

Shallow vs Deep NNs

DSE 3151 Deep Learning

DSE 3151, B.Tech Data Science & Engineering

August 2023

Rohini R Rao & Abhilash Pai

Department of Data Science and Computer Applications

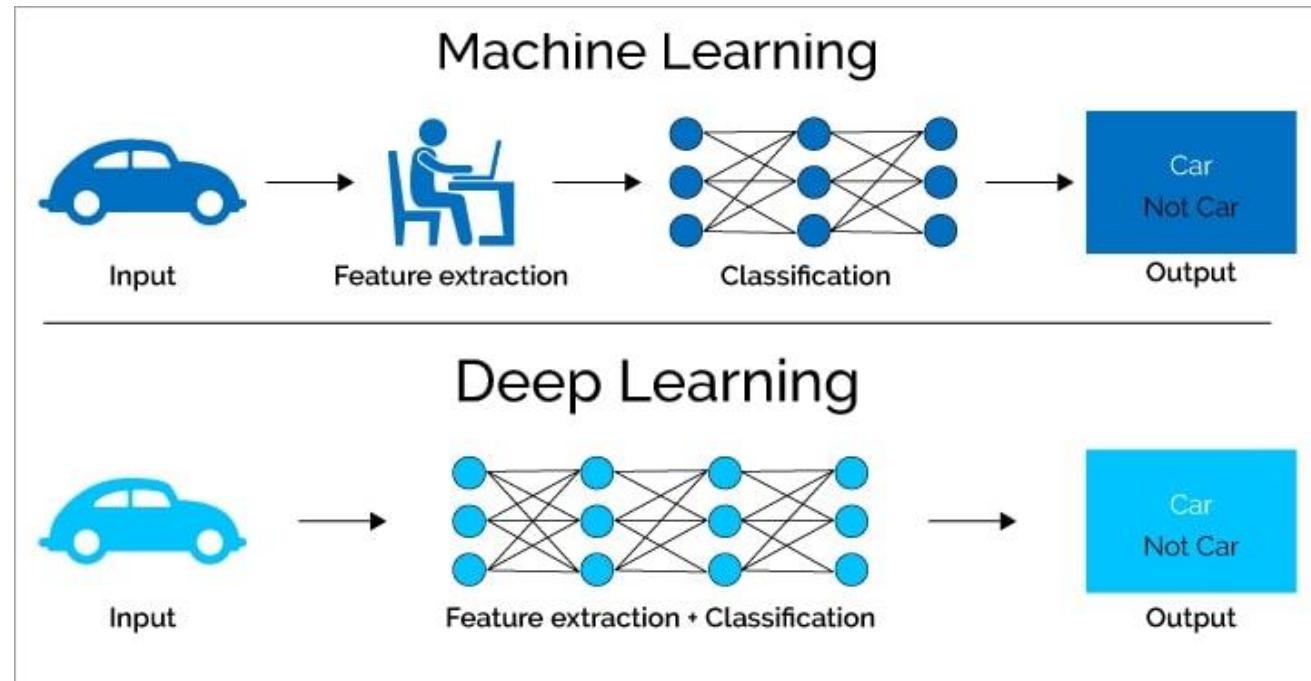
MIT Manipal

Slide -1 of 5

Contents

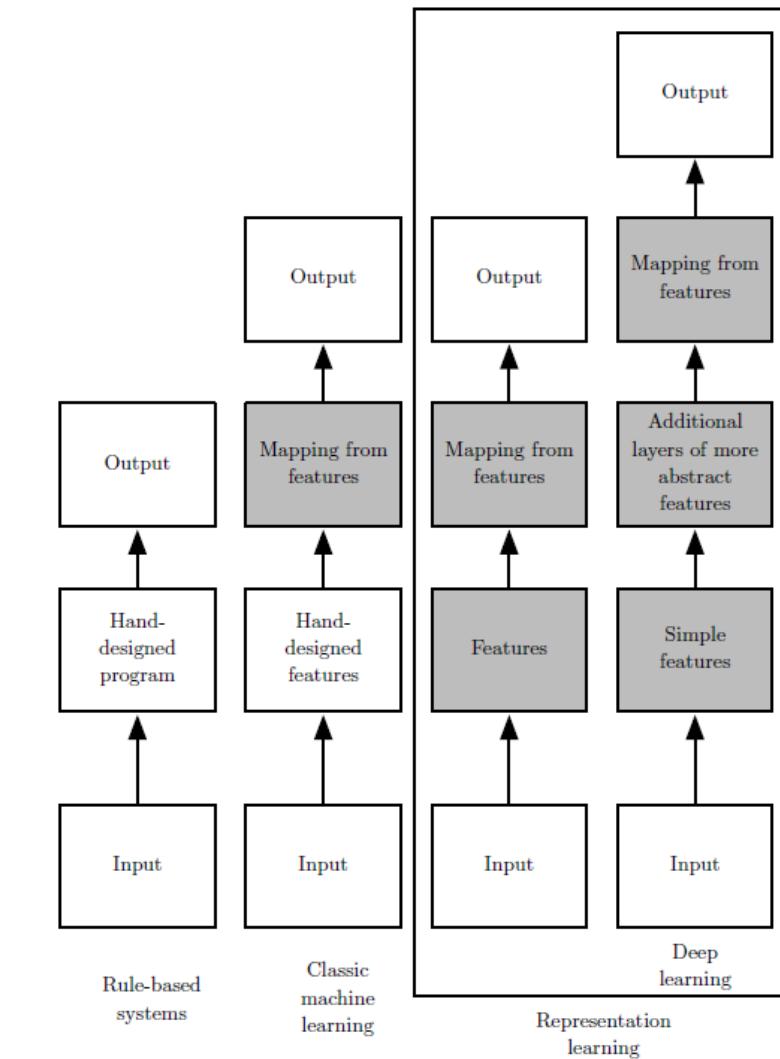
- Shallow Networks
 - Neural Network Representation
 - Back Propagation
 - Vectorization
- Deep Neural Networks
 - Example: Learning XOR
 - Architecture Design
 - Loss Functions
 - Metrics
 - Gradient-Based Learning
 - Optimization
 - Diagnosing Learning Curves
 - Strategies for overfitting
 - Learning rate scheduling

Deep Learning and Machine Learning

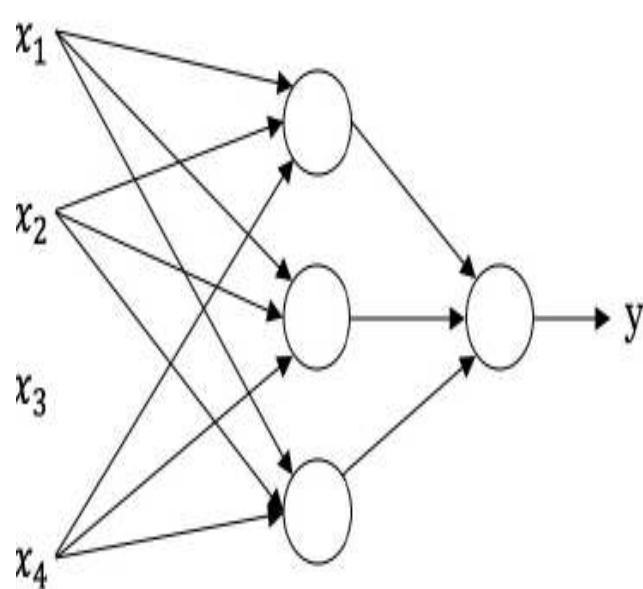


(Source: softwaretestinghelp.com)

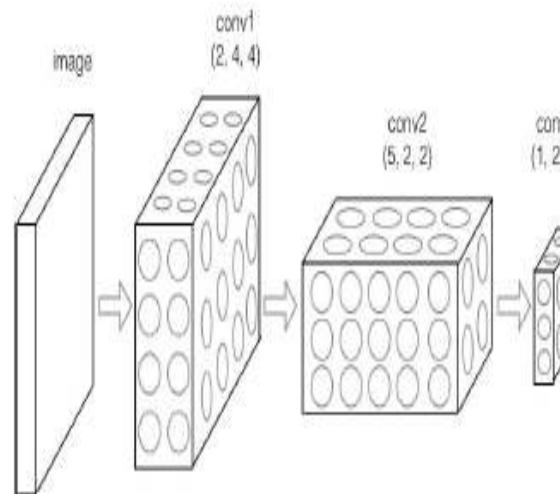
Learning Multiple Components



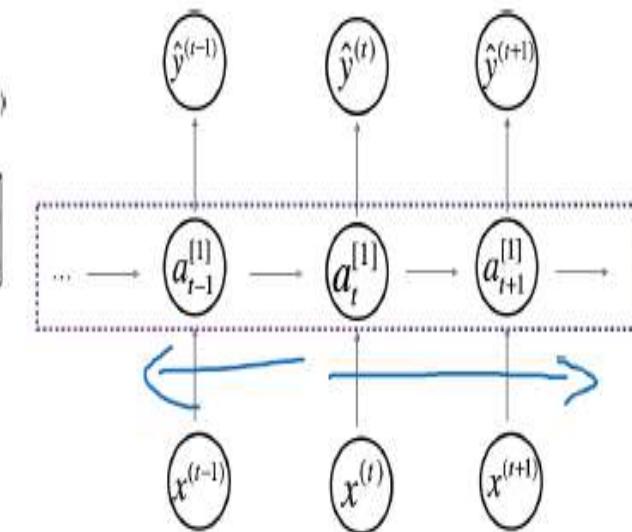
Neural Network Examples



Standard NN



Convolutional NN



Recurrent NN

Structured Data

Size	#bedrooms	...	Price (1000\$s)
2104	3		400
1600	3		330
2400	3		369
...
3000	4		540

Unstructured Data



Audio

Image

User Age	Ad Id	...	Click
41	93242		1
80	93287		0
18	87312		1
...
27	71244		1

Four scores and seven
years ago...

Text

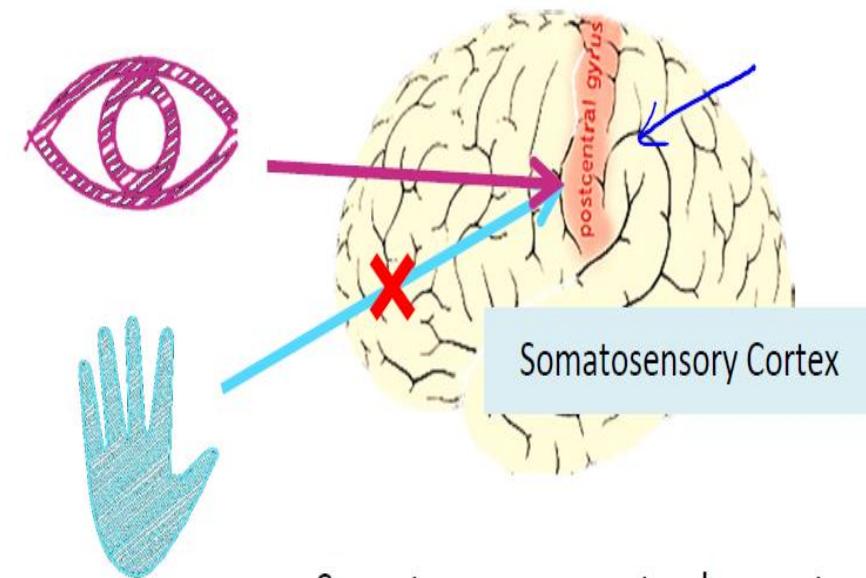
Applications of Deep Learning

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info	Position of other cars	Autonomous driving

Neural Network

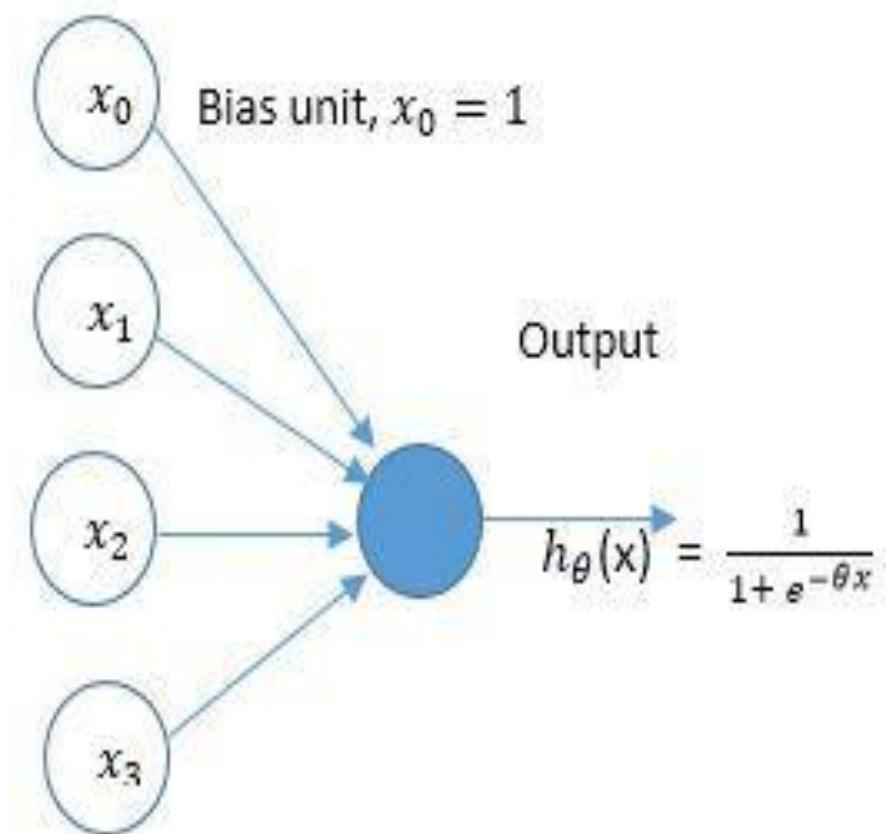
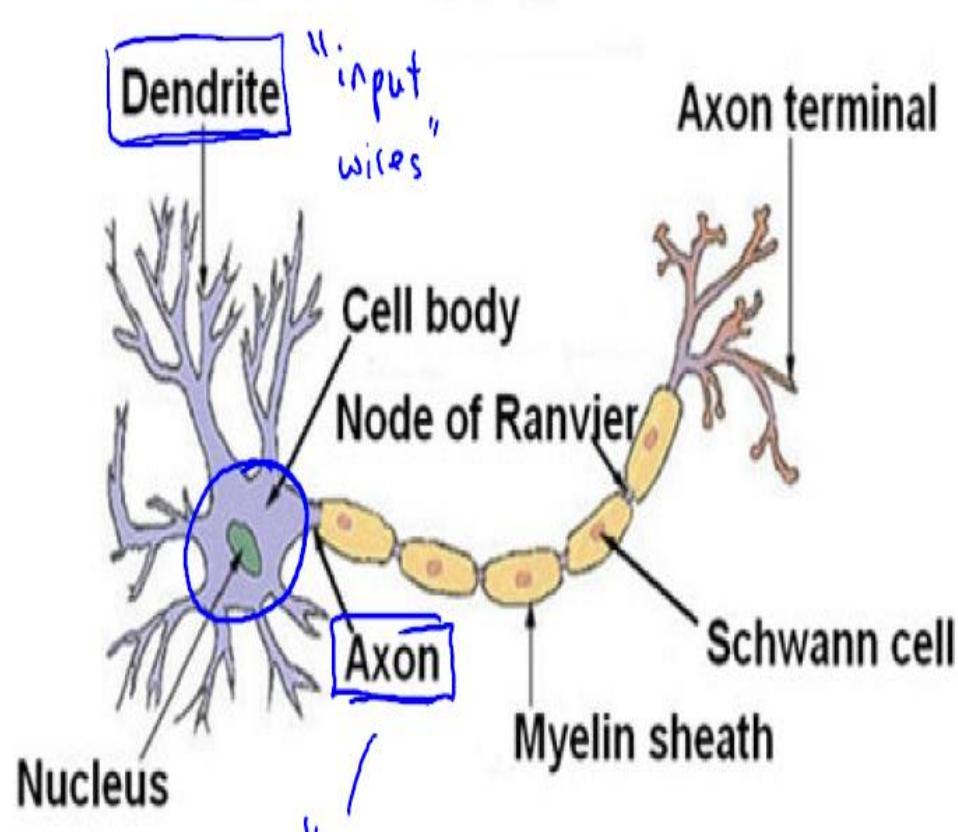
- is a set of connected input/output units in which each connection has
- a weight associated with it.
- During the learning phase, the network learns by adjusting
- the weights so as to be able to predict the correct class label of the input tuples.
- Also referred to as **Connectionist learning**

The “one learning algorithm” hypothesis



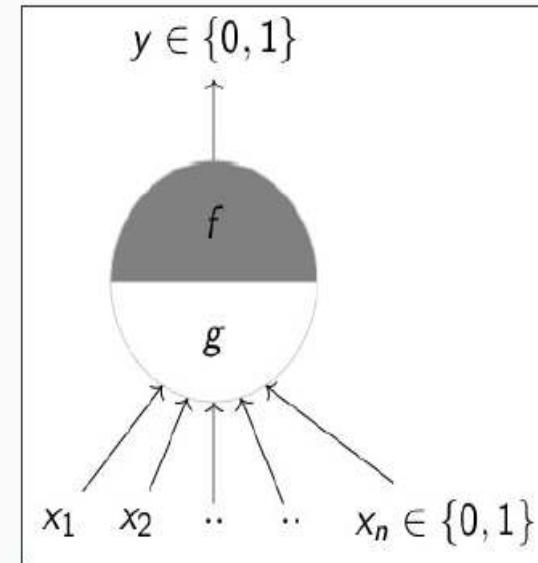
Somatosensory cortex learns to see

Neuron is a computational , logistic unit



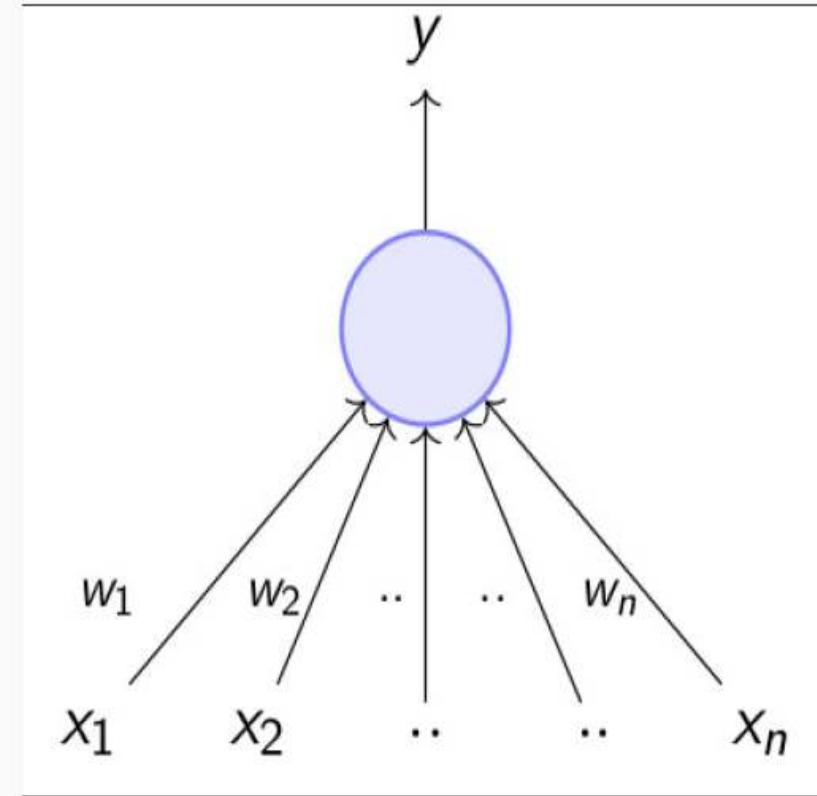
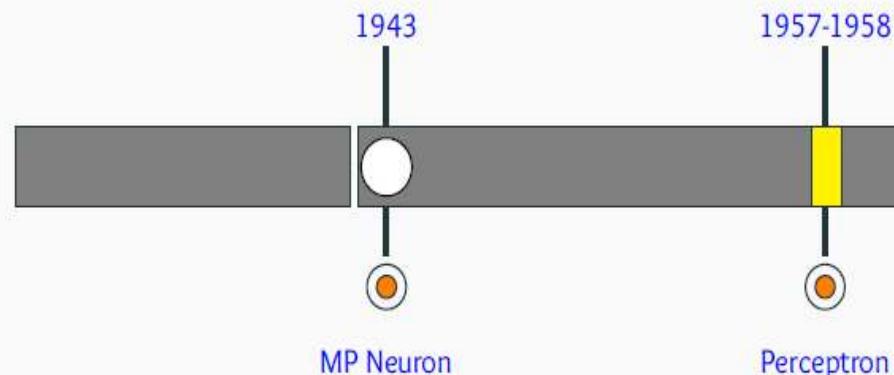
McCulloch Pitts Neuron

McCulloch (neuroscientist) and Pitts (logician) proposed a highly simplified model of the neuron (1943)^[2]



Perceptron

“the perceptron may eventually be able to learn, make decisions, and translate languages” -Frank Rosenblatt

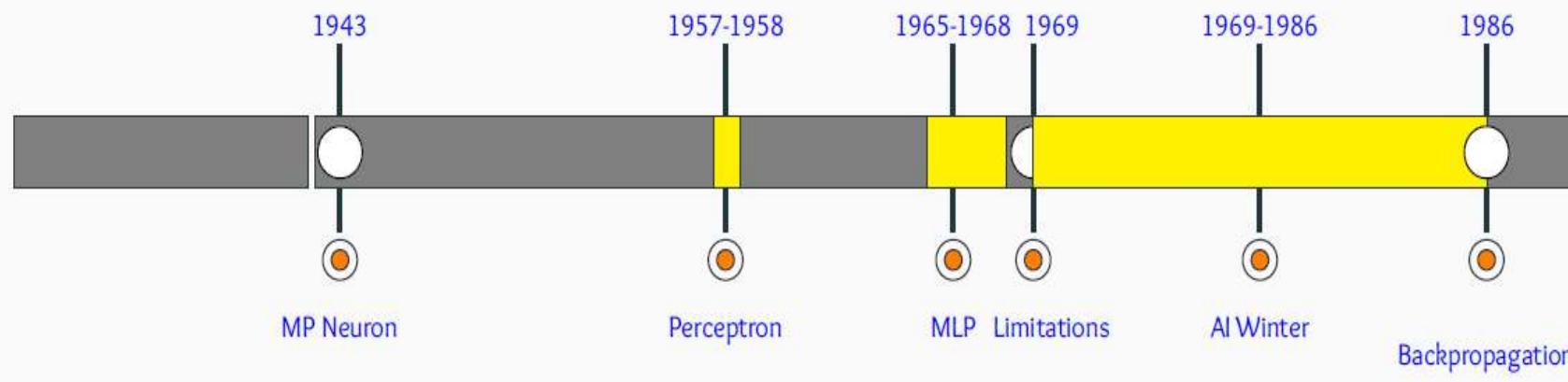
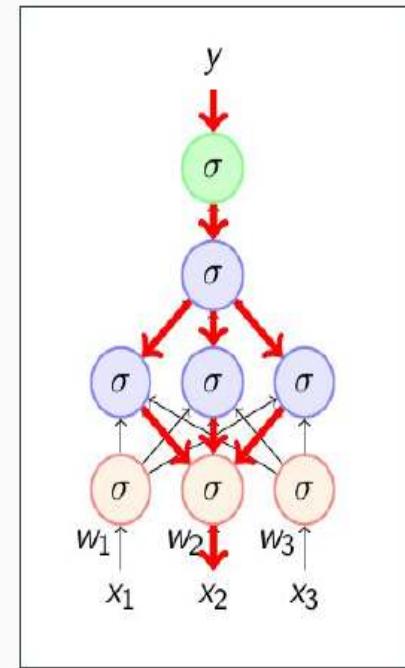


Backpropagation

Discovered and rediscovered several times throughout 1960's and 1970's

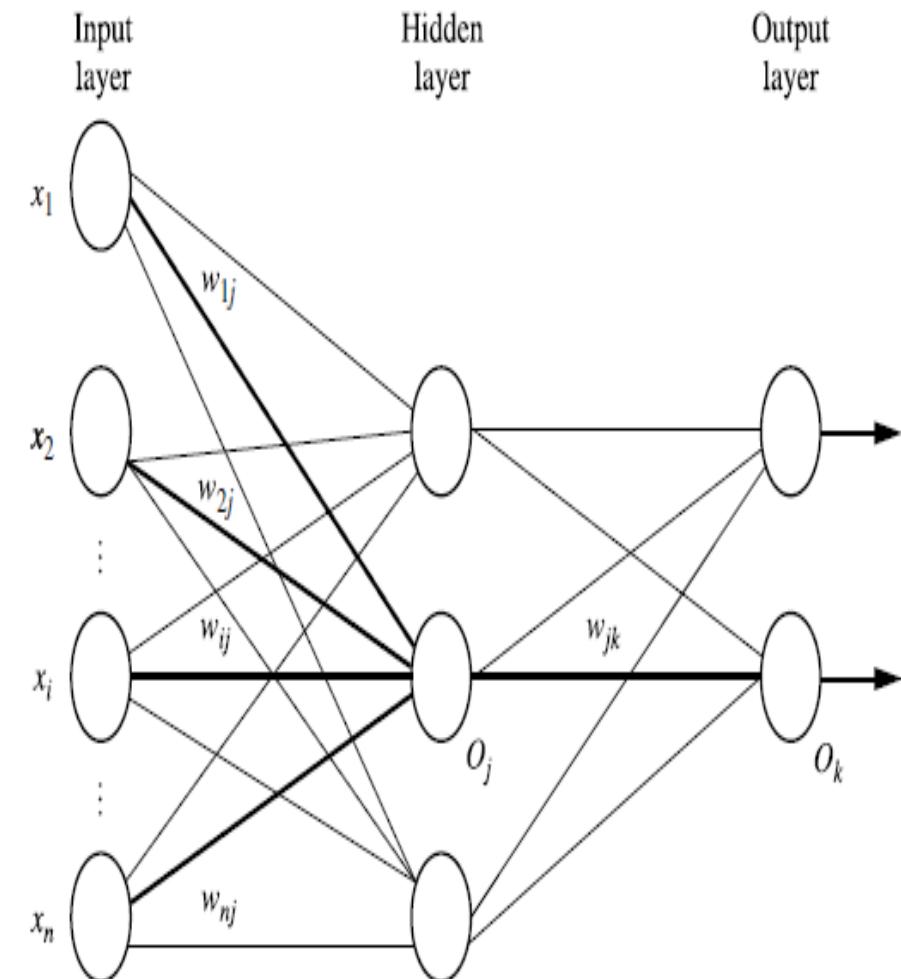
Werbos(1982)^[5] first used it in the context of artificial neural networks

Eventually popularized by the work of Rumelhart et. al. in 1986^[6]



Multilayer Feed-Forward Neural Network

- consists of an *input layer*, one or more *hidden layers*, and an *output layer*
- units in
 - input layer are **input units**
 - hidden and output layer are **neurodes**
- feed-forward network since none of the weights cycles back



Back Propagation Algorithm

```
(1) Initialize all weights and biases in network;  
(2) while terminating condition is not satisfied {  
(3)   for each training tuple X in D {  
(4)     // Propagate the inputs forward:  
(5)     for each input layer unit j {  
(6)        $O_j = I_j$ ; // output of an input unit is its actual input value  
(7)       for each hidden or output layer unit j {  
(8)          $I_j = \sum_i w_{ij} O_i + \theta_j$ ; //compute the net input of unit j with respect to  
             the previous layer, i  
(9)          $O_j = \frac{1}{1+e^{-I_j}}$ ; } // compute the output of each unit j  
(10)      // Backpropagate the errors:  
(11)      for each unit j in the output layer  
(12)         $Err_j = O_j(1 - O_j)(T_j - O_j)$ ; // compute the error  
(13)      for each unit j in the hidden layers, from the last to the first hidden layer  
(14)         $Err_j = O_j(1 - O_j) \sum_k Err_k w_{jk}$ ; // compute the error with respect to  
             the next higher layer, k  
(15)      for each weight  $w_{ij}$  in network {  
(16)         $\Delta w_{ij} = (l)Err_j O_i$ ; // weight increment  
(17)         $w_{ij} = w_{ij} + \Delta w_{ij}$ ; } // weight update  
(18)      for each bias  $\theta_j$  in network {  
(19)         $\Delta \theta_j = (l)Err_j$ ; // bias increment  
(20)         $\theta_j = \theta_j + \Delta \theta_j$ ; } // bias update  
(21)    } }
```

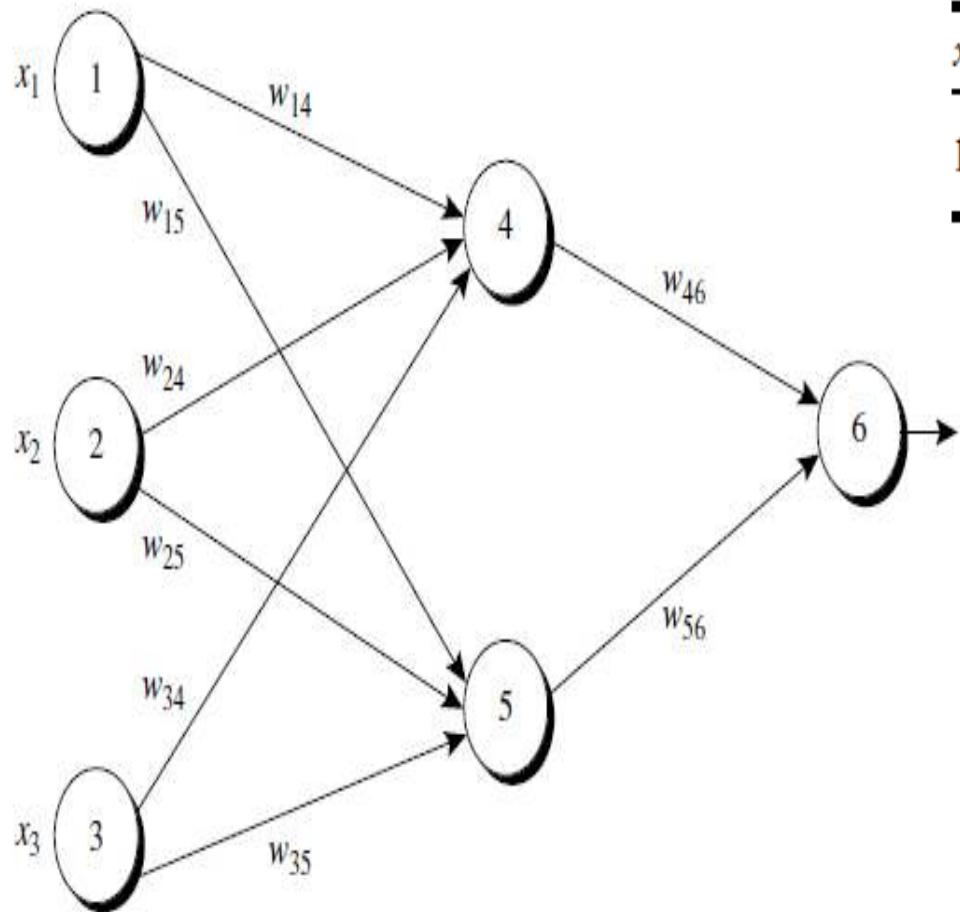
Back Propagation Algorithm

- learns using a gradient descent method to search for a set of weights that fits the training data so as to minimize the mean squared error
- L is the **learning rate**, a constant typically having a value between 0.0 and 1.0.
 - helps avoid getting stuck at a local minimum in decision space and encourages finding the global minimum.
- If L is
 - too small, then learning will occur at a very slow pace.
 - too large, then oscillation between inadequate solutions may occur
- rule of thumb is to set the learning rate to $1=t$
 - where t is the number of iterations through the training set so far
- one iteration through the training set is an **epoch**.
- **Case Updating**
 - updating the weights and biases after the presentation of each tuple.
 - Alternatively,
- **Epoch updating**
 - the weight and bias increments could be accumulated in variables
 - so that the weights and biases are updated after all the tuples in the training set have been presented.

Back Propagation Algorithm

- **Terminating condition:**
 - Training stops when
 - All changes in w_{ij} in the previous epoch are so small as to be below some specified threshold
 - Or The percentage of tuples misclassified in the previous epoch is below some threshold,
 - Or A prespecified number of epochs has expired.
- **The computational efficiency** depends on the
 - time spent training the network.
 - Given $|D|$ tuples and w weights, each epoch requires $O(|D| * w)$ time.
 - In the worst-case scenario, the number of epochs can be exponential in n , the number of inputs.
 - In practice, the time required for the networks to converge is highly variable.

Example



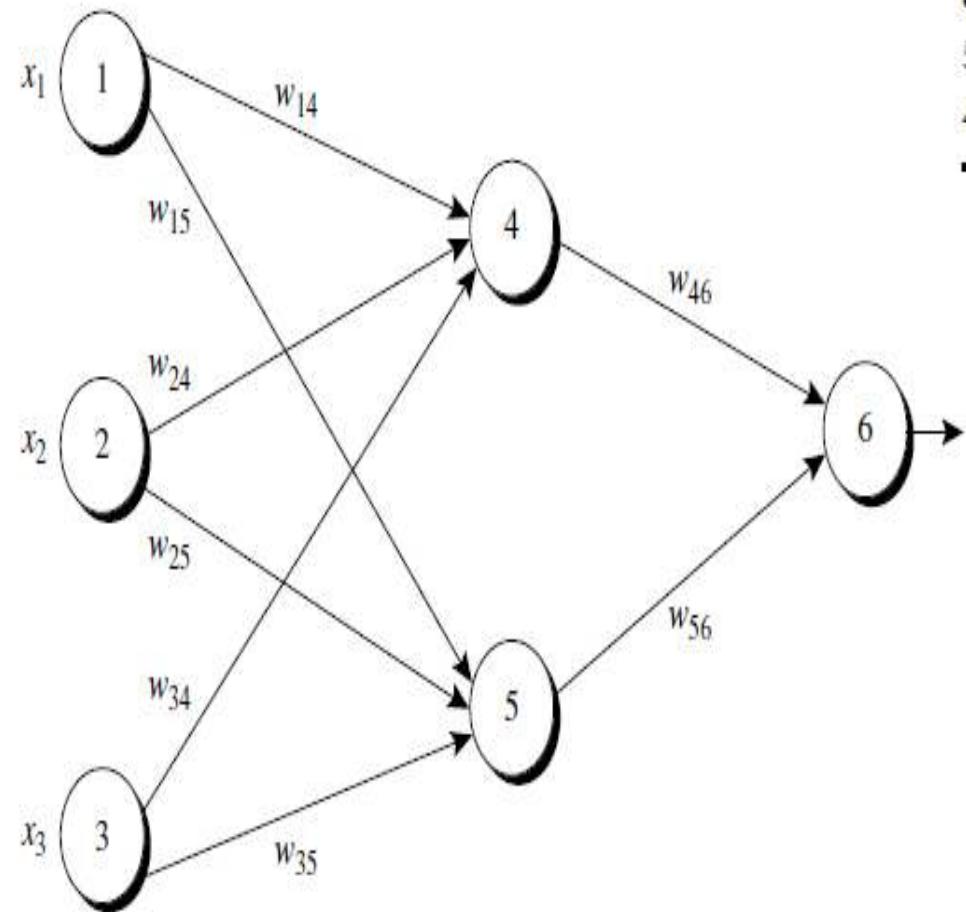
Initial Input, Weight, and Bias Values

x_1	x_2	x_3	w_{14}	w_{15}	w_{24}	w_{25}	w_{34}	w_{35}	w_{46}	w_{56}	θ_4	θ_5	θ_6
1	0	1	0.2	-0.3	0.4	0.1	-0.5	0.2	-0.3	-0.2	-0.4	0.2	0.1

Net Input and Output Calculations

Unit, j	Net Input, I_j	Output, O_j
4	$0.2 + 0 - 0.5 - 0.4 = -0.7$	$1/(1 + e^{-0.7}) = 0.332$
5	$-0.3 + 0 + 0.2 + 0.2 = 0.1$	$1/(1 + e^{-0.1}) = 0.525$
6	$(-0.3)(0.332) - (0.2)(0.525) + 0.1 = -0.105$	$1/(1 + e^{0.105}) = 0.474$

Example



Calculation of the Error at Each Node

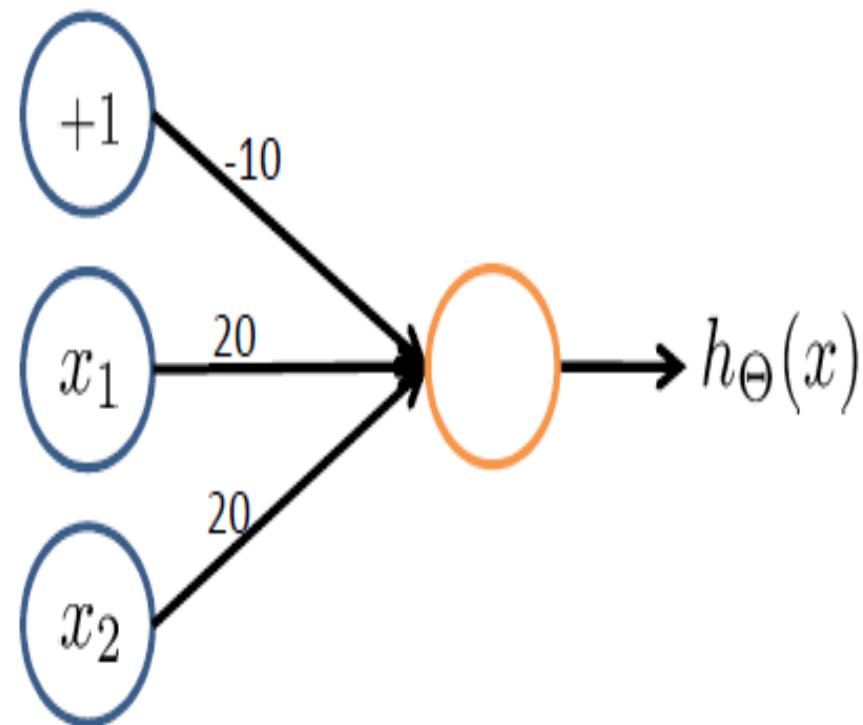
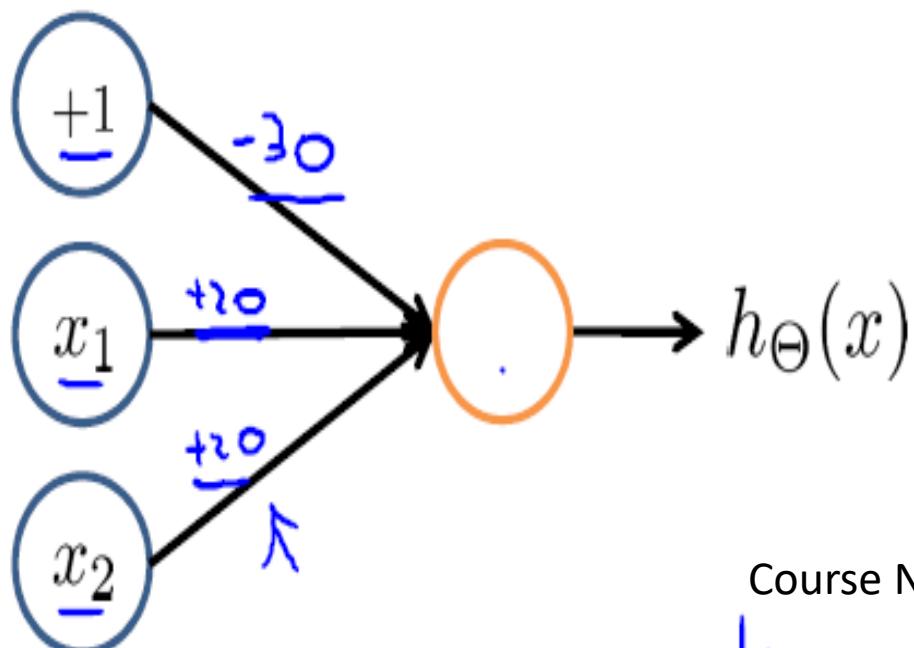
<i>Unit, j</i>	<i>Err_j</i>
6	$(0.474)(1 - 0.474)(1 - 0.474) = 0.1311$
5	$(0.525)(1 - 0.525)(0.1311)(-0.2) = -0.0065$
4	$(0.332)(1 - 0.332)(0.1311)(-0.3) = -0.0087$

Calculations for Weight and Bias Updating

<i>Weight or Bias</i>	<i>New Value</i>
w_{46}	$-0.3 + (0.9)(0.1311)(0.332) = -0.261$
w_{56}	$-0.2 + (0.9)(0.1311)(0.525) = -0.138$
w_{14}	$0.2 + (0.9)(-0.0087)(1) = 0.192$
w_{15}	$-0.3 + (0.9)(-0.0065)(1) = -0.306$
w_{24}	$0.4 + (0.9)(-0.0087)(0) = 0.4$
w_{25}	$0.1 + (0.9)(-0.0065)(0) = 0.1$
w_{34}	$-0.5 + (0.9)(-0.0087)(1) = -0.508$
w_{35}	$0.2 + (0.9)(-0.0065)(1) = 0.194$
θ_6	$0.1 + (0.9)(0.1311) = 0.218$
θ_5	$0.2 + (0.9)(-0.0065) = 0.194$
θ_4	$-0.4 + (0.9)(-0.0087) = -0.408$

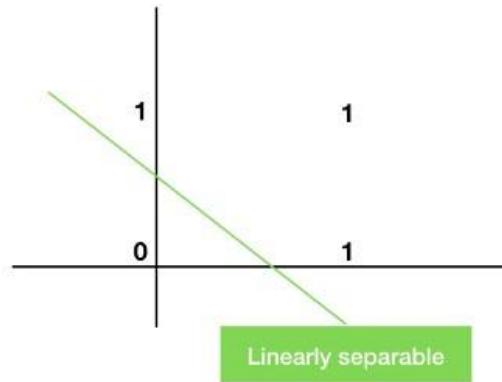
Simple AND and Simple OR operations

- $x_1, x_2 \in \{0, 1\}$
- $y = x_1 \text{ AND } x_2$



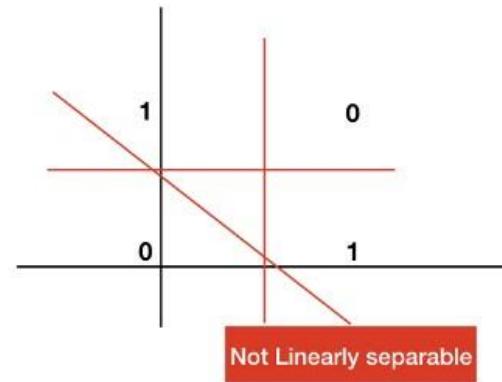
XOR Problem - Perceptron Learning

Inclusive-OR



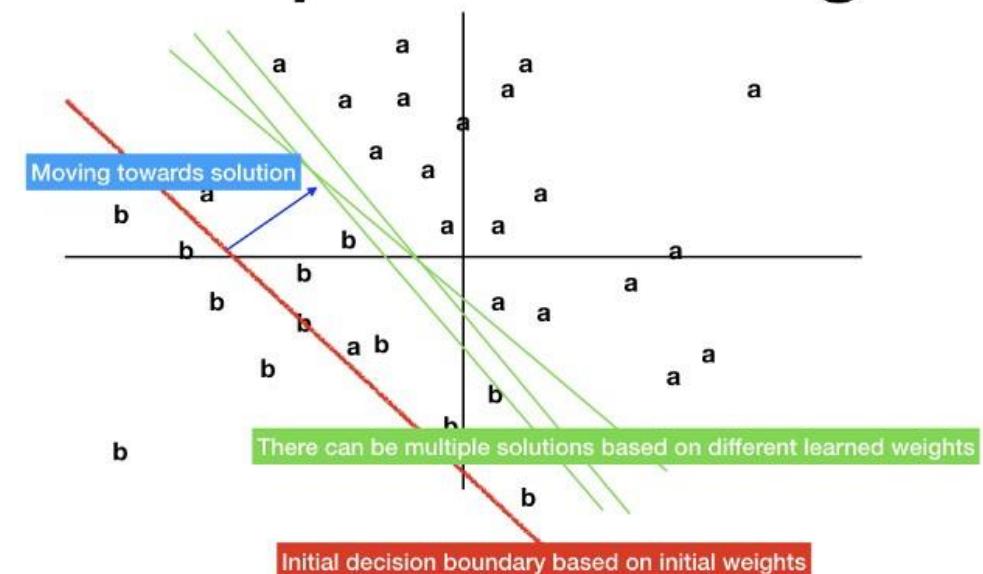
a	b	c
0	0	0
0	1	1
1	0	1
1	1	1

Exclusive-OR



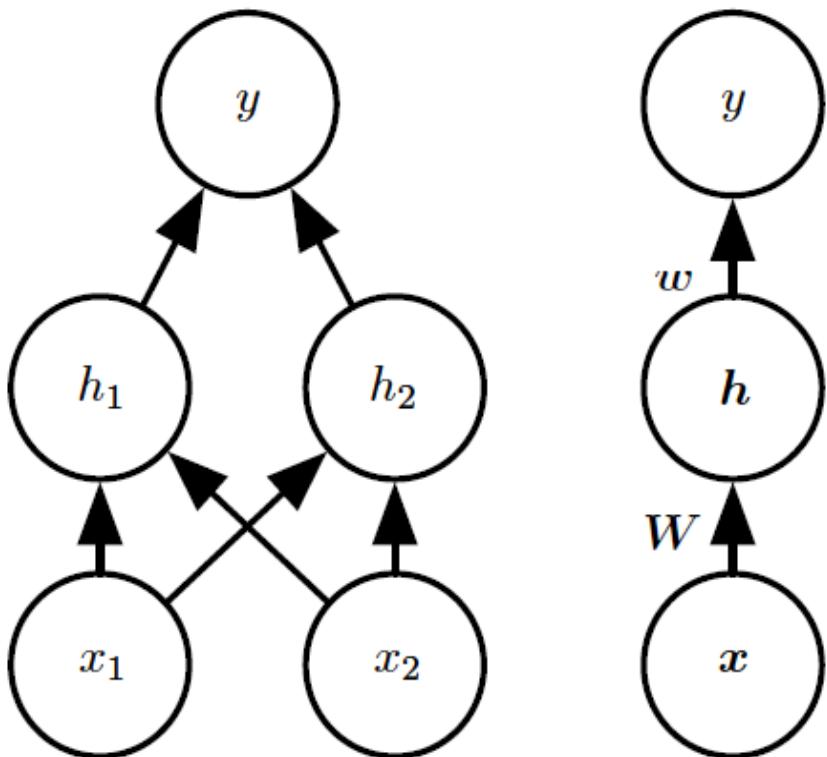
a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

Perceptron Learning



XOR Problem

Network Diagram

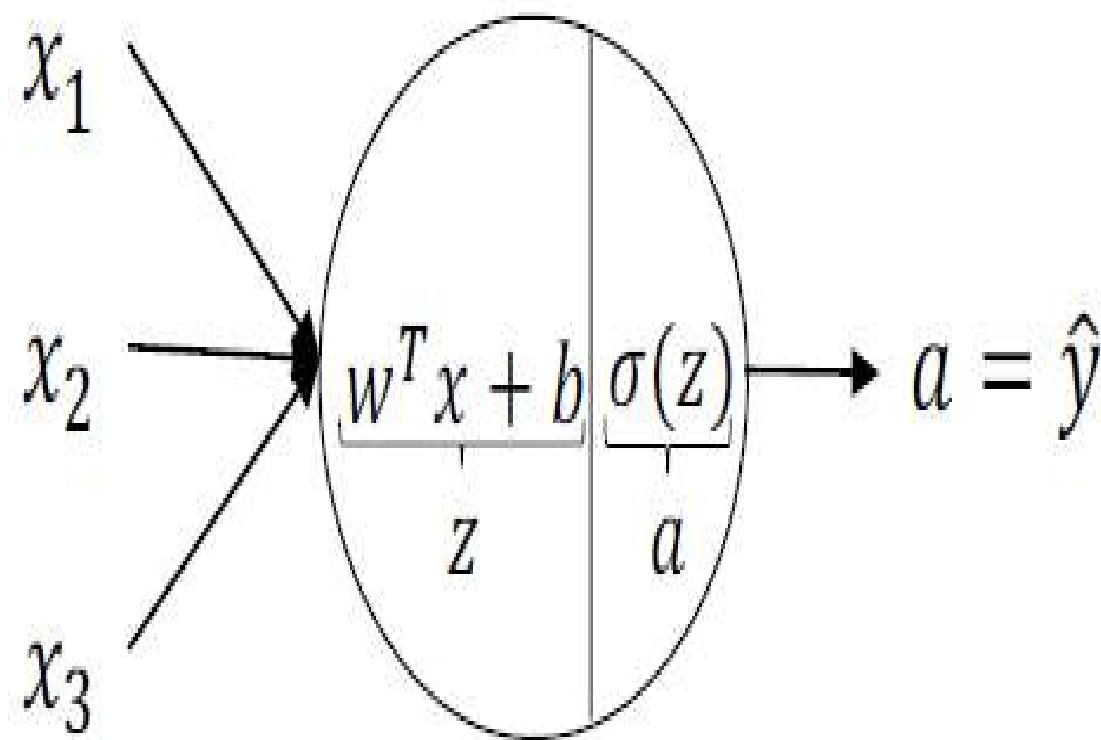


Solving XOR

$$f(x; \mathbf{W}, c, \mathbf{w}, b) = \mathbf{w}^\top \max\{0, \mathbf{W}^\top x + c\} + b.$$

$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix}$$
$$c = \begin{bmatrix} 0 \\ -1 \end{bmatrix}, \quad b = 0$$
$$\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

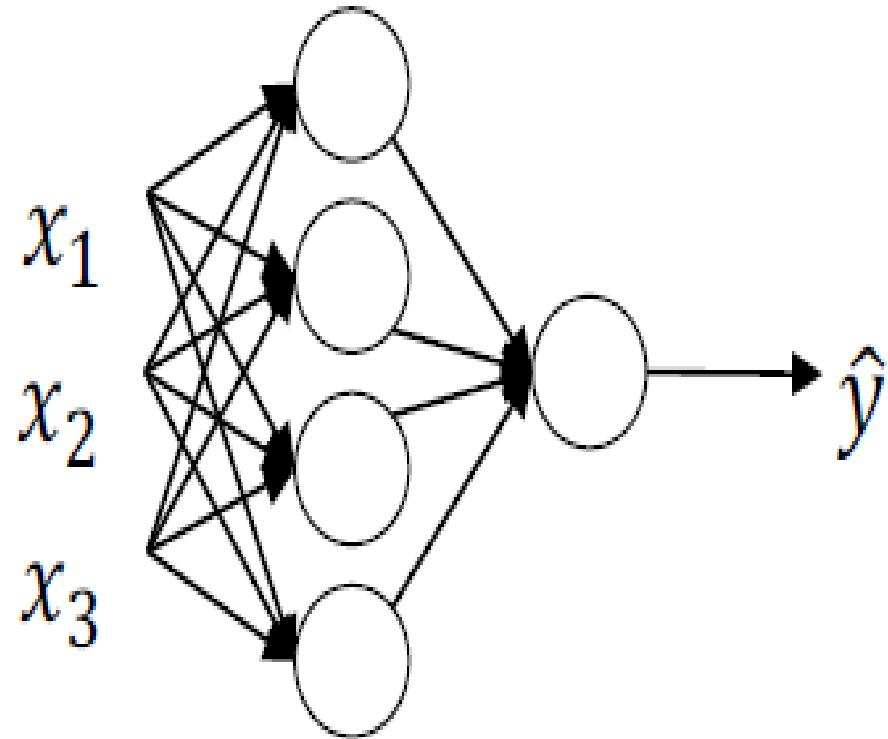
Neural Network Representation



$$z = w^T x + b$$

$$a = \sigma(z)$$

Neural Network Representation



$$z_1^{[1]} = w_1^{[1]T}x + b_1^{[1]}, a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T}x + b_2^{[1]}, a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T}x + b_3^{[1]}, a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T}x + b_4^{[1]}, a_4^{[1]} = \sigma(z_4^{[1]})$$

$$Z^{[1]} = X^{[1]T}X + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

The Shallow Neural Network

Vectorizing across multiple examples

for i = 1 to m:

$$Z^{[1]} = W^{[1]T} X + b^{[1]}$$

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$



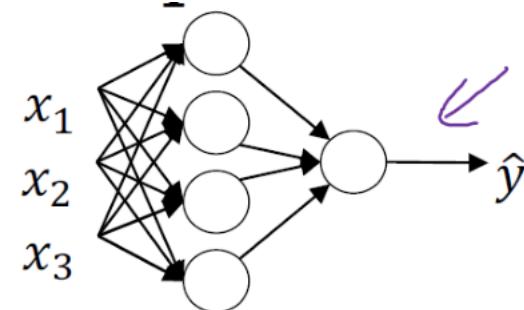
$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]T} A^{[1]} + b^{[2]}$$

$$\hat{y} = A^{[2]} = \sigma(Z^{[2]})$$

The Shallow Neural Network

Vectorizing across multiple examples



$$X = \begin{bmatrix} | & | & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & | \end{bmatrix}$$

$$\underline{A^{[1]}} = \begin{bmatrix} | & | & | \\ a^{1} & a^{[1](2)} & \dots & a^{[1](m)} \\ | & | & | \end{bmatrix}$$

for $i = 1$ to m

$$\rightarrow z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$\rightarrow a^{[1](i)} = \sigma(z^{[1](i)})$$

$$\rightarrow z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$\rightarrow a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]}X + b^{[1]} \quad \leftarrow \quad w^{[1]}A^{[1]} + b^{[1]}$$

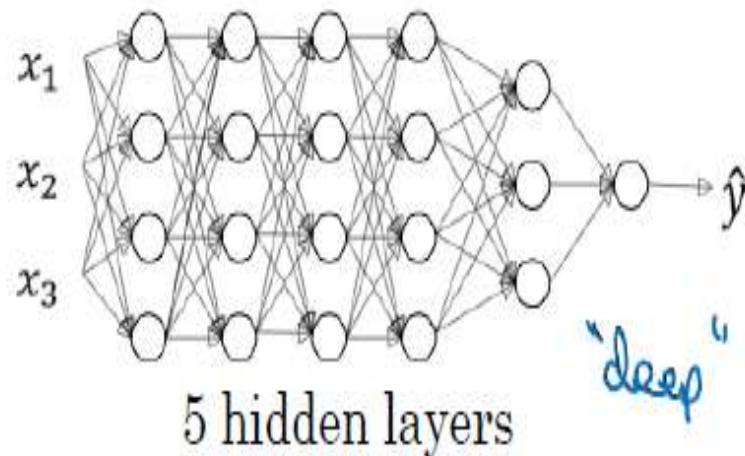
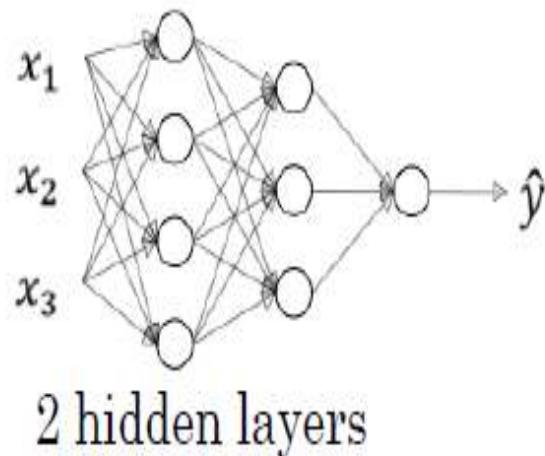
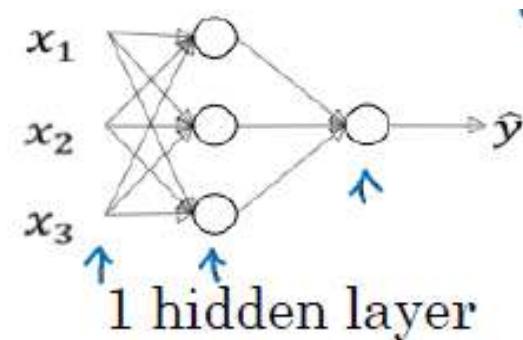
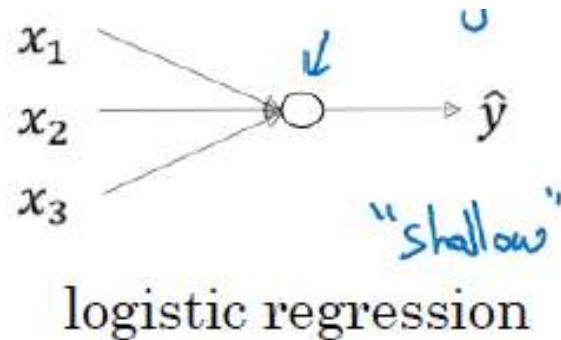
$$A^{[1]} = \sigma(Z^{[1]})$$

$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

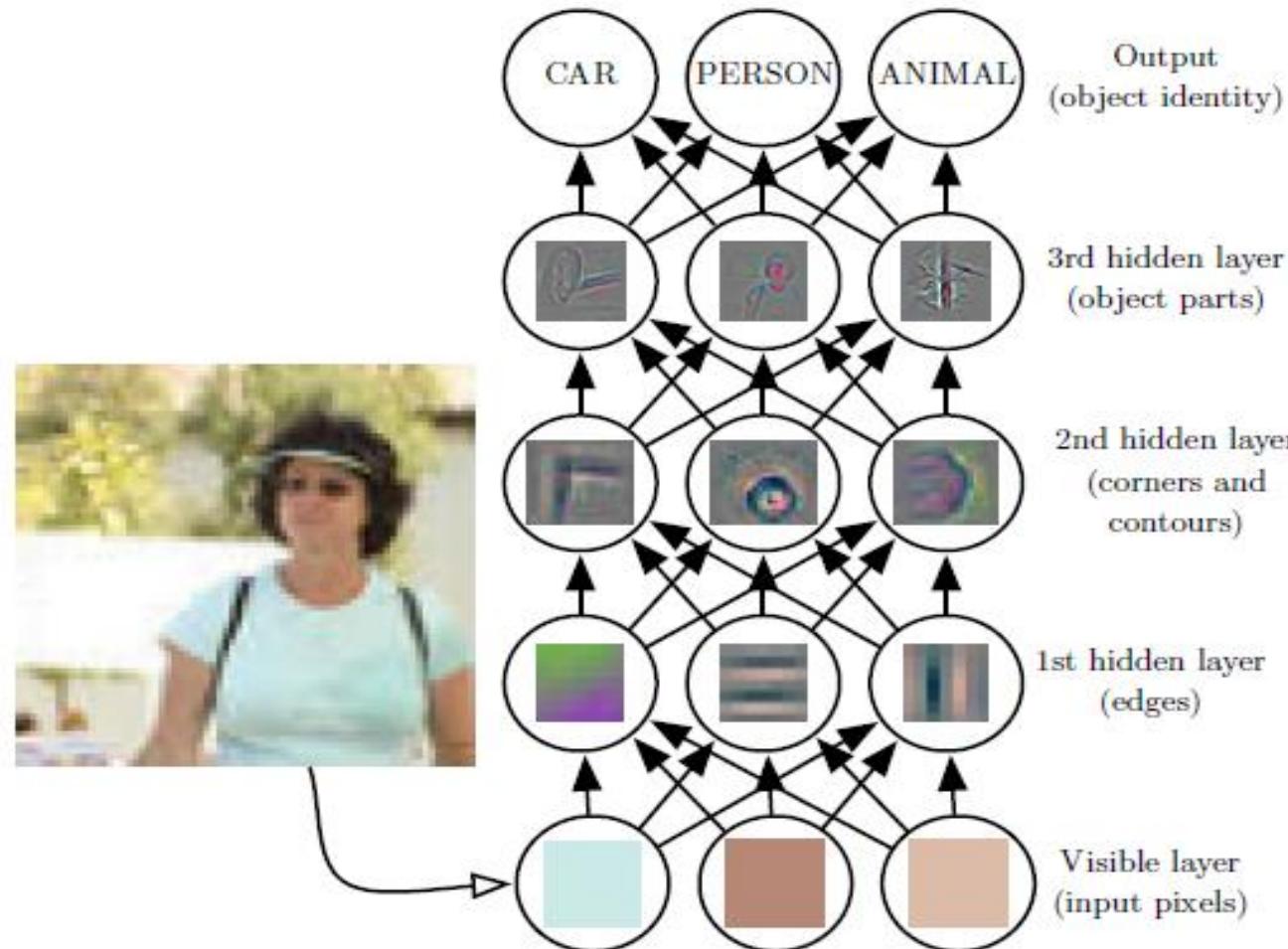
$$A^{[2]} = \sigma(Z^{[2]})$$

Andrew Ng

Deep vs Shallow Network



Depth: Repeated Composition



Neural Network learning as optimization

- Cannot calculate the perfect weights for a neural network since there are too many unknowns
- Instead, the problem of learning is as a search or optimization problem
- An algorithm is used to navigate the space of possible sets of weights the model may use in order to make good or good enough predictions
- **Objective of optimizer** is to minimize the loss function or the error term.
- **Loss function**
 - gives the difference between observed value from the predicted value.
 - must be
 - **Continuous**
 - **Differentiable at each point** (allows the use of gradient-based optimization)
 - To minimize the loss generated from any model compute
 - The magnitude that is by how much amount to decrease or increase, and
 - direction in which to move

Machine Learning Setup

Data: $\{x_i, y_i\}_{i=1}^n$

Model: Our approximation of the relation between x and y . For example,

$$\hat{y} = \frac{1}{1 + e^{-(w^T x)}}$$

or $\hat{y} = w^T x$

or $\hat{y} = x^T W x$

or just about any function

Parameters: In all the above cases, w is a parameter which needs to be learned from the data

Learning algorithm: An algorithm for learning the parameters (w) of the model (for example, perceptron learning algorithm, gradient descent, etc.)

Objective/Loss/Error function: To guide the learning algorithm

Activation Functions

- To make the network robust use of **Activation or Transfer functions**.
- Activations functions introduce non-linear properties in the neural networks.
- A good Activation function has the following properties:
- **Monotonic Function:**
 - should be either entirely non-increasing or non-decreasing.
 - If not monotonic then increasing the neuron's weight might cause it to have less influence on reducing the error of the cost function.
- **Differential:**
 - mathematically means the change in y with respect to change in x .
 - should be differential because we want to calculate the change in error with respect to given weights at the time of gradient descent.
- **Quickly Converging:**
 - Should reach its desired value fast.

Gradient Descent Rule

The direction u that we intend to move in should be at 180° w.r.t. the gradient

- In other words, move in a direction opposite to the gradient

Parameter Update Equations

$$w_{t+1} = w_t - \eta \nabla w_t$$

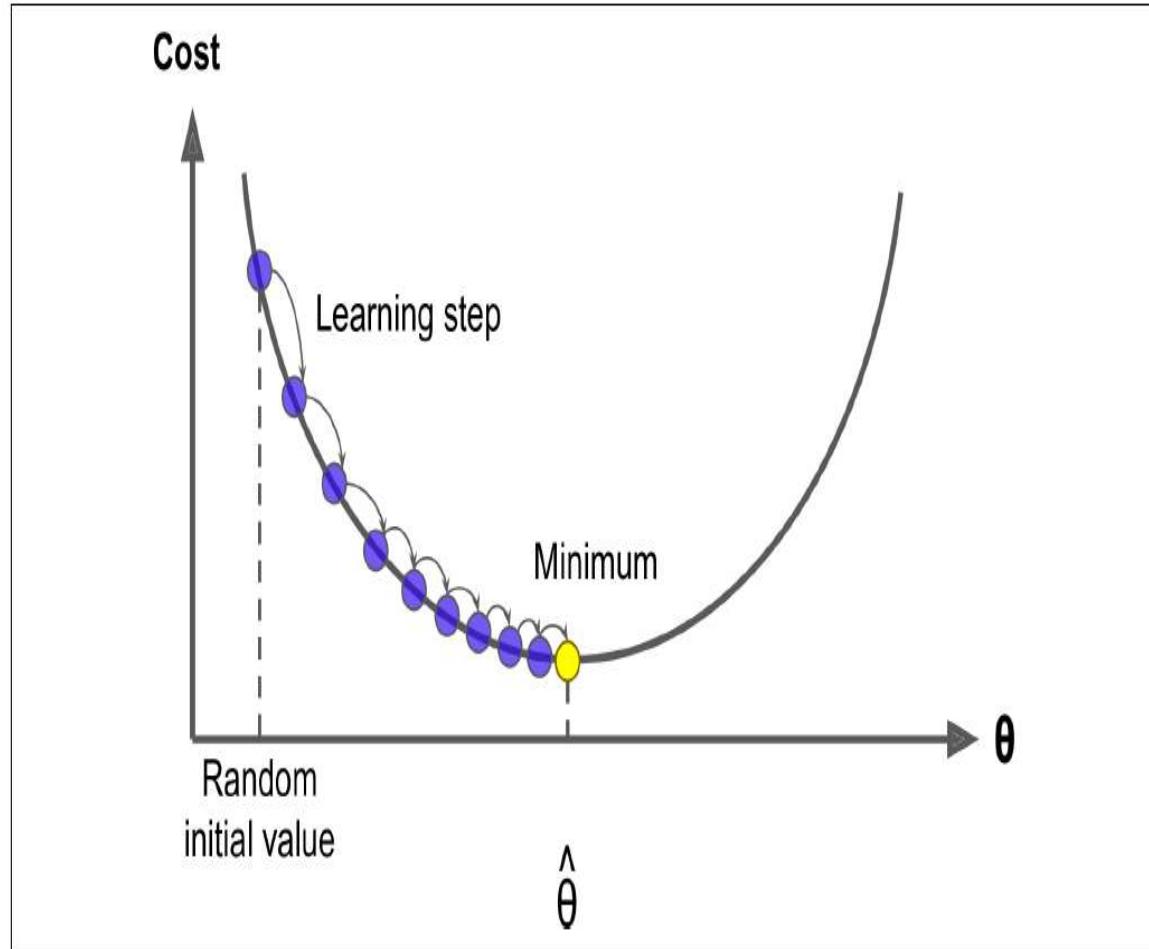
$$b_{t+1} = b_t - \eta \nabla b_t$$

where, $\nabla w_t = \frac{\partial \mathcal{L}(w, b)}{\partial w}$ at $w = w_t, b = b_t$, $\nabla b = \frac{\partial \mathcal{L}(w, b)}{\partial b}$ at $w =$

Algorithm: gradient_descent()

```
t ← 0;  
max_iterations ← 1000;  
while  $t < max\_iterations$  do  
     $w_{t+1} \leftarrow w_t - \eta \nabla w_t;$   
     $b_{t+1} \leftarrow b_t - \eta \nabla b_t;$   
     $t \leftarrow t + 1;$   
end
```

Gradient Descent



- Generic Optimization Algorithm
- Starts with random values
- Improves gradually , in an attempt to decrease loss function
- Until algorithm converges to minimum
- Learning Rate-
 - Hyperparameter
 - Indicates size of steps

Gradient Descent & Learning Rate

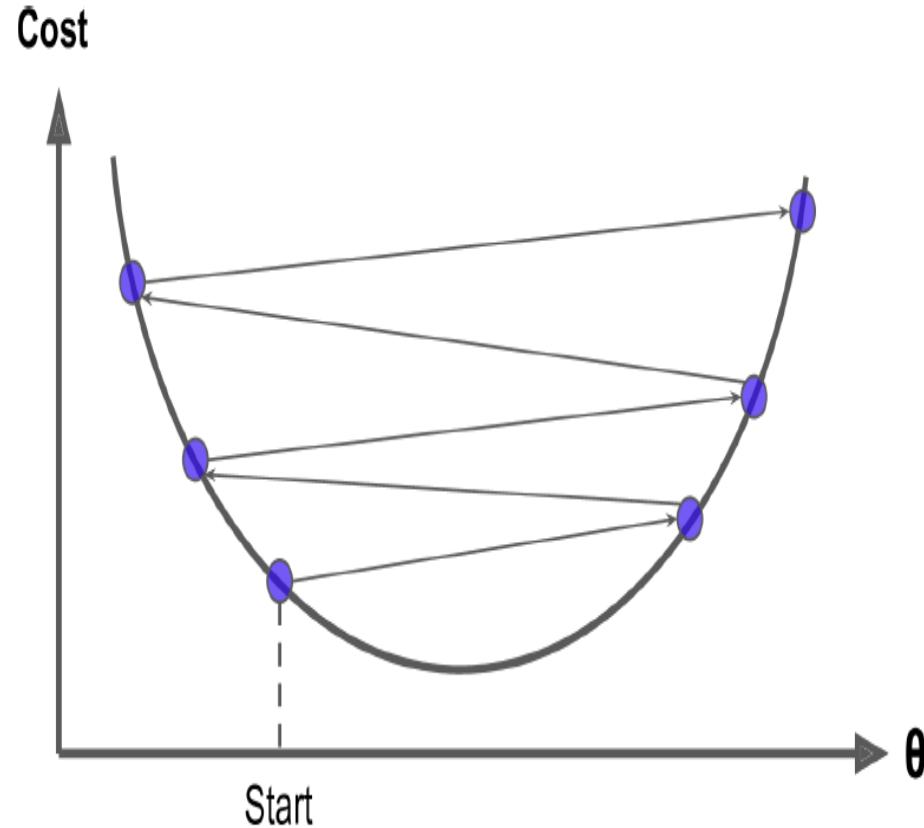


Figure 4-5. Learning rate too large

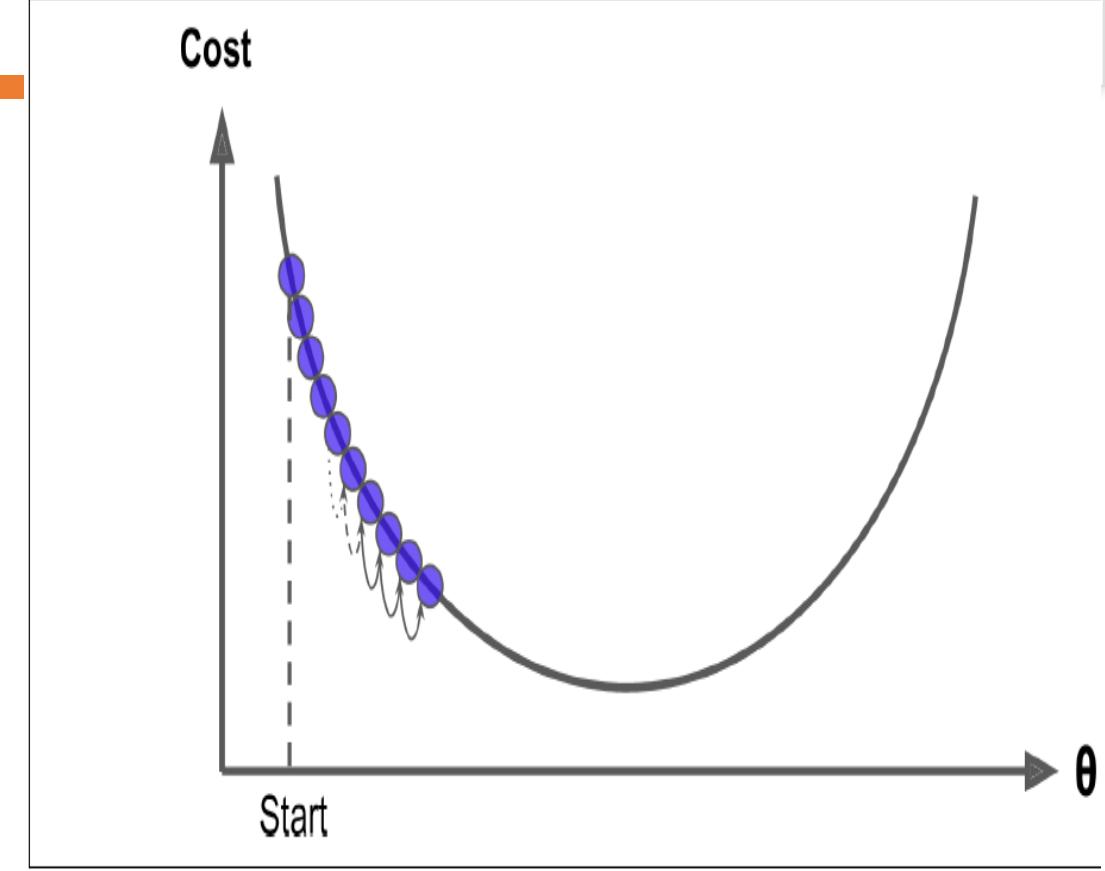
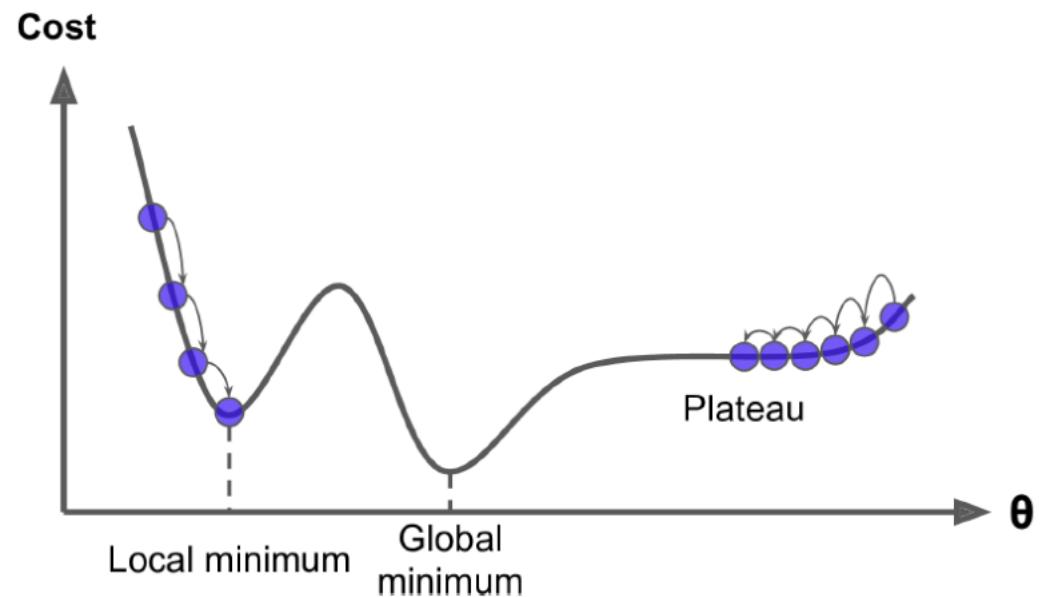


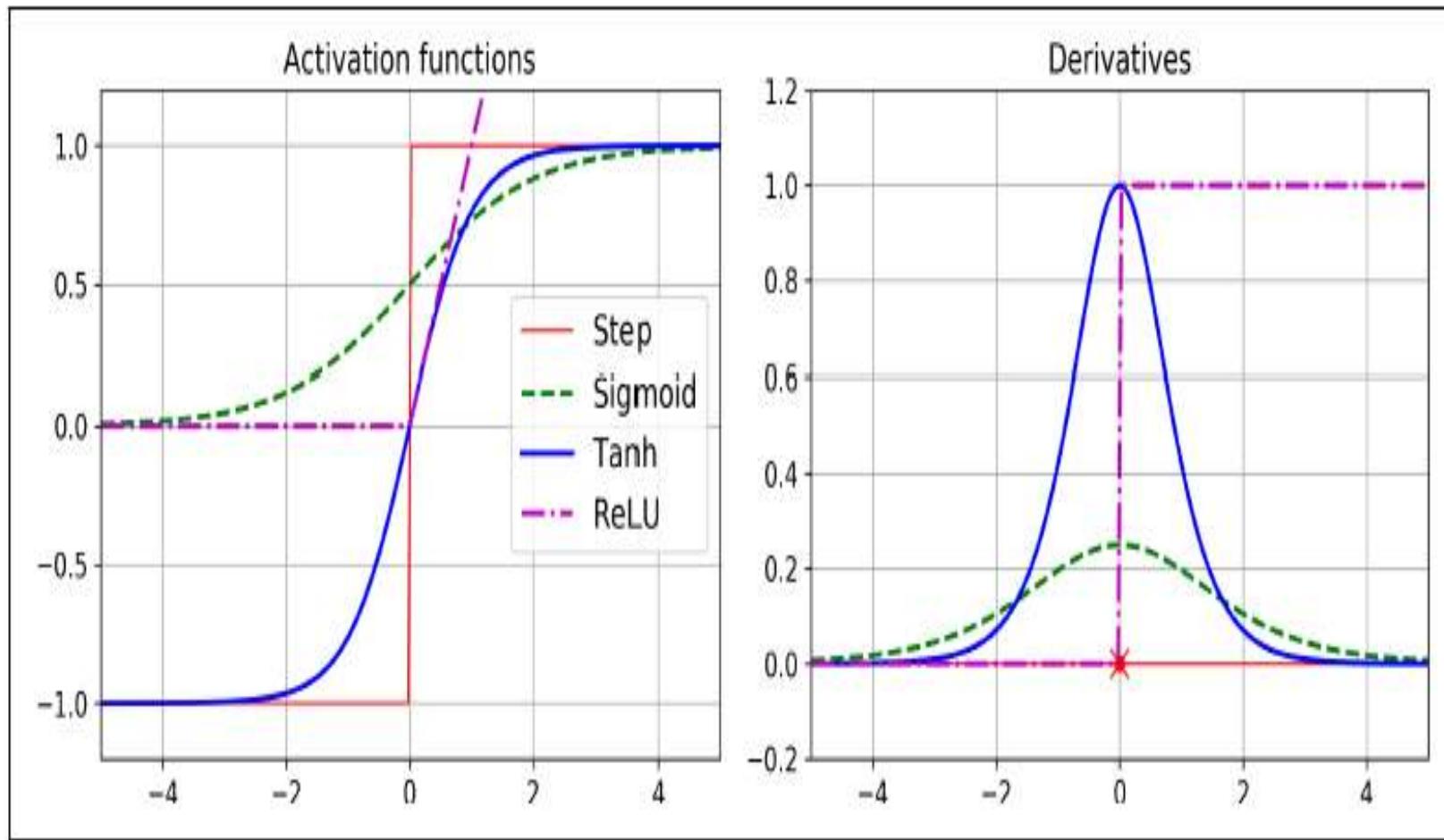
Figure 4-4. Learning rate too small

Gradient Descent Pitfalls

- If it starts on left can reach local minimum
- If it stars from right can hit plateau
- So pick Cost Functions which are convex functions
 - Has no local minimum
 - Continuous function with slope that does not change abruptly
- Then Gradient Descent will approach close to global minimum



Activation Functions and their derivatives

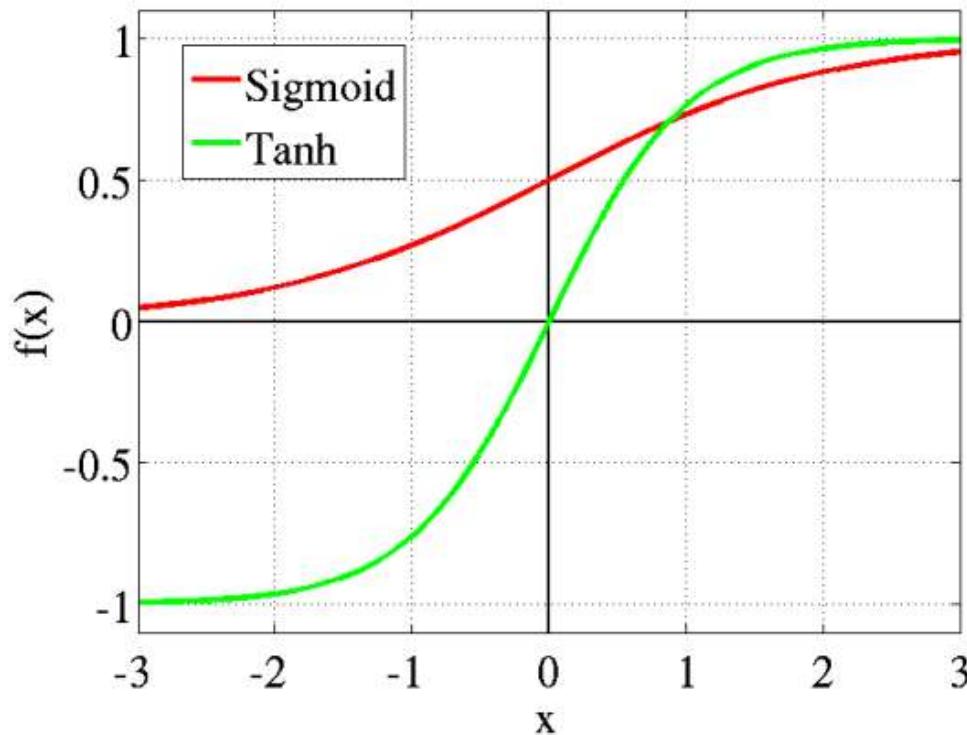


Aurelien Geron, "Hands-On Machine Learning with Scikit-Learn , Keras & Tensorflow, O'Reilly Publications

Rohini R Rao & Abhilash Pai, Dept of Data Science and CA

35

Activation Functions – Sigmoid vs Tanh



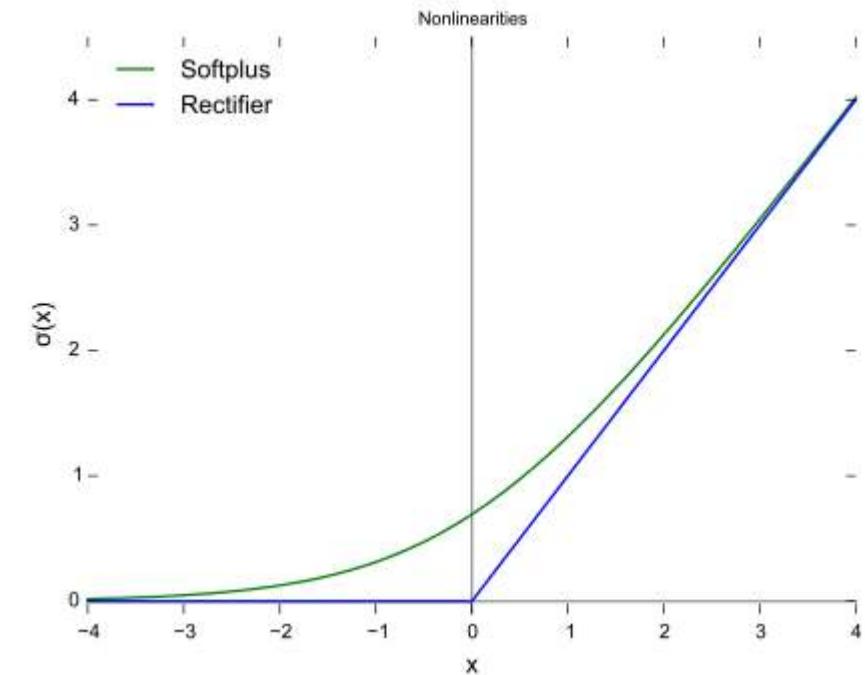
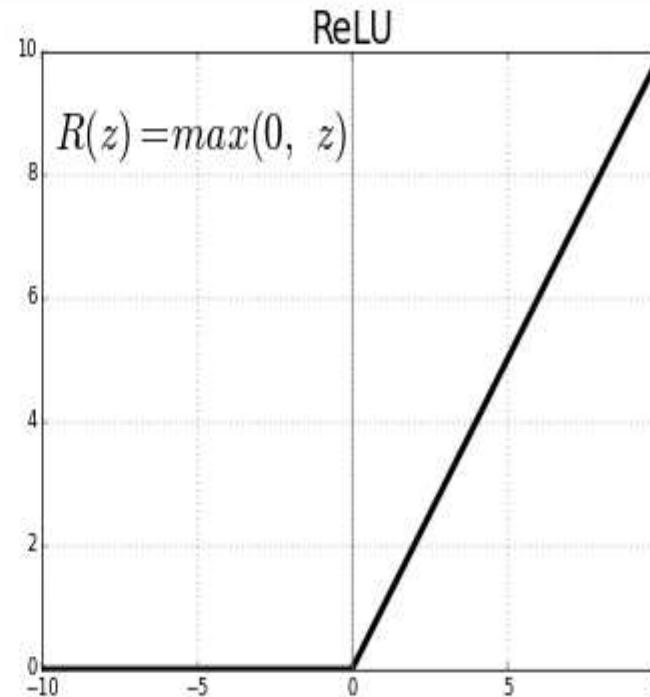
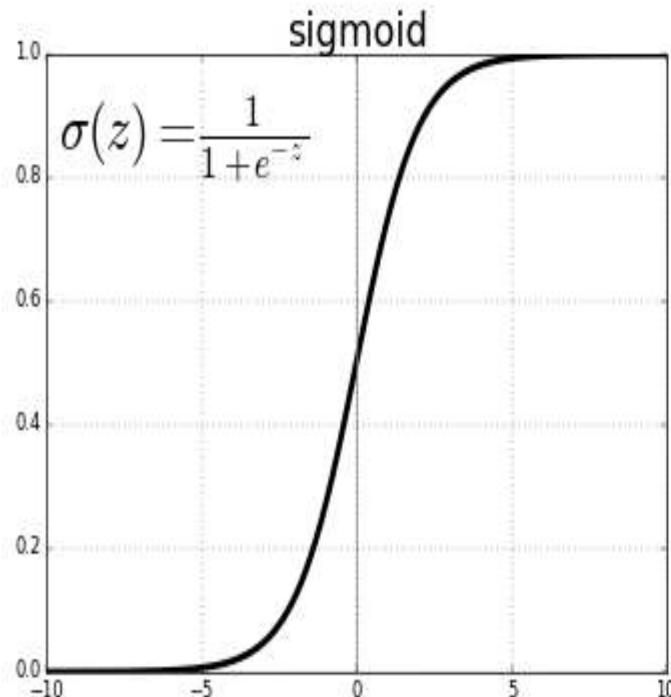
- **Sigmoid**

- $s(x) = 1/(1 + e^{-x})$ where $e \approx 2.71$ is the base of the natural logarithm
- to predict the probability
- between the range of **0 and 1**, sigmoid is the right choice.
- is **differentiable**-, we can find the slope of the sigmoid curve at any two points.
- The function is **monotonic** but function's derivative is not.
- can cause a neural network to get stuck at the training time.

- **Tanh**

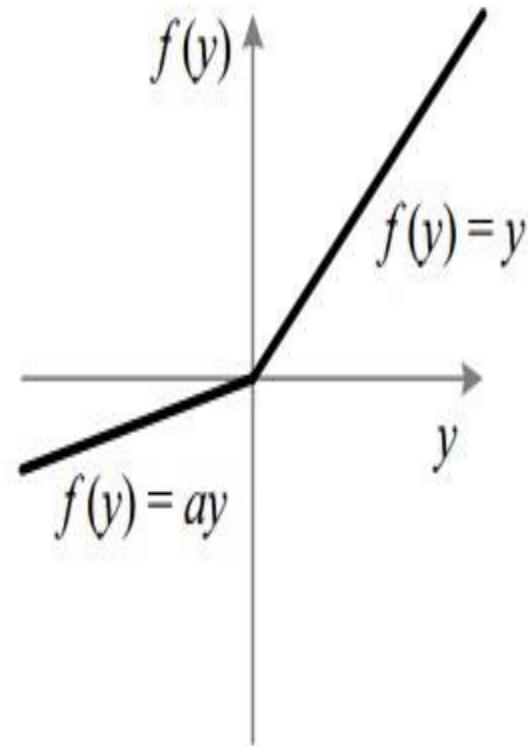
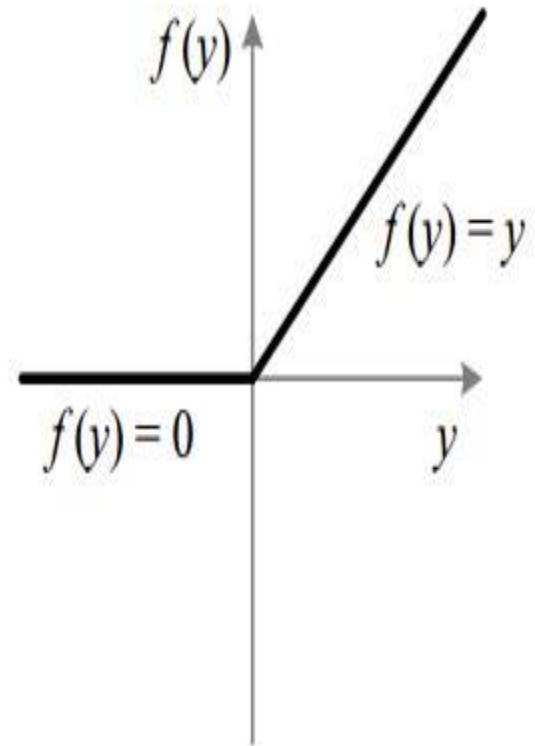
- Range is from (-1 to 1)
- Centering data – mean = 0
- is also sigmoidal (s - shaped)
- Advantage is that the -ve inputs will be mapped strongly negative
- Disadvantage – If x is very small or very large slope or gradient becomes 0 which slows down gradient descent

Activation Functions – Sigmoid vs ReLU



- ReLU is half rectified
- $f(z)$ is 0 when $z < 0$ and $f(z)$ is equal to z when $z \geq 0$.
- Derivative = 1 when z is +ve and 0 when z is 0-ve
- The function and its derivative **both are monotonic**
- **Alternate to ReLU is softplus activation function**
 - $\text{Softplus}(z) = \ln(1+\exp(z))$, Close to 0 when z is -ve and close to z when z is +ve

Activation Functions- ReLU vs Leaky ReLU



- The leak helps to increase the range of the ReLU function.
- Usually, the value of a is 0.01.
- When a is not 0.01 then it is called **Randomized ReLU**.
- **range** of the Leaky ReLU is (-infinity to infinity)

Activation Functions

SoftMax

- *Usually last activation function*
- *to normalize the output of a network to a probability distribution over predicted output classes*

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^n \exp(y_j)}$$

where,

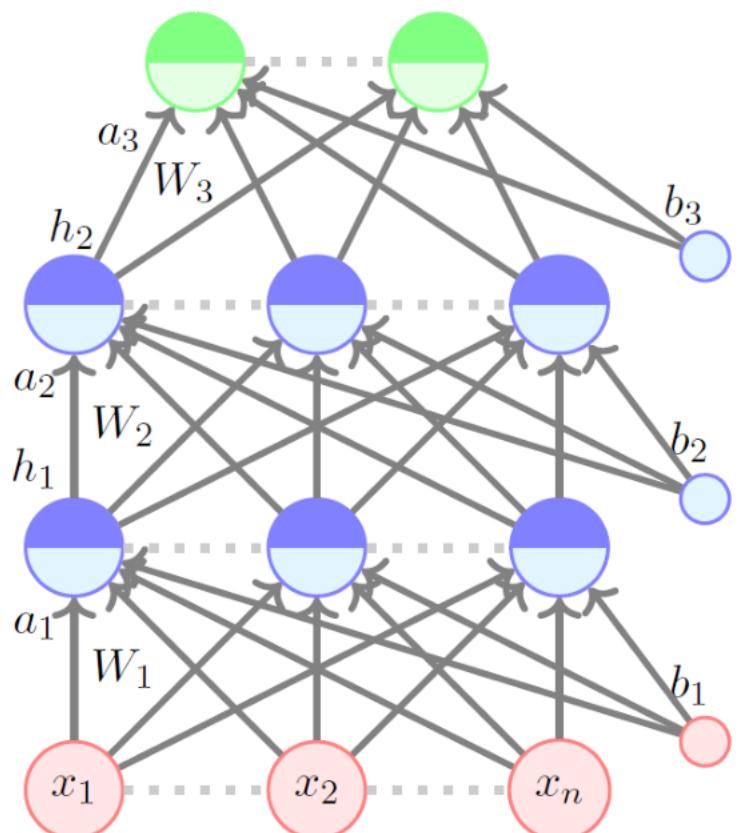
y	is an input vector to a softmax function, S. It consists of n elements for n classes (possible outcomes)
y_i	the i -th element of the input vector. It can take any value between -inf and +inf
$\exp(y_i)$	standard exponential function applied on y_i . The result is a small value (close to 0 but never 0) if $y_i < 0$ and a large value if y_i is large. eg <ul style="list-style-type: none">• $\exp(55) = 7.69e+23$ (A very large value)• $\exp(-55) = 1.30e-24$ (A very small value close to 0) <p>Note: $\exp(*)$ is just e^* where $e = 2.718$, the Euler's number.</p>
$\sum_{j=1}^n \exp(y_j)$	A normalization term. It ensures that the values of output vector $S(y)_i$ sum to 1 for i -th class and each of them is in the range 0 and 1 which makes up a valid probability distribution.
n	Number of classes (possible outcomes)

Activation Function Cheat Sheet

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) ^[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) ^[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$

Output Functions

$$h_L = \hat{y} = f(x)$$



- The pre-activation at layer i is given by

$$a_i(x) = b_i + W_i h_{i-1}(x)$$

- The activation at layer i is given by

$$h_i(x) = g(a_i(x))$$

where g is called the activation function (for example, logistic, tanh, linear, etc.)

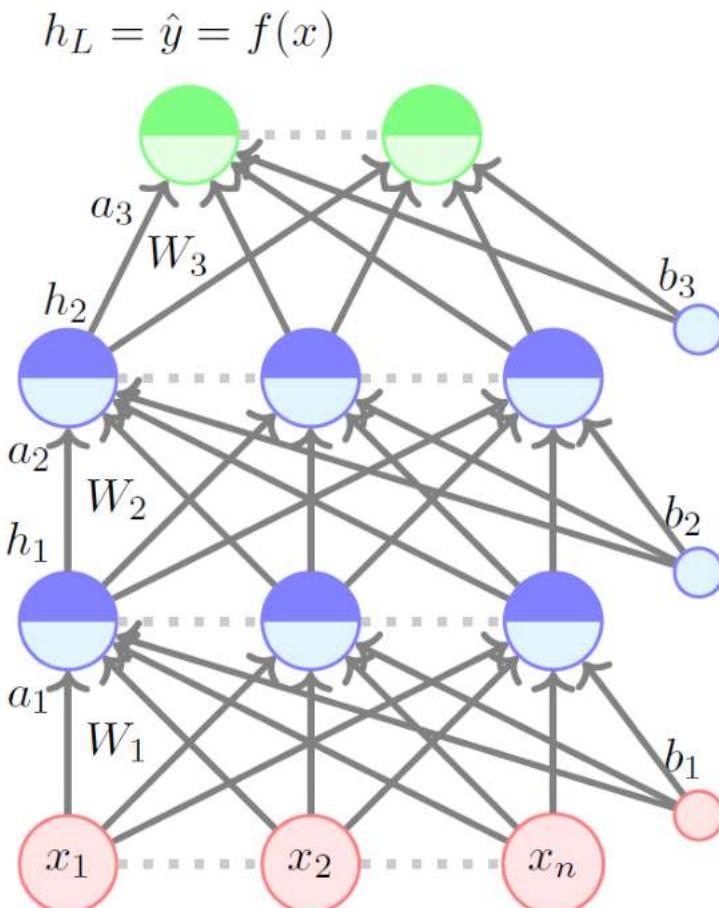
- The activation at the output layer is given by

$$f(x) = h_L(x) = O(a_L(x))$$

where O is the output activation function (for example, softmax, linear, etc.)

- To simplify notation we will refer to $a_i(x)$ as a_i and $h_i(x)$ as h_i

Regression problems - Output Function

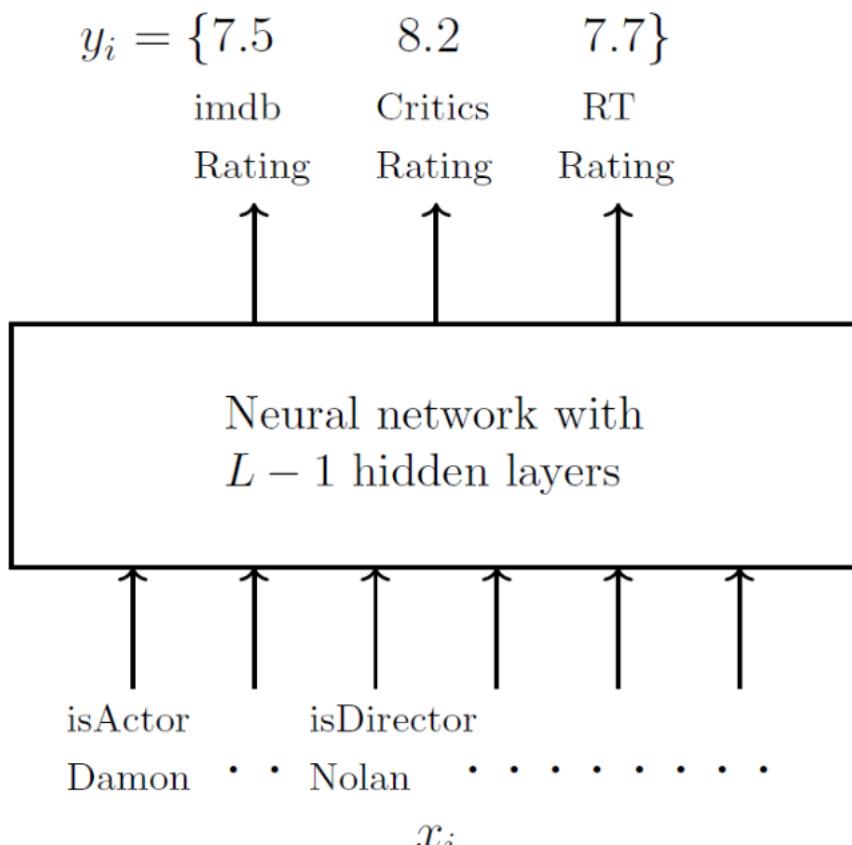


- A related question: What should the output function ' O ' be if $y_i \in \mathbb{R}$?
- More specifically, can it be the logistic function?
- No, because it restricts \hat{y}_i to a value between 0 & 1 but we want $\hat{y}_i \in \mathbb{R}$
- So, in such cases it makes sense to have ' O ' as linear function

$$\begin{aligned}f(x) &= h_L = O(a_L) \\&= W_O a_L + b_O\end{aligned}$$

- $\hat{y}_i = f(x_i)$ is no longer bounded between 0 and 1

Regression problems - Loss Function

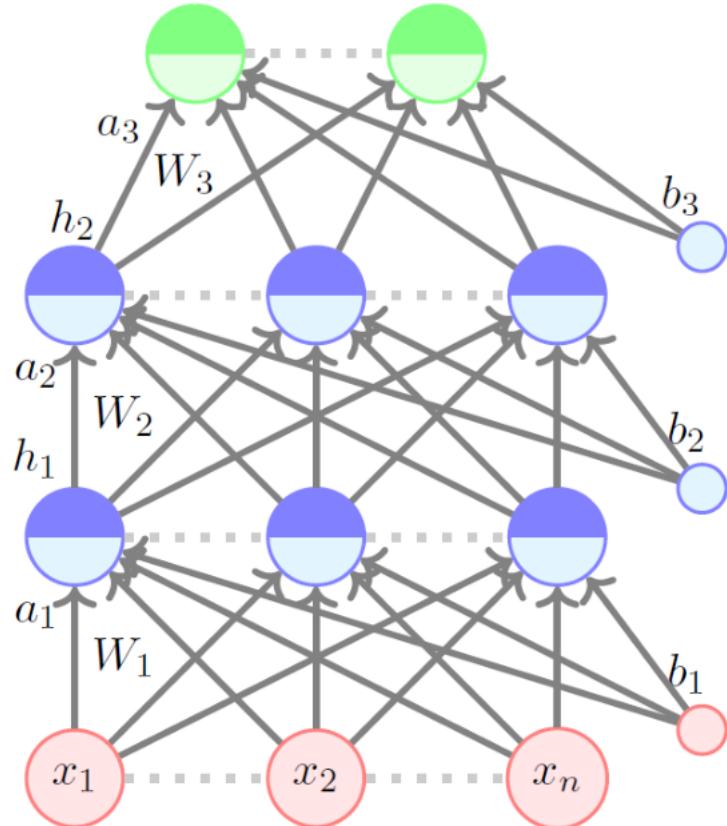


- The choice of loss function depends on the problem at hand
- We will illustrate this with the help of two examples
- Consider our movie example again but this time we are interested in predicting ratings
- Here $y_i \in \mathbb{R}^3$
- The loss function should capture how much \hat{y}_i deviates from y_i
- If $y_i \in \mathbb{R}^n$ then the squared error loss can capture this deviation

$$\mathcal{L}(\theta) = \frac{1}{N} \sum_{i=1}^N \sum_{j=1}^3 (\hat{y}_{ij} - y_{ij})^2$$

Classification problem- Output Function

$$h_L = \hat{y} = f(x)$$



- Notice that y is a probability distribution
- Therefore we should also ensure that \hat{y} is a probability distribution
- What choice of the output activation ' O ' will ensure this ?

$$a_L = W_L h_{L-1} + b_L$$

$$\hat{y}_j = O(a_L)_j = \frac{e^{a_{L,j}}}{\sum_{i=1}^k e^{a_{L,i}}}$$

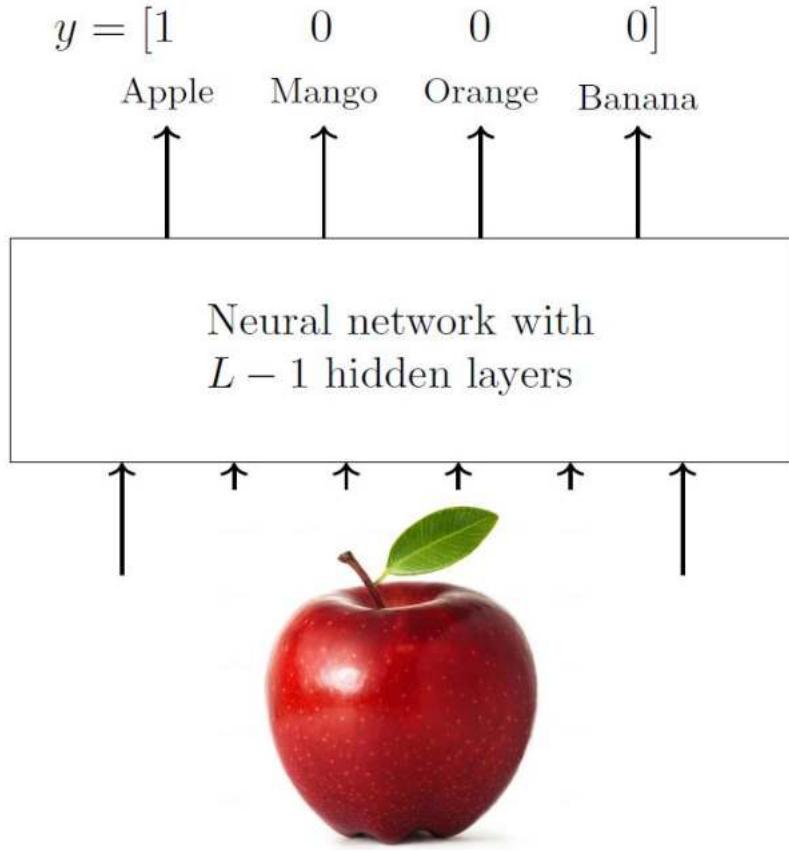
$O(a_L)_j$ is the j^{th} element of \hat{y} and $a_{L,j}$ is the j^{th} element of the vector a_L .

- This function is called the *softmax* function
What does \hat{y}_ℓ encode?

It is the probability that x belongs to the ℓ^{th} class (bring it as close to 1).

$\log \hat{y}_\ell$ is called the *log-likelihood* of the data.

Classification Problems- Loss Function



- Now that we have ensured that both y & \hat{y} are probability distributions can you think of a function which captures the difference between them?
- Cross-entropy

$$\mathcal{L}(\theta) = - \sum_{c=1}^k y_c \log \hat{y}_c$$

- Notice that

$$\begin{aligned} y_c &= 1 && \text{if } c = \ell \text{ (the true class label)} \\ &= 0 && \text{otherwise} \end{aligned}$$

$\therefore \mathcal{L}(\theta) = -\log \hat{y}_\ell$ So, for classification problem (where you have to choose 1 of K classes), we use the following objective function

$$\underset{\theta}{\text{minimize}} \quad \mathcal{L}(\theta) = -\log \hat{y}_\ell$$

$$\text{or} \quad \underset{\theta}{\text{maximize}} \quad -\mathcal{L}(\theta) = \log \hat{y}_\ell$$

Typical MLP architecture

Regression

Hyperparameter	Typical Value
# input neurons	One per input feature (e.g., $28 \times 28 = 784$ for MNIST)
# hidden layers	Depends on the problem. Typically 1 to 5.
# neurons per hidden layer	Depends on the problem. Typically 10 to 100.
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None or ReLU/Softplus (if positive outputs) or Logistic/Tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

Classification

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax

Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Loss function	Cross-Entropy	Cross-Entropy	Cross-Entropy

Regression Loss Functions

- **Mean Squared Error (MSE)**

- values with a large error are penalized.
- is a convex function with a clearly defined global minimum
- Can be used in **gradient descent optimization** to set the weight values
- Very sensitive to outliers , will significantly increase the loss.

$$MSE = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2$$

- **Mean Absolute Error (MAE)**

- used in cases when the training data has a large number of outliers as the average distance approaches 0, gradient descent optimization will not work

$$MAE = \frac{1}{n} \sum_{i=1}^n |y^{(i)} - \hat{y}^{(i)}|$$

- **Huber Loss**

- Based on absolute difference between the actual and predicted value and threshold value, δ
- Is quadratic when error is smaller than δ but linear when error is larger than δ

$$Huber Loss = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \hat{y}^{(i)})^2 \quad |y^{(i)} - \hat{y}^{(i)}| \leq \delta$$

$$\frac{1}{n} \sum_{i=1}^n \delta(|y^{(i)} - \hat{y}^{(i)}| - \frac{1}{2}\delta) \quad |y^{(i)} - \hat{y}^{(i)}| > \delta$$

Classification Loss Functions

Cross entropy measures entropy between two probability distributions

- **Binary Cross-Entropy/Log Loss**

- Compares the actual value (0 or 1) with the probability that the input aligns with that category
 - $p(i)$ = probability that the category is 1
 - $1 - p(i)$ = probability that the category is 0

$$CE\ Loss = \frac{1}{n} \sum_{i=1}^N - (y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i))$$

- **Categorical Cross-Entropy Loss**

- In cases where the number of classes is greater than two

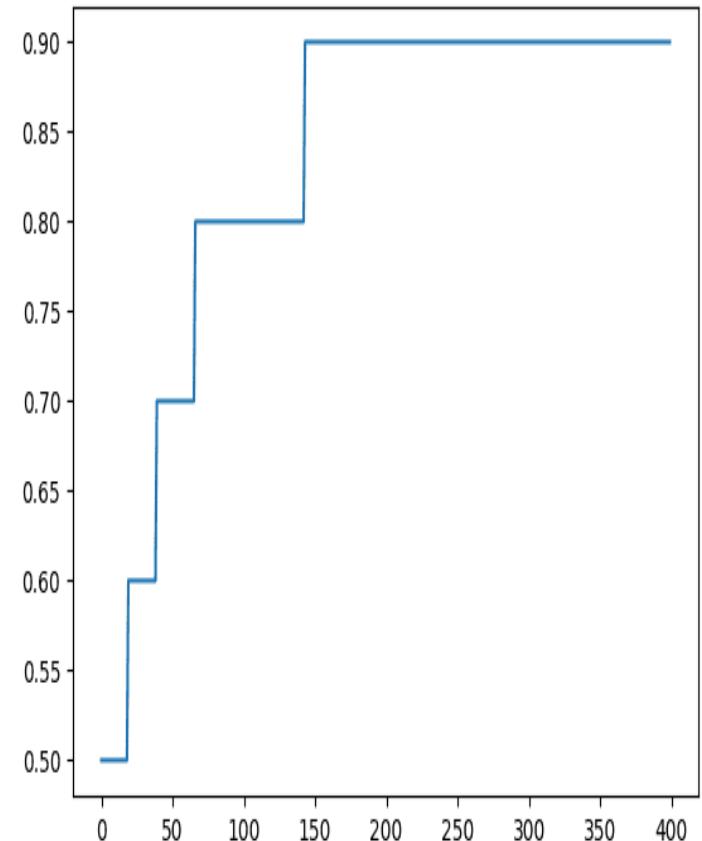
$$CE\ Loss = -\frac{1}{n} \sum_{i=1}^N \sum_{j=1}^M y_{ij} \cdot \log(p_{ij})$$

Keras Metrics

- `model.compile(..., metrics=['mse'])`
- Metric values are recorded at the end of each epoch on the training dataset.
- If a validation dataset is also provided, then is also calculated for the validation dataset.
- All metrics are reported in verbose output and in the history object returned from calling the `fit()` function.

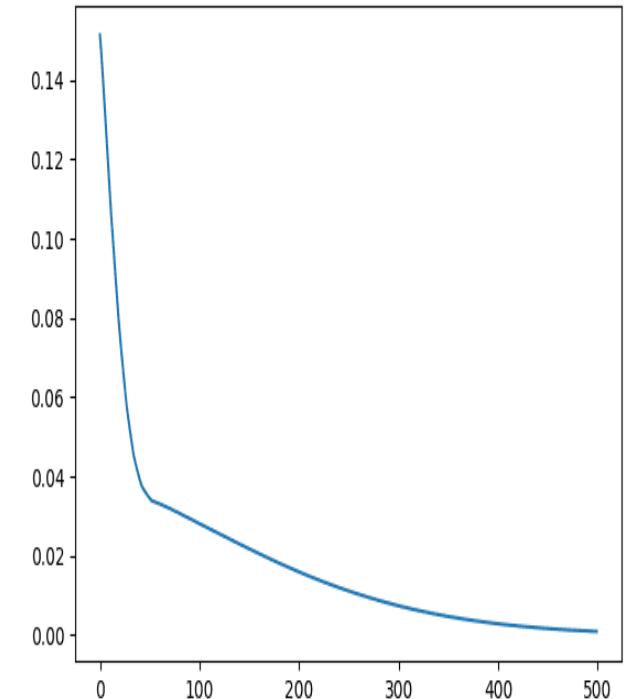
Keras Metrics

- **Accuracy metrics**
 - Accuracy
 - Calculates how often predictions equal labels.
 - Binary Accuracy
 - Calculates how often predictions match binary labels.
 - Categorical Accuracy
 - Calculates how often predictions match one-hot labels
 - Sparse Categorical Accuracy
 - Calculates how often predictions match integer labels.
 - TopK Categorical Accuracy
 - calculates the percentage of records for which the targets are in the top K predictions
 - rank the predictions in the descending order of probability values.
 - If the rank of the yPred is less than or equal to K, it is considered accurate.
 - Sparse TopK Categorical Accuracy class
 - Computes how often integer targets are in the top K predictions.



Keras Metrics

- **Regression metrics**
 - Mean Squared Error
 - Computes the mean squared error between `y_true` and `y_pred`
 - Root Mean Squared Error
 - Computes root mean SE metric between `y_true` and `y_pred`
 - Mean Absolute Error
 - Computes the mean absolute error between the labels and predictions
 - Mean Absolute Percentage Error
 - **MAPE** = $(1/n) * \sum(|\text{actual} - \text{prediction}| / |\text{actual}|) * 100$
 - Average difference between the predicted and the actual in %
 - Mean Squared Logarithmic Error
 - measure of the ratio between the true and predicted values.

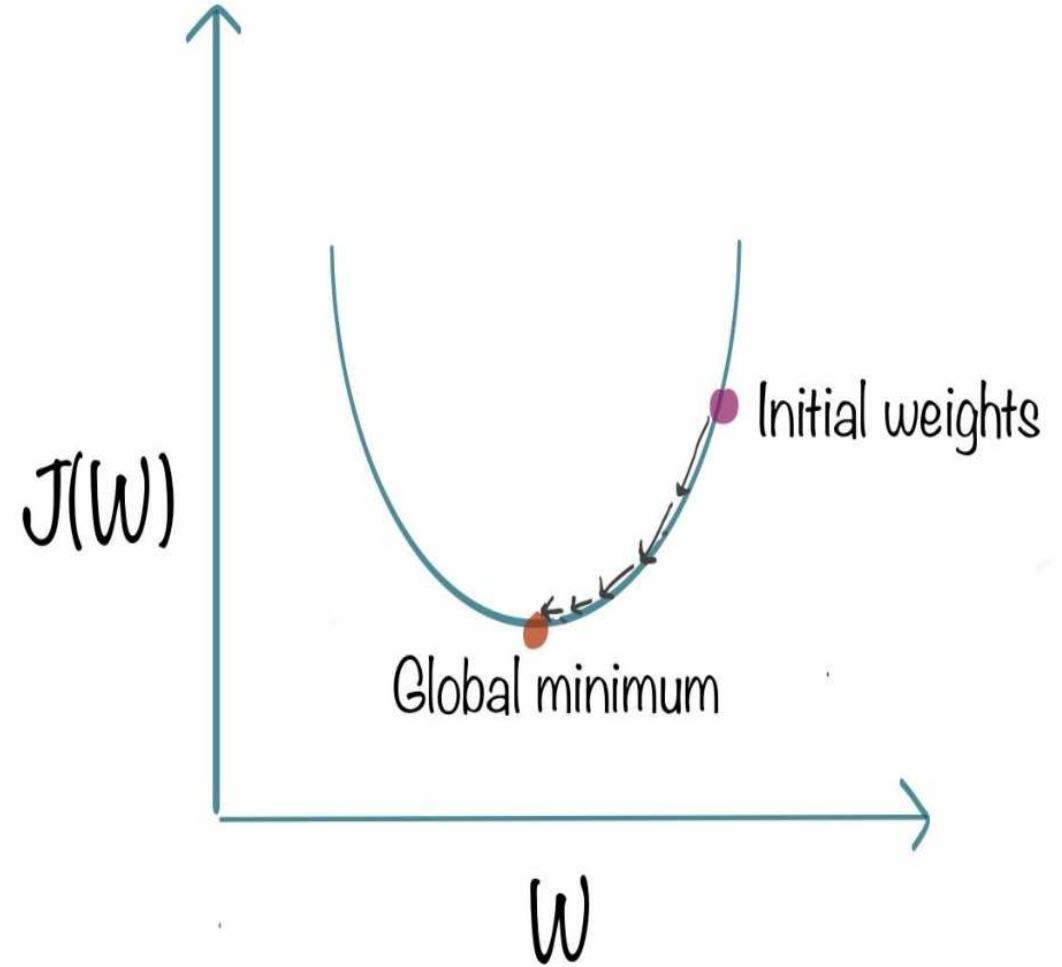


Keras Metrics

- AUC
- Precision
- Recall
- TruePositives
- TrueNegatives
- FalsePositives
- FalseNegatives
- PrecisionAtRecall
 - Computes best precision where recall is \geq specified value
- SensitivityAtSpecificity
 - Computes best sensitivity where specificity is \geq specified value
- SpecificityAtSensitivity
 - Computes best specificity where sensitivity is \geq specified value
- **Probabilistic metrics**
 - Binary Crossentropy
 - Categorical Crossentropy
 - Sparse Categorical Crossentropy
 - KLDivergence
 - a measure of how two probability distributions are different from each other
 - Poisson
 - if dataset comes from a Poisson distribution

Gradient based learning

- **Gradient Based Learning**
 - seeks to change the weights so that the next evaluation reduces the error
 - is navigating down the gradient (or slope) of error
 - The process repeats until the global minimum is reached.
 - works well for convex functions
 - It is expensive to calculate the gradients if the size of the data is huge.



The Vanishing/Exploding Gradient Problems

- Algorithm computes the gradient of the cost function with regard to each parameter in the network
- **Problems include**
- **Vanishing Gradients**
 - Gradients become very small as algorithm progresses down to lower layers
 - So connection weights remain unchanged and training never converges to a solution
- **Exploding Gradients**
 - Gradients become so large until layers get huge weight updates and algorithm diverges
- Deep Neural Networks suffer from unstable gradients, different layers may learn at different speeds.
- **Reasons for unstable gradients Glorot & Bengio (2010)**
 - Because of combination of Sigmoid activation and weight initialization (normal distribution with mean 0 and std dev 1)
 - Variance of outputs of each layer is much greater than the variance of its inputs
 - Variance goes on increasing after each layer
 - until the activation function saturates(0 or 1, with derivative close to 0) at the top layers

Solutions include : 1. Weight Initialization

- ***The variance of the outputs of each layer has to be equal to the variance of its inputs***

- **Xavier's Initialization**

$$W^{[l]} \sim \mathcal{N} \left(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}} \right)$$

$$b^{[l]} = 0$$

- Weight Matrix W of a particular layer l
- picked randomly from a *normal distribution* with
 - *mean $\mu=0$*
 - *variance $\sigma^2 = \text{multiplicative inverse of the number of neurons in layer } l-1$* .
- The bias b of all layers is initialized with 0

Solutions Include : Weight Initialization

1. Xavier's or Glorot Initialization

Variance of outputs of each layer to be equal to the variance of its inputs

Gradients to have equal variance before and after flowing through a layer in the reverse direction

Fan-in – Number of inputs

Fan-out- Number of neurons

$$fan_{avg} = (fan_{in} + fan_{out})/2.$$

Table 11-1. Initialization parameters for each type of activation function

Initialization	Activation functions	σ^2 (Normal)
Glorot	None, Tanh, Logistic, Softmax	$1 / fan_{avg}$
He	ReLU & variants	$2 / fan_{in}$
LeCun	SELU	$1 / fan_{in}$

Solutions include

2. Using Non-saturating Activation functions

1. ReLU does not saturate for +ve values
2. Suffer from problem of dying ReLU- neurons output only 0
 - Weighted sum of its inputs are negative for all instances in training set
3. Use Leaky ReLU instead – (only go into coma don't die, may wake up)
 1. $\alpha = 0.01$, sometimes 0.2
4. Flavors include

Randomized leaky ReLU(RReLU)

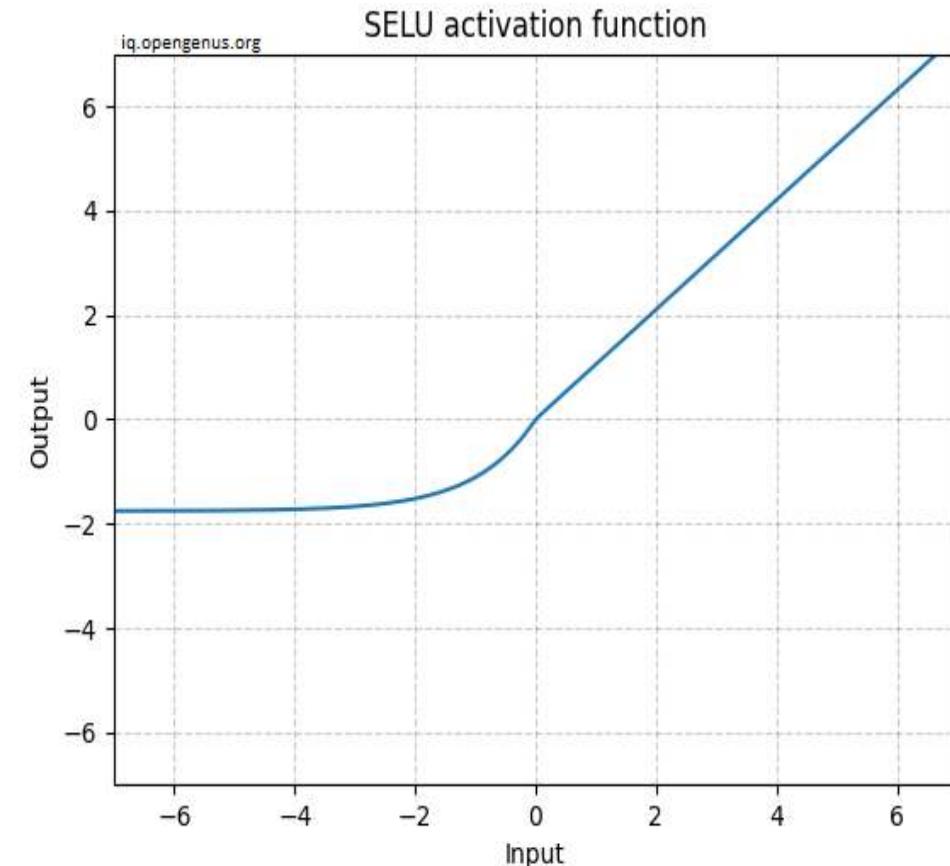
α is picked randomly in a given range during training and is fixed to an average value during testing

Parametric Leaky Relu (PReLU)

α is authorised to be learned during training as a parameter

SELU

- **Scaled Exponential Linear Units**
 - Induce self-normalization
 - the output of each layer will tend to preserve mean 0 and standard deviation 1 during training
 - $f(x) = \lambda x$ if $x \geq 0$
 - $f(x) = \lambda \alpha(\exp(x)-1)$ if $x < 0$
 - $\alpha = 1.6733$, $\lambda = 1.0507$
- conditions for self-normalization to happen:
 - The input features must be standardized (mean 0 and standard deviation 1).
 - Every hidden layer's weights must also be initialized using normal initialization.
 - The network's architecture must be sequential.



Solutions Include

3. Batch Normalization -Ioffe and Szegedy (2015)

- designed to solve the vanishing/exploding gradients problems, is also a good regularizer
- BN layer performs the standardizing and normalizing operations on the input of a layer coming from a previous layer.
- Normalization
 - brings the numerical data to a common scale without distorting its shape.
 - (mean = 0, std dev= 1)
- BN adds extra operations in the model , before activation
 - Operation zero centres and normalizes each input
 - Then scale and shift the result using two new parameter vectors per layer
 - Each BN Layer learn 4 parameter vectors
 - Output Scale Vector
 - Output offset vector
 - Input mean vector
 - Input standard deviation
- To zero-centre and normalize the inputs , mean and standard deviation of input needs to be computed
- Current mini-batch is used to evaluate mean and standard deviation

Solutions Include

3. Batch Normalization -Ioffe and Szegedy (2015)

$$1. \quad \mu_B = \frac{1}{m_B} \sum_{i=1}^{m_B} \mathbf{x}^{(i)}$$

$$2. \quad \sigma_B^2 = \frac{1}{m_B} \sum_{i=1}^{m_B} (\mathbf{x}^{(i)} - \mu_B)^2$$

$$3. \quad \hat{\mathbf{x}}^{(i)} = \frac{\mathbf{x}^{(i)} - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$4. \quad \mathbf{z}^{(i)} = \gamma \otimes \hat{\mathbf{x}}^{(i)} + \beta$$

μ_B is the vector of input means, evaluated over the whole mini-batch B (it contains one mean per input).

- σ_B is the vector of input standard deviations, also evaluated over the whole mini-batch (it contains one standard deviation per input).
- m_B is the number of instances in the mini-batch.
- $\hat{\mathbf{x}}^{(i)}$ is the vector of zero-centered and normalized inputs for instance i .
- γ is the output scale parameter vector for the layer (it contains one scale parameter per input).
- \otimes represents element-wise multiplication (each input is multiplied by its corresponding output scale parameter).
- β is the output shift (offset) parameter vector for the layer (it contains one offset parameter per input). Each input is offset by its corresponding shift parameter.
- ϵ is a tiny number to avoid division by zero (typically 10^{-5}). This is called a *smoothing term*.
- $\mathbf{z}^{(i)}$ is the output of the BN operation: it is a rescaled and shifted version of the inputs.

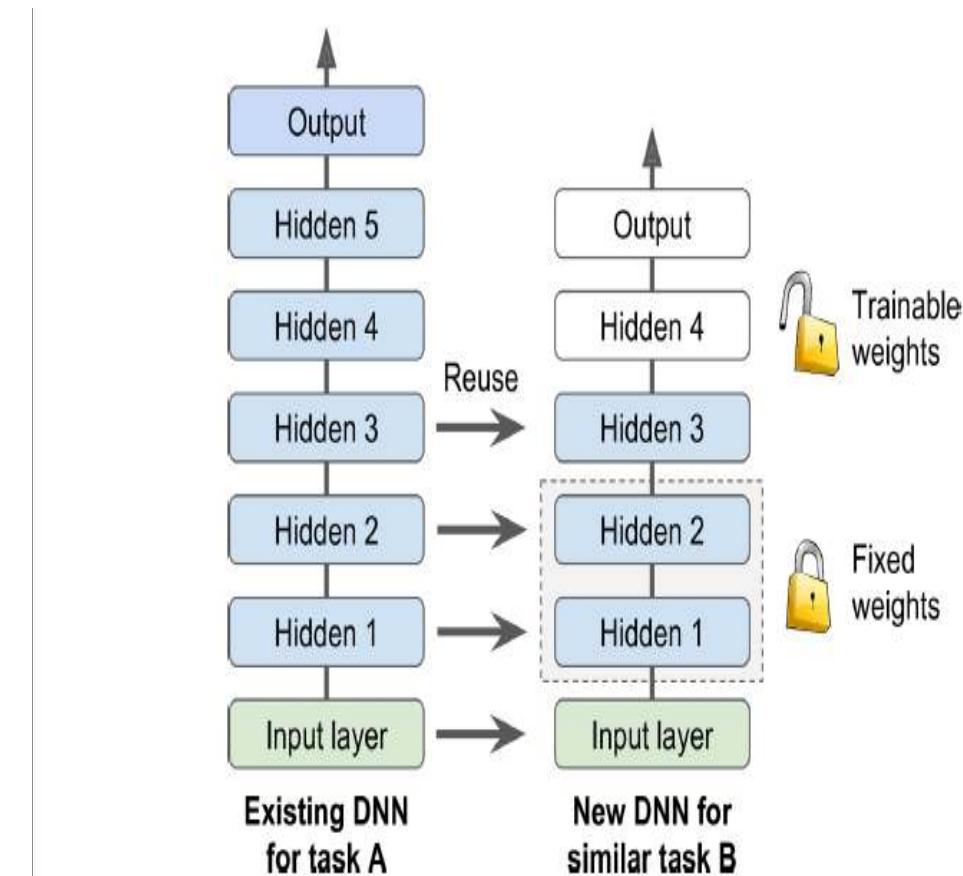
Solutions include

4. Gradient Clipping

- Clip gradient during back propagation so that they never exceed some threshold
- All the partial derivatives of the loss will be clipped between -0.1 to 0.1
- Threshold can also be a hyperparameter to tune

5. Reusing Pretrained Layers

- Transfer Learning



6. Faster Optimizers

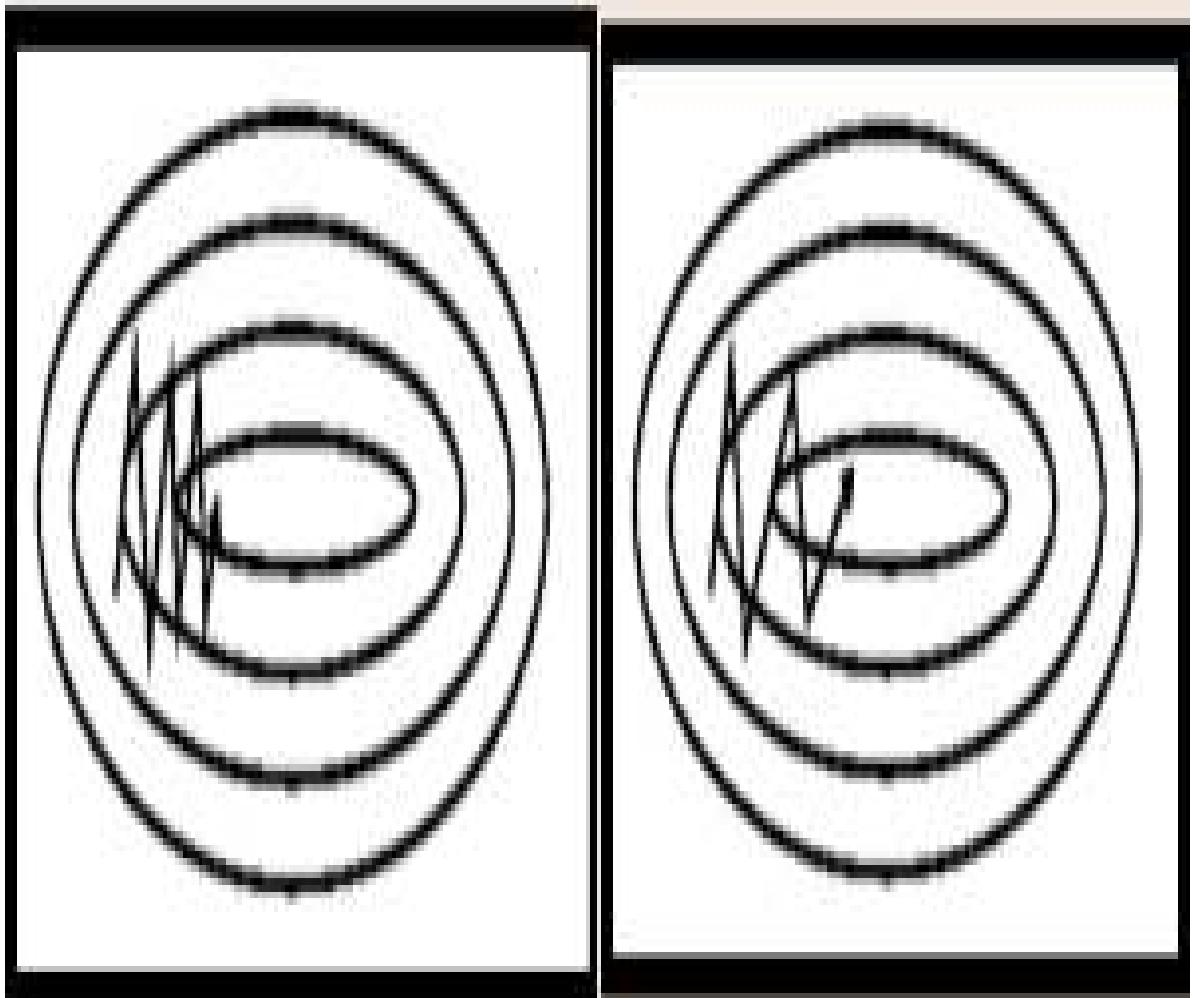
- Optimizer
 - is a function or an algorithm that modifies the attributes of the neural network, such as weights and learning rate.
- Terminology
 - Weights/ Bias – The learnable parameters in a model that controls the signal between two neurons.
 - Epoch – The number of times the algorithm runs on the whole training dataset.
 - Sample – A single row of a dataset.
 - Batch –denotes the number of samples to be taken for updating the model parameters.
 - Learning rate –defines a scale of how much model weights should be updated.
 - Cost Function/Loss Function -is used to calculate the cost that is the difference between the predicted value and the actual value.

Mini Batch Gradient Descent Deep Learning Optimizer

- **Batch gradient descent:**
 - gradient is average of gradients computed from ALL the samples in dataset
- Mini Batch GD:
 - subset of the dataset is used for calculating the loss function, therefore fewer iterations are needed.
 - batch size of 32 is considered to be appropriate for almost every case.
 - Yann Lecun (2018) – “Friends don’t let friends use mini batches larger than 32”
- is faster , more efficient and robust than the earlier variants of gradient descent.
- the cost function is noisier than the batch GD but smoother than SDG.
- Provides a good balance between speed and accuracy.
- It needs a hyperparameter that is “mini-batch-size”, which needs to be tuned to achieve the required accuracy.

Stochastic GD & SGD with Momentum DL

- stochastic means randomness on which the algorithm is based upon.
- Instead of taking the whole dataset for each iteration, randomly select the batches of data
- The path taken is full of noise as compared to the gradient descent algorithm.
- Uses a higher number of iterations to reach the local minima, thereby the overall computation time increases.
- The computation cost is still less than that of the gradient descent optimizer.
- If the data is enormous and computational time is an essential factor, SGD should be preferred over batch gradient descent algorithm.
- **Stochastic Gradient Descent with Momentum Deep Learning Optimizer**
 - momentum helps in faster convergence of the loss function.



SGD with Momentum Optimizer

- GD takes small , regular steps down the slope so algorithm takes more time to reach the bottom
- adding a fraction of the previous update to the current update will make the process a bit faster.
- Hyperparameter β - Momentum
 - To simulate a friction mechanism and prevent momentum from becoming too large
- Also rolls past local minima
- learning rate should be decreased with a high momentum term.

$$1. \quad \mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\theta} J(\theta)$$

$$2. \quad \theta \leftarrow \theta + \mathbf{m}$$

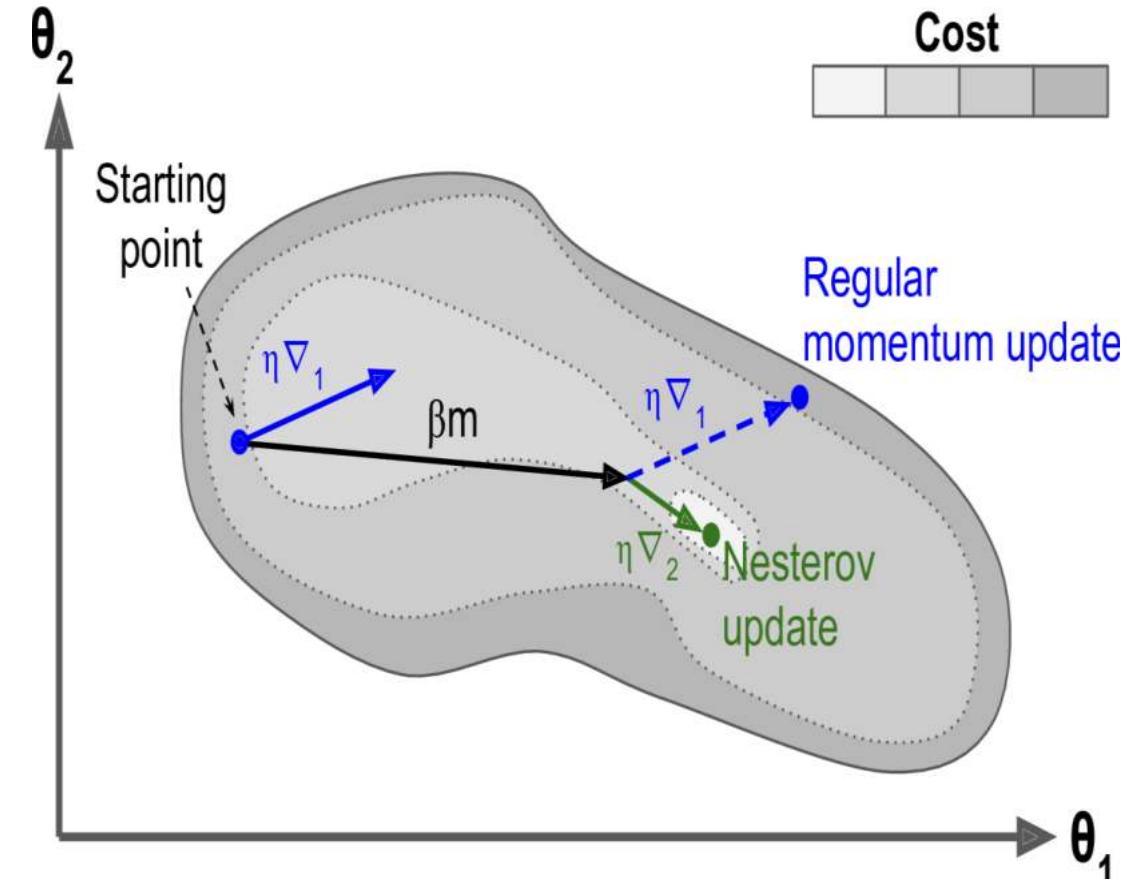
- β - Momentum
- set between 0 (high friction) and 1 (low friction)
- Typically 0.9

```
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9)
```

SGD with Nesterov Momentum Optimization

- Yurii Nesterov in 1983
- to measure the gradient of the cost function not at the local position but slightly ahead in the direction of the momentum
- the momentum vector will be pointing in the right direction (i.e., toward the optimum)
- it will be slightly more accurate to use the gradient measured a bit farther in that direction rather than using the gradient at the original position

1. $\mathbf{m} \leftarrow \beta\mathbf{m} - \eta\nabla_{\theta}J(\theta + \beta\mathbf{m})$
2. $\theta \leftarrow \theta + \mathbf{m}$



```
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9, nesterov=True)
```

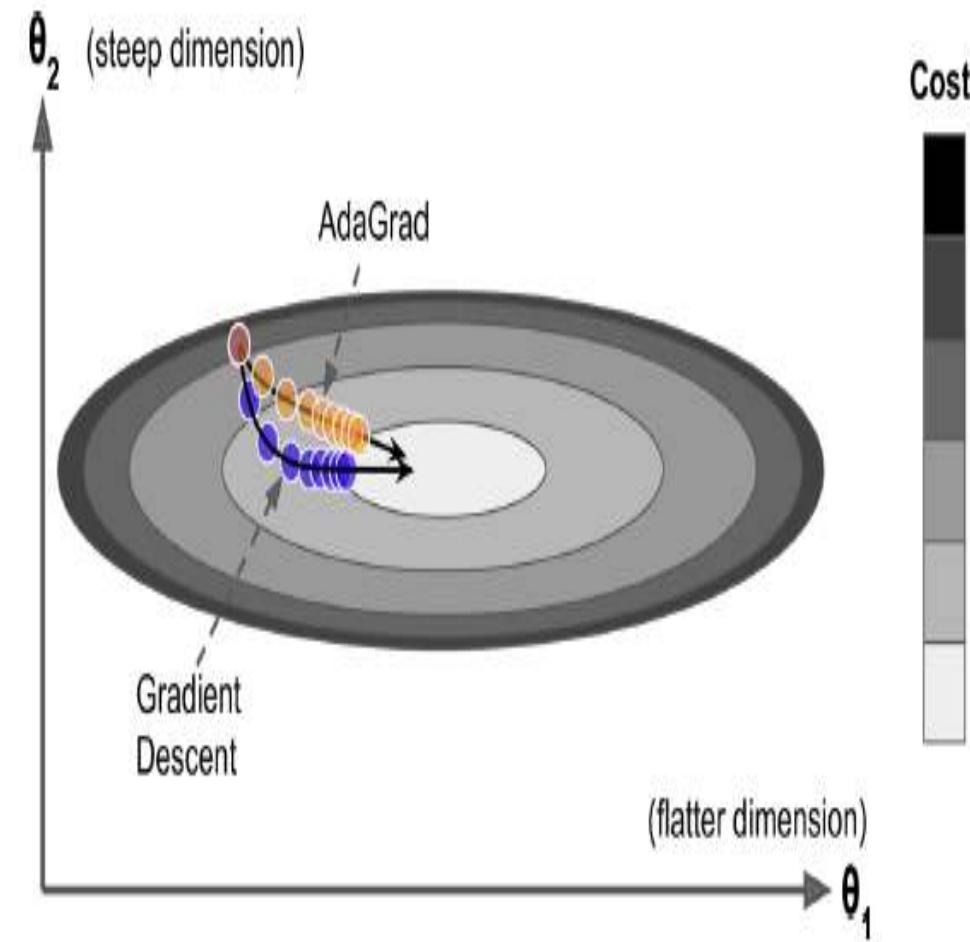
Adagrad (Adaptive Gradient Descent) Deep Learning Optimizer

- Adaptive Learning Rate
- Scaling down the gradient vector along the steepest dimension
- If the cost function is steep along the i th dimension, then s will get larger and larger at each iteration
- No need to modify the learning rate manually
- more reliable than gradient descent algorithms, and it reaches convergence at a higher speed.

$$s \leftarrow s + \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$$

$$\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta) \odot \sqrt{s + \epsilon}$$

- Disadvantage
 - it decreases the learning rate aggressively and monotonically.
 - Due to small learning rates, the model eventually becomes unable to acquire more knowledge, and hence the accuracy of the model is compromised.



