "max pooling" operation with "stride" greater than 1 results in an output smaller than the input
  - One output per stride
  - The output is "downsampled"

# Downsampling requires Stride>1



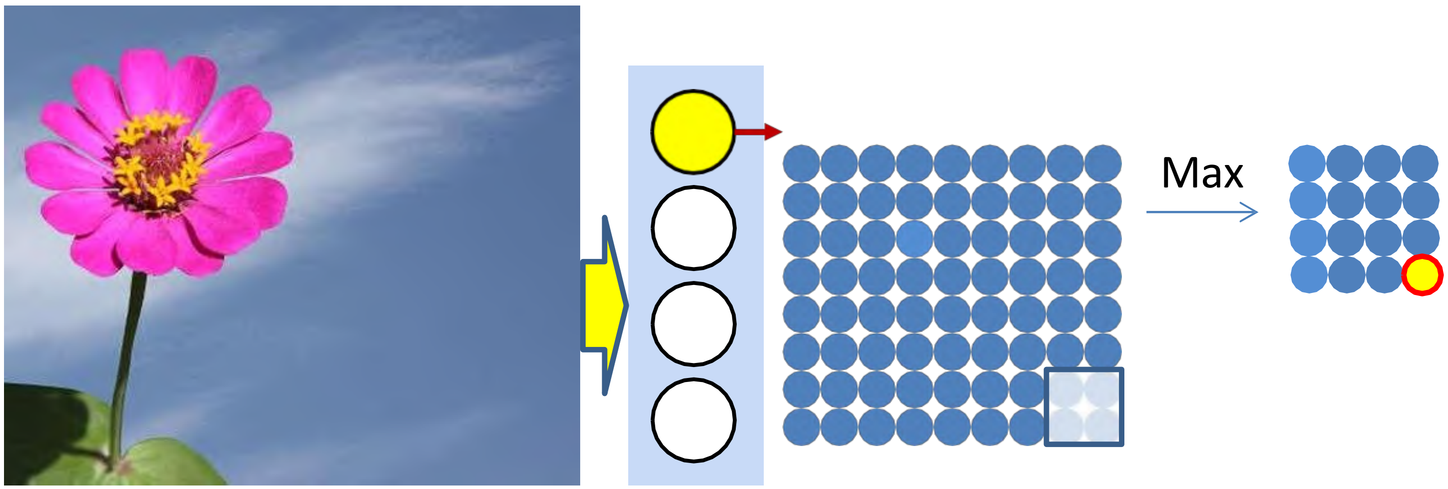

- The “max pooling” operation with “stride” greater than 1 results in an output smaller than the input
  - One output per stride
  - The output is “downsampled”

# Downsampling requires Stride>1




- The "max pooling" operation with "stride" greater than 1 results in an output smaller than the input
  - One output per stride
  - The output is "downsampled"

# Downsampling requires Stride>1

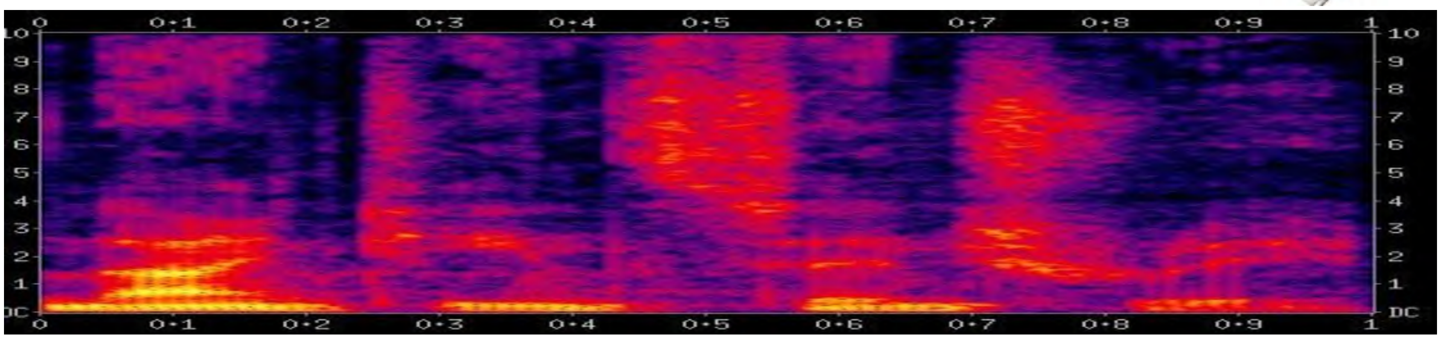



- The "max pooling" operation with "stride" greater than 1 results in an output smaller than the input
  - One output per stride
  - The output is "downsampled"

# Deep Learning
# Recurrent Networks

# What did I say?

“To be” or not “to be”??




- Speech Recognition
  - Analyze a series of spectral vectors, determine what was said
- Note: Inputs are sequences of vectors. Output is a classification result

# What is he talking about?

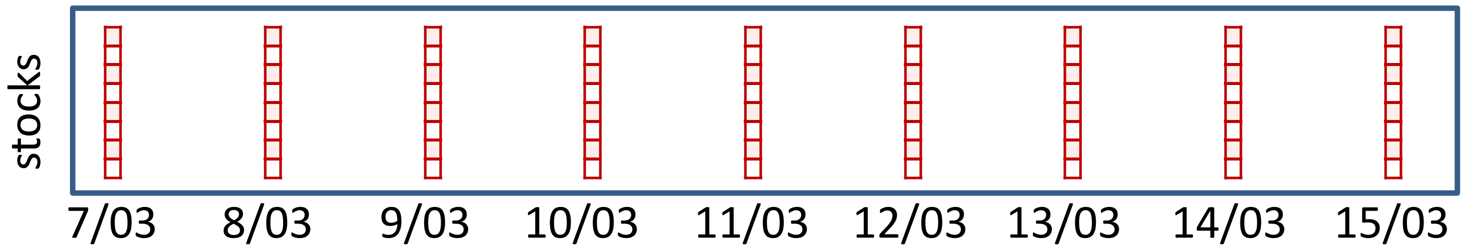
“Football” or “basketball”?

The Steelers, meanwhile, continue to struggle to make stops on defense. They've allowed, on average, 30 points a game, and have shown no signs of improving anytime soon.

- Text analysis
  - E.g. analyze document, identify topic
    - Input series of words, output classification output
  - E.g. read English, output French
    - Input series of words, output series of words

# Should I invest..

To invest or not to invest?



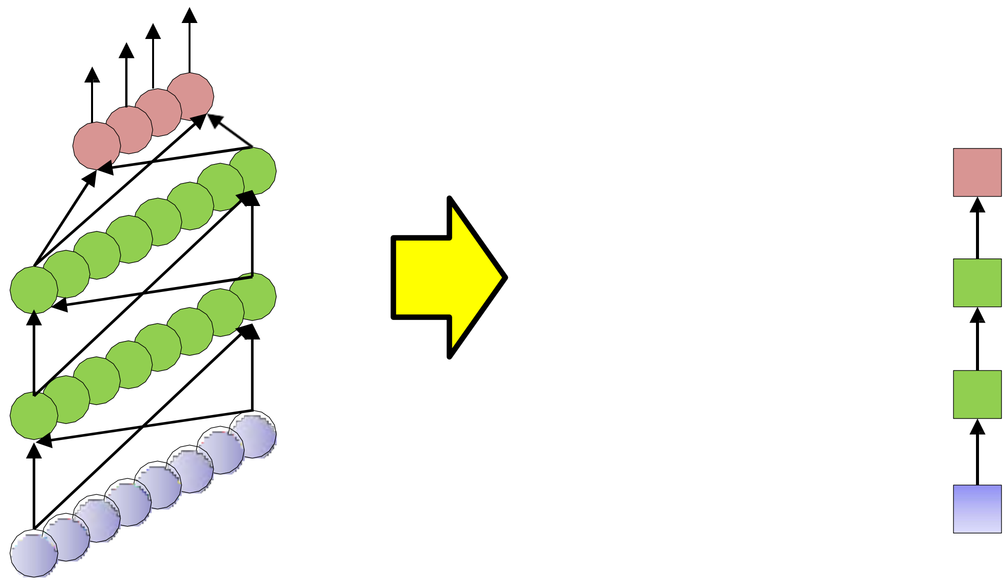

- Note: Inputs are sequences of vectors. Output may be scalar or vector
  - Should I invest, vs. should I not invest in X?
  - Decision must be taken considering how things have fared over time

# These are classification and prediction problems

- Consider a sequence of inputs
  - Input vectors
- Produce one or more outputs

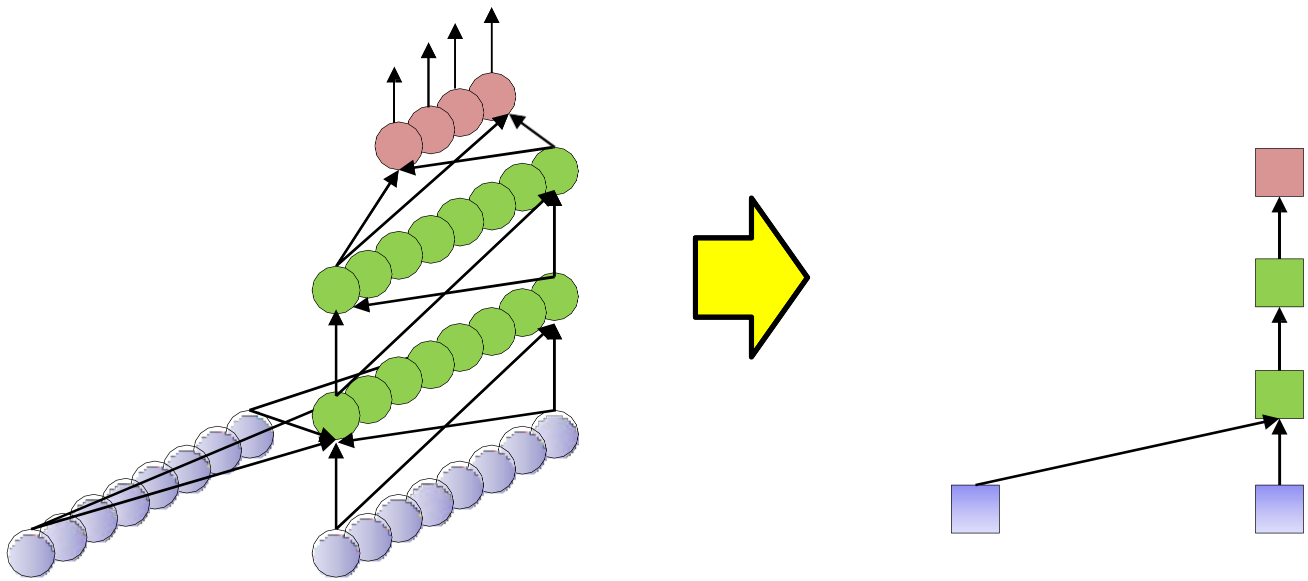
- This can be done with neural networks
  - Obviously

# Representational shortcut



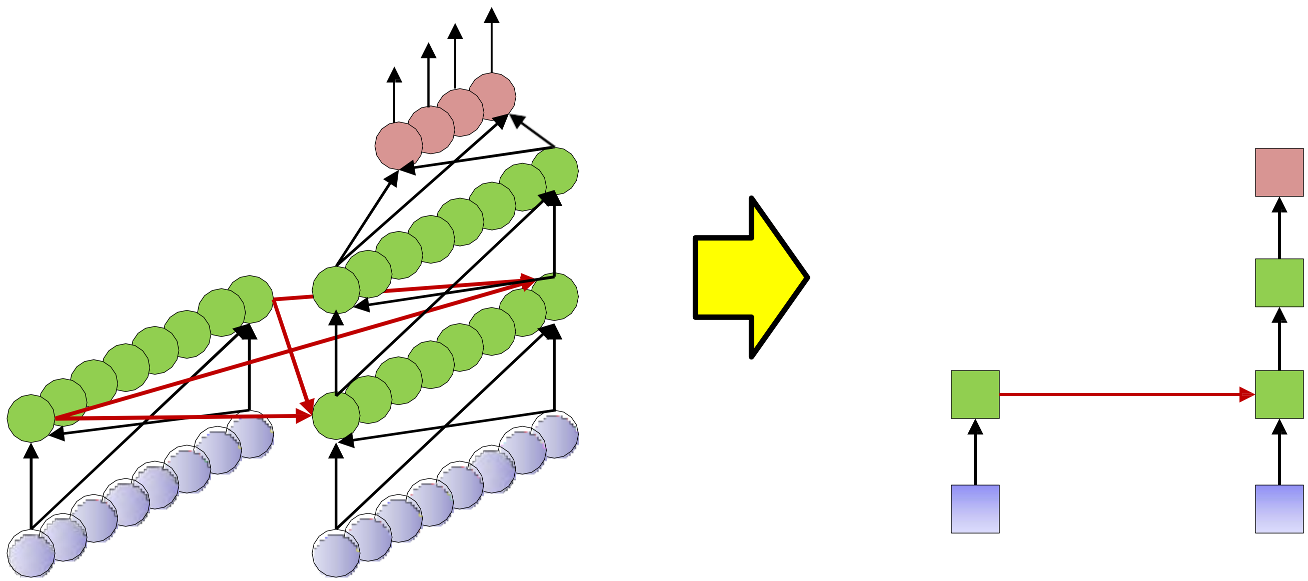
- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything by simple boxes
  - Each box actually represents an entire *layer with many units*

# Representational shortcut



- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything by simple boxes
  - Each box actually represents an entire *layer with many units*

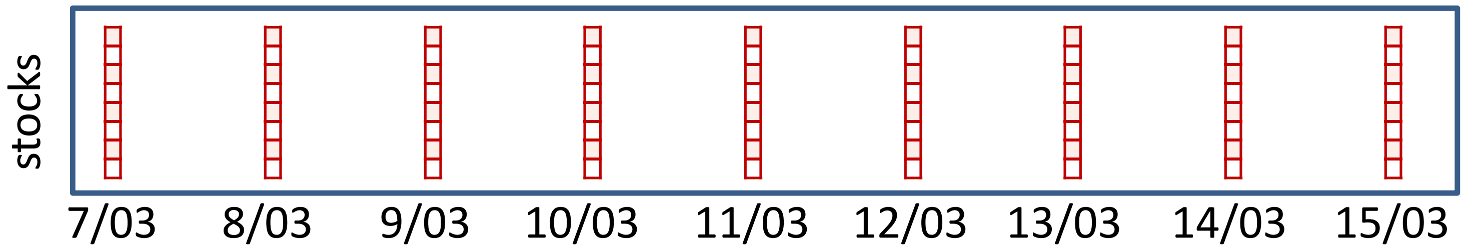
# Representational shortcut



- Input at each time is a *vector*
- Each layer has many neurons
  - Output layer too may have many neurons
- But will represent everything as simple boxes
  - Each box actually represents an entire *layer with many units*

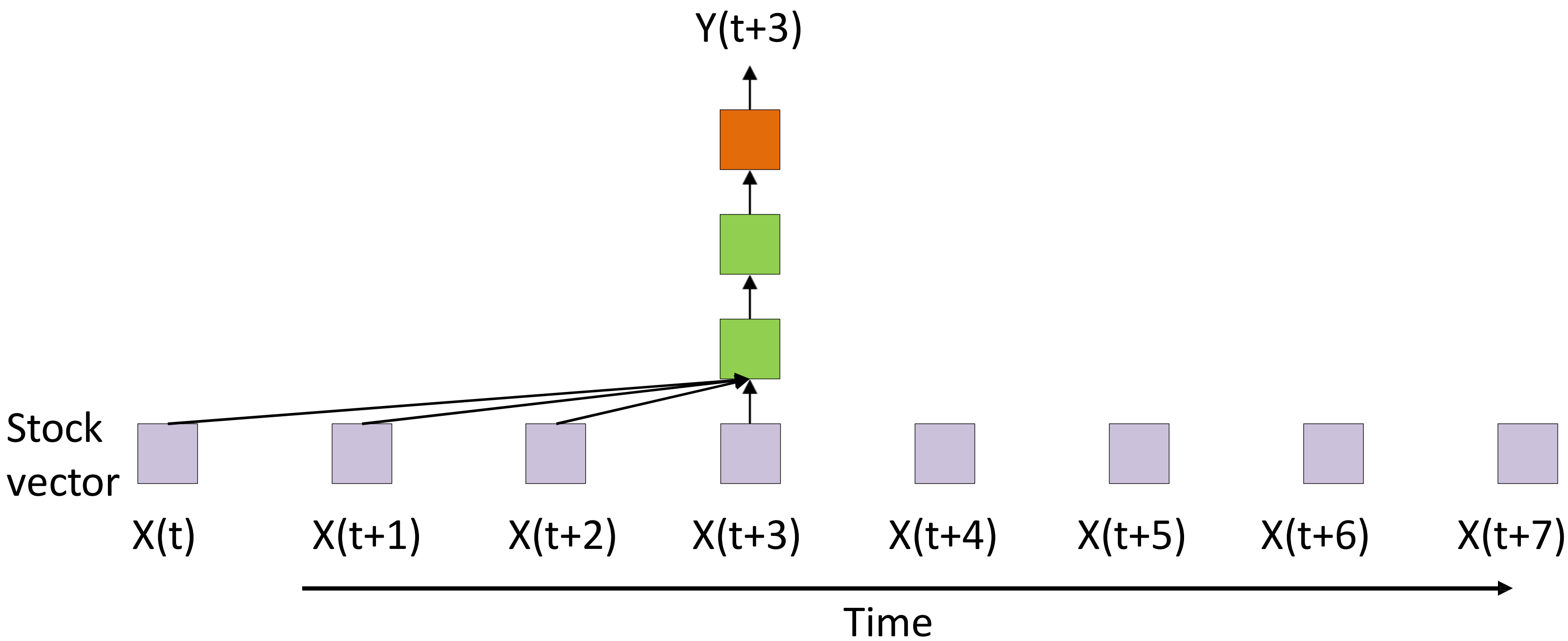
# The stock prediction problem…

To invest or not to invest?



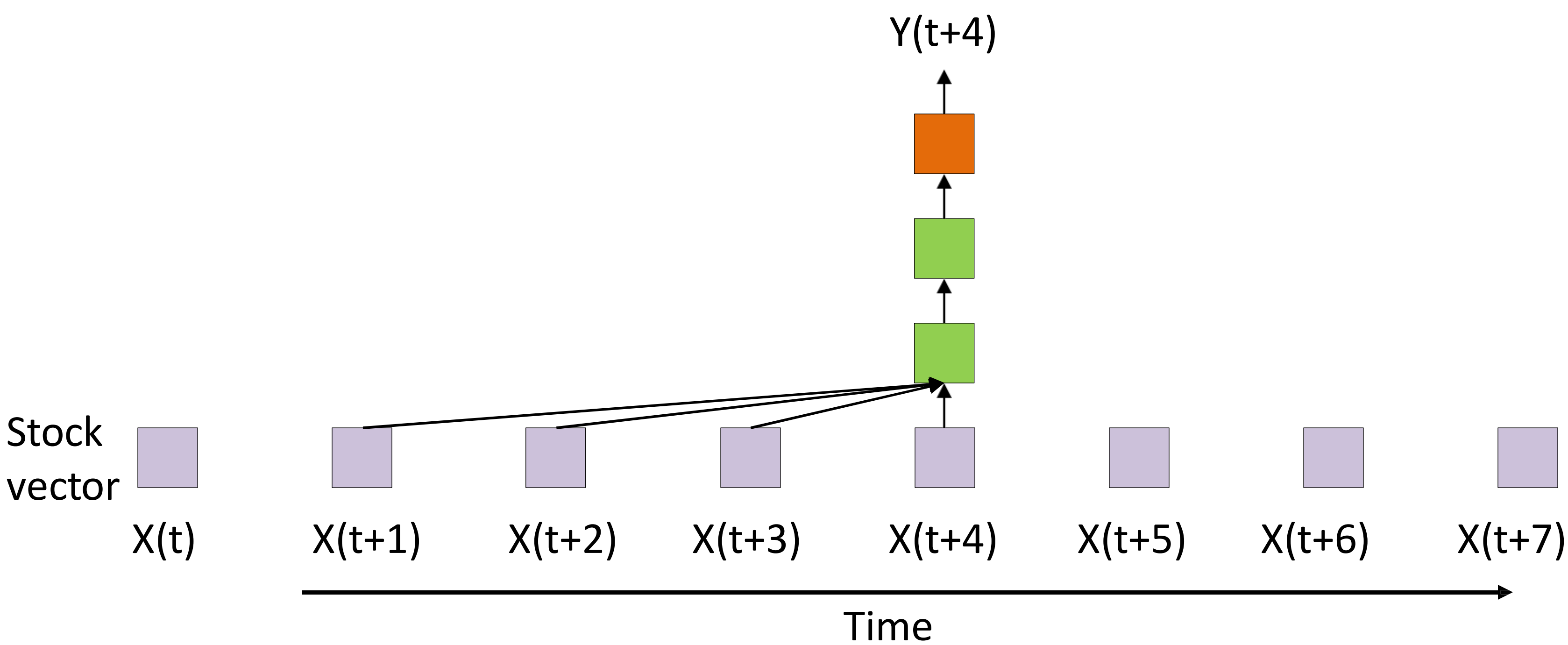

- Stock market
  - Must consider the series of stock values in the past several days to decide if it is wise to invest today
    - Ideally consider *all* of history

# The stock predictor network



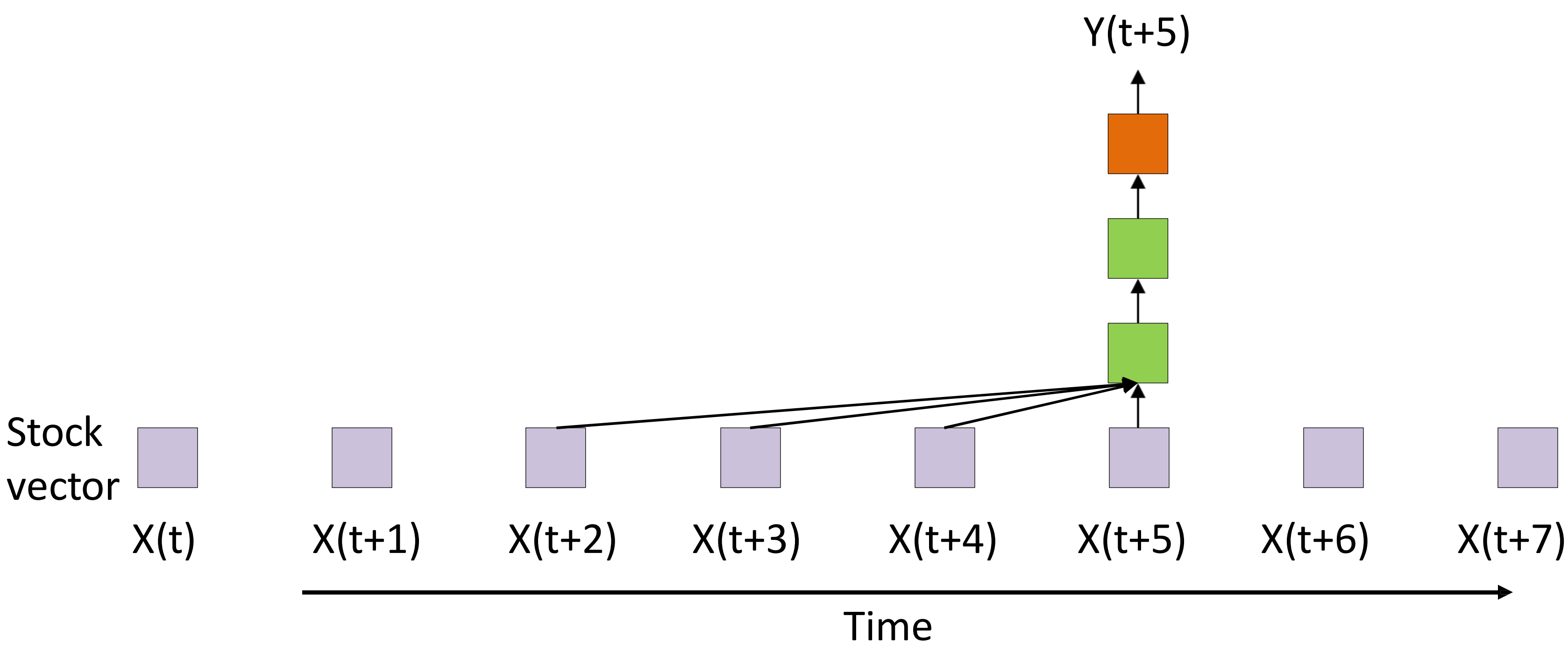

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

# The stock predictor network



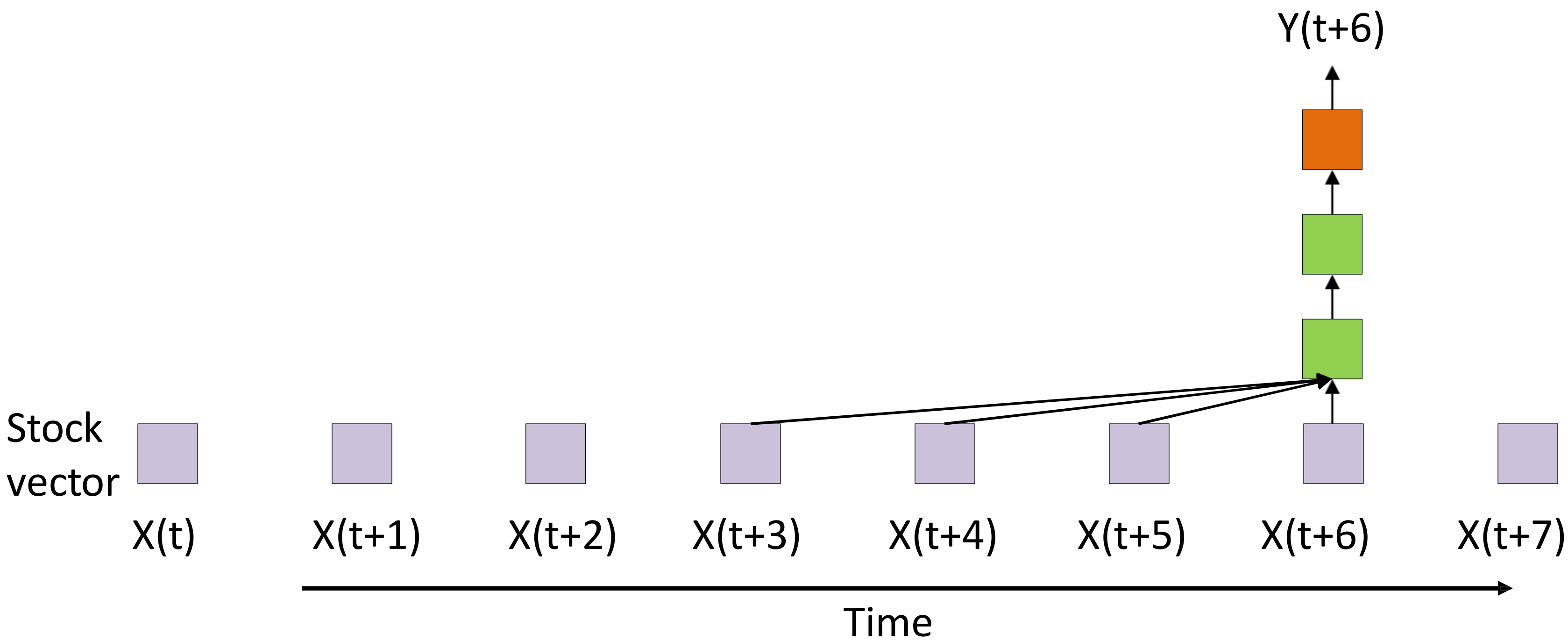

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

# The stock predictor network



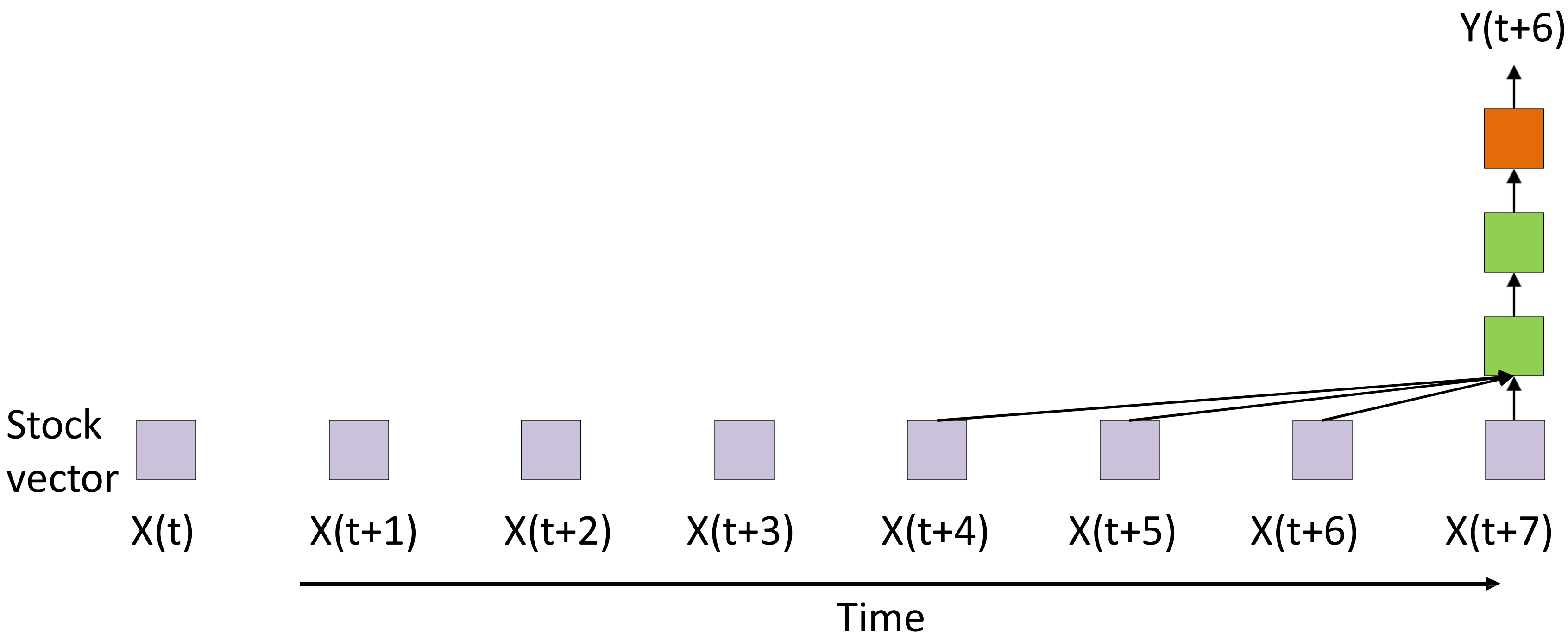

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

# The stock predictor network




- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

# The stock predictor network



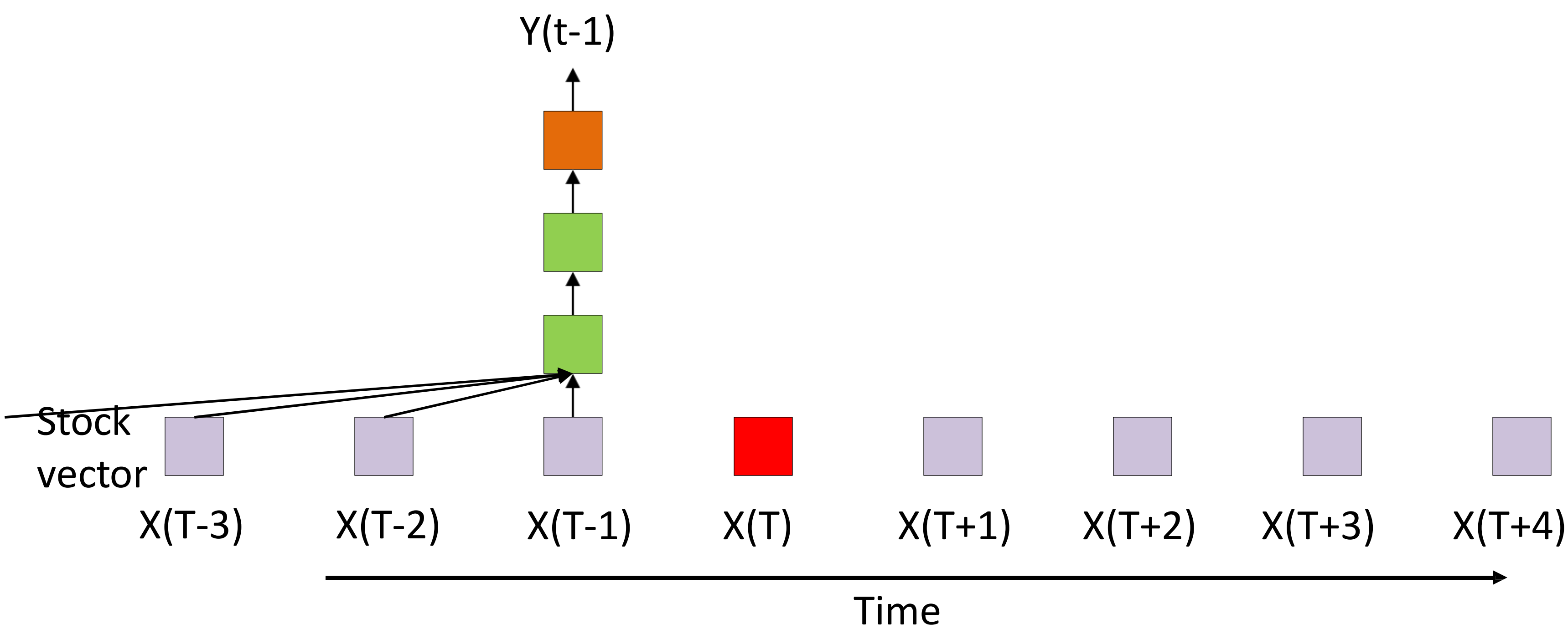

- The sliding predictor
  - Look at the last few days
  - This is just a convolutional neural net applied to series data
    - Also called a *Time-Delay neural network*

# Finite-response model

- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for $N$ days into the future
    - $N$ is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-N})$$

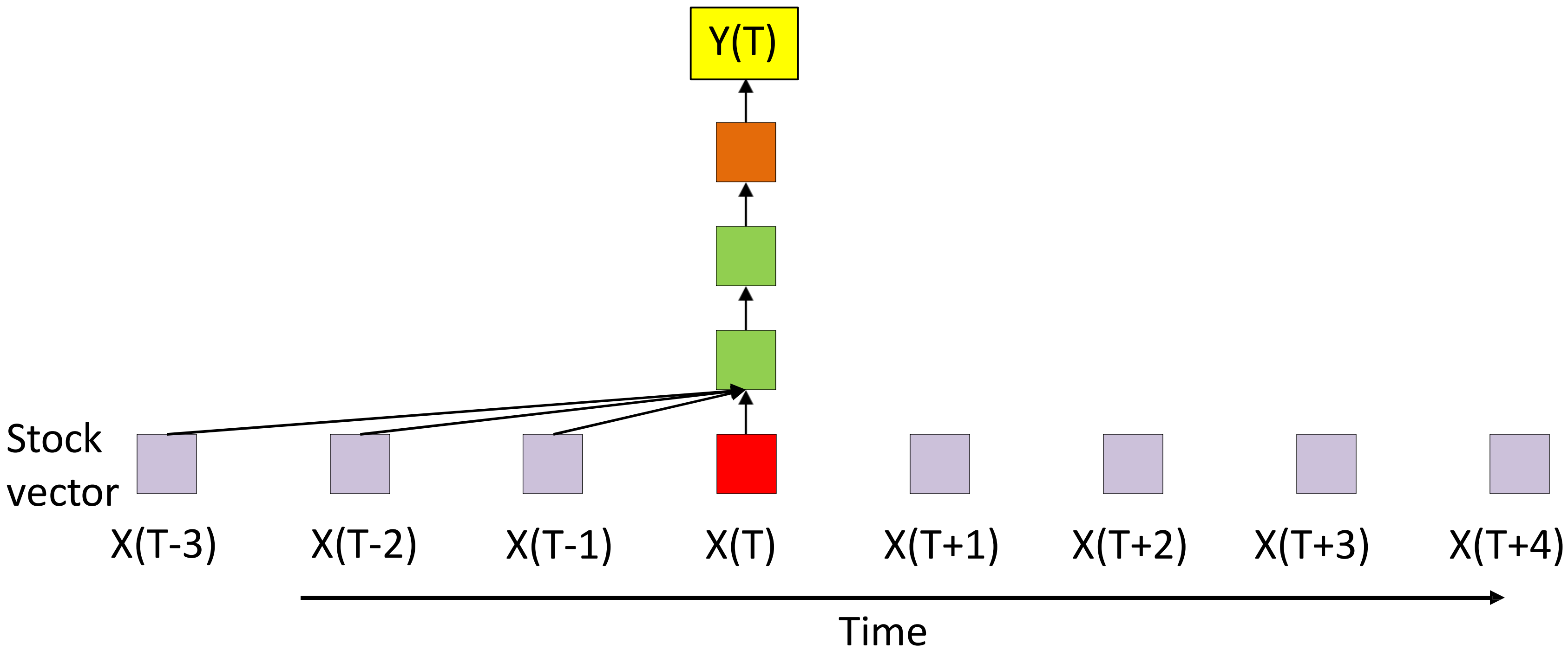
# The stock predictor




- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for $N$ days into the future
    - $N$ is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-N})$$

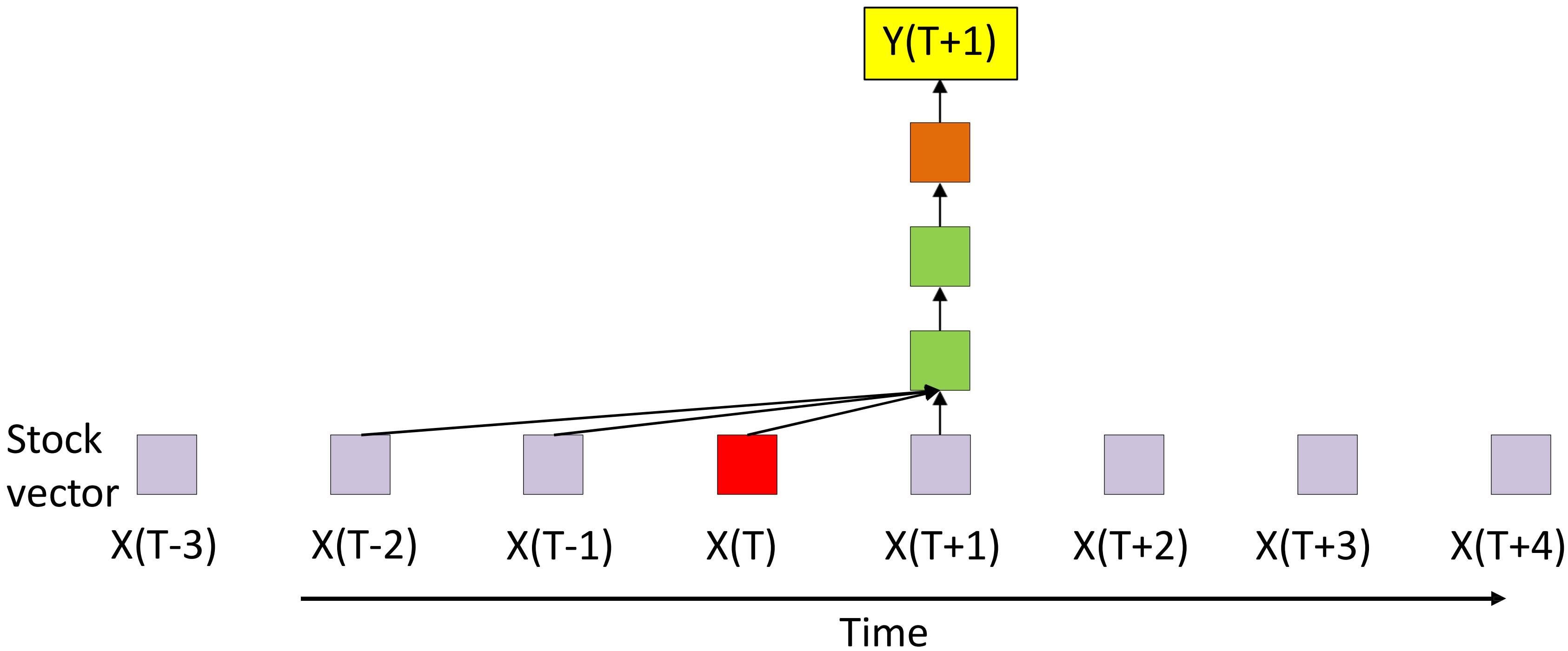
# The stock predictor




- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for $N$ days into the future
    - $N$ is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-N})$$

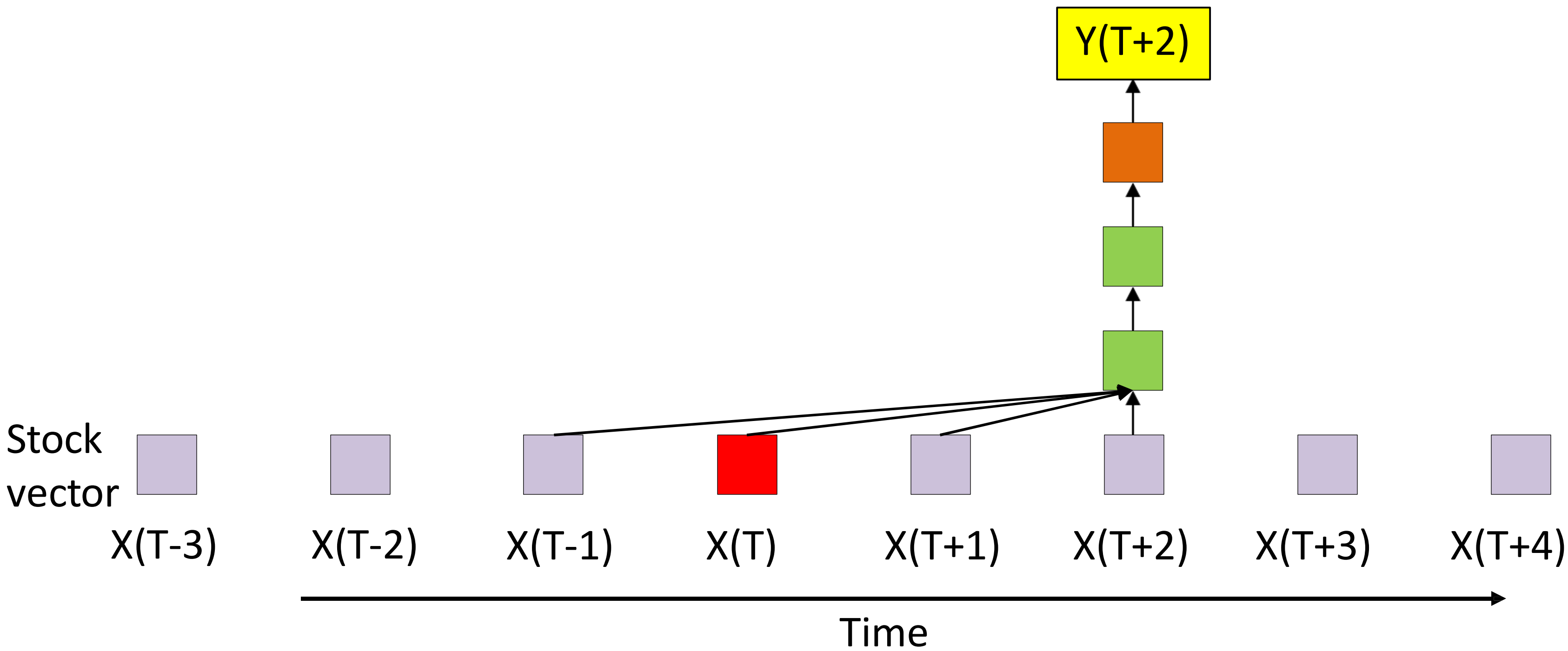
# The stock predictor




- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for $N$ days into the future
    - $N$ is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-N})$$

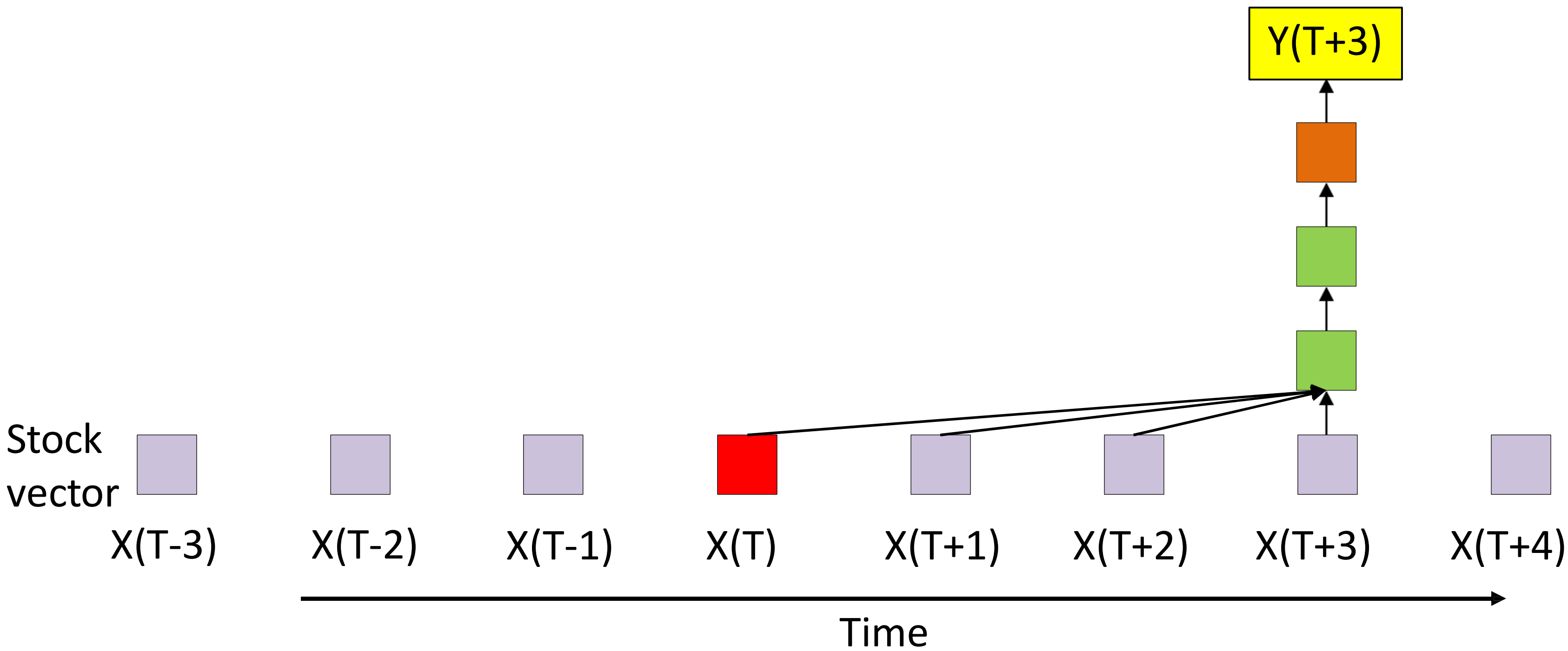
# The stock predictor




- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for $N$ days into the future
    - $N$ is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-N})$$

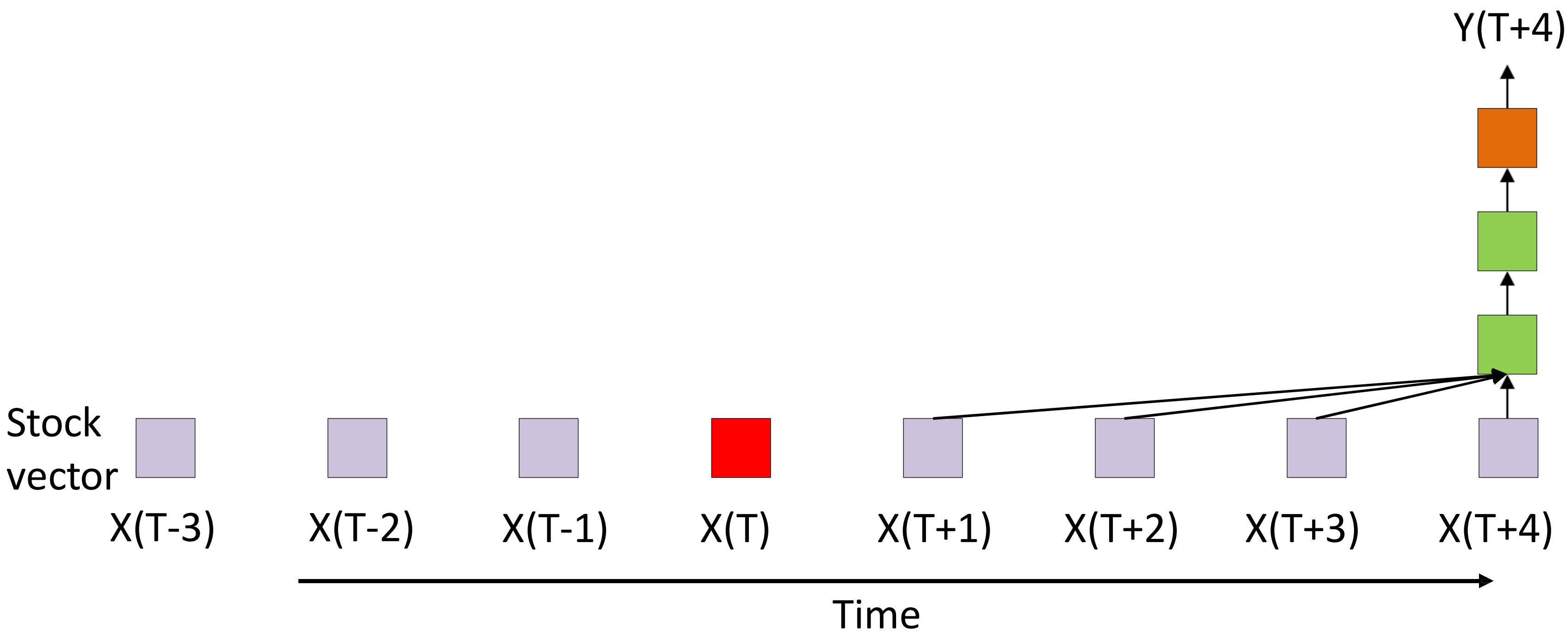
# The stock predictor




- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for $N$ days into the future
    - $N$ is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, \ldots, X_{t-N})$$

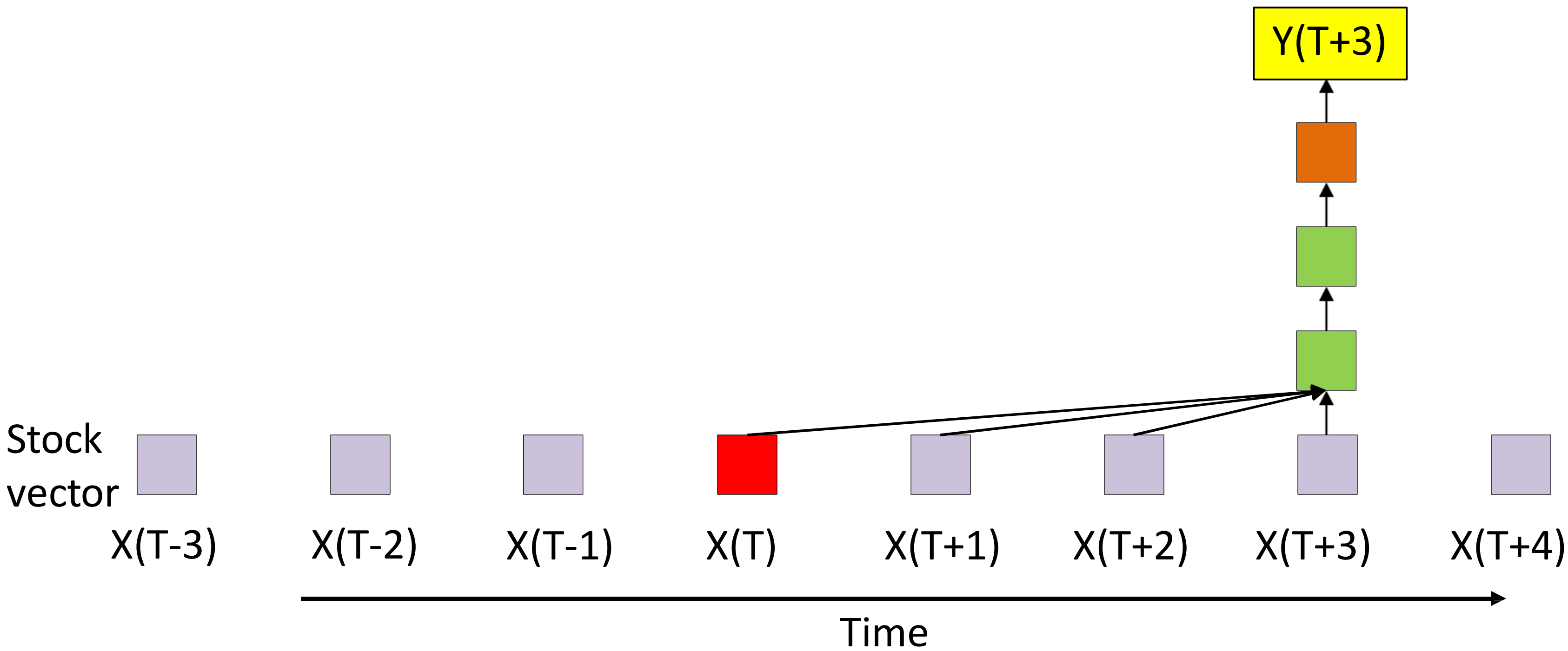
# The stock predictor




- This is a *finite response* system
  - Something that happens *today* only affects the output of the system for $N$ days into the future
    - $N$ is the *width* of the system

$$Y_t = f(X_t, X_{t-1}, \dots, X_{t-N})$$

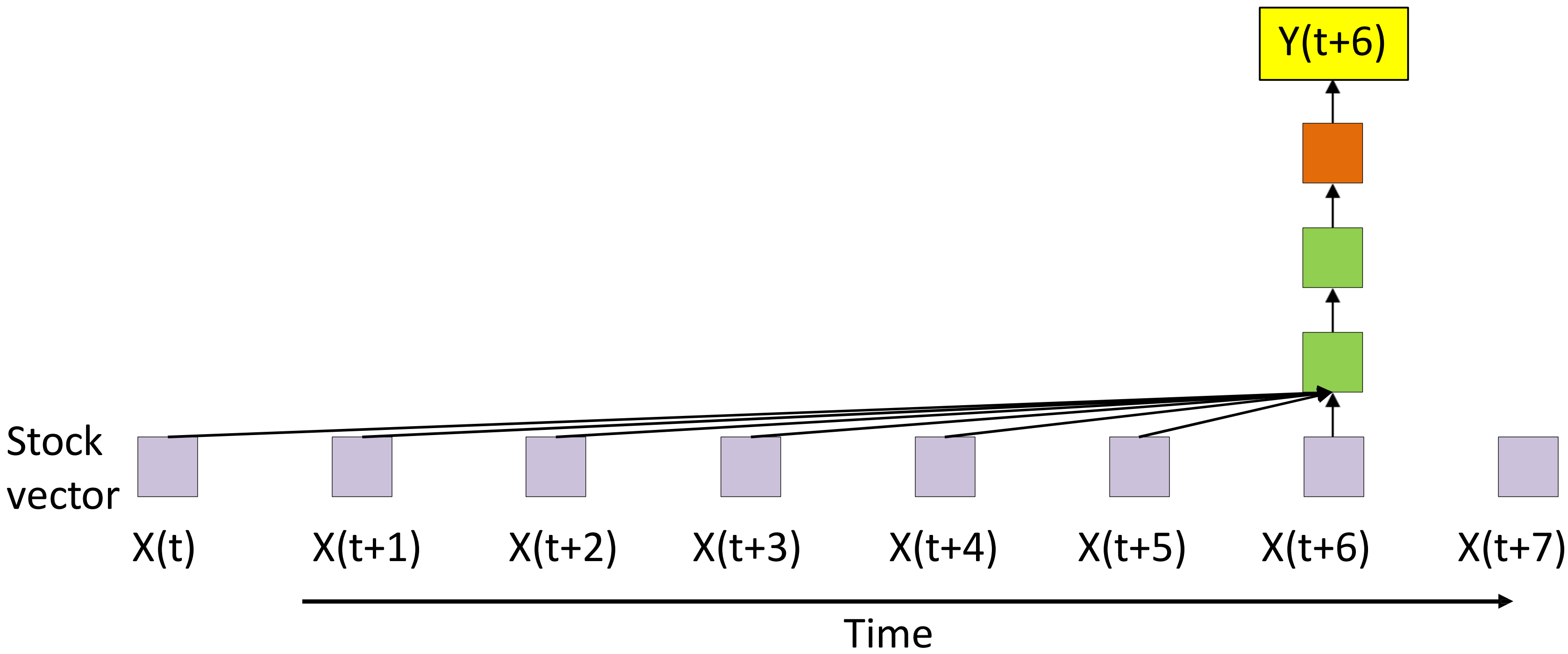
# Finite-response model




- Something that happens *today* only affects the output of the system for $N$ days into the future
  - **Predictions consider *N* days of history**

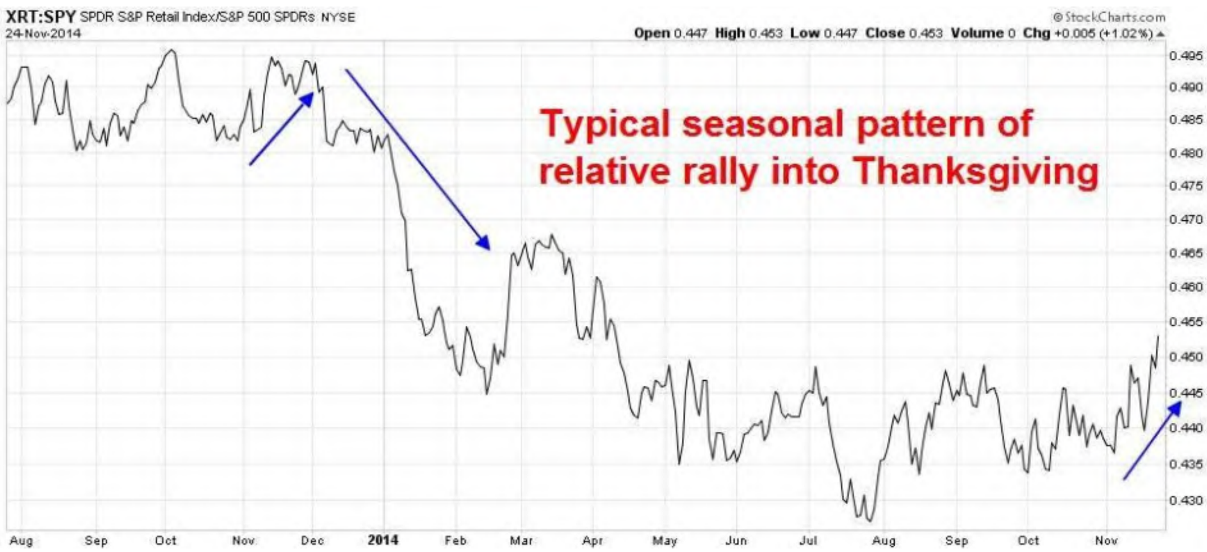
- To consider more of the past to make predictions, you must increase the "history" considered by the system

# Finite-response



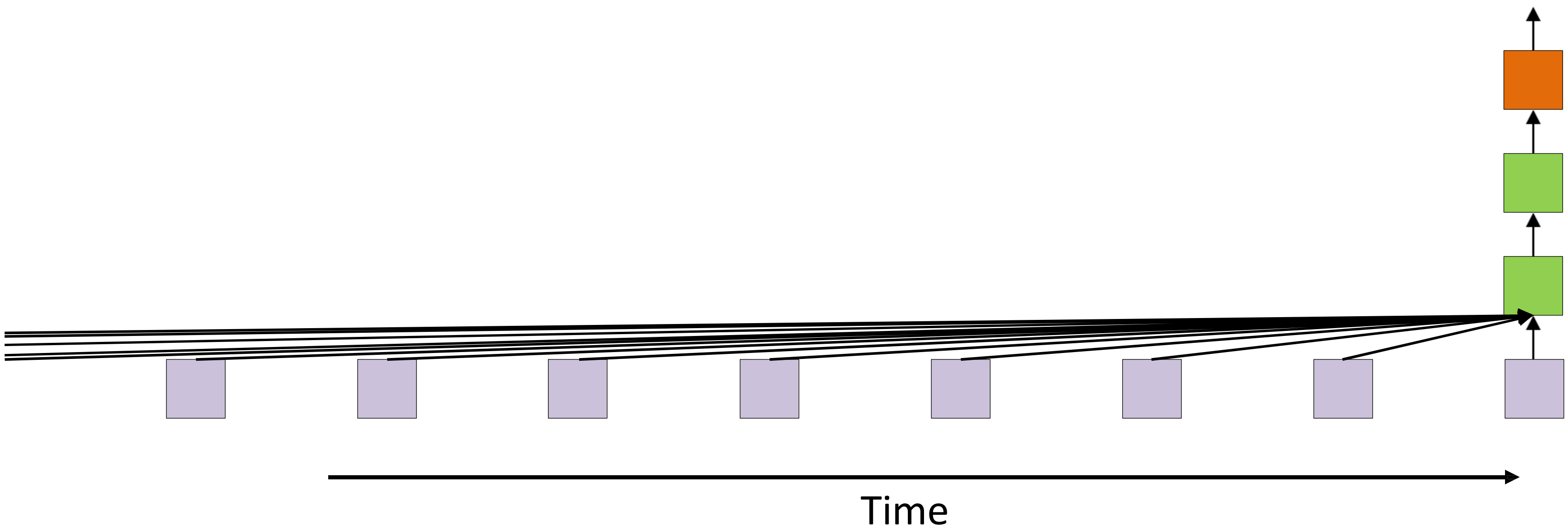

- Problem: Increasing the “history” makes the network more complex
  - No worries, we have the CPU and memory
    - Or do we?

# Systems often have long-term dependencies




- Longer-term trends –
  - Weekly trends in the market
  - Monthly trends in the market
  - Annual trends
  - Though longer historic tends to affect us less than more recent events..