RMS Prop(Root Mean Square) Deep Learning Optimizer

- The problem with the gradients some are small while others may be huge
- Defining a single learning rate might not be the best idea.
- accumulating only the gradients from the most recent iterations (as opposed to all the gradients since the beginning of training).

$$s \leftarrow \beta s + (1 - \beta) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$$

$$\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta) \oslash \sqrt{s + \epsilon}$$

`optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9)`

- Works better than Adagrad , was popular until ADAM

Adam Deep Learning Optimizer

- is derived from adaptive moment estimation.
- inherit the features of both Adagrad and RMS prop algorithms.
 - like Momentum optimization keeps track of an exponentially decaying average of past gradients,
 - like RMSProp it keeps track of an exponentially decaying average of past squared gradients
 - β_1 and β_2 represent the decay rate of the average of the gradients.
- Advantages
 - Is straightforward to implement
 - faster running time
 - low memory requirements, and requires less tuning
- Disadvantages
 - Focusses on faster computation time, whereas SGD focus on data points.
 - Therefore SGD generalize the data in a better manner at the cost of low computation speed.

$$\begin{aligned}\mathbf{m} &\leftarrow \beta_1 \mathbf{m} - (1 - \beta_1) \nabla_{\theta} J(\theta) \\ \mathbf{s} &\leftarrow \beta_2 \mathbf{s} + (1 - \beta_2) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta) \\ \widehat{\mathbf{m}} &\leftarrow \frac{\mathbf{m}}{1 - \beta_1^t} \\ \widehat{\mathbf{s}} &\leftarrow \frac{\mathbf{s}}{1 - \beta_2^t} \\ \theta &\leftarrow \theta + \eta \widehat{\mathbf{m}} \oslash \sqrt{\widehat{\mathbf{s}}} + \epsilon\end{aligned}$$

t represents iteration

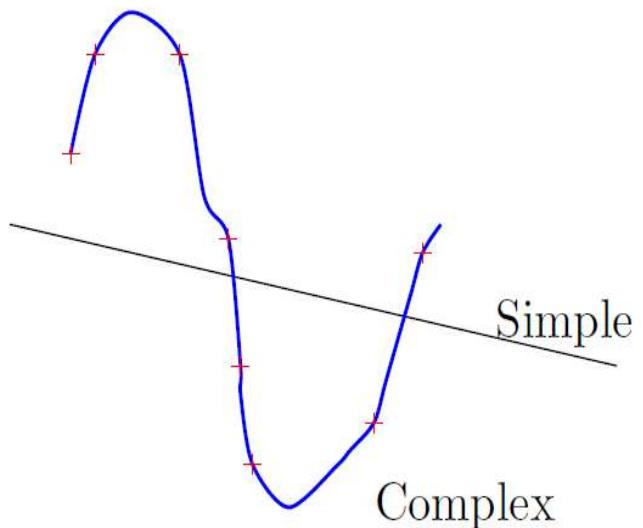
$\beta_1 = 0.9$

$\beta_2 = 0.999$

Smoothing term ϵ is $= 10^{-7}$

```
optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
```

Curve Fitting – True Function is Sinusoidal



The points were drawn from a sinusoidal function (the true $f(x)$)

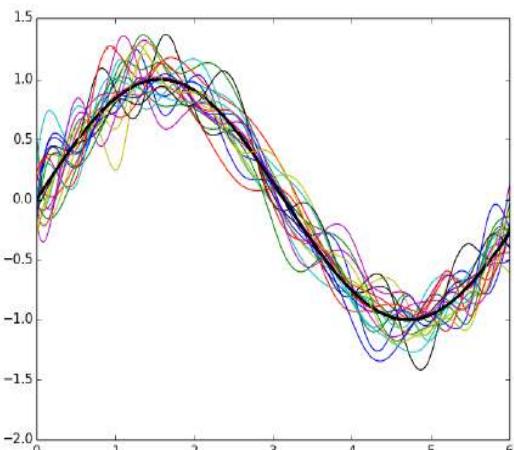
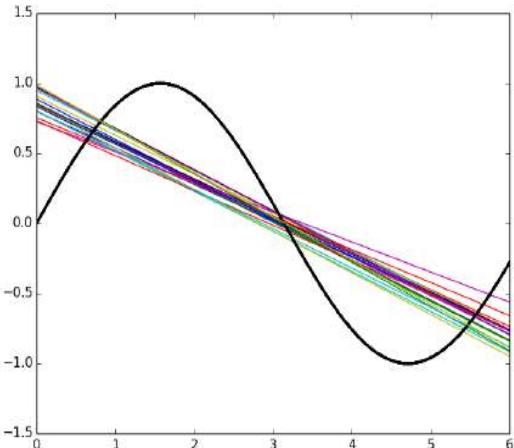
- Let us consider the problem of fitting a curve through a given set of points
- We consider two models :

$$\begin{array}{ll} \text{Simple} \\ (\text{degree:1}) & y = \hat{f}(x) = w_1 x + w_0 \end{array}$$

$$\begin{array}{ll} \text{Complex} \\ (\text{degree:25}) & y = \hat{f}(x) = \sum_{i=1}^{25} w_i x^i + w_0 \end{array}$$

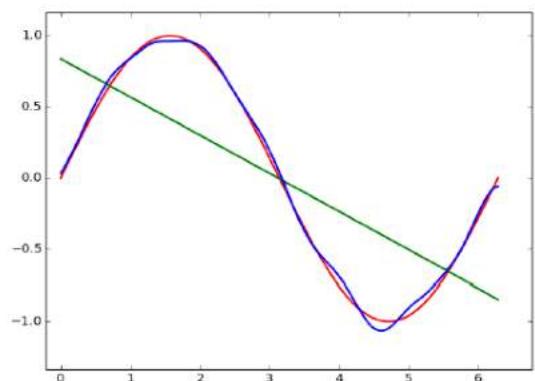
- Note that in both cases we are making an assumption about how y is related to x . We have no idea about the true relation $f(x)$

Curve Fitting – True Function is Sinusoidal



- Simple models trained on different samples of the data do not differ much from each other
- However they are very far from the true sinusoidal curve (under fitting)
- On the other hand, complex models trained on different samples of the data are very different from each other (high variance)

Bias



Green Line: Average value of $\hat{f}(x)$ for the simple model

Blue Curve: Average value of $\hat{f}(x)$ for the complex model

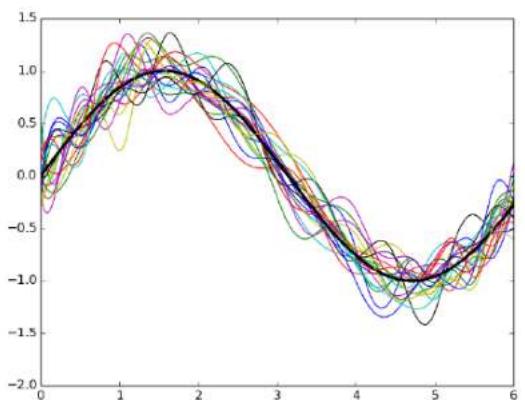
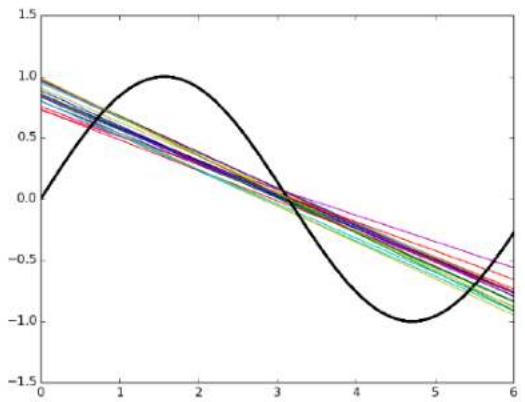
Red Curve: True model ($f(x)$)

- Let $f(x)$ be the true model (sinusoidal in this case) and $\hat{f}(x)$ be our estimate of the model (simple or complex, in this case) then,

$$\text{Bias } (\hat{f}(x)) = E[\hat{f}(x)] - f(x)$$

- $E[\hat{f}(x)]$ is the average (or expected) value of the model
- We can see that for the simple model the average value (green line) is very far from the true value $f(x)$ (sinusoidal function)
- Mathematically, this means that the simple model has a high bias
- On the other hand, the complex model has a low bias

Variance



- We now define,

Variance ($\hat{f}(x)$) = $E[(\hat{f}(x) - E[\hat{f}(x)])^2]$
(Standard definition from statistics)

- Roughly speaking it tells us how much the different $\hat{f}(x)$'s (trained on different samples of the data) differ from each other
- It is clear that the simple model has a low variance whereas the complex model has a high variance

Mean Square Error

- We can show that

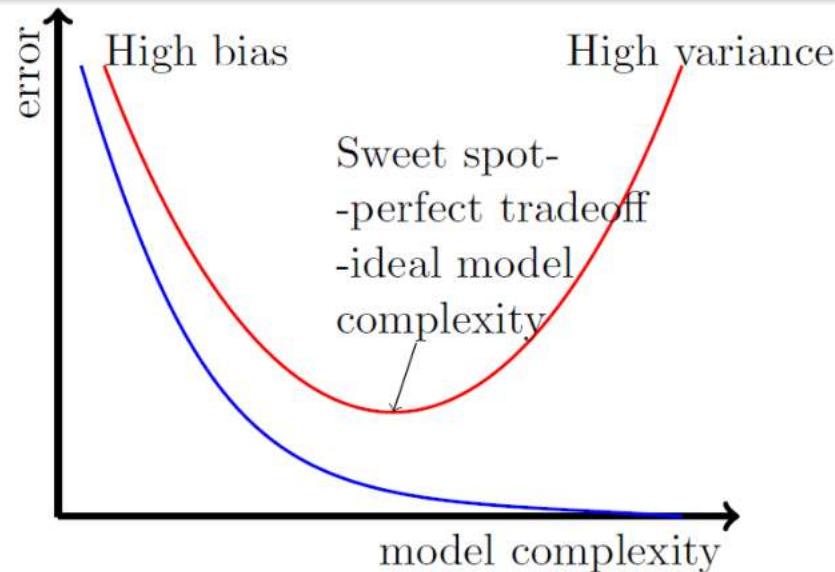
$$\begin{aligned} E[(y - \hat{f}(x))^2] &= \text{Bias}^2 \\ &\quad + \text{Variance} \\ &\quad + \sigma^2 \text{ (irreducible error)} \end{aligned}$$

- Consider a new point (x, y) which was not seen during training
- If we use the model $\hat{f}(x)$ to predict the value of y then the mean square error is given by

$$E[(y - \hat{f}(x))^2]$$

(average square error in predicting y for many such unseen points)

Train vs Test Error



- The parameters of $\hat{f}(x)$ (all w_i 's) are trained using a training set $\{(x_i, y_i)\}_{i=1}^n$
- However, at test time we are interested in evaluating the model on a validation (unseen) set which was not used for training
- This gives rise to the following two entities of interest:
 $train_{err}$ (say, mean square error)
 $test_{err}$ (say, mean square error)
- Typically these errors exhibit the trend shown in the adjacent figure

Train vs Test Error

- Let there be n training points and m test (validation) points

$$train_{err} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$$

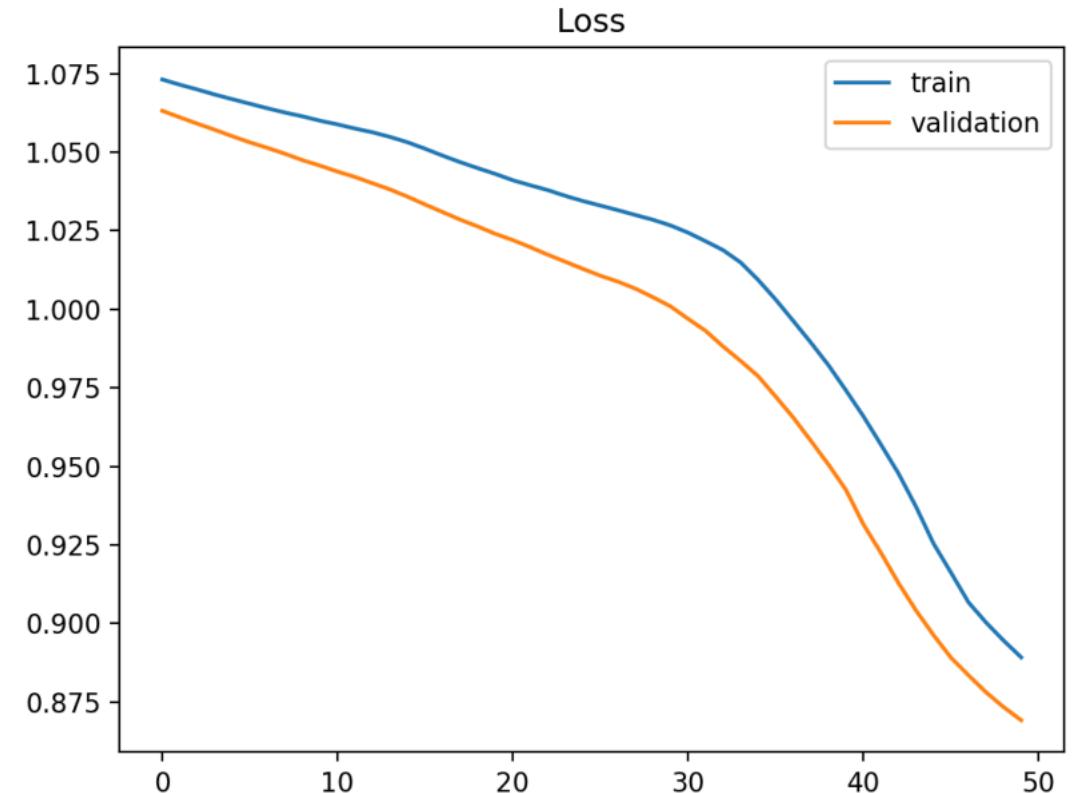
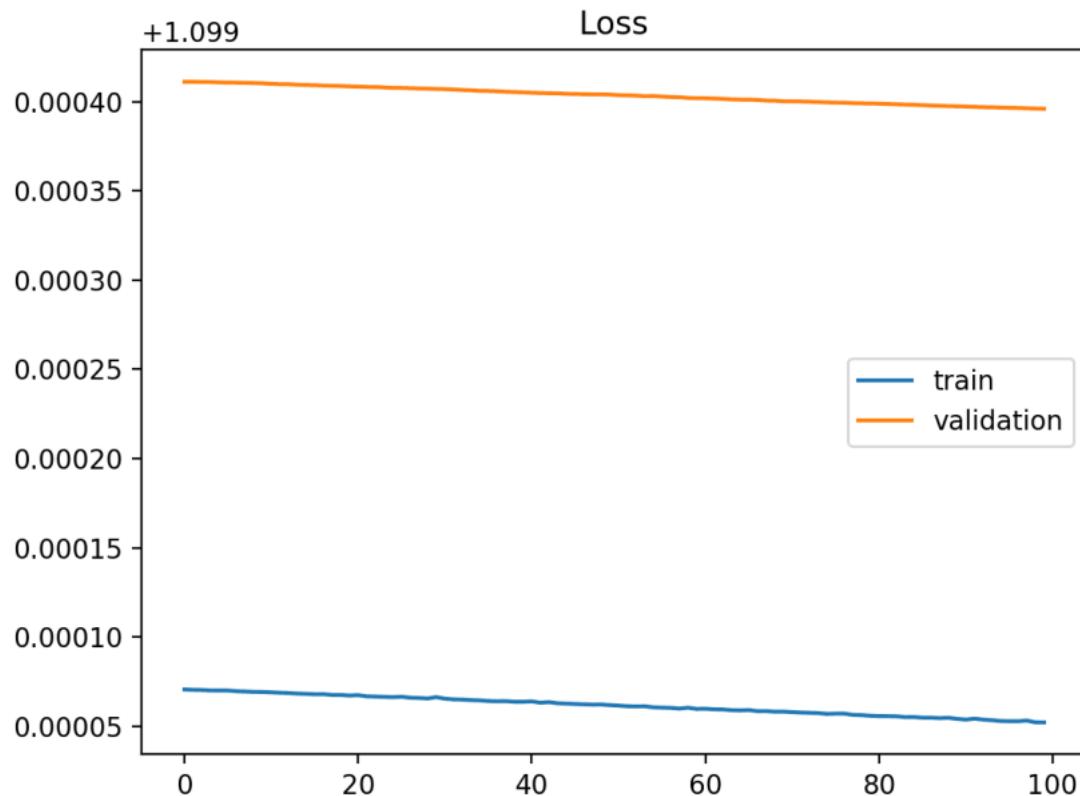
$$test_{err} = \frac{1}{m} \sum_{i=n+1}^{n+m} (y_i - \hat{f}(x_i))^2$$

- As the model complexity increases $train_{err}$ becomes overly optimistic and gives us a wrong picture of how close \hat{f} is to f
- The validation error gives the real picture of how close \hat{f} is to f

Learning Curves

- Line plot of learning (y-axis) over experience (x-axis)
- The metric used to evaluate learning could be
- **Optimization Learning Curves:**
 - calculated on the metric by which the parameters of the model are being optimized, e.g. loss.
 - Minimizing, such as loss or error
- **Performance Learning Curves:**
 - calculated on the metric by which the model will be evaluated and selected, e.g. accuracy.
 - Maximizing metric , such as classification accuracy
- **Train Learning Curve:**
 - calculated from the training dataset that gives an idea of how well the model is learning.
- **Validation Learning Curve:**
 - calculated from a hold-out validation dataset that gives an idea of how well the model is generalizing.

Underfit Learning Curves

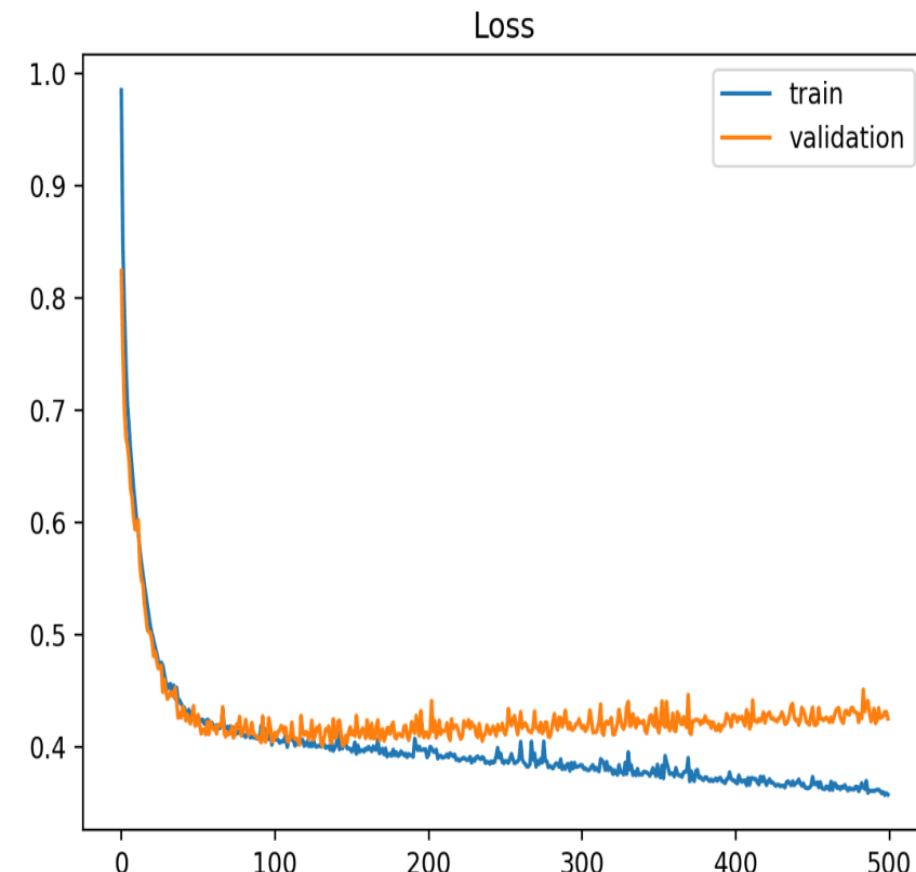


A plot of learning curves shows underfitting if:

- The training loss remains flat regardless of training.
- The training loss continues to decrease until the end of training.

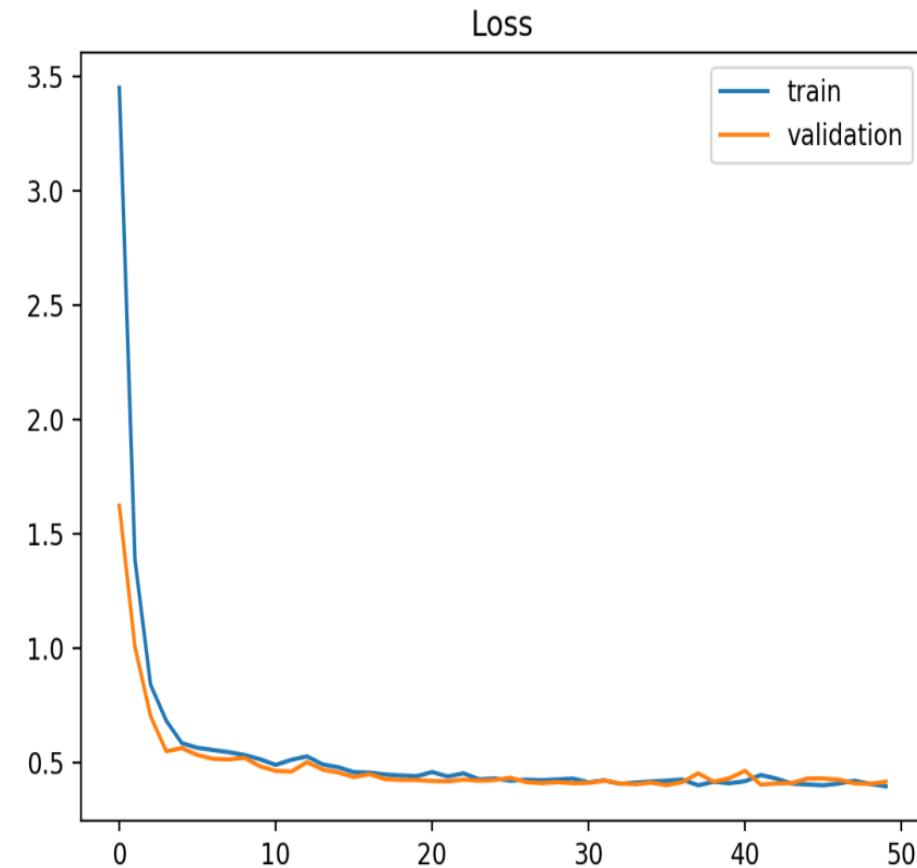
Overfitting Curves

- Overfitting
 - Model specialized on training data, it is not able to generalize to new data
 - Results in increase in generalization error.
 - generalization error can be measured by the performance of the model on the validation dataset.
- A plot of learning curves shows overfitting if:
 - The plot of training loss continues to decrease with experience.
 - The plot of validation loss decreases to a point and begins increasing again.
 - The inflection point in validation loss may be the point at which training could be halted as experience after that point shows the dynamics of overfitting.



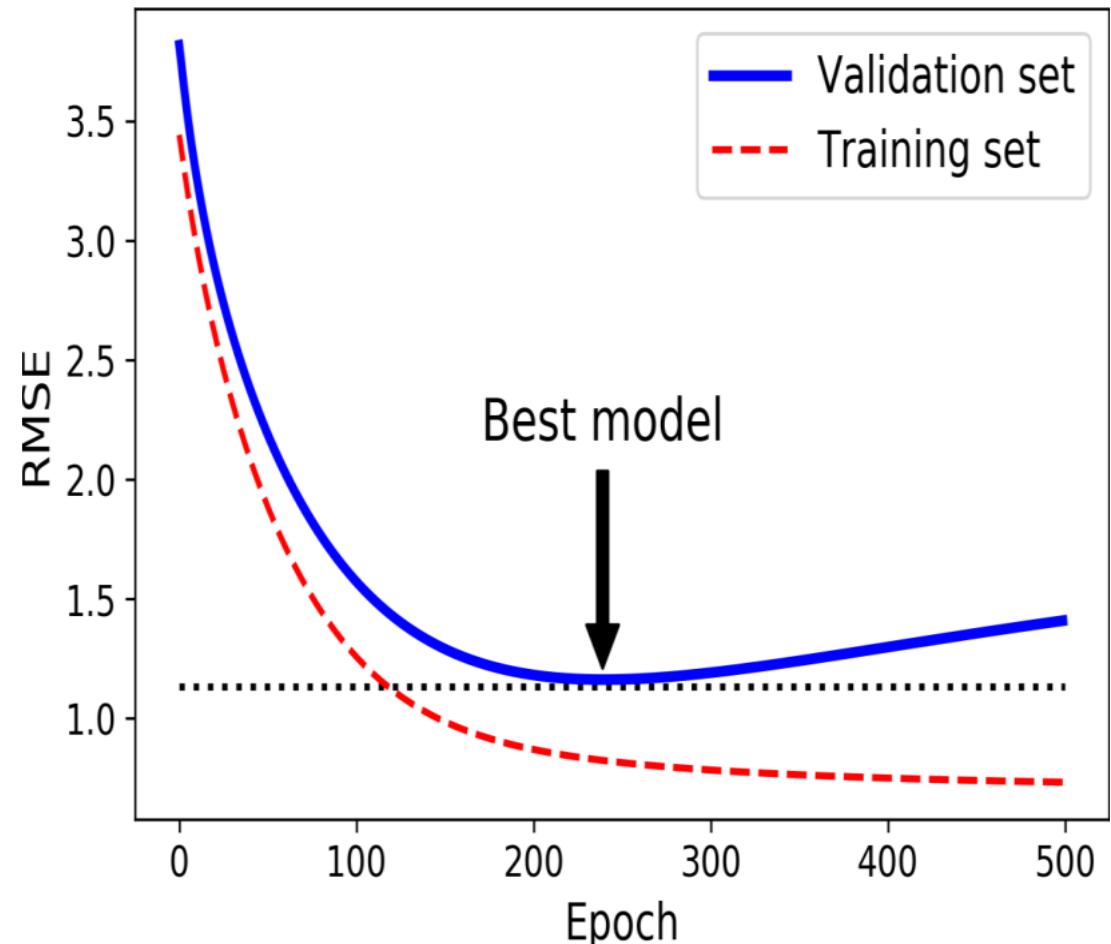
Good Fit Learning Curves

- A good fit is the goal of the learning algorithm and exists between an overfit and underfit model.
- A plot of learning curves shows a good fit if:
 - Plot of training loss decreases to a point of stability
 - Plot of validation loss decreases to a point of stability and has a small gap with the training loss.
- Loss of the model will almost always be lower on the training than the validation dataset.
- We should expect some gap between the train and validation loss learning curves.
- This gap is referred to as the “generalization gap.”



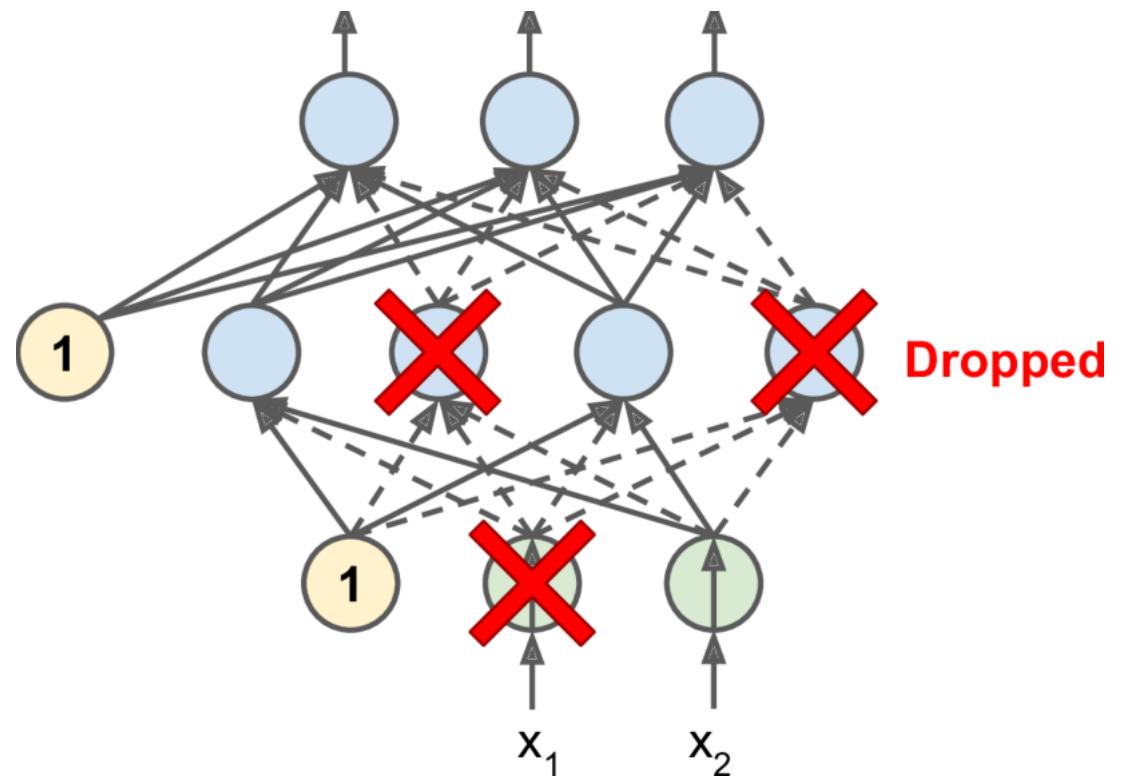
Avoiding Overfitting

- With so many parameters, DNN can fit complex datasets.
- But also prone to overfitting the training set.
- Early Stopping
 - stop training as soon as the validation error reaches a minimum
 - With Stochastic and Mini-batch Gradient Descent, the curves are not so smooth, and it may be hard to know whether you have reached the minimum or not.
 - Stop only after the validation error has been above the minimum for some time



Avoiding overfitting

- Dropout
 - Proposed by Geoffrey Hinton in 2012
 - at every training step, every neuron has a probability p of being temporarily “dropped out,”
 - it will be entirely ignored during this training step, but it may be active during the next step
 - p is called the dropout rate, usually 50%.
 - After training, neurons don’t get dropped
 - If $p = 50\%$
 - during testing a neuron will be connected to twice as many input neurons as it was (on average) during training.
 - multiply each input connection weight by the keep probability ($1 - p$) after training
 - Alternatively, divide each neuron’s output by the keep probability during training



Overfitting using Regularization

- L1, l2 regularization
 - Regularization can be used to constrain the NN weights
 - Lasso – (l1) least absolute shrinkage and selection operator
 - adds the “absolute value of magnitude” of the coefficient as a penalty term to the loss function.
 - Ridge regression (l2)
 - adds the “squared magnitude” of the coefficient as the penalty term to the loss function.
 - Use l1(), l2(), l1_l2() function
 - which returns a regularizer that will compute the regularization loss at each step during training
 - Regularization loss is added to final loss
- Max-Norm Normalization
 - It **constrains** the weights w of the incoming connections

$$\cdot (\mathbf{w} \leftarrow \mathbf{w} \frac{r}{\|\mathbf{w}\|_2}).$$

- r is max-norm hyper parameter and $\|\cdot\|_2$ is the ℓ_2 norm
- Reducing r increases the amount of regularization and helps reduce overfitting
- Can also help alleviate the unstable gradients problem if we are not using Batch normalization

Constant learning rate not ideal

- Better to start with a high learning rate
- then reduce it once it stops making fast progress
- can reach a good solution faster
- Learning Schedule strategies can be applied
 - Power Scheduling
 - Exponential Scheduling
 - Piecewise Constant Scheduling
 - Performance Scheduling



Learning Rate Scheduling

- *Power scheduling*
 - Learning rate set to a function of the iteration number ‘t’
 - $t: \eta(t) = \eta_0 / (1 + t/k)^c$
 - The initial learning rate η_0 , the power c (typically set to 1)
 - The learning rate drops at each step, and after s steps it is down to $\eta_0 / 2$ and so on
 - schedule first drops quickly, then more and more slowly
 - `optimizer = keras.optimizers.SGD(lr=0.01, decay=1e-4)`
 - *The decay is the number of steps it takes to divide the learning rate by one more unit,*
 - *Keras assumes that c is equal to 1.*
- *Exponential scheduling*
 - Set the learning rate to: $\eta(t) = \eta_0 0.1^{t/s}$
 - learning rate will gradually drop by a factor of 10 every s steps.

Learning Rate Scheduling

- *Piecewise constant scheduling*
 - Constant learning rate for a number of epochs
 - e.g., $\eta_0 = 0.1$ for 5 epochs
 - then a smaller learning rate for another number of epochs
 - e.g., $\eta_1 = 0.001$ for 50 epochs and so on
- *Performance scheduling*
 - Measure the validation error every N steps (just like for early stopping)
 - reduce the learning rate by a factor of λ when the error stops dropping

References

- Ian Goodfellow, Yoshua Bengio and Aaron Courville, “Deep Learning”, MIT Press 2016
- Swayam NPTEL Notes- Deep Learning, Mitesh Khapra
- Course Notes – Neural Networks and Deep Learning, Andrew NG
- Aurelien Geron, “Hands-On Machine Learning with Scikit-Learn , Keras & Tensorflow, OReilly Publications
- Jiawei Han and Micheline Kamber, “Data Mining Concepts And Techniques”, 3rd Edition, Morgan Kauffmann

DSE 3151 DEEP LEARNING

Convolutional Neural Networks

Dr. Rohini Rao & Dr. Abhilash K Pai

Dept. of Data Science and Computer Applications

MIT Manipal

The Convolution Operation - 1D

- Convolution is a linear operation on two functions of a real-valued argument, where one function is applied over the other to yield element-wise dot products.
- Example: Consider a discrete signal ' x_t ' which represents the position of a spaceship at time ' t ' recorded by a laser sensor.
- Now, suppose that this sensor is noisy.
- To obtain a less noisy measurement we would like to average several measurements.
- Considering that, the most recent measurements are more important, we would like to take a weighted average over ' x_t '. The new estimate at time ' t ' is computed as follows:

$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} = (x * w)_t$$

convolution
↑
input Filter/Mask/Kernel



x_0



x_1



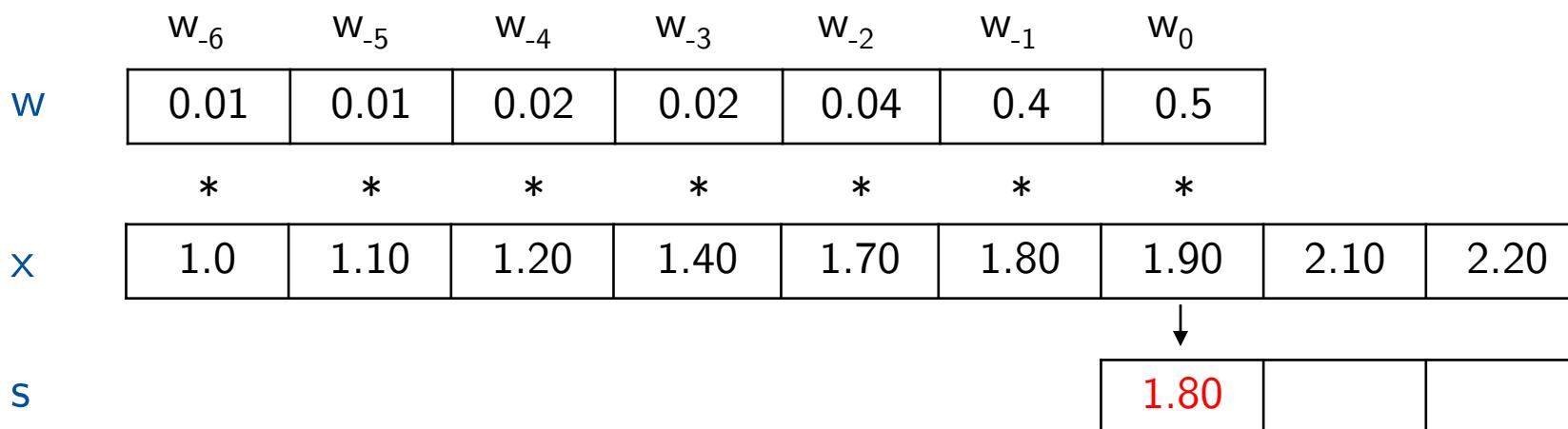
x_2

The Convolution Operation - 1D

- In practice, we would sum only over a small window.

For example: $s_t = \sum_{a=0}^6 x_{t-a} w_{-a}$

- We just slide the filter over the input and compute the value of s_t based on a window around x_t



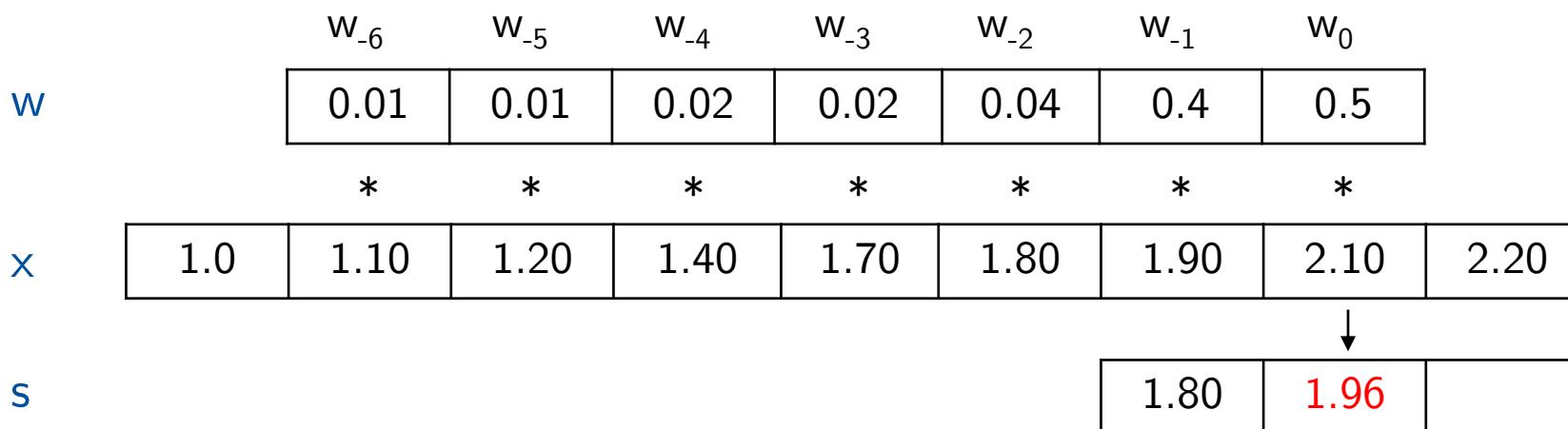
Content adapted from : CS7015 Deep Learning, Dept. of CSE, IIT Madras

The Convolution Operation - 1D

- In practice, we would sum only over a small window.

For example: $s_t = \sum_{a=0}^6 x_{t-a} w_{-a}$

- We just slide the filter over the input and compute the value of s_t based on a window around x_t



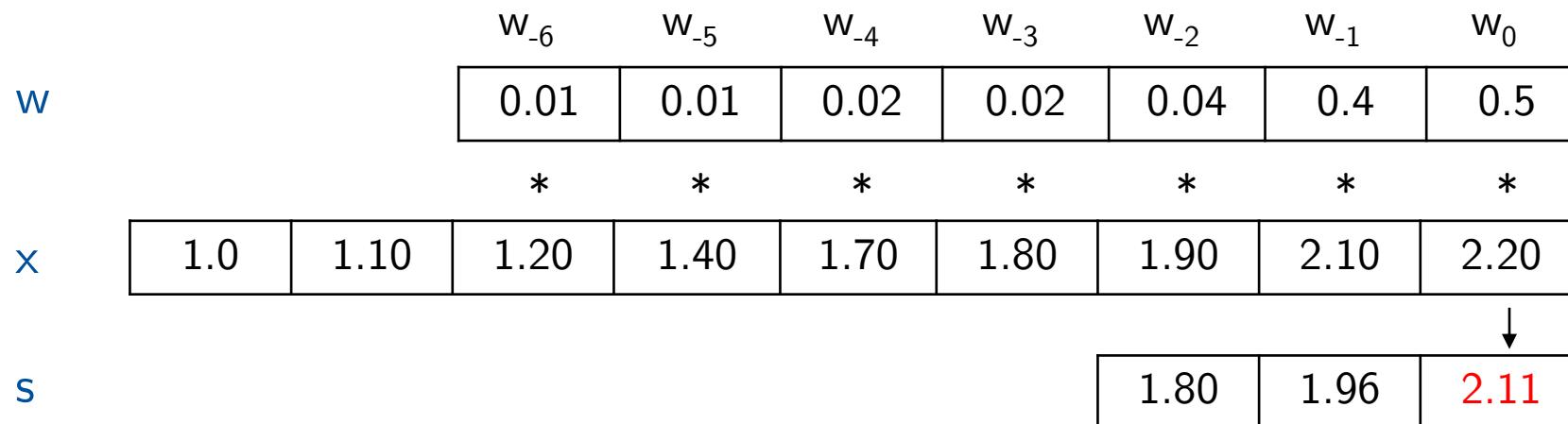
Content adapted from : CS7015 Deep Learning, Dept. of CSE, IIT Madras

The Convolution Operation - 1D

- In practice, we would sum only over a small window.

$$\text{For example: } s_t = \sum_{a=0}^6 x_{t-a} w_{-a}$$

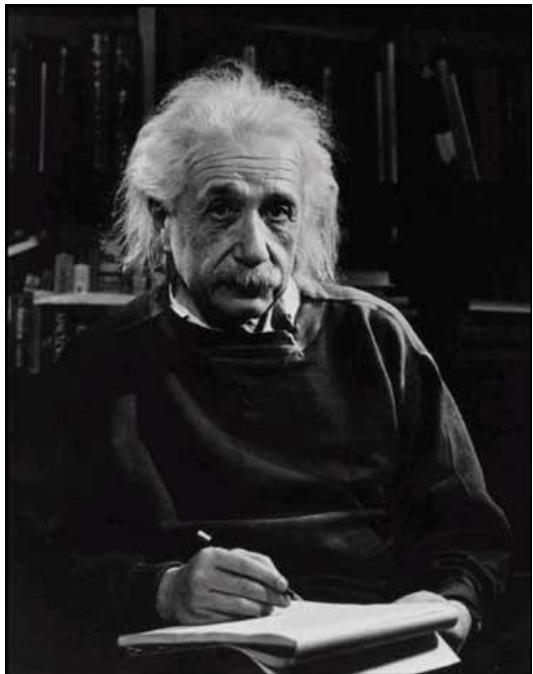
- We just slide the filter over the input and compute the value of s_t based on a window around x_t



- Use cases of 1-D convolution : Audio signal processing, stock market analysis, time series analysis etc.

Content adapted from : CS7015 Deep Learning, Dept. of CSE, IIT Madras

Convolution in 2-D using Images : What is an Image?



What we see

0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

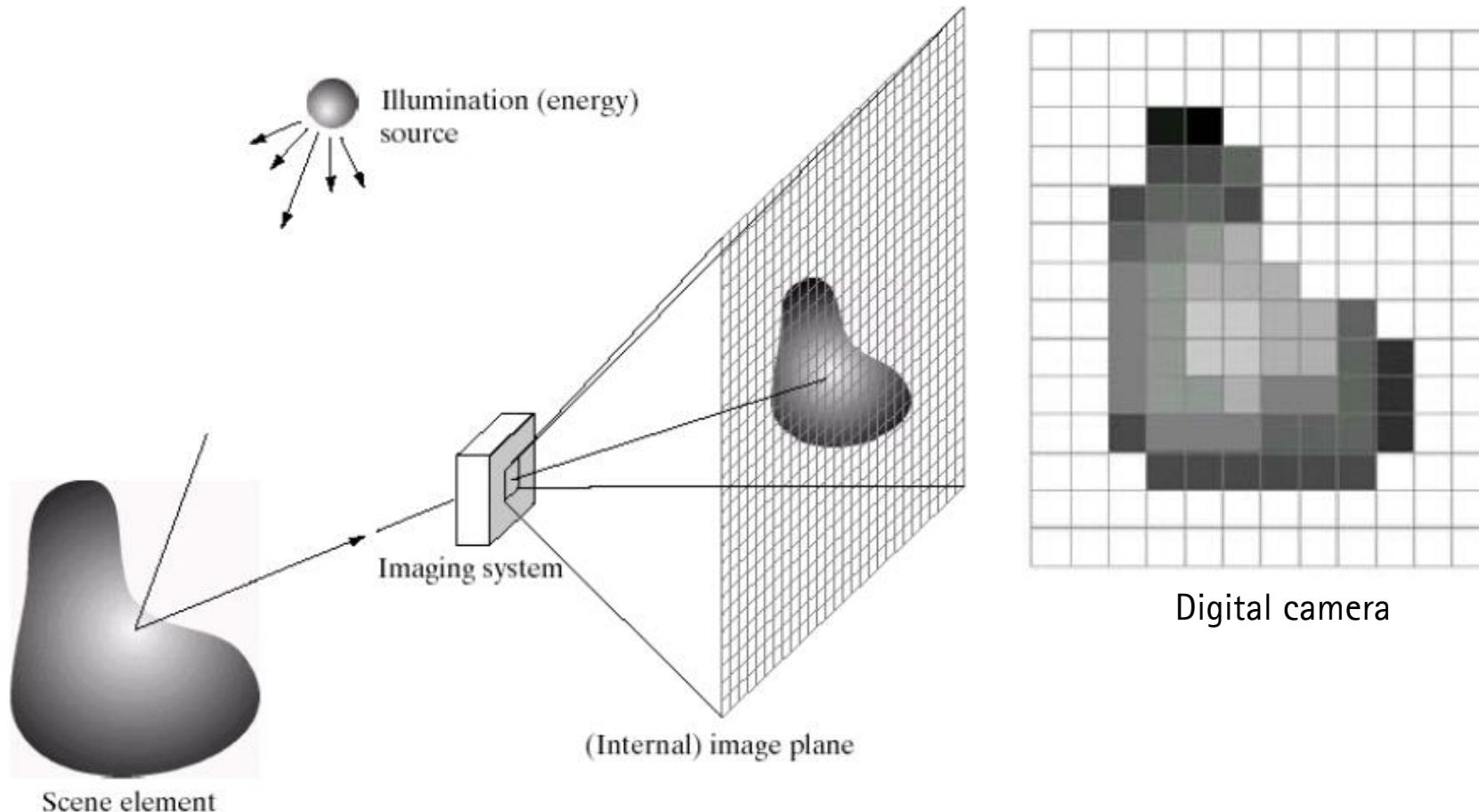
What a computer sees

Convolution in 2-D using Images : What is an Image?

- An image can be represented mathematically as a function $f(x,y)$ which gives the intensity value at position (x,y) , where, $f(x,y) \in \{0,1,\dots,I_{\max-1}\}$ and $x,y \in \{0,1,\dots,N-1\}$.
- Larger the value of N , more is the clarity of the picture (larger resolution), but more data to be analyzed in the image.
- If the image is a **Gray-scale** (8-bit per pixel) image, then it requires N^2 Bytes for storage.
- If the image is color - **RGB**, each pixel requires 3 Bytes of storage space.

N is the resolution of the image and I_{\max} is the level of discretized brightness value.

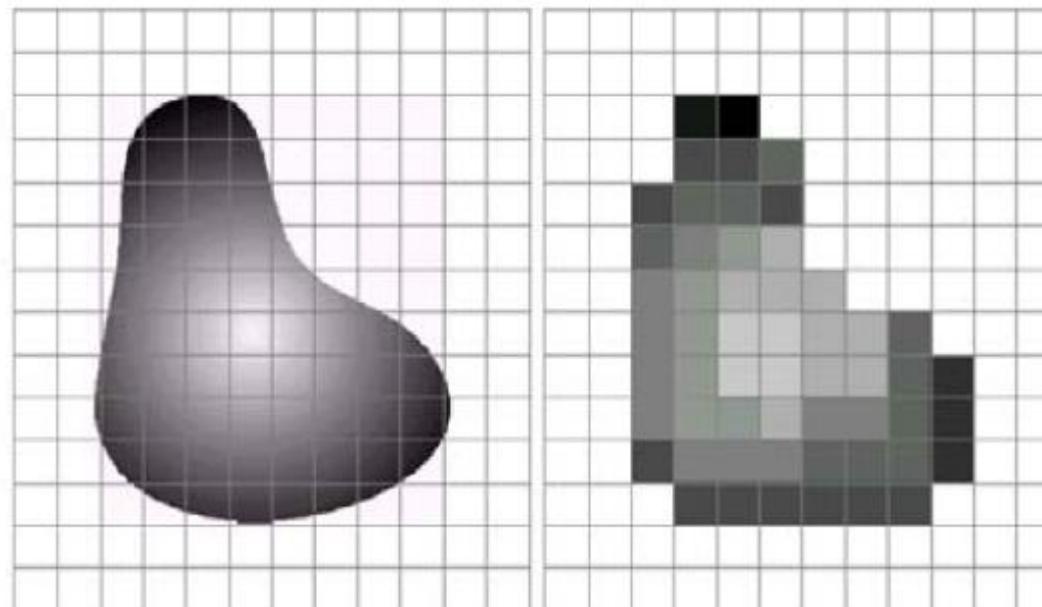
Convolution in 2-D using Images : What is an Image?



[Source: D. Hoiem]

Convolution in 2-D using Images : What is an Image?

- **Sample** the 2-D space on a regular grid.
- **Quantize** each sample, i.e., the photons arriving at each active cell are integrated and then digitized.



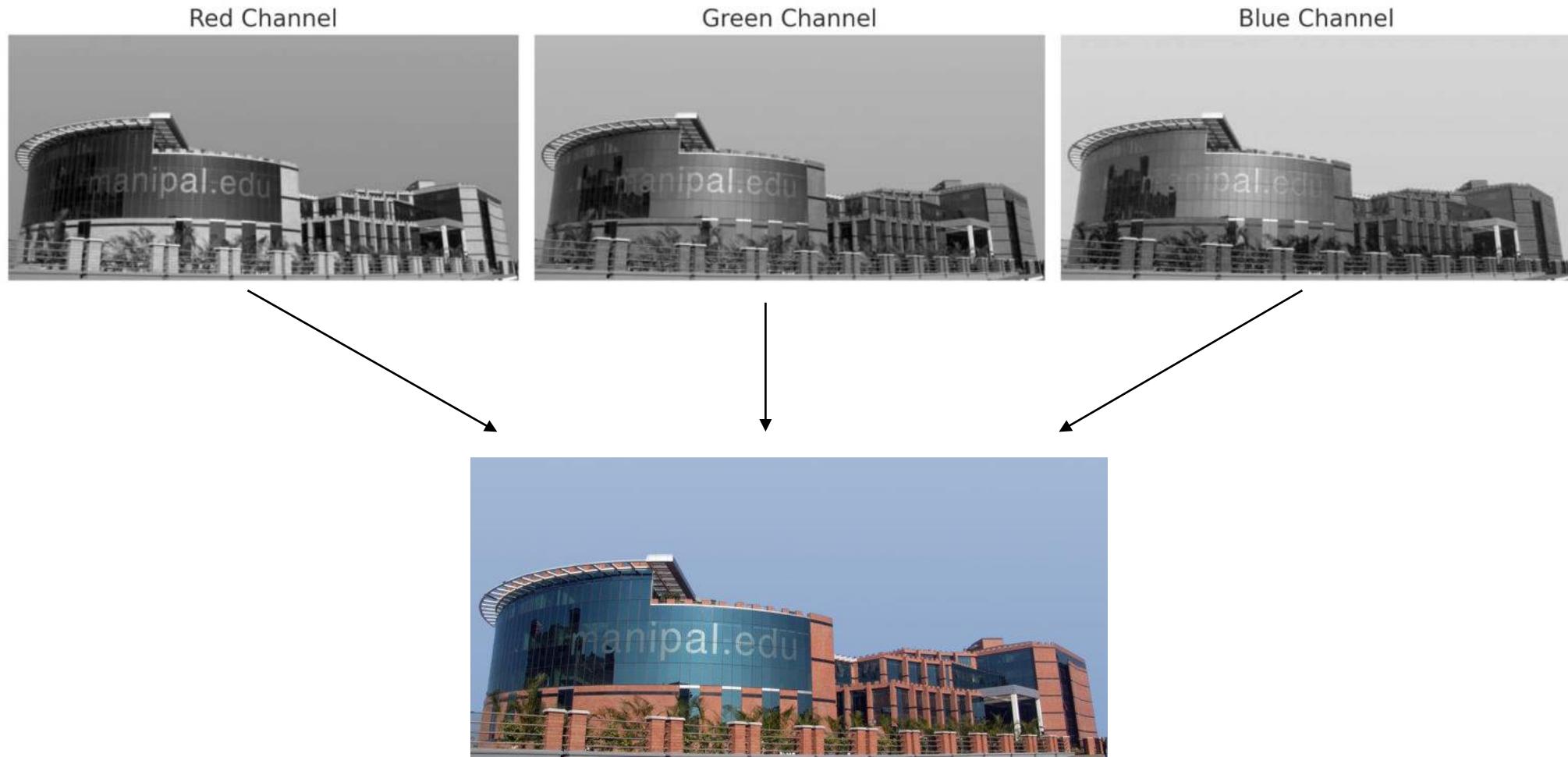
[Source: D. Hoiem]

Convolution in 2-D using Images : What is an Image?

- A grid (matrix) of intensity values.

255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	0	0	255	255	255	255	255	255	255	255	255	255
255	255	255	75	75	75	255	255	255	255	255	255	255	255	255
255	255	75	95	95	75	255	255	255	255	255	255	255	255	255
255	255	96	127	145	175	255	255	255	255	255	255	255	255	255
255	255	127	145	175	175	175	255	95	255	255	255	255	255	255
255	255	127	145	200	200	175	175	95	255	255	255	255	255	255
255	255	127	145	145	175	127	127	95	47	255	255	255	255	255
255	255	127	145	145	175	127	127	95	47	255	255	255	255	255
255	255	74	127	127	127	95	95	95	47	255	255	255	255	255
255	255	255	74	74	74	74	74	74	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255
255	255	255	255	255	255	255	255	255	255	255	255	255	255	255

Convolution in 2-D using Images : What is an Image?



The Convolution Operation - 2D

- Images are good examples of 2-D inputs.
- A 2-D convolution of an **Image 'I'** using a **filter 'K'** of size '**m x n**' is now defined as (looking at previous pixels):

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i-a,j-b} K_{a,b}$$

- In practice, one of the way is to look at the succeeding pixels:

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i+a,j+b} K_{a,b}$$

The Convolution Operation - 2D

- Another way is to consider center pixel as reference pixel, and then look at its surrounding pixels:

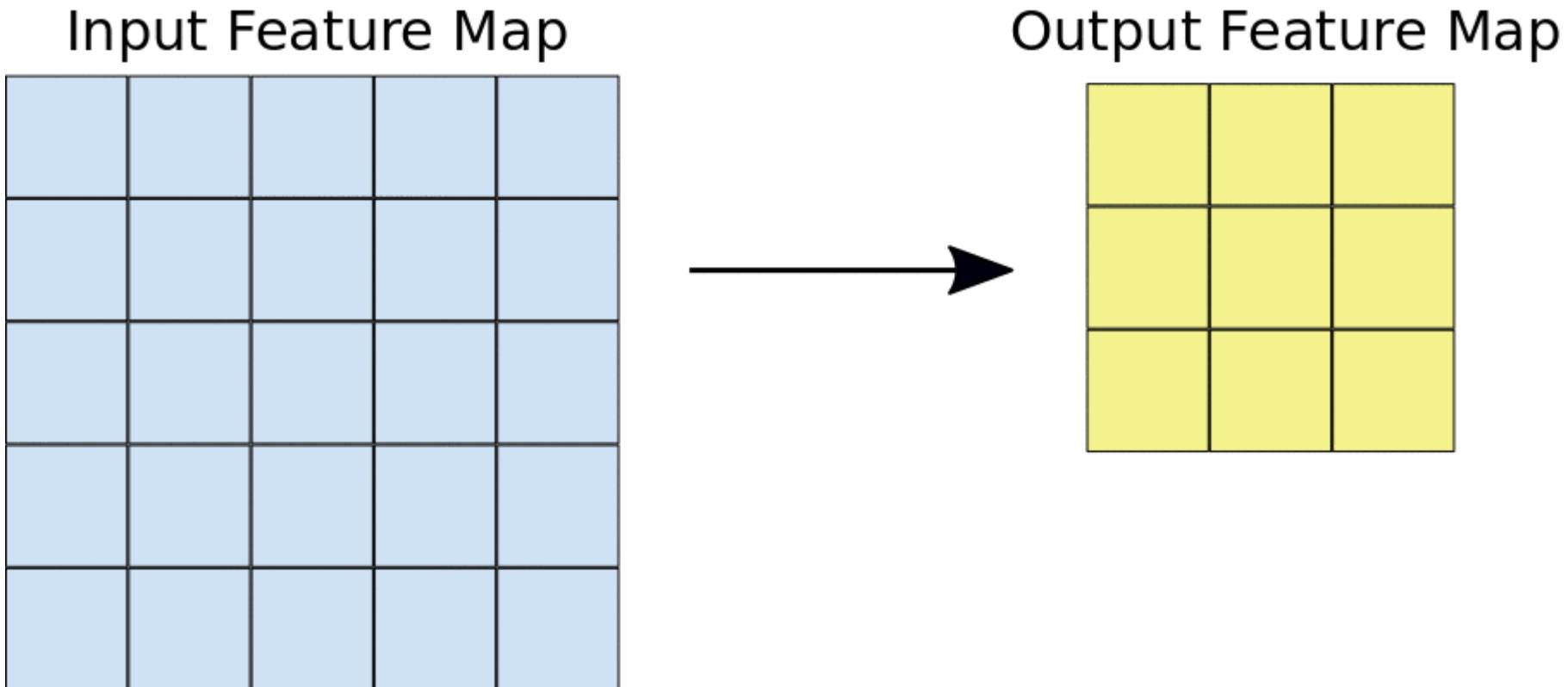
$$S_{ij} = (I * K)_{ij} = \sum_{a=[-m/2]}^{[m/2]} \sum_{b=[-n/2]}^{[n/2]} I_{i-a,j-b} * K_{(m/2)+a, (n/2)+b}$$

Pixel of interest

0	1	0	0	1
0	0	1	1	0
1	0	0	0	1
0	1	0	0	1
0	0	1	0	1

Content adapted from : CS7015 Deep Learning, Dept. of CSE, IIT Madras

The Convolution Operation - 2D



Source: <https://developers.google.com/>

The Convolution Operation - 2D

Input Image

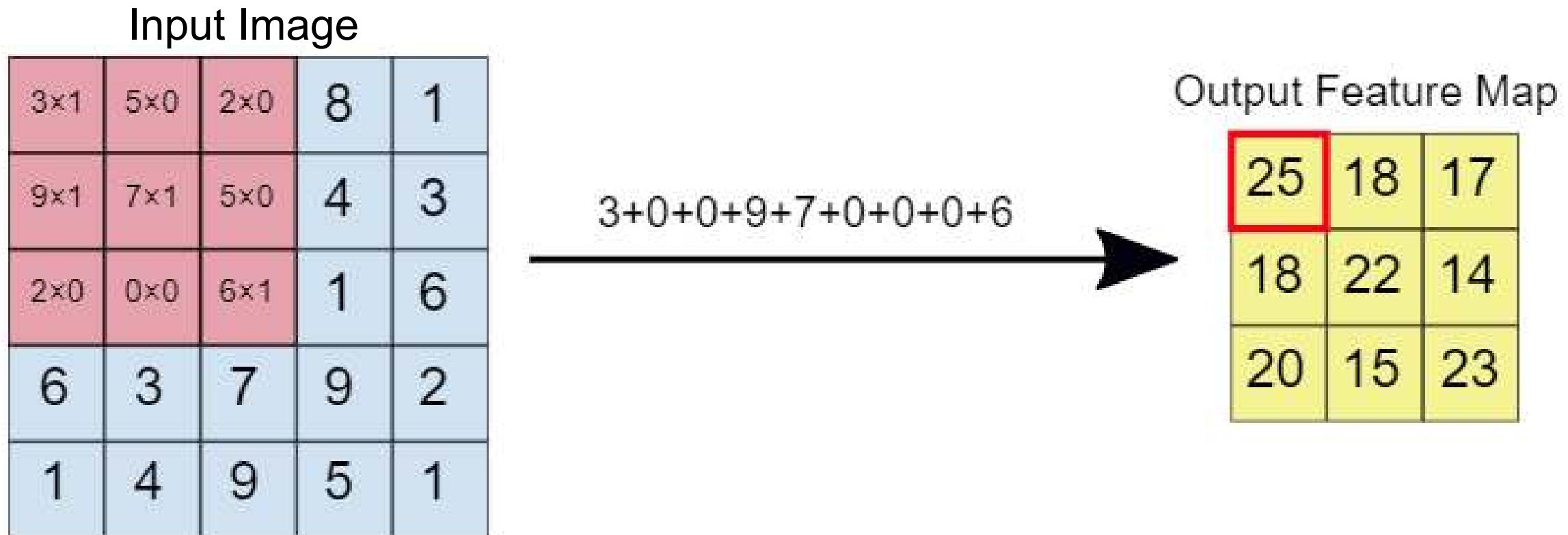
3	5	2	8	1
9	7	5	4	3
2	0	6	1	6
6	3	7	9	2
1	4	9	5	1

Convolutional Filter

1	0	0
1	1	0
0	0	1

Source: <https://developers.google.com/>

The Convolution Operation - 2D



Source: <https://developers.google.com/>

The Convolution Operation - 2D



$$\begin{matrix} & 1 & 1 & 1 \\ * & 1 & 1 & 1 = \\ & 1 & 1 & 1 \end{matrix}$$



Smoothening Filter

The Convolution Operation - 2D



$$* \begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix} =$$



Sharpening Filter

The Convolution Operation - 2D



$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

Filter for edge
detection



The Convolution Operation – 2D : Various filters (edge detection)

Prewitt

-1	0	1
-1	0	1
-1	0	1

S_x

1	1	1
0	0	0
-1	-1	-1

S_y



Input image



After applying
Horizontal edge
detection filter

Sobel

-1	0	1
-2	0	2
-1	0	1

S_x

1	2	1
0	0	0
-1	-2	-1

S_y



After applying
Vertical edge
detection filter

Laplacian

0	1	0
1	-4	1
0	1	0

Roberts

0	1
-1	0

S_x

1	0
0	-1

S_y

The Convolution Operation - 2D

stride=1

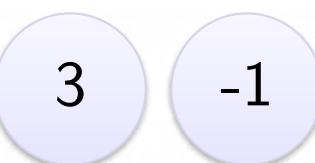
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input image: 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

Dot
product



Note: Stride is the number of “unit” the kernel is shifted per slide over rows/ columns

The Convolution Operation - 2D

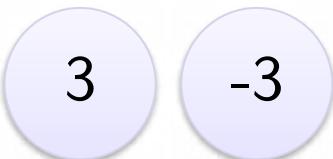
If stride=2

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input image: 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



Note: Stride is the number of “unit” the kernel is shifted per slide over rows/ columns

The Convolution Operation - 2D

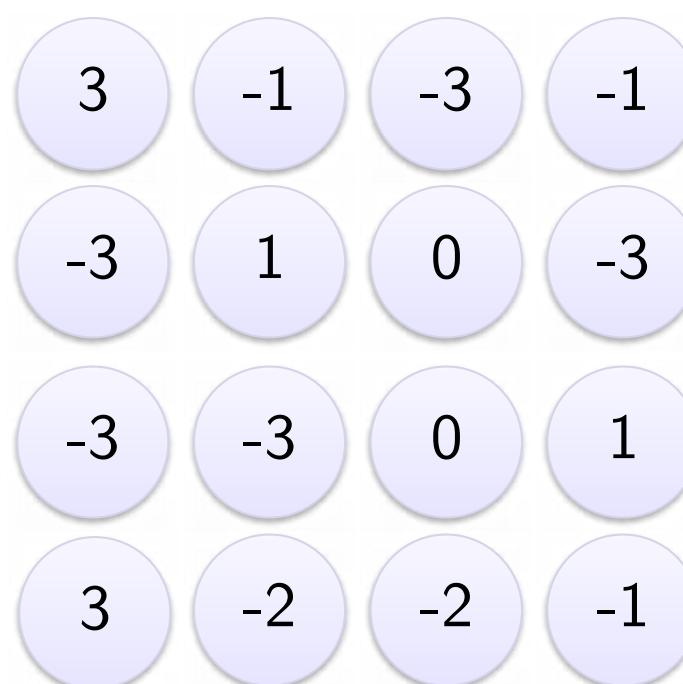
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input image: 6×6

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1



4×4 Feature Map

The Convolution Operation - 2D

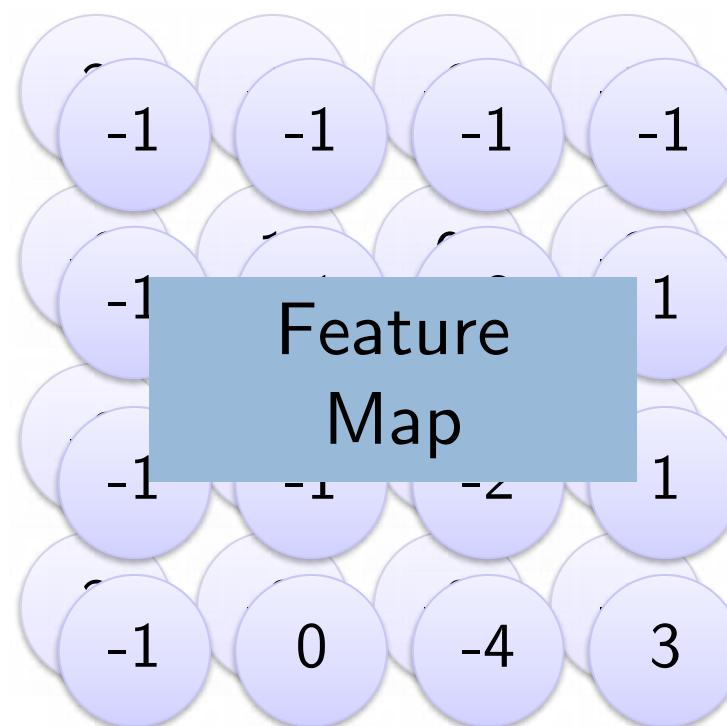
stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Input image: 6×6

-1	1	-1
-1	1	-1
-1	1	-1

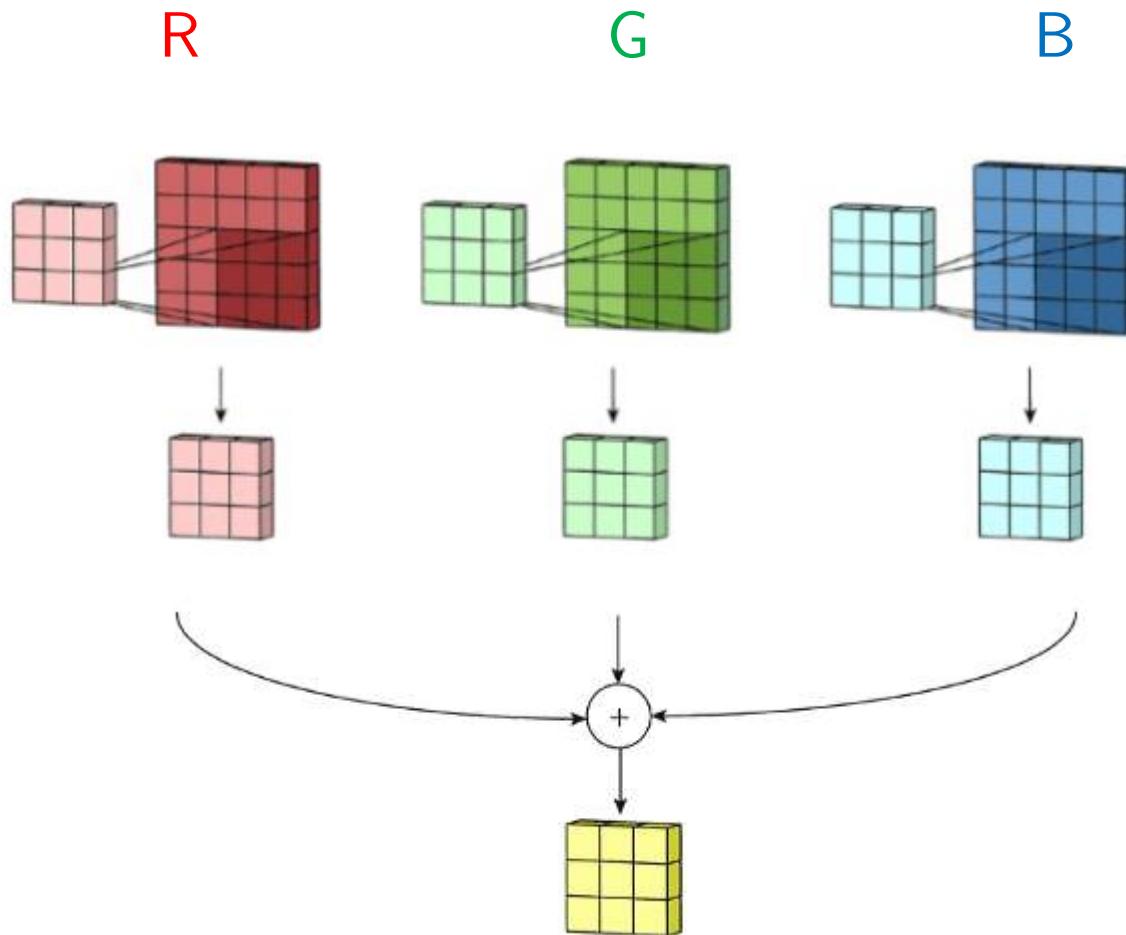
Filter 2



Repeat for each filter!

Two 4×4 images
Forming $4 \times 4 \times 2$ matrix

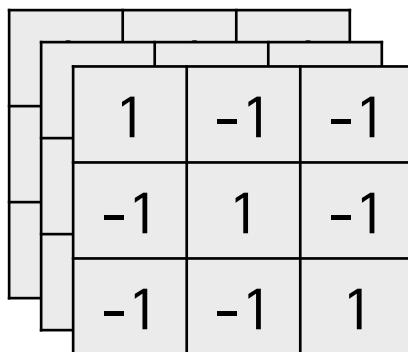
The Convolution Operation –RGB Images



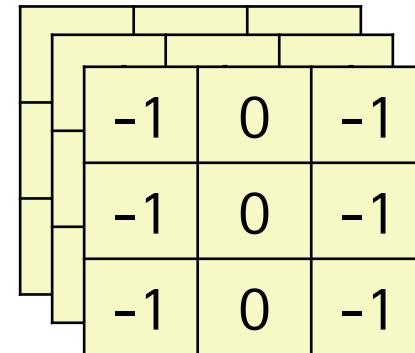
Apply the filter to R, G, and B channels of the image and combine the resultant feature maps to obtain a 2-D feature map.

Source: Intuitively Understanding Convolutions for Deep Learning | by Irhum Shafkat | Towards Data Science

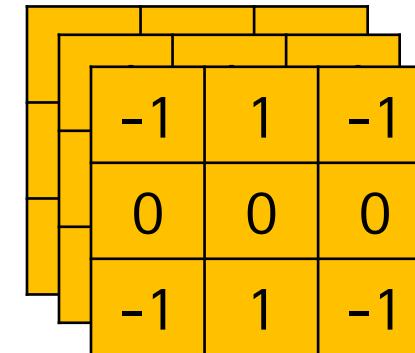
The Convolution Operation –RGB Images multiple filters



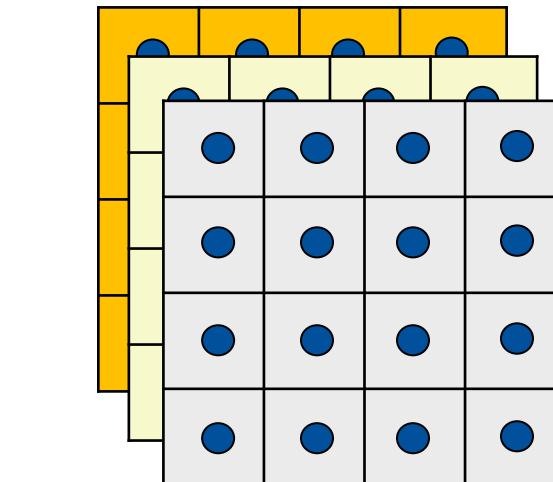
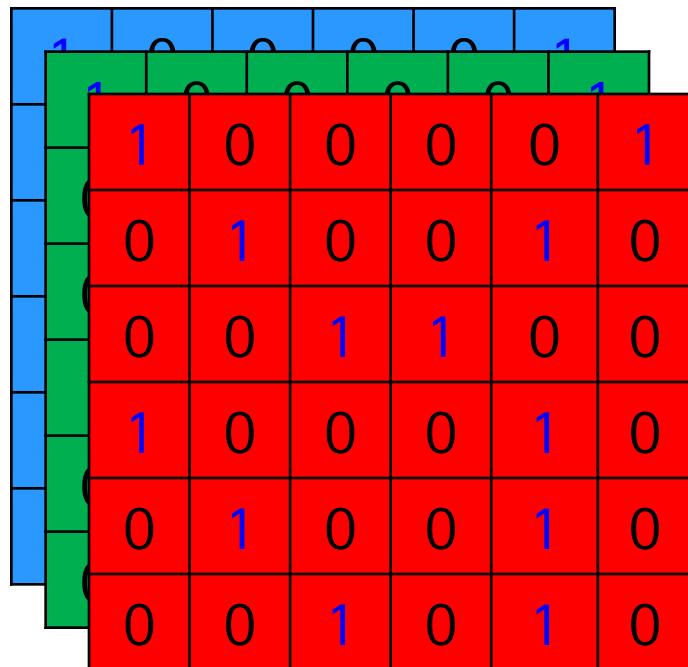
Filter 1



Filter 2



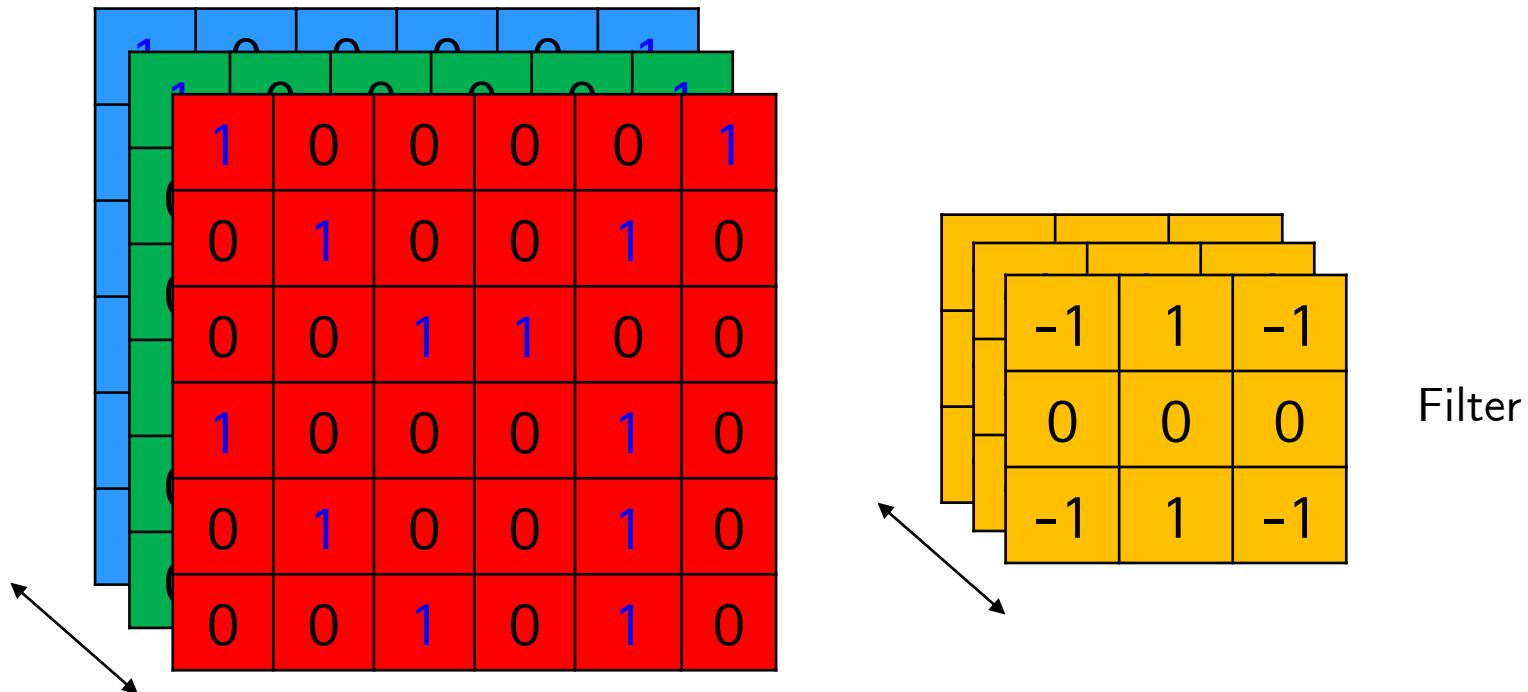
Filter K



K-filters = K-Feature Maps

Depth of feature map = No. of feature maps = No. of filters

The Convolution Operation : Terminologies



1. Depth of an Input Image = No. of channels in the Input Image = Depth of a filter
2. Assuming square filters, Spatial Extent (F) of a filter is the size of the filter

The Convolution Operation : Zero Padding



conv_{3x3}

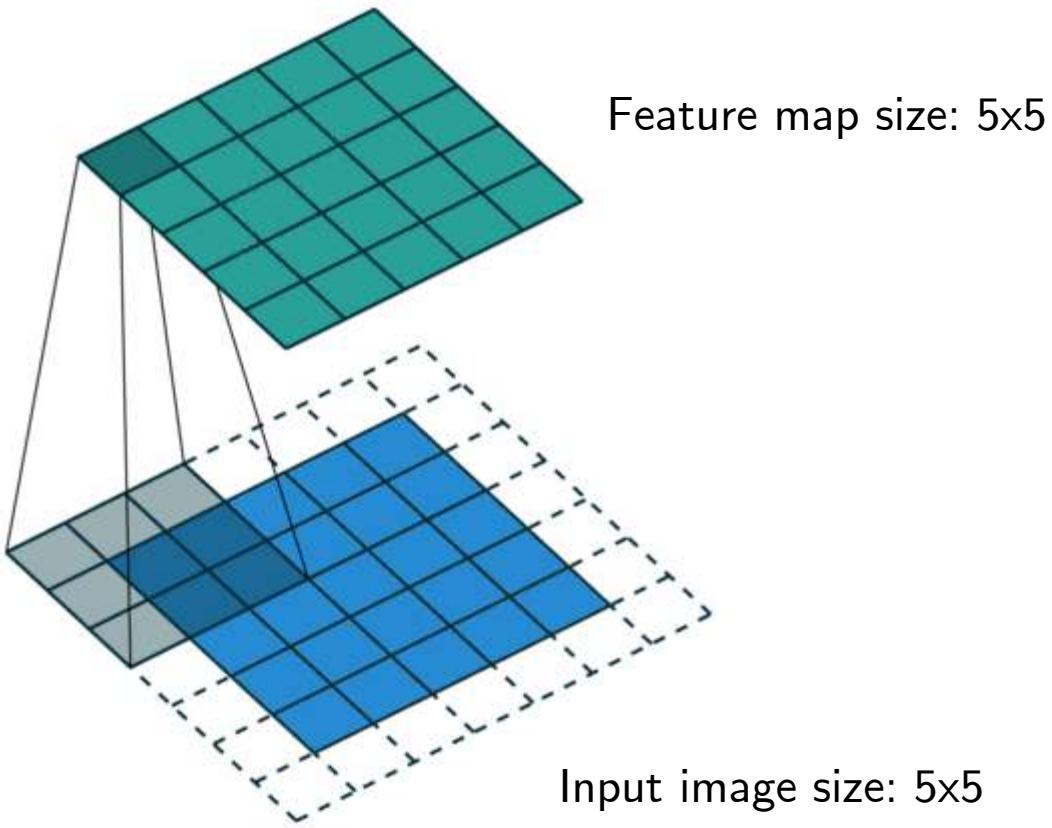
4x4

2x2

0	0	0	0	0	0
0	35	19	25	6	0
0	13	22	16	53	0
0	4	3	7	10	0
0	9	8	1	3	0
0	0	0	0	0	0

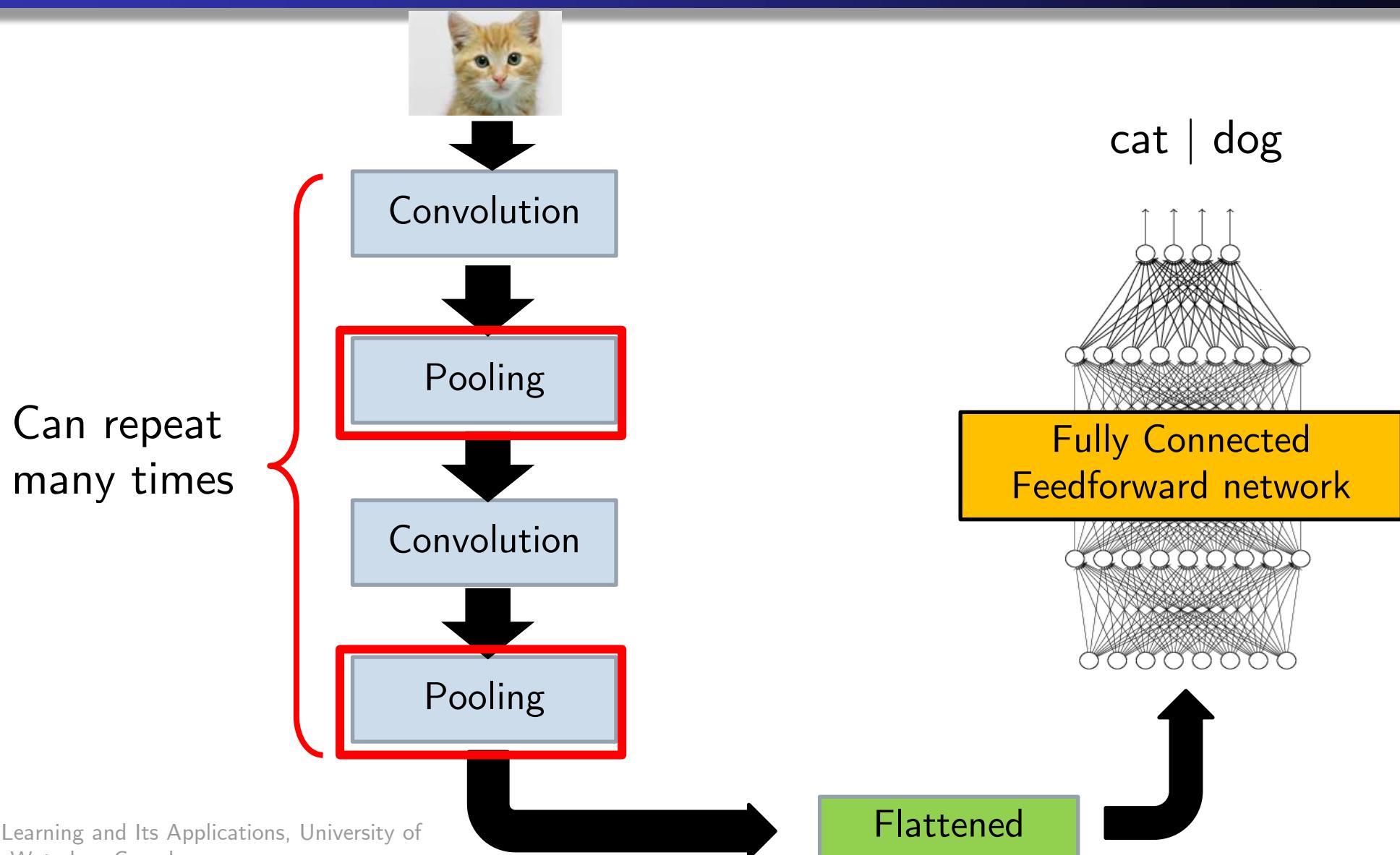
Pad Zeros and then convolve to obtain a feature map with dimension = input image dimension

The Convolution Operation : Zero Padding



[Source: Intuitively Understanding Convolutions for Deep Learning | by Irhum Shafkat | Towards Data Science](#)

Convolutional Neural Network (CNN) : At a glance



Source: CS 898: Deep Learning and Its Applications, University of Waterloo, Canada.

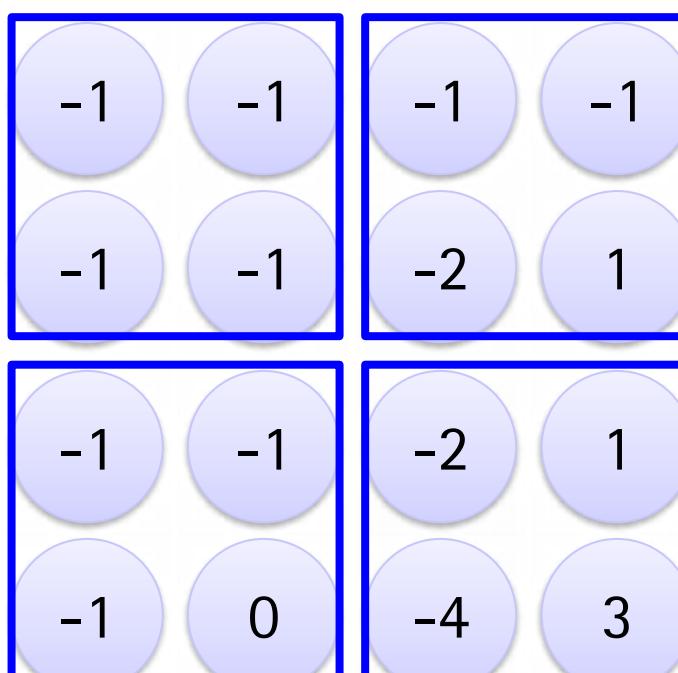
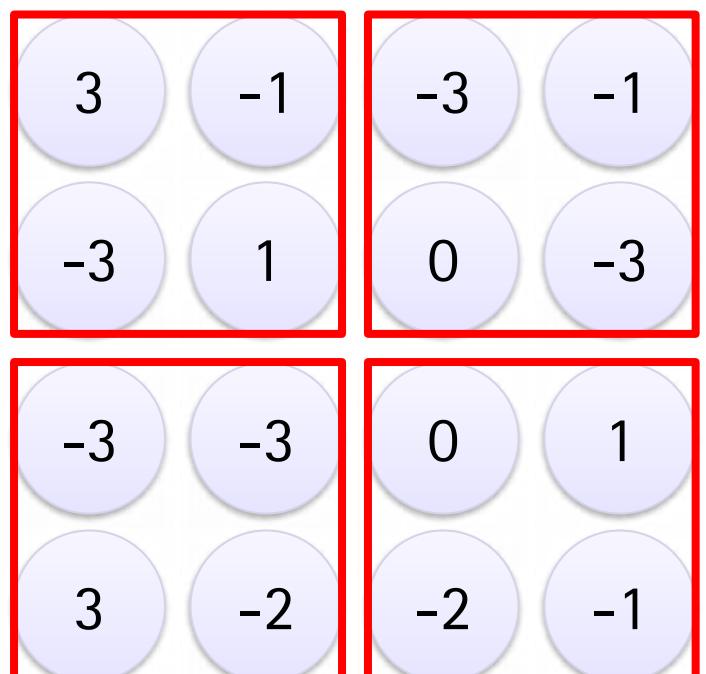
Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

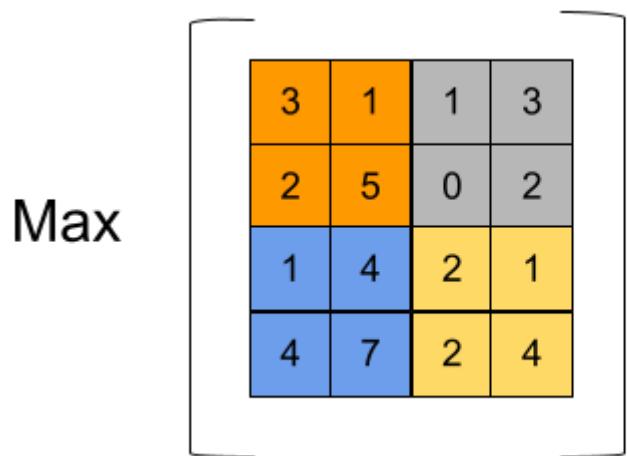
Filter 2



- Max Pooling
- Average Pooling

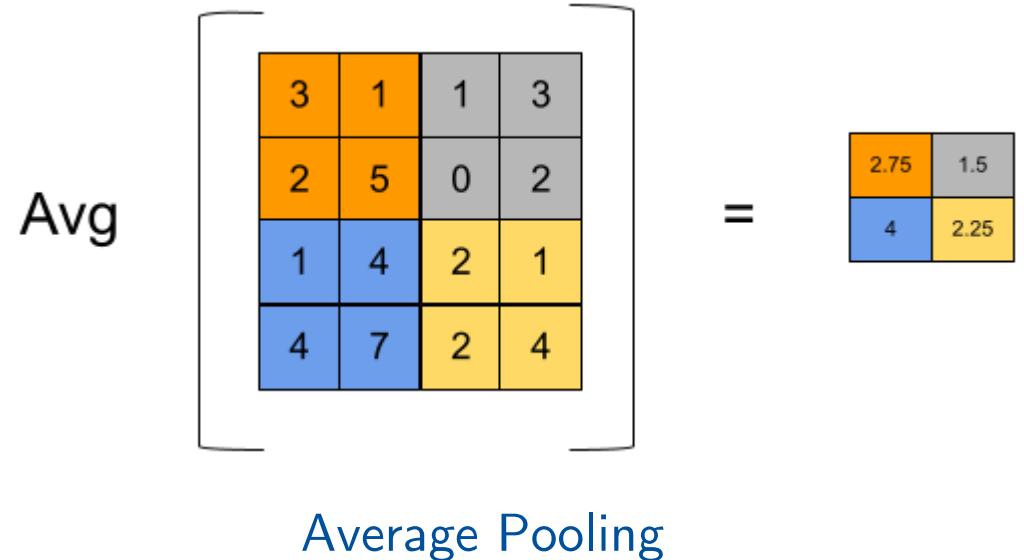
Source: CS 898: Deep Learning and Its Applications, University of Waterloo, Canada.

Pooling



$$= \begin{matrix} 5 & 3 \\ 7 & 4 \end{matrix}$$

Max. Pooling

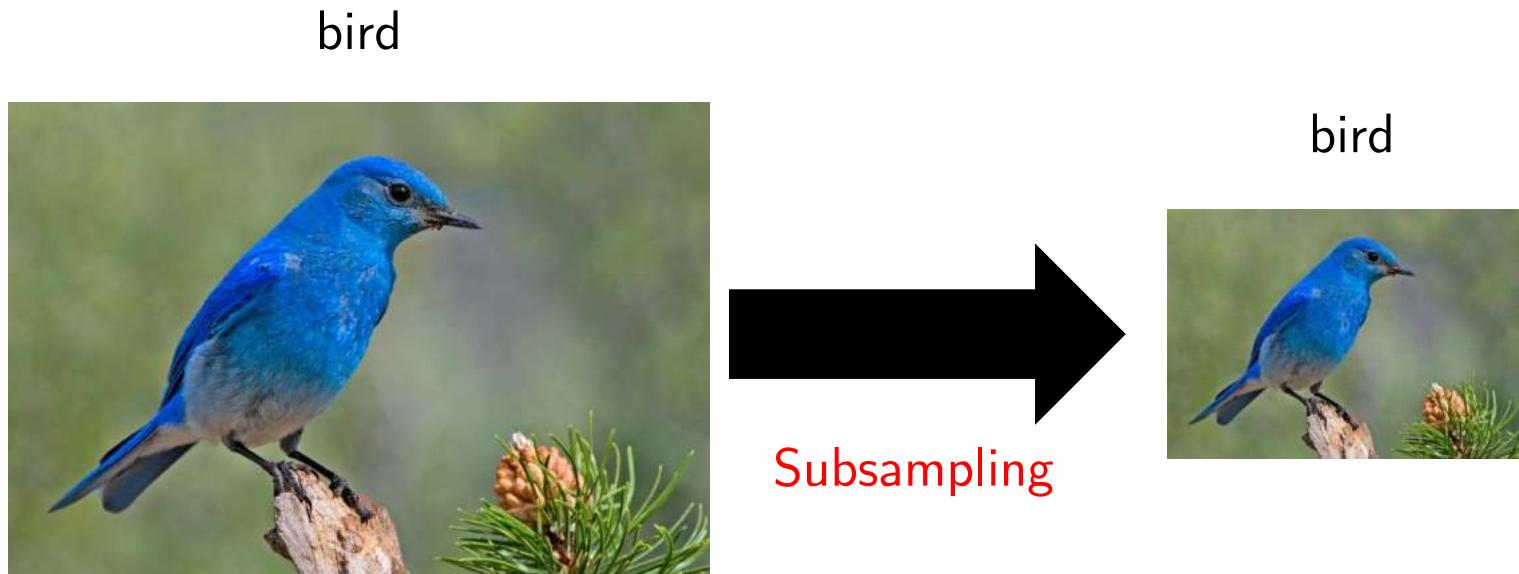


Average Pooling

Stride ?

Why Pooling ?

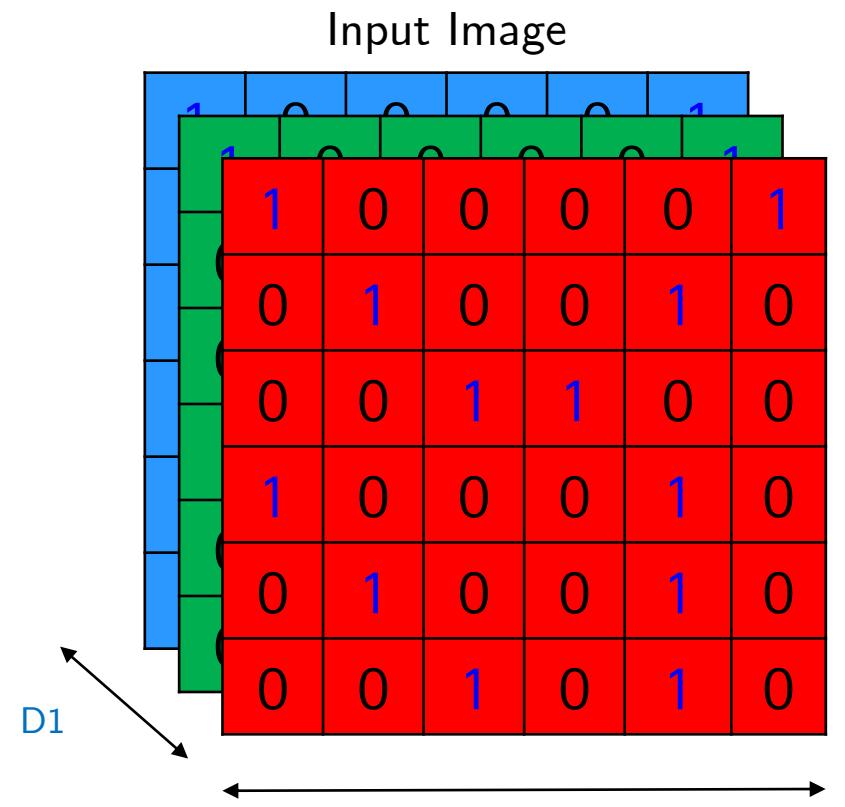
- Subsampling pixels will not change the object



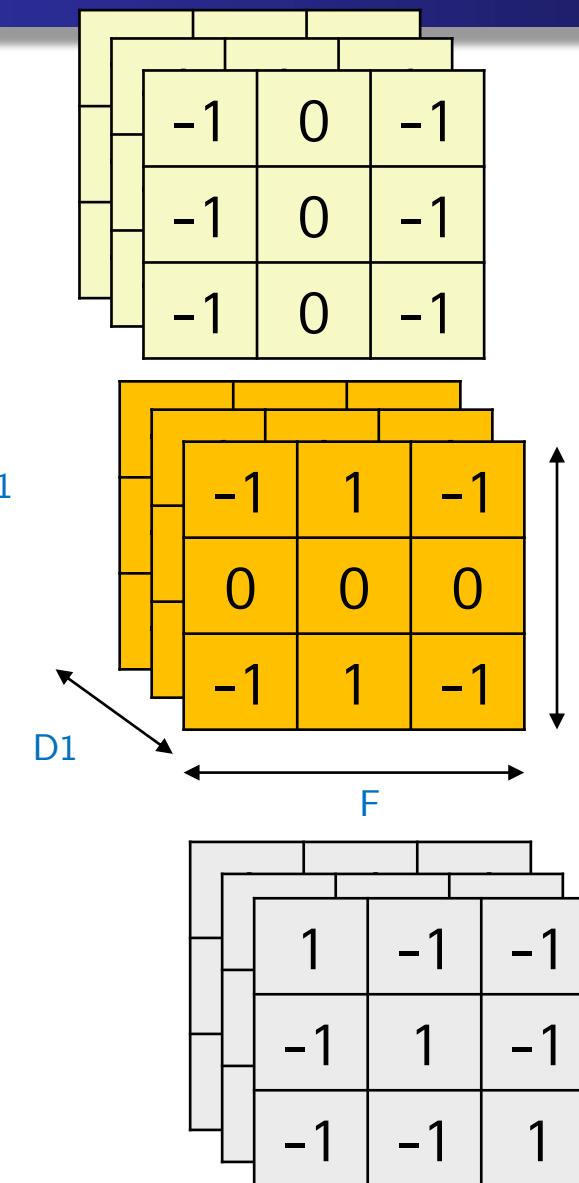
- We can subsample the pixels to make image smaller
- Therefore, fewer parameters to characterize the image

Source: CS 898: Deep Learning and Its Applications, University of Waterloo, Canada.

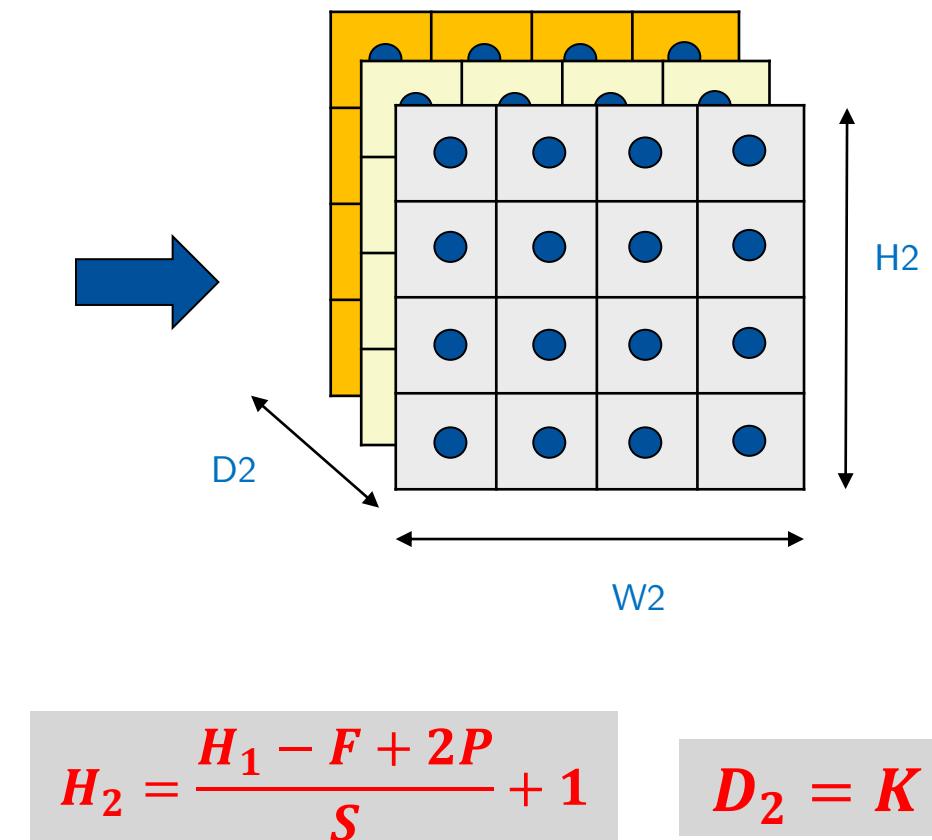
Relation between i/p size, feature map size, filter size



$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$



Stride length = S
No. of Filters = K
Padding = P



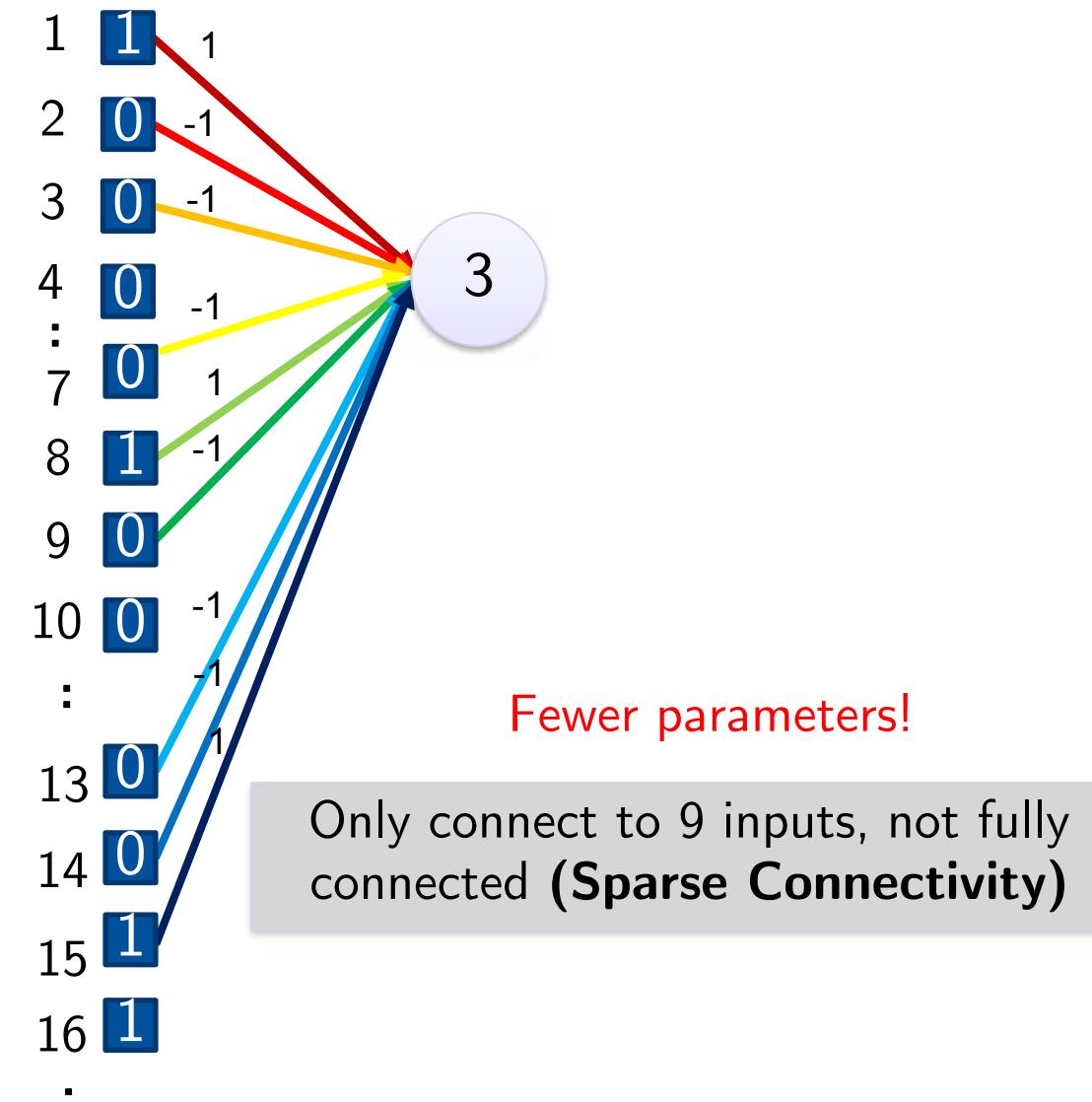
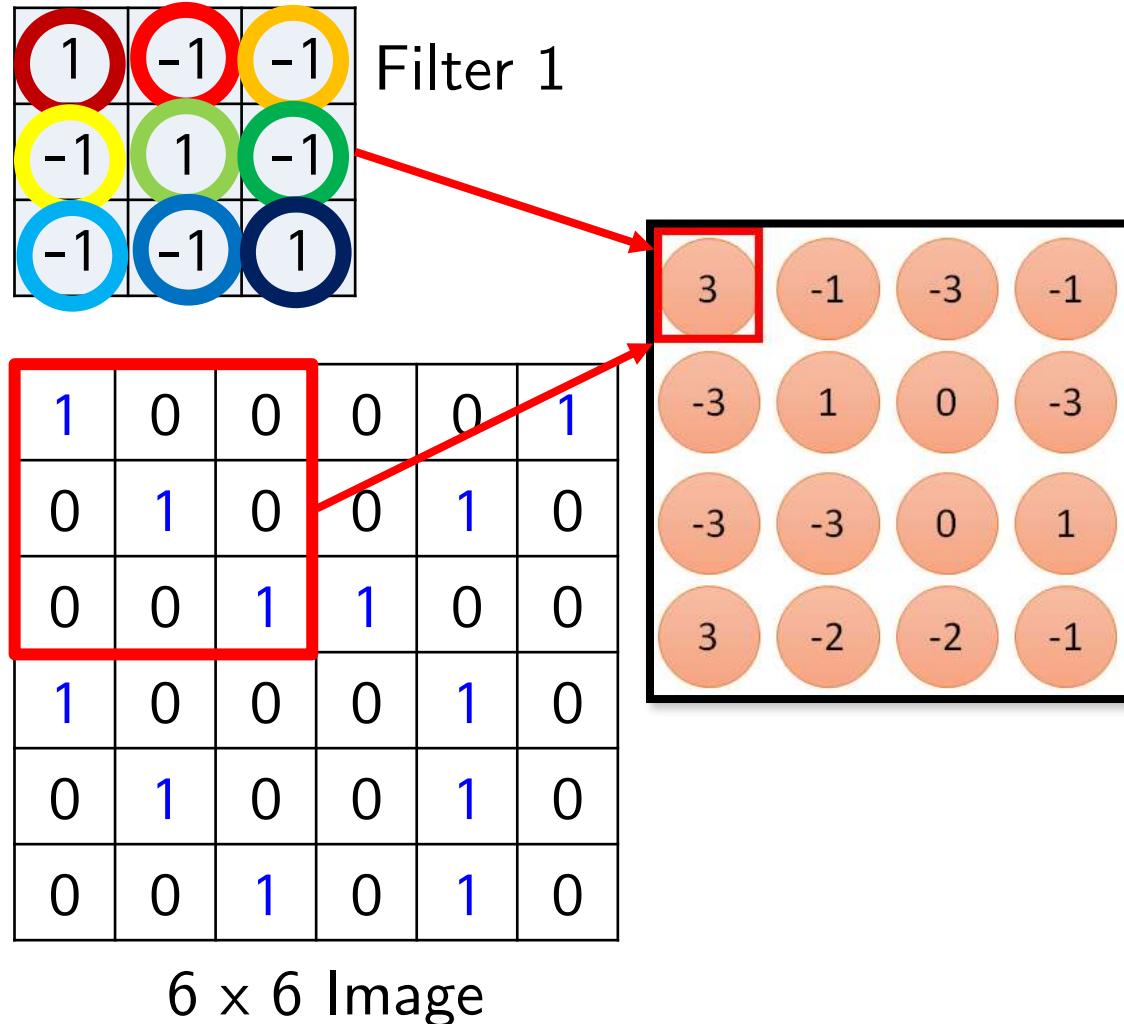
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$D_2 = K$$

Important properties of CNN

- Sparse Connectivity
- Shared weights
- Equivariant representation

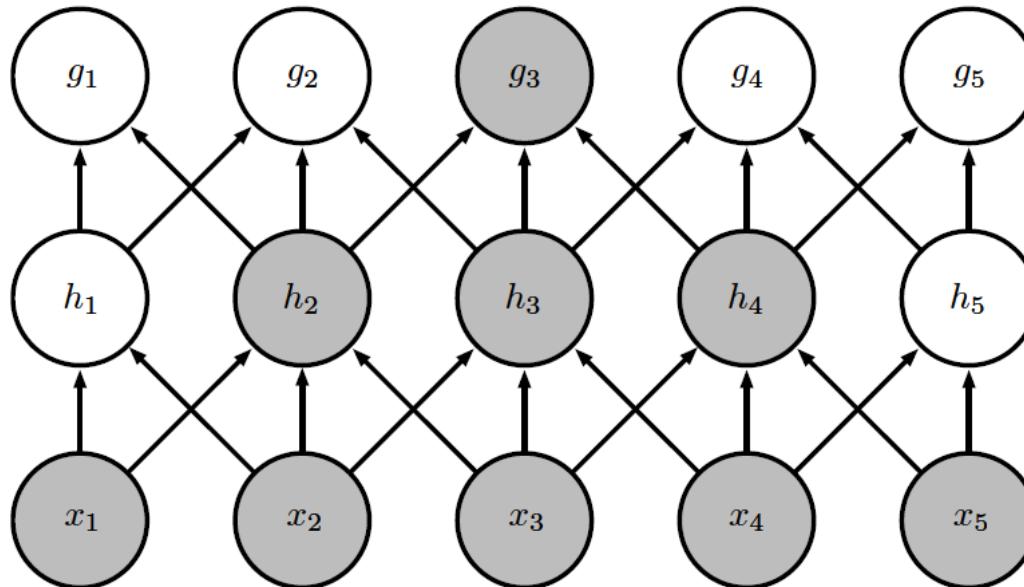
Properties of CNN



Source: CS 898: Deep Learning and Its Applications, University of Waterloo, Canada.

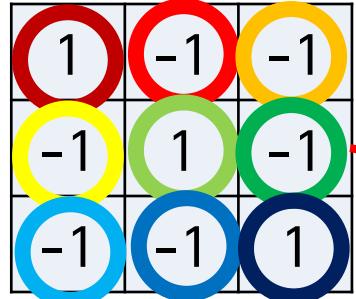
Properties of CNN

Is sparse connectivity good?



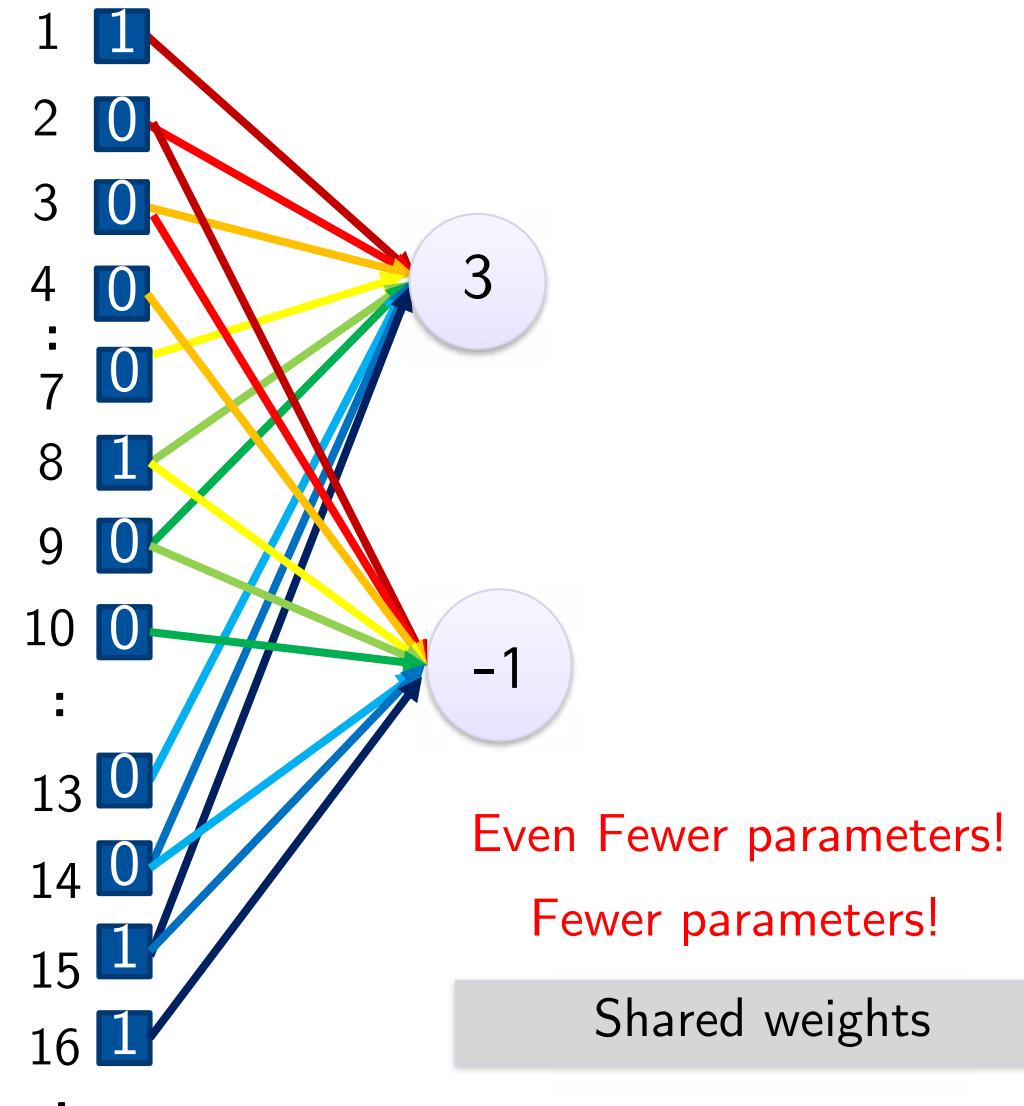
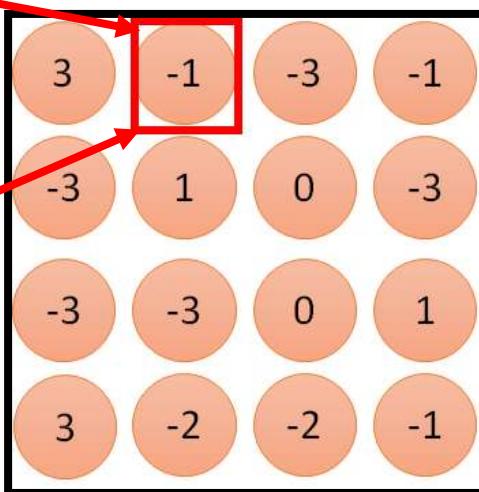
Ian Goodfellow et al. 2016

Properties of CNN



1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

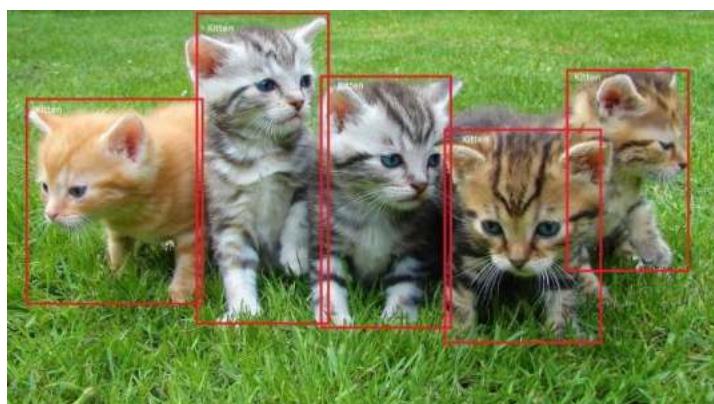
6 x 6 Image



Source: CS 898: Deep Learning and Its Applications, University of Waterloo, Canada.

Equivariance to translation

- A function f is equivariant to a function g if $f(g(x)) = g(f(x))$ or if the output changes in the same way as the input.
- This is achieved by the concept of weight sharing.
- As the same weights are shared across the images, hence if an object occurs in any image, it will be detected irrespective of its position in the image.

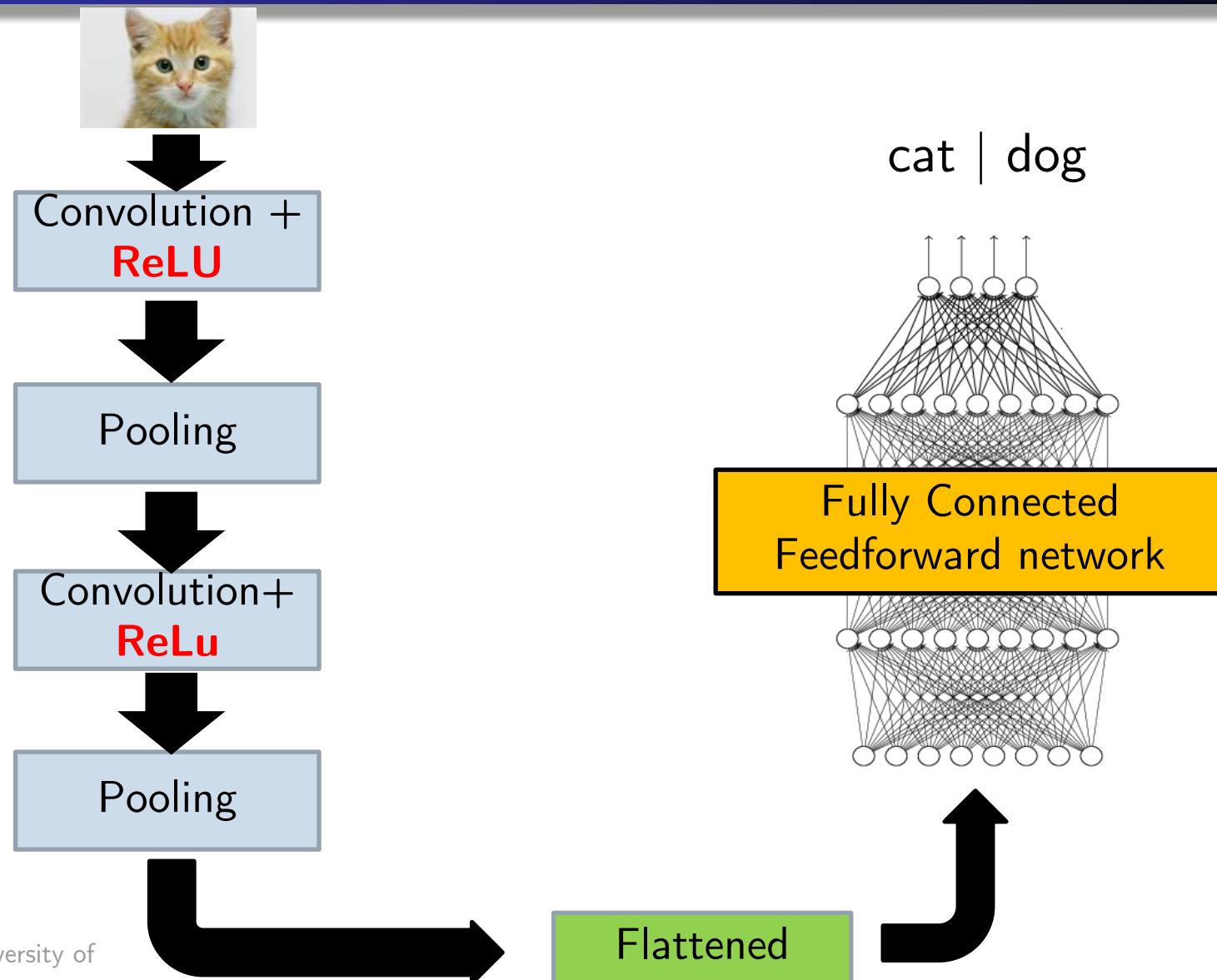


Source: [Translational Invariance Vs Translational Equivariance | by Divyanshu Mishra | Towards Data Science](#)

CNN vs Fully Connected NN

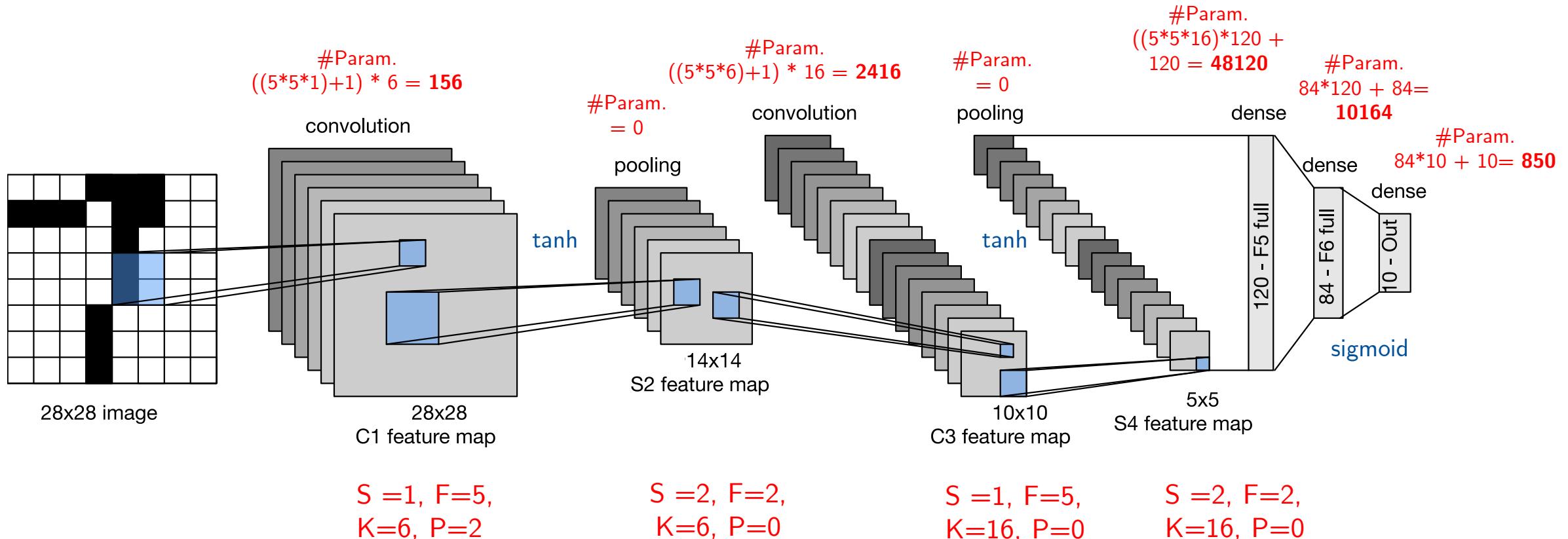
- A CNN compresses the fully connected NN in two ways:
 - Reducing the number of connections
 - Shared weights
- Max pooling further reduces the parameters to characterize an image.

Convolutional Neural Network (CNN) : Non-linearity with activation



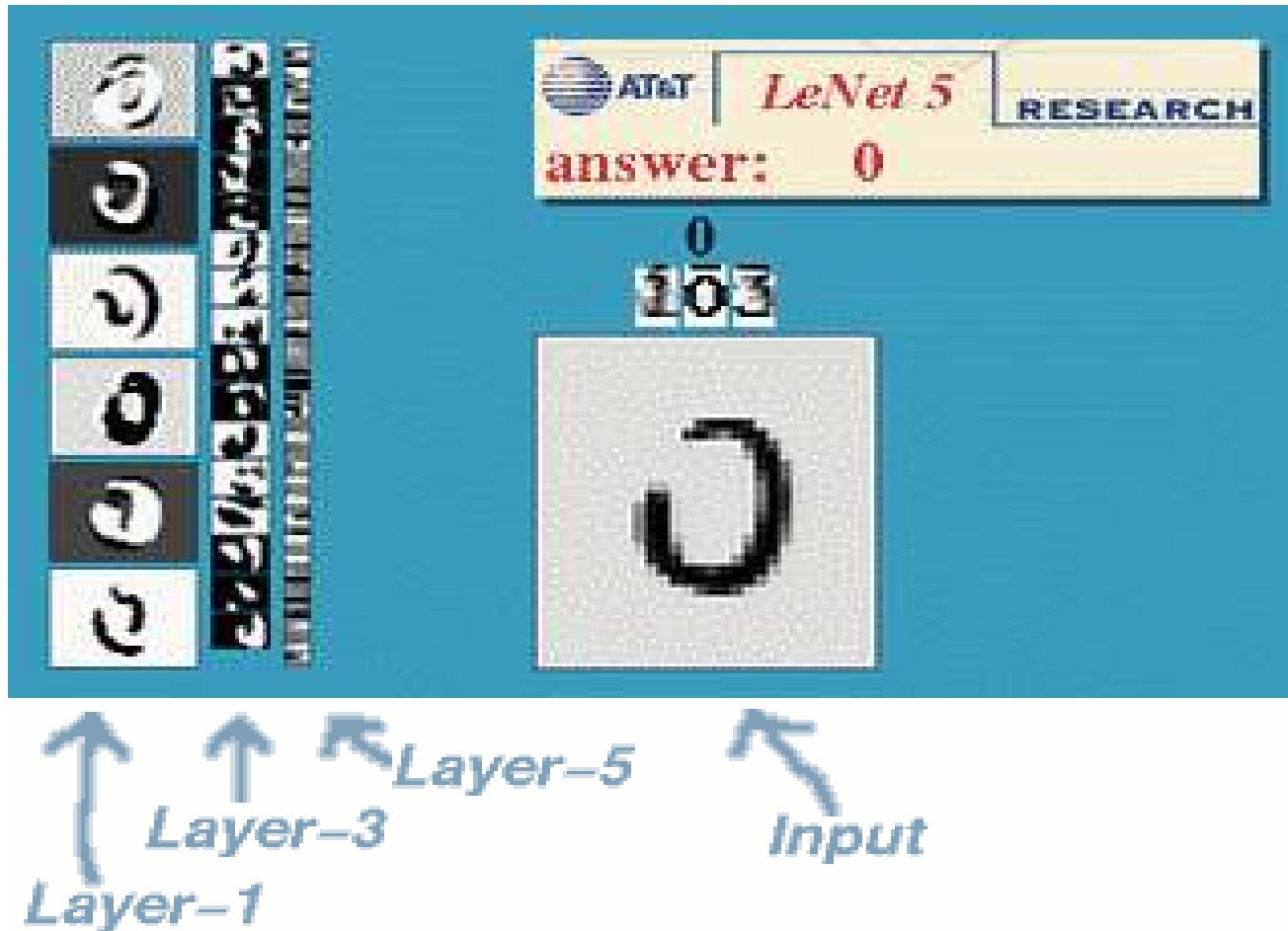
Source: CS 898: Deep Learning and Its Applications, University of Waterloo, Canada.

LeNet-5 Architecture for handwritten text recognition



LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., & others. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

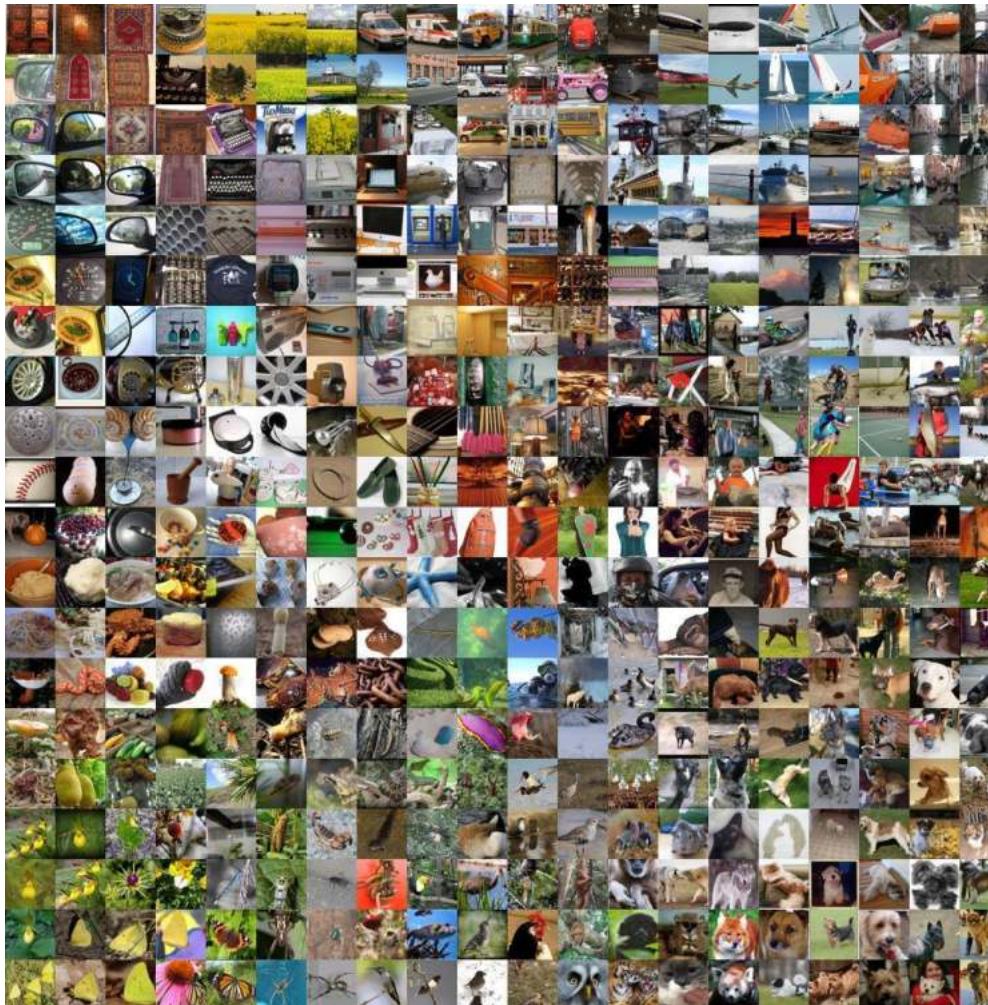
LeNet-5 Architecture for handwritten number recognition



Source: <http://yann.lecun.com/>

ImageNet Dataset

More than 14 million images.

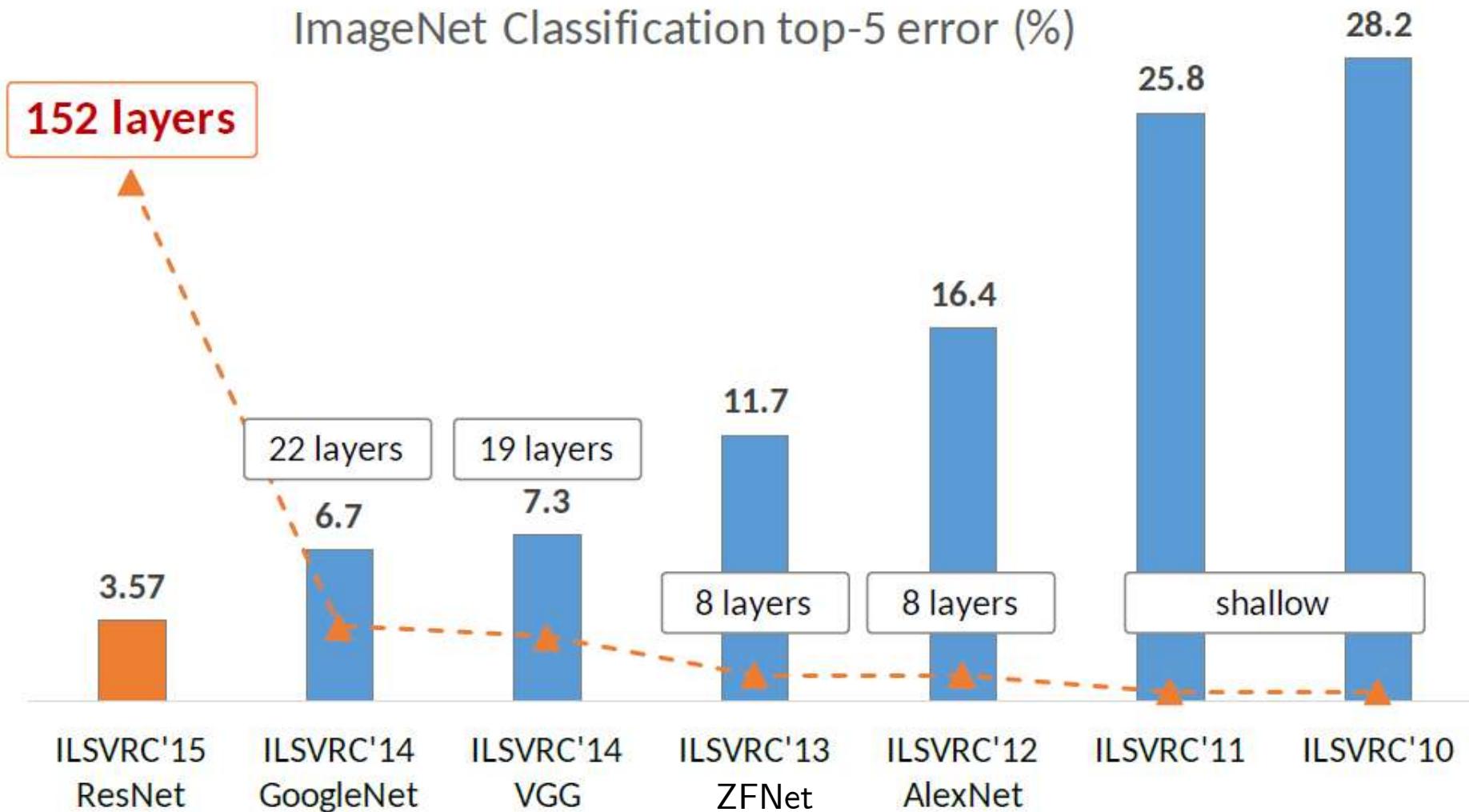


22,000 Image categories

Deng, Jia, et al. "Imagenet: A large-scale hierarchical image database." *IEEE conference on computer vision and pattern recognition*. IEEE, 2009.

ImageNet Large Scale Visual Recognition Challenge

- 1000 ImageNet Categories



ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

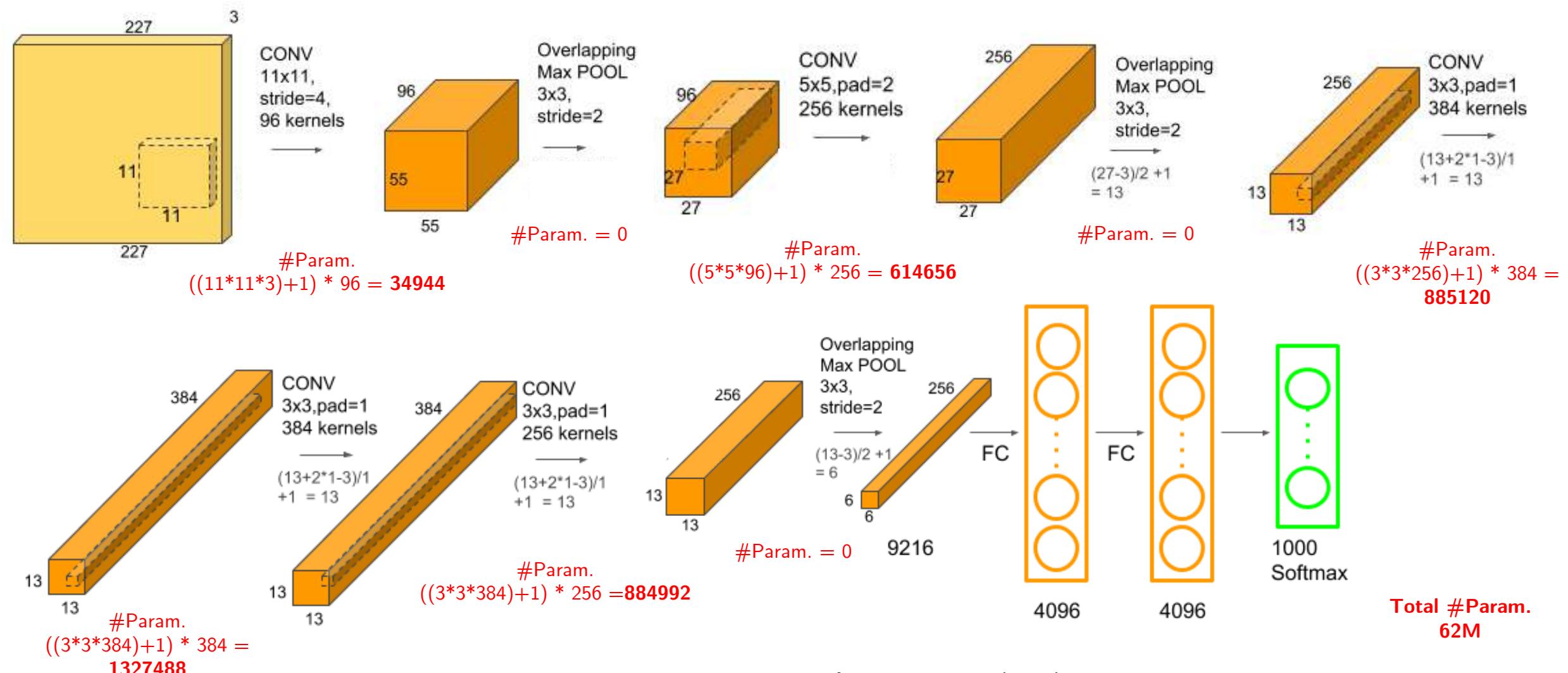
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called “dropout” that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

- Used **ReLU** activation function instead of sigmoid and tanh.
- Used **data augmentation** techniques that consisted of image translations, horizontal reflections, and patch extractions.
- Implemented **dropout** layers.

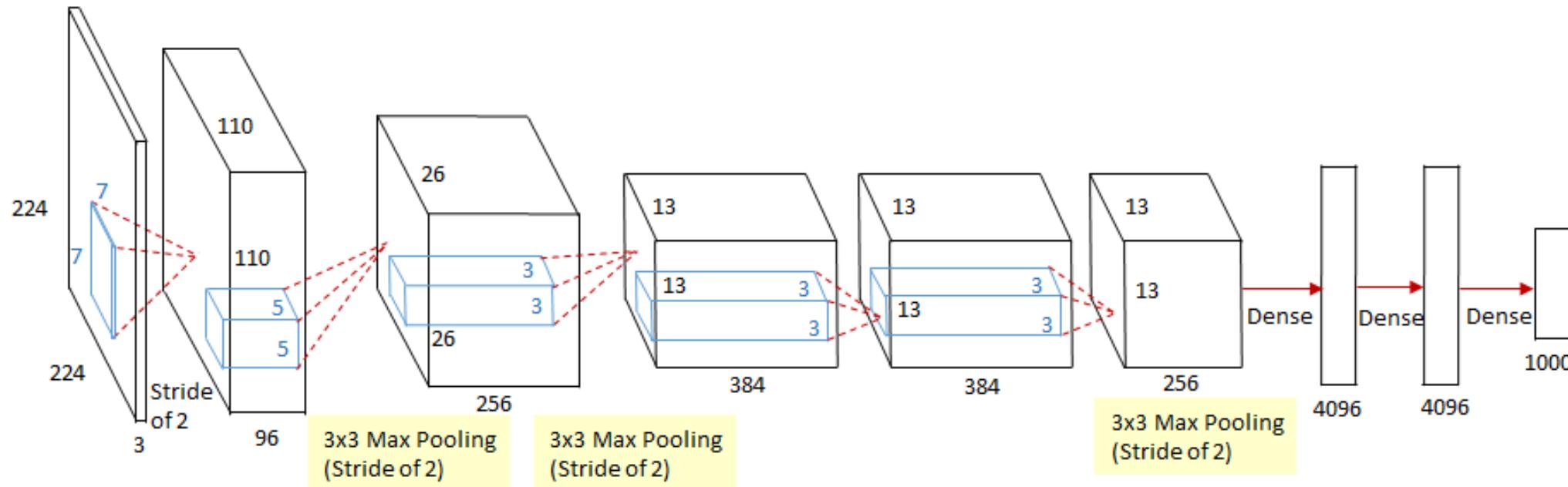
AlexNet Architecture



Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012).

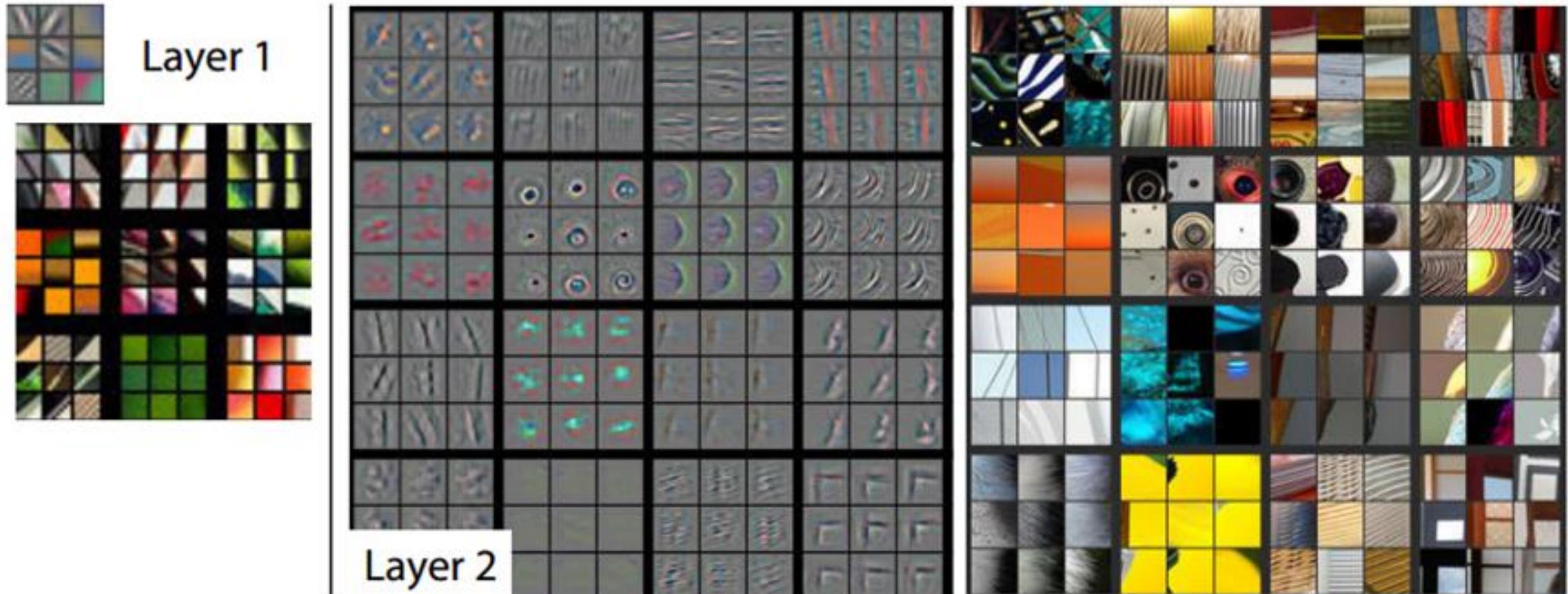
Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25.

ZFNet Architecture (2013)



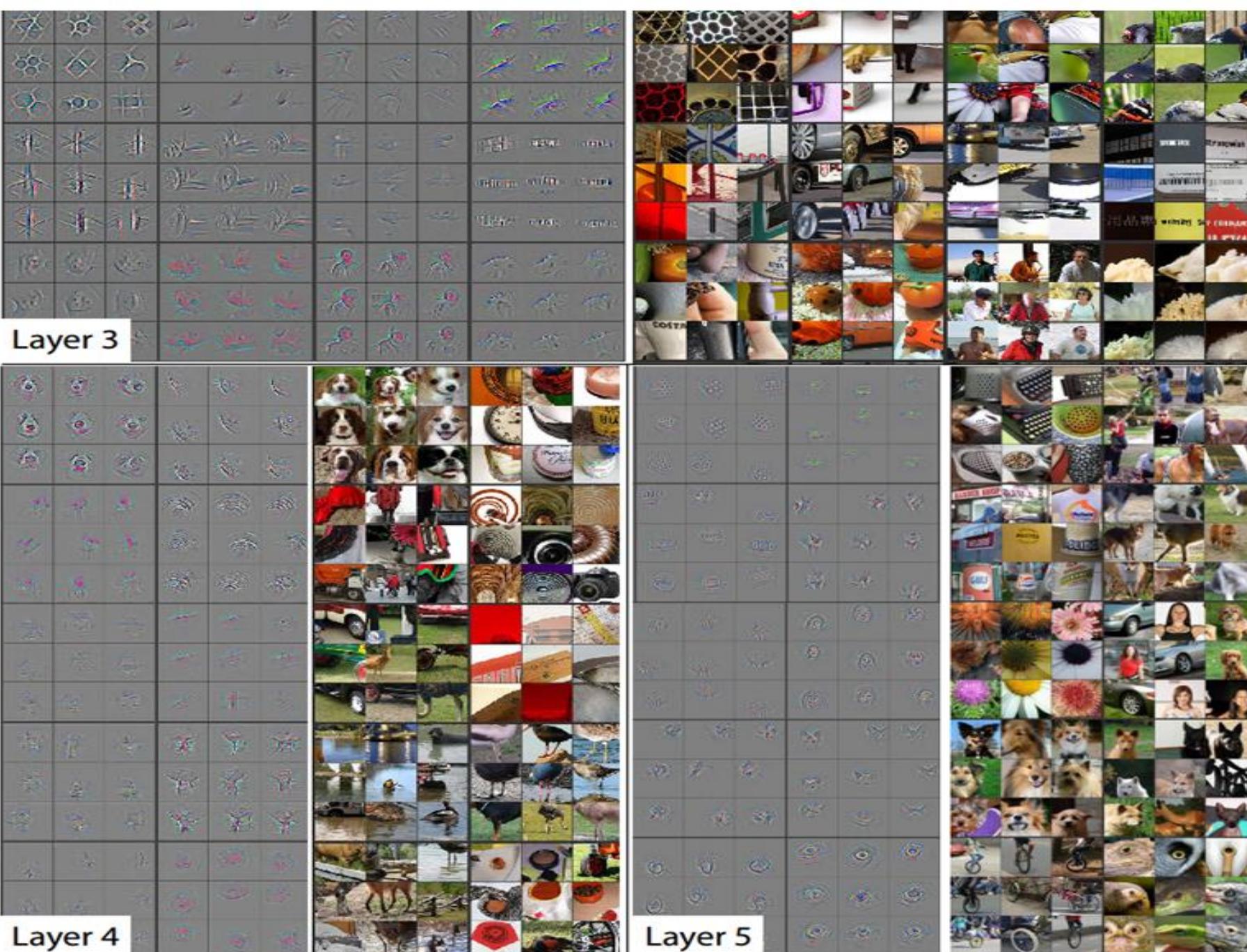
- Used filters of size 7x7 instead of 11x11 in AlexNet
- Used Deconvnet to visualize the intermediate results.

Zeiler, M. D., & Fergus, R. (2013). Visualizing and understanding convolutional networks.
In *European conference on computer vision* (pp. 818-833). Springer, Cham.



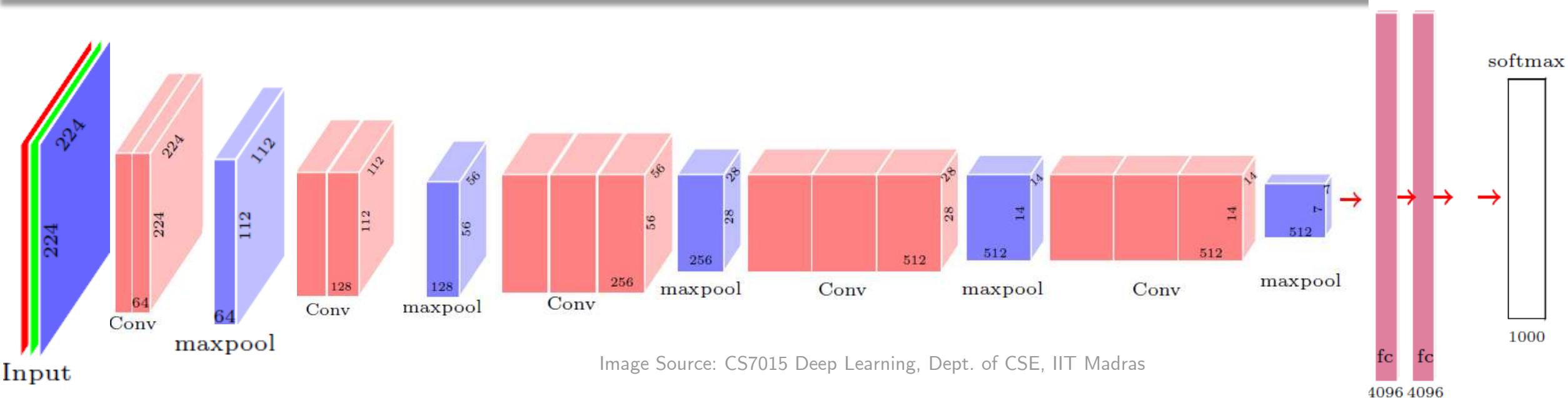
Visualizations of Layer 1 and 2. Each layer illustrates 2 pictures, one which shows the filters themselves and one that shows what part of the image are most strongly activated by the given filter. For example, in the space labeled Layer 2, we have representations of the 16 different filters (on the left)

[Visualizing and Understanding Deep Neural Networks by Matt Zeiler - YouTube](#)



Visualizations of Layers 3, 4, and 5

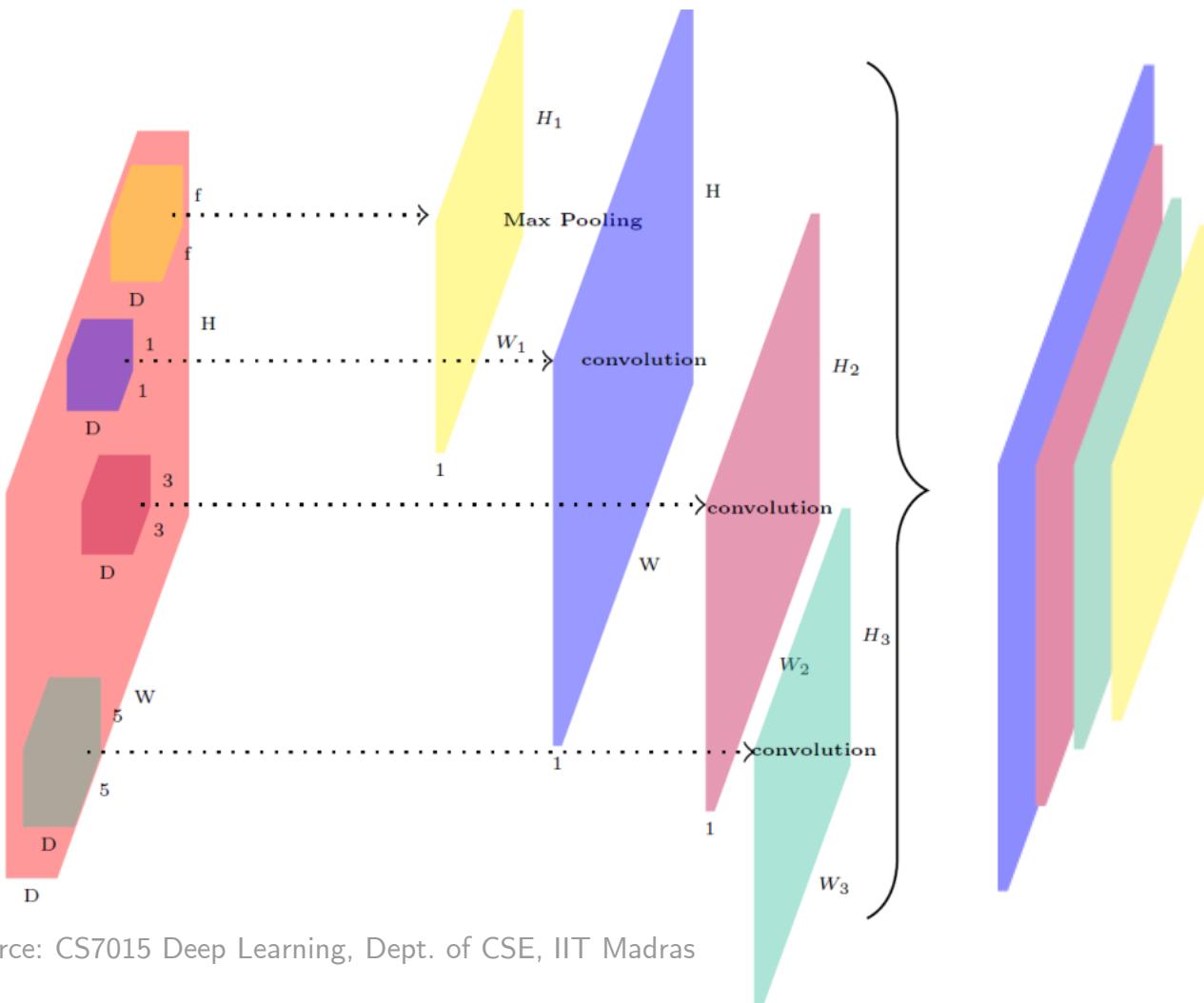
VGGNet Architecture (2014)



- Used filters of size 3x3 in all the convolution layers.
- 3 conv layers back-to-back have an effective receptive field of 7x7.
- Also called VGG-16 as it has 16 layers.
- This work reinforced the notion that convolutional neural networks have to have a deep network of layers in order for this hierarchical representation of visual data to work

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition , International Conference on Learning Representations (ICLR14)

GoogleNet Architecture (2014)

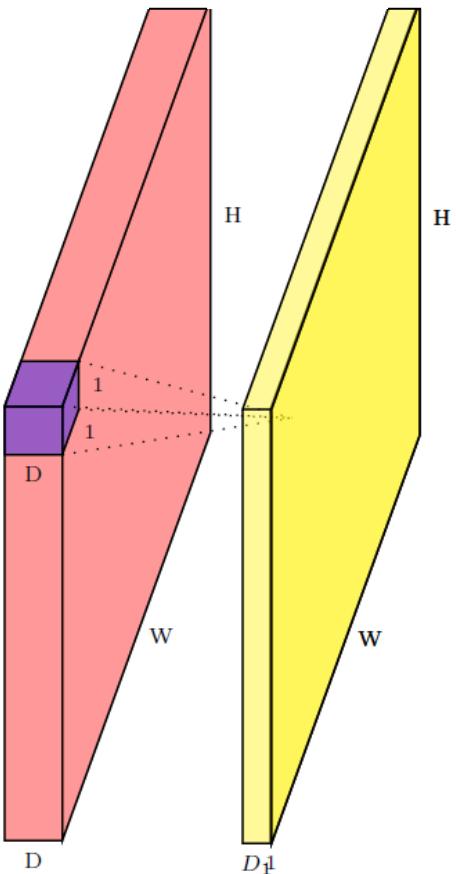


Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

- Most of the architectures discussed till now apply either of the following after each convolution operation:
 - Max Pooling
 - 3x3 convolution
 - 5x5 convolution
- Idea: Why cant we apply them all together at the same time and concatenate the feature maps.
- Problem: This will result in large number of computations.
- Specifically, each element of the output required **$O(F \times F \times D)$** computations

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions.
In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR'15)*

GoogleNet Architecture (2014)

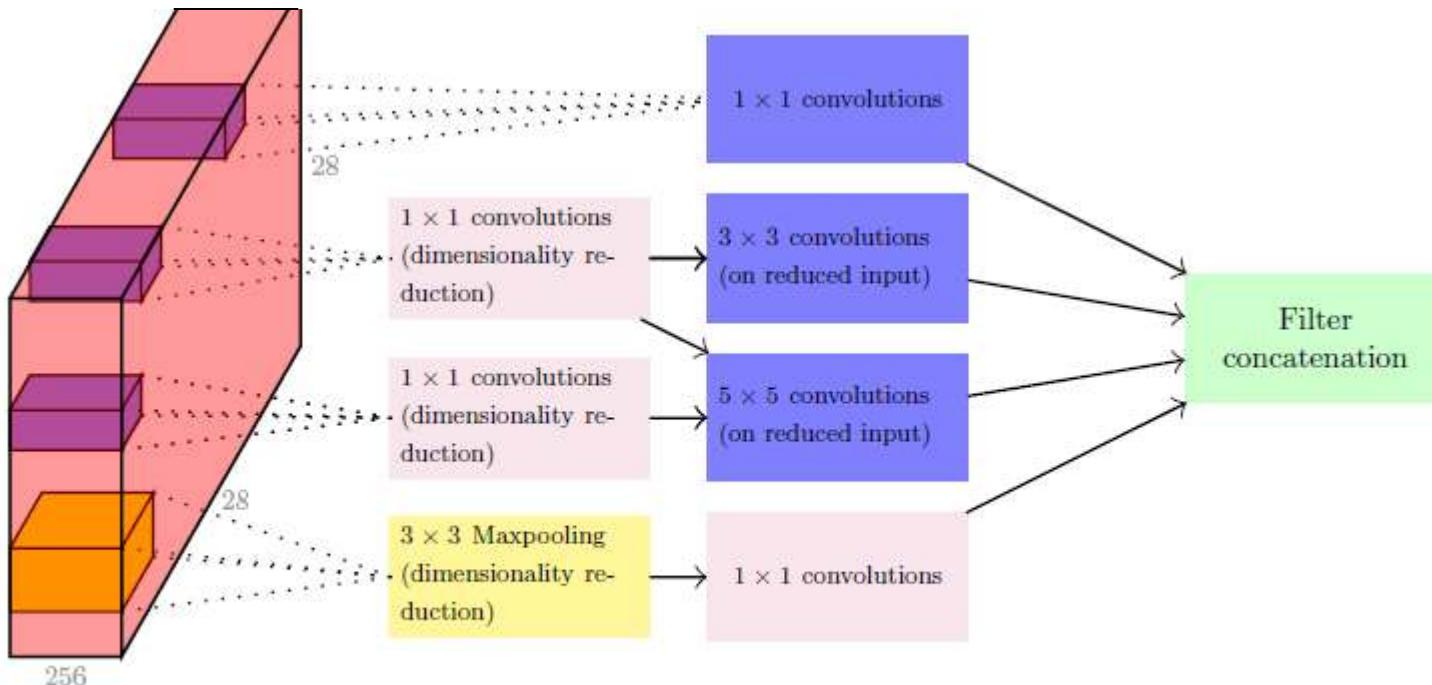


- Solution: Apply 1×1 convolutions
- 1×1 convolution aggregates along the depth.
- So, if we apply D_1 1×1 convolutions ($D_1 < D$), we will get an output of size $W \times H \times D_1$
- So, the total number of computations will reduce to **$O(F \times F \times D_1)$**
- We could then apply subsequent 3×3 , 5×5 filters on this reduced output

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions.
In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR'15)*

GoogleNet Architecture (2014)



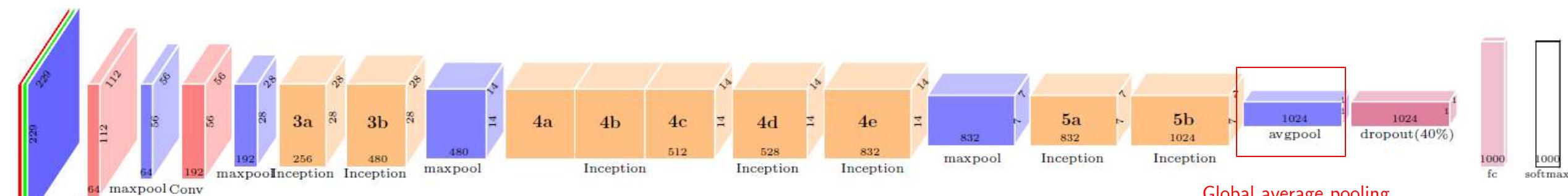
The Inception module

- Also, we might want to use different dimensionality reductions (applying 1×1 convolutions of different sizes) before the 3×3 and 5×5 filters.
- We can also add the maxpooling layer followed by 1×1 convolution.
- After this, we concatenate all these layers.
- This is called the **Inception module**.
- **GoogleNet** contains many such inception modules.

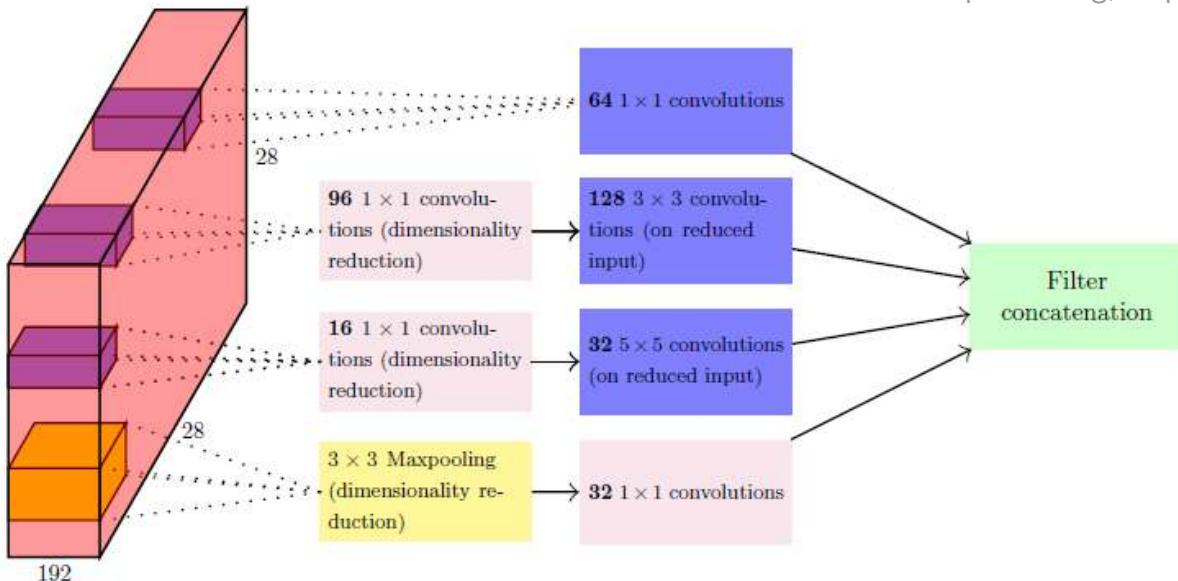
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR'15)*

GoogleNet Architecture (2014)



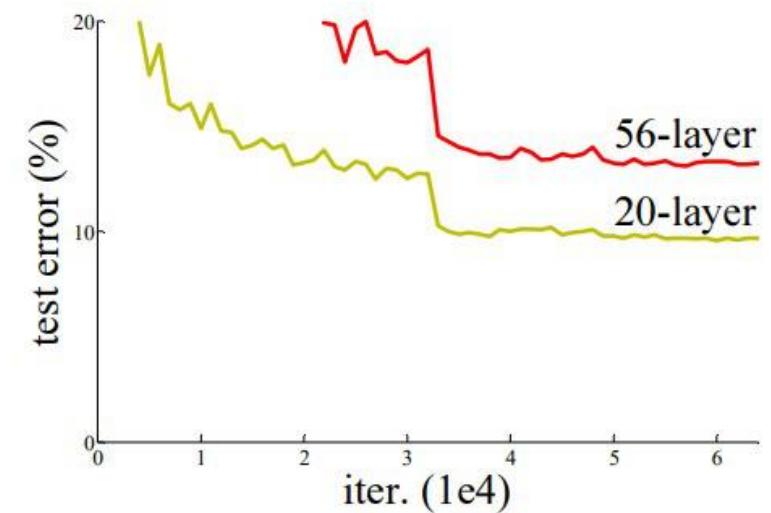
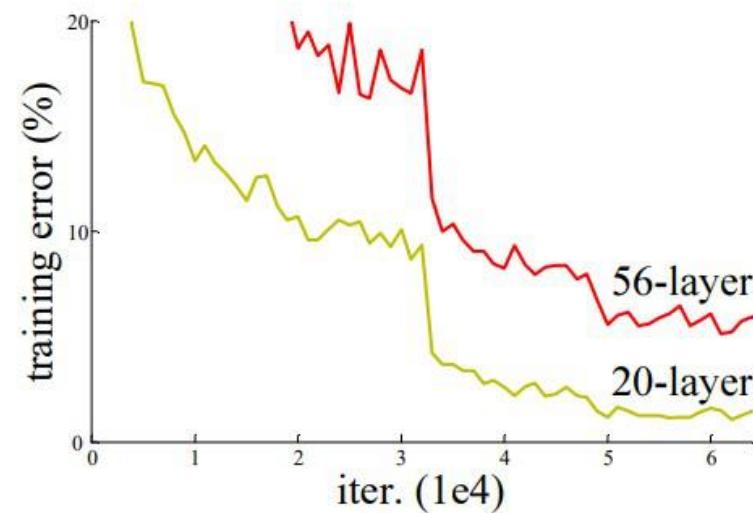
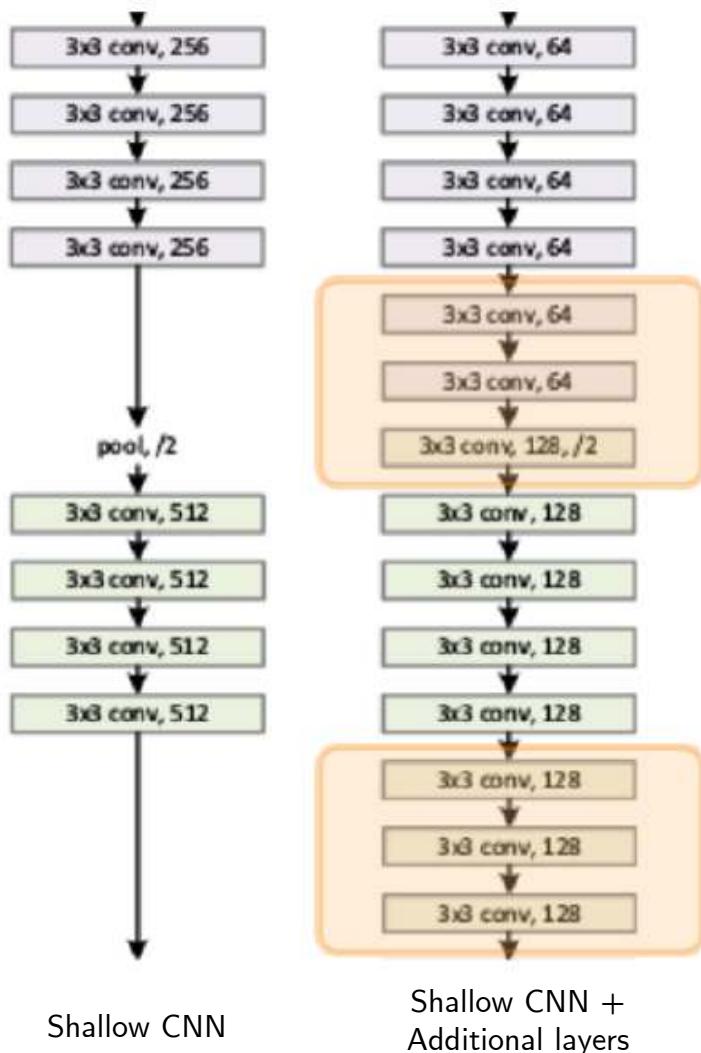
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras



- 12 times less parameters and 2 times more computations than AlexNet
- Used Global Average Pooling instead of Flattening.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., & Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR'15)*

ResNet Architecture (2015)

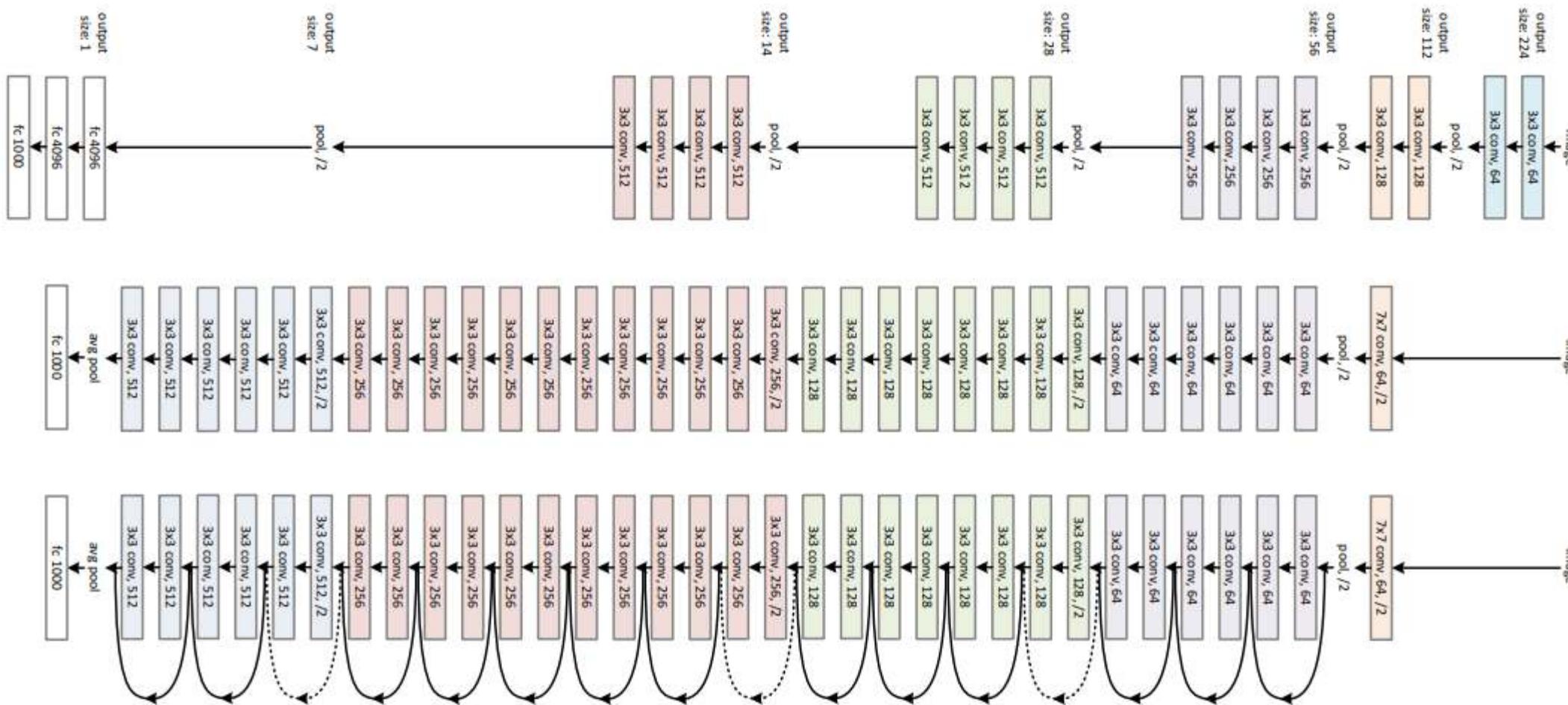


Effect of increasing layers of shallow CNN when experimented over the CIFAR dataset

Source: [Residual Networks \(ResNet\) - Deep Learning - GeeksforGeeks](#)

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

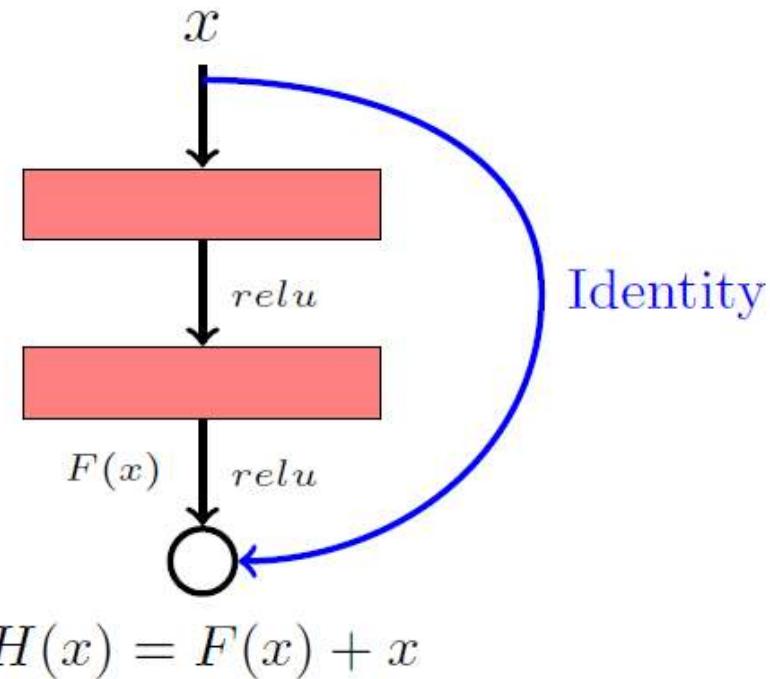
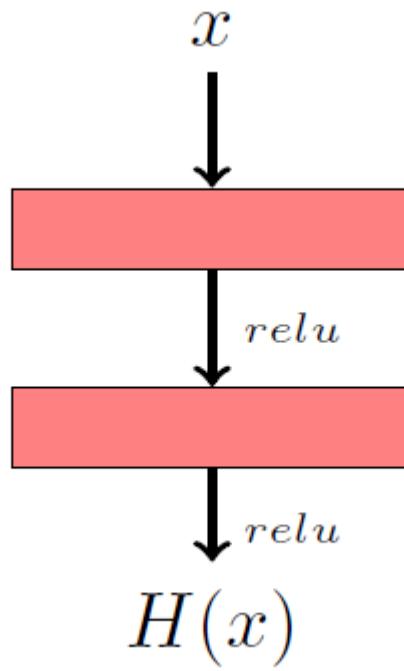
ResNet Architecture (2015)



Source: [Residual Networks \(ResNet\) - Deep Learning - GeeksforGeeks](#)

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

ResNet Architecture (2015)



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

Sequence Modeling using RNN

DSE 3151 DEEP LEARNING

Dr. Rohini Rao & Dr. Abhilash K Pai

Dept. of Data Science and Computer Applications

MIT Manipal

Examples of Sequence Data

- Speech Recognition



Mary had a little lamb

- Music Generation
- Sentiment Classification
- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition

Examples of Sequence Data

- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition

La



Examples of Sequence Data

- Speech Recognition

- Music Generation

- Sentiment Classification

“Its an average movie”



- DNA Sequence Analysis

- Machine Translation

- Video Activity Recognition

- Name Entity Recognition

Examples of Sequence Data

- Speech Recognition

- Music Generation

- Sentiment Classification

- DNA Sequence Analysis

AGCCCCTGTGAGGAACTAG → AGCCCCTGTGAGGAACTAG

- Machine Translation

- Video Activity Recognition

- Name Entity Recognition

Examples of Sequence Data

- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition

ARE YOU FEELING SLEEPY →

क्या आपको नींद आ रही है

Examples of Sequence Data

- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition



WAVING

Examples of Sequence Data

- Speech Recognition
- Music Generation
- Sentiment Classification
- DNA Sequence Analysis
- Machine Translation
- Video Activity Recognition
- Name Entity Recognition

“Alice wants to discuss about
Deep Learning with Bob”



“**Alice** wants to discuss about
Deep Learning with **Bob**”

Issues with using ANN/CNN on sequential data

Issues with using ANN/CNN on sequential data

- In feedforward and convolutional neural networks, the size of the **input was always fixed**.
 - In many applications with **sequence data**, the **input is not of a fixed size**.

Issues with using ANN/CNN on sequential data

- In feedforward and convolutional neural networks, the size of the **input was always fixed**.
 - In many applications with **sequence data**, the **input is not of a fixed size**.
- Further, each input to the ANN/CNN network was **independent of the previous or future inputs**.
 - With sequence data, **successive inputs** may **not** be **independent** of each other.

Modelling Sequence Learning Problems: Introduction

- The model needs to look at a sequence of inputs and produce an output (or outputs).

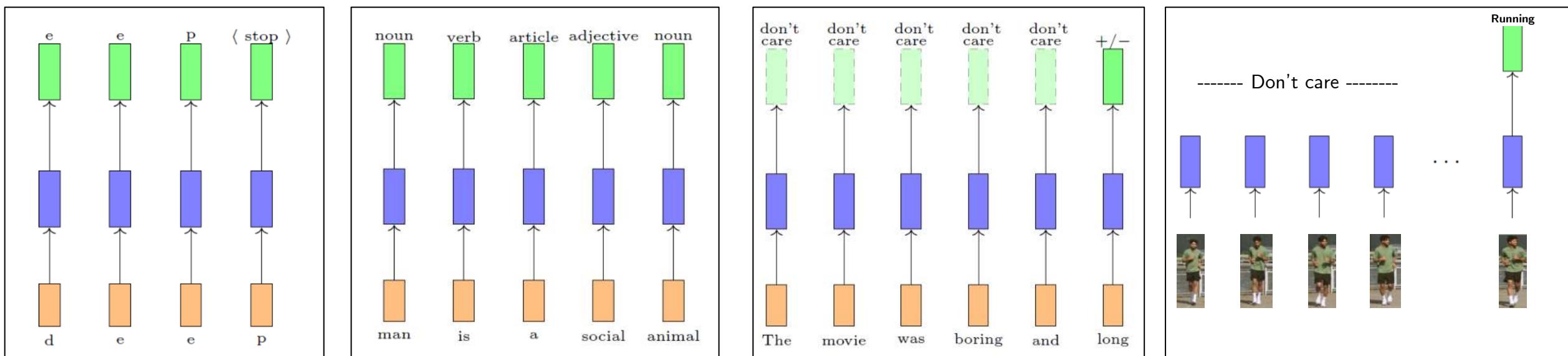
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Modelling Sequence Learning Problems: Introduction

- The model needs to look at a sequence of inputs and produce an output (or outputs).
- For this purpose, let's consider each input to be corresponding to one time step.

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Modelling Sequence Learning Problems: Introduction



Task: Auto-complete

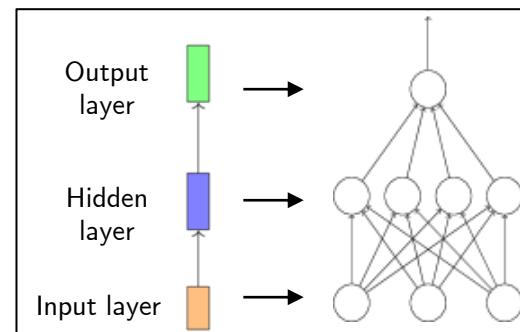
Task: P-o-S tagging

Task: Movie Review

Task: Action Recognition

- The model needs to look at a sequence of inputs and produce an output (or outputs).
- For this purpose, let's consider each input to be corresponding to one time step.
- Next, build a network for each time step/input, where each network performs the same task (eg: Auto complete: input=character, output=character)

Legend



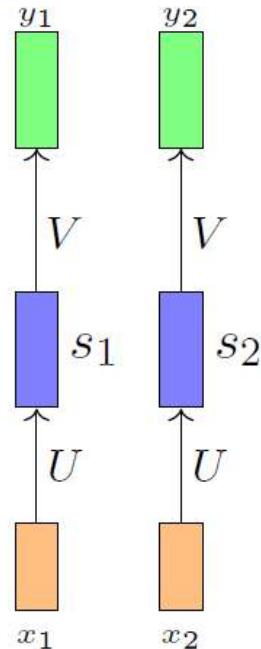
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

How to Model Sequence Learning Problems?

1. Model the dependence between inputs.
 - Eg: The next word after an ‘adjective’ is most probably a ‘noun’.
2. Account for variable number of inputs.
 - A sentence can have arbitrary no. of words.
 - A video can have arbitrary no. of frames.
3. Make sure that the function executed at each time step is the same.
 - Because at each time step we are doing the same task.

Modelling Sequence Learning Problems using Recurrent Neural Networks (RNN)

Introduction



Considering the network at each time step to be a fully connected network, the general equation for the network at each time step is:

$$s_i = \sigma(Ux_i + b)$$

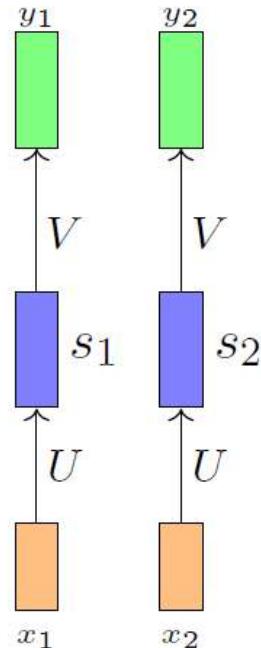
$$y_i = \mathcal{O}(Vs_i + c)$$

i = timestep

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Modelling Sequence Learning Problems using Recurrent Neural Networks (RNN)

Introduction



Considering the network at each time step to be a fully connected network, the general equation for the network at each time step is:

$$s_i = \sigma(Ux_i + b)$$

$$y_i = \mathcal{O}(Vs_i + c)$$

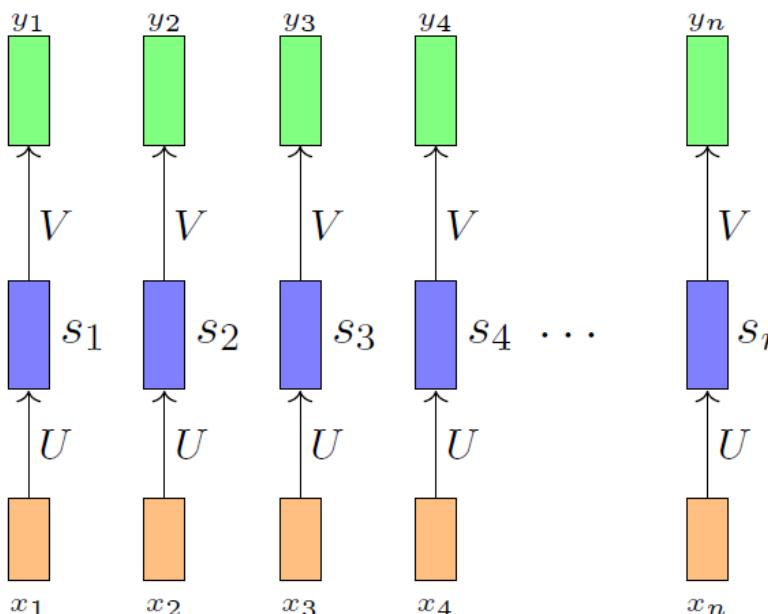
i = timestep

Since we want the same function to be executed at each timestep we should share the same network (i.e., **same parameters at each timestep**)

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Recurrent Neural Networks (RNN): Introduction

- If the input sequence is of length ‘n’, we would create ‘n’ networks for each input, as seen previously.

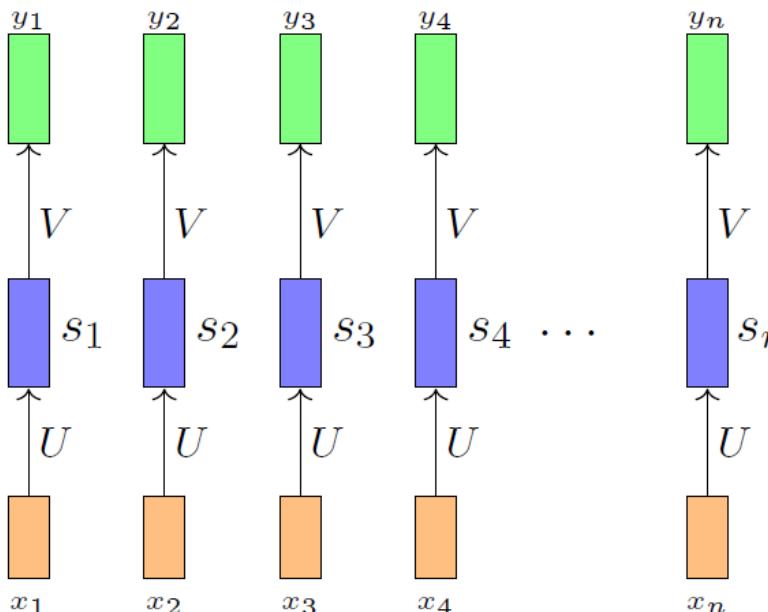


Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

By doing so, we have addressed the issue of variable input size!!

Recurrent Neural Networks (RNN): Introduction

- If the input sequence is of length ‘n’, we would create ‘n’ networks for each input, as seen previously.



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

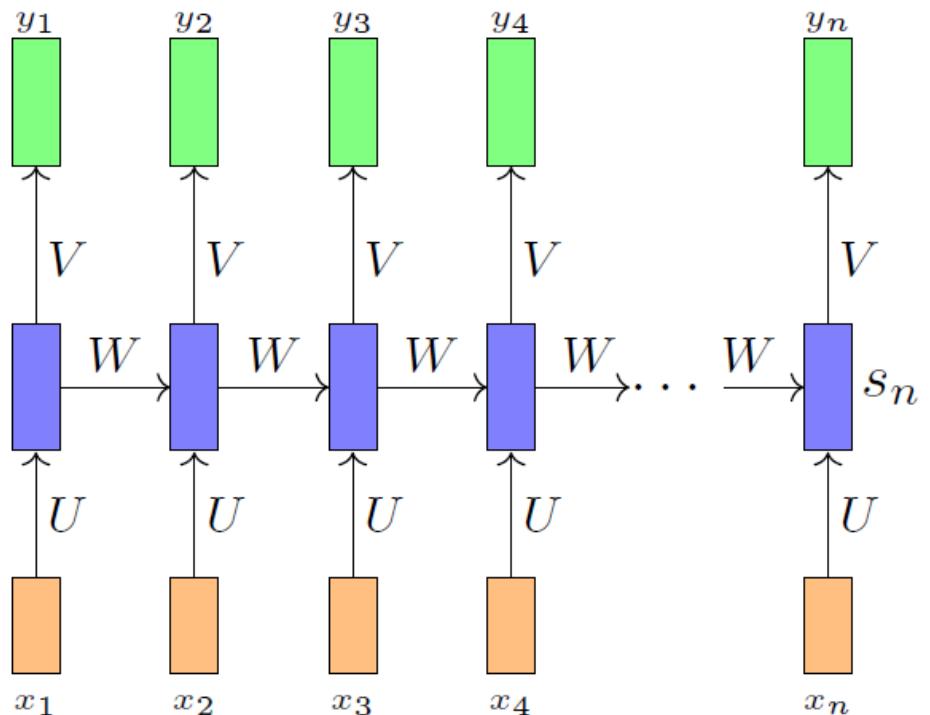
But, how to model the dependencies between the inputs ?

Recurrent Neural Networks (RNN)

Solution: Add recurrent connection in the network.

Recurrent Neural Networks (RNN)

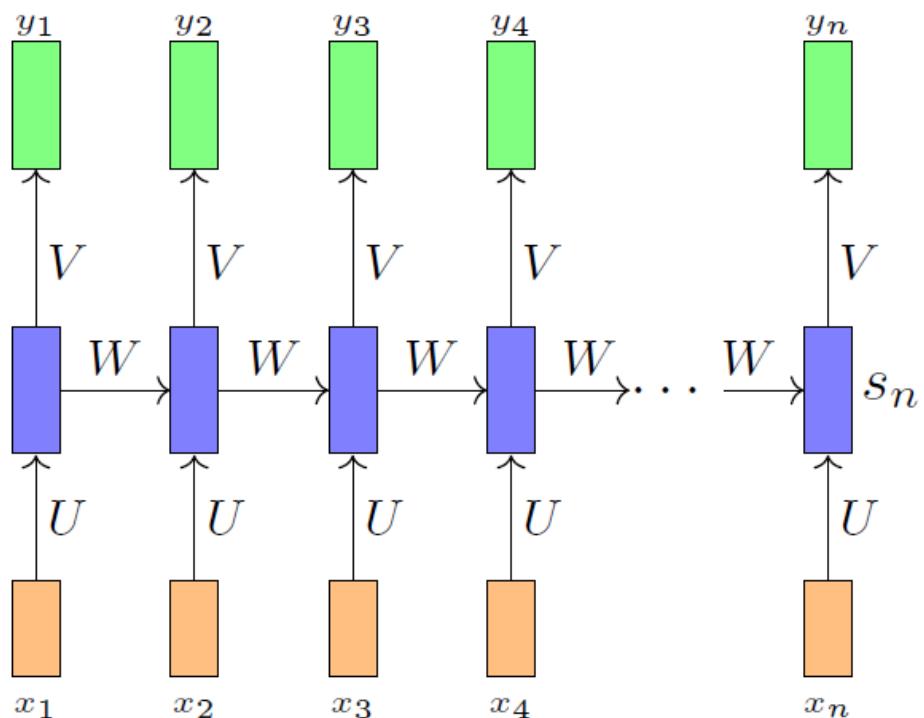
Solution: Add recurrent connection in the network.



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Recurrent Neural Networks (RNN)

Solution: Add recurrent connection in the network.



- So, the RNN equation:

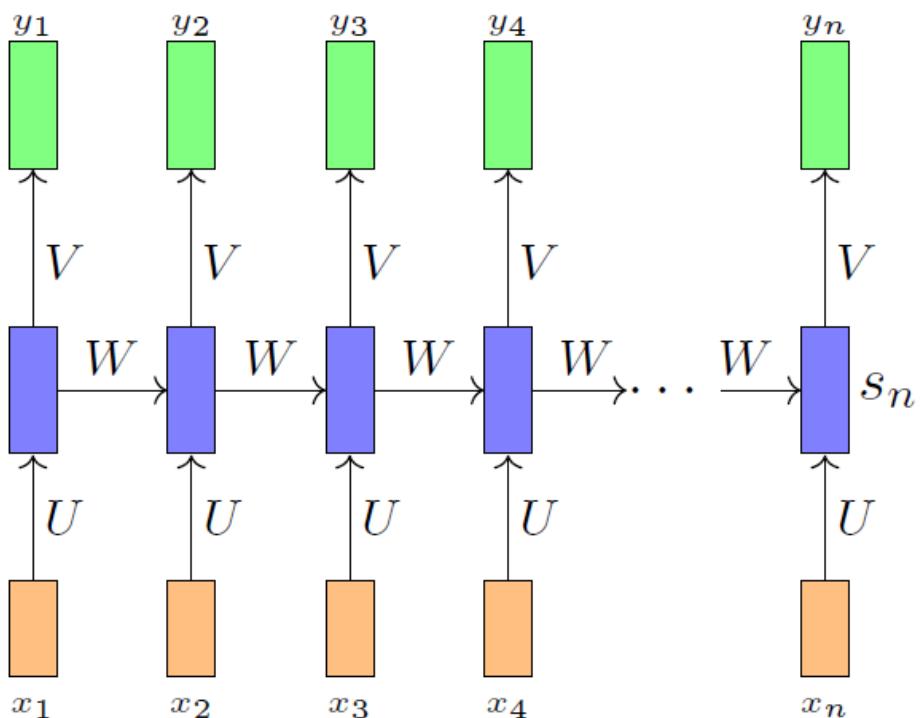
$$s_i = \sigma(Ux_i + Ws_{i-1} + b)$$

$$y_i = \mathcal{O}(Vs_i + c)$$

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Recurrent Neural Networks (RNN)

Solution: Add recurrent connection in the network.



- So, the RNN equation:

$$s_i = \sigma(Ux_i + Ws_{i-1} + b)$$

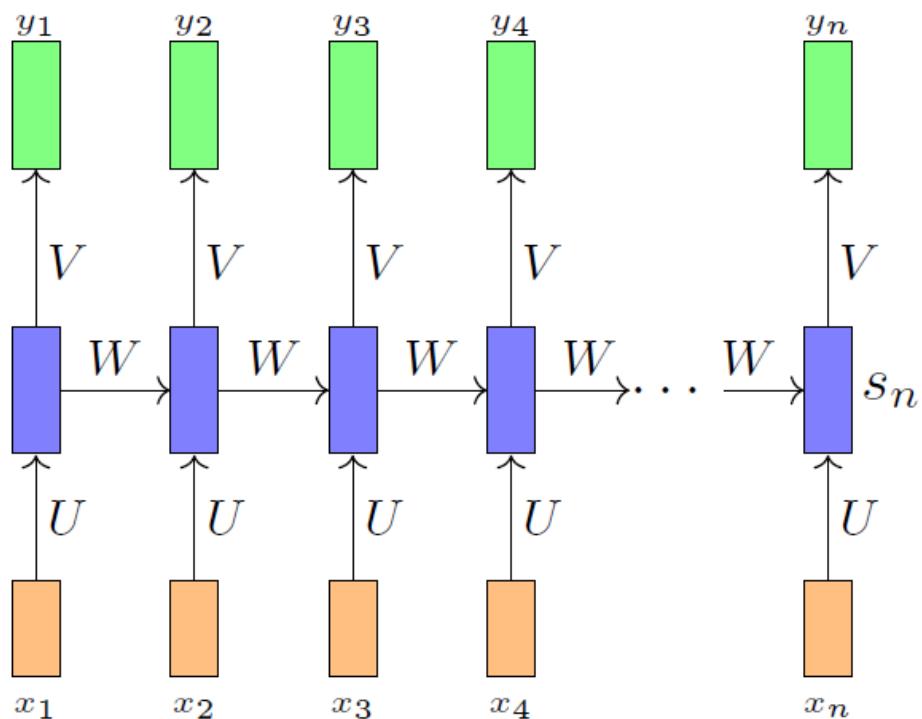
$$y_i = \mathcal{O}(Vs_i + c)$$

U, W, V, b, c are parameters of the network

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Recurrent Neural Networks (RNN)

Solution: Add recurrent connection in the network.



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

- So, the RNN equation:

$$s_i = \sigma(Ux_i + Ws_{i-1} + b)$$

$$y_i = \mathcal{O}(Vs_i + c)$$

U, W, V, b, c are parameters of the network

The dimensions of each term is as follows:

X_i -- [1 x no. of i/p neurons]

s_i -- [1 x no. of neurons in the hidden state]

W -- [no. of neurons in the hidden state x no. of neurons in the hidden state]

U -- [no. of i/p neurons x no. of neurons in the hidden state]

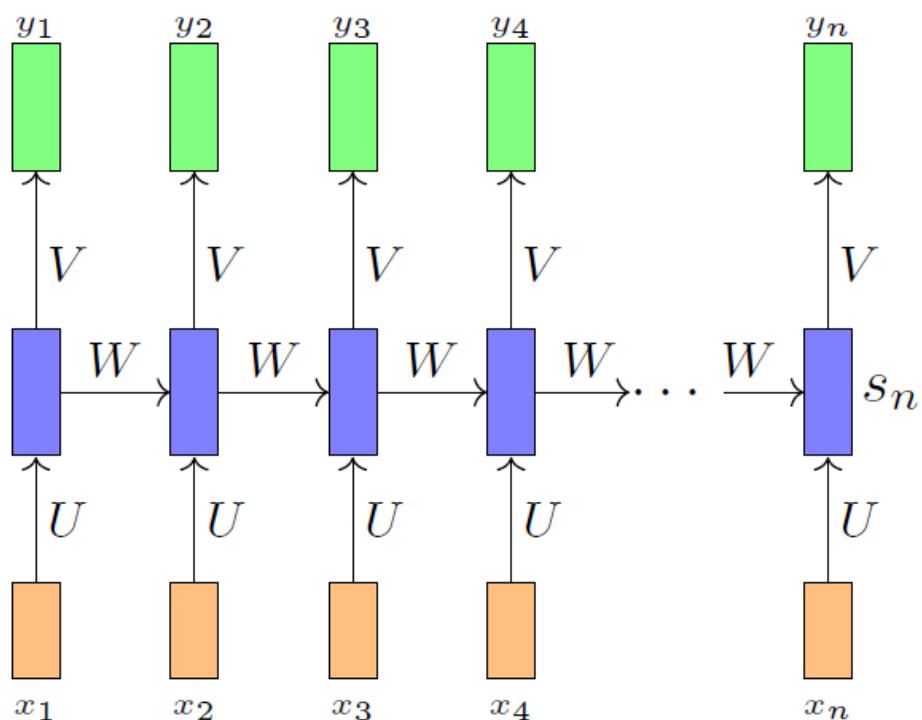
V -- [no. of neurons in the hidden state x no. of neurons in the o/p state]

b -- [1 x no. of neurons in the hidden state]

c -- [1 x no. of neurons in the o/p state]

Recurrent Neural Networks (RNN)

Solution: Add recurrent connection in the network.



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

- So, the RNN equation:

$$s_i = \sigma(Ux_i + Ws_{i-1} + b)$$

$$y_i = \mathcal{O}(Vs_i + c)$$

U, W, V, b, c are parameters of the network

The dimensions of each term is as follows:

X_i -- [1 x no. of i/p neurons]

s_i -- [1 x no. of neurons in the hidden state]

W -- [no. of neurons in the hidden state x no. of neurons in the hidden state]

U -- [no. of i/p neurons x no. of neurons in the hidden state]

V -- [no. of neurons in the hidden state x no. of neurons in the o/p state]

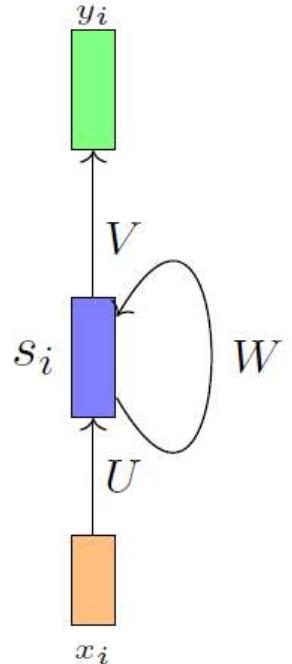
b -- [1 x no. of neurons in the hidden state]

c -- [1 x no. of neurons in the o/p state]

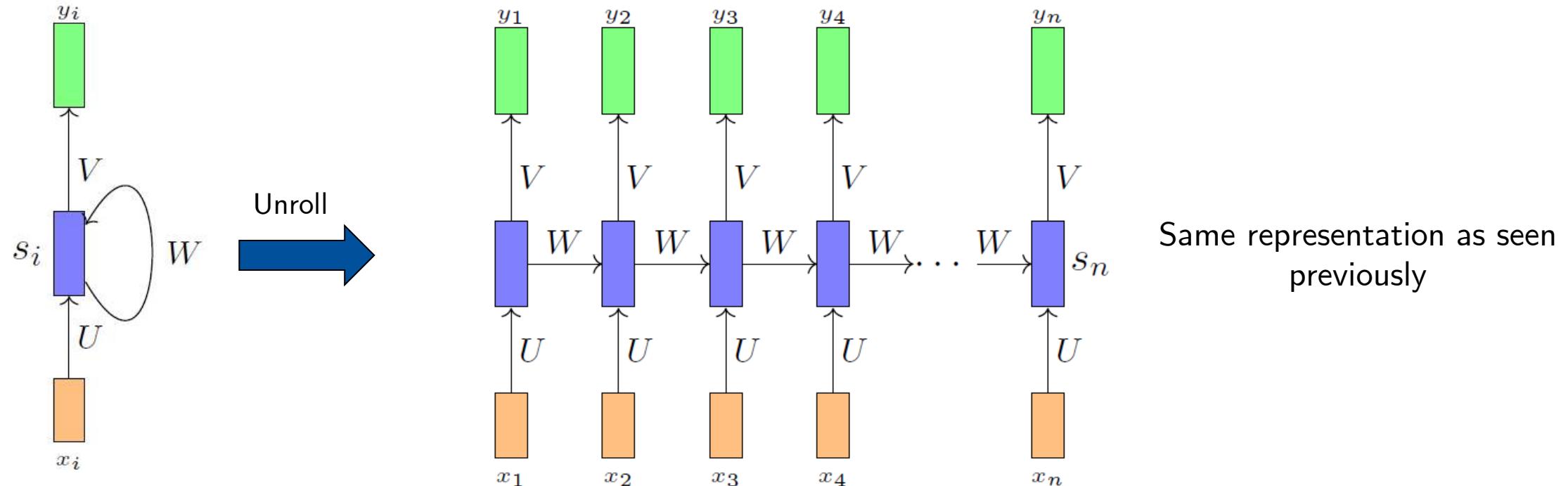
- At time step $i=0$ there are no previous inputs, so they are typically assumed to be all zeros.
- Since, the output of s_i at time step i is a function of all the inputs from previous time steps, we could say it has a form of **memory**.
- A part of a neural network that preserves some state across time steps is called a **memory cell** (or simply a **cell**)

Recurrent Neural Networks (RNN)

Compact representation of a RNN:



Recurrent Neural Networks (RNN)

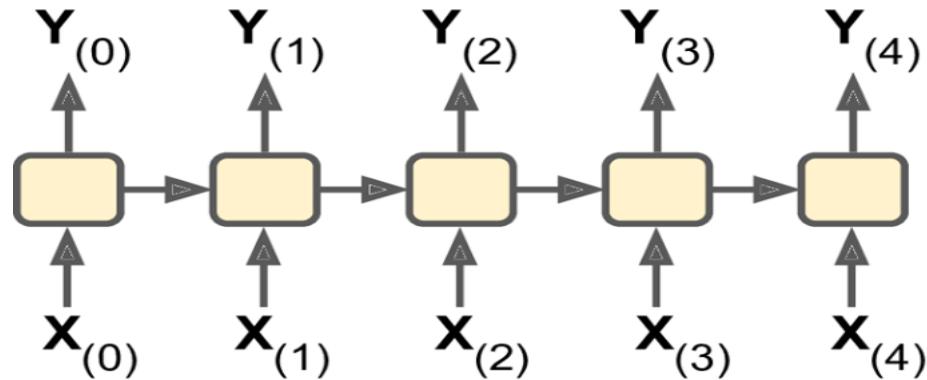


Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

- Unrolling the network through time = representing network against time axis.
- At each time step t (also called a **frame**) RNN receives inputs x_i as well as output from previous step y_{i-1}

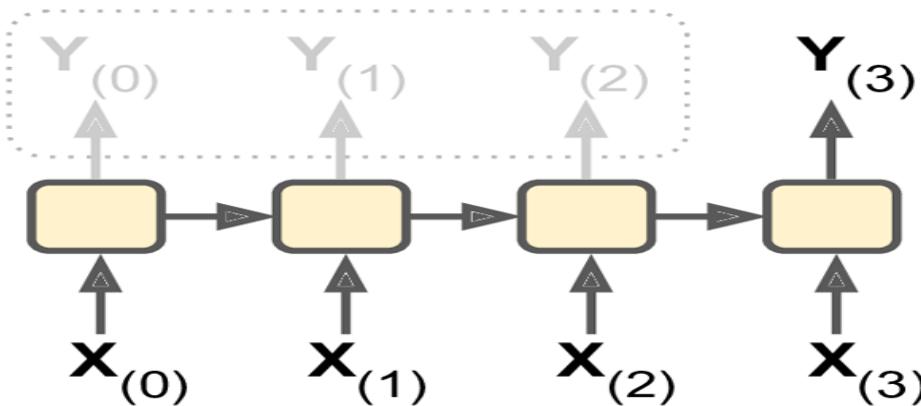
Input and Output Sequences

Seq-to-Seq

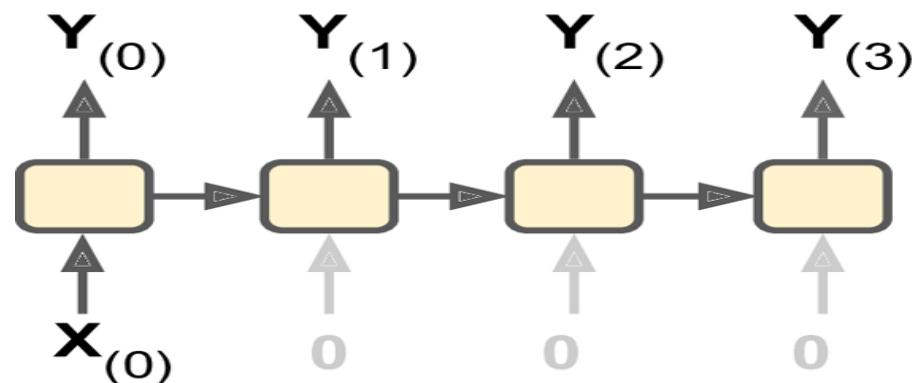


Seq-to-Vector

Ignored outputs

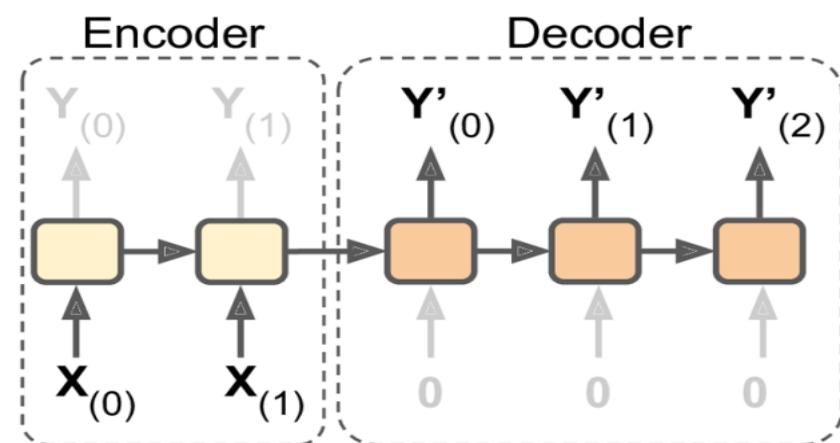


Vector-to-Seq

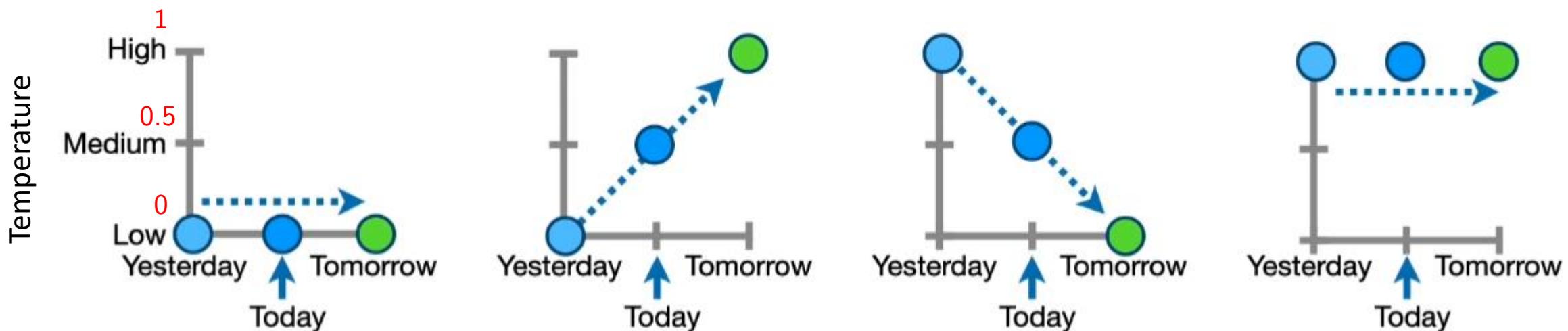


Encoder

Decoder

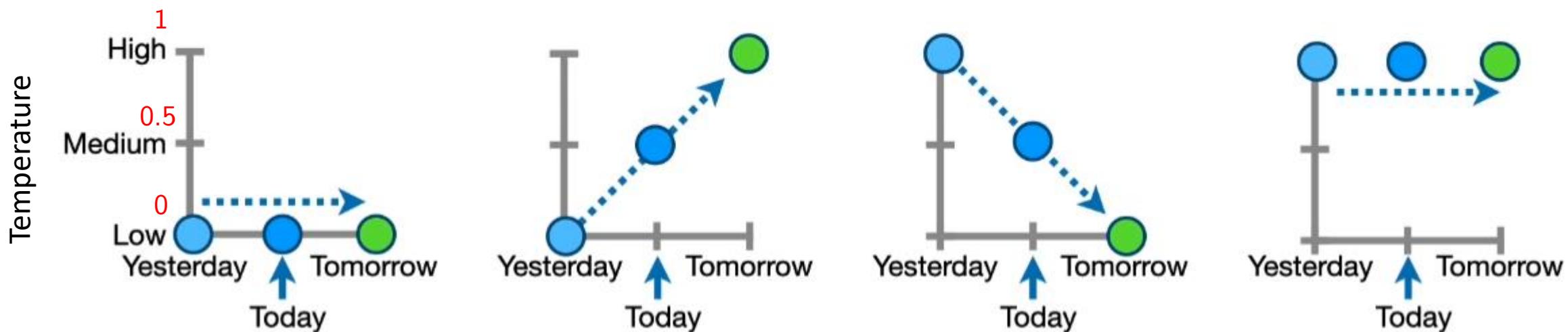


Recurrent Neural Networks (RNN) : Example



Source: <https://www.youtube.com/c/joshstarmer>

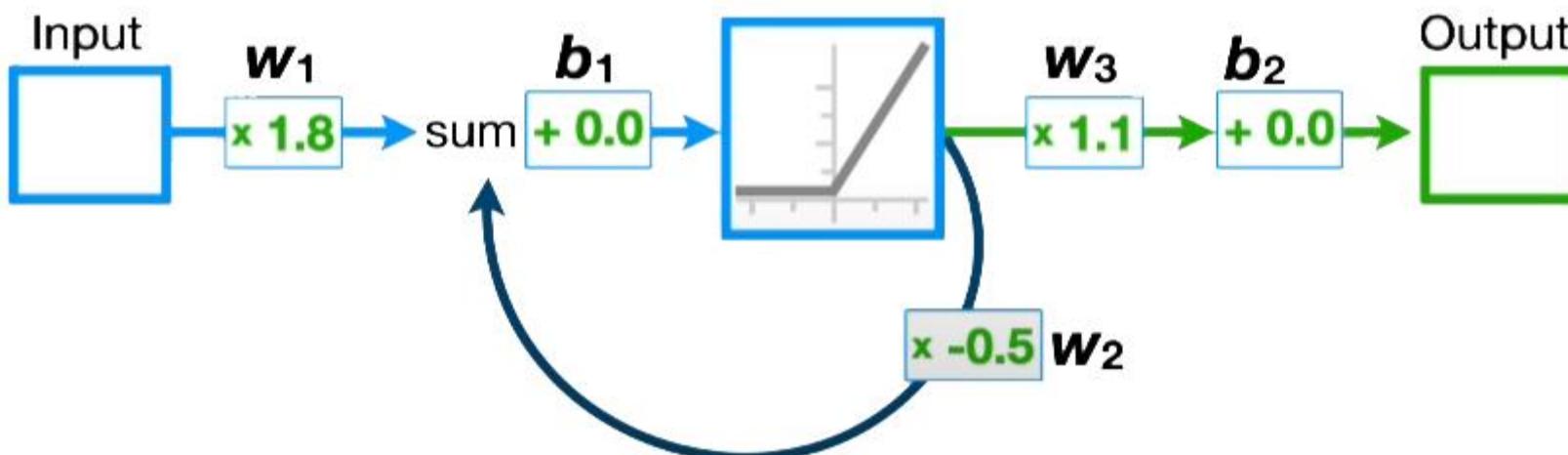
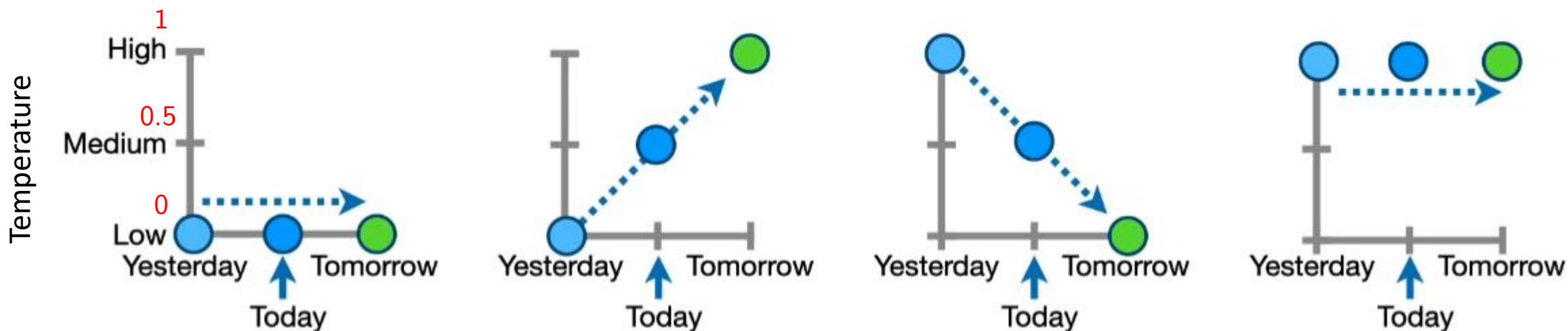
Recurrent Neural Networks (RNN) : Example



Problem : Given the temperatures of yesterday and today predict tomorrow's temperature.

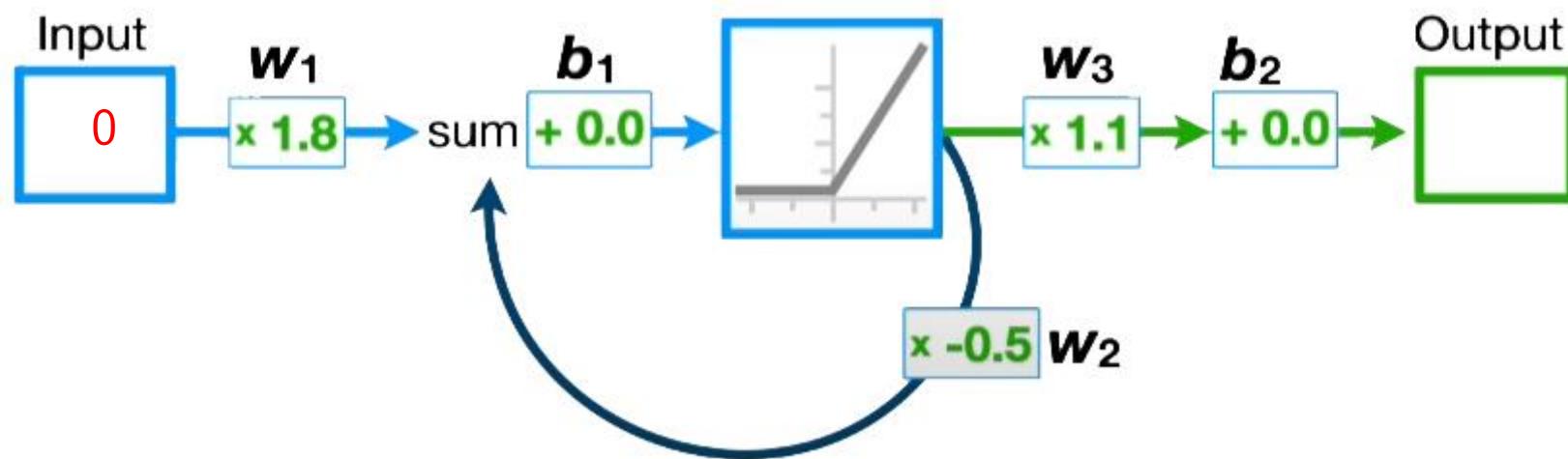
Source: <https://www.youtube.com/c/joshstarmer>

Recurrent Neural Networks (RNN) : Example



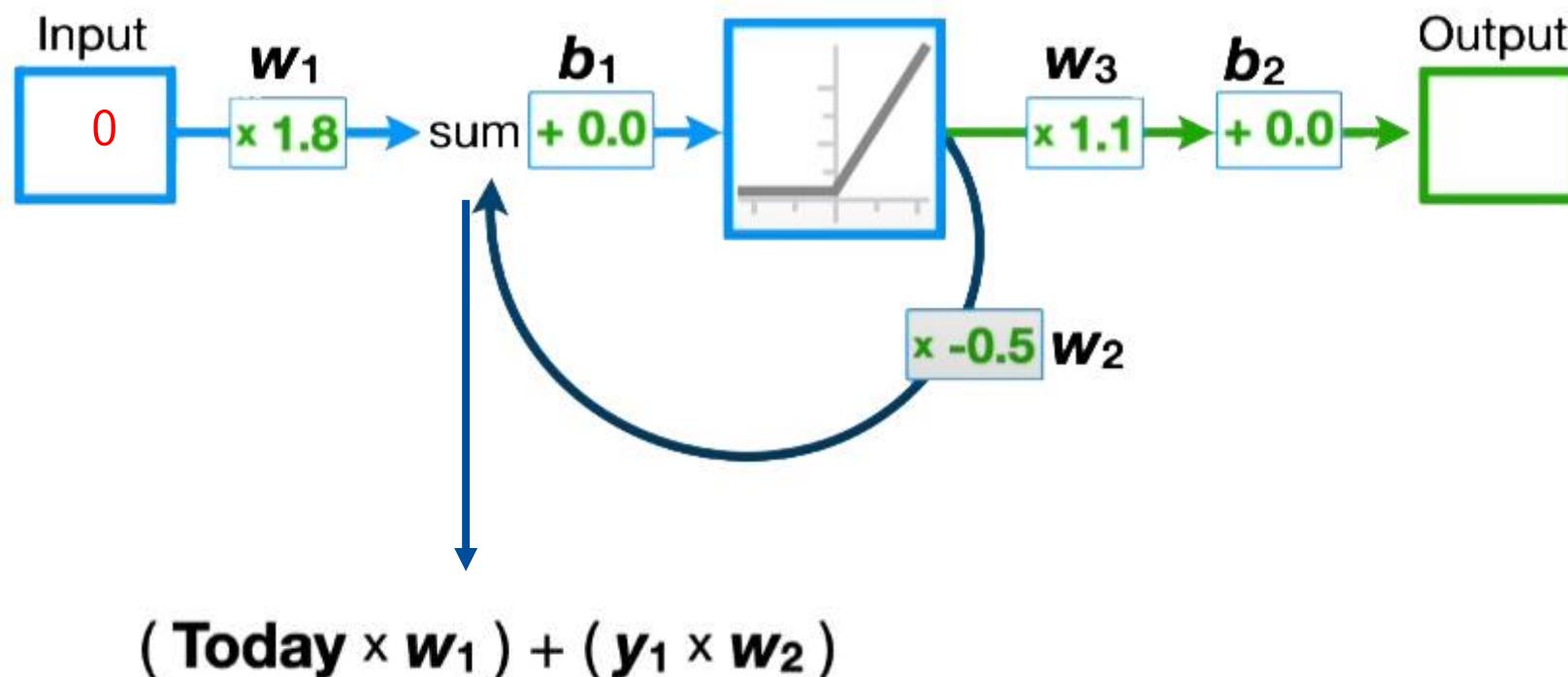
Source: <https://www.youtube.com/c/joshstammer>

Recurrent Neural Networks (RNN) : Example



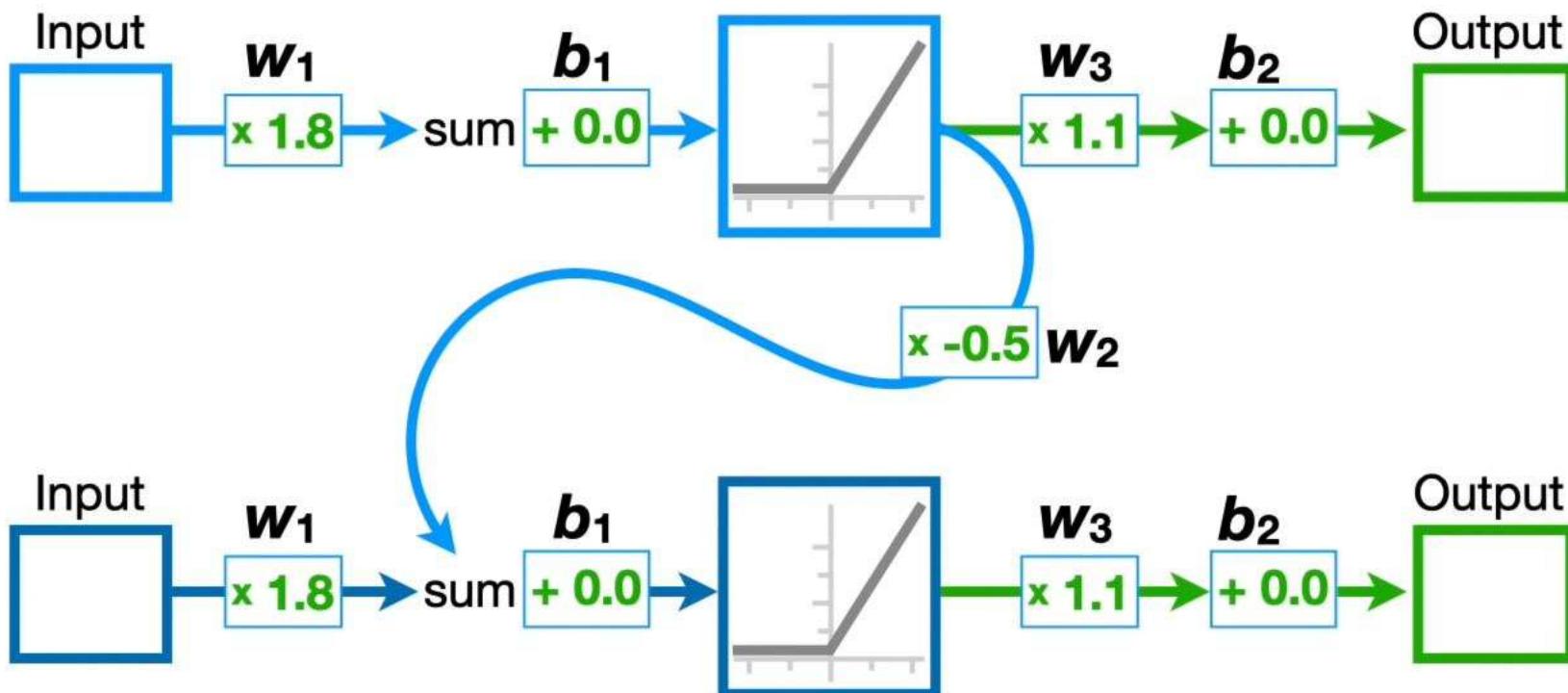
Source: <https://www.youtube.com/c/joshstarmer>

Recurrent Neural Networks (RNN) : Example



Source: <https://www.youtube.com/c/joshstarmer>

Recurrent Neural Networks (RNN) : Example



Unrolling the feedback loop by making a copy of NN for each input value

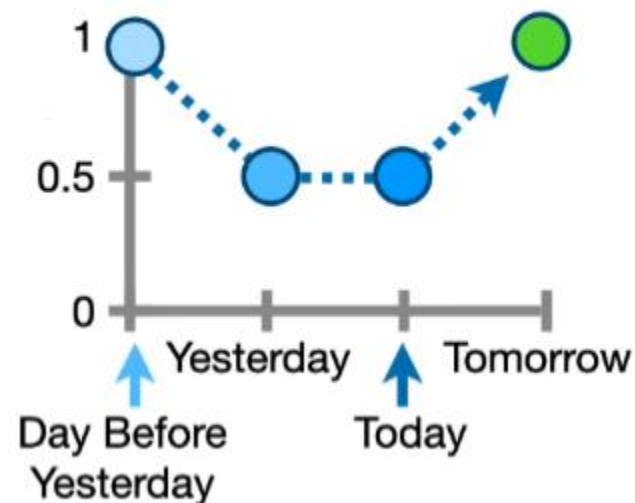
Source: <https://www.youtube.com/c/joshstarmer>

Recurrent Neural Networks (RNN) : Example



Source: <https://www.youtube.com/c/joshstarmer>

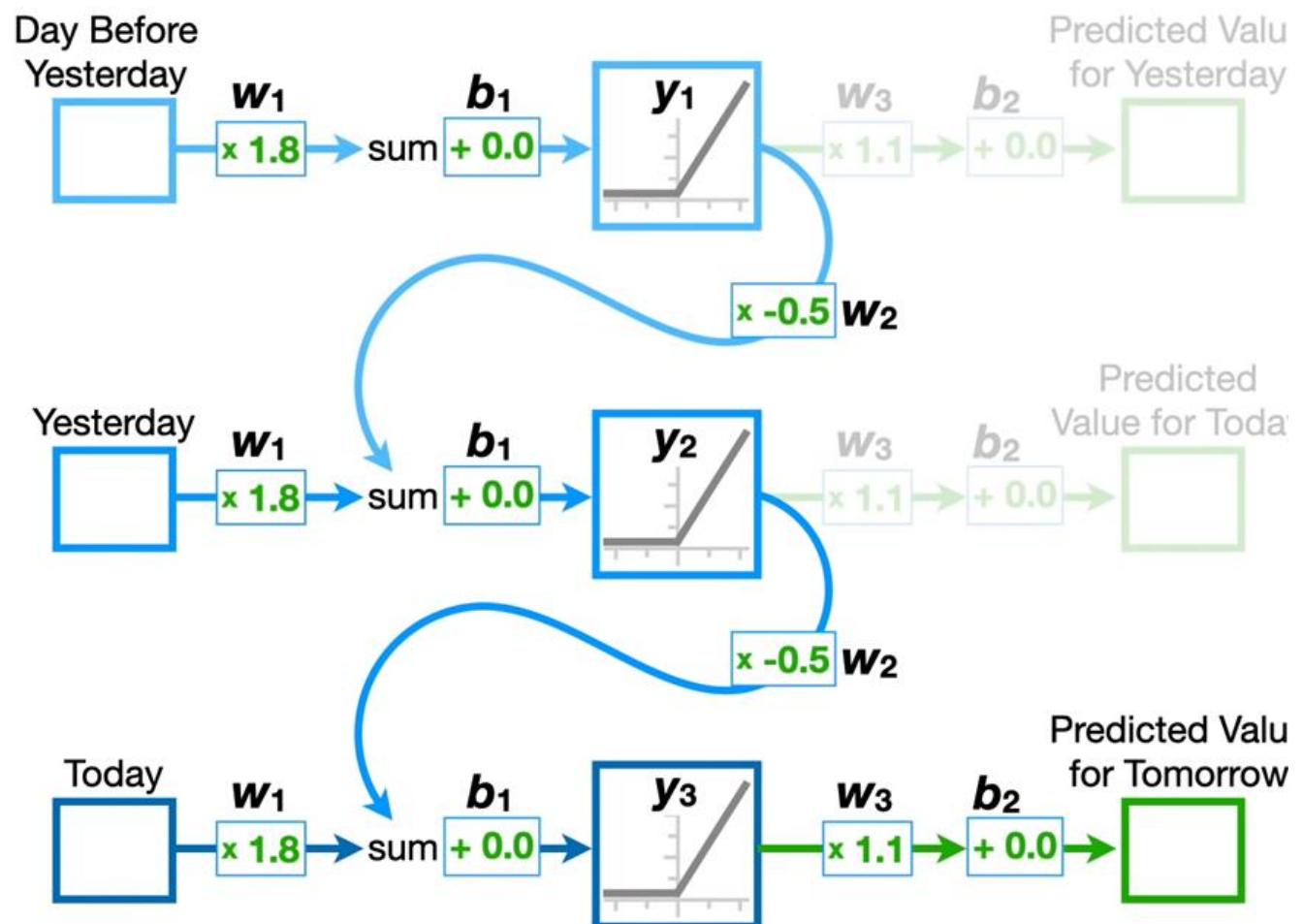
Recurrent Neural Networks (RNN) : Example



Problem : Given the temperature of 3 days (today, yesterday and day before yesterday), Predict tomorrow's temperature?

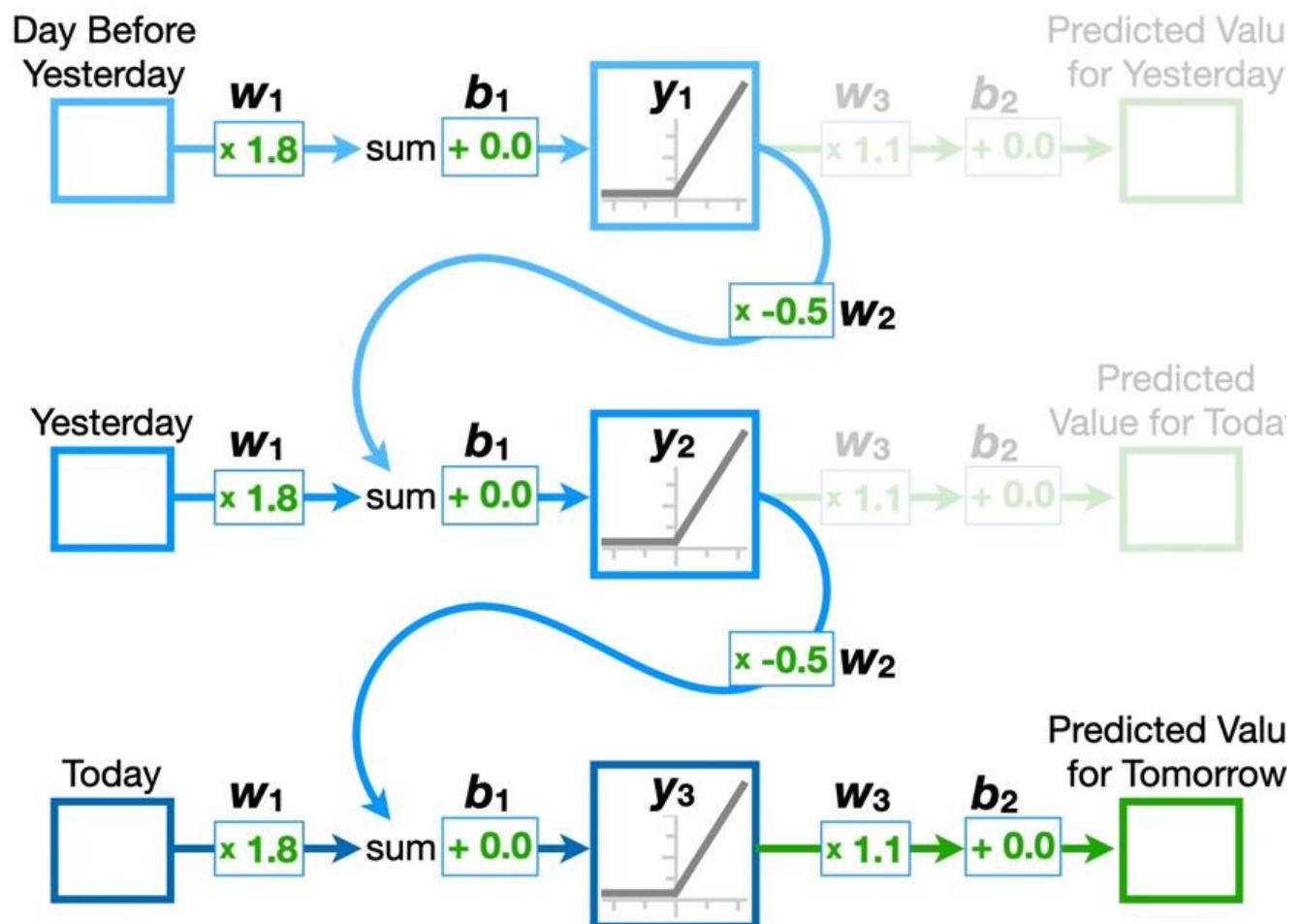
Source: <https://www.youtube.com/c/joshstarmer>

Recurrent Neural Networks (RNN) : Example



Source: <https://www.youtube.com/c/joshstarmer>

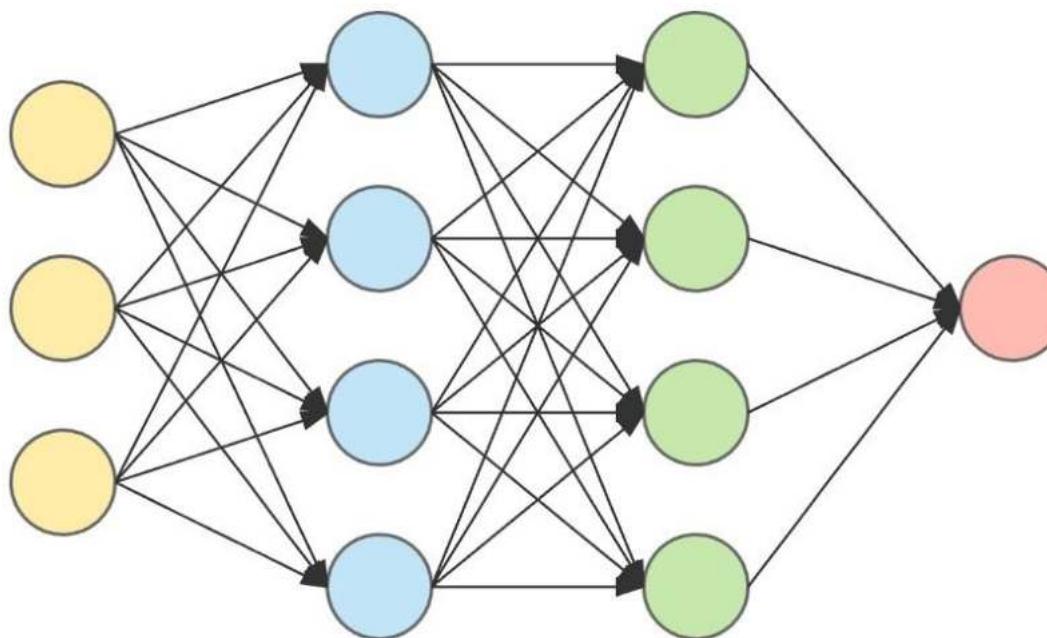
Recurrent Neural Networks (RNN) : Example



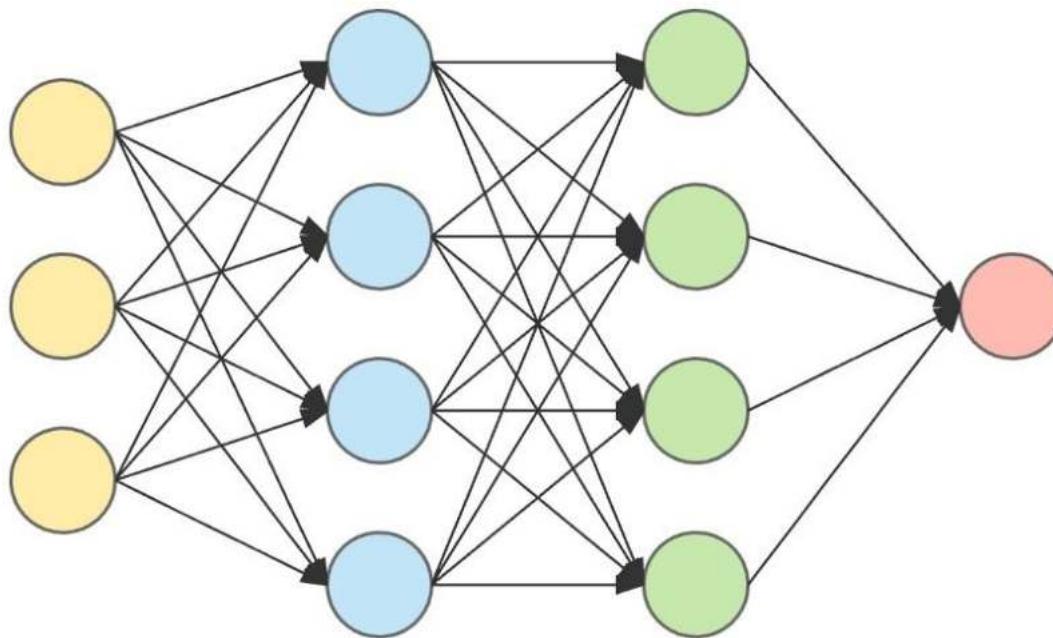
So, the no. of networks = no. of inputs

Source: <https://www.youtube.com/c/joshstarmer>

Backpropagation in ANN : Recap

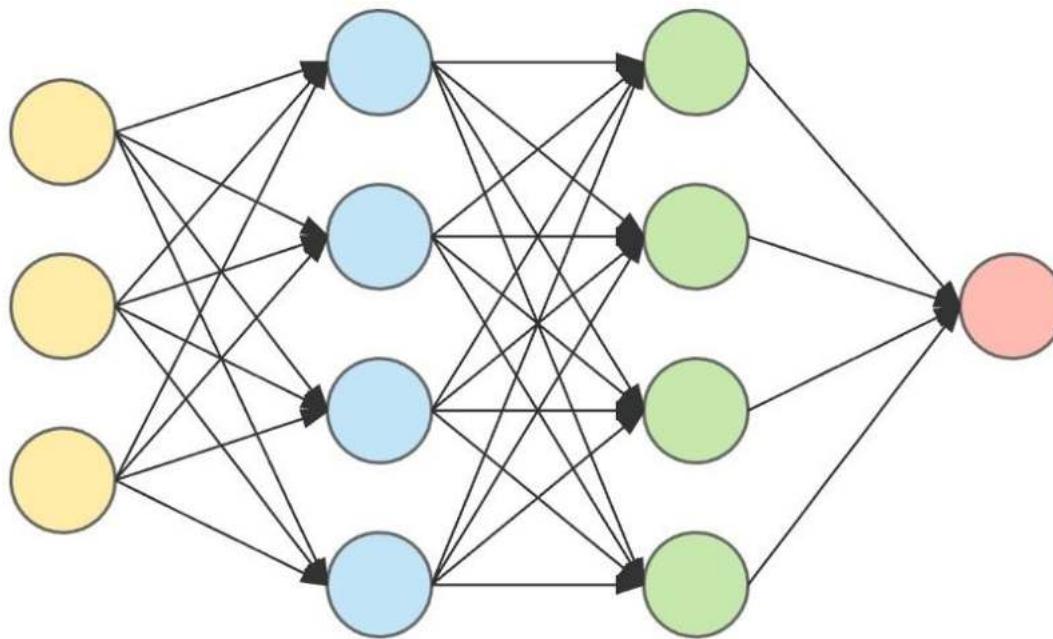


Backpropagation in ANN : Recap



For simplicity, lets represent the above network as follows:

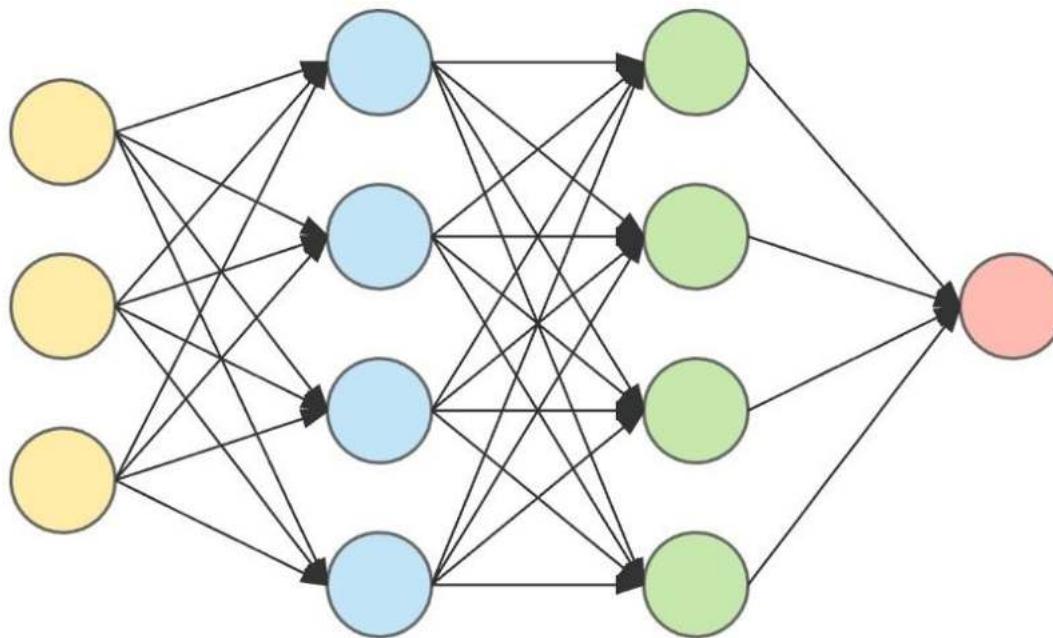
Backpropagation in ANN : Recap



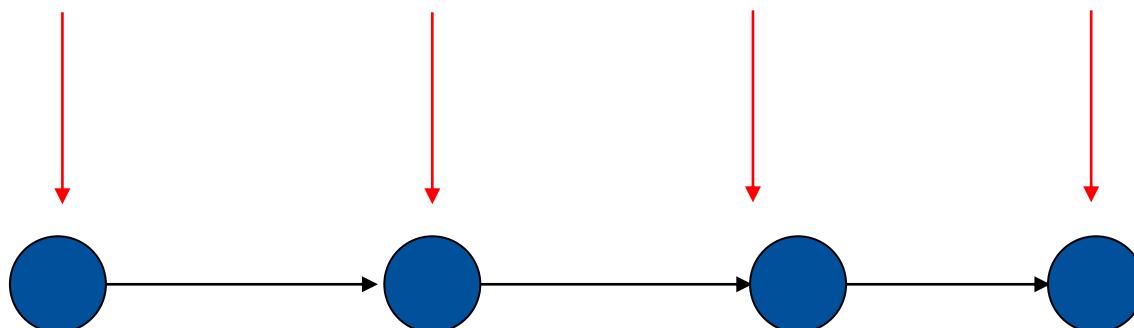
For simplicity, lets represent the above network as follows:



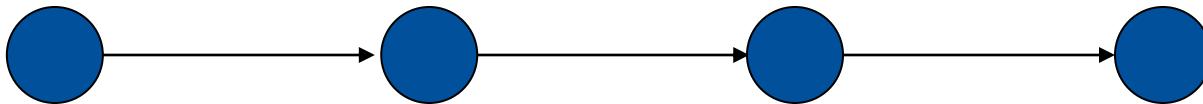
Backpropagation in ANN : Recap



For simplicity, let's represent the above network as follows:

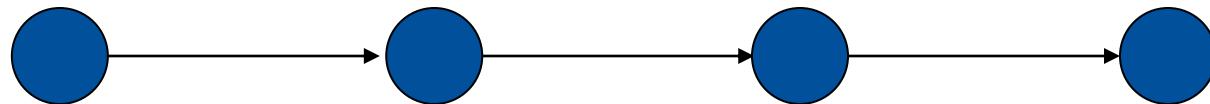


Backpropagation in ANN : Recap



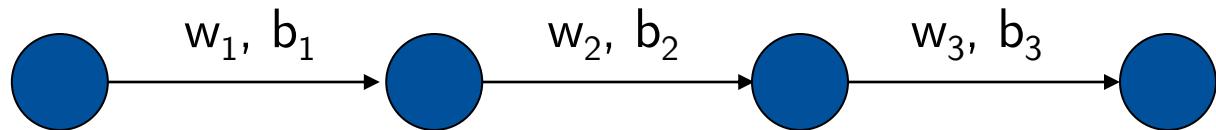
Backpropagation in ANN : Recap

The Loss function: $L(w_1, b_1, w_2, b_2, w_3, b_3)$



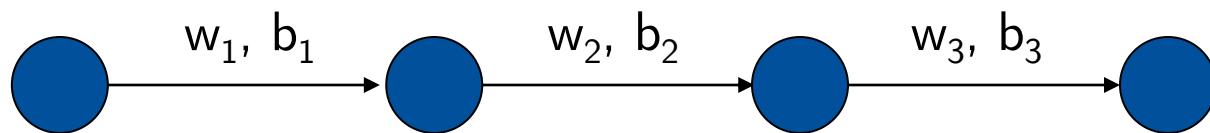
Backpropagation in ANN : Recap

The Loss function: $L(w_1, b_1, w_2, b_2, w_3, b_3)$



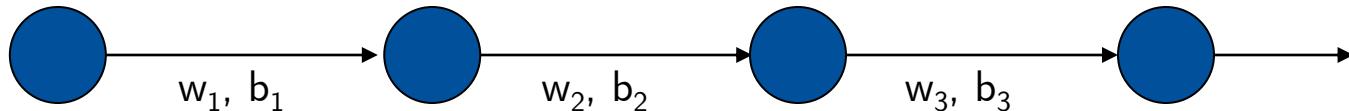
Backpropagation in ANN : Recap

The Loss function: $L(w_1, b_1, w_2, b_2, w_3, b_3)$

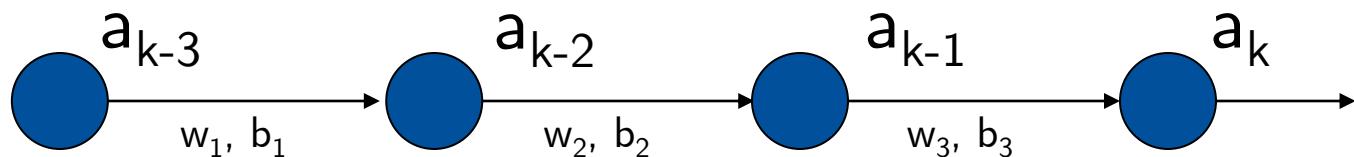


By how much should the parameters be changed to make an efficient decrease in the loss L ?

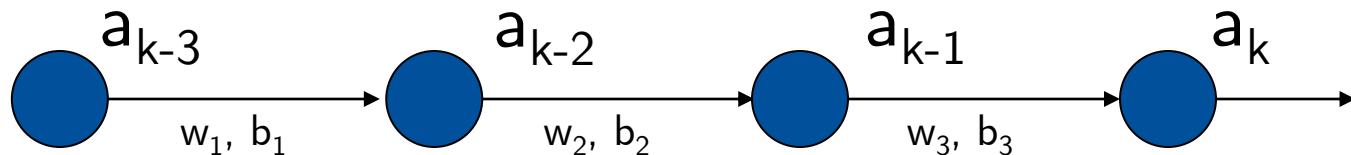
Backpropagation in ANN : Recap



Backpropagation in ANN : Recap

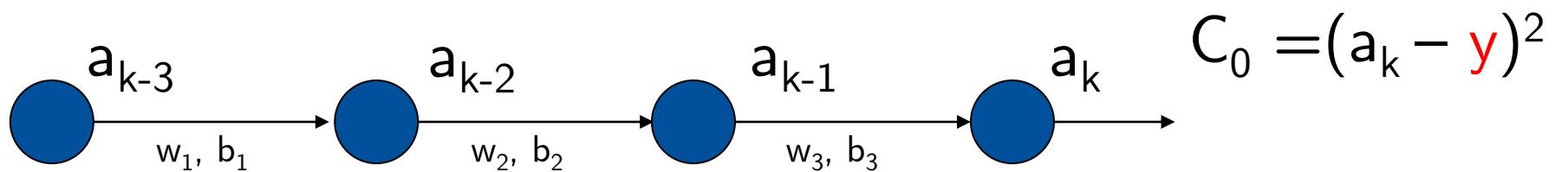


Backpropagation in ANN : Recap



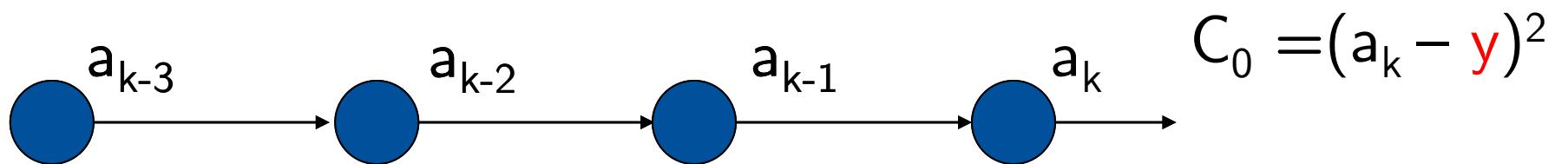
$$a_k = \sigma(w_k a_{k-1} + b_k)$$

Backpropagation in ANN : Recap



$$a_k = \sigma(w_k a_{k-1} + b_k)$$

Backpropagation in ANN : Recap



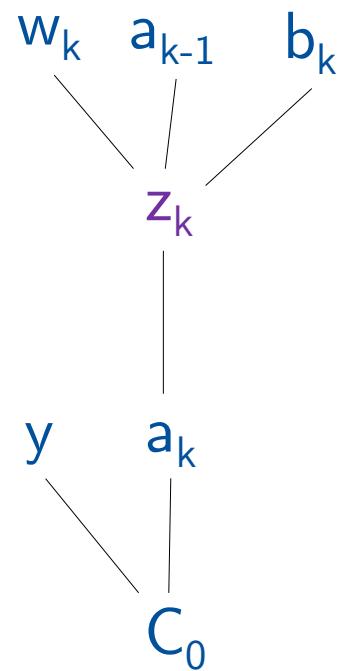
$$C_0 = (a_k - y)^2$$

$$a_k = \sigma(w_k a_{k-1} + b_k)$$

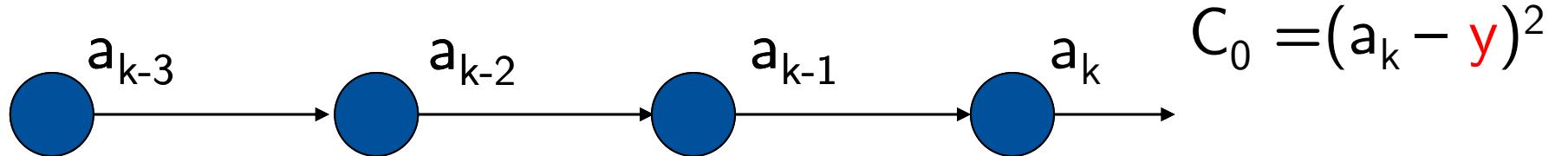
Now, if $z_k = w_k a_{k-1} + b_k$

Then, $a_k = \sigma(z_k)$

Backpropagation in ANN : Recap



Dependency graph



$$a_k = \sigma(w_k a_{k-1} + b_k)$$

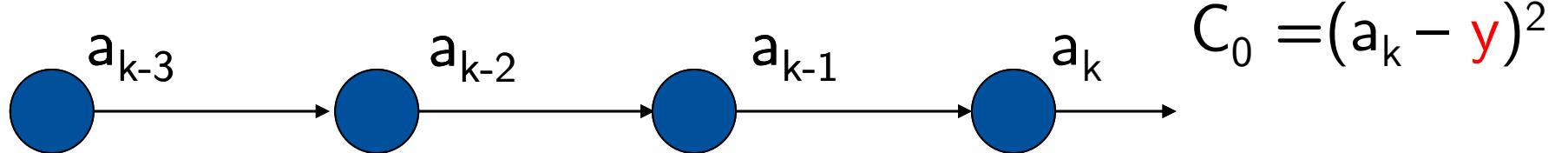
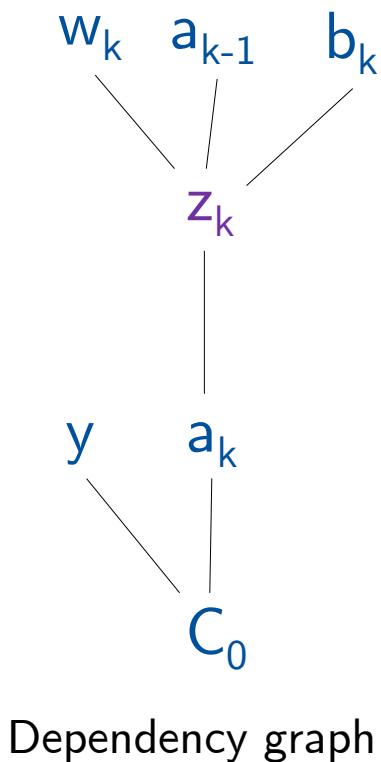
Now, if $z_k = w_k a_{k-1} + b_k$

Then, $a_k = \sigma(z_k)$

$$C_0 = (a_k - y)^2$$

Backpropagation in ANN : Recap

Aim is to compute : $\frac{\partial C_0}{\partial w_k}$



$$a_k = \sigma(w_k a_{k-1} + b_k)$$

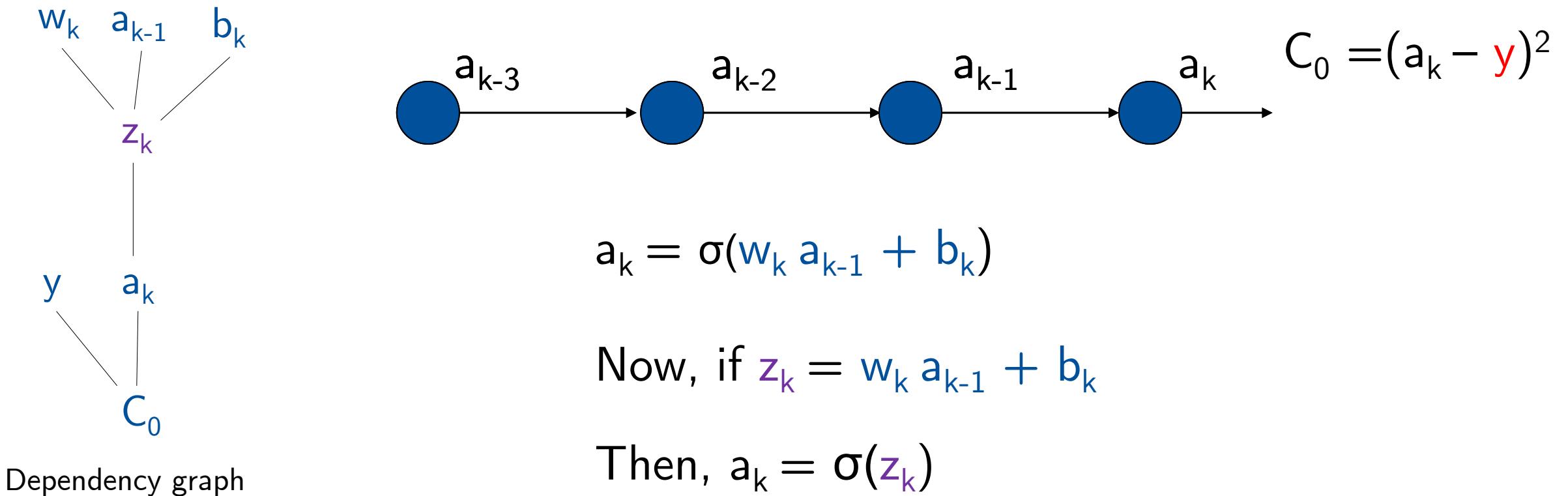
Now, if $z_k = w_k a_{k-1} + b_k$

Then, $a_k = \sigma(z_k)$

Backpropagation in ANN : Recap

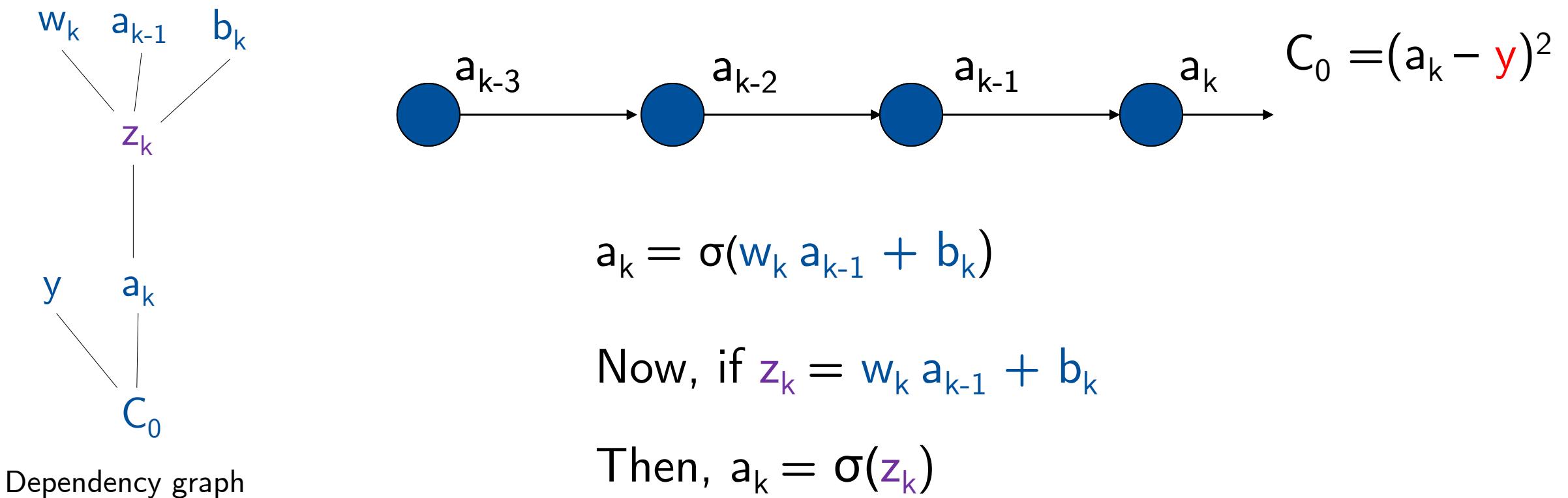
As there is a dependency, we need to apply chain rule

$$\frac{\partial C_0}{\partial w_k} = \frac{\partial z_k}{\partial w_k} \frac{\partial a_k}{\partial z_k} \frac{\partial C_0}{\partial a_k}$$

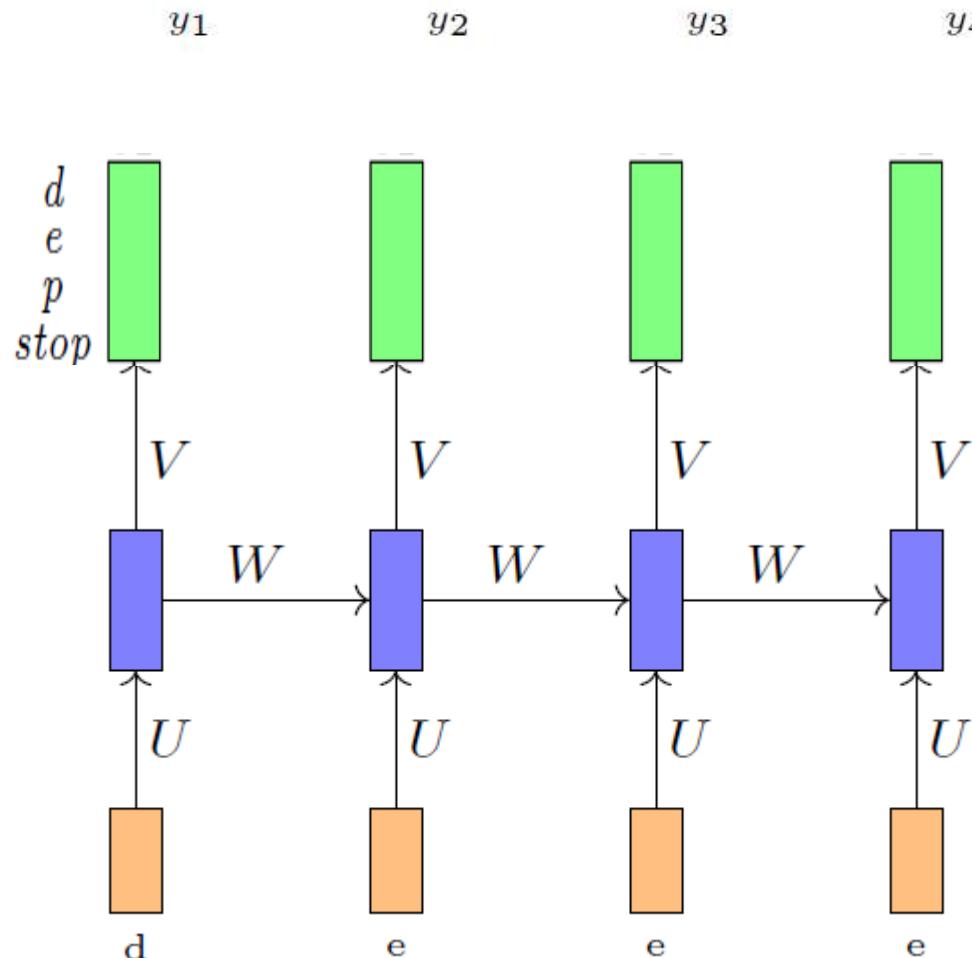


Backpropagation in ANN : Recap

$$\frac{\partial C_0}{\partial w_k} = \frac{\partial z_k}{\partial w_k} \frac{\partial a_k}{\partial z_k} \frac{\partial C_0}{\partial a_k} = a_{k-1} \sigma'(z_k) * 2*(a_k - y)$$



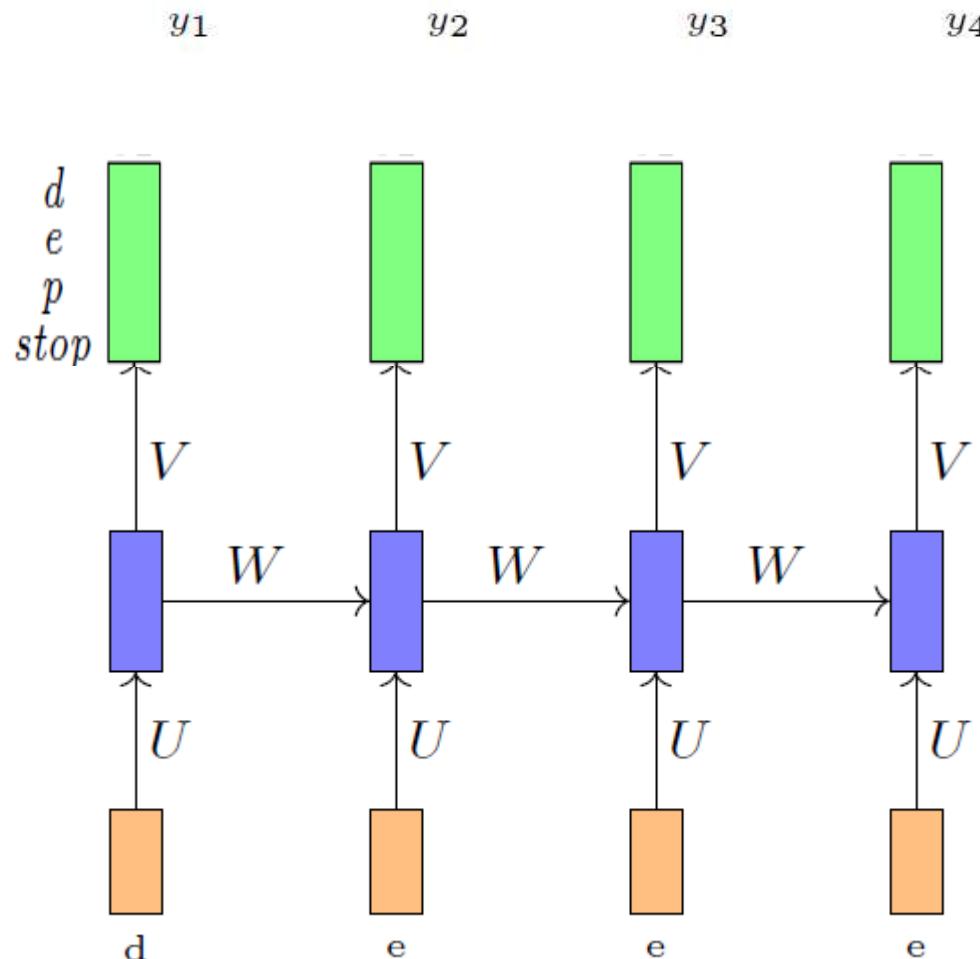
Training RNNs : Backpropagation through time (BPTT)



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

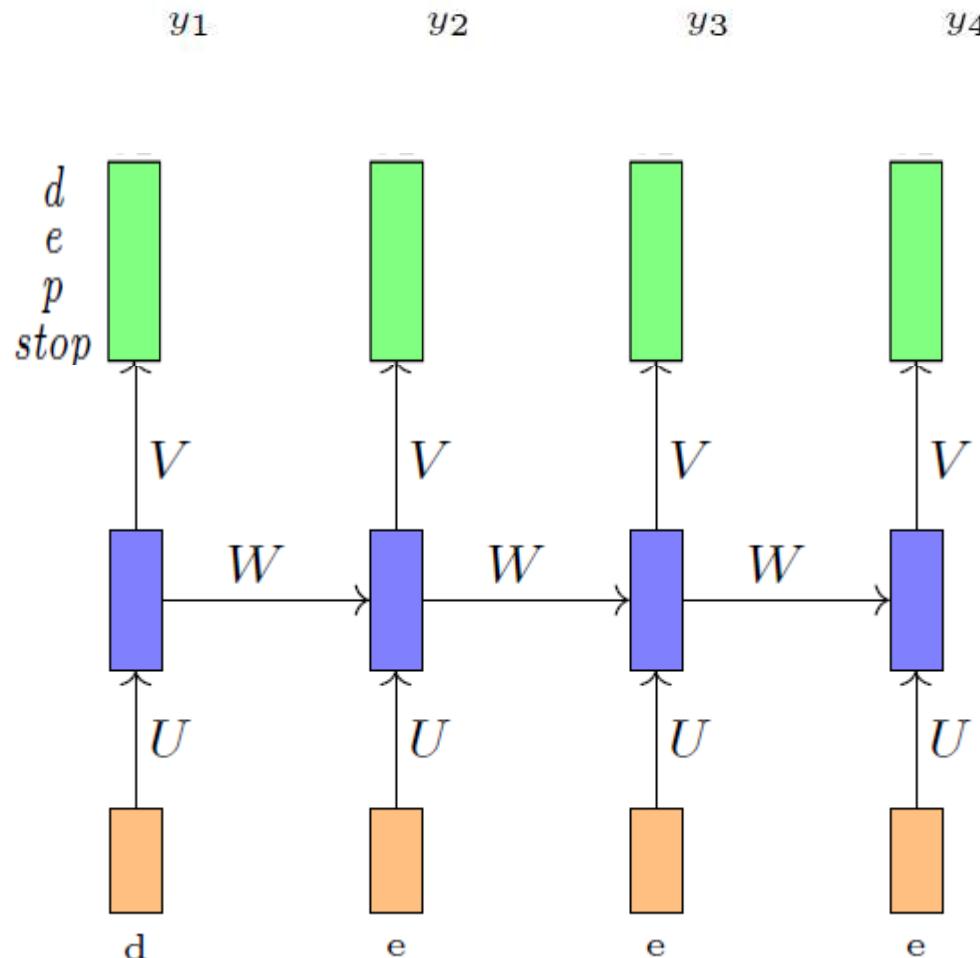
Training RNNs : Backpropagation through time (BPTT)

- For instance, consider the task of auto-completion (predicting the next character).



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

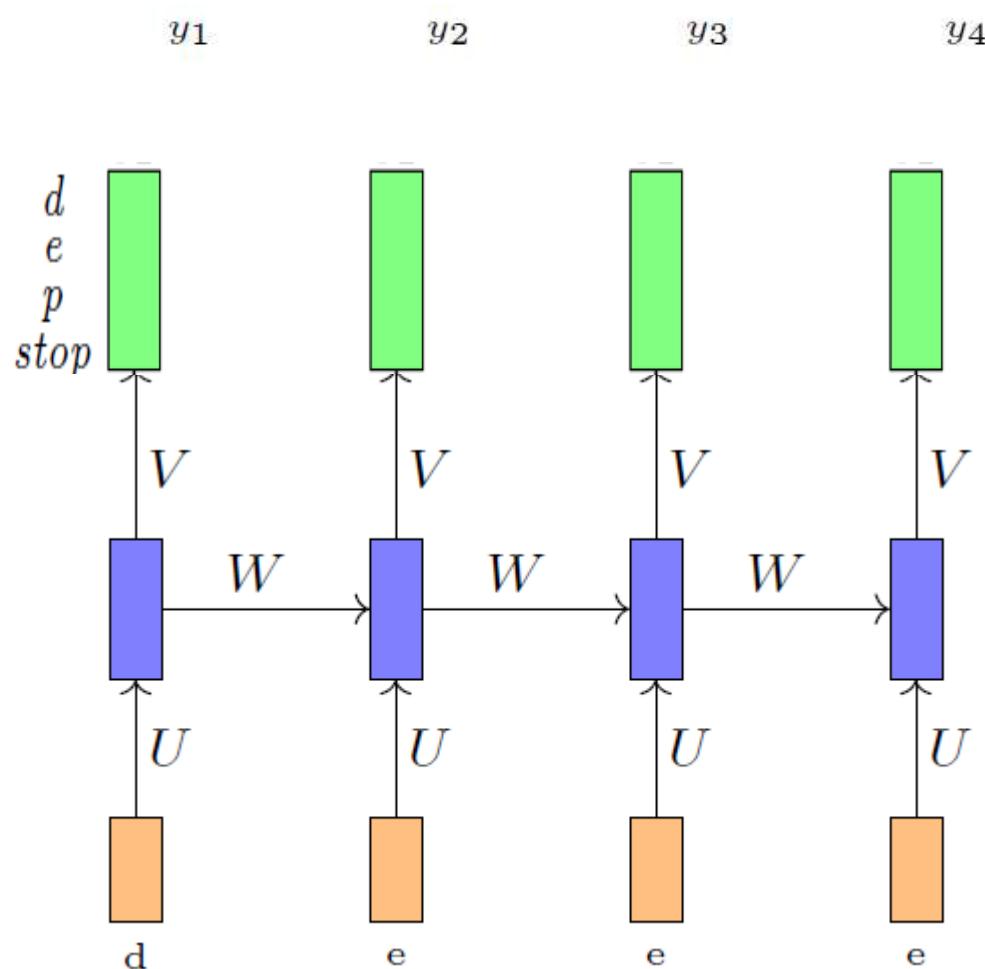
Training RNNs : Backpropagation through time (BPTT)



- For instance, consider the task of auto-completion (predicting the next character).
- For simplicity we assume that there are only 4 characters in our vocabulary (d, e, p, <stop>).

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

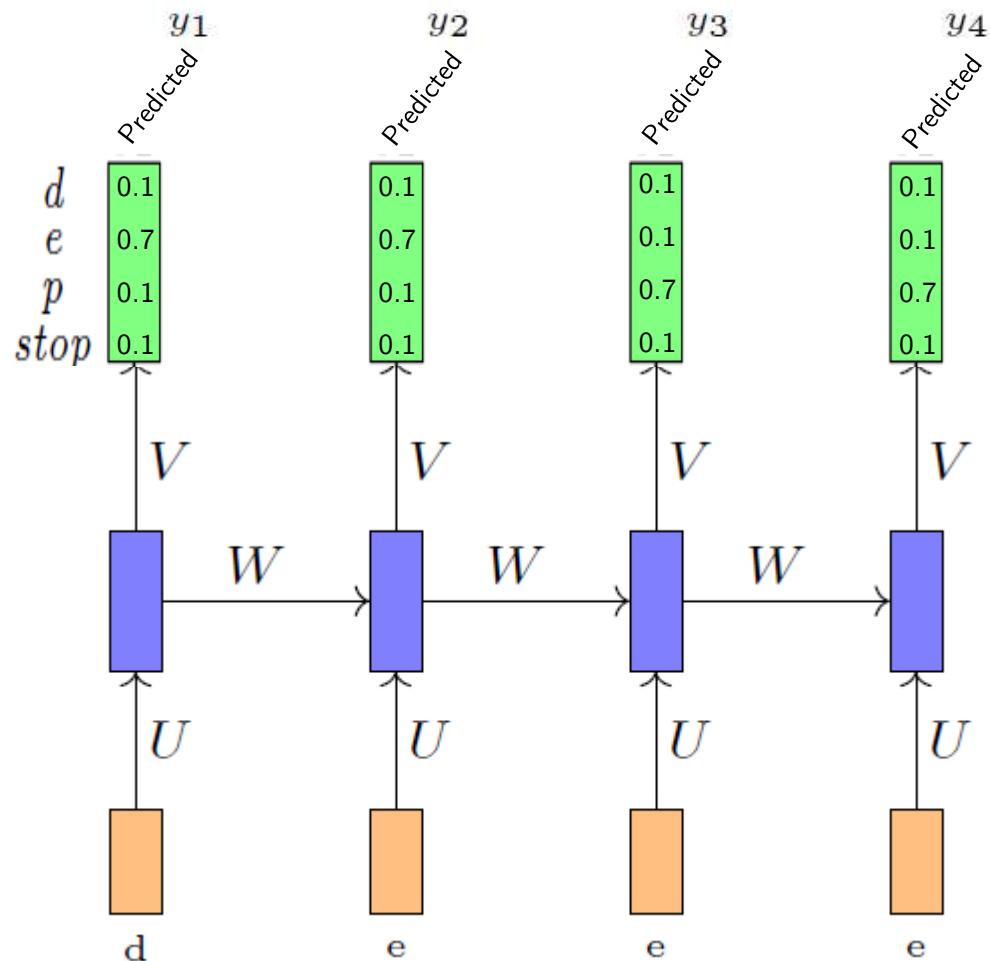
Training RNNs : Backpropagation through time (BPTT)



- For instance, consider the task of auto-completion (predicting the next character).
- For simplicity we assume that there are only 4 characters in our vocabulary (d, e, p, <stop>).
- At each timestep we want to predict one of these 4 characters.

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

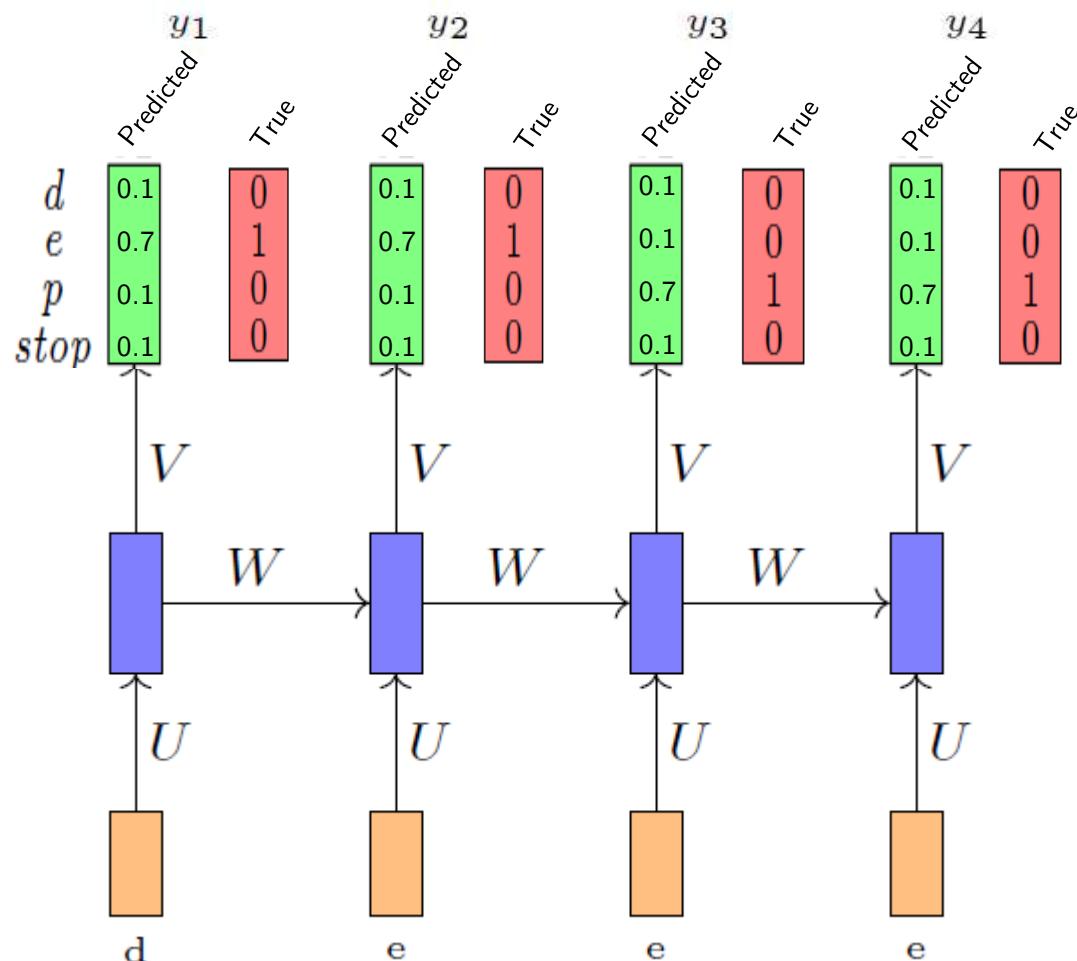
Training RNNs : Backpropagation through time (BPTT)



- For instance, consider the task of auto-completion (predicting the next character).
- For simplicity we assume that there are only 4 characters in our vocabulary ($d, e, p, \langle\text{stop}\rangle$).
- At each timestep we want to predict one of these 4 characters.
- Suppose we initialize U, V, W randomly and the network predicts the probabilities (green block)

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

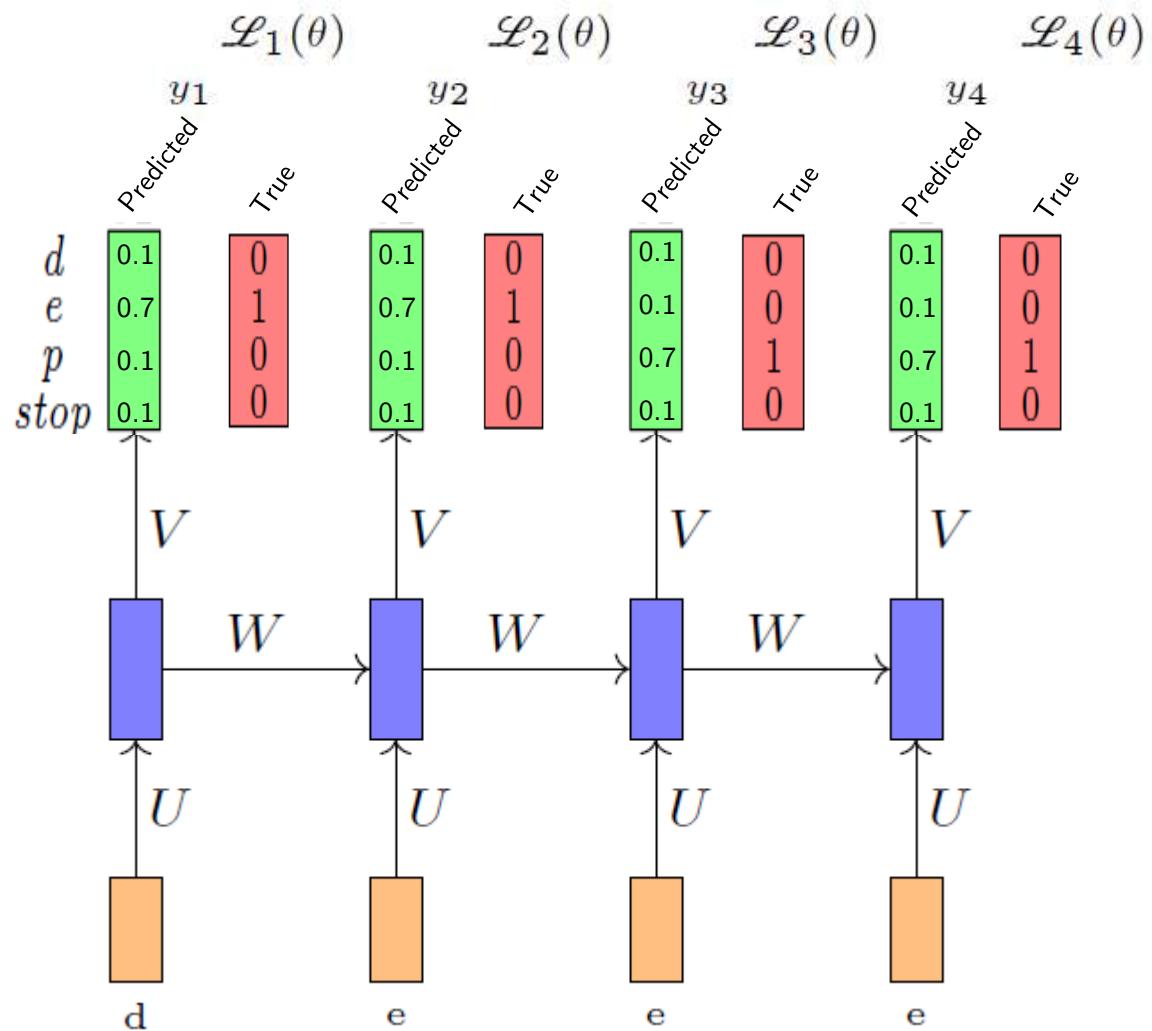
Training RNNs : Backpropagation through time (BPTT)



- For instance, consider the task of auto-completion (predicting the next character).
- For simplicity we assume that there are only 4 characters in our vocabulary (d, e, p, <stop>).
- At each timestep we want to predict one of these 4 characters.
- Suppose we initialize U , V , W randomly and the network predicts the probabilities (green block)
- And the true probabilities are as shown (red block).