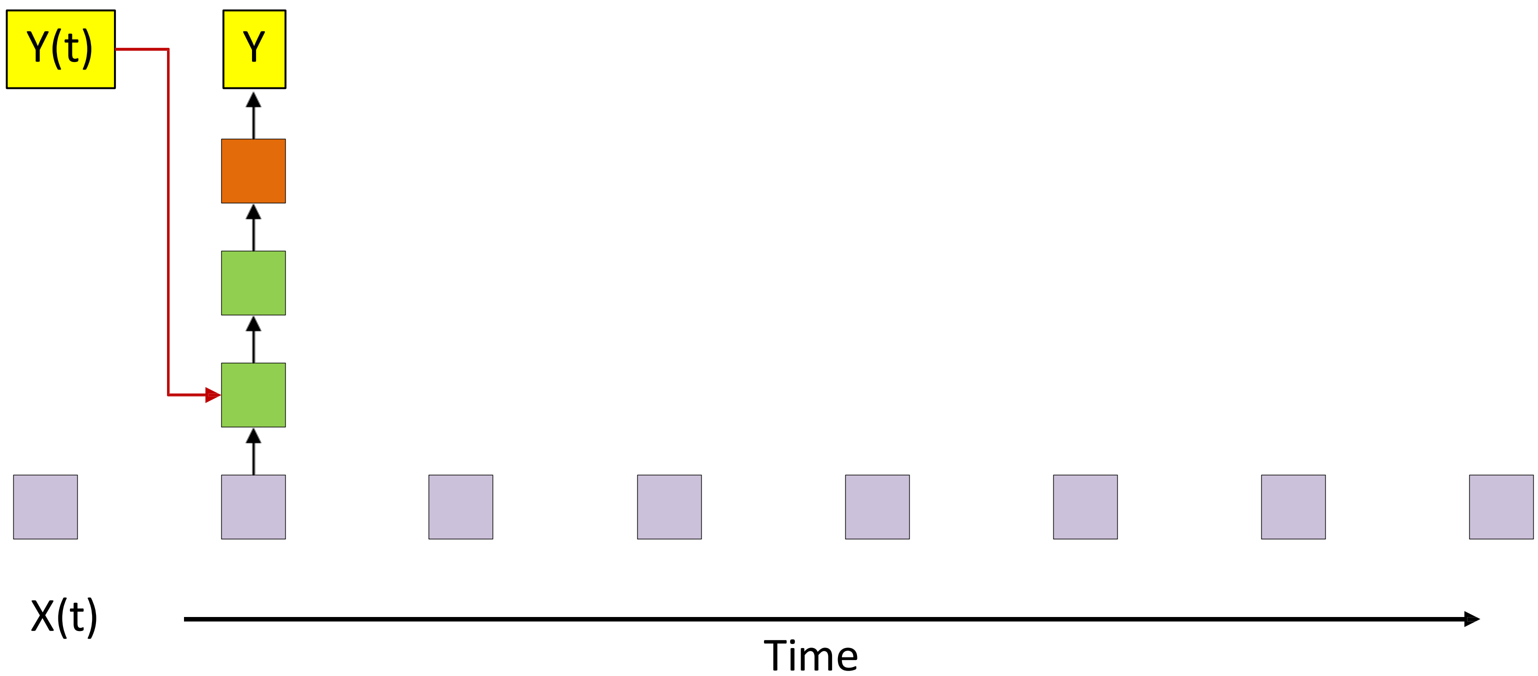
# We want *infinite* memory




- Required: *Infinite* response systems
  - What happens today can continue to affect the output forever
    - Possibly with weaker and weaker influence

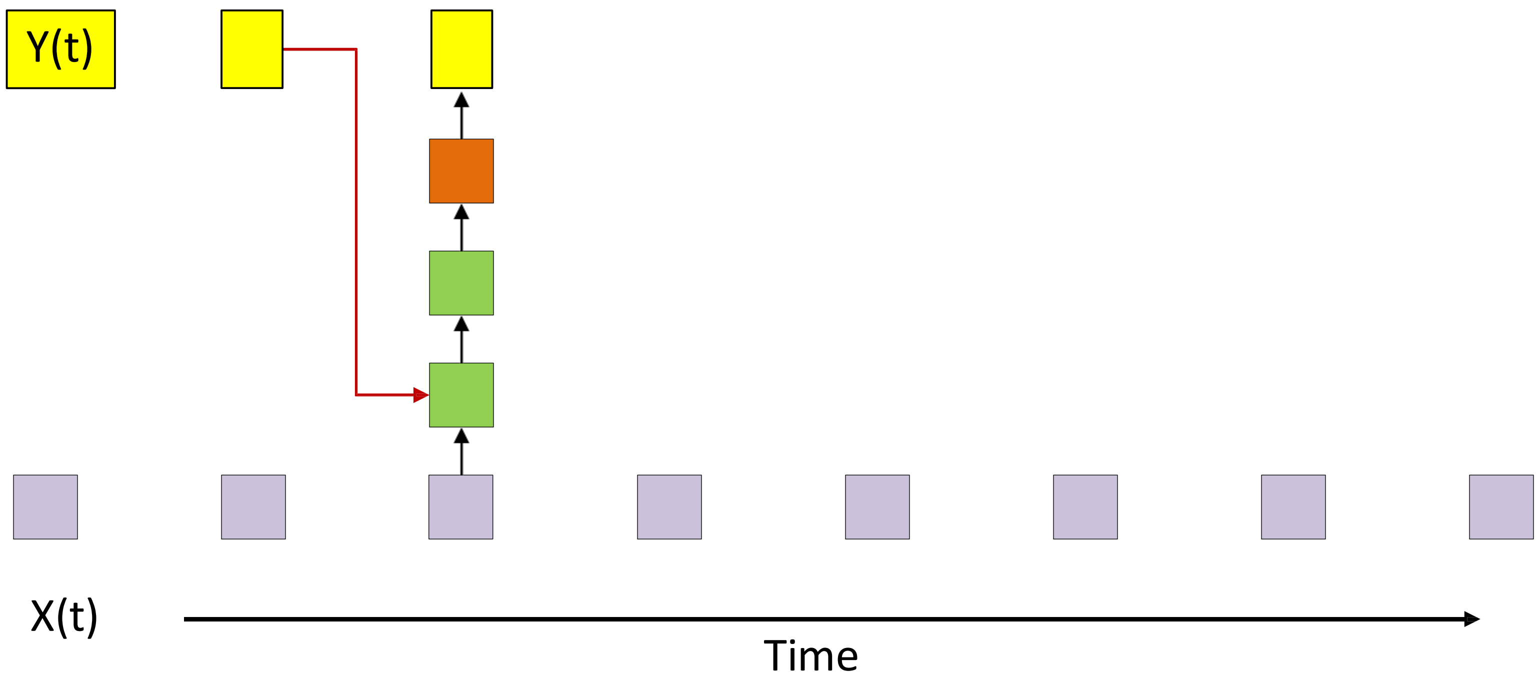
$$Y_t = f(X_t, X_{t-1}, \ldots, X_{t-\infty})$$

# A one-tap NARX network



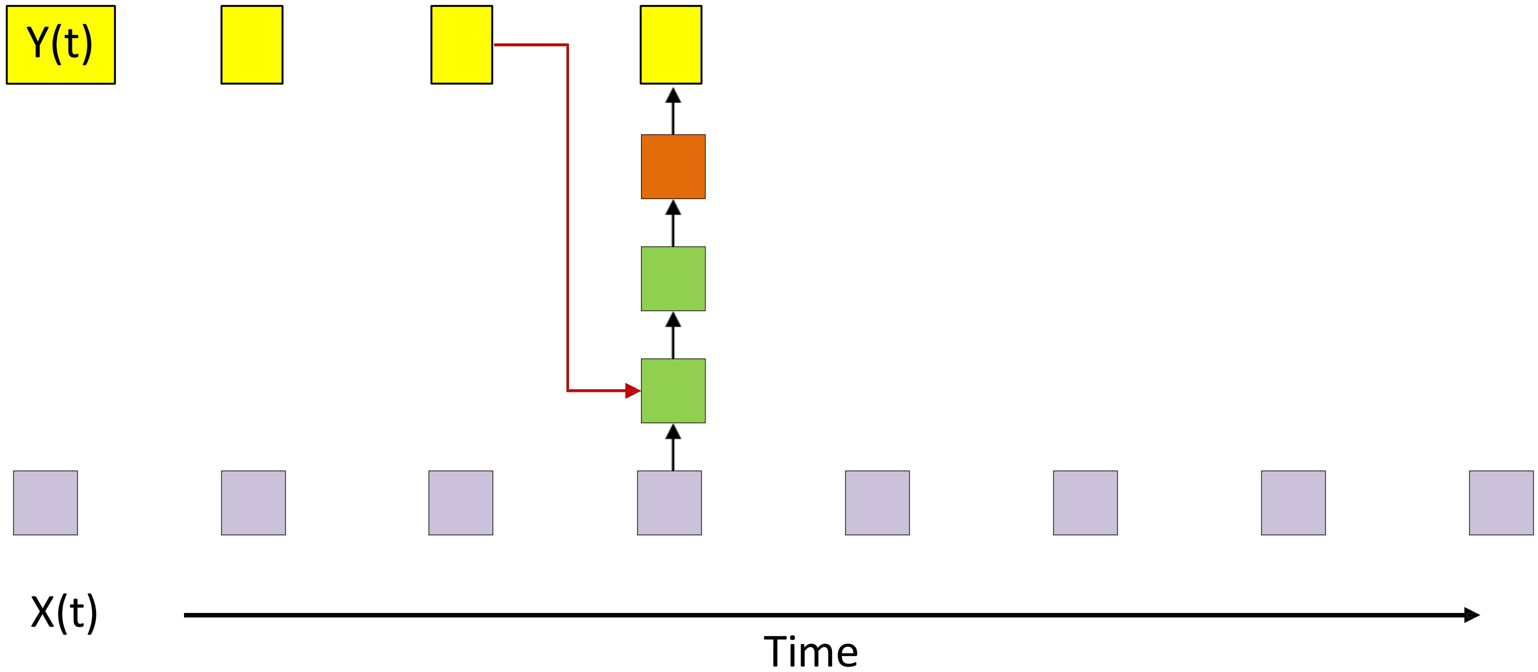

- A NARX net with recursion from the output

# A one-tap NARX network



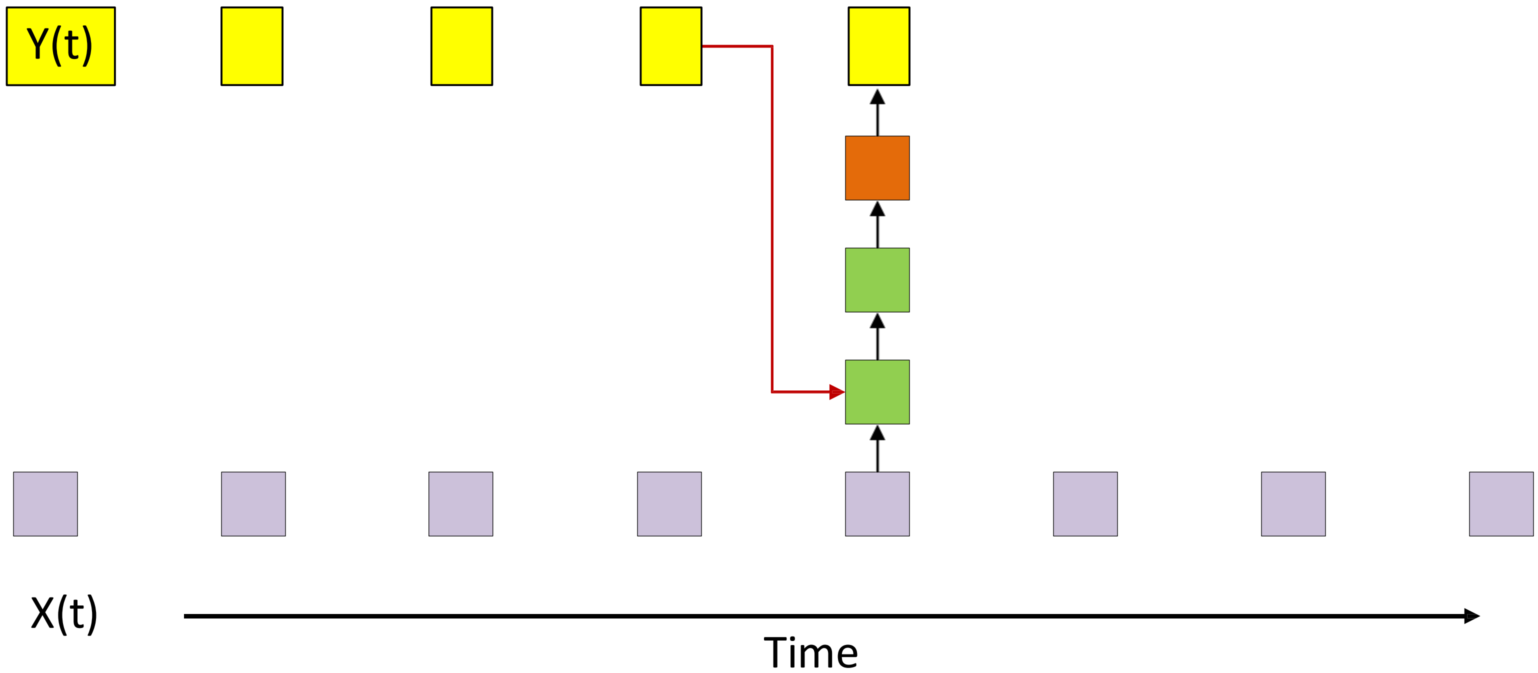

- A NARX net with recursion from the output

# A one-tap NARX network



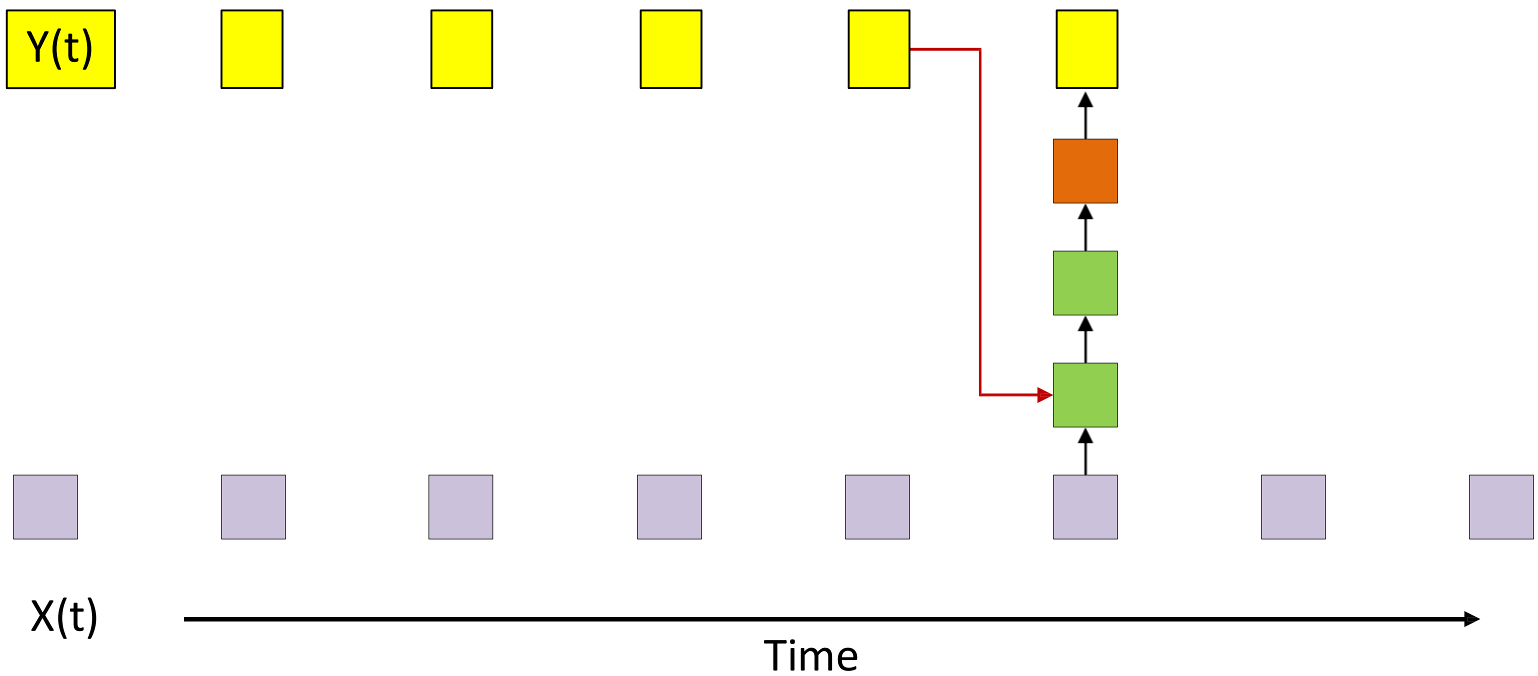

- A NARX net with recursion from the output

# A one-tap NARX network



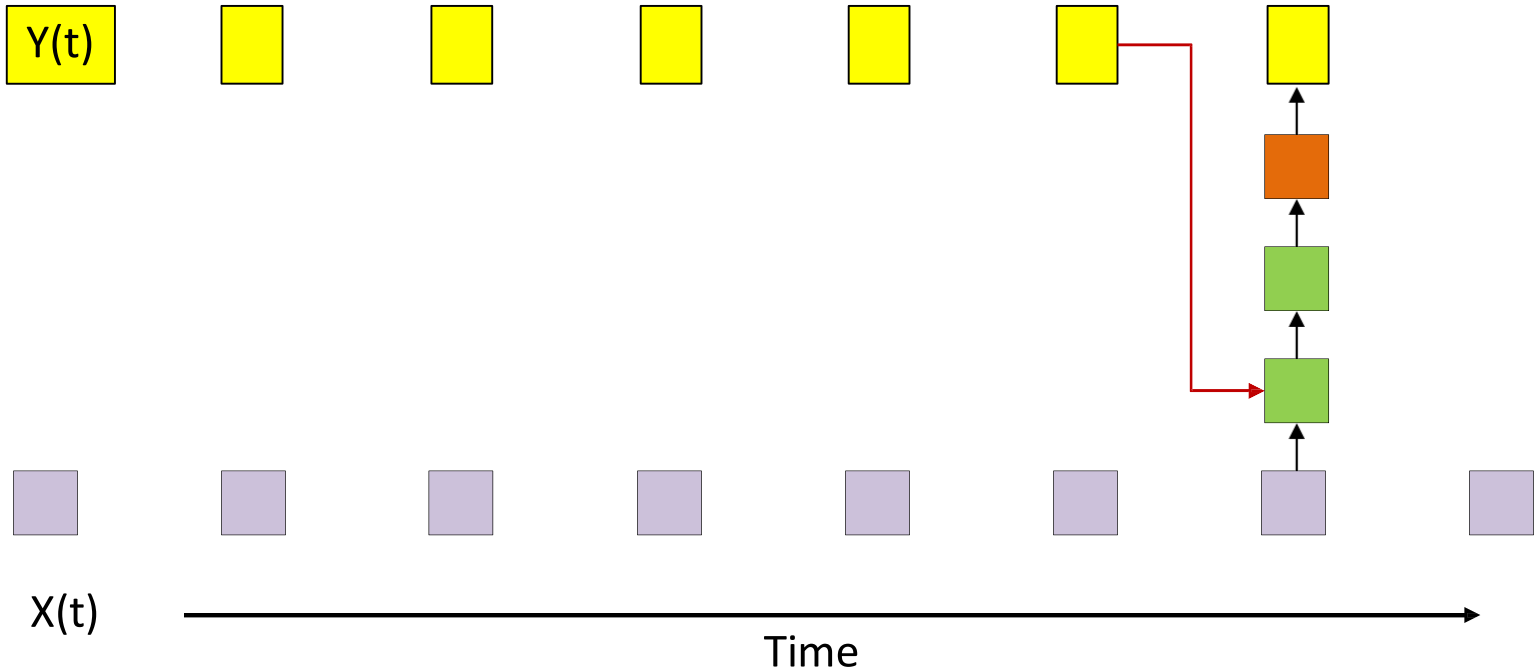

- A NARX net with recursion from the output

# A one-tap NARX network



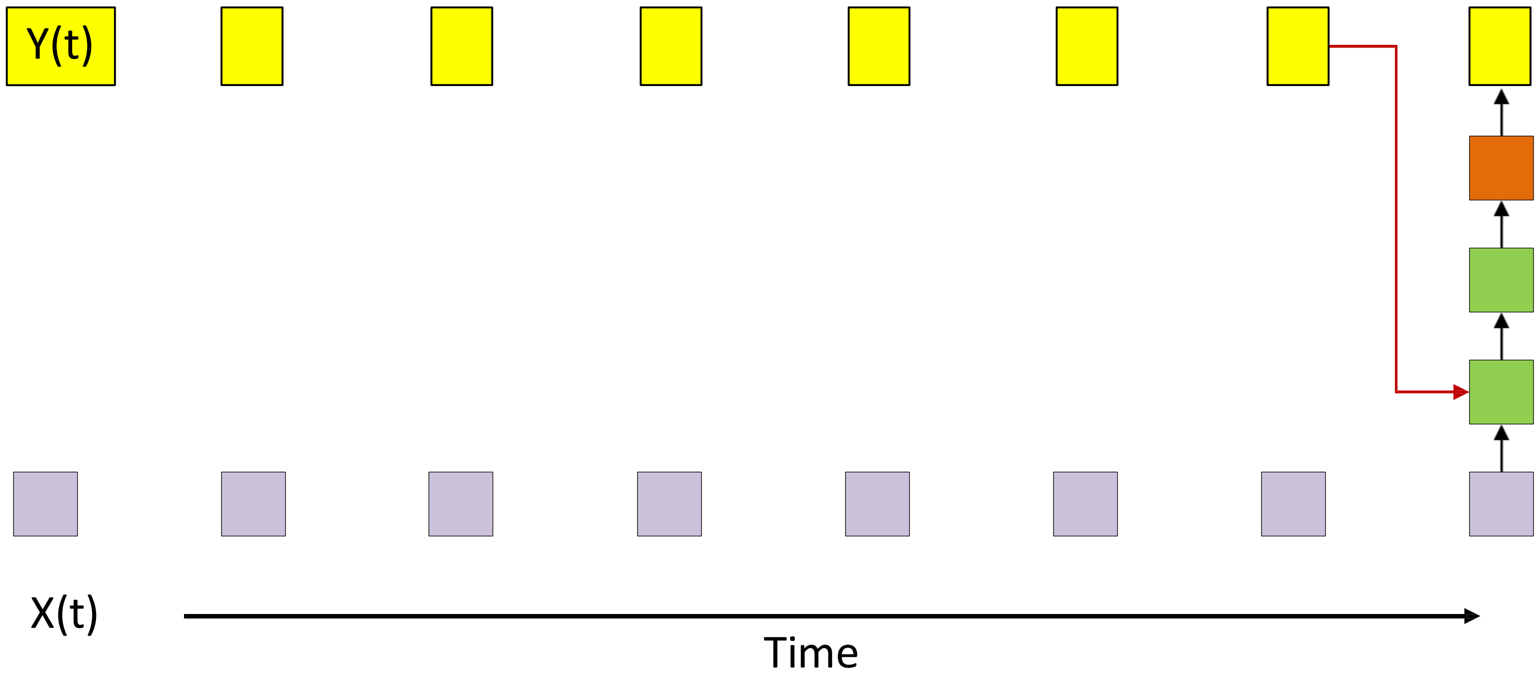

- A NARX net with recursion from the output

# A one-tap NARX network



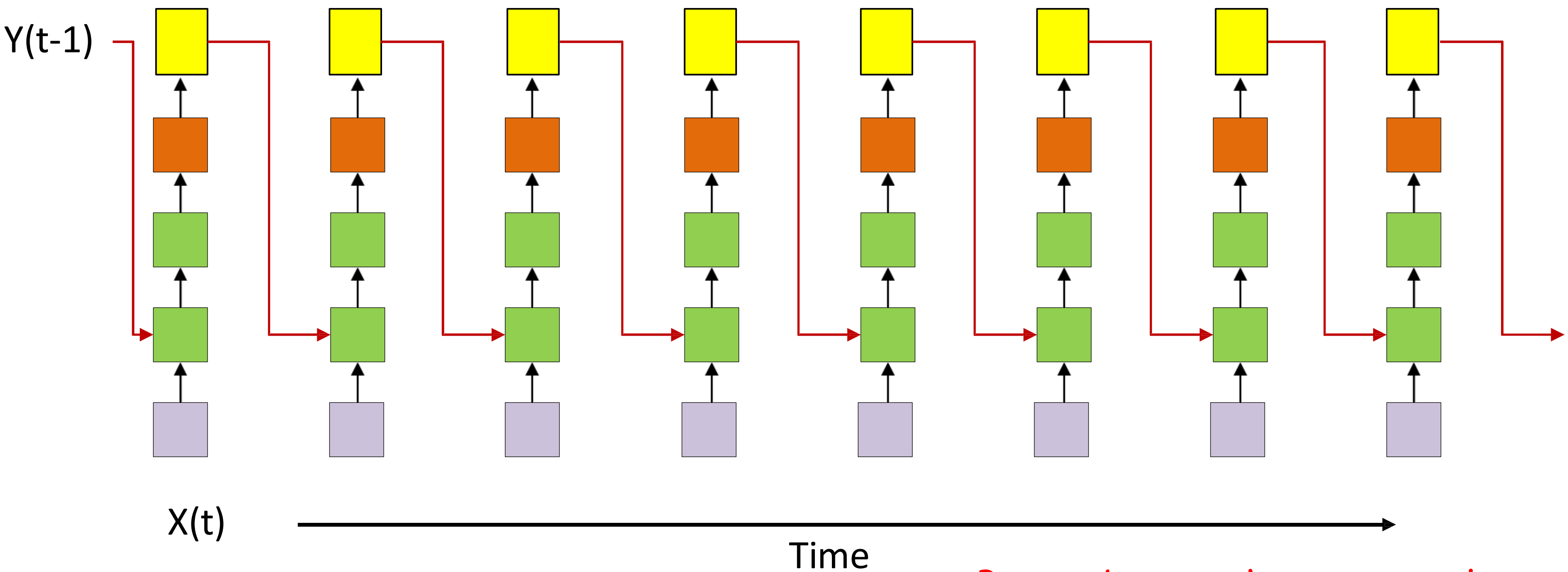

- A NARX net with recursion from the output

# A one-tap NARX network




- A NARX net with recursion from the output

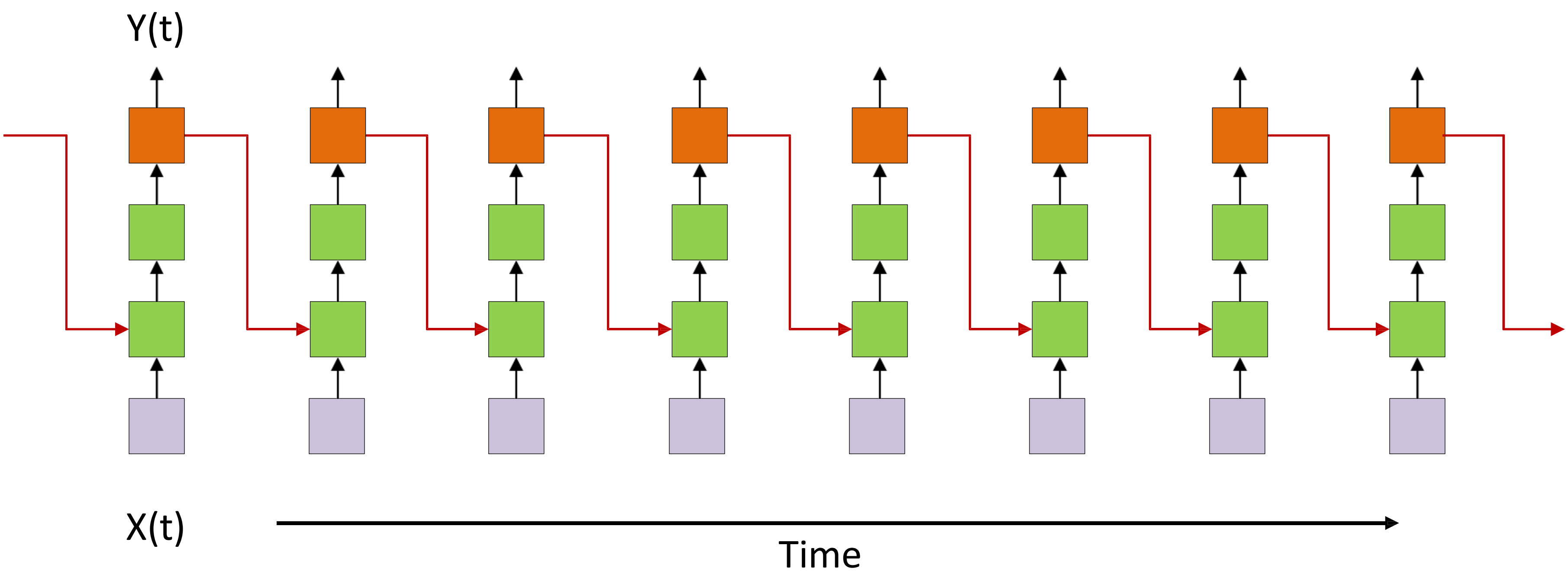
# A more complete representation




Brown boxes show output layers
Yellow boxes are outputs

- A NARX net with recursion from the output
- Showing all computations
- All columns are identical
- *An input at t=0 affects outputs forever*

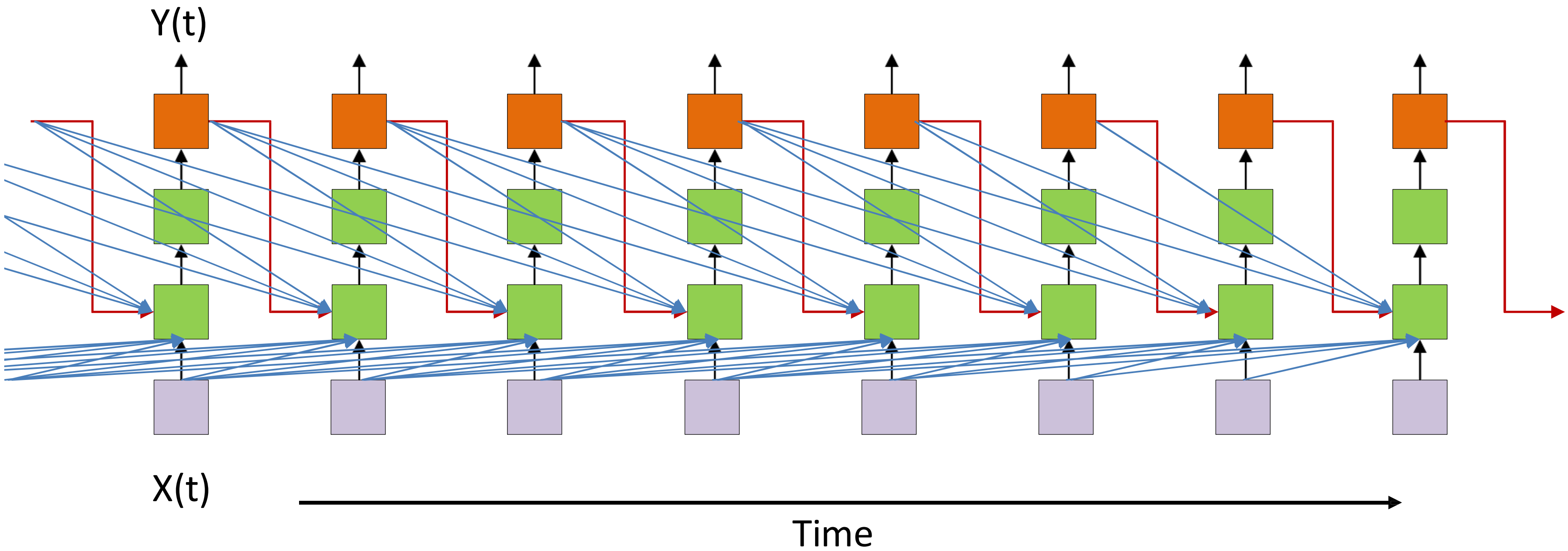
# Same figure redrawn




Brown boxes show output layers
All outgoing arrows are the same output

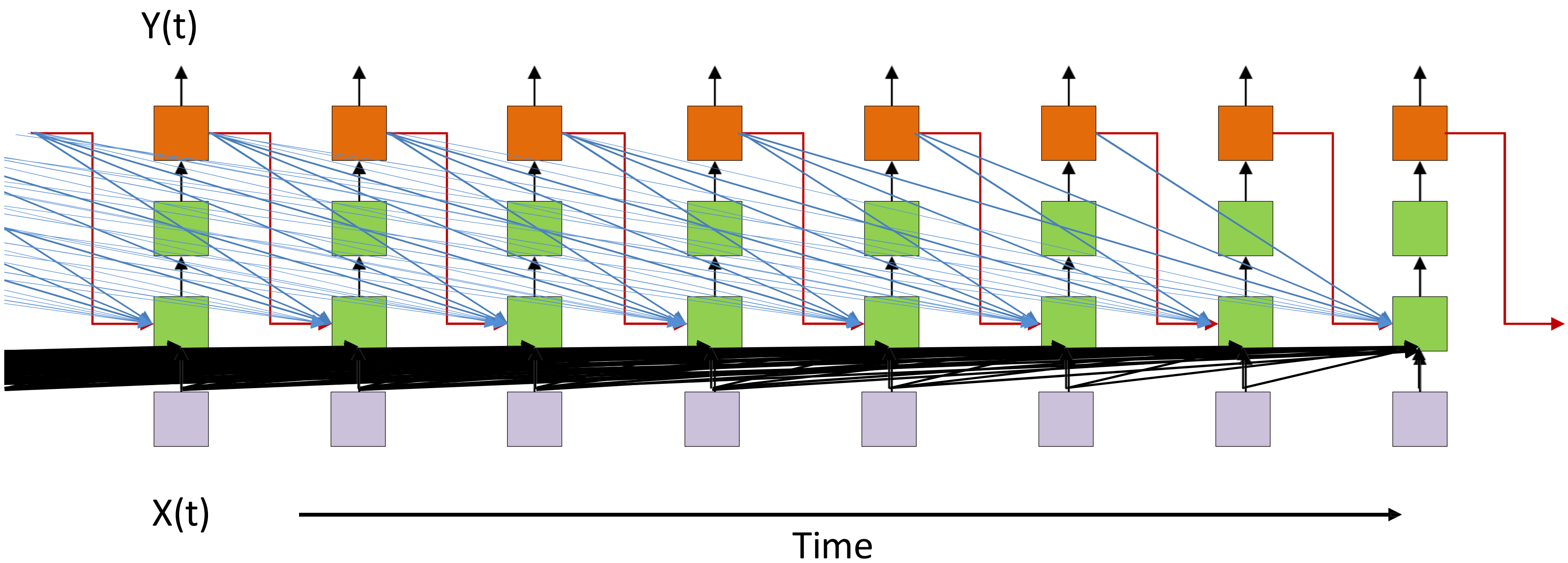
- A NARX net with recursion from the output
- Showing all computations
- All columns are identical
- *An input at t=0 affects outputs forever*

# A more generic NARX network



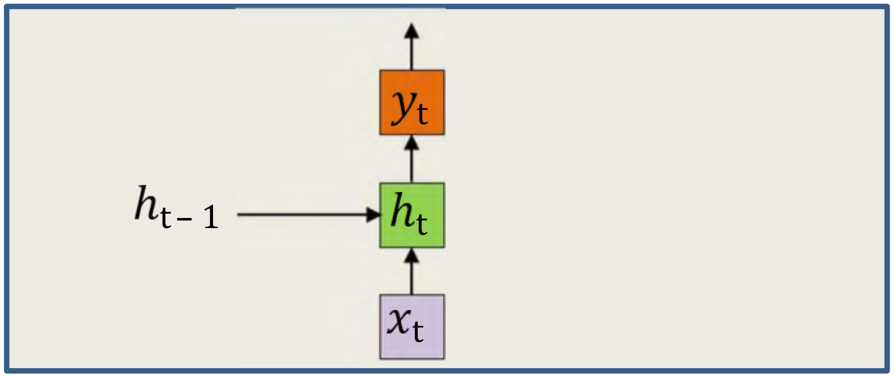

- The output $Y_t$ at time $t$ is computed from the past $K$ outputs $Y_{t-1}, \ldots, Y_{t-K}$ and the current and past $L$ inputs $X_t, \ldots, X_{t-L}$

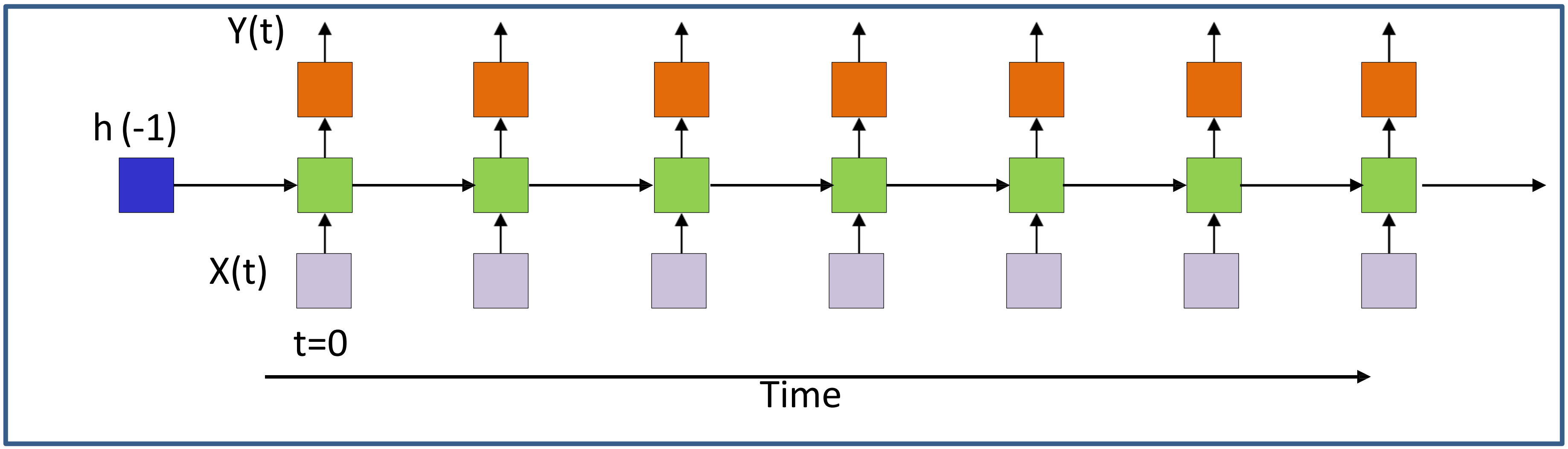
# A "complete" NARX network




- The output $Y_t$ at time $t$ is computed from *all* past outputs and *all* inputs until time $t$
  - Not really a practical model

# The simple state-space model

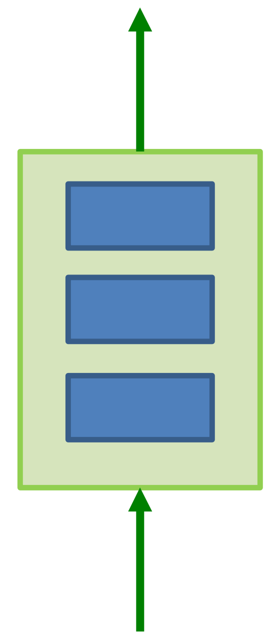



- The state (green) at any time is determined by the input at that time, and the state at the previous time
- *An input at t=0 affects outputs forever*
- Also known as a recurrent neural net

# An alternate model for infinite response systems: the state-space model

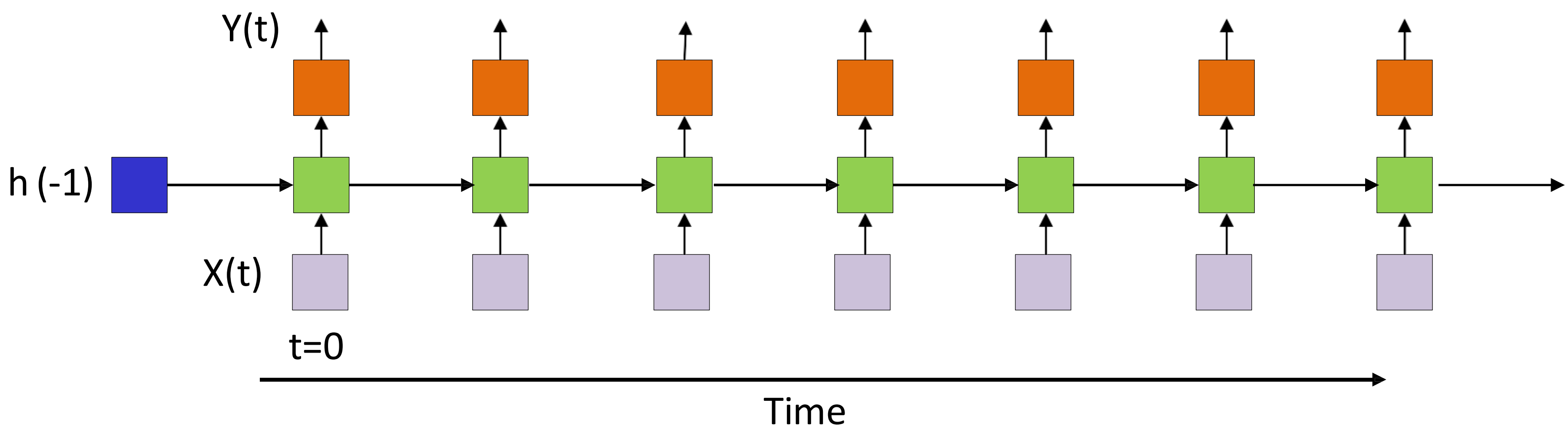
$$h_t = f(x_t, h_{t-1})$$
$$y_t = g(h_t)$$



- $h_t$ is the *state* of the network
- Need to define initial state $h_{-1}$

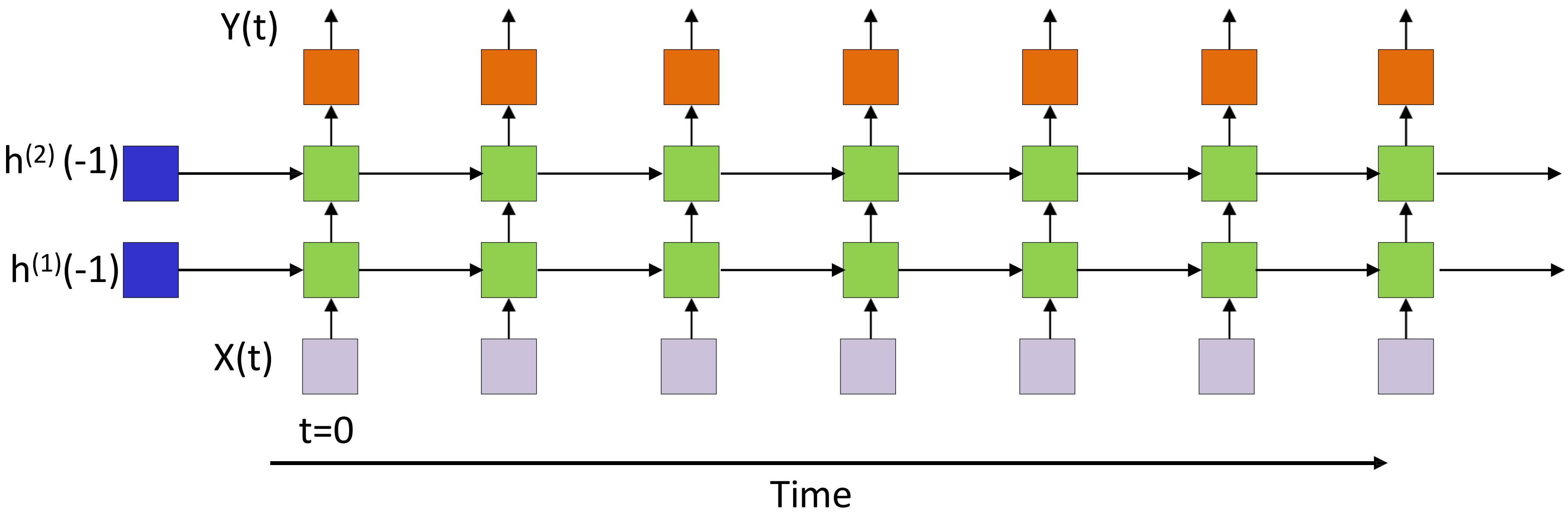
- The state an be arbitrarily complex

# Single hidden layer RNN



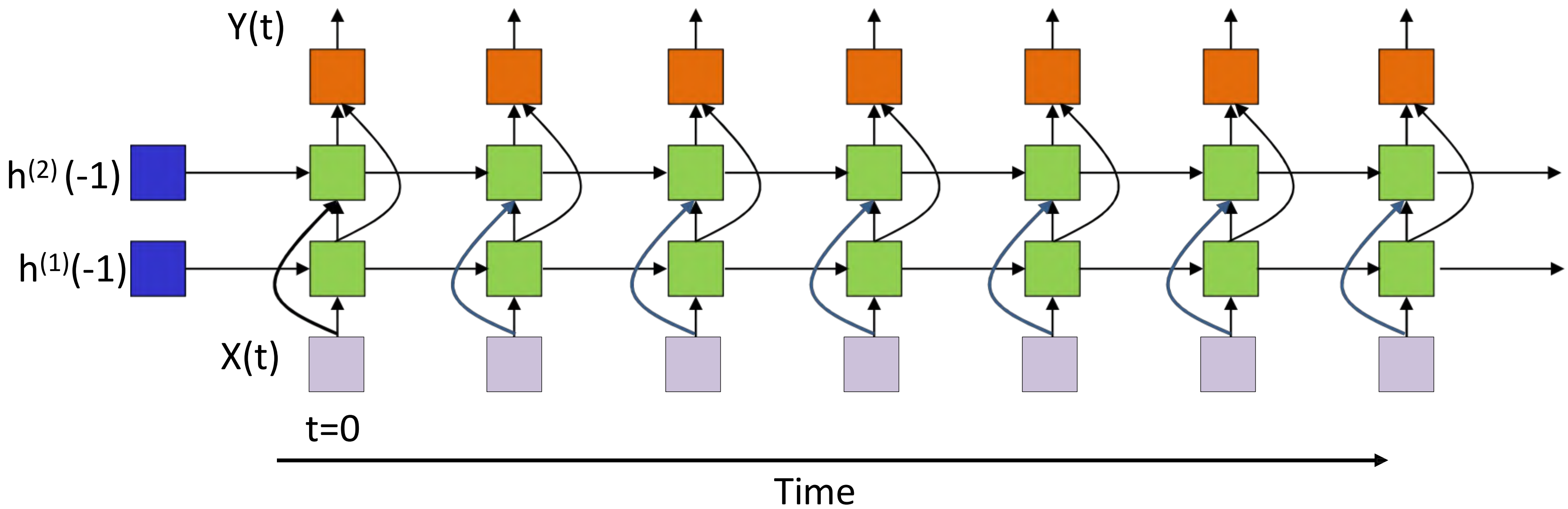

- Recurrent neural network
- All columns are identical
- *An input at t=0 affects outputs forever*

# Multiple recurrent layer RNN



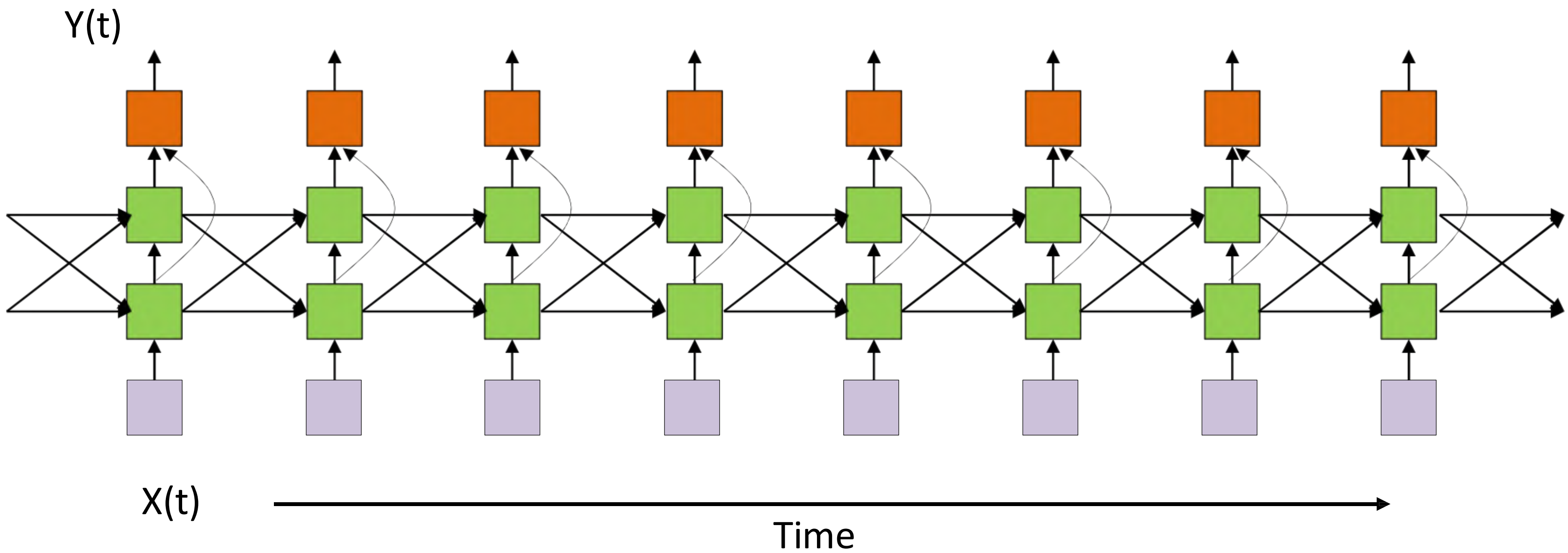

- Recurrent neural network
- All columns are identical
- *An input at t=0 affects outputs forever*

# Multiple recurrent layer RNN



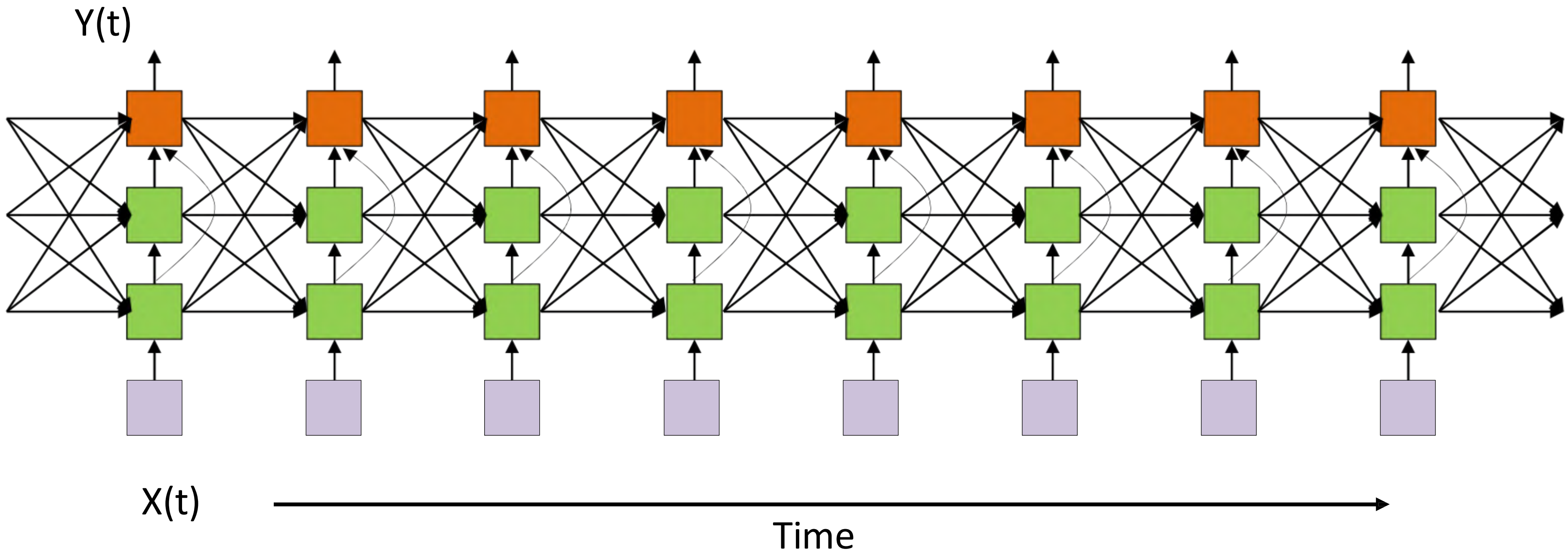

- We can also have skips..

# A more complex state



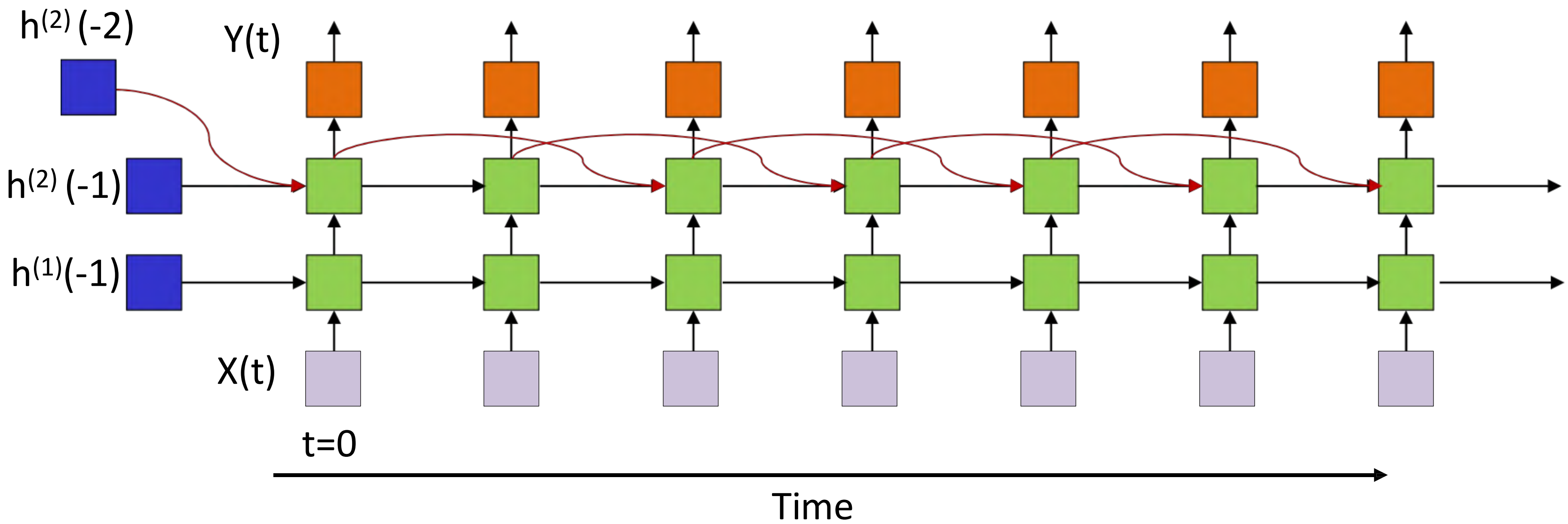

- All columns are identical
- *An input at t=0 affects outputs forever*

# Or the network may be even more complicated



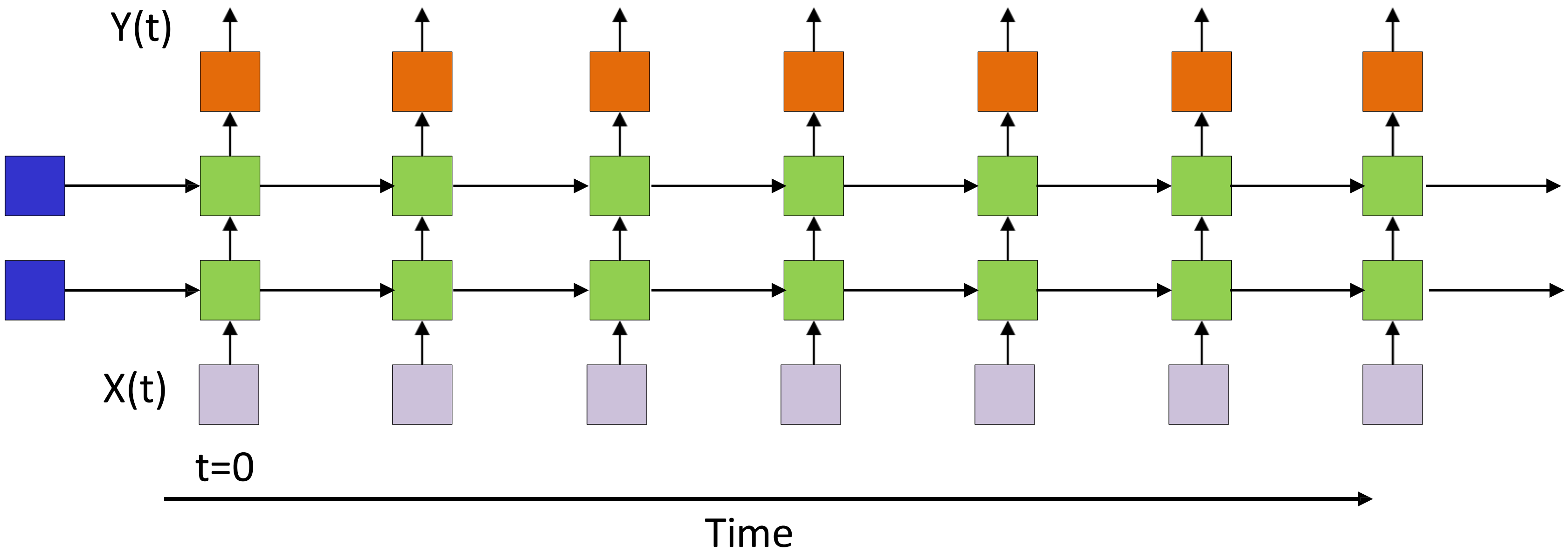

- Shades of NARX
- All columns are identical
- *An input at t=0 affects outputs forever*

# Generalization with other recurrences



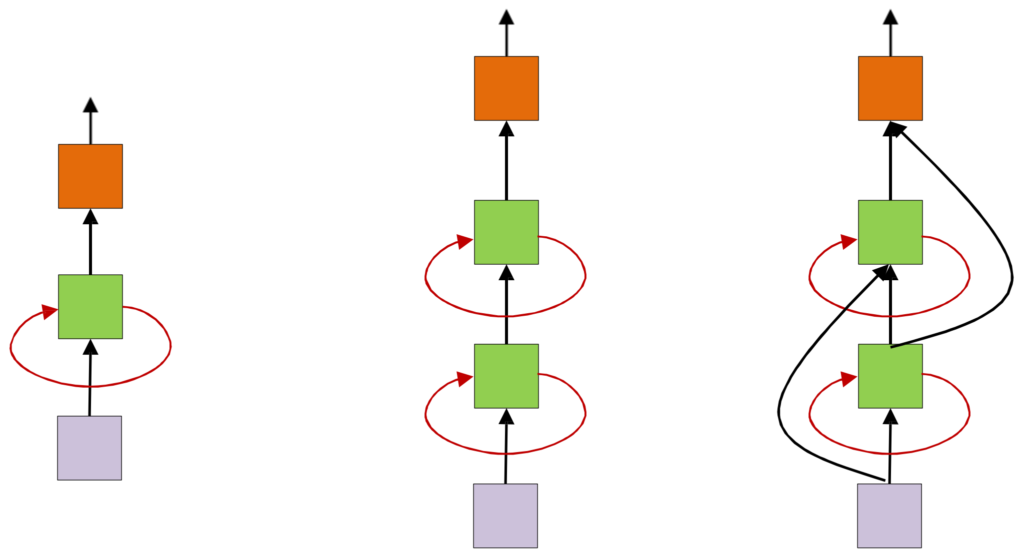

- All columns (including incoming edges) are identical

# The simplest structures are most popular



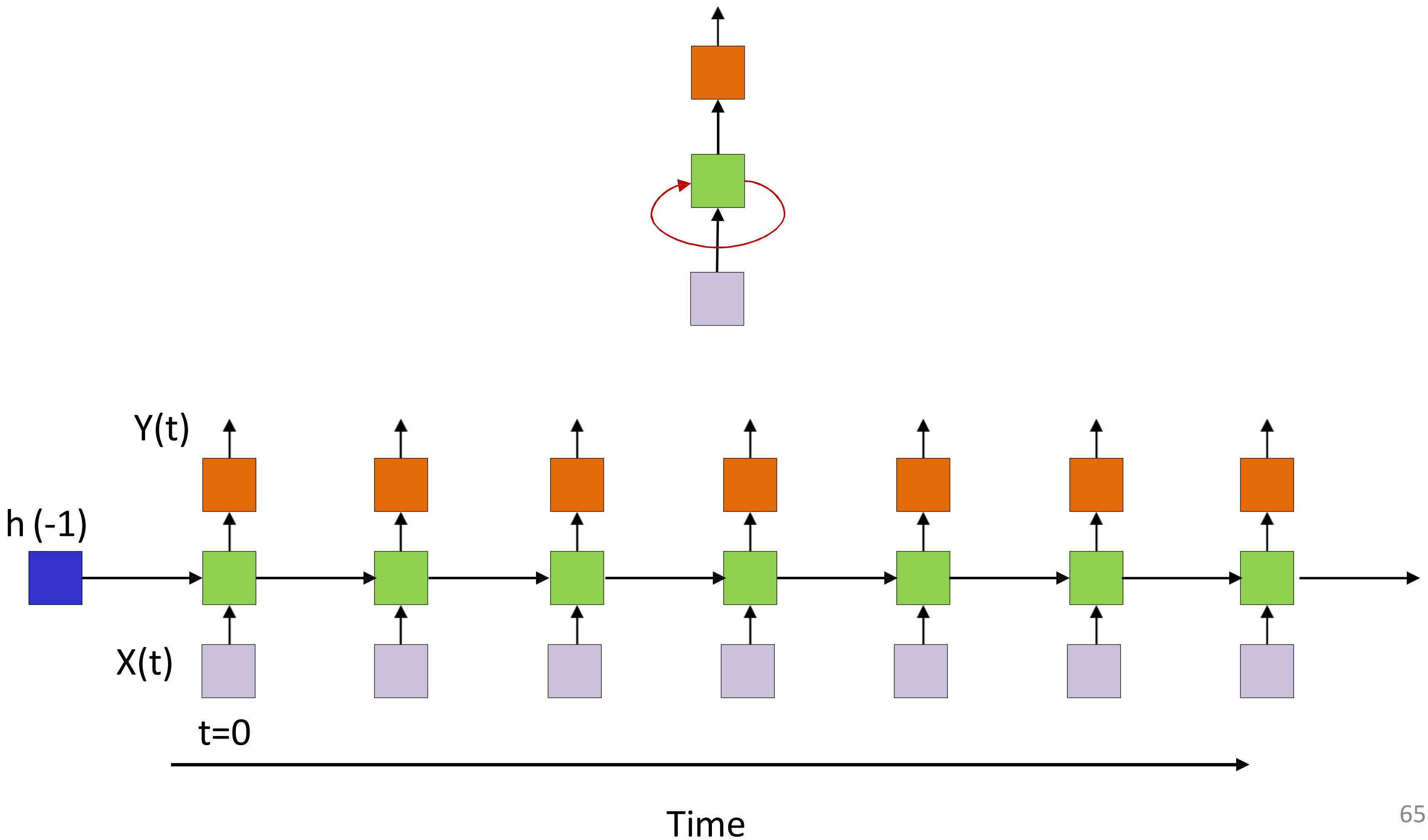

- Recurrent neural network
- All columns are identical
- *An input at t=0 affects outputs forever*

# A Recurrent Neural Network

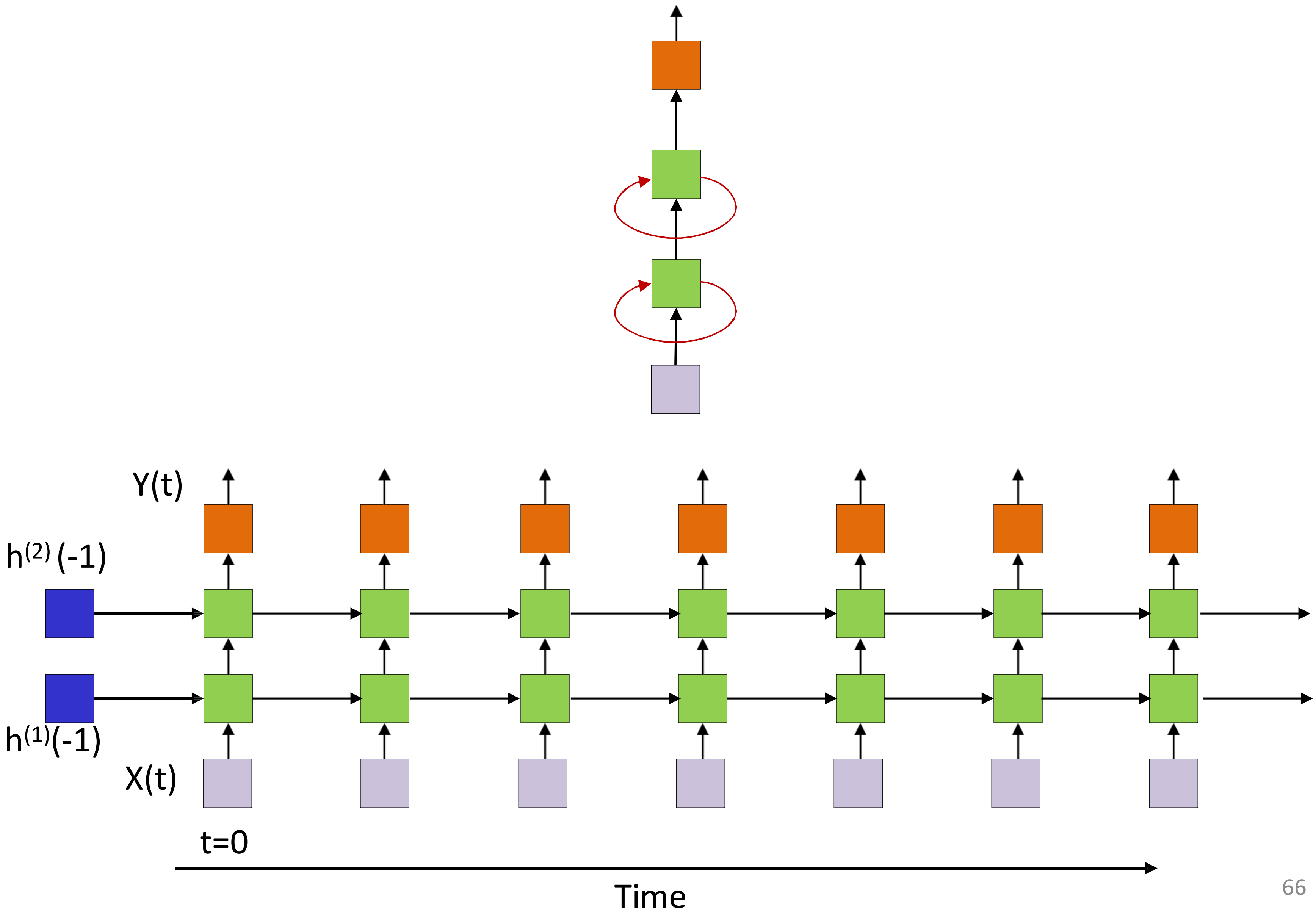


- Simplified models often drawn
- The loops imply recurrence

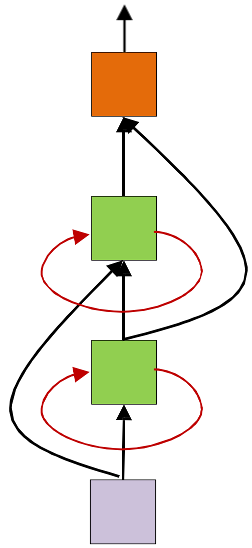
# The detailed version of the simplified representation

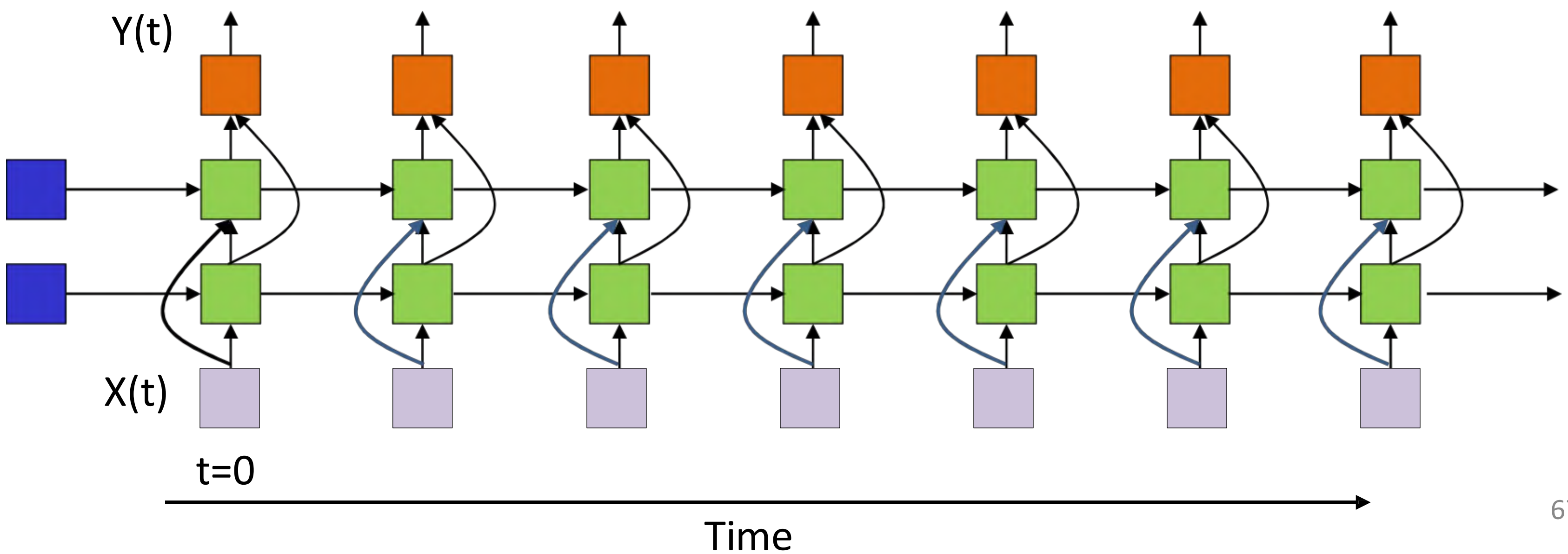


# Multiple recurrent layer RNN

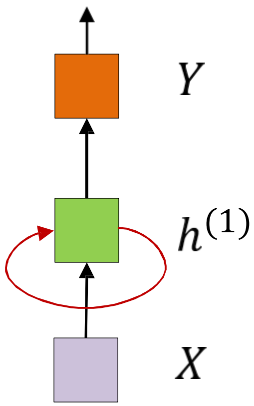


# Multiple recurrent layer RNN





# Equations




Current weights

Recurrent weights

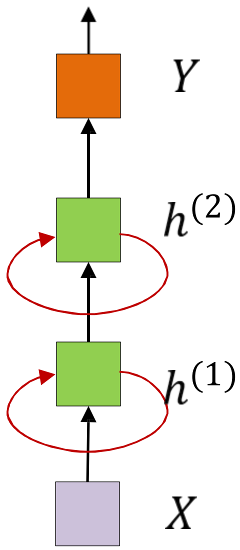
$$h_{i}^{(1)}(-1) = part\ of\ network\ parameters$$

$$h_{i}^{(1)}(t) = f_1\left(\sum_{j} w_{ji}^{(1)} X_{j}(t) + \sum_{j} w_{ji}^{(11)} h_{i}^{(1)}(t-1) + b_{i}^{(1)}\right)$$

$$Y(t) = f_2\left(\sum_{j} w_{jk}^{(2)} h_{j}^{(1)}(t) + b_{k}^{(2)}, k = 1..M\right)$$

- Note superscript in indexing, which indicates layer of network from which inputs are obtained
- Assuming vector function at output, e.g. softmax
- The *state* node activation, $f_1()$ is typically $\tanh()$
- Every neuron also has a *bias* input

# Equations




$$h_i^{(1)}(-1) = part\ of\ network\ parameters$$

$$h_i^{(2)}(-1) = part\ of\ network\ parameters$$

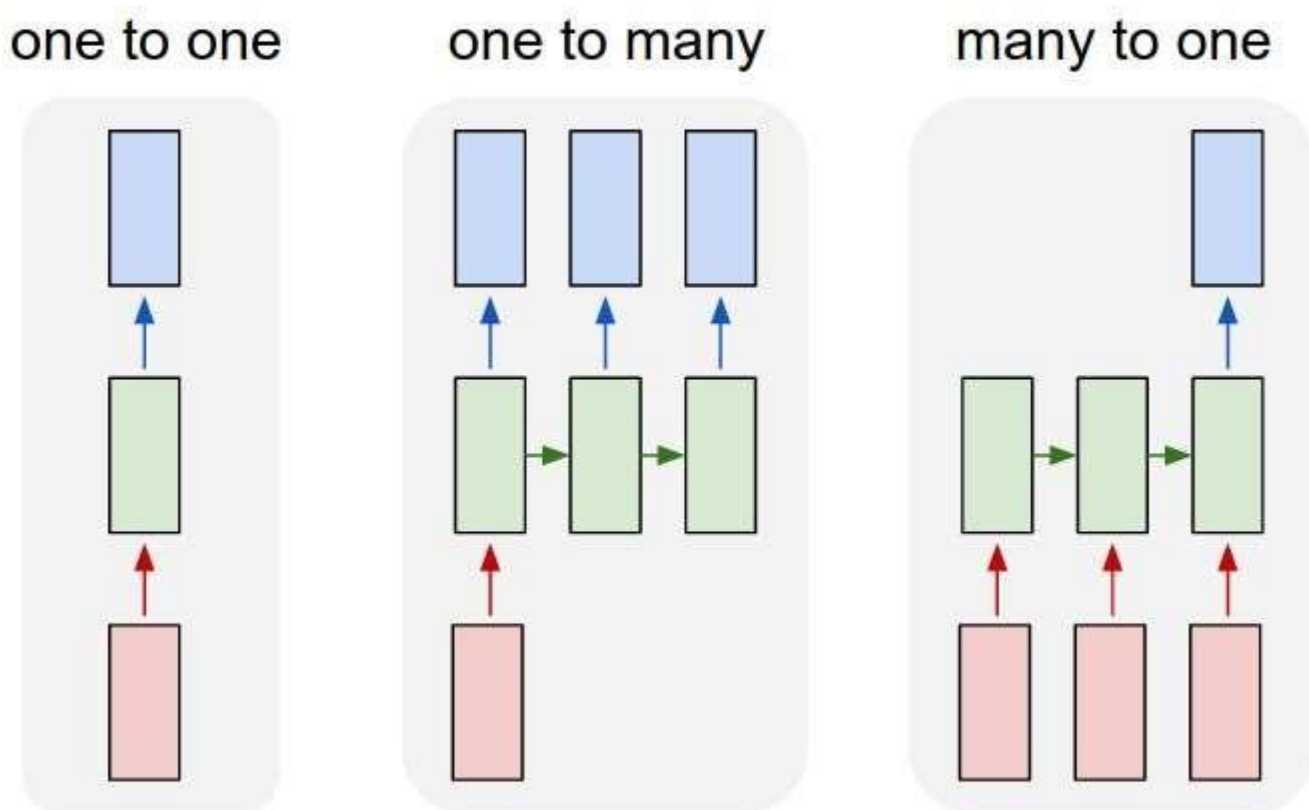
$$h_i^{(1)}(t) = f_1\left(\sum_j w_{ji}^{(1)} X_j(t) + \sum_j w_{ji}^{(11)} h_i^{(1)}(t-1) + b_i^{(1)}\right)$$

$$h_i^{(2)}(t) = f_2\left(\sum_j w_{ji}^{(2)} h_j^{(1)}(t) + \sum_j w_{ji}^{(22)} h_i^{(2)}(t-1) + b_i^{(2)}\right)$$

$$Y(t) = f_3\left(\sum_j w_{jk}^{(3)} h_j^{(2)}(t) + b_k^{(3)}, k = 1..M\right)$$

- Assuming vector function at output, e.g. softmax $f_3()$
- The *state* node activations, $f_k()$ are typically $\tanh()$
- Every neuron also has a *bias* input

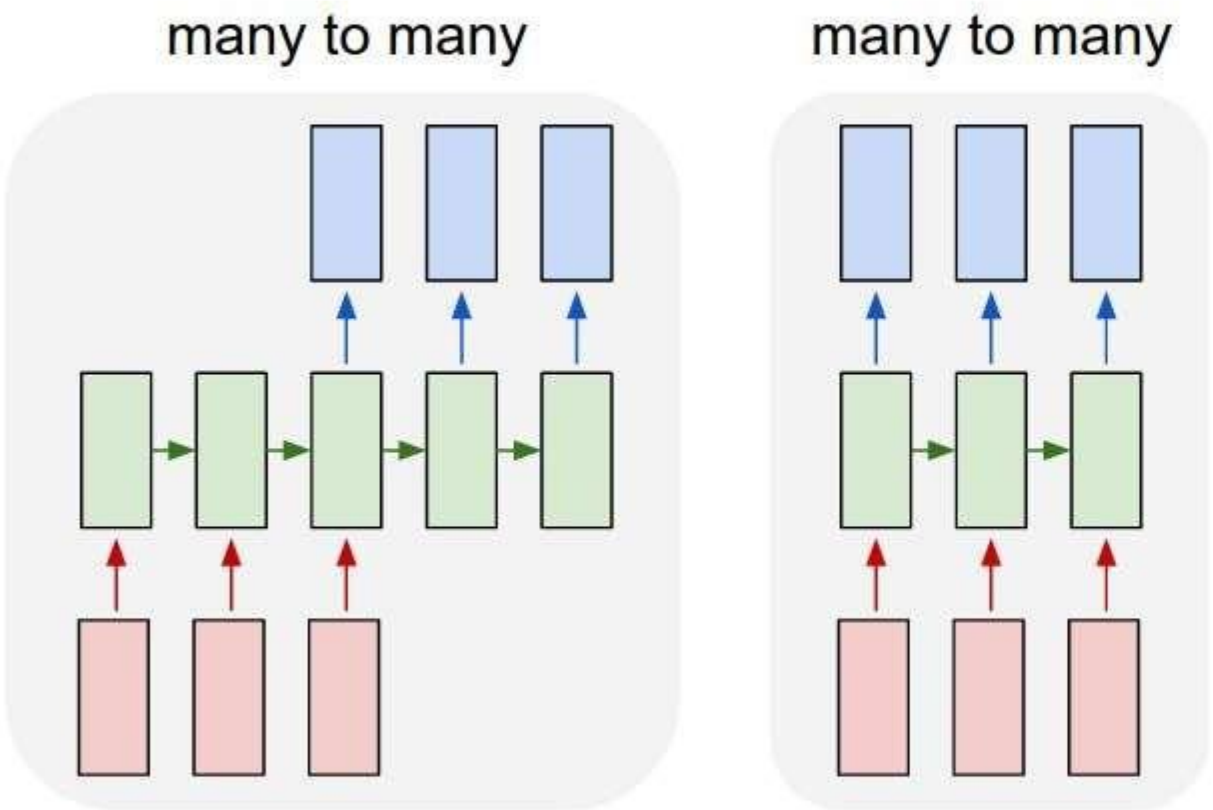
# Variants on recurrent nets




Images from Karpathy

- 1: Conventional MLP
- 2: Sequence *generation*, e.g. image to caption
- 3: Sequence based *prediction or classification,* e.g. Speech recognition, text classification

# Variants




Images from Karpathy

- 1: *Delayed* sequence to sequence, e.g. machine translation
- 2: Sequence to sequence, e.g. stock problem, label prediction
- Etc…

# Deep Learning
# Sequence to Sequence models: Attention Models

# Sequence-to-sequence modelling

- Problem:
  - A sequence $X_1 \ldots X_\mathrm{N}$ goes in
  - A different sequence $Y_1 \ldots Y_\mathrm{M}$ comes out
- E.g.
  - Speech recognition:  Speech goes in, a word sequence comes out
    - Alternately output may be phoneme or character sequence
  - Machine translation: Word sequence goes in, word sequence comes out
  - Dialog :  User statement goes in,  system response comes out
  - Question answering :  Question comes in, answer goes out

- In general $N \neq M$
  - No synchrony between $X$ and $Y$.

# Sequence to sequence

 → Seq2seq → I ate an apple

I ate an apple → Seq2seq → Ich habe einen apfel gegessen

- Sequence goes in, sequence comes out
- No notion of "time synchrony" between input and output
  - May even not even maintain order of symbols
    - E.g. "I ate an apple" → "Ich habe einen apfel gegessen"
  - Or even seem related to the input
    - E.g. "My screen is blank" → "Please check if your computer is plugged in."

# Recap: Predicting text

- Simple problem: Given a series of symbols (characters or words) $w_1$ $w_2$… $w_n$, predict the next symbol (character or word) $w_{n+1}$

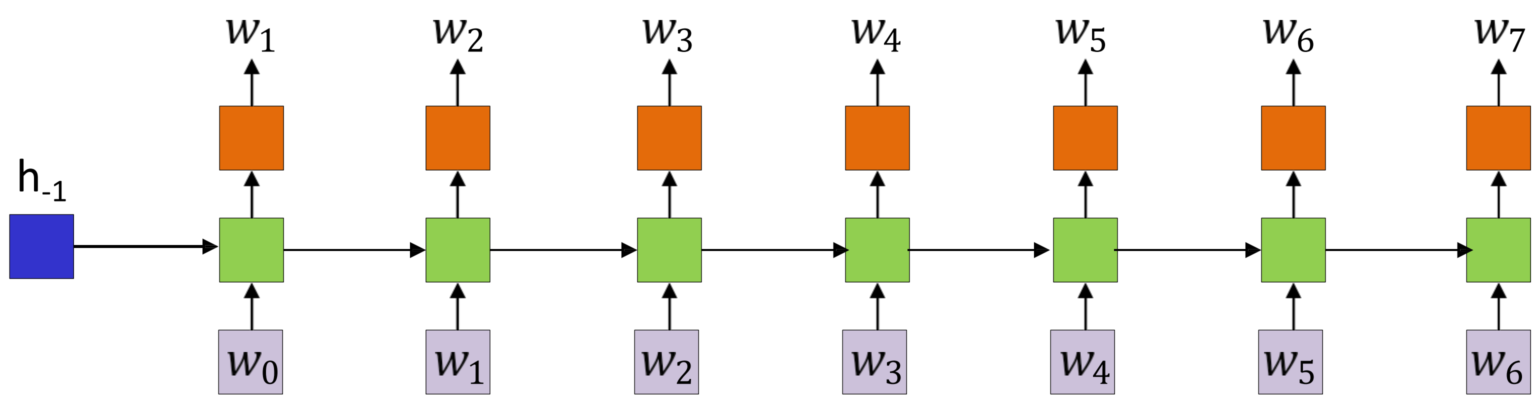
# Language modelling using RNNs

Four score and seven years ???

A B R A H A M L I N C O L ??

- Problem: Given a sequence of words (or characters) predict the next one
  - The problem of learning the sequential structure of language

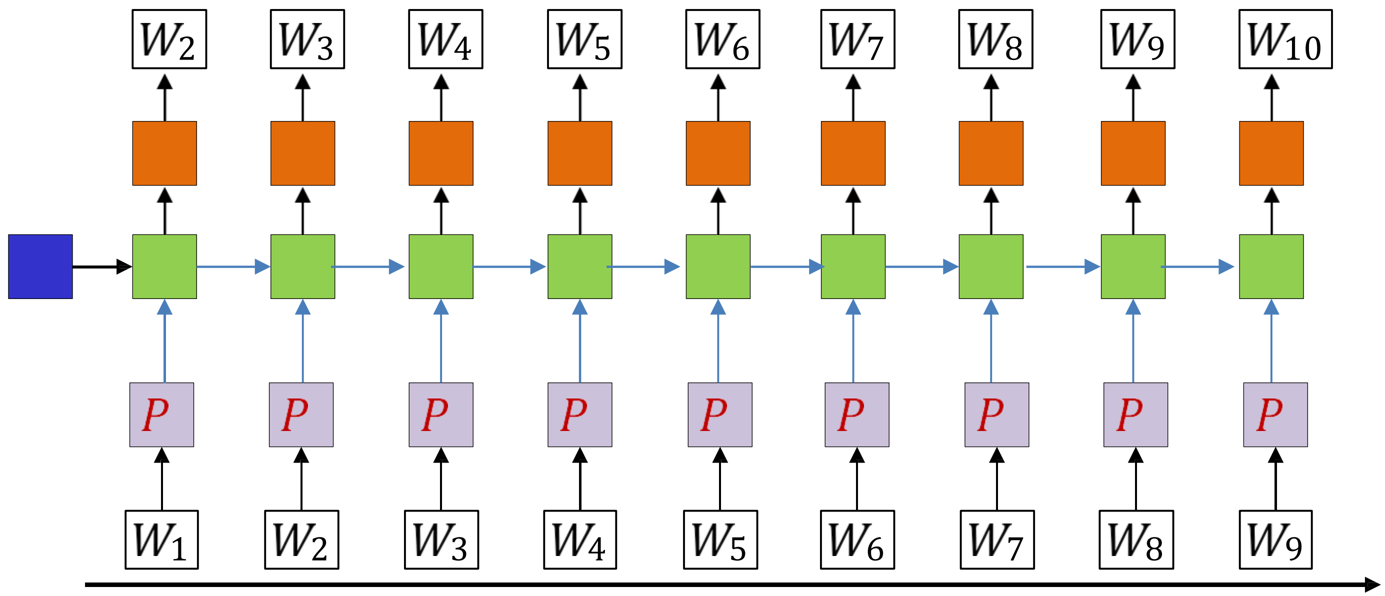
# Simple recurrence : Text Modelling




- Learn a model that can predict the next symbol given a sequence of symbols
  - Characters or words

- After observing inputs $w_0 \ldots w_k$ it predicts $w_{k+1}$
  - In reality, outputs a probability distribution for $w_{k+1}$

# Generating Language: The model

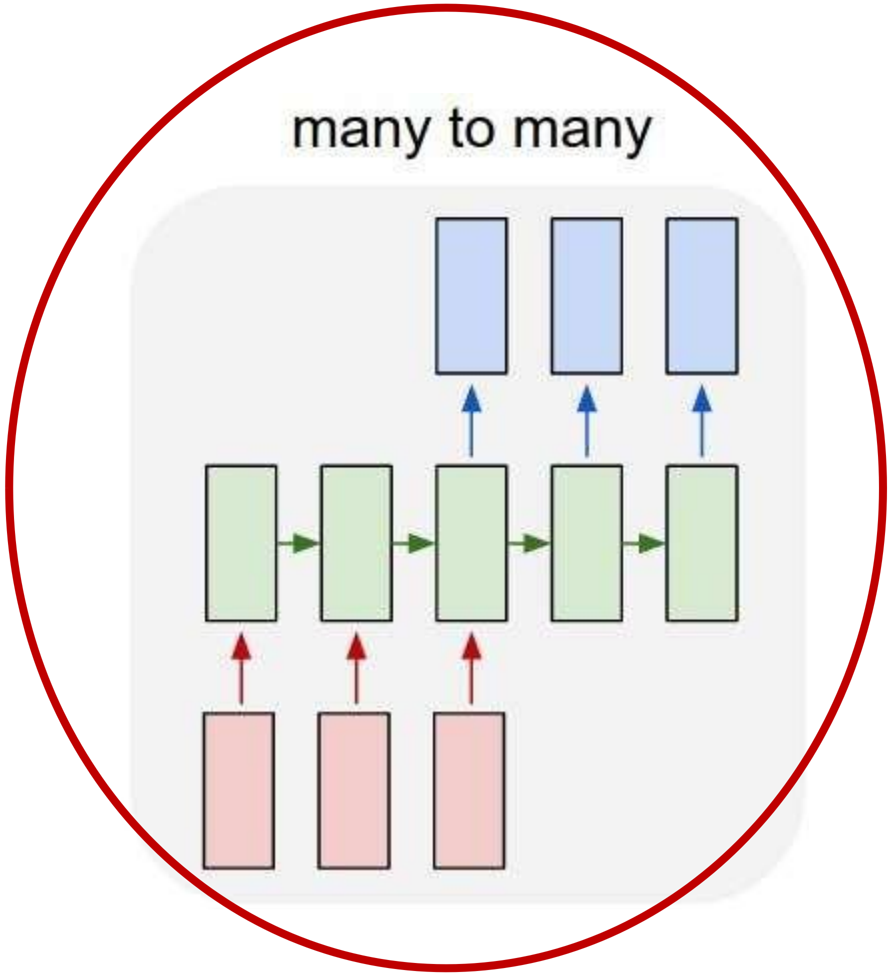



- Input: symbols as one-hot vectors
  - Dimensionality of the vector is the size of the "vocabulary"
  - Projected down to lower-dimensional "embeddings"

- The hidden units are (one or more layers of) LSTM units
- Output at each time: A probability distribution for the next word in the sequence

- All parameters are trained via backpropagation from a lot of text

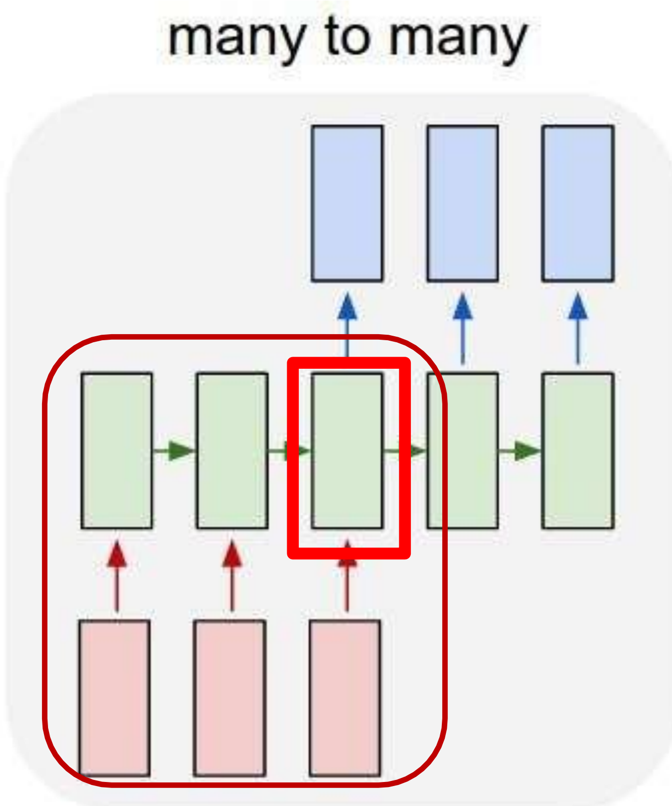
# Modelling the problem




- *Delayed* sequence to sequence

# Modelling the problem

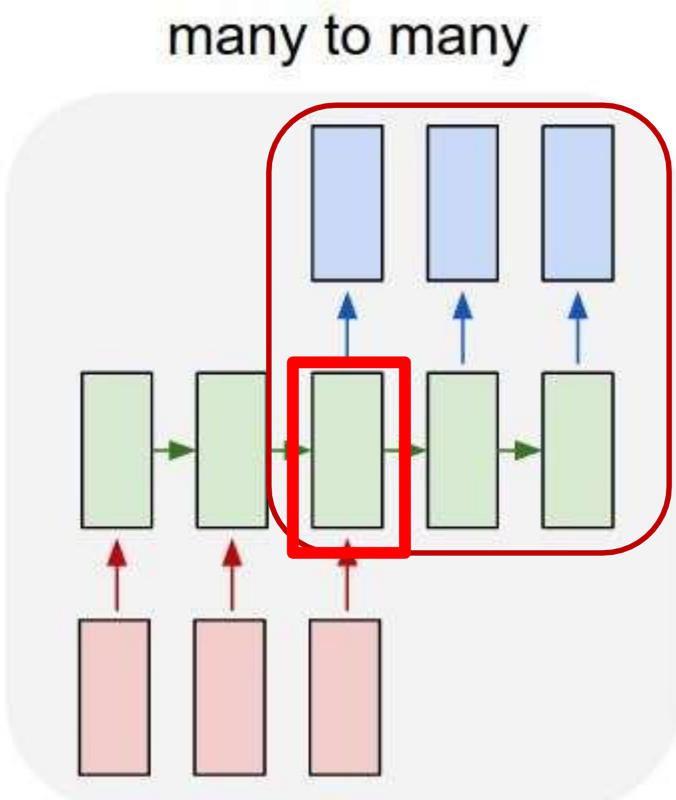
First process the input and generate a hidden representation for it




- *Delayed* sequence to sequence

# Modelling the problem

First process the input and generate a hidden representation for it

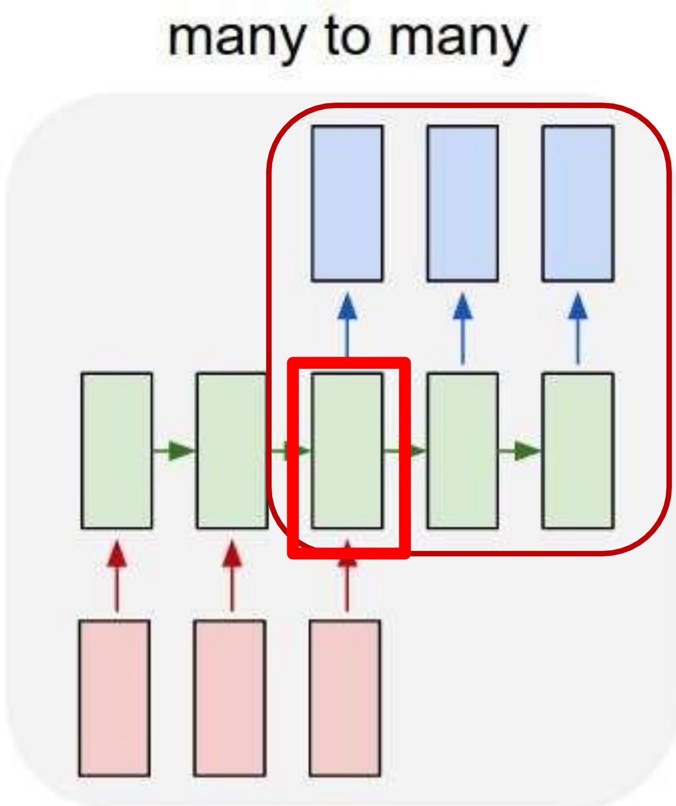



Then use it to generate an output

- *Delayed* sequence to sequence

# Modelling the problem

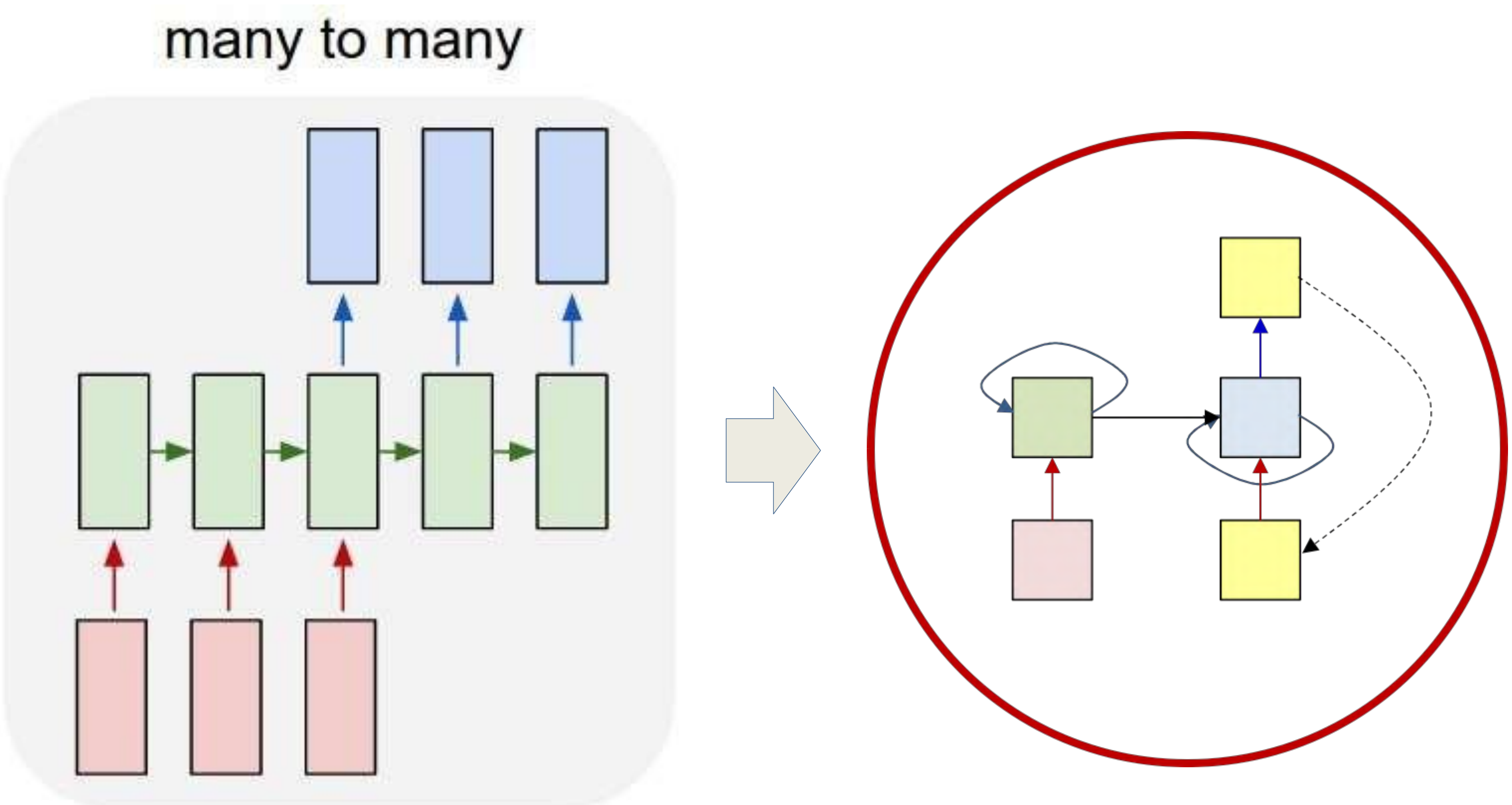
First process the input and generate a hidden representation for it




Then use it to generate an output

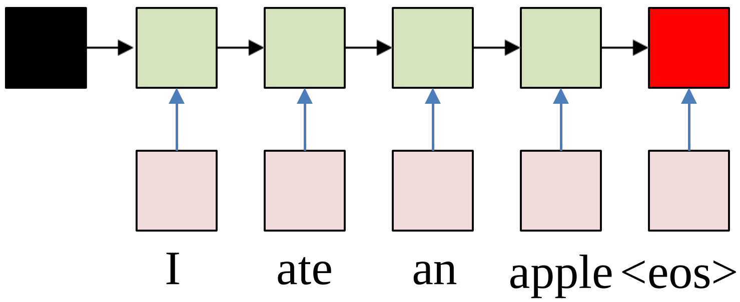
- *Problem:* Each word that is output depends only on current hidden state, and not on previous outputs

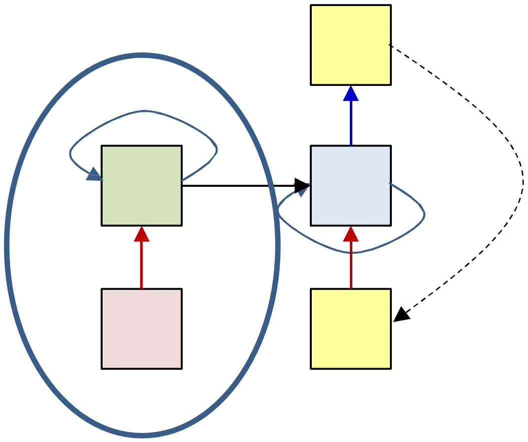
# Modelling the problem




- *Delayed* sequence to sequence
  - Delayed *self-referencing* sequence-to-sequence

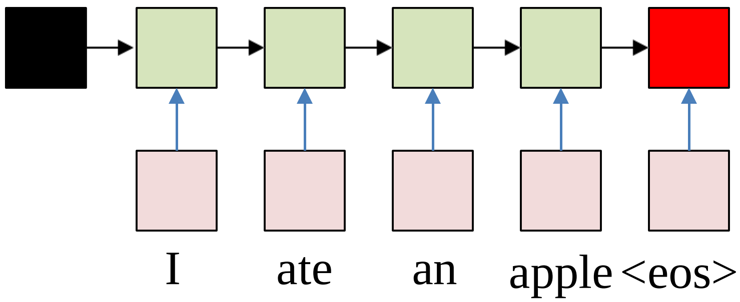
# The “simple” translation model

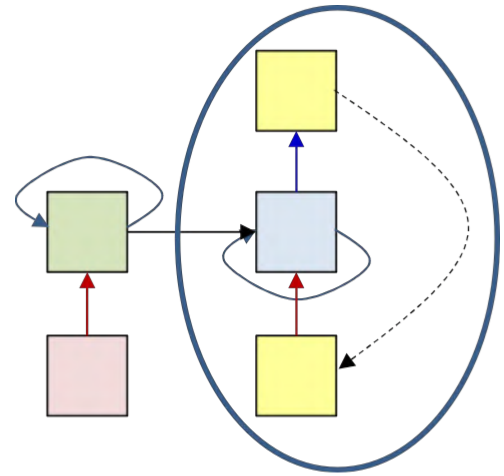


- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

# The “simple” translation model

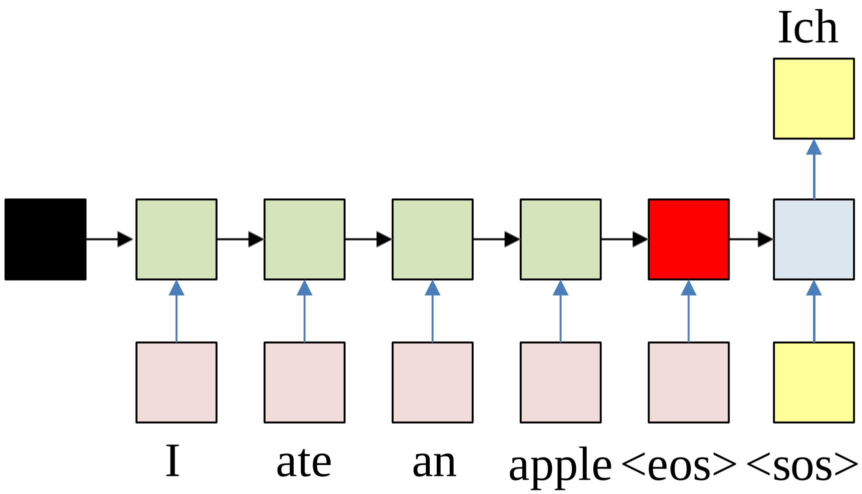


- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence

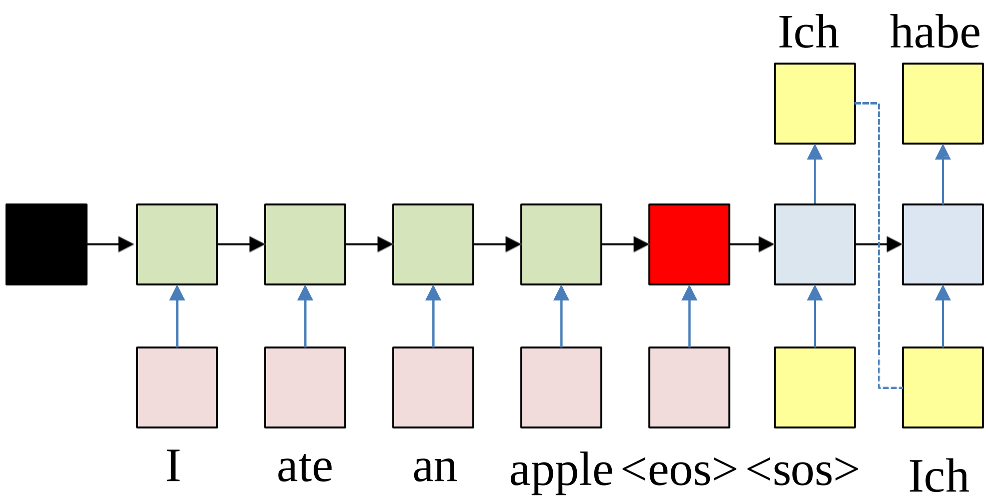
- Subsequently a *second* RNN uses the hidden activation as initial state, and <sos> as initial symbol, to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

# The "simple" translation model



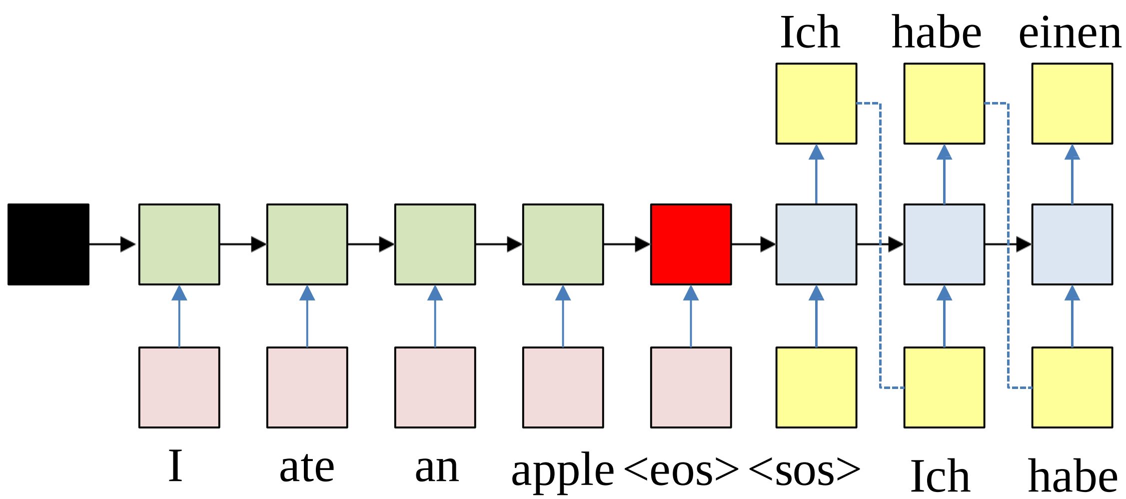

- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> "stores" all information about the sentence
- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

# The “simple” translation model



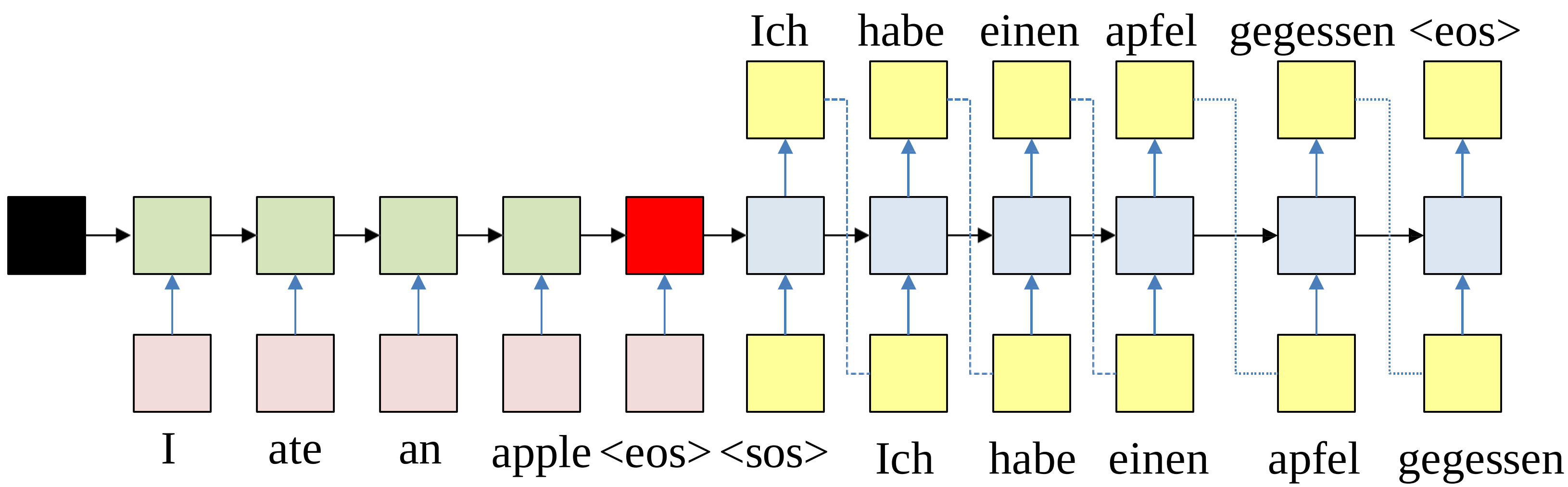

- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence
- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

# The “simple” translation model



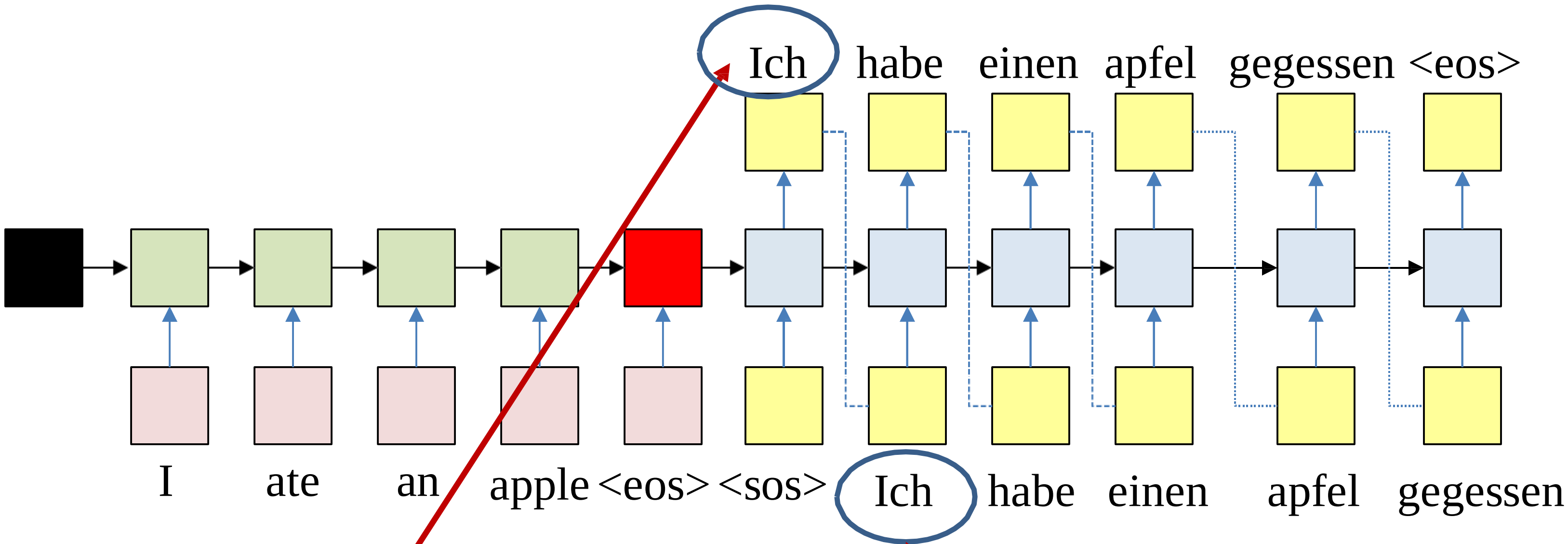

- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence
- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

# The “simple” translation model



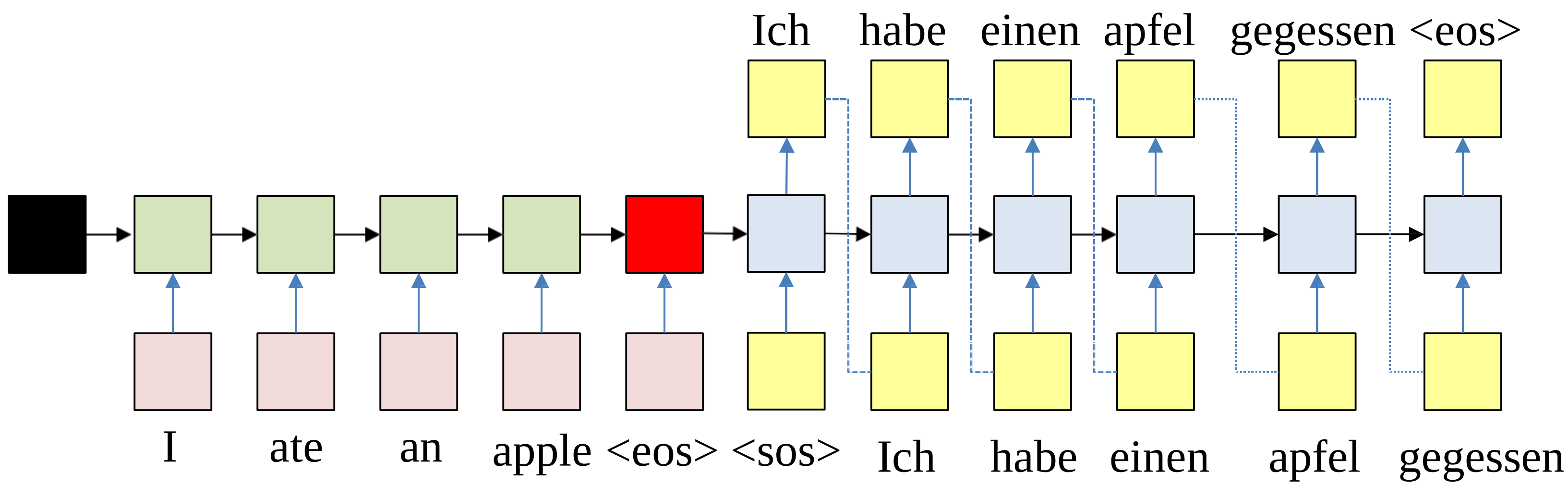

- The input sequence feeds into a recurrent structure
- The input sequence is terminated by an explicit <eos> symbol
  - The hidden activation at the <eos> “stores” all information about the sentence
- Subsequently a *second* RNN uses the hidden activation as initial state to produce a sequence of outputs
  - The output at each time becomes the input at the next time
  - Output production continues until an <eos> is produced

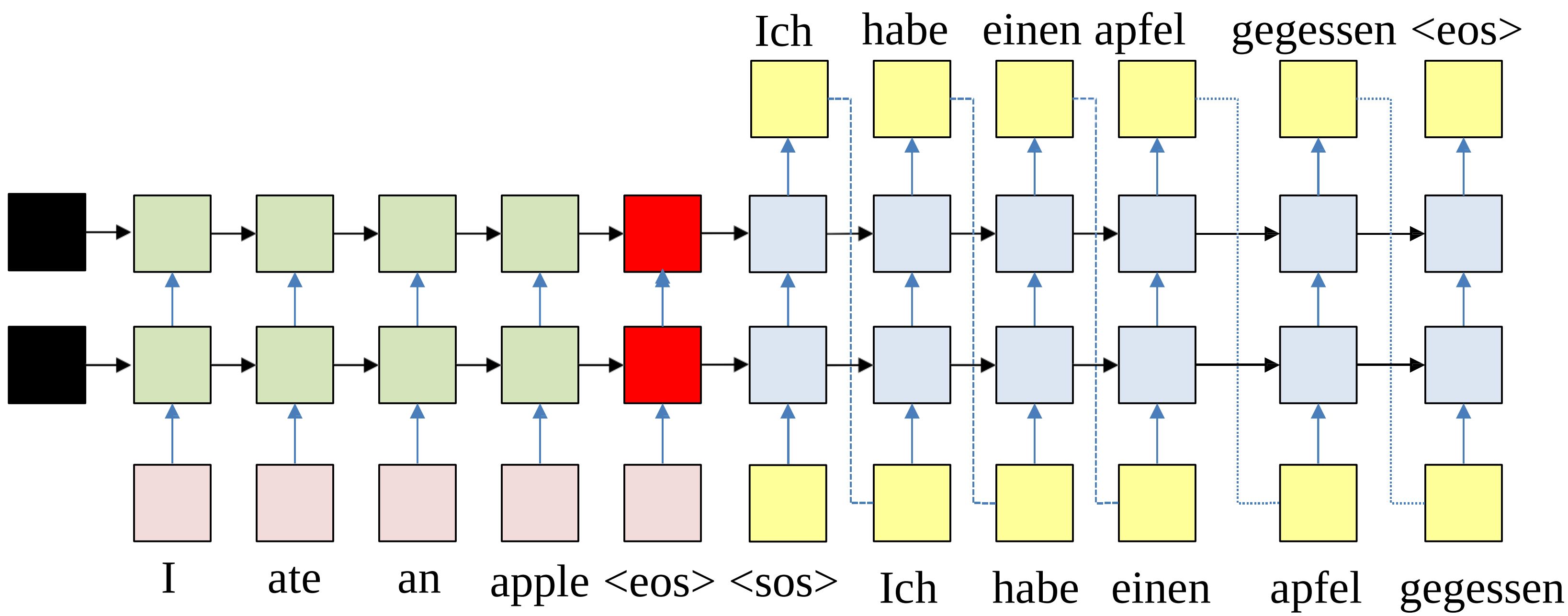
# The “simple” translation model




Note that drawing a different word here

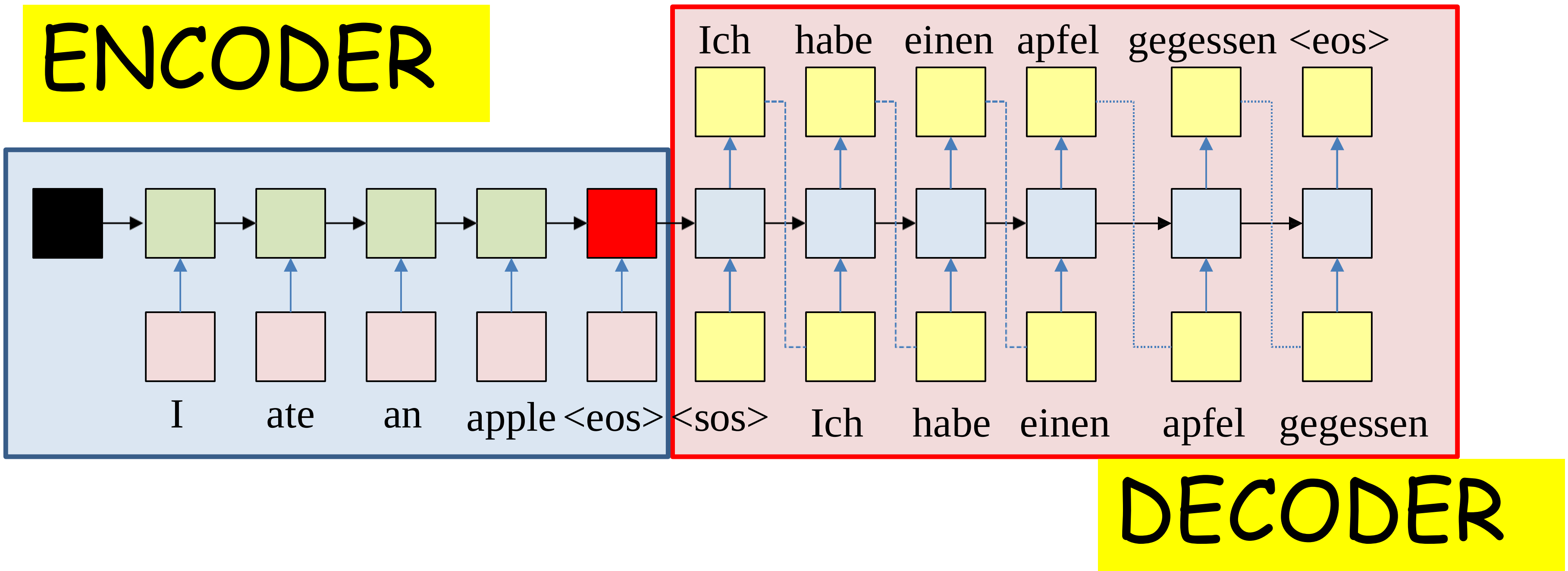
Would result in a different word being input here, and as a result the output here and subsequent outputs would all change



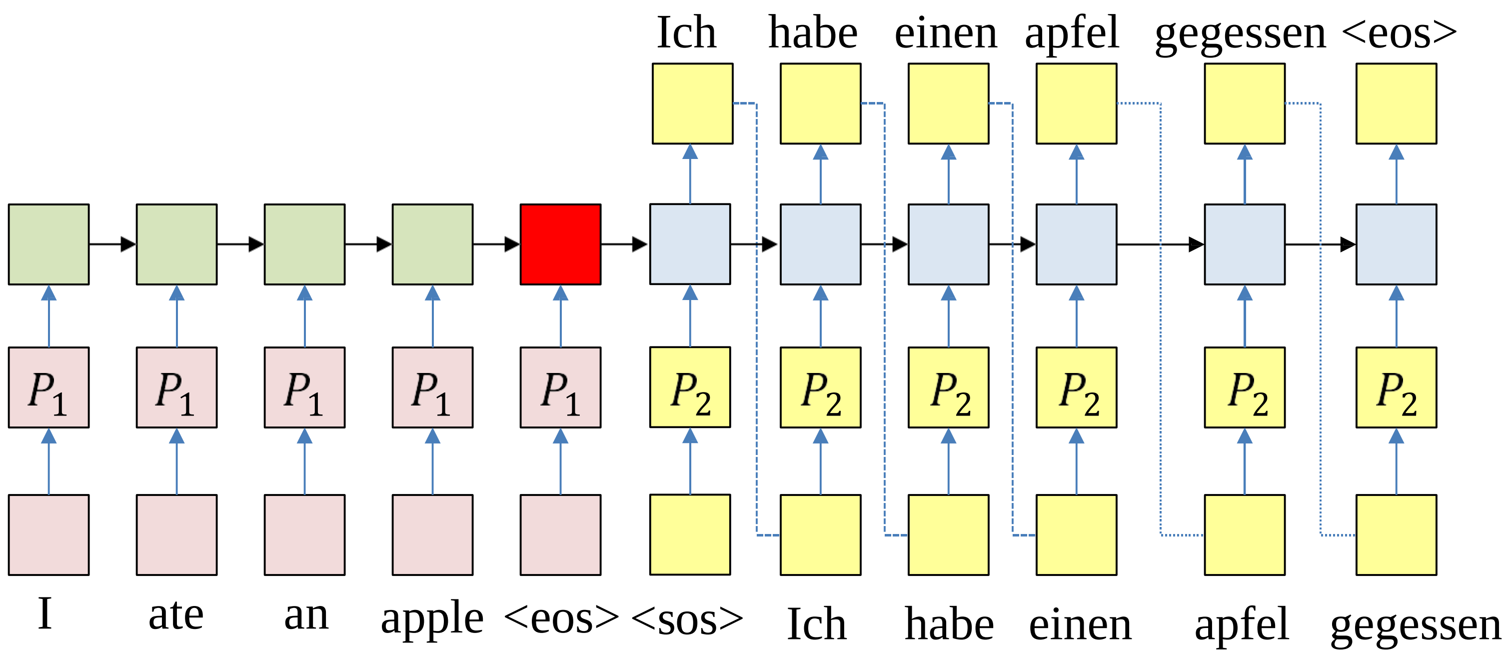

- We will illustrate with a single hidden layer, but the discussion generalizes to more layers

# The “simple” translation model



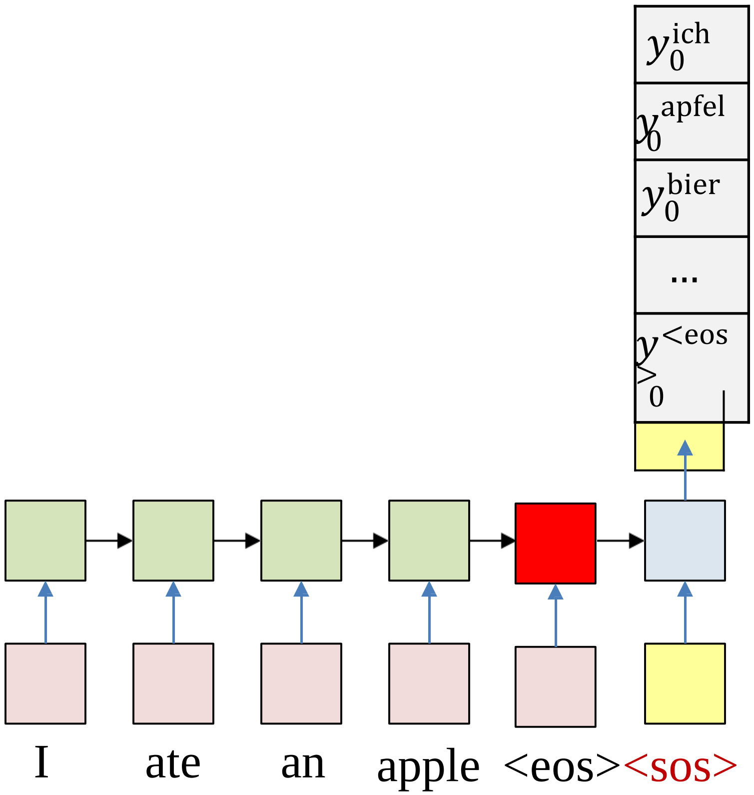

- The recurrent structure that extracts the hidden representation from the input sequence is the *encoder*
- The recurrent structure that utilizes this representation to produce the output sequence is the *decoder*

# The “simple” translation model



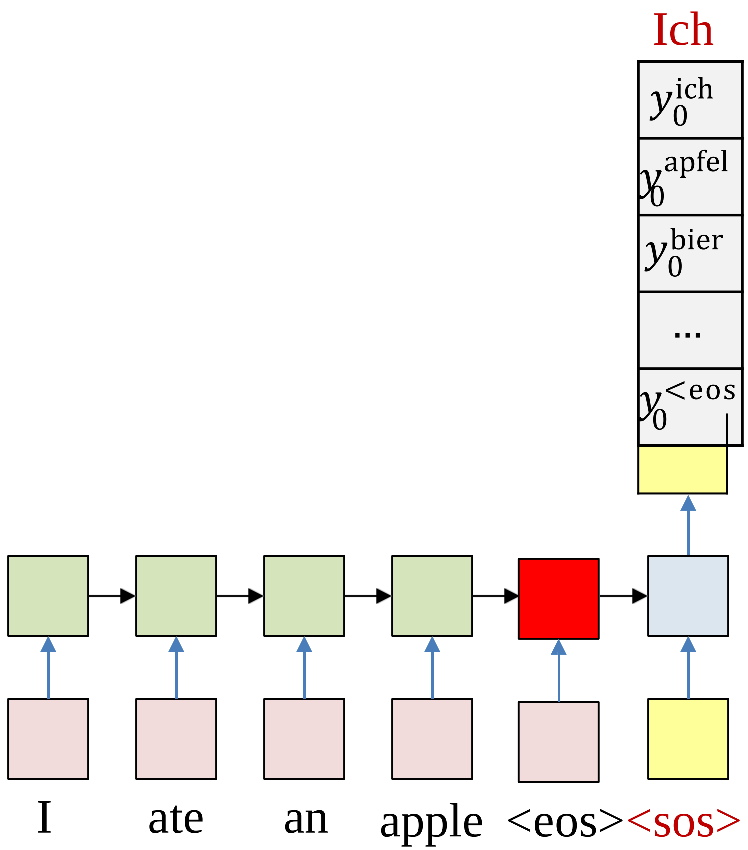

- A more detailed look: The one-hot word representations may be compressed via embeddings
  - Embeddings will be learned along with the rest of the net
  - In the following slides we will not represent the projection matrices

# What the network actually produces



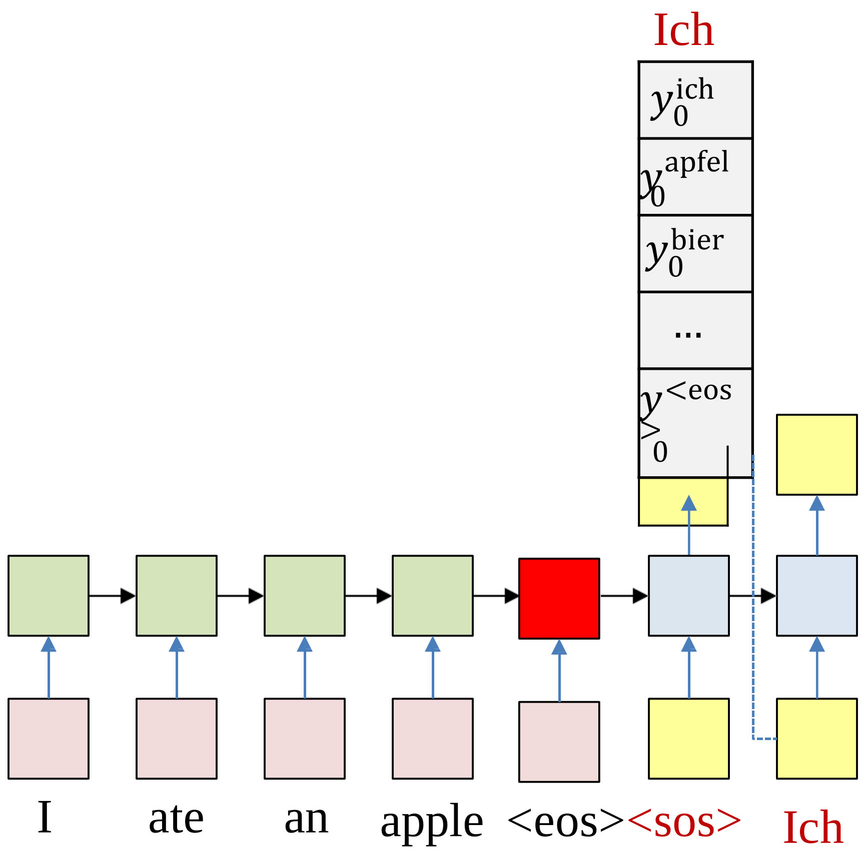

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until $k$
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

# What the network actually produces



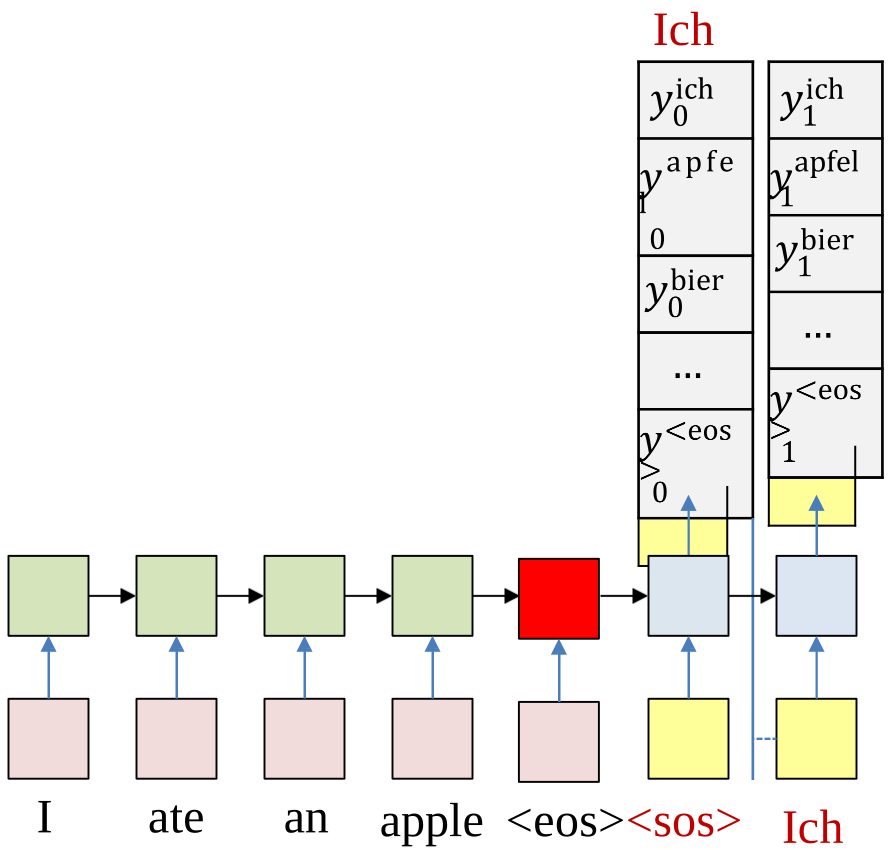

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until $k$
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

# What the network actually produces



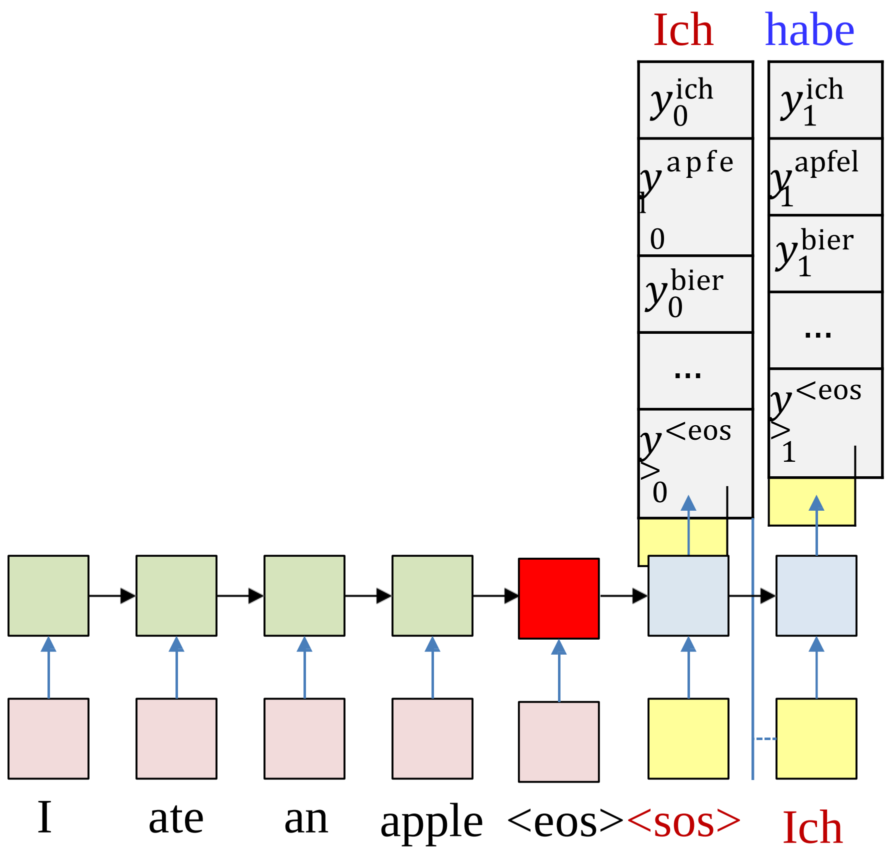

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until $k$
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

# What the network actually produces



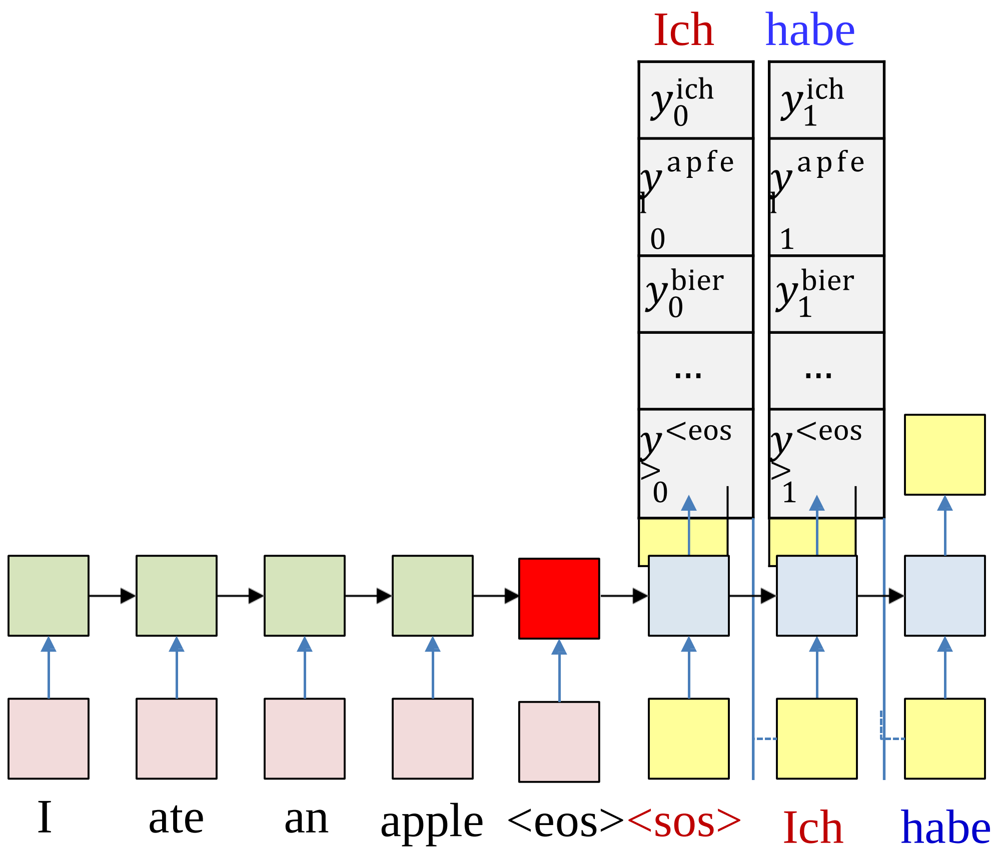

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until $k$
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

# What the network actually produces



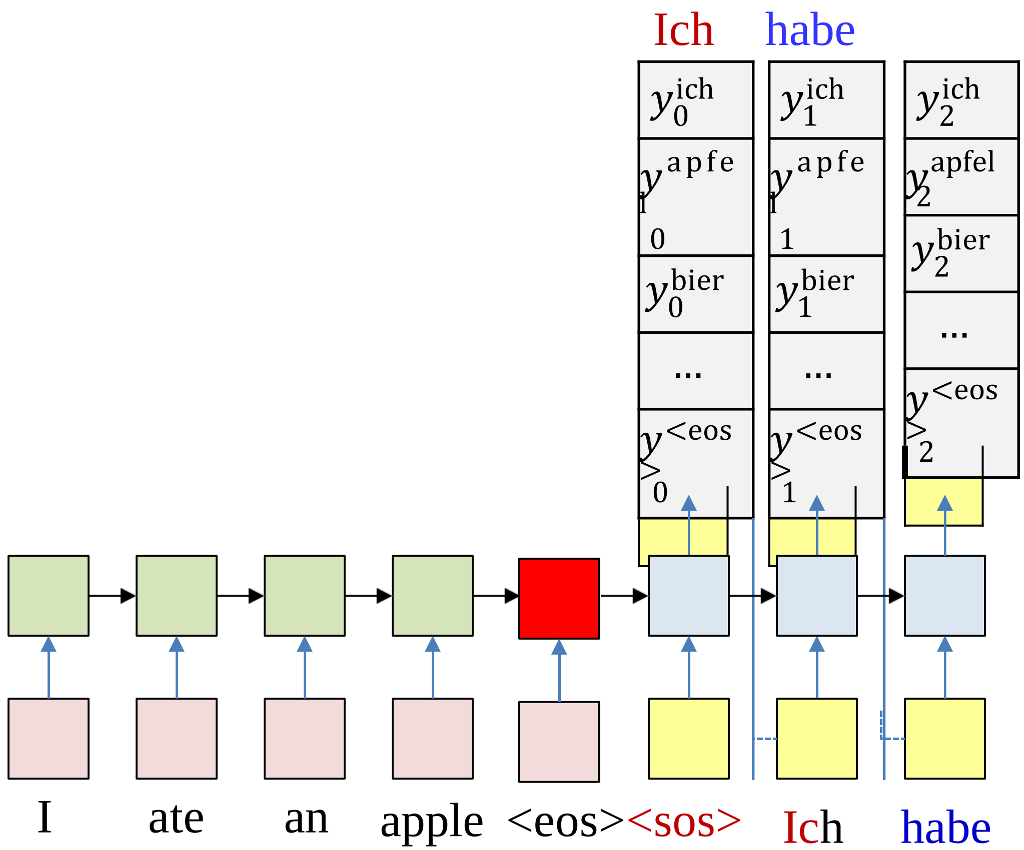

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until $k$
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

# What the network actually produces



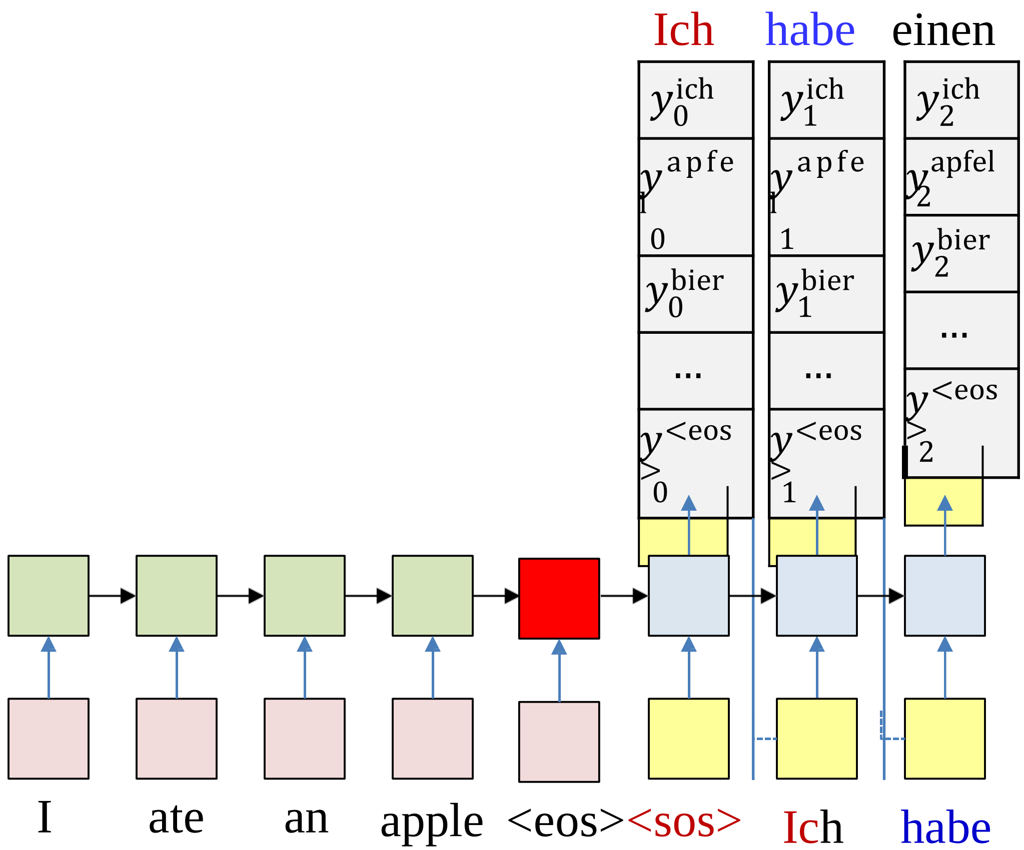

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until $k$
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