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

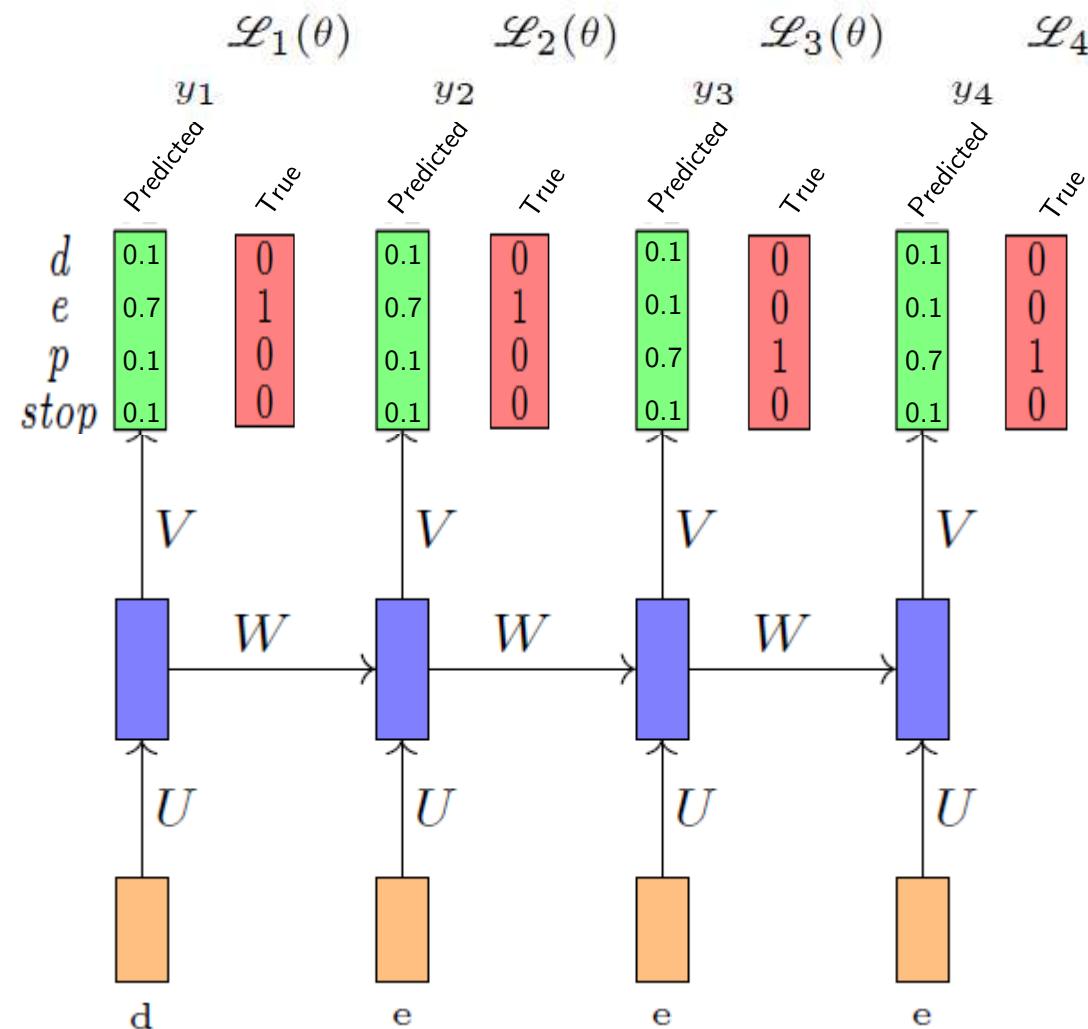
Training RNNs : Backpropagation through time (BPTT)



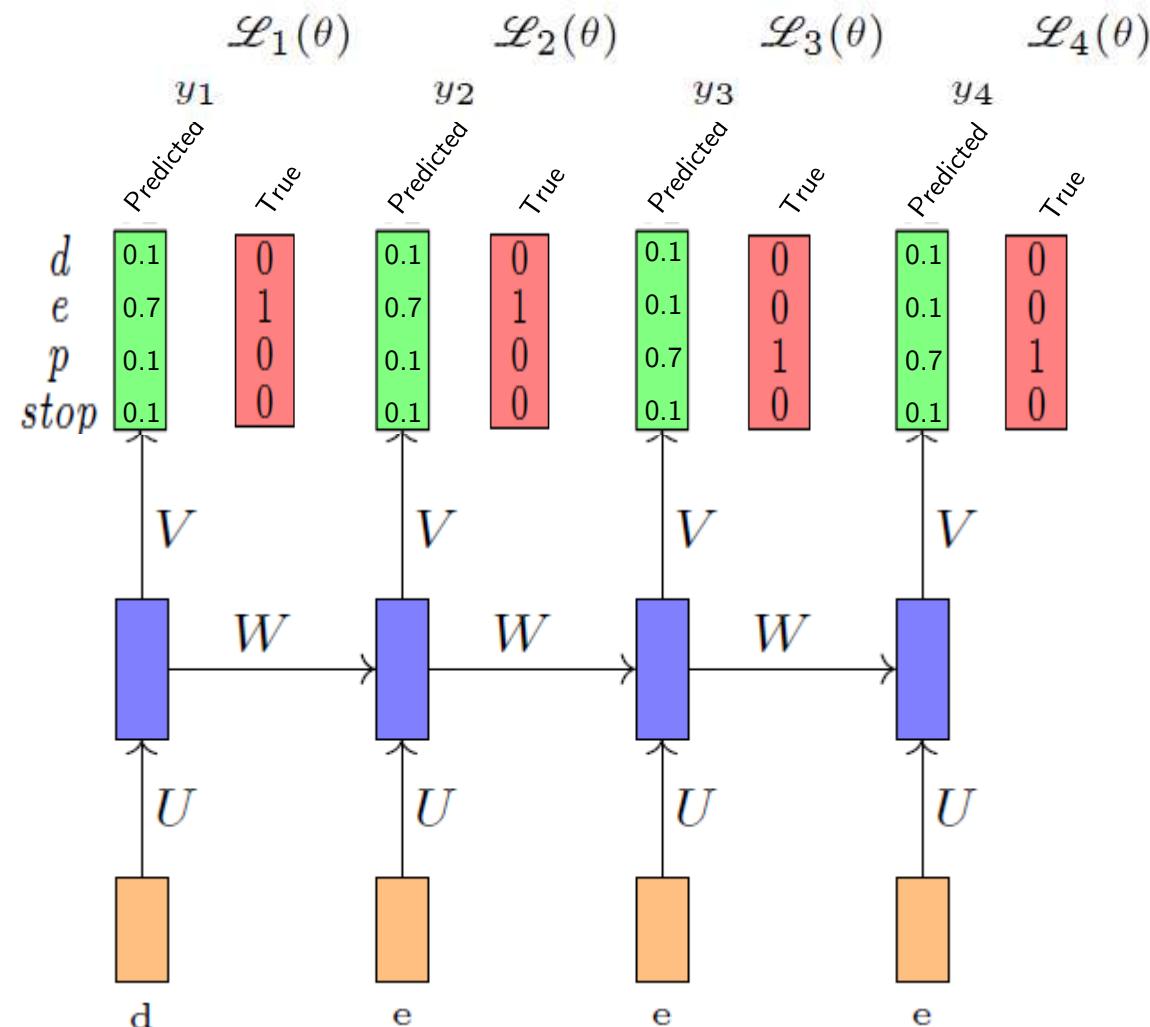
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

- For instance, consider the task of auto-completion (predicting the next character).
- For simplicity we assume that there are only 4 characters in our vocabulary ($d, e, p, \langle\text{stop}\rangle$).
- At each timestep we want to predict one of these 4 characters.
- Suppose we initialize U, V, W randomly and the network predicts the probabilities (green block).
- And the true probabilities are as shown (red block).
- At each time step, the loss $L_i(\theta)$ is calculated, where $\theta = \{U, V, W, b, c\}$ is the set of parameters.

Training RNNs : Backpropagation through time (BPTT)

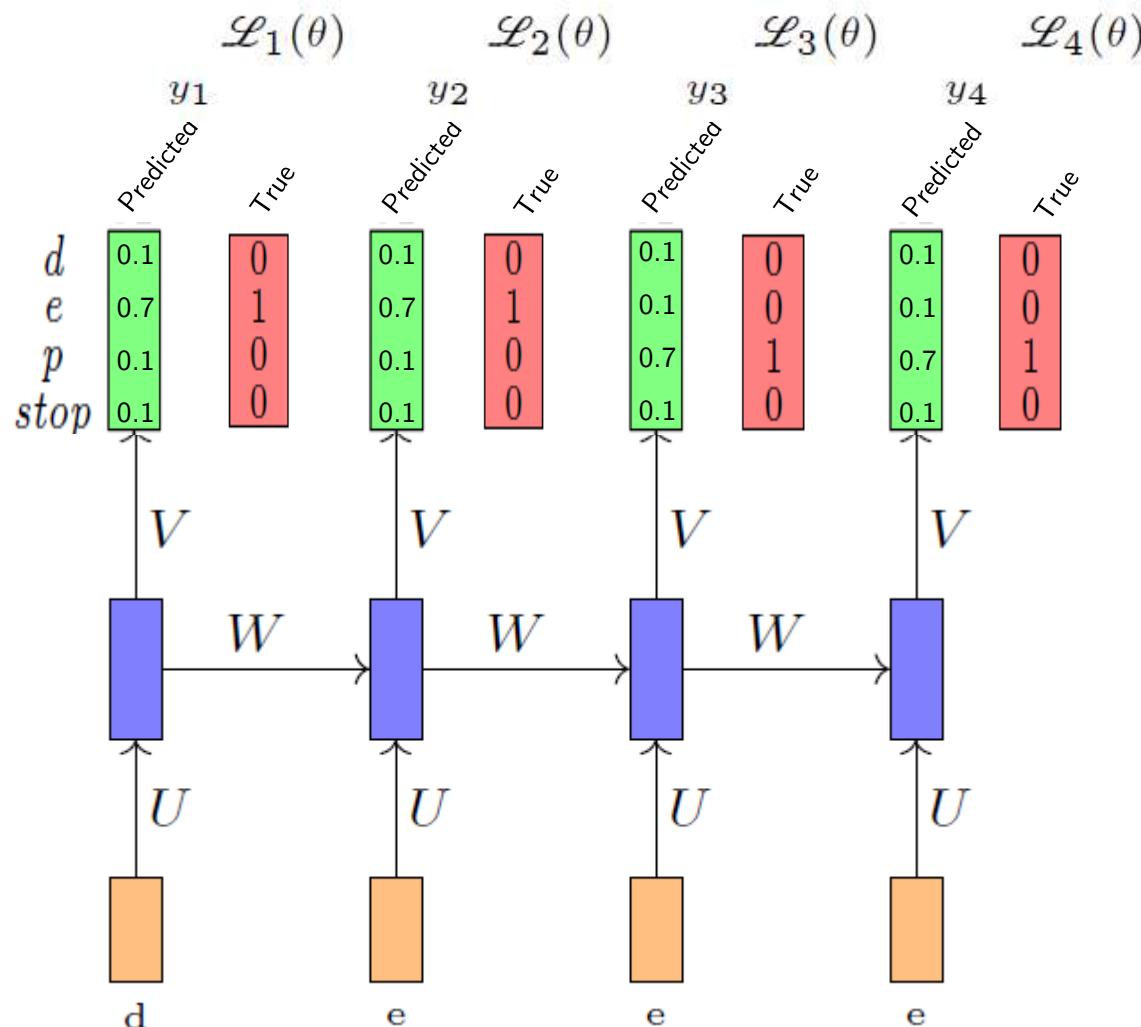


Training RNNs : Backpropagation through time (BPTT)



To train the RNNs we need to answer two questions:

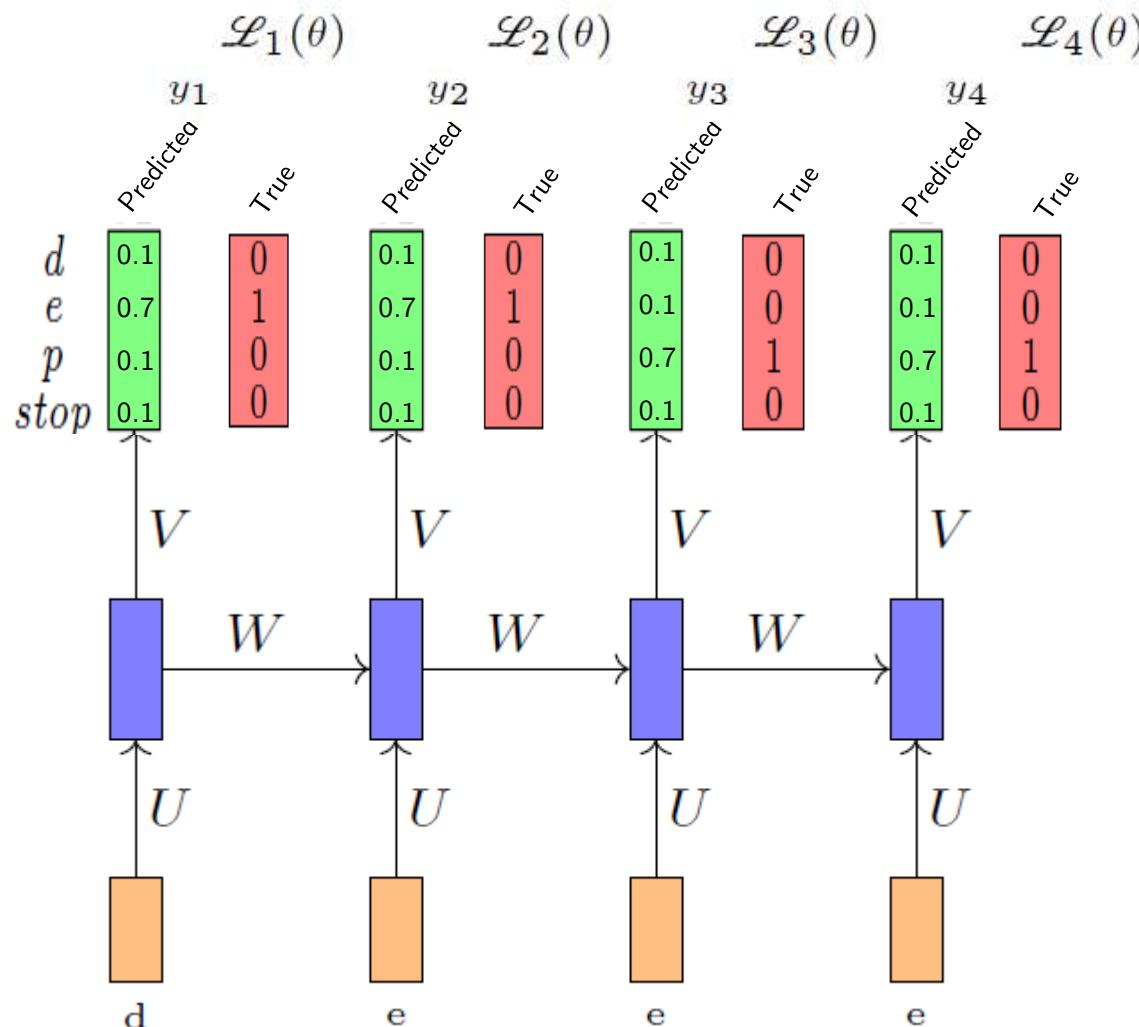
Training RNNs : Backpropagation through time (BPTT)



To train the RNNs we need to answer two questions:

- 1) What is the total loss made by the model ?

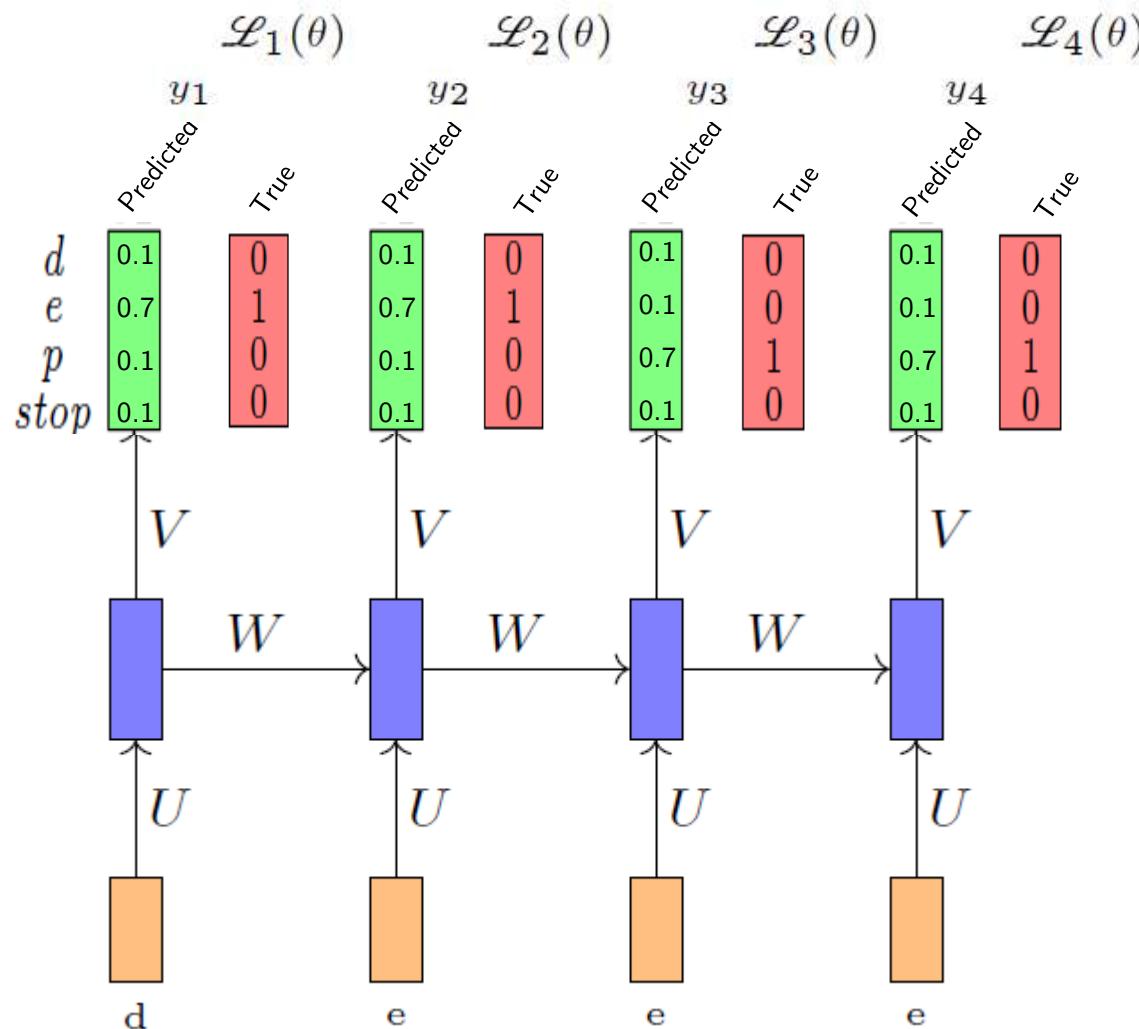
Training RNNs : Backpropagation through time (BPTT)



To train the RNNs we need to answer two questions:

- 1) What is the total loss made by the model ?
- 2) How do we backpropagate this loss and update the parameters of the network ?

Training RNNs : Backpropagation through time (BPTT)



To train the RNNs we need to answer two questions:

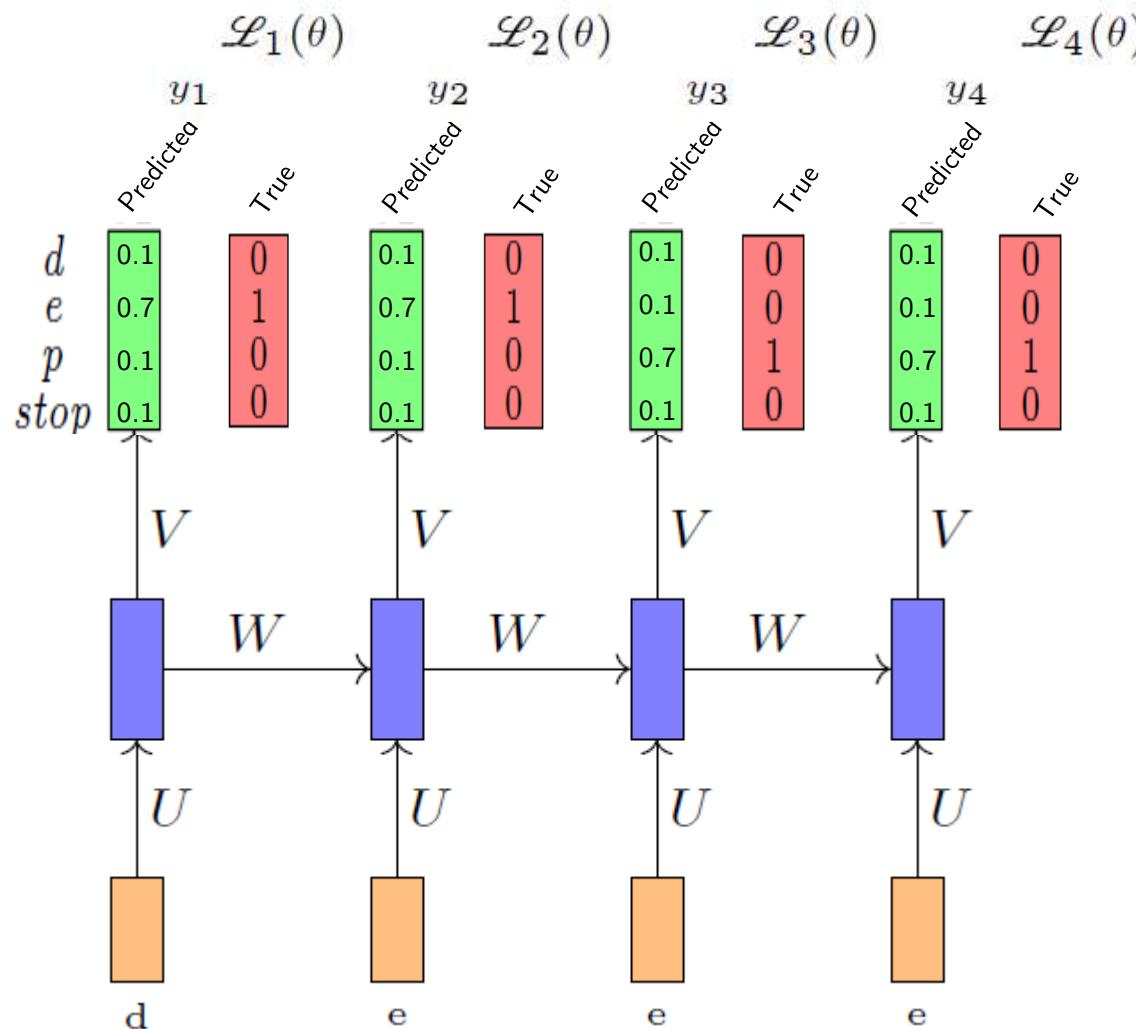
- 1) What is the total loss made by the model ?

Ans: the Sum of individual losses

$$\mathcal{L}(\theta) = \sum_{t=1}^T \mathcal{L}_t(\theta)$$

- 2) How do we backpropagate this loss and update the parameters of the network ?

Training RNNs : Backpropagation through time (BPTT)



To train the RNNs we need to answer two questions:

- 1) What is the total loss made by the model ?

Ans: the Sum of individual losses

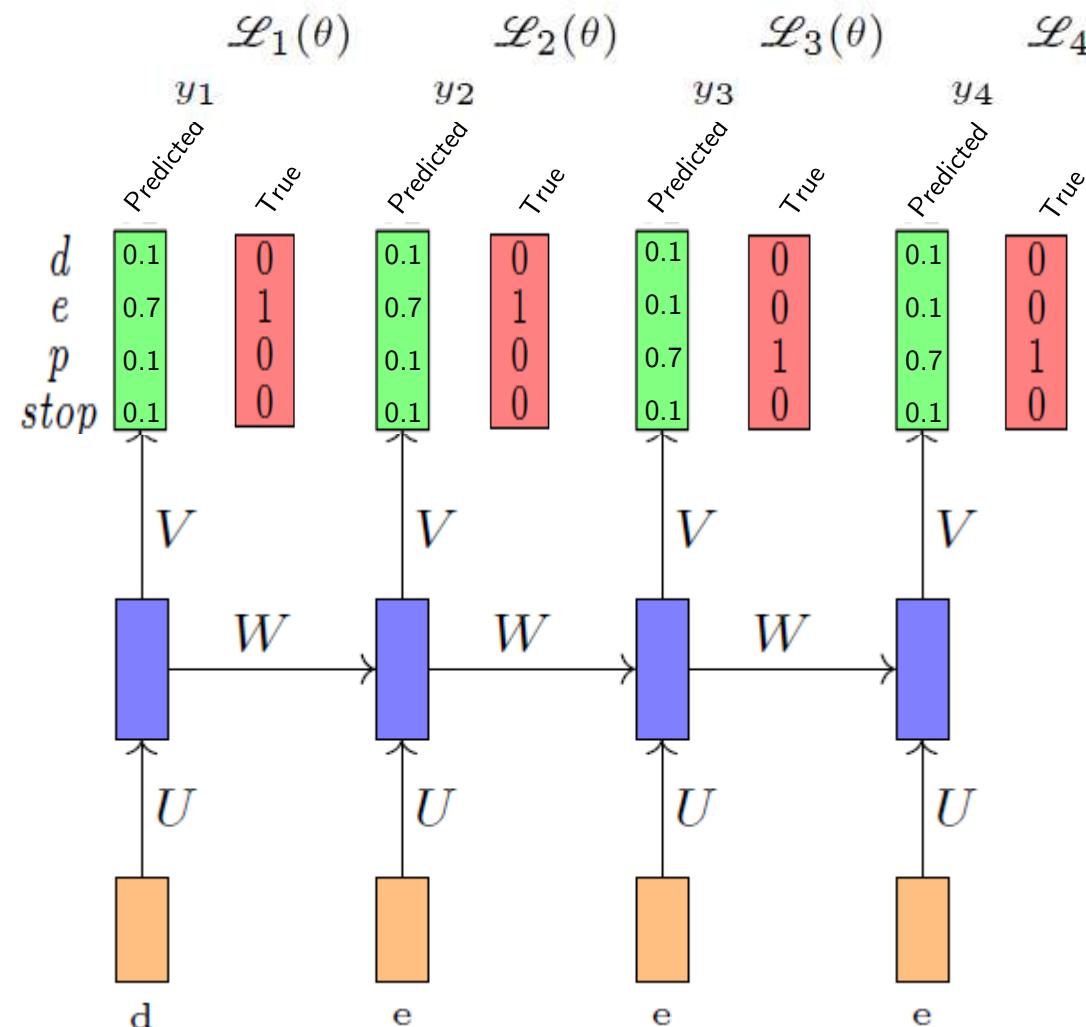
$$\mathcal{L}(\theta) = \sum_{t=1}^T \mathcal{L}_t(\theta)$$

- 2) How do we backpropagate this loss and update the parameters of the network ?

Ans: BPTT by computing the partial derivative of L w.r.t U, V, W, b, c

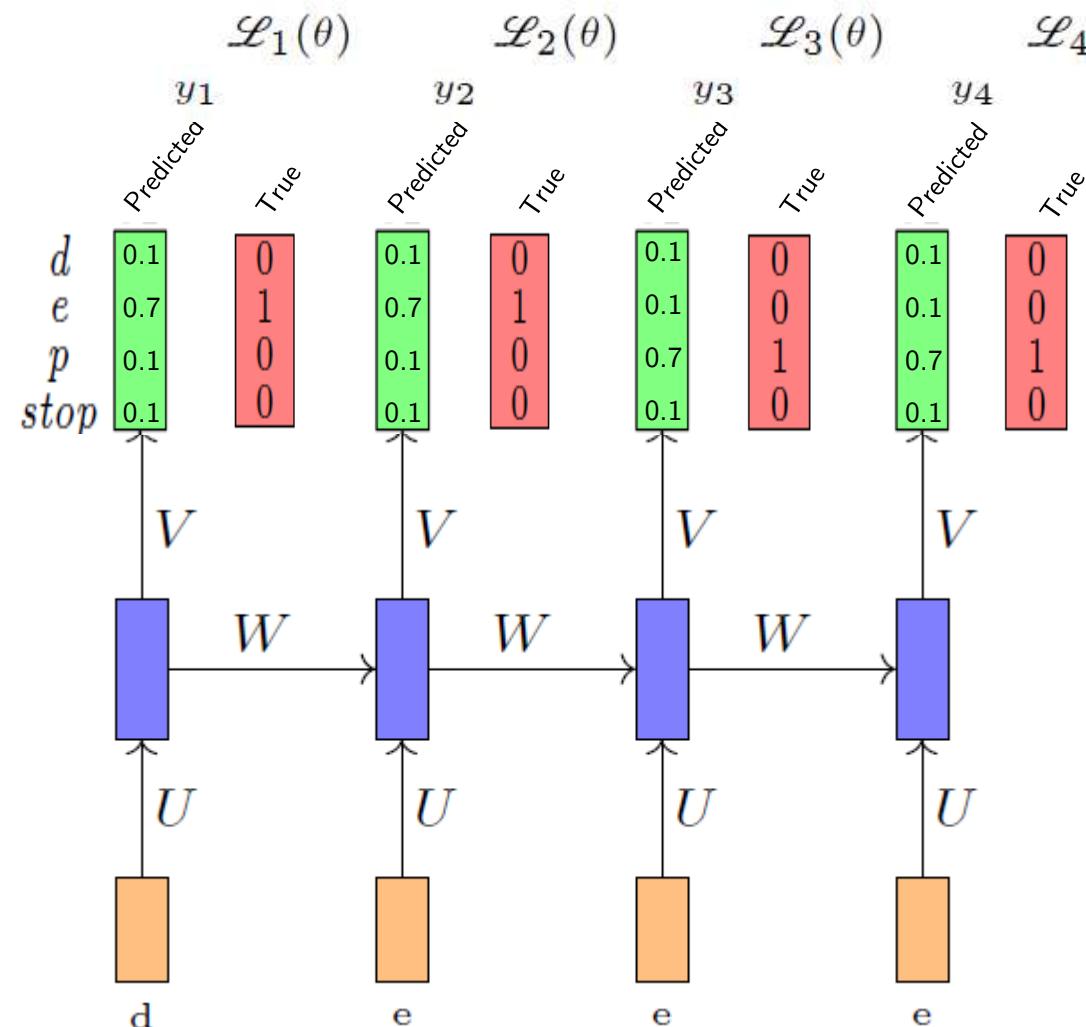
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)



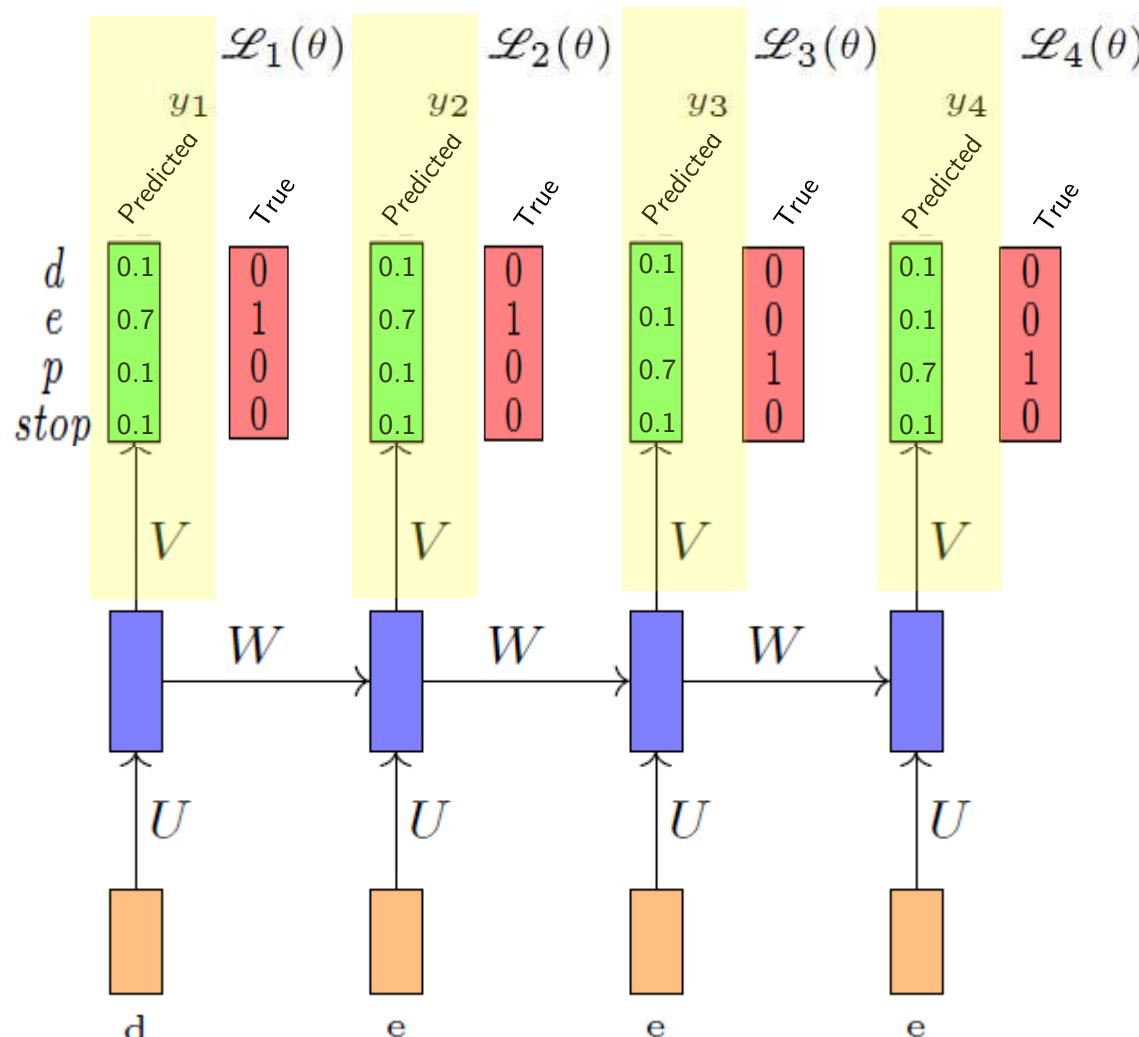
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Let us consider $\frac{\partial \mathcal{L}(\theta)}{\partial V}$ (V is a matrix so ideally we should write $\nabla_v \mathcal{L}(\theta)$)

$$\frac{\partial \mathcal{L}(\theta)}{\partial V} = \sum_{t=1}^T \frac{\partial \mathcal{L}_t(\theta)}{\partial V}$$

For example, if:

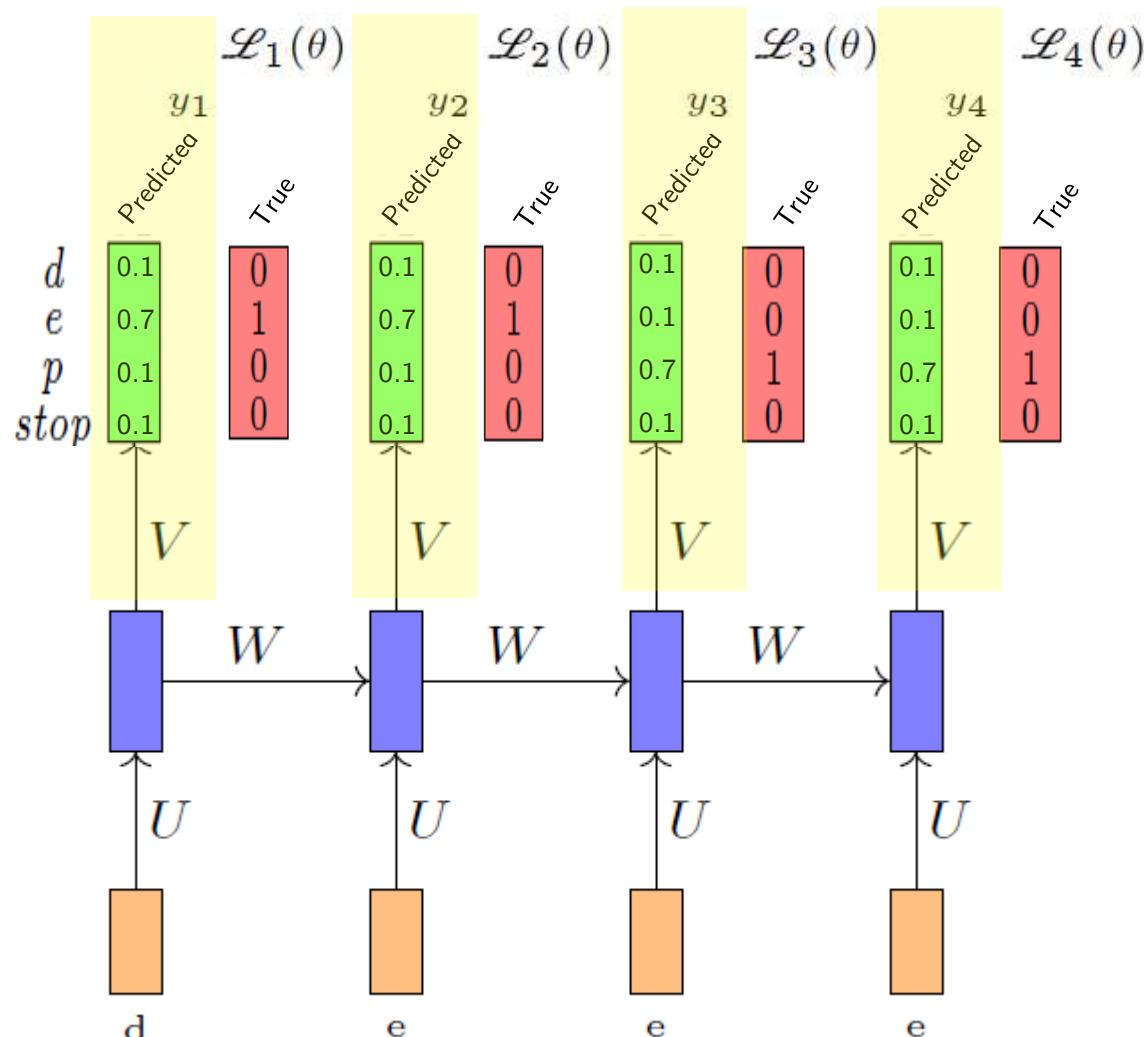
$$\hat{y}_4 = O(VS_4 + c) \quad \text{and} \quad L_4 = \frac{1}{2}(y_4 - \hat{y}_4)^2$$

Ignoring bias and considering O as linear:

$$\frac{\partial L_4}{\partial V} = \frac{\partial L_4}{\partial \hat{y}_4} \frac{\partial \hat{y}_4}{\partial V}$$

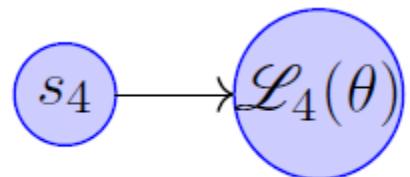
$$\frac{\partial L_4}{\partial V} = -(y_4 - \hat{y}_4) \cdot s_4$$

Training RNNs : Backpropagation through time (BPTT)



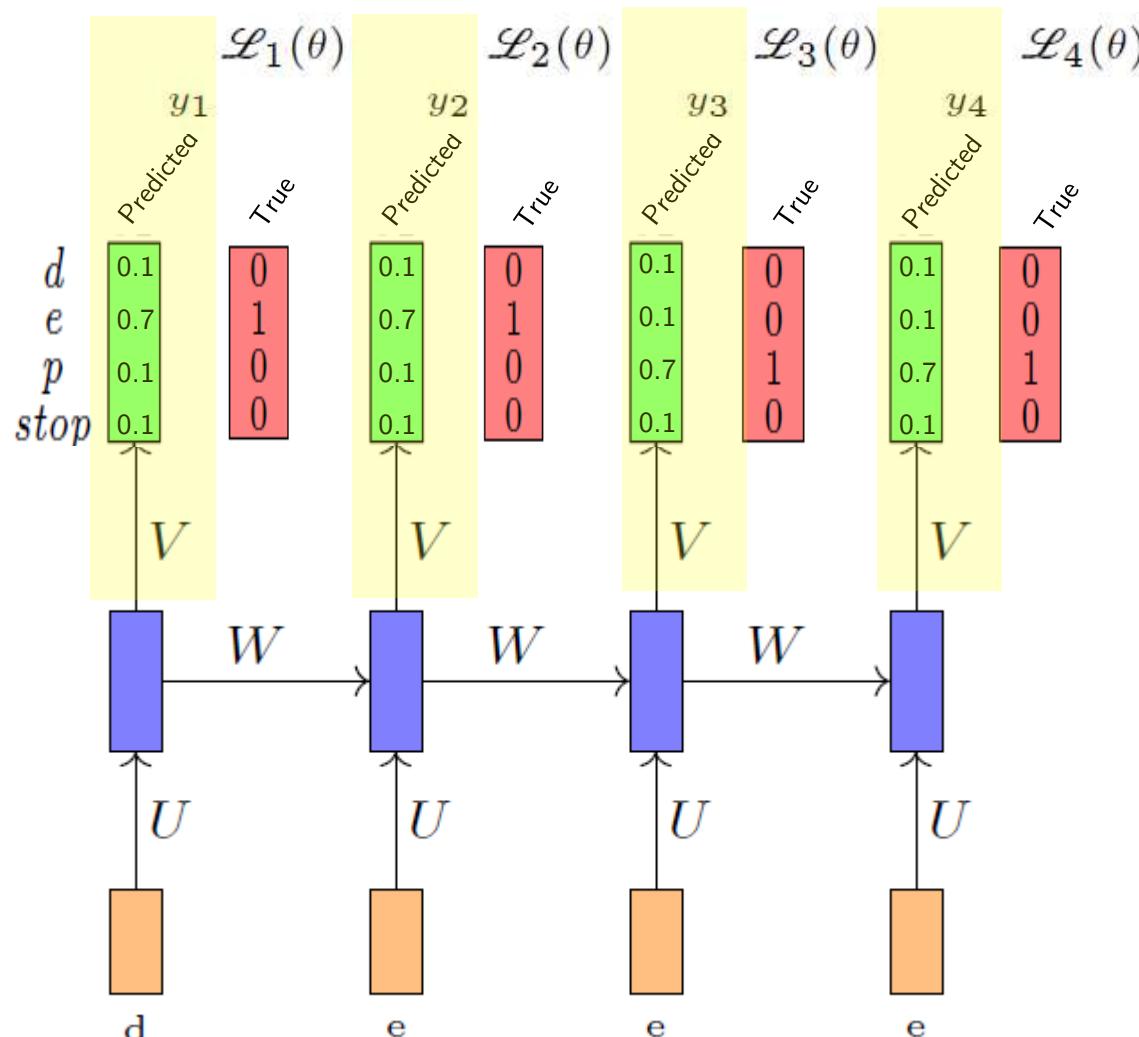
Let us consider the derivative $\frac{\partial \mathcal{L}(\theta)}{\partial W}$

$$\frac{\partial \mathcal{L}(\theta)}{\partial W} = \sum_{t=1}^T \frac{\partial \mathcal{L}_t(\theta)}{\partial W}$$



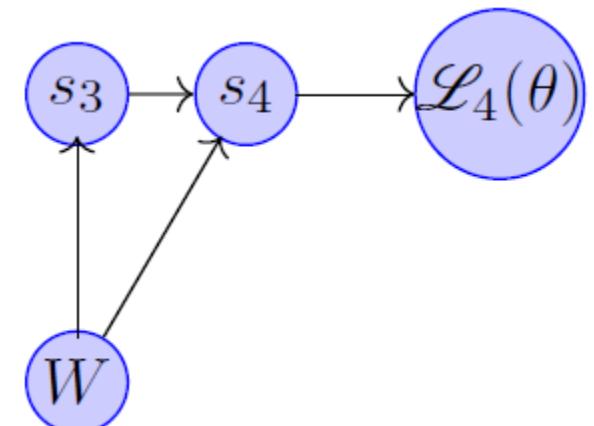
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)



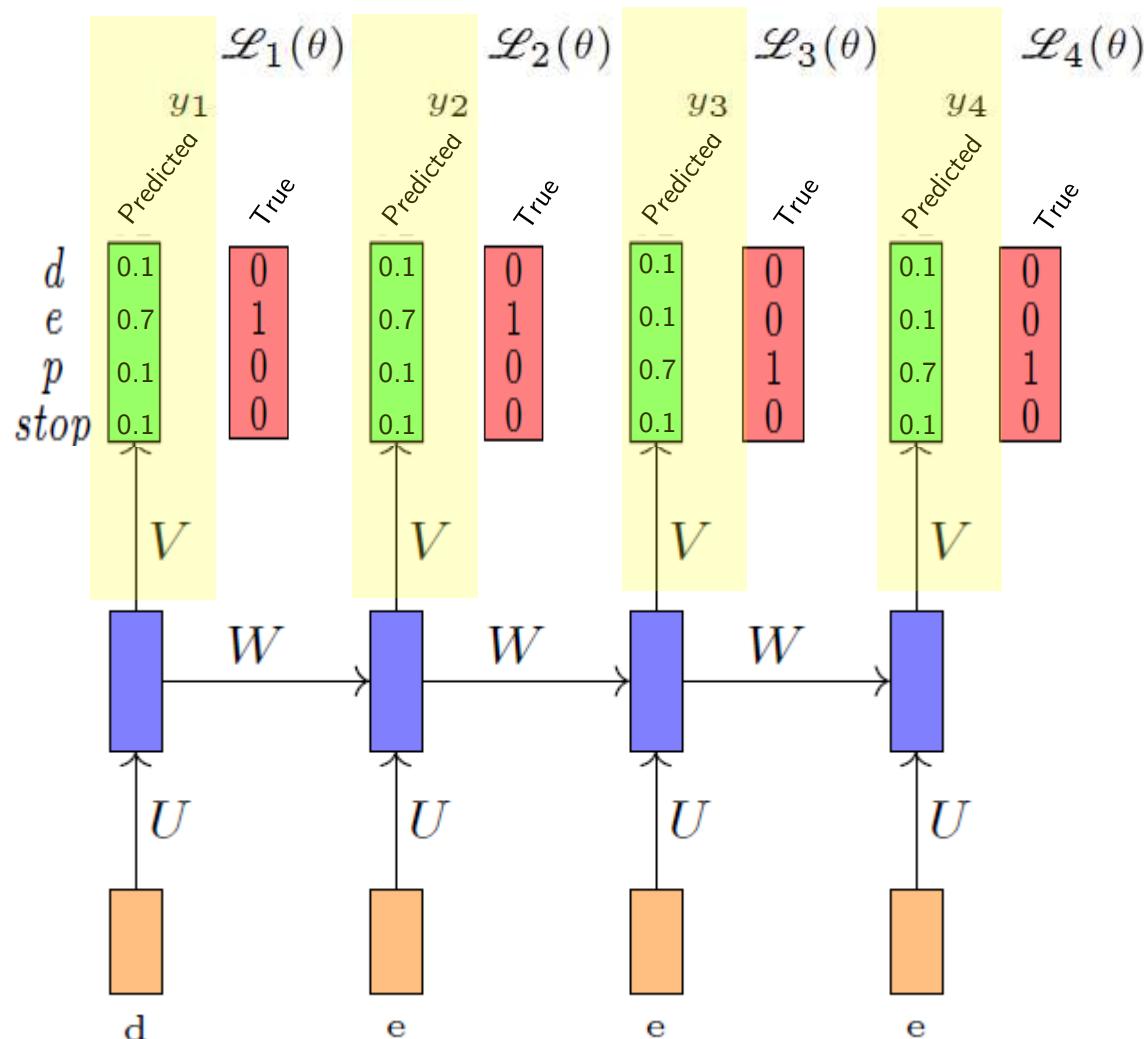
Let us consider the derivative $\frac{\partial \mathcal{L}(\theta)}{\partial W}$

$$\frac{\partial \mathcal{L}(\theta)}{\partial W} = \sum_{t=1}^T \frac{\partial \mathcal{L}_t(\theta)}{\partial W}$$



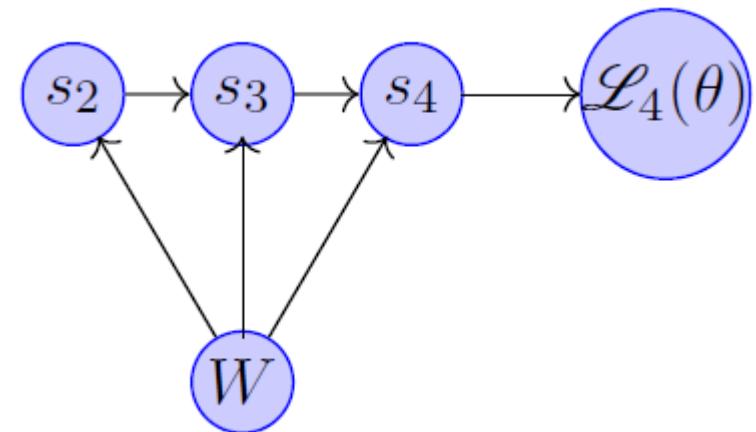
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)



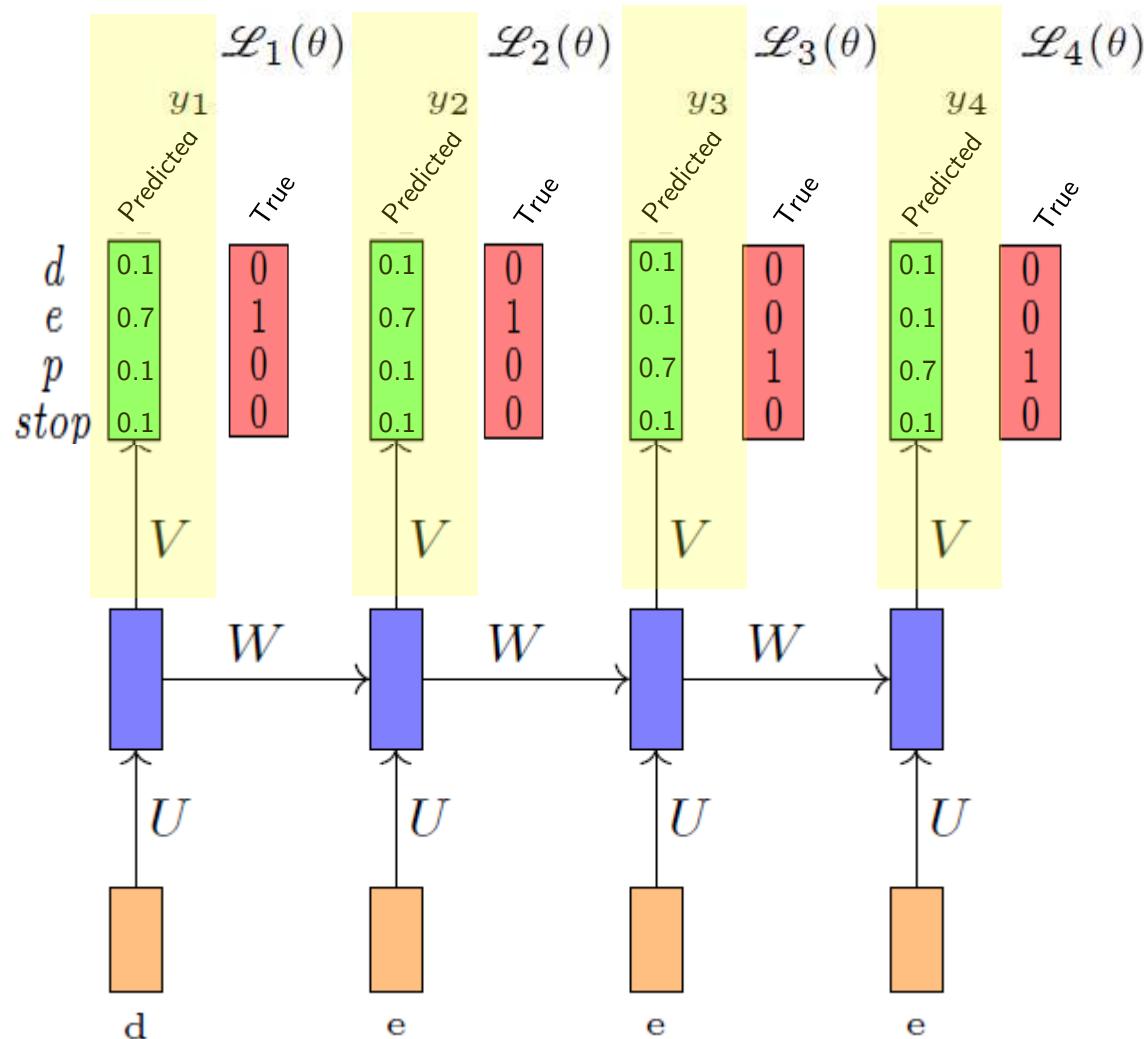
Let us consider the derivative $\frac{\partial \mathcal{L}(\theta)}{\partial W}$

$$\frac{\partial \mathcal{L}(\theta)}{\partial W} = \sum_{t=1}^T \frac{\partial \mathcal{L}_t(\theta)}{\partial W}$$



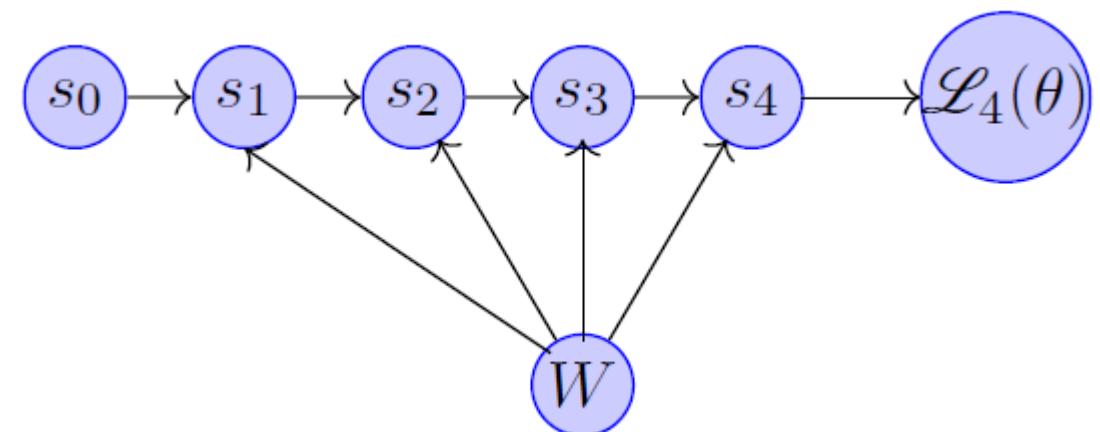
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)



Let us consider the derivative $\frac{\partial \mathcal{L}(\theta)}{\partial W}$

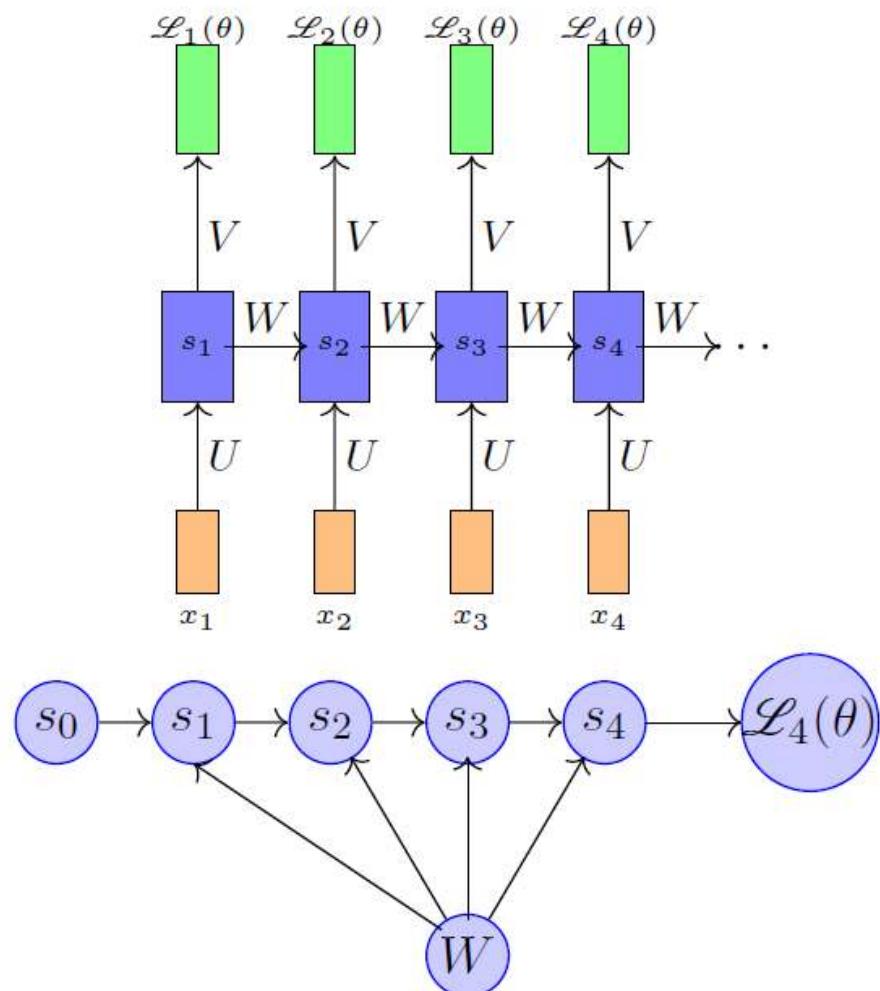
$$\frac{\partial \mathcal{L}(\theta)}{\partial W} = \sum_{t=1}^T \frac{\partial \mathcal{L}_t(\theta)}{\partial W}$$



Ordered network

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)



$$\frac{\partial \mathcal{L}_4(\theta)}{\partial W} = \frac{\partial \mathcal{L}_4(\theta)}{\partial s_4} \frac{\partial s_4}{\partial W}$$

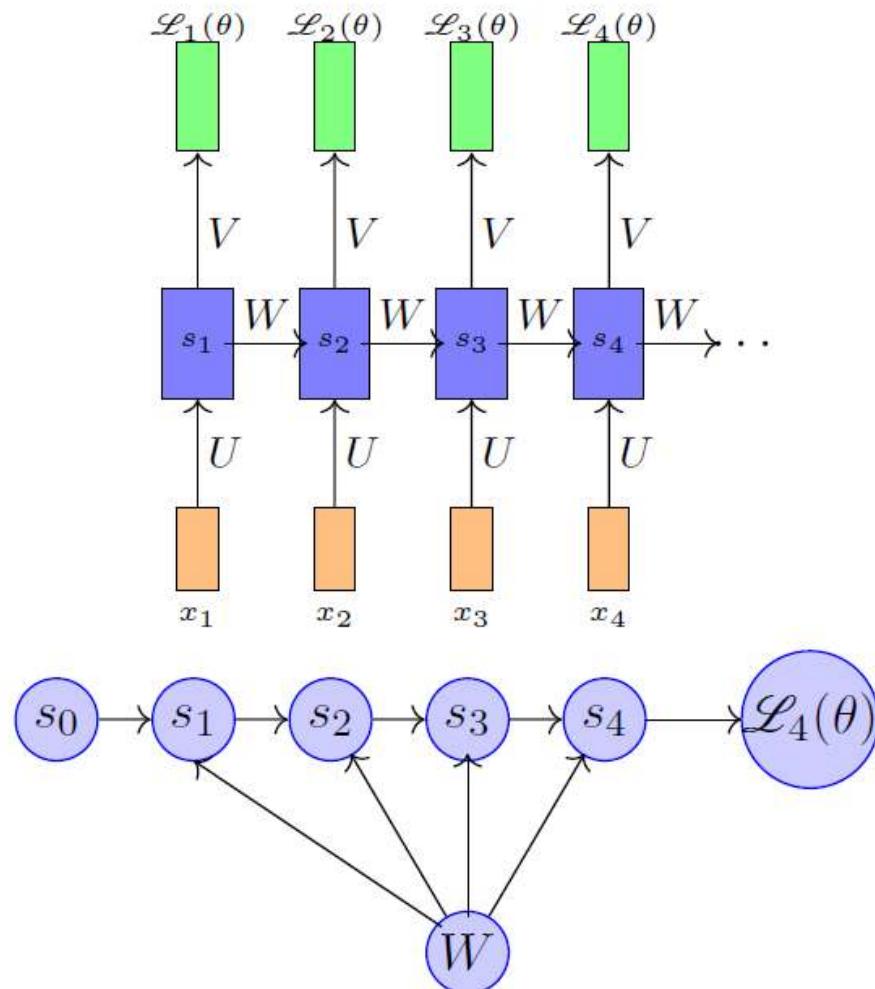
$\frac{\partial \mathcal{L}_4(\theta)}{\partial s_4}$ computation is straight forward

But how do we compute $\frac{\partial s_4}{\partial W}$

$$s_4 = \sigma(Ws_3 + b)$$

In such an ordered network, we can't compute $\frac{\partial s_4}{\partial W}$ by simply treating s_3 as a constant (because it also depends on W)

Training RNNs : Backpropagation through time (BPTT)



But how do we compute $\frac{\partial s_4}{\partial W}$

In such networks the total derivative $\frac{\partial s_4}{\partial W}$ has two parts

Explicit : $\frac{\partial^+ s_4}{\partial W}$, treating all other inputs as constant

Implicit : Summing over all indirect paths from s_4 to W

Training RNNs : Backpropagation through time (BPTT)

$$\frac{\partial s_4}{\partial W} = \underbrace{\frac{\partial^+ s_4}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial W}}_{\text{implicit}}$$

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)

$$\begin{aligned}\frac{\partial s_4}{\partial W} &= \underbrace{\frac{\partial^+ s_4}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial W}}_{\text{implicit}} \\ &= \frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3} \left[\underbrace{\frac{\partial^+ s_3}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial W}}_{\text{implicit}} \right]\end{aligned}$$

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)

$$\begin{aligned}\frac{\partial s_4}{\partial W} &= \underbrace{\frac{\partial^+ s_4}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial W}}_{\text{implicit}} \\ &= \frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3} \left[\underbrace{\frac{\partial^+ s_3}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial W}}_{\text{implicit}} \right] \\ &= \frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3} \frac{\partial^+ s_3}{\partial W} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \left[\frac{\partial^+ s_2}{\partial W} + \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W} \right]\end{aligned}$$

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)

$$\frac{\partial s_4}{\partial W} = \underbrace{\frac{\partial^+ s_4}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial W}}_{\text{implicit}}$$

$$= \frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3} \left[\underbrace{\frac{\partial^+ s_3}{\partial W}}_{\text{explicit}} + \underbrace{\frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial W}}_{\text{implicit}} \right]$$

$$= \frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3} \frac{\partial^+ s_3}{\partial W} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \left[\frac{\partial^+ s_2}{\partial W} + \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial W} \right]$$

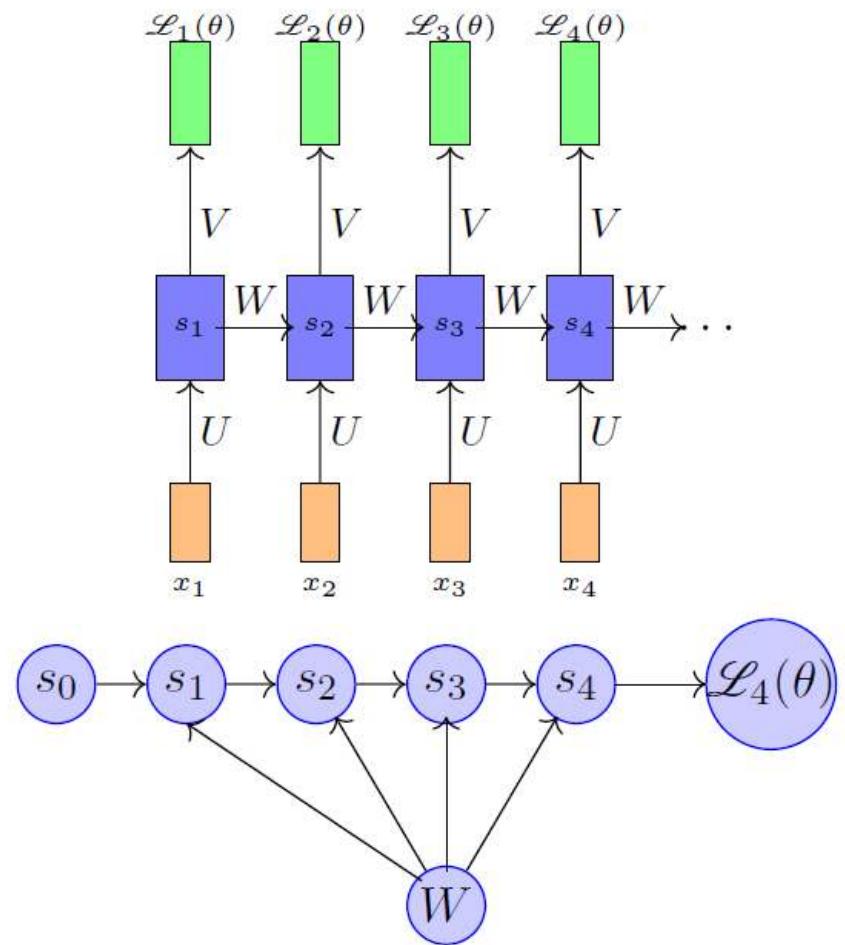
$$= \frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3} \frac{\partial^+ s_3}{\partial W} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial^+ s_2}{\partial W} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \left[\frac{\partial^+ s_1}{\partial W} \right]$$

For simplicity we will short-circuit some of the paths

$$\frac{\partial s_4}{\partial W} = \frac{\partial s_4}{\partial s_4} \frac{\partial^+ s_4}{\partial W} + \frac{\partial s_4}{\partial s_3} \frac{\partial^+ s_3}{\partial W} + \frac{\partial s_4}{\partial s_2} \frac{\partial^+ s_2}{\partial W} + \frac{\partial s_4}{\partial s_1} \frac{\partial^+ s_1}{\partial W} = \sum_{k=1}^4 \frac{\partial s_4}{\partial s_k} \frac{\partial^+ s_k}{\partial W}$$

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Training RNNs : Backpropagation through time (BPTT)



Finally we have

$$\frac{\partial \mathcal{L}_4(\theta)}{\partial W} = \frac{\partial \mathcal{L}_4(\theta)}{\partial s_4} \frac{\partial s_4}{\partial W}$$

$$\frac{\partial s_4}{\partial W} = \sum_{k=1}^4 \frac{\partial s_4}{\partial s_k} \frac{\partial^+ s_k}{\partial W}$$

$$\therefore \frac{\partial \mathcal{L}_t(\theta)}{\partial W} = \frac{\partial \mathcal{L}_t(\theta)}{\partial s_t} \sum_{k=1}^t \frac{\partial s_t}{\partial s_k} \frac{\partial^+ s_k}{\partial W}$$

This algorithm is called backpropagation through time (BPTT) as we backpropagate over all previous time steps

Back Propagation through time – Vanishing & Exploding gradient

We will now focus on $\frac{\partial s_t}{\partial s_k}$ and highlight an important problem in training RNN's using BPTT

$$\frac{\partial s_t}{\partial s_k} = \frac{\partial s_t}{\partial s_{t-1}} \frac{\partial s_{t-1}}{\partial s_{t-2}} \dots \frac{\partial s_{k+1}}{\partial s_k} = \prod_{j=k}^{t-1} \frac{\partial s_{j+1}}{\partial s_j}$$

Let us look at one such term in the product (i.e., $\frac{\partial s_{j+1}}{\partial s_j}$)

Recall that:

$$a_j = Ws_{j-1} + b$$

$$s_j = \sigma(a_j)$$

Therefore:

$$\frac{\partial s_{j+1}}{\partial s_j} = \frac{\partial s_j}{\partial s_{j-1}} = \frac{\frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}}{\frac{\partial a_j}{\partial s_{j-1}}}$$

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd}]$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]$$

$$\frac{\partial s_j}{\partial a_j} = \begin{bmatrix} \cdot & \cdot & \cdot & \cdots & \cdot \end{bmatrix}$$

Back Propagation through time – Vanishing & Exploding gradient

We will now focus on $\frac{\partial s_t}{\partial s_k}$ and highlight an important problem in training RNN's using BPTT

$$\frac{\partial s_t}{\partial s_k} = \frac{\partial s_t}{\partial s_{t-1}} \frac{\partial s_{t-1}}{\partial s_{t-2}} \cdots \frac{\partial s_{k+1}}{\partial s_k}$$

Let us look at one such term in the product (i.e., $\frac{\partial s_{j+1}}{\partial s_j}$)

$$a_j = Ws_{j-1} + b$$

$$s_j = \sigma(a_j)$$

$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd}]$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]$$

$$\frac{\partial s_j}{\partial a_j} = \begin{bmatrix} \frac{\partial s_{j1}}{\partial a_{j1}} & \frac{\partial s_{j2}}{\partial a_{j1}} & \frac{\partial s_{j3}}{\partial a_{j1}} & \dots \\ \frac{\partial s_{j1}}{\partial a_{j2}} & \frac{\partial s_{j2}}{\partial a_{j2}} & \ddots & \\ \vdots & \vdots & \vdots & \frac{\partial s_{jd}}{\partial a_{jd}} \end{bmatrix}$$

$$= \begin{bmatrix} \sigma'(a_{j1}) & 0 & 0 & 0 \\ 0 & \sigma'(a_{j2}) & 0 & 0 \\ 0 & 0 & \ddots & \\ 0 & 0 & \dots & \sigma'(a_{jd}) \end{bmatrix}$$

Back Propagation through time – Vanishing & Exploding gradient

We will now focus on $\frac{\partial s_t}{\partial s_k}$ and highlight an important problem in training RNN's using BPTT

$$\frac{\partial s_t}{\partial s_k} = \frac{\partial s_t}{\partial s_{t-1}} \frac{\partial s_{t-1}}{\partial s_{t-2}} \cdots \frac{\partial s_{k+1}}{\partial s_k}$$

Let us look at one such term in the product (i.e., $\frac{\partial s_{j+1}}{\partial s_j}$)

$$a_j = Ws_{j-1} + b$$

$$s_j = \sigma(a_j)$$

$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd}]$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]$$

$$\frac{\partial s_j}{\partial a_j} = \begin{bmatrix} \frac{\partial s_{j1}}{\partial a_{j1}} & \frac{\partial s_{j2}}{\partial a_{j1}} & \frac{\partial s_{j3}}{\partial a_{j1}} & \dots \\ \frac{\partial s_{j1}}{\partial a_{j2}} & \frac{\partial s_{j2}}{\partial a_{j2}} & \ddots & \\ \vdots & \vdots & \vdots & \frac{\partial s_{jd}}{\partial a_{jd}} \end{bmatrix}$$

$$= \begin{bmatrix} \sigma'(a_{j1}) & 0 & 0 & 0 \\ 0 & \sigma'(a_{j2}) & 0 & 0 \\ 0 & 0 & \ddots & \\ 0 & 0 & \dots & \sigma'(a_{jd}) \end{bmatrix}$$

$$= diag(\sigma'(a_j))$$

Back Propagation through time – Vanishing & Exploding gradient

We will now focus on $\frac{\partial s_t}{\partial s_k}$ and highlight an important problem in training RNN's using BPTT

$$\frac{\partial s_t}{\partial s_k} = \frac{\partial s_t}{\partial s_{t-1}} \frac{\partial s_{t-1}}{\partial s_{t-2}} \cdots \frac{\partial s_{k+1}}{\partial s_k}$$

Let us look at one such term in the product (i.e., $\frac{\partial s_{j+1}}{\partial s_j}$)

$$a_j = Ws_{j-1} + b$$
$$s_j = \sigma(a_j)$$

$$\frac{\partial s_j}{\partial s_{j-1}} = \frac{\partial s_j}{\partial a_j} \frac{\partial a_j}{\partial s_{j-1}}$$

We are interested in the magnitude of $\frac{\partial s_j}{\partial s_{j-1}} \leftarrow$ if it is small (large) $\frac{\partial s_t}{\partial s_k}$ and hence $\frac{\partial \mathcal{L}_t}{\partial W}$ will vanish (explode)

$$a_j = [a_{j1}, a_{j2}, a_{j3}, \dots, a_{jd}]$$

$$s_j = [\sigma(a_{j1}), \sigma(a_{j2}), \dots, \sigma(a_{jd})]$$

$$\frac{\partial s_j}{\partial a_j} = \begin{bmatrix} \frac{\partial s_{j1}}{\partial a_{j1}} & \frac{\partial s_{j2}}{\partial a_{j1}} & \frac{\partial s_{j3}}{\partial a_{j1}} & \dots \\ \frac{\partial s_{j1}}{\partial a_{j2}} & \frac{\partial s_{j2}}{\partial a_{j2}} & \ddots & \\ \vdots & \vdots & \vdots & \frac{\partial s_{jd}}{\partial a_{jd}} \end{bmatrix}$$

$$= \begin{bmatrix} \sigma'(a_{j1}) & 0 & 0 & 0 \\ 0 & \sigma'(a_{j2}) & 0 & 0 \\ 0 & 0 & \ddots & \\ 0 & 0 & \dots & \sigma'(a_{jd}) \end{bmatrix}$$
$$= diag(\sigma'(a_j))$$

Back Propagation through time – Vanishing & Exploding gradient

$$\left\| \frac{\partial s_j}{\partial s_{j-1}} \right\| = \left\| \text{diag}(\sigma'(a_j))W \right\|$$

$$\leq \left\| \text{diag}(\sigma'(a_j)) \right\| \|W\|$$

$\because \sigma(a_j)$ is a bounded function (sigmoid, tanh) $\sigma'(a_j)$ is bounded

$$\left\| \frac{\partial s_t}{\partial s_k} \right\| = \left\| \prod_{j=k+1}^t \frac{\partial s_j}{\partial s_{j-1}} \right\|$$

$$\leq \prod_{j=k+1}^t \gamma\lambda$$

$$\leq (\gamma\lambda)^{t-k}$$

$$\sigma'(a_j) \leq \frac{1}{4} = \gamma \text{ [if } \sigma \text{ is logistic]}$$

$$\leq 1 = \gamma \text{ [if } \sigma \text{ is tanh]}$$

If $\gamma\lambda < 1$ the gradient will vanish

If $\gamma\lambda > 1$ the gradient could explode

$$\left\| \frac{\partial s_j}{\partial s_{j-1}} \right\| \leq \gamma \|W\|$$

$$\leq \gamma\lambda$$

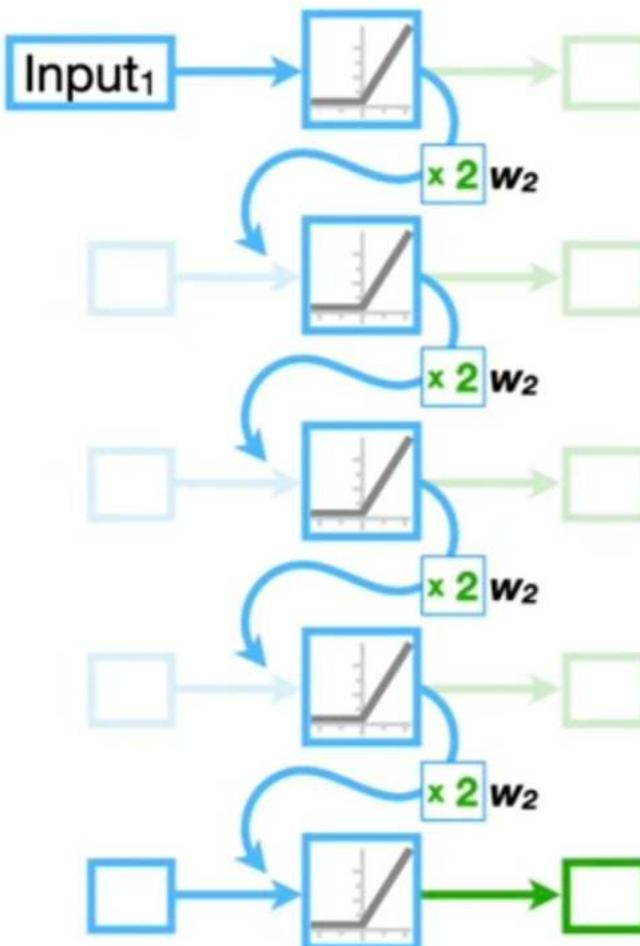
Back Propagation through time – Vanishing & Exploding gradient

input value is amplified
16 times before it gets
to the final copy of the
network.

$$\text{Input}_1 \times 2 \times 2 \times 2 \times 2$$

$$= \text{Input}_1 \times 2^4$$

$$= \text{Input}_1 \times w_2^{\text{Num. Unroll}}$$



A gist of the exploding gradient (same case with vanishing gradient if instead of 2 the value is less than 1)

Back Propagation through time in RNNs : Issues & Solutions

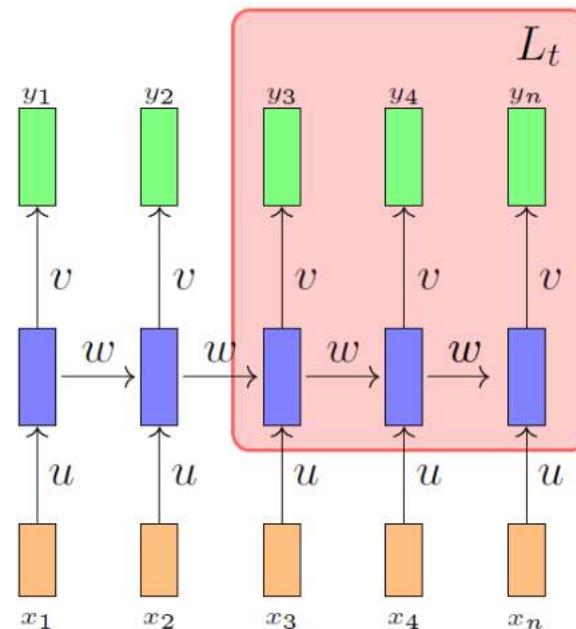
1. Gradient calculations are expensive
(slow training for long sequences)

- Solution: Truncated BPTT

2. Exploding gradients (long sequences)

3. Vanishing gradients (long sequences)

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras



- Instead of looking at all 'n' time steps, we would look at lesser time steps allowing us to estimate rather than calculate the gradient used to update the weights.

Back Propagation through time in RNNs : Issues & Solutions

1. Gradient calculations are expensive
(slow training for long sequences)

- Solution: Truncated BPTT

2. Exploding gradients (long sequences)

- Solution: Gradient Clipping

3. Vanishing gradients (long sequences)

Let $g = \frac{\partial L}{\partial W}$

I. Clipping by value:

if $\|g\| \geq \text{max_threshold}$ then:

$g \leftarrow \text{threshold}$

end if

II. Clipping by norm:

if $\|g\| \geq \text{threshold}$ then:

$g \leftarrow \text{threshold} * g / \|g\|$

end if

Back Propagation through time in RNNs : Issues & Solutions

1. Gradient calculations are expensive
(slow training for long sequences)
 - Solution: Truncated BPTT
2. Exploding gradients (long sequences)
 - Solution: Gradient Clipping
3. Vanishing gradients (long sequences)
 - Solution: Use alternate RNN architectures such as LSTM and GRU.

LSTM & GRU

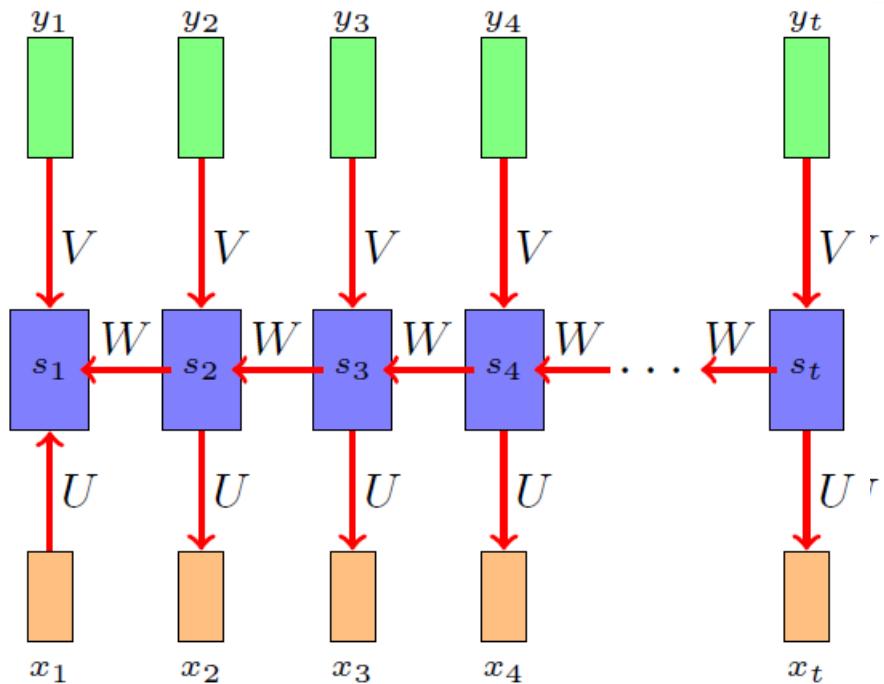
DSE 3151 DEEP LEARNING

Dr. Rohini Rao & Dr. Abhilash K Pai

Dept. of Data Science and Computer Applications

MIT Manipal

LSTM and GRU : Introduction

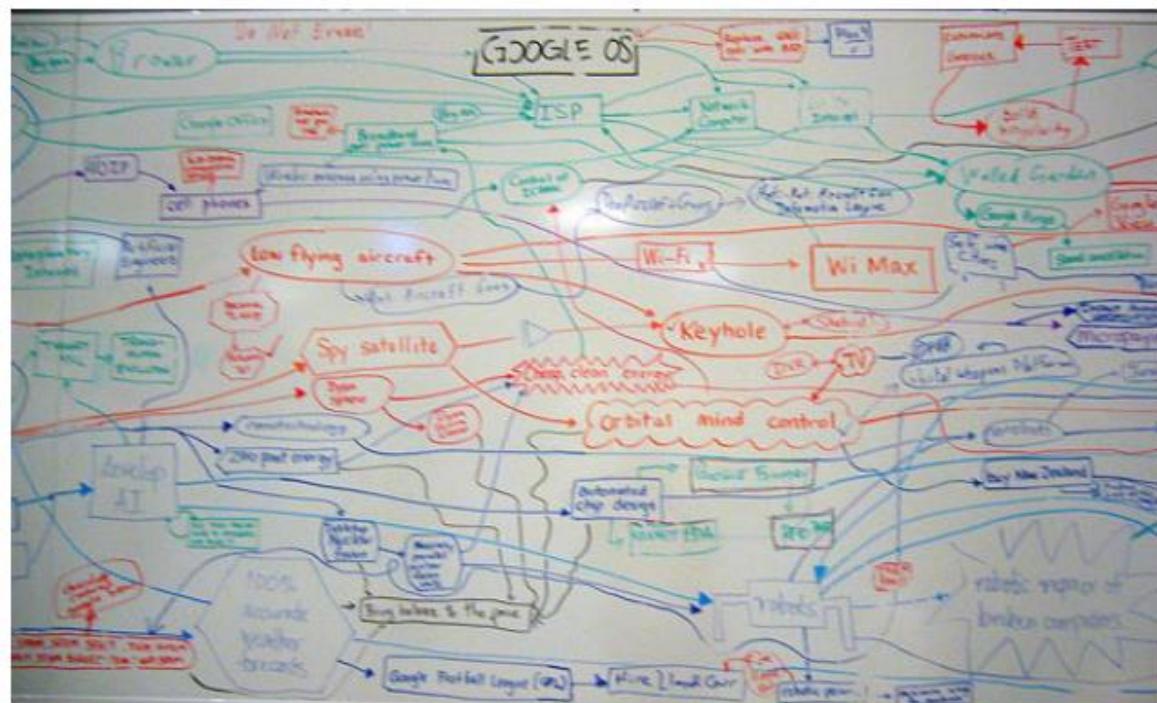


Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

- The state (s_i) of an RNN records information from all previous time steps.
- At each new timestep the old information gets morphed by the current input.
- After ' t ' steps the information stored at time step $t - k$ (for some $k < t$) gets completely morphed.
- It would be impossible to extract the original information stored at time step $t - k$.
- Also, there is the Vanishing gradients problem!

LSTM and GRU : Introduction

The white board analogy:



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

- Consider a scenario where we have to evaluate the expression on a whiteboard:

**Evaluate “ $ac(bd+a) + ad$ ”
given that $a= 1$, $b= 3$, $c= 5$, $d=11$**
 - Normally, the evaluation in white board would look like:

$ac = 5$
 $bd = 33$
 $bd + a = 34$
 $ac(bd + a) = 170$
 $ad = 11$
 $ac(bd + a) + ad = 181$
 - Now, if the white board has space to accommodate only 3 steps, the above evaluation cannot fit in the required space and would lead to loss of information.

LSTM and GRU : Introduction

Evaluate “ $ac(bd+a) + ad$ ”
given that $a= 1$, $b= 3$, $c= 5$, $d=11$

- A solution is to do the following:

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

$$ac(bd + a) = 170$$

$$ad = 11$$

$$ac(bd + a) + ad = 181$$

LSTM and GRU : Introduction

Evaluate “ $ac(bd+a) + ad$ ”

given that $a= 1$, $b= 3$, $c= 5$, $d=11$

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

$$ac(bd + a) = 170$$

$$ad = 11$$

$$ac(bd + a) + ad = 181$$

- A solution is to do the following:

- Selectively write:

$$ac = 5$$

$$bd = 33$$

LSTM and GRU : Introduction

Evaluate “ $ac(bd+a) + ad$ ”

given that $a= 1$, $b= 3$, $c= 5$, $d=11$

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

$$ac(bd + a) = 170$$

$$ad = 11$$

$$ac(bd + a) + ad = 181$$

- A solution is to do the following:

- Selectively write:

$$ac = 5$$

$$bd = 33$$

- Selectively read:

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

LSTM and GRU : Introduction

Evaluate “ $ac(bd+a) + ad$ ”

given that $a= 1$, $b= 3$, $c= 5$, $d=11$

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

$$ac(bd + a) = 170$$

$$ad = 11$$

$$ac(bd + a) + ad = 181$$

- A solution is to do the following:

- Selectively write:

$$ac = 5$$

$$bd = 33$$

- Selectively read:

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

Now the board is full

LSTM and GRU : Introduction

Evaluate “ $ac(bd+a) + ad$ ”

given that $a= 1$, $b= 3$, $c= 5$, $d=11$

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

$$ac(bd + a) = 170$$

$$ad = 11$$

$$ac(bd + a) + ad = 181$$

- A solution is to do the following:

- Selectively write:

$$ac = 5$$

$$bd = 33$$

- Selectively read:

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

Now the board is full

- So, Selectively forget:

$$ac = 5$$

$$bd + a = 34$$

LSTM and GRU : Introduction

Evaluate “ $ac(bd+a) + ad$ ”

given that $a= 1$, $b= 3$, $c= 5$, $d=11$

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

$$ac(bd + a) = 170$$

$$ad = 11$$

$$ac(bd + a) + ad = 181$$

- A solution is to do the following:

- Selectively write:

$$ac = 5$$

$$bd = 33$$

- Selectively read:

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

Now the board is full

- So, Selectively forget:

$$ac = 5$$

$$ac(bd + a) = 170$$

$$bd + a = 34$$

LSTM and GRU : Introduction

Evaluate “ $ac(bd+a) + ad$ ”

given that $a= 1$, $b= 3$, $c= 5$, $d=11$

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

$$ac(bd + a) = 170$$

$$ad = 11$$

$$ac(bd + a) + ad = 181$$

- A solution is to do the following:

- Selectively write:

$$ac = 5$$

$$bd = 33$$

- Selectively read:

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

Now the board is full

- So, Selectively forget:

$$ac = 5$$

$$ac(bd + a) = 170$$

$$ad = 11$$

LSTM and GRU : Introduction

Evaluate “ $ac(bd+a) + ad$ ”

given that $a= 1$, $b= 3$, $c= 5$, $d=11$

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

$$ac(bd + a) = 170$$

$$ad = 11$$

$$ac(bd + a) + ad = 181$$

- A solution is to do the following:

- Selectively write:

$$ac = 5$$

$$bd = 33$$

- Selectively read:

$$ac = 5$$

$$bd = 33$$

$$bd + a = 34$$

Now the board is full

- So, Selectively forget:

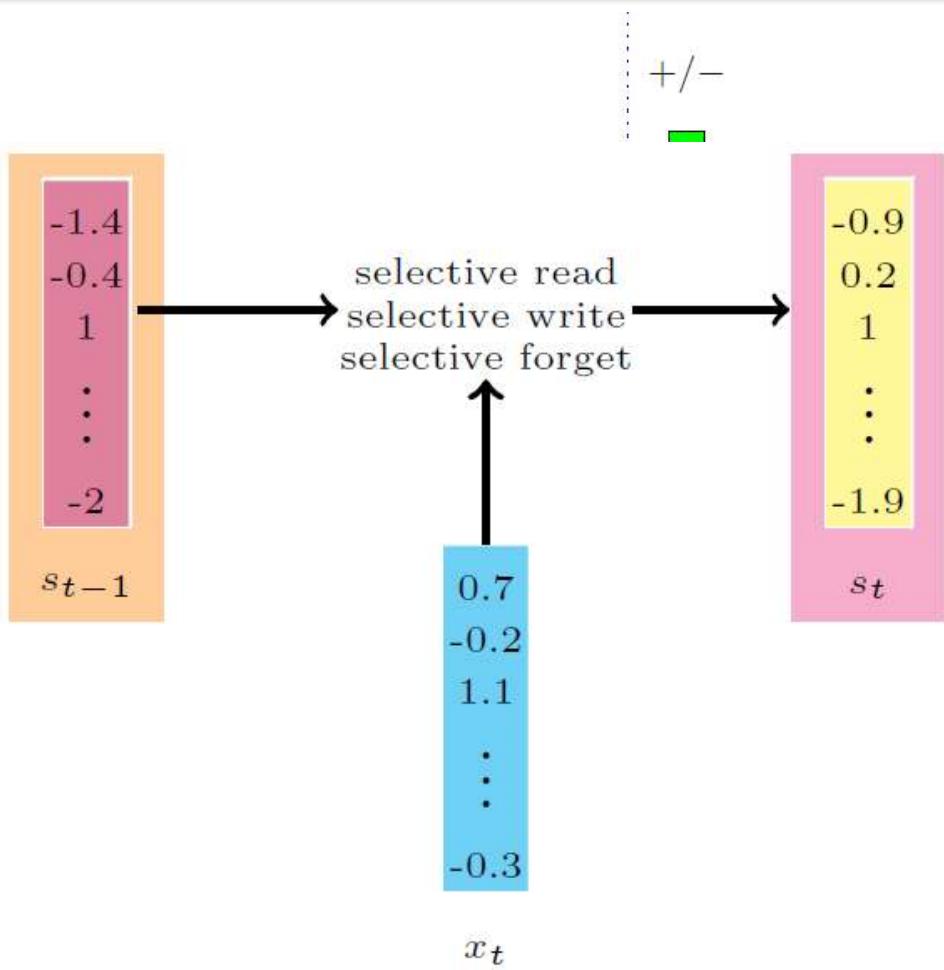
$$ac(bd + a) + ad = 181$$

$$ac(bd + a) = 170$$

$$ad = 11$$

Since the RNN also has a finite state size, we need to figure out a way to allow it to selectively read, write and forget

LSTM and GRU : Introduction

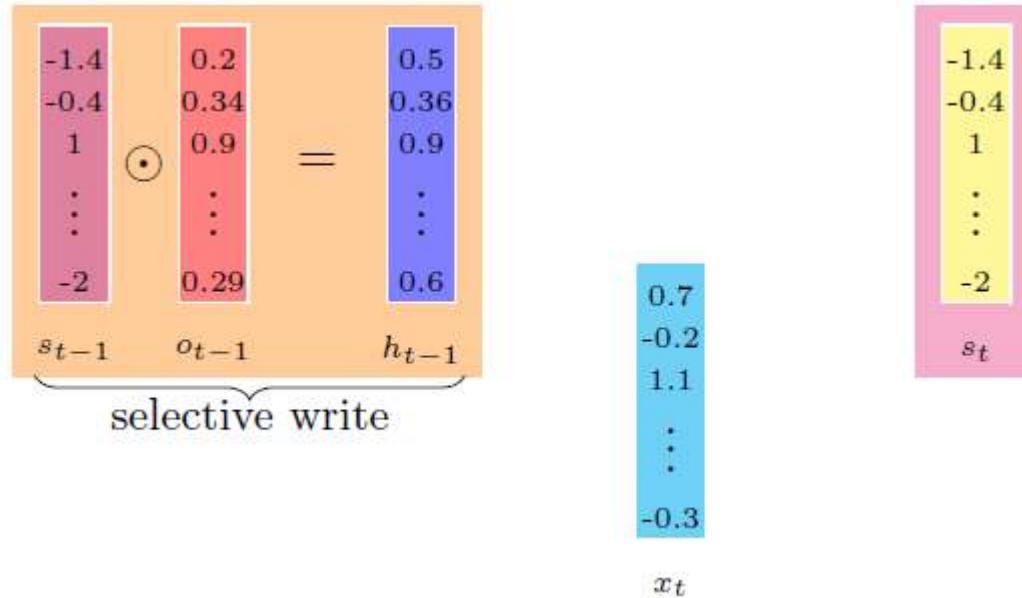


- RNN reads the document from left to right and after every word updates the state.
- By the time we reach the end of the document the information obtained from the first few words is completely lost.
- In our improvised network, ideally, we would like to:
 - **Forget** the information added by stop words (a, the, etc.)
 - **Selectively read** the information added by previous sentiment bearing words (awesome, amazing, etc.)
 - **Selectively write** new information from the current word to the state.

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

LSTM and GRU : Introduction

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras



The RNN has to learn o_{t-1} along with other parameters (W, U, V)

$$o_{t-1} = \sigma(W_o h_{t-2} + U_o x_{t-1} + b_o)$$

$$h_{t-1} = o_{t-1} \odot \sigma(s_{t-1})$$

New parameters to be learned are: W_o , U_o , b_o

O_t is called the output gate as it decides how much to pass (write) to the next time step.

Selectively write:

- In an RNN, the state s_t is defined as follows:

$$s_t = \sigma(W s_{t-1} + U x_t) \text{ (ignoring bias)}$$

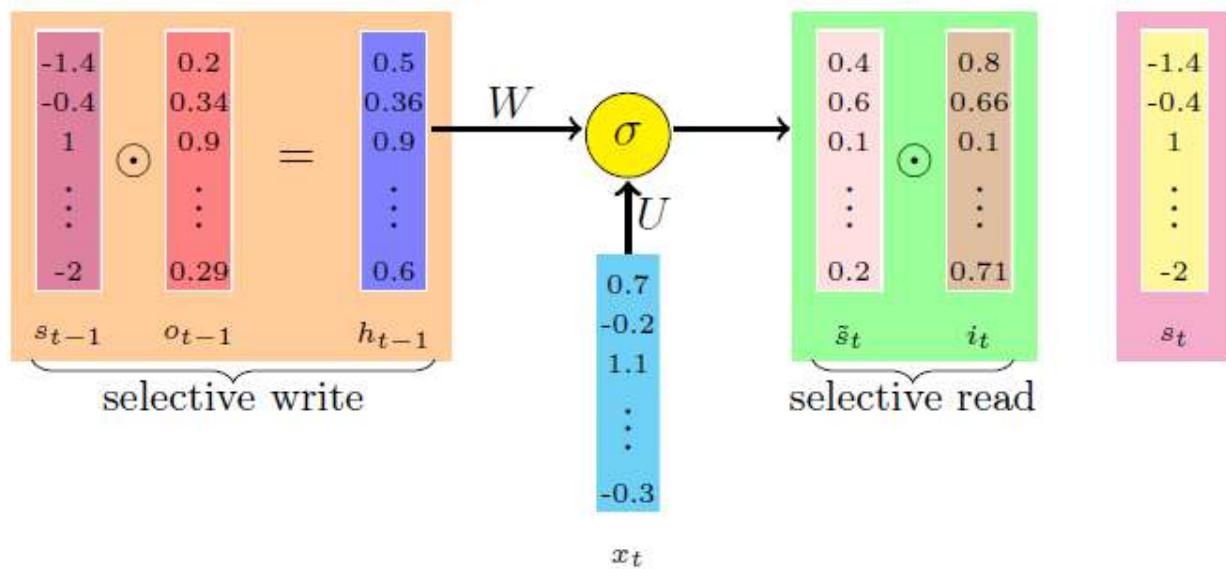
- Instead of passing s_{t-1} as it is, we need to pass (write) only some portions of it.

- To do this, we introduce a vector o_{t-1} which decides what fraction of each element of s_{t-1} should be passed to the next state.

- Each element of o_{t-1} (restricted to be between 0 and 1) gets multiplied with s_{t-1}

- How does RNN know what fraction of the state to pass on?

LSTM and GRU : Introduction



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Selectively read:

- We will now use h_{t-1} and x_t to compute the new state at the time step t :

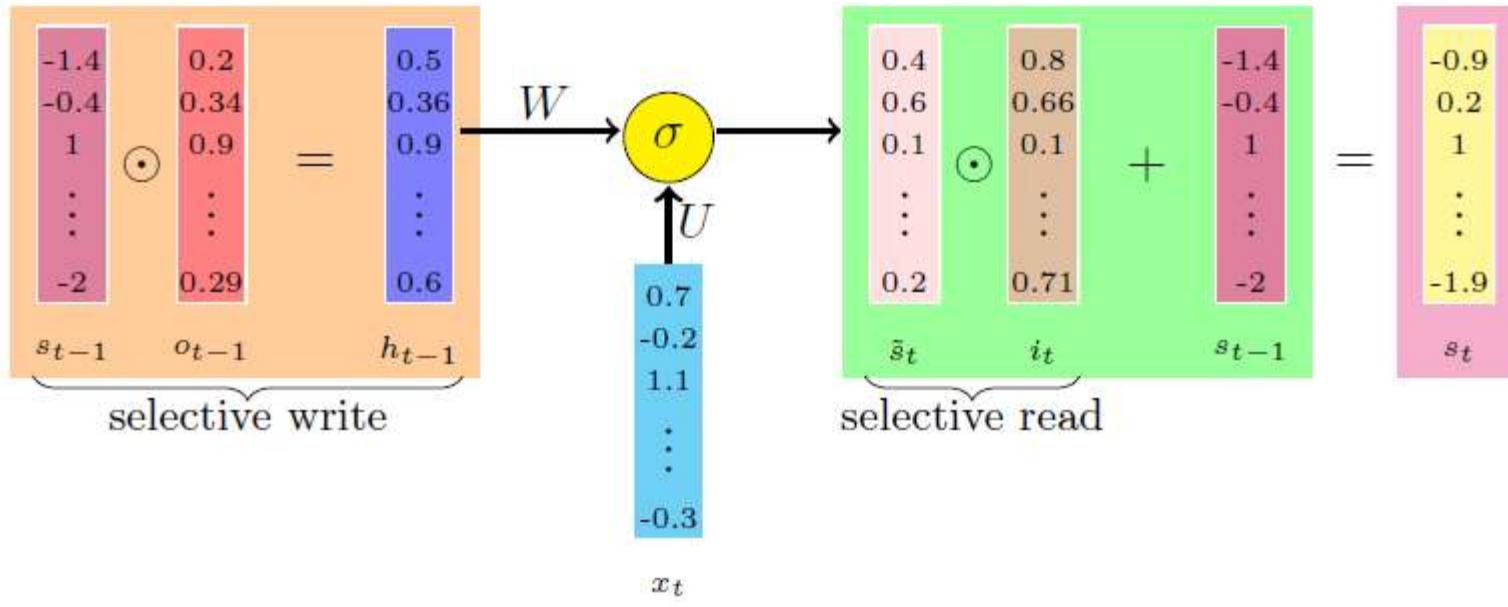
$$\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$$

- Again, to pass only useful information from \tilde{s}_t to s_t , we selectively read from it before constructing the new cell state.
- To do this we introduce another gate called as the **input gate**:

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

- And use $i_t \odot \tilde{s}_t$ to selectively read the information.

LSTM and GRU : Introduction

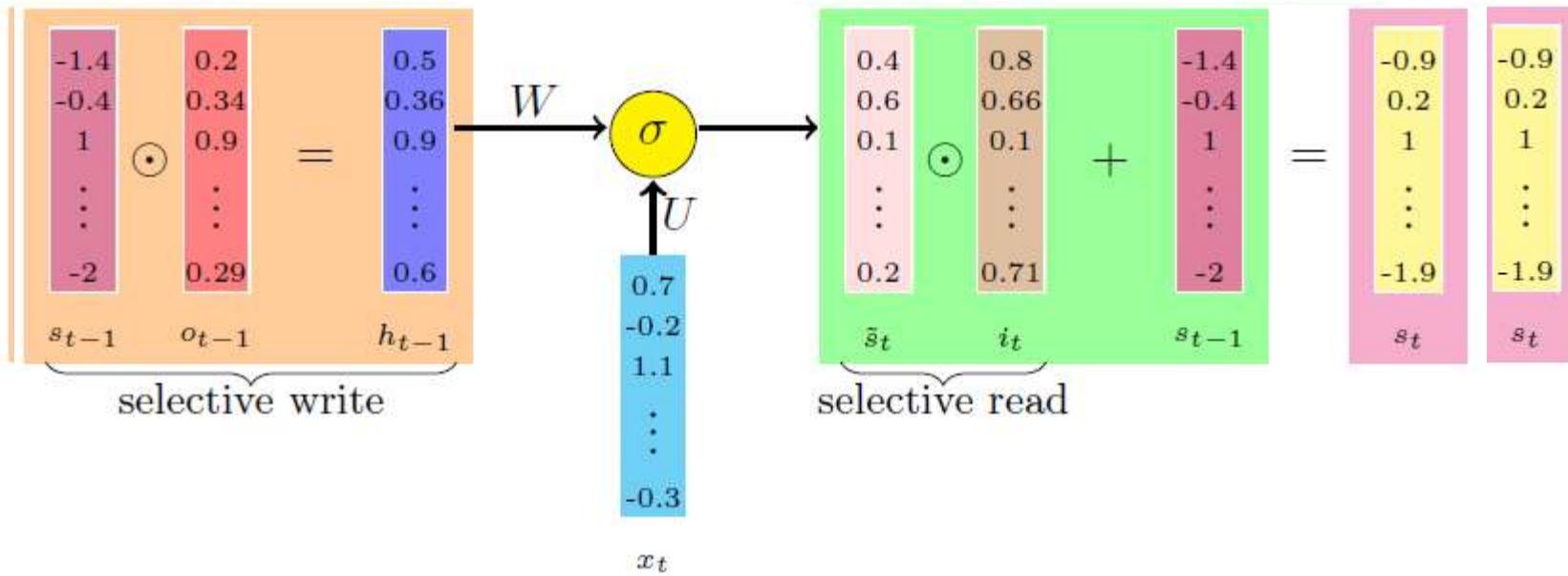


Selectively forget

- How do we combine s_{t-1} and \tilde{s}_t to get the new state?

$$s_t = s_{t-1} + i_t \odot \tilde{s}_t$$

LSTM and GRU : Introduction



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Selectively forget

- How do we combine s_{t-1} and \tilde{s}_t to get the new state?

$$s_t = s_{t-1} + i_t \odot \tilde{s}_t$$

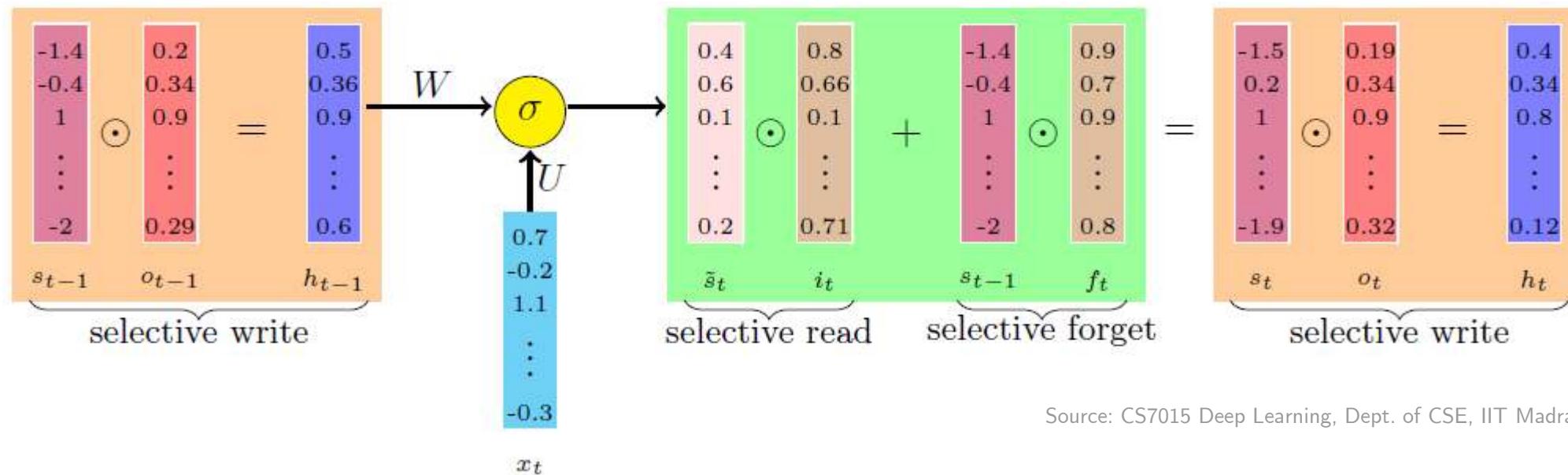
↓

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

- But we may not want to use the whole of s_{t-1} but forget some parts of it.
- To do this a **forget gate** is introduced:

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

LSTM (Long Short-Term Memory)



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Gates:

$$o_t = \sigma(W_o h_{t-1} + U_o x_t + b_o)$$

$$i_t = \sigma(W_i h_{t-1} + U_i x_t + b_i)$$

$$f_t = \sigma(W_f h_{t-1} + U_f x_t + b_f)$$

States:

Long-term memory

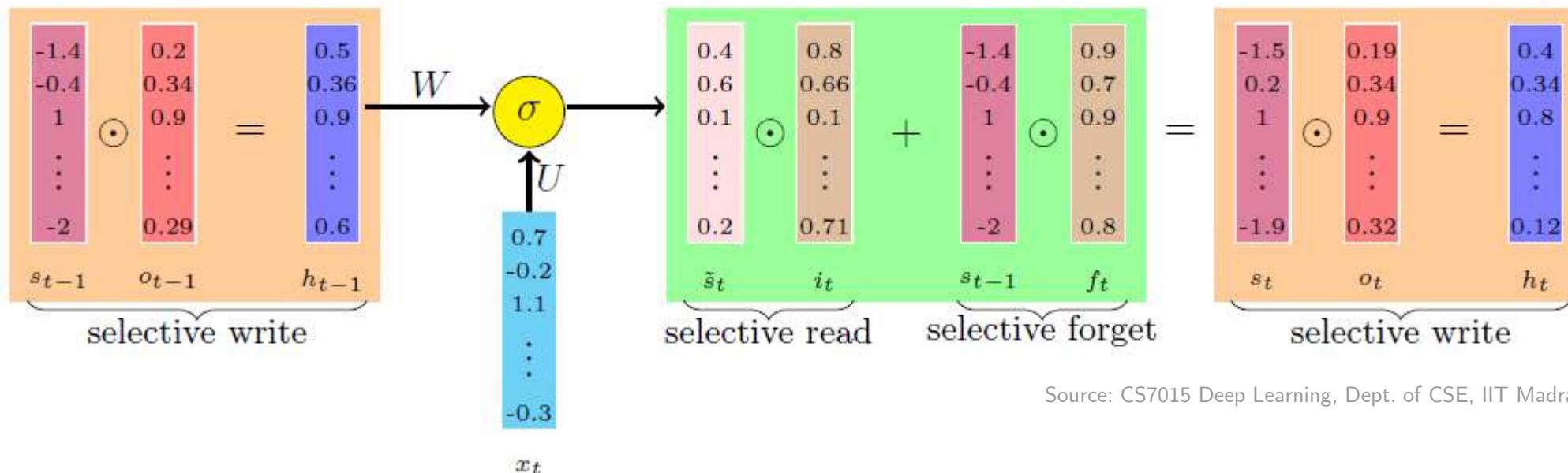
$$\tilde{s}_t = \sigma(W h_{t-1} + U x_t + b)$$

$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

Short-term memory

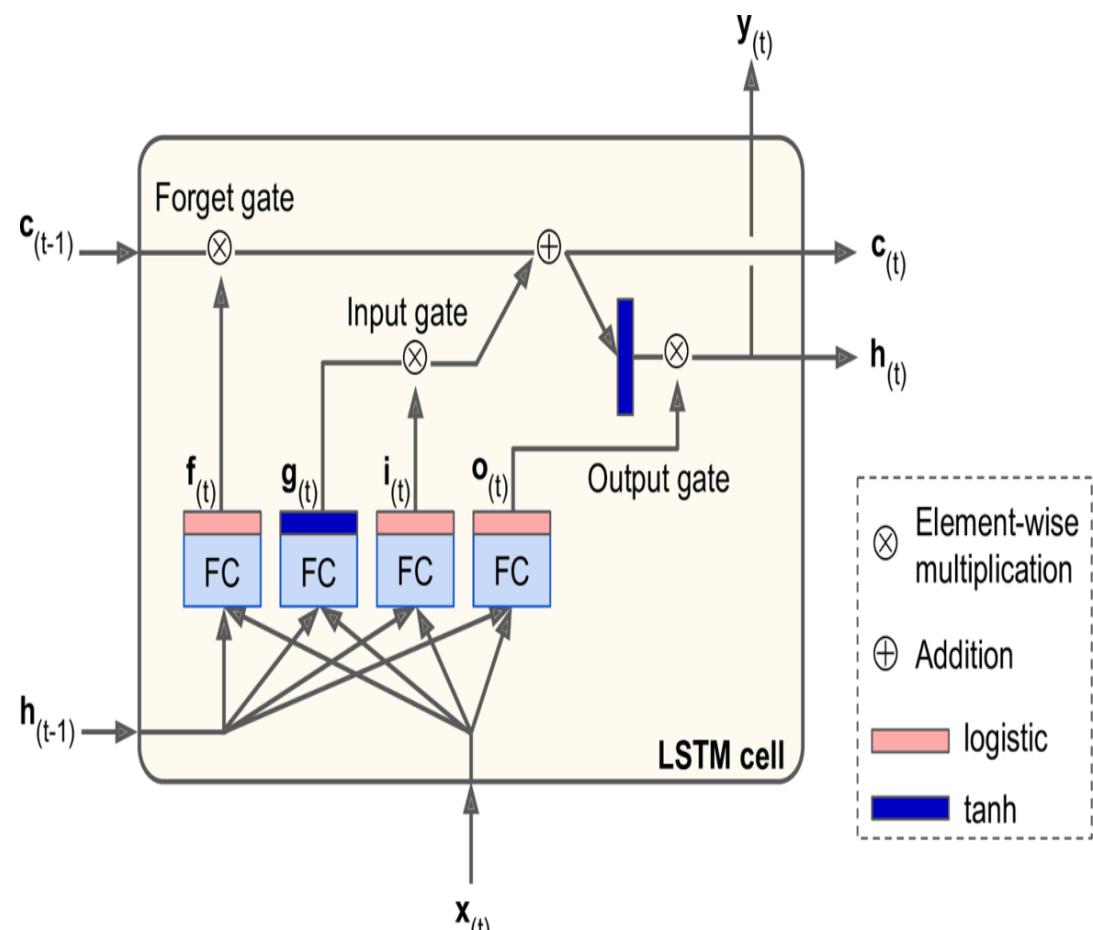
$$h_t = o_t \odot \sigma(s_t) \text{ and } rnn_{out} = h_t$$

LSTM (Long Short-Term Memory)



- LSTM has many variants which include different number of gates and also different arrangement of gates.
- A popular variant of LSTM is the Gated Recurrent Unit (GRU).

LSTM Cell

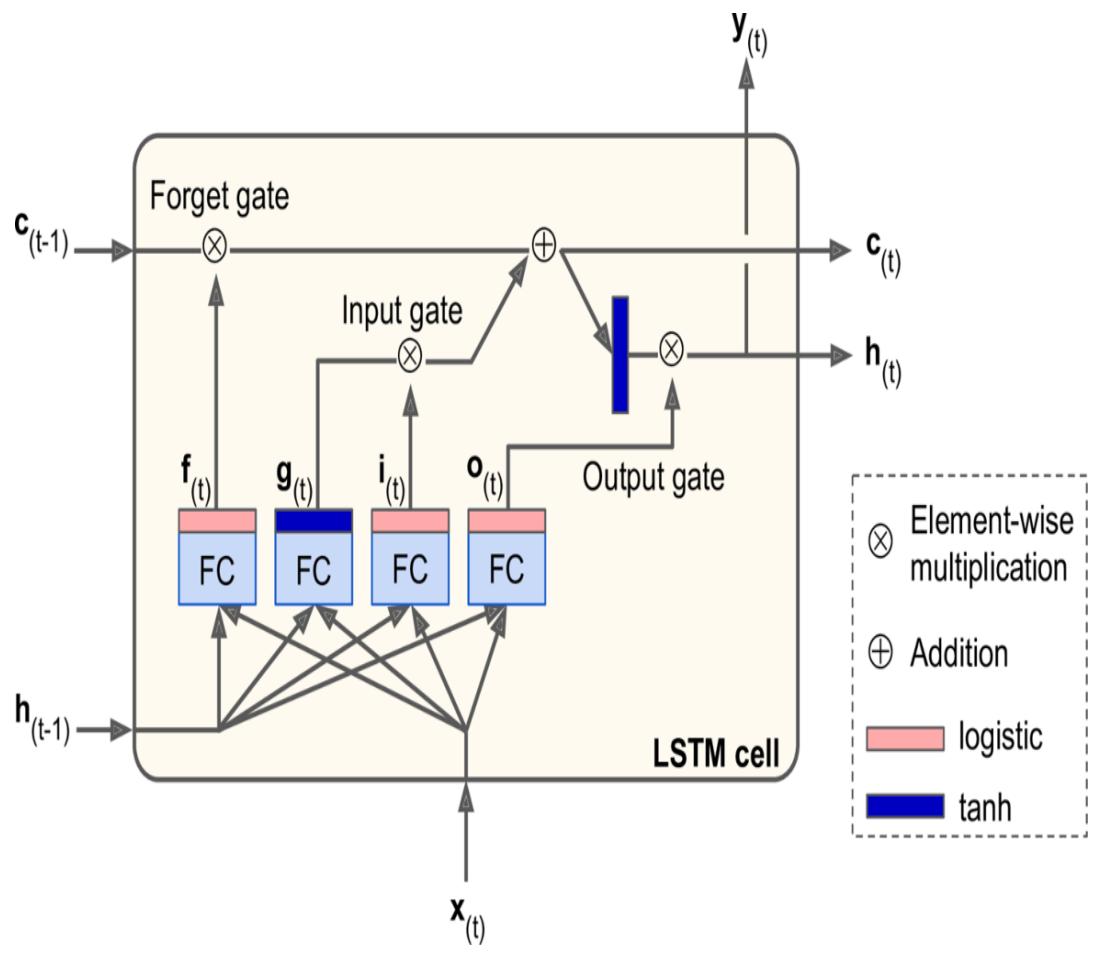


- Note: c_t is same as s_t

- Neuron called a Cell
- FC are fully connected layers
- Long Term state $c_{(t-1)}$ traverses through forget gate forgetting some memories and adding some new memories
- Long term state $c_{(t-1)}$ is passed through tanh and then filtered by an output gate, which produces short term state $h_{(t)}$
- Update gate- $g_{(t)}$ takes current input $x_{(t)}$ and previous short term state $h_{(t-1)}$
- Important parts of output $g_{(t)}$ goes to long term state

Source: Aurélien Géron. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.", 2019.

LSTM Cell



- **Gating Mechanism-** regulates information that the network stores
- Other 3 layers are gate controllers
- **Forget gate $f_{(t)}$** controls which part of the long—term state should be erased
 - **Input gate $i_{(t)}$** controls which part of $g_{(t)}$ should be added to long term state
 - **Output gate $o_{(t)}$** controls which parts of long term state should be read and output at this time state
 - both to $h_{(t)}$ and to $y_{(t)}$

Source: Aurélien Géron. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.", 2019.

LSTM computations

$$\begin{aligned}\mathbf{i}_{(t)} &= \sigma(\mathbf{W}_{xi}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i) \\ \mathbf{f}_{(t)} &= \sigma(\mathbf{W}_{xf}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f) \\ \mathbf{o}_{(t)} &= \sigma(\mathbf{W}_{xo}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o) \\ \mathbf{g}_{(t)} &= \tanh (\mathbf{W}_{xg}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g) \\ \mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\ \mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh (\mathbf{c}_{(t)})\end{aligned}$$

An LSTM cell can learn to recognize an important input (role of the input gate), store it in long term state, preserve it for as long as possible it is needed (role of forget gate), and extract it whenever it is needed.

\mathbf{W}_{xi} , \mathbf{W}_{xf} , \mathbf{W}_{xo} , \mathbf{W}_{xg} are the weight matrices of each of the four layers for their connection to the input vector $\mathbf{x}_{(t)}$.

\mathbf{W}_{hi} , \mathbf{W}_{hf} , \mathbf{W}_{ho} , and \mathbf{W}_{hg} are the weight matrices of each of the four layers for their connection to the previous short-term state $\mathbf{h}_{(t-1)}$.

\mathbf{b}_i , \mathbf{b}_f , \mathbf{b}_o , and \mathbf{b}_g are the bias terms for each of the four layers.

Source: Aurélien Géron. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.", 2019.

Gated Recurrent Unit (GRU)

-1.4
-0.4
1
⋮
-2

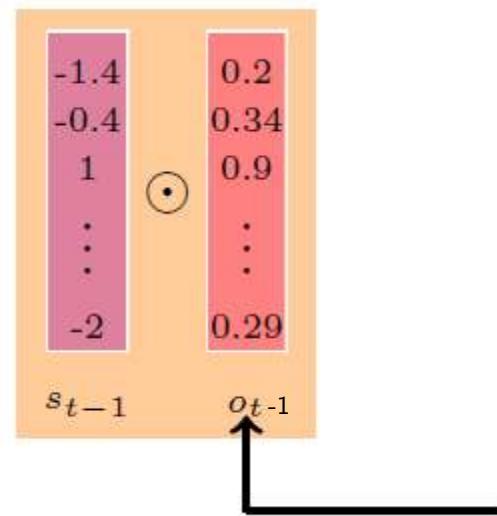
s_{t-1}

Gates:

States:

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Gated Recurrent Unit (GRU)



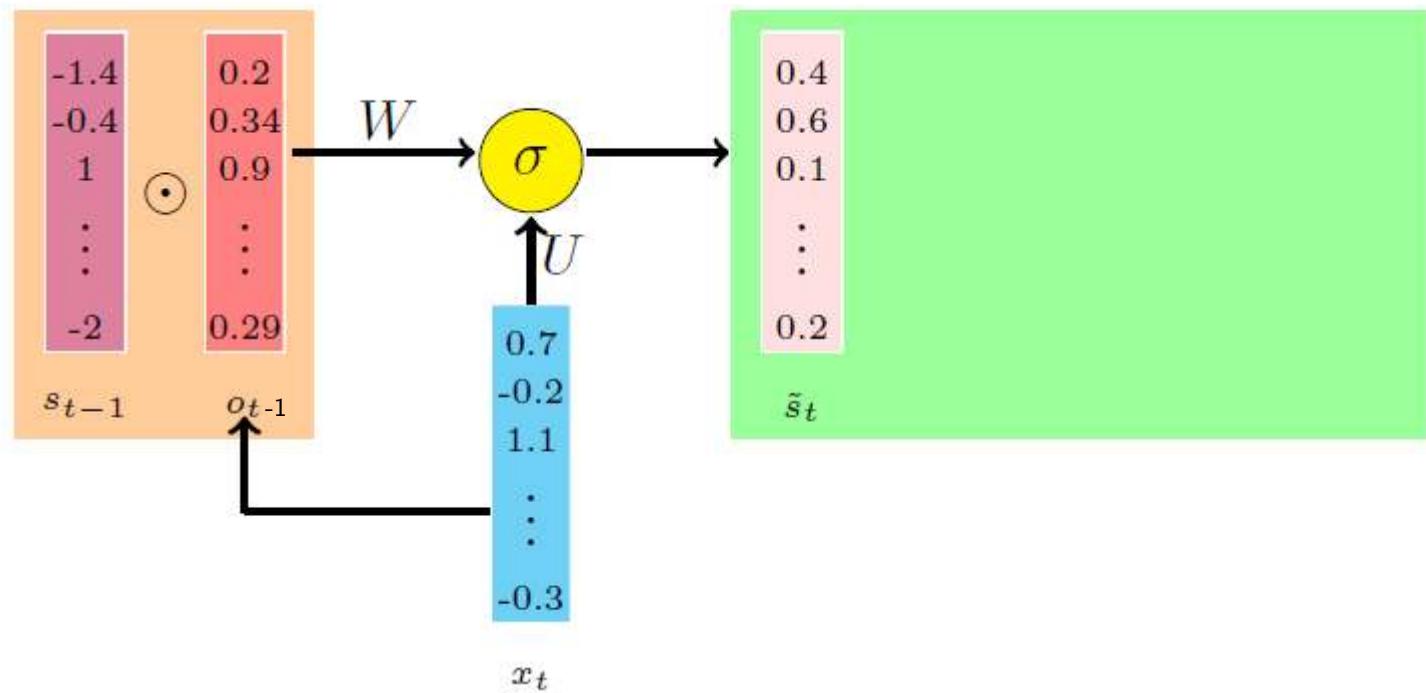
Gates:

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$

States:

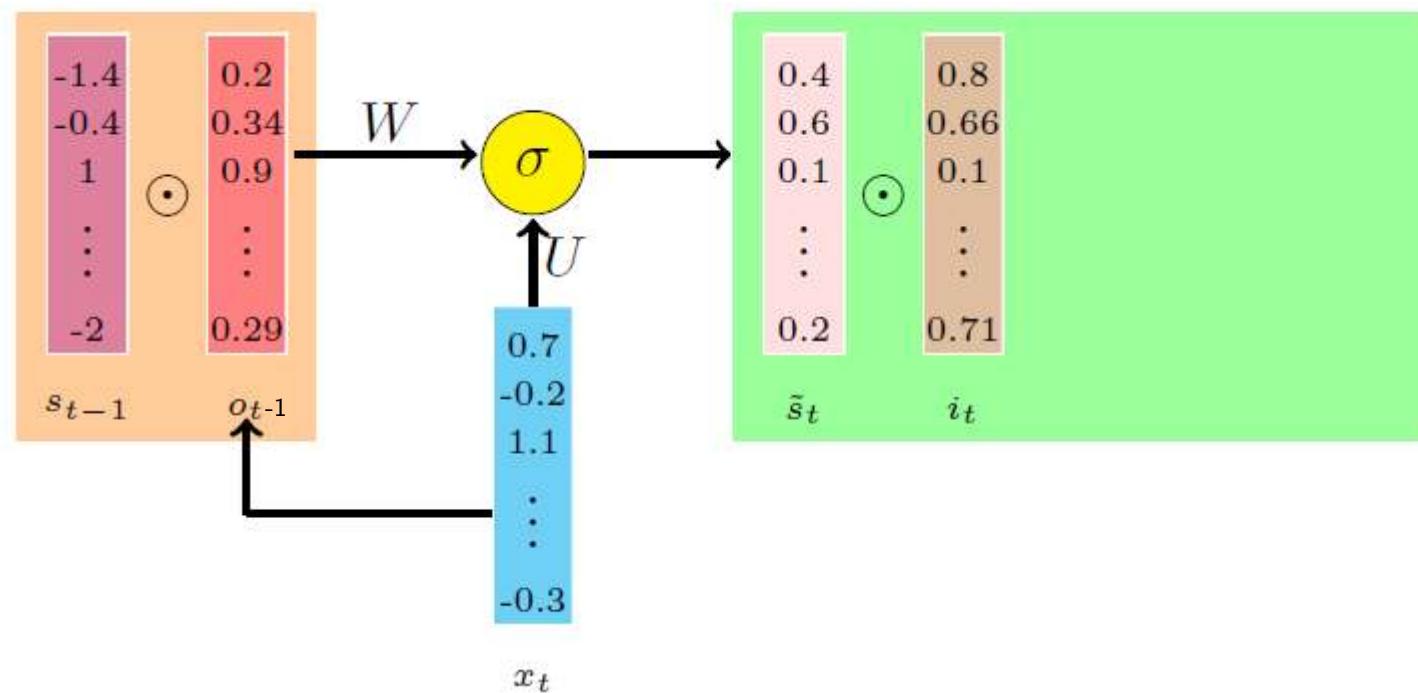
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Gated Recurrent Unit (GRU)



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Gated Recurrent Unit (GRU)



Gates:

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$

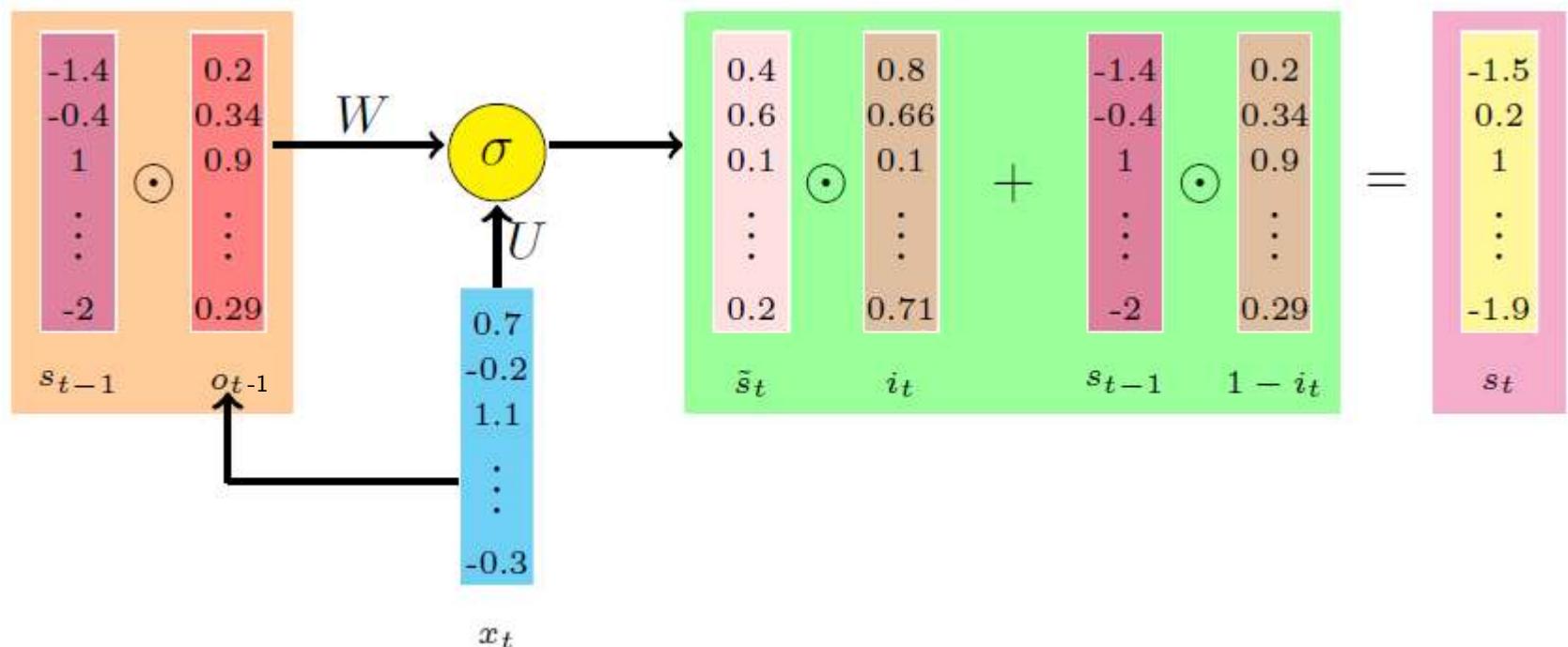
$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

States:

$$\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$$

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Gated Recurrent Unit (GRU)



Gates:

$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$

$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

States:

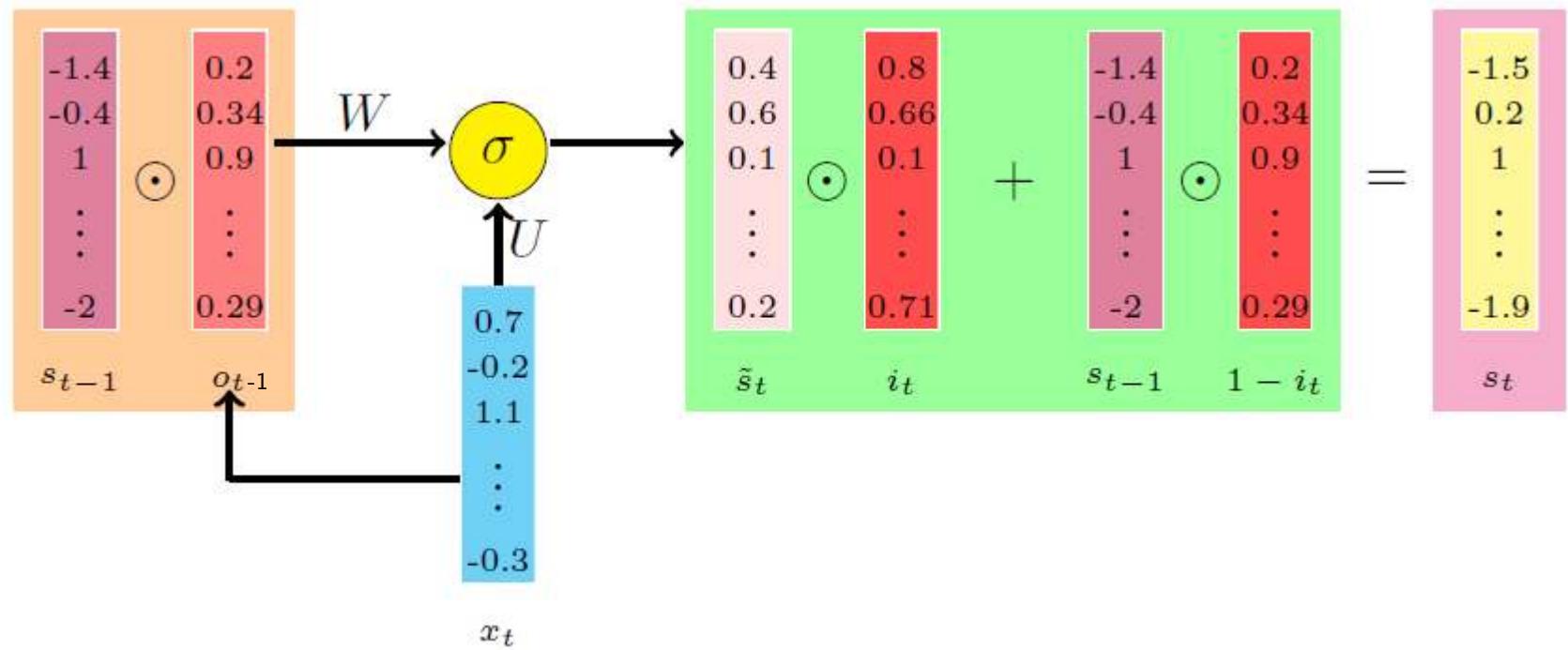
$$\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$$

$$s_t = (1 - i_t) \odot s_{t-1} + i_t \odot \tilde{s}_t$$

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Gated Recurrent Unit (GRU)

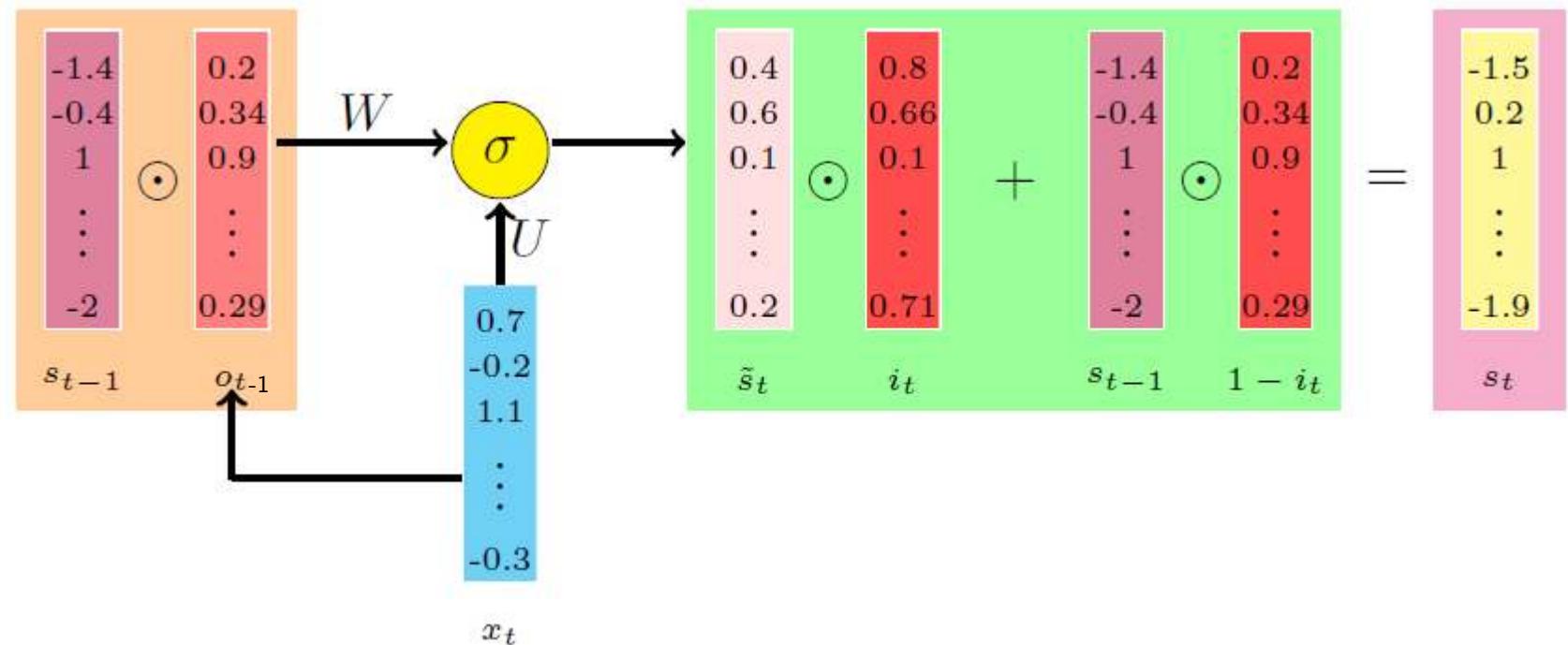
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras



No explicit forget gate (the forget gate and input gates are tied)

Gated Recurrent Unit (GRU)

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras



$$o_t = \sigma(W_o s_{t-1} + U_o x_t + b_o)$$

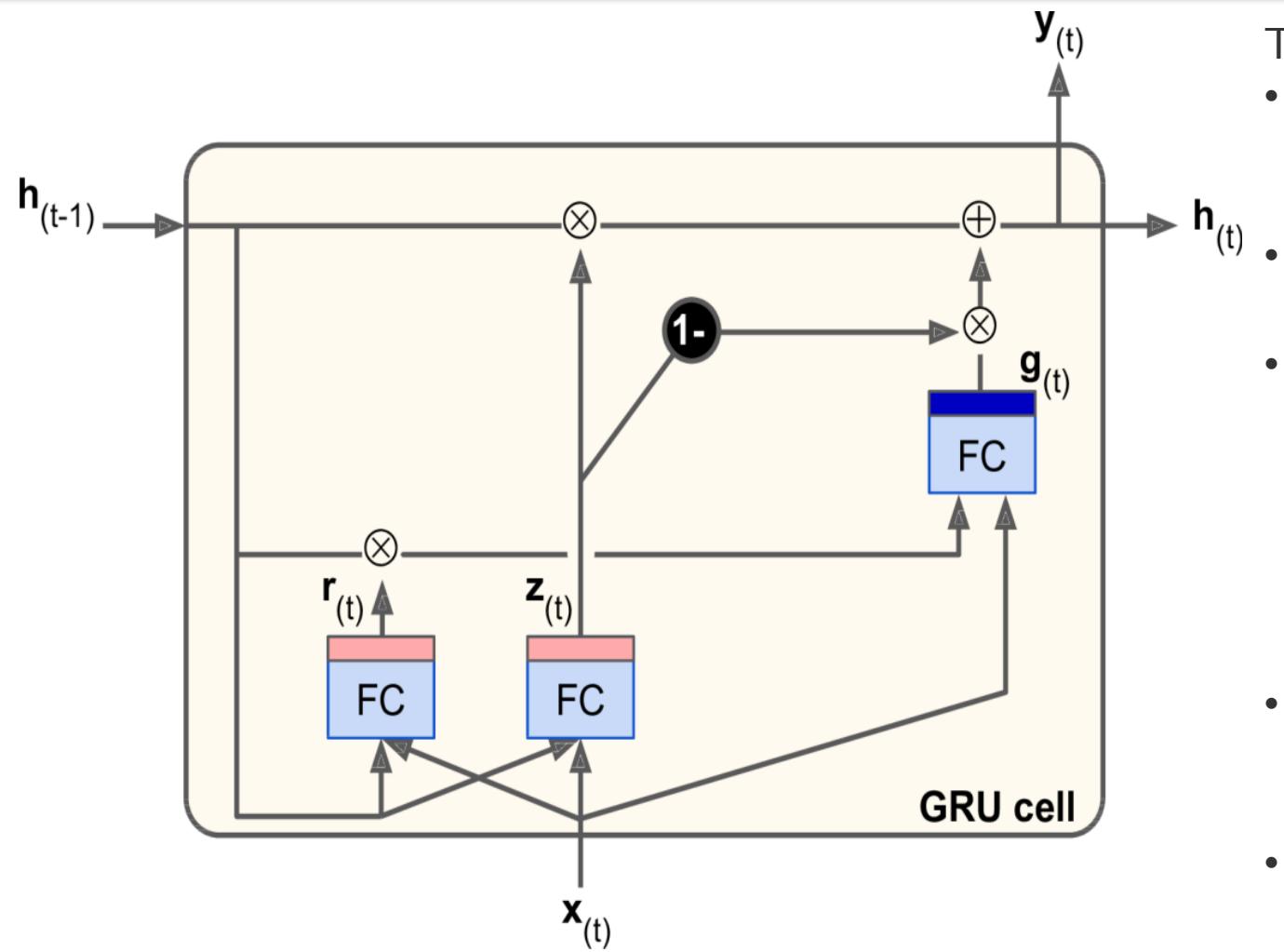
$$i_t = \sigma(W_i s_{t-1} + U_i x_t + b_i)$$

$$\tilde{s}_t = \sigma(W(o_t \odot s_{t-1}) + U x_t + b)$$

$$s_t = (1 - i_t) \odot s_{t-1} + i_t \odot \tilde{s}_t$$

The gates depend directly on s_{t-1} and not the intermediate h_{t-1} as in the case of LSTMs

Gated Recurrent Unit CELL (Kyunghyun Cho et al, 2014)



The main simplifications of LSTM are:

- Both state vectors (short and long term) are merged into a single vector $h_{(t)}$.
- **Gate controller** $z_{(t)}$ controller controls both the forget gate and the input gate.
- If the gate controller outputs
 - 1, the forget gate is open and the input gate is closed.
 - 0, the opposite happens
 - whenever a memory must be written, the location where it will be stored is erased first.
- No output gate, the full state vector is output at every time step.
- **Reset gate controller** $r_{(t)}$ that controls which part of the previous state will be shown to the main layer $g_{(t)}$.