# What the network actually produces



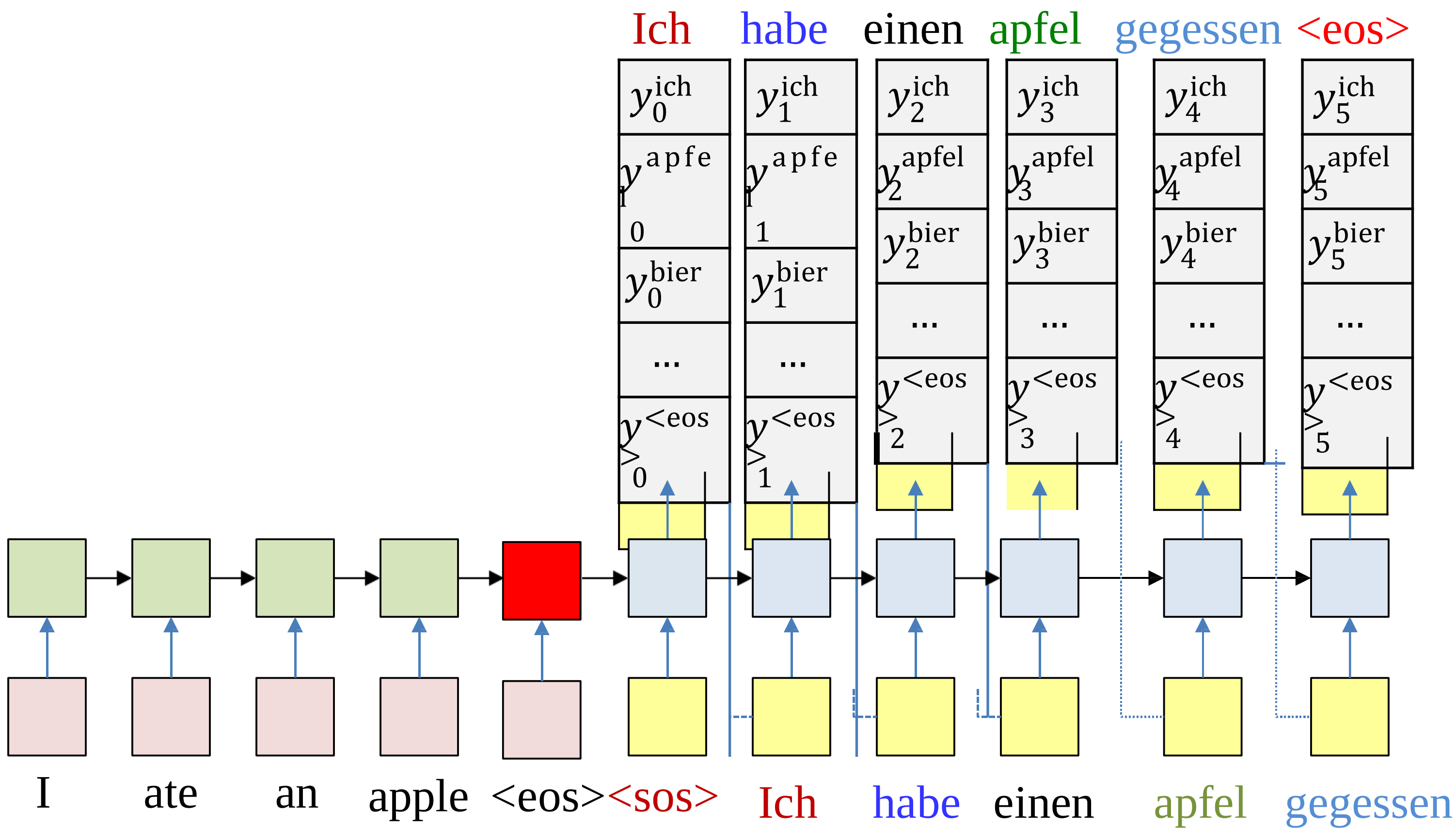

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \dots, O_1, I_1, \dots, I_N)$
  - The probability given the entire input sequence $I_1, \dots, I_N$ and the partial output sequence $O_1, \dots, O_{k-1}$ until $k$
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

# What the network actually produces



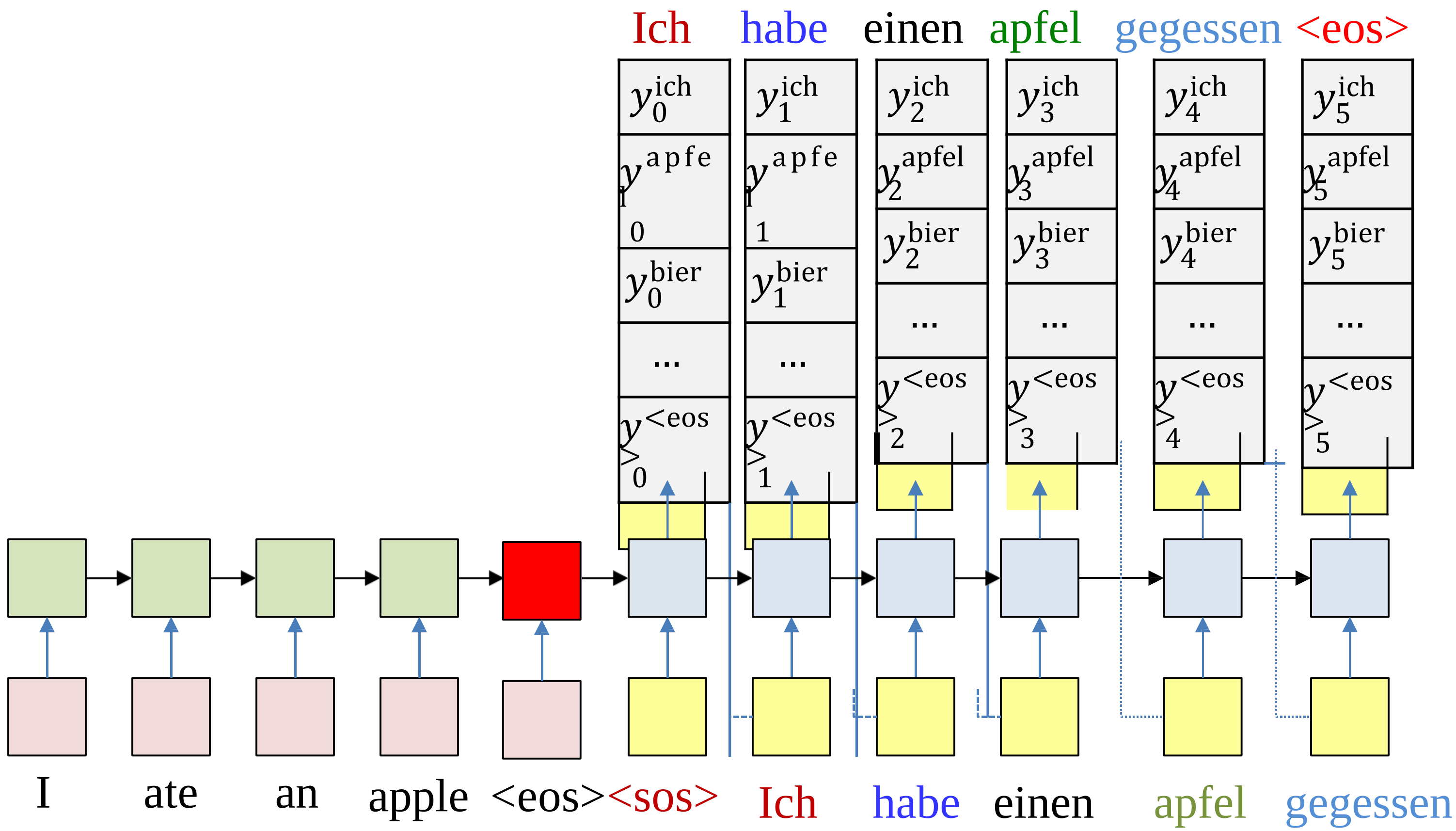

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until $k$
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

# What the network actually produces



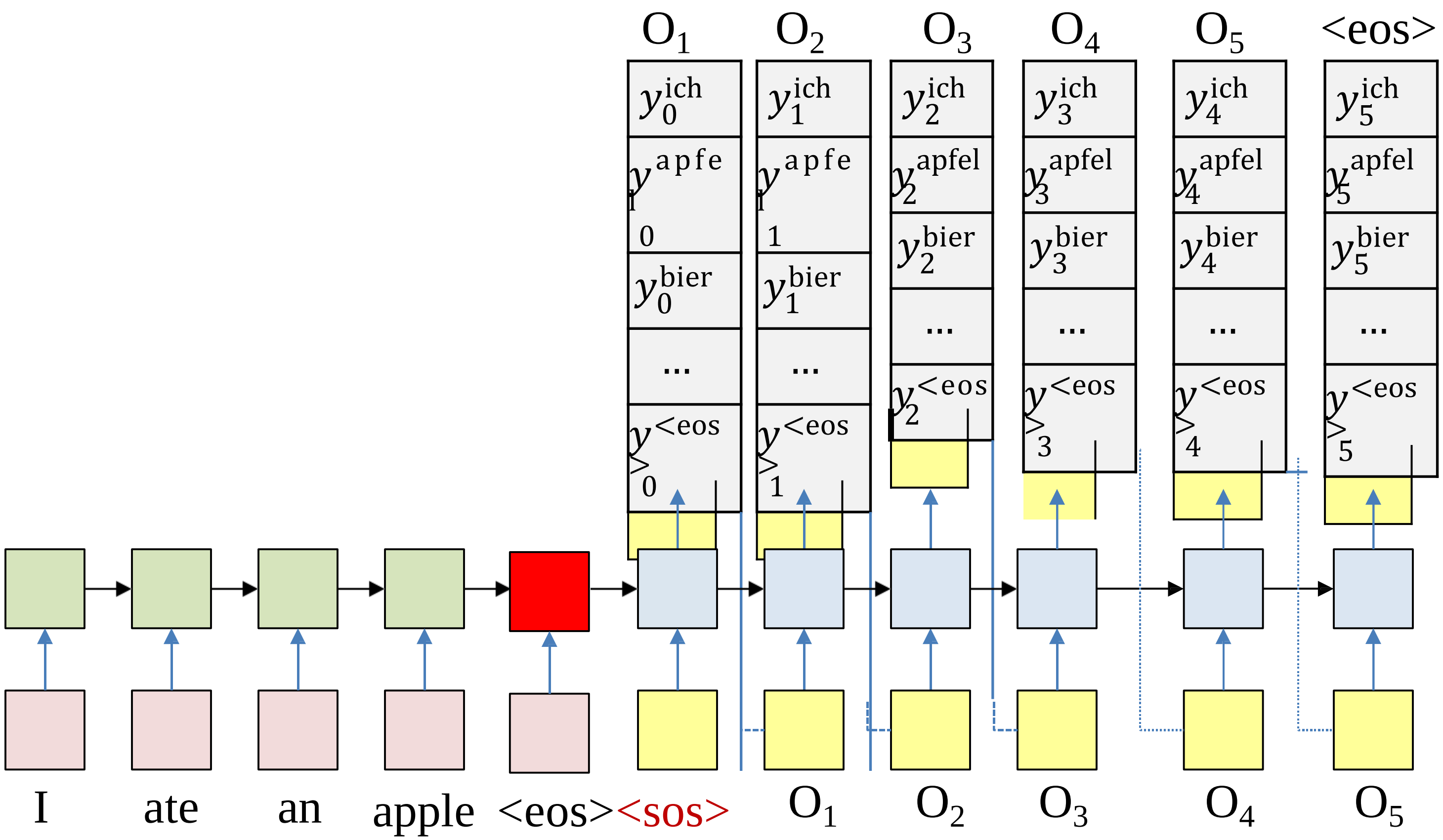

- At each time $k$ the network actually produces a probability distribution over the output vocabulary
  - $y_k^w = P(O_k = w | O_{k-1}, \ldots, O_1, I_1, \ldots, I_N)$
  - The probability given the entire input sequence $I_1, \ldots, I_N$ and the partial output sequence $O_1, \ldots, O_{k-1}$ until $k$
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time

# Generating an output from the net




- At each time the network produces a probability distribution over words, given the entire input and entire output sequence so far
- At each time a word is *drawn* from the output distribution
- The drawn word is provided as input to the next time
- The process continues until an <eos> is generated

# The probability of the output

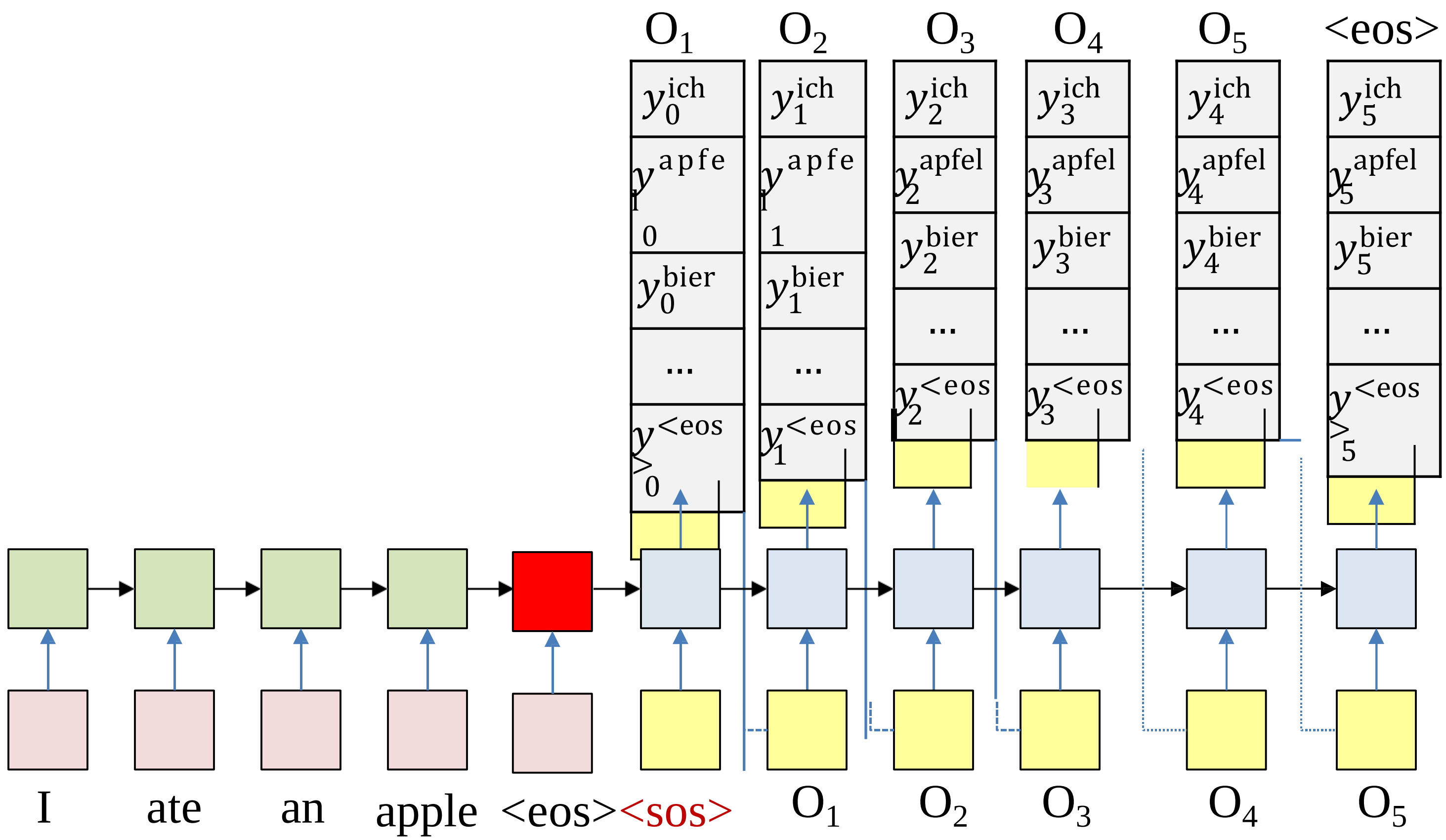



$$P(O_1, \dots, O_L | I_1, \dots, I_N)$$

$$= P(O_1 | I_{1\dots N}, \dots, I_N) P(O_2 | O_1, I_1, \dots, I_N) P(O_3 | O_1, O_2, I_1, \dots, I_N) \dots P(O_L | O_1, \dots, O_{L-1}, I_1, \dots, I_N)$$

$$= y_1^{O_1} y_2^{O_2} \dots y_L^{O_L}$$

# The probability of the output




- ***The objective of drawing: Produce the most likely output (that ends in an <eos>)***

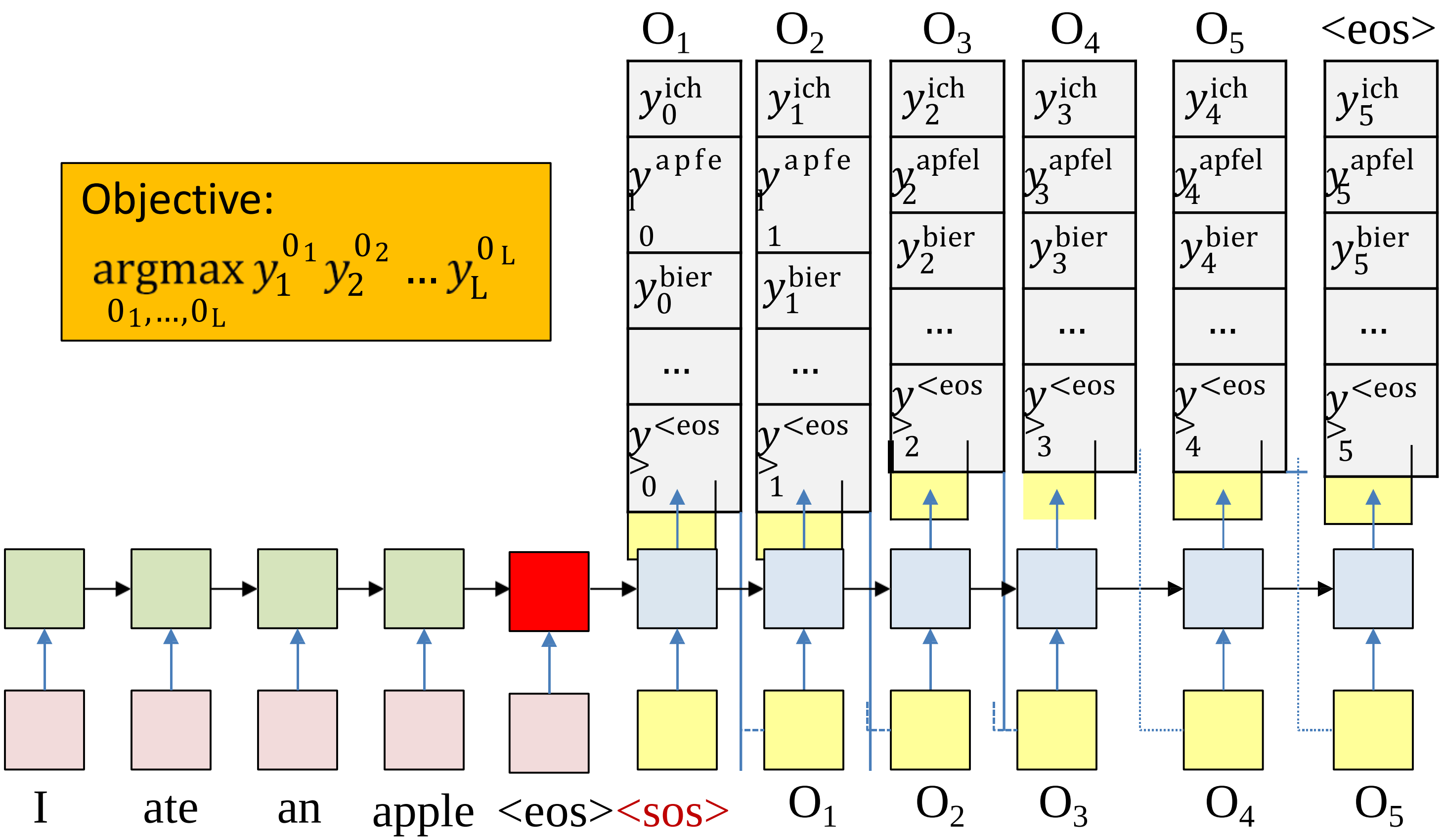
$$\underset{O_1,\ldots,O_L}{\operatorname{argmax}} P\left(O_1,\ldots,O_L \mid W_1^{\text{in}},\ldots,W_N^{\text{in}}\right)$$

$$= \underset{O_1,\ldots,O_L}{\operatorname{argmax}}\, y_1^{O_1} y_2^{O_2} \ldots y_L^{O_L}$$

# Greedy drawing

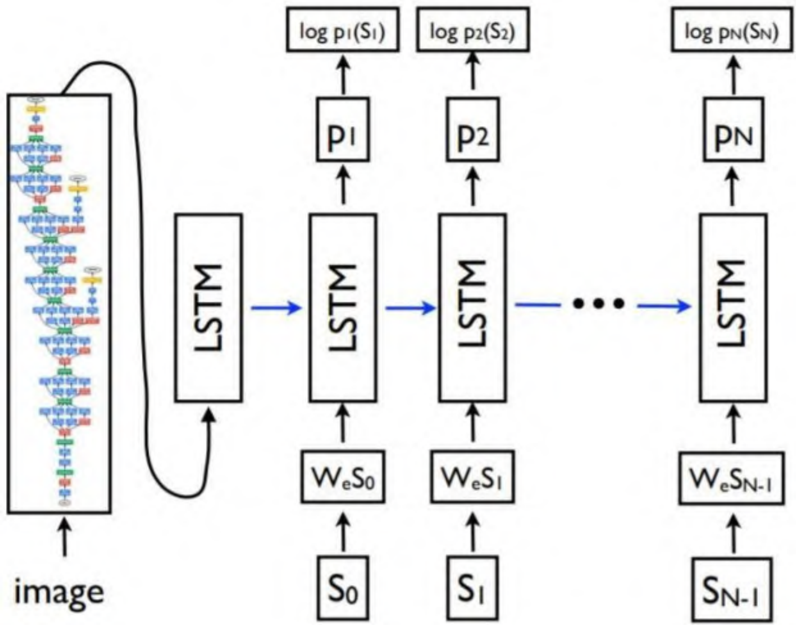
Objective:

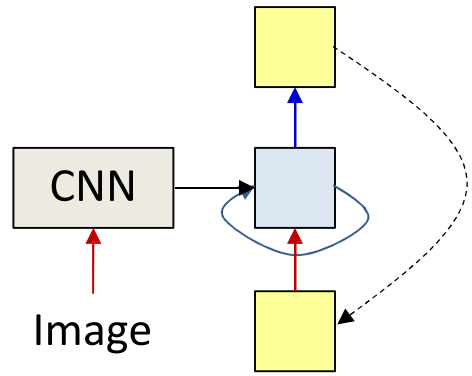
$$\underset{O_1,\dots,O_L}{\operatorname{argmax}}\; y_1^{O_1} y_2^{O_2} \dots y_L^{O_L}$$




- So how do we draw words at each time to get the most likely word sequence?
- *Greedy* answer – select the most probable word at each time

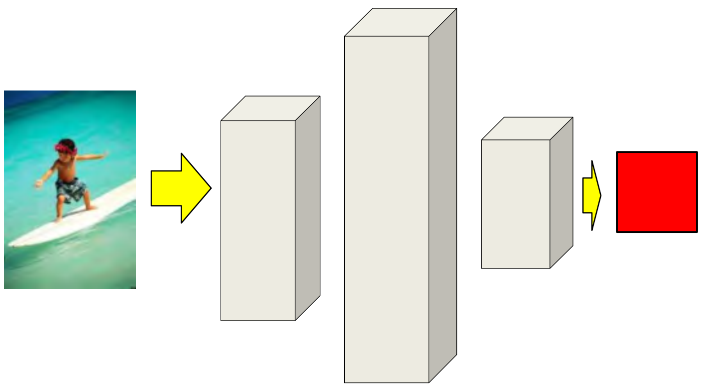
# Generating Image Captions



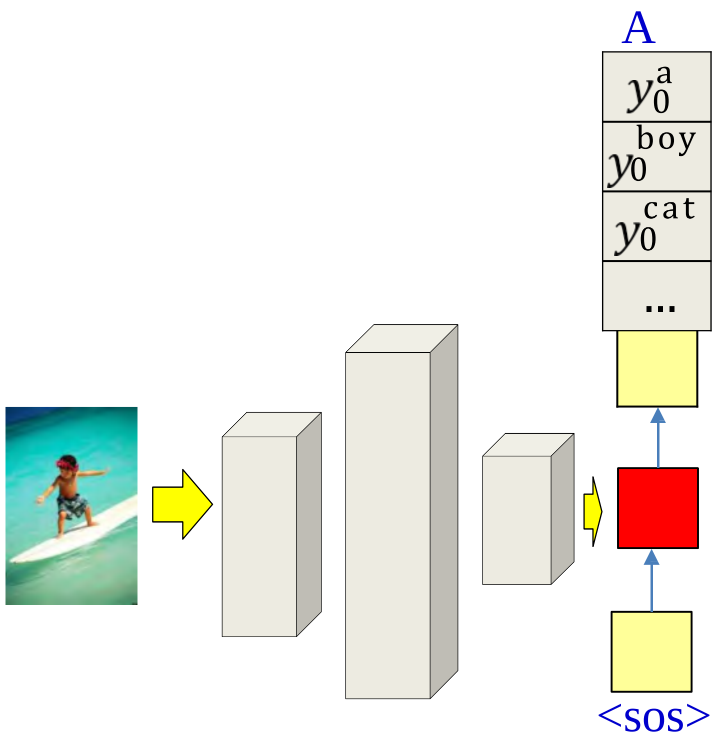

- Not really a seq-to-seq problem, more an image-to-sequence problem
- Initial state is produced by a state-of-art CNN-based image classification system
  - Subsequent model is just the decoder end of a seq-to-seq model
    - "Show and Tell: A Neural Image Caption Generator", O. Vinyals, A. Toshev, S. Bengio, D. Erhan

# Generating Image Captions



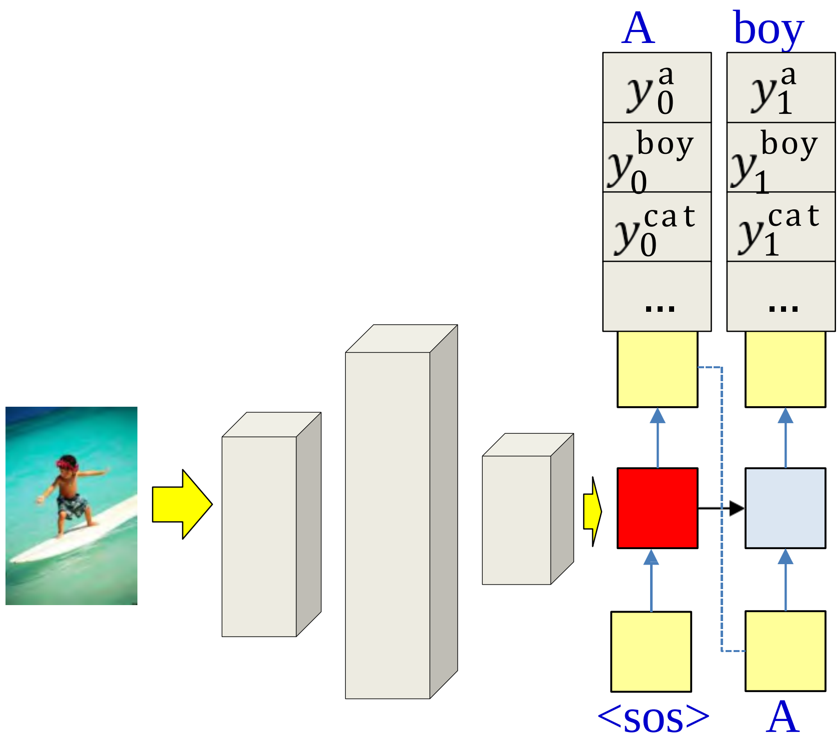
- Decoding: Given image
  - Process it with CNN to get output of classification layer

# Generating Image Captions



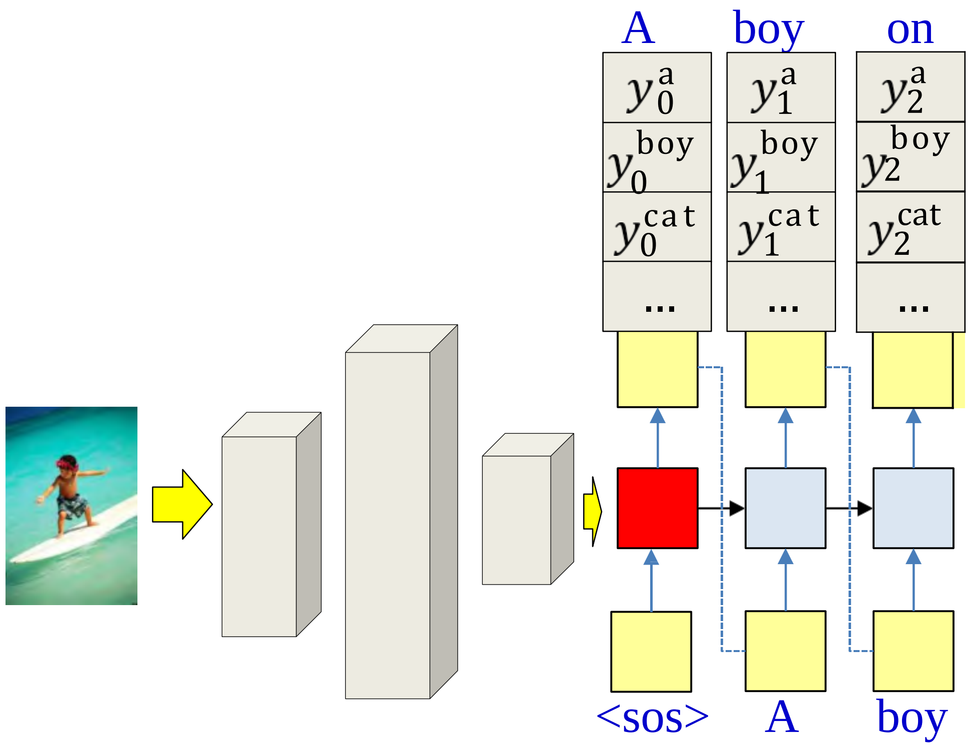

- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t | W_0 W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

# Generating Image Captions



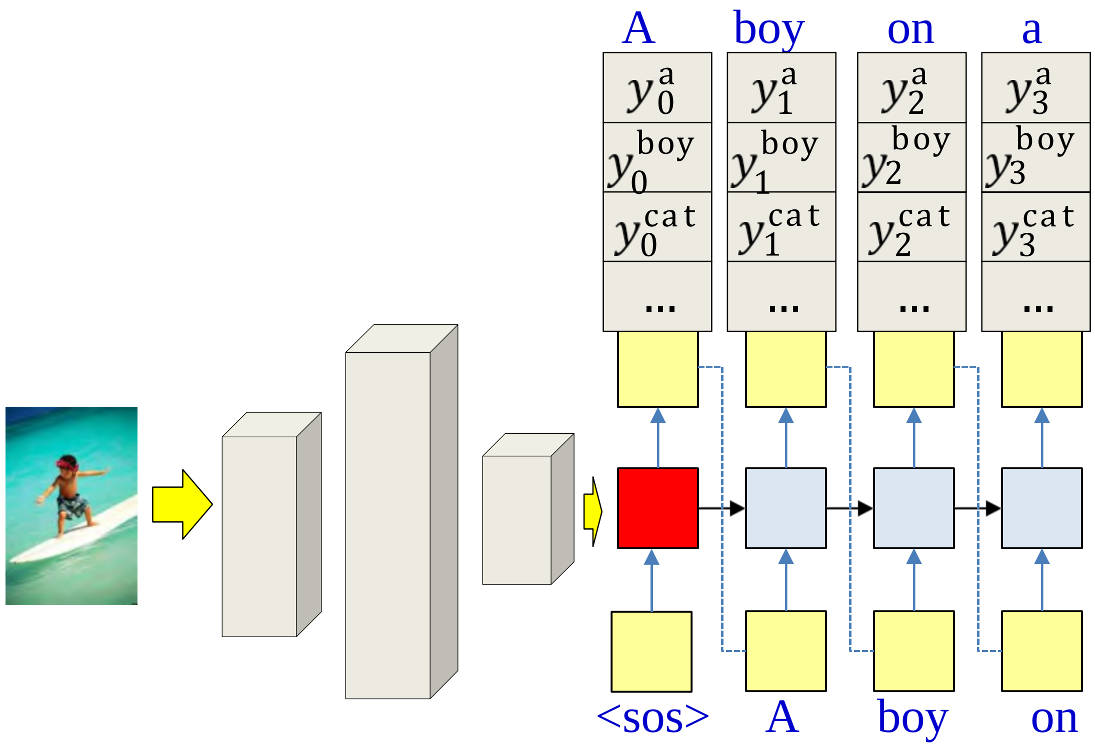

- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t | W_0 W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

# Generating Image Captions



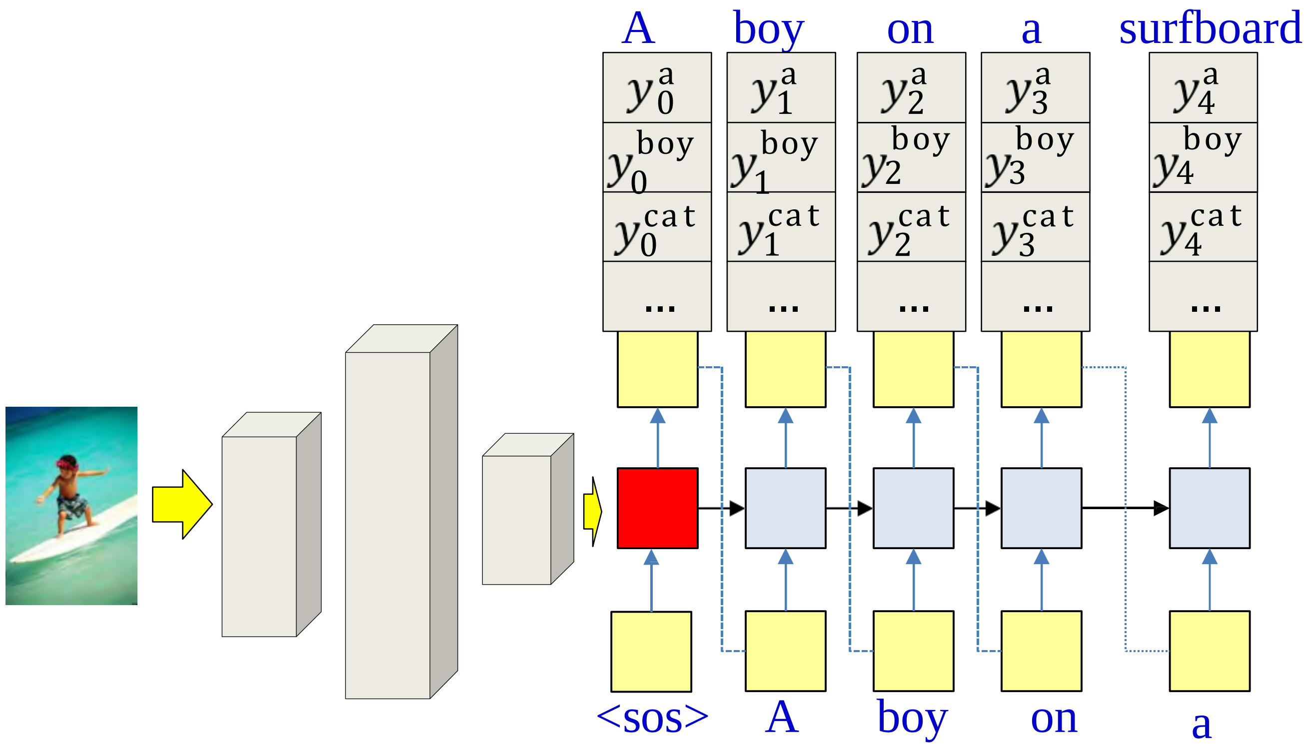

- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t | W_0 W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

# Generating Image Captions



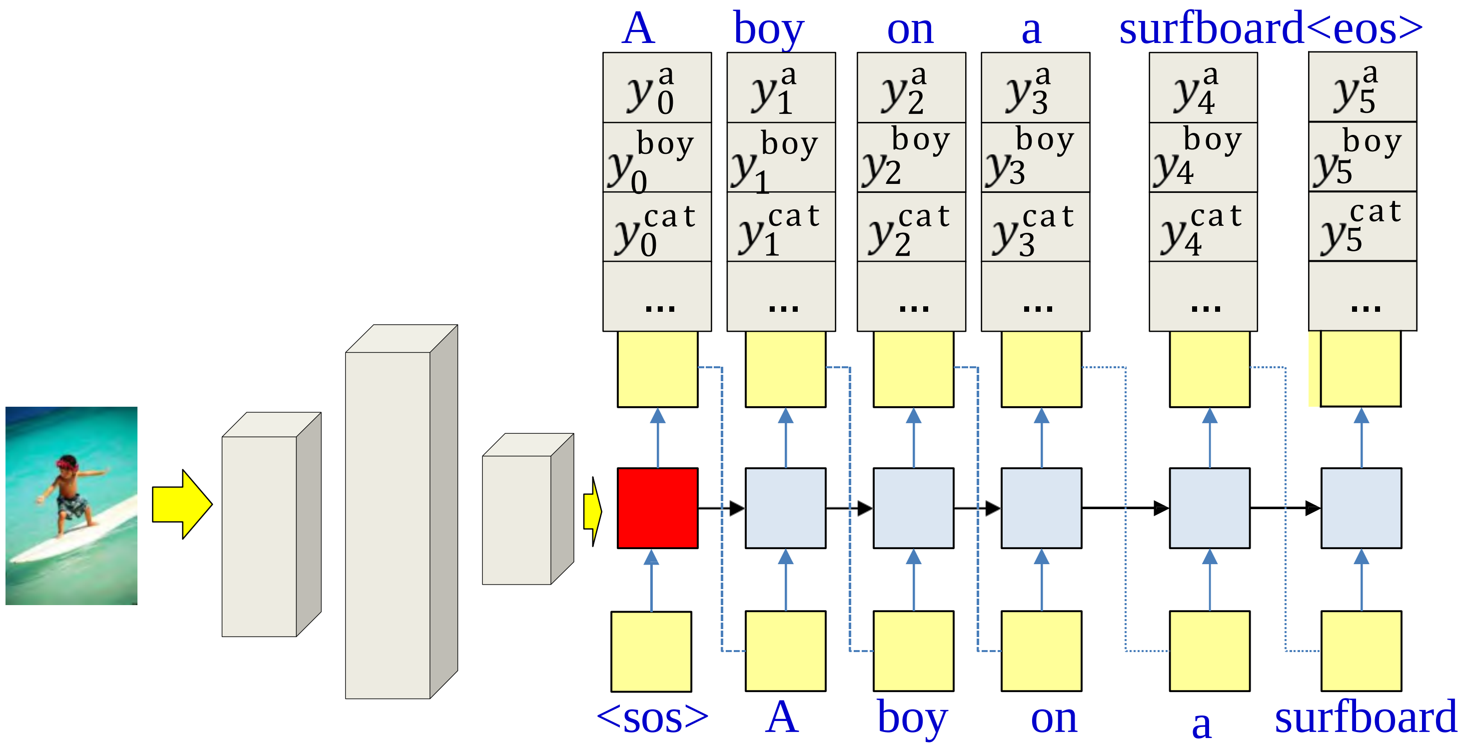

- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t | W_0 W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

# Generating Image Captions




- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t|W_0W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

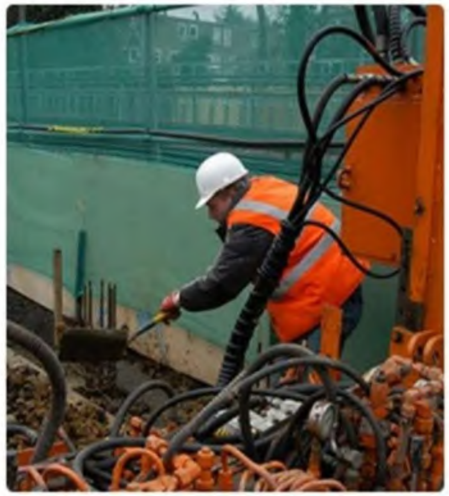
# Generating Image Captions




- Decoding: Given image
  - Process it with CNN to get output of classification layer
  - Sequentially generate words by drawing from the conditional output distribution $P(W_t | W_0 W_1 \ldots W_{t-1}, Image)$
  - In practice, we can perform the beam search explained earlier

# Examples from Vinyals et al.

"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

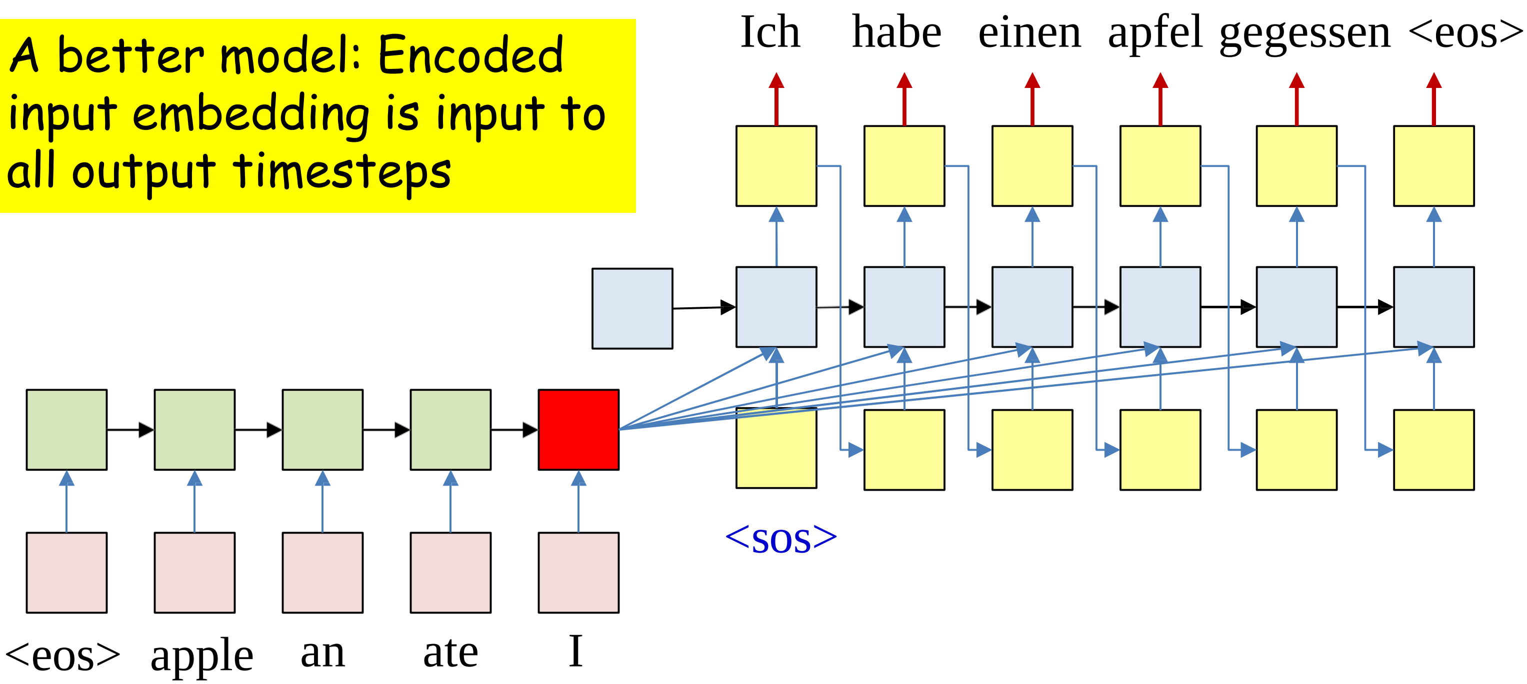
"a cat is sitting on a couch with a remote control."

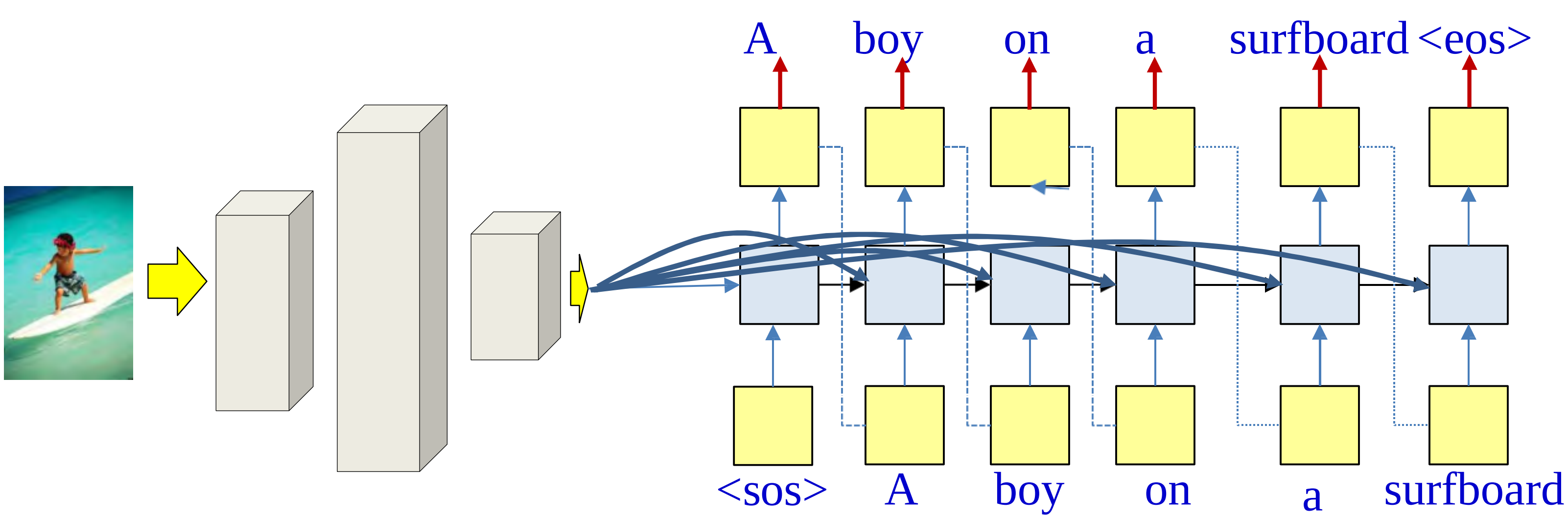

"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

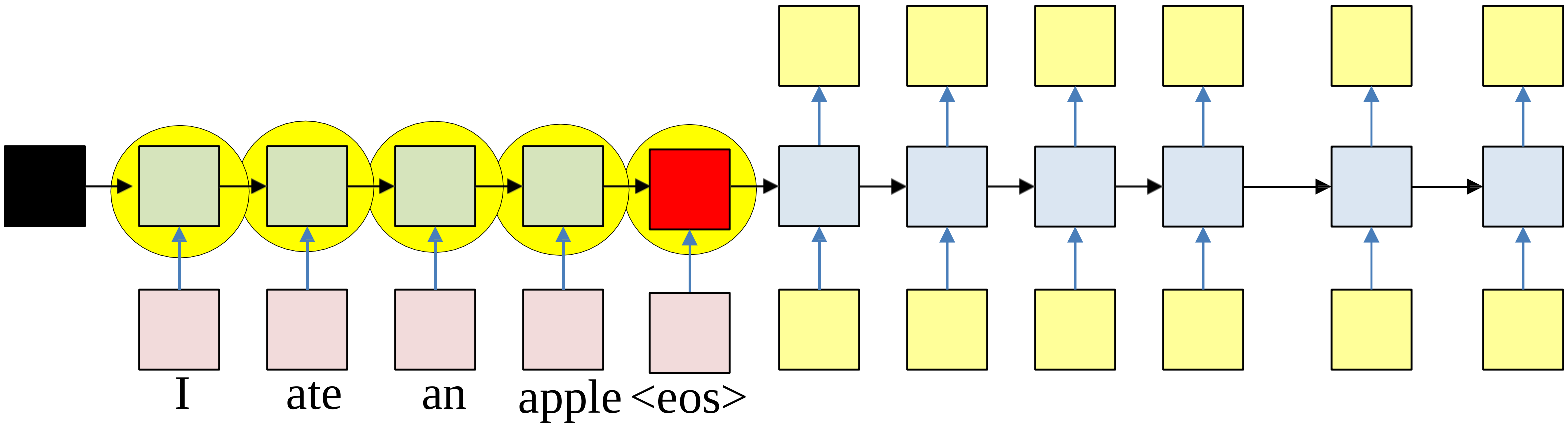
# Variants

A better model: Encoded input embedding is input to all output timesteps



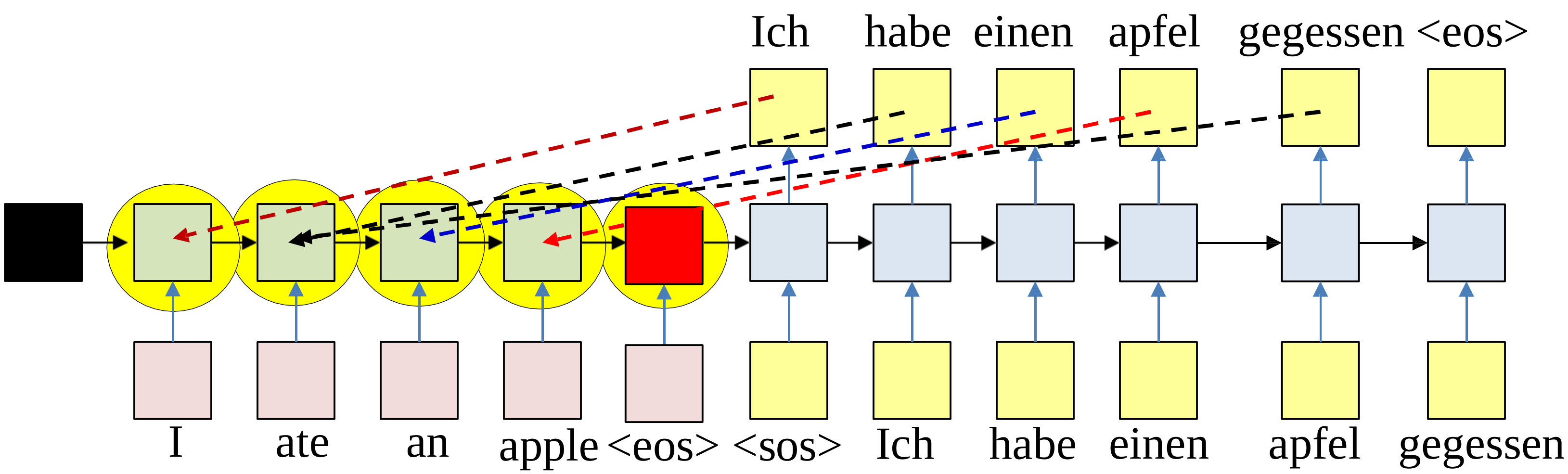



# A problem with this framework



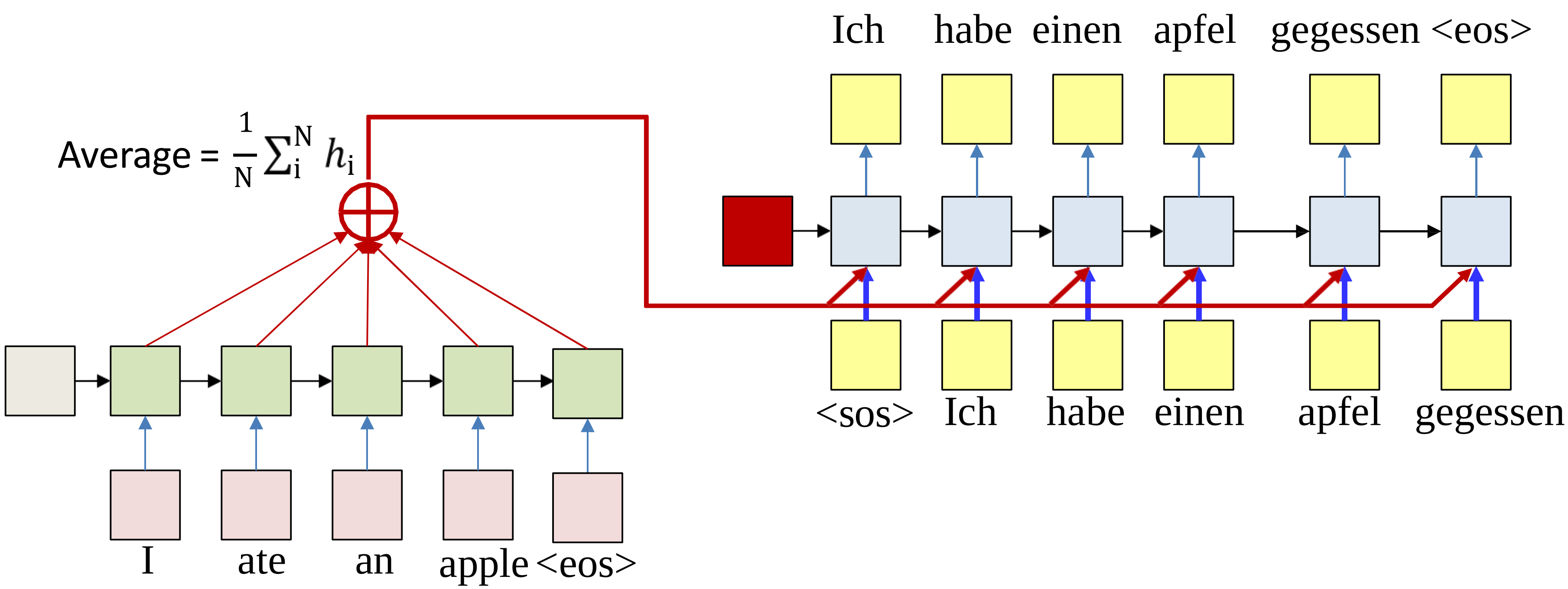

- In reality: *All* hidden values carry information
  - Some of which may be diluted by the time we get to the final state of the encoder

# A problem with this framework



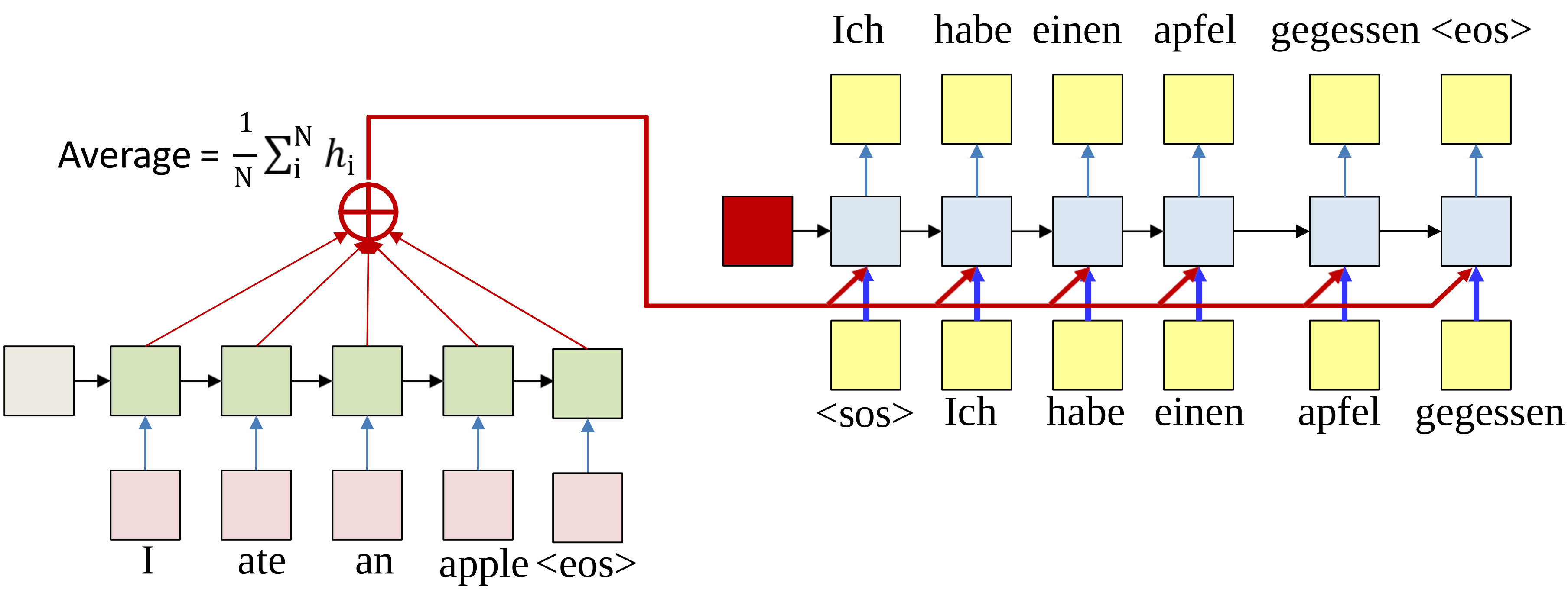

- In reality: *All* hidden values carry information
  - Some of which may be diluted by the time we get to the final state of the encoder
- *Every* output is related to the input directly
  - Not sufficient to have the encoder hidden state to *only* the initial state of the decoder
  - Misses the direct relation of the outputs to the inputs

# Using all input hidden states



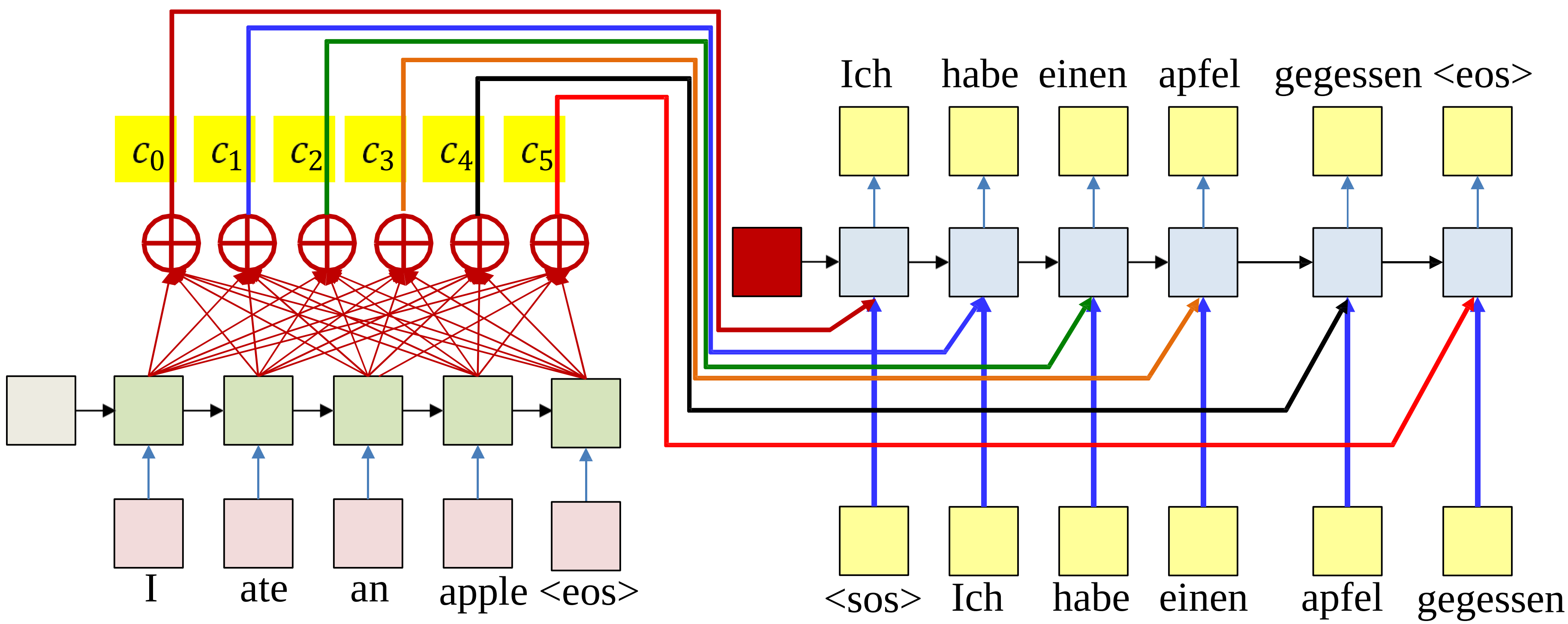

- Simple solution: Compute the average of all encoder hidden states
- Input this average to every stage of the decoder
- The initial decoder hidden state is now separate from the encoder
  - And may be a learnable parameter

# Using all input hidden states




- **Problem:** The average applies the same weight to every input
- It supplies the same average to every output word
- In practice, different outputs may be related to different inputs
  - E.g. "Ich" is most related to "I", and "habe" and "gegessen" are both most related to "ate"

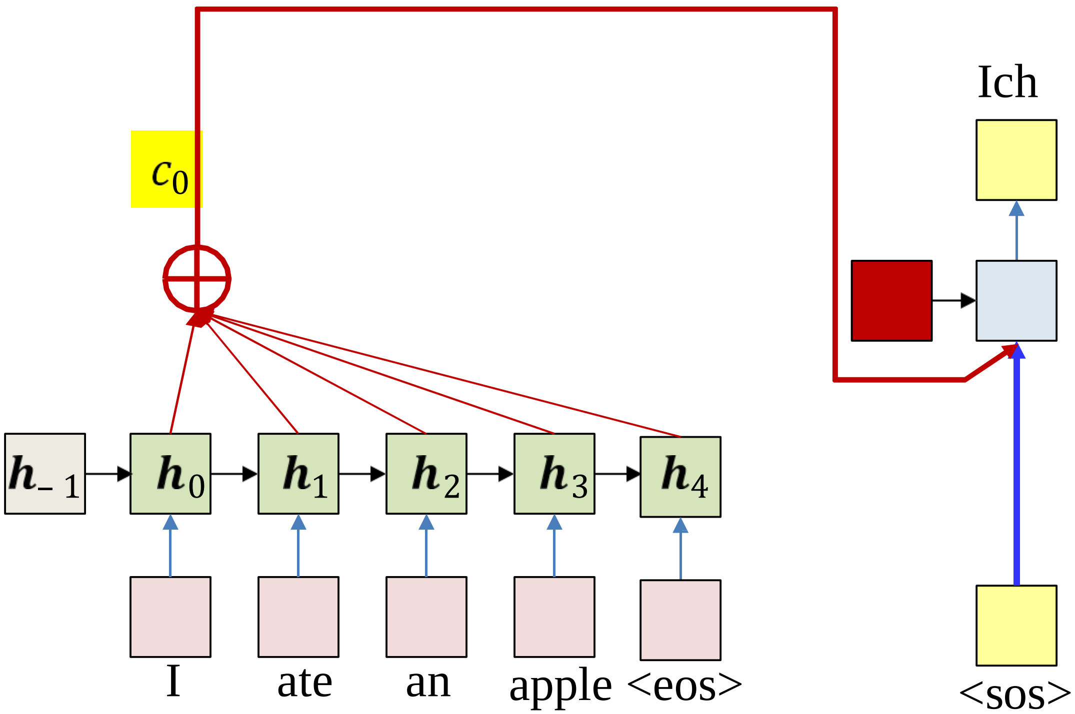
# Using all input hidden states




- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_t = \frac{1}{N}\sum_{i}^{N} w_i(t) h_i$$

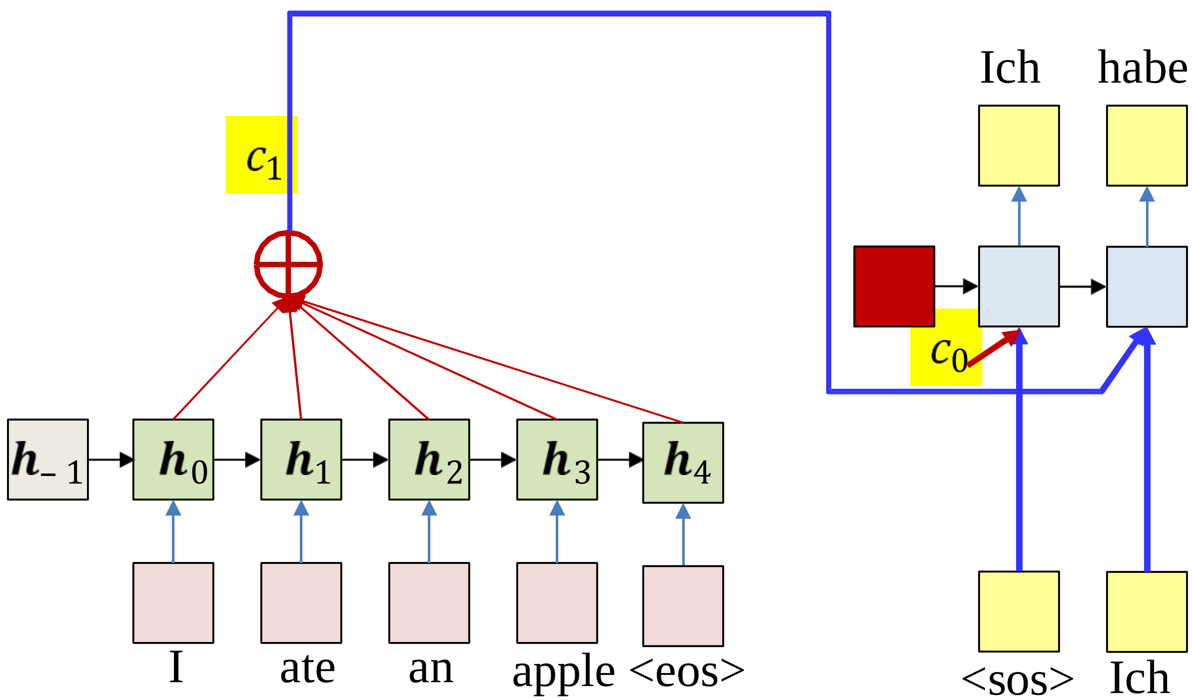
# Using all input hidden states




- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_0 = \frac{1}{N}\sum_{i}^{N} w_i(0)h_i$$

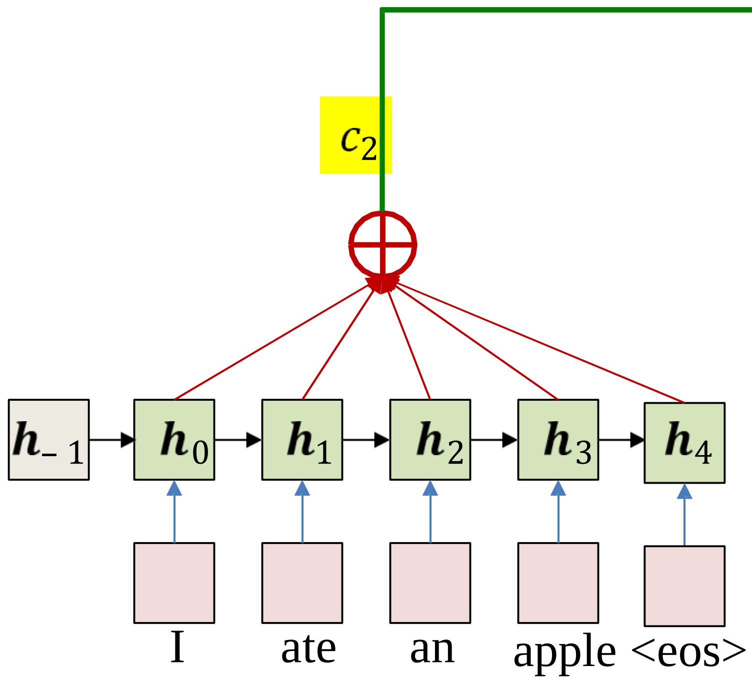
# Using all input hidden states




- **Solution:** Use a *different* weighted average for each output word
  – The weighted average provided for the kth output word is:

$$c_1 = \frac{1}{N} \sum_{i}^{N} w_i(1) h_i$$

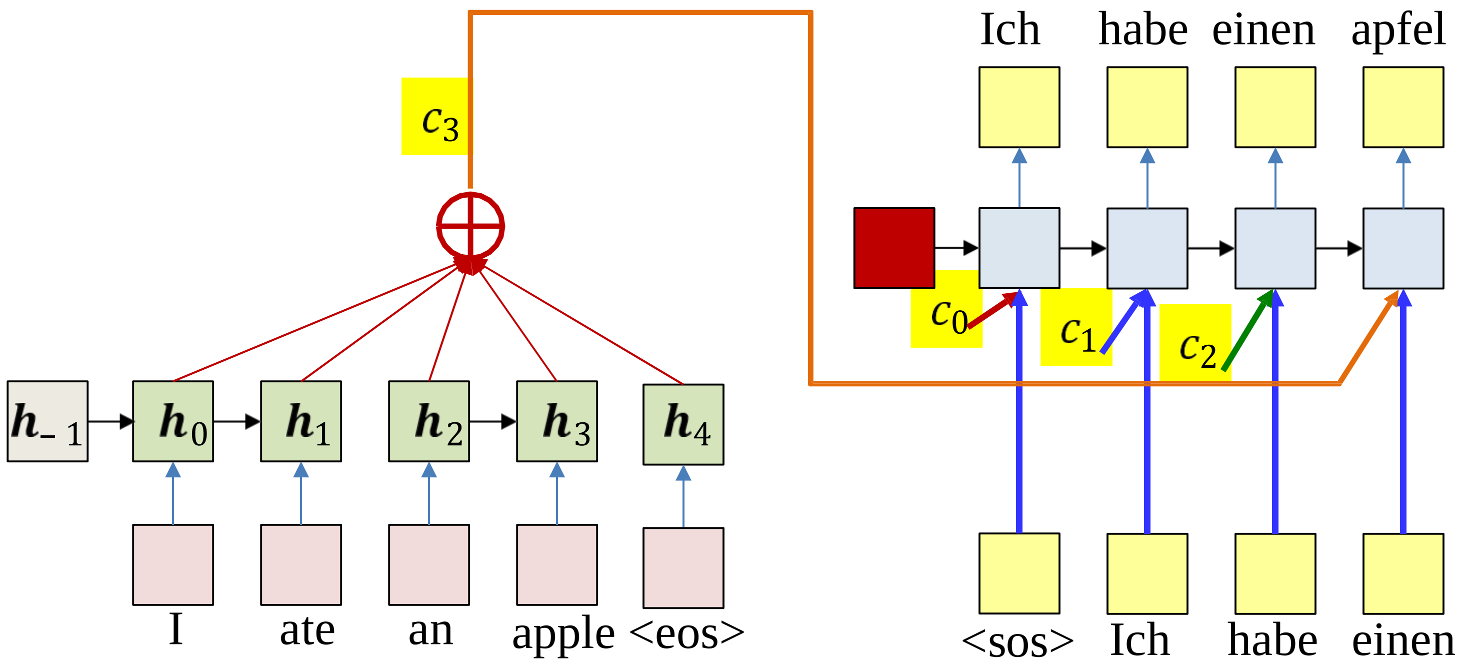
# Using all input hidden states




- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_2 = \frac{1}{N}\sum_{i}^{N} w_i(2) h_i$$

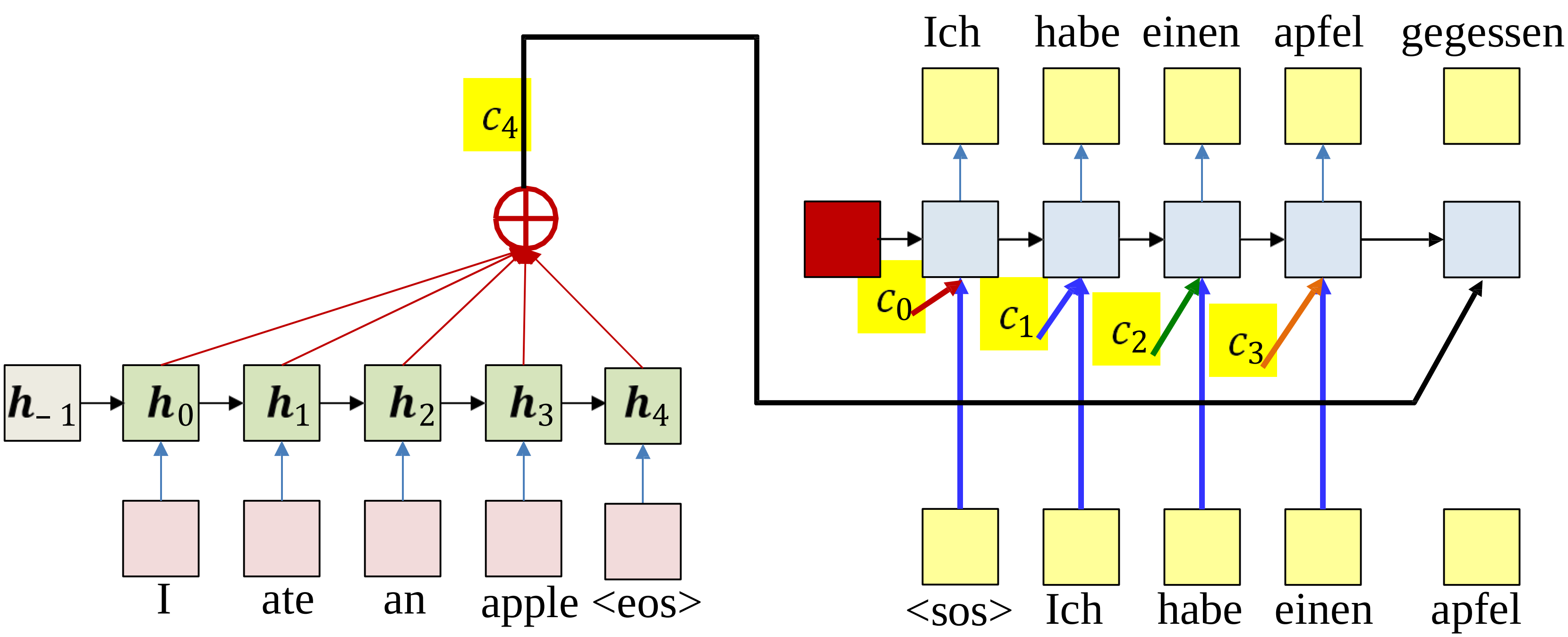
# Using all input hidden states




- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_3 = \frac{1}{N}\sum_{i}^{N} w_i(3)h_i$$

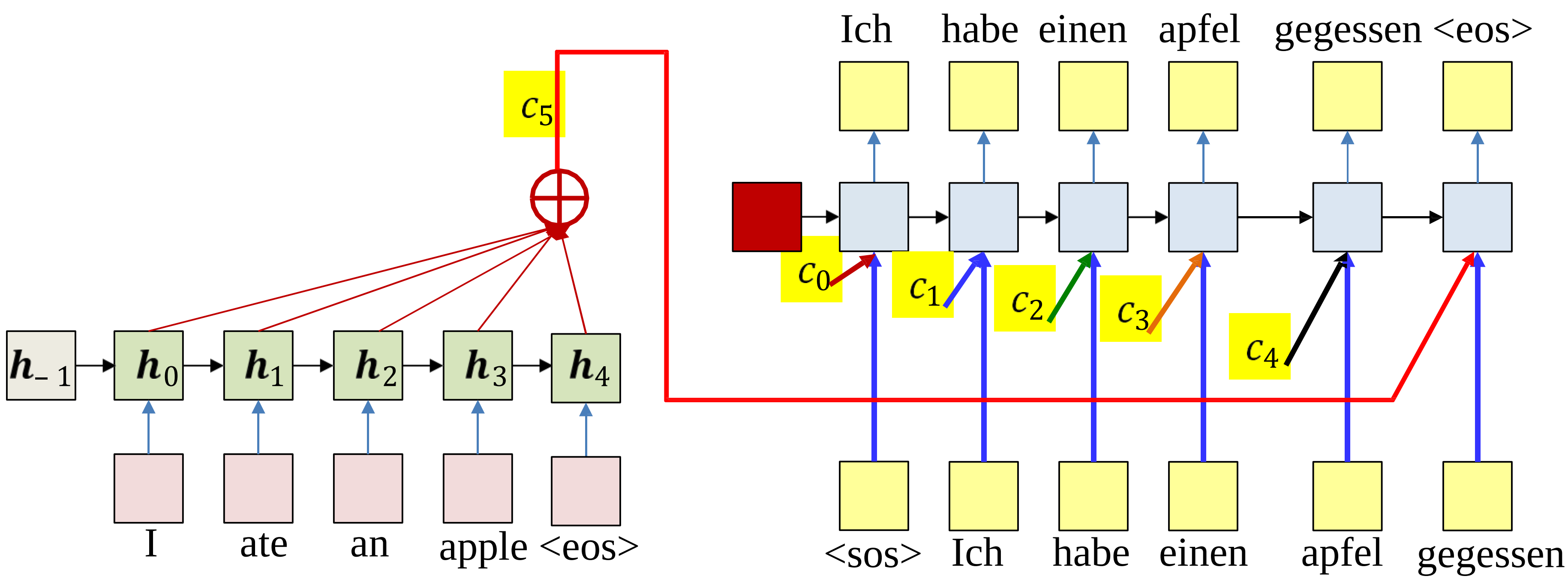
# Using all input hidden states




- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_4 = \frac{1}{N}\sum_{i}^{N} w_i(4)h_i$$

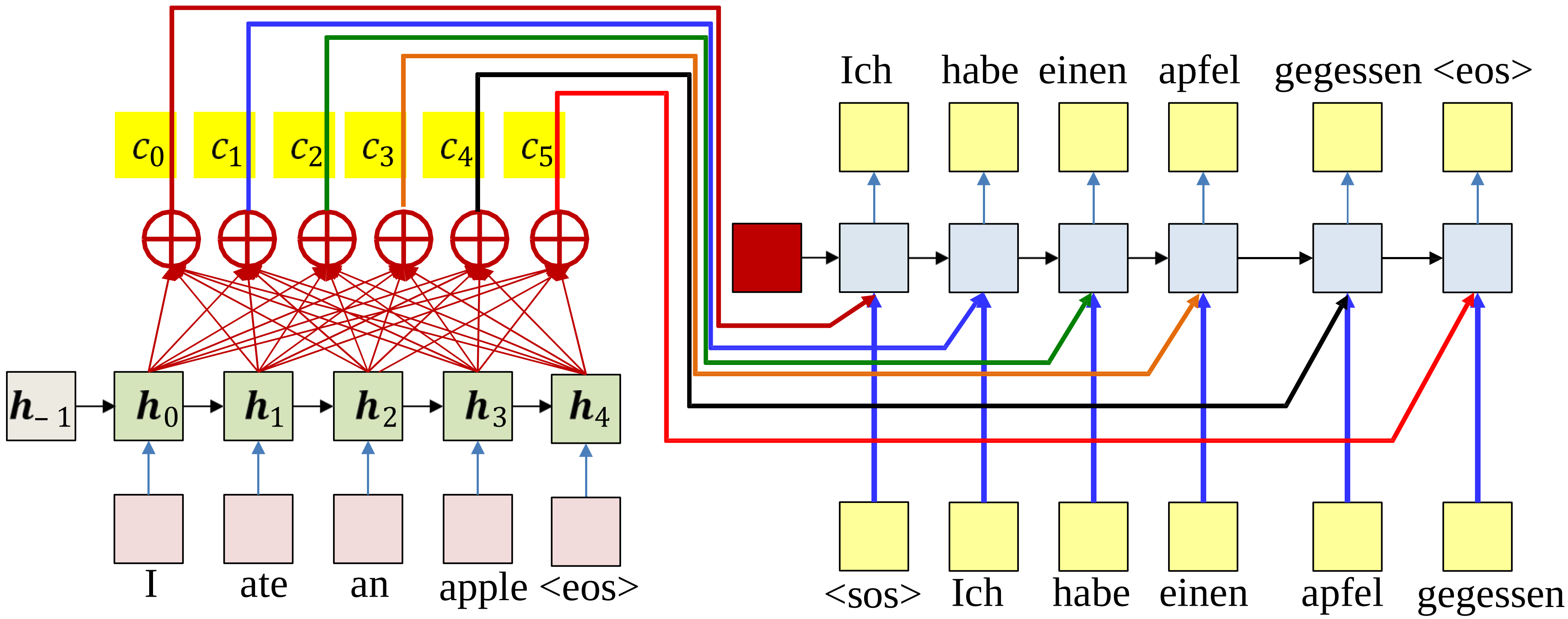
# Using all input hidden states




- **Solution:** Use a *different* weighted average for each output word
  - The weighted average provided for the kth output word is:

$$c_5 = \frac{1}{N}\sum_{i}^{N} w_i(5)h_i$$

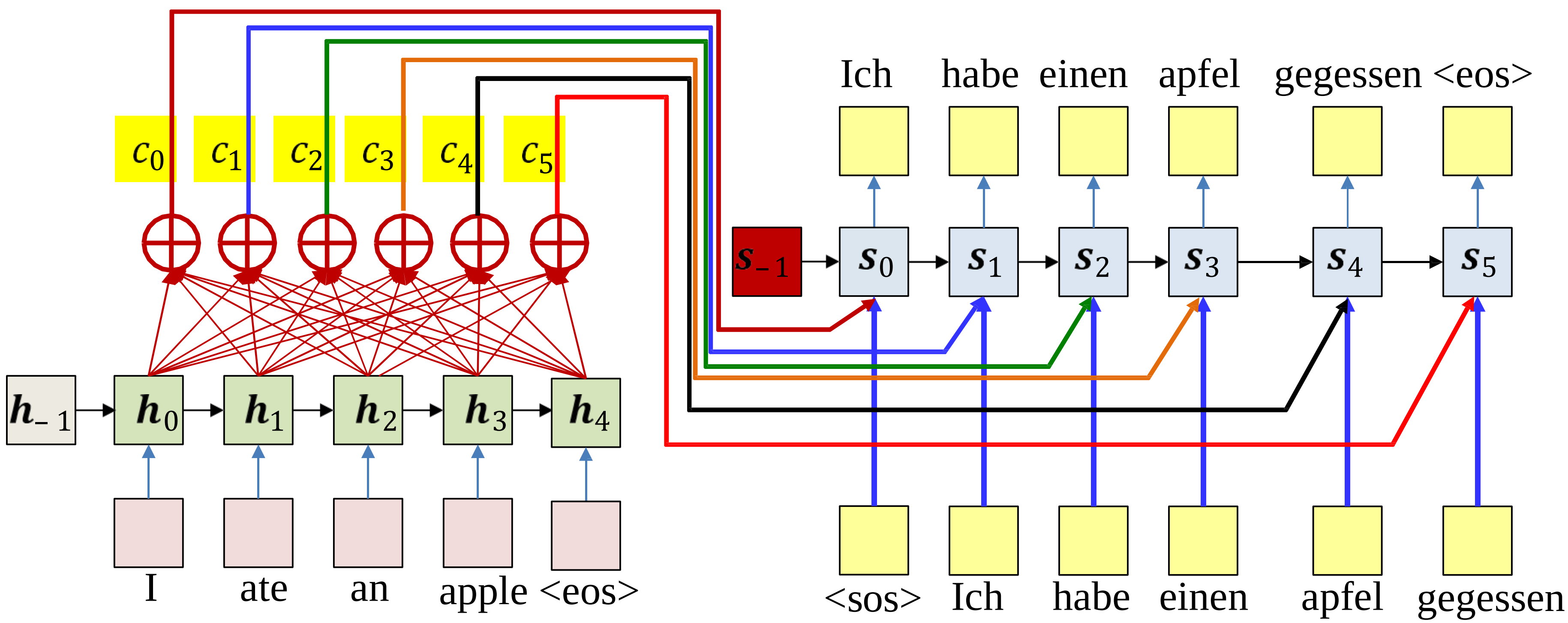
# Using all input hidden states




$$c_t = \frac{1}{N}\sum_{i}^{N} w_i(t)h_i$$

- This solution will work if the weights $w_{ki}$ can somehow be made to "focus" on the right input word
  - E.g., when predicting the word "apfel", $w_3(4)$, the weight for "apple" must be high while the rest must be low
- How do we generate such weights??

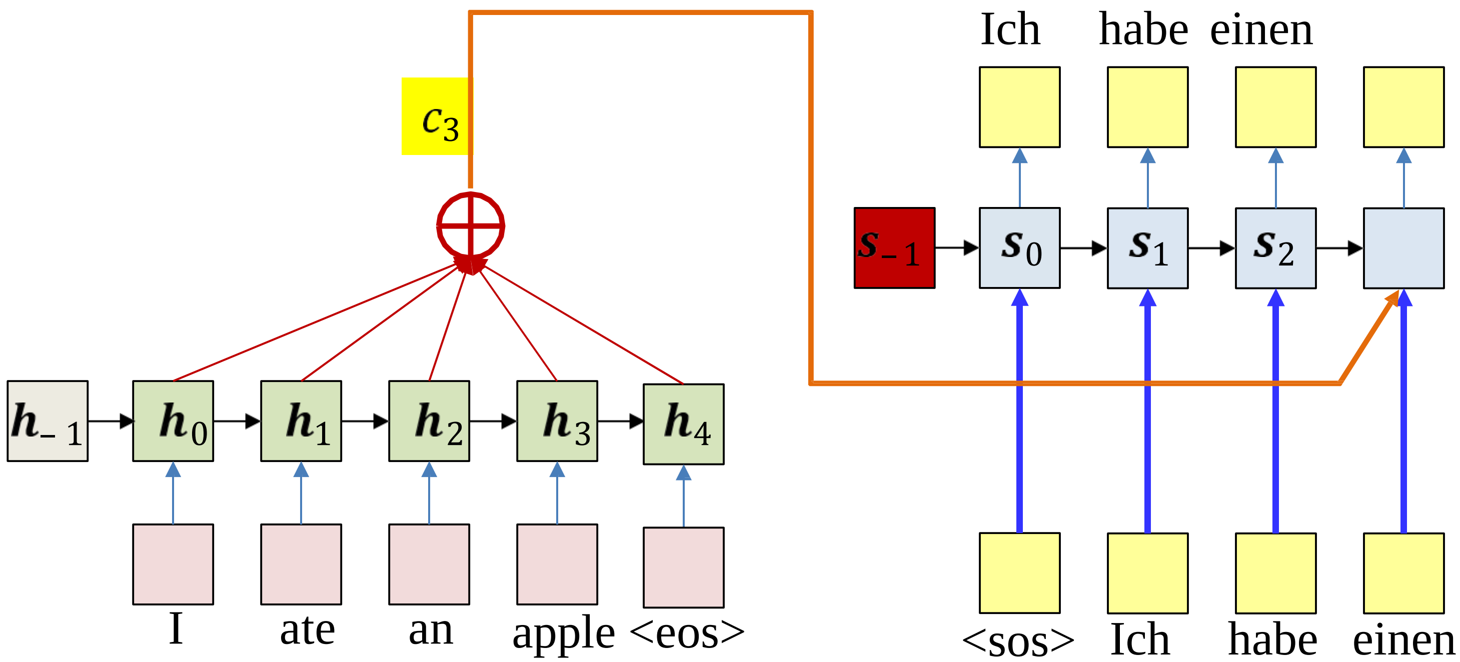
# Attention Models




$$c_t = \frac{1}{N}\sum_{i}^{N} w_i(t) h_i$$

- **Attention weights:** The weights $w_i(t)$ are dynamically computed as functions of decoder state
  - Expectation: if the model is well-trained, this will automatically "highlight" the correct input
- But how are these computed?

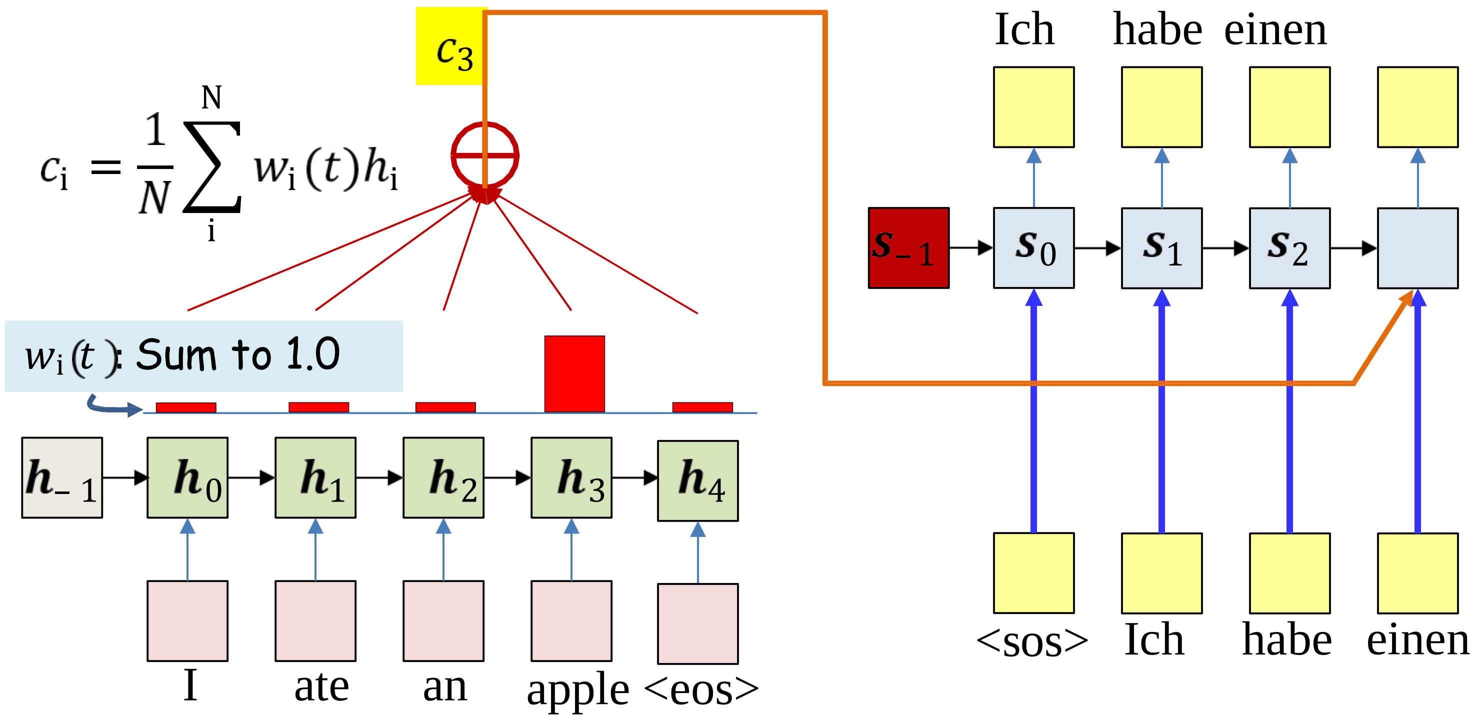
# Attention weights at time $t$




- The weights $w_i(t)$ at time $t$ must be computed from available information at time $t$
- The primary information is $s_{t-1}$ (the state at time time $t-1$)
  - Also, the input at time $t$, but generally not used for simplicity

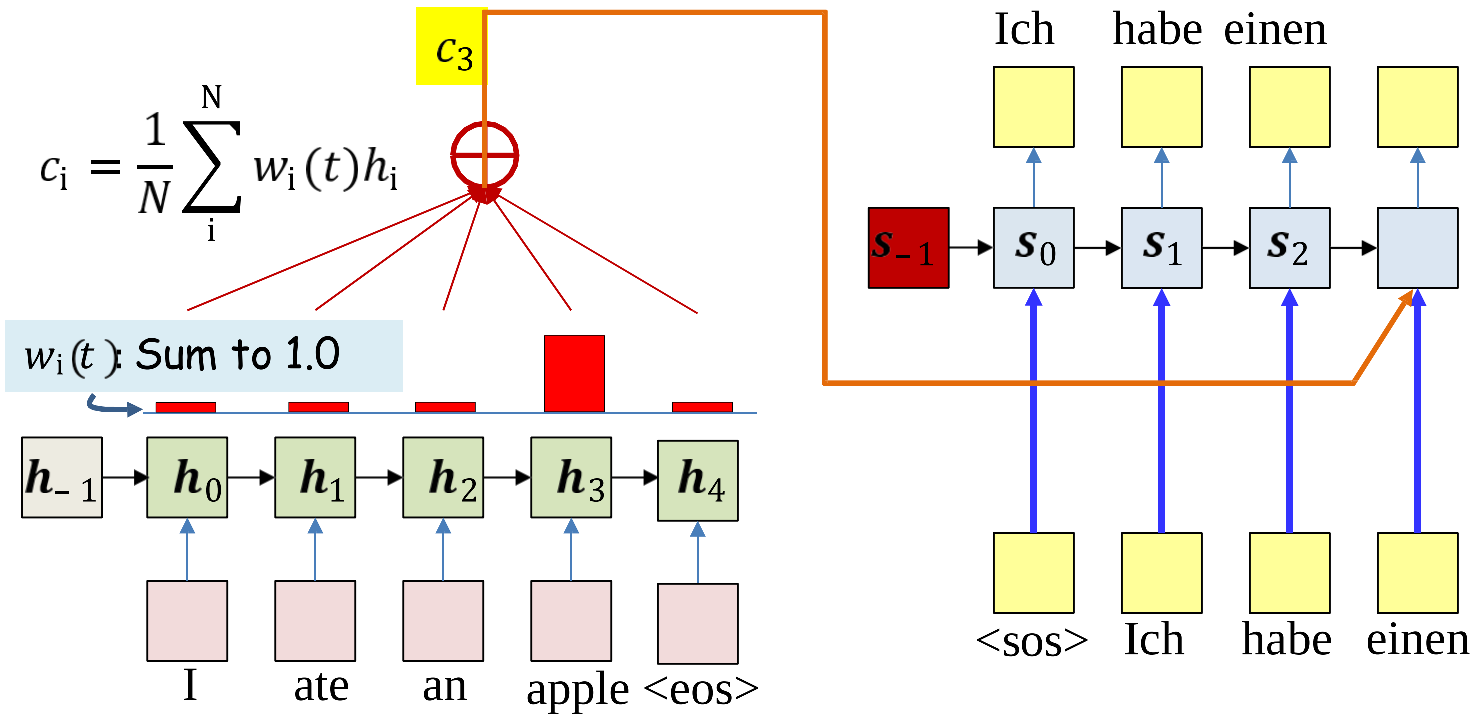
$$w_i(t) = a(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

# Requirement on attention weights




- The weights $w_i(t)$ must be positive and sum to 1.0
  - I.e. be a distribution
  - Ideally, they must be high for the most relevant inputs for the ith output and low elsewhere

# Requirement on attention weights

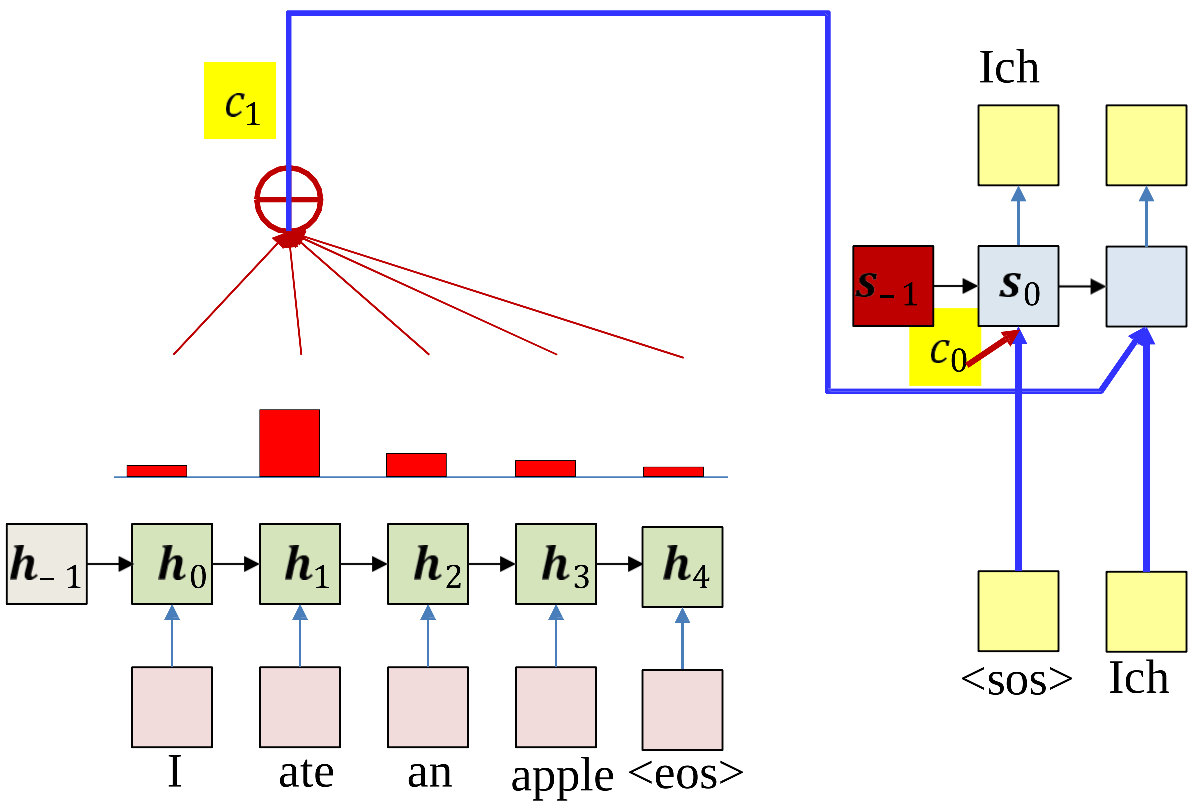



- The weights $w_i(t)$ must be positive and sum to 1.0
  - I.e. be a distribution
  - Ideally, they must be high for the most relevant inputs for the ith output and low elsewhere
- Solution: A two step weight computation
  - First compute *raw* weights (which could be +ve or –ve)
  - Then softmax them to convert them to a distribution

$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

# Using all input hidden states

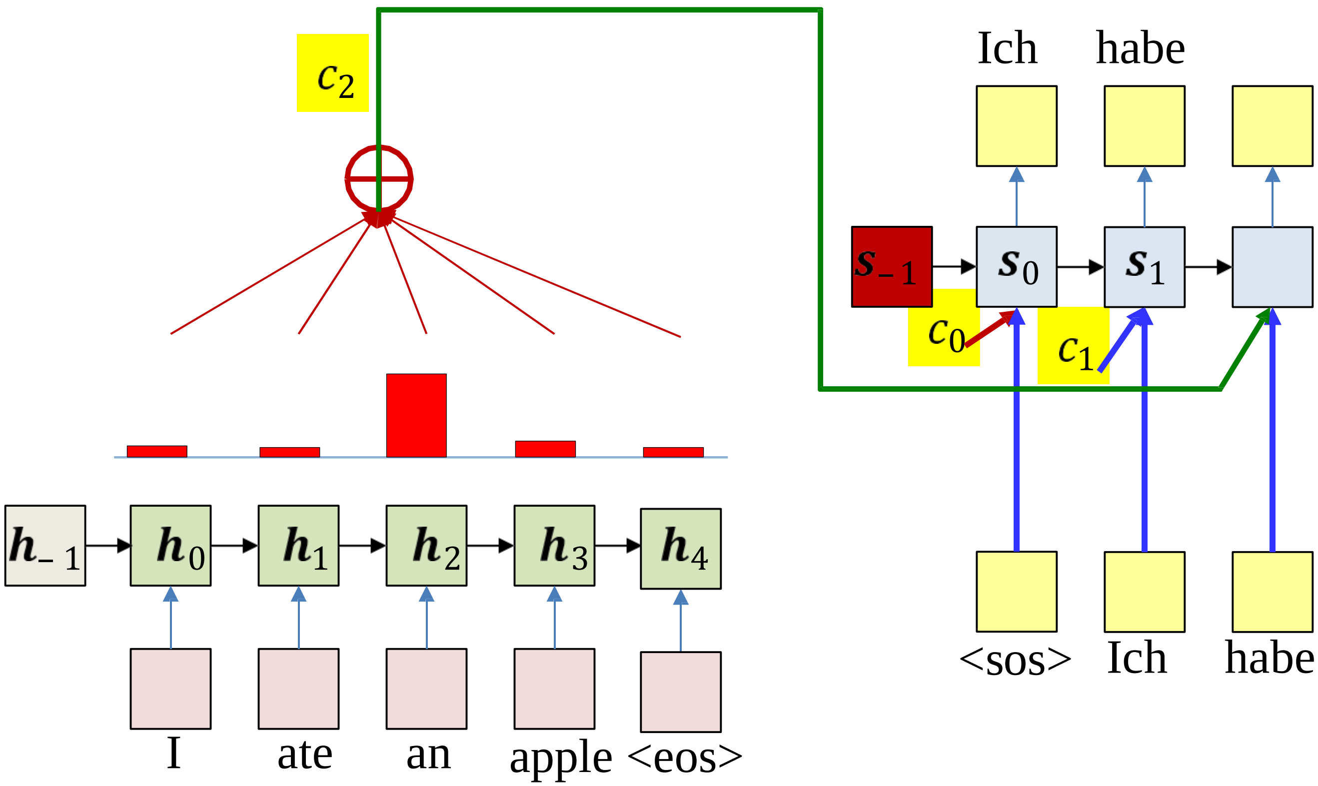



$$e_i(1) = g(\boldsymbol{h}_i, \boldsymbol{s}_0)$$

$$w_i(1) = \frac{\exp(e_i(1))}{\sum_j \exp(e_j(1))}$$

$$c_1 = \frac{1}{N}\sum_{i}^{N} w_i(1)h_i$$

# Using all input hidden states

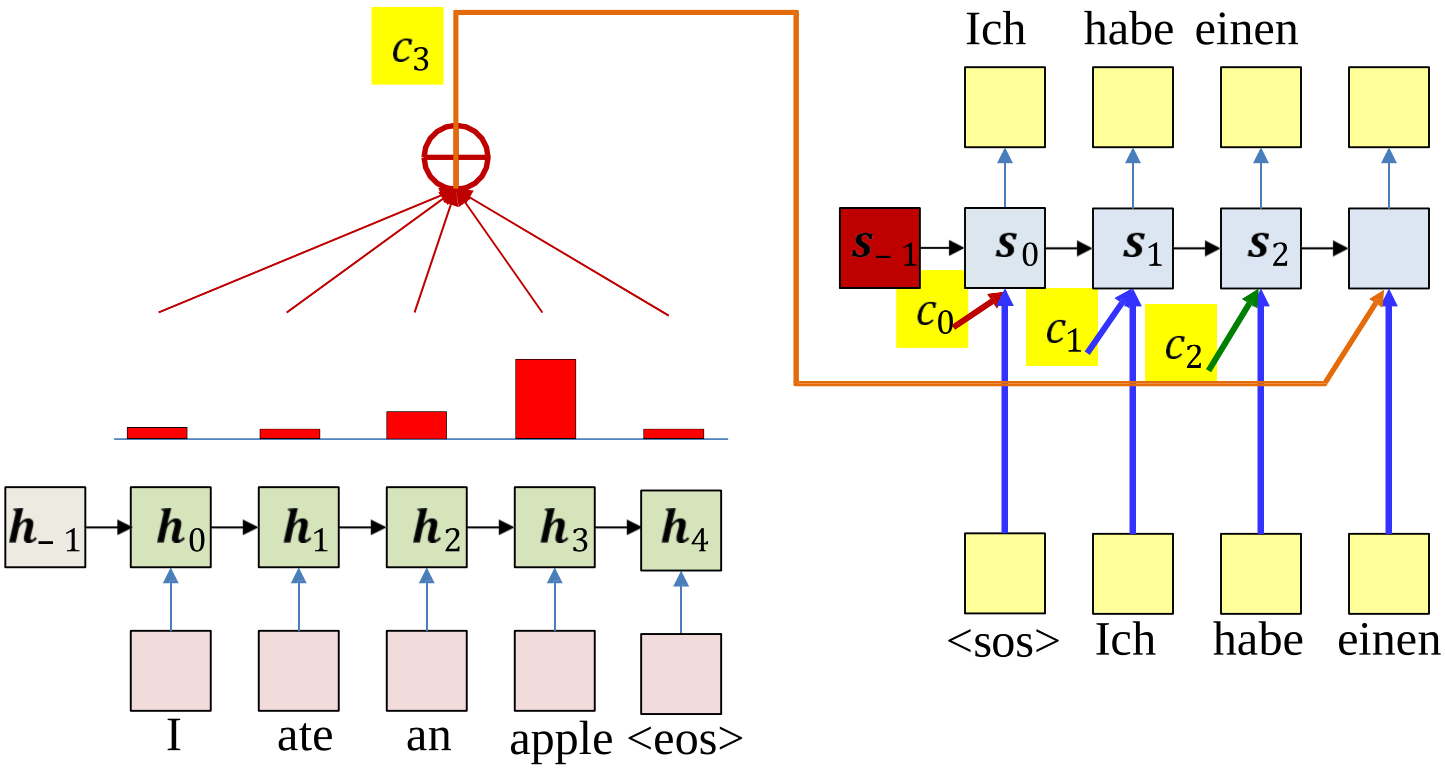



$$e_i(2) = g(\boldsymbol{h}_i, \boldsymbol{s}_1)$$

$$w_i(2) = \frac{\exp(e_i(2))}{\sum_j \exp(e_j(2))}$$

$$c_2 = \frac{1}{N}\sum_{i}^{N} w_i(2) h_i$$

# Using all input hidden states

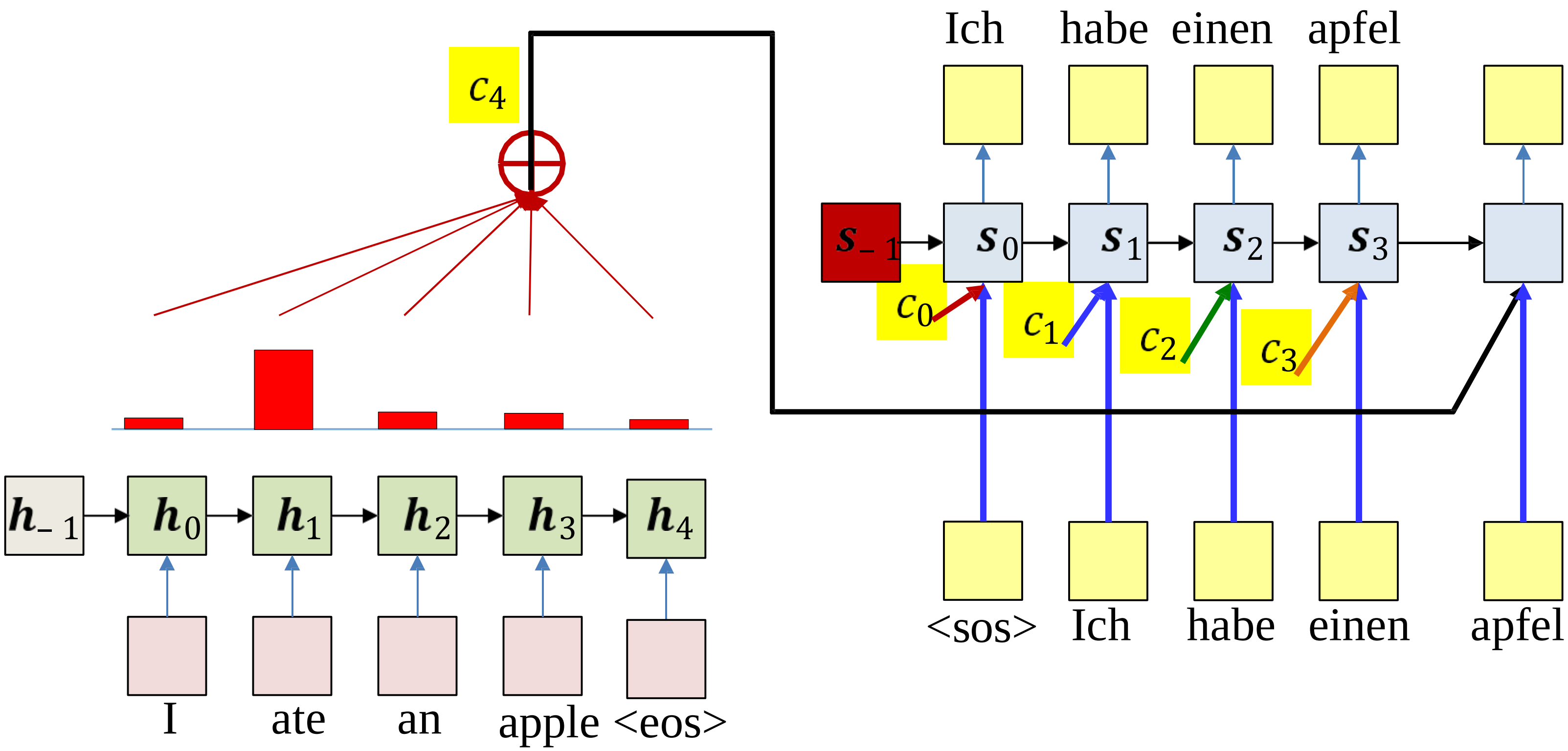



$$e_i(3) = g(\boldsymbol{h}_i, \boldsymbol{s}_2)$$

$$w_i(3) = \frac{\exp(e_i(3))}{\sum_j \exp(e_j(3))}$$

$$c_3 = \frac{1}{N}\sum_{i}^{N} w_i(3) h_i$$

# Using all input hidden states

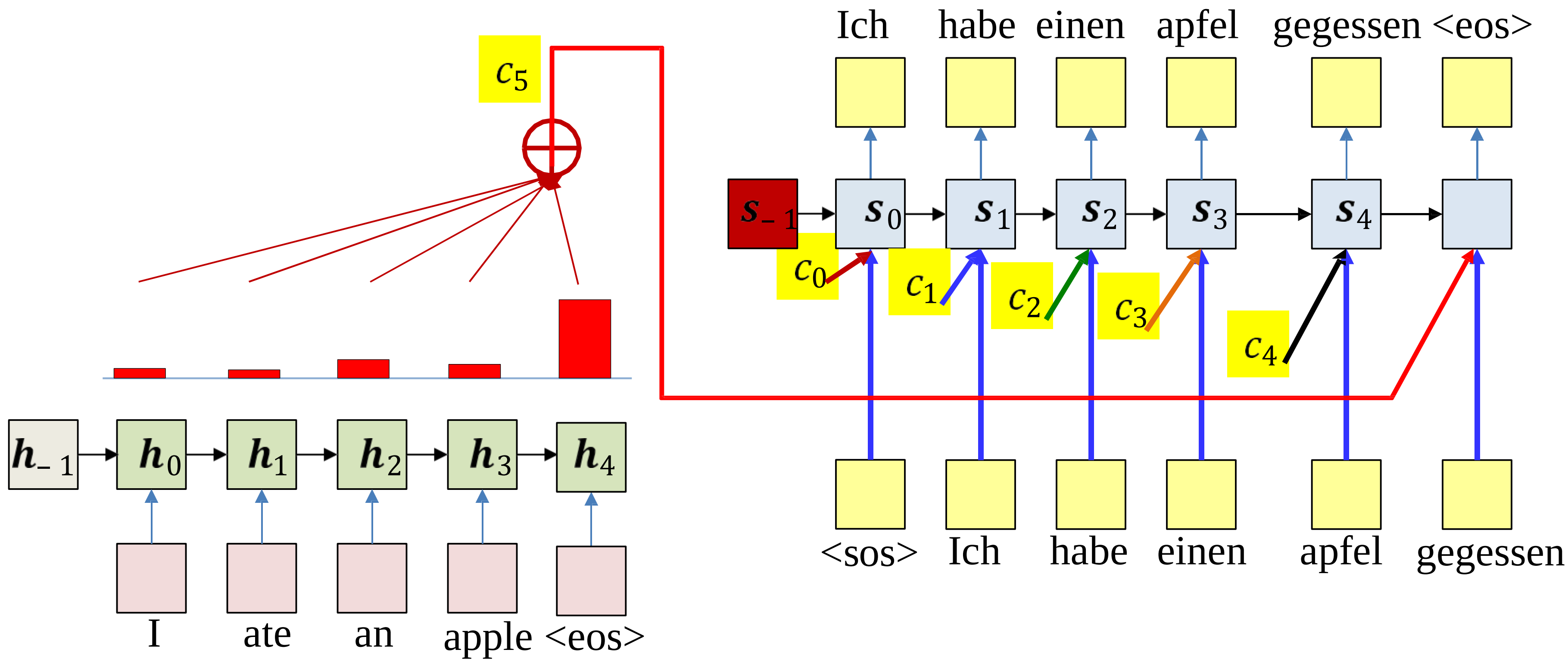



$$e_i(4) = g(h_i, s_3)$$

$$w_i(4) = \frac{\exp(e_i(4))}{\sum_j \exp(e_j(4))}$$

$$c_4 = \frac{1}{N}\sum_i^N w_i(4) h_i$$

# Using all input hidden states

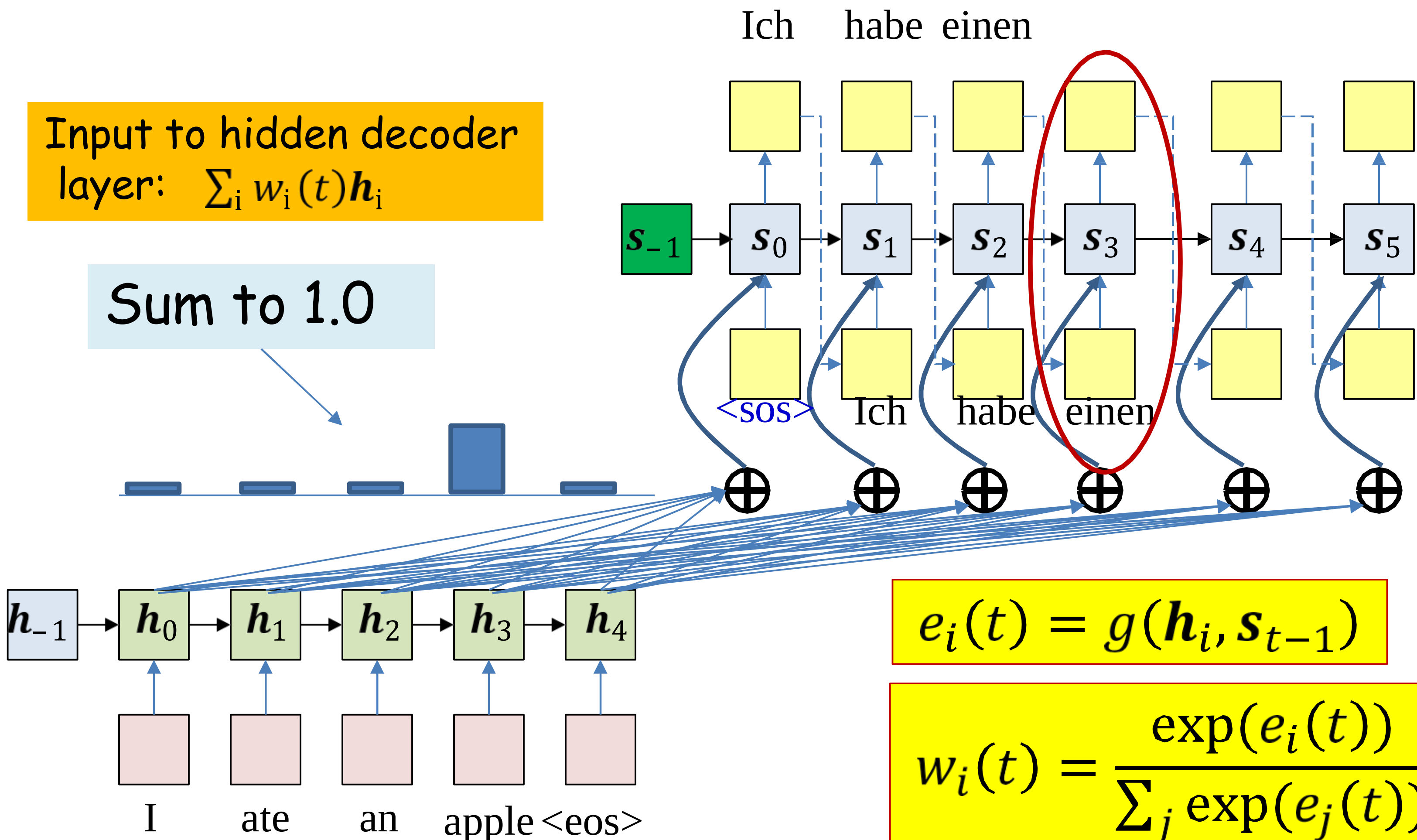



$$e_i(5) = g(h_i, s_4)$$

$$w_i(5) = \frac{\exp(e_i(5))}{\sum_j \exp(e_j(5))}$$

$$c_5 = \frac{1}{N}\sum_{i}^{N} w_i(5) h_i$$

# Summarizing the computation

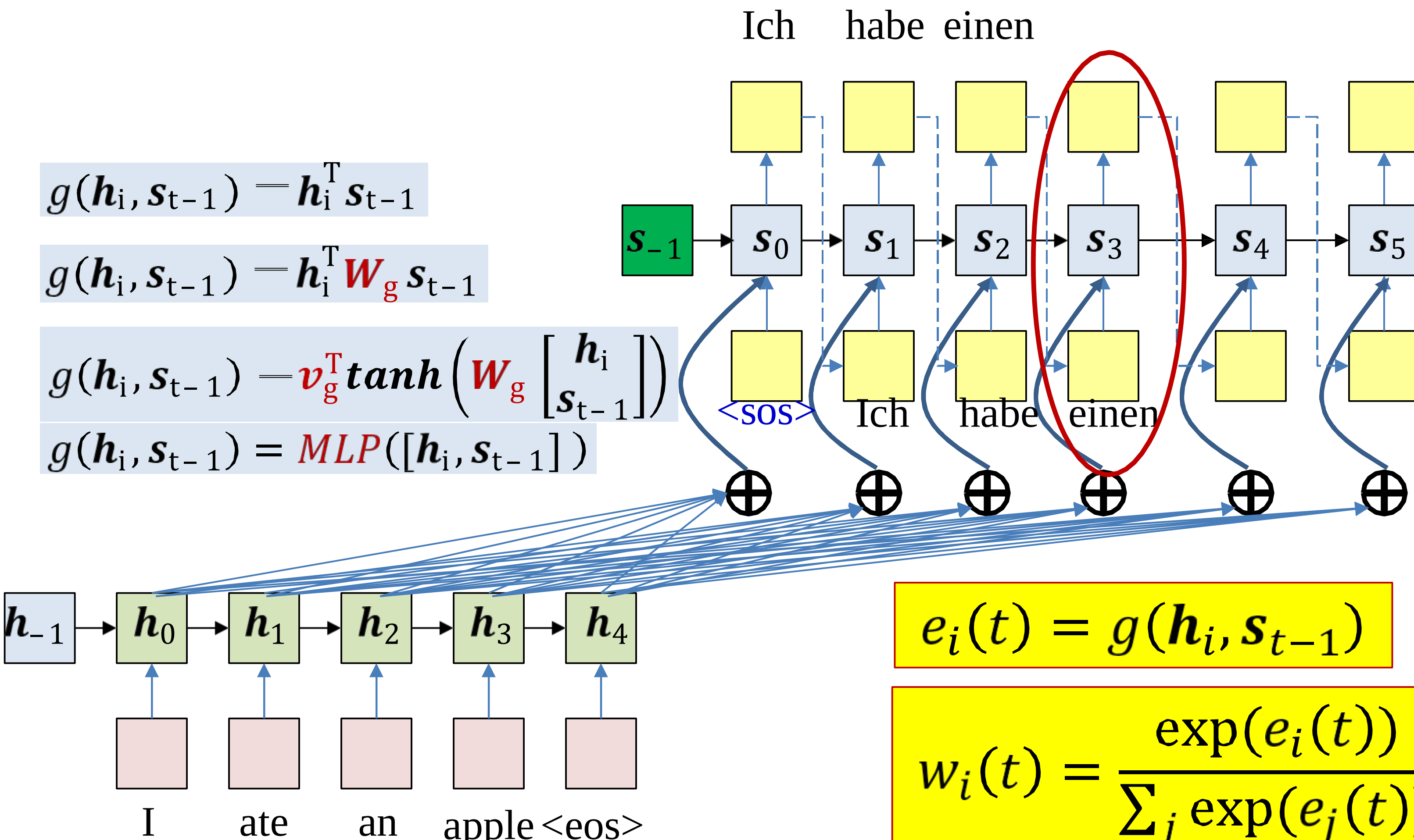



$$e_i(t) = g(\boldsymbol{h}_i, \boldsymbol{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

- “Raw” weight at any time: A function $g()$ that works on the two hidden states
- Actual weight: softmax over raw weights

# Attention models




$$g(\mathbf{h}_i, \mathbf{s}_{t-1}) = \mathbf{h}_i^T \mathbf{s}_{t-1}$$

$$g(\mathbf{h}_i, \mathbf{s}_{t-1}) = \mathbf{h}_i^T \mathbf{W}_g \mathbf{s}_{t-1}$$

$$g(\mathbf{h}_i, \mathbf{s}_{t-1}) = \mathbf{v}_g^T \boldsymbol{tanh}\left(\mathbf{W}_g \begin{bmatrix} \mathbf{h}_i \\ \mathbf{s}_{t-1} \end{bmatrix}\right)$$

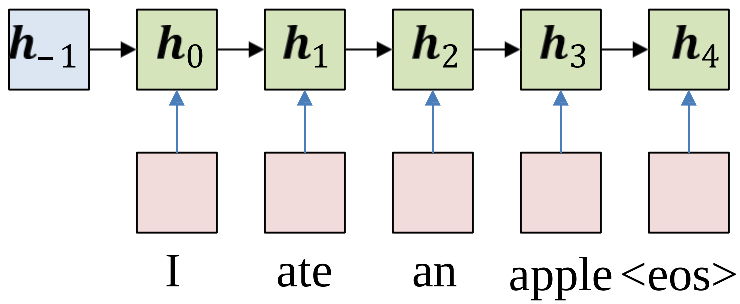
$$g(\mathbf{h}_i, \mathbf{s}_{t-1}) = MLP([\mathbf{h}_i, \mathbf{s}_{t-1}])$$

$$e_i(t) = g(\mathbf{h}_i, \mathbf{s}_{t-1})$$

$$w_i(t) = \frac{\exp(e_i(t))}{\sum_j \exp(e_j(t))}$$

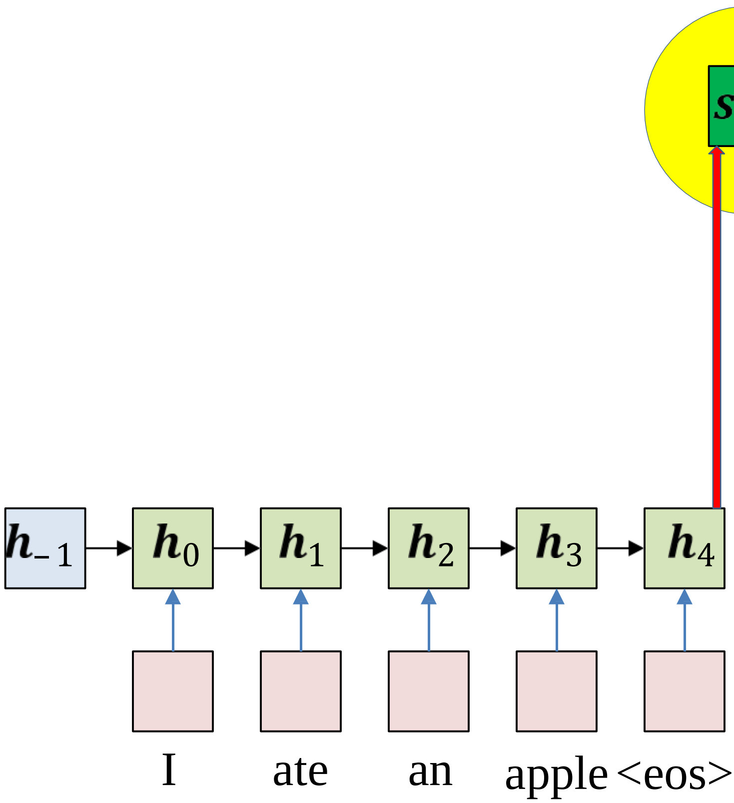
- Typical options for $g()$…
  - Variables in red are to be learned

# Converting an input (forward pass)




- Pass the input through the encoder to produce hidden representations $h_i$

# Converting an input (forward pass)




**What is this?**
**Multiple options**

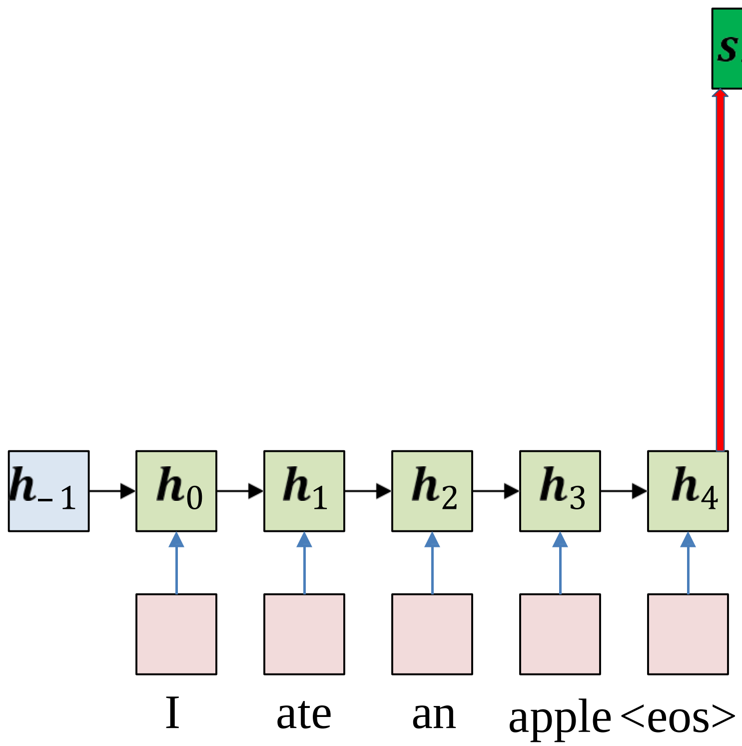
Simplest: $\boldsymbol{s}_{-1} = 0$
Alternative: learn $\boldsymbol{s}_{-1}$
Alternative 2: $\boldsymbol{s}_{-1} = \boldsymbol{h}_{\mathrm{N}}$
If $\boldsymbol{s}$ and $\boldsymbol{h}$ are different sizes:

$$\boldsymbol{s}_{-1} = \boldsymbol{W}_{\mathrm{s}} \boldsymbol{h}_{\mathrm{N}}$$

$\boldsymbol{W}_{\mathrm{s}}$ is learnable parameter

- Initialize decoder hidden state

# Converting an input (forward pass)




$$g(\boldsymbol{h}_i, \boldsymbol{s}_{-1}) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_{-1}$$

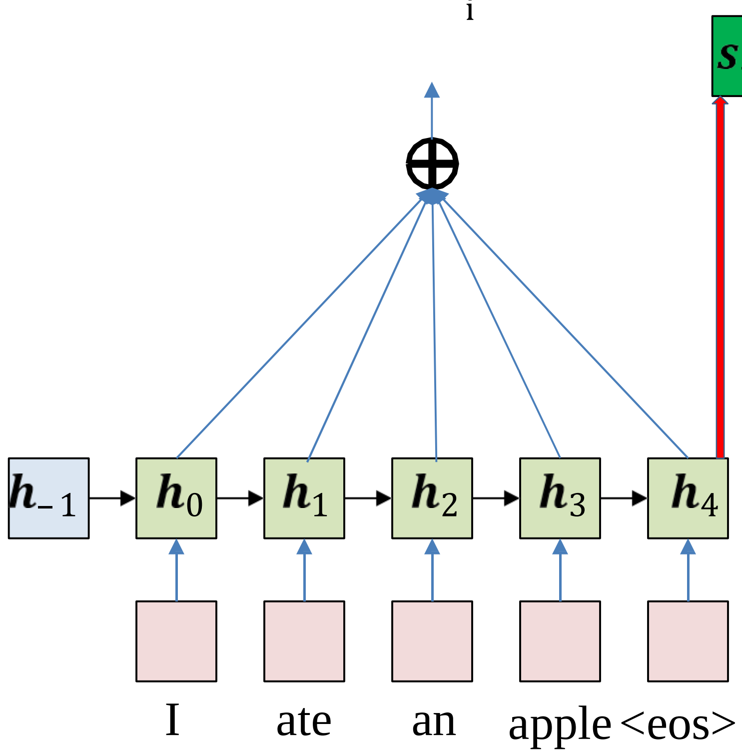
$$e_i(0) = g(\boldsymbol{h}_i, \boldsymbol{s}_{-1})$$

$$w_i(0) = \frac{\exp(e_i(0))}{\sum_j \exp(e_j(0))}$$

- Compute weights (for every $\boldsymbol{h}_i$) for first output

# Converting an input (forward pass)

$$c_0 = \sum_i w_i(0) \boldsymbol{h}_i$$



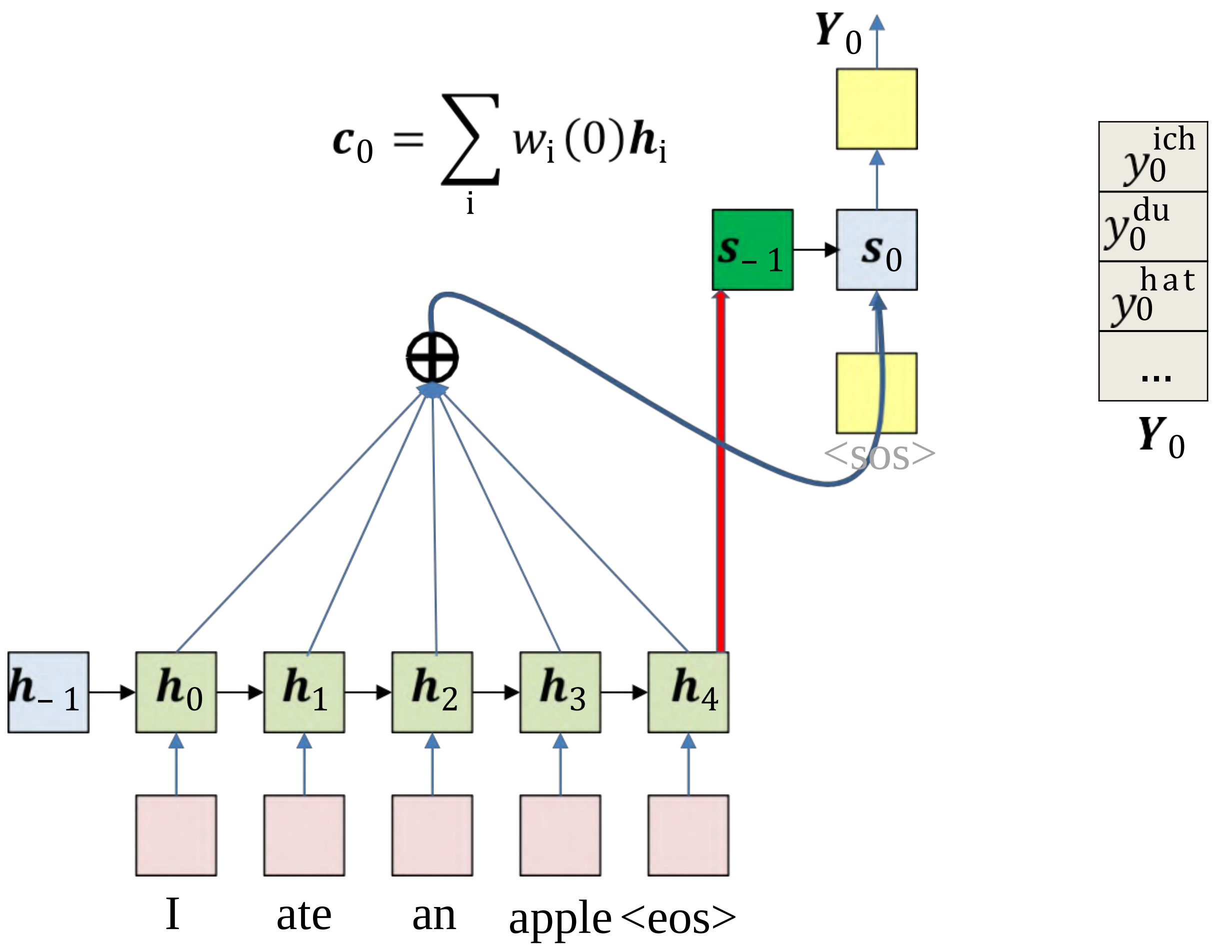

$$g(\boldsymbol{h}_i, \boldsymbol{s}_{-1}) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_{-1}$$

$$e_i(0) = g(\boldsymbol{h}_i, \boldsymbol{s}_{-1})$$

$$w_i(0) = \frac{\exp(e_i(0))}{\sum_j \exp(e_j(0))}$$

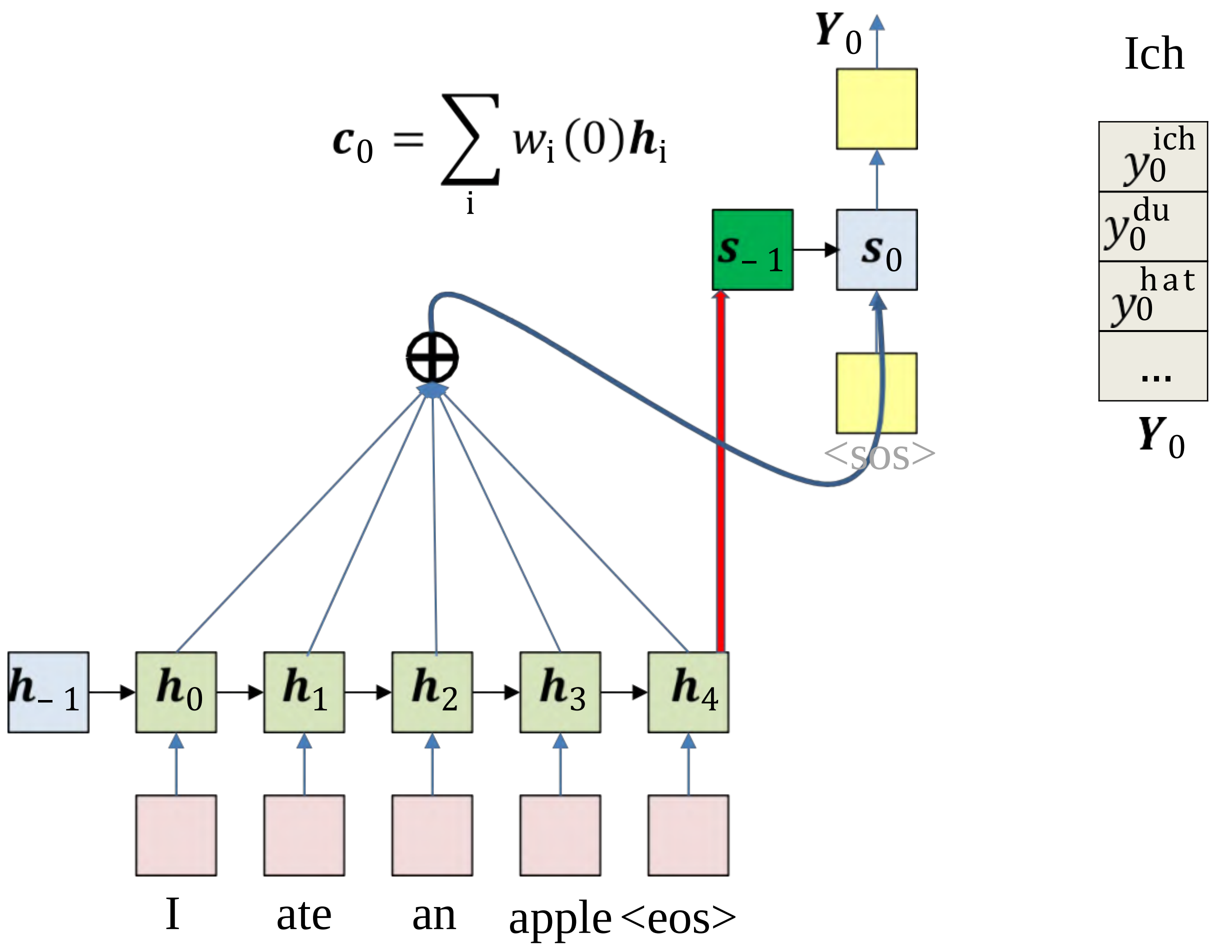
- Compute weights (for every $\boldsymbol{h}_i$) for first output
- Compute weighted combination of hidden values

# Converting an input (forward pass)

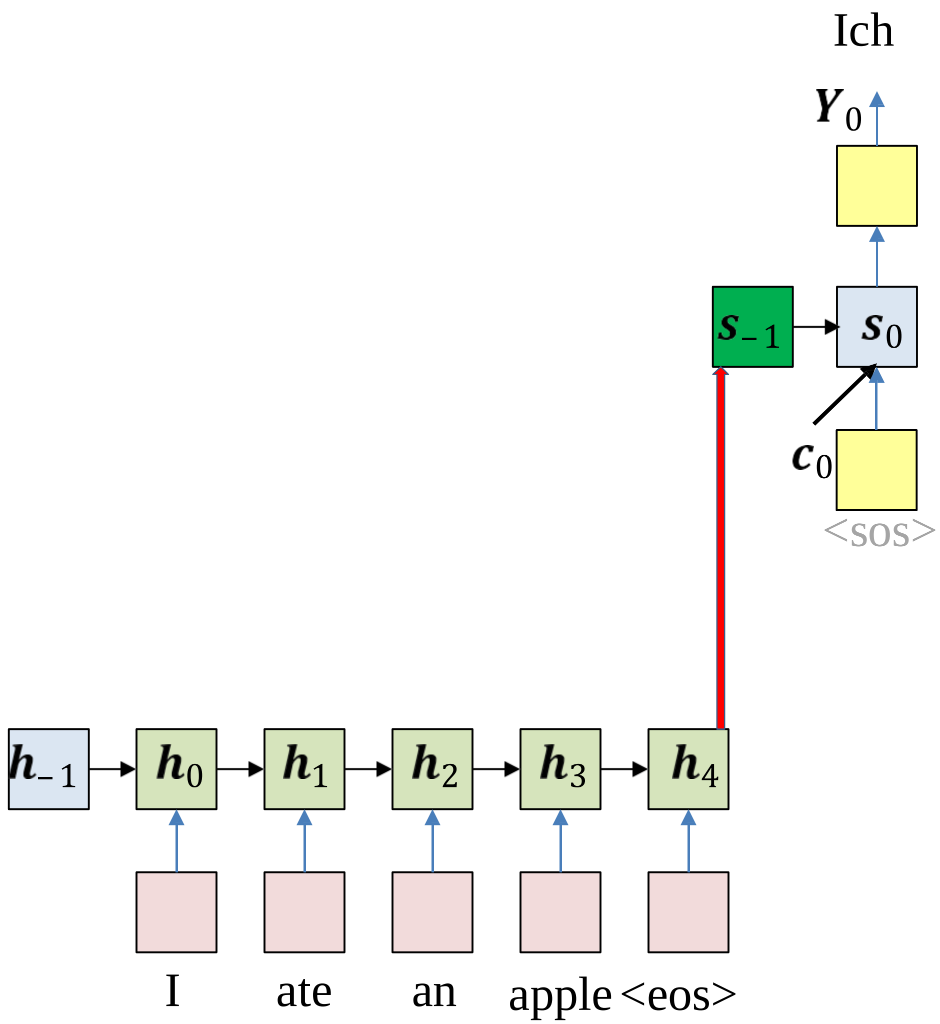



- Produce the first output
  - Will be distribution over words

# Converting an input (forward pass)




- Produce the first output
  - Will be distribution over words
  - Draw a word from the distribution