Source: Aurélien Géron. Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: Concepts, tools, and techniques to build intelligent systems. " O'Reilly Media, Inc.", 2019.

LSTM vs GRU computation

$$\begin{aligned}\mathbf{i}_{(t)} &= \sigma(\mathbf{W}_{xi}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hi}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_i) \\ \mathbf{f}_{(t)} &= \sigma(\mathbf{W}_{xf}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hf}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_f) \\ \mathbf{o}_{(t)} &= \sigma(\mathbf{W}_{xo}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{ho}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_o) \\ \mathbf{g}_{(t)} &= \tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_g) \\ \mathbf{c}_{(t)} &= \mathbf{f}_{(t)} \otimes \mathbf{c}_{(t-1)} + \mathbf{i}_{(t)} \otimes \mathbf{g}_{(t)} \\ \mathbf{y}_{(t)} &= \mathbf{h}_{(t)} = \mathbf{o}_{(t)} \otimes \tanh(\mathbf{c}_{(t)})\end{aligned}$$

$$\begin{aligned}\mathbf{z}_{(t)} &= \sigma(\mathbf{W}_{xz}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hz}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_z) \\ \mathbf{r}_{(t)} &= \sigma(\mathbf{W}_{xr}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hr}^T \cdot \mathbf{h}_{(t-1)} + \mathbf{b}_r) \\ \mathbf{g}_{(t)} &= \tanh(\mathbf{W}_{xg}^T \cdot \mathbf{x}_{(t)} + \mathbf{W}_{hg}^T \cdot (\mathbf{r}_{(t)} \otimes \mathbf{h}_{(t-1)}) + \mathbf{b}_g) \\ \mathbf{h}_{(t)} &= \mathbf{z}_{(t)} \otimes \mathbf{h}_{(t-1)} + (1 - \mathbf{z}_{(t)}) \otimes \mathbf{g}_{(t)}\end{aligned}$$

- GRU Performance is good but may have a slight dip in the accuracy
- But lesser number of trainable parameters which makes it advantageous to use

Avoiding vanishing gradients with LSTMs: Intuition

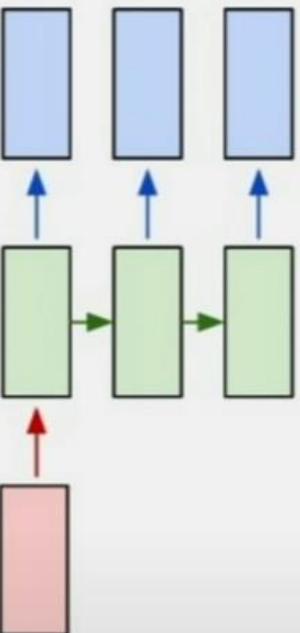
- During forward propagation the gates control the flow of information.
- They prevent any irrelevant information from being written to the state.
- Similarly during backward propagation they control the flow of gradients.
- It is easy to see that during backward pass the gradients will get multiplied by the gate.
- If the state at time $t-1$ did not contribute much to the state at time t then during backpropagation the gradients flowing into s_{t-1} will vanish $\|f_t\| \rightarrow 0$ and $\|o_{t-1}\| \rightarrow 0$
- The key difference from vanilla RNNs is that the flow of information and gradients is controlled by the gates which ensure that the gradients vanish only when they should.

Different RNNs

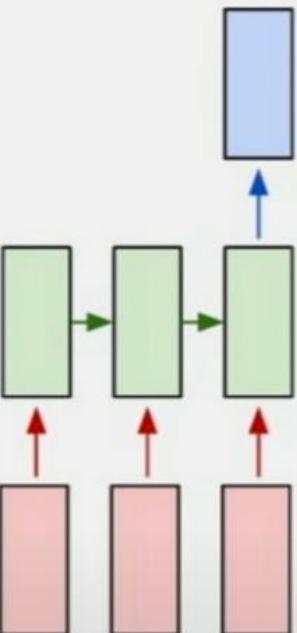
one to one



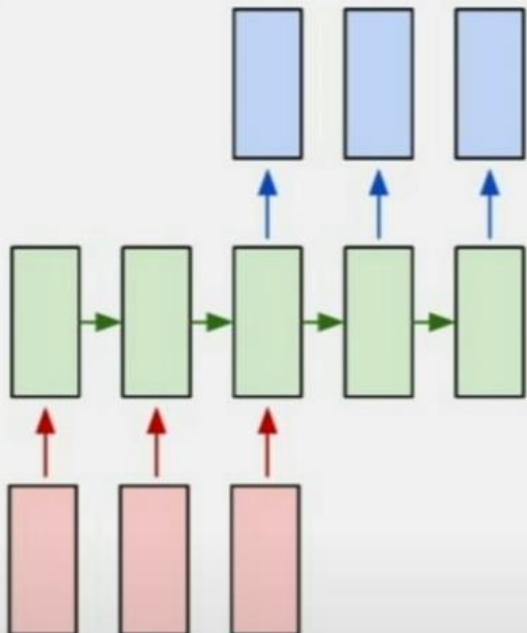
one to many



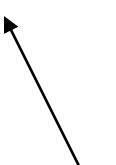
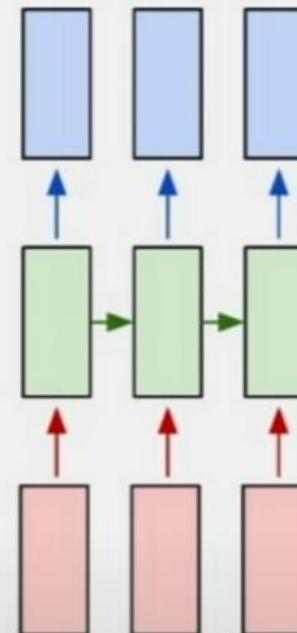
many to one



many to many

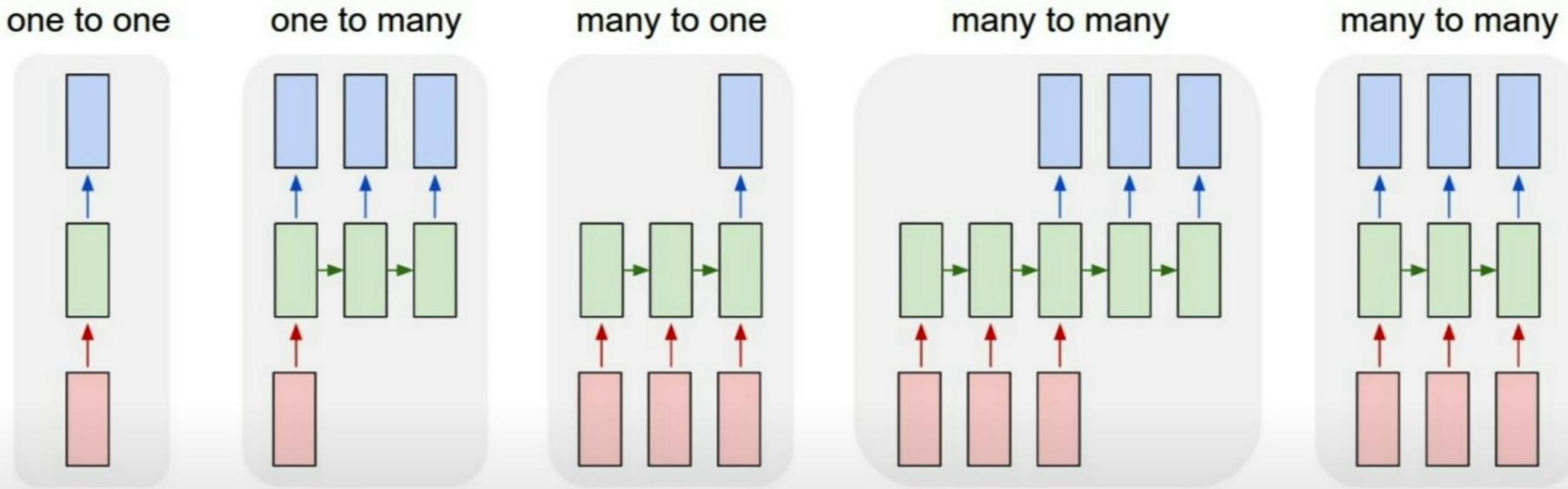


many to many



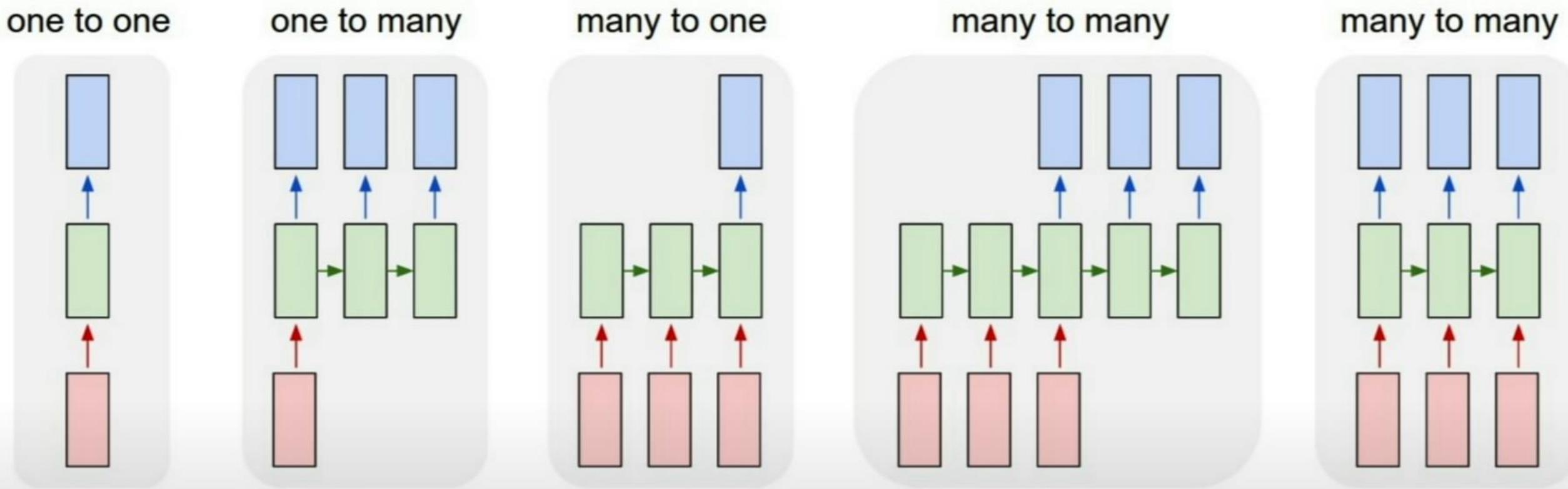
Vanilla RNNs

Different RNNs



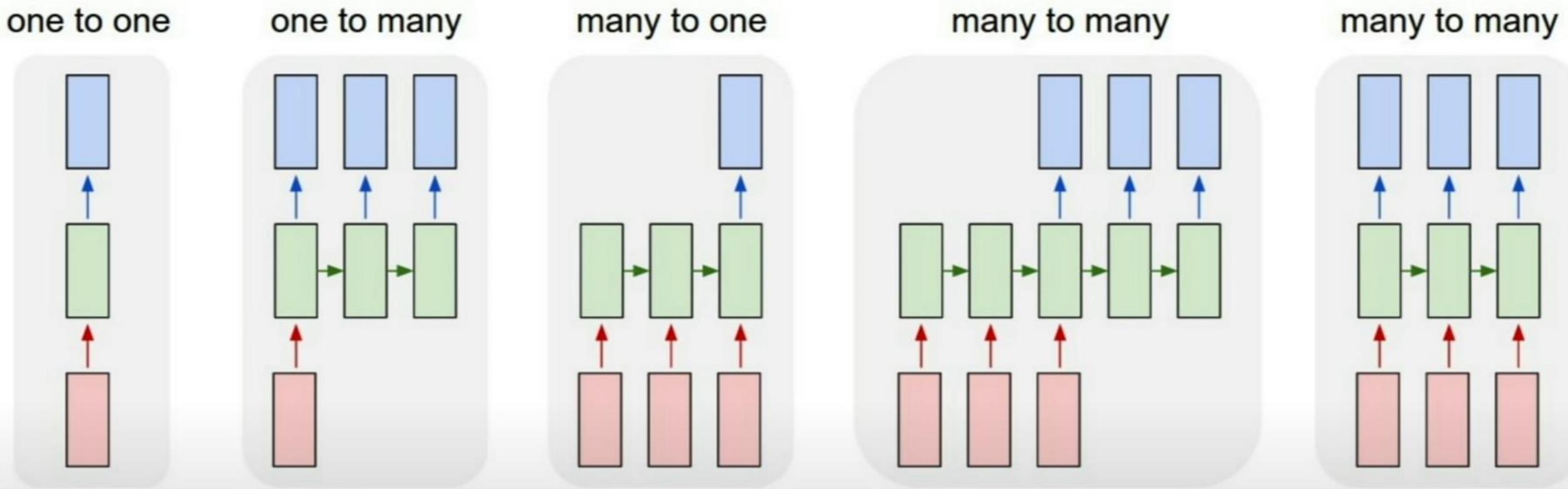
Eg: Image Captioning
Image -> Sequence of words

Different RNNs



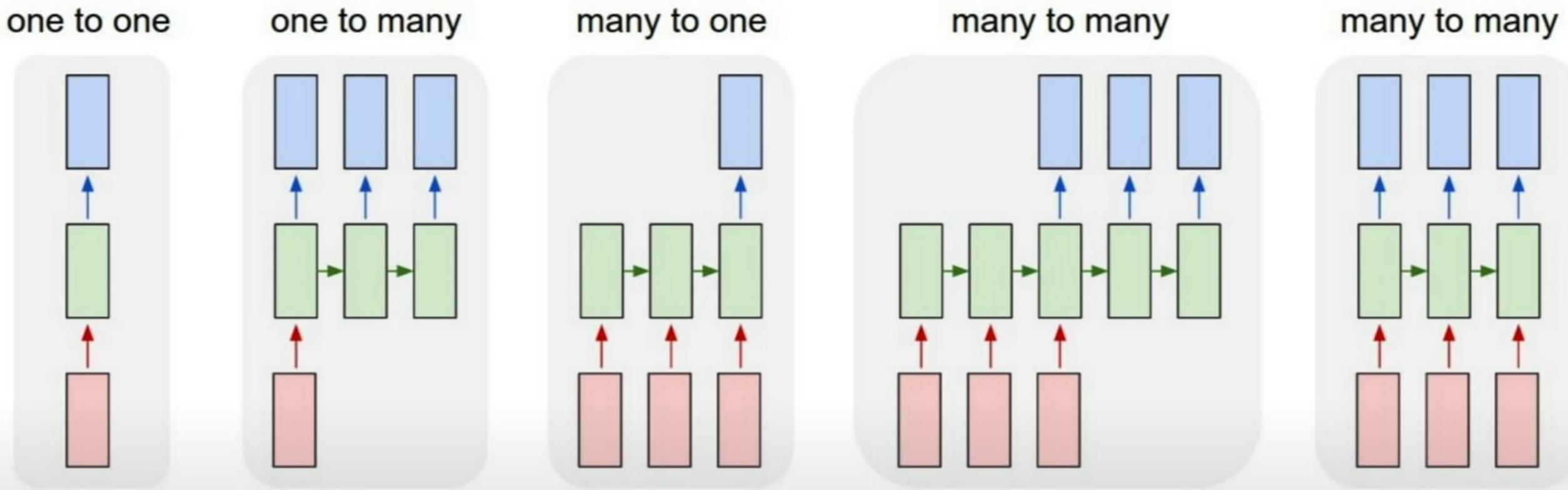
Eg: Sentiment classification
Sequence of words -> Sentiment

Different RNNs



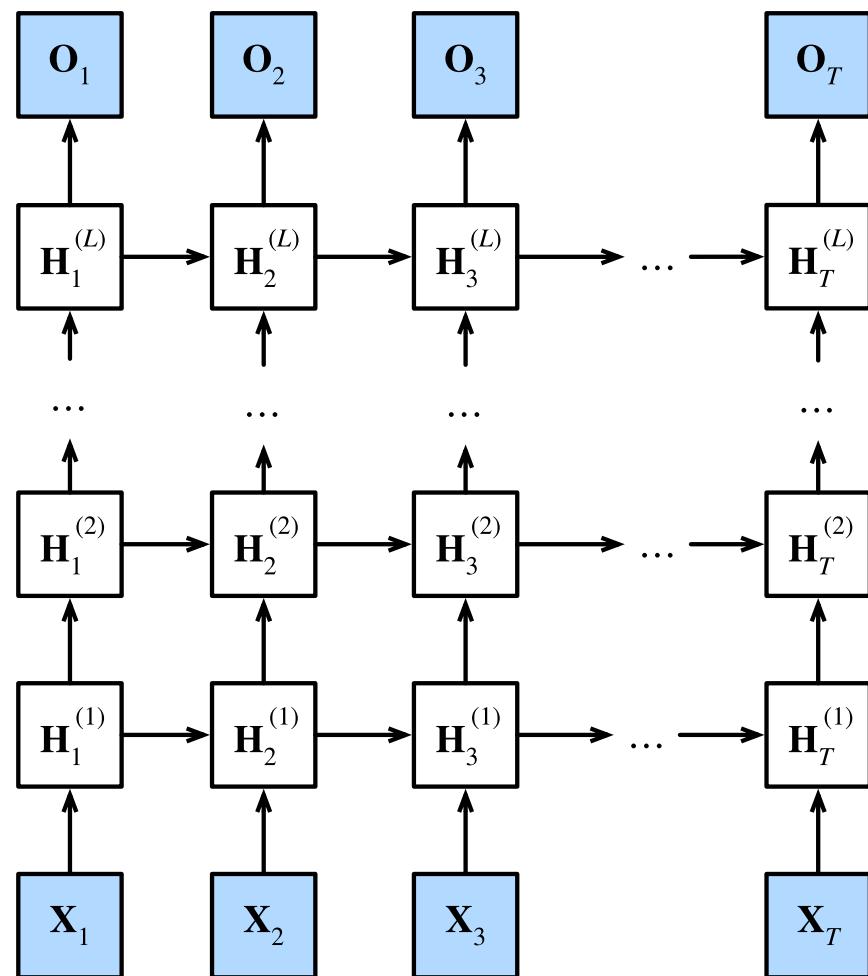
Eg: Machine Translation
Sequence of words -> Sequence of words

Different RNNs



Eg: Video Classification on frame level

Deep RNNs



RNNs that are deep not only in the time direction but also in the input-to-output direction.

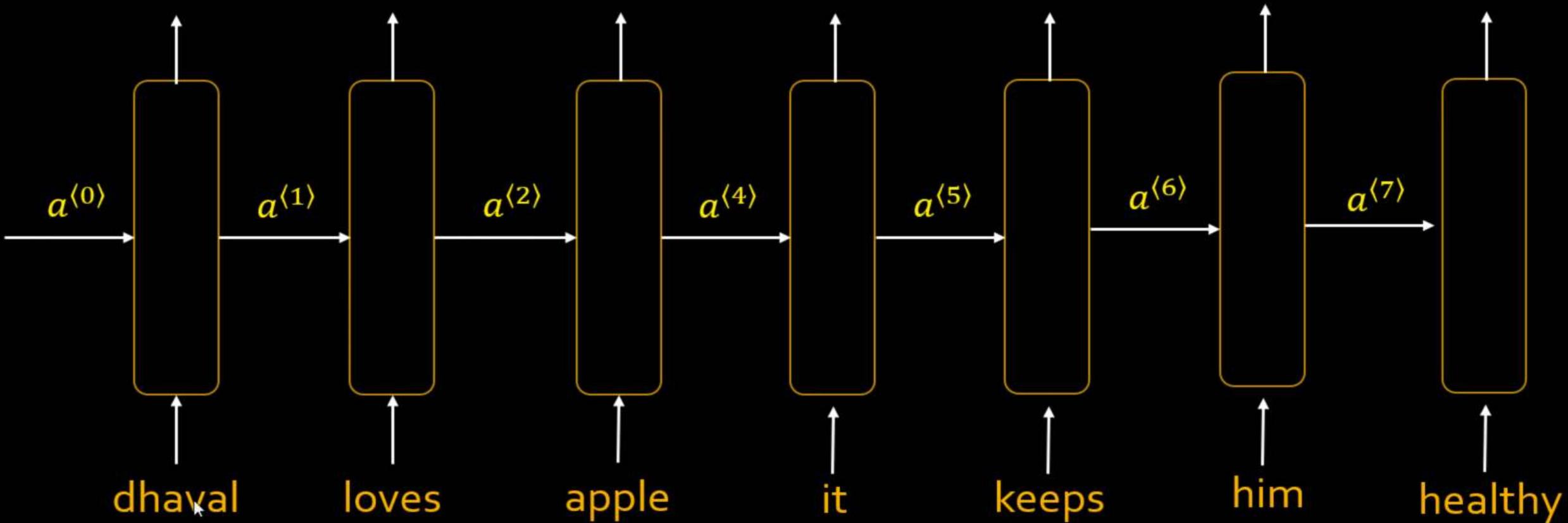
$$\mathbf{H}_t^{(l)} = \phi_l(\mathbf{H}_t^{(l-1)} \mathbf{W}_{xh} + \mathbf{H}_{t-1}^{(l)} \mathbf{W}_{hh} + \mathbf{b}_h^{(l)})$$

$$\mathbf{O}_t = \mathbf{H}_t^{(L)} \mathbf{W}_{hq} + \mathbf{b}_q$$

Source: Deep Recurrent Neural Networks — Dive into Deep Learning 1.0.0-alpha1.post0 documentation (d2l.ai)

Bi-Directional RNNs: Intuition

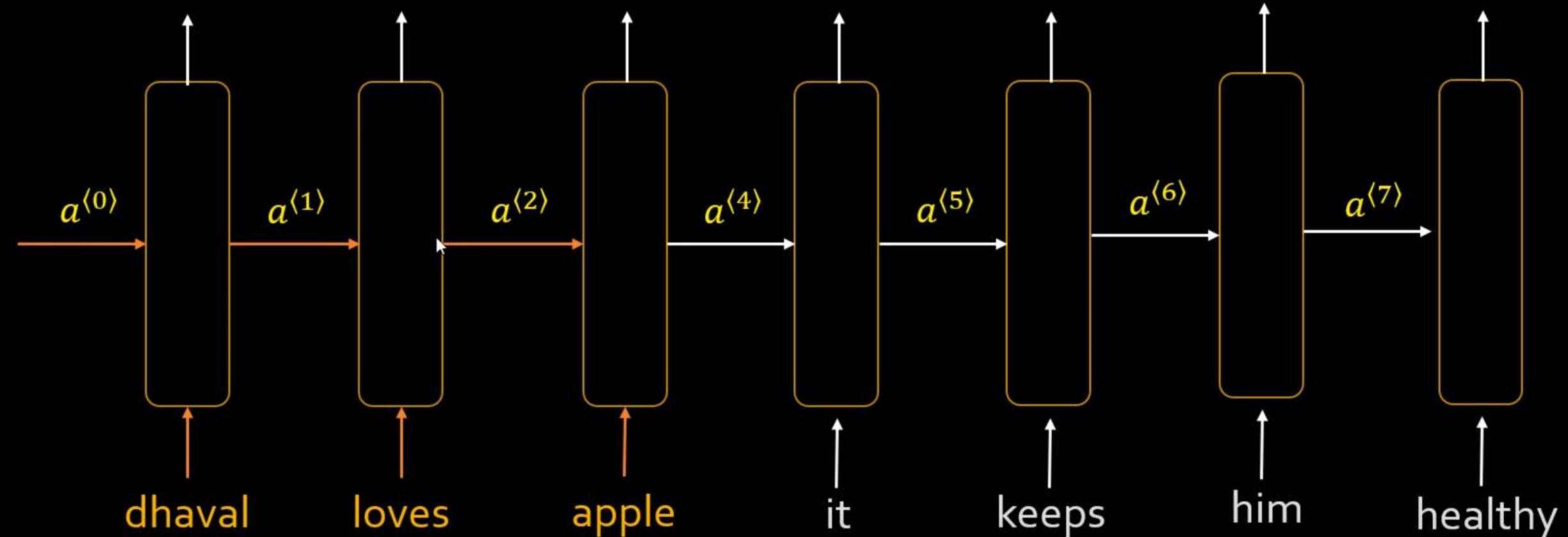
fruit or company?



Source: codebasics - YouTube

Bi-Directional RNNs: Intuition

fruit or company?

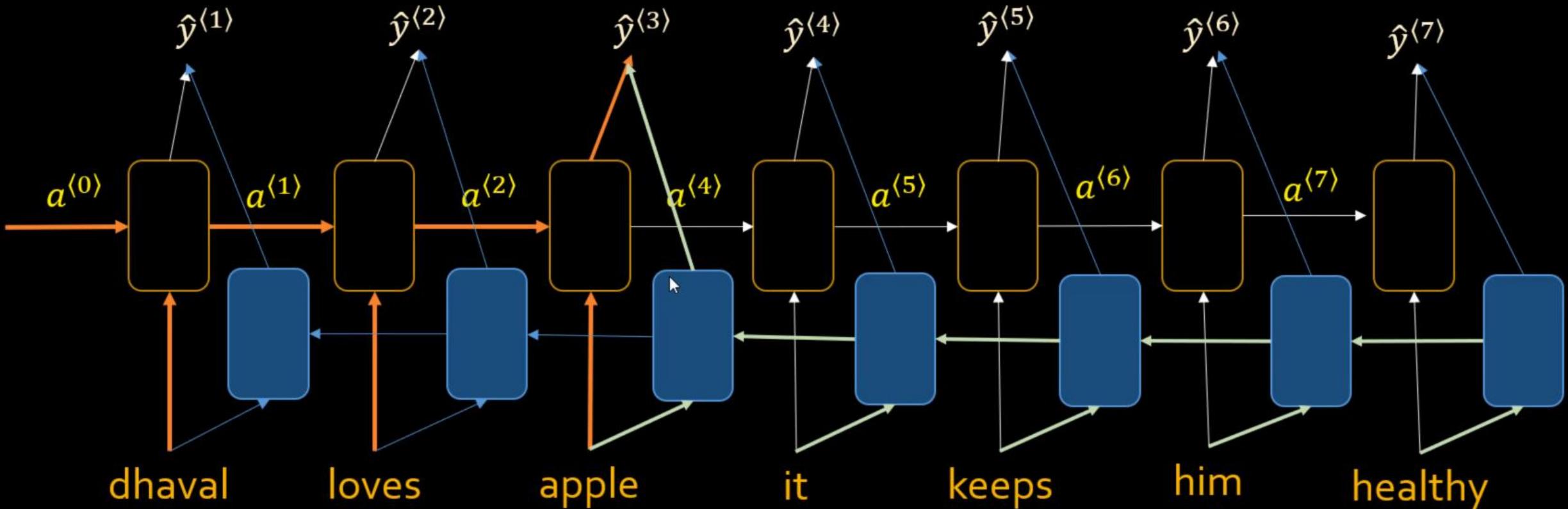


- The o/p at the third time step (where input is the string “apple”) depends on only previous two i/p/s

Source: codebasics - YouTube

Bi-Directional RNNs

fruit or company?

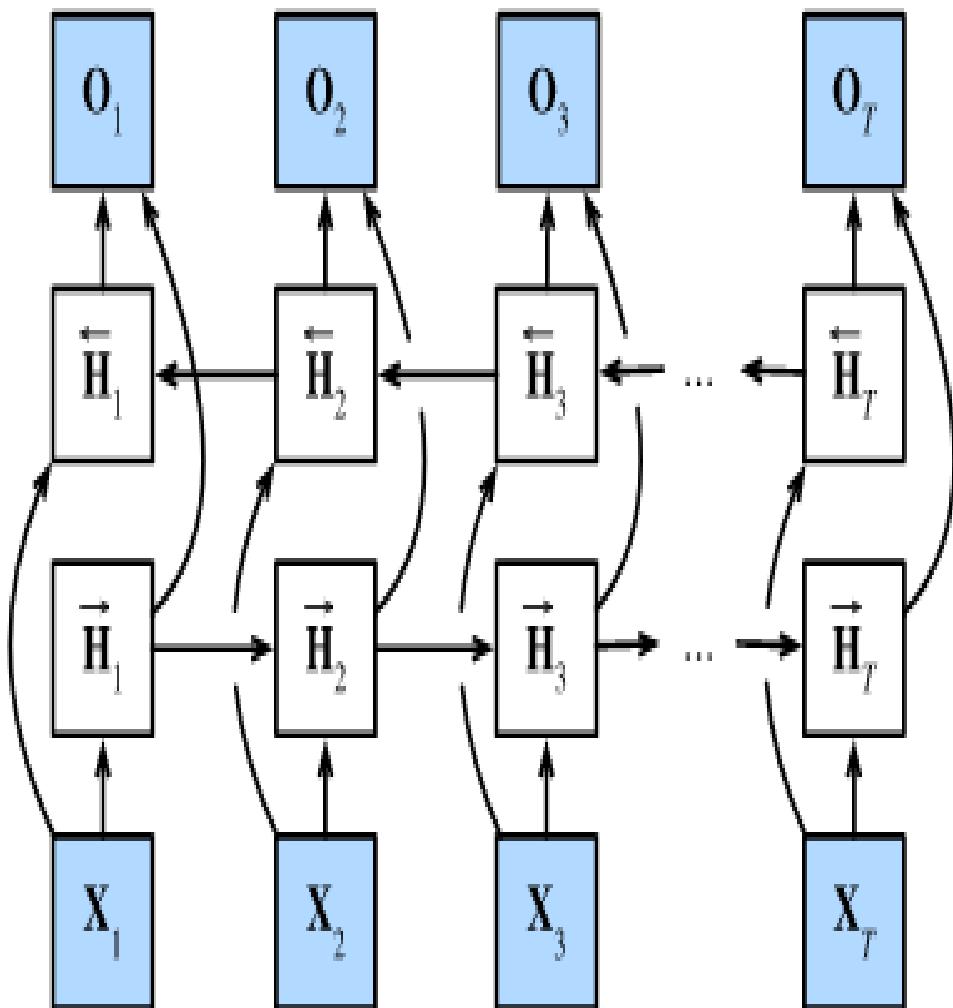


- Adding an additional backward layer with connections as shown above makes the o/p at a time step depend on both previous as well as future i/p's.

Source: codebasics - YouTube

Bi-directional RNNs

- Example - speech detection
- I am ____.
- I am ____ hungry.
- I am ____ hungry, and I can eat half a cake.
- Regular RNNs are causal
 - look at past and present inputs to generate output.
- Use 2 recurrent layers on the same inputs
 - One reading words from left to right
 - Another reading words from right to left
- Combine their outputs at each time step



Source: Bidirectional Recurrent Neural Networks — Dive into Deep Learning 1.0.0-alpha1.post0 documentation (d2l.ai)

Bi-directional RNN computation

$$H_t(\text{Forward}) = A(X_t * W_{xH}(\text{forward}) + H_{t-1}(\text{Forward}) * W_{HH}(\text{Forward}) + b_H(\text{Forward}))$$

$$H_t(\text{Backward}) = A(X_t * W_{xH}(\text{Backward}) + H_{t+1}(\text{Backward}) * W_{HH}(\text{Backward}) + b_H(\text{Backward}))$$

where,

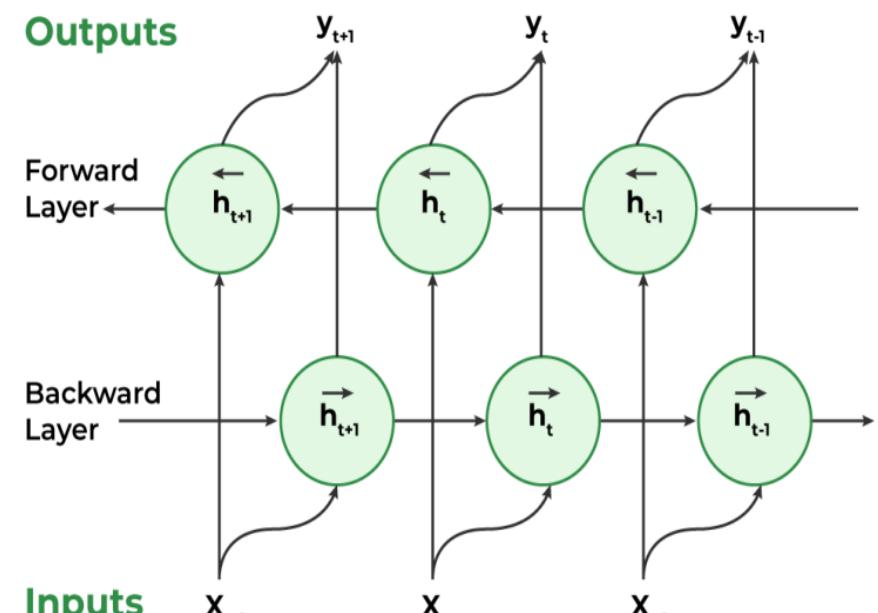
A = activation function,

W = weight matrix

b = bias

The output at any given hidden state is :

$$Y_t = H_t * W_{AY} + b_y, \text{ where } H_t \text{ is a concatenation of } H_t(\text{Forward}) \text{ and } H_t(\text{Backward})$$



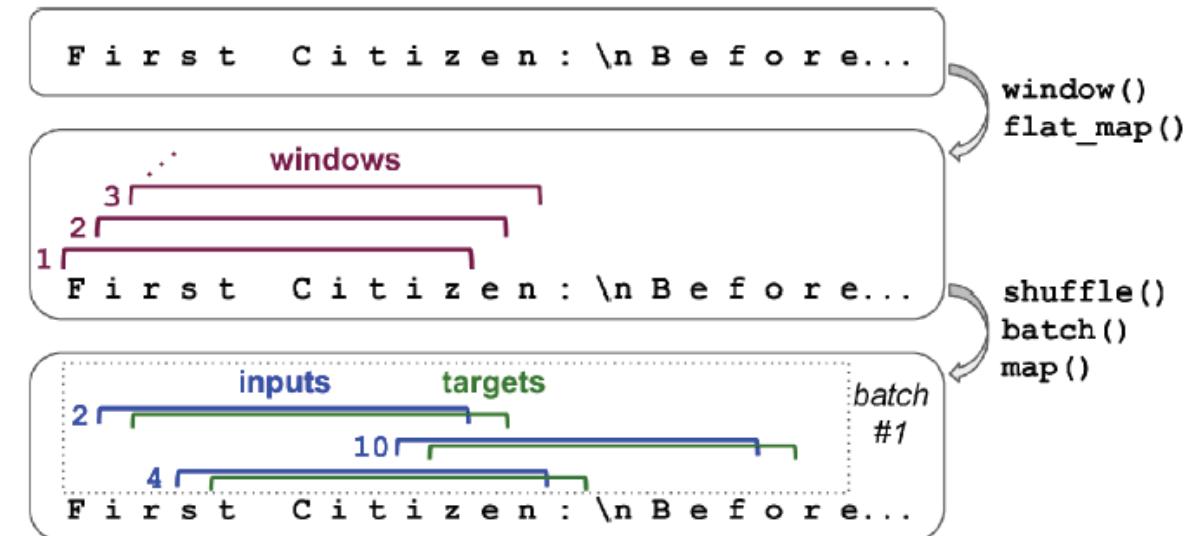
[Bidirectional Recurrent Neural Network - GeeksforGeeks](#)

Generating Shakespearan Text using a Character RNN

- “The unreasonable Effectiveness of Recurrent Neural Networks” – Andrej Karpathy (2015)
- 3-layer RNN with 512 hidden nodes on each layer
- Char-RNN was trained on Shakespeare’s work used to generate novel text- one character at a time
- PANDARUS:

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

- Chop the Sequential dataset into multiple windows



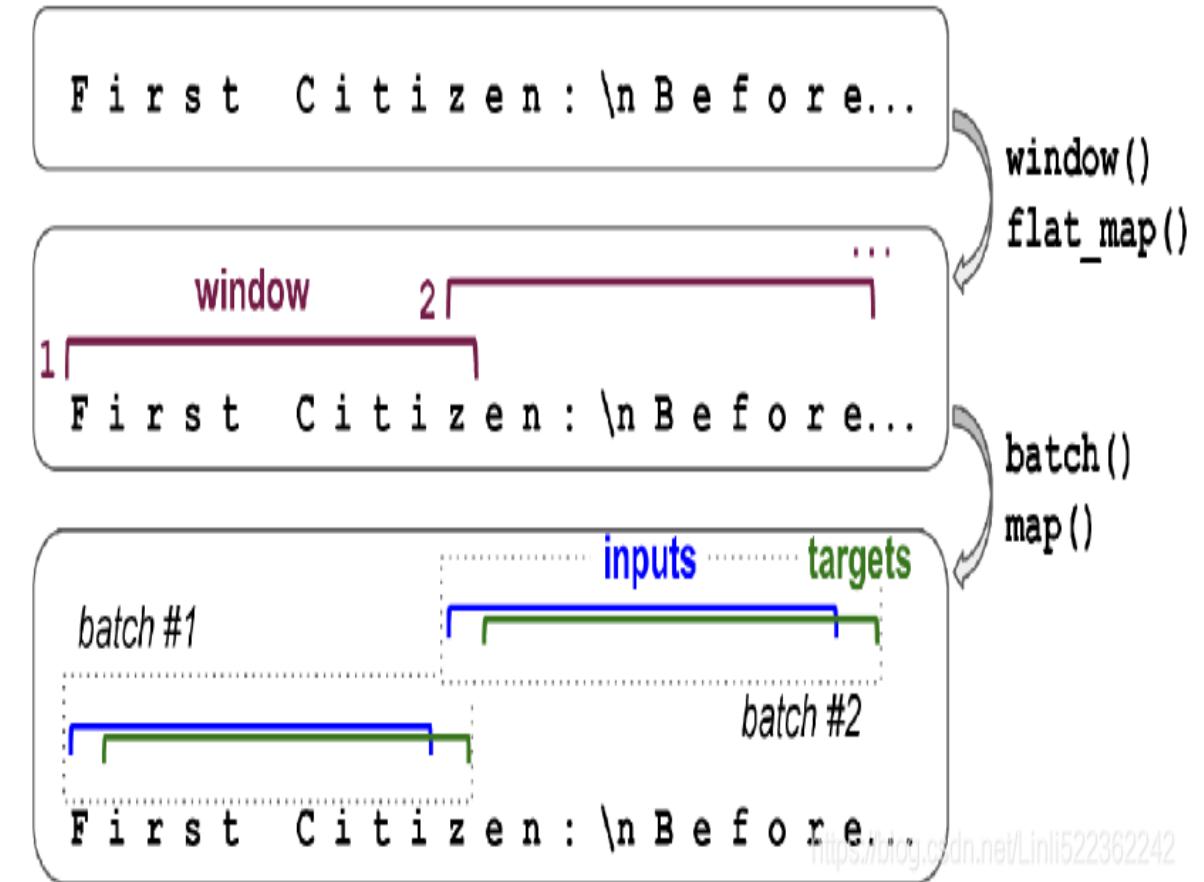
Stateful RNN

- **Stateless RNNs**

- at each training iteration the model starts with a hidden state full of 0s
- Update this state at each time step
- Discards the output at the final state when moving onto next training batch

- **Stateful RNN**

- Uses sequential nonoverlapping input sequences
- Preserves the final state after processing one training batch
- use it as initial state for next training batch
- Model will learn long-term patterns despite only backpropagating through short sequences



Encoder Decoder Models

DSE 5251 DEEP LEARNING

Rohini R Rao & Abhilash K Pai

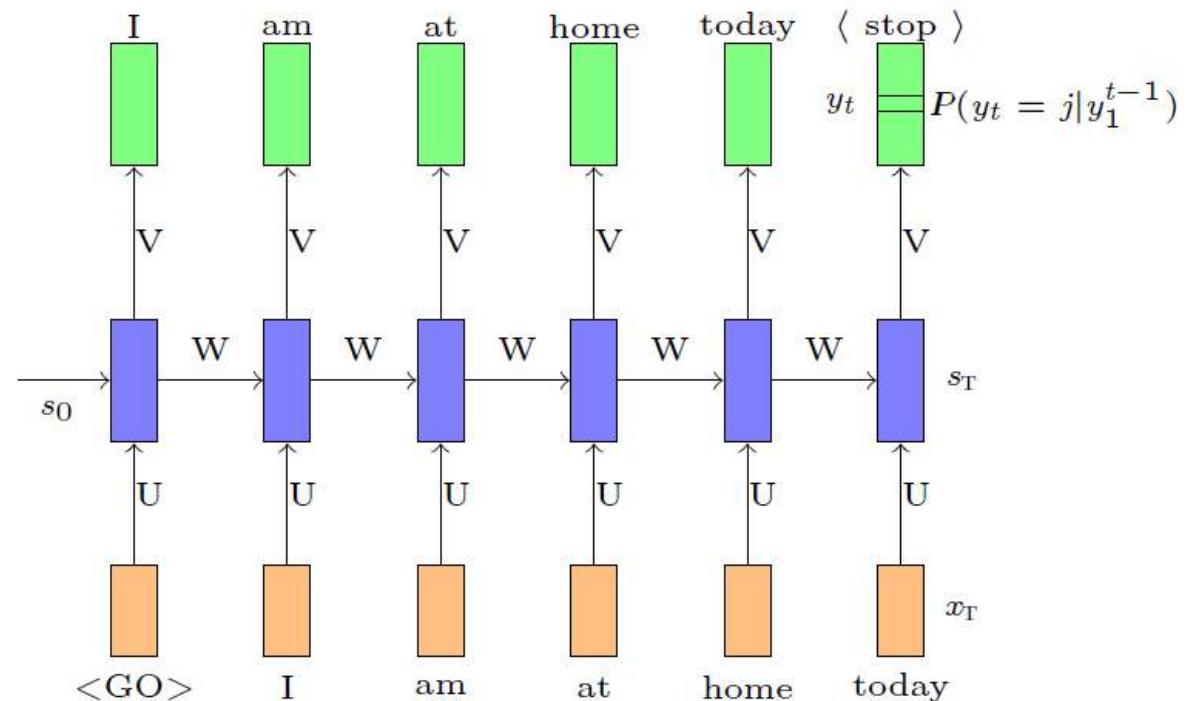
Department of Data Science and Computer Applications

MIT Manipal

Encoder Decoder Models : Introduction

- Language Modeling:

Given the $t-i$ words predict the t^{th} word



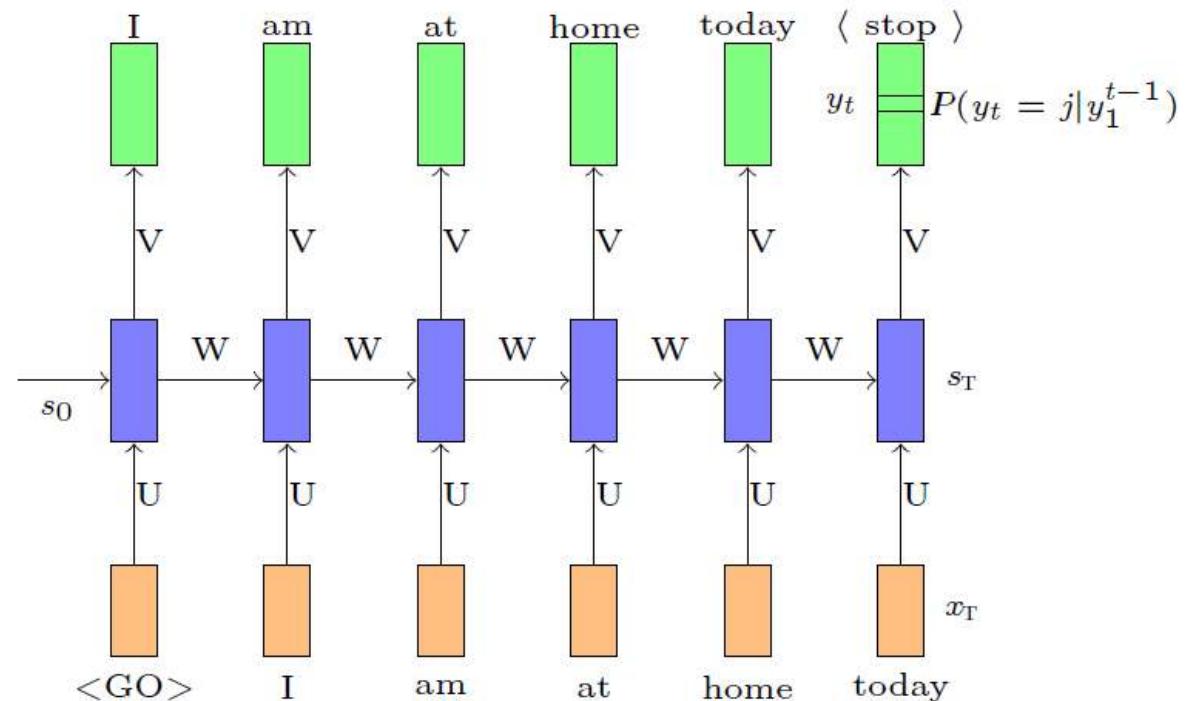
Encoder Decoder Models : Introduction

- Language Modeling:

Given the $t-i$ words predict the t^{th} word

More formally, given y_1, y_2, \dots, y_{t-1} we want to find

$$y^* = \operatorname{argmax} P(y_t | y_1, y_2, \dots, y_{t-1})$$



Encoder Decoder Models : Introduction

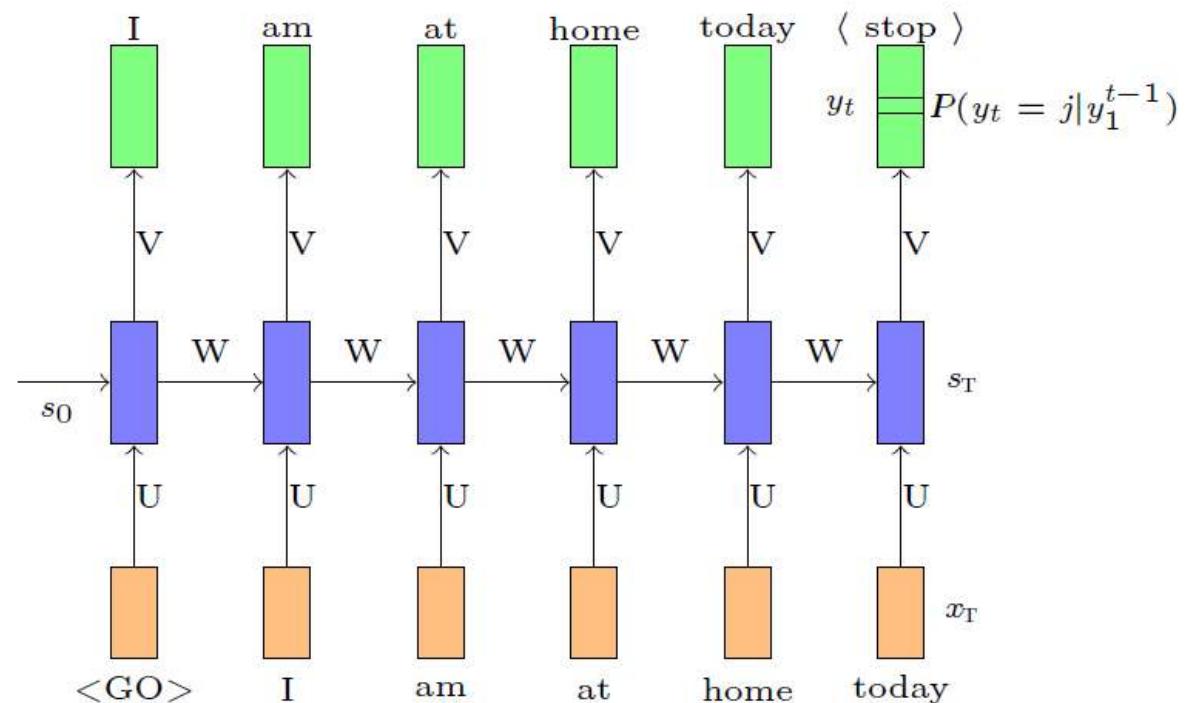
- Language Modeling:

Given the $t-i$ words predict the t^{th} word

More formally, given y_1, y_2, \dots, y_{t-1} we want to find

$$y^* = \operatorname{argmax} P(y_t | y_1, y_2, \dots, y_{t-1})$$

For simplicity let us denote



Encoder Decoder Models : Introduction

- Language Modeling:

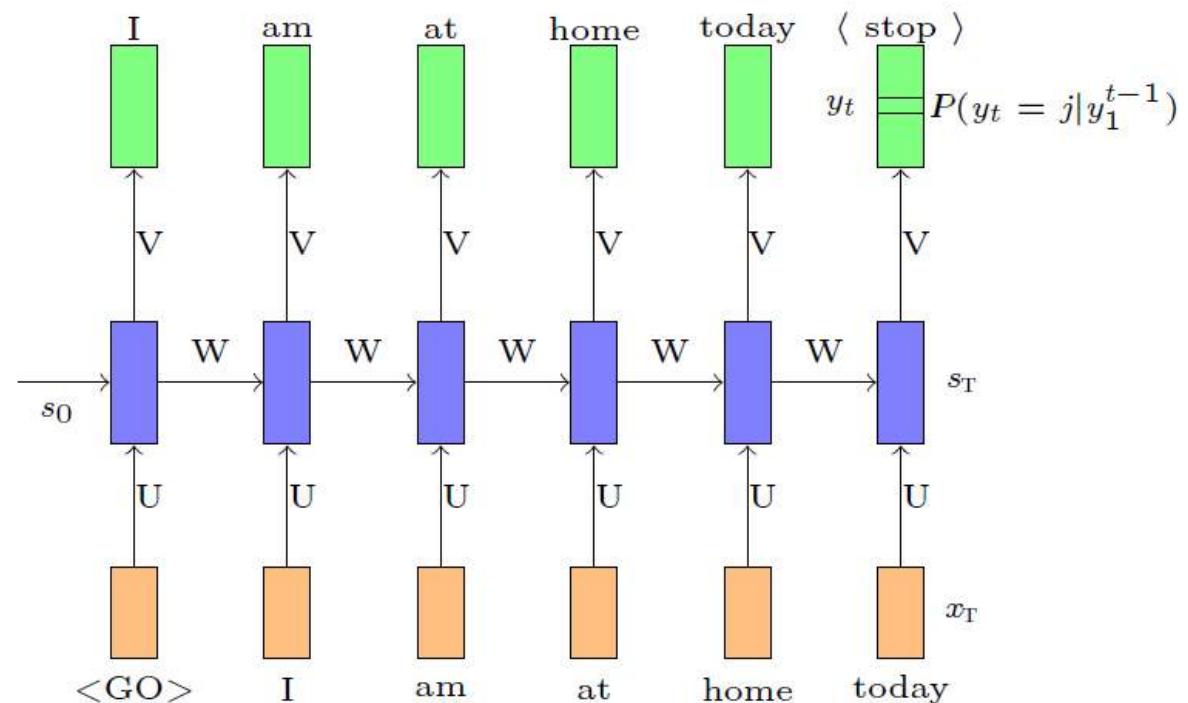
Given the t - i words predict the t^{th} word

More formally, given y_1, y_2, \dots, y_{t-1} we want to find

$$y^* = \operatorname{argmax} P(y_t | y_1, y_2, \dots, y_{t-1})$$

For simplicity let us denote

$$P(y_t | y_1, y_2 \dots y_{t-1}) = P(y_t | y_1^{t-1})$$



Encoder Decoder Models : Introduction

- Language Modeling:

Given the t - i words predict the t^{th} word

More formally, given y_1, y_2, \dots, y_{t-1} we want to find

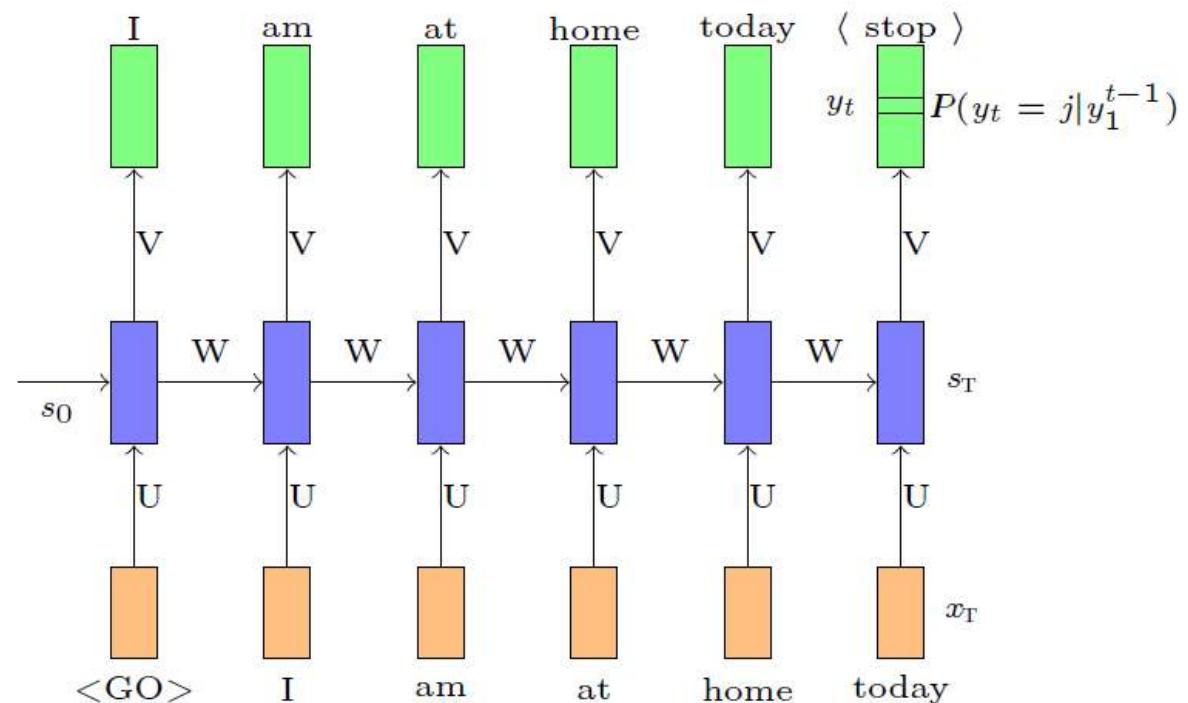
$$y^* = \operatorname{argmax} P(y_t | y_1, y_2, \dots, y_{t-1})$$

For simplicity let us denote

$$P(y_t | y_1, y_2 \dots y_{t-1}) = P(y_t | y_1^{t-1})$$

We are interested in:

$$P(y_t = j | y_1, y_2 \dots y_{t-1})$$



Encoder Decoder Models : Introduction

- Language Modeling:

Given the $t-i$ words predict the t^{th} word

More formally, given y_1, y_2, \dots, y_{t-1} we want to find

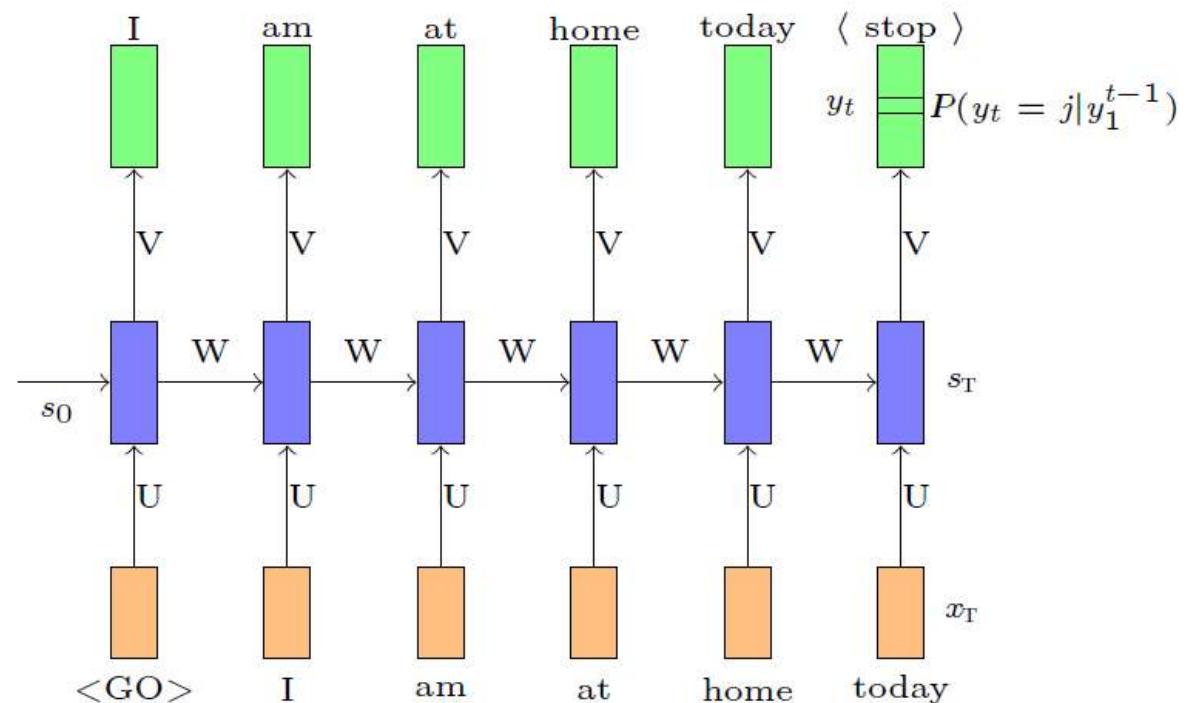
$$y^* = \operatorname{argmax} P(y_t | y_1, y_2, \dots, y_{t-1})$$

For simplicity let us denote

$$P(y_t | y_1, y_2 \dots y_{t-1}) = P(y_t | y_1^{t-1})$$

We are interested in:

$$P(y_t = j | y_1, y_2 \dots y_{t-1}) \xrightarrow{\text{j}^{\text{th}} \text{ word in the vocabulary}}$$



Encoder Decoder Models : Introduction

- Language Modeling:

Given the $t-i$ words predict the t^{th} word

More formally, given y_1, y_2, \dots, y_{t-1} we want to find

$$y^* = \operatorname{argmax} P(y_t | y_1, y_2, \dots, y_{t-1})$$

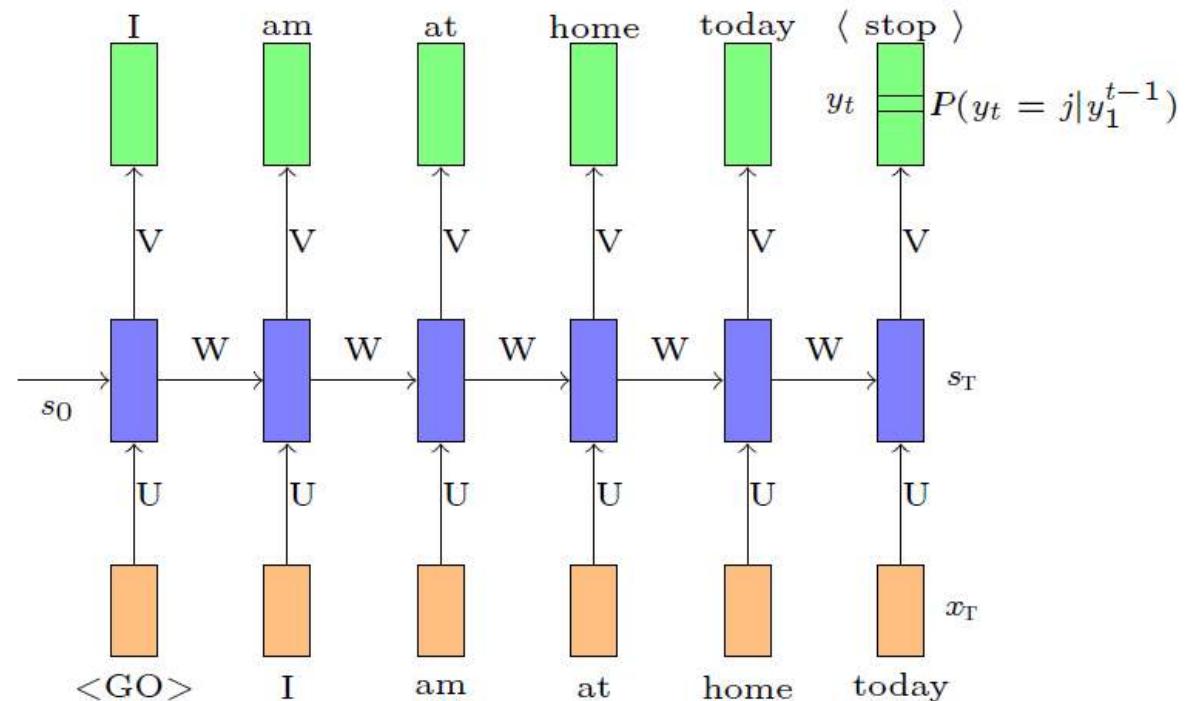
For simplicity let us denote

$$P(y_t | y_1, y_2 \dots y_{t-1}) = P(y_t | y_1^{t-1})$$

We are interested in:

$$P(y_t = j | y_1, y_2 \dots y_{t-1}) \xrightarrow{\text{j}^{\text{th}} \text{ word in the vocabulary}}$$

Using an RNN we will compute this as:



Encoder Decoder Models : Introduction

- Language Modeling:

Given the t - i words predict the t^{th} word

More formally, given y_1, y_2, \dots, y_{t-1} we want to find

$$y^* = \operatorname{argmax} P(y_t | y_1, y_2, \dots, y_{t-1})$$

For simplicity let us denote

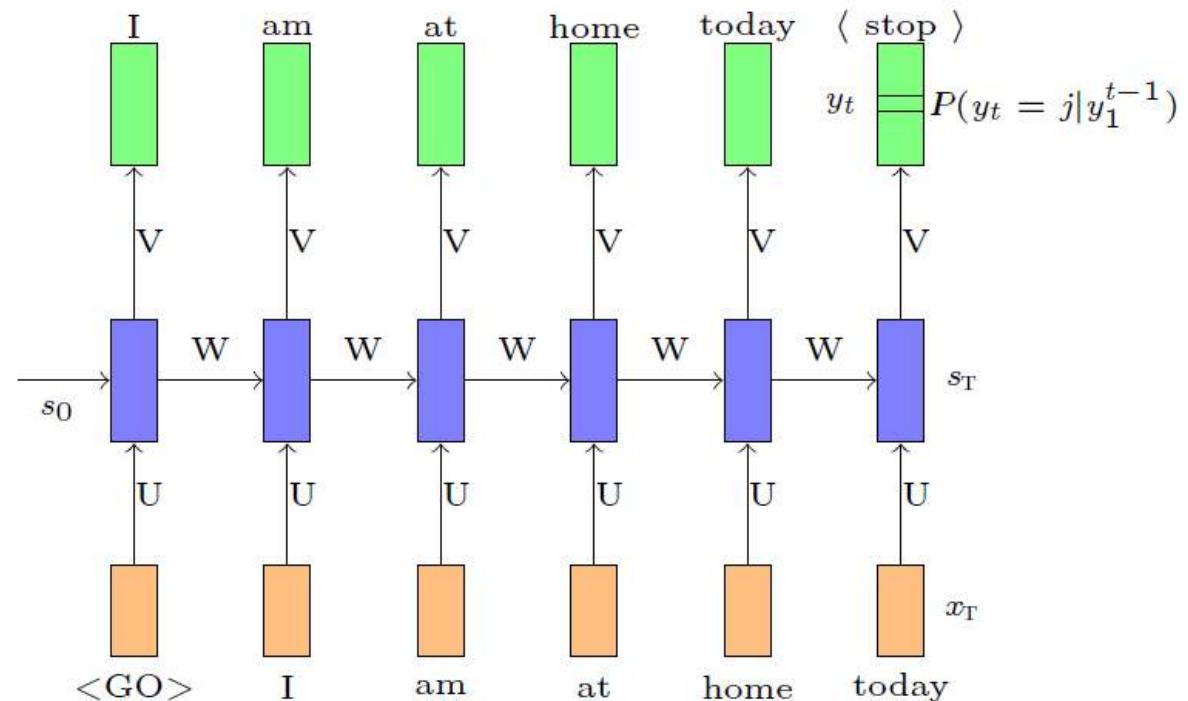
$$P(y_t | y_1, y_2 \dots y_{t-1}) = P(y_t | y_1^{t-1})$$

We are interested in:

$$P(y_t = j | y_1, y_2 \dots y_{t-1}) \xrightarrow{\text{j}^{\text{th}} \text{ word in the vocabulary}}$$

Using an RNN we will compute this as:

$$P(y_t = j | y_1^{t-1}) = \operatorname{softmax}(Vs_t + c)_j$$



Encoder Decoder Models : Introduction

- Language Modeling:

Given the t - i words predict the t^{th} word

More formally, given y_1, y_2, \dots, y_{t-1} we want to find

$$y^* = \operatorname{argmax} P(y_t | y_1, y_2, \dots, y_{t-1})$$

For simplicity let us denote

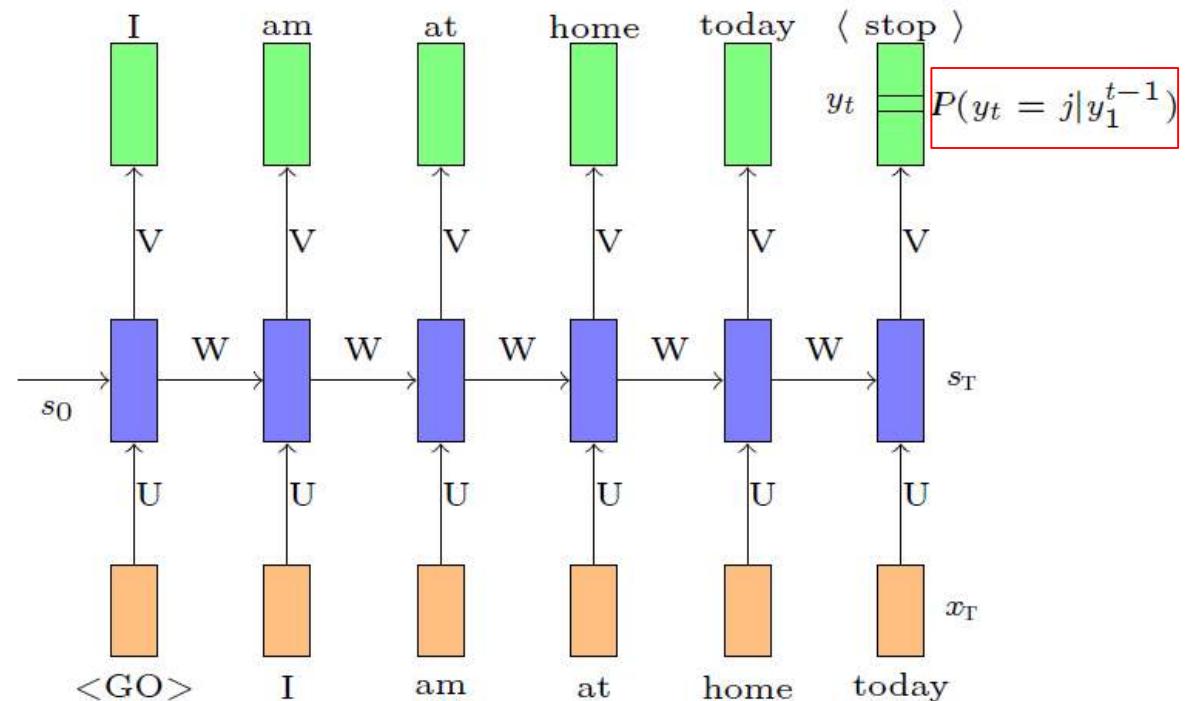
$$P(y_t | y_1, y_2 \dots y_{t-1}) = P(y_t | y_1^{t-1})$$

We are interested in:

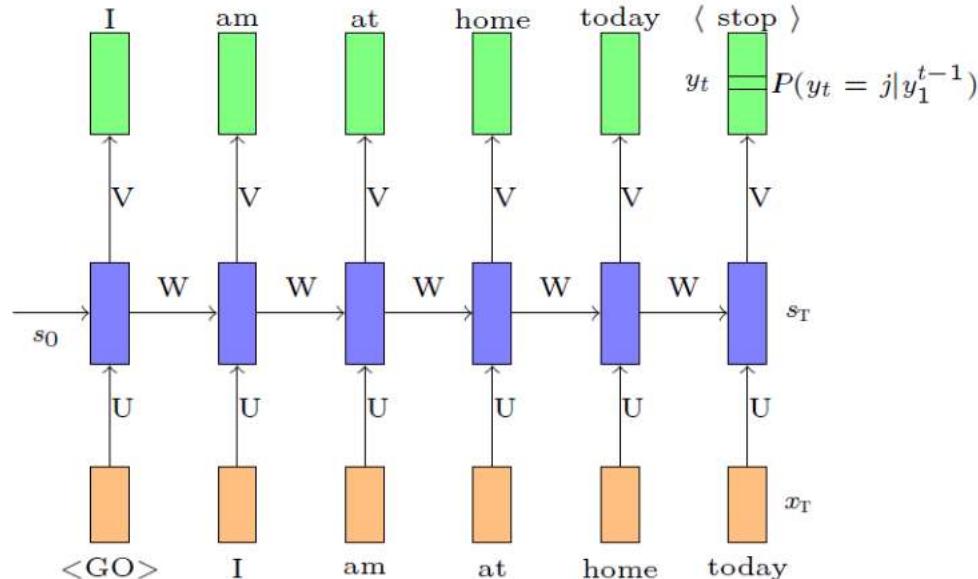
$$P(y_t = j | y_1, y_2 \dots y_{t-1}) \xrightarrow{\text{j}^{\text{th}} \text{ word in the vocabulary}}$$

Using an RNN we will compute this as:

$$P(y_t = j | y_1^{t-1}) = \operatorname{softmax}(Vs_t + c)_j$$



Encoder Decoder Models : Introduction



Data:

India, officially the Republic of India, is a country in South Asia. It is the seventh-largest country by area,

- **Data:** All sentences from any large corpus (say wikipedia)
- **Model:**

$$s_t = \sigma(Ws_{t-1} + Ux_t + b)$$

$$P(y_t = j|y_1^{t-1}) = \text{softmax}(Vs_t + c)_j$$

- **Parameters:** U, V, W, b, c
- **Loss:**

$$\mathcal{L}(\theta) = \sum_{t=1}^T \mathcal{L}_t(\theta)$$

$$\mathcal{L}_t(\theta) = -\log P(y_t = \ell_t|y_1^{t-1})$$

where ℓ_t is the true word at time step t

Encoder Decoder Models : Introduction

- Shorthand notations:

$$s_t = \sigma(U \textcolor{red}{x}_t + W \textcolor{red}{s}_{t-1} + b) \quad \tilde{s}_t = \sigma(W(o_t \odot \textcolor{red}{s}_{t-1}) + U \textcolor{red}{x}_t + b) \quad \tilde{s}_t = \sigma(W h_{t-1} + U \textcolor{red}{x}_t + b)$$

$$s_t = i_t \odot \textcolor{red}{s}_{t-1} + (1 - i_t) \odot \tilde{s}_t \quad s_t = f_t \odot \textcolor{red}{s}_{t-1} + i_t \odot \tilde{s}_t$$

$$h_t = o_t \odot \sigma(s_t)$$



$$s_t = \text{RNN}(\textcolor{red}{s}_{t-1}, \textcolor{red}{x}_t)$$

$$s_t = \text{GRU}(\textcolor{red}{s}_{t-1}, \textcolor{red}{x}_t)$$

$$h_t, s_t = \text{LSTM}(\textcolor{red}{h}_{t-1}, \textcolor{red}{s}_{t-1}, \textcolor{red}{x}_t)$$

Encoder Decoder Models : Introduction

Task: generate a sentence given an image



A man throwing
a frisbee in a park

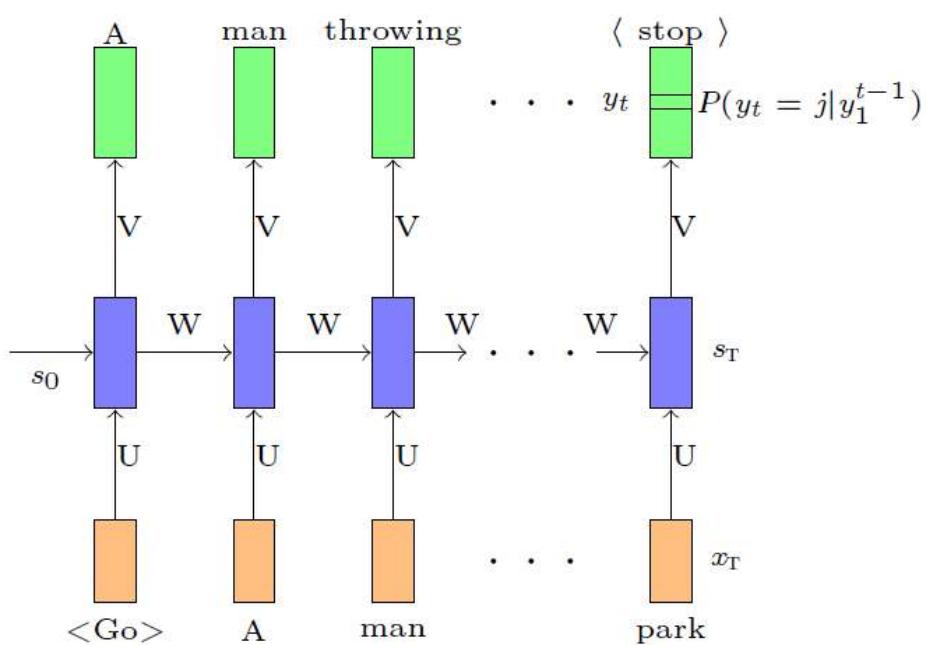
We are now interested in $P(y_t|y_1^{t-1}, I)$
instead of $P(y_t|y_1^{t-1})$ where I is an
image

Encoder Decoder Models : Introduction

- Earlier we modeled $P(y_t|y_1^{t-1})$ as

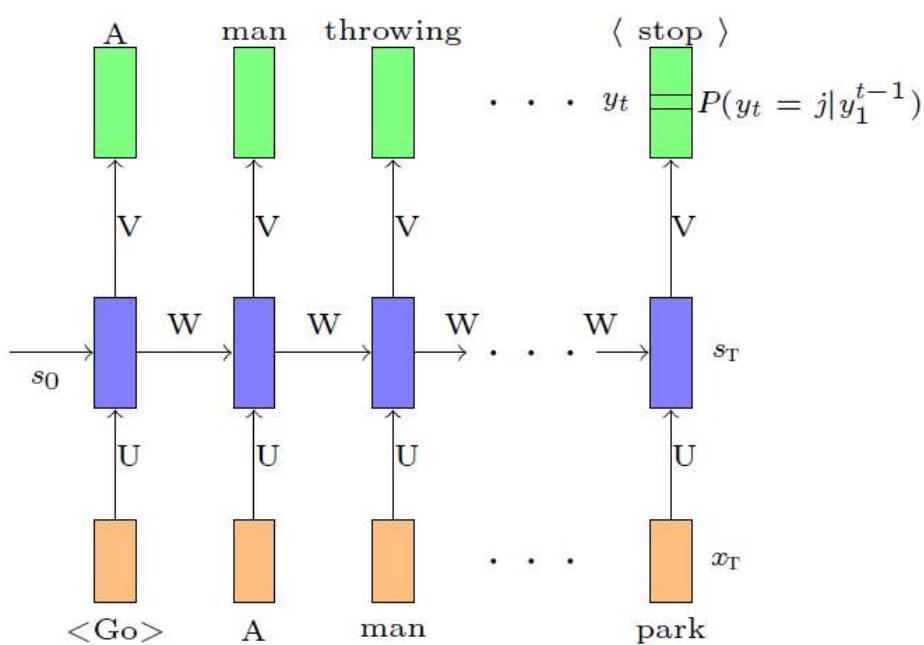
$$P(y_t|y_1^{t-1}) = P(y_t = j|s_t)$$

Encoder Decoder Models : Introduction



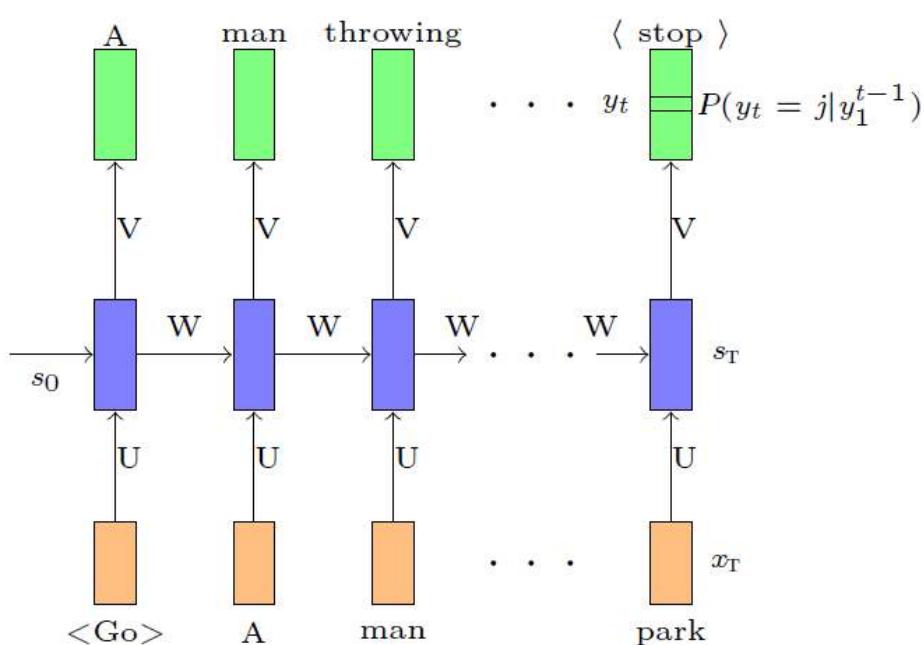
- Earlier we modeled $P(y_t | y_1^{t-1})$ as
$$P(y_t | y_1^{t-1}) = P(y_t = j | s_t)$$

Encoder Decoder Models : Introduction



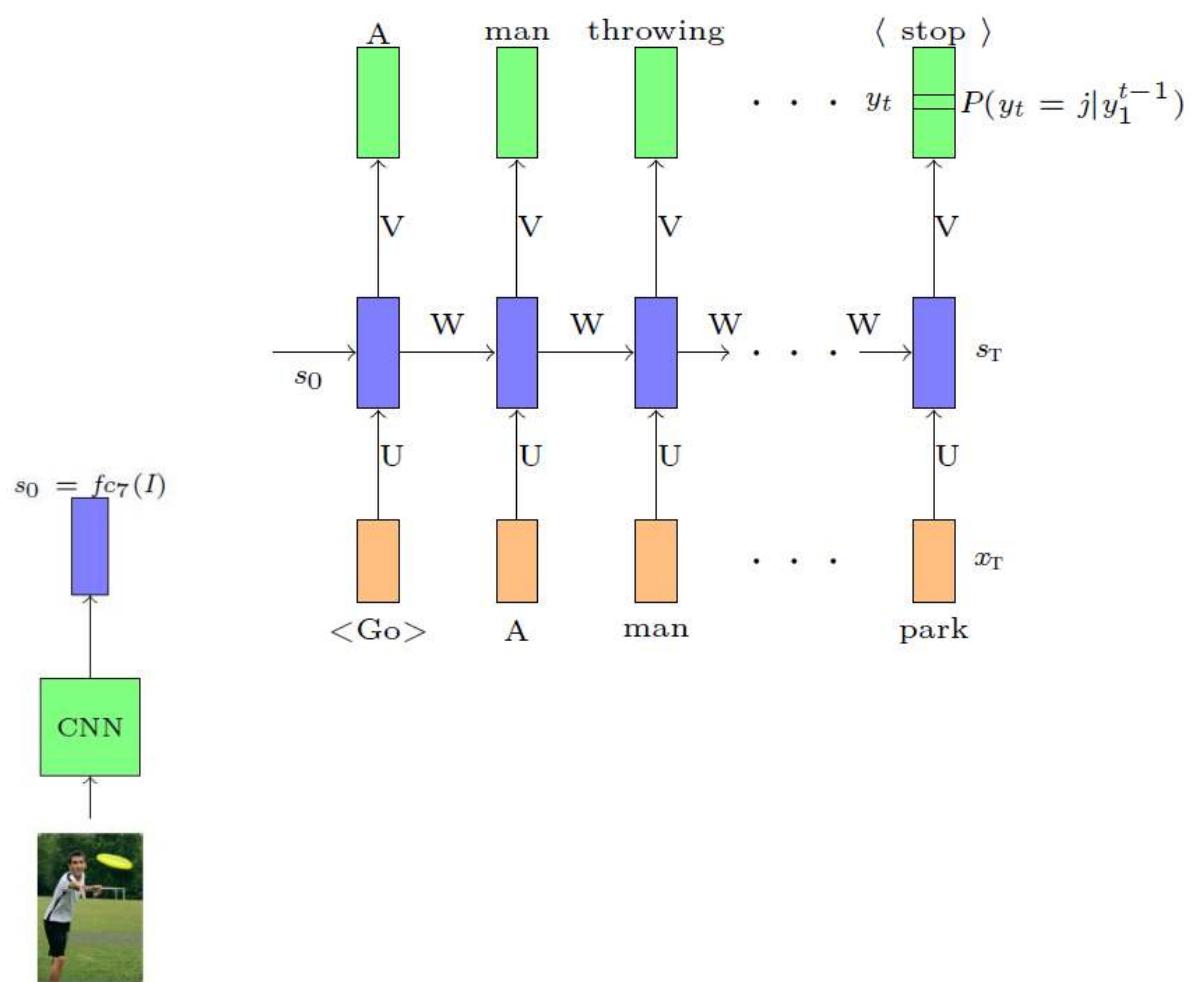
- Earlier we modeled $P(y_t | y_1^{t-1})$ as
$$P(y_t | y_1^{t-1}) = P(y_t = j | s_t)$$
- Where s_t was a state capturing all the previous words

Encoder Decoder Models : Introduction



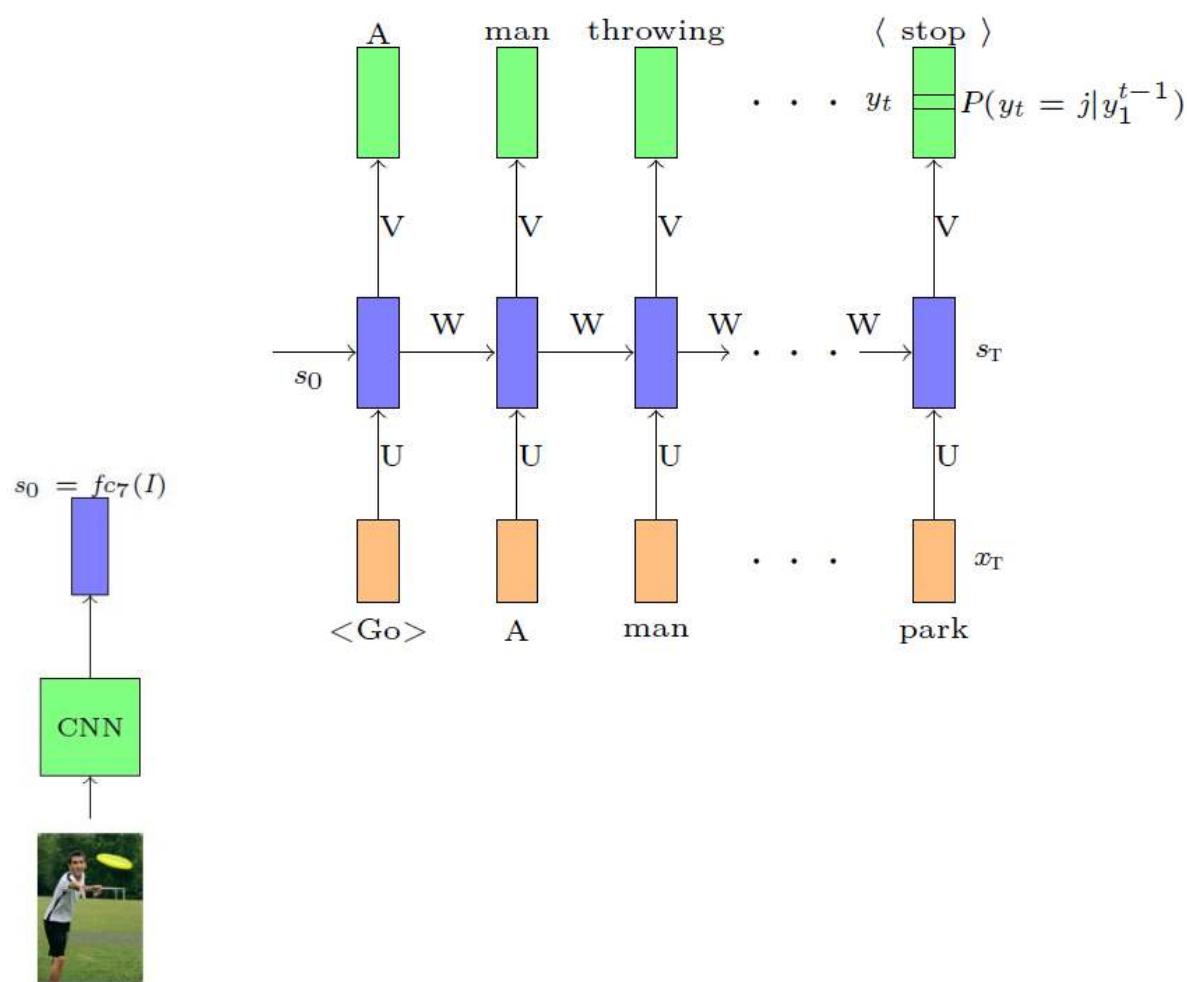
- Earlier we modeled $P(y_t | y_1^{t-1})$ as
$$P(y_t | y_1^{t-1}) = P(y_t = j | s_t)$$
- Where s_t was a state capturing all the previous words
- We could now model $P(y_t = j | y_1^{t-1}, I)$ as $P(y_t = j | s_t, f_{c7}(I))$

Encoder Decoder Models : Introduction



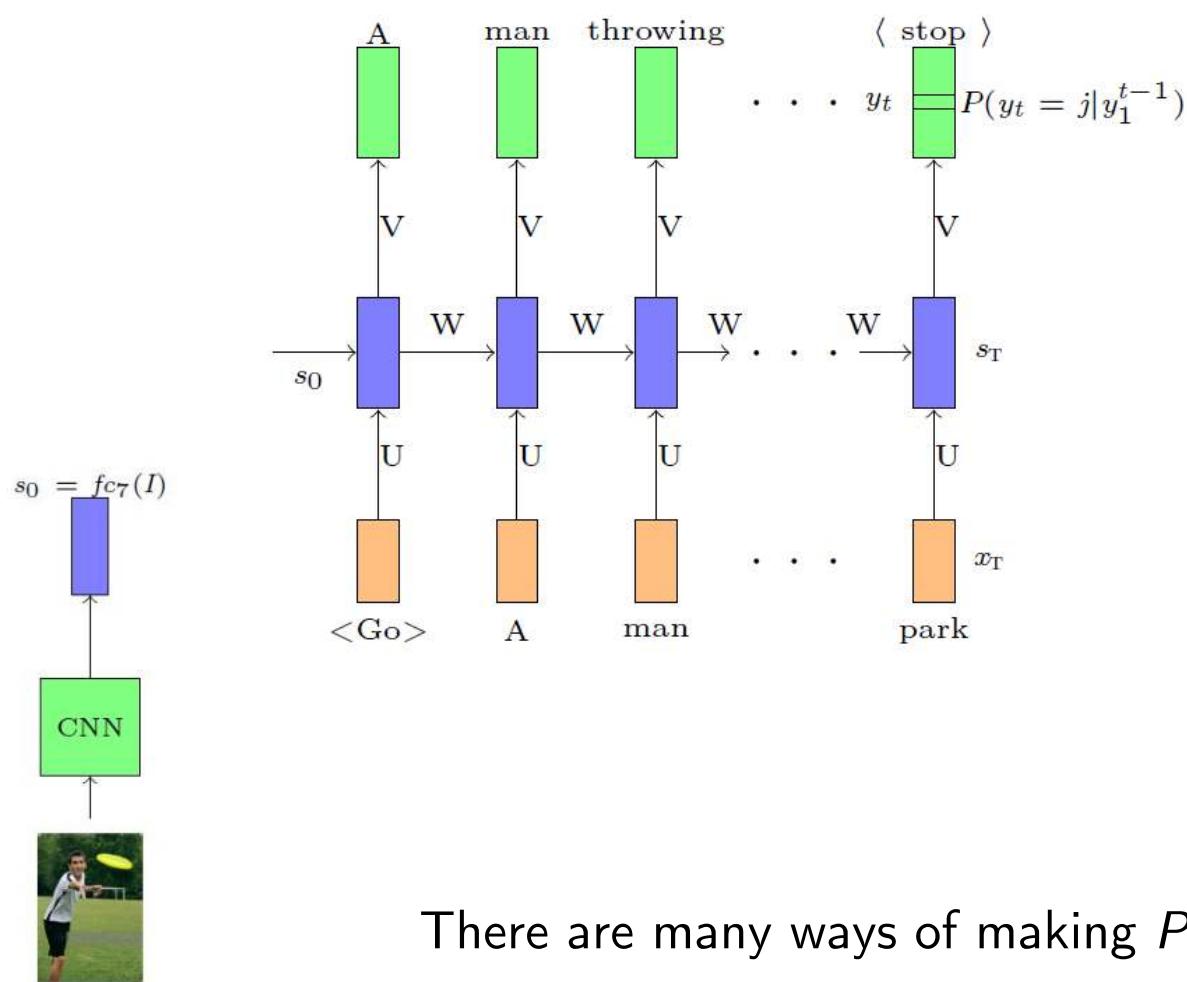
- Earlier we modeled $P(y_t | y_1^{t-1})$ as
$$P(y_t | y_1^{t-1}) = P(y_t = j | s_t)$$
- Where s_t was a state capturing all the previous words
- We could now model $P(y_t = j | y_1^{t-1}, I)$ as $P(y_t = j | s_t, f_{c7}(I))$

Encoder Decoder Models : Introduction



- Earlier we modeled $P(y_t|y_1^{t-1})$ as
$$P(y_t|y_1^{t-1}) = P(y_t = j|s_t)$$
- Where s_t was a state capturing all the previous words
- We could now model $P(y_t = j|y_1^{t-1}, I)$ as $P(y_t = j|s_t, f_{c7}(I))$
- where $f_{c7}(I)$ is the representation obtained from the f_{c7} layer of an image

Encoder Decoder Models : Introduction



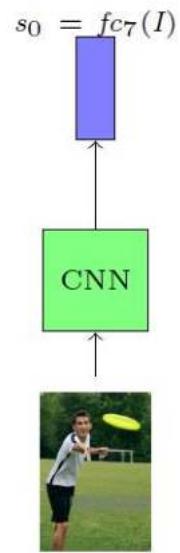
- Earlier we modeled $P(y_t|y_1^{t-1})$ as
$$P(y_t|y_1^{t-1}) = P(y_t = j|s_t)$$
- Where s_t was a state capturing all the previous words
- We could now model $P(y_t = j|y_1^{t-1}, I)$ as $P(y_t = j|s_t, f_{c7}(I))$
- where $f_{c7}(I)$ is the representation obtained from the f_{c7} layer of an image

There are many ways of making $P(y_t = j)$ conditional on $f_{c7}(I)$

Encoder Decoder Models : Introduction

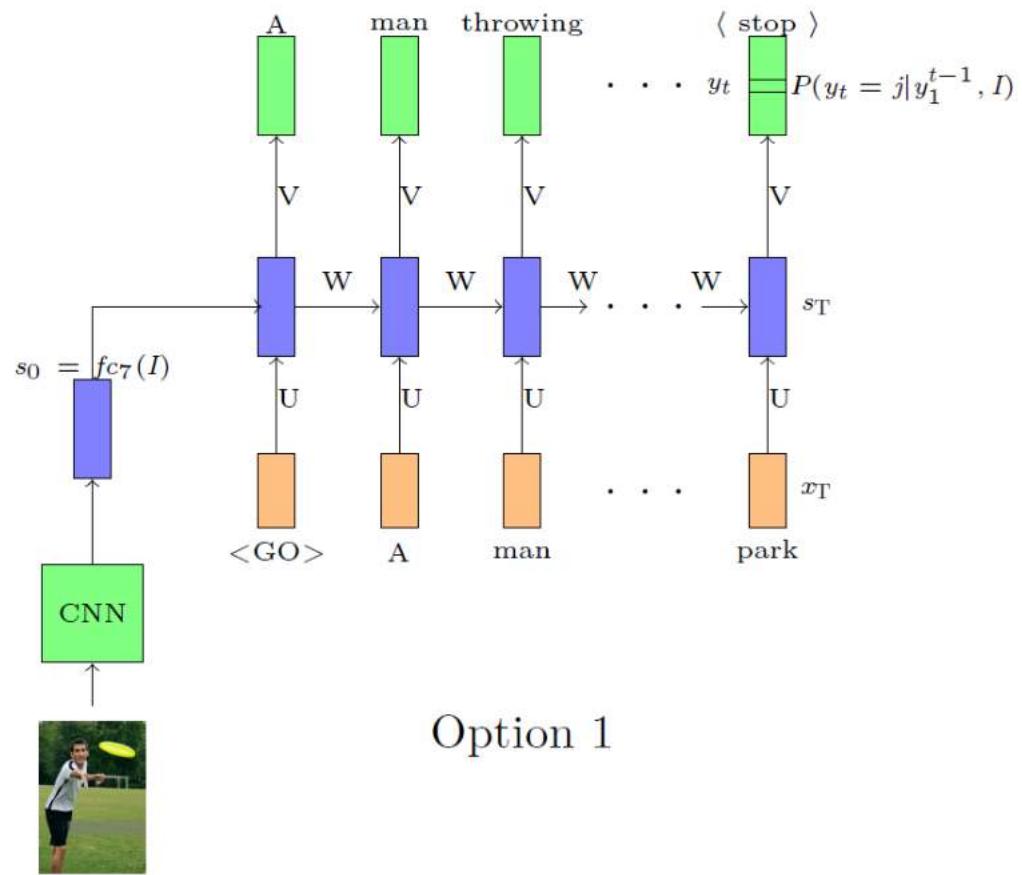
Option 1: Set $s_0 = f_{c7}(I)$

Now s_0 and hence all subsequent s_t 's depend on $f_{c7}(I)$



Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Encoder Decoder Models : Introduction



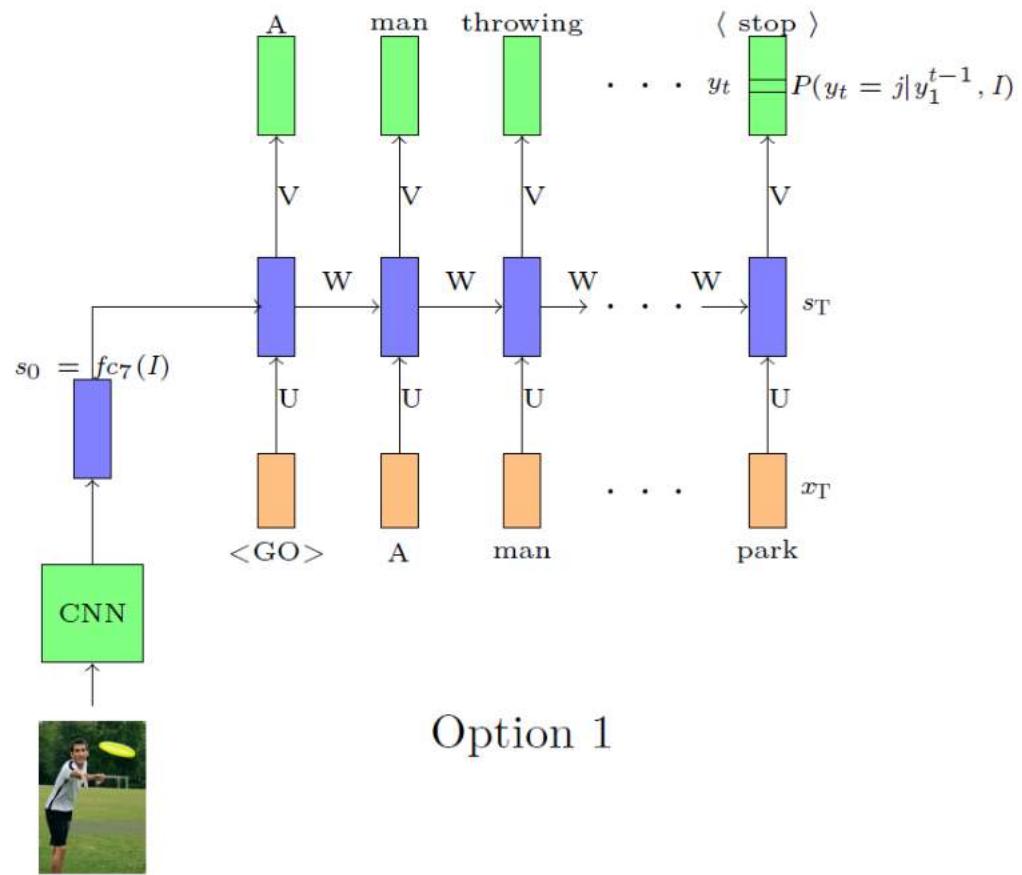
Option 1: Set $s_0 = f_{c7}(I)$

Now s_0 and hence all subsequent s_t 's depend on $f_{c7}(I)$

We can thus say that $P(y_t = j)$ depends on $f_{c7}(I)$

In other words, we are computing $P(y_t = j | s_t, f_{c7}(I))$

Encoder Decoder Models : Introduction



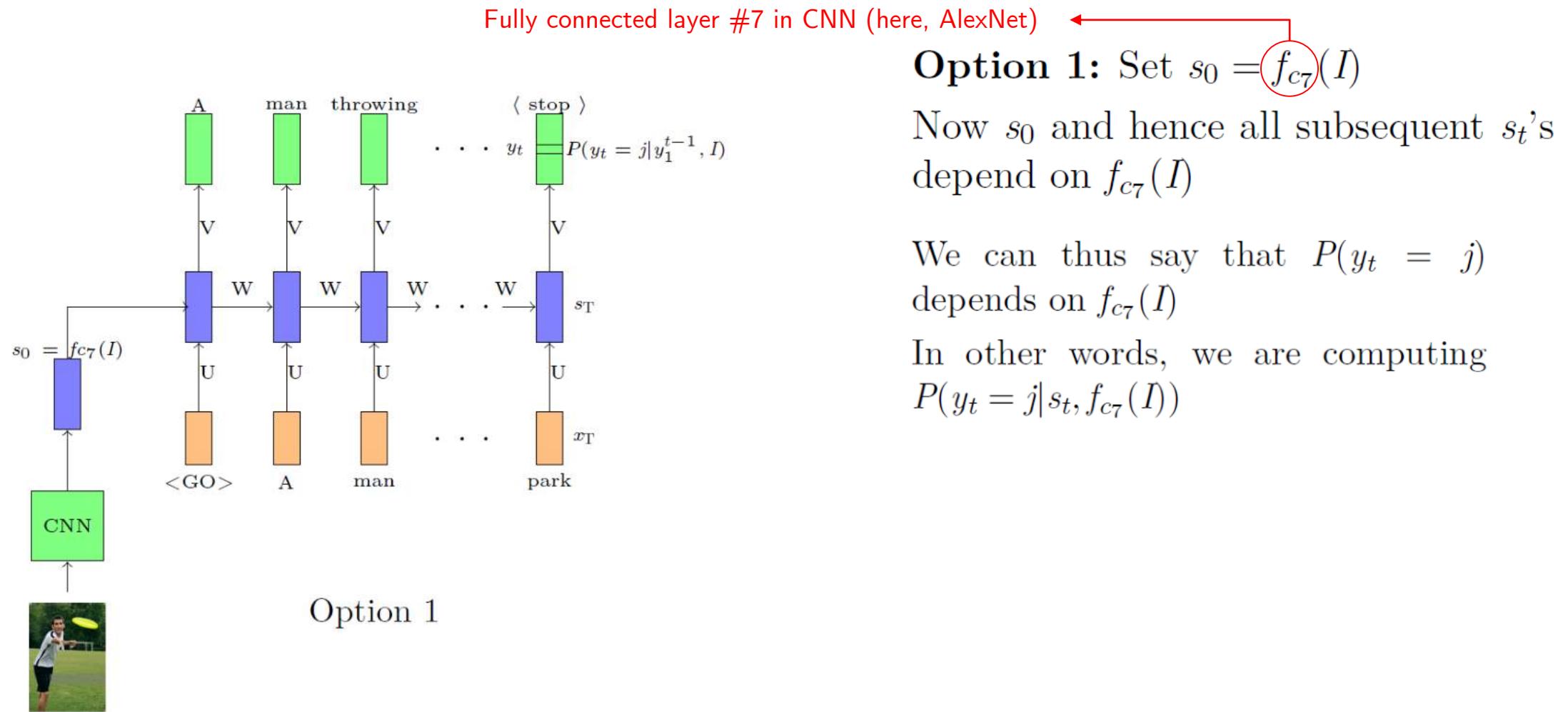
Option 1: Set $s_0 = f_{c7}(I)$

Now s_0 and hence all subsequent s_t 's depend on $f_{c7}(I)$

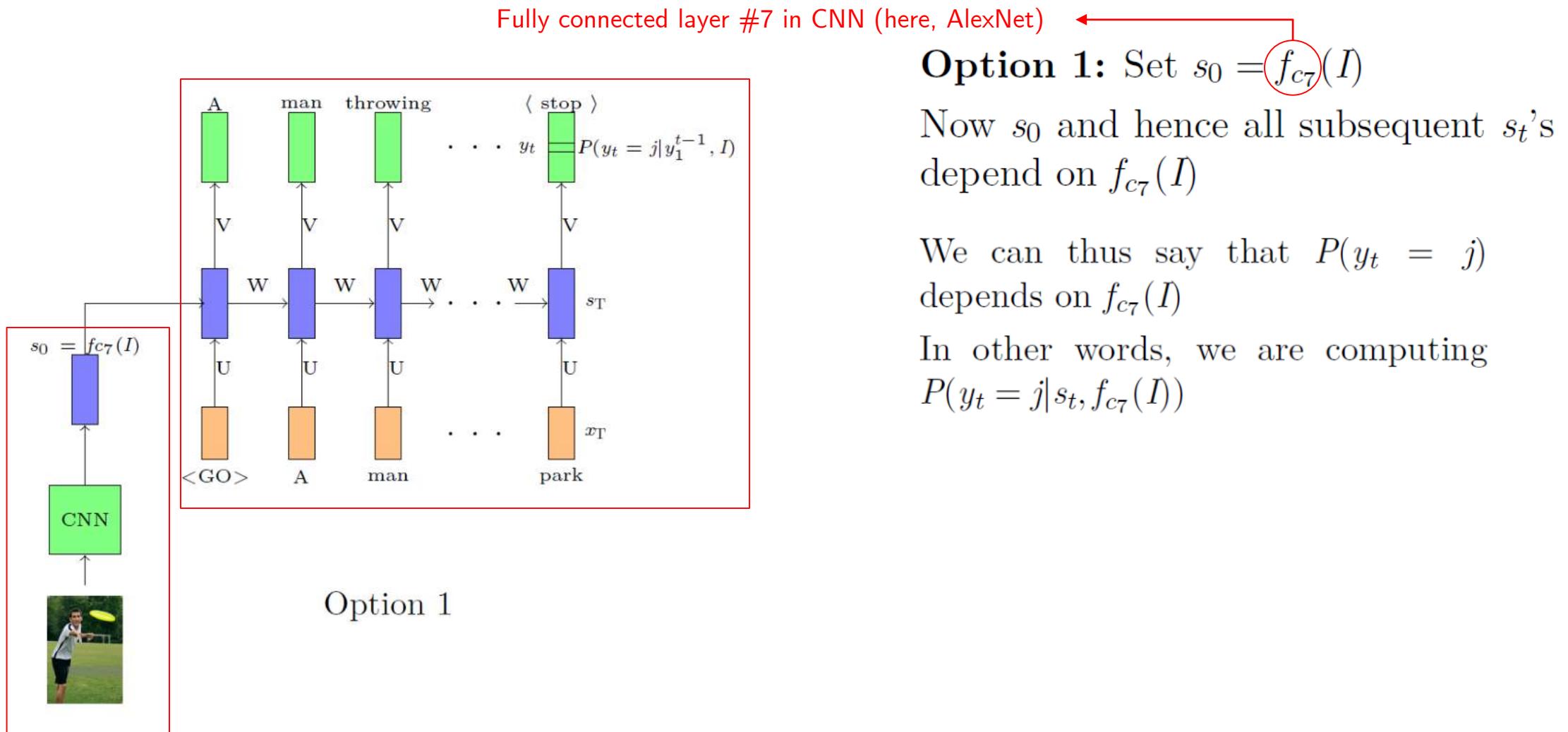
We can thus say that $P(y_t = j)$ depends on $f_{c7}(I)$

In other words, we are computing $P(y_t = j | s_t, f_{c7}(I))$

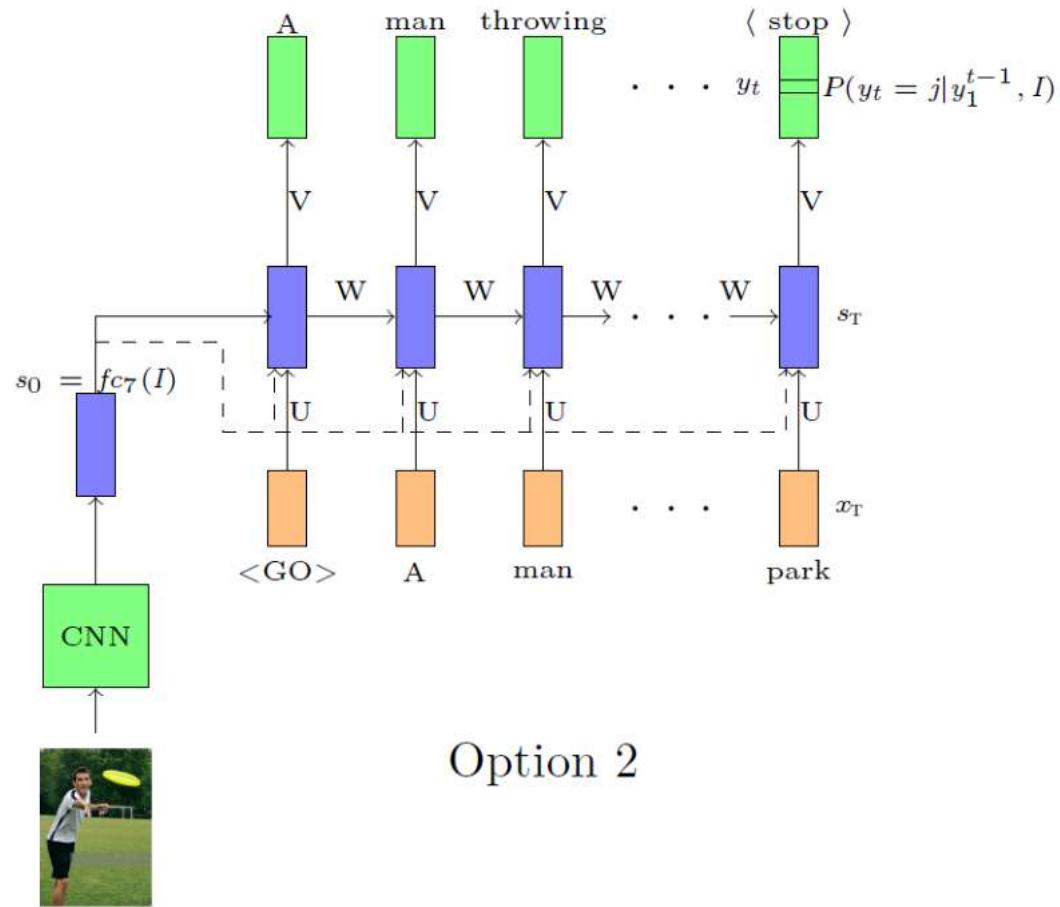
Encoder Decoder Models : Introduction



Encoder Decoder Models : Introduction



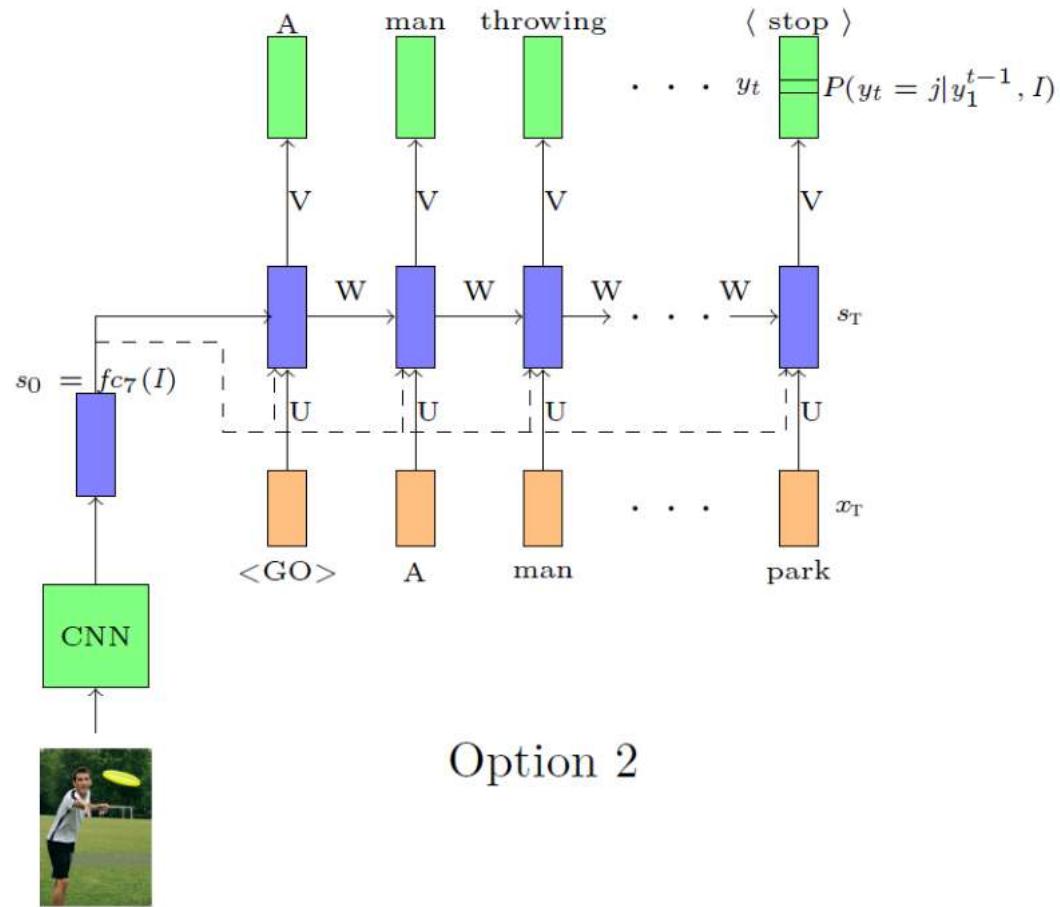
Encoder Decoder Models : Introduction



Option 2: Another more explicit way of doing this is to compute

$$s_t = RNN(s_{t-1}, [x_t, f_{c7}(I)])$$

Encoder Decoder Models : Introduction

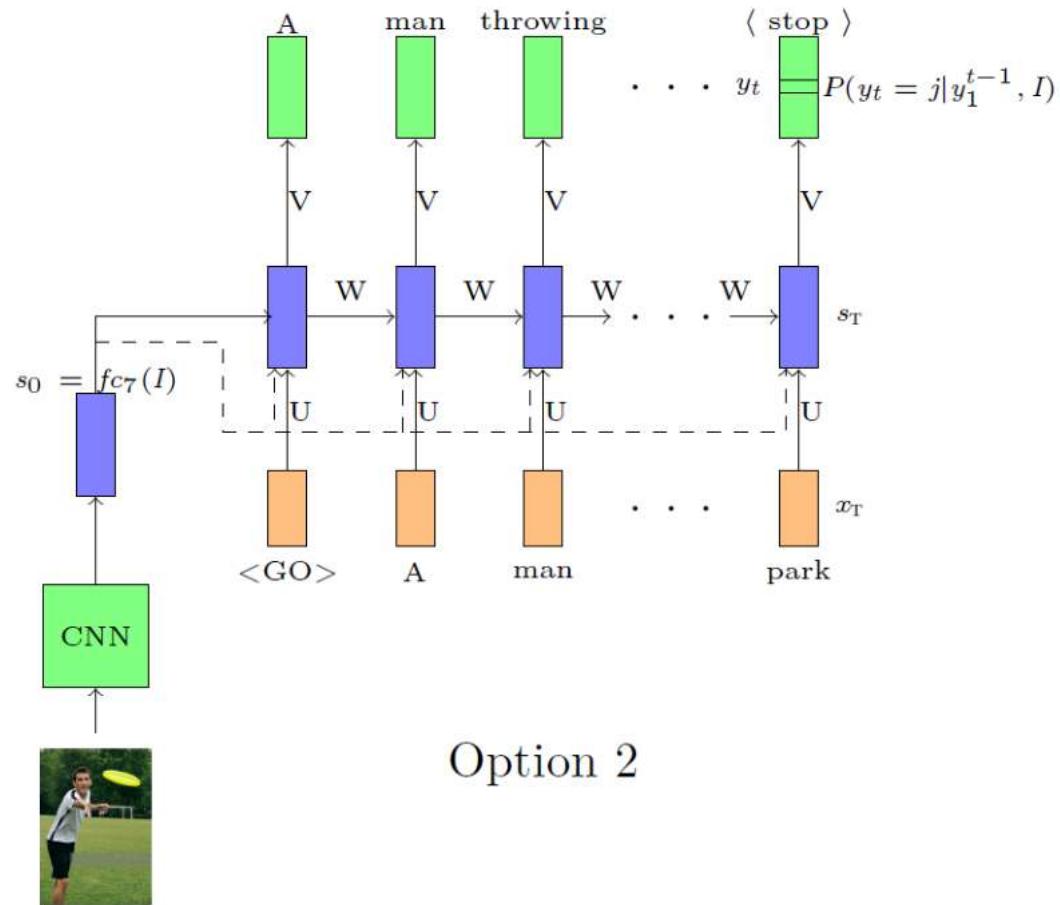


Option 2: Another more explicit way of doing this is to compute

$$s_t = RNN(s_{t-1}, [x_t, f_{c7}(I)])$$

In other words we are explicitly using $f_{c7}(I)$ to compute s_t and hence $P(y_t = j)$

Encoder Decoder Models : Introduction



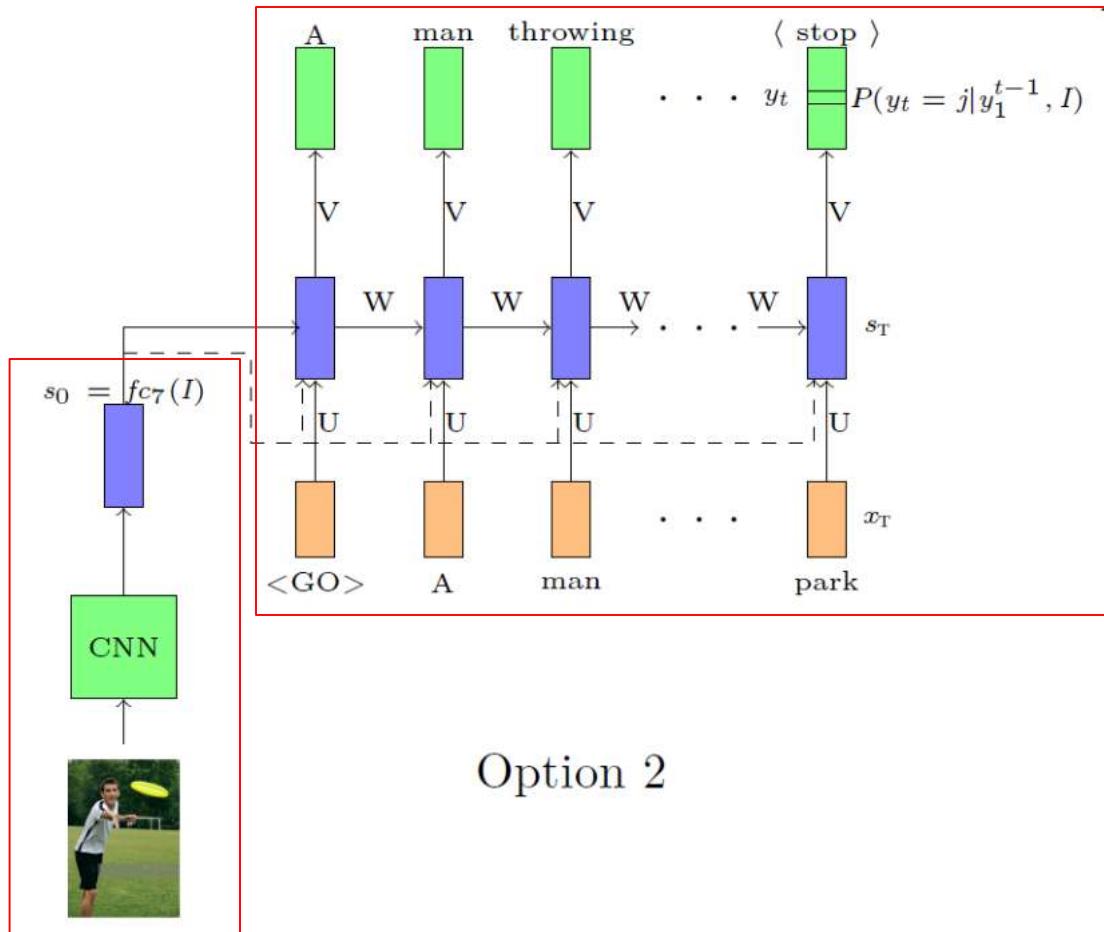
Option 2: Another more explicit way of doing this is to compute

$$s_t = RNN(s_{t-1}, [x_t, f_{c7}(I)])$$

In other words we are explicitly using $f_{c7}(I)$ to compute s_t and hence $P(y_t = j)$

The o/p of CNN is concatenated with the input embedding and then fed to the RNN as input at each time step.

Encoder Decoder Models : Introduction



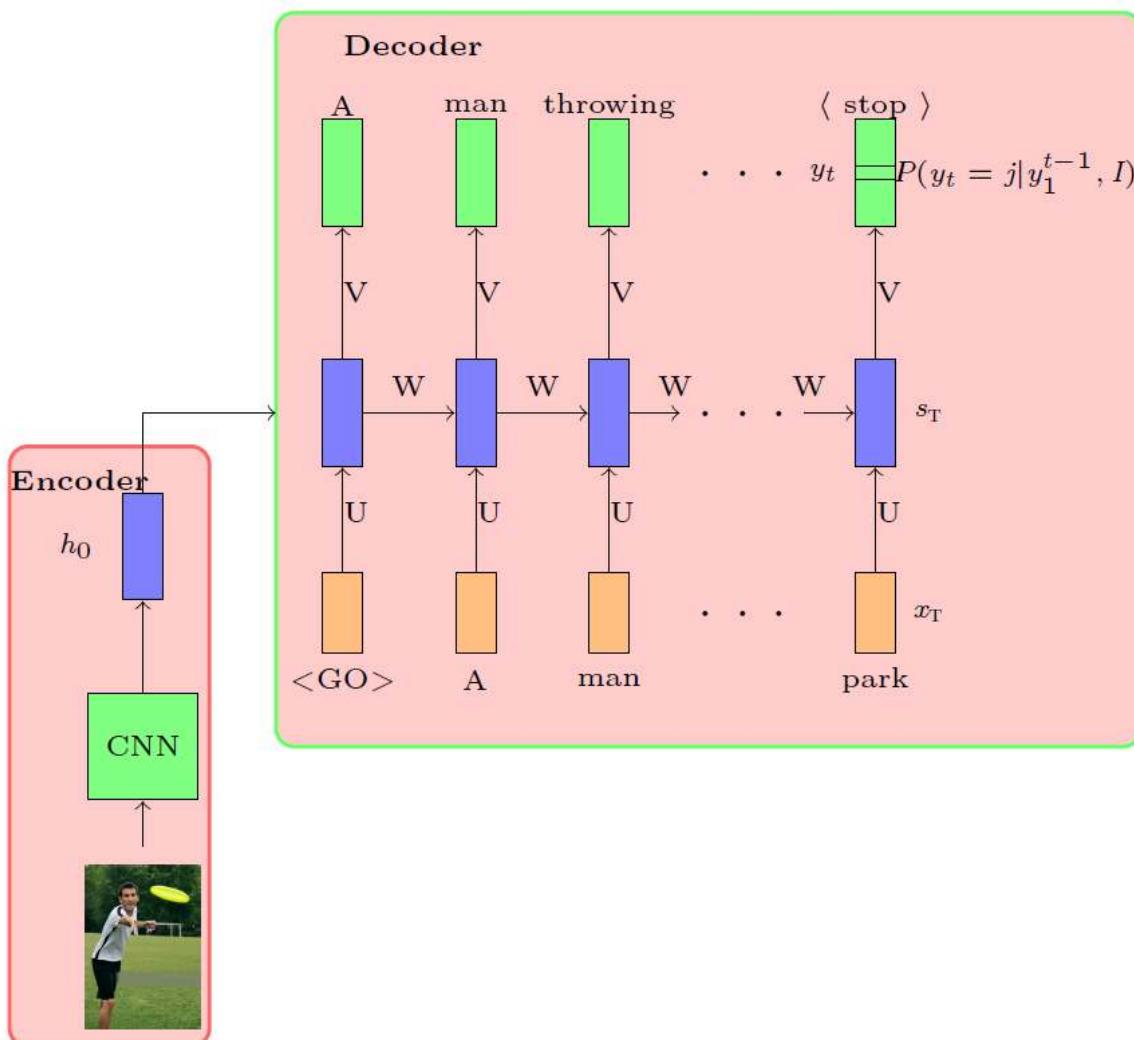
Option 2: Another more explicit way of doing this is to compute

$$s_t = RNN(s_{t-1}, [x_t, f_{c7}(I)])$$

In other words we are explicitly using $f_{c7}(I)$ to compute s_t and hence $P(y_t = j)$

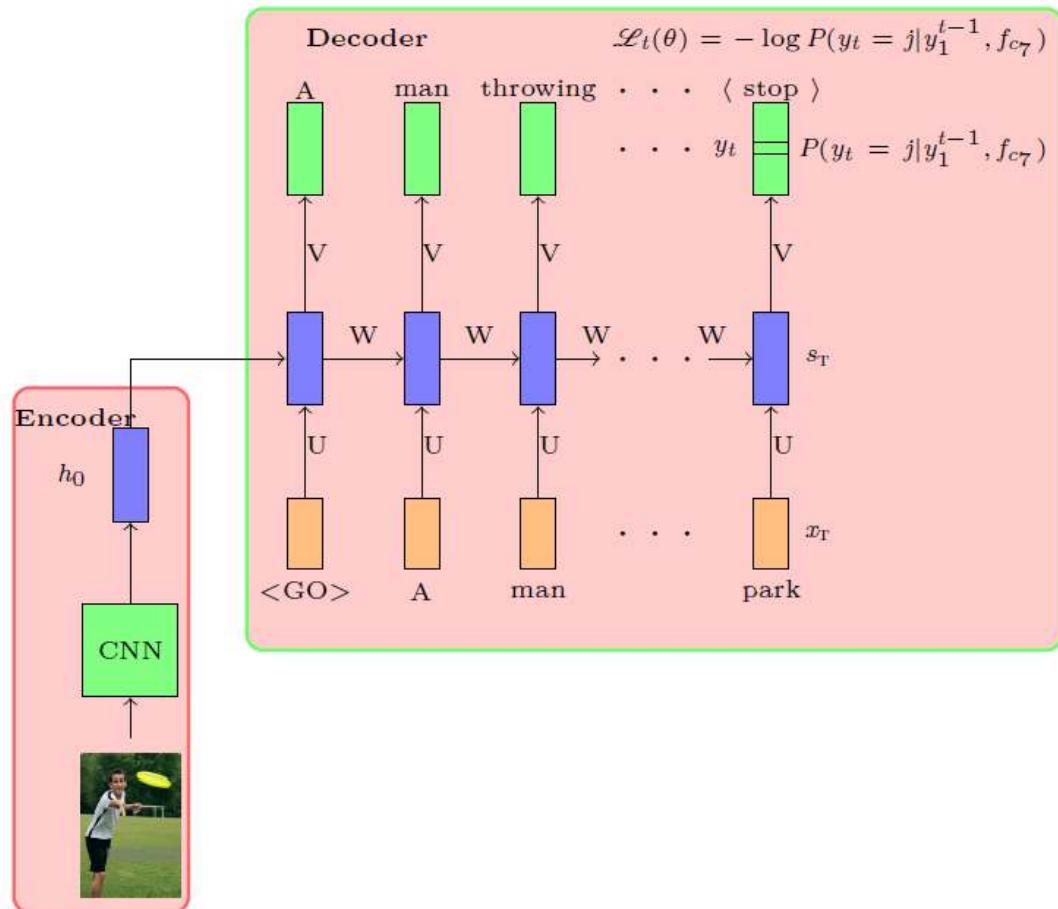
The o/p of CNN is concatenated with the input embedding and then fed to the RNN as input at each time step.

Encoder Decoder Models : Introduction



- This is typical **encoder decoder** architecture

Applications of Encoder Decoder Models: Image Captioning

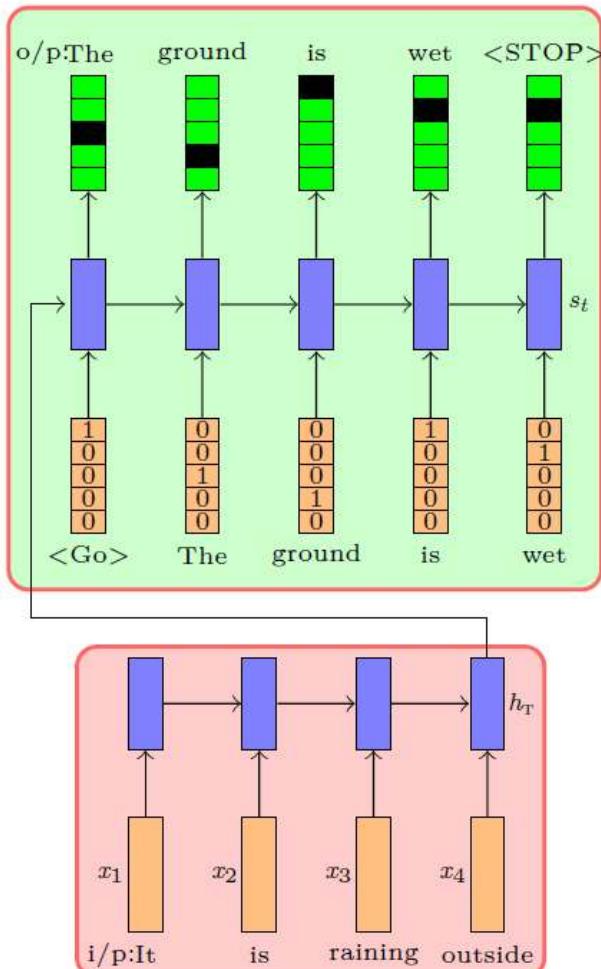


- **Task:** Image captioning
- **Data:** $\{x_i = \text{image}_i, y_i = \text{caption}_i\}_{i=1}^N$
- **Model:**
 - **Encoder:**
$$s_0 = \text{CNN}(x_i)$$
 - **Decoder:**
$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
$$P(y_t | y_1^{t-1}, I) = \text{softmax}(Vs_t + b)$$
- **Parameters:** $U_{dec}, V, W_{dec}, W_{conv}, b$
- **Loss:**
$$\mathcal{L}(\theta) = \sum_{i=1}^T \mathcal{L}_t(\theta) = - \sum_{t=1}^T \log P(y_t = \ell_t | y_1^{t-1}, I)$$
- **Algorithm:** Gradient descent with backpropagation

Represents the input embedding of the output obtained at time step $t-1$

Applications of Encoder Decoder Models: Textual Entailment

o/p : The ground is wet



i/p : It is raining outside

- **Task:** Textual entailment
- **Data:** $\{x_i = \text{premise}_i, y_i = \text{hypothesis}_i\}_{i=1}^N$

- **Model (Option 1):**

- **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

- **Decoder:**

$$s_0 = h_T \quad (T \text{ is length of input})$$

$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$

$$P(y_t | y_1^{t-1}, x) = \text{softmax}(V s_t + b)$$

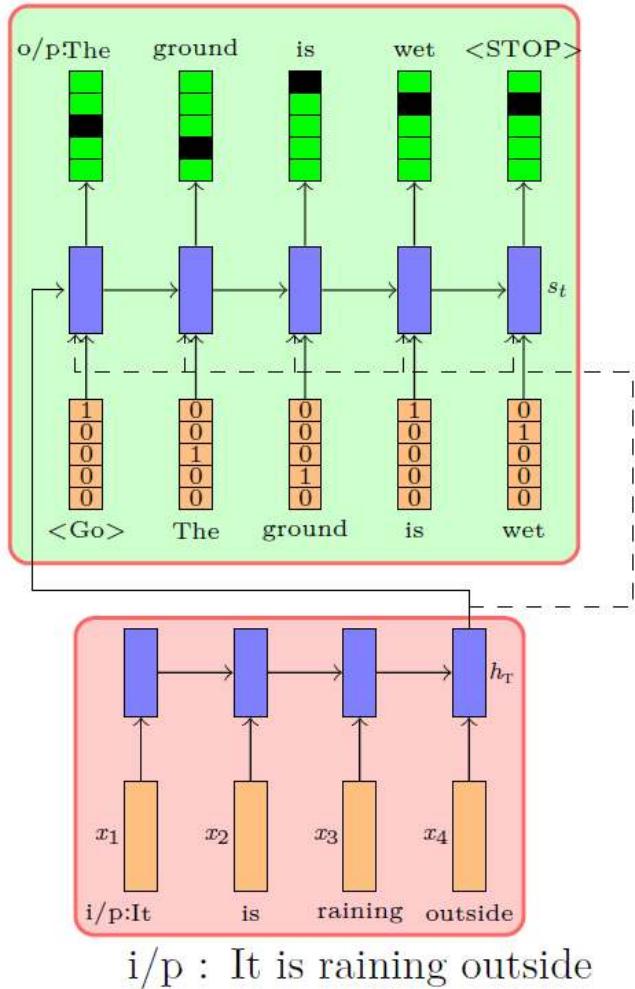
- **Parameters:** $U_{dec}, V, W_{dec}, U_{enc}, W_{enc}, b$

- **Loss:**

$$\mathcal{L}(\theta) = \sum_{i=1}^T \mathcal{L}_t(\theta) = - \sum_{t=1}^T \log P(y_t = \ell_t | y_1^{t-1}, x)$$

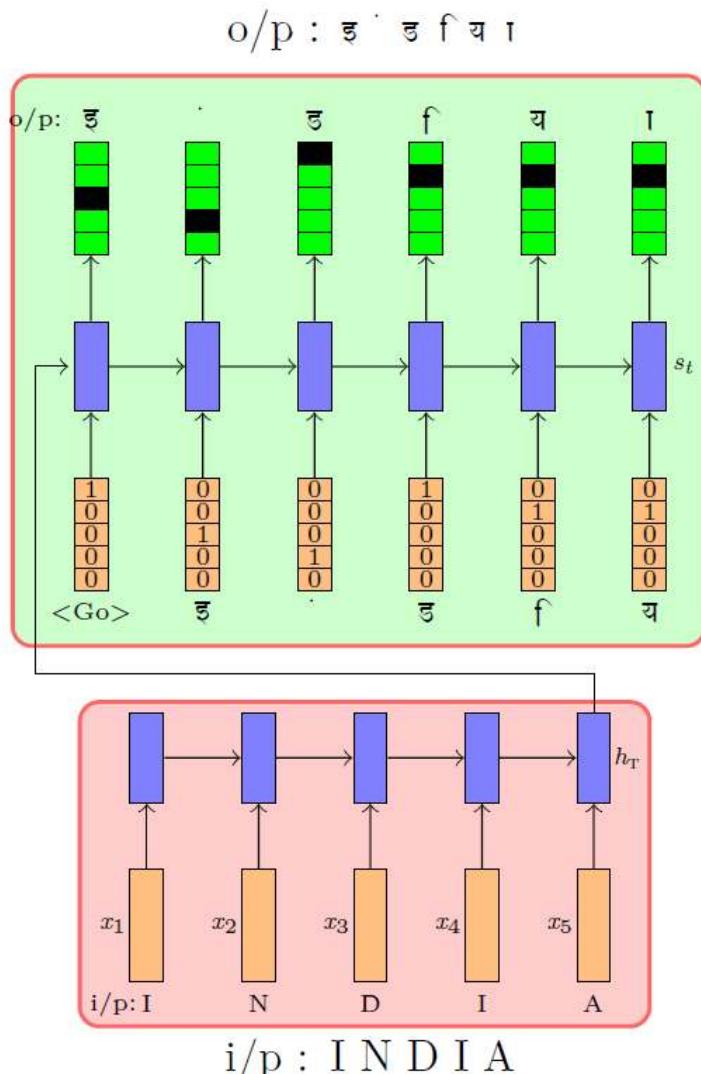
Applications of Encoder Decoder Models: Textual Entailment

o/p : The ground is wet



- **Task:** Textual entailment
- **Data:** $\{x_i = \text{premise}_i, y_i = \text{hypothesis}_i\}_{i=1}^N$
- **Model (Option 2):**
 - **Encoder:**
$$h_t = RNN(h_{t-1}, x_{it})$$
 - **Decoder:**
$$s_0 = h_T \quad (T \text{ is length of input})$$
$$s_t = RNN(s_{t-1}, [h_T, e(\hat{y}_{t-1})])$$
$$P(y_t | y_1^{t-1}, x) = \text{softmax}(V s_t + b)$$
- **Parameters:** $U_{dec}, V, W_{dec}, U_{enc}, W_{enc}, b$
- **Loss:**
$$\mathcal{L}(\theta) = \sum_{i=1}^T \mathcal{L}_t(\theta) = - \sum_{t=1}^T \log P(y_t = \ell_t | y_1^{t-1}, x)$$
- **Algorithm:** Gradient descent with backpropagation

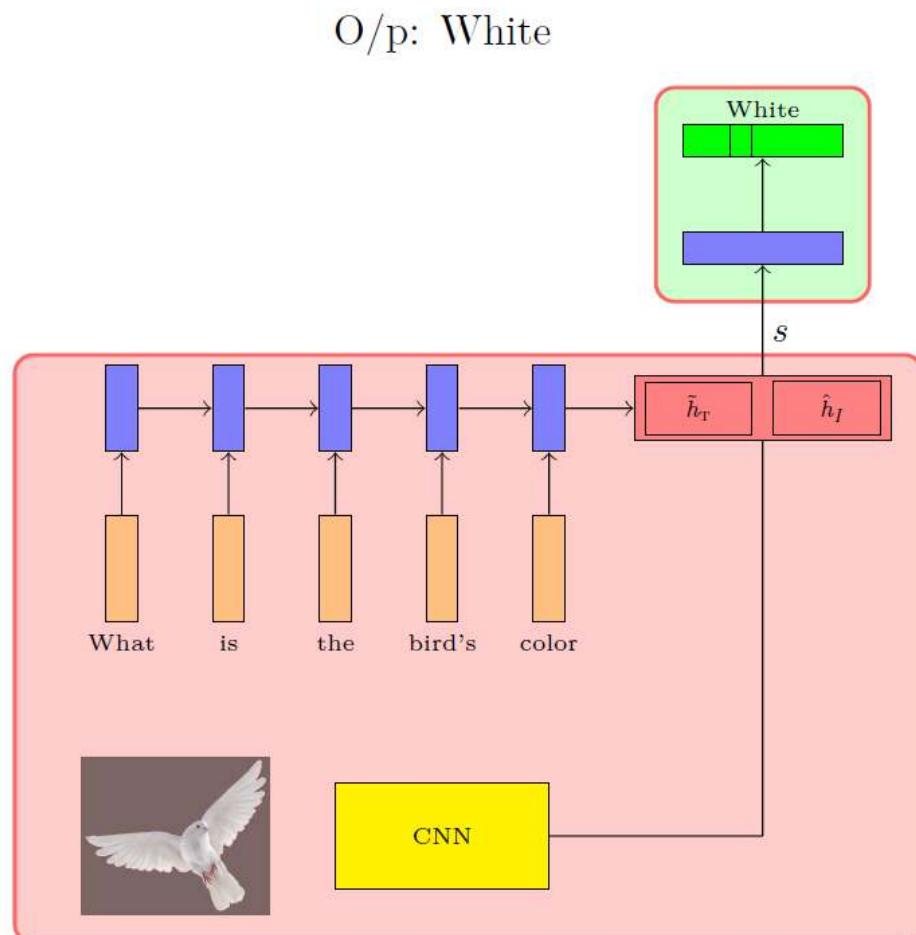
Applications of Encoder Decoder Models: Transliteration



- **Task:** Transliteration
- **Data:** $\{x_i = \text{srcword}_i, y_i = \text{tgtword}_i\}_{i=1}^N$
- **Model**
 - **Encoder:**
$$h_t = RNN(h_{t-1}, x_t)$$
 - **Decoder:**
$$s_0 = h_T \quad (T \text{ is length of input})$$
$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
$$P(y_t|y_1^{t-1}, x) = \text{softmax}(V s_t + b)$$
- **Parameters:** $U_{dec}, V, W_{dec}, U_{enc}, W_{enc}, b$
- **Loss:**
$$\mathcal{L}(\theta) = \sum_{i=1}^T \mathcal{L}_t(\theta) = - \sum_{t=1}^T \log P(y_t = \ell_t | y_1^{t-1}, x)$$
- **Algorithm:** Gradient descent with backpropagation

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

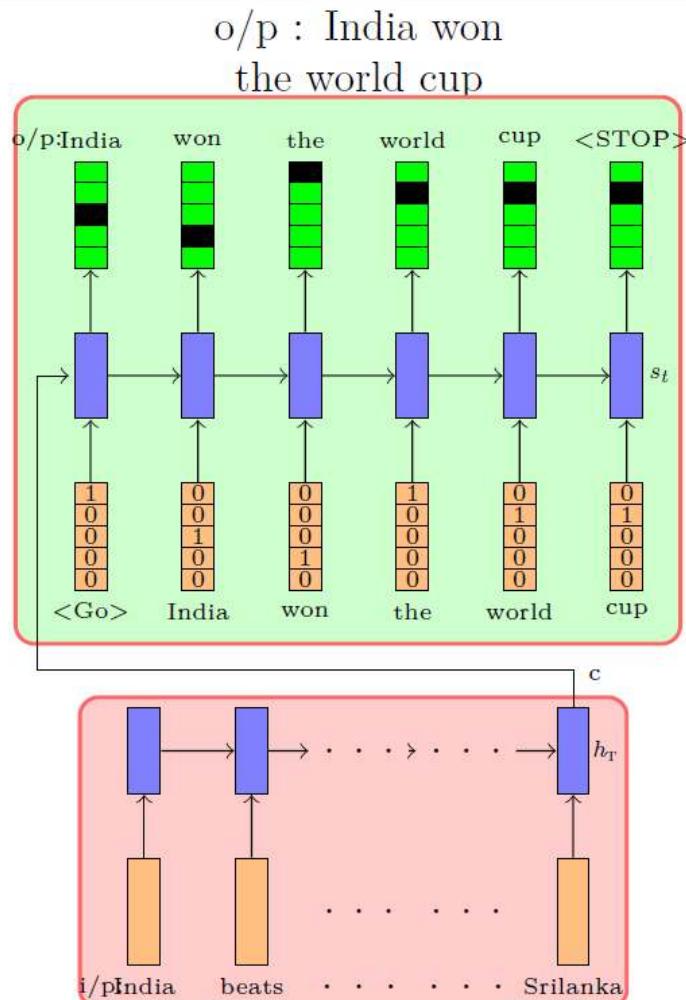
Applications of Encoder Decoder Models: Image Question Answering



Question: What
is the bird's color

- **Task:** Image Question Answering
- **Data:** $\{x_i = \{I, q\}_i, y_i = Answer_i\}_{i=1}^N$
- **Model:**
 - **Encoder:**
$$\hat{h}_I = CNN(I), \tilde{h}_t = RNN(\tilde{h}_{t-1}, q_{it})$$
$$s = [\tilde{h}_T; \hat{h}_I]$$
 - **Decoder:**
$$P(y|q, I) = softmax(Vs + b)$$
- **Parameters:** $V, b, U_q, W_q, W_{conv}, b$
- **Loss:**
$$\mathcal{L}(\theta) = -\log P(y = \ell|I, q)$$
- **Algorithm:** Gradient descent with backpropagation

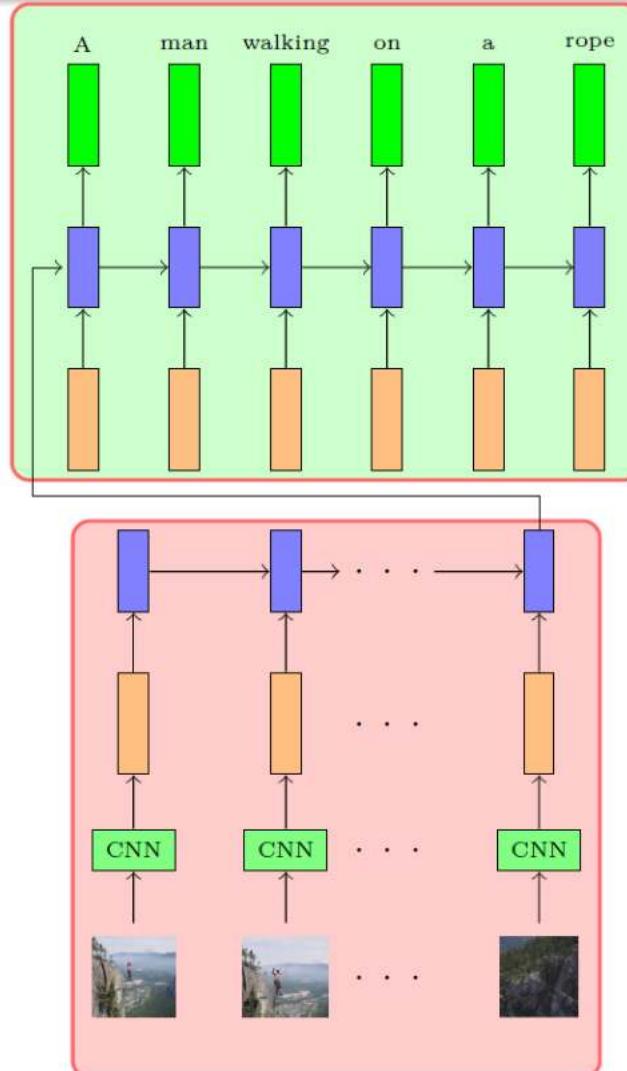
Applications of Encoder Decoder Models: Document Summarization



i/p : India beats Srilanka to win ICC WC 2011.
Dhoni and Gambhir's half centuries help beat SL

- **Task:** Document Summarization
- **Data:** $\{x_i = Document_i, y_i = Summary_i\}_{i=1}^N$
- **Model:**
 - **Encoder:**
$$h_t = RNN(h_{t-1}, x_{it})$$
 - **Decoder:**
$$s_0 = h_T$$
$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$
$$P(y_t|y_1^{t-1}, x) = softmax(Vs_t + b)$$
- **Parameters:** $U_{dec}, V, W_{dec}, U_{enc}, W_{enc}, b$
- **Loss:**
$$\mathcal{L}(\theta) = \sum_{i=1}^T \mathcal{L}_t(\theta) = - \sum_{t=1}^T \log P(y_t = \ell_t | y_1^{t-1}, x)$$
- **Algorithm:** Gradient descent with backpropagation

Applications of Encoder Decoder Models: Video Captioning



- **Task:** Video Captioning

- **Data:** $\{x_i = \text{video}_i, y_i = \text{desc}_i\}_{i=1}^N$

- **Model:**

- **Encoder:**

$$h_t = RNN(h_{t-1}, CNN(x_{it}))$$

- **Decoder:**

$$s_0 = h_T$$

$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$

$$P(y_t|y_1^{t-1}, x) = \text{softmax}(Vs_t + b)$$

- **Parameters:** $U_{dec}, W_{dec}, V, b, W_{conv}, U_{enc}, W_{enc}, b$

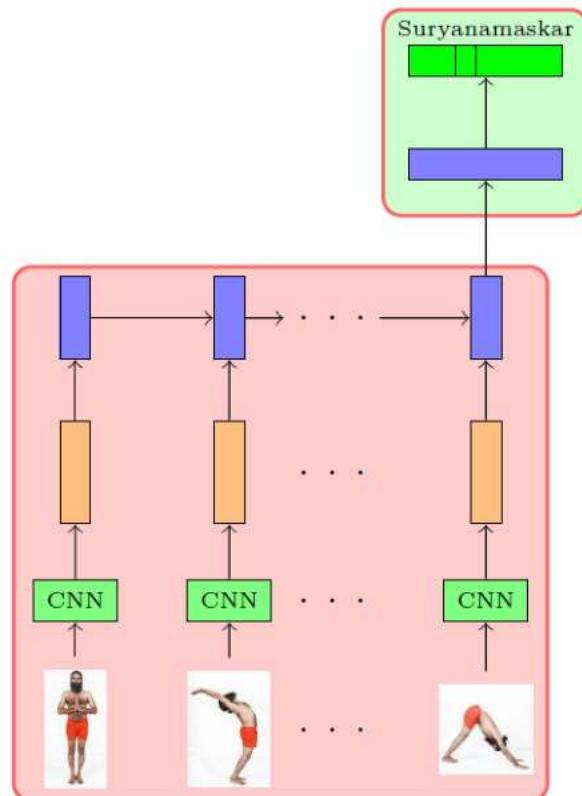
- **Loss:**

$$\mathcal{L}(\theta) = \sum_{i=1}^T \mathcal{L}_t(\theta) = - \sum_{t=1}^T \log P(y_t = \ell_t | y_1^{t-1}, x)$$

- **Algorithm:** Gradient descent with backpropagation

Applications of Encoder Decoder Models: Video Classification

o/p: Surya Namaskar



- **Task:** Video Classification
- **Data:** $\{x_i = \text{Video}_i, y_i = \text{Activity}_i\}_{i=1}^N$

- **Model:**

- **Encoder:**

$$h_t = RNN(h_{t-1}, CNN(x_{it}))$$

- **Decoder:**

$$s = h_T$$

$$P(y|I) = \text{softmax}(Vs + b)$$

- **Parameters:** $V, b, W_{conv}, U_{enc}, W_{enc}, b$

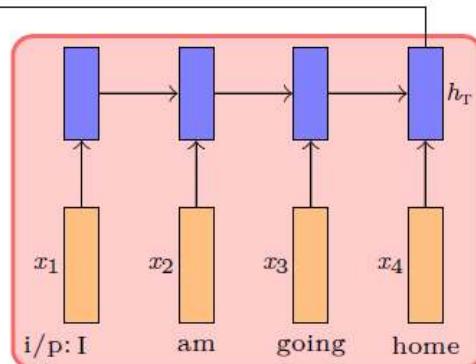
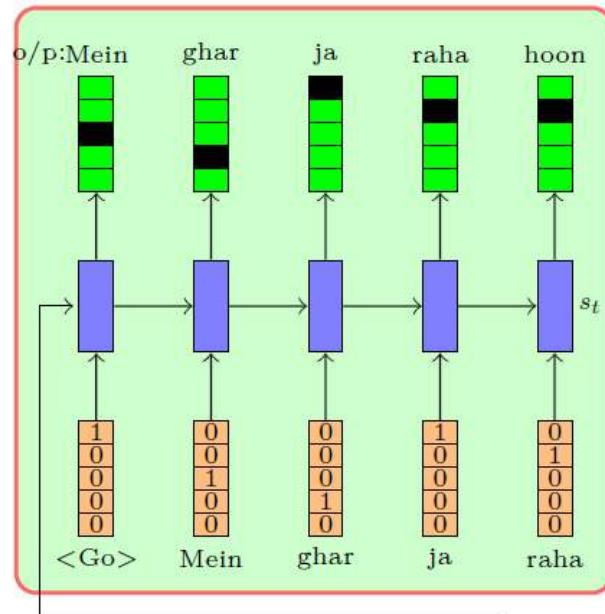
- **Loss:**

$$\mathcal{L}(\theta) = -\log P(y = \ell | \text{Video})$$

- **Algorithm:** Gradient descent with backpropagation

Applications of Encoder Decoder Models: Machine Translation

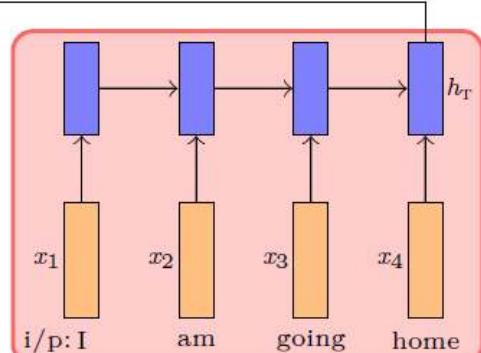
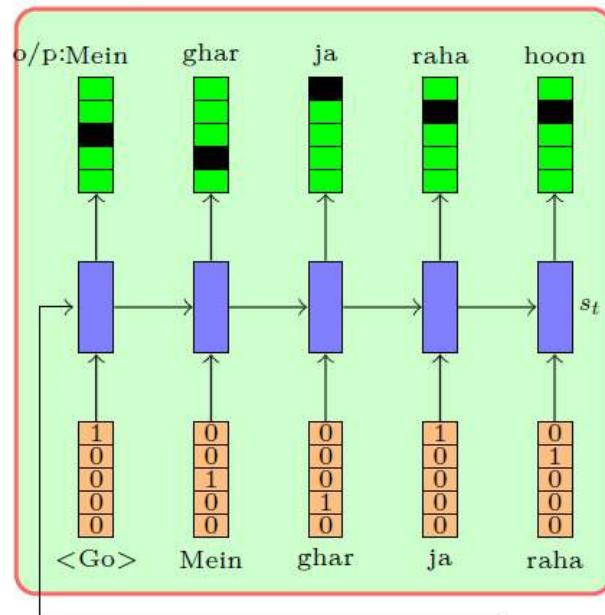
o/p : Mein ghar ja raha hoon



i/p : I am going home

Applications of Encoder Decoder Models: Machine Translation

o/p : Mein ghar ja raha hoon



i/p : I am going home

- **Task:** Machine translation
- **Data:** $\{x_i = \text{source}_i, y_i = \text{target}_i\}_{i=1}^N$

- **Model (Option 1):**

- **Encoder:**

$$h_t = RNN(h_{t-1}, x_{it})$$

- **Decoder:**

$$s_0 = h_T \quad (T \text{ is length of input})$$

$$s_t = RNN(s_{t-1}, e(\hat{y}_{t-1}))$$

$$P(y_t|y_1^{t-1}, x) = \text{softmax}(V s_t + b)$$

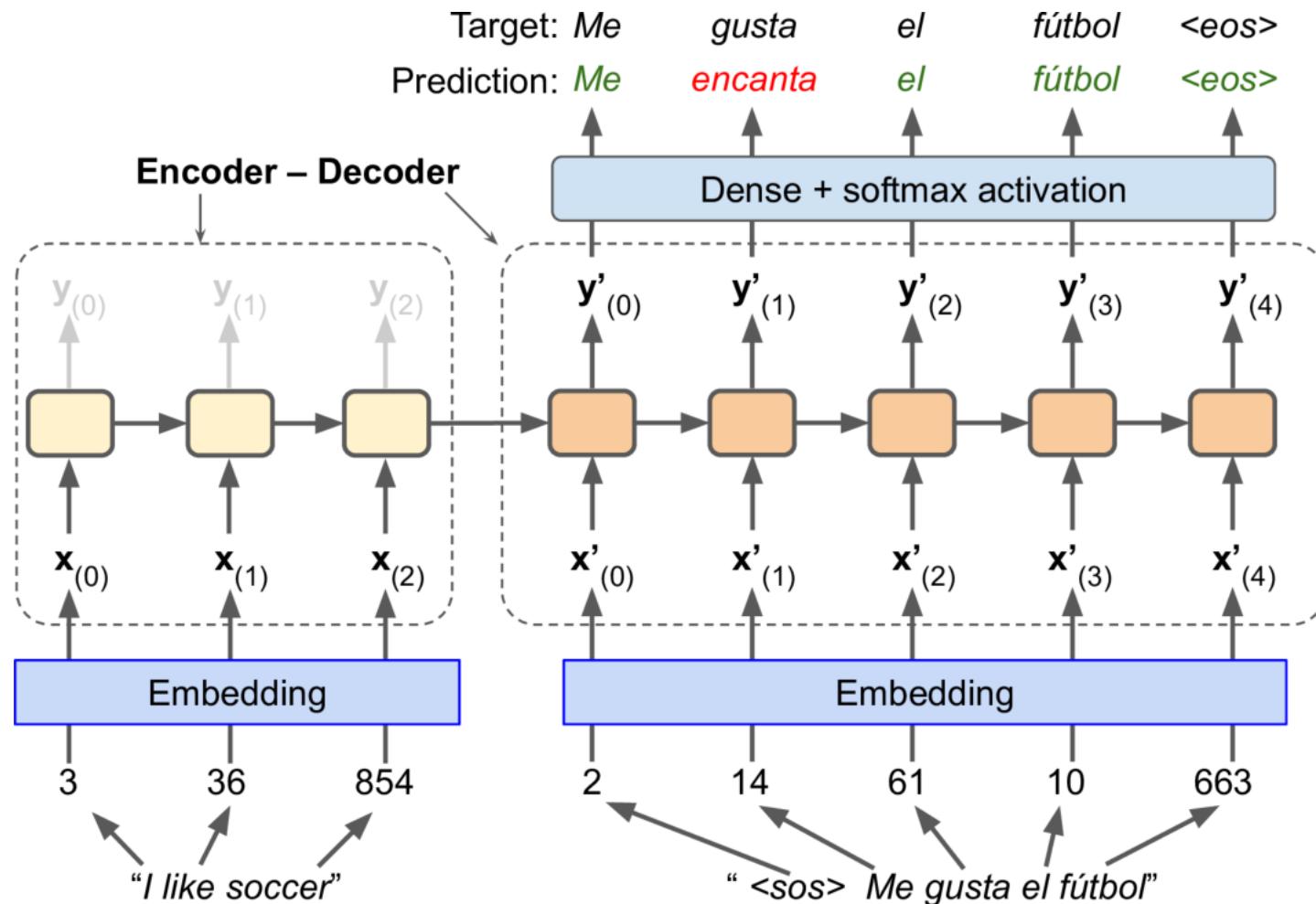
- **Parameters:** $U_{dec}, V, W_{dec}, U_{enc}, W_{enc}, b$

- **Loss:**

$$\mathcal{L}(\theta) = \sum_{i=1}^T \mathcal{L}_t(\theta) = - \sum_{t=1}^T \log P(y_t = \ell_t | y_1^{t-1}, x)$$

- **Algorithm:** Gradient descent with backpropagation

Applications of Encoder Decoder Models: Machine Translation



Machine Translation as Conditional Learning Problem

- $Y = P(y_1, y_2, y_3 \dots, y_n | x_1, x_2, \dots, x_m)$
- Jane visitr l'Afrique en Septembre (x:French)
 - Jane is visiting Africa in September (y: English)
 - Jane is going to be visiting Africa in September September (y: English)
 - In September, Jane will visit Africa.
- $Y_{\text{argmax}(y_1 \dots y_n)} = P(y_1, y_2, y_3 \dots, y_n | x_1, x_2, \dots, x_m)$

Beam Search

“Comment vas-tu?” – How are you?
“Comment vas-tu jouer?” – How will you play?
Model could output “How will you?”

- Since model greedily outputs the most likely word at each step, it ended with suboptimal translation
- Can model go back and correct mistakes?
- **Beam Search**
 - Keeps track of a short list of the k most promising sentences
 - Parameter k is beam width
 - At each decoder step it tries to extend them by 1 word
- **Step 1:** First word could be “How” (Prob: 75%), “what” or “you”
- **Step 2:** 3 copies of model made to find the next word
- **Step 3:** First model will look for next word after “How” by computing conditional probabilities
 - Output could be “will (Prob:36%) “are”, “do”,
- **Step 4:** Compute the probabilities of each of the two word sentences the model considered and so on
- Advantage is that good translation for fairly short sentences can be obtained
- Disadvantage – really bad at translating long sentences.
- Due to limited short term memory, led to **ATTENTION MECHANISM**

Attention Mechanism (Bahdanau et al , 2014)

- Technique that allowed the decoder to focus on the appropriate words at each time steps
- Ensures that the path to from an input word to its translation is much shorter
- For instance , at the time step where the decoder needs to output the word 'lait' it focusses on the word "milk"
- Path from input word to translation is smaller, so not affected by the short term limitations of RNNS
- **Visual Attention**
 - For example : Generating image captions using visual attentions
 - CNN processes image and outputs some feature maps
 - Then decoder RNN with attention generates the caption , one word at a time
- **Leads to Explainability (Ribeiro et al. 2016)**
 - **What led the model to produce its output**
 - **Which leads to fairness in results**
 - Google apologizes after its Vision AI produced racist results



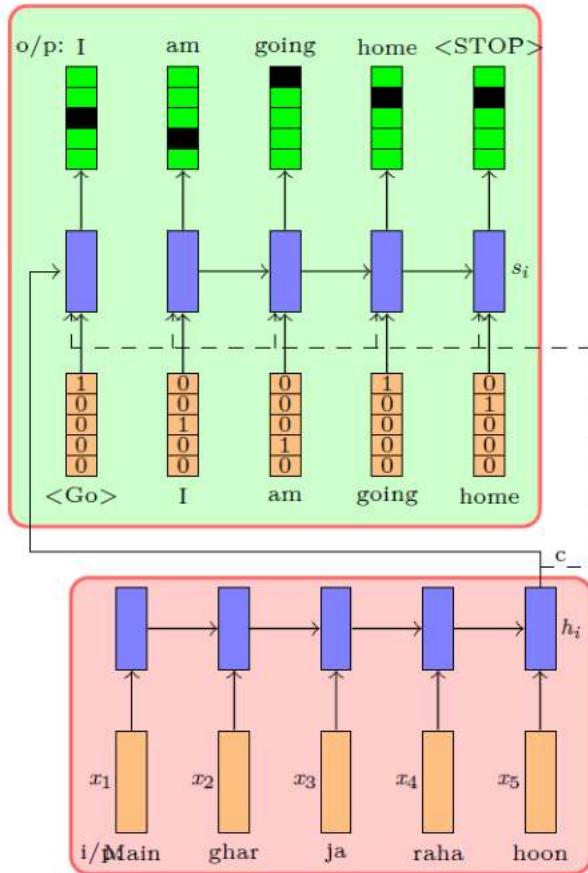
Attention Mechanism (Bahdanau et al , 2014)

- Attention is proposed as a method to both align and translate.
 - **Alignment**
 - is the problem in machine translation that identifies which parts of the input sequence are relevant to each word in the output
 - **translation**
 - is the process of using the relevant information to select the appropriate output
- *“... we introduce an extension to the encoder–decoder model which learns to align and translate jointly. Each time the proposed model generates a word in a translation, it (soft-)searches for a set of positions in a source sentence where the most relevant information is concentrated. The model then predicts a target word based on the context vectors associated with these source positions and all the previous generated target words.”*
 - [Neural Machine Translation by Jointly Learning to Align and Translate, 2015.](#)

Attention Mechanism: Introduction

Task: Machine translation

o/p : I am going home



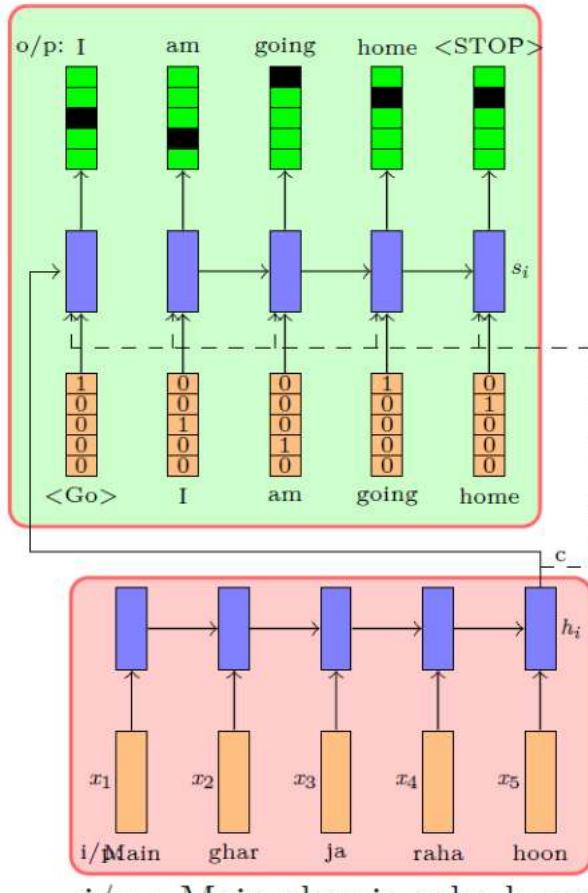
i/p : Main ghar ja raha hoon

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Attention Mechanism: Introduction

Task: Machine translation

o/p : I am going home

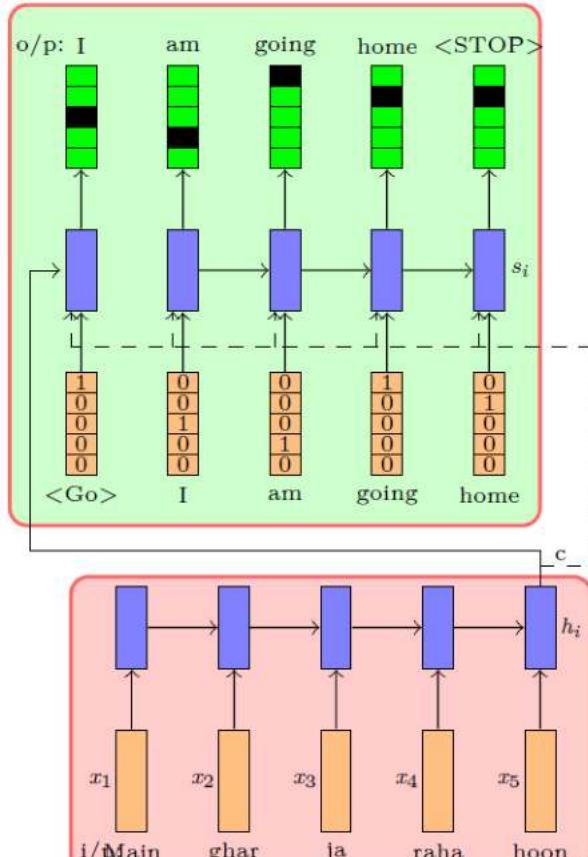


- In a typical encoder decoder network, each time step of the decoder uses the information obtained from the last time step of the encoder.

Attention Mechanism: Introduction

Task: Machine translation

o/p : I am going home



- In a typical encoder decoder network, each time step of the decoder uses the information obtained from the last time step of the encoder.
- However, the translation would be effective if the network could focus/or pay attention to specific input word that would contribute to the prediction.

Attention Mechanism: Introduction

- Consider the task of machine translation:

o/p : I am going home

i/p : Main ghar ja raha hoon

Attention Mechanism: Introduction

While predicting each word in the o/p we would like our model to mimic humans and focus on specific words in the i/p

o/p : I am going home
 $t_1 : [1 \ 0 \ 0 \ 0 \ 0]$

i/p : Main ghar ja raha hoon

Attention Mechanism: Introduction

While predicting each word in the o/p we would like our model to mimic humans and focus on specific words in the i/p

o/p : I **am** going home

t_1 : [1 0 0 0 0]

t_2 : [0 0 0 0 **1**]

i/p : Main ghar ja raha **hoon**

Attention Mechanism: Introduction

While predicting each word in the o/p we would like our model to mimic humans and focus on specific words in the i/p

o/p : I am **going** home

t_1 : [1 0 0 0 0]

t_2 : [0 0 0 0 1]

t_3 : [0 0 **0.5** **0.5** 0]

i/p : Main ghar **ja** **raha** hoon

Attention Mechanism: Introduction

o/p : I am going **home**

t_1 : [1 0 0 0 0]

t_2 : [0 0 0 0 1]

t_3 : [0 0 0.5 0.5 0]

t_4 : [0 **1** 0 0 0]

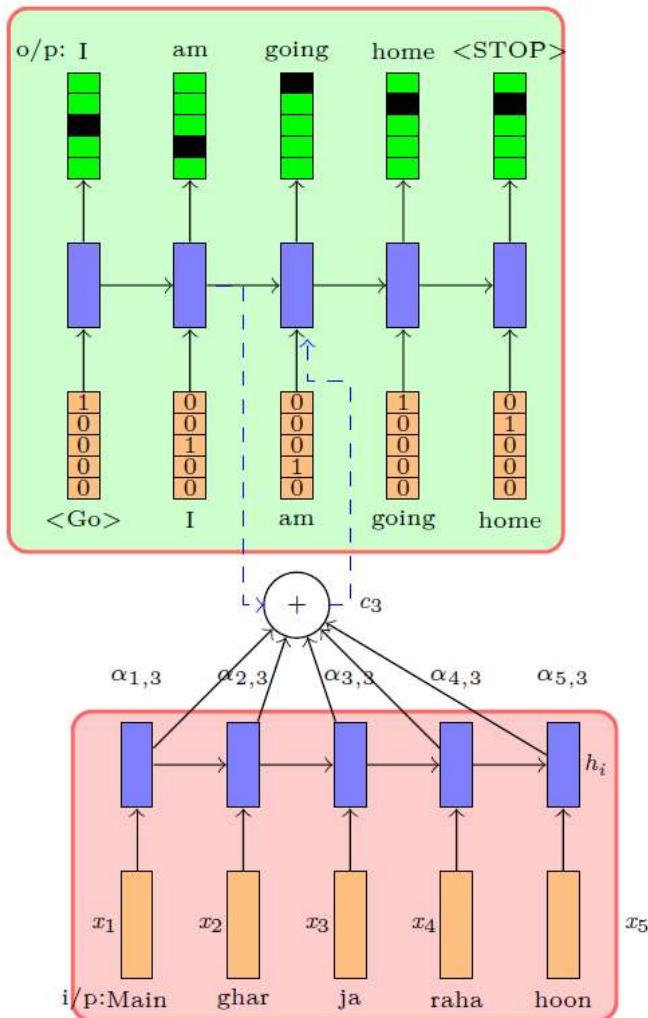
i/p : Main **ghar** ja raha hoon

- Essentially, at each time step, a distribution on the input words must be introduced.
- This distribution tells the model how much attention to pay to each input words at each time step.

Encoder Decoder with Attention Mechanism

Task: Machine translation

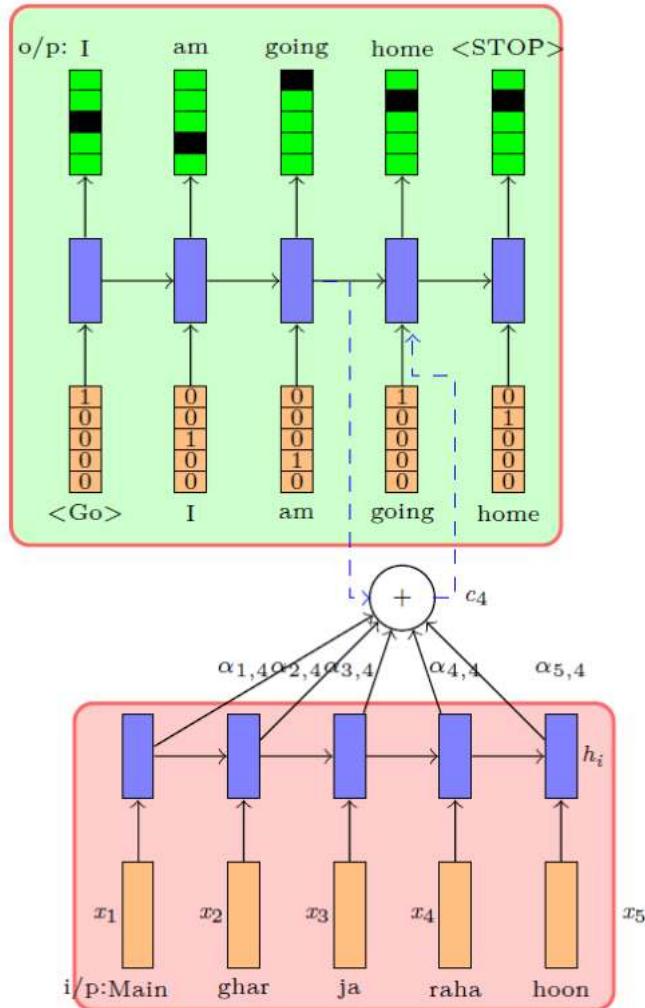
- To do this, we could just take a weighted average of the corresponding word representations and feed it to the decoder



Encoder Decoder with Attention Mechanism

Task: Machine translation

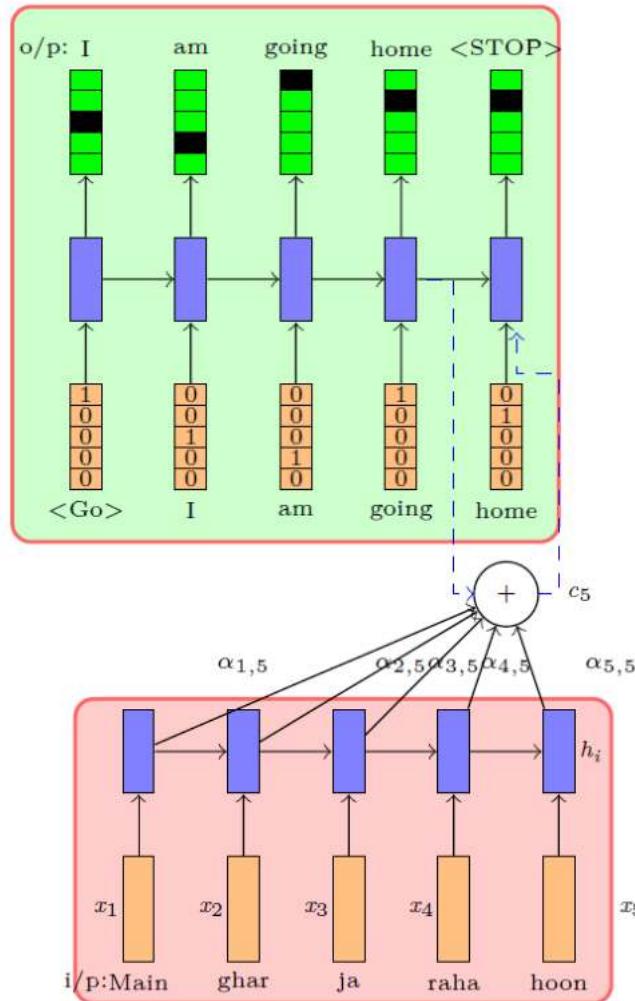
- To do this, we could just take a weighted average of the corresponding word representations and feed it to the decoder



Encoder Decoder with Attention Mechanism

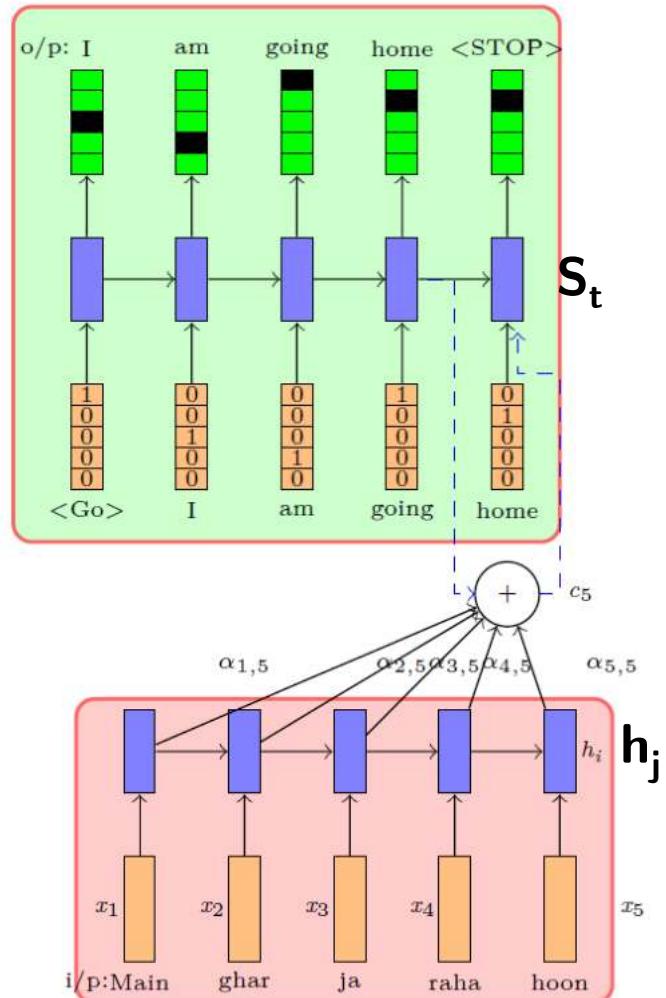
Task: Machine translation

- To do this, we could just take a weighted average of the corresponding word representations and feed it to the decoder



Encoder Decoder with Attention Mechanism

Task: Machine translation

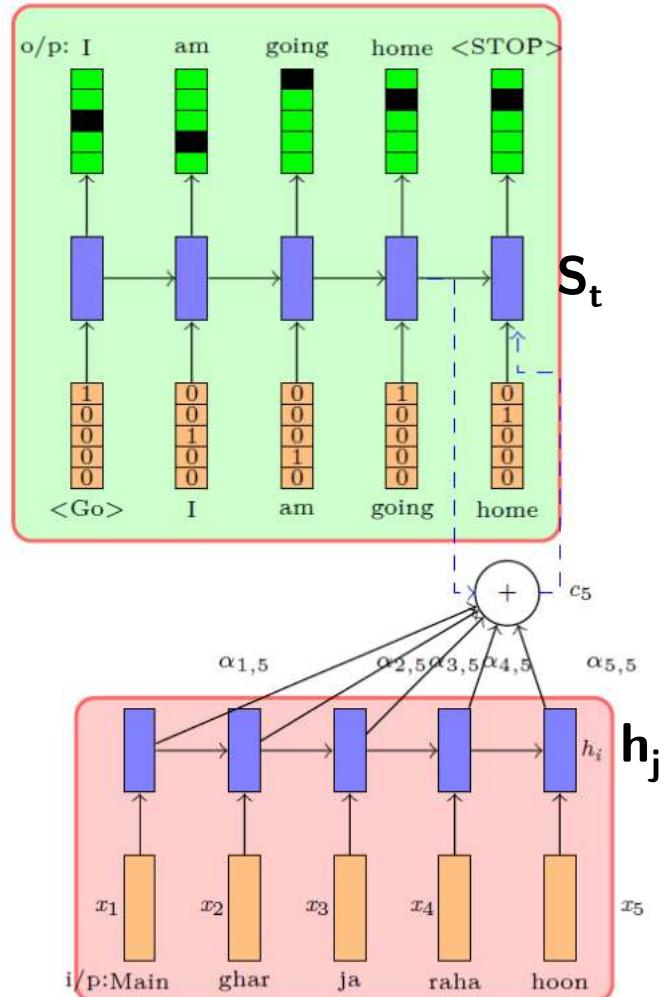


Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Encoder Decoder with Attention Mechanism

Task: Machine translation

- To enable the network to focus on certain data we define the following function:

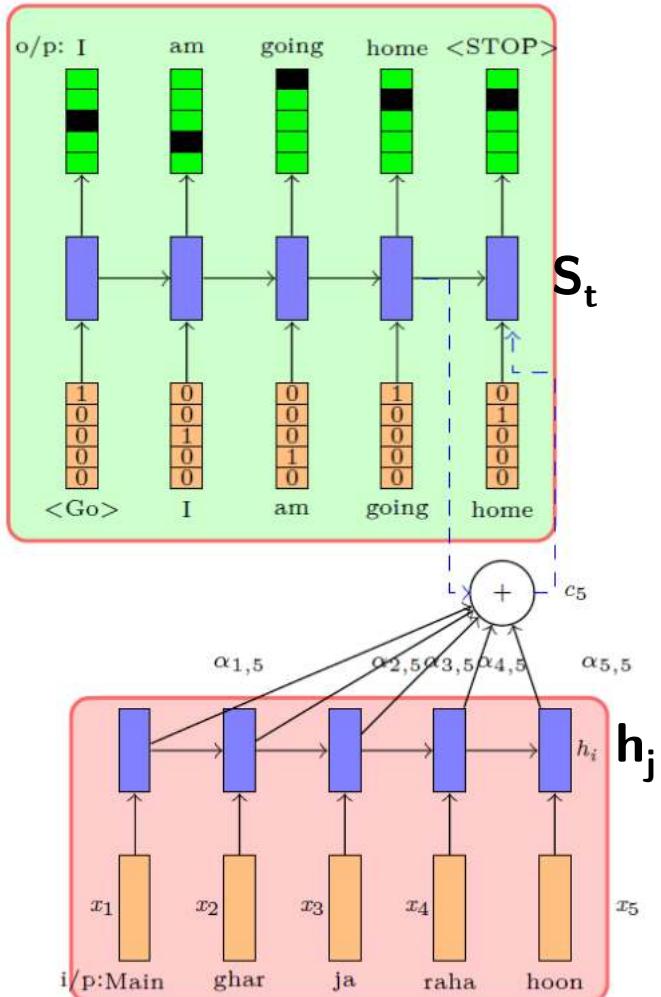


Encoder Decoder with Attention Mechanism

Task: Machine translation

- To enable the network to focus on certain data we define the following function:

$$e_{jt} = f_{ATT}(s_{t-1}, \mathbf{h}_j)$$



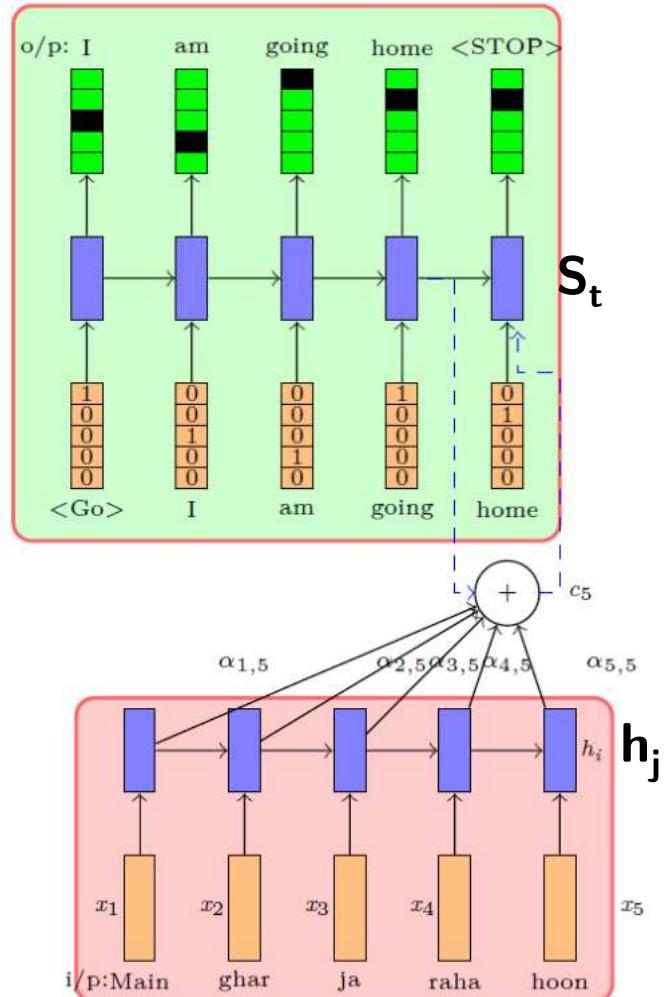
Encoder Decoder with Attention Mechanism

Task: Machine translation

- To enable the network to focus on certain data we define the following function:

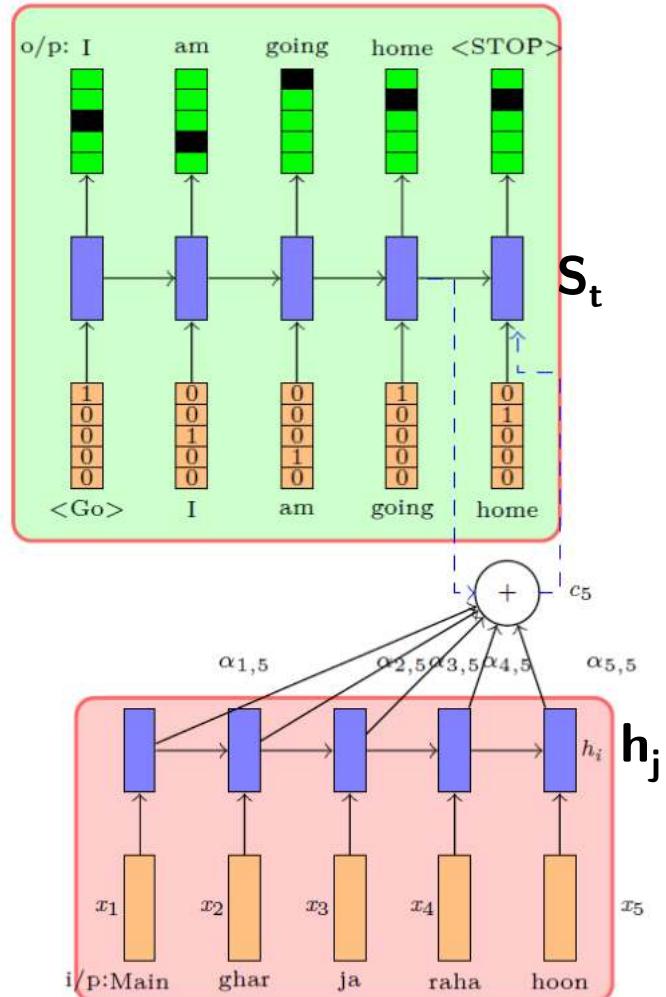
$$e_{jt} = f_{ATT}(s_{t-1}, h_j)$$

Can be considered as a separate feed forward network



Encoder Decoder with Attention Mechanism

Task: Machine translation



- To enable the network to focus on certain data we define the following function:

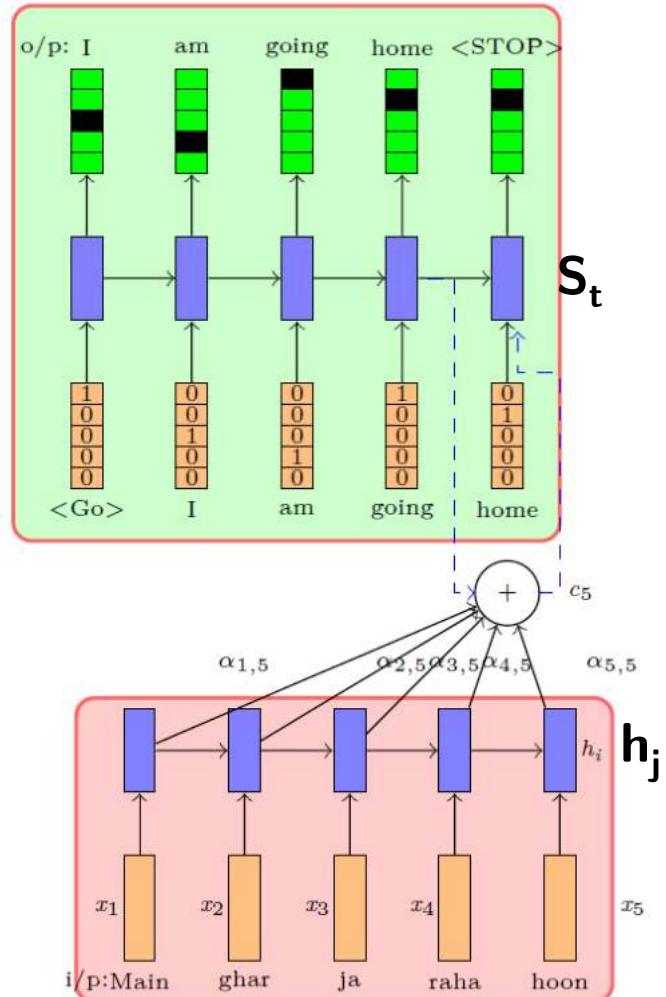
$$e_{jt} = f_{ATT}(s_{t-1}, h_j)$$

Can be considered as a separate feed forward network

- This quantity captures the importance of the j^{th} input word for decoding the t^{th} output word.

Encoder Decoder with Attention Mechanism

Task: Machine translation



- To enable the network to focus on certain data we define the following function:

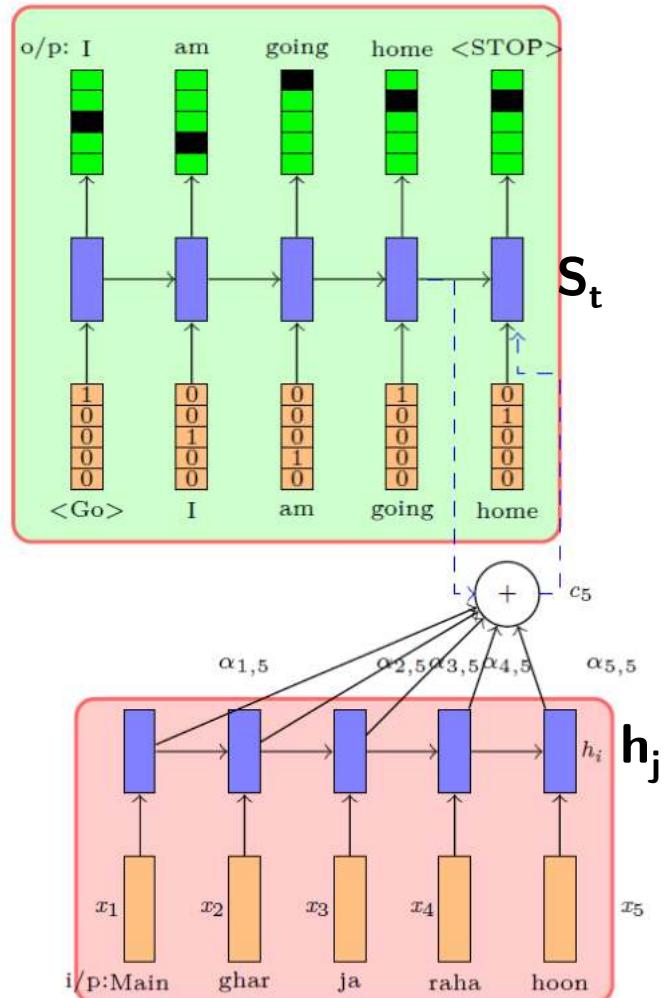
$$e_{jt} = f_{ATT}(s_{t-1}, h_j)$$

Can be considered as a separate feed forward network

- This quantity captures the importance of the j^{th} input word for decoding the t^{th} output word.
- We could compute α_{jt} by normalizing these weights using the softmax function.

Encoder Decoder with Attention Mechanism

Task: Machine translation



- To enable the network to focus on certain data we define the following function:

$$e_{jt} = f_{ATT}(s_{t-1}, h_j)$$

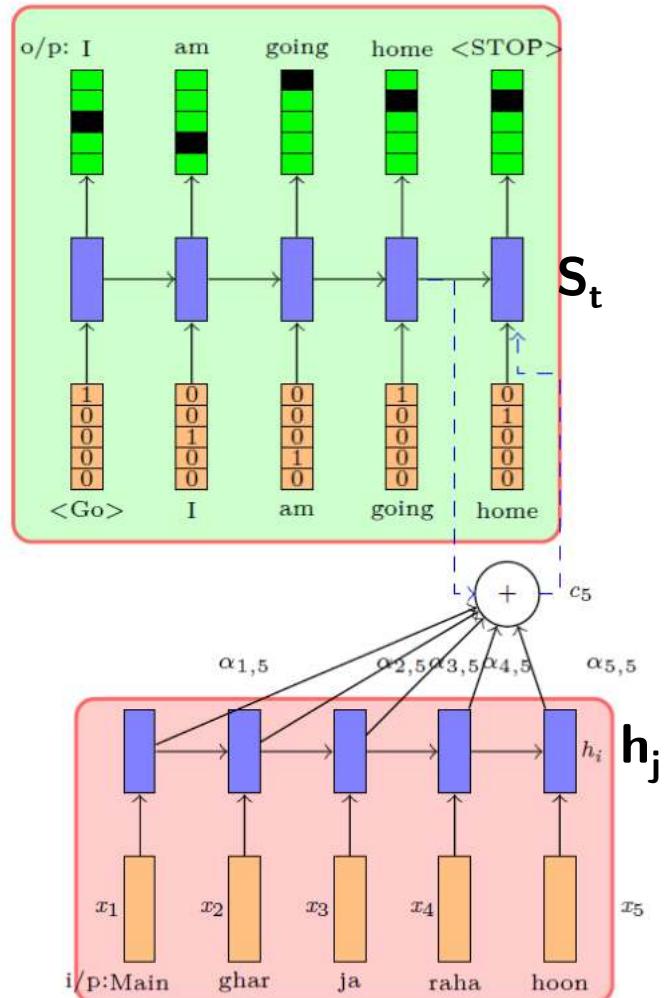
Can be considered as a separate feed forward network

- This quantity captures the importance of the jth input word for decoding the tth output word.
- We could compute α_{jt} by normalizing these weights using the softmax function.

$$\alpha_{jt} = \frac{\exp(e_{jt})}{\sum_{j=1}^M \exp(e_{jt})}$$

Encoder Decoder with Attention Mechanism

Task: Machine translation



- To enable the network to focus on certain data we define the following function:

$$e_{jt} = f_{ATT}(s_{t-1}, h_j)$$

Can be considered as a separate feed forward network

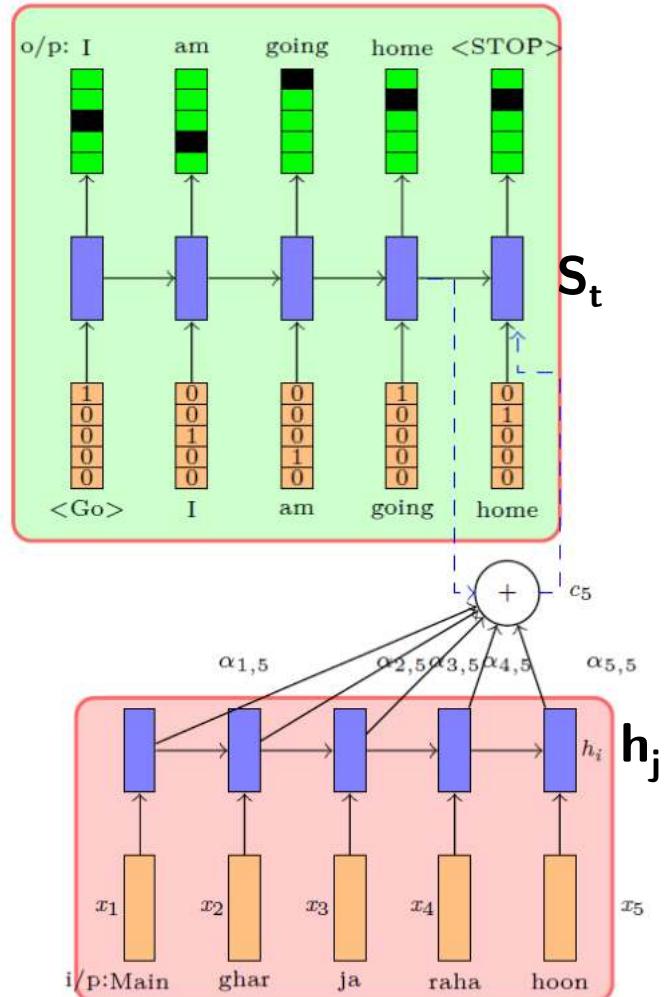
- This quantity captures the importance of the j^{th} input word for decoding the t^{th} output word.
- We could compute α_{jt} by normalizing these weights using the softmax function.

$$\alpha_{jt} = \frac{\exp(e_{jt})}{\sum_{j=1}^M \exp(e_{jt})}$$

- Where, α_{jt} denotes the probability of focusing on the j^{th} word to produce the t^{th} output word

Encoder Decoder with Attention Mechanism

Task: Machine translation

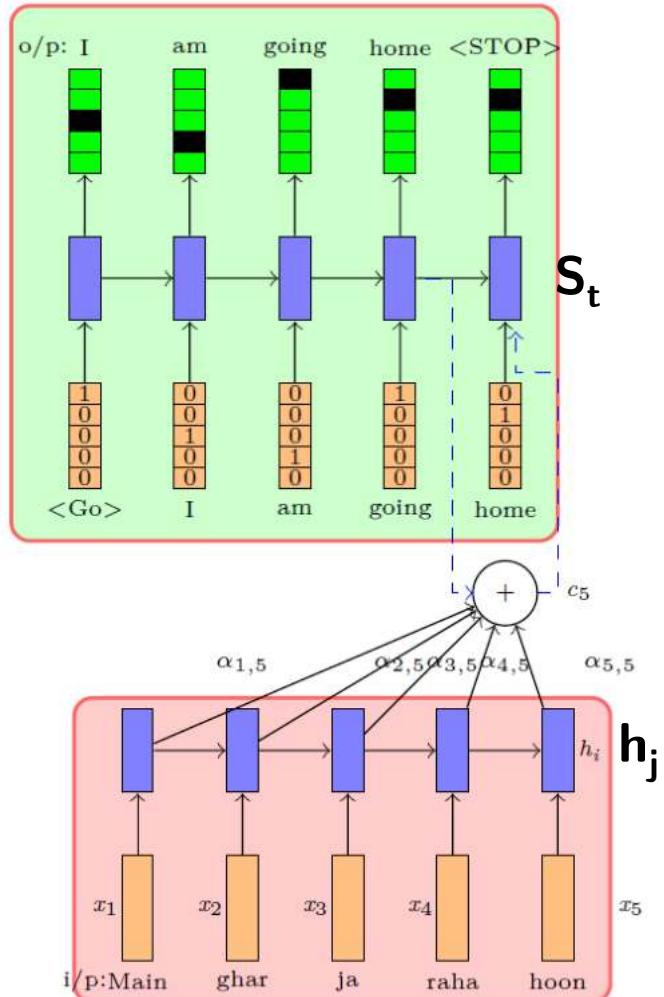


Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Encoder Decoder with Attention Mechanism

Task: Machine translation

- To enable the network to focus on certain data we define the following function:



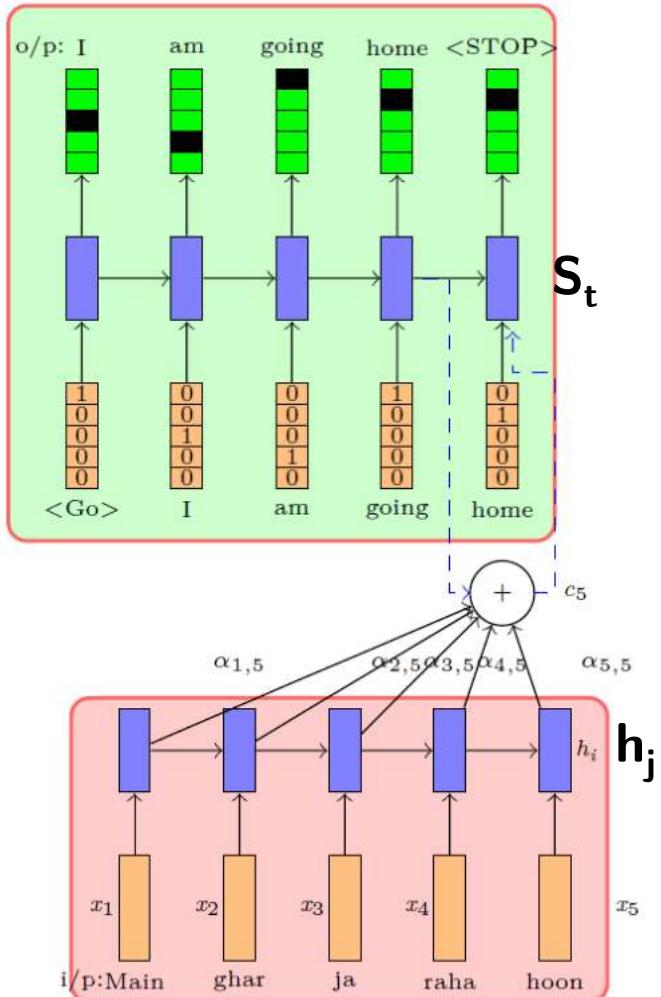
Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Encoder Decoder with Attention Mechanism

Task: Machine translation

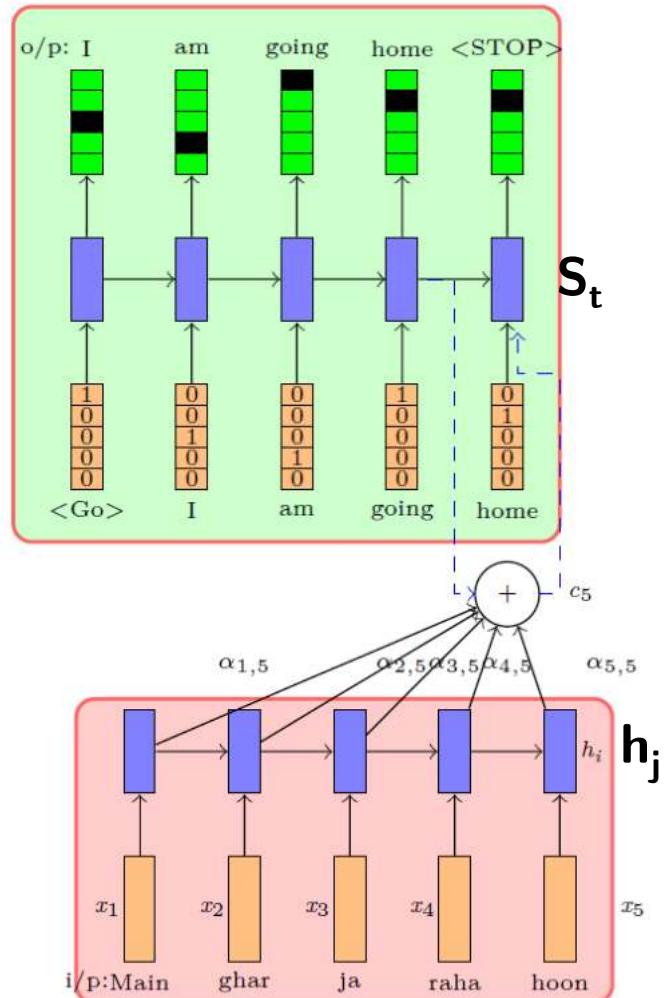
- To enable the network to focus on certain data we define the following function:

$$e_{jt} = f_{ATT}(s_{t-1}, h_j)$$



Encoder Decoder with Attention Mechanism

Task: Machine translation



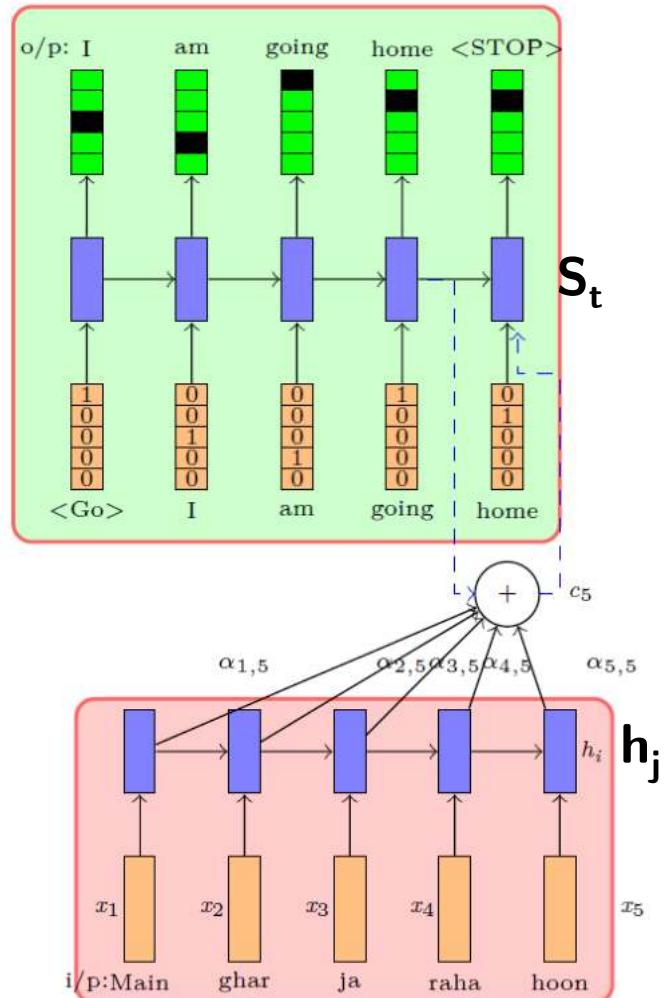
- To enable the network to focus on certain data we define the following function:

$$e_{jt} = f_{ATT}(s_{t-1}, h_j)$$

- This quantity captures the importance of the j^{th} input word for decoding the t^{th} output word.

Encoder Decoder with Attention Mechanism

Task: Machine translation



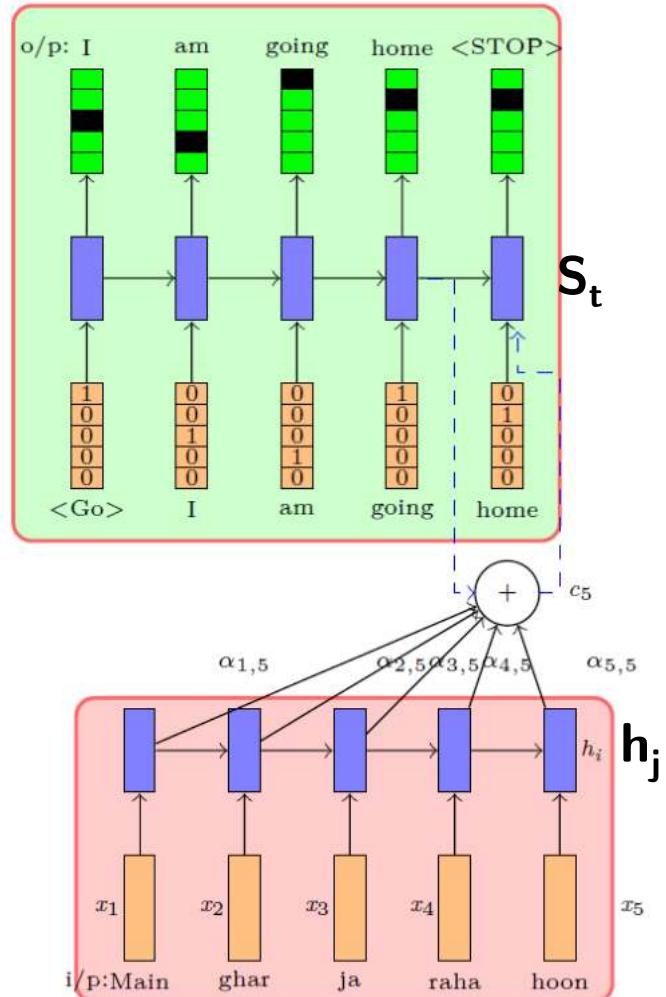
- To enable the network to focus on certain data we define the following function:

$$e_{jt} = f_{ATT}(s_{t-1}, \mathbf{h}_j)$$

- This quantity captures the importance of the jth input word for decoding the tth output word.
- We could compute α_{jt} by normalizing these weights using the softmax function.

Encoder Decoder with Attention Mechanism

Task: Machine translation



- To enable the network to focus on certain data we define the following function:

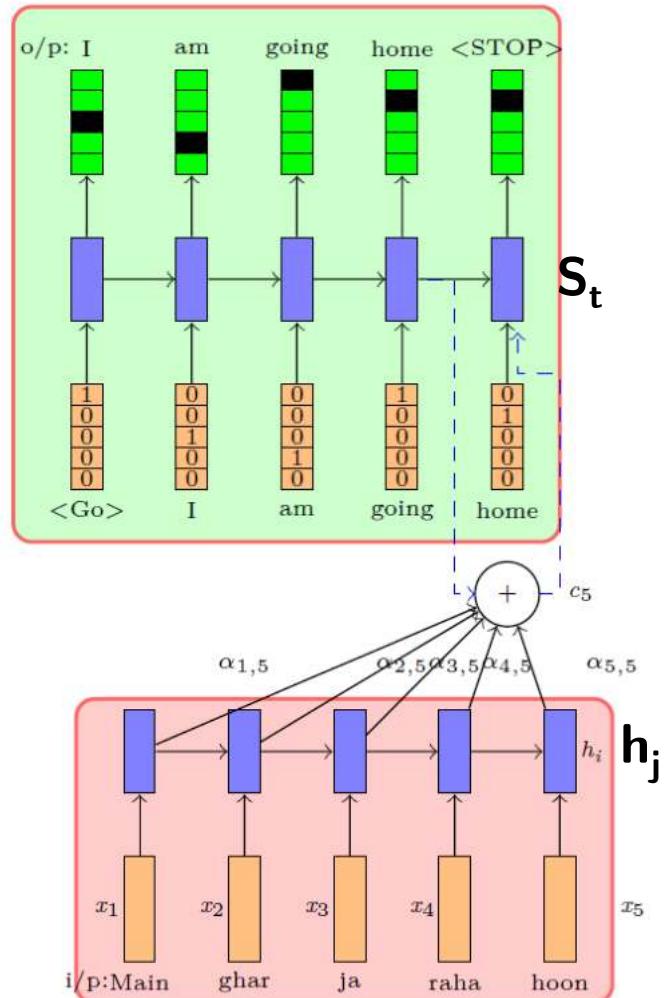
$$e_{jt} = f_{ATT}(s_{t-1}, \mathbf{h}_j)$$

- This quantity captures the importance of the jth input word for decoding the tth output word.
- We could compute α_{jt} by normalizing these weights using the softmax function.

$$\alpha_{jt} = \frac{\exp(e_{jt})}{\sum_{j=1}^M \exp(e_{jt})}$$

Encoder Decoder with Attention Mechanism

Task: Machine translation



- To enable the network to focus on certain data we define the following function:

$$e_{jt} = f_{ATT}(s_{t-1}, h_j)$$

- This quantity captures the importance of the j^{th} input word for decoding the t^{th} output word.
- We could compute α_{jt} by normalizing these weights using the softmax function.

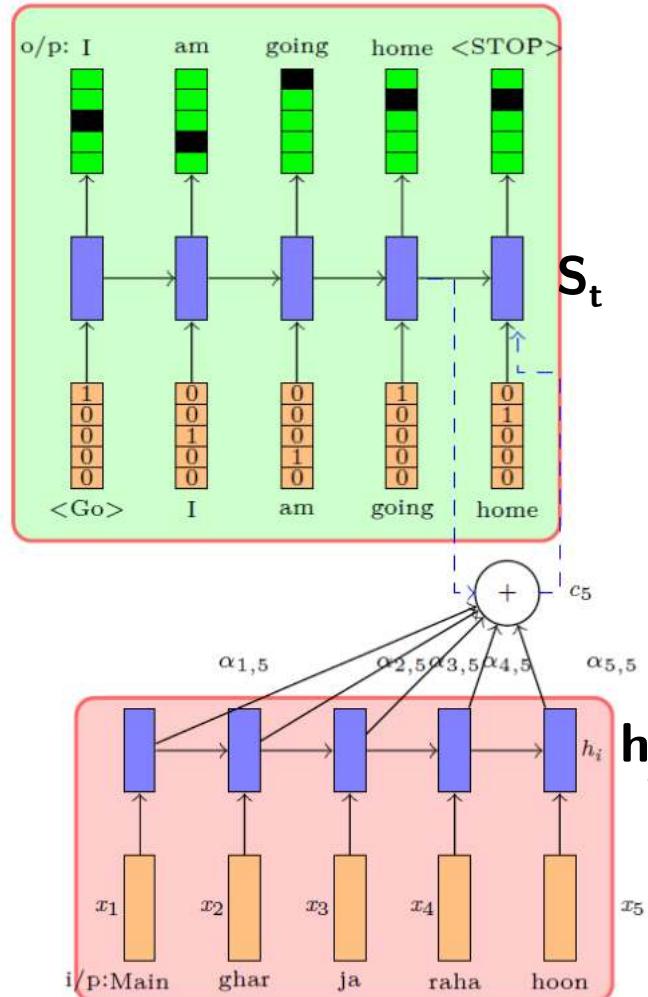
$$\alpha_{jt} = \frac{\exp(e_{jt})}{\sum_{j=1}^M \exp(e_{jt})}$$

- Where, α_{jt} denotes the probability of focusing on the j^{th} word to produce the t^{th} output word

Encoder Decoder with Attention Mechanism

Task: Machine translation

- Introducing the parametric form of α :



$$e_{jt} = V_{attn}^T \tanh(U_{attn}h_j + W_{attn}s_t)$$

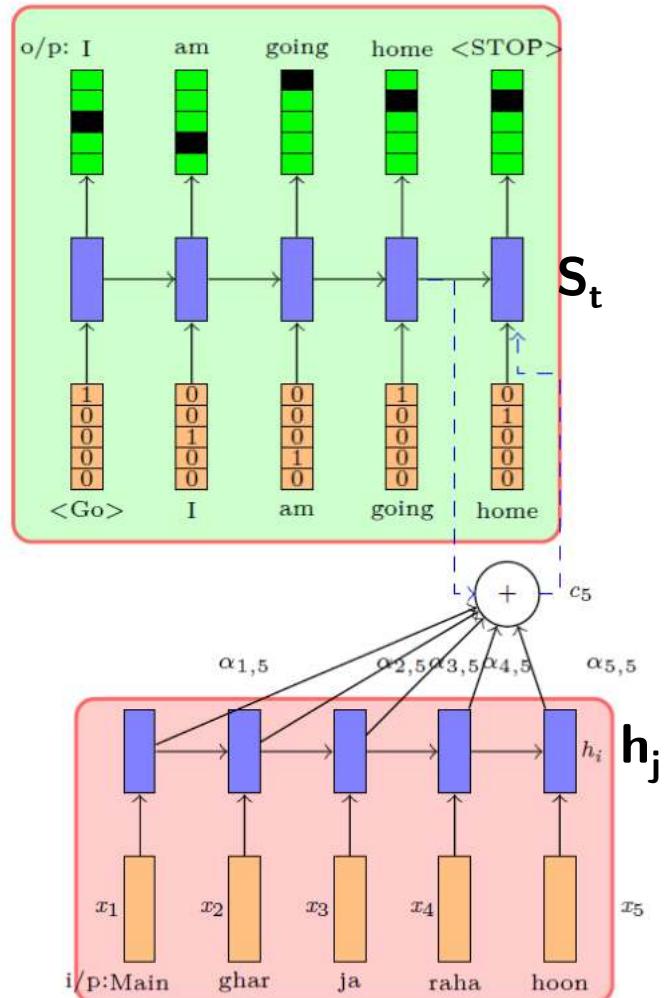
$$\alpha_{jt} = \frac{\exp(e_{jt})}{\sum_{j=1}^M \exp(e_{jt})}$$

$$c_t = \sum_{j=1}^T \alpha_{jt} h_j$$

- Where, c_t (context) gives a weighted sum over the inputs.

Encoder Decoder with Attention Mechanism

Task: Machine translation



- **Data:** $\{x_i = \text{source}_i, y_i = \text{target}_i\}_{i=1}^N$

- **Encoder:**

$$h_t = RNN(h_{t-1}, x_t)$$

$$s_0 = h_T$$

- **Decoder:**

$$e_{jt} = V_{attn}^T \tanh(U_{attn} h_j + W_{attn} s_t)$$

$$\alpha_{jt} = \text{softmax}(e_{jt})$$

$$c_t = \sum_{j=1}^T \alpha_{jt} h_j$$

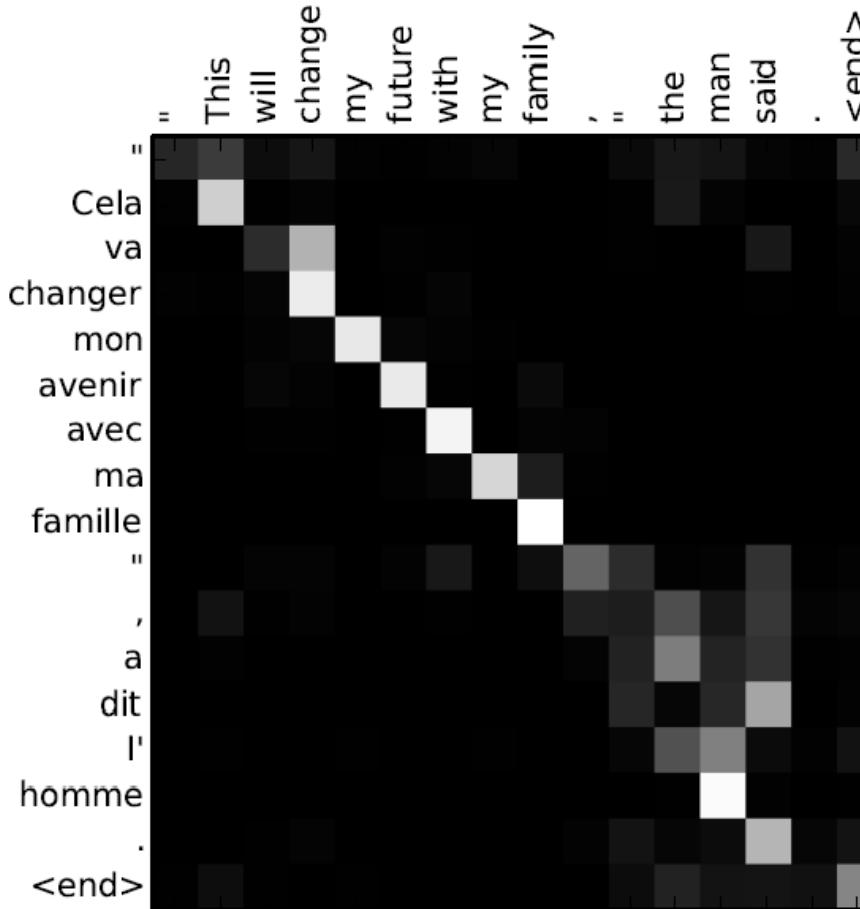
$$s_t = RNN(s_{t-1}, [e(\hat{y}_{t-1}), c_t])$$

$$\ell_t = \text{softmax}(Vs_t + b)$$

- **Parameters:** $U_{dec}, V, W_{dec}, U_{enc}, W_{enc}, b, U_{attn}, V_{attn}$

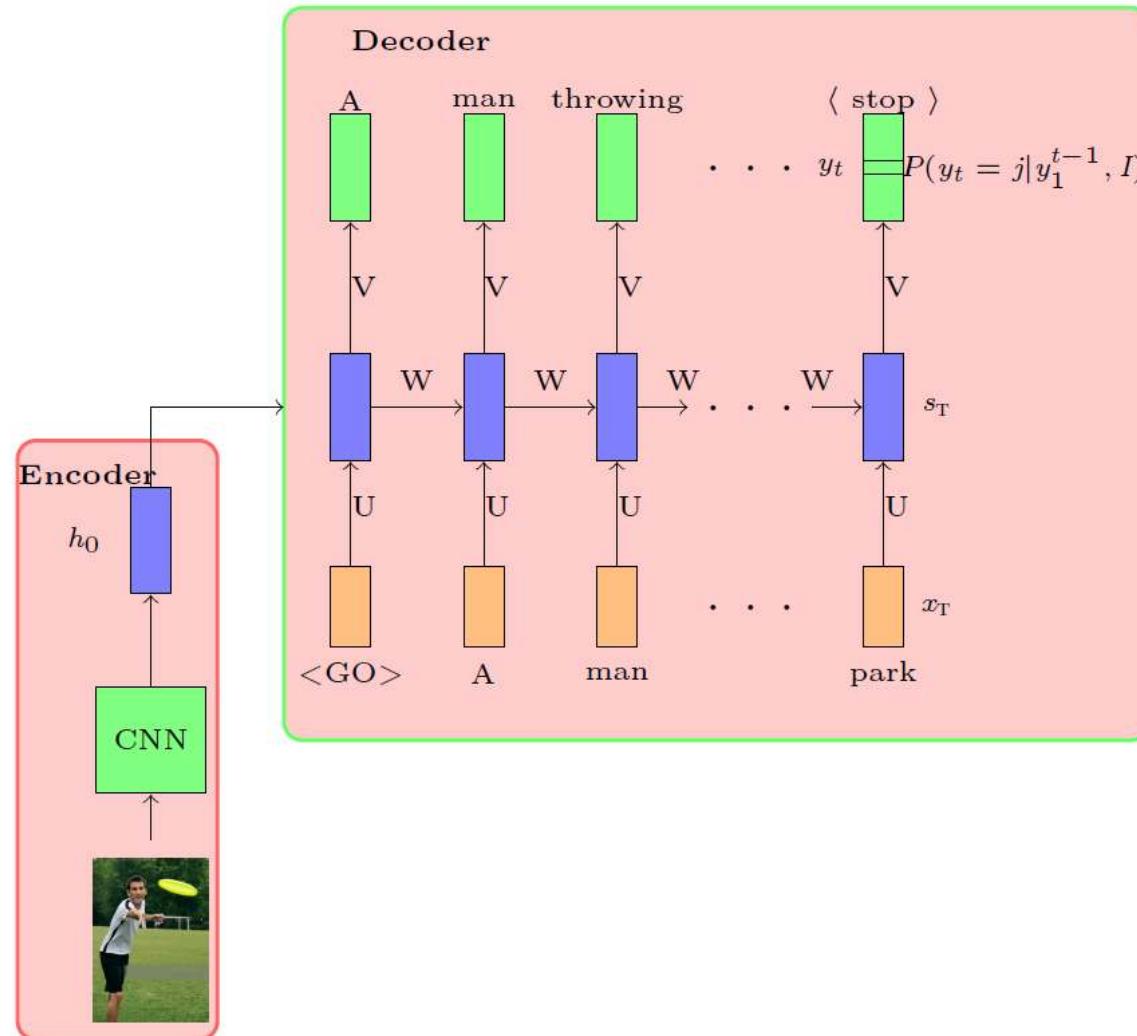
- **Loss and Algorithm** remains same

Encoder Decoder with Attention Mechanism: Visualization

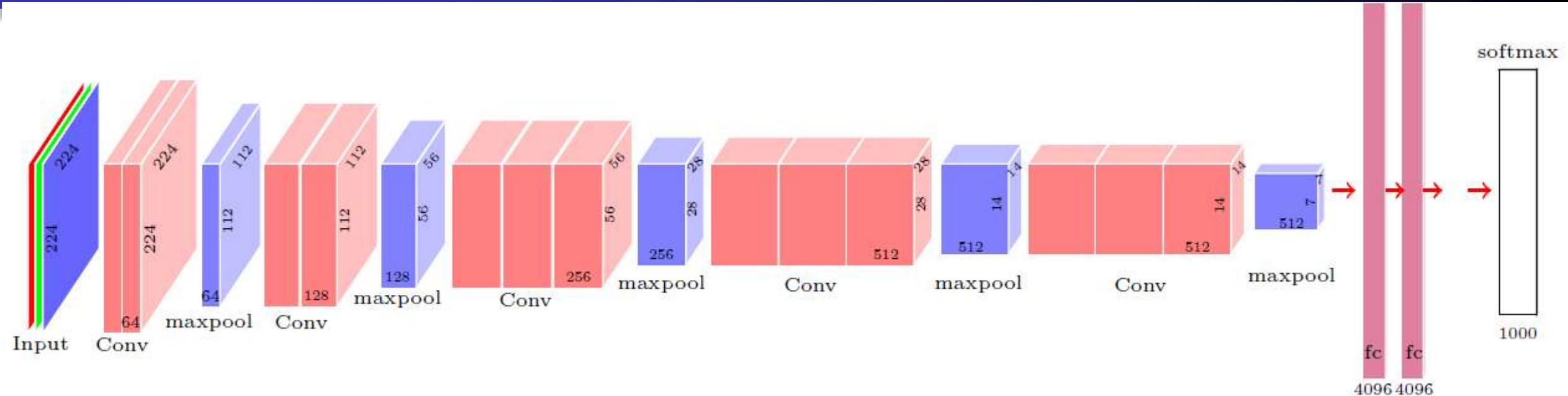


Example output of attention-based neural machine translation model
Bahdanau et al. 2015

Attention over Images

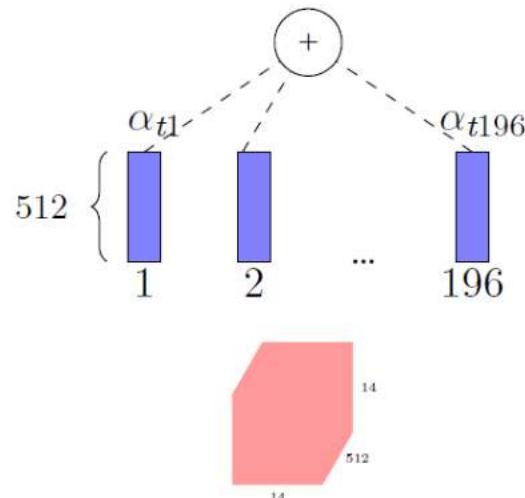
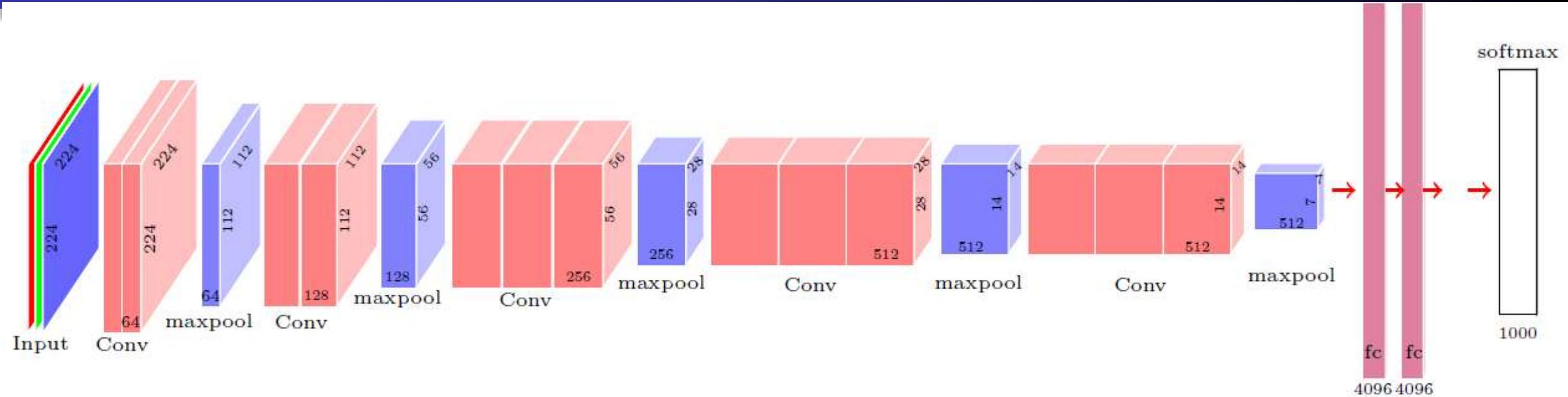


Attention over Images



- For a CNN (eg: VGG-16) we would consider the convolution layer to be an input to the decoder, instead of the fully connected layers.
- This is because, the information about the image is contained in the feature maps in the convolution layer.
- Therefore, we could add attention weights to each pixel of the feature map volume to make the model focus on a particular pixel or region in the image.

Attention over Images



- For a CNN (eg: VGG-16) we would consider the convolution layer to be an input to the decoder, instead of the fully connected layers.
- This is because, the information about the image is contained in the feature maps in the convolution layer.
- Therefore, we could add attention weights to each pixel of the feature map volume to make the model focus on a particular pixel or region in the image.

Source: CS7015 Deep Learning, Dept. of CSE, IIT Madras

Attention over Images



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.



A giraffe standing in a forest with trees in the background.

Figure: Examples of the attention-based model attending to the correct object (*white* indicates the attended regions, *underlines* indicates the corresponding word) [Kyunghyun Cho et al. 2015.]

Evaluation of Machine Translation: BLUE score

- Bilingual Evaluation Understudy (BLUE) is a score for comparing a candidate translation of text to one or more reference translations.
- Correlates to human judgment of quality
- Scores are calculated for individual translated segments by comparing them with a set of good quality reference translations.
- Scores are then averaged over the whole corpus to reach an estimate of the translation's overall quality.
- Intelligibility or grammatical correctness are not taken into account
- Number between 0 and 1

Evaluation of Machine Translation: BLUE score

German: Ich bin zur Zeit nicht im Büro

Reference English: I am currently out of the office

MT English: am am am am am am

Precision (*uni-gram*):

$$p_1 = \frac{(1 + 1 + 1 + 1 + 1 + 1)}{6} = 1$$

Evaluation of Machine Translation: BLUE score

German: Ich bin zur Zeit nicht im Büro

Reference English: I am currently out of the office

MT English: am am am am am am

Modified Precision (uni-gram):

$$p_1 = \frac{(1 + 0 + 0 + 0 + 0 + 0)}{6} = 0.16$$

Evaluation of Machine Translation: BLUE score

German: Ich bin zur Zeit nicht im Büro

Reference English: I am currently out of the office

MT English: I am currently not in the office

Modified Precision (*unigram*):

$$p_1 = \frac{1 + 1 + 1 + 0 + 0 + 1 + 1}{7} = \frac{5}{7} = 0.71$$

Modified Precision (*bi-gram*):

English: (I am), (am currently), (currently out), (out, of), (of the), (the office)

MT English: (I am), (am currently), (currently not), (not in), (in the), (the office)

$$p_2 = \frac{1 + 1 + 1 + 0 + 0 + 1}{6} = \frac{4}{6} = 0.66$$

Evaluation of Machine Translation: BLUE score

German: Ich bin zur Zeit nicht im Büro

Reference English: I am currently out of the office

MT English: I am

Modified Precision (unigram): $p_1 = 1$

Modified Precision (bigram): $p_2 = 1$

Modifier Precision (n-gram): $p_n = 1$

Evaluation of Machine Translation: BLUE score

German: Ich bin zur Zeit nicht im Büro

Reference English: I am currently out of the office

MT English: I am

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

Brevity Penalty

$$\text{Bleu Score} = BP \cdot e^{\left(\frac{1}{N} \sum_{n=1}^N P_n\right)}$$

Evaluation of Machine Translation: BLUE score

German: Ich bin zur Zeit nicht im Büro

Reference English: I am currently out of the office

MT English: I am

$$BP = \begin{cases} 1 & \text{if } c > r \\ e^{(1-r/c)} & \text{if } c \leq r \end{cases}$$

Brevity Penalty

$$\text{Bleu Score} = BP \cdot e^{\left(\frac{1}{N} \sum_{n=1}^N P_n\right)}$$

Evaluation of machine translation

NIST

- n-gram precision
 - counts how many ($i=1, \dots, 4$) grams match their n-gram counterpart in the reference translations.
- BLEU simply calculates n-gram precision adding equal weight to each segment
- NIST also calculates how informative a particular n-gram is.
- when a correct n-gram is found, the rarer that n-gram is, the more weight it will be given

Evaluation of machine translation

- **WORD ERROR RATE**
 - metric based on the Levenshtein distance at the word level.
 - based on the calculation of the number of words that differ between a piece of machine-translated text and a reference translation.
- **METEOR (Metric for Evaluation of Translation with Explicit ORdering)**
 - Recall – the proportion of the matched n-grams out of the total number of n-grams in the reference translation
 - based on the harmonic mean of unigram precision and recall, with recall weighted higher than precision
 - Features include stemming and synonymy matching along with exact word matching
- **LEPOR (Length Penalty, Precision, n-gram Position difference Penalty and Recall)**
 - **Language Independent – overcomes language bias problem**
 - factors of
 - enhanced length penalty – Translator is punished if longer or shorter than the reference translation
 - precision - score reflects the accuracy of the hypothesis translation
 - recall - score reflects the loyalty of the hypothesis translation to the reference translation or source language.
 - n-gram word order penalty- is designed for the different position orders between the hypothesis translation and reference translation.

Hierarchical Attention : Introduction to Hierarchical Models

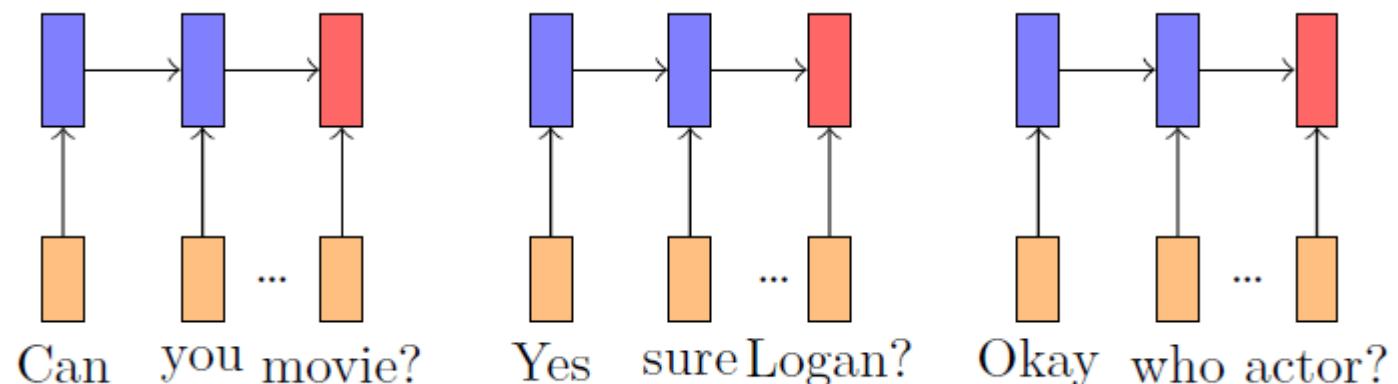
Task: Chat Bot

Context

U: Can you suggest a good movie?
B: Yes, sure. How about Logan?
U: Okay, who is the lead actor?

Response

B: Hugh Jackman, of course



Hierarchical Attention : Introduction to Hierarchical Models

Task: Chat Bot

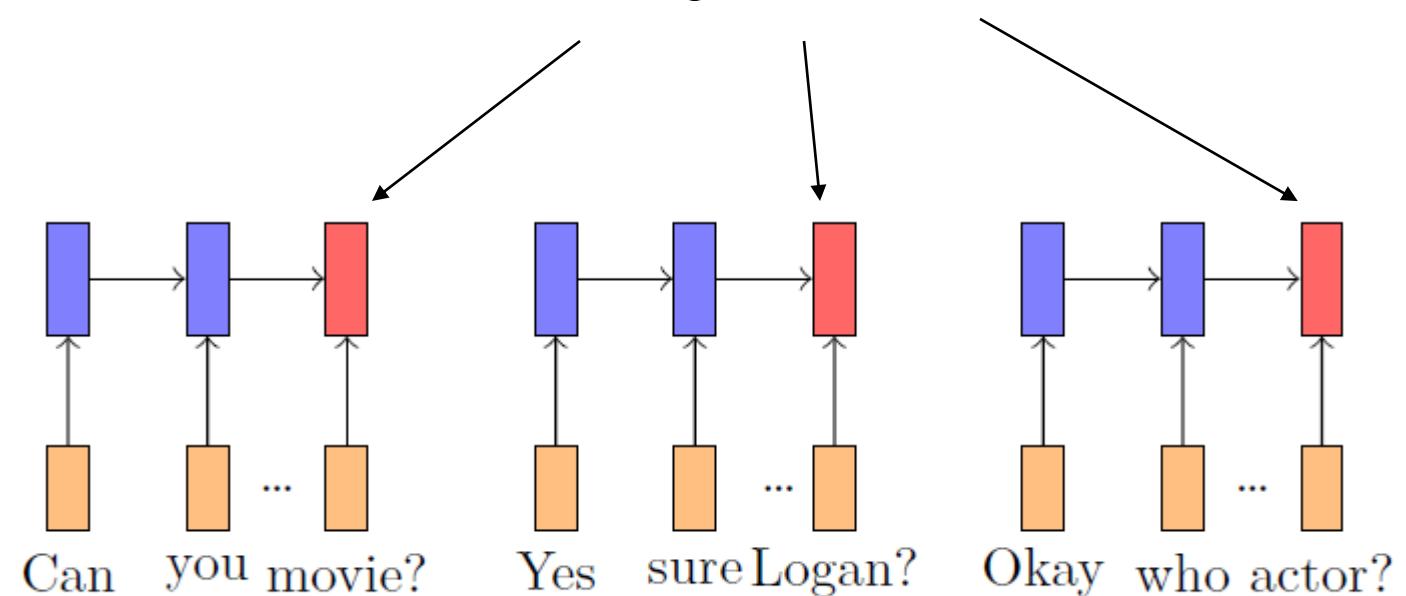
Context

U: Can you suggest a good movie?
B: Yes, sure. How about Logan?
U: Okay, who is the lead actor?

Response

B: Hugh Jackman, of course

Encoding of utterances



Hierarchical Attention : Introduction to Hierarchical Models

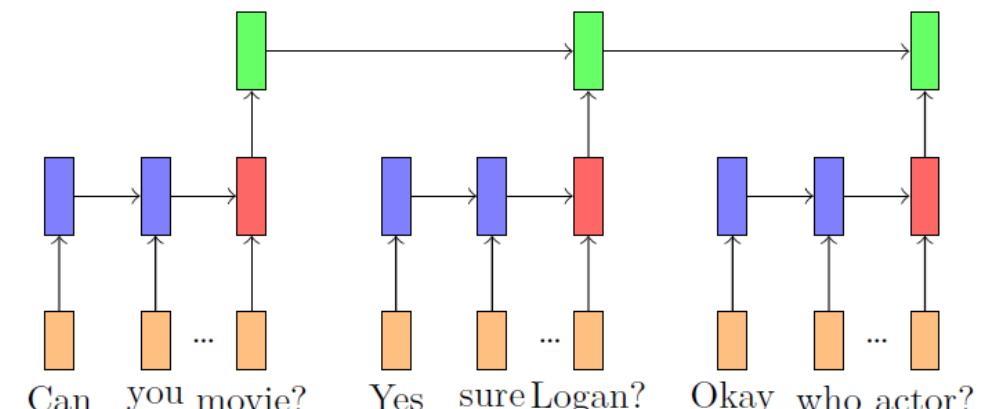
Task: Chat Bot

Context

U: Can you suggest a good movie?
B: Yes, sure. How about Logan?
U: Okay, who is the lead actor?

Response

B: Hugh Jackman, of course



Hierarchical Attention : Introduction to Hierarchical Models

Task: Chat Bot

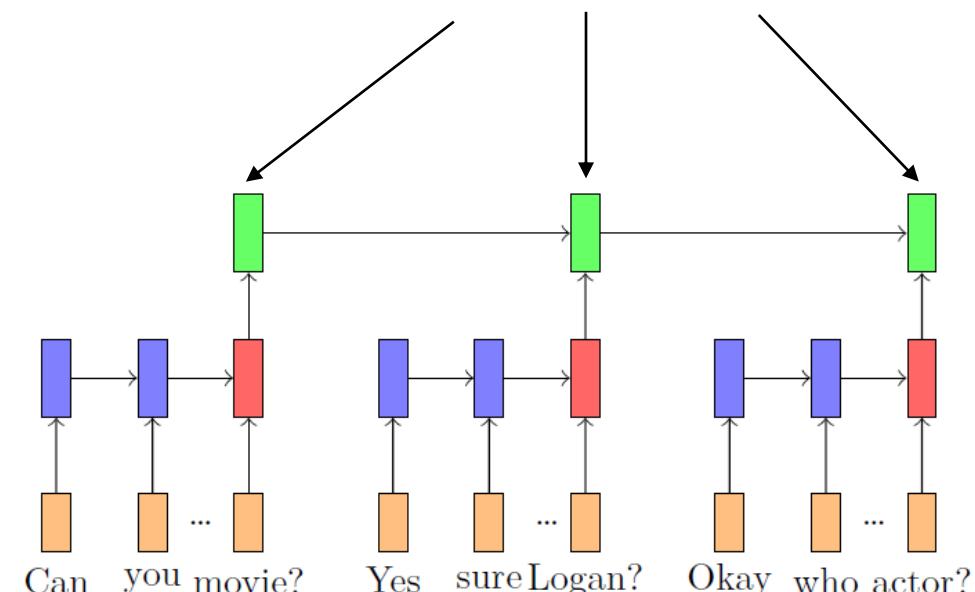
Context

U: Can you suggest a good movie?
B: Yes, sure. How about Logan?
U: Okay, who is the lead actor?

Response

B: Hugh Jackman, of course

Encoding of sequence of utterances



Hierarchical Attention : Introduction to Hierarchical Models

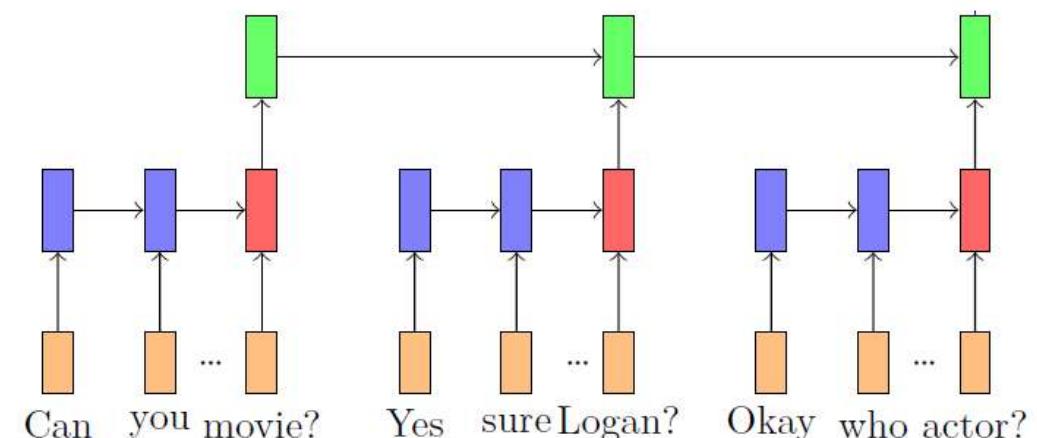
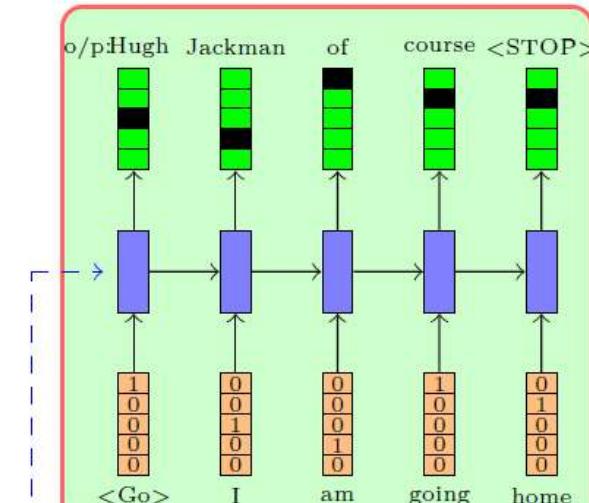
Task: Chat Bot

Context

U: Can you suggest a good movie?
B: Yes, sure. How about Logan?
U: Okay, who is the lead actor?

Response

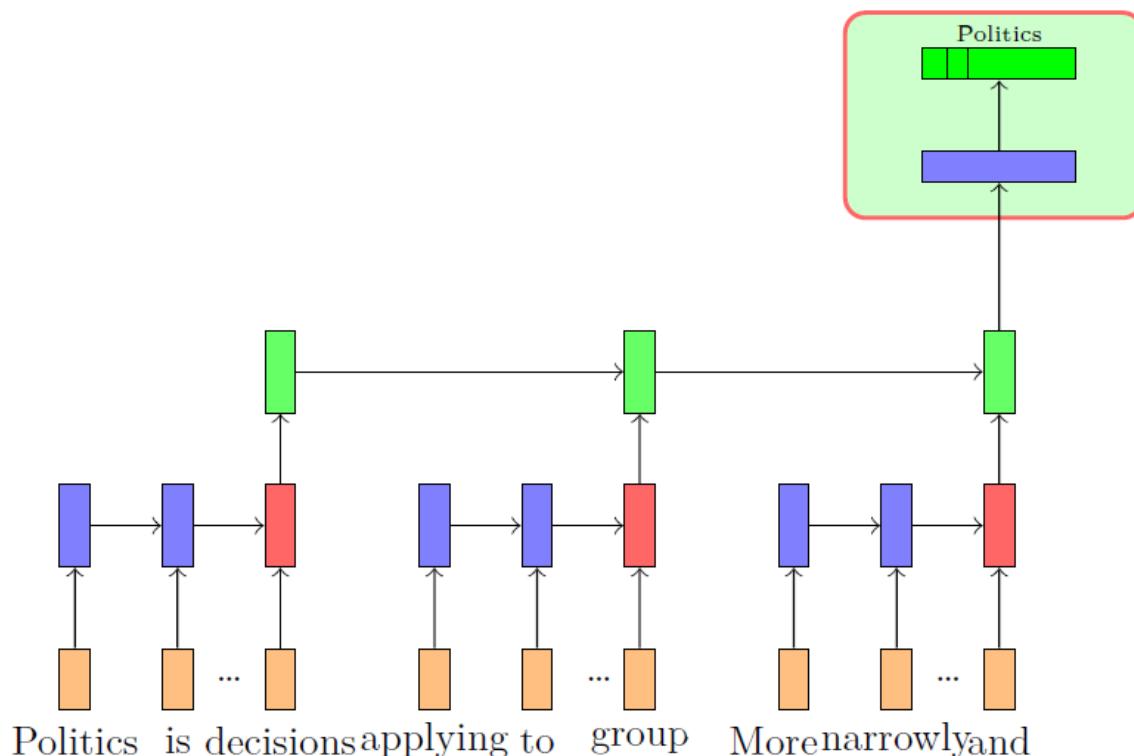
B: Hugh Jackman, of course



Hierarchical Attention : Introduction to Hierarchical Models

Task: Document Summarization

Politics is the process of making decisions applying to all members of each group.
More narrowly, it refers to achieving and ...

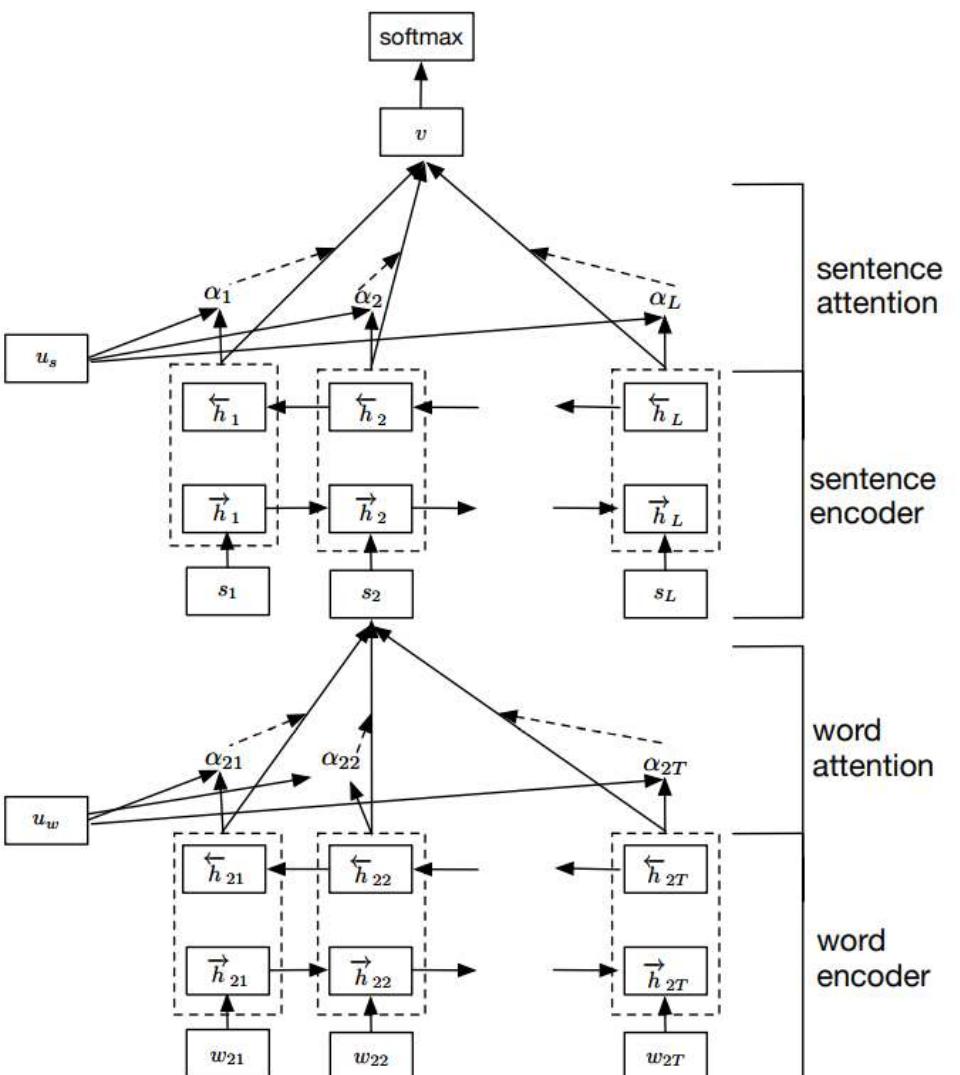


- **Data:** $\{Document_i, class_i\}_{i=1}^N$
- **Word level (1) encoder:**
$$h_{ij}^1 = RNN(h_{ij-1}^1, w_{ij})$$
$$s_i = h_{iT_i}^1 \quad [T \text{ is length of sentence } i]$$
- **Sentence level (2) encoder:**
$$h_i^2 = RNN(h_{i-1}^2, s_i)$$
$$s = h_K^2 \quad [K \text{ is number of sentences}]$$
- **Decoder:**
$$P(y|document) = softmax(Vs + b)$$
- **Params:** $W_{enc}^1, U_{enc}^1, W_{enc}^2, U_{enc}^2, V, b$
- **Loss:** Cross Entropy
- **Algorithm:** Gradient Descent with backpropagation

Document Classification using Hierarchical Attention Networks

- To understand the main message of a document
 - Not every word in a sentence and every sentence in a document are equally important
- Meaning of word depends on context
 - For example - “The bouquet of flowers is pretty” vs. “The food is pretty bad”.
- HAN
 - considers the hierarchical structure of documents (document - sentences - words)
 - Includes an attention mechanism that is able to find the most important words and sentences in a document while taking the context into consideration

Hierarchical Attention Networks



- Data: $\{Document_i, class_i\}_{i=1}^N$
- Word level (1) encoder:

$$h_{ij} = RNN(h_{ij-1}, w_{ij})$$

$$u_{ij} = \tanh(W_w h_{ij} + b_w)$$

$$\alpha_{ij} = \frac{\exp(u_{ij}^T u_w)}{\sum_t \exp(u_{it}^T u_w)}$$

$$s_i = \sum_j \alpha_{ij} h_{ij}$$

- Sentence level (2) encoder:

$$h_i = RNN(h_{i-1}, s_i)$$

$$u_i = \tanh(W_s h_i + b_s)$$

$$\alpha_i = \frac{\exp(u_i^T u_s)}{\sum_i \exp(u_i^T u_s)}$$

$$s = \sum_i \alpha_i h_i$$

- Decoder:

$$P(y|document) = \text{softmax}(Vs + b)$$

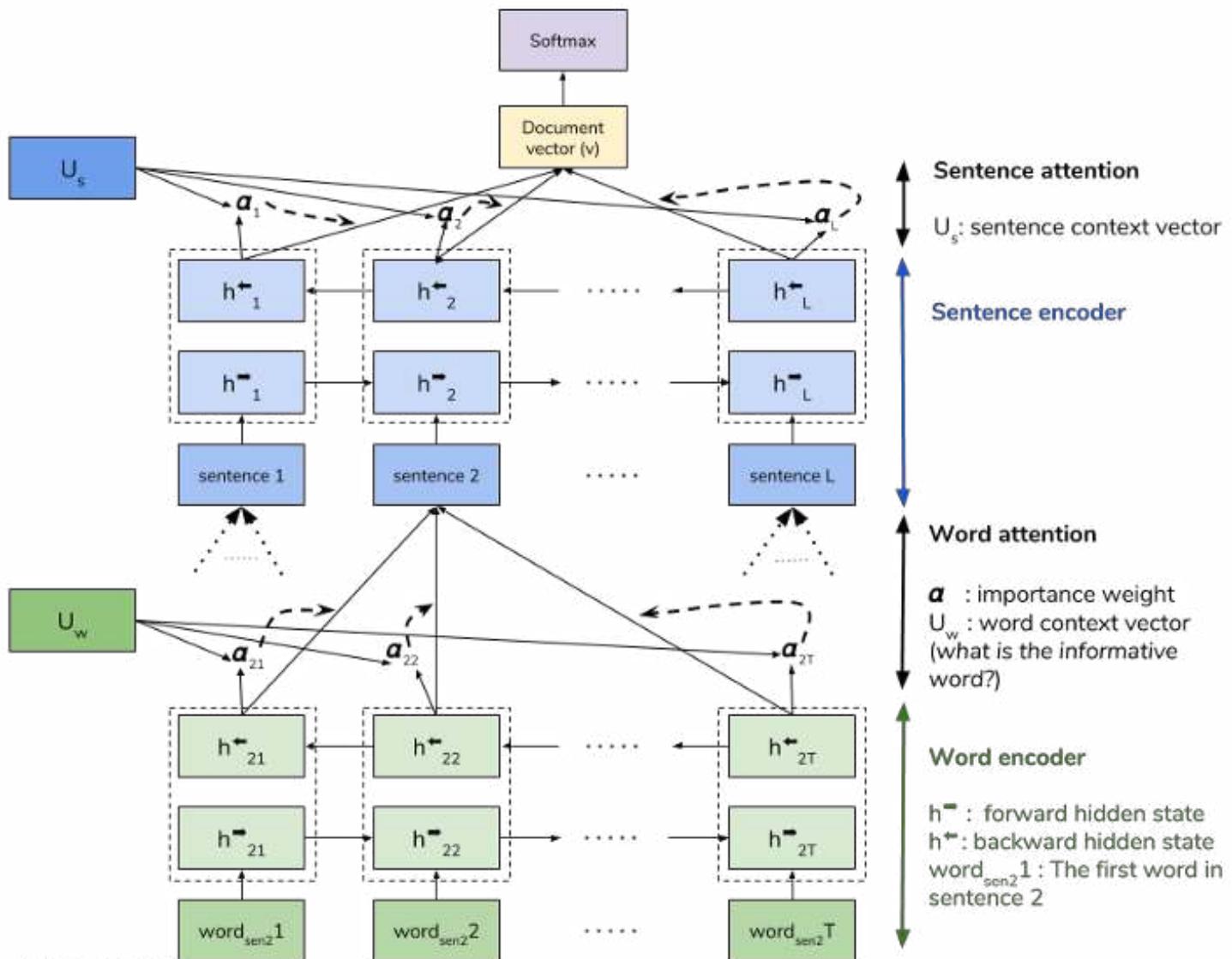
- Parameters:

$$W_w, W_s, V, b_w, b_s, b, u_w, u_s$$

- Loss: cross entropy

- Algorithm: Gradient Descent and backpropagation

Hierarchical Attention Networks (Yang et al. 2016)



GT: 4 Prediction: 4
pork belly = delicious .
scallops ?
i do n't .
even .
like .
scallops , and these were a-m-a-z-i-n-g .
fun and tasty cocktails .
next time i 'm in phoenix , i will go back here .
highly recommend .

GT: 0 Prediction: 0
terrible value .
ordered pasta entree .
\$ 16.95 good taste but size was an appetizer size .
no salad , no bread no vegetable .
this was .
our and tasty cocktails .
our second visit .
i will not go back .

Figure 5: Documents from Yelp 2013. Label 4 means star 5, label 0 means star 1.

GT: 1 Prediction: 1
why does zebras have stripes ?
what is the purpose or those stripes ?
who do they serve the zebras in the wild life ?
this provides camouflage - predator vision is such that it is usually difficult for them to see complex patterns

GT: 4 Prediction: 4
how do i get rid of all the old web searches i have on my web browser ?
i want to clean up my web browser go to tools > options .
then click " delete history " and " clean up temporary internet files . "

Figure 6: Documents from Yahoo Answers. Label 1 denotes Science and Mathematics and label 4 denotes Computers and Internet.

Transformers & Recursive Networks

DSE 3151 DEEP LEARNING

B.Tech Data Science & Engineering

August 2023

[Rohini R Rao & Abhilash K Pai](#)

Department of Data Science and Computer Applications

MIT Manipal

[Slide -4 of 5](#)

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaiser@google.com

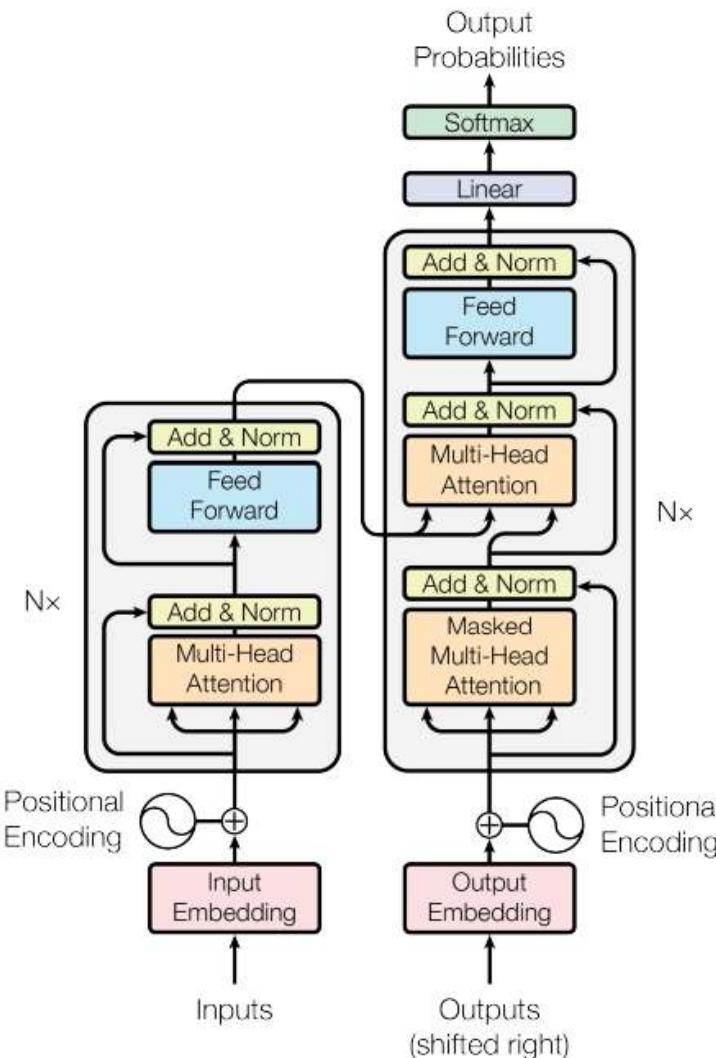
Illia Polosukhin* ‡
illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.0 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature.

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

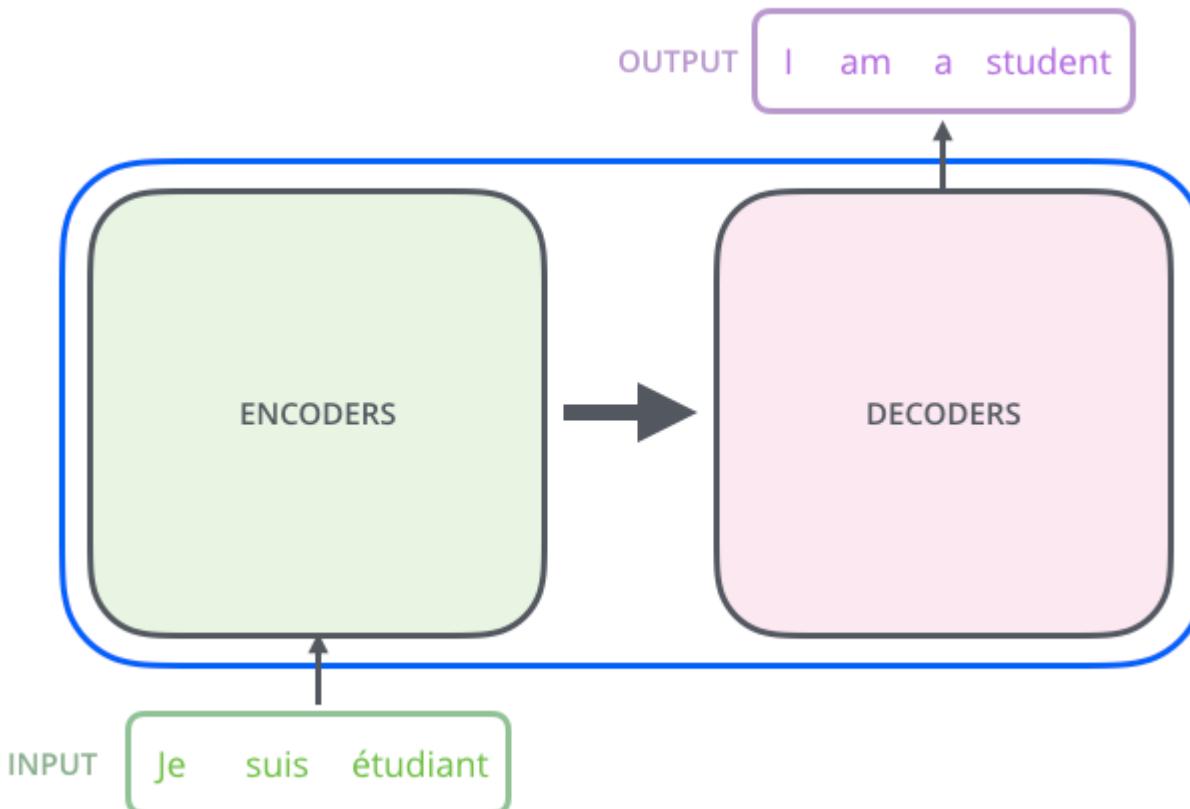
Transformers



- Vaswani et al. 2017 proposed the transformer model, entirely built on **self attention mechanism without using sequence aligned recurrent architectures.**
- Key components:
 - Self attention
 - Multi-head attention
 - Positional encoding
 - Encoder-Decoder architecture

Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

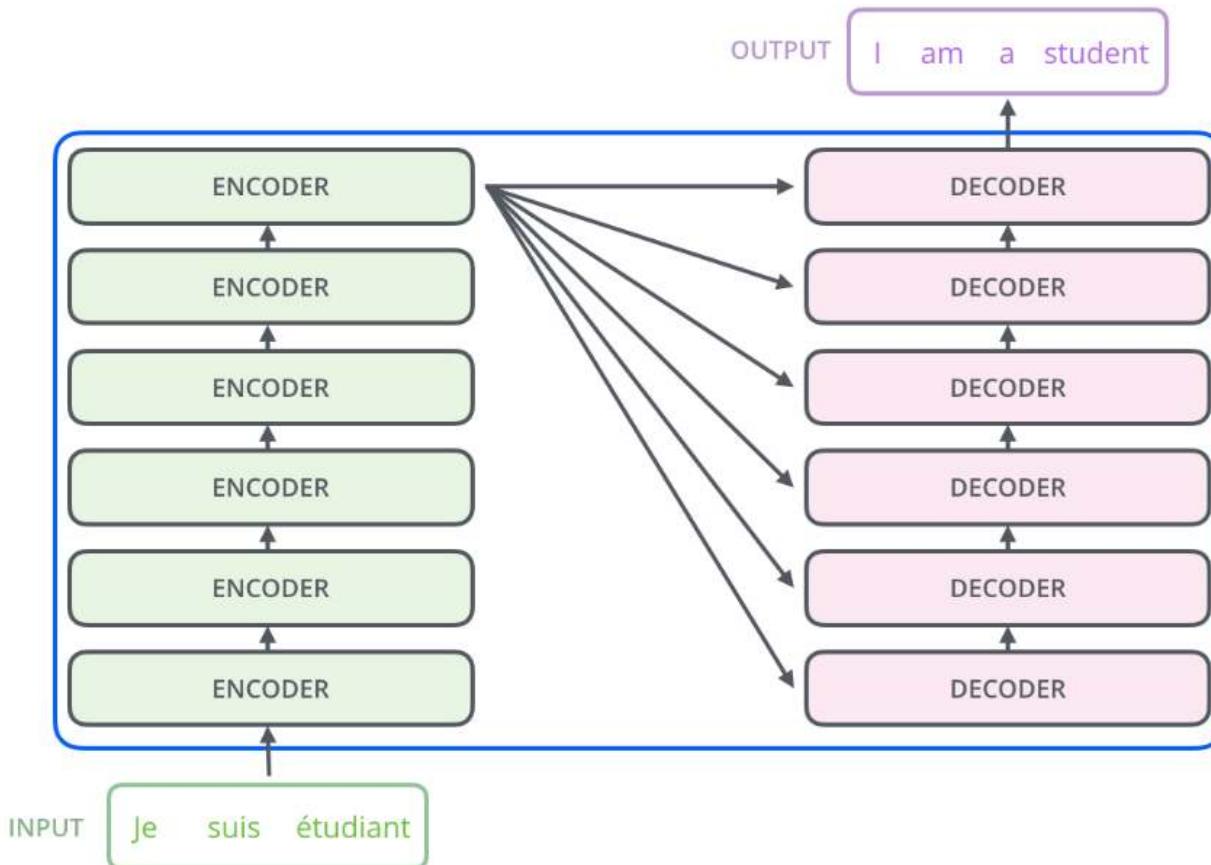
Transformers



The entire network can be viewed as an encoder decoder architecture

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalamar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

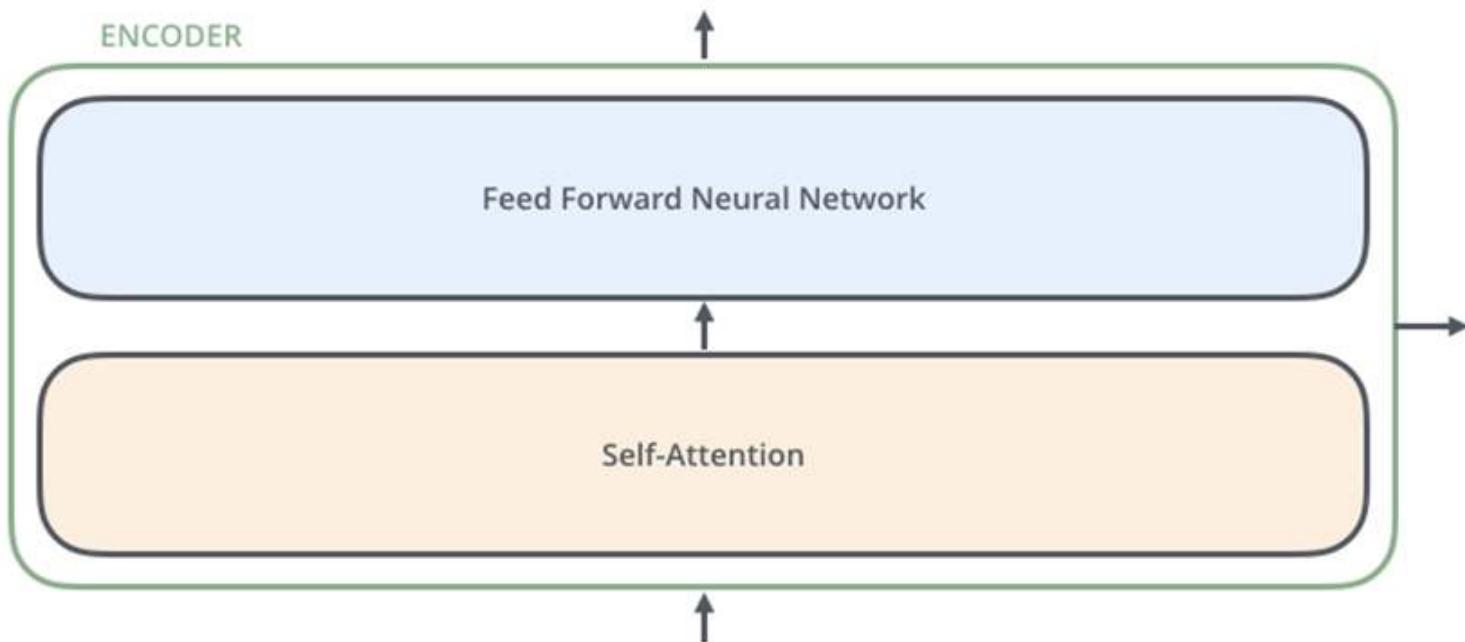
Transformers



The encoding component is a stack of encoders and the decoding component is also a stack of encoders of the same number (in the paper, this number = 6)

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalammar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

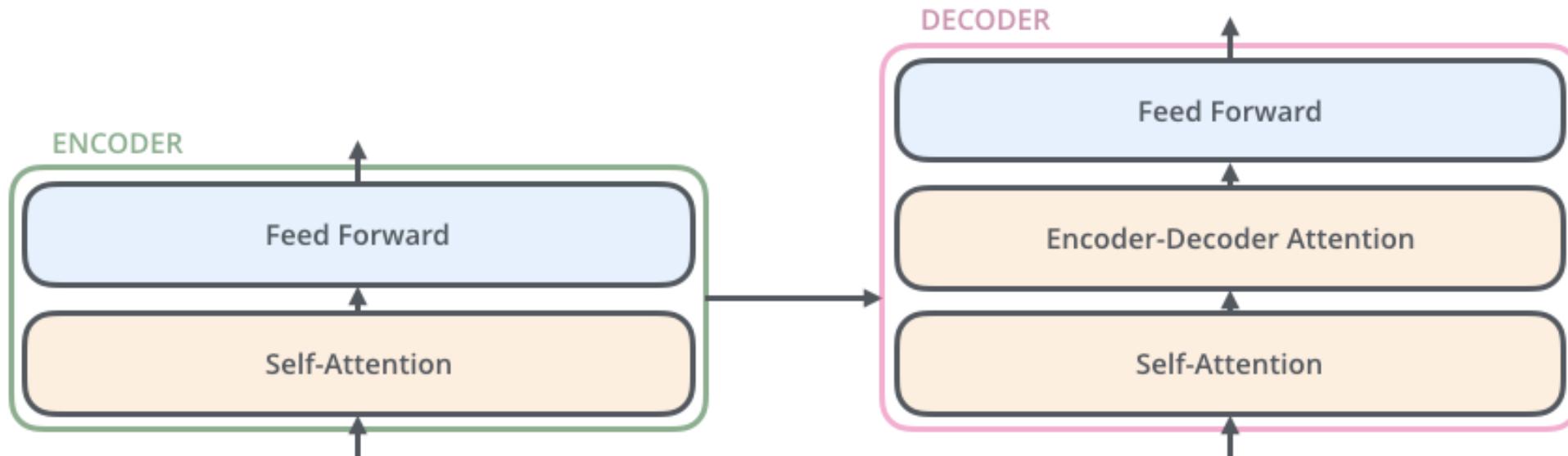
Transformers



- The encoders are all identical in structure (yet they do not share weights).
- Each one is broken down into two sub-layers
- The encoder's inputs first flow through a self-attention layer – a layer that helps the encoder look at other words in the input sentence as it encodes a specific word – to the feed forward neural network.

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalammar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

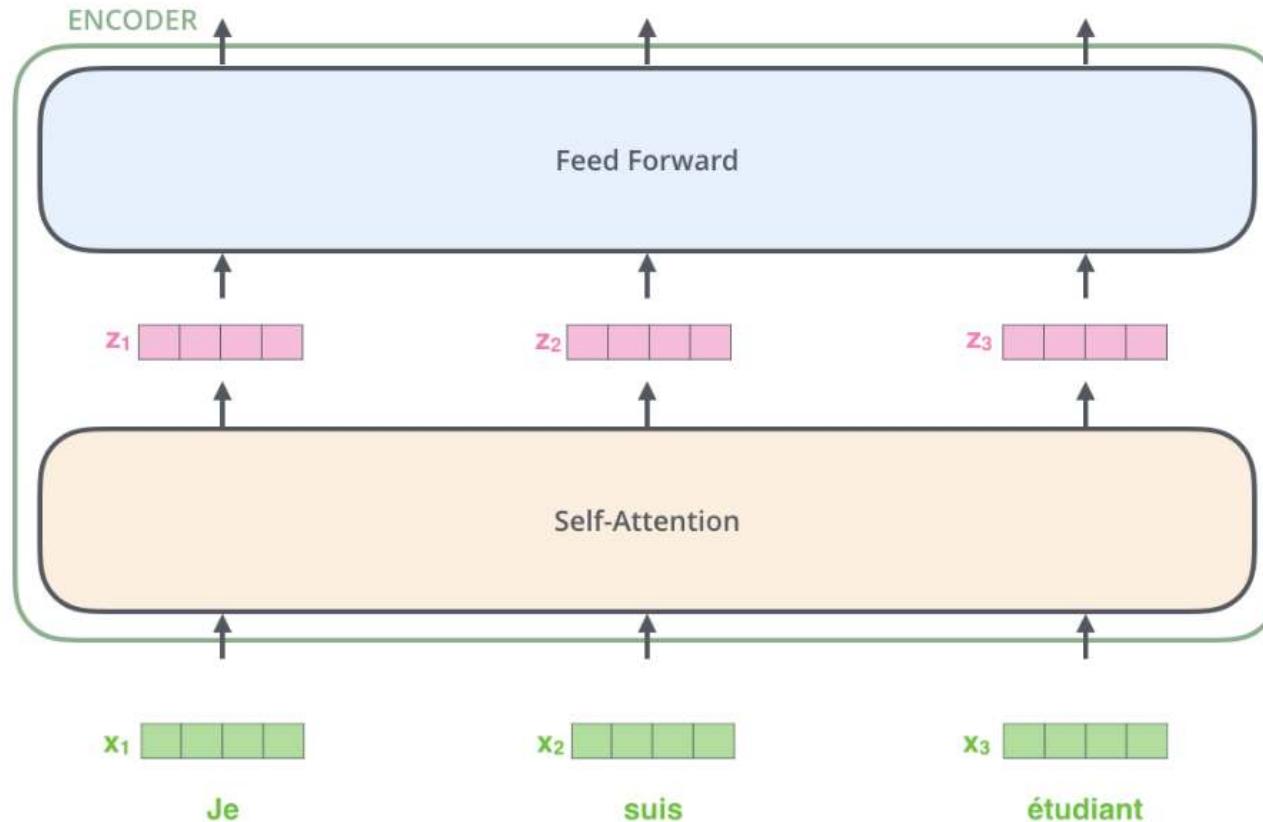
Transformers



The decoder has both those layers, but between them is an attention layer that helps the decoder focus on relevant parts of the input sentence (similar what attention does in seq2seq models).

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalamar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

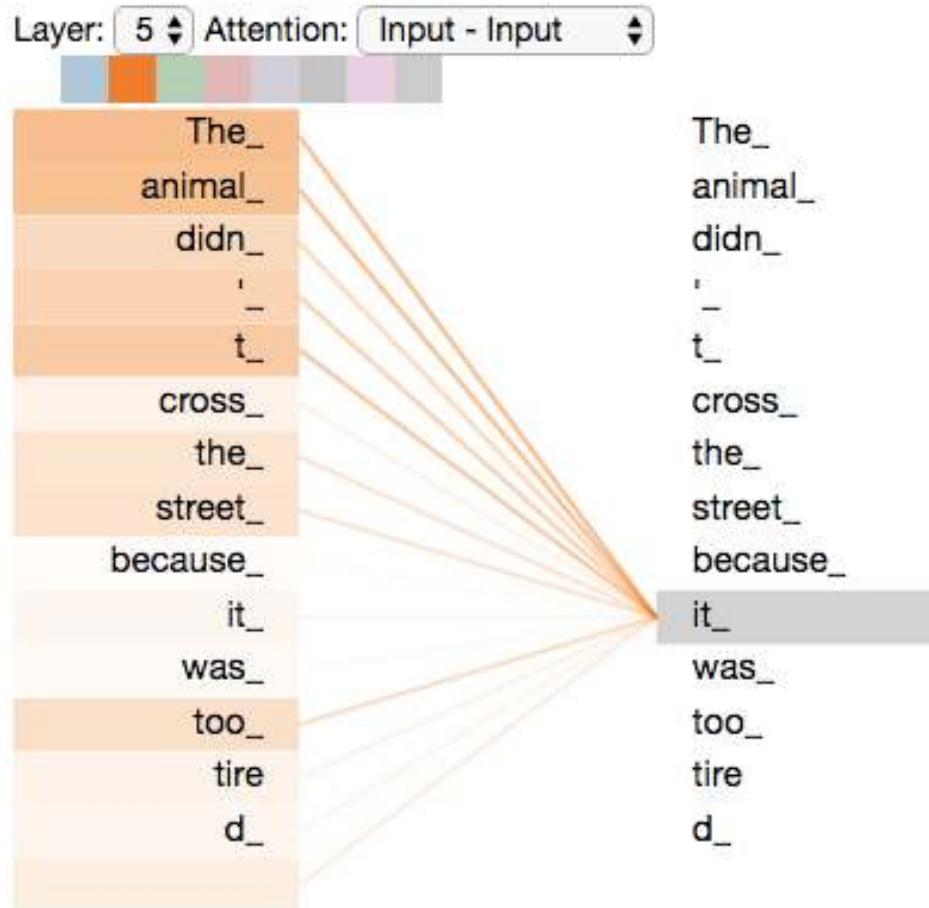
Transformers



- A word in each position flows through its own path in the encoder.
- There are dependencies between these paths in the self-attention layer.
- The feed-forward layer does not have those dependencies.
- Thus, the various paths can be executed in parallel while flowing through the feed-forward layer.
- This is a key property of the transformer.

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalammar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

Transformers : Self attention



S1: Animal didn't cross the street because **it was too tired**

S2: Animal didn't cross the **street because **it** was too wide**

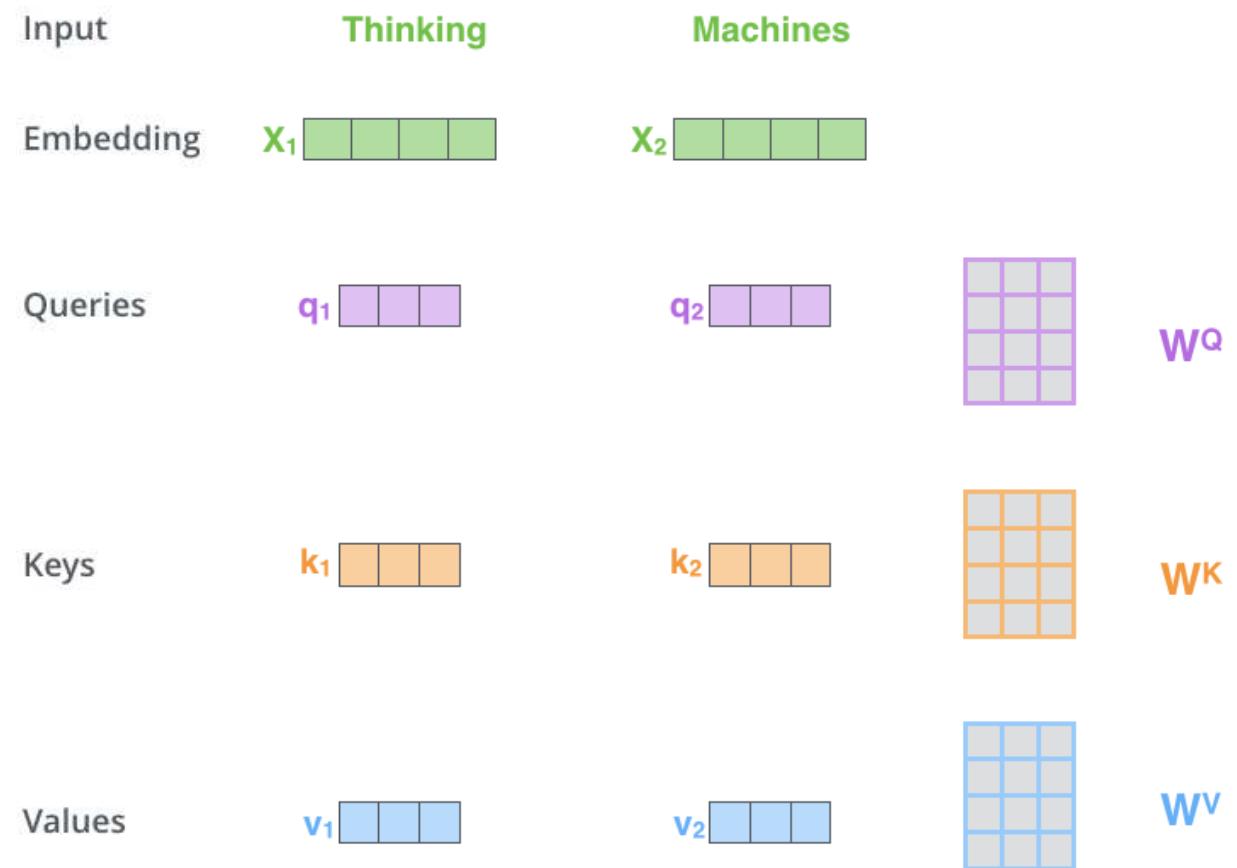
- In sentence S1, “it” refers to “animal” and in sentence S2 “it” refers to “street”.
- Such deductions are hard for traditional sequence to sequence models.
- While processing each word in a sequence, self-attention mechanism allows the model to decide as to which other parts of the same sequence it needs to focus on, which makes such deductions easier and allows better encoding.
- RNNs which maintain a hidden state to incorporate the representation of previous vectors are no longer needed!

Transformers : Self attention

Step 1: Create three vectors from encoder's input vector (x_i):

- Query vector (q_i)
- Key vector (k_i)
- Value vector (v_i)

These are created by multiplying the input with weight matrices W^q , W^k , W^v , learned during training.