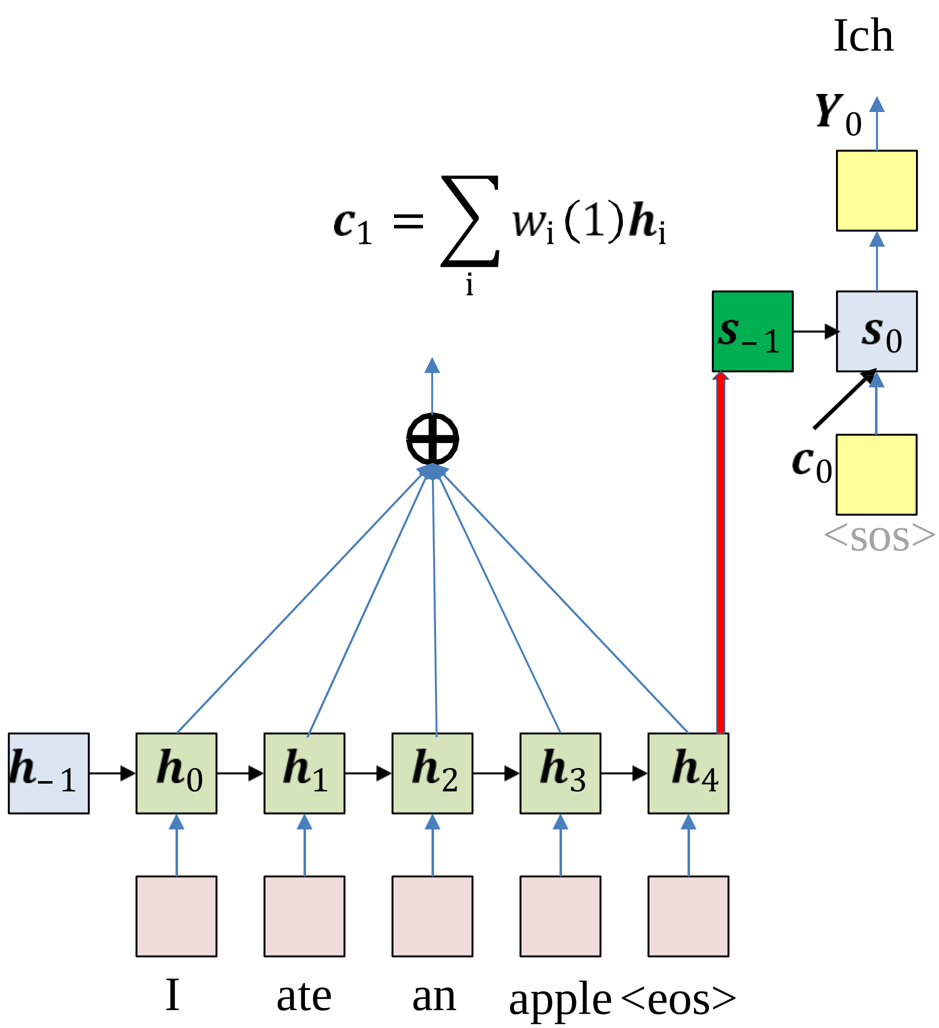



$$g(\boldsymbol{h}_i, \boldsymbol{s}_0) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_0$$

$$e_i(1) = g(\boldsymbol{h}_i, \boldsymbol{s}_0)$$

$$w_i(1) = \frac{\exp(e_i(1))}{\sum_j \exp(e_j(1))}$$

- Compute the weights for all instances for time = 1

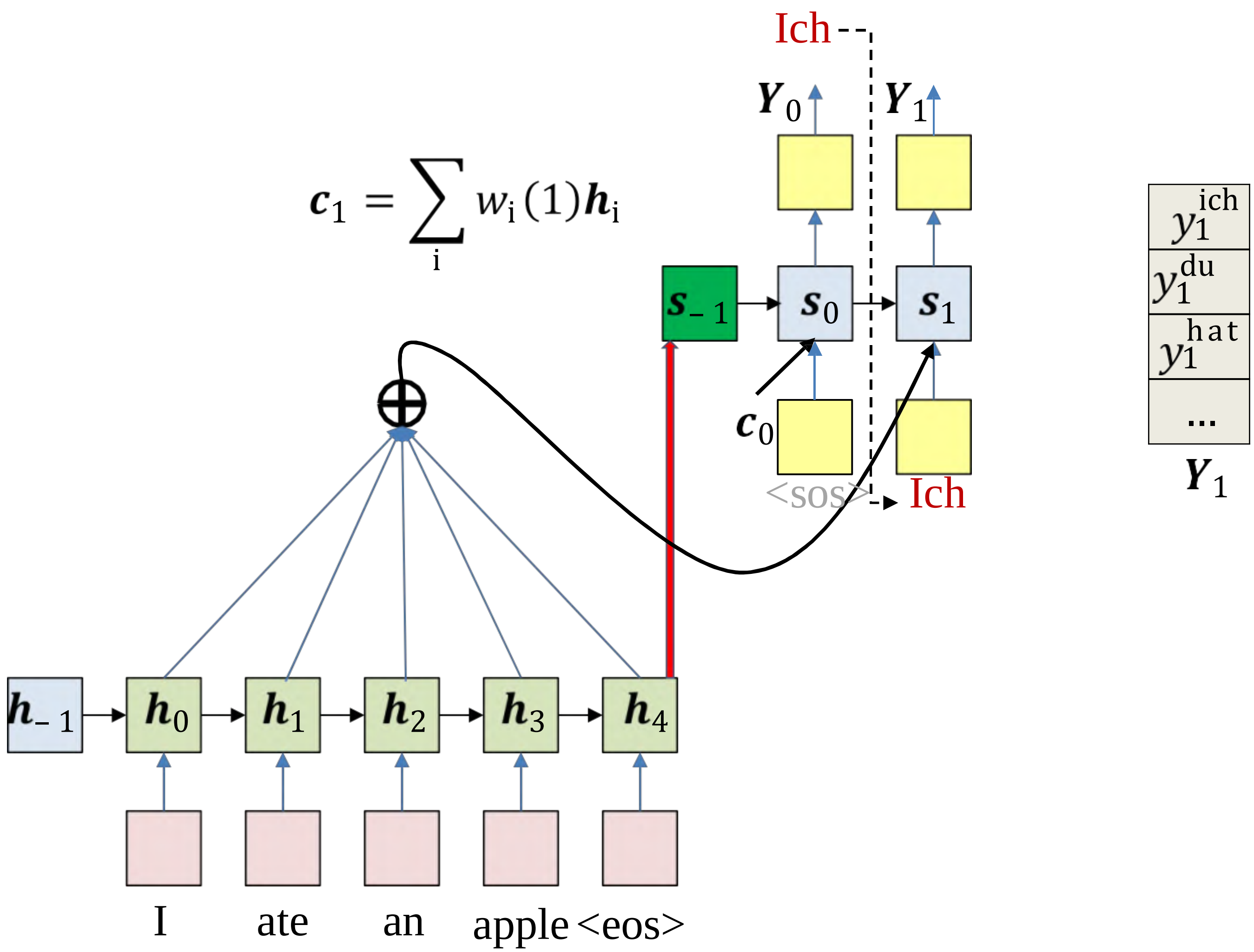



$$g(\boldsymbol{h}_i, \boldsymbol{s}_0) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_0$$

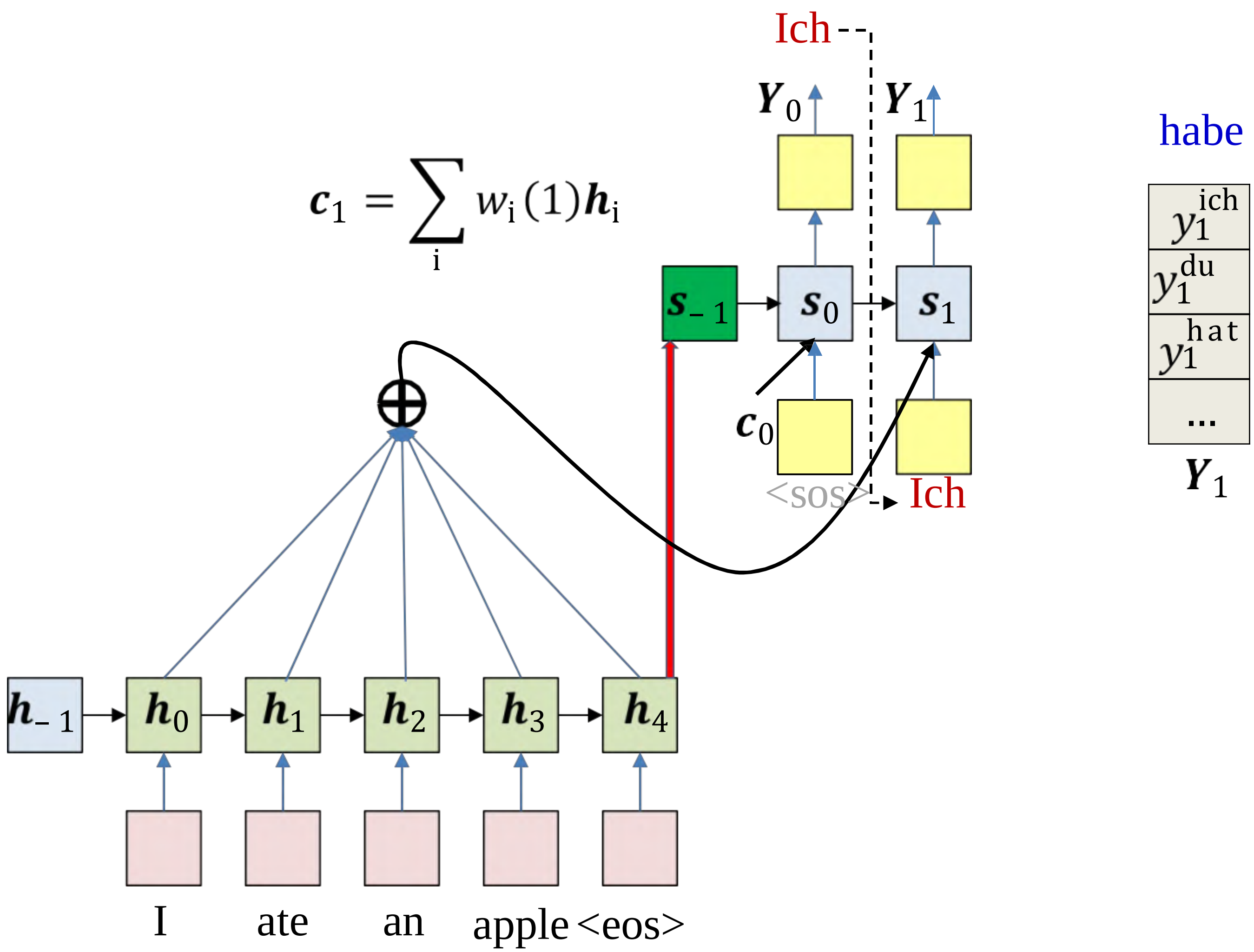
$$e_i(1) = g(\boldsymbol{h}_i, \boldsymbol{s}_0)$$

$$w_i(1) = \frac{\exp(e_i(1))}{\sum_j \exp(e_j(1))}$$

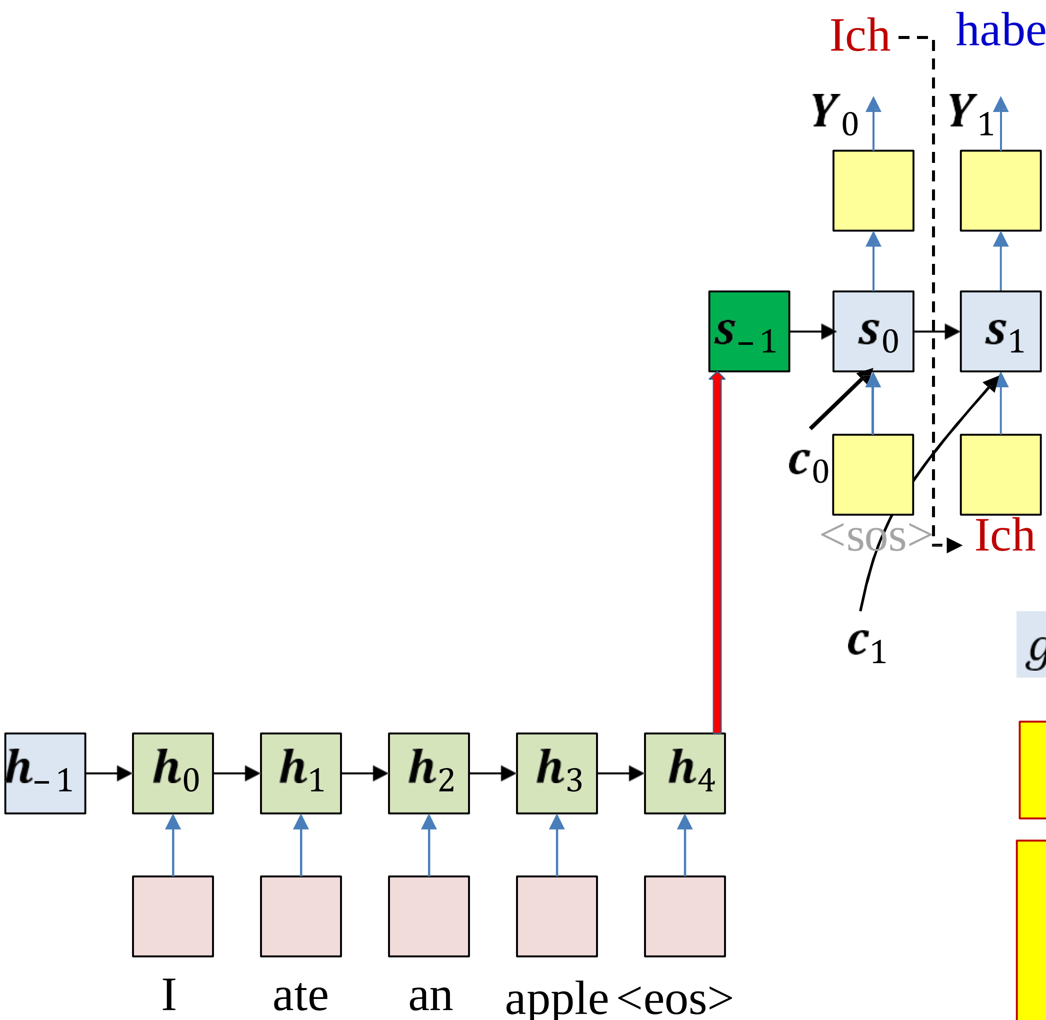
- Compute the weighted sum of hidden input values at t=1




- Compute the output at t=1
  - Will be a probability distribution over words




- Draw a word from the output distribution at t=1

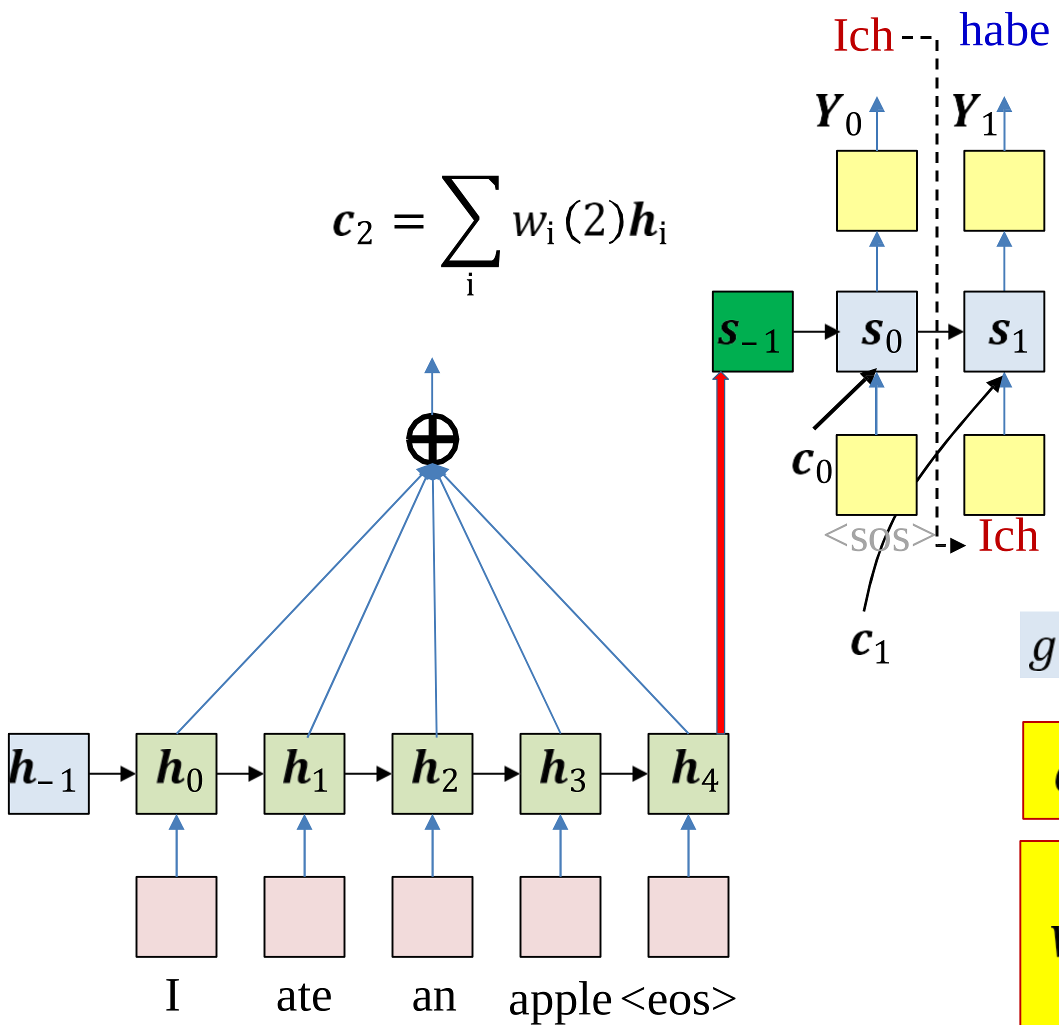



$$g(\boldsymbol{h}_i, \boldsymbol{s}_1) = \boldsymbol{h}_i^T \boldsymbol{W}_g \boldsymbol{s}_1$$

$$e_i(2) = g(\boldsymbol{h}_i, \boldsymbol{s}_1)$$

$$w_i(2) = \frac{\exp(e_i(2))}{\sum_j \exp(e_j(2))}$$

- Compute the weights for all instances for time = 2

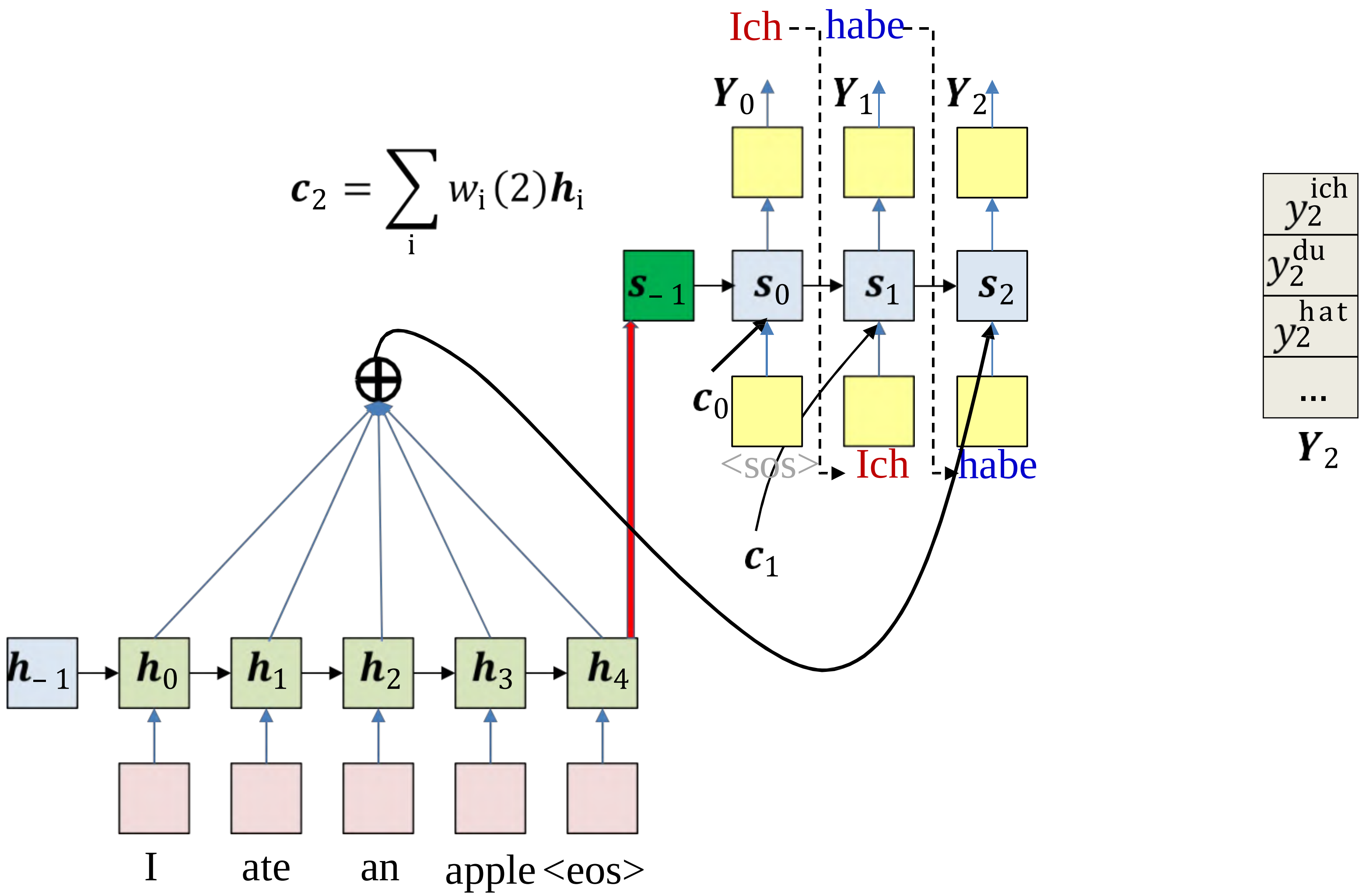



$$g(h_i, s_1) = h_i^T W_g s_1$$

$$e_i(2) = g(h_i, s_1)$$

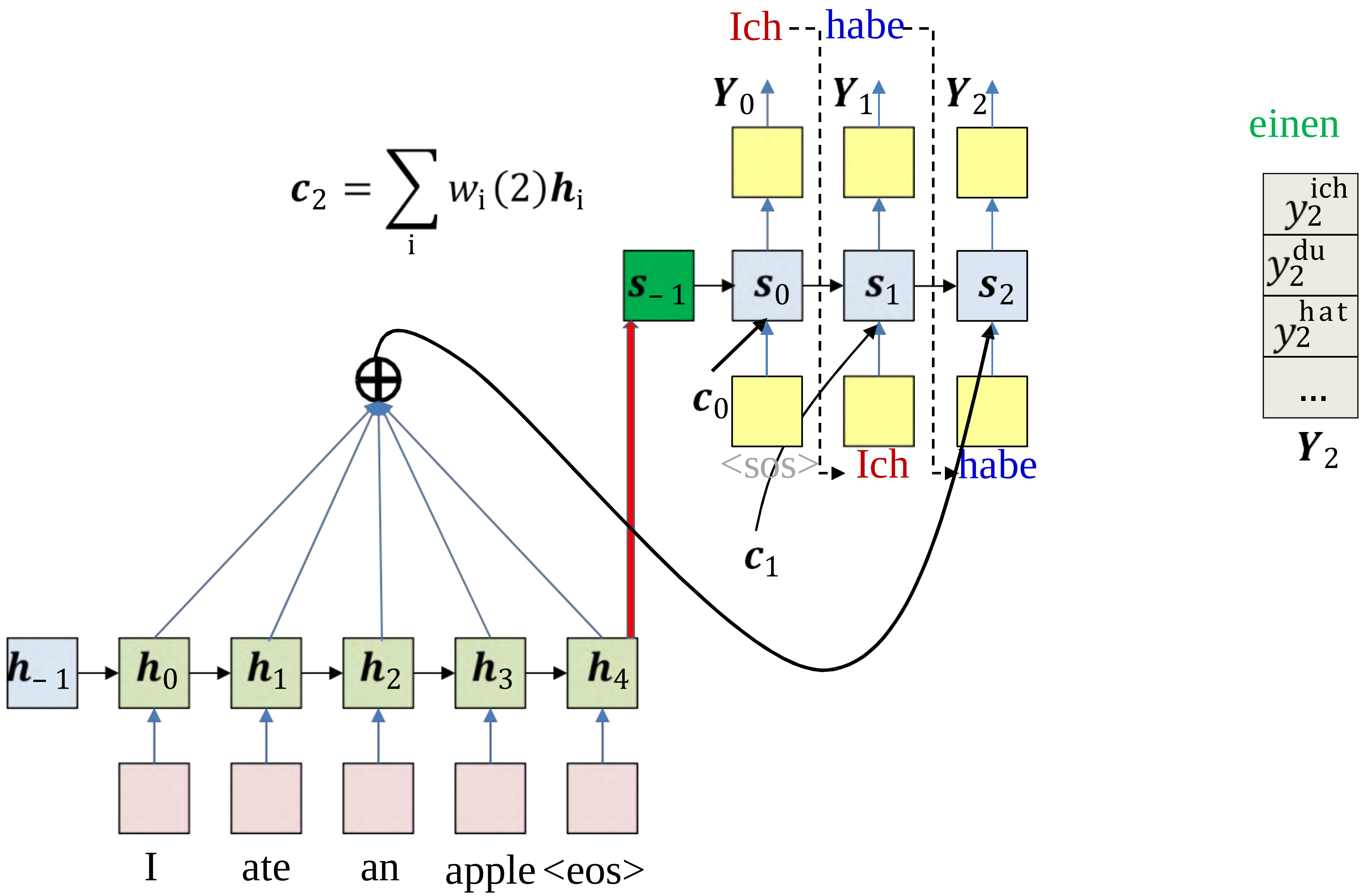
$$w_i(2) = \frac{\exp(e_i(2))}{\sum_j \exp(e_j(2))}$$

- Compute the weighted sum of hidden input values at t=2

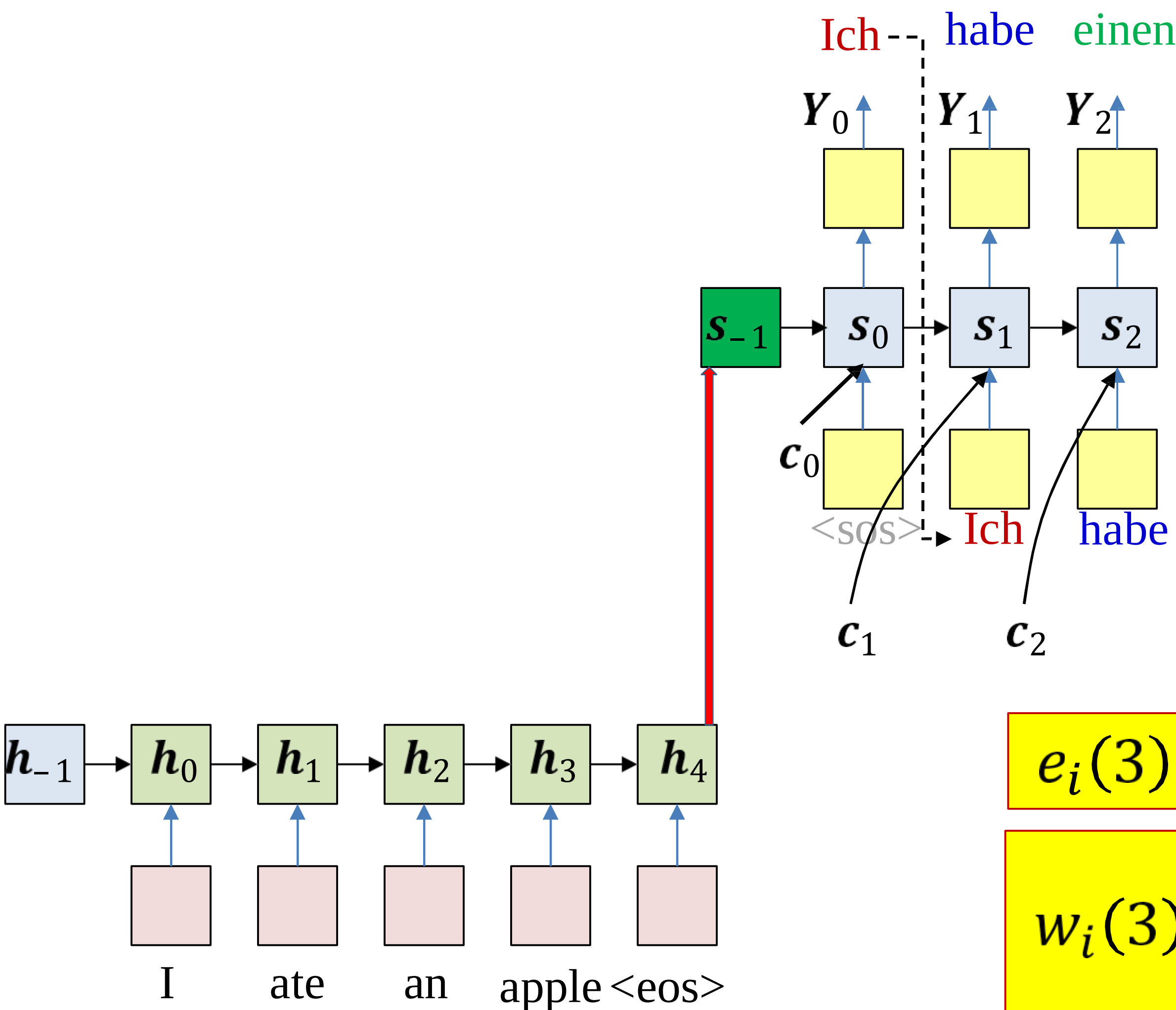



- Compute the output at t=2
  - Will be a probability distribution over words

$$c_2 = \sum_i w_i(2)\boldsymbol{h}_i$$



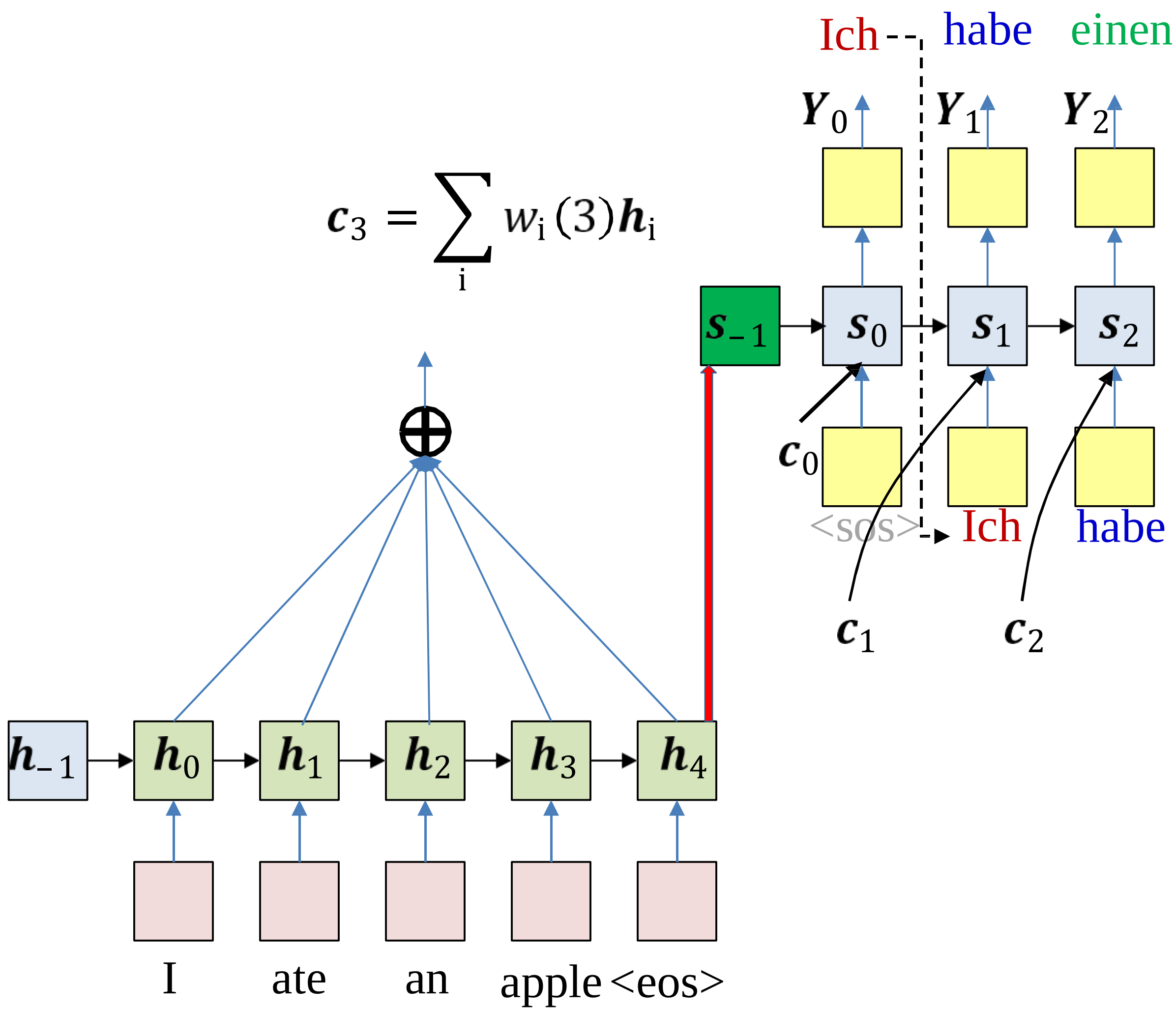

- Draw a word from the distribution



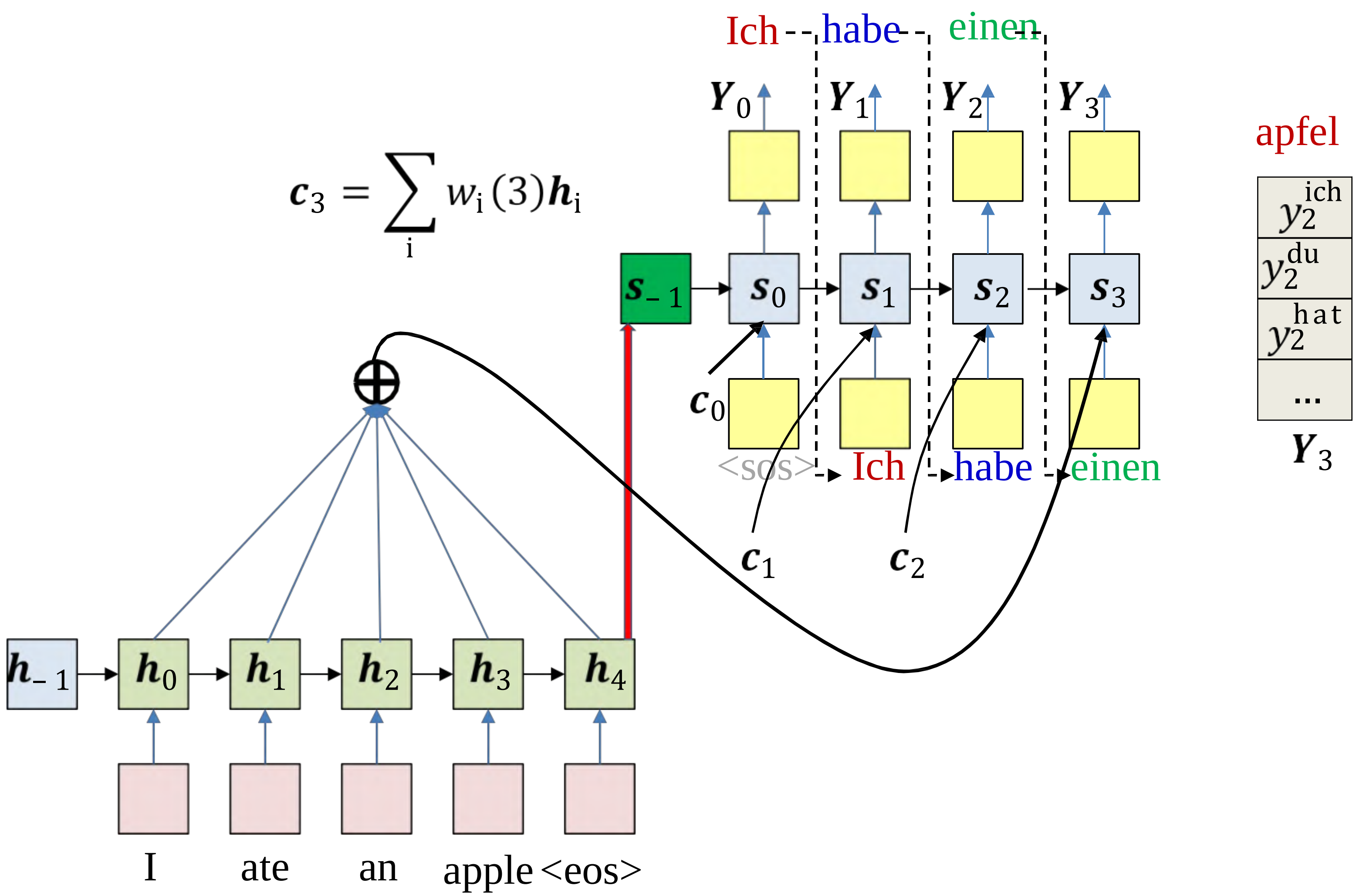

$$\boxed{e_i(3) = g(\boldsymbol{h}_i, \boldsymbol{s}_2)}$$

$$\boxed{w_i(3) = \frac{\exp(e_i(3))}{\sum_j \exp(e_j(3))}}$$

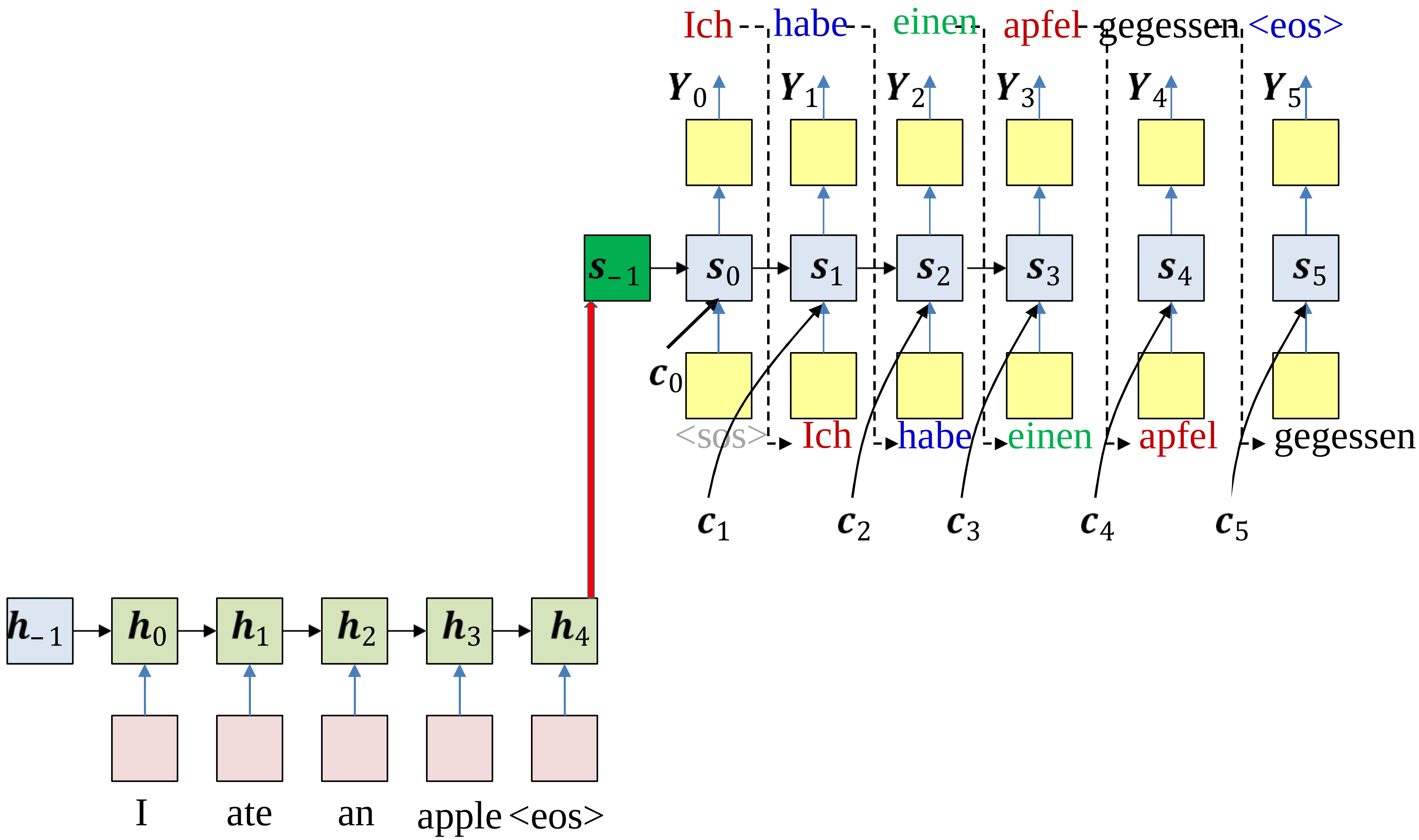
- Compute the weights for all instances for time = 3




- Compute the weighted sum of hidden input values at t=3




- Compute the output at t=3
  - Will be a probability distribution over words
  - Draw a word from the distribution



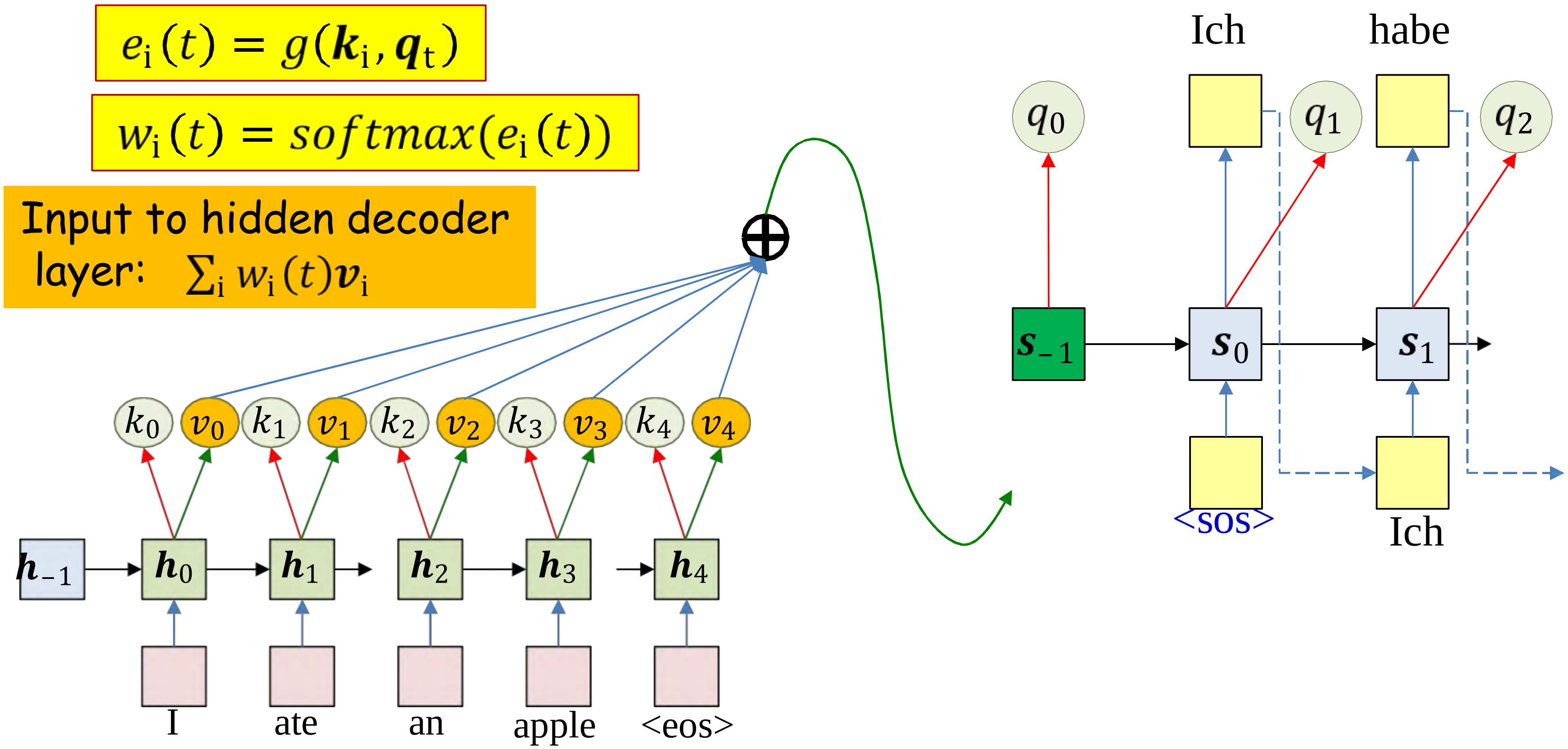

- Continue the process until an end-of-sequence symbol is produced

# Modification: Query key value

$$e_i(t) = g(\boldsymbol{k}_i, \boldsymbol{q}_t)$$

$$w_i(t) = softmax(e_i(t))$$

Input to hidden decoder layer: $\sum_i w_i(t)\boldsymbol{v}_i$




- Encoder outputs an explicit “key” and “value” at each input time
  – Key is used to evaluate the importance of the input at that time, for a given output
- Decoder outputs an explicit “query” at each output time
  – Query is used to evaluate which inputs to pay attention to
- The weight is a function of key and query
- The actual context is a weighted sum of value

# TRANSFORMERS

# CONTENT

- Introduction to Transformers
- Transformers Background
- The Attention Mechanism
- The Transformer Architecture
- GPT and BERT

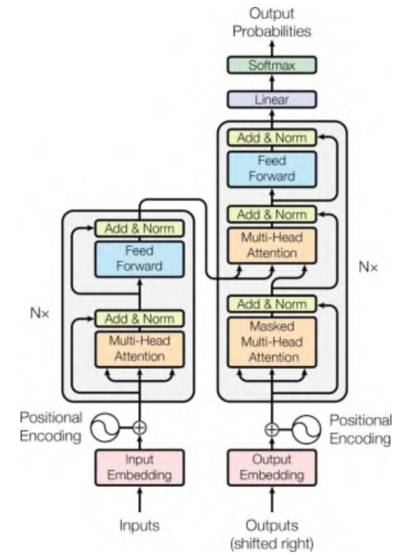
# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper :**
**Attention Is All You Need**

June 12, 2017

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

| Model | BLEU | | Training Cost (FLOPs) | |
|---|---|---|---|---|
| | EN-DE | EN-FR | EN-DE | EN-FR |
| ByteNet [18] | 23.75 | | | |
| Deep-Att + PosUnk [39] | | 39.2 | | $1.0 \cdot 10^{20}$ |
| GNMT + RL [38] | 24.6 | 39.92 | $2.3 \cdot 10^{19}$ | $1.4 \cdot 10^{20}$ |
| ConvS2S [9] | 25.16 | 40.46 | $9.6 \cdot 10^{18}$ | $1.5 \cdot 10^{20}$ |
| MoE [32] | 26.03 | 40.56 | $2.0 \cdot 10^{19}$ | $1.2 \cdot 10^{20}$ |
| Deep-Att + PosUnk Ensemble [39] | | 40.4 | | $8.0 \cdot 10^{20}$ |
| GNMT + RL Ensemble [38] | 26.30 | 41.16 | $1.8 \cdot 10^{20}$ | $1.1 \cdot 10^{21}$ |
| ConvS2S Ensemble [9] | 26.36 | **41.29** | $7.7 \cdot 10^{19}$ | $1.2 \cdot 10^{21}$ |
| Transformer (base model) | 27.3 | 38.1 | $\mathbf{3.3 \cdot 10^{18}}$ | |
| Transformer (big) | **28.4** | **41.8** | $2.3 \cdot 10^{19}$ | |



# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper :**
**Attention Is All You Need**

June 12, 2017

**First GPT paper by OpenAI:**
**Improving Language Understanding**
**by Generative Pre-Training**

June 11, 2018

# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper :**
**Attention Is All You Need**

June 12, 2017

**First GPT paper by OpenAI:**
**Improving Language Understanding**
**by Generative Pre-Training**

June 11, 2018

Table 5: Analysis of various model ablations on different tasks. Avg. score is a unweighted average of all the results. (*mc*= Mathews correlation, *acc*=Accuracy, *pc*=Pearson correlation)

| Method | Avg. Score | CoLA (mc) | SST2 (acc) | MRPC (F1) | STSB (pc) | QQP (F1) | MNLI (acc) | QNLI (acc) | RTE (acc) |
|---|---|---|---|---|---|---|---|---|---|
| Transformer w/ aux LM (full) | 74.7 | 45.4 | 91.3 | 82.3 | 82.0 | **70.3** | **81.8** | **88.1** | **56.0** |
| Transformer w/o pre-training | 59.9 | 18.9 | 84.0 | 79.4 | 30.9 | 65.5 | 75.7 | 71.2 | 53.8 |
| Transformer w/o aux LM | **75.0** | **47.9** | **92.0** | **84.9** | **83.2** | 69.8 | 81.1 | 86.9 | 54.4 |
| LSTM w/ aux LM | 69.1 | 30.3 | 90.5 | 83.2 | 71.8 | 68.1 | 73.7 | 81.1 | 54.6 |

# INTRODUCTION TO TRANSFORMERS

| Original Transformers Paper : Attention Is All You Need | First GPT paper by OpenAI: Improving Language Understanding by Generative Pre-Training | First BERT paper by Google: Pre-Training Deep Bidirectional Transformers for Language Understanding |
|---|---|---|
| June 12, 2017 | June 11, 2018 | Oct 11, 2018 |

# INTRODUCTION TO TRANSFORMERS

**Original Transformers Paper : Attention Is All You Need**

June 12, 2017

**First GPT paper by OpenAI: Improving Language Understanding by Generative Pre-Training**

June 11, 2018

**First BERT paper by Google: Pre-Training Deep Bidirectional Transformers for Language Understanding**

Oct 11, 2018

| System | MNLI-(m/mm) | QQP | QNLI | SST-2 | CoLA | STS-B | MRPC | RTE | **Average** |
|---|---|---|---|---|---|---|---|---|---|
| | 392k | 363k | 108k | 67k | 8.5k | 5.7k | 3.5k | 2.5k | - |
| Pre-OpenAI SOTA | 80.6/80.1 | 66.1 | 82.3 | 93.2 | 35.0 | 81.0 | 86.0 | 61.7 | 74.0 |
| BiLSTM+ELMo+Attn | 76.4/76.1 | 64.8 | 79.8 | 90.4 | 36.0 | 73.3 | 84.9 | 56.8 | 71.0 |
| OpenAI GPT | 82.1/81.4 | 70.3 | 87.4 | 91.3 | 45.4 | 80.0 | 82.3 | 56.0 | 75.1 |
| $\text{BERT}_{\text{BASE}}$ | 84.6/83.4 | 71.2 | 90.5 | 93.5 | 52.1 | 85.8 | 88.9 | 66.4 | 79.6 |
| $\text{BERT}_{\text{LARGE}}$ | **86.7/85.9** | **72.1** | **92.7** | **94.9** | **60.5** | **86.5** | **89.3** | **70.1** | **82.1** |

Table 1: GLUE Test results, scored by the evaluation server (https://gluebenchmark.com/leaderboard). The number below each task denotes the number of training examples. The "Average" column is slightly different than the official GLUE score, since we exclude the problematic WNLI set.[8] BERT and OpenAI GPT are single-model, single task. F1 scores are reported for QQP and MRPC, Spearman correlations are reported for STS-B, and accuracy scores are reported for the other tasks. We exclude entries that use BERT as one of their components.

# TRANSFORMERS : BACKGROUND

# TRANSFORMERS : BACKGROUND

- Word Embeddings
- Encoder Decoder Models
- Attention

# TRANSFORMERS : BACKGROUND

- **Word Embeddings**
- Encoder Decoder Models
- Attention

# TRANSFORMERS : BACKGROUND

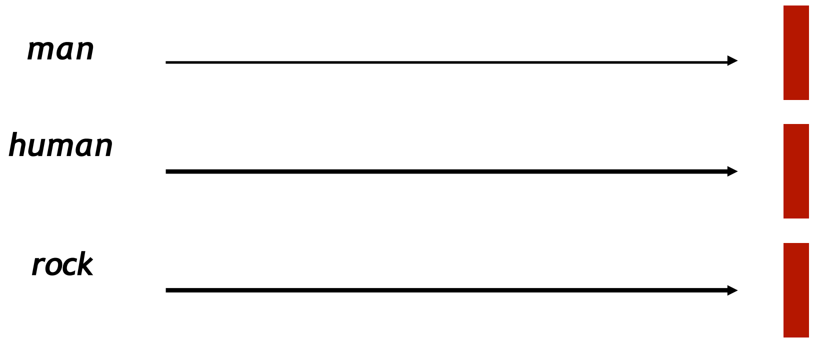
- Word Embeddings

Representing words in the form of vectors
that incorporate information such as word meaning and
context.

# TRANSFORMERS : BACKGROUND

- Word Embeddings

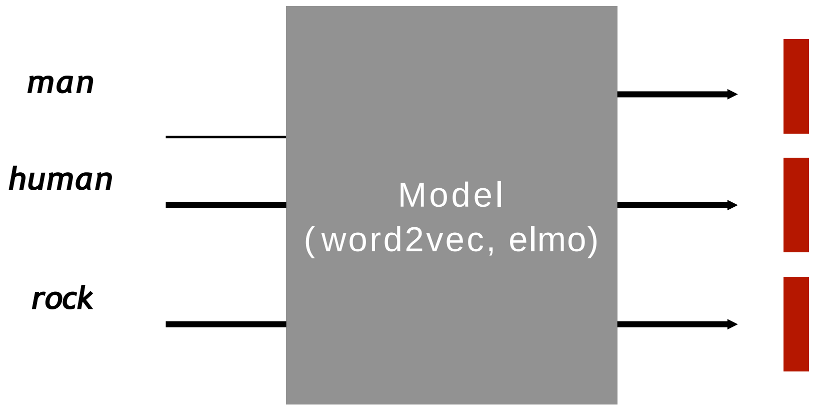
Representing words in the form of vectors
that incorporate information such as word meaning and
context.



# TRANSFORMERS : BACKGROUND

- Word Embeddings

Representing words in the form of vectors
that incorporate information such as word meaning and
context.



# TRANSFORMERS : BACKGROUND

- Word Embeddings
- **Encoder Decoder Models**
- Attention

# TRANSFORMERS : BACKGROUND

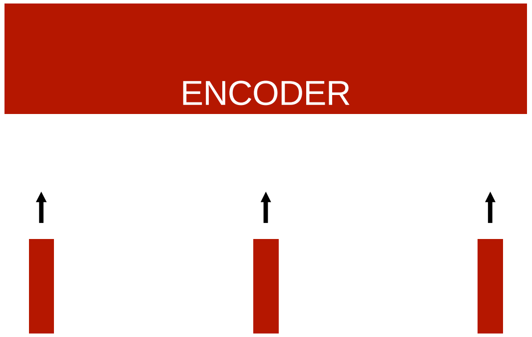
- Encoder Decoder Models

Formalizes tasks into two steps -
maps the input into an encoded representation used by the decoder to generate output

# TRANSFORMERS : BACKGROUND

- Encoder Decoder Models

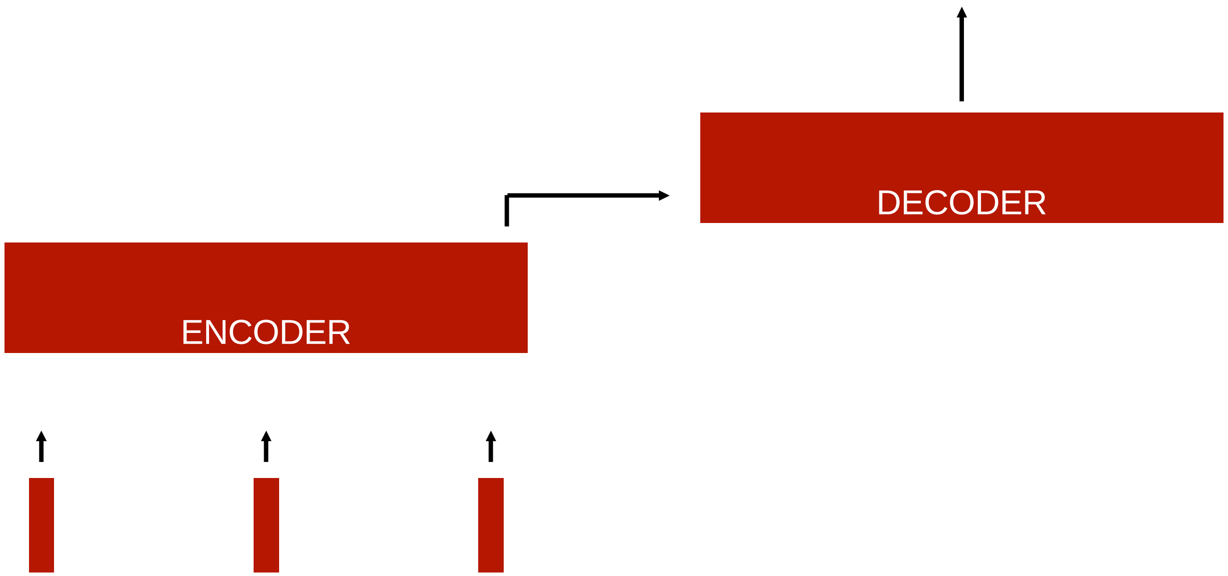
Formalizes tasks into two steps -
maps the input into an encoded representation used by the decoder to generate output



# TRANSFORMERS : BACKGROUND

- Encoder Decoder Models

Formalizes tasks into two steps -
maps the input into an encoded representation used by the decoder to generate output

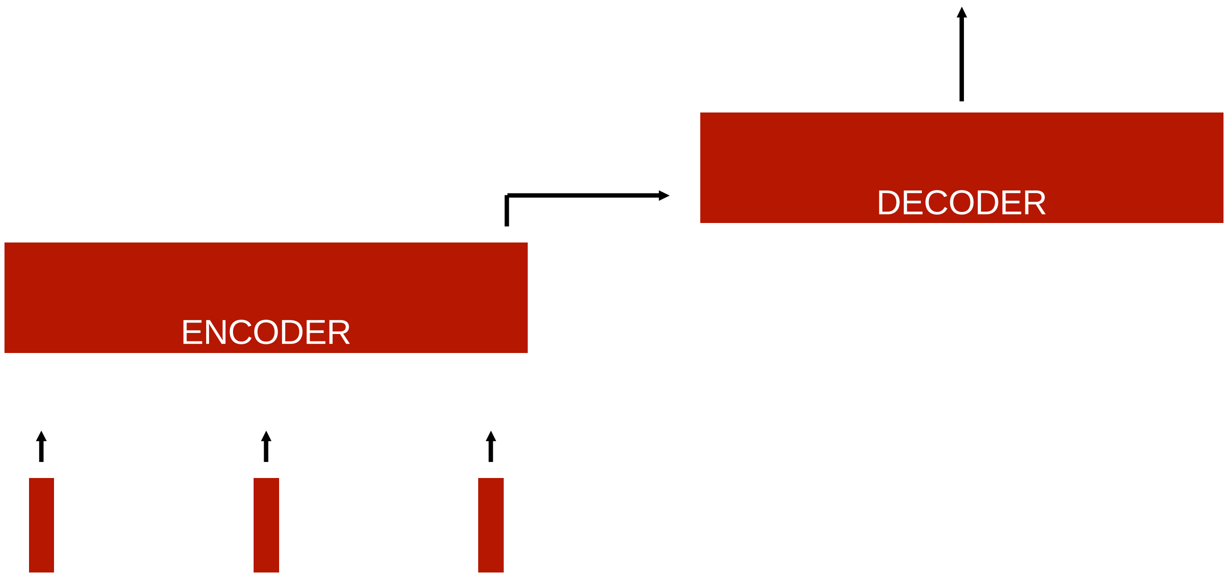


# TRANSFORMERS : BACKGROUND

- Encoder Decoder Models

Formalizes tasks into two steps -
maps the input into an encoded representation used by the decoder to generate output

**See any problems here?**



# TRANSFORMERS : BACKGROUND

- Word Embeddings
- Encoder Decoder Models
- **Attention**

# TRANSFORMERS : BACKGROUND

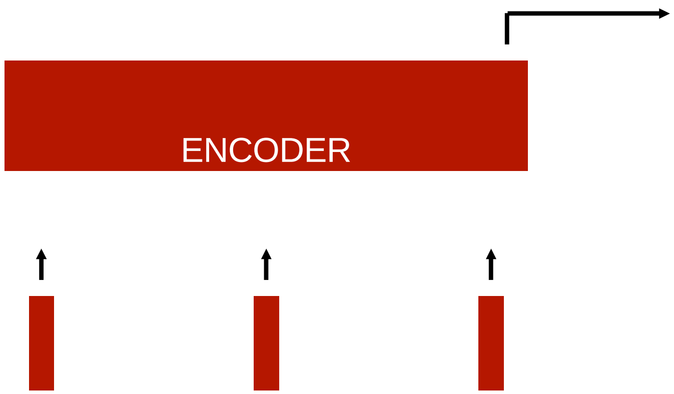
- Attention Mechanism

  A method of dynamically giving weight (attention) to different parts of the input to make a decision.

# TRANSFORMERS : BACKGROUND

- Attention Mechanism

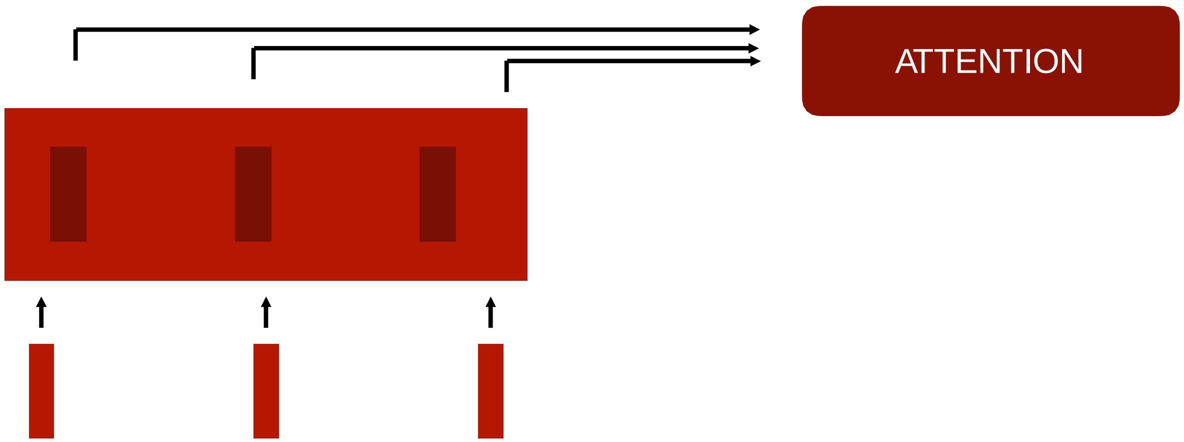
  A method of dynamically giving weight (attention) to different parts of the input to make a decision.



# TRANSFORMERS : BACKGROUND

- Attention Mechanism

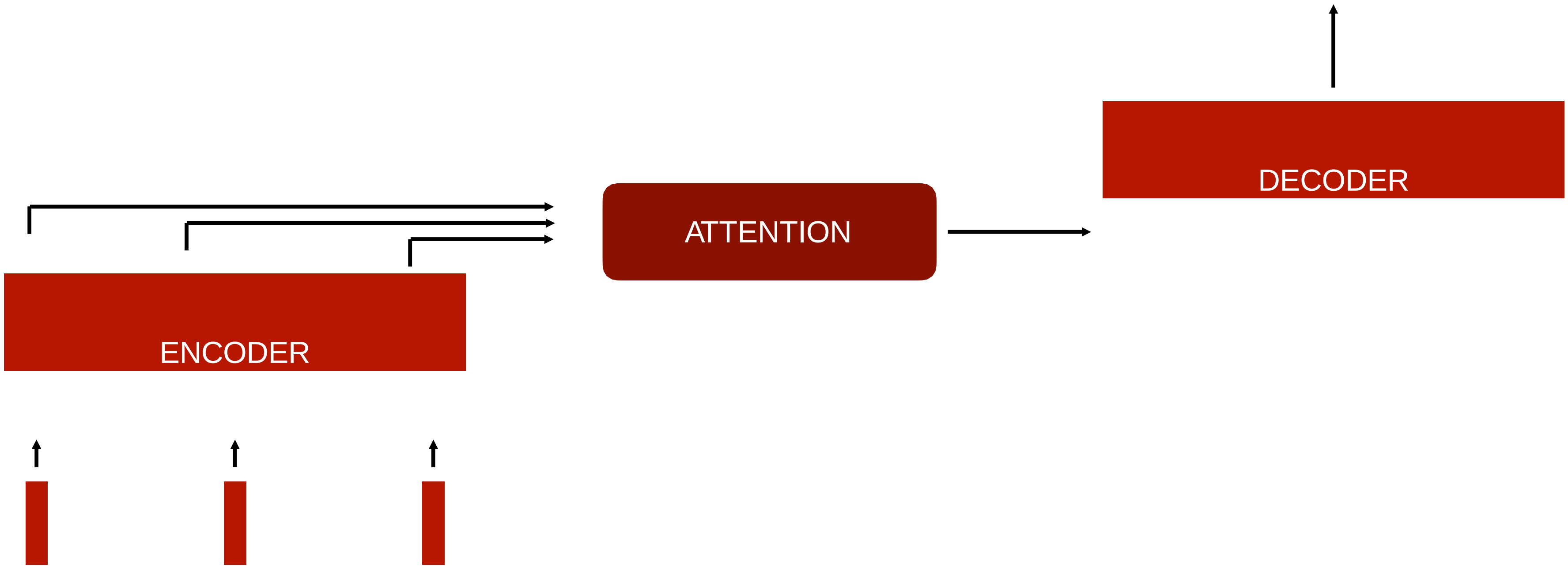
A method of dynamically giving weight (attention) to different parts of the input to make a decision.



# TRANSFORMERS : BACKGROUND

- Attention Mechanism

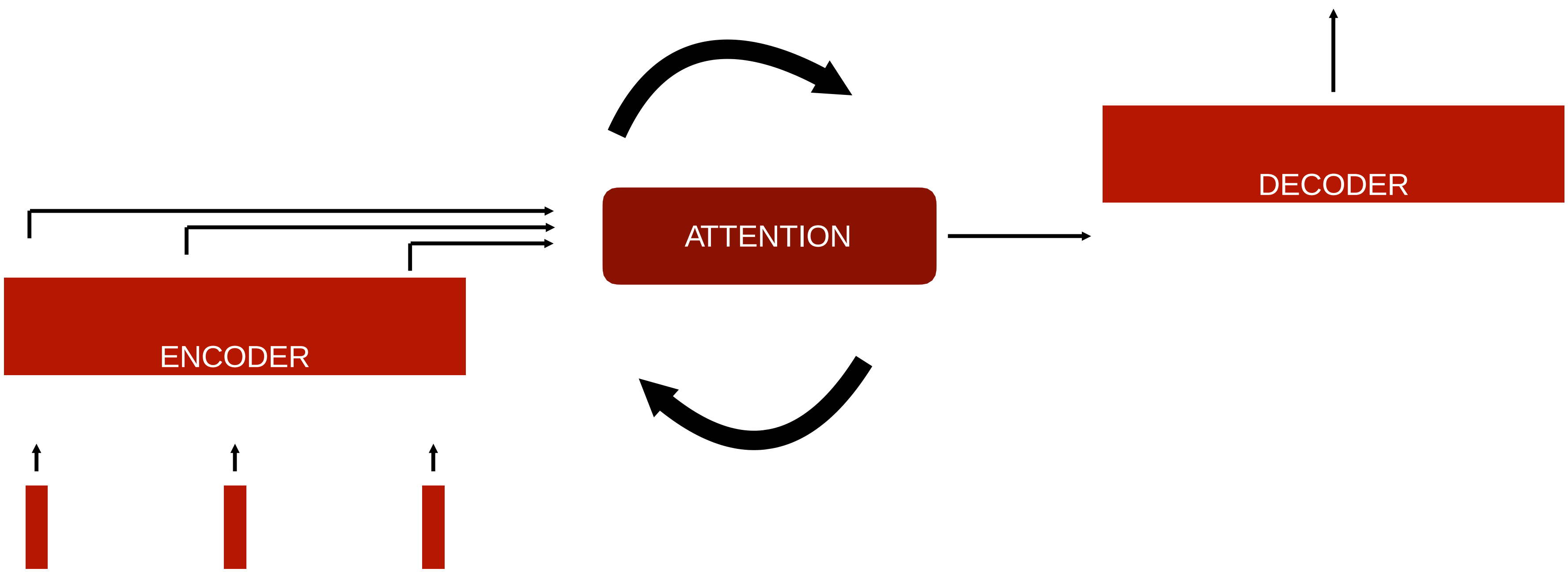
A method of dynamically giving weight (attention) to different parts of the input to make a decision.



# TRANSFORMERS : BACKGROUND

- Attention Mechanism

A method of dynamically giving weight (attention) to different parts of the input to make a decision.



# TRANSFORMERS :
# THE ATTENTION MECHANISM

# THE ATTENTION MECHANISM

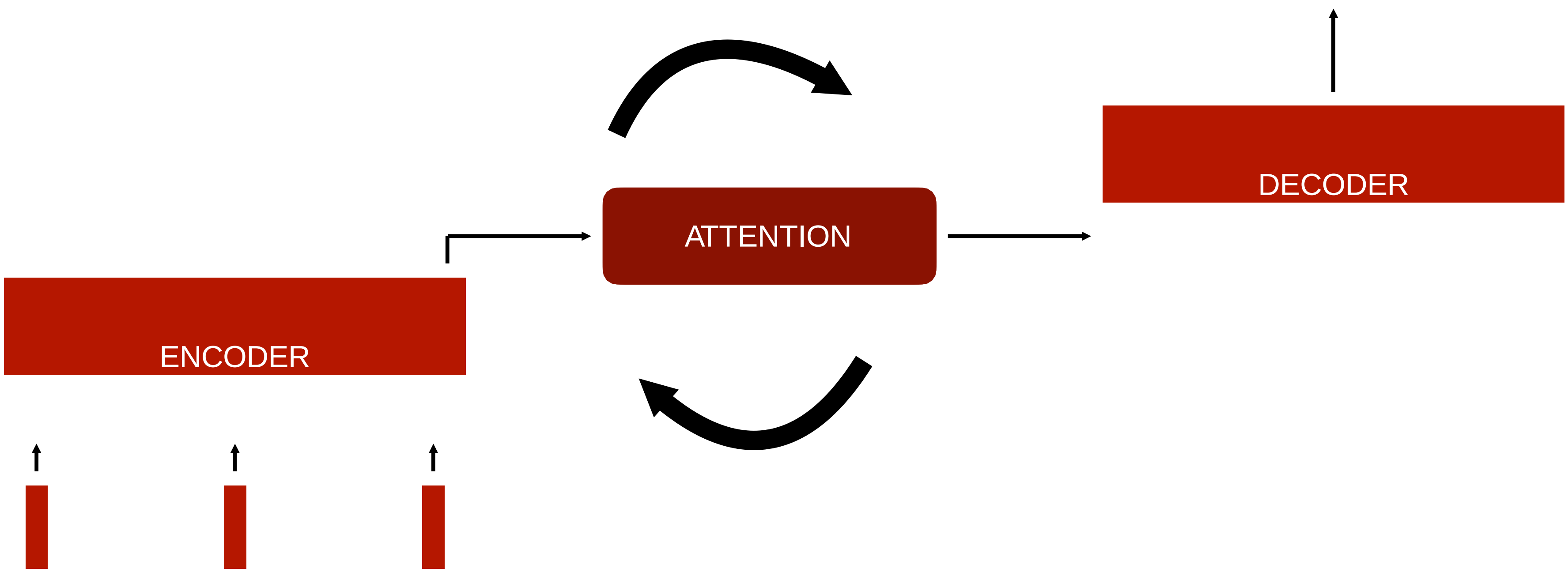
- Attention
- Self-Attention
- Multi-Head Attention

# THE ATTENTION MECHANISM

- **Attention**
- Self-Attention
- Multi-Head Attention

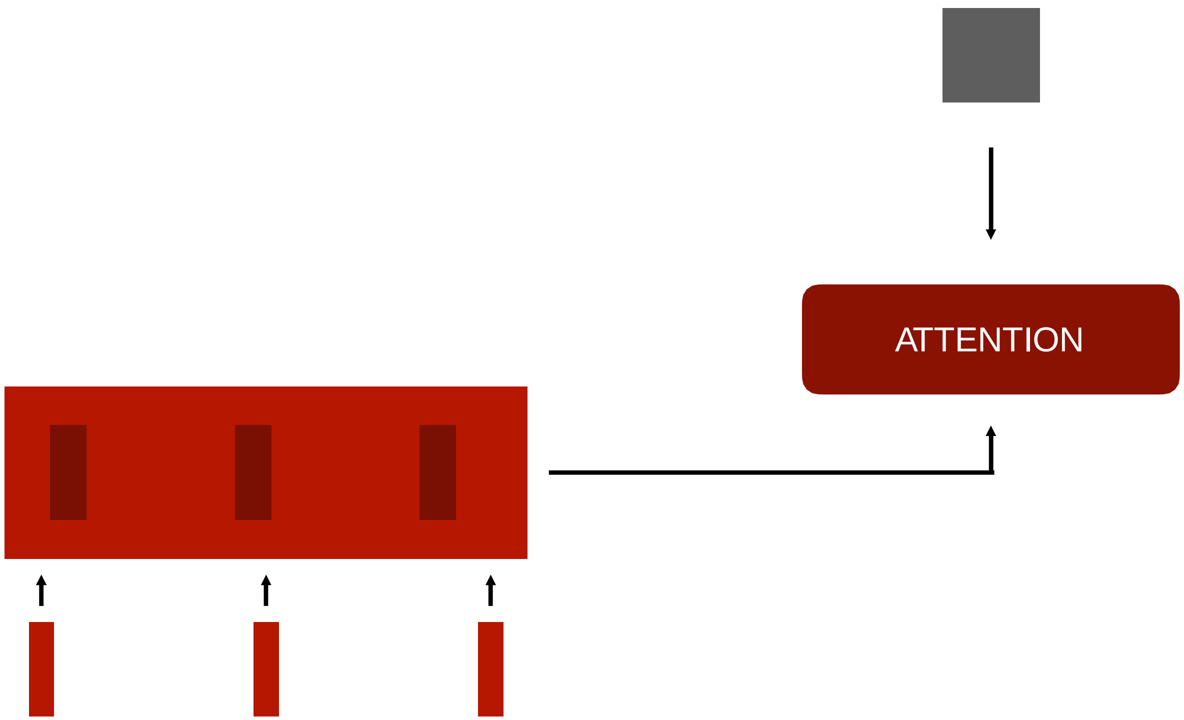
# THE ATTENTION MECHANISM

- Attention



# THE ATTENTION MECHANISM

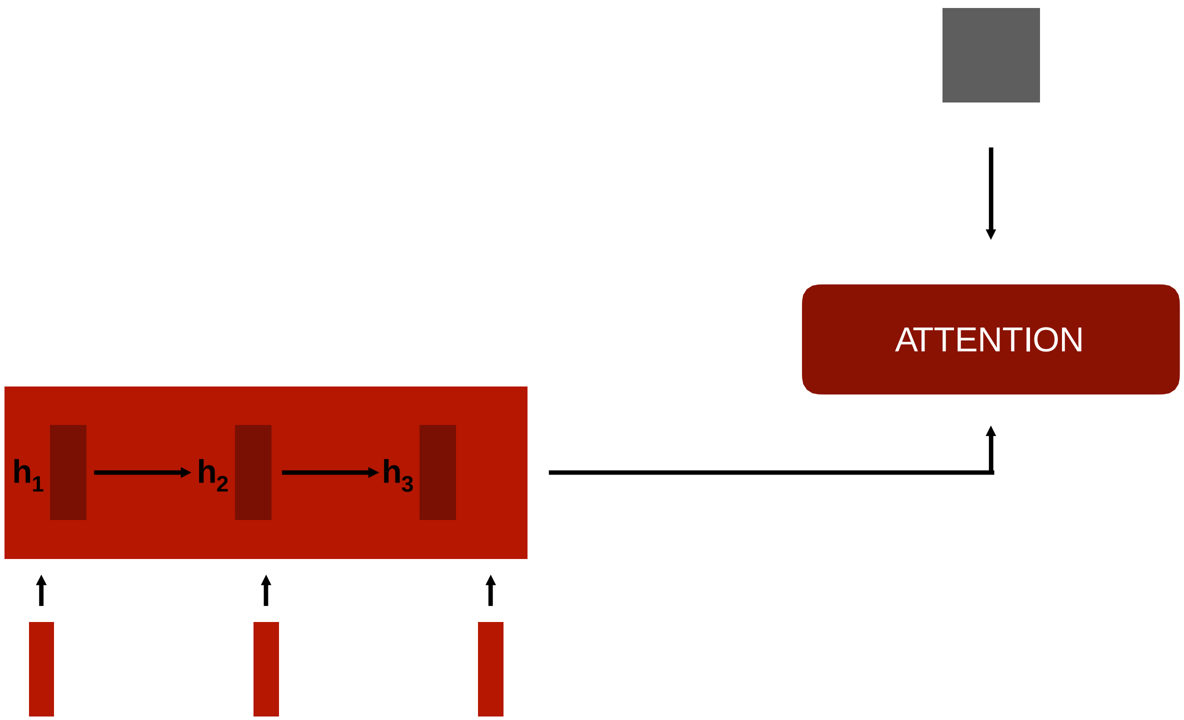
- Attention



# THE ATTENTION MECHANISM

- Attention

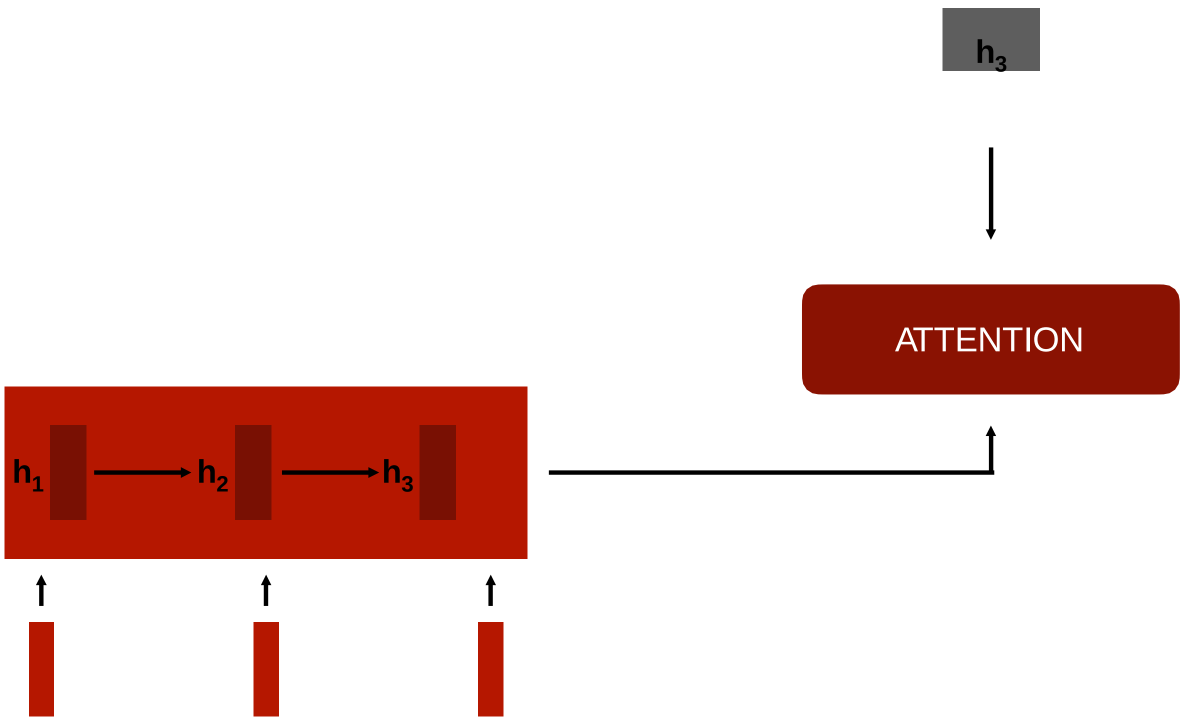
What is the other input to the attention module at the beginning of generation?



# THE ATTENTION MECHANISM

- Attention

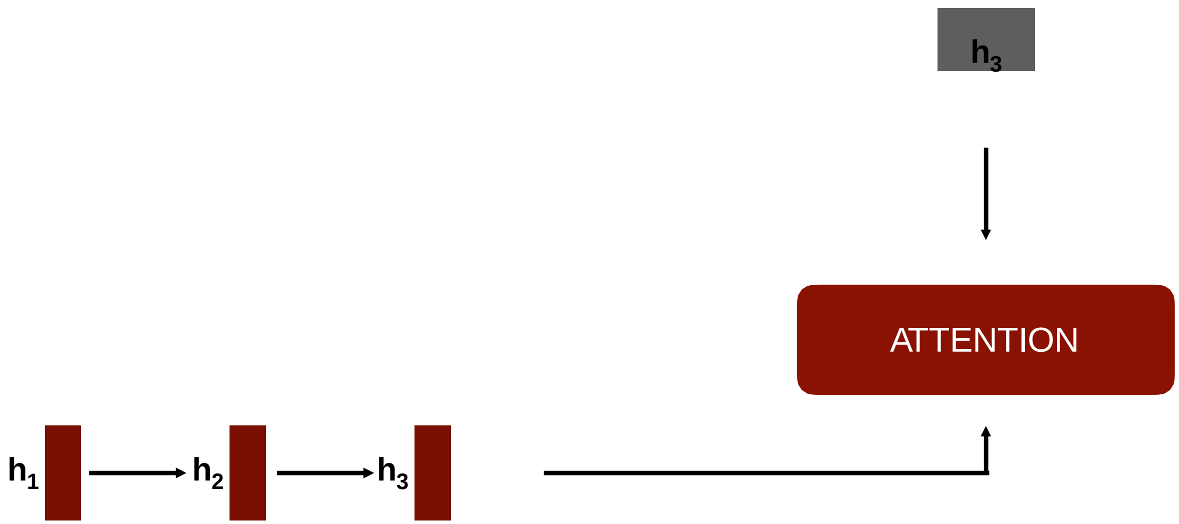
**At t = 0,**
**First time step of generation**



# THE ATTENTION MECHANISM

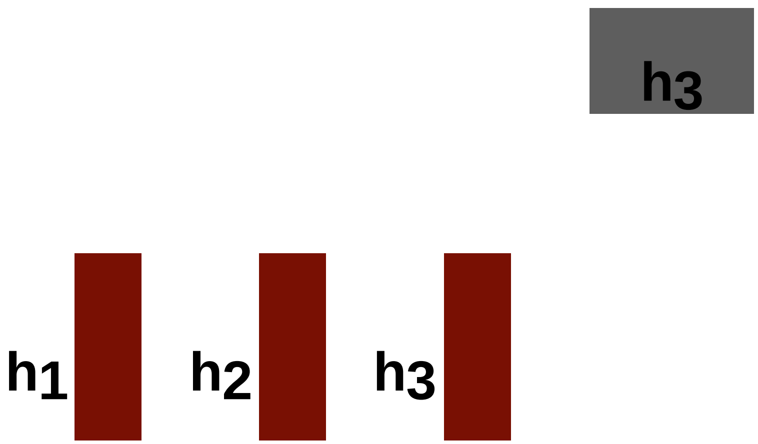
- Attention

At t = 0,
First time step of generation



# THE ATTENTION MECHANISM

- **Attention : Set Up**

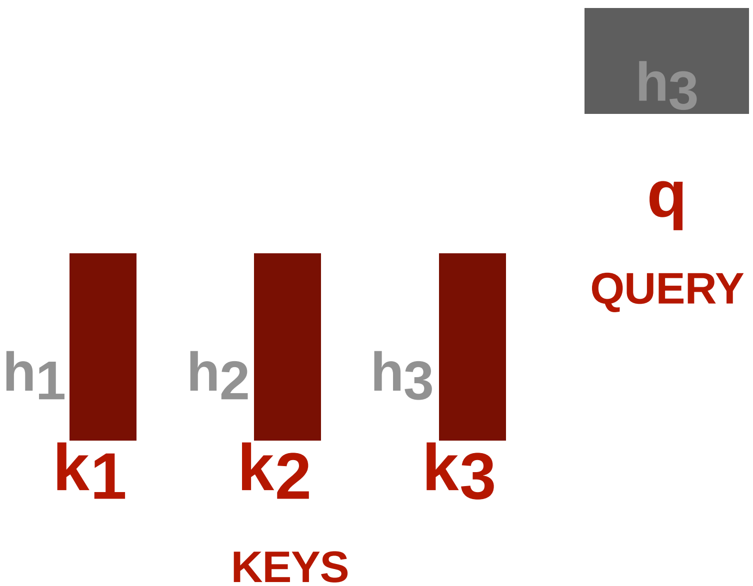



**At t = 0,**
**First time step of generation**

# THE ATTENTION MECHANISM

- Attention : Set Up

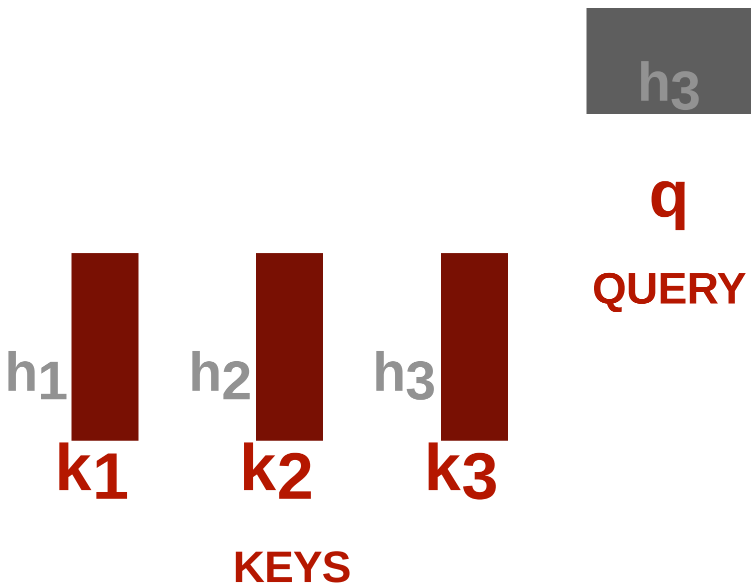
At t = 0,
First time step of generation



# THE ATTENTION MECHANISM

- Calculating Attention

At t = 0,
First time step of generation

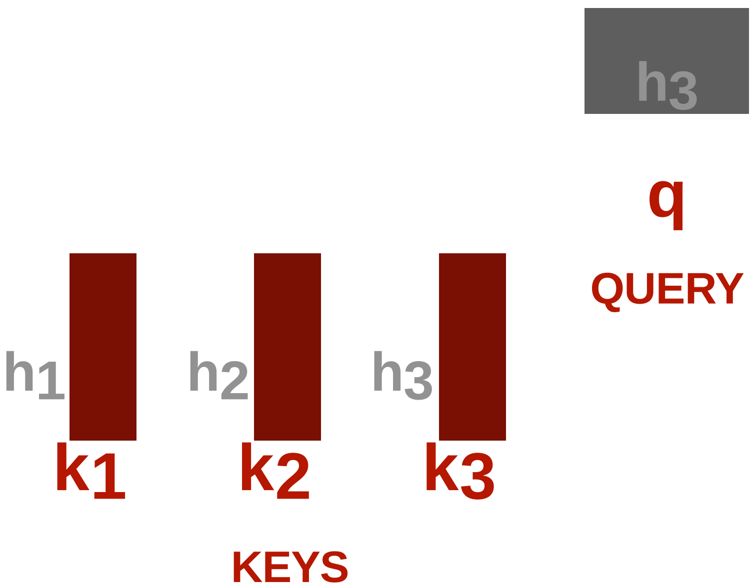


# THE ATTENTION MECHANISM

- **Calculating Attention**

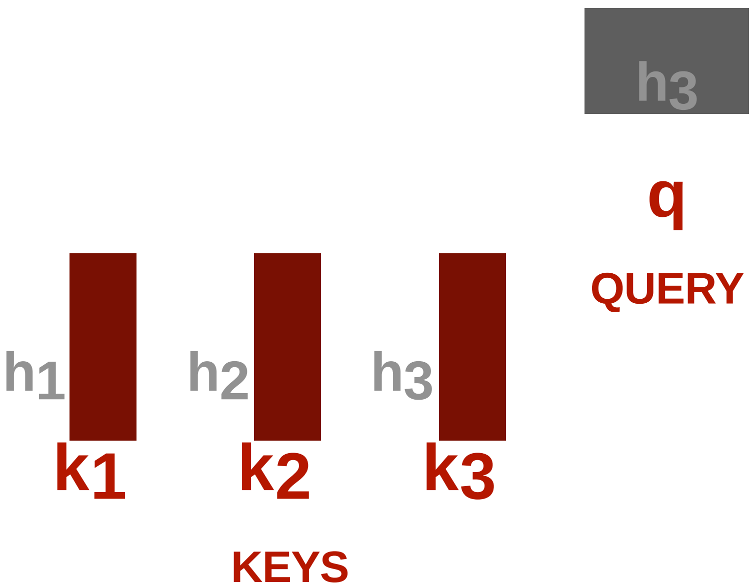
STEP -1 : CALCULATE A SIMILARITY MEASURE BETWEEN QUERY AND EACH KEY

$$e_i(t) = g(\boldsymbol{q}(t), \boldsymbol{k_i})$$



# THE ATTENTION MECHANISM

- **Calculating Attention**




STEP -1 : CALCULATE A SIMILARITY MEASURE BETWEEN QUERY AND EACH KEY

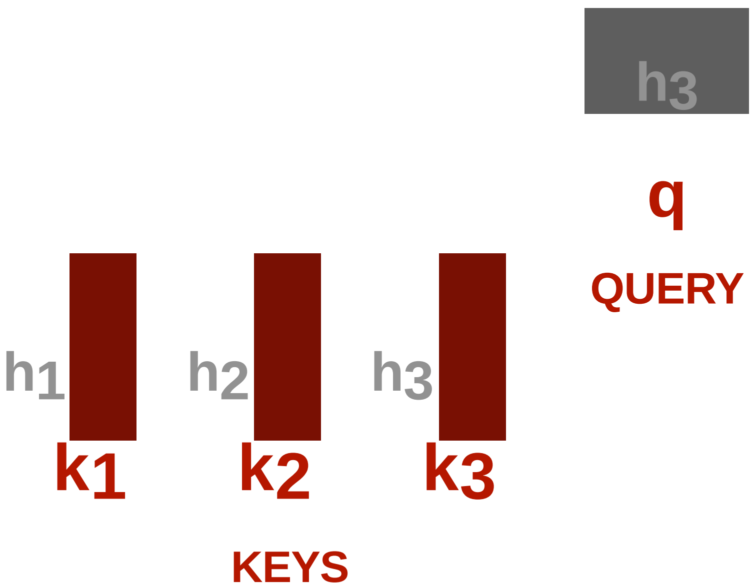
$$e_i(t) = g(\boldsymbol{q}(t), \boldsymbol{k}_i)$$

**Examples:**

$$g(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^\top \boldsymbol{k}_i$$

# THE ATTENTION MECHANISM

- **Calculating Attention**




STEP -1 : CALCULATE A SIMILARITY MEASURE BETWEEN QUERY AND EACH KEY

$$e_i(t) = g(\boldsymbol{q}(t), \boldsymbol{k}_i)$$

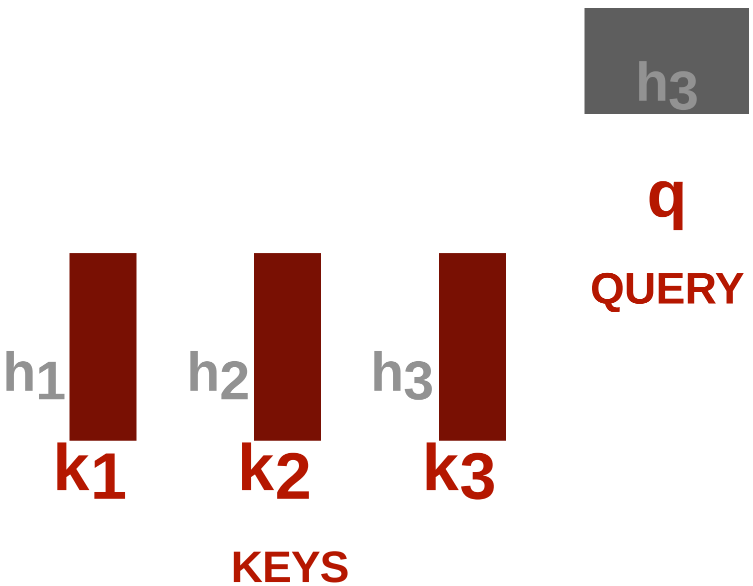
**Examples:**

$$g(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^{\mathrm{T}}\boldsymbol{k}_i$$

$$g(\boldsymbol{q}, \boldsymbol{k}_i) = \boldsymbol{q}^{T}\,\mathbf{W}\,\boldsymbol{k}_i$$

# THE ATTENTION MECHANISM

- **Calculating Attention**




STEP -1 : CALCULATE A SIMILARITY MEASURE BETWEEN QUERY AND EACH KEY

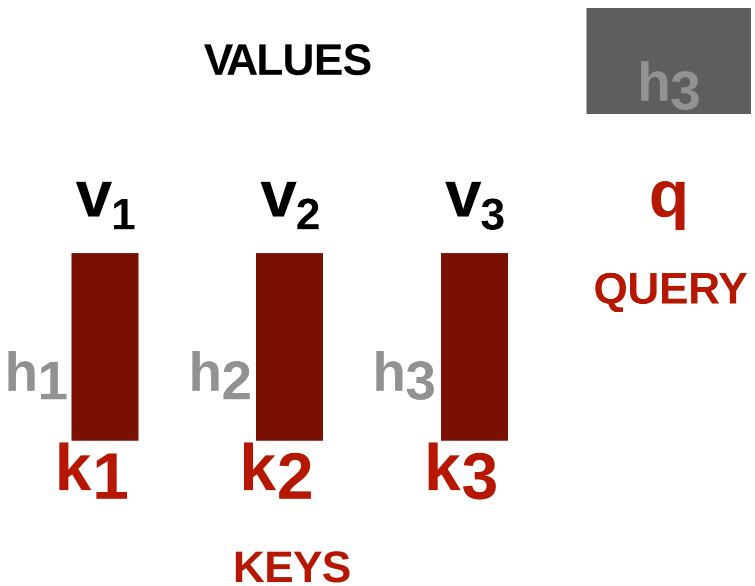
$$e_i(t) = g(\boldsymbol{q}(t), \boldsymbol{k_i})$$

STEP -2 : TAKE SOFTMAX OVER RAW WEIGHTS

$$w_i(t) = \frac{\exp(\, e_i(\boldsymbol{q}(t), \boldsymbol{k_i})\,)}{\sum_j \exp(\, e_j(\boldsymbol{q}(t), \boldsymbol{k_j})\,)}$$

# THE ATTENTION MECHANISM

- **Calculating Attention**




STEP -1 : CALCULATE A SIMILARITY MEASURE BETWEEN QUERY AND EACH KEY

$$e_i(t) = g(\boldsymbol{q}(t), \boldsymbol{k}_i)$$

STEP -2 : TAKE SOFTMAX OVER RAW WEIGHTS

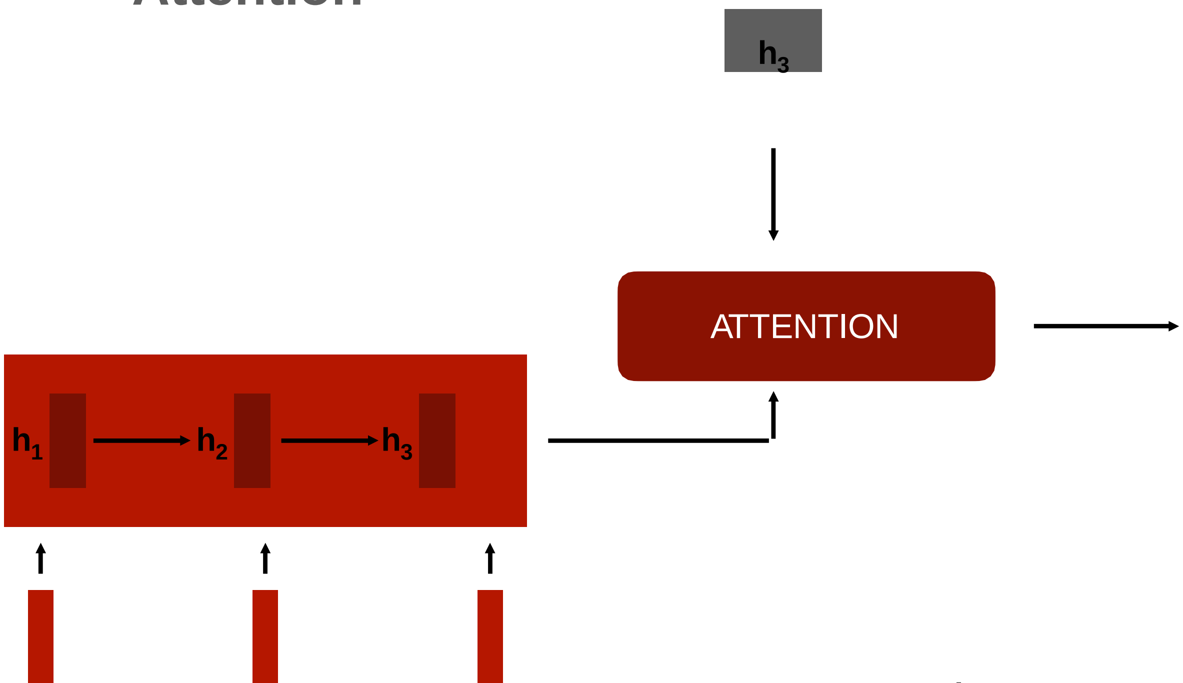
$$w_i(t) = \frac{\exp(\, e_i(\boldsymbol{q}(t), \boldsymbol{k}_i)\,)}{\sum_j \exp(\, e_j(\boldsymbol{q}(t), \boldsymbol{k}_j)\,)}$$

STEP -3 : TAKE A LINEAR COMBINATION

$$\boldsymbol{O}(\boldsymbol{k}, \boldsymbol{q}(t), \boldsymbol{v}) = \sum_i w_i(t)\, \boldsymbol{v}_i$$

# THE ATTENTION MECHANISM

- Attention




$$O(\boldsymbol{k},\boldsymbol{q}(t),\boldsymbol{v}) = \sum_i w_i(t)\ \boldsymbol{v}_i$$

**NOTE : Query, Key, Values are generalizations of the input to the attention mechanism.**

# THE ATTENTION MECHANISM

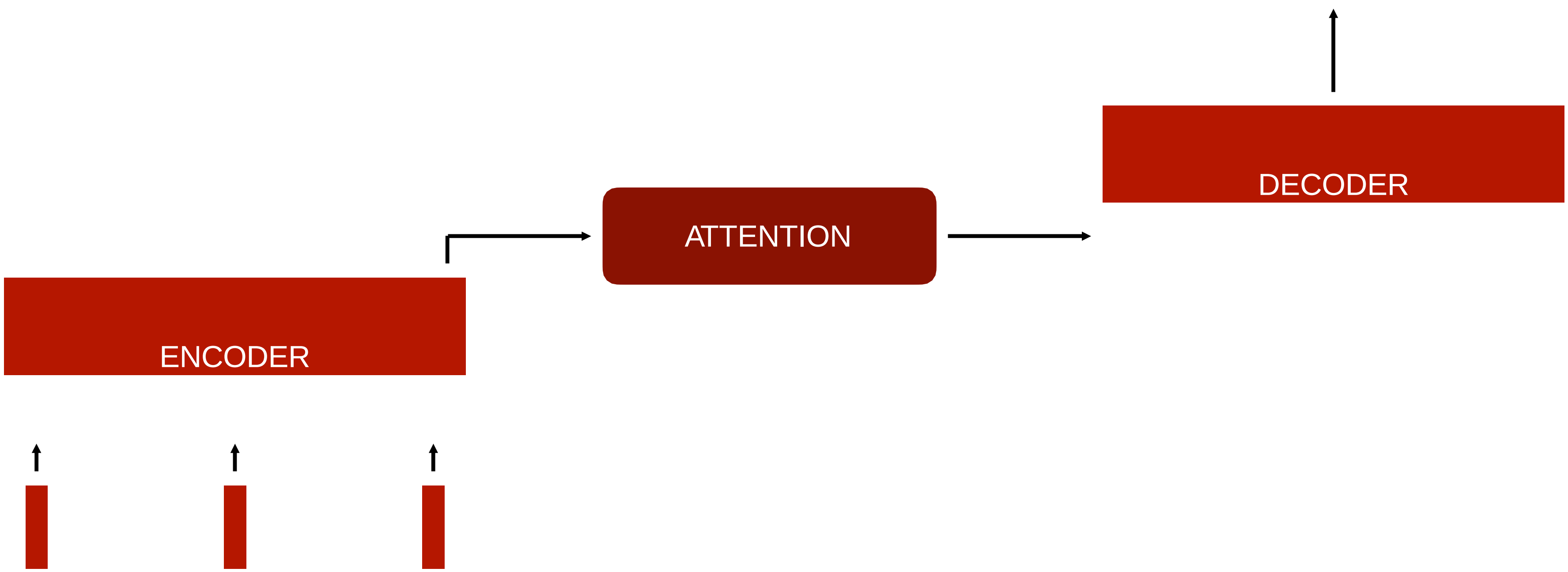
- Attention
- **Self-Attention**
- Multi-Head Attention

# THE ATTENTION MECHANISM

- **Self-Attention**

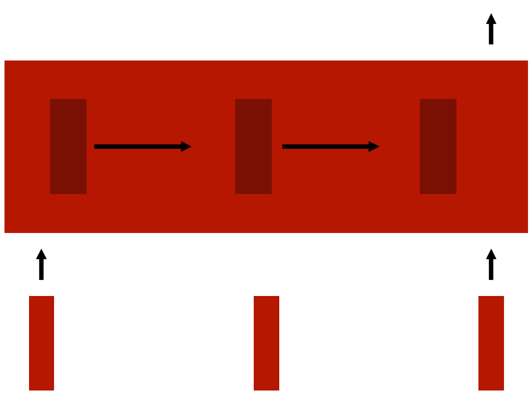
# THE ATTENTION MECHANISM

- Self-Attention



# THE ATTENTION MECHANISM

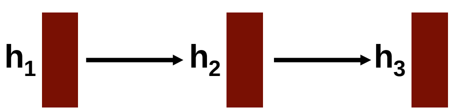
- **Self-Attention**



# THE ATTENTION MECHANISM

- Self-Attention

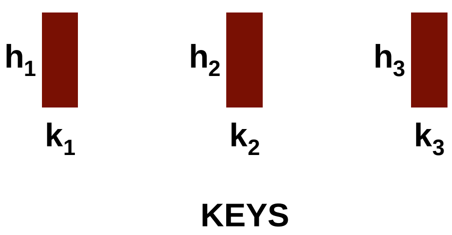
**Find the attention of each hidden state with every other hidden state in the sequence**



# THE ATTENTION MECHANISM

- Self-Attention

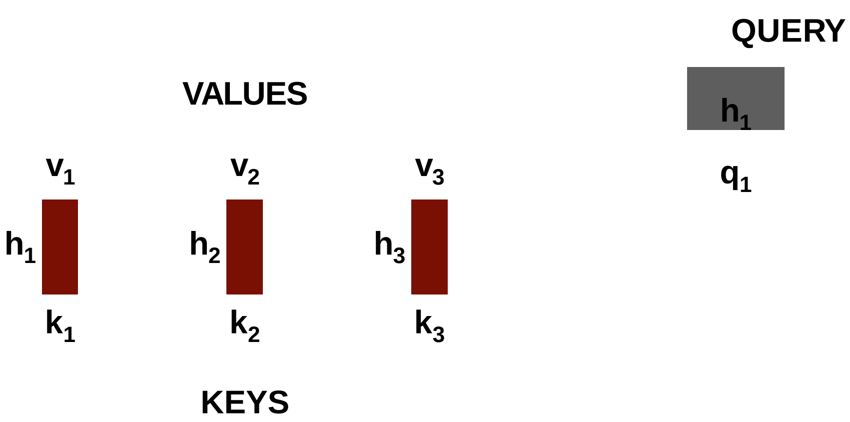
**Find the attention of each hidden state with every other hidden state in the sequence**



# THE ATTENTION MECHANISM

- **Self-Attention**

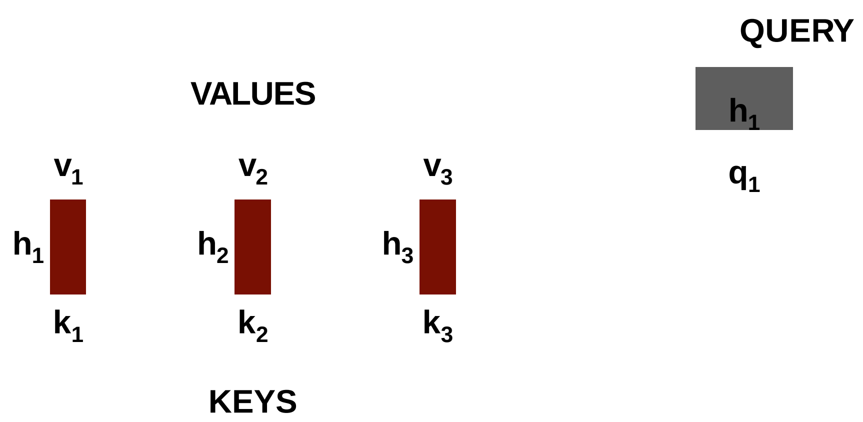
**Find the attention of each hidden state with every other hidden state in the sequence**



# THE ATTENTION MECHANISM

- Self-Attention

**Find the attention of each hidden state with every other hidden state in the sequence**



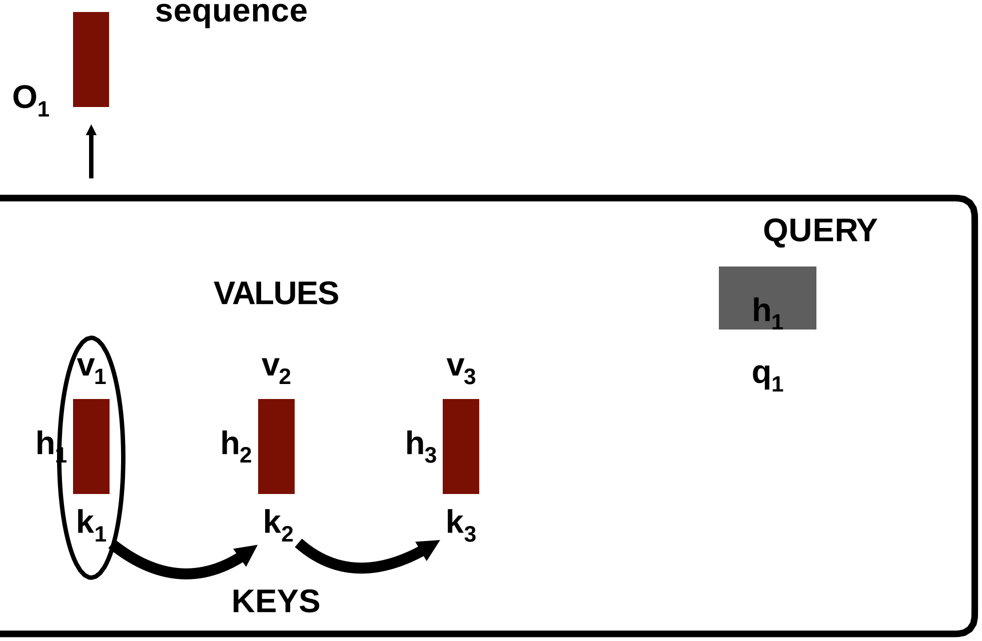

$$\boldsymbol{O}_1\,(\boldsymbol{k}_i,\ \boldsymbol{q}_1) = \sum w_i\, \boldsymbol{v}_i$$

$$\boxed{w_{i(t)} = \frac{\exp(\, e_i(\boldsymbol{q}_{(t)}, \boldsymbol{k}_i)\,)}{\sum_j \exp(\, e_j(\boldsymbol{q}_{(t)}, \boldsymbol{k}_j)\,)}}$$

# THE ATTENTION MECHANISM

- Self-Attention

**Find the attention of each hidden state with every other hidden state in the sequence**



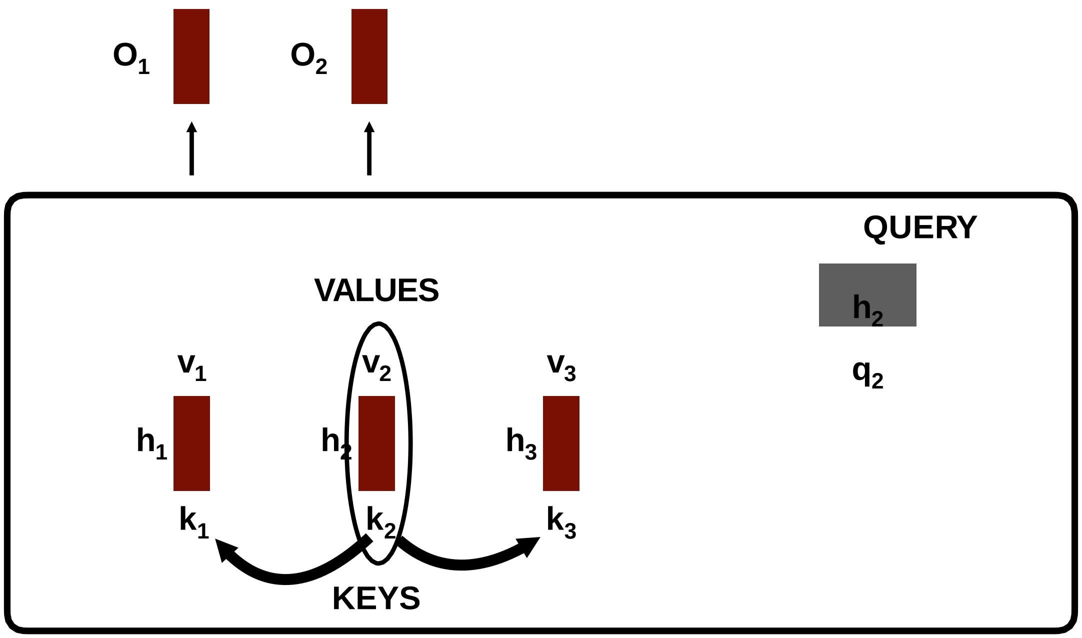

$$\boldsymbol{O}_1 (\boldsymbol{k}_i, \boldsymbol{q}_1) = \sum w_i \boldsymbol{v}_i$$

$$w_{i(t)} = \frac{\exp(\, e_i(\boldsymbol{q}_{(t)}, \boldsymbol{k}_i)\,)}{\sum_j \exp(\, e_j(\boldsymbol{q}_{(t)}, \boldsymbol{k}_j)\,)}$$

# THE ATTENTION MECHANISM

- Self-Attention

**Find the attention of each hidden state with every other hidden state in the sequence**



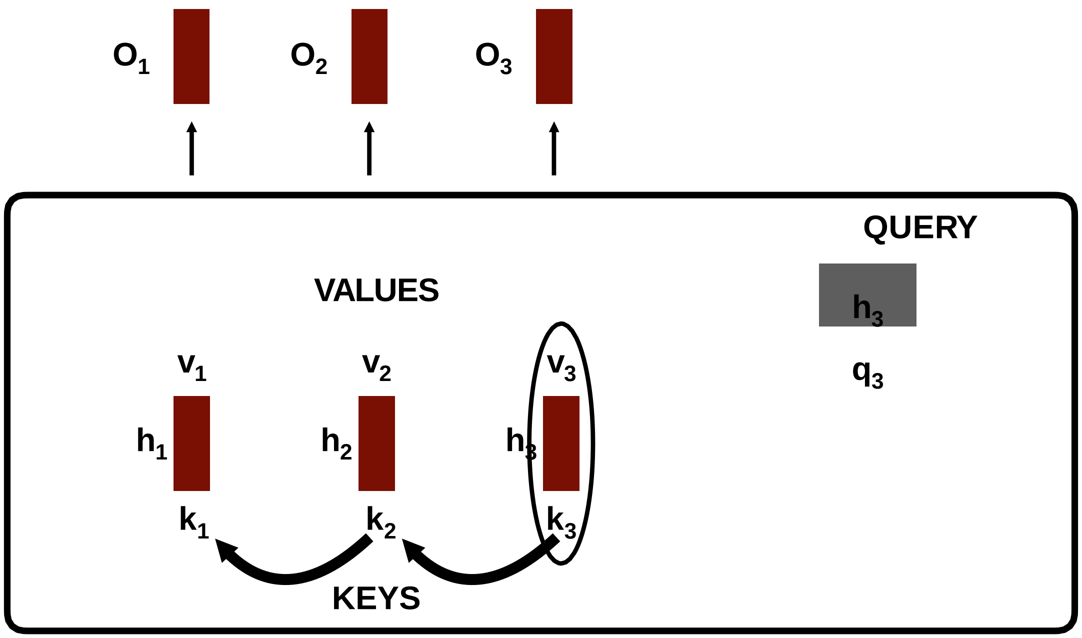

$$O_2\,(\boldsymbol{k}_i,\ \boldsymbol{q}_2) = \sum w_i\ \boldsymbol{v}_i$$

$$w_{i(t)} = \frac{\exp(\ e_i(\boldsymbol{q}_{(t)},\ \boldsymbol{k}_i)\ )}{\sum_j \exp(\ e_j(\boldsymbol{q}_{(t)},\ \boldsymbol{k}_j)\ )}$$

# THE ATTENTION MECHANISM

- **Self-Attention**

**Find the attention of each hidden state with every other hidden state in the sequence**



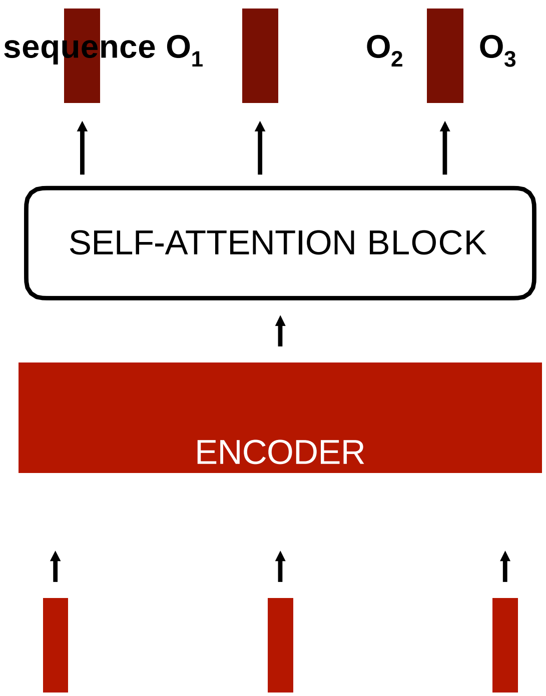

$$O_3\,(\boldsymbol{k}_i, \boldsymbol{q}_3) = \sum w_i\, \boldsymbol{v}_i$$

$$\boxed{w_{i(t)} = \frac{\exp(\, e_i(\boldsymbol{q}_{(t)}, \boldsymbol{k}_i)\,)}{\sum_j \exp(\, e_j(\boldsymbol{q}_{(t)}, \boldsymbol{k}_j)\,)}}$$

# THE ATTENTION MECHANISM

- **Self-Attention**

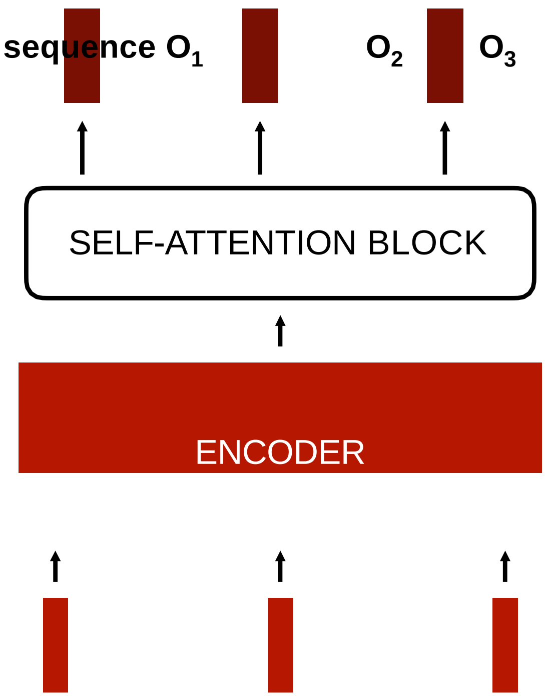
**Find the attention of each hidden state with every other hidden state in the sequence**



# THE ATTENTION MECHANISM

- Self-Attention

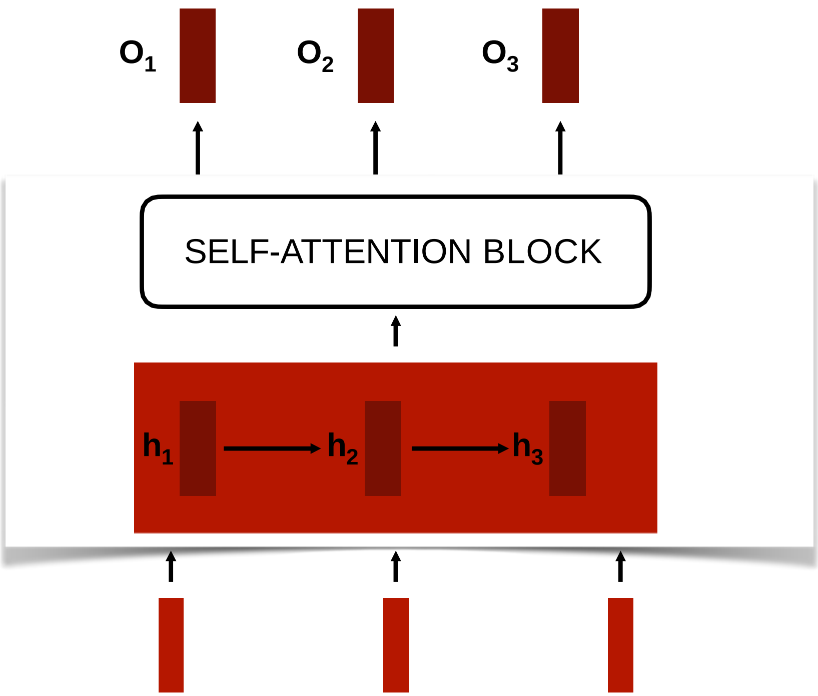
**Find the attention of each hidden state with every other hidden state in the sequence**




**Note: Our attention mechanism has no learnable parameter if we use dot product attention**

# THE ATTENTION MECHANISM

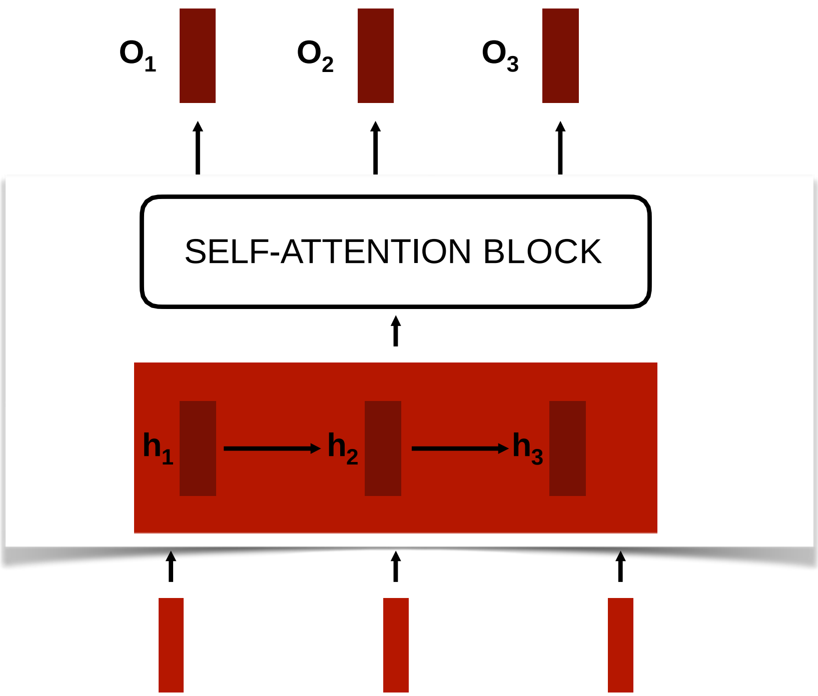
- **Self-Attention**




ENCODER BLOCK

# THE ATTENTION MECHANISM

- Self-Attention

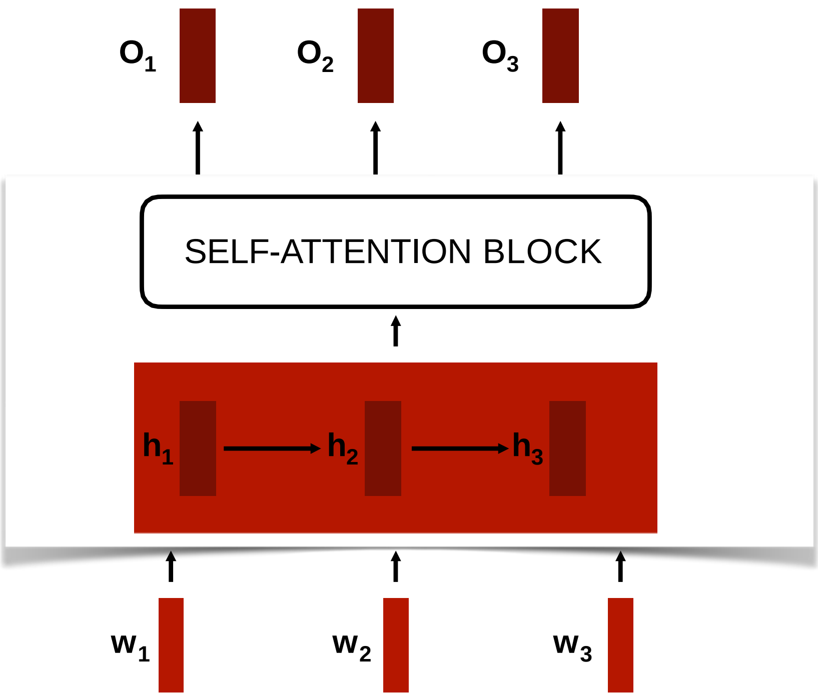



ENCODER BLOCK

What does self attention do here?

# THE ATTENTION MECHANISM

- **Self-Attention**

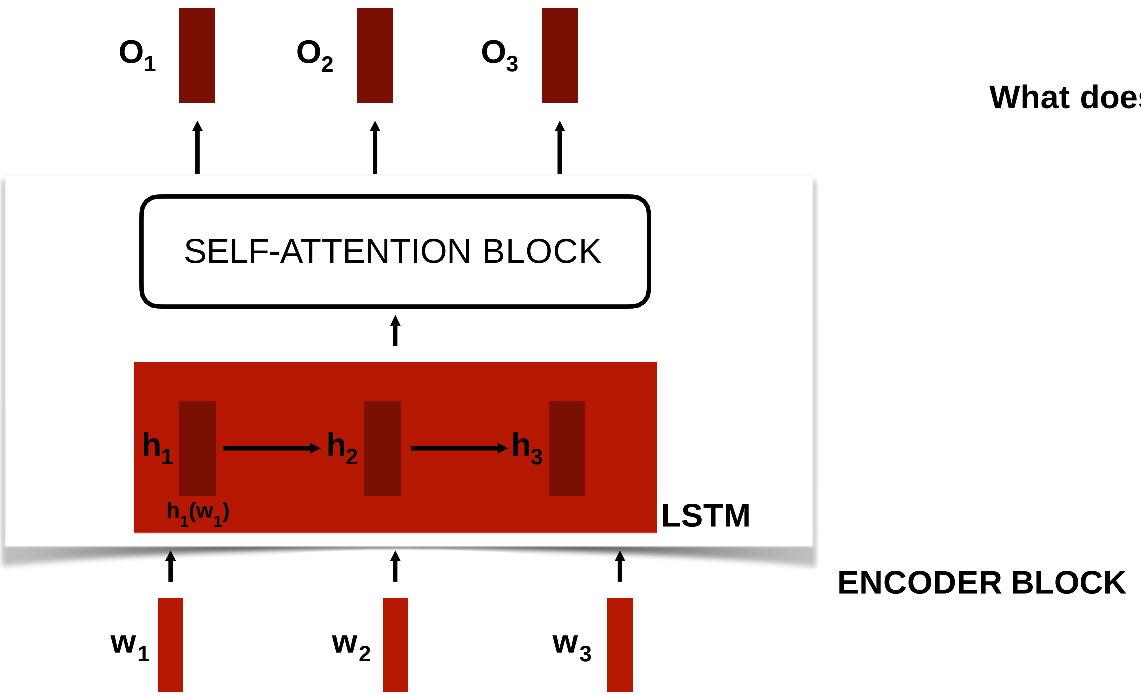



**What does self attention do here?**

**ENCODER BLOCK**

# THE ATTENTION MECHANISM

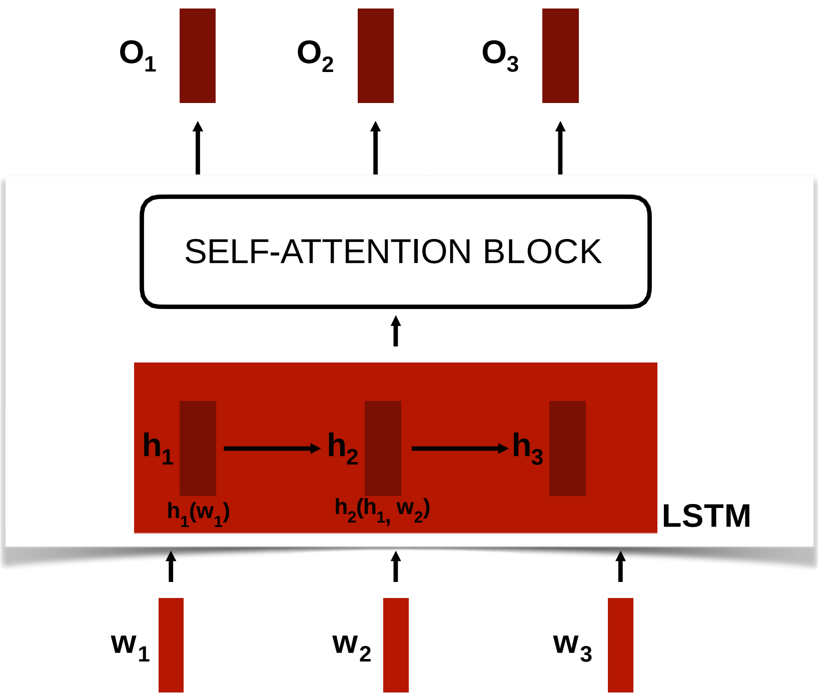
- Self-Attention




What does self attention do here?

# THE ATTENTION MECHANISM

- Self-Attention

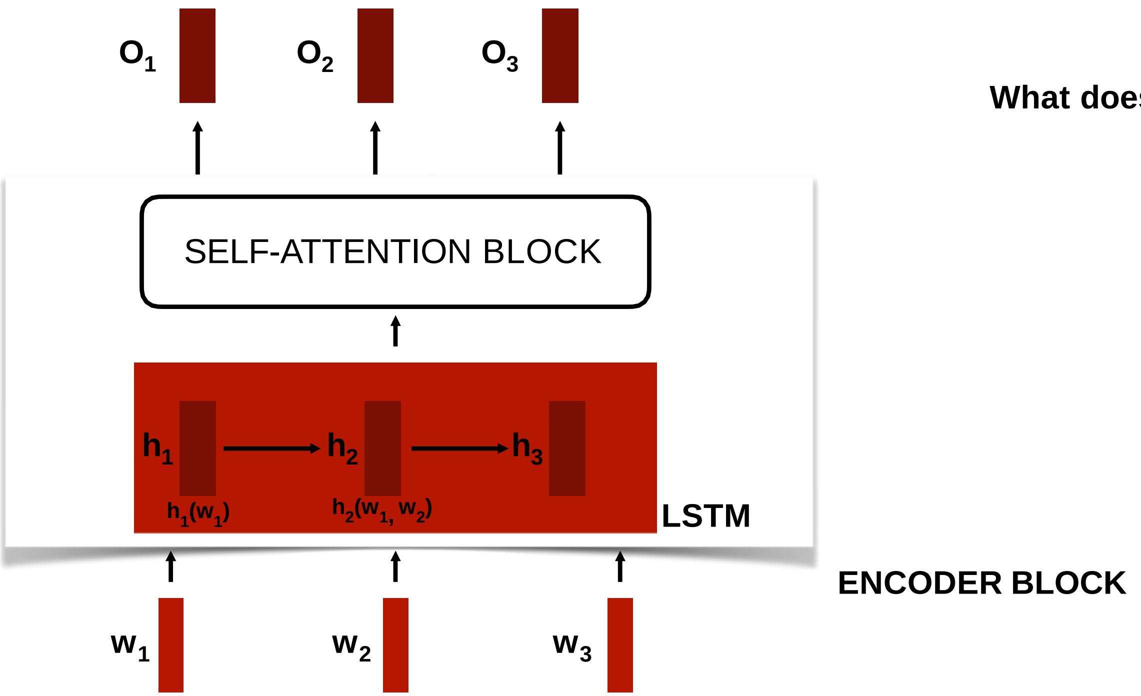



ENCODER BLOCK

What does self attention do here?

# THE ATTENTION MECHANISM

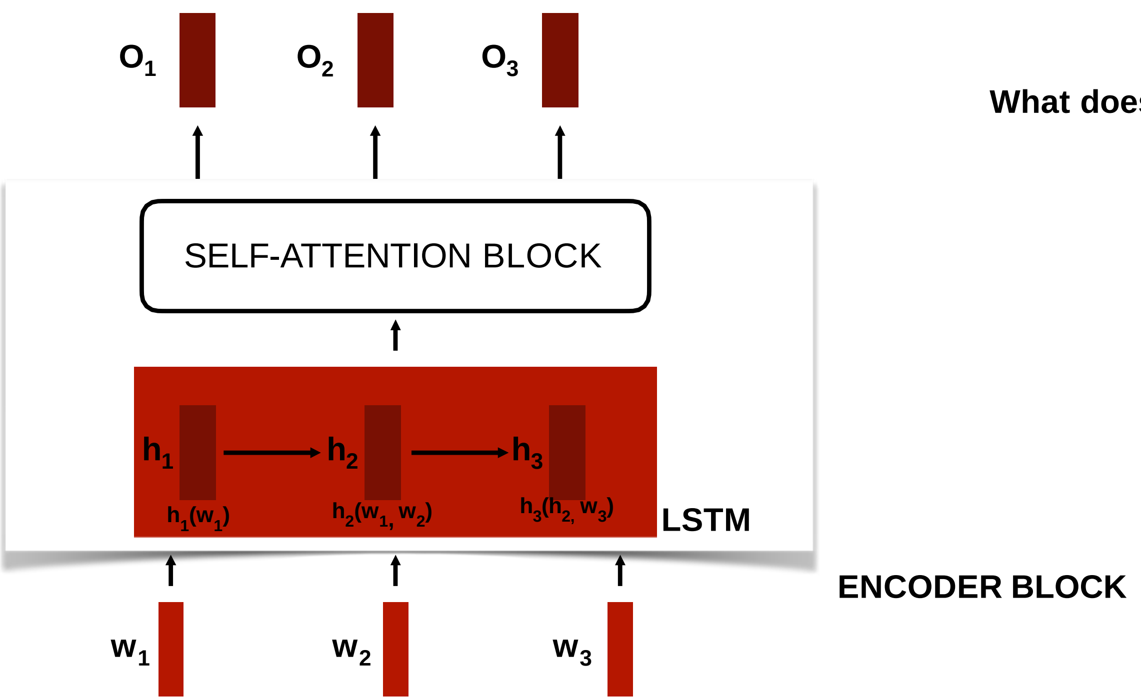
- Self-Attention




**What does self attention do here?**

# THE ATTENTION MECHANISM

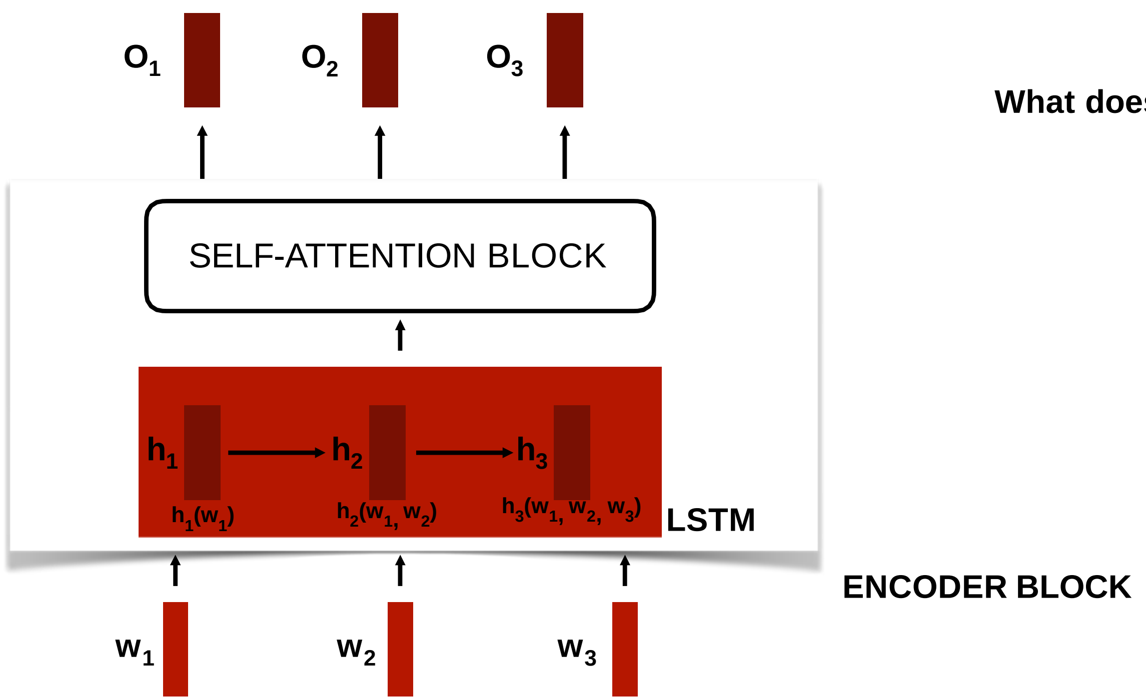
- Self-Attention




What does self attention do here?