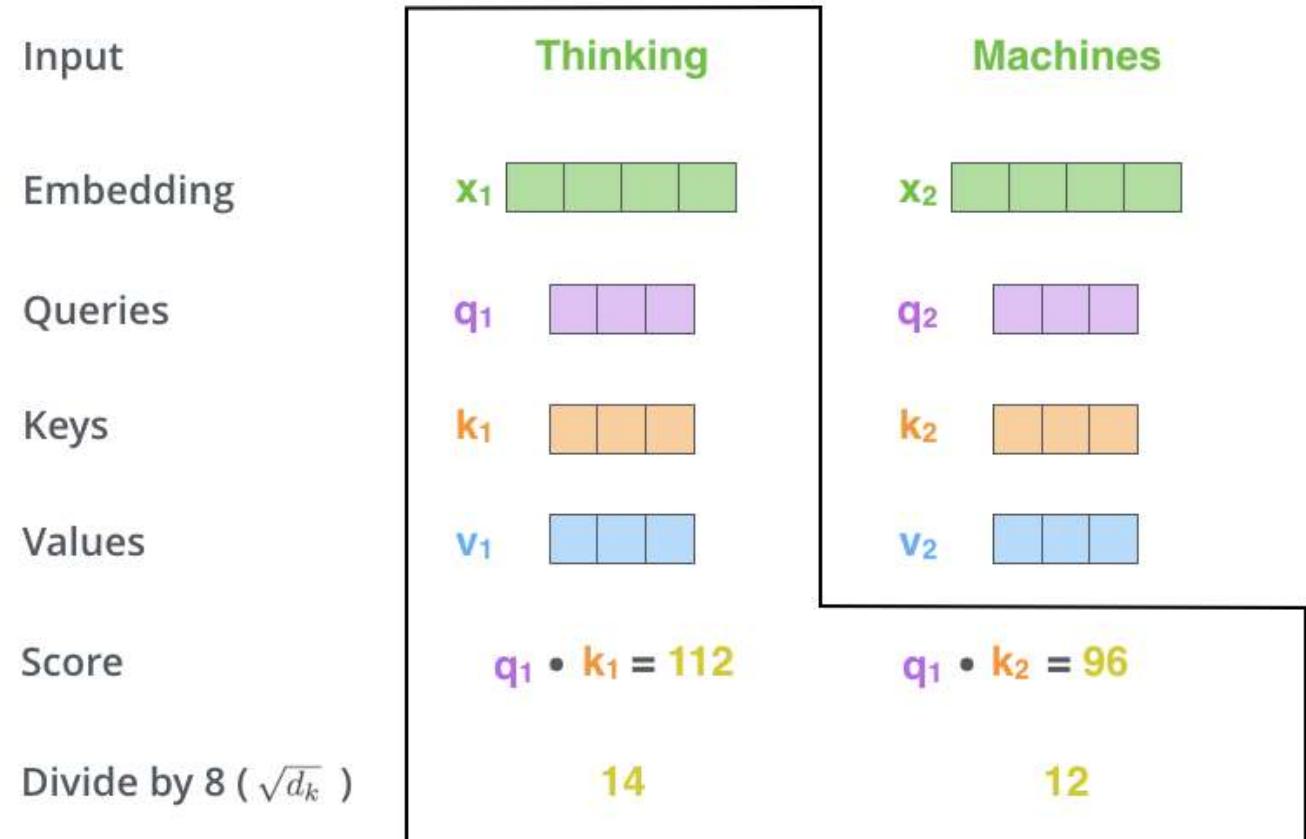
- Note: In the paper by Vaswani et al, the **q, k and w** vectors has dimension of **64** and the input vector **x** has a dimension of **512**,

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalammar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

Transformers : Self attention

Step 2: Calculate self attention scores of each word against the other words in the same sentence.

- This is done by taking the dot product of query vector with the key vector of respective words.
- The scores are divided by square root of the key length.
- This is called **Scaled Dot-Product attention**
 - it leads to more stable gradients.



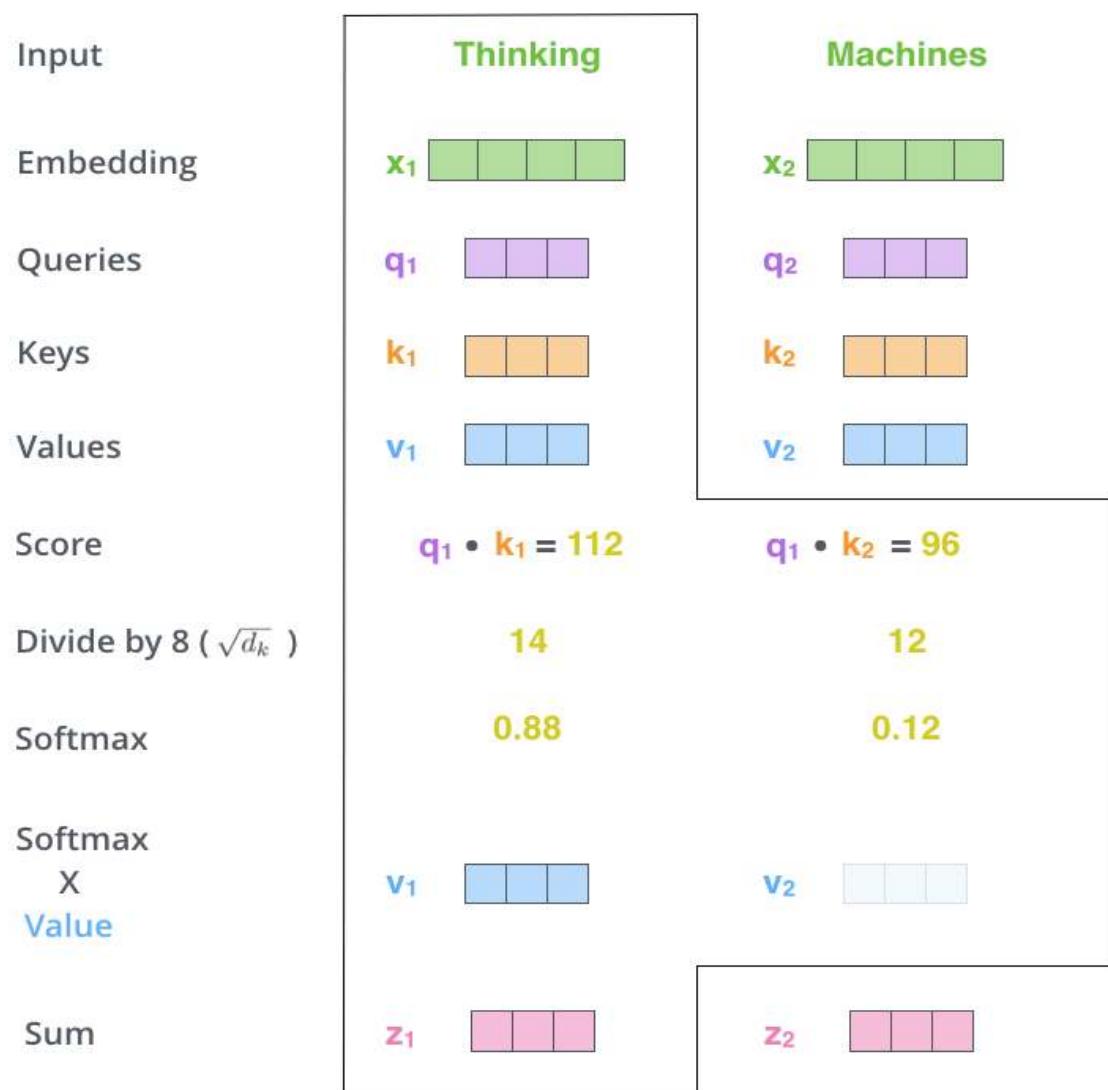
[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalamar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

Transformers : Self attention

Step 3: Softmax is used to obtain normalized probability scores; determines how much each word will be expressed at this position.

Step 4: Multiply each value vector by the Softmax score; to keep values of words we want to focus on and drown out irrelevant words.

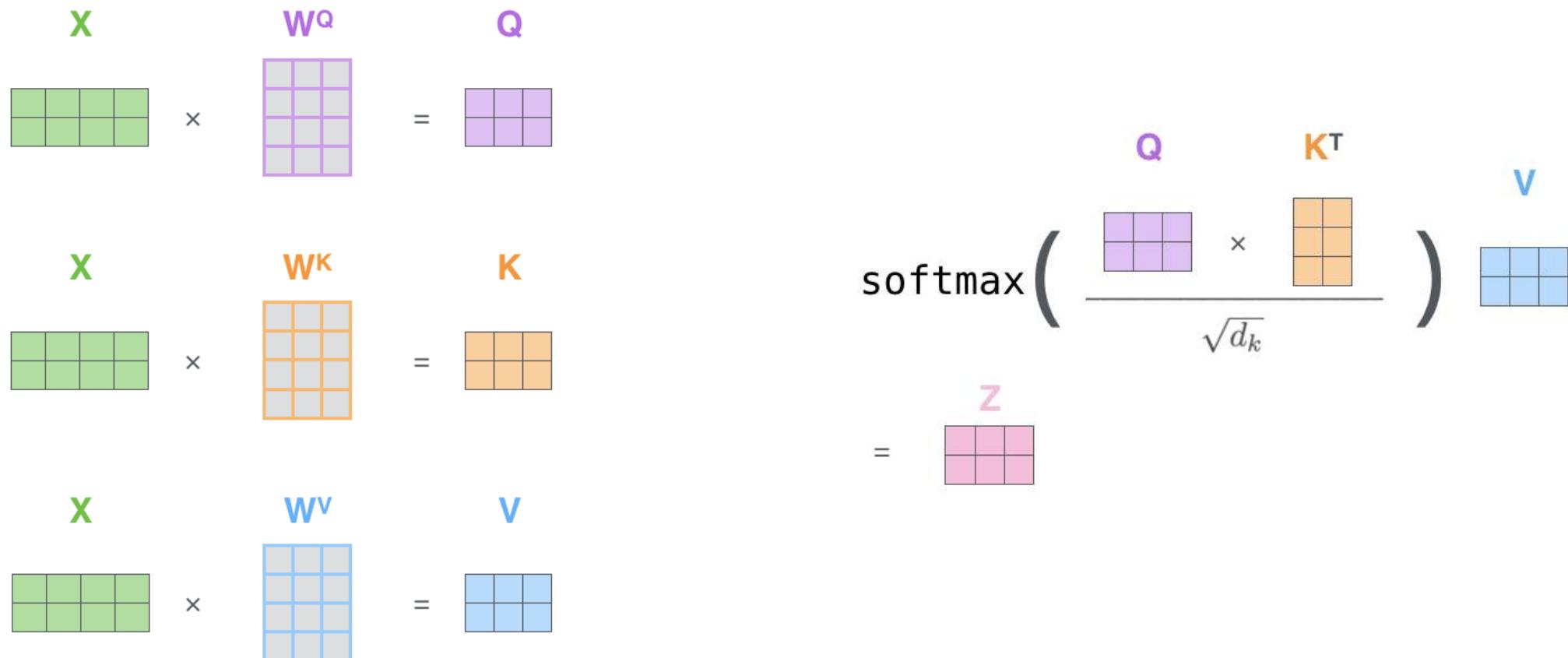
Step 5: Sum up the weighted value vectors ; produces the output of self-attention layer at this position (for first word)



[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalamar.github.io\)](http://jalammar.github.io)

Transformers : Self attention

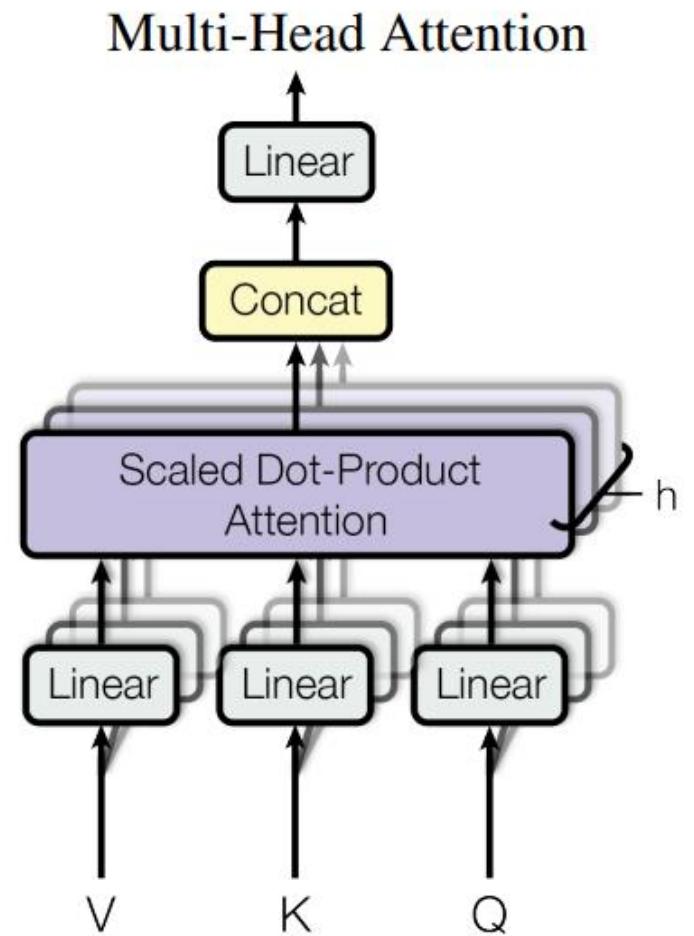
- In the actual implementation, however, Step 1 to Step 4 is done in matrix form for faster processing.



[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalamar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

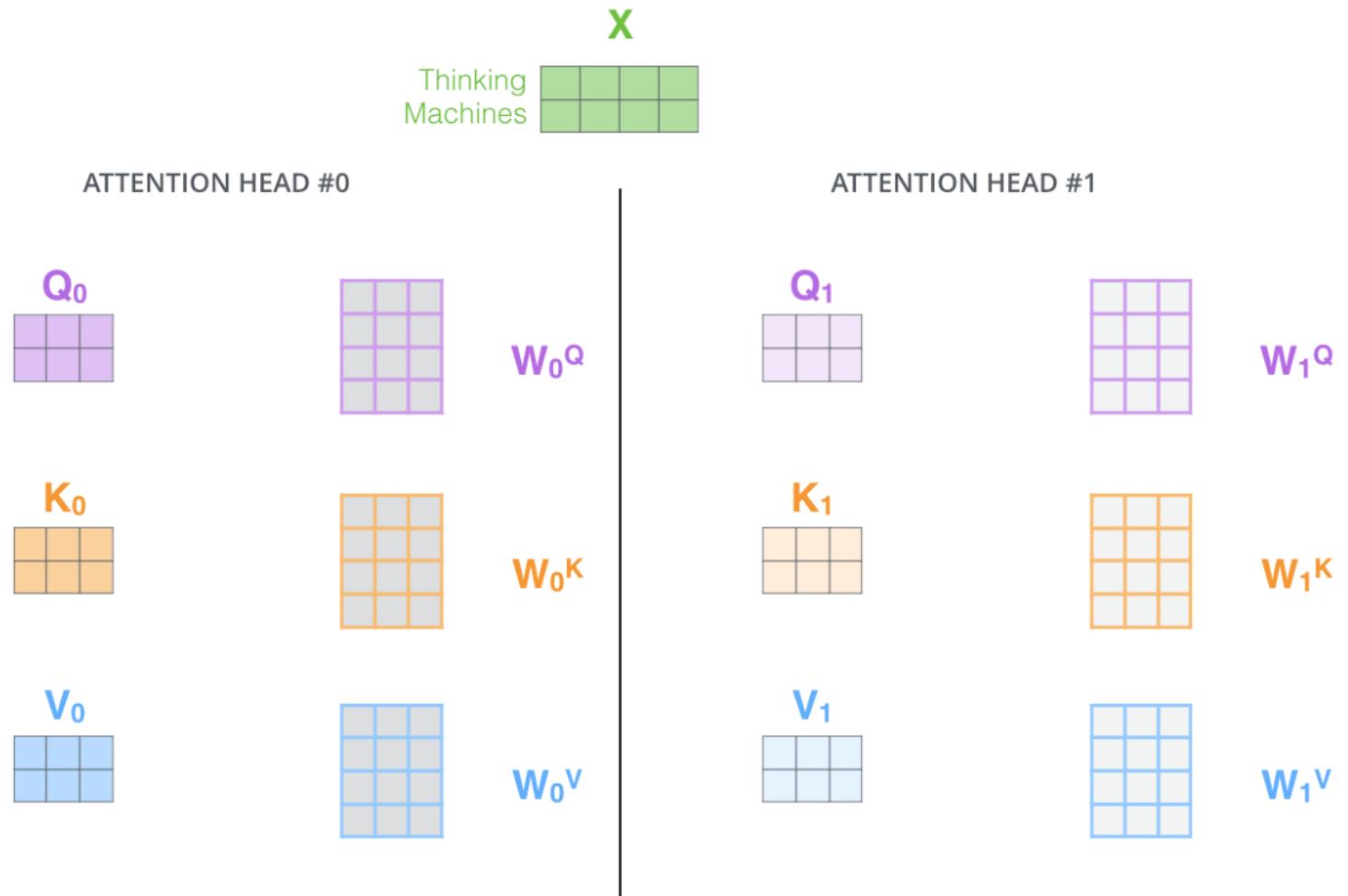
Transformers : Multi-head Attention

- Multi-head attention improves the performance of attention layer in two ways:
 - It expands the model's ability to focus on different positions.
 - It gives the attention layer multiple "representation subspaces" i.e., we have not only one, but multiple sets of Query/Key/Value weight matrices.



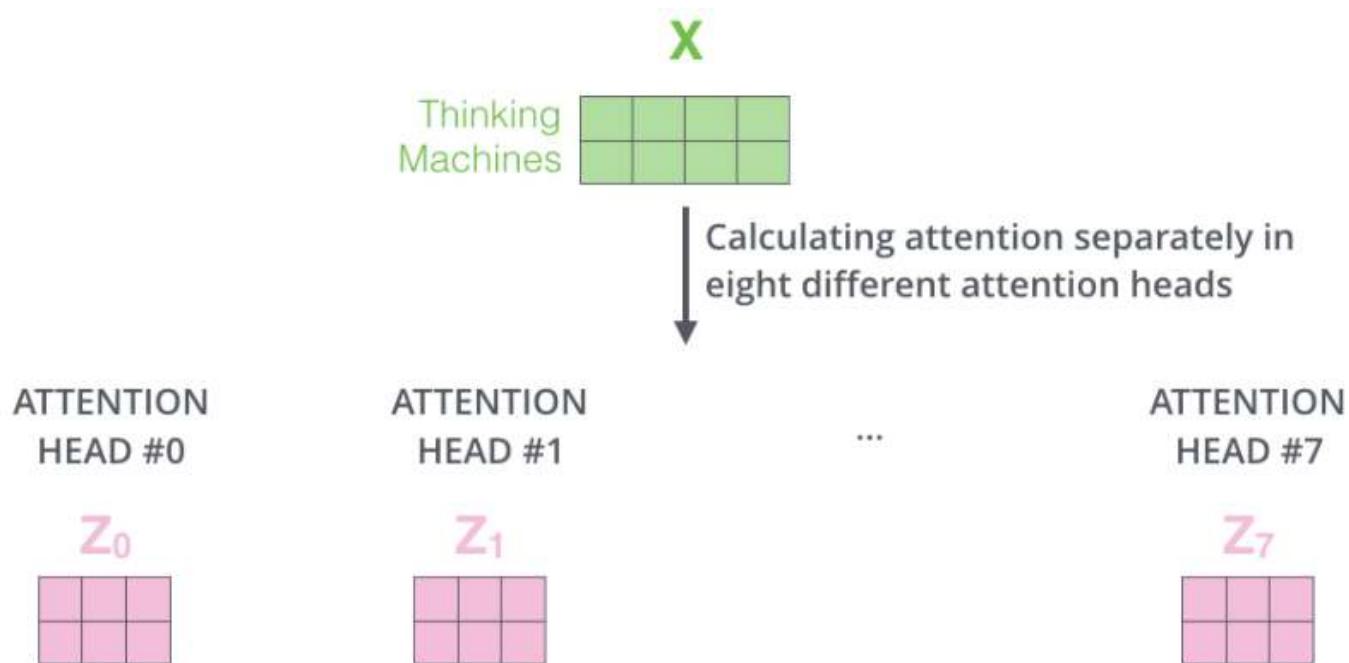
[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalammar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

Transformers : Multi-head Attention



With multi-headed attention, we maintain separate Q , K , V weight matrices for each head resulting in different Q , K , V matrices.

Transformers : Multi-head Attention



- Eight different weight matrices are obtained as an o/p.
- However, the feed-forward layer is not expecting eight matrices – it's expecting a single matrix (a vector for each word).

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalamar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

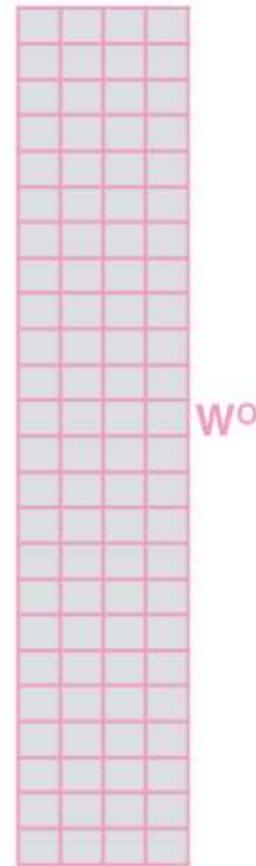
Transformers : Multi-head Attention

1) Concatenate all the attention heads



2) Multiply with a weight matrix W^o that was trained jointly with the model

\times



3) The result would be the Z matrix that captures information from all the attention heads. We can send this forward to the FFNN

$$= \begin{matrix} Z \\ \hline \end{matrix}$$

To tackle the issue, the eight matrices are concatenated and multiplied with additional weight matrix W^o

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalammar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

Transformers : Multi-head Attention

1) This is our input sentence*

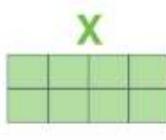
2) We embed each word*

3) Split into 8 heads. We multiply X or R with weight matrices

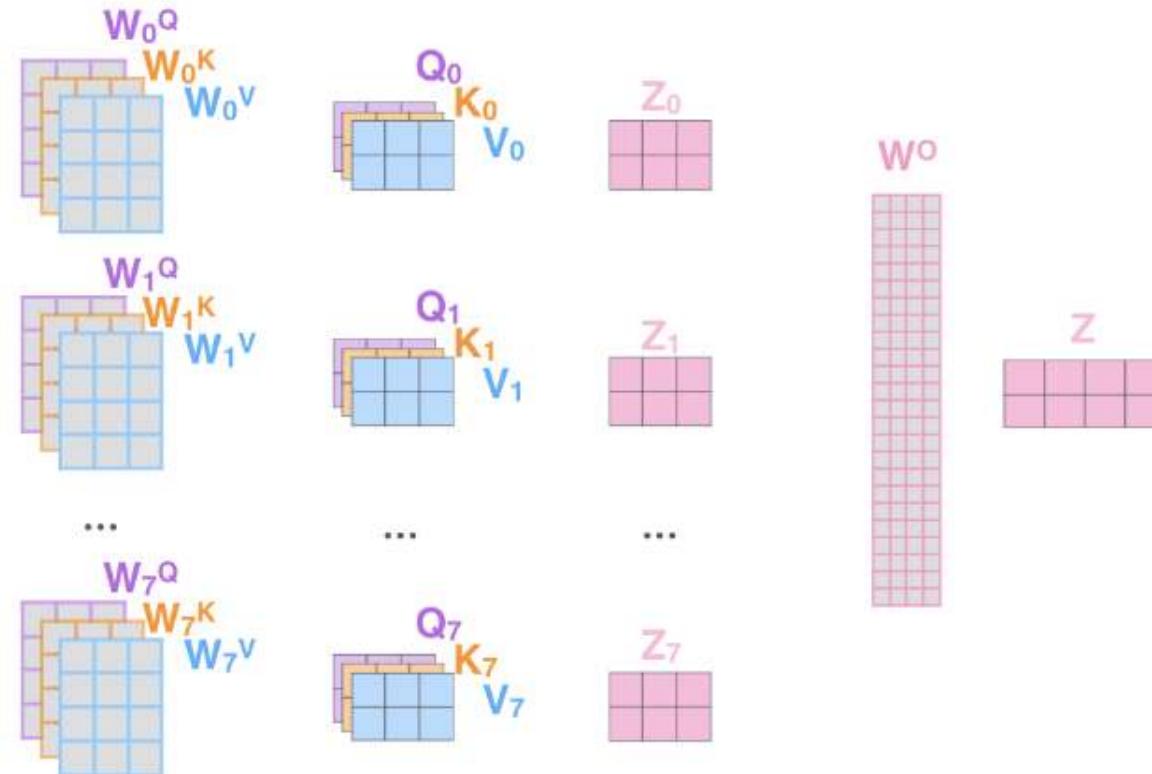
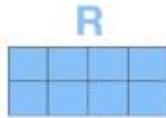
4) Calculate attention using the resulting $Q/K/V$ matrices

5) Concatenate the resulting Z matrices, then multiply with weight matrix W^o to produce the output of the layer

Thinking
Machines

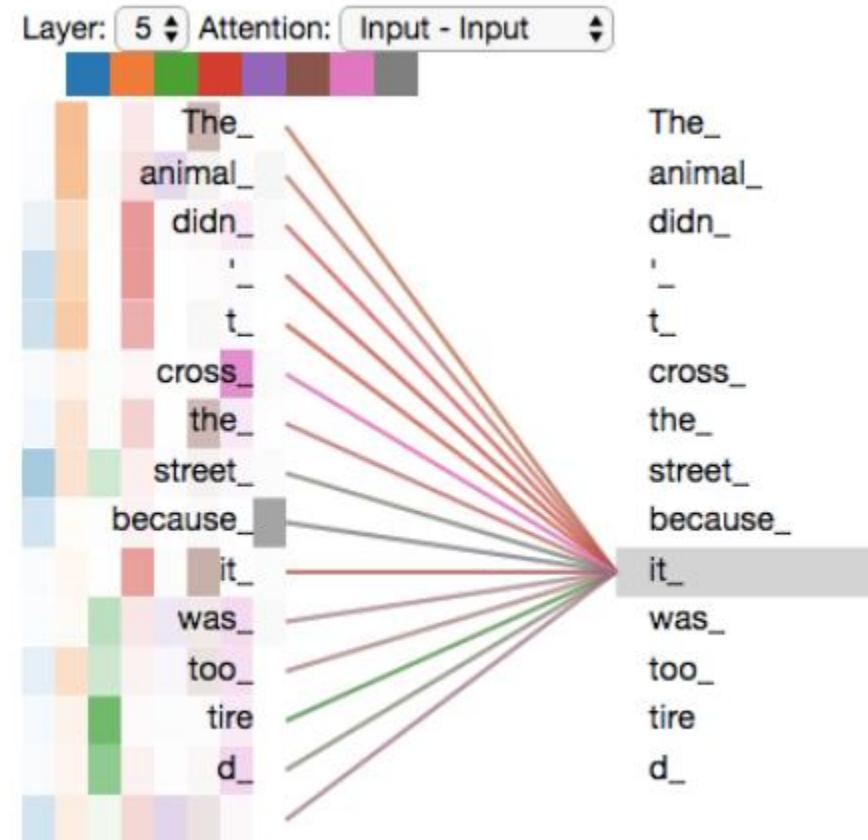
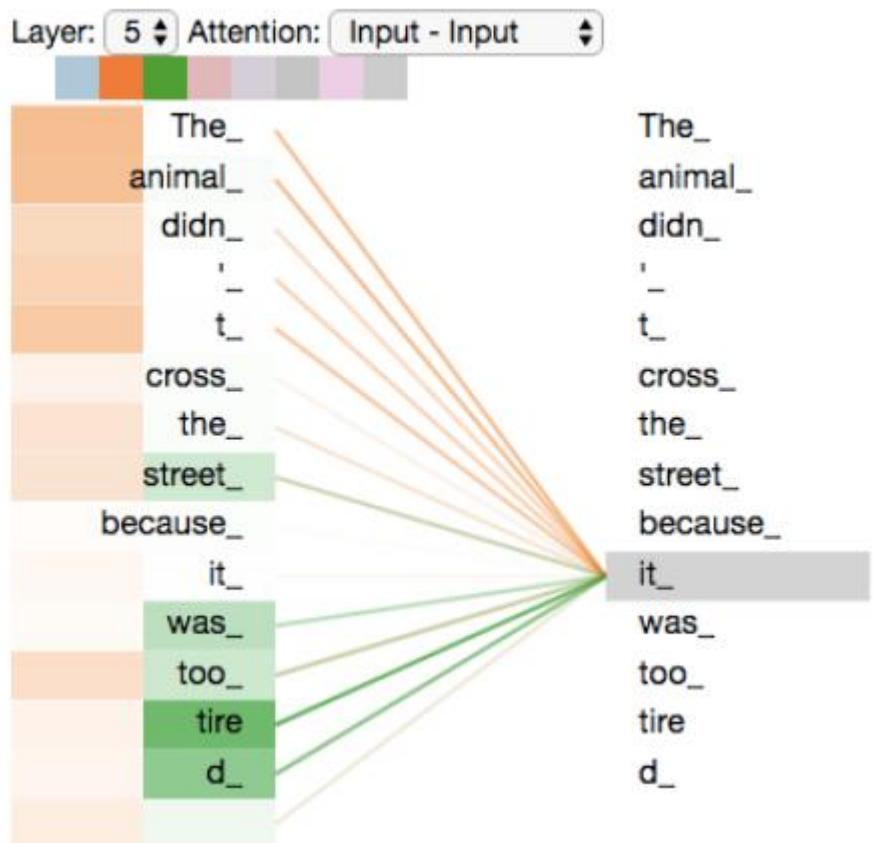


* In all encoders other than #0, we don't need embedding. We start directly with the output of the encoder right below this one



[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalamar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

Transformers : Multi-head Attention

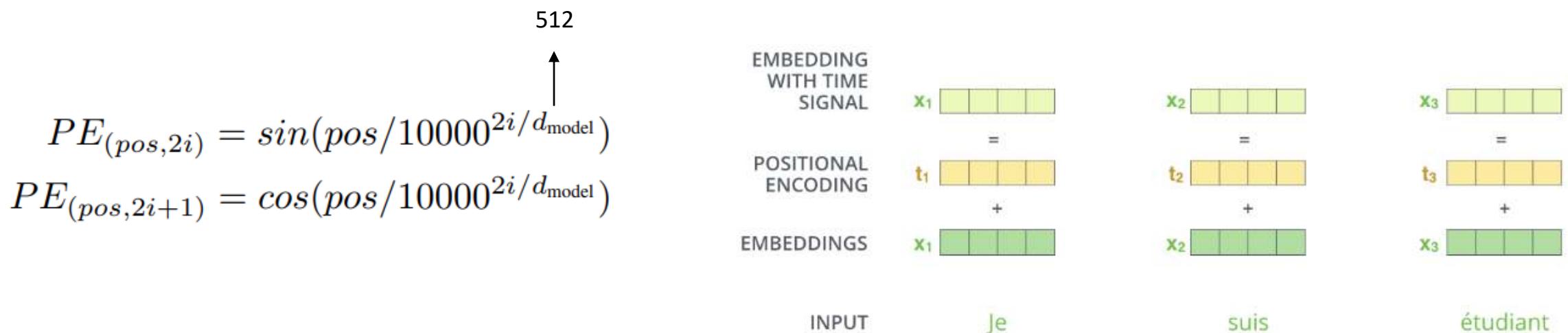


- As we encode the word "it", one attention head is focusing most on "the animal", while another is focusing on "tired" – in a sense, the model's representation of the word "it" bakes in some of the representation of both "animal" and "tired".

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalammar.github.io\)](http://jalammar.github.io)

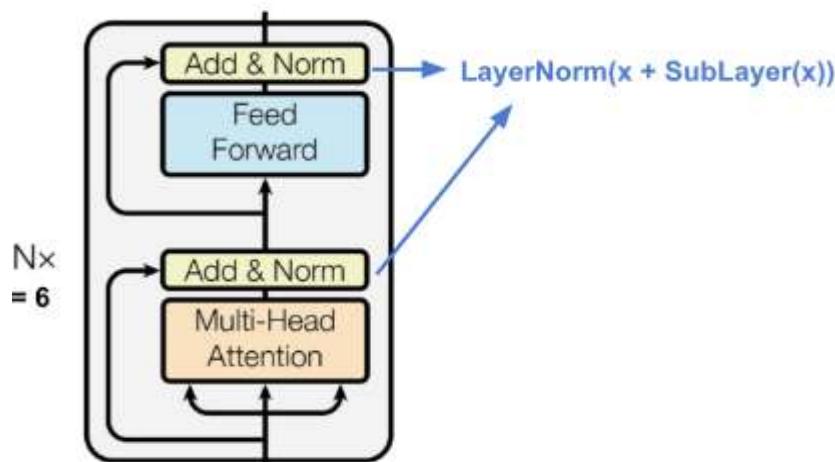
Transformers : Positional Encoding

- Order of the sequence conveys important information for machine translation tasks and language modeling.
- Position encoding is a way to account for the order of words in the input sequence.
- The positional information of the input token in the sequence is added to the input embedding vectors.



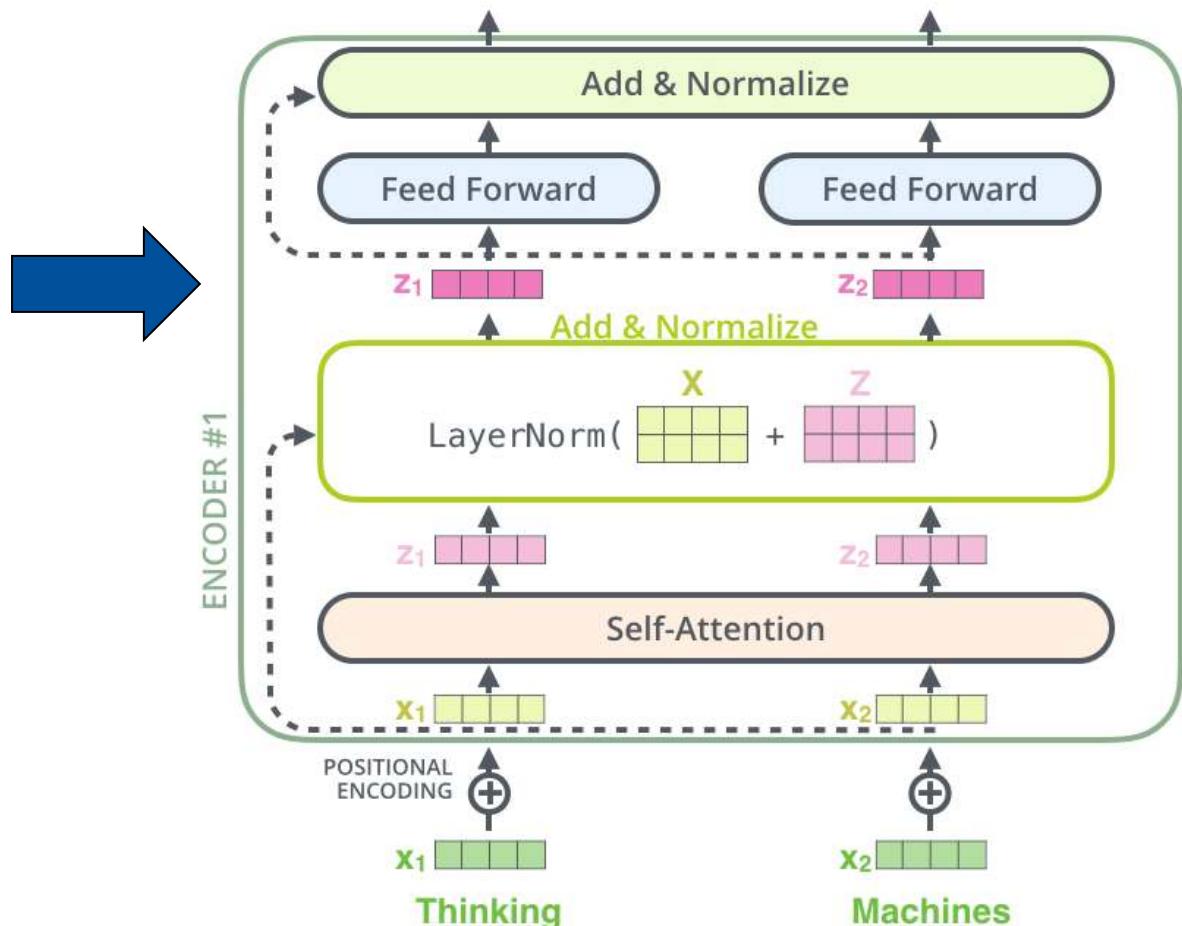
[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalamar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

Transformers : Encoder



Layer Norm

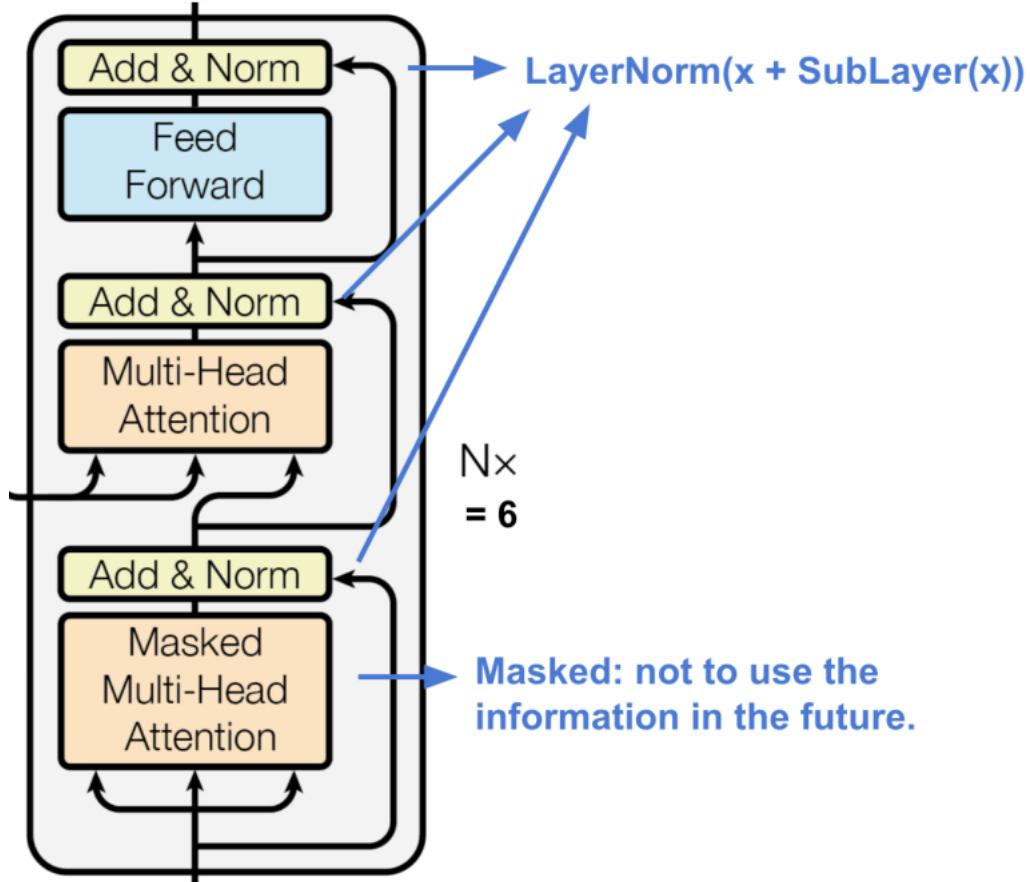
statistics are calculated across all features and all elements (words), for each instance(sentence) independently.



[Attention? Attention! | Lil'Log \(lilianweng.github.io\)](https://lilianweng.github.io/lil-log/2018/05/11/attention.html)

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalammar.github.io\)](https://jalammar.github.io/illustrated-transformer/)

Transformers : Decoder



- **The Encoder**

- processes the input sequence.
- The output of the top encoder is transformed into a set of attention vectors K and V.

- Vectors K & V are used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence

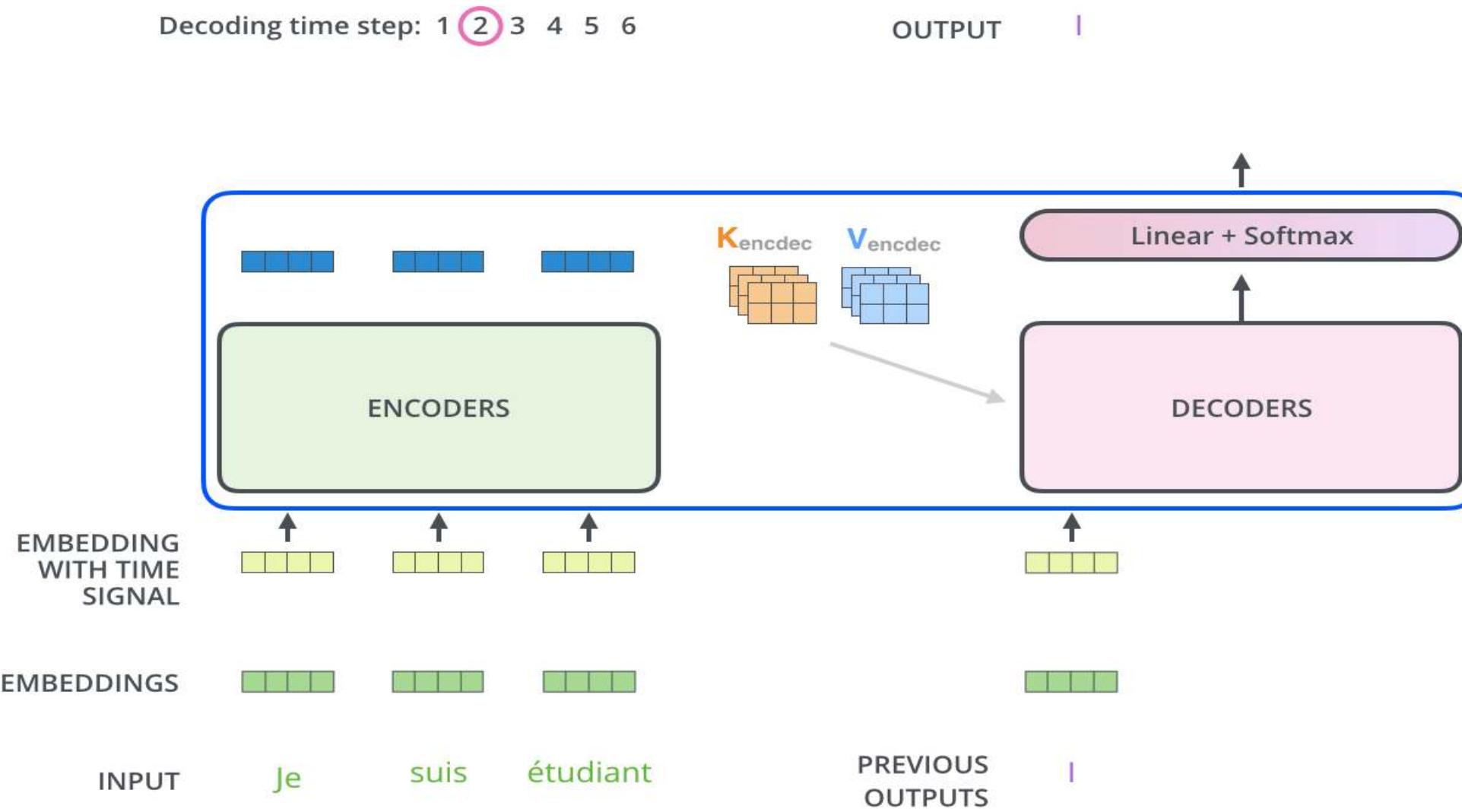
- **The Decoder**

- has masked multi-head attention layer to prevent the positions from seeing subsequent positions
- The “Encoder-Decoder Attention” layer creates its Queries matrix from the layer below it, and takes the Keys and Values matrix from the output of the encoder stack

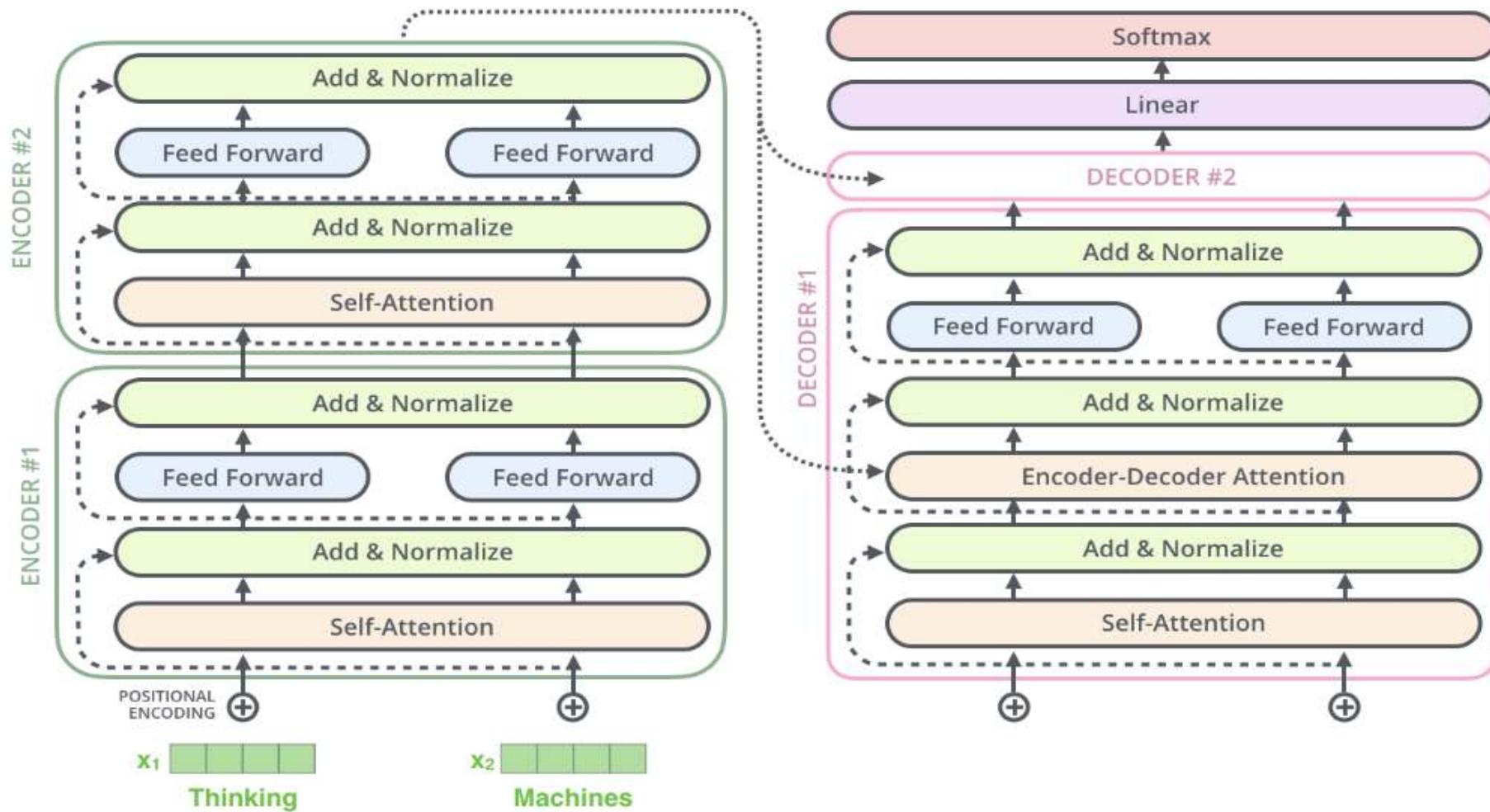
[Attention? Attention! | Lil'Log \(lilianweng.github.io\)](https://lilianweng.github.io/lil-log/2018/05/11/attention-attention.html)

[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalammar.github.io\)](https://jalammar.github.io/illustrated-transformer/)

Transformers: Encoders & Decoders



Transformers : Decoder



[The Illustrated Transformer – Jay Alammar – Visualizing machine learning one concept at a time. \(jalamar.github.io\)](https://jalammar.github.io/the-illustrated-transformer/)

Massive Deep Learning Language models

- **Language models**
 - estimate the probability of words appearing in a sentence, or of the sentence itself existing.
 - building blocks in a lot of NLP applications
- **Massive deep learning language models**
 - pretrained using an enormous amount of unannotated data to provide a general-purpose deep learning model.
 - Downstream users can create task-specific models with smaller annotated training datasets (transfer learning)
- **Tasks executed with BERT and GPT models:**
 - **Natural language inference**
 - enables models to determine whether a statement is true, false or undetermined based on a premise.
 - For example, if the premise is “tomatoes are sweet” and the statement is “tomatoes are fruit” it might be labelled as undetermined.
 - **Question answering**
 - model receives a question regarding text content and returns the answer in text, specifically marking the beginning and end of each answer.
 - **Text classification**
 - is used for sentiment analysis, spam filtering, news categorization

GPT (Generative Pre-Training) by Open AI Pretraining

- Unsupervised learning served as **pre-training** objective for supervised fine-tuned models
 - generative language model using unlabeled data
 - then fine-tuning the model by providing examples of specific downstream tasks like classification, sentiment analysis, textual entailment etc.
- **Semi supervised learning using 3 components**
 1. **Unsupervised Language Modelling (Pre-training)**

$$L_1(T) = \sum_i \log P(t_i | t_{i-k}, \dots, t_{i-1}; \theta)$$

2. where
 1. k is the size of the context window, and ii) conditional probability P is modeled with the help of a neural network (NN) with parameters Θ

GPT (Generative Pre-Training) by Open AI

2. **Supervised Fine-Tuning:** maximising the likelihood of observing label y , given features or tokens x_1, \dots, x_n .

$$L_2(C) = \sum_{x,y} \log P(y|x_1, \dots, x_n)$$

where C was the labeled dataset made up of training examples.

3. **Auxiliary learning objective for supervised fine-tuning to get better generalisation and faster convergence.**

$$L_3(C) = L_2(C) + \lambda L_1(C)$$

where $L_1(C)$ was the auxiliary objective of learning language model

λ was the weight given to this secondary learning objective. λ was set to 0.5.

- Supervised fine-tuning is achieved by adding a linear and a softmax layer to the transformer model to get the task labels for downstream tasks.
- “zero-shot” framework
 - measure a model’s performance having *never* been trained on the task.

GPT-n series created by OpenAI (2018 onwards)

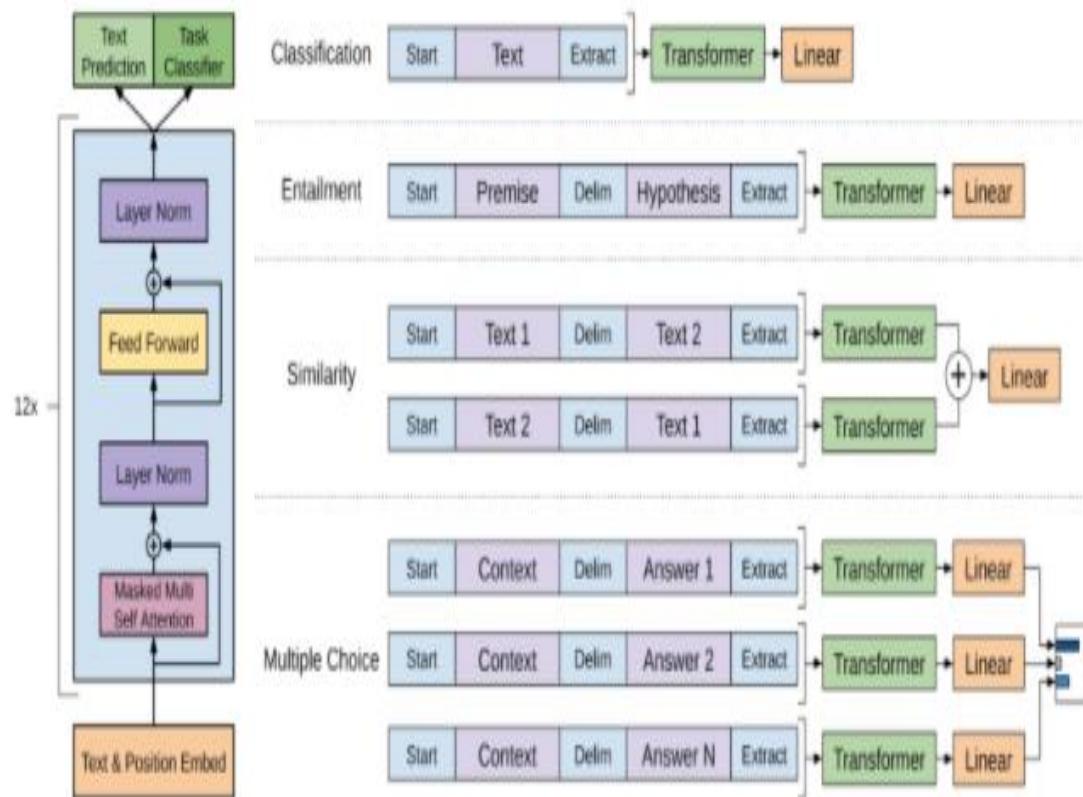


Figure 1: (left) Transformer architecture and training objectives used in this work. (right) Input transformations for fine-tuning on different tasks. We convert all structured inputs into token sequences to be processed by our pre-trained model, followed by a linear+softmax layer.

- **Generative** models are a type of statistical model that are used to generate new data points.
 - learn the underlying relationships between variables in a dataset in order to generate new data points similar to those in the dataset.
- Trained on BooksCorpus dataset contains over 7,000 unique unpublished books from a variety of genres
- 12-layer decoder-only transformer with masked self-attention heads
- GPT-2
 - Application - generate long passages of coherent text
 - <https://transformer.huggingface.co/doc/gpt2-large>

- step to Artificial General Intelligence(AGI)
- "I am open to the idea that a worm with 302 neurons is conscious, so I am open to the idea that GPT-3 with 175 billion parameters is conscious too." — David Chalmers*
- 3rd-gen language prediction model in capacity of **175 billion** parameters.
 - trained with 499 Billion tokens
 - trained using next word prediction
 - Context window size was increased from 1024 for GPT-2 to 2048 tokens for GPT-3.
 - Size of word embeddings was increased to 12888 for GPT-3 from 1600 for GPT-2.
 - To train models of different sizes, the batch size is increased according to number of parameters, while the learning rate is decreased accordingly.

Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

zero-shot task transfer OR meta learning

- The model is supposed to understand the task based on the examples and instruction.
- For English to French translation task, the model was given an English sentence followed by the word French and a prompt (:)

Zero-shot

The model predicts the answer given only a natural language description of the task. No gradient updates are performed.



One-shot

In addition to the task description, the model sees a single example of the task. No gradient updates are performed.



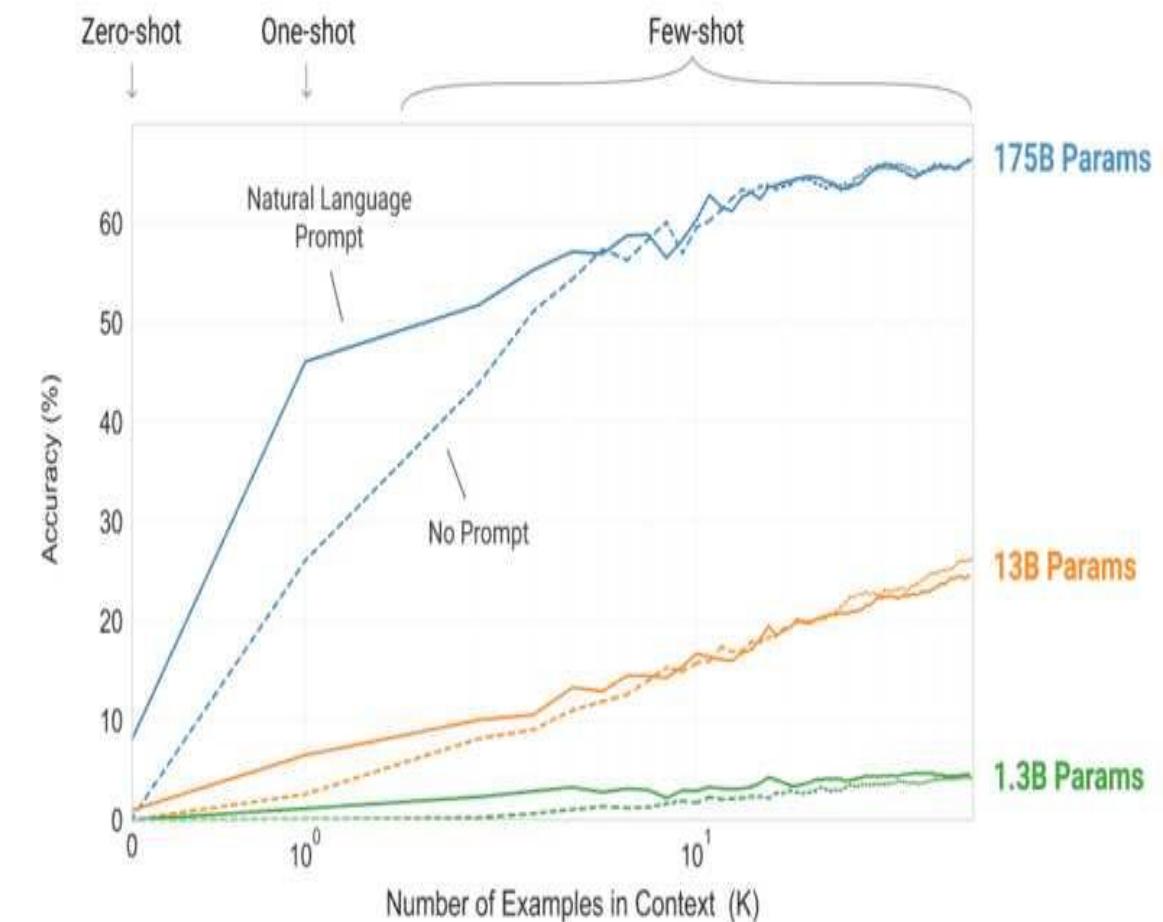
Few-shot

In addition to the task description, the model sees a few examples of the task. No gradient updates are performed.



GPT-3 Task Agnostic Model

- **LIMITATIONS OF GPT-3**
- GPT-3 can perform a wide variety of operations such as compose prose, write code and fiction, business articles
- Does not have any internal representation of what these words even mean. misses the *semantically grounded model* of the topics on which it discusses.
- If the model is faced with data that is not in a similar form or is unavailable from the Internet's corpus of existing text that was used initially in the training phase, then the language generated is a loss.
- Expensive and complex inferencing due to hefty architecture, less interpretability of the language, and uncertainty around what helps the model achieve its few-shot learning behavior.
- The text generated carries bias of the language it is initially trained on.
- The articles, blogs, memos generated by GPT-3 may face gender, ethnicity, race, or religious bias.
- model is capable of producing high-quality text, sometimes loses coherence with data while generating long sentences and thus may repeat sequences of text again and again in a paragraph.

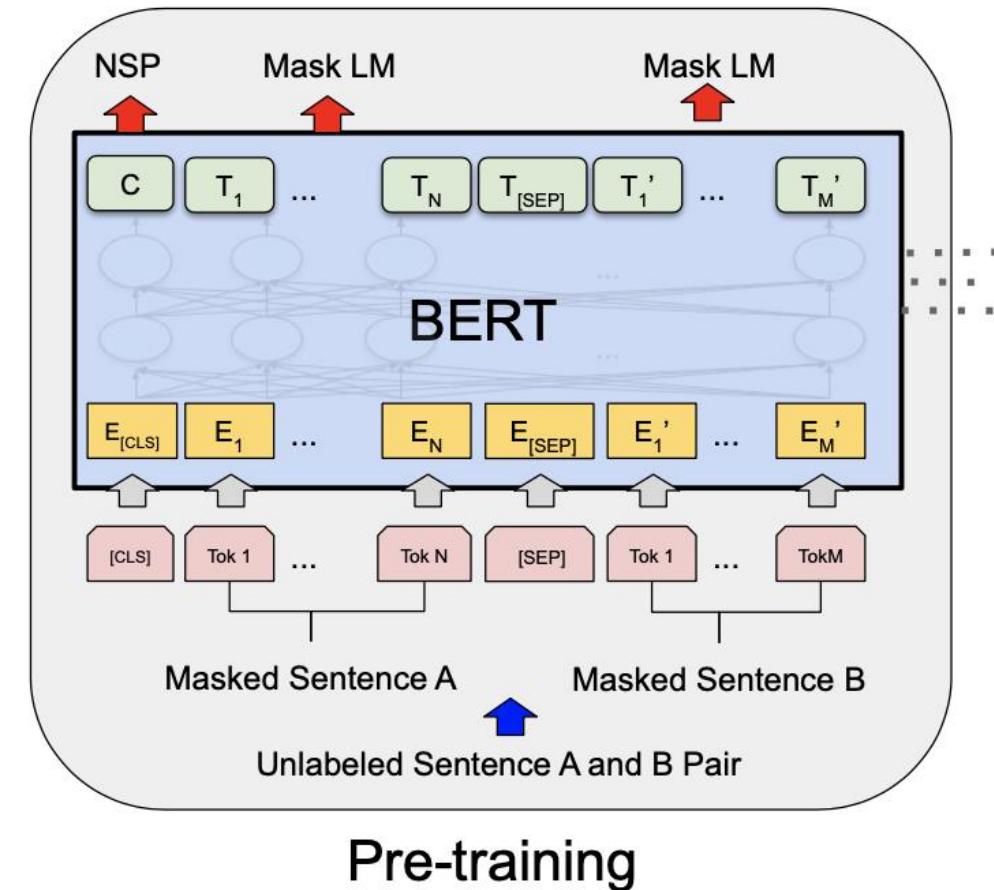


BERT (Bidirectional Encoder Representations from Transformers) by google

- “BERT stands for Bidirectional Encoder Representations from Transformers. It is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks.”
- Two variants
 - BERT Base: 12 layers (transformer blocks), 12 attention heads, and 110 million parameters
 - BERT Large: 24 layers (transformer blocks), 16 attention heads and, 340 million parameters
- **BERT is pre-trained on two NLP tasks:**
 - Masked Language Modeling
 - replace 15% of the input sequence with [MASK] and model learns to detect the masked word
 - Next Sentence Prediction
 - two sentences A and B are separated with the special token [SEP] and are formed in such a way that 50% of the time B is the actual next sentence and 50% of the time is a random sentence.

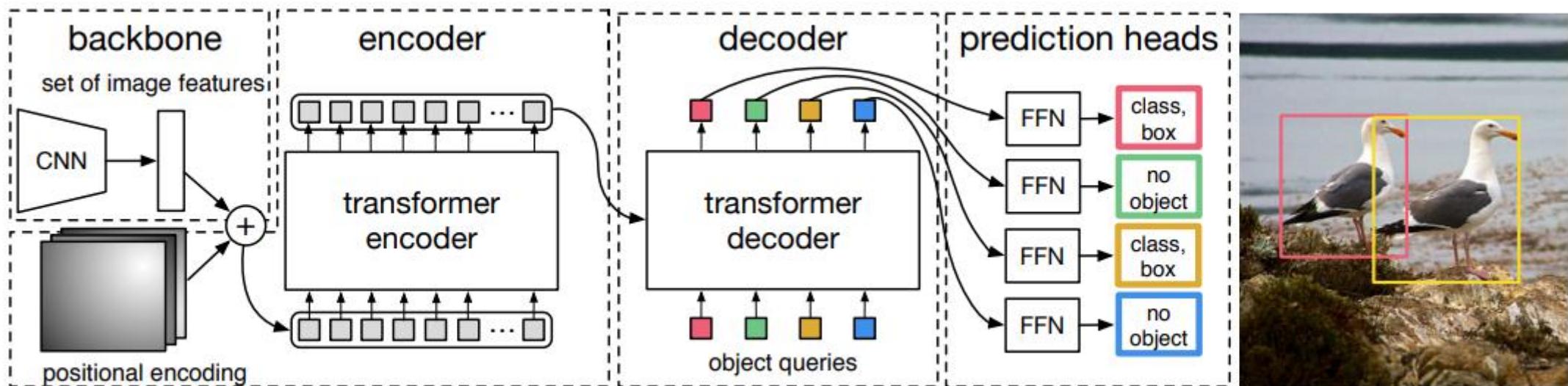
BERT (Bidirectional Encoder Representations from Transformers) by google

- every input embedding is a combination of 3 embeddings:
 - Position Embeddings: captures “sequence” or “order” information
 - Segment Embeddings: can also take sentence pairs as inputs for tasks (Question-Answering)
 - Token Embeddings: learned for the specific token from the WordPiece token vocabulary
- Output is an embedding since it has only encoder and no decoder



Transformers in Computer Vision

DEtection TRansformer (DETR)

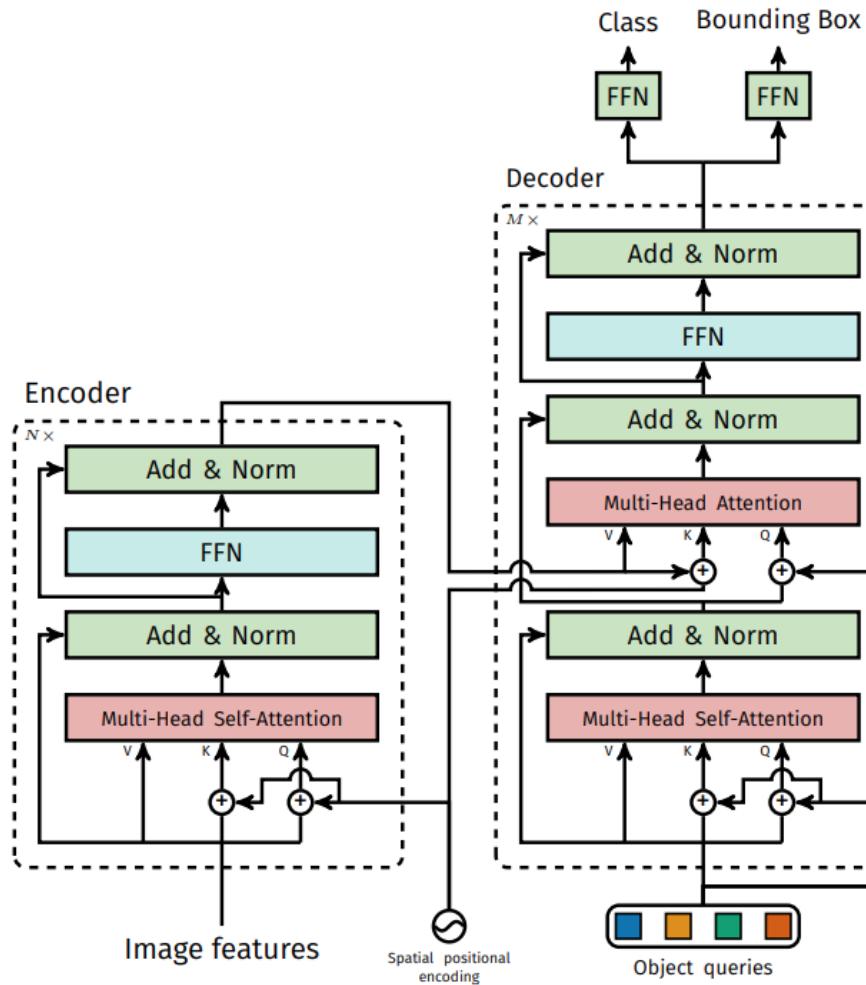


<https://github.com/facebookresearch/detr>

Carion, Nicolas, et al. "End-to-end object detection with transformers." *European conference on computer vision*. Springer, Cham, 2020.

Transformers in Computer Vision

DEtection TRansformer (DETR)



<https://github.com/facebookresearch/detr>

Carion, Nicolas, et al. "End-to-end object detection with transformers." *European conference on computer vision*. Springer, Cham, 2020.

Transformers in Computer Vision

DEtection TRansformer (DETR): Results on COCO 2017 dataset (AP = Average Precision)

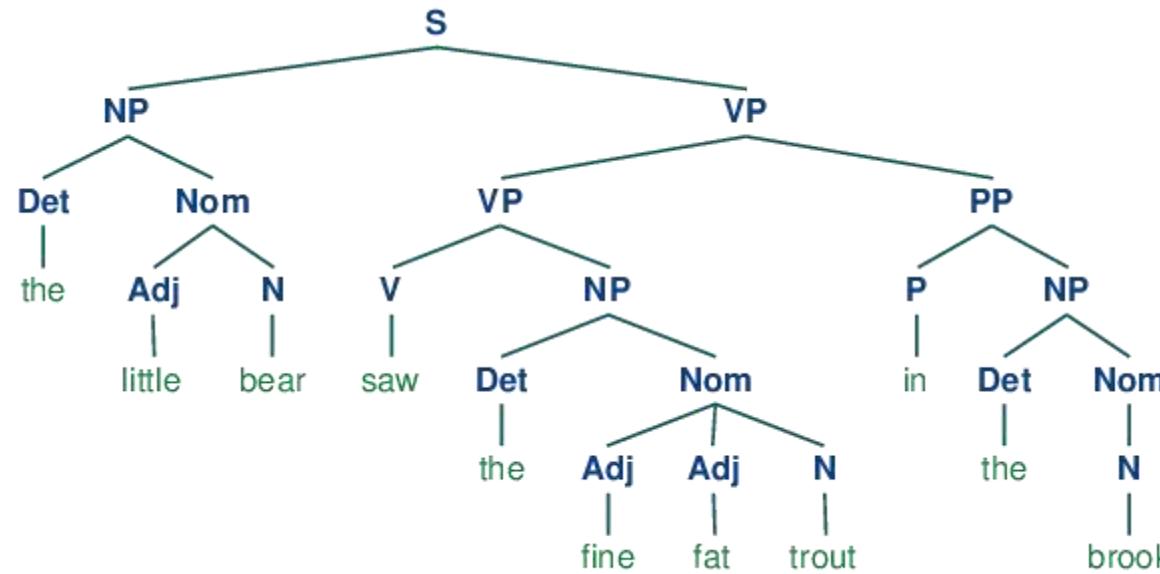
Model	GFLOPS/FPS	#params	AP	AP ₅₀	AP ₇₅	AP _S	AP _M	AP _L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	47.8	27.2	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	44.9	64.7	47.7	23.7	49.5	62.3

<https://github.com/facebookresearch/detr>

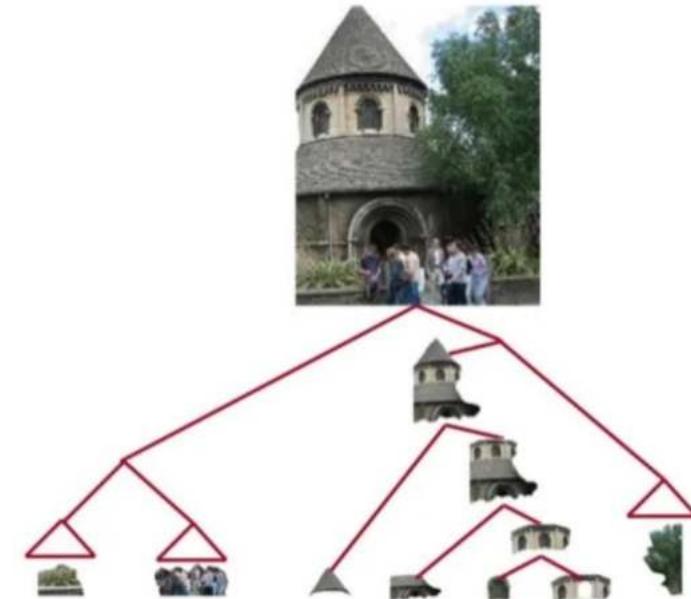
Carion, Nicolas, et al. "End-to-end object detection with transformers." *European conference on computer vision*. Springer, Cham, 2020.

Recursive Neural Networks

- Many real world entities have recursive structure.



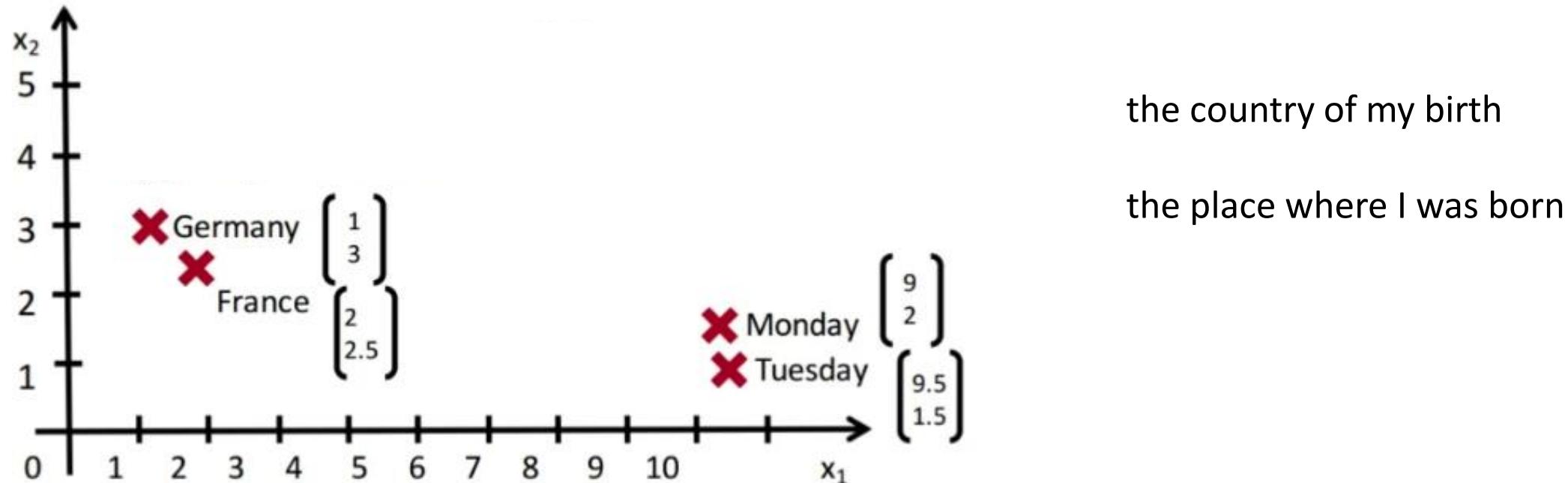
Eg: A syntactic tree structure representing a sentence.



Eg: A tree representation of different segments in an image

Recursive Neural Networks: Introduction

- Can we learn a good representation for these recursive structures?

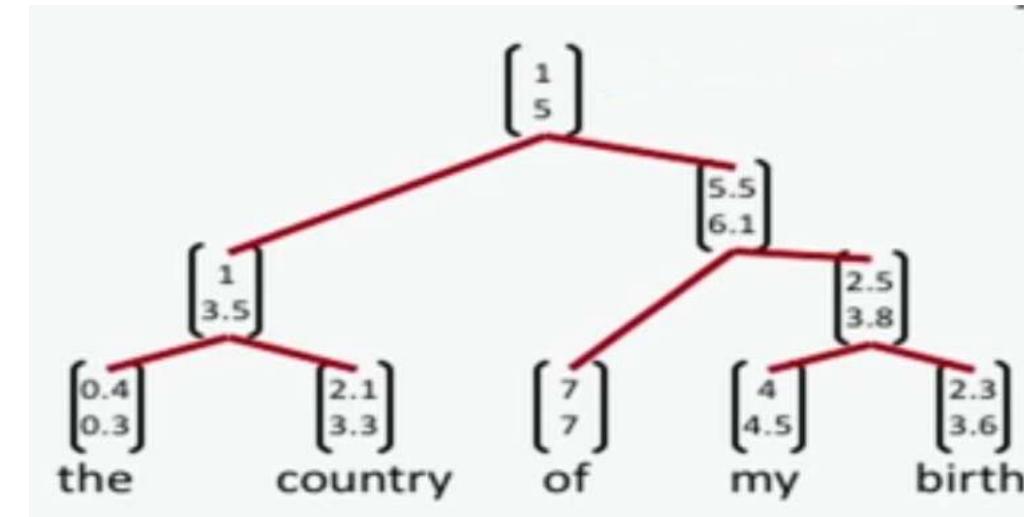
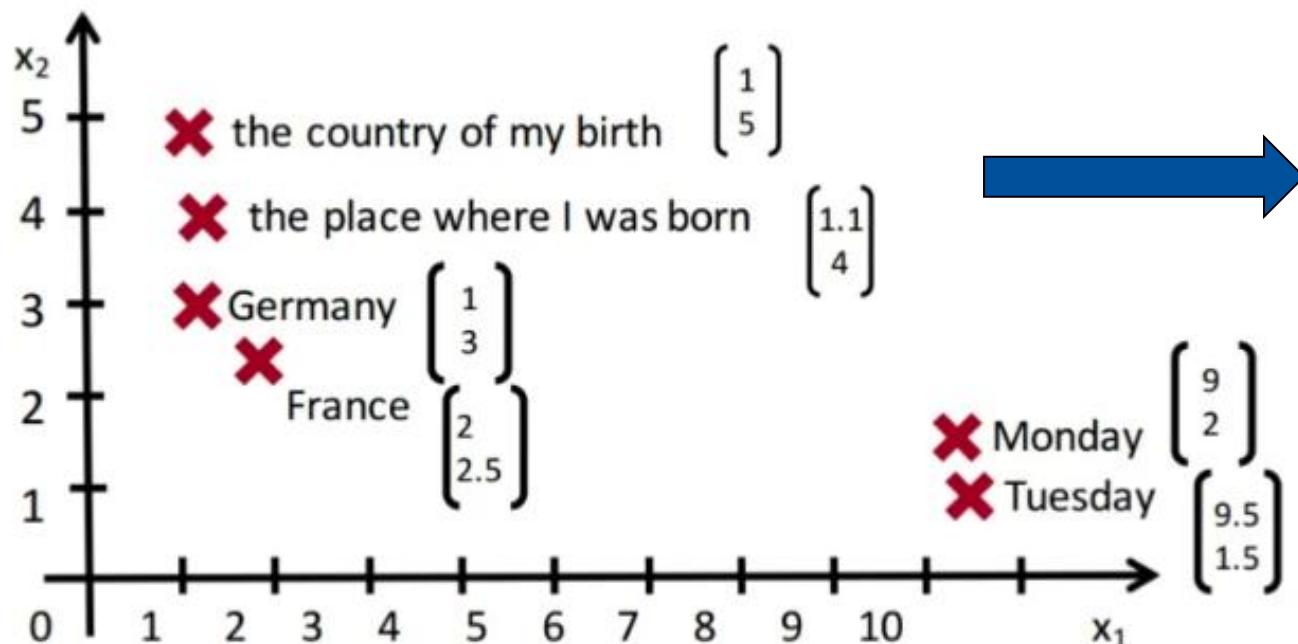


Eg: Word phrase representations (learning representations of phrases of arbitrary length)

Source: Socher et al. Parsing Natural Scenes and Natural Language with Recursive Neural Networks, ICML'11

Recursive Neural Networks: Introduction

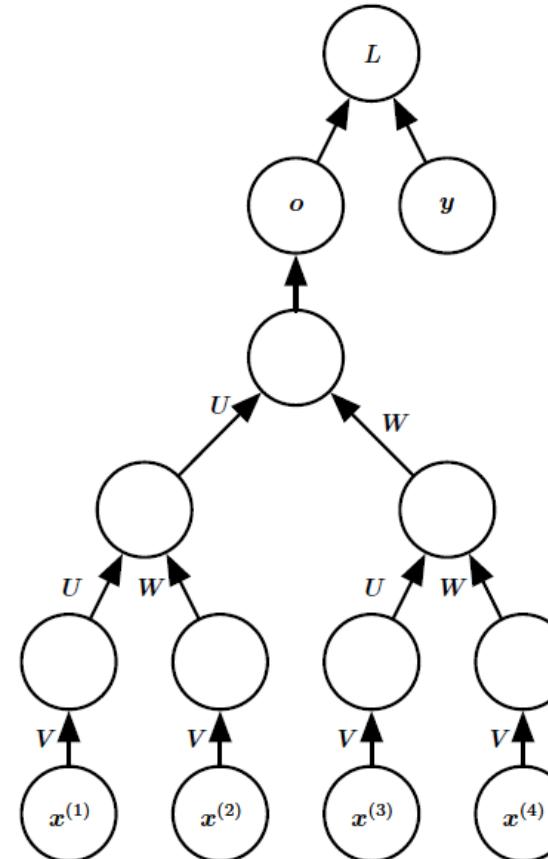
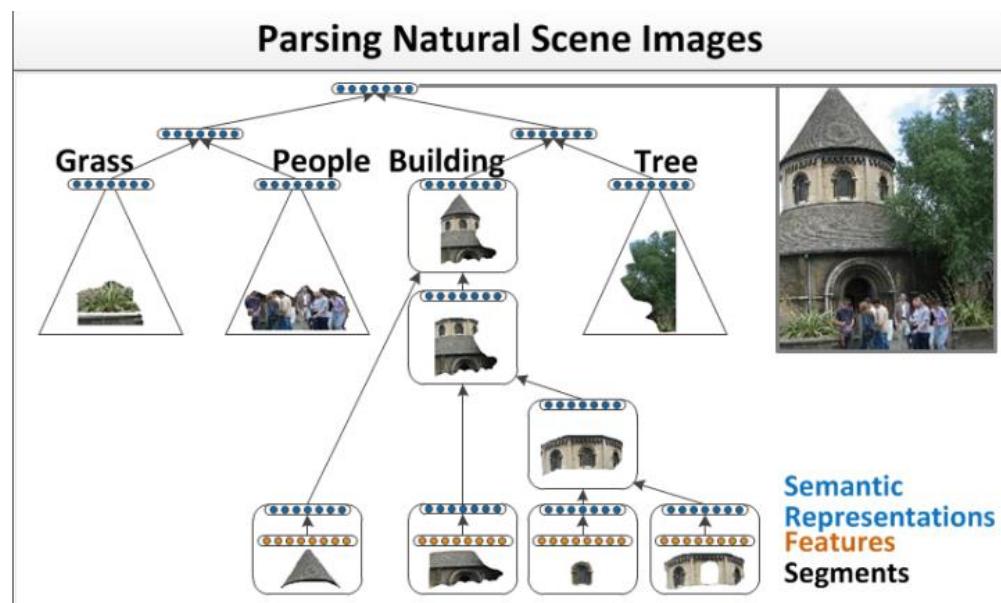
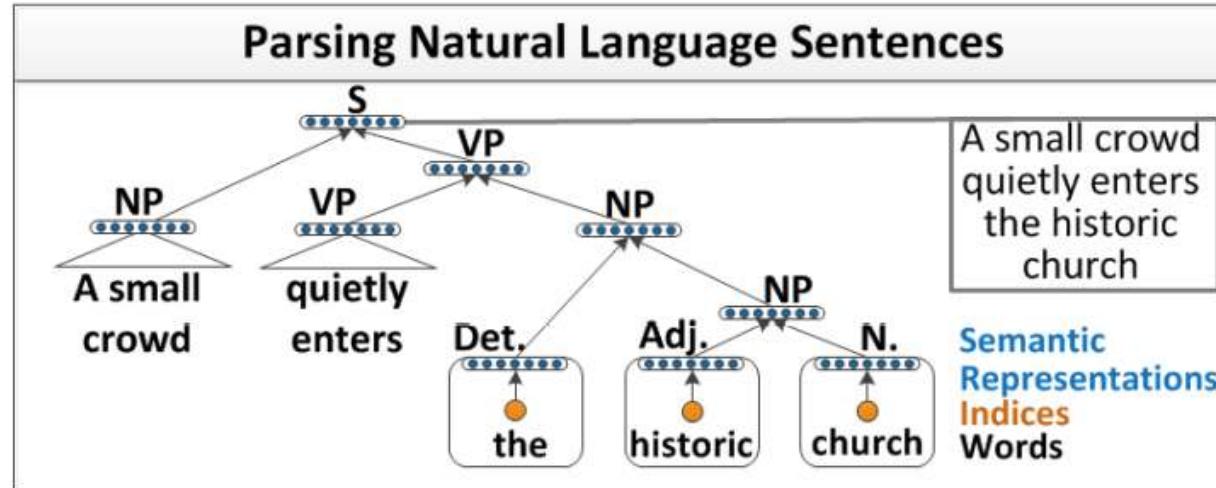
- Can we learn a good representation for these recursive structures?



- The meaning of a sentence is determined by meaning of its words and the rules that combine them.

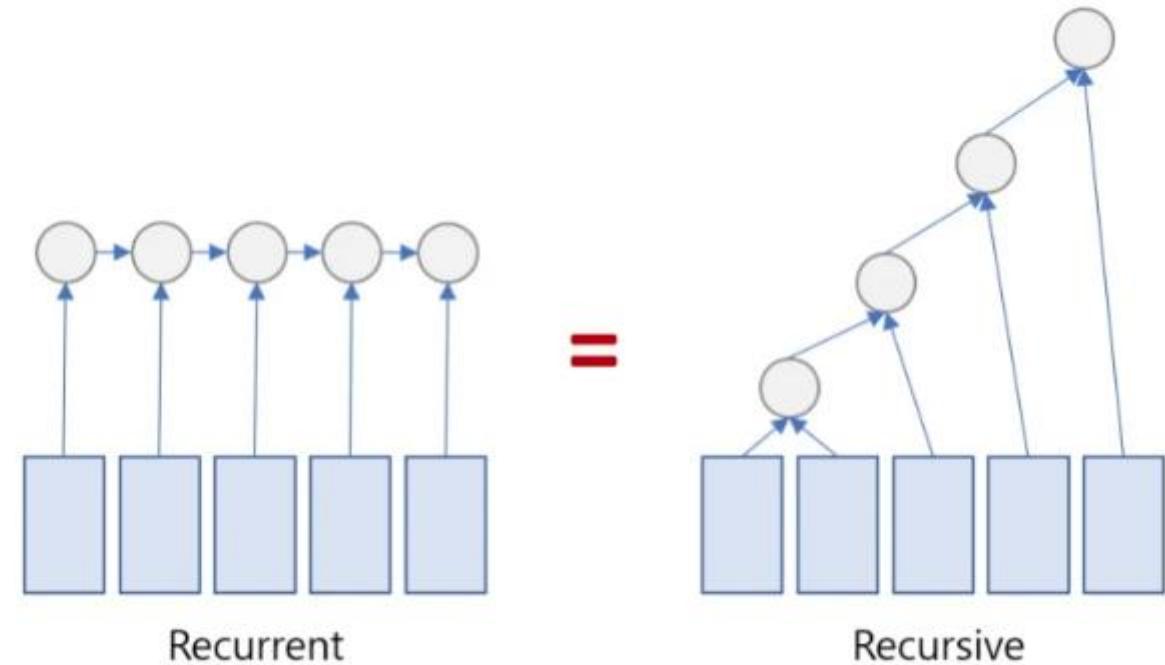
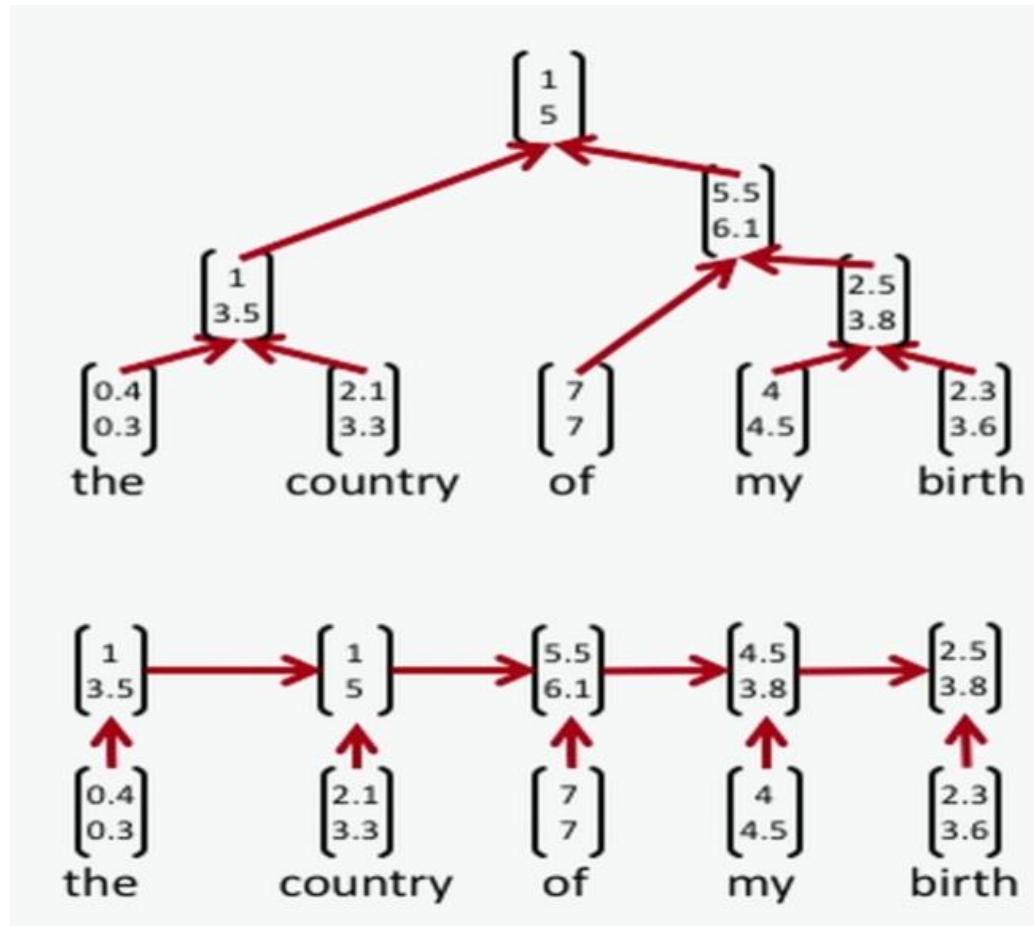
Source: Socher et al. Parsing Natural Scenes and Natural Language with Recursive Neural Networks, ICML'11

Recursive Neural Networks



Source: Ian Goodfellow et al. "Deep Learning" MIT Press'15

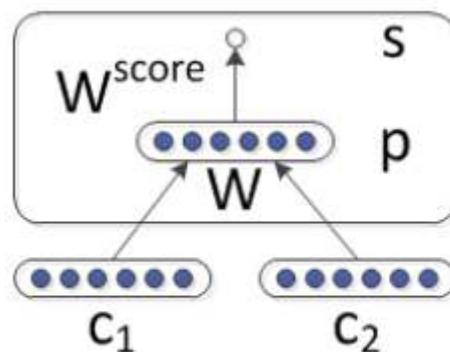
Recursive Neural Networks vs Recurrent Neural Networks



Source: Socher et al. Parsing Natural Scenes and Natural Language with Recursive Neural Networks, ICML'11

Recursive Neural Networks

- To build a recursive neural network, we need two things:
 - A model that merges pairs of representations.
 - A model that determines the tree structure.



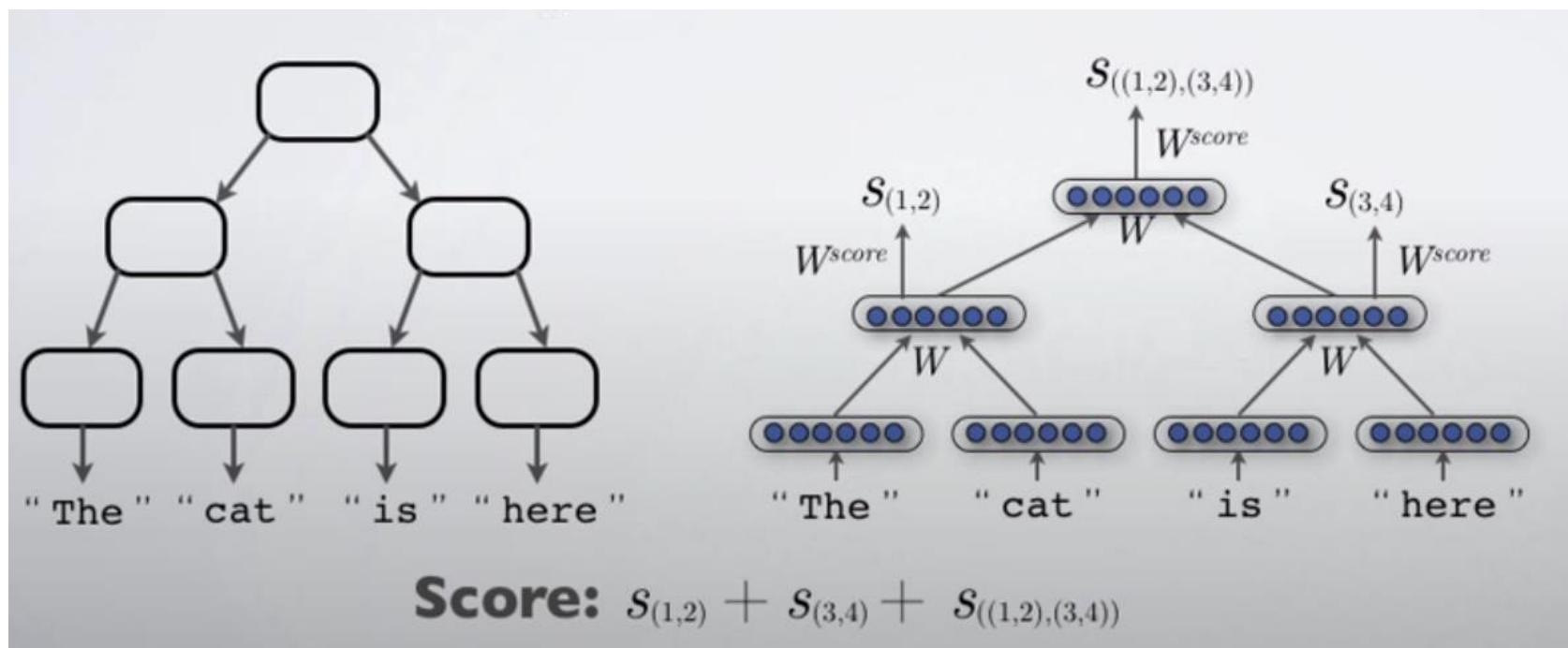
$$\begin{aligned} s &= W^{score} p \\ p &= f(W[c_1; c_2] + b) \end{aligned}$$

A score to determine which pairs of representations to merge first

Source: Socher et al. Parsing Natural Scenes and Natural Language with Recursive Neural Networks, ICML'11

Recursive Neural Networks

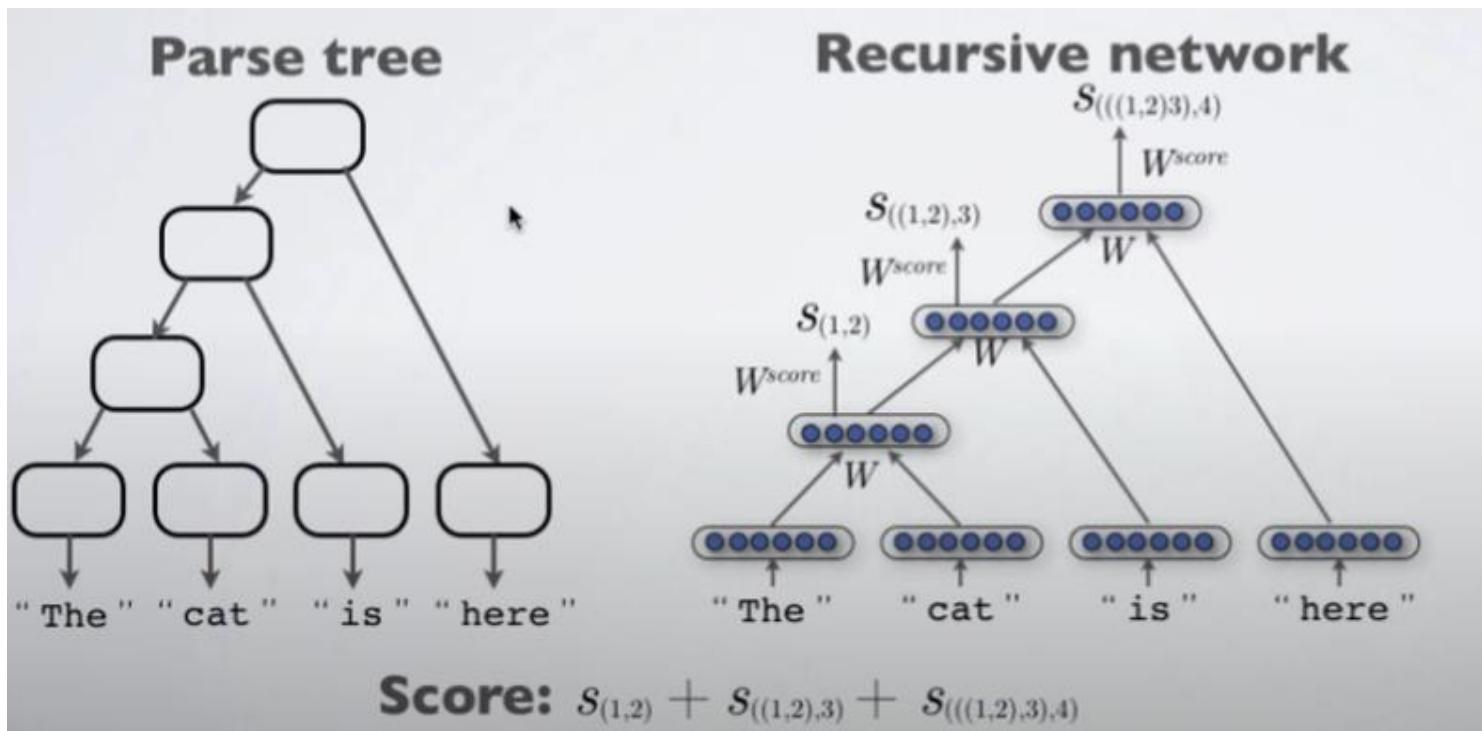
- To build a recursive neural network, we need two things:
 - A model that merges pairs of representations.
 - A model that determines the tree structure.



Source: Socher et al. Parsing Natural Scenes and Natural Language with Recursive Neural Networks, ICML'11

Recursive Neural Networks

- To build a recursive neural network, we need two things:
 - A model that merges pairs of representations.
 - A model that determines the tree structure.



- Approximate the best tree by locally maximizing each subtree

Source: Socher et al. Parsing Natural Scenes and Natural Language with Recursive Neural Networks, ICML'11

Autoencoders

DSE 3151 DEEP LEARNING

Dr. Rohini Rao & Dr. Abhilash K Pai

Department of Data Science and Computer Applications

MIT Manipal

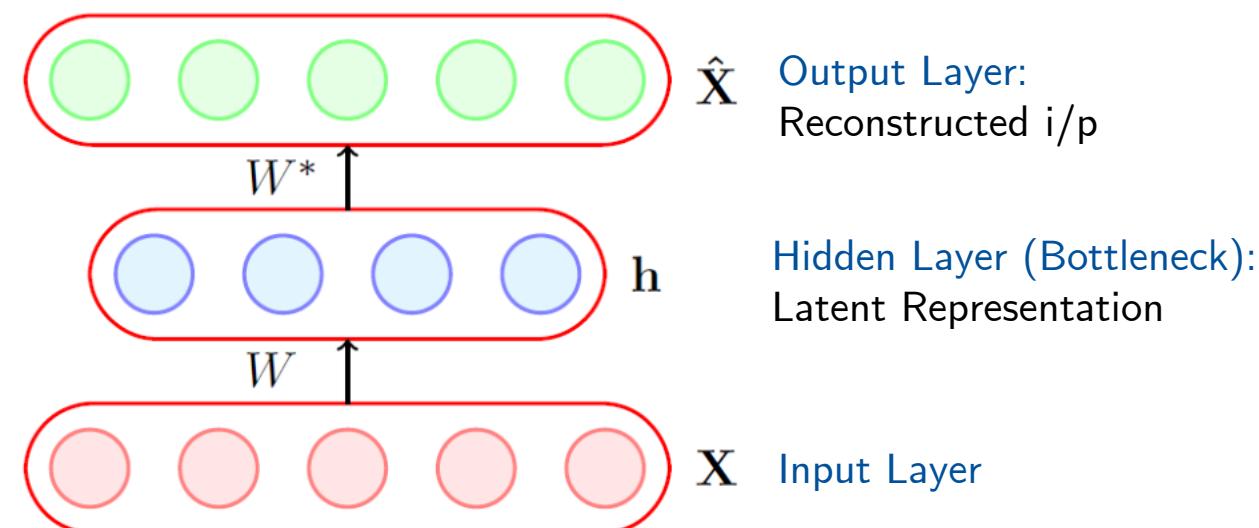
Unsupervised Learning

Unsupervised learning : only use the inputs $X^{(t)}$ for learning.

- Automatically extract meaningful features/representations for the data.
- Compress data while maintaining the structure and complexity of the original dataset (dimensionality reduction).
- Leverage the availability of unlabeled data (which can be used for unsupervised pre-training).
- Some well-known neural networks for unsupervised learning are:
 - Restricted Boltzmann Machines (RBM)
 - Sparse Coding Model
 - **Autoencoders**

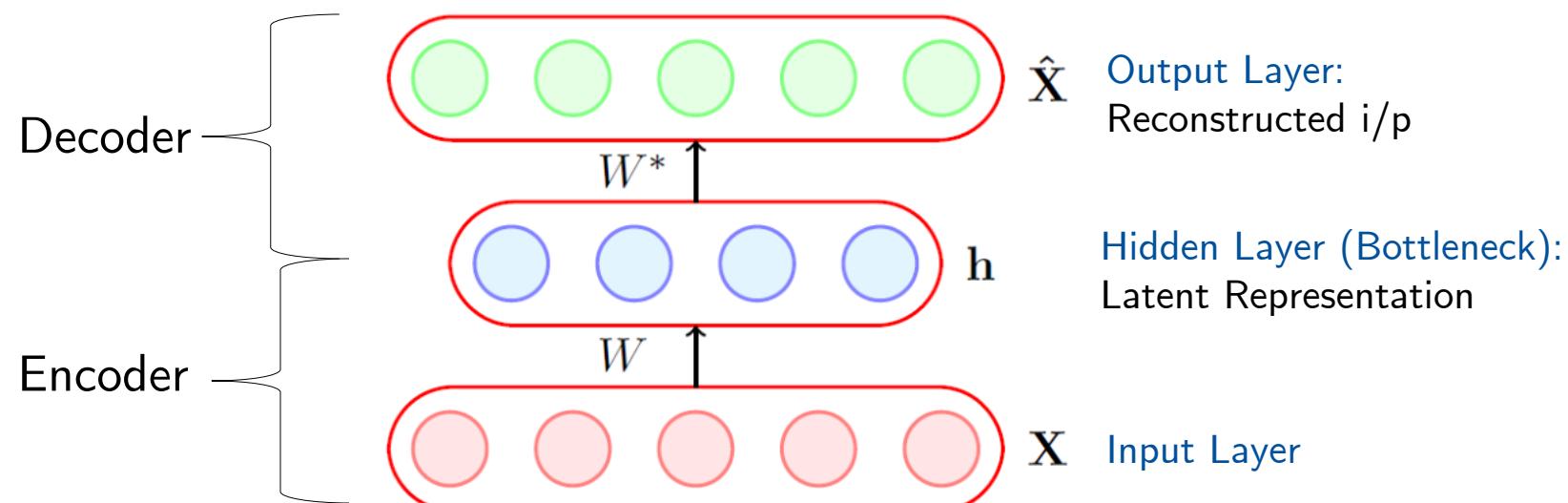
Autoencoders: Introduction

- Autoencoders are special type feed forward neural networks which are trained to reproduce their input at the output layer.

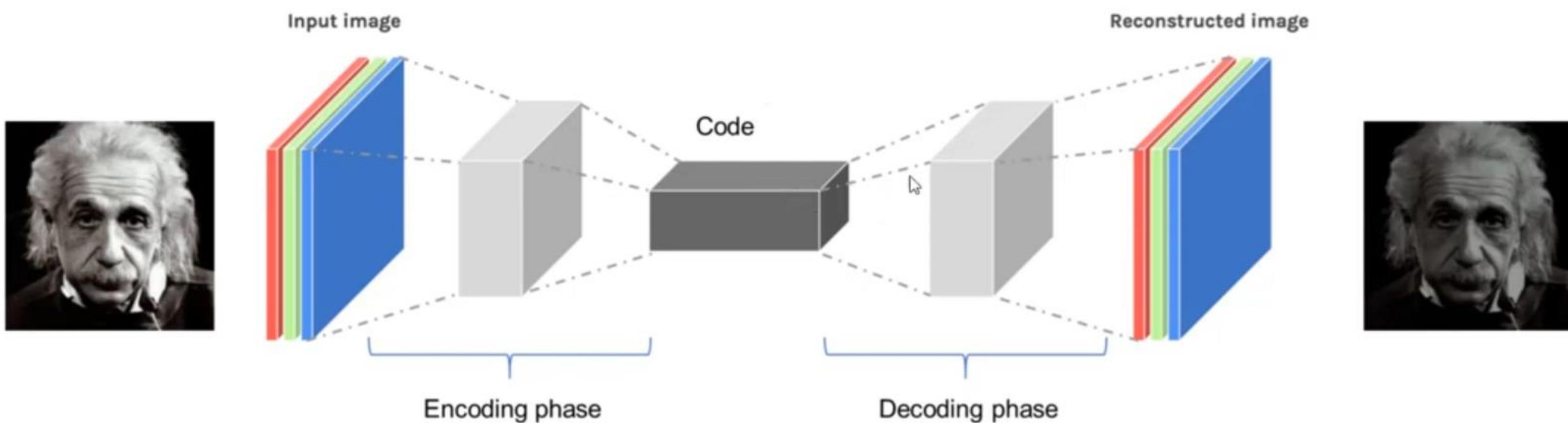


Autoencoders: Introduction

- It consists of an **ENCODER** that encodes its **input X** into a **hidden representation h** , and a **DECODER** which decodes the input again from this hidden representation.

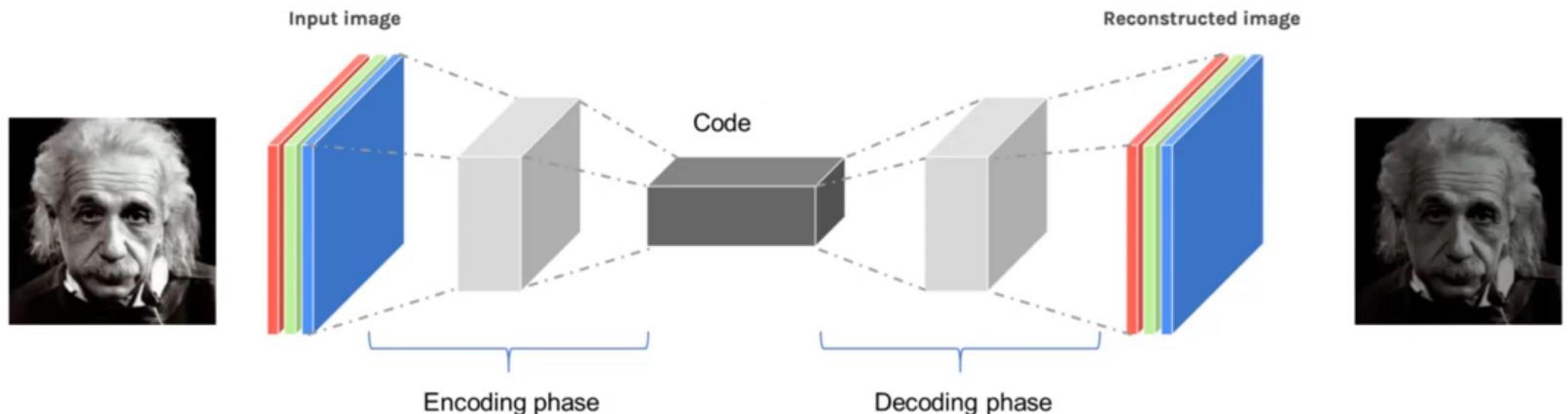


Autoencoders: Introduction



Source: [85b - An introduction to autoencoders - in Python](#)

Autoencoders: Introduction



```
model.add(Conv2D(8, (3, 3), activation='relu', padding='same'))  
model.add(MaxPooling2D((2, 2), padding='same'))
```

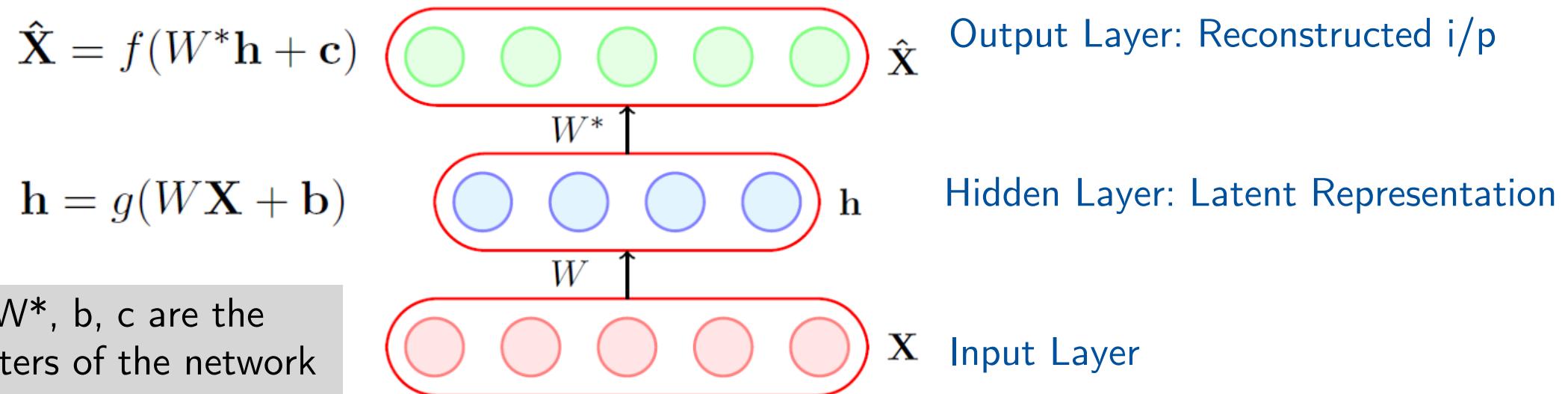
```
model.fit(x, x, epochs=100, shuffle=True)
```

```
model.add(Conv2D(8, (3, 3), activation='relu', padding='same'))  
model.add(UpSampling2D((2, 2)))
```

Source: [85b - An introduction to autoencoders - in Python](#)

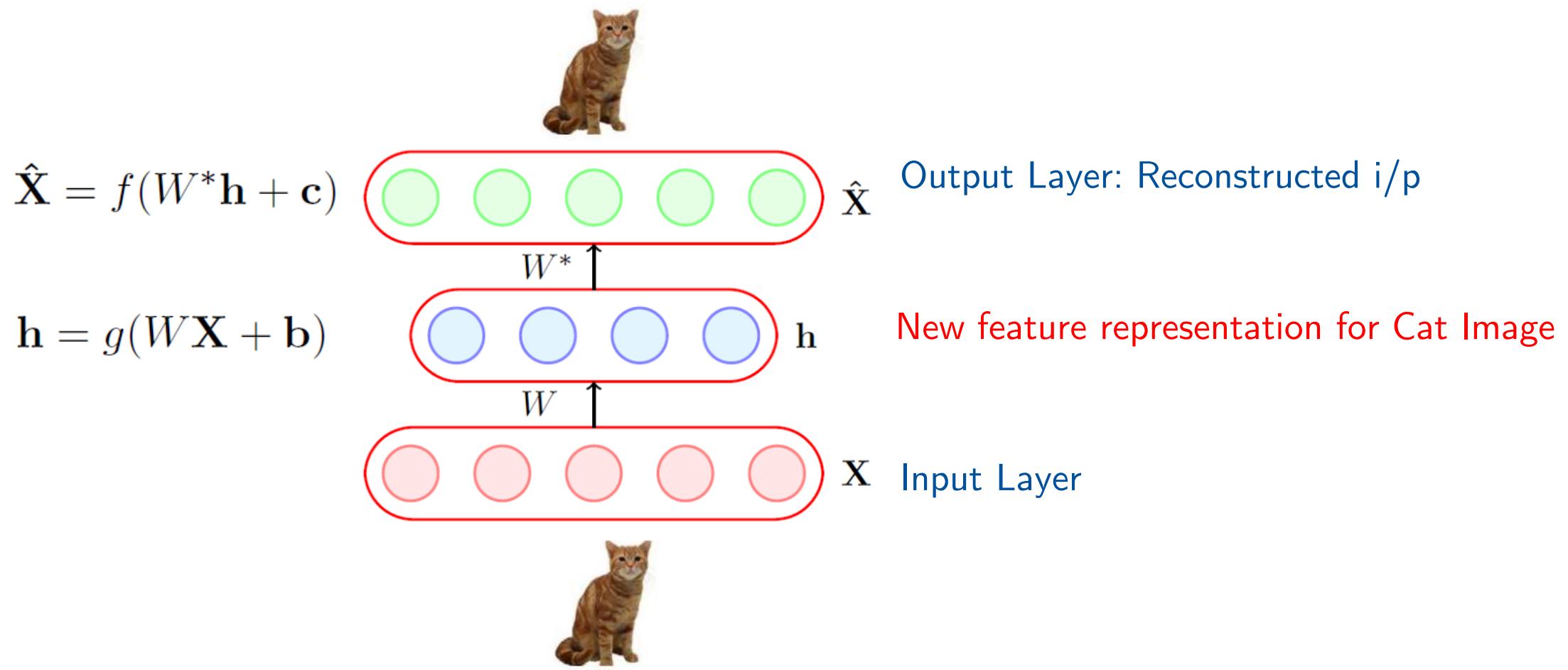
Autoencoders: Introduction

- The autoencoder model is trained to minimize a certain loss function which will ensure that \mathbf{X} is close to $\hat{\mathbf{X}}$



- As the hidden layer could produce a reduced representation of the input, autoencoders (as the one shown above) can be used for dimensionality reduction.

Autoencoders: Introduction



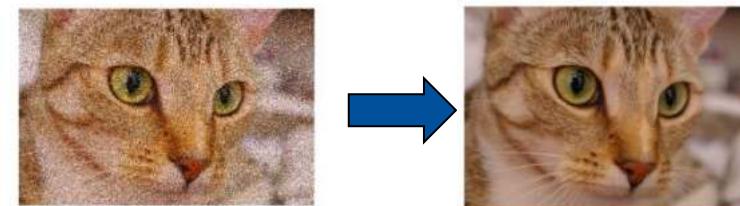
- Also, the latent representation can be used as a new feature representation of the input X.

Autoencoders: Applications

- Feature Learning and Dimensionality reduction

- Pretraining of Deep Neural Networks

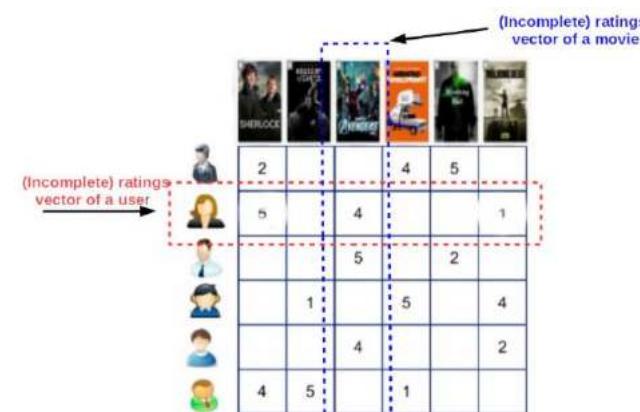
- Denoising and Inpainting



- Generate Images

- Anomaly Detection

- Recommender Systems (e.g. matrix completion)



Sedhain et al, AutoRec: Autoencoders Meet Collaborative Filtering, WWW 2015

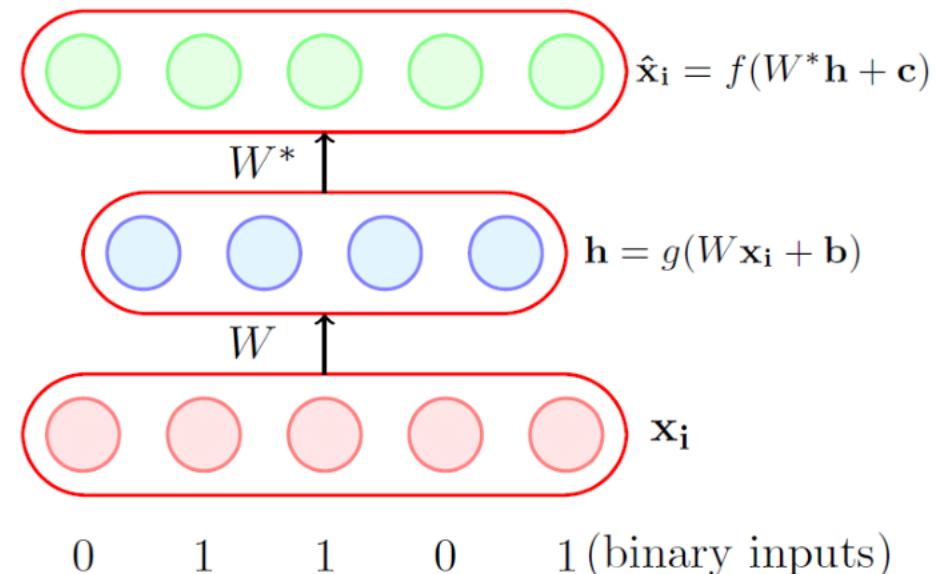
Choice of Loss and Activation functions

- For **binary inputs** (each $X_{ij} \in \{0,1\}$):

- The activation function “f” (o/p layer) is chosen as **Sigmoid**.
- The loss function is $L([W, W^*, c, b])$ is (**Cross Entropy**):

$$\min\left\{-\sum_{i=1}^n (x_{ij} \log \hat{x}_{ij} + (1 - x_{ij}) \log(1 - \hat{x}_{ij}))\right\}$$

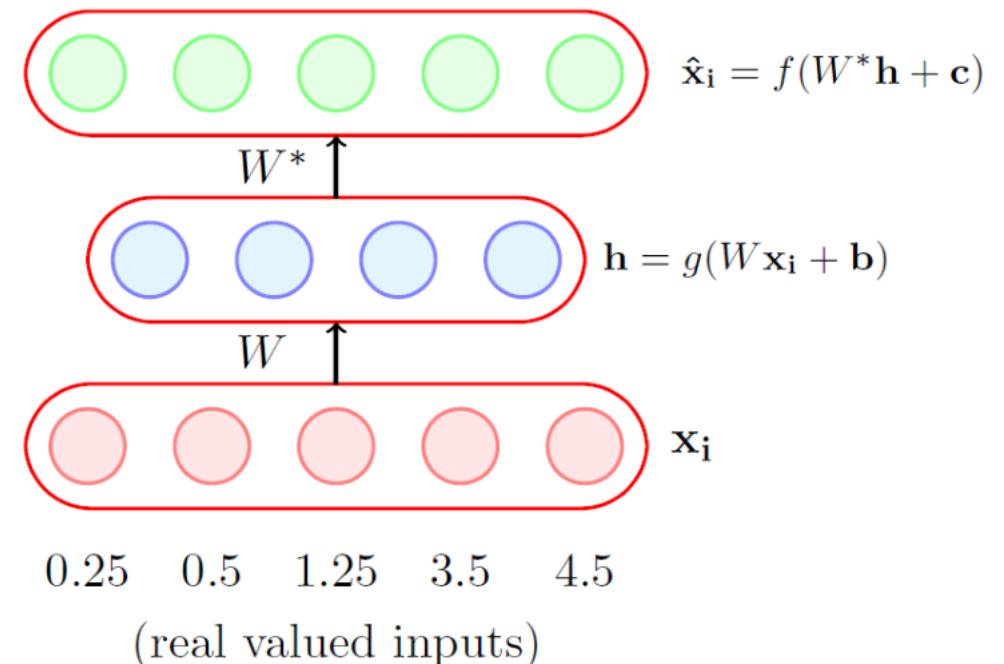
- The activation function “g” is typically chosen as Sigmoid.



Choice of Loss and Activation functions

- For **real valued inputs** (each $X_{ij} \in R$):
 - The activation function “f” is chosen as **Linear**.
 - The loss function is $L([W, W^*, c, b])$ is **(SSD)**:

$$\min_{W, W^*, c, b} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$



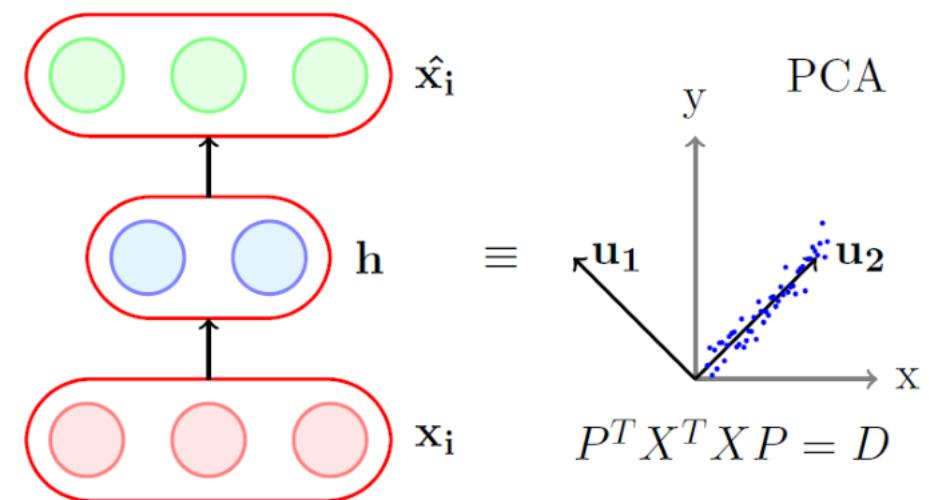
- The activation function “g” is typically chosen as Sigmoid.

Autoencoders and PCA

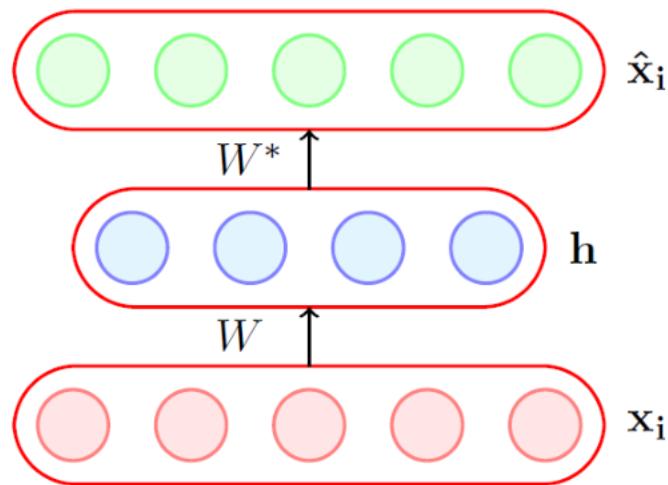
- An autoencoder's **encoder part** is **equivalent to PCA** if we:

- Use a Linear Encoder.
- Use a Linear Decoder.
- Use a Squared Error Loss function.
- Normalize the inputs to:

$$\hat{x}_{ij} = \frac{1}{\sqrt{m}} \left(x_{ij} - \frac{1}{m} \sum_{k=1}^m x_{kj} \right)$$

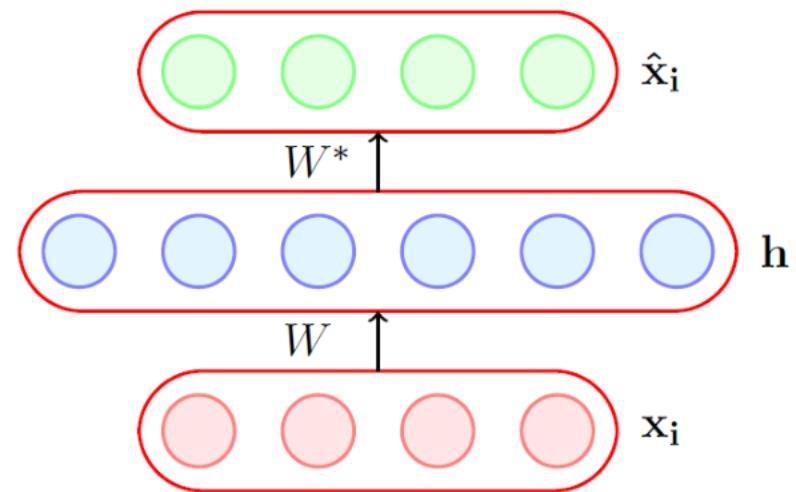


Undercomplete and Overcomplete Autoencoders



Undercomplete AE

- Dimension of h is less than dimension x

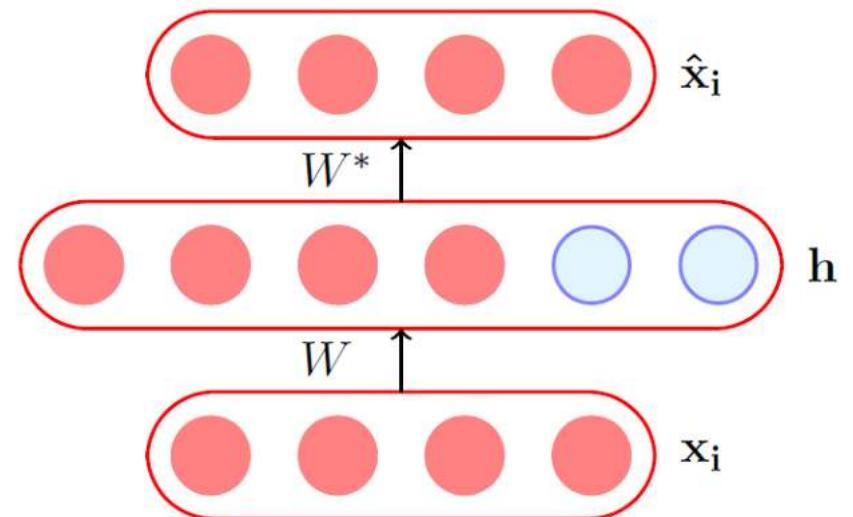


Overcomplete AE

- Dimension of h is greater than dimension x

Regularization in Autoencoders

- An overcomplete autoencoder would learn to copy the inputs x_i to hidden representation h_i and then copy h_i to \hat{x}_i (learn an identity function).
- Such an identity encoding is useless in practice as it does not really tell us anything about the important characteristics of the data. This is a case of overfitting!
- Overfitting (Poor generalization) could happen even in undercomplete autoencoders.
- To avoid poor generalization, we need to introduce regularization.



Regularization in Autoencoders

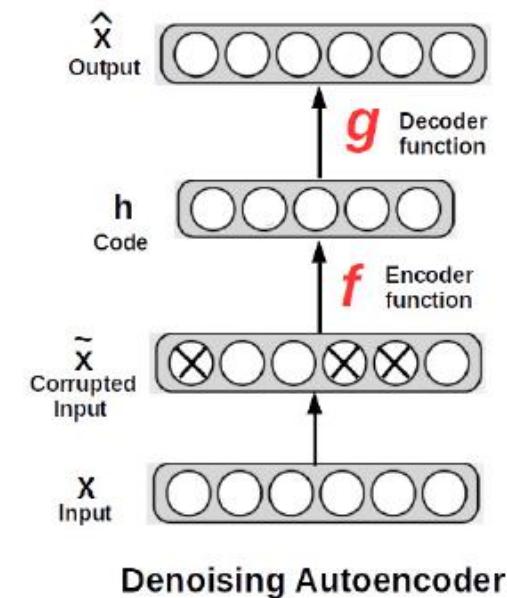
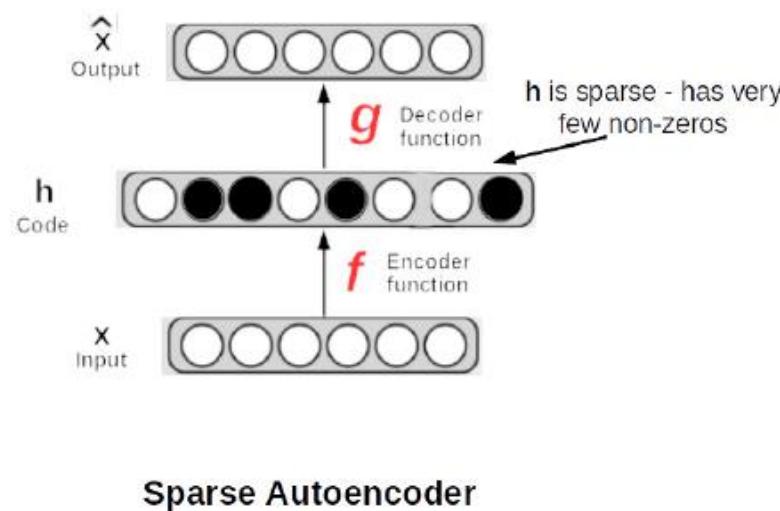
- The simplest solution is to add an L_2 regularization term to the objective function:

$$\min_{\theta, w, w^*, \mathbf{b}, \mathbf{c}} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2 + \lambda \|\theta\|^2$$

- Another trick is to tie the weights of encoder and decoder ($W^* = W^T$). This effectively reduces the number of parameters and acts as a regularizer.

Regularized Autoencoders

- Several ways to regularize the model, e.g.
 - Make the learned code sparse ([Sparse Autoencoders](#))
 - Make the model robust against noisy/incomplete inputs ([Denoising Autoencoders](#))



- Make the model robust against small changes in the input ([Contractive Autoencoders](#))

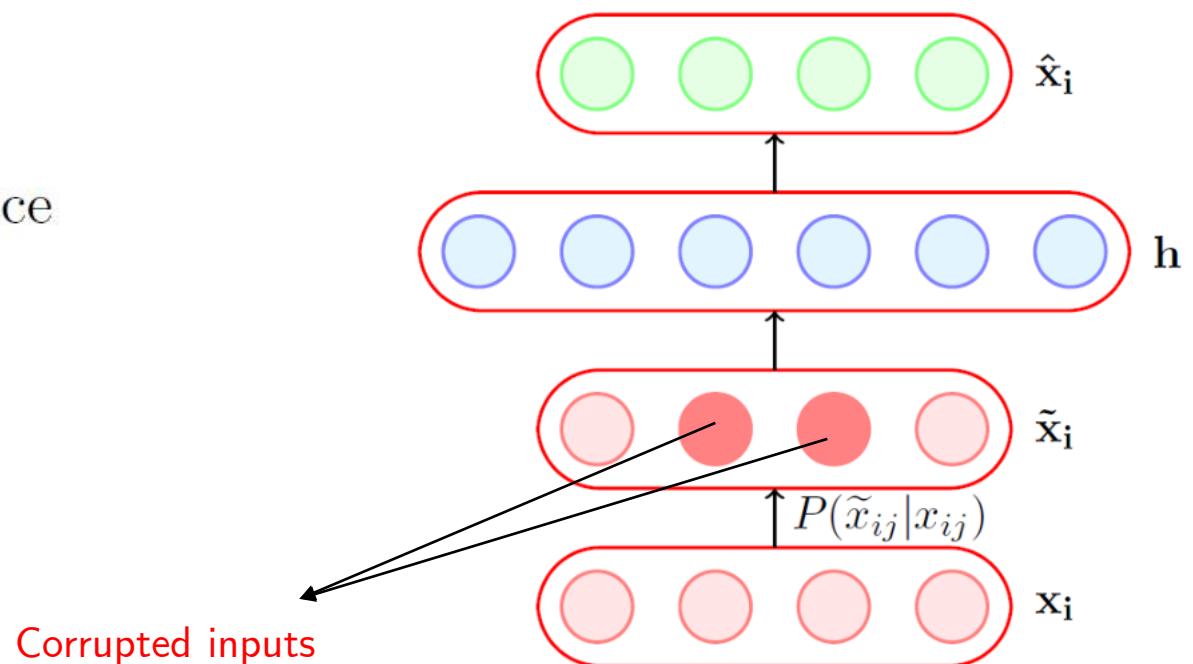
Denoising Autoencoders

- A denoising encoder simply corrupts the input data using a probabilistic process ($P(\tilde{x}_{ij}|x_{ij})$) before feeding it to the network.

A simple $P(\tilde{x}_{ij}|x_{ij})$ used in practice is the following

$$P(\tilde{x}_{ij} = 0|x_{ij}) = q$$

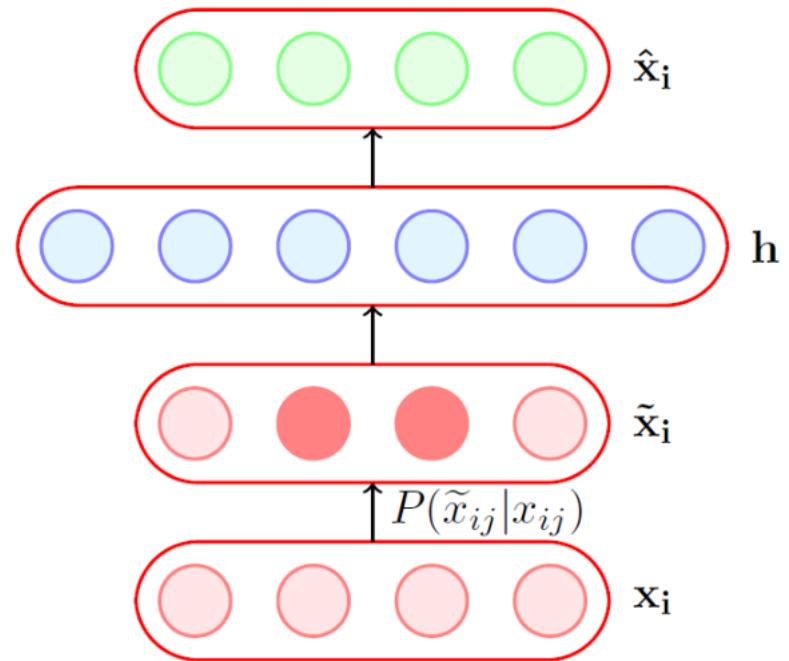
$$P(\tilde{x}_{ij} = x_{ij}|x_{ij}) = 1 - q$$



- In other words, with probability q the input is flipped to 0 and with probability $(1 - q)$ it is retained as it is.

Denoising Autoencoders

- How does this help?
 - This helps because the objective is still to reconstruct the original (un-corrupted) x_i :
$$\arg \min_{\theta} \frac{1}{m} \sum_{i=1}^m \sum_{j=1}^n (\hat{x}_{ij} - x_{ij})^2$$
 - It no longer makes sense for the model learn to copy corrupted input to h (the objective function will not be minimized by doing so)
 - Instead, the model will now have to capture the characteristics of the data correctly.

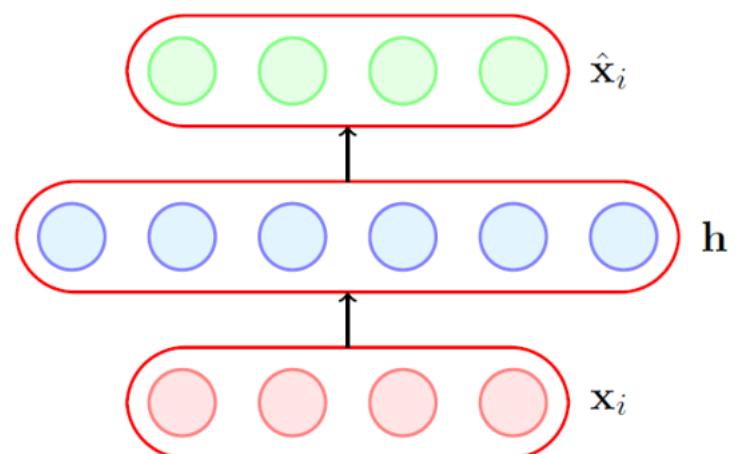


Sparse Autoencoders

- A hidden neuron with sigmoid activation will have values between 0 and 1.
- We say that the neuron is activated when its output is close to 1 and not activated when its output is close to 0.
- A sparse autoencoder tries to ensure the neuron is inactive most of the times.

The average value of the activation of a neuron l is given by

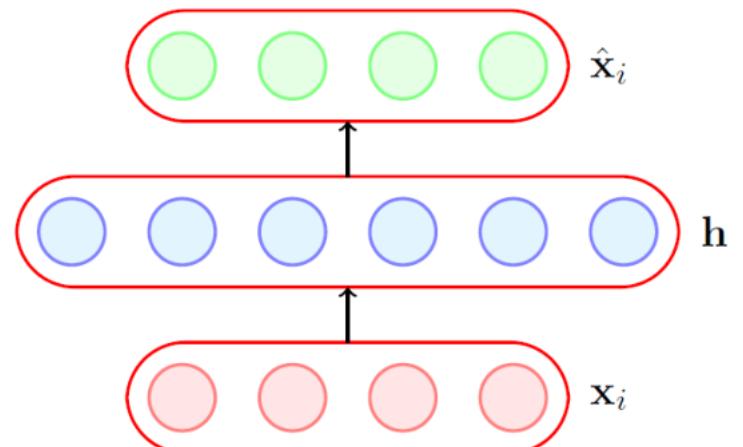
$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^m h(\mathbf{x}_i)_l$$



Sparse Autoencoders

- If the neuron l is sparse (i.e. mostly inactive) then $\hat{\rho}_l \rightarrow 0$
- A sparse autoencoder uses a sparsity parameter ρ (typically very close to 0, say, 0.005) and tries to enforce the constraint $\hat{\rho}_l = \rho$
- One way of ensuring this is to add the following term to the objective function

$$\Omega(\theta) = \sum_{l=1}^k \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l}$$



The average value of the activation of a neuron l is given by

$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^m h(\mathbf{x}_i)_l$$

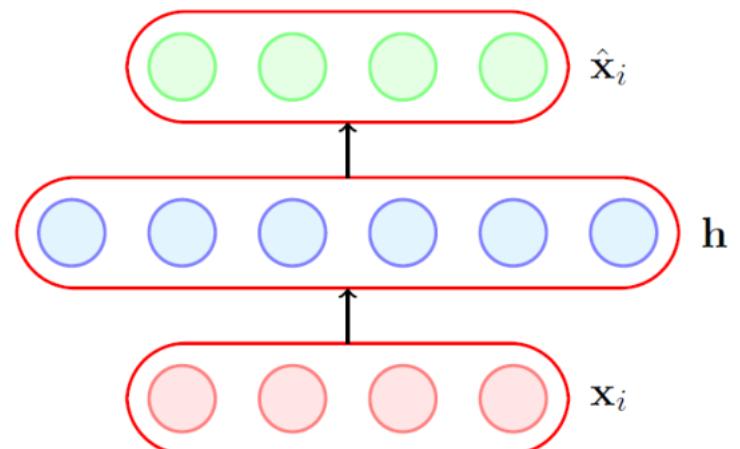
Sparse Autoencoders

- Now, the equation for the loss function will look like:

$$L'(\theta) = L(\theta) + \Omega(\theta)$$

$$\Omega(\theta) = \sum_{l=1}^k \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l}$$

- When will this term ($\Omega(\theta)$) reach its minimum value and what is the minimum value?



The average value of the activation of a neuron l is given by

$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^m h(\mathbf{x}_i)_l$$

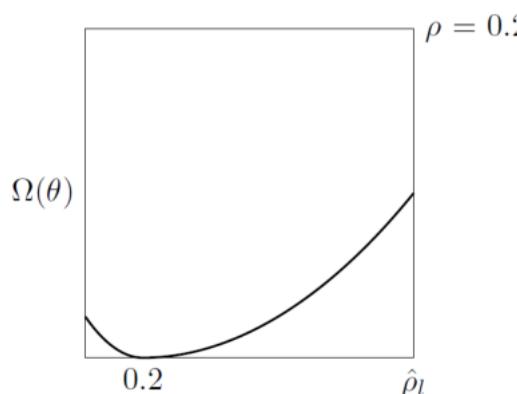
Sparse Autoencoders

- Now, the equation for the loss function will look like:

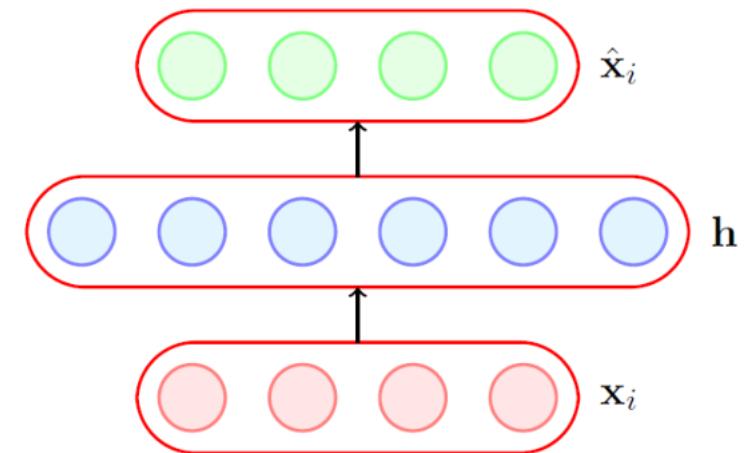
$$L'(\theta) = L(\theta) + \Omega(\theta) \rightarrow \text{Sparsity constraint}$$

$$\Omega(\theta) = \sum_{l=1}^k \rho \log \frac{\rho}{\hat{\rho}_l} + (1 - \rho) \log \frac{1 - \rho}{1 - \hat{\rho}_l}$$

- When will this term ($\Omega(\theta)$) reach its minimum value and what is the minimum value?