# THE ATTENTION MECHANISM

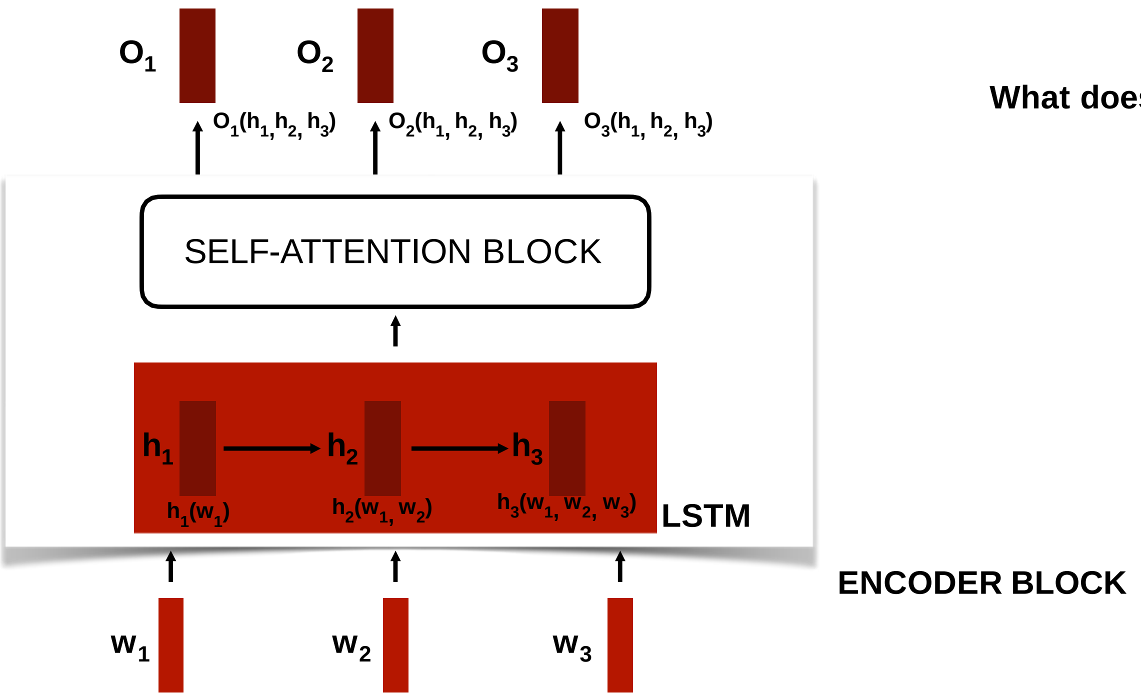
- Self-Attention




What does self attention do here?

# THE ATTENTION MECHANISM

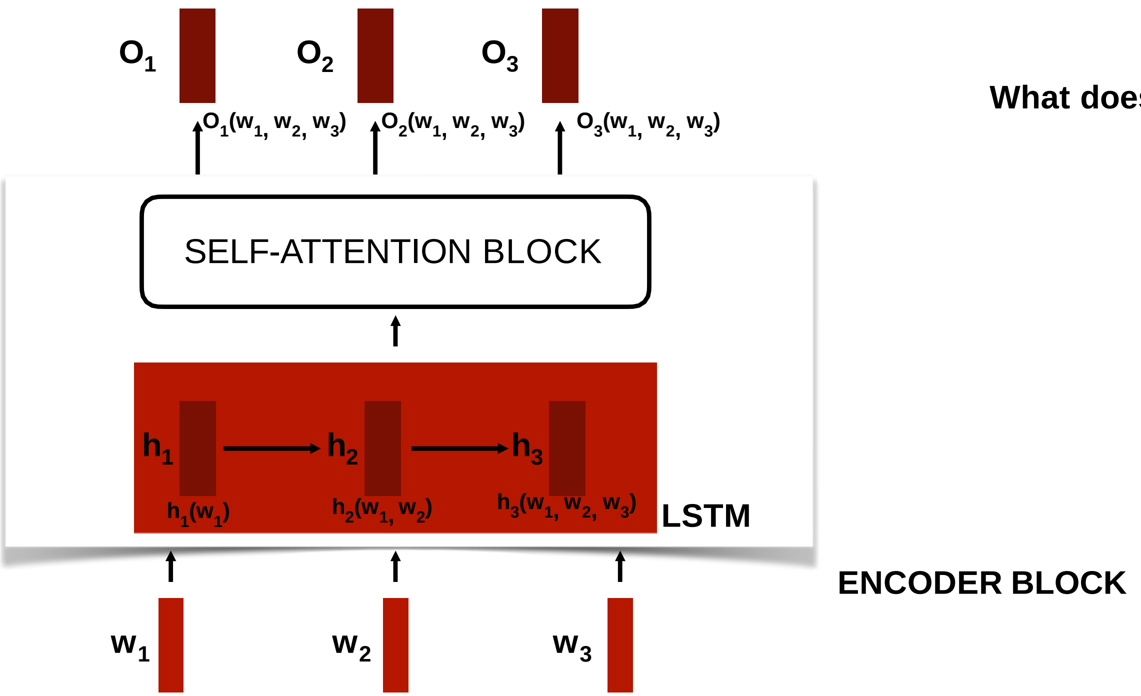
- Self-Attention




What does self attention do here?

# THE ATTENTION MECHANISM

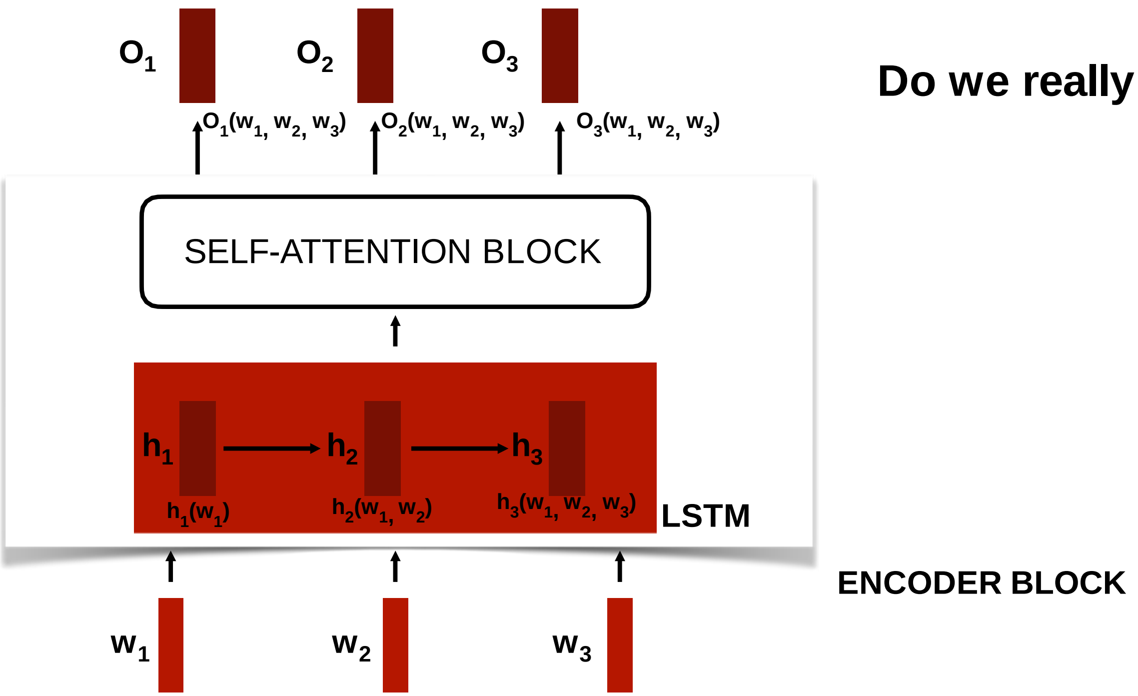
- Self-Attention




What does self attention do here?

# THE ATTENTION MECHANISM

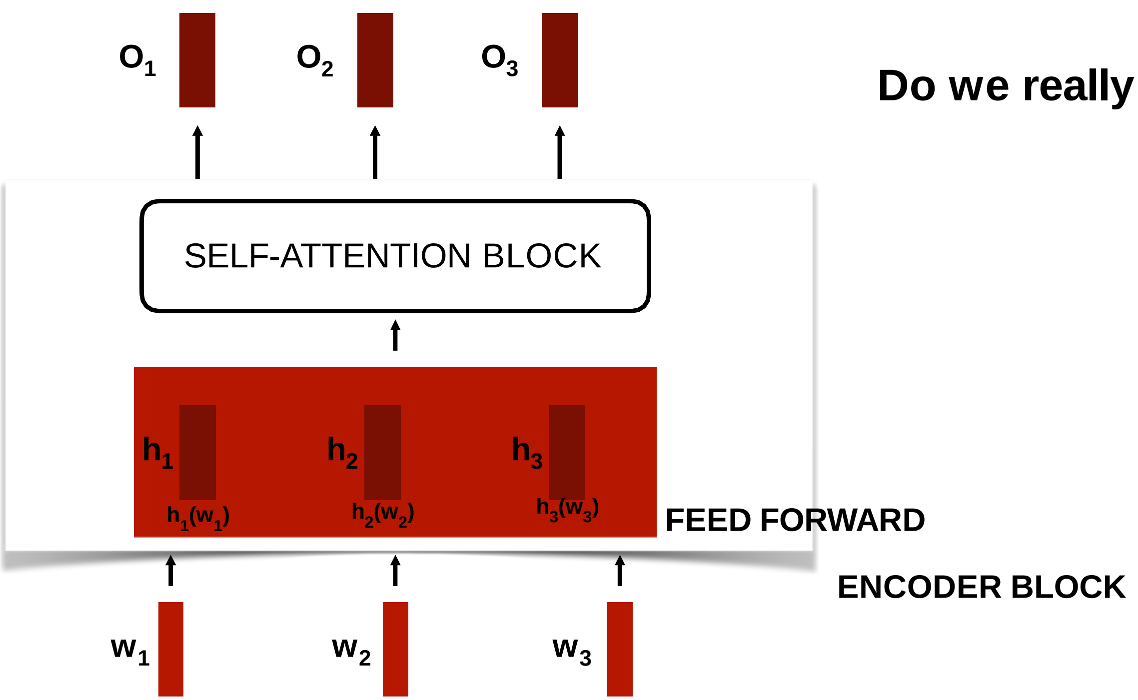
- **Self-Attention**




**Do we really need the LSTM to model sequences?**

# THE ATTENTION MECHANISM

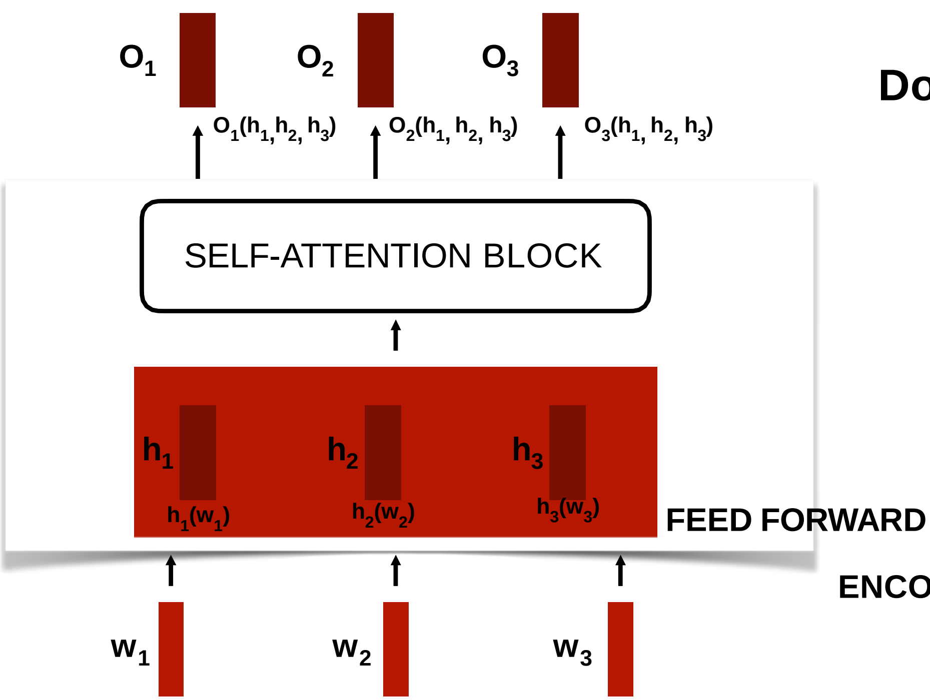
- Self-Attention




**Do we really need the LSTM to model sequences?**

# THE ATTENTION MECHANISM

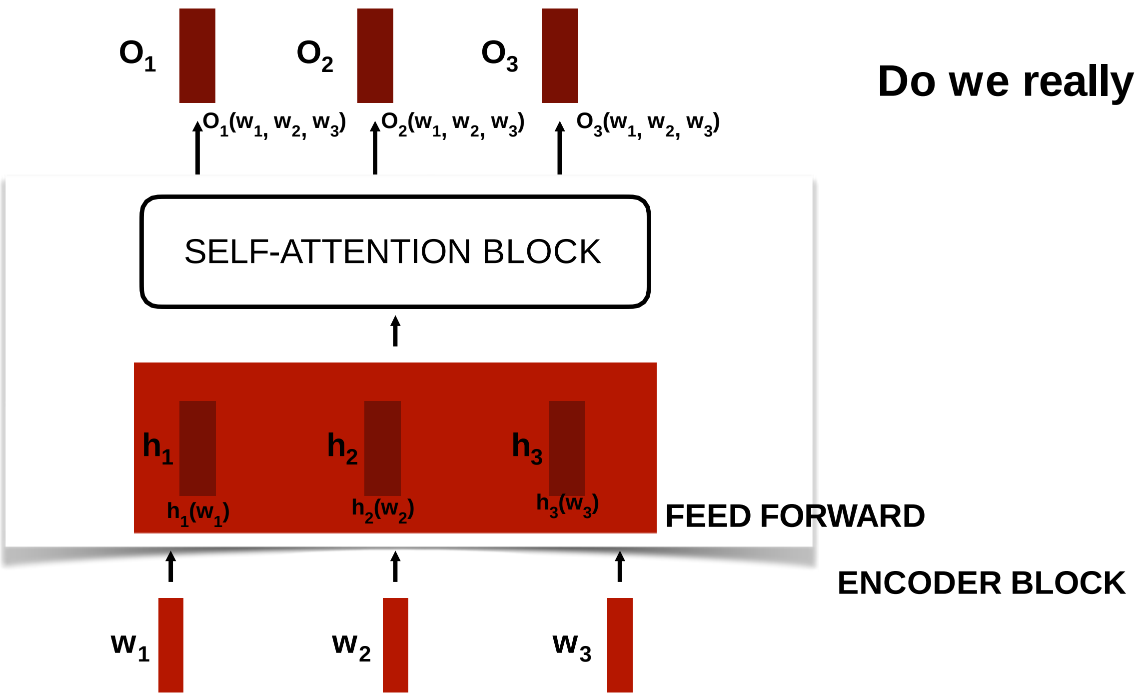
- Self-Attention




**Do we really need the LSTM to model sequences?**

# THE ATTENTION MECHANISM

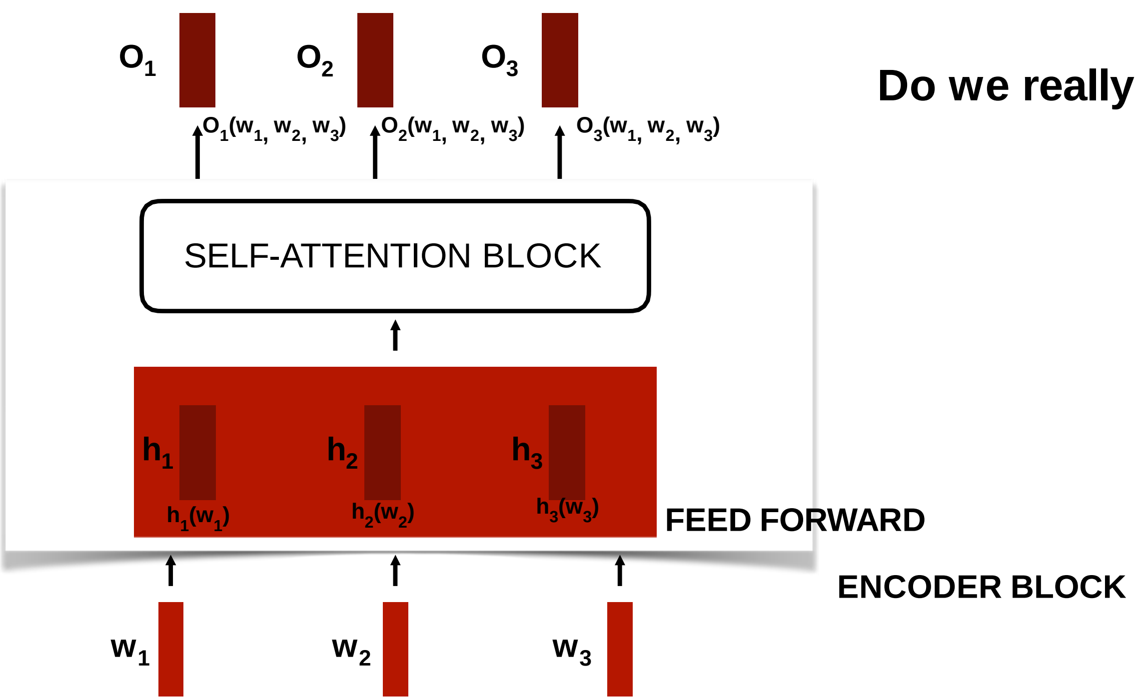
- Self-Attention




Do we really need the LSTM to model sequences?

# THE ATTENTION MECHANISM

- **Self-Attention**

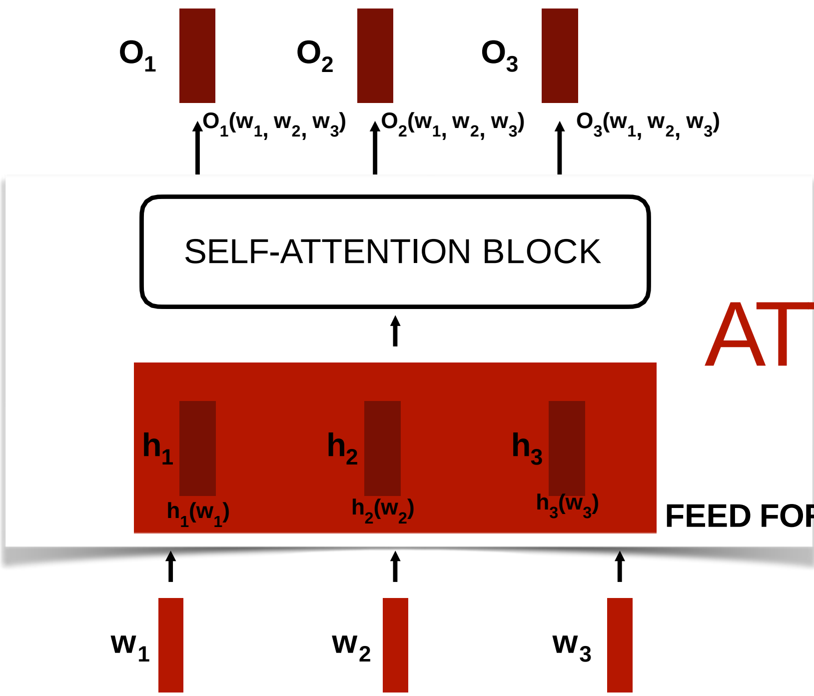



**Do we really need the LSTM to model sequences?**

**NO!**

# THE ATTENTION MECHANISM

- **Self-Attention**




**Do we really need the LSTM to model sequences?**

**NO!**

ATTENTION IS ALL YOU NEED!

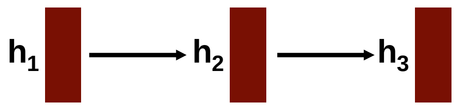
# THE ATTENTION MECHANISM

- Attention
- Self-Attention
- **Multi-Head Attention**

# THE ATTENTION MECHANISM

- **Multi-Head Attention**

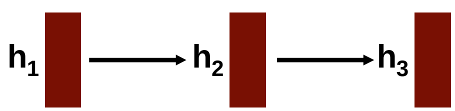
**NOTE : Query, Key, Values are generalizations of the input to the attention mechanism.**



# THE ATTENTION MECHANISM

- **Multi-Head Attention**

**NOTE : Query, Key, Values are generalizations of the input to the attention mechanism.**




$k_1 = W_k h_1$ $k_2 = W_k h_2$ $k_3 = W_k h_3$

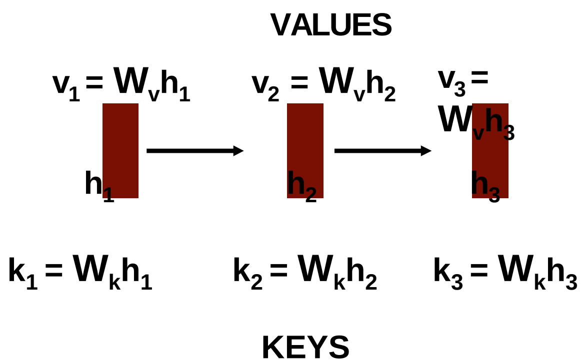
**KEYS**

$W_k$ =: **To convert input sequence to keys**

# THE ATTENTION MECHANISM

- Multi-Head Attention

NOTE : Query, Key, Values are generalizations of the input to the attention mechanism.




$W_k$ =: To convert input sequence to keys

$W_v$ =: To convert input sequence to values

# THE ATTENTION MECHANISM

- Multi-Head Attention

NOTE : Query, Key, Values are generalizations of the input to the attention mechanism.

VALUES

$v_1 = W_v h_1$ $v_2 = W_v h_2$ $v_3 = W_v h_3$

$h_1 \longrightarrow h_2 \longrightarrow h_3$

$k_1 = W_k h_1$ $k_2 = W_k h_2$ $k_3 = W_k h_3$

KEYS

$h_1$ $h_2$ $h_3$

$q_1 = W_q h_1$ $q_2 = W_q h_2$ $q_3 = W_q h_3$

QUERIES

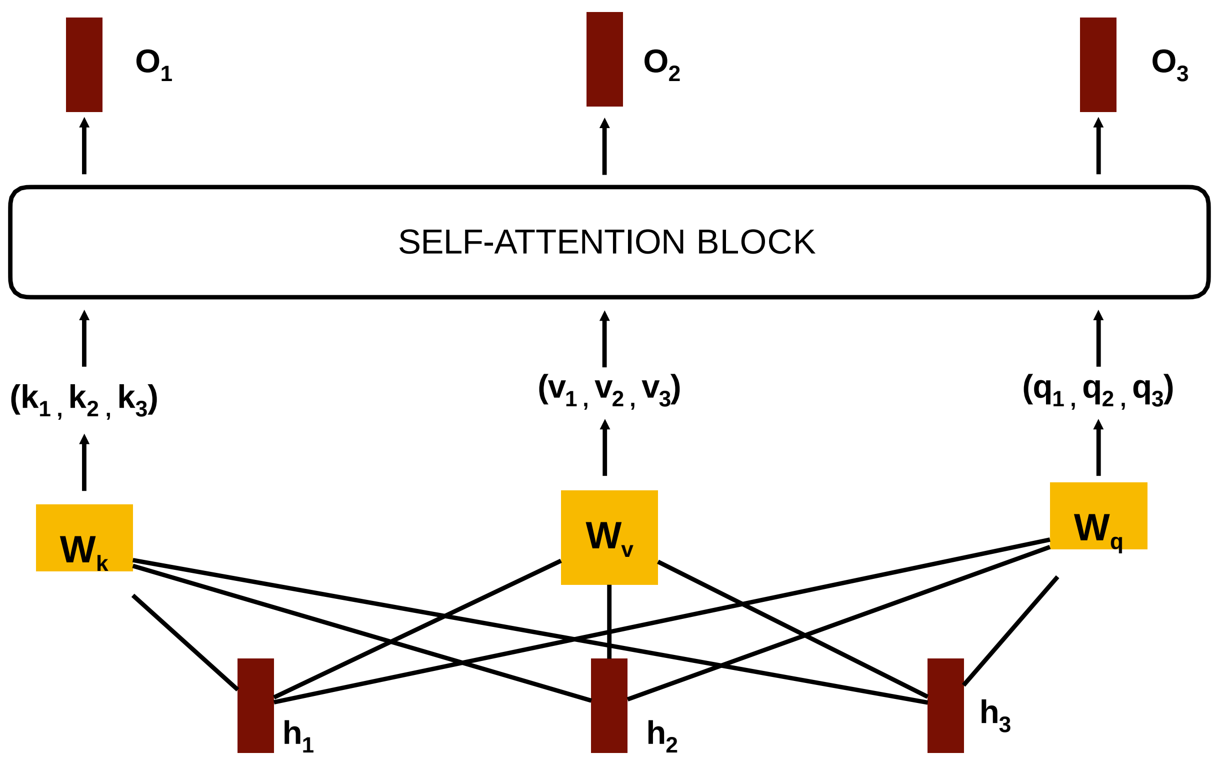
$W_k$ =: To convert input sequence to keys

$W_v$ =: To convert input sequence to values

$W_q$=: To convert input sequence to values

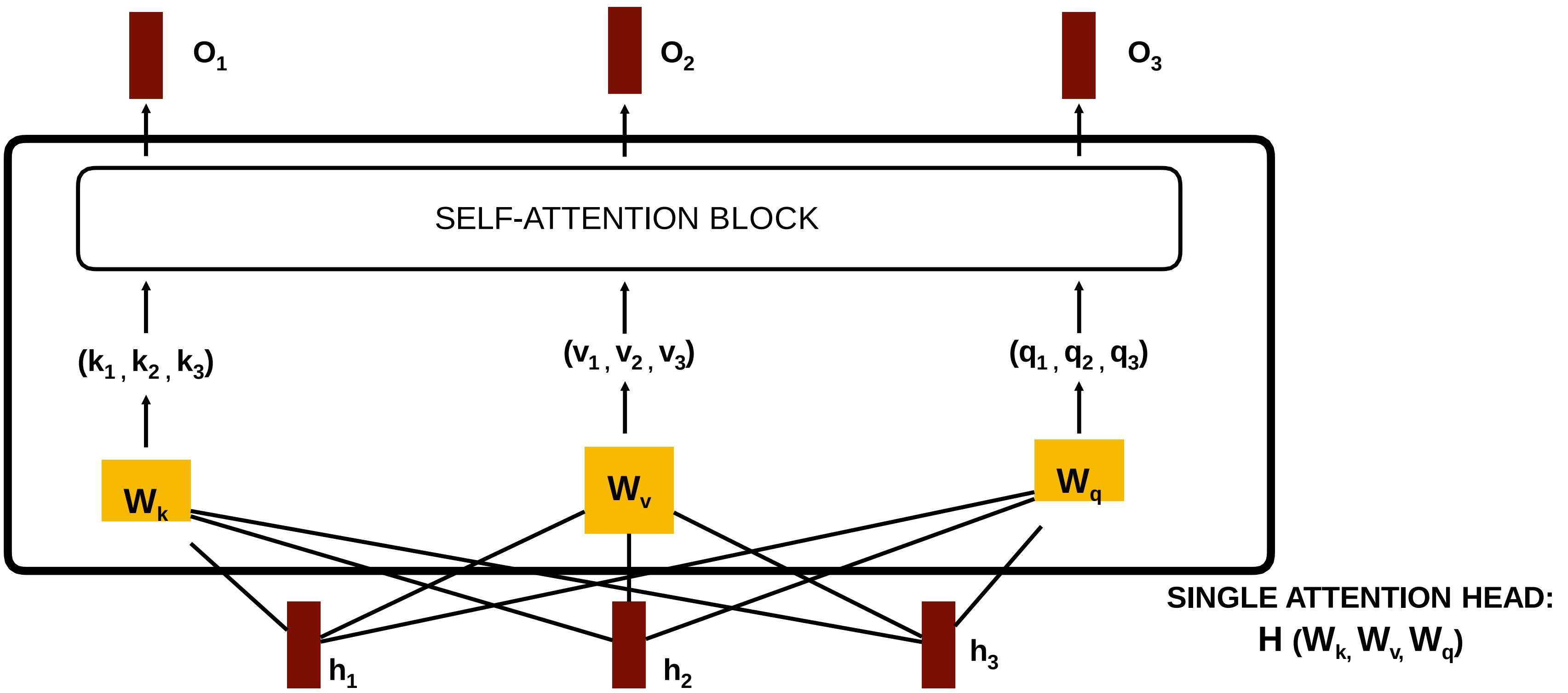
# THE ATTENTION MECHANISM

- Multi-Head Attention



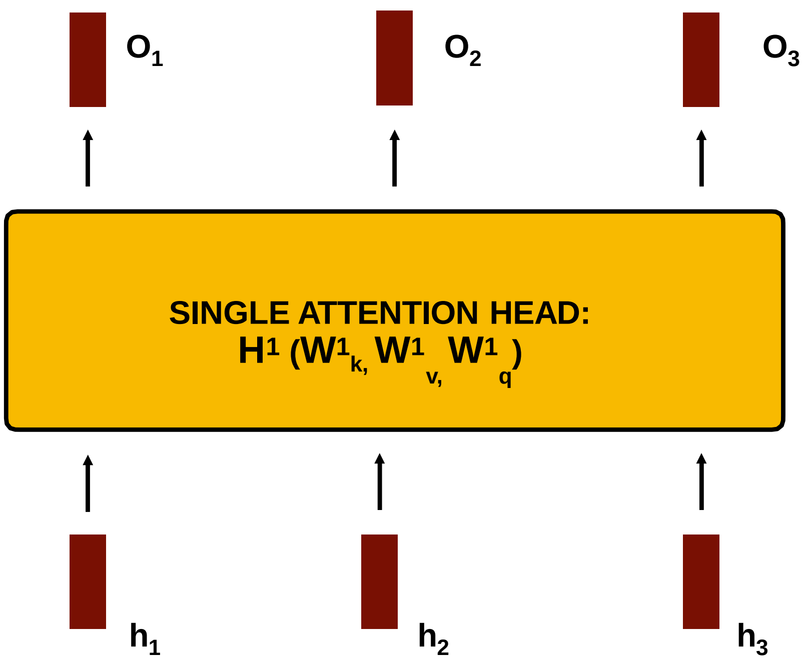
# THE ATTENTION MECHANISM

- **Multi-Head Attention**



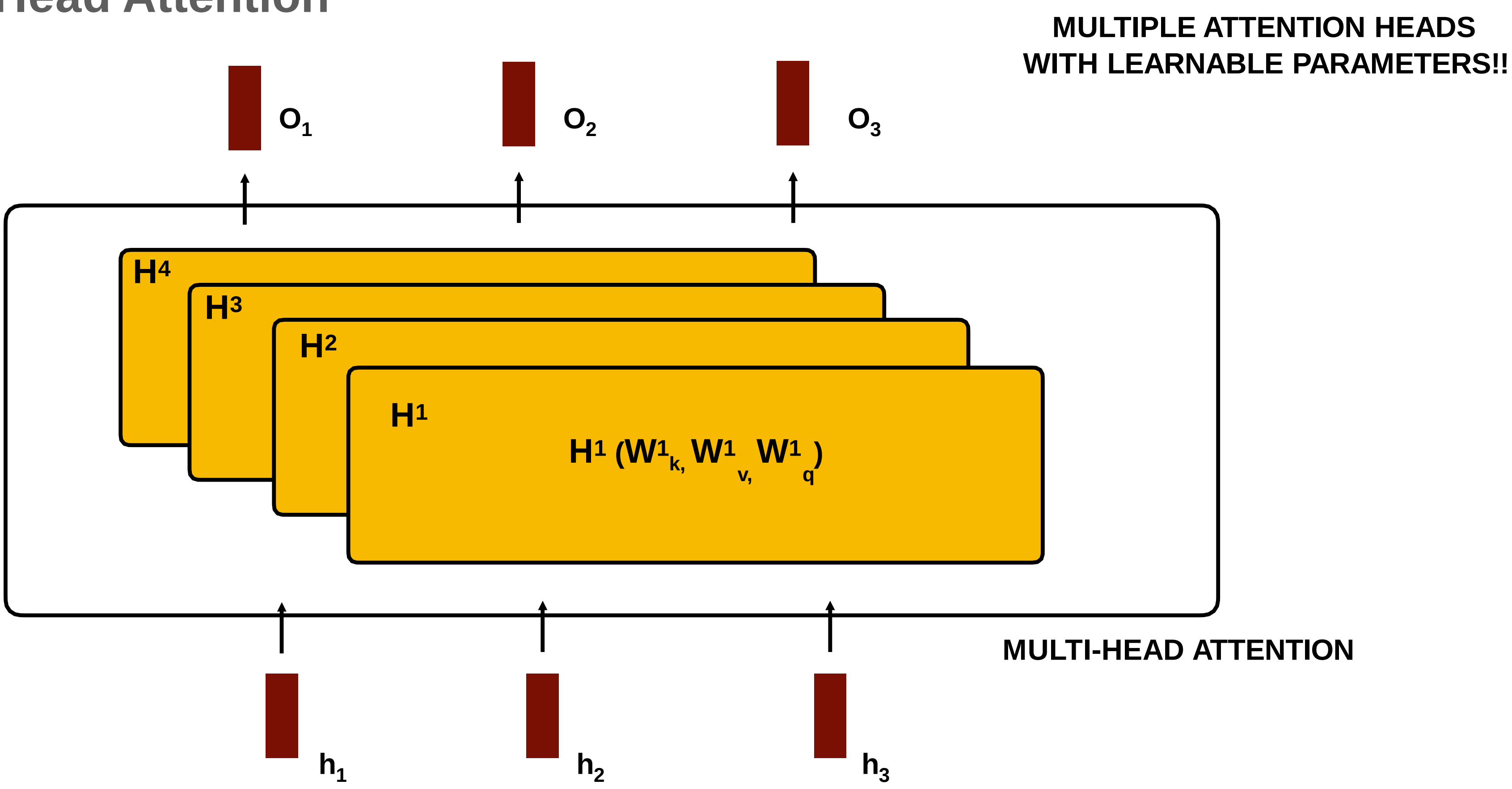
# THE ATTENTION MECHANISM

- Multi-Head Attention



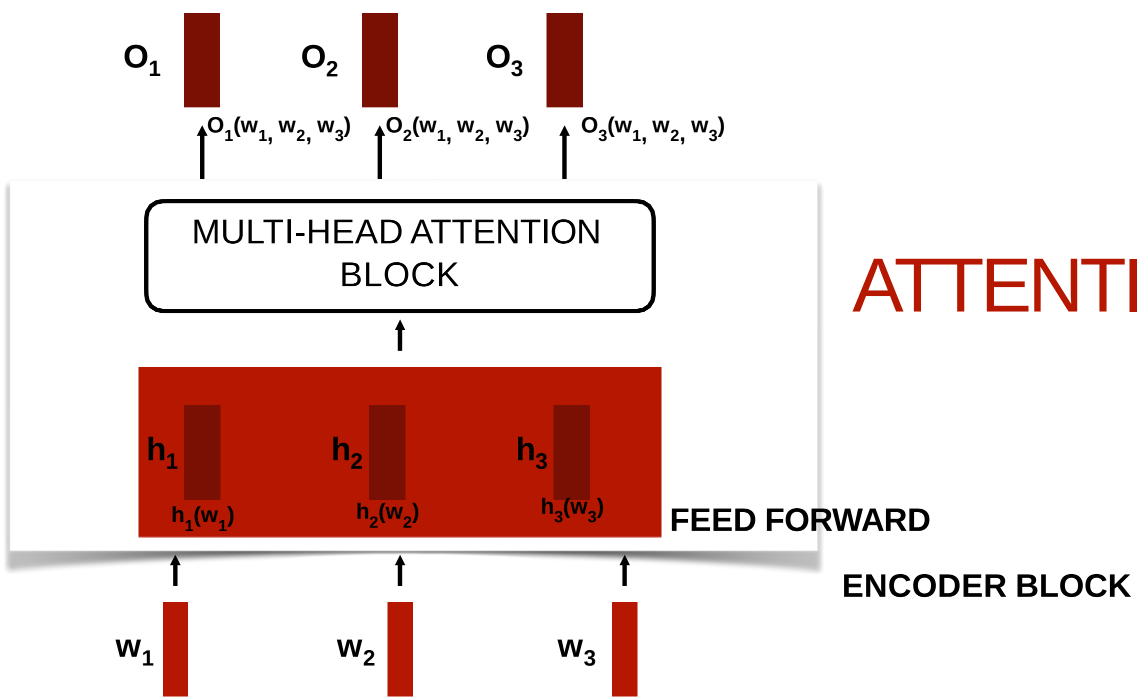
# THE ATTENTION MECHANISM

- Multi-Head Attention



# THE ATTENTION MECHANISM

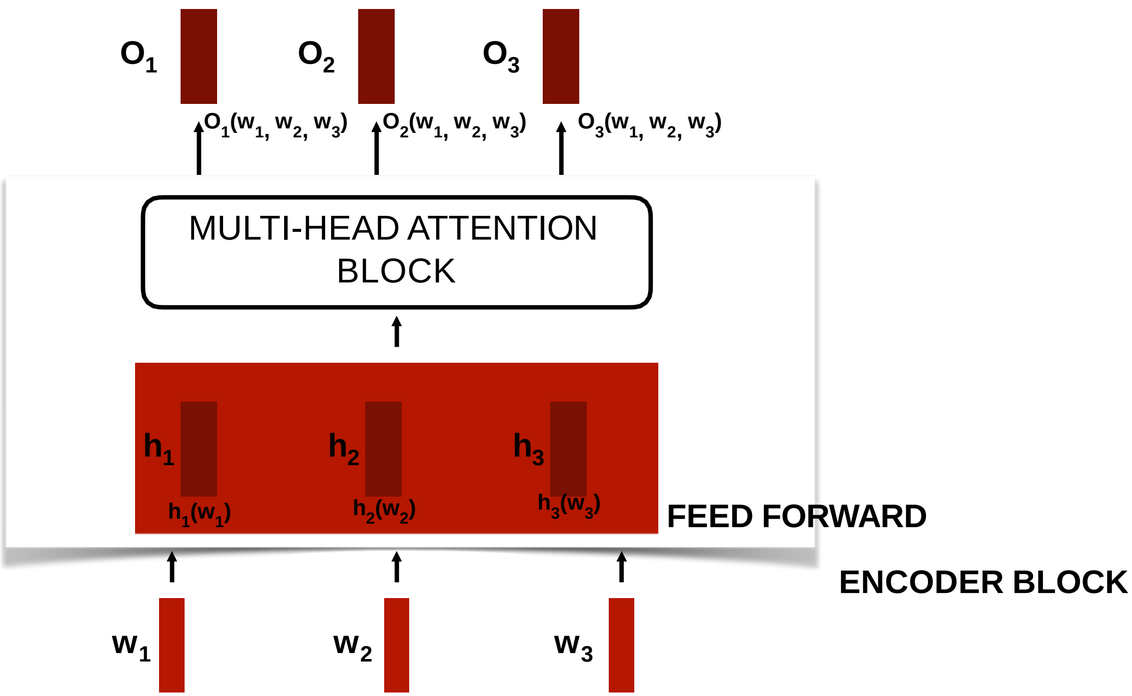
- Multi-Head Attention



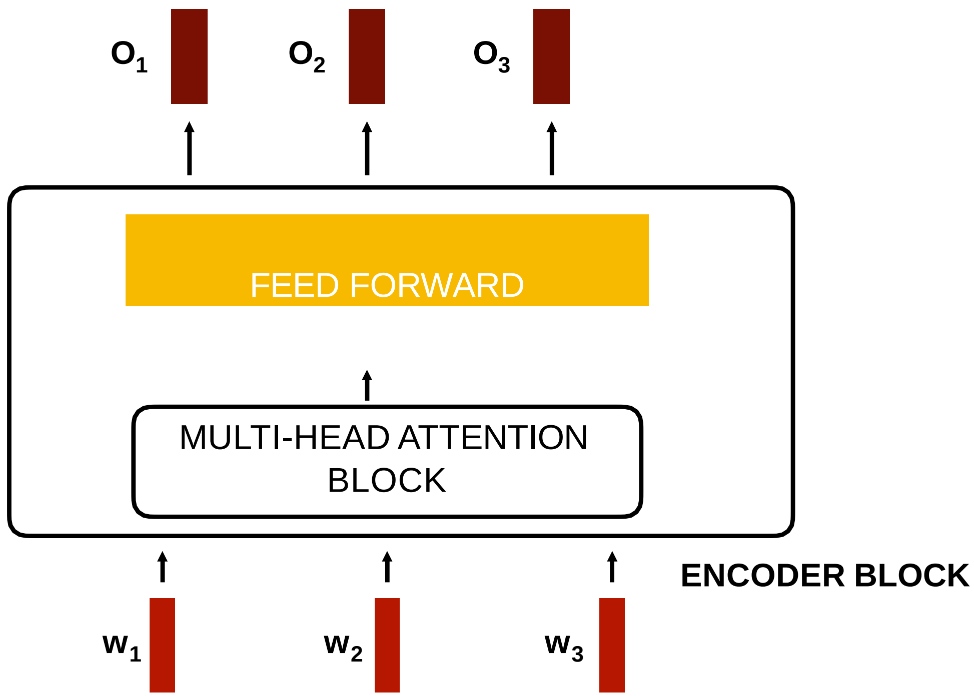

ATTENTION IS ALL YOU NEED!

# THE TRANSFORMER ARCHITECTURE

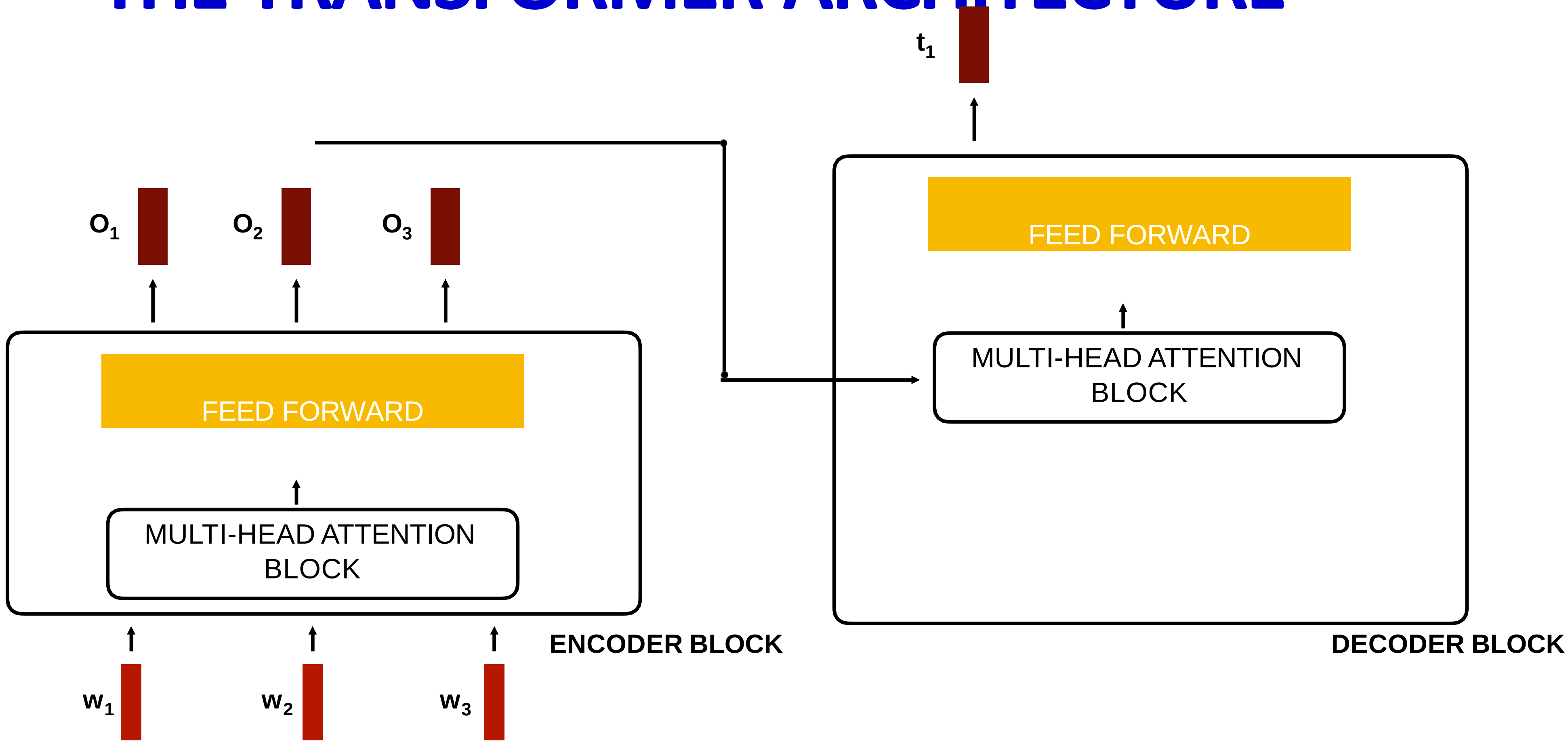
# THE TRANSFORMER ARCHITECTURE



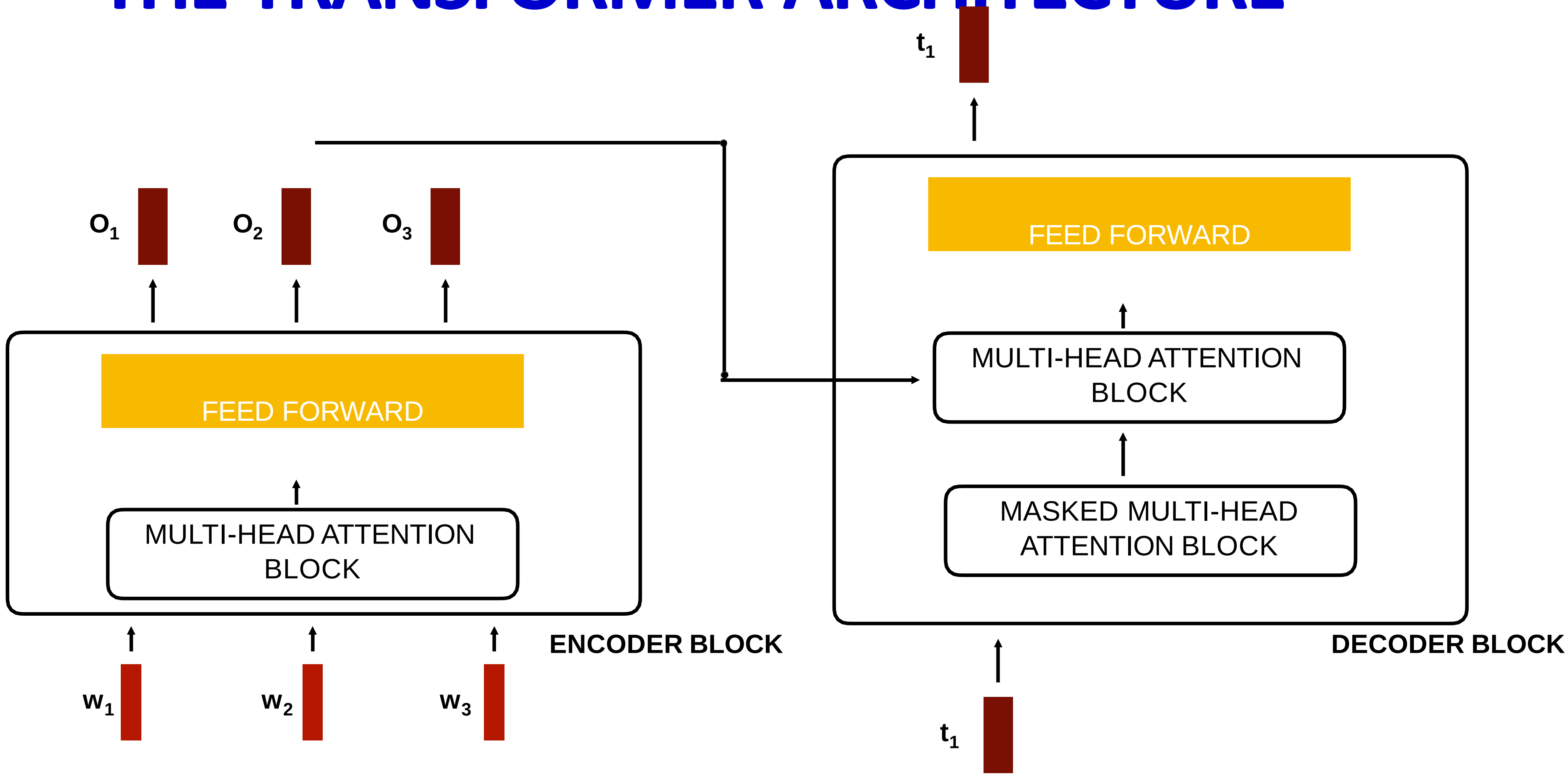
# THE TRANSFORMER ARCHITECTURE



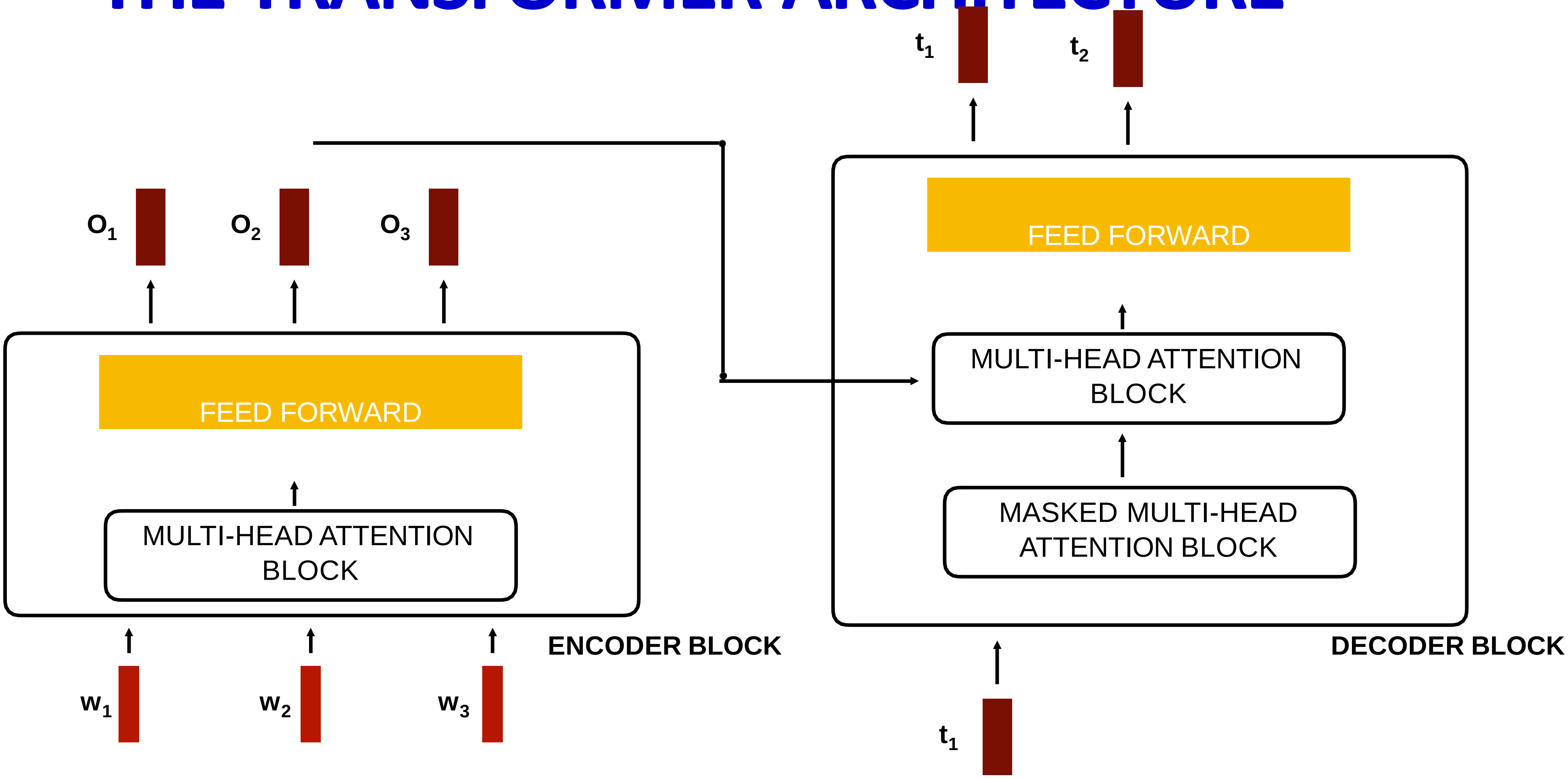
# THE TRANSFORMER ARCHITECTURE



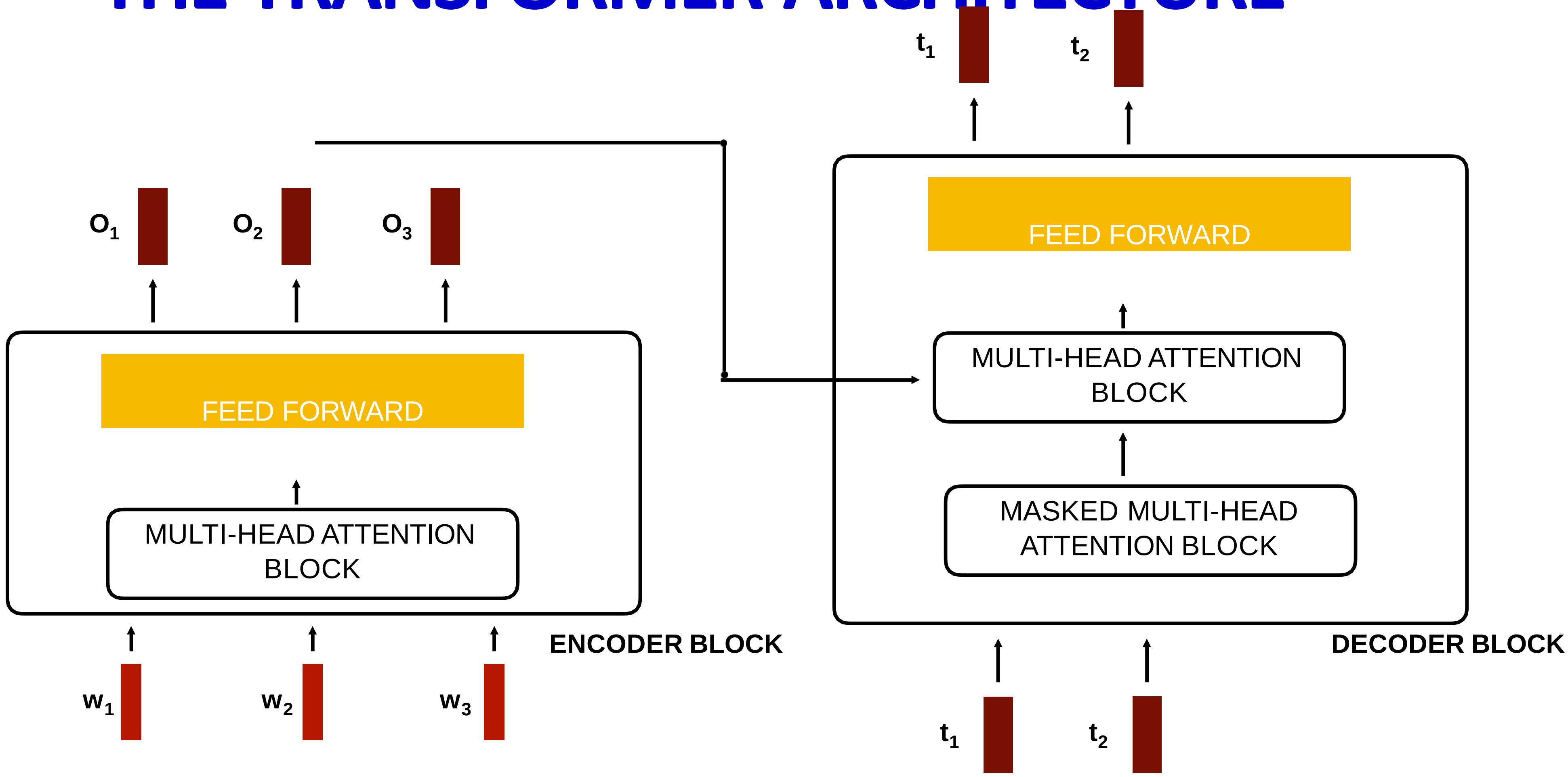
# THE TRANSFORMER ARCHITECTURE



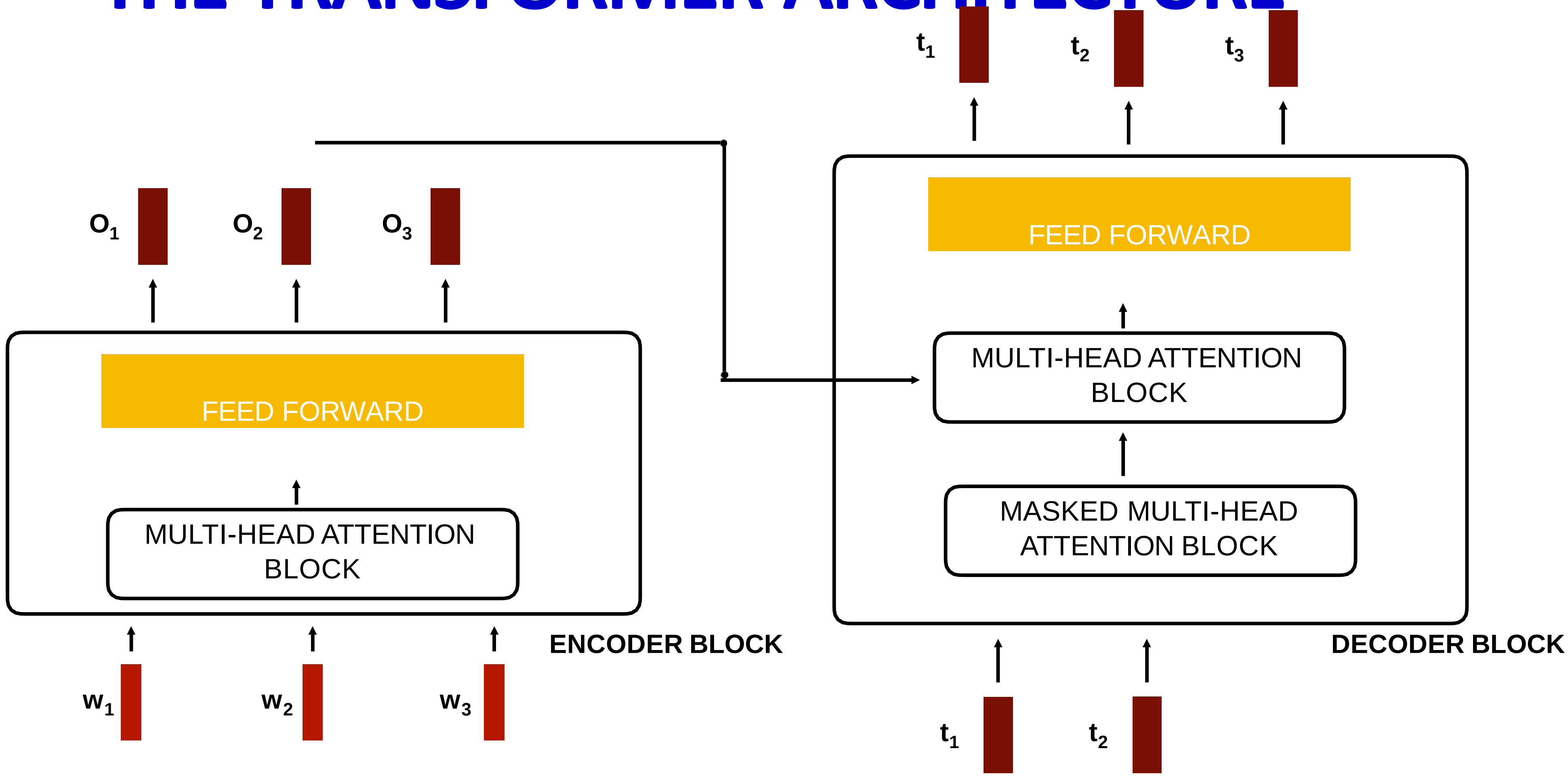
# THE TRANSFORMER ARCHITECTURE



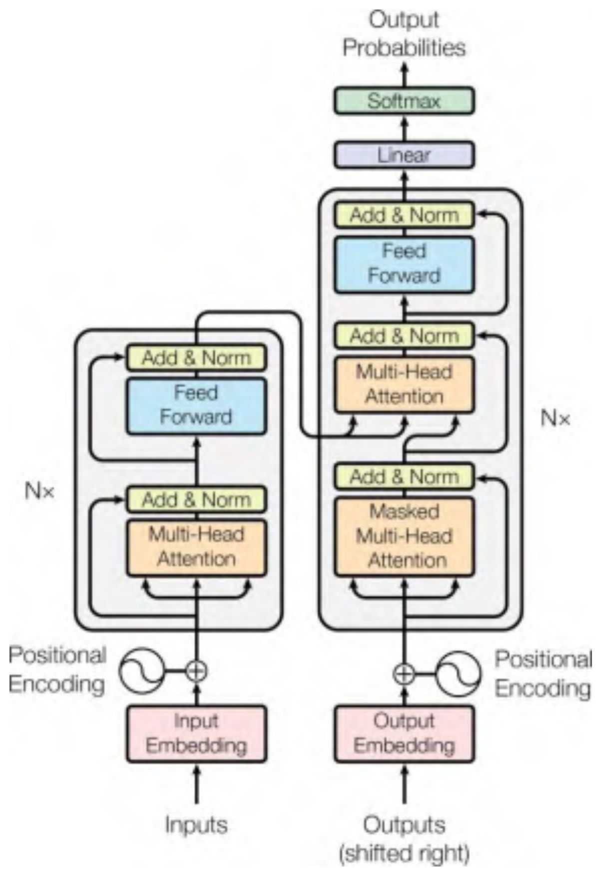
# THE TRANSFORMER ARCHITECTURE



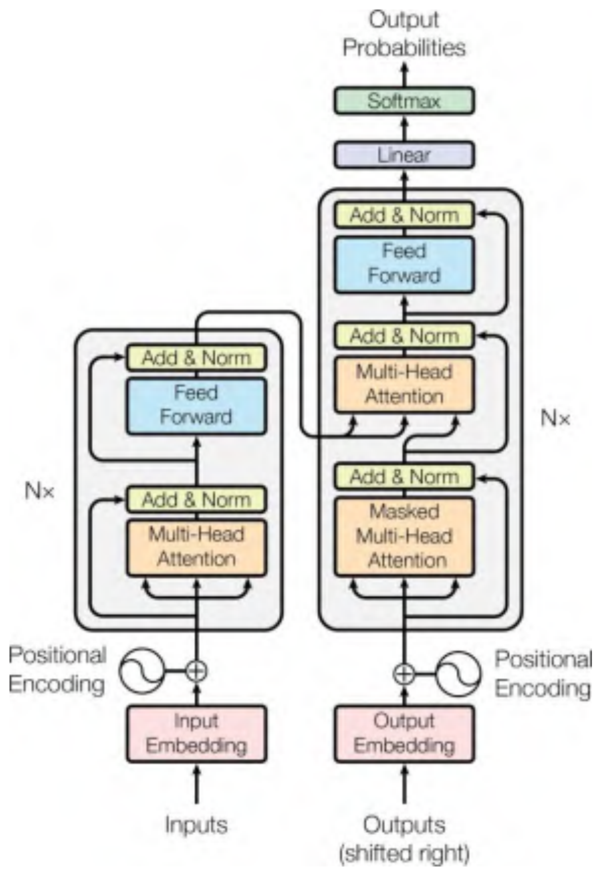
# THE TRANSFORMER ARCHITECTURE



# THE TRANSFORMER ARCHITECTURE

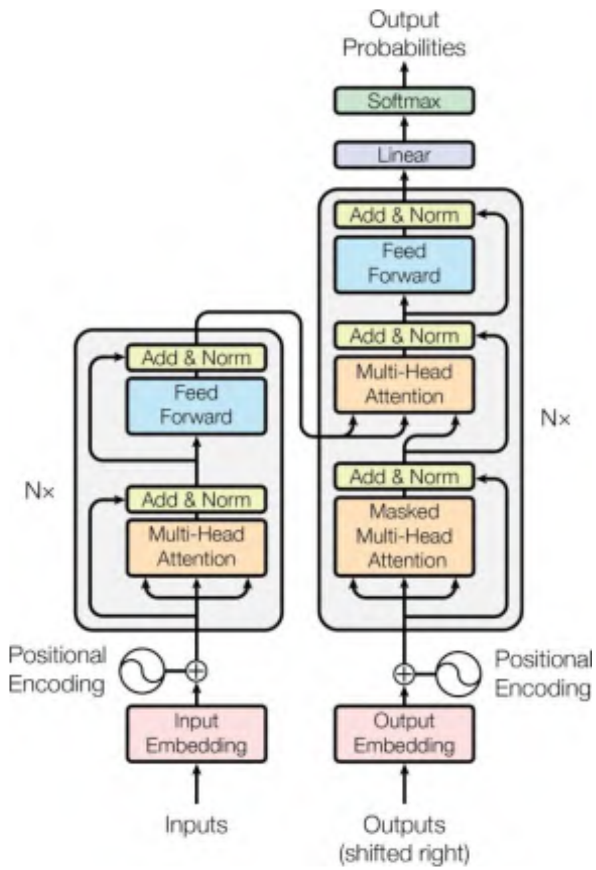


# THE TRANSFORMER ARCHITECTURE




**Multiple stacked encoder and decoder blocks!**

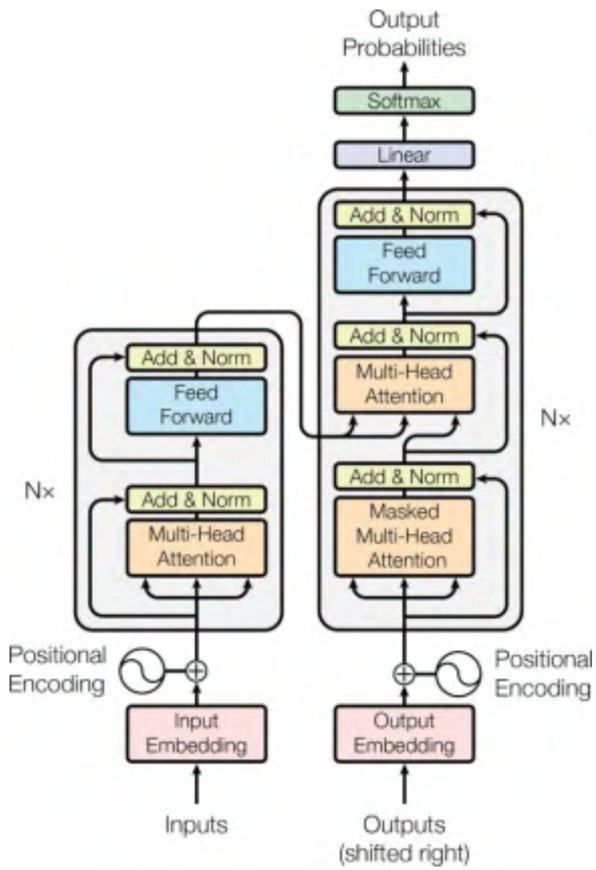
# THE TRANSFORMER ARCHITECTURE




**Multiple stacked encoder and decoder blocks!**

**Layer Normalization!**

# THE TRANSFORMER ARCHITECTURE



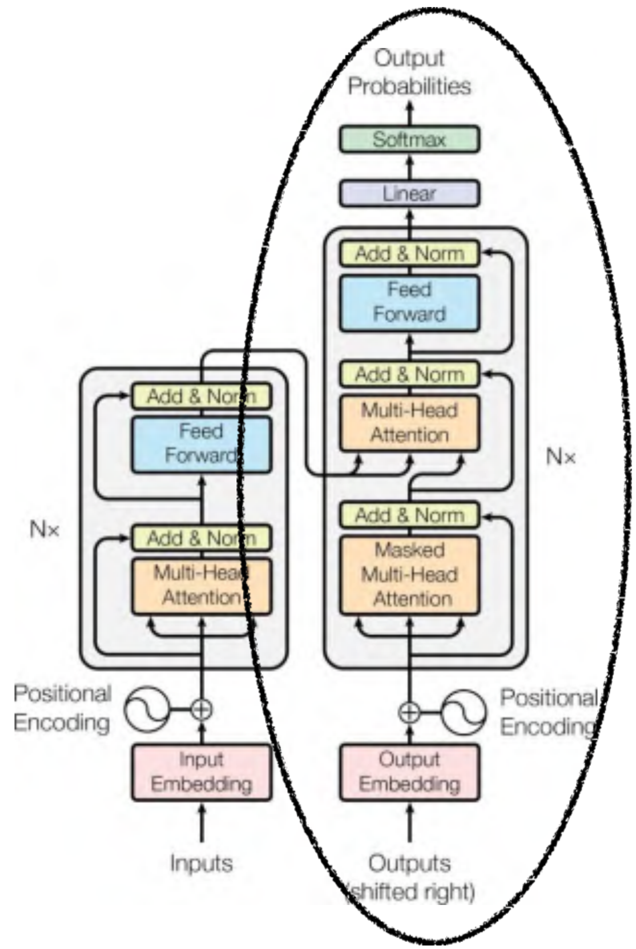

**Multiple stacked encoder and decoder blocks!**

**Layer Normalization!**

**Positional Embeddings!**

# GPT ARCHITECTURE

# GPT ARCHITECTURE



# BERT ARCHITECTURE

# BERT ARCHITECTURE