$\Omega(\theta)$ will reach min. value
when $\hat{\rho}_l = \rho$



The average value of the activation of a neuron l is given by

$$\hat{\rho}_l = \frac{1}{m} \sum_{i=1}^m h(\mathbf{x}_i)_l$$

Contractive Autoencoders

- A contractive autoencoder also tries to prevent an overcomplete autoencoder from learning the identity function.
- It does so by adding the following regularization term to the loss function:

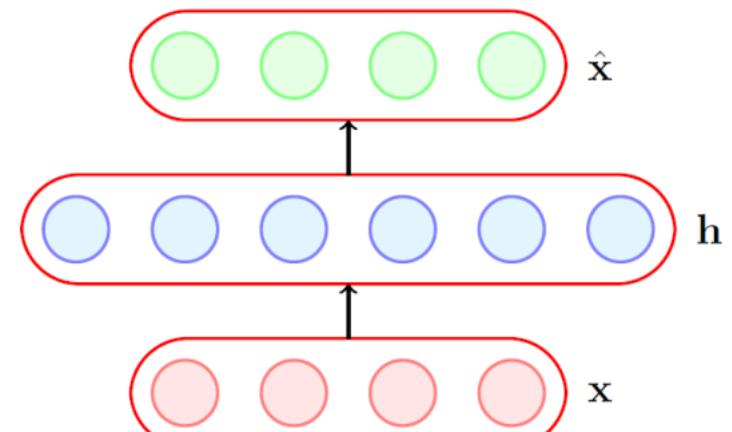
$$\Omega(\theta) = \|J_{\mathbf{x}}(\mathbf{h})\|_F^2$$

↓ ↓
Jacobian of the encoder Frobenius Norm

Variation in output of
2nd neuron in the
hidden layer with a
small variation in the
1st input.

$$J_{\mathbf{x}}(\mathbf{h}) = \begin{bmatrix} \frac{\partial h_1}{\partial x_1} & \dots & \dots & \dots & \frac{\partial h_1}{\partial x_n} \\ \frac{\partial h_2}{\partial x_1} & \dots & \dots & \dots & \frac{\partial h_2}{\partial x_n} \\ \vdots & & \ddots & & \vdots \\ \frac{\partial h_k}{\partial x_1} & \dots & \dots & \dots & \frac{\partial h_k}{\partial x_n} \end{bmatrix}$$

$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^n \sum_{l=1}^k \left(\frac{\partial h_l}{\partial x_j} \right)^2$$



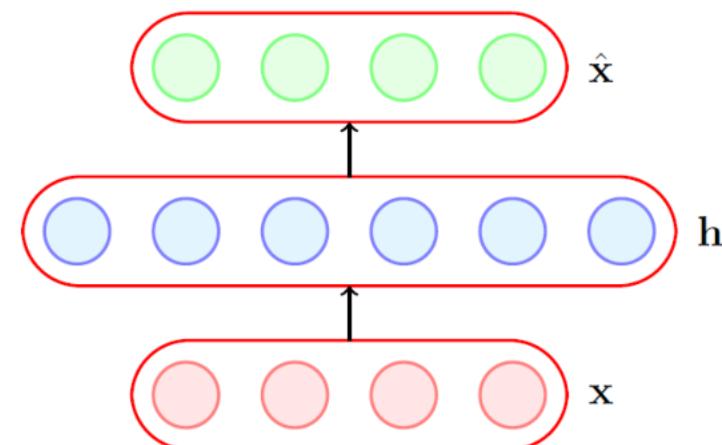
Contractive Autoencoders

- What is the intuition behind this ?
- Consider $\frac{\partial h_1}{\partial x_1}$, what does it mean if $\frac{\partial h_1}{\partial x_1} = 0$
- It means that this neuron is not very sensitive to variations in the input x_1 .

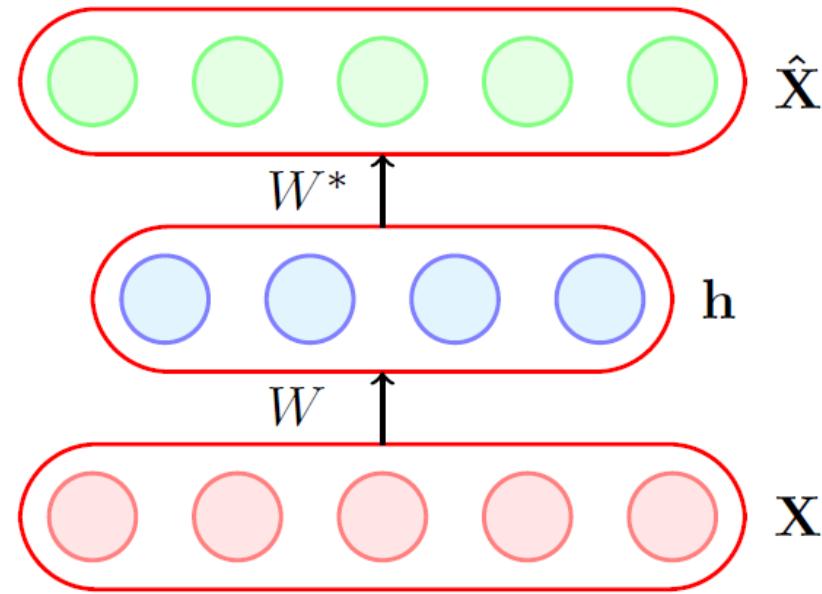
$$\hat{\mathcal{L}}(\theta) = \mathcal{L}(\theta) + \Omega(\theta)$$

- $\mathcal{L}(\theta)$ - capture important variations in data
- $\Omega(\theta)$ - do not capture variations in data
- Tradeoff - capture only very important variations in the data

$$\|J_{\mathbf{x}}(\mathbf{h})\|_F^2 = \sum_{j=1}^n \sum_{l=1}^k \left(\frac{\partial h_l}{\partial x_j} \right)^2$$



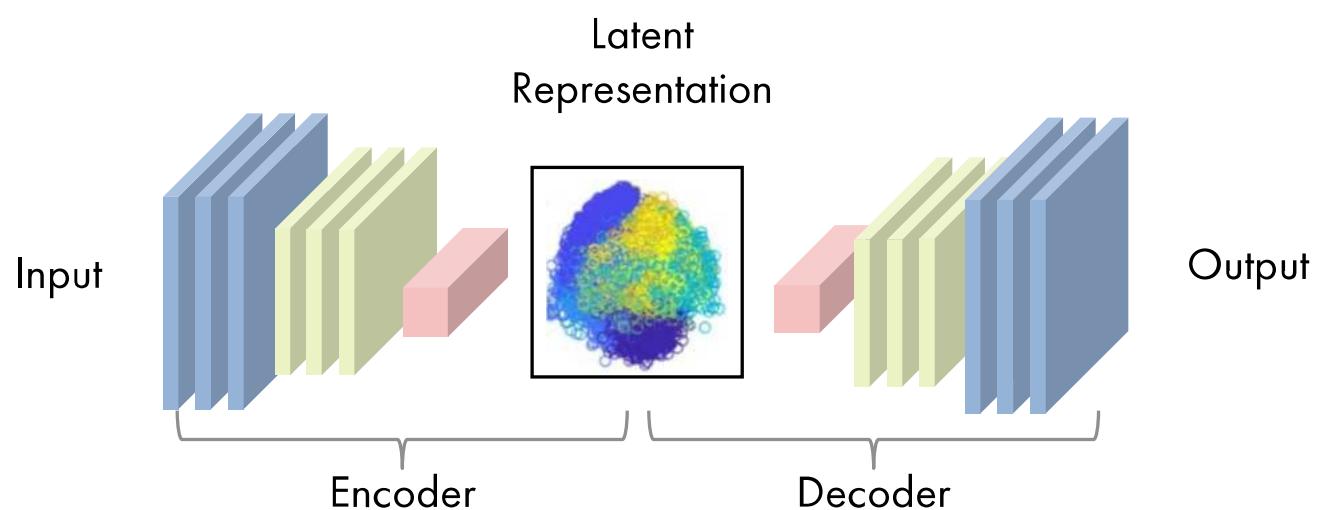
Generative Models: Introduction



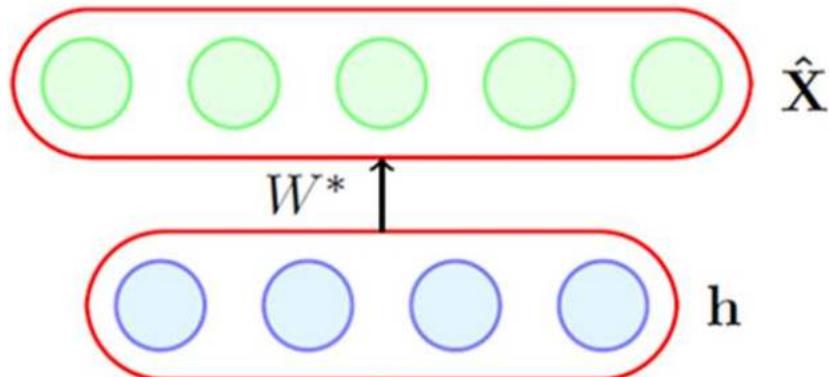
$$h = g(WX + b)$$

$$\hat{X} = f(W^*h + c)$$

- An autoencoder contains an encoder which takes the input x and maps it to a hidden representation.
- The decoder then takes this hidden representation and tries to reconstruct the input from it as \hat{x} .
- The hidden layer produces a good abstraction of the input, in the form of h .

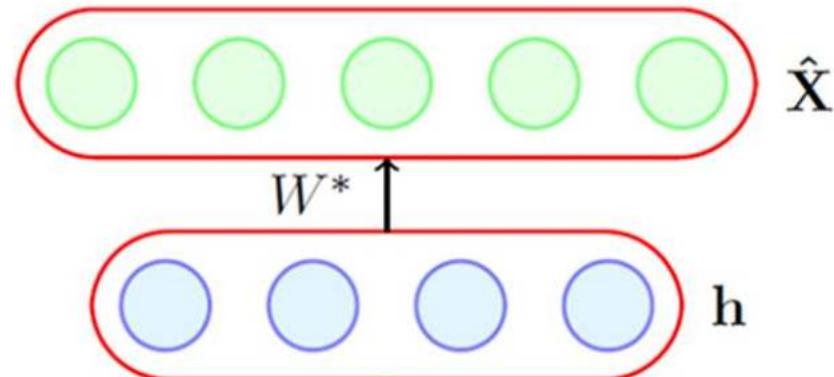


Generative Models: Introduction



- An autoencoder contains an encoder which takes the input x and maps it to a hidden representation.
- The decoder then takes this hidden representation and tries to reconstruct the input from it as \hat{x} .
- The hidden layer produces a good abstraction of the input, in the form of h .
- Now, once the autoencoder is trained can we remove the encoder, feed a hidden representation h to the decoder and decode a \hat{x} from it ?
- In other words, can we do generation with autoencoders?

Generative Models: Introduction



- Remember classic autoencoders are deterministic i.e., for an X , the encoder will give the same hidden representation every time.

- In principle, yes! But in practice there is a problem with this approach.
- h is a very high dimensional vector and only a few vectors in this space would actually correspond to meaningful latent representations of our input.
- So, of all the possible values of h which values should I feed to the decoder?
- Ideally, we should feed only those values of h which are highly *likely*.
- In other words, we are interested in sampling from a distribution over h 's ($P(h|X)$) so that we pick only those h 's which have high probability.
- Let's understand this concept clearly with an example.

Generative Models: Introduction

Example: Movie Review

M1: An unexpected and necessary masterpiece

M2: Delightfully merged information and comedy

M3: Director's first true masterpiece

M4: Sci- perfection, truly mesmerizing film

M5: Waste of time and money

M6: Best Lame Historical Movie Ever

- Consider a movie critic who writes reviews for movies
- For simplicity let us assume that he always writes reviews containing a **maximum of 5 words**
- Further, let us assume that there are a total of **50 words** in his **vocabulary**
- Each of the 5 words in his review can be treated as a **random variable** which takes one of the 50 values

Generative Models: Introduction

Example: Movie Review

M1: An unexpected and necessary masterpiece

M2: Delightfully merged information and comedy

M3: Director's first true masterpiece

M4: Sci- perfection, truly mesmerizing film

M5: Waste of time and money

M6: Best Lame Historical Movie Ever

- Given many such reviews written by the reviewer we could learn the joint probability distribution

$$P(X_1, X_2, \dots, X_5)$$

- Consider a movie critic who writes reviews for movies
- For simplicity let us assume that he always writes reviews containing a **maximum of 5 words**
- Further, let us assume that there are a total of **50 words** in his **vocabulary**
- Each of the 5 words in his review can be treated as a **random variable** which takes one of the 50 values

Generative Models: Introduction

Example: Movie Review

M1: An unexpected and necessary masterpiece

M2: Delightfully merged information and comedy

M3: Director's first true masterpiece

M4: Sci- perfection, truly mesmerizing film

M5: Waste of time and money

M6: Best Lame Historical Movie Ever

- We can think of a very simple **factorization** of this model by assuming that the i^{th} word only **depends on the previous 2 words** and not anything before that:

$$P(X_1, X_2, \dots, X_5) = P(X_1) \prod_{i=2:5} P(X_i | X_{i-1}, X_{i-2})$$

- Let us consider one such factor:

$$P(X_i = \text{time} \mid X_{i-2} = \text{waste}, X_{i-1} = \text{of})$$

- This could be estimated as:

$$\frac{\text{count (waste of time)}}{\text{count (waste of)}}$$

- And the two counts mentioned above can be computed by going over all the reviews

Generative Models: Introduction

- What can we do with this joint distribution?

Joint distribution

w	$P(X_i = w X_{i-2} = \text{more}, X_{i-1} = \text{realistic})$	$P(X_i = w X_{i-2} = \text{realistic}, X_{i-1} = \text{than})$	$P(X_i = w X_{i-2} = \text{than}, X_{i-1} = \text{real})$...
than	0.61	0.01	0.20	...
as	0.12	0.10	0.16	...
for	0.14	0.09	0.05	...
real	0.01	0.50	0.01	...
the	0.02	0.12	0.12	...
life	0.05	0.11	0.33	...

Generative Models: Introduction

M7: More realistic than real life

w	$P(X_i = w X_{i-2} = \text{more}, X_{i-1} = \text{realistic})$	$P(X_i = w X_{i-2} = \text{realistic}, X_{i-1} = \text{than})$	$P(X_i = w X_{i-2} = \text{than}, X_{i-1} = \text{real})$...
than	0.61	0.01	0.20	...
as	0.12	0.10	0.16	...
for	0.14	0.09	0.05	...
real	0.01	0.50	0.01	...
the	0.02	0.12	0.12	...
life	0.05	0.11	0.33	...

- What can we do with this joint distribution?
- Given a **review**, **classify** if this was written by the reviewer

$$\begin{aligned} P(M7) &= P(X_1 = \text{more}).P(X_2 = \text{realistic}|X_1 = \text{more}). \\ &\quad P(X_3 = \text{than}|X_1 = \text{more}, X_2 = \text{realistic}). \\ &\quad P(X_4 = \text{real}|X_2 = \text{realistic}, X_3 = \text{than}). \\ &\quad P(X_5 = \text{life}|X_3 = \text{than}, X_4 = \text{real}) \\ &= 0.2 \times 0.25 \times 0.61 \times 0.50 \times 0.33 = 0.005 \end{aligned}$$

Generative Models: Introduction

M7: More realistic than real life

w	$P(X_i = w X_{i-2} = \text{more}, X_{i-1} = \text{realistic})$	$P(X_i = w X_{i-2} = \text{realistic}, X_{i-1} = \text{than})$	$P(X_i = w X_{i-2} = \text{than}, X_{i-1} = \text{real})$...
than	0.61	0.01	0.20	...
as	0.12	0.10	0.16	...
for	0.14	0.09	0.05	...
real	0.01	0.50	0.01	...
the	0.02	0.12	0.12	...
life	0.05	0.11	0.33	...

- What can we do with this joint distribution?
- Given a review, classify if this was written by the reviewer
- **What else can we do?**

Generative Models: Introduction

M7: More realistic than real life

w	$P(X_i = w X_{i-2} = \text{more}, X_{i-1} = \text{realistic})$	$P(X_i = w X_{i-2} = \text{realistic}, X_{i-1} = \text{than})$	$P(X_i = w X_{i-2} = \text{than}, X_{i-1} = \text{real})$...
than	0.61	0.01	0.20	...
as	0.12	0.10	0.16	...
for	0.14	0.09	0.05	...
real	0.01	0.50	0.01	...
the	0.02	0.12	0.12	...
life	0.05	0.11	0.33	...

- What can we do with this joint distribution?
- Given a review, classify if this was written by the reviewer
- What else can we do?
- **Generate** new reviews which would look like reviews written by this reviewer

Generative Models: Introduction

M7: More realistic than real life

w	$P(X_i = w X_{i-2} = \text{more}, X_{i-1} = \text{realistic})$	$P(X_i = w X_{i-2} = \text{realistic}, X_{i-1} = \text{than})$	$P(X_i = w X_{i-2} = \text{than}, X_{i-1} = \text{real})$...
than	0.61	0.01	0.20	...
as	0.12	0.10	0.16	...
for	0.14	0.09	0.05	...
real	0.01	0.50	0.01	...
the	0.02	0.12	0.12	...
life	0.05	0.11	0.33	...

- What can we do with this joint distribution?
- Given a review, classify if this was written by the reviewer
- What else can we do?
- **Generate** new reviews which would look like reviews written by this reviewer
- **How to do this?**

Generative Models: Introduction

w	$P(X_1 = w)$			
the	0.62			
movie	0.10			
amazing	0.01			
useless	0.01			
was	0.01			
:	:			

- How does the reviewer start his reviews (what is the first word that he chooses)?

Generative Models: Introduction

w	$P(X_1 = w)$			
the	0.62			
movie	0.10			
amazing	0.01			
useless	0.01			
was	0.01			
:	:			

- How does the reviewer start his reviews (what is the first word that he chooses)?
- We could take the word which has the highest probability and put it as the first word in our review

The

Generative Models: Introduction

w	$P(X_1 = w)$	$P(X_2 = w X_1 = \text{the})$		
the	0.62	0.01		
movie	0.10	0.40		
amazing	0.01	0.22		
useless	0.01	0.20		
was	0.01	0.00		
:	:	:		

- How does the reviewer start his reviews (what is the first word that he chooses)?
- We could take the word which has the highest probability and put it as the first word in our review
- Having selected this what is the most likely second word that the reviewer uses?

The movie

Generative Models: Introduction

w	$P(X_1 = w)$	$P(X_2 = w X_1 = \text{the})$	$P(X_i = w X_{i-2} = \text{the}, X_{i-1} = \text{movie})$	
the	0.62	0.01	0.01	
movie	0.10	0.40	0.01	
amazing	0.01	0.22	0.01	
useless	0.01	0.20	0.03	
was	0.01	0.00	0.60	
:	:	:	:	

- How does the reviewer start his reviews (what is the first word that he chooses)?
- We could take the word which has the highest probability and put it as the first word in our review
- Having selected this what is the most likely second word that the reviewer uses?
- Having selected first two words what is the most likely third word that the reviewer uses?

The movie was

Generative Models: Introduction

w	$P(X_1 = w)$	$P(X_2 = w X_1 = \text{the})$	$P(X_i = w X_{i-2} = \text{the}, X_{i-1} = \text{movie})$...
the	0.62	0.01	0.01	...
movie	0.10	0.40	0.01	...
amazing	0.01	0.22	0.01	...
useless	0.01	0.20	0.03	...
was	0.01	0.00	0.60	...
:	:	:	:	...

The movie was really amazing

- How does the reviewer start his reviews (what is the first word that he chooses)?
- We could take the word which has the highest probability and put it as the first word in our review
- Having selected this what is the most likely second word that the reviewer uses?
- Having selected first two words what is the most likely third word that the reviewer uses?
- And so on ...

Generative Models: Introduction

w	$P(X_1 = w)$	$P(X_2 = w X_1 = \text{the})$	$P(X_i = w X_{i-2} = \text{the}, X_{i-1} = \text{movie})$...
the	0.62	0.01	0.01	...
movie	0.10	0.40	0.01	...
amazing	0.01	0.22	0.01	...
useless	0.01	0.20	0.03	...
was	0.01	0.00	0.60	...
:	:	:	:	...

The movie was really amazing

- How does the reviewer start his reviews (what is the first word that he chooses)?
- We could take the word which has the highest probability and put it as the first word in our review
- Having selected this what is the most likely second word that the reviewer uses?
- Having selected first two words what is the most likely third word that the reviewer uses?
- And so on ...
- But, if we select the most likely word at each time step, then it will give us the same review again and again

Generative Models: Introduction

w	$P(X_1 = w)$	$P(X_2 = w X_1 = \text{the})$	$P(X_i = w X_{i-2} = \text{the}, X_{i-1} = \text{movie})$...
the	0.62	0.01	0.01	...
movie	0.10	0.40	0.01	...
amazing	0.01	0.22	0.01	...
useless	0.01	0.20	0.03	...
was	0.01	0.00	0.60	...
:	:	:	:	...

The movie was really amazing

We should instead **sample** from this distribution!

- How does the reviewer start his reviews (what is the first word that he chooses)?
- We could take the word which has the highest probability and put it as the first word in our review
- Having selected this what is the most likely second word that the reviewer uses?
- Having selected first two words what is the most likely third word that the reviewer uses?
- And so on ...
- But, if we select the most likely word at each time step, then it will give us the same review again and again

Generative Models: Introduction

w	$P(X_1 = w)$	$P(X_2 = w X_1 = \text{the})$	$P(X_i = w X_{i-2} = \text{the}, X_{i-1} = \text{movie})$...
the	0.62	0.01	0.01	...
movie	0.10	0.40	0.01	...
amazing	0.01	0.22	0.01	...
useless	0.01	0.20	0.03	...
was	0.01	0.00	0.60	...
is	0.01	0.00	0.30	...
masterpiece	0.01	0.11	0.01	...
I	0.21	0.00	0.01	...
liked	0.01	0.01	0.01	...
decent	0.01	0.02	0.01	...

- Suppose there are 10 words in our vocabulary, and we have computed the joint probability distribution over all the random variables.
- Now, consider that we need to generate the 3rd word in the review, given the first two words of the review.
- There are 10 possibilities :

Index	Word
0	the
1	movie
2	amazing
3	useless
4	was
5	is
6	masterpiece
7	I
8	liked
9	decent

Generative Models: Introduction

w	$P(X_1 = w)$	$P(X_2 = w X_1 = \text{the})$	$P(X_i = w X_{i-2} = \text{the}, X_{i-1} = \text{movie})$...
the	0.62	0.01	0.01	...
movie	0.10	0.40	0.01	...
amazing	0.01	0.22	0.01	...
useless	0.01	0.20	0.03	...
was	0.01	0.00	0.60	...
is	0.01	0.00	0.30	...
masterpiece	0.01	0.11	0.01	...
I	0.21	0.00	0.01	...
liked	0.01	0.01	0.01	...
decent	0.01	0.02	0.01	...

- Suppose there are 10 words in our vocabulary, and we have computed the joint probability distribution over all the random variables.
- Now, consider that we need to generate the 3rd word in the review, given the first two words of the review.
- There are 10 possibilities :
- We can think of this as a 10-sided dice where each side correspond to a word and has a certain probability to show up.

Index	Word	$P(X_i = w X_{i-2} = \text{the}, X_{i-1} = \text{movie})$.
0	the	0.01	.
1	movie	0.01	.
2	amazing	0.01	.
3	useless	0.03	.
4	was	0.60	.
5	is	0.30	.
6	masterpiece	0.01	.
7	I	0.01	.
8	liked	0.01	.
9	decent	0.01	.

Generative Models: Introduction

Generated Reviews

- the movie is liked decent
- I liked the amazing movie
- the movie is masterpiece

Such models which tries to estimate the probability $P(X)$ from a large number of samples are called as generative models

- Suppose there are 10 words in our vocabulary, and we have computed the joint probability distribution over all the random variables.

Now, consider that we need to generate the 3rd word in the review, given the first two words of the review.

There are 10 possibilities :

We can think of this as a 10-sided dice where each side correspond to a word and has a certain probability to show up.

- Roll this dice and pick the side (word) which comes up.
- Every run will now give a different review.

Index	Word	$P(X_i = w X_{i-2} = \text{the}, X_{i-1} = \text{movie})$.
0	the	0.01	.
1	movie	0.01	.
2	amazing	0.01	.
3	useless	0.03	.
4	was	0.60	.
5	is	0.30	.
6	masterpiece	0.01	.
7	I	0.01	.
8	liked	0.01	.
9	decent	0.01	.

Generative Models: Introduction



What about images?

- For 32x32 images we want to learn: $P(V_1, V_2, \dots, V_{1024})$ where V_i is a random variable corresponding to each pixel, which could possibly have values from 0-255.
- We could factorize this joint distribution by assuming that each pixel is dependent on its neighboring pixels.

Generative Models: Introduction



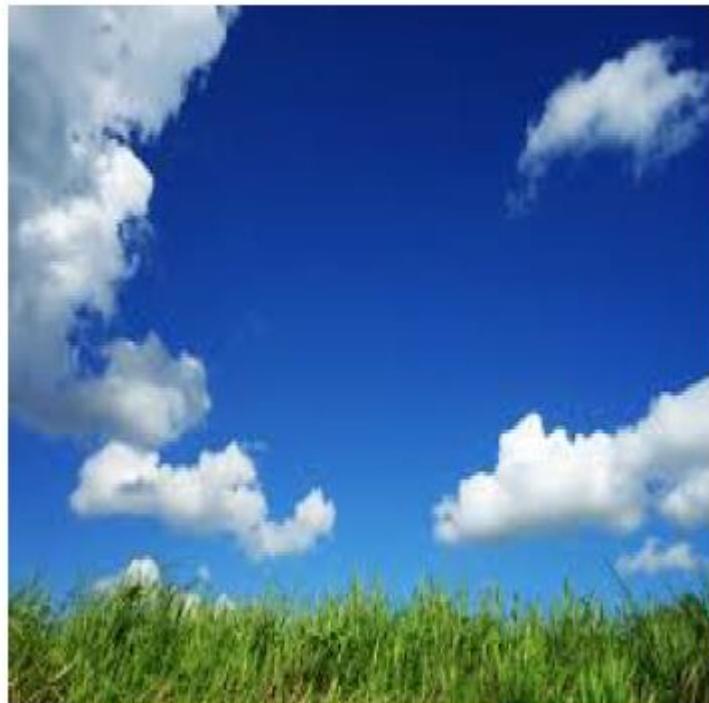
Input Image



Output Images

- Again, what can we do with joint distribution $P(V_1, V_2, \dots, V_{1024})$?
- Apart from **classifying** and **generating** (as discussed for previous language modelling in prev. slides), we can also **correct noisy inputs** (here, images) or help in **completing incomplete images**.

Generative Models: The concept of latent variables



- Latent variables : Blue sky, Green grass, White clouds
- There are certain underlying hidden (latent) characteristics which are determining the pixels and their interactions.
- We could think of these as additional (latent) random variables in our distribution
- And the interactions between the visible pixels are captured through the latent variables.

Generative Models: Introduction



Latent Variable = daytime



Latent Variable = night

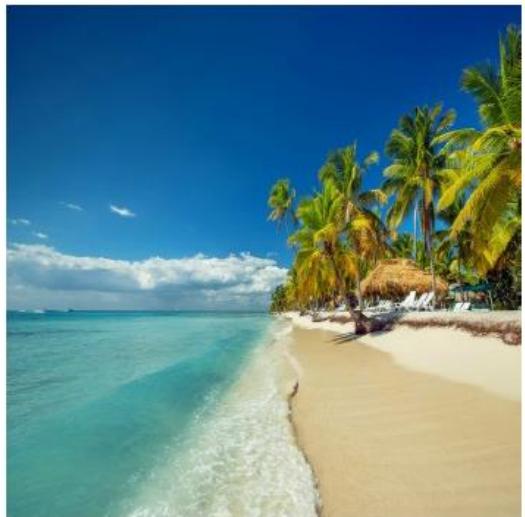


Latent Variable = cloudy

- Based on this notion, we would now like learn **the joint distribution $P(V, H)$** where, $V = \{V_1, V_2, \dots, V_{\#pixels}\}$ is observed variable and $H = \{H_1, H_2, \dots, H_{\#\text{latent features}}\}$ is hidden variables.

- Using this we can find:
$$P(H|V) = \frac{P(V, H)}{\sum_H P(V, H)}$$
- That is, given an image, we can **find the most likely latent configuration ($H = h$) that generated this image**, where h captures a latent (abstract) representation (imp. properties of the image)

Generative Models: Introduction



- Under this abstraction, all these images would look very similar (i.e., they would have very similar latent configurations h)
- Even though in the original feature space (pixels) there is a significant difference between these images, in the latent space they would be very close to each other
- Once again, assume that we are able to learn the joint distribution $P(V, H)$
- Using this distribution, we can find
$$P(V|H) = \frac{P(V, H)}{\sum_V P(V, H)}$$
- By using this, we can now say "**Create an image which is cloudy, has a beach and depicts daytime**"
- In other words, I can now generate images given certain latent variables

Variational Autoencoders (VAE)

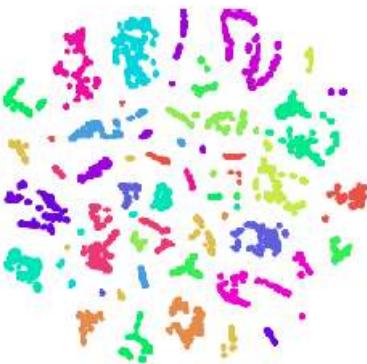


Figure: Abstraction

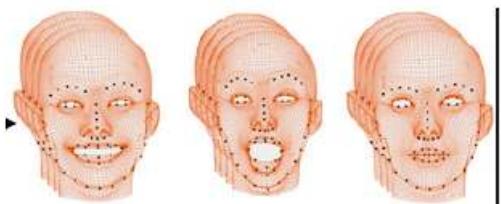
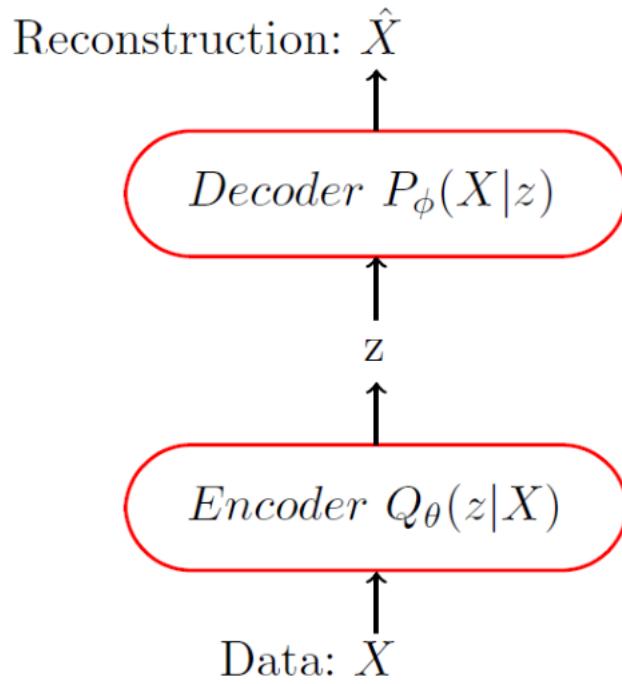


Figure: Generation

- Let $\{X = x_i\}_{i=1}^N$ be the training data
- We can think of X as a random variable in R^n
- For example, X could be an image and the dimensions of X correspond to pixels of the image
- We are interested in learning an abstraction (i.e., given an X find the hidden representation z)
- We are also interested in generation (i.e., given a hidden representation generate an X)
- In probabilistic terms we are interested in $P(z|X)$ and $P(X|z)$ (to be consistent with the iteration on VAEs we will use z instead of H and X instead of V)

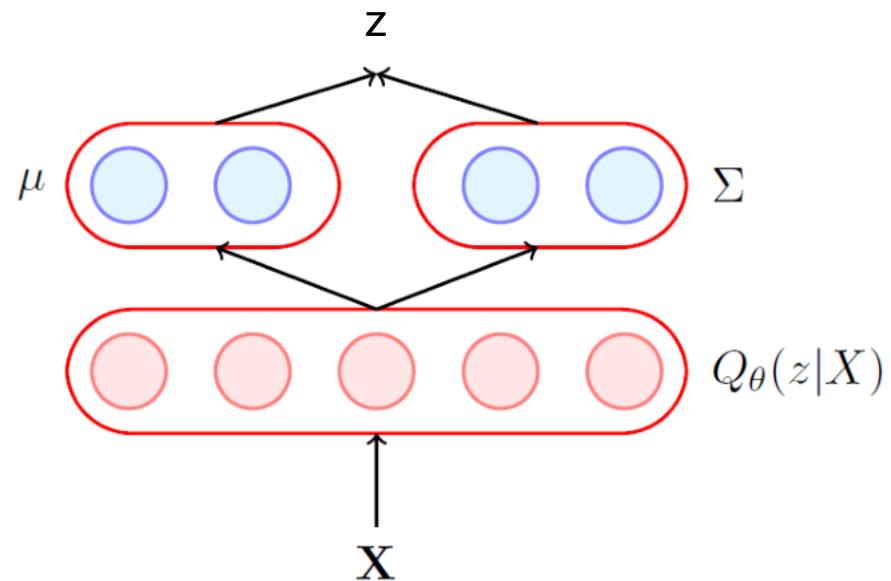
Variational Autoencoders (VAE)



θ : the parameters of the encoder neural network
 ϕ : the parameters of the decoder neural network

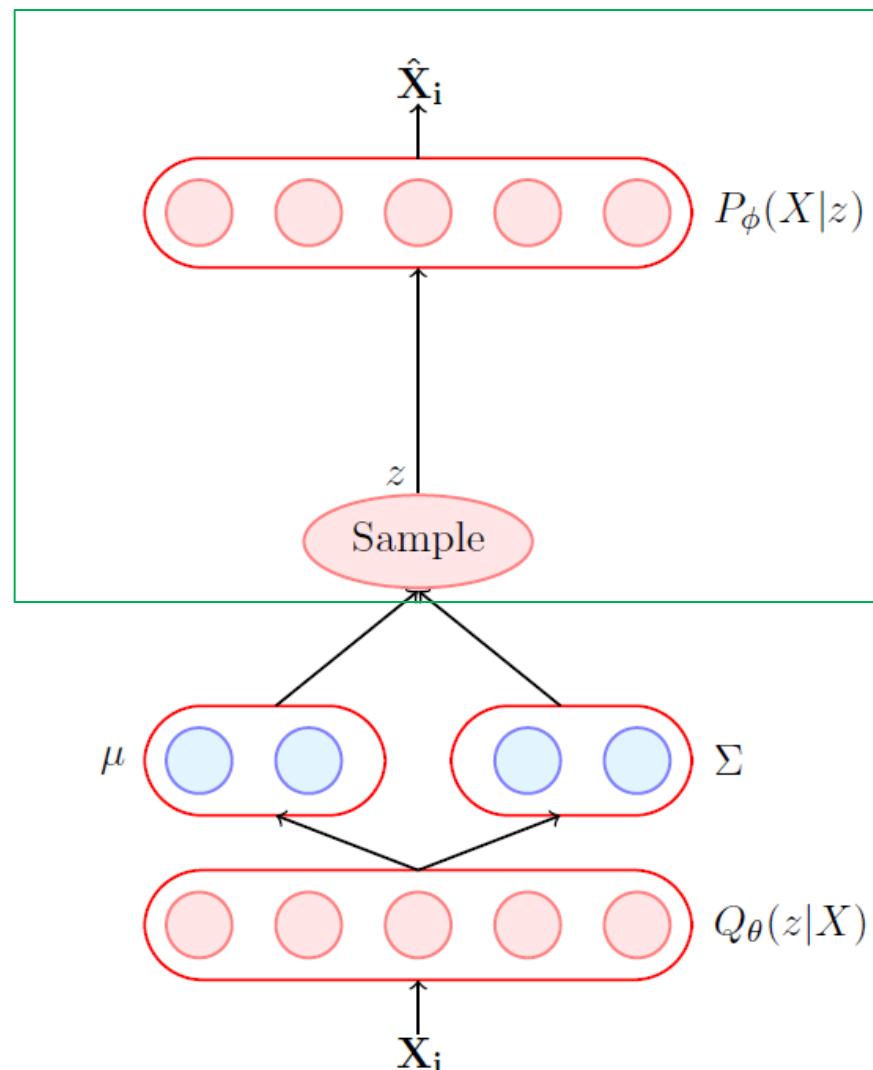
- **Encoder Goal:** Learn a distribution over the latent variables ($Q(z | X)$)
- **Decoder Goal:** Learn a distribution over the visible variables ($P(X | z)$)
- The training data is mapped to a latent space using a Neural network where the latent space **posterior distribution** ($Q(z|X)$) and **prior distribution** ($Q(z)$) are modelled as **Gaussian** and the **output of this network is two parameters – mean and covariance** (parameters of the posterior distribution)
- The latent space vector is mapped to input image using another neural network.

Variational Autoencoders: Encoder



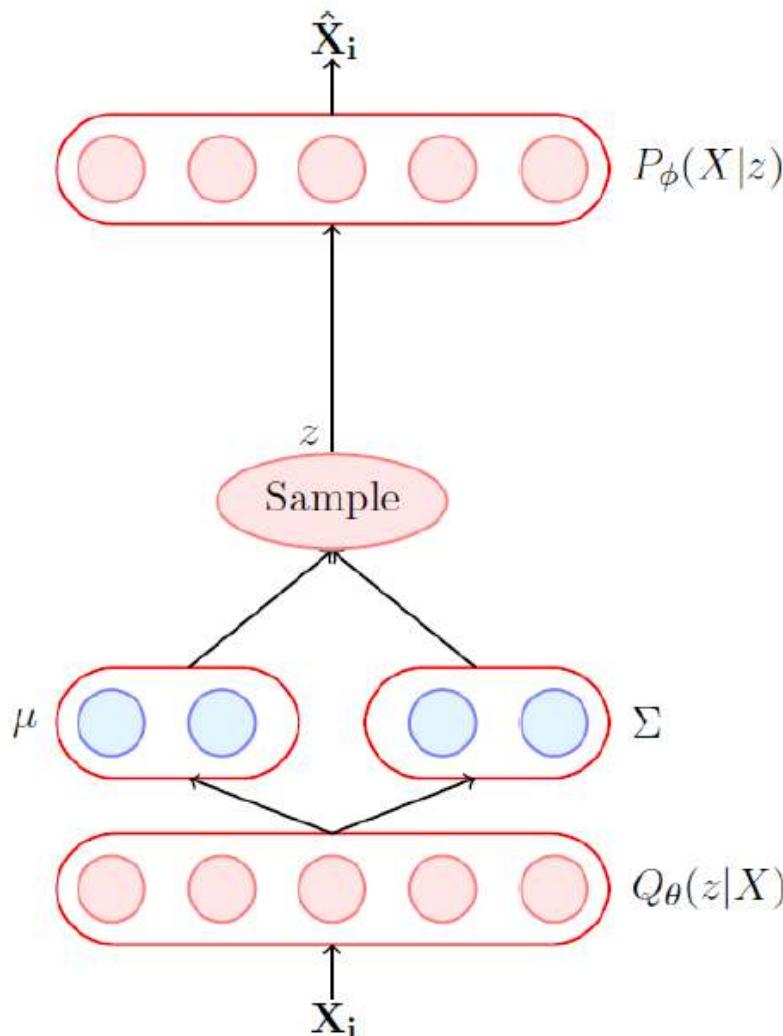
In VAEs we assume that the latent variables come from a standard normal distribution $\mathcal{N}(0, I)$ and the job of the encoder is to then predict the parameters of this distribution

Variational Autoencoders: Decoder



- The job of the decoder is to predict a probability distribution over $X : P(X|z)$
- Once again we will assume a certain form for this distribution
- For example, if we want to predict 28×28 pixels and each pixel belongs to \mathbb{R} (*i.e.*, $X \in \mathbb{R}^{784}$) then what would be a suitable family for $P(X|z)$?
- We could assume that $P(X|z)$ is a Gaussian distribution with unit variance
- The job of the decoder f would then be to predict the mean of this distribution as $f_{\phi}(z)$

Variational Autoencoders



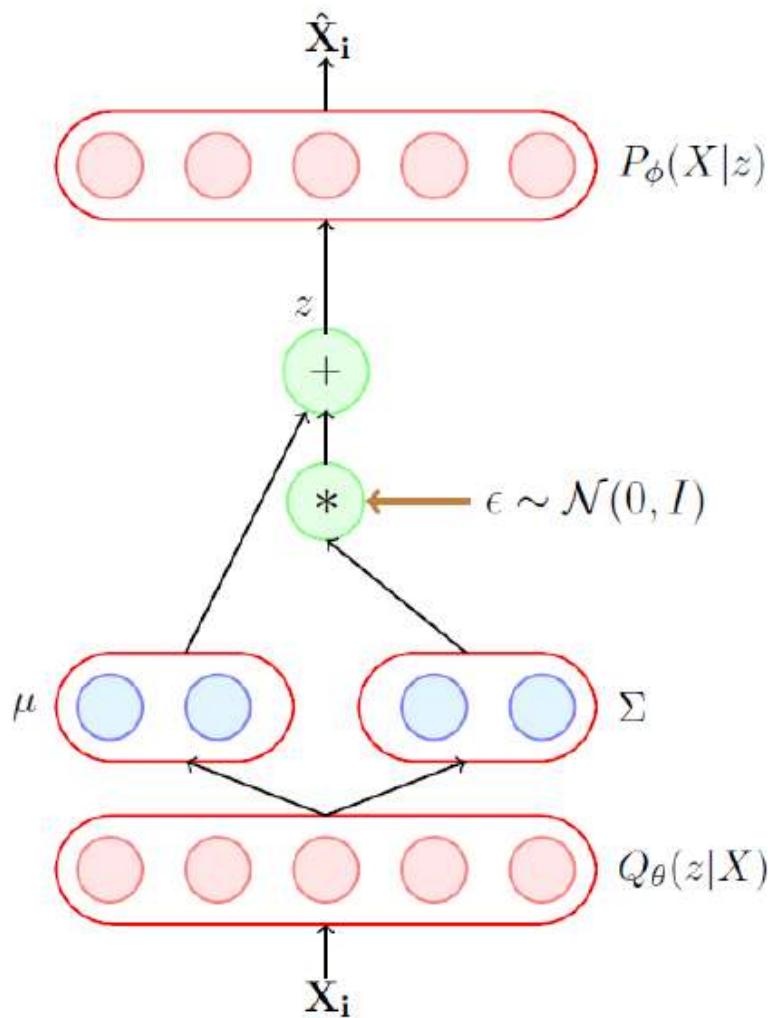
- The following is the total loss function (summed over all the data points):

$$\mathcal{L}(\theta) = \sum_{i=1}^m l_i(\theta, \phi) = -\mathbb{E}_{z \sim Q_{\theta}(z|x_i)} [\log P_{\phi}(x_i|z)] + KL(Q_{\theta}(z|x_i) || P(z))$$

KL-divergence → captures the difference between the two distributions

This term acts as a regularizer by forcing the encoder to produce latent representations that look like samples from a standard Gaussian distribution.

Variational Autoencoders



- The latent space in VAEs is typically modeled as a Gaussian distribution, and sampling from this distribution is a non-differentiable operation. This makes it difficult to compute gradients with respect to the model parameters.
- VAEs use a **reparameterization trick** to tackle the problem where we move the process of sampling to the input layer.

For 1 dimensional case, given μ and σ we can sample from $\mathcal{N}(\mu, \sigma)$ by first sampling $\epsilon \sim \mathcal{N}(0, 1)$, and then computing

$$z = \mu + \sigma * \epsilon$$

Markov Models

DSE 3151 DEEP LEARNING

B.Tech Data Science & Engineering

August 2023

[Rohini R Rao & Abhilash Pai](#)

Department of Data Science and Computer Applications

MIT Manipal

[Slide -4 of 5](#)

Markov models

- **Weak Law of Large Numbers**
 - When you collect independent samples, as the number of samples gets bigger, the mean of those samples converges to the true mean of the population
- **Markov**
 - independence was not a necessary condition for the mean to converge
 - average of the outcomes from a process involving *dependent* random variables could converge over time.
- **Markov Model:**
 - describe how random(stochastic) systems or processes evolve over time
 - The system is modeled as a sequence of states
 - where all states are observable
 - Model encodes dependencies and reach a steady-state over time
 - ie. it moves in between states with a specific probability
- **Claude Shannon used *Markov chains***
 - to model the English language as a sequence of letters that have a certain degree of randomness and dependencies between each other

<https://towardsdatascience.com/markov-models-and-markov-chains-explained-in-real-life-probabilistic-workout-routine-65e47b5c9a73>

Application of Markov Models

- **Parking lots** have a fixed number of spots available, but how many of these are available at any given point in time can be described as a combination of multiple factors or variables:
 - Day of the week,
 - Time of the day,
 - Parking fee,
 - Proximity to transit,
 - Proximity to businesses,
 - Number of free parking spots in the vicinity,
 - Number of available spots in the parking lot itself (a full-parking lot may deter some people to park there)
 - Some of these factors may be independent of each others are not
 - For instance, *Parking Fee* typically depends on *Time of day* and *Day of week*.

<https://towardsdatascience.com/markov-models-and-markov-chains-explained-in-real-life-probabilistic-workout-routine-65e47b5c9a73>

Applications of Markov Models

- Since Markov models describe the behavior over time, can be used to answer questions about the future state of the system:
- **How it evolves over time**
 - In what state is the system going to be in N time steps?
- **Tracing possible sequences in the process:**
 - When the system goes from *State A* to *State B* in N steps, what is the likelihood that it follows a specific path p ?
- **Parking Lot Example**
 - What is the occupancy rate of the parking lot 3 hours from now?
 - How likely is the parking lot to be at 50% capacity and then at 25% capacity in 5 hours?
- **Markov assumption**
 - It assumes the transition probability between each state only depends on the current state you are in
 - A Markov chain has short-term memory, it only remembers where you are now and where you want to go next.

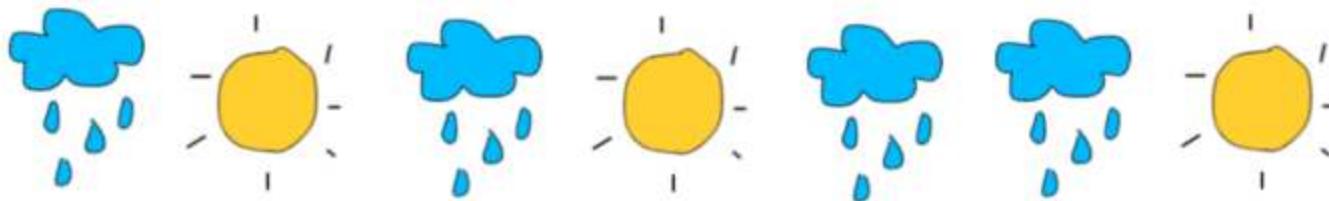
Markov Chains

- *A Markov chain is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.*



- **Ques 1:** calculate the percentage of instances its sunny on days directly following rainy days.
- **Ques 2:** calculate the percentage of instances its rainy on days directly following sunny days.

Markov Chains



- **Draw the Transition Matrix**
- **Draw the Markov Chain**
- The previous 3 days are [rainy, sunny, rainy]. What's the probability of rainy weather tomorrow?
- The previous 2 days are [rainy, rainy], What's the probability of rainy weather tomorrow?
- The previous 3 days are [sunny, rainy, sunny]. What's the probability of rainy weather tomorrow?
- Best starting point is a simple model which performs better than a random guess

Example : Modeling my Workout

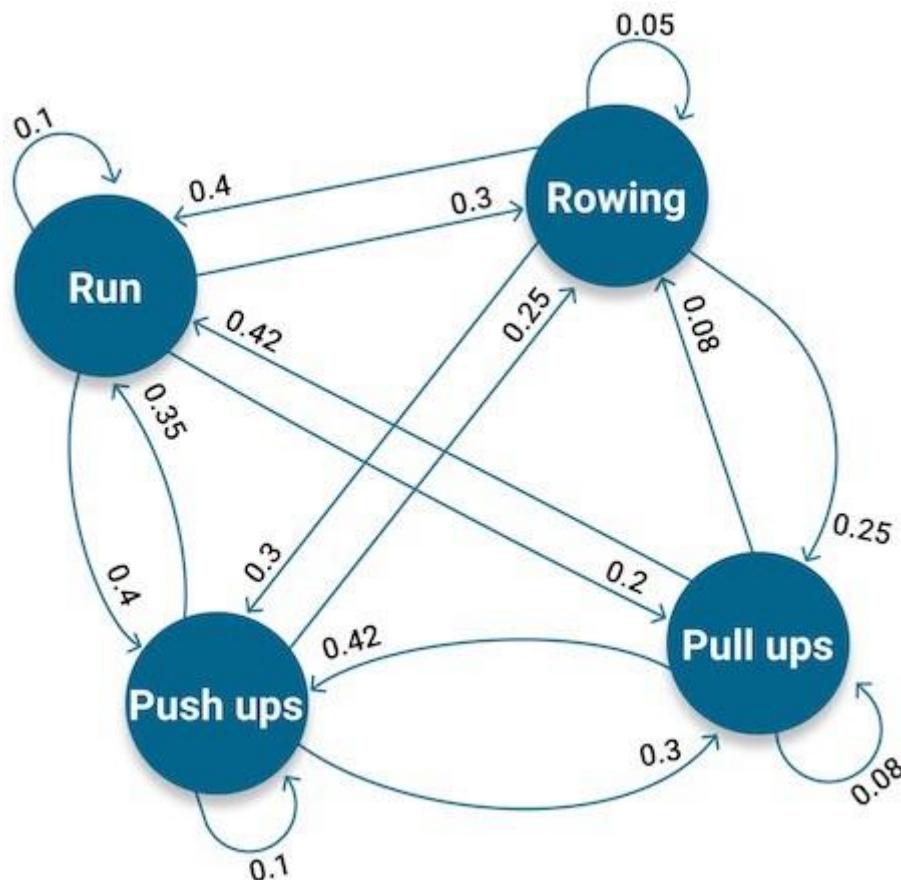
Set	Next set				Total
	Run (3 miles)	Push-ups (20)	Rowing (15 minutes)	Pull-ups (20)	
Run (3 miles)	1	4	3	2	10
Push-ups (20)	7	2	5	6	20
Rowing (15 minutes)	8	6	1	5	20
Pull-ups (20)	5	5	1	1	12
	21	17	10	14	62

Example: Modeling my workout – Transitional Matrix

Set	P(Next set Set)				Total
	Run (3 miles)	Push-ups (20)	Rowing (15 minutes)	Pull-ups (20)	
Run (3 miles)	0.10	0.40	0.30	0.20	1.00
Push-ups (20)	0.35	0.10	0.25	0.30	1.00
Rowing (15 minutes)	0.40	0.30	0.05	0.25	1.00
Pull-ups (20)	0.42	0.42	0.08	0.08	1.00

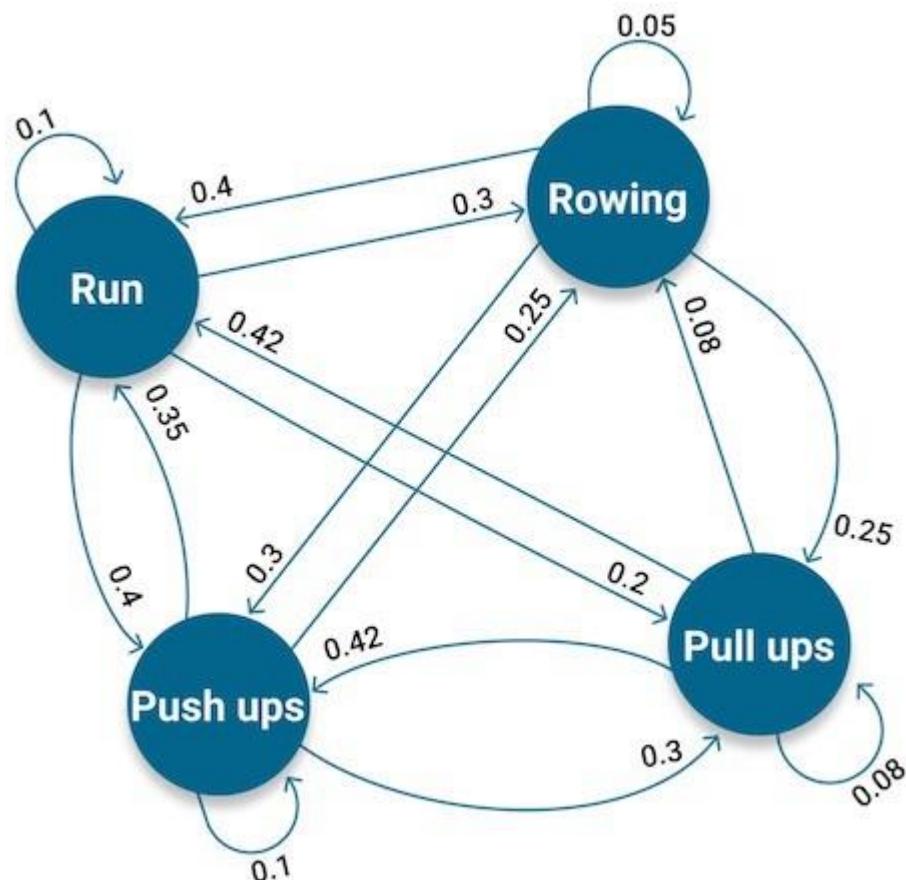
$$P = \begin{pmatrix} \text{Run} & \text{Push-ups} & \text{Rowing} & \text{Pull-ups} \\ \text{Run} & \begin{pmatrix} 0.10 & 0.40 & 0.30 & 0.20 \end{pmatrix} \\ \text{Push-ups} & \begin{pmatrix} 0.35 & 0.10 & 0.25 & 0.30 \end{pmatrix} \\ \text{Rowing} & \begin{pmatrix} 0.40 & 0.30 & 0.05 & 0.25 \end{pmatrix} \\ \text{Pull-ups} & \begin{pmatrix} 0.42 & 0.42 & 0.08 & 0.08 \end{pmatrix} \end{pmatrix}$$

Ex: WORKOUT – Markov Chain



- To understand how the workout routine evolves over time take into account three components :
 - Starting state
 - End state
 - Time-frame
 - how long it will take the model to get from the start to the end state.
- For example
 - Ques 1 - If I start the workout with a run, how likely am I to do push-ups on the second set?**
 - Ques 2 - In 3-set workout, how likely am I to do: 1) run, 2) push-ups and 3) pull-ups?
 - Ques 3- What's the likelihood of doing any of the exercises on the fourth set?

Ex: WORKOUT – Markov Chain

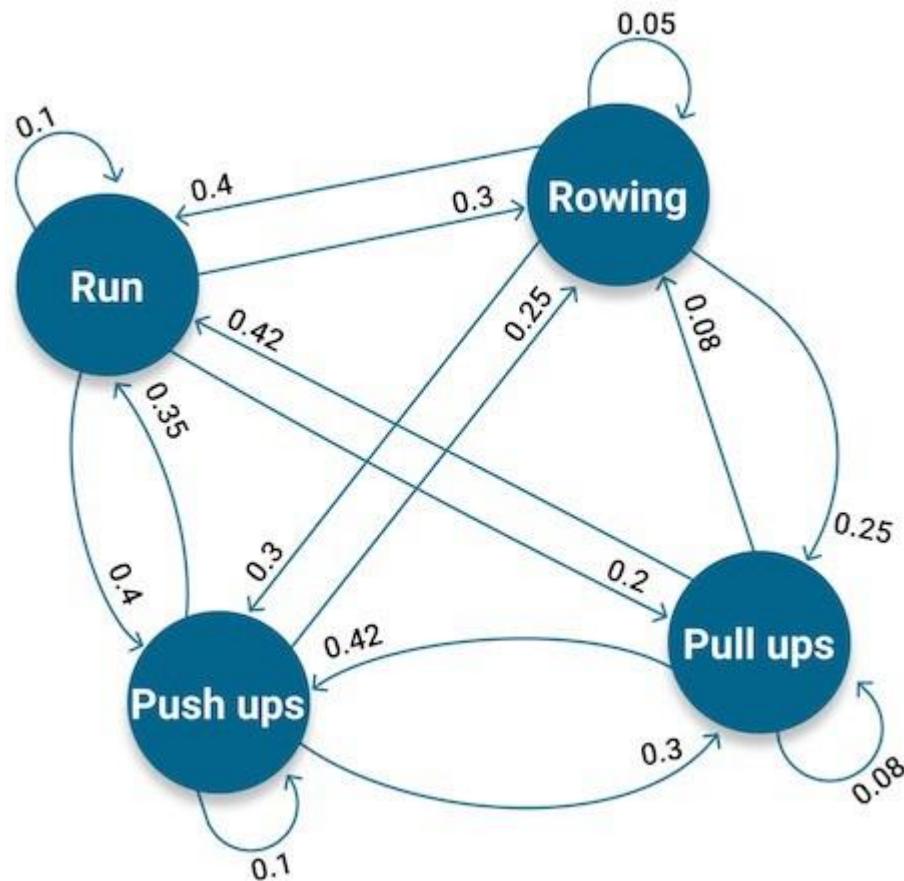


- **Ques 1 - If I start the workout with a run, how likely am I to do push-ups on the second set?**

- Start state is running 3 miles
- End state is doing a 20 push-up set.
- time-frame is two

$$\begin{aligned} P_{\text{Run, Push-ups}} &= P_{\text{Run}} P_{\text{Push-ups} | \text{Run}} + P_{\text{Push-ups}} P_{\text{Push-ups} | \text{Run}} + P_{\text{Rowing}} P_{\text{Push-ups} | \text{Rowing}} + P_{\text{Pull-ups}} P_{\text{Push-ups} | \text{Pull-ups}} \\ &\equiv [P_{\text{Run}} \mid \text{Run}] P_{\text{Push-ups} \mid \text{Run}} + [P_{\text{Push-ups}} \mid \text{Run}] P_{\text{Push-ups} \mid \text{Run}} + [P_{\text{Rowing}} \mid \text{Run}] P_{\text{Push-ups} \mid \text{Rowing}} + [P_{\text{Pull-ups}} \mid \text{Run}] P_{\text{Push-ups} \mid \text{Pull-ups}} \\ &\equiv (0.1 \times 0.1) + (0.1 \times 0.1) + (0.3 \times 0.3) + (0.2 \times 0.1) \\ &\approx 0.01 + 0.01 + 0.09 + 0.08 \\ &\approx 0.25 \end{aligned}$$

Ex: WORKOUT – Markov Chain



Second power of the transition matrix
the state of the Markov Chain at time-step 2

$$P^2 = \begin{pmatrix} \text{Run} & \text{Push-ups} & \text{Rowing} & \text{Pull-ups} \\ \text{Run} & 0.35 & 0.25 & 0.16 & 0.23 \\ \text{Push-ups} & 0.30 & 0.35 & 0.17 & 0.19 \\ \text{Rowing} & 0.28 & 0.31 & 0.22 & 0.2 \\ \text{Pull-ups} & 0.26 & 0.27 & 0.24 & 0.24 \end{pmatrix}$$

Power of the transition matrix

- Future states are calculated using recursion
- Over a large enough number of iterations, all transition probabilities will converge to a value and remain unchanged
- we can say the transition probabilities reached a **steady-state**

$$r_{i,j}^{(n)} = \sum_k r_{i,k}^{(n-1)} p_{k,j}$$

How many time steps in the future

Going from state i to state j

Transition probability from state k to state j

Result of the recursion from state i to state k, which is somewhere in between states i and j

Summary of Markov Model

- **Stochastic Model**
 - a **discrete-time process** indexed at time 1,2,3,...that takes values called **states** which are observed
 - *Example states (S) = {hot , cold }*
 - *State series over time => $z \in S_T$*
 - *Weather for 4 days can be a sequence => {z1=hot, z2 =cold, z3 =cold, z4 =hot}*
- **Markov model** is engineered to handle data which can be represented as ‘sequence’ of observations over time.
- **Markov Assumptions**
 1. **Limited Horizon assumption:** Probability of being in a state at a time t depend only on the state at the time (t-1).
$$P(z_t, z_{t-1}, z_{t-2}, \dots, z_1) = P(z_t / z_{t-1})$$
- means state at time t represents enough summary of the past reasonably to predict the future.

Summary of Markov Model

- **Stationary Process Assumption:** Conditional (probability) distribution over the next state, given the current state, doesn't change over time.

$$P(z_t / z_{t-1}) = P(z_2 / z_1) \longrightarrow t \in 2 \dots T$$

$P(s_{\text{hot}} / s_0)$

i / j	s_0	s_{hot}	s_{cold}	s_{rain}
s_0		0.33	0.33	0.34
s_{hot}	0	0.8	0.1	0.1
s_{cold}	0	0.2	0.6	0.2
s_{rain}	0	0.1	0.2	0.7

State Transition Matrix

find out the probability of sequence —>
 $\{z_1 = s_{\text{hot}}, z_2 = s_{\text{cold}}, z_3 = s_{\text{rain}}, z_4 = s_{\text{rain}}, z_5 = s_{\text{cold}}\}$

Summary of Markov Model

Two Main Questions in Markov-model

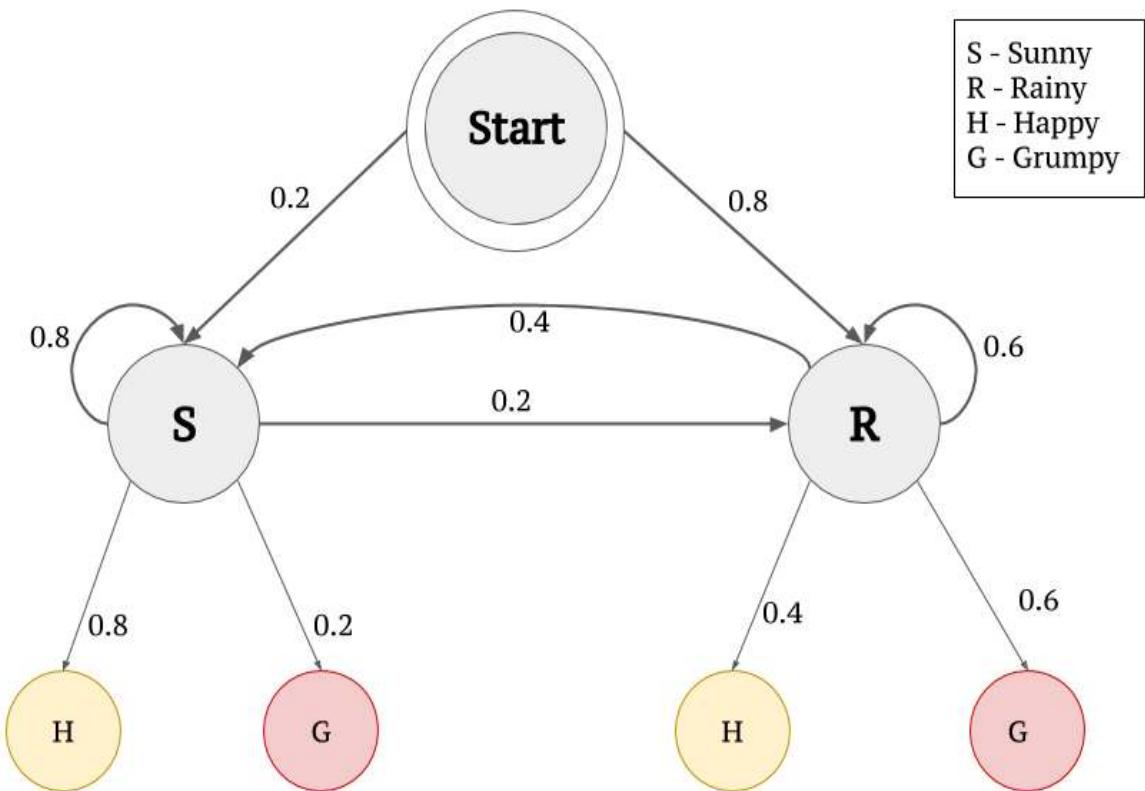
1. Probability of particular sequences of state z?
2. How do we estimate the parameter of state transition matrix A to maximize the likelihood of the observed sequence?

$$\begin{aligned} P(\vec{z}) &= P(z_t, z_{t-1}, \dots, z_1; A) \\ &= P(z_t, z_{t-1}, \dots, z_1, z_0; A) && \cdot \text{ Initial State} \\ &= P(z_t / z_{t-1}, \dots, z_1, z_0; A) P(z_{t-1} / z_{t-2}, \dots, z_0; A) \dots P(z_1 / z_0; A) && \text{Chain rule of probability} \\ &= P(z_t | z_{t-1}; A) P(z_{t-1} / z_{t-2}; A) \dots P(z_2 / z_1; A) P(z_1 / z_0; A) && \text{Limited Horizon Assumption} \\ &= \prod_{z_{t-1}, z_t} P(z_t / z_{t-1}; A) \\ &= \prod_{A_{z_{t-1}, z_t}} \end{aligned}$$

<https://towardsdatascience.com/markov-and-hidden-markov-model-3eec42298d75>

Hidden Markov Model

- HMM
 - is a probabilistic model to infer unobserved information from observed data
 - Cannot observe the state themselves but only the result of some probability function(observation) of the states.
- HMM is a statistical **Markov model** in which the system being modeled is assumed to be a **Markov process** with unobserved (**hidden**) states.
 - **Markov Model:** Series of (hidden) states $z=\{z_1, z_2, \dots\}$ drawn from state alphabet $S=\{s_1, s_2, \dots, s_{|S|}\}$ where z_i belongs to S .
 - **Hidden Markov Model:** Series of observed output $x = \{x_1, x_2, \dots\}$ drawn from an output alphabet $V = \{v_1, v_2, \dots, v_{|V|}\}$ where x_i belongs to V



Set of states (S) = {Happy, Grumpy}

Set of hidden states (Q) = {Sunny , Rainy}

State series over time = $z \in S_T$

Observed States for 4 days = { $z_1=$ Happy, $z_2=$ Grumpy, $z_3=$ Grumpy, $z_4=$ Happy}

Assumptions of HMM

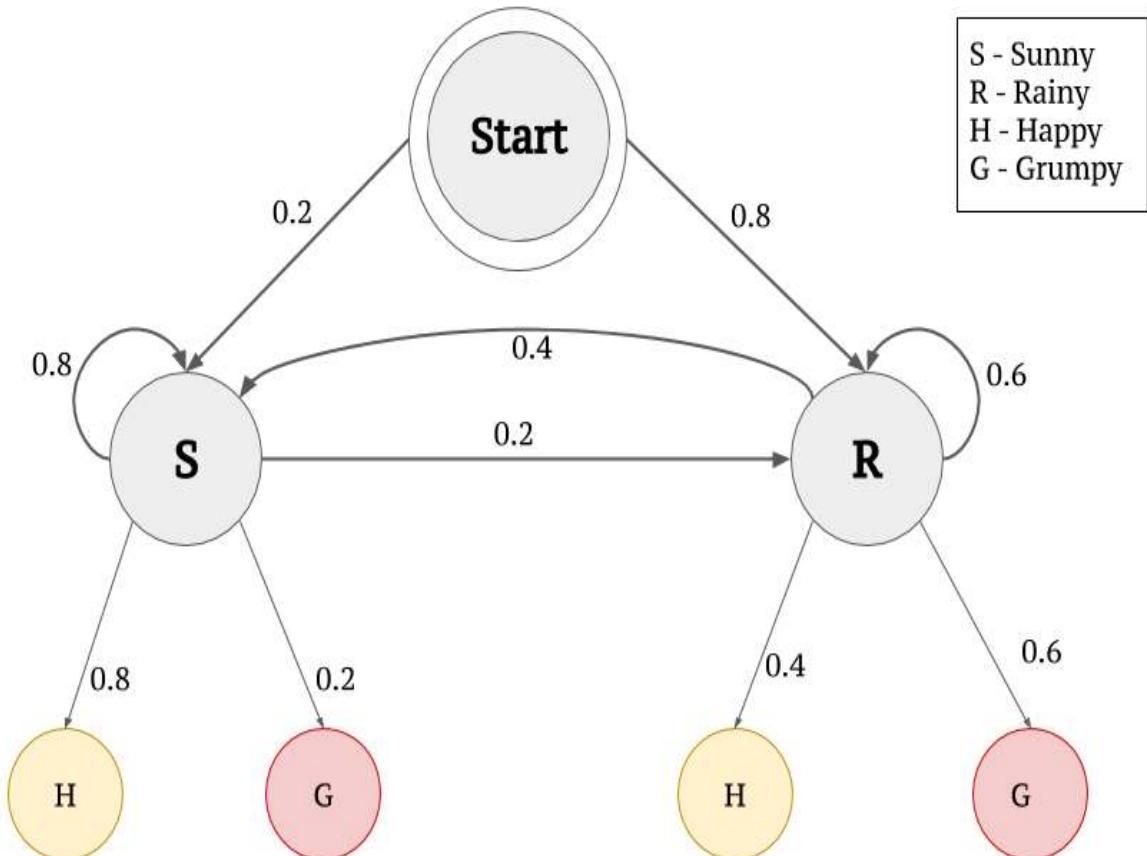
Output independence assumption:

- Output observation is conditionally independent of all other hidden states and all other observations when *given the current hidden state.*

$$P(x_t = v_i / z_t = s_j) = P(x_t = v_i / x_1, x_2, \dots, x_T, z_1, z_2, \dots, z_T) = B_{ji}$$

- **Emission Probability Matrix:**

- Probability of hidden state generating output v_i given that state at the corresponding time was s_j .



Three important questions in HMM are

1. What is the probability of an observed sequence?
2. What is the most likely series of states to generate an observed sequence?
3. How can we learn the values for the HMMs parameters A and B given some data?

Probability of Observed Sequence

1. Forward Procedure

Calculate the total probability of all the observations (from t_1) up to time t.

$$\alpha_i(t) = P(x_1, x_2, \dots, x_t, z_t = s_i; A, B)$$

2. Backward Procedure

Similarly calculate total probability of all the observations from final time (T) to t.

$$\beta_i(t) = P(x_T, x_{T-1}, \dots, x_{t+1}, z_t = s_i; A, B)$$

<https://towardsdatascience.com/markov-and-hidden-markov-model-3eec42298d75>

HMM – Forward Procedure

$$A \text{ (State Transition Matrix)} = \begin{matrix} & H & C \\ H & \left\{ \begin{array}{cc} 0.7 & 0.3 \\ 0.4 & 0.6 \end{array} \right\} \\ C & \end{matrix}$$

- $S = \{\text{hot}, \text{cold}\}$
- $v = \{v_1=1 \text{ ice cream}, v_2=2 \text{ ice cream}, v_3=3 \text{ ice cream}\}$
 - where V is the Number of ice creams consumed on a day.
- Example Sequence =
 $\{x_1=v_2, x_2=v_3, x_3=v_1, x_4=v_2\}$

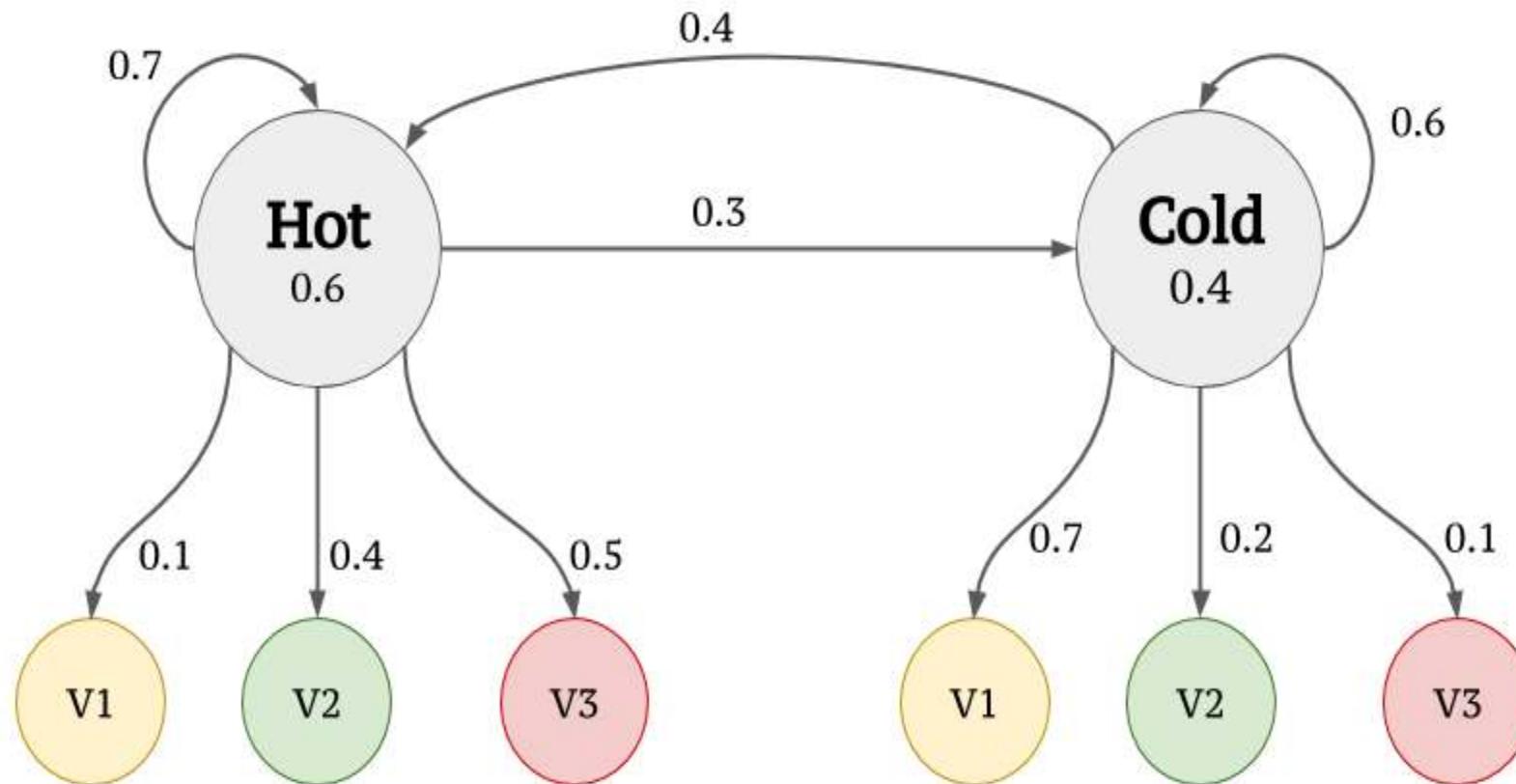
$$B \text{ (Emission Matrix)} = \begin{matrix} & V1 & V2 & V3 \\ H & \left\{ \begin{array}{ccc} 0.1 & 0.4 & 0.5 \end{array} \right\} \\ C & \left\{ \begin{array}{ccc} 0.7 & 0.2 & 0.1 \end{array} \right\} \end{matrix}$$

$$\pi \text{ (Initial state } S_0) = \left\{ \begin{array}{cc} 0.6 & 0.4 \end{array} \right\}$$

$$A \text{ (State Transition Matrix)} = \begin{matrix} & H & C \\ \pi & \left\{ \begin{array}{cc} 0.6 & 0.4 \end{array} \right\} \\ H & \left\{ \begin{array}{cc} 0.7 & 0.3 \end{array} \right\} \\ C & \left\{ \begin{array}{cc} 0.4 & 0.6 \end{array} \right\} \end{matrix}$$

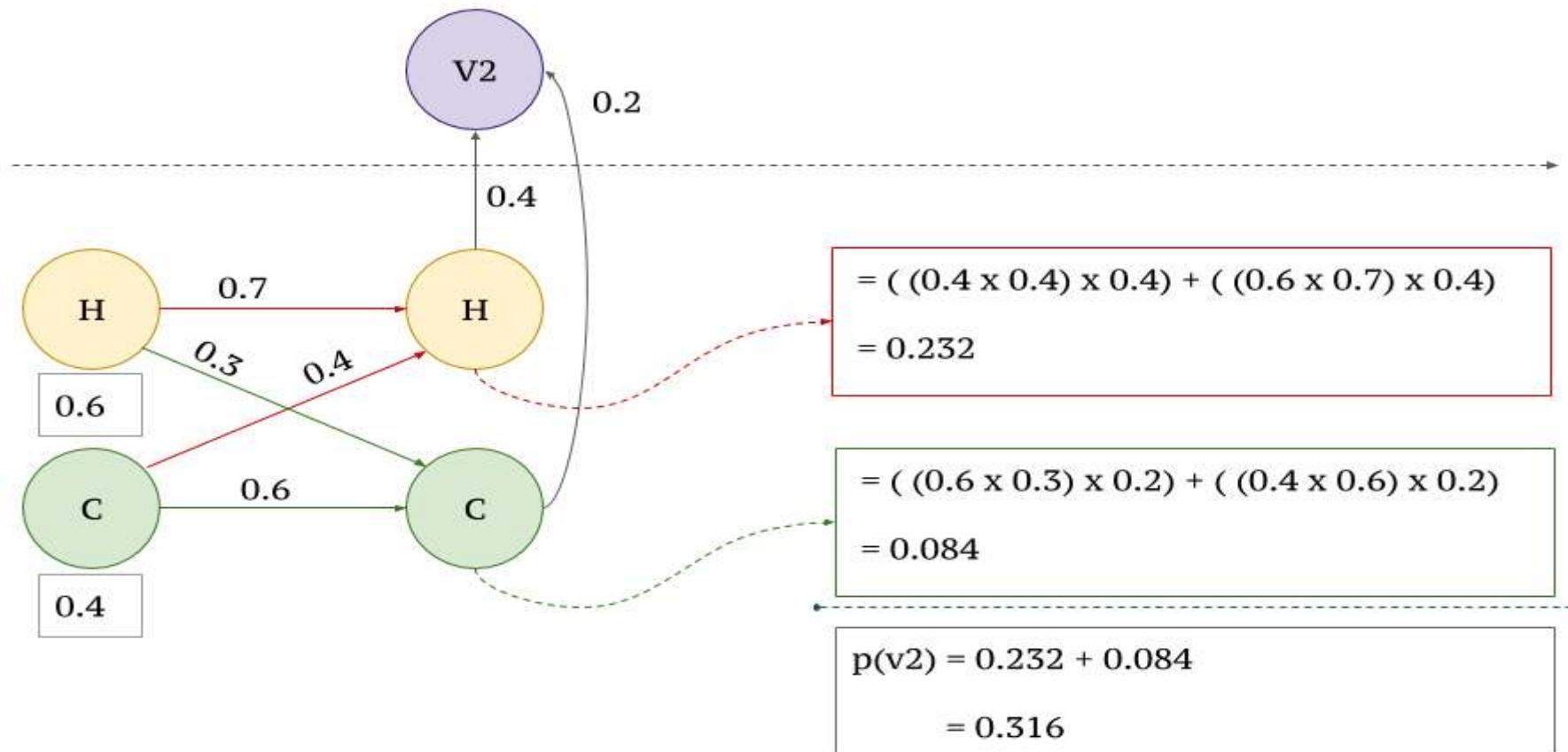
<https://towardsdatascience.com/markov-and-hidden-markov-model-3eec42298d75>

HMM – Forward Procedure



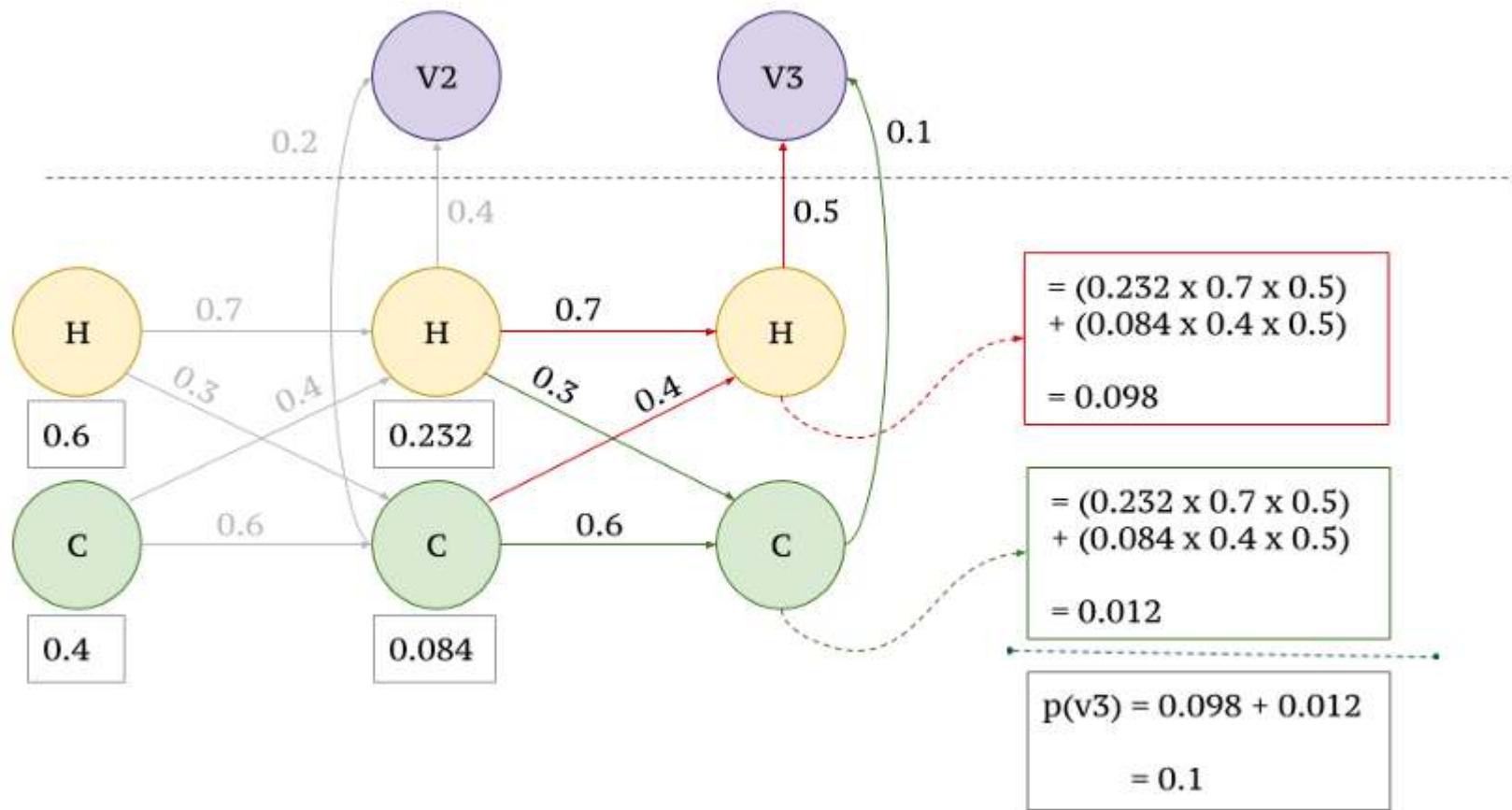
HMM- Forward Procedure

- For first observed output $x_1=v_2$



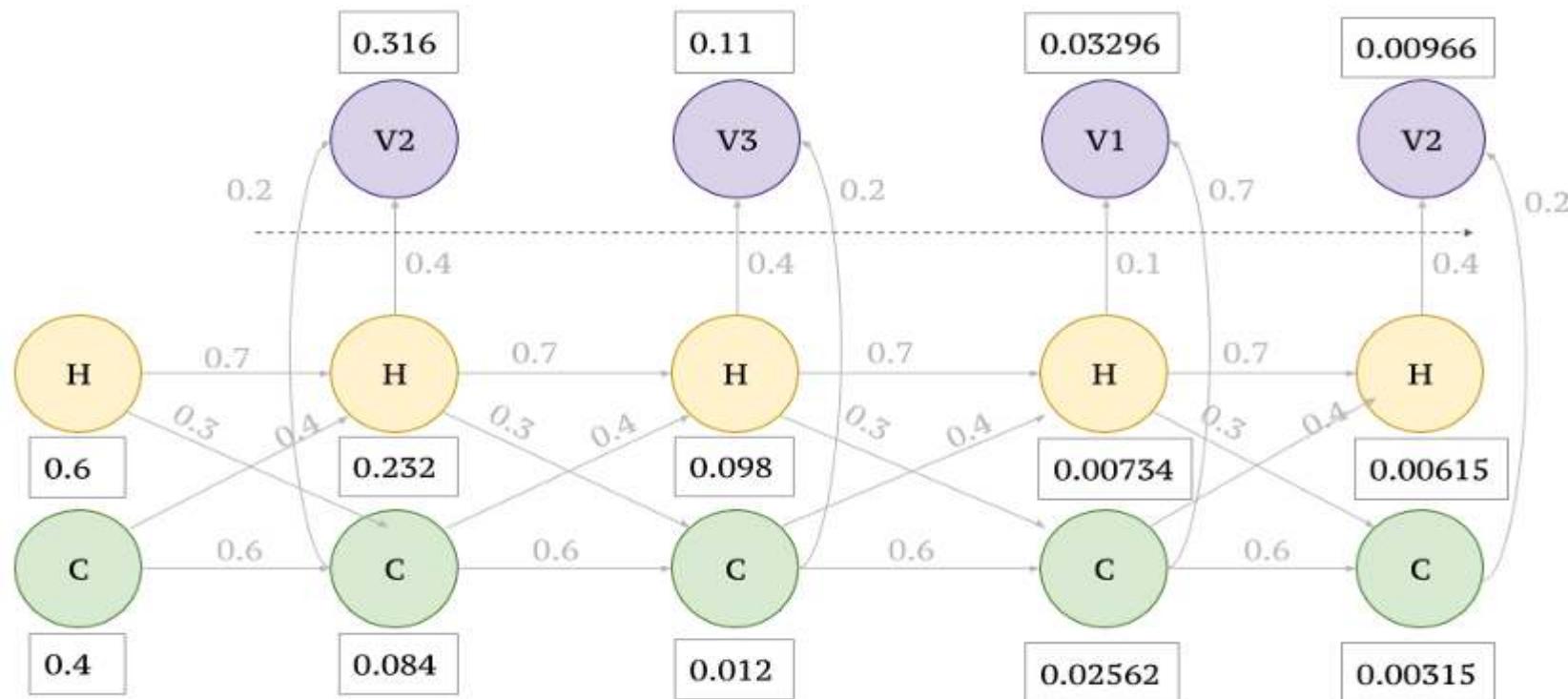
HMM-Forward Procedure

- 2. for observed output $x_2=v_3$



HMM-Forward Procedure

- for observed output x_3 and x_4
 - Similarly for $x_3=v_1$ and $x_4=v_2$, we have to multiply the paths that lead to v_1 and v_2 .



Hidden Markov Model Algorithm

- **Step 1: Define the state space and observation space**
 - The state space is the set of all possible hidden states, and the observation space is the set of all possible observations
- **Step 2: Define the initial state distribution**
- **Step 3: Define the state transition probabilities**
- **Step 4: Define the observation likelihoods:**
 - Emission matrix - Probabilities of generating each observation from each hidden state
 - <https://www.geeksforgeeks.org/hidden-markov-model-in-machine-learning/>

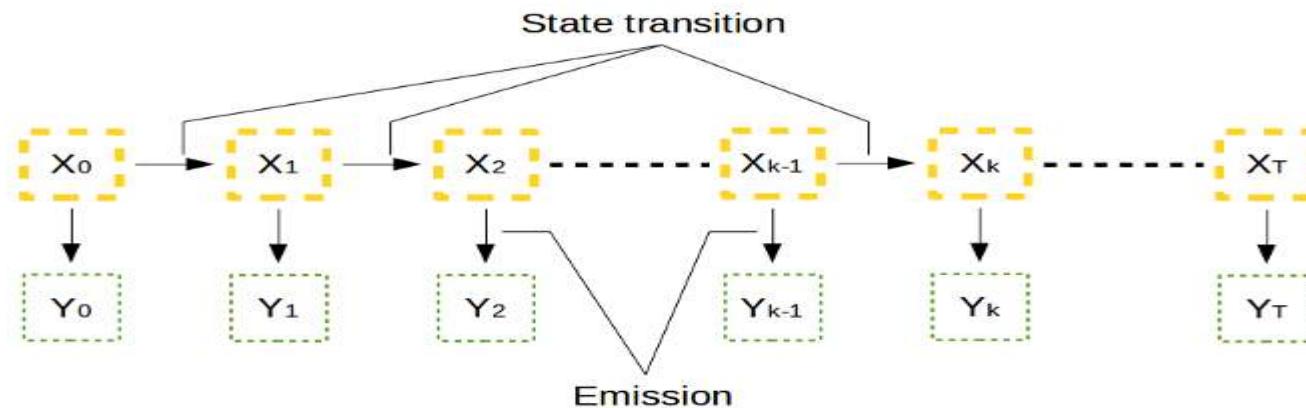
Hidden Markov Model Algorithm

- **Step 5:Train the model**
 - Baum-Welch algorithm, or the forward-backward algorithm used to estimate parameters of the state transition probabilities and the observation likelihoods
 - Iteratively updating the parameters until convergence.
- **Step 6:Decode the most likely sequence of hidden states**
 - Given the observed data, the Viterbi algorithm is used to compute the most likely sequence of hidden states
 - This can be used to predict future observations, classify sequences, or detect patterns in sequential data.
- **Step 7:Evaluate the model**
 - using various metrics, such as accuracy, precision, recall, or F1 score.

Baum-Welch Algorithm

- Also known as the **forward-backward** algorithm
- is a dynamic programming approach and a special case of the expectation-maximization algorithm
- purpose is to tune the parameters of the HMM such that the model is maximally like the observed data.
 - the state transition matrix \mathbf{A}
 - the emission matrix \mathbf{B}
 - the initial state distribution π_0 , such that the model is maximally like the observed data.
- Includes the
 1. **Initial phase**
 2. **Forward phase**
 3. **Backward phase**
 4. **Update phase. λ**
- The forward and the backward phase form the E-step of the EM algorithm
 - computes expected hidden states given the observed data and the set of parameter matrices before-tuned
 - the update phase itself is the M-step
 - update formulas then tune the parameter matrices to best fit the observed data and the expected hidden states

Baum-Welch Algorithm



1. Initial phase

- parameter matrices A, B, π_0 are initialized
- Can be done randomly if there is no prior knowledge about them.

2. Forward phase

$$\alpha(X_k) \rightarrow P[Y_{0:k}, X_k] \rightarrow \sum_{X_{k-1}} \alpha(X_{k-1}) P(X_k | X_{k-1}) P(Y_k | X_k)$$

- α is the joint probability of the observed data up to time k and the state at time k

$$\alpha(X_0) = P[Y_0, X_0] = P[Y_0 | X_0] P[X_0]$$

<https://medium.com/analytics-vidhya/baum-welch-algorithm-for-training-a-hidden-markov-model-part-2-of-the-hmm-series-d0e393b4fb86>

Baum-Welch Algorithm

1. Backward Phase

$$\beta(X_k) = P[Y_{k+1:T}|X_k] = \sum_{X_{k+1}} \beta(X_{k+1})P(X_{k+1}|X_k)P(Y_{k+1}|X_{k+1})$$

- β function is defined as the conditional probability of the observed data from time k+1 given the state at time k
- The second term of the R.H.S. is the state transition probability from A, while the last term is the emission probability from B.
- The R.H.S. is summed over all possible states at time k +1.

$$\beta(X_T) = 1$$

Baum-Welch Algorithm

$$P[X_k|Y_{0:T}] \xrightarrow{\text{blue arrow}} \frac{\alpha(X_k)\beta(X_k)}{P[Y_{0:T}]} \propto \alpha(X_k)\beta(X_k)$$

4. Update phase

Probability distribution of a state at time k given all observed data we have

$$\eta(X_k) \xrightarrow{\text{blue arrow}} P[X_k|Y_{0:T}] = \frac{\alpha(X_k)\beta(X_k)}{\sum_{X_k} \alpha(X_k)\beta(X_k)}$$

Joint probability of two consecutive states given the data

$$\begin{aligned}\xi(X_k, X_{k+1}) &= P[X_k, X_{k+1}|Y_{0:T}] \\ &= \frac{\alpha(X_k)\beta(X_{k+1})P[X_{k+1}|X_k]P[Y_{k+1}|X_{k+1}]}{\sum_{X_k} \alpha(X_k)\beta(X_{k+1})P[X_{k+1}|X_k]P[Y_{k+1}|X_{k+1}]}\end{aligned}$$

Baum-Welch Algorithm

4. Update phase

$$\pi_0^* = \eta(X_0)$$

$$a_{ij}^* = P[X_k = j | X_{k-1} = i] = \frac{\sum_k \xi(X_k = j, X_{k-1} = i)}{\sum_k \eta(X_{k-1} = i)}$$

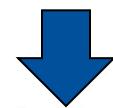
$$b_{ij}^* = P[Y_k = j | X_k = i] = \frac{\sum_k \eta(X_k = i) \times 1_{Y_k=j}}{\sum_k \eta(X_k = i)}$$

HMM – Viterbi Algorithm

$$X_{0:T}^* = \operatorname{argmax}_{X_{0:T}} P[X_{0:T}|Y_{0:T}]$$

To find the best set of states

$$\mu(X_k) = \max_{X_{0:k-1}} P[X_{0:k}, Y_{0:k}] = \max_{X_{k-1}} \mu(X_{k-1}) P[X_k|X_{k-1}] P[Y_k|X_k]$$



$$\mu(X_0) = P[Y_0|X_0]P[X_0]$$

$$\mu(X_1) = \max_{X_0} \mu(X_0) P[X_1|X_0] P[Y_1|X_1]$$

$$\mu(X_2) = \max_{X_1} \mu(X_1) P[X_2|X_1] P[Y_2|X_2]$$

$$\mu(X_3) = \max_{X_2} \mu(X_2) P[X_3|X_2] P[Y_3|X_3]$$

<https://medium.com/analytics-vidhya/baum-welch-algorithm-for-training-a-hidden-markov-model-part-2-of-the-hmm-series-d0e393b4fb86>

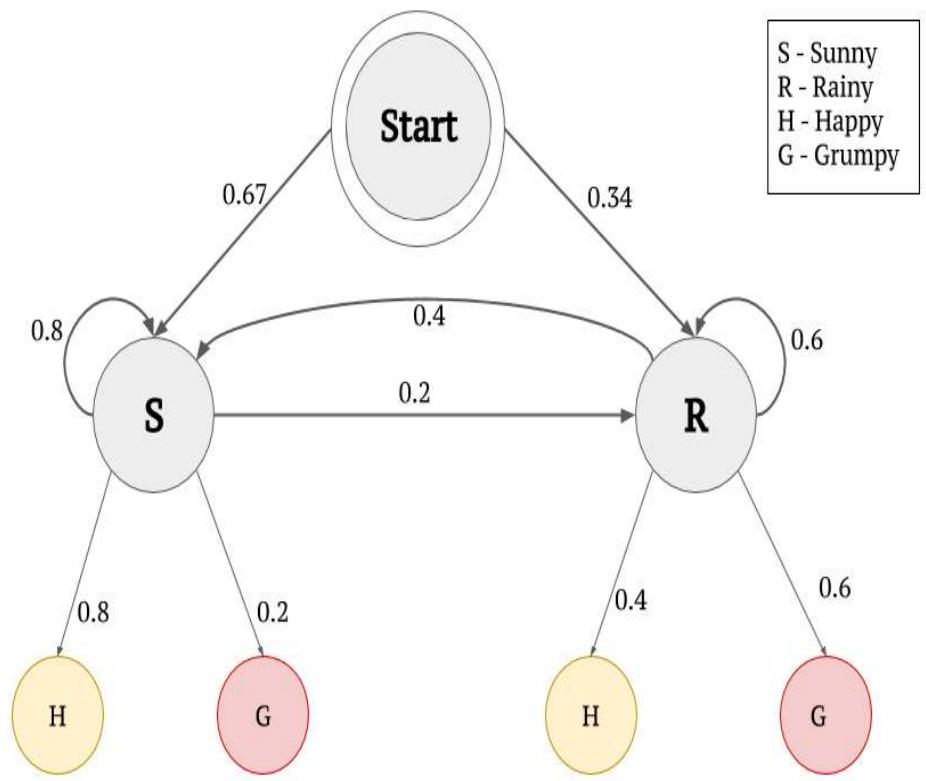
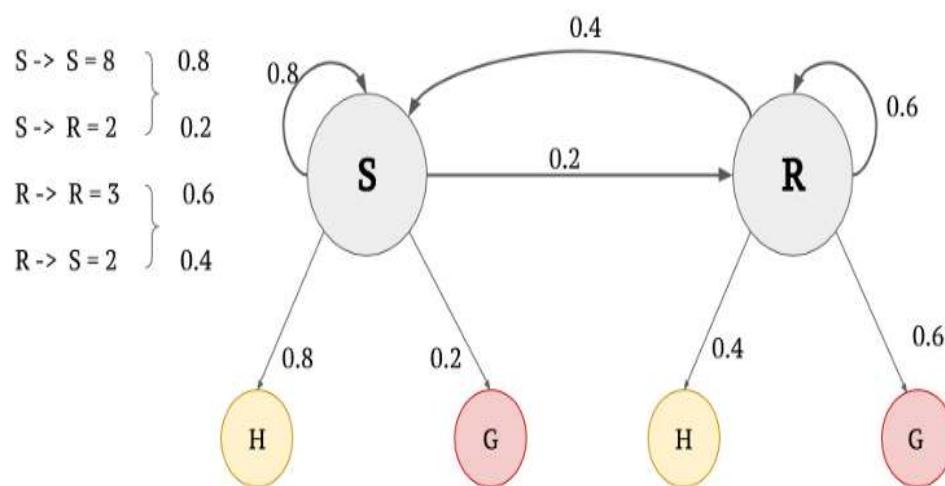
HMM- Maximum Likelihood

Maximum likelihood assignment

- For a given observed sequence of outputs $x \in V_T$, we intend to find the most likely series of states $z \in S_T$. We can understand this with an example found below.

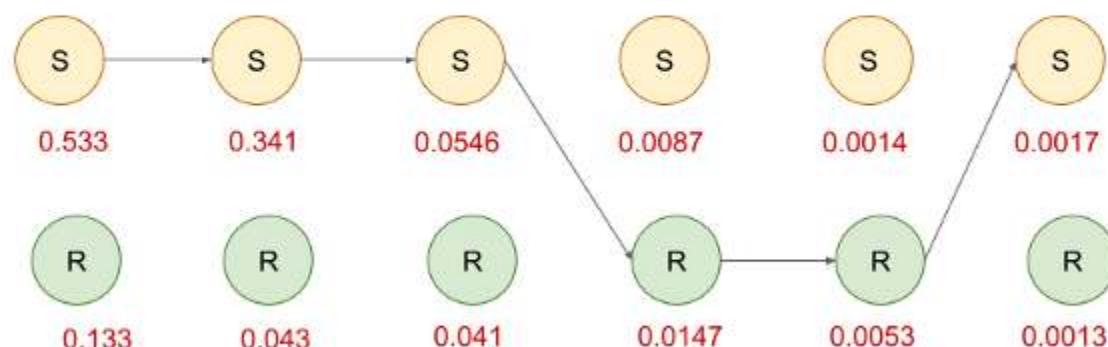
Climate	S	S	S	S	R	R	R	S	S	S	S	R	R	S	S	S
Emotion	G	H	H	H	G	H	H	G	H	H	H	G	H	H	H	H

S - Sunny
R - Rainy
H - Happy
G - Grumpy



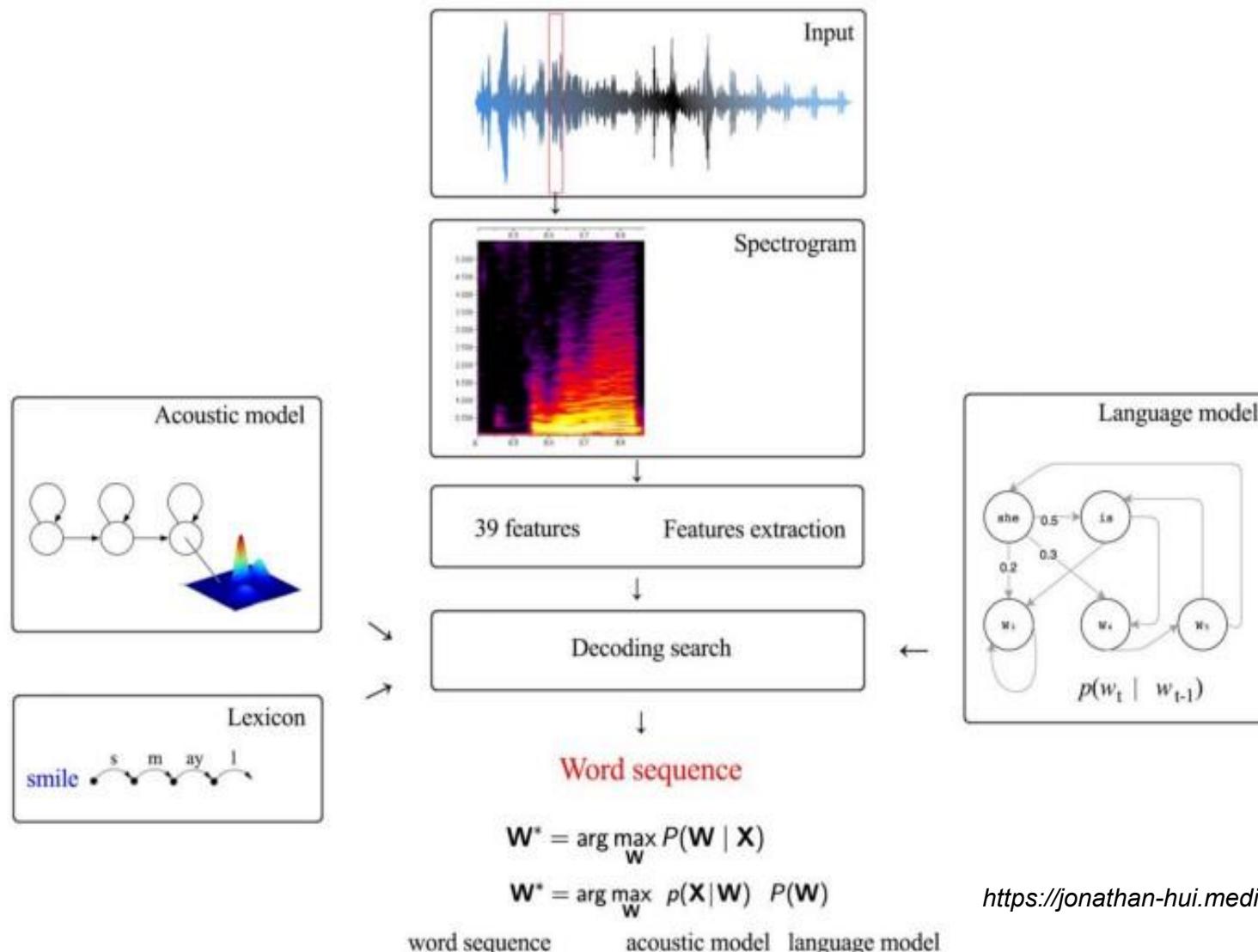
HMM – Viterbi Algorithm

Days	M	T	W	Th	F	S
Emotion	H	H	G	G	G	H
Best Path						
Best Path						
S - Sunny R - Rainy H - Happy G - Grumpy						



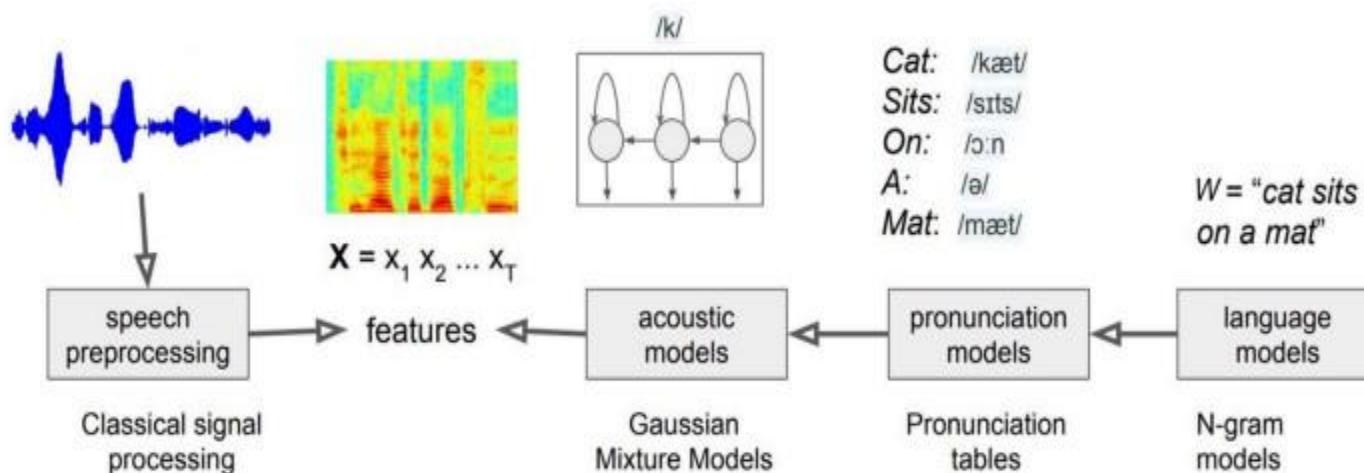
More likelihood sequence = **S S S R R S**
For the given output sequence H H G G G H

Automatic Speech Recognition



- **ACOUSTIC MODEL**
Uses audio recordings of speech & compiles to statistical representations of the sounds for words.
- **Language model**
 - which gives the probabilities of sequences of words.
- **Lexicon**
 - set of words with their pronunciations broken down into phonemes

Building blocks of ASR



$$W^* = \arg \max_W p(X|W) p(W)$$

acoustic model language model

$$W^* = \arg \max_W P(W|X)$$

$$W^* = \arg \max_W P(X|W) P(W) / P(X) \quad \text{Apply Bayes' Theorem}$$

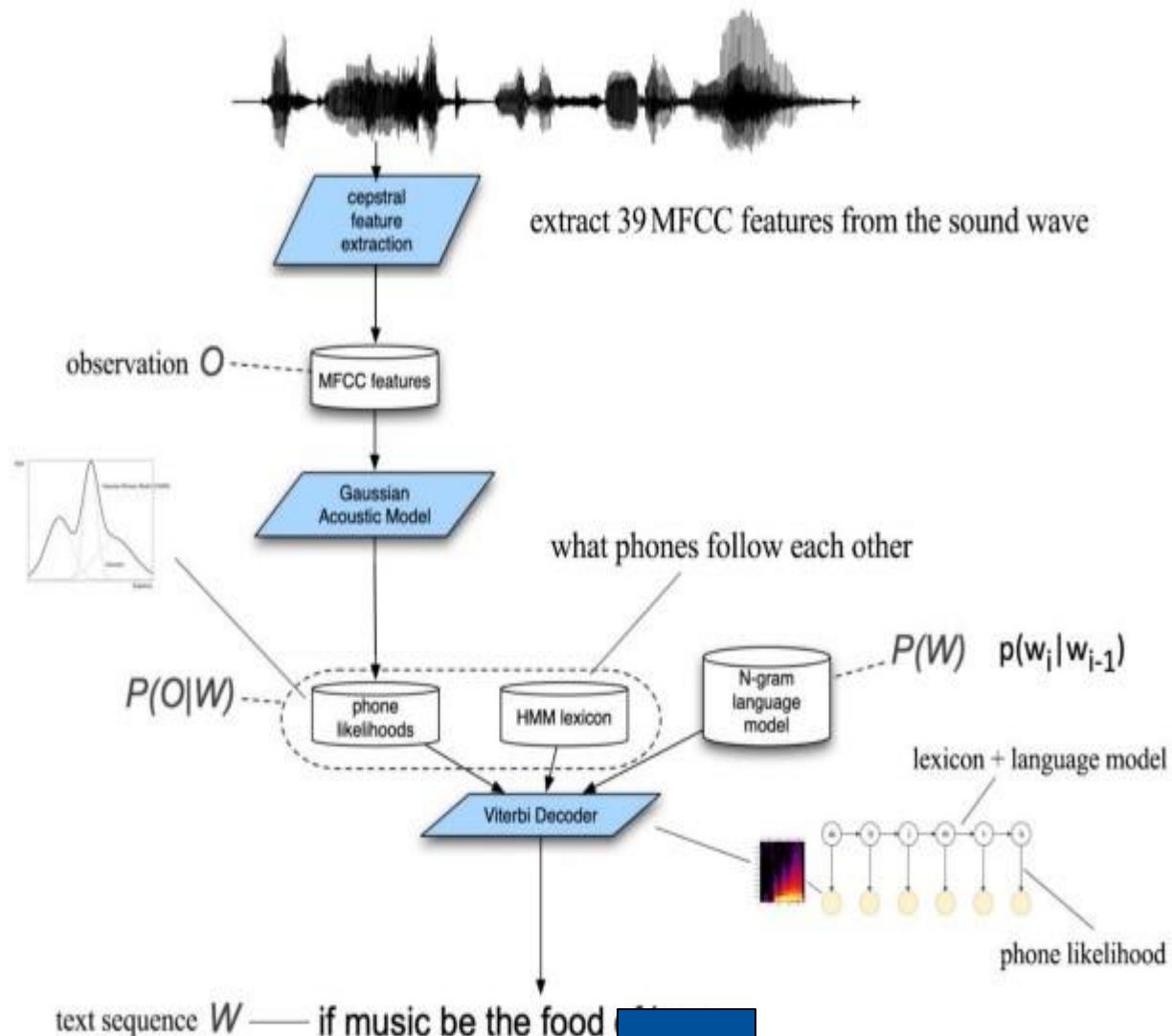
$$W^* = \arg \max_W P(X|W) P(W)$$

word sequence acoustic model language model

Remove $P(X)$ because X does not change w.r.t. W

<https://jonathan-hui.medium.com/speech-recognition-gmm-hmm-8bb5eff8b196>

Realization of ASR



Optimizers & Practical Methodology

DSE 3151, B.Tech Data Science & Engineering

Rohini R Rao & Abhilash Pai

Department of Data Science and Computer Applications

MIT Manipal

Gradient Descent Algorithm

Algorithm: gradient_descent()

$t \leftarrow 0;$

$max_iterations \leftarrow 1000;$

Initialize $\theta_0 = [W_1^0, \dots, W_L^0, b_1^0, \dots, b_L^0];$

while $t++ < max_iterations$ **do**

$h_1, h_2, \dots, h_{L-1}, a_1, a_2, \dots, a_L, \hat{y} = forward_propagation(\theta_t);$

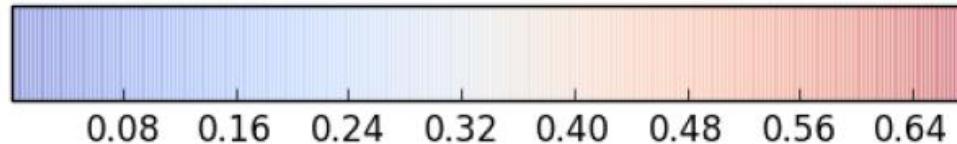
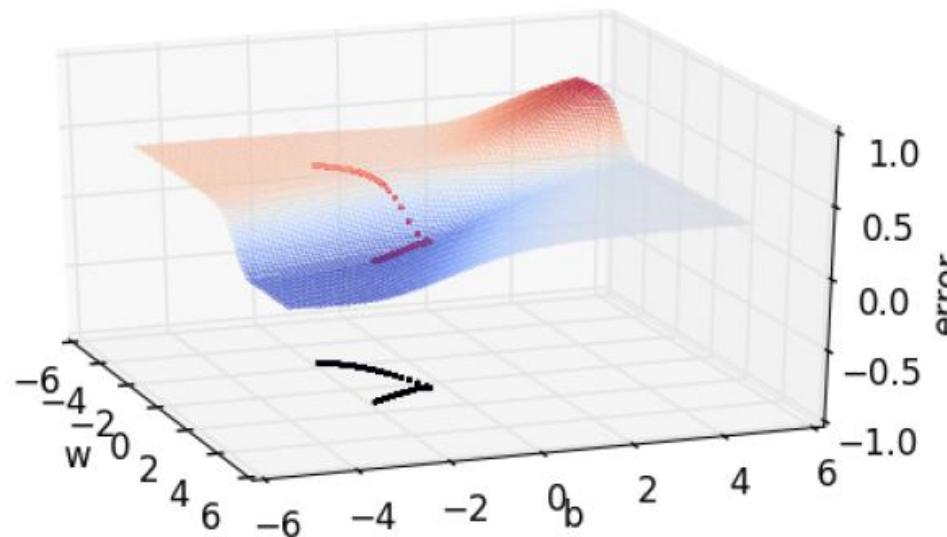
$\nabla\theta_t = backward_propagation(h_1, h_2, \dots, h_{L-1}, a_1, a_2, \dots, a_L, y, \hat{y});$

$\theta_{t+1} \leftarrow \theta_t - \eta \nabla\theta_t;$

end

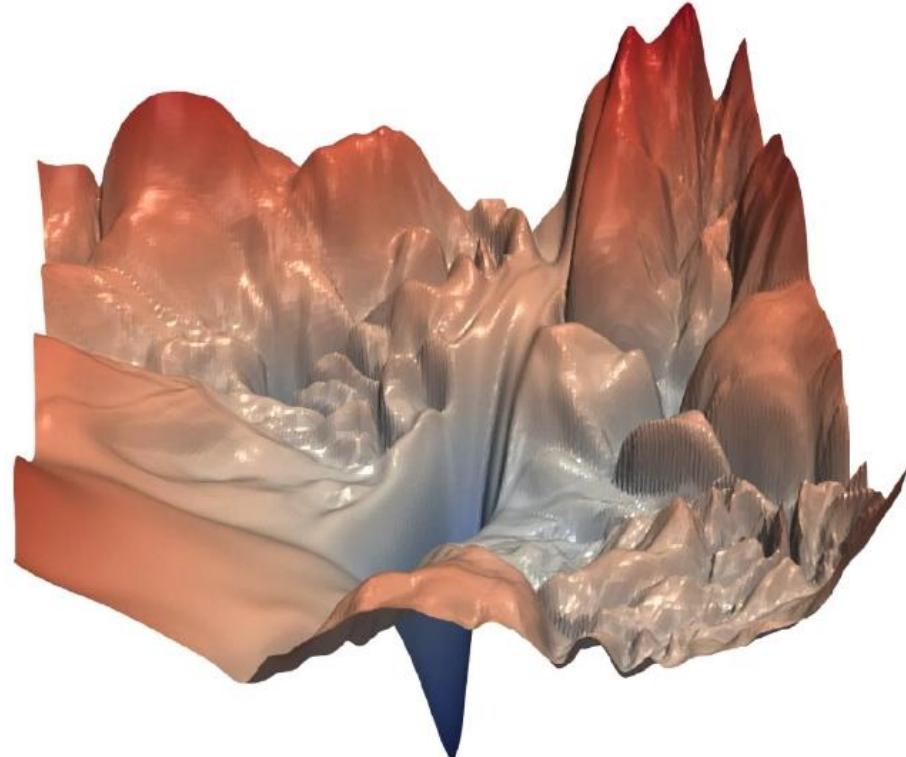
Gradient descent: Error surface

Gradient descent on the error surface



Gradient descent: Error surface

Visualization of error surface of a neural network (ResNet-56)



Gradient descent and its variants

Vanilla (Batch) Gradient Descent

Require: Learning rate α , initial parameters θ_t , training dataset \mathcal{D}_{tr}

- 1: **while** stopping criterion not met **do**
 - 2: Initialize parameter updates $\Delta\theta_t = 0$
 - 3: **for each** $(x^{(i)}, y^{(i)})$ in \mathcal{D}_{tr} **do**
 - 4: Compute gradient using backpropagation $\nabla_{\theta_t} \mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$
 - 5: Aggregate gradient $\Delta\theta_t = \Delta\theta_t + \nabla_{\theta_t} \mathcal{L}$
 - 6: **end for**
 - 7: Apply update $\theta_{t+1} = \theta_t - \alpha \frac{1}{|\mathcal{D}_{tr}|} \Delta\theta_t$
 - 8: **end while**
-

What is the issue here?

Think of how the algorithm will work for large Dataset (Eg: ImageNet with Billions of Data)

Gradient descent and its variants

Stochastic Gradient Descent

Require: Learning rate α , initial parameters θ_t , training dataset \mathcal{D}_{tr}

```
1: while stopping criterion not met do
2:   for each  $(x^{(i)}, y^{(i)})$  in  $\mathcal{D}_{tr}$  do
3:     Compute gradient using backpropagation  $\nabla_{\theta_t} \mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$ 
4:     Gradient  $\Delta\theta_t = \nabla_{\theta_t} \mathcal{L}$ 
5:     Apply update  $\theta_{t+1} = \theta_t - \alpha \Delta\theta_t$ 
6:   end for
7: end while
```

Mini Batch Gradient Descent Deep Learning Optimizer

- **Batch gradient descent:**
 - gradient is average of gradients computed from ALL the samples in dataset
- Mini Batch GD:
 - subset of the dataset is used for calculating the loss function, therefore fewer iterations are needed.
 - batch size of 32 is considered to be appropriate for almost every case.
 - Yann Lecun (2018) – “Friends don’t let friends use mini batches larger than 32”
- is faster , more efficient and robust than the earlier variants of gradient descent.
- the cost function is noisier than the batch GD but smoother than SDG.
- Provides a good balance between speed and accuracy.
- It needs a hyperparameter that is “mini-batch-size”, which needs to be tuned to achieve the required accuracy.

Gradient descent and its variants

Mini-Batch Stochastic Gradient Descent

Require: Learning rate α , initial parameters θ_t , mini-batch size m, training dataset \mathcal{D}_{tr}

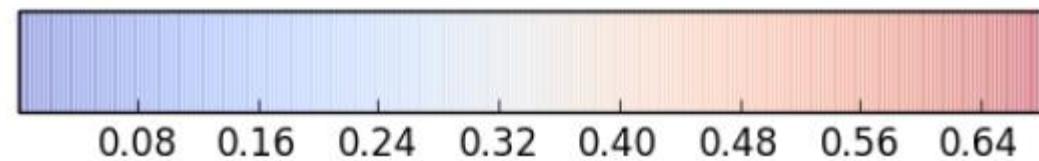
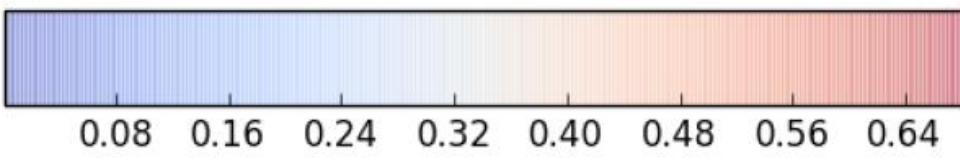
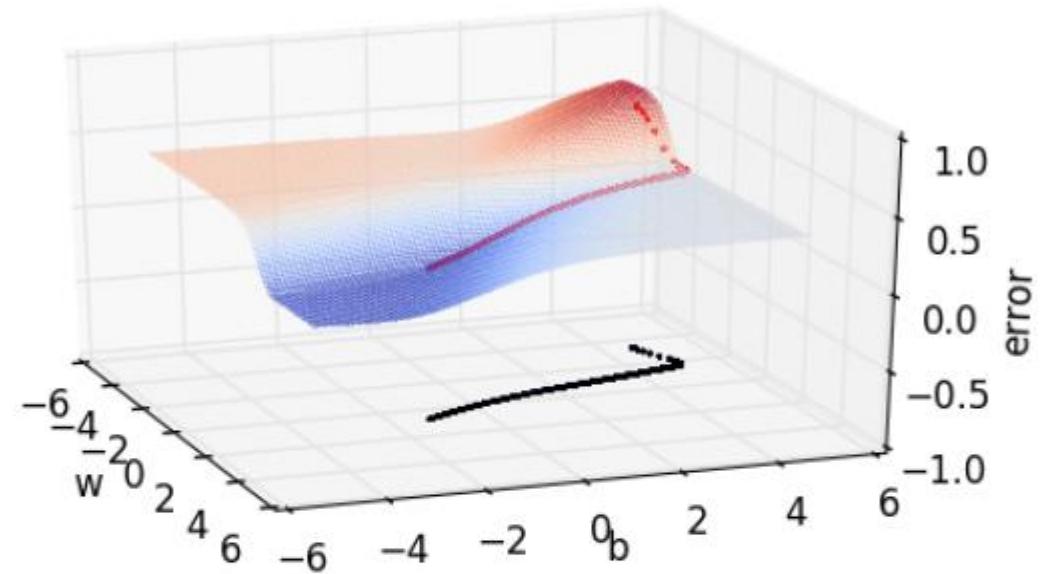
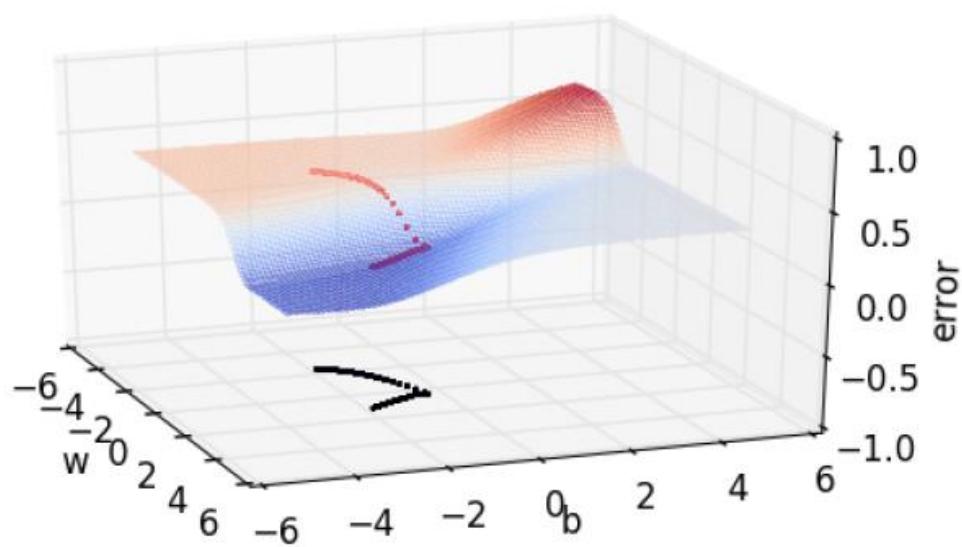
```
1: while stopping criterion not met do
2:   Initialize gradients  $\Delta\theta_t = 0$ 
3:   Sample m examples from  $\mathcal{D}_{tr}$  (call it  $\mathcal{D}_{mini}$ )
4:   for each  $(x^{(i)}, y^{(i)})$  in  $\mathcal{D}_{mini}$  do
5:     Compute gradient using backpropagation  $\nabla_{\theta_t} \mathcal{L}(\theta_t; x^{(i)}, y^{(i)})$ 
6:     Aggregate gradient  $\Delta\theta_t = \Delta\theta_t + \nabla_{\theta_t} \mathcal{L}$ 
7:   end for
8:   Apply update  $\theta_{t+1} = \theta_t - \alpha \Delta\theta_t$ 
9: end while
```

Gradient descent and its variants

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

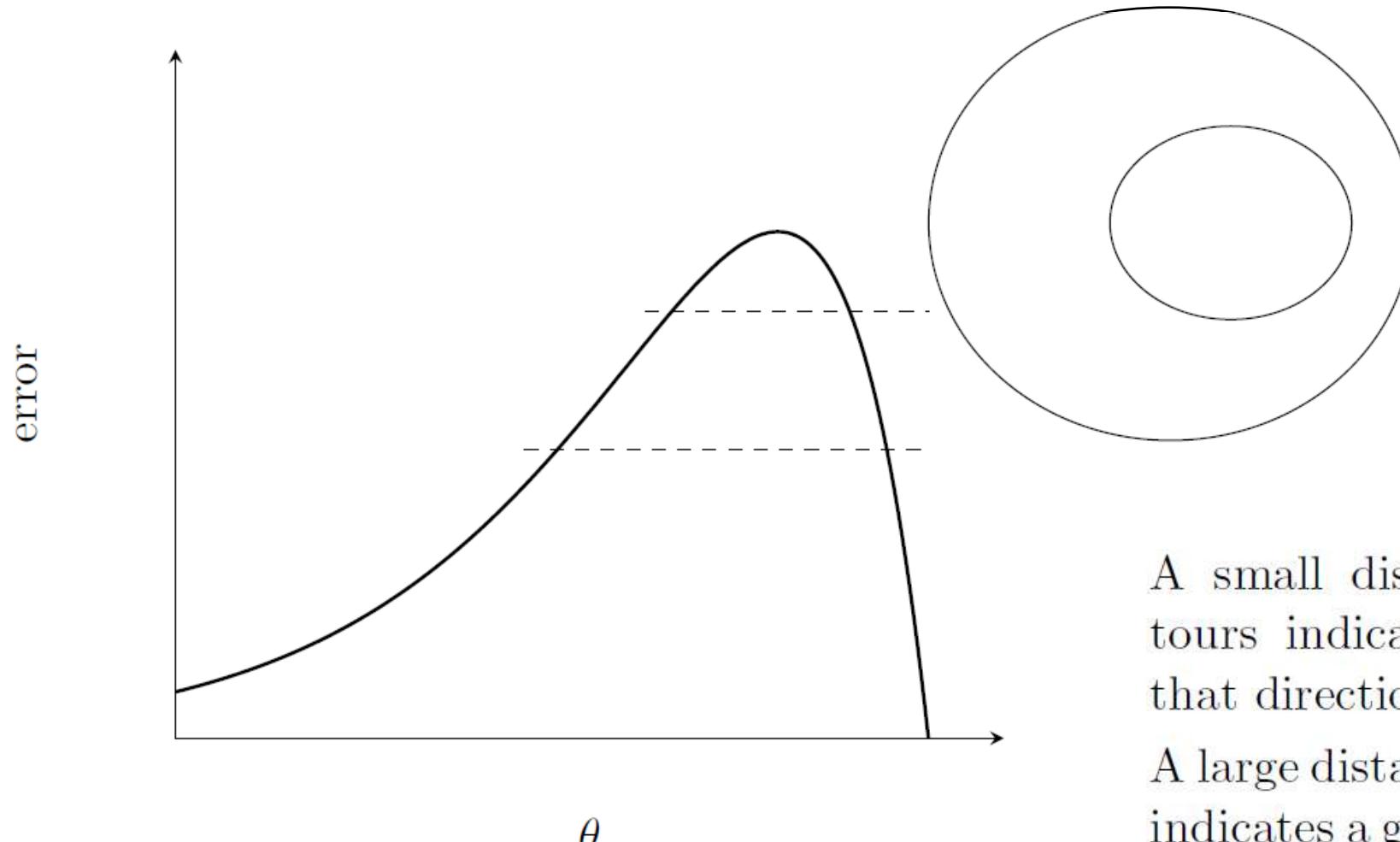
Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	1
Stochastic Gradient Descent	N
Mini-Batch Gradient Descent	$\frac{N}{B}$

Gradient descent: Behaviour for different initializations



Slow in flat regions and fast in steep slopes

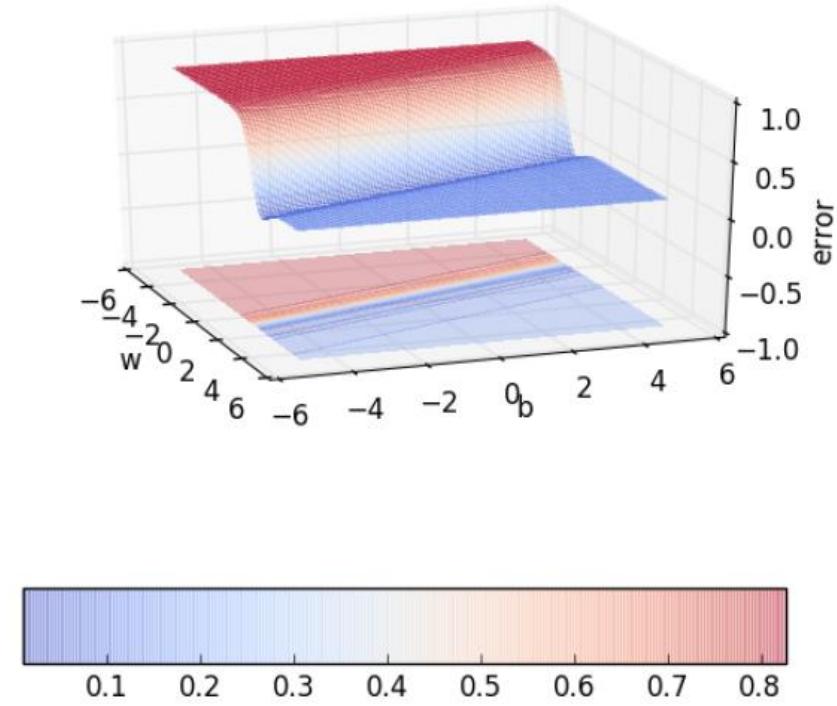
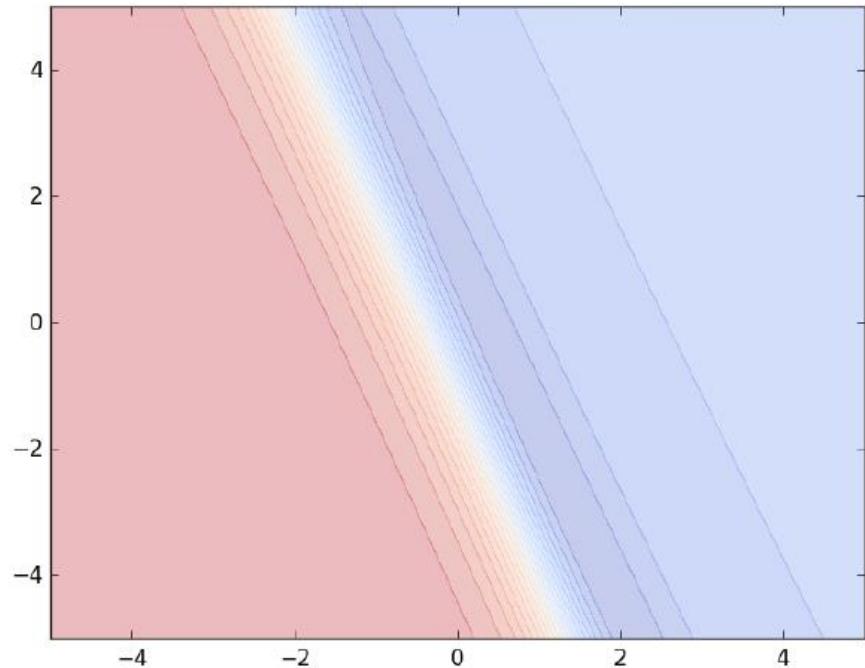
Contour Plot: a better way of visualizing error surface in 2D



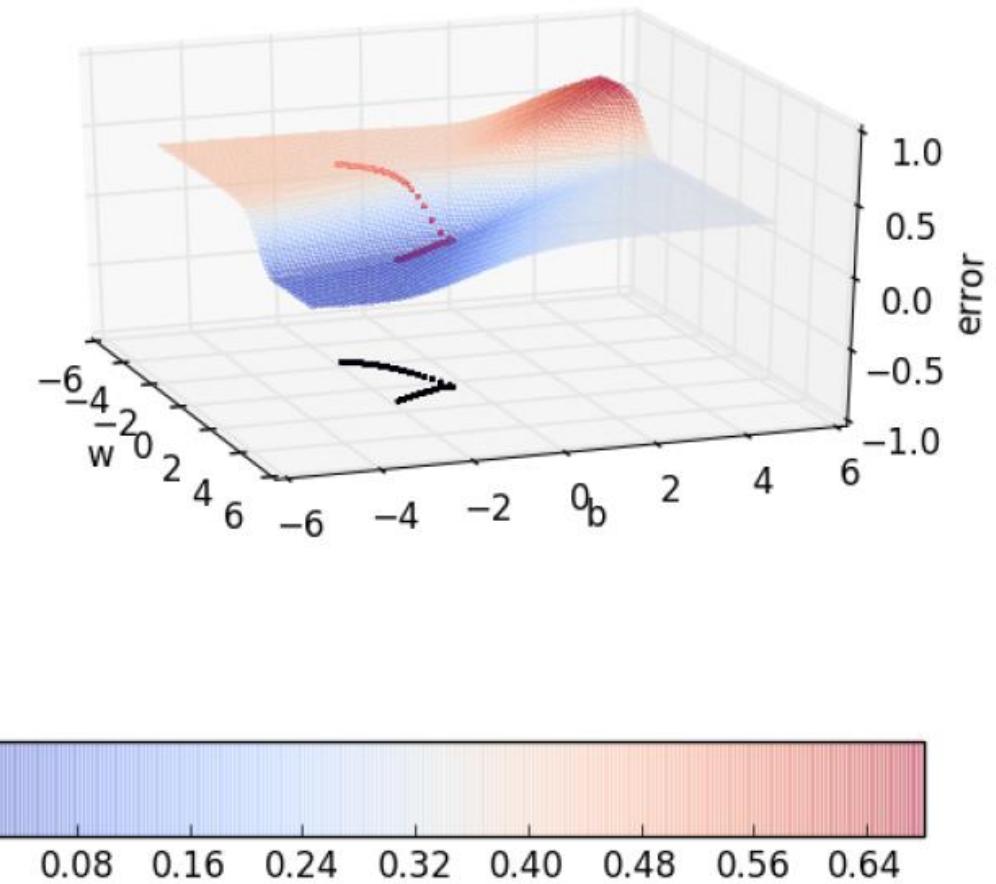
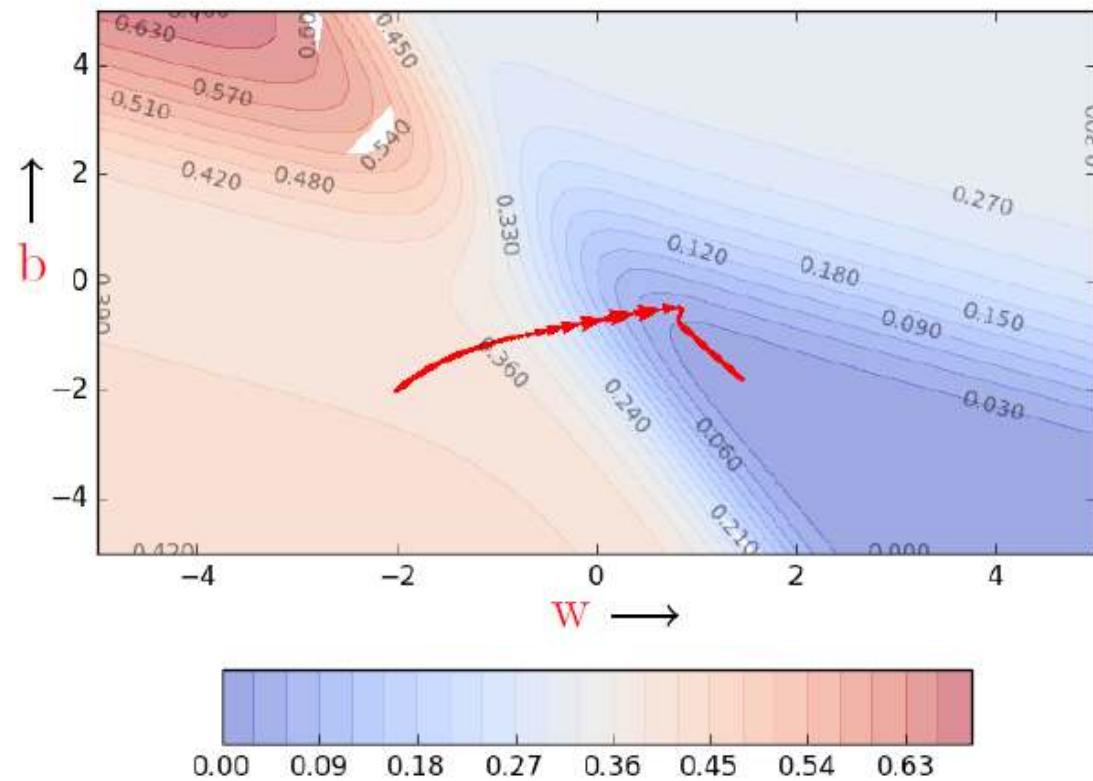
A small distance between the contours indicates a steep slope along that direction

A large distance between the contours indicates a gentle slope along that direction

Contour Plot vs 3D surface plot



Contour Plot vs 3D surface plot



Gradient descent: Problems

Plateaus and Flat Regions

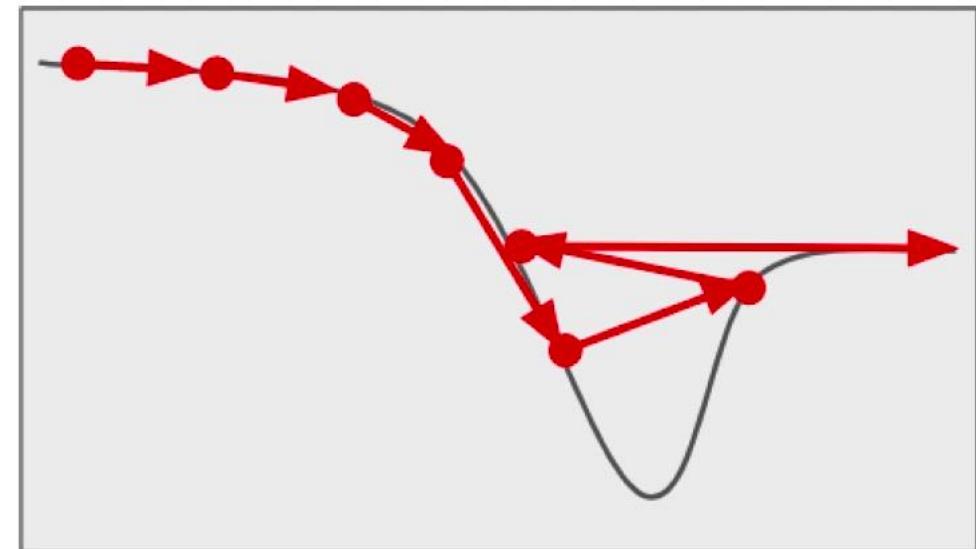
They constitute portions of error surface where gradient is highly non-spherical

Gradient descent spends a long time traversing in these regions as the updates are small

Can we expedite this process?

How about increasing the learning rate?

Though traversal becomes faster in plateaus, there is a **risk of divergence**



Gradient descent and its variants

Momentum-based Gradient Descent

Range : [0,1] (typically closer to 1)

Accumulated history of weight updates

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t$$
$$w_{t+1} = w_t - update_t$$

$$update_0 = 0$$

$$update_1 = \gamma \cdot update_0 + \eta \nabla w_1 = \eta \nabla w_1$$

$$update_2 = \gamma \cdot update_1 + \eta \nabla w_2 = \gamma \cdot \eta \nabla w_1 + \eta \nabla w_2$$

$$\begin{aligned} update_3 &= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 + \eta \nabla w_2) + \eta \nabla w_3 \\ &= \gamma \cdot update_2 + \eta \nabla w_3 = \gamma^2 \cdot \eta \nabla w_1 + \gamma \cdot \eta \nabla w_2 + \eta \nabla w_3 \end{aligned}$$

$$update_4 = \gamma \cdot update_3 + \eta \nabla w_4 = \gamma^3 \cdot \eta \nabla w_1 + \gamma^2 \cdot \eta \nabla w_2 + \gamma \cdot \eta \nabla w_3 + \eta \nabla w_4$$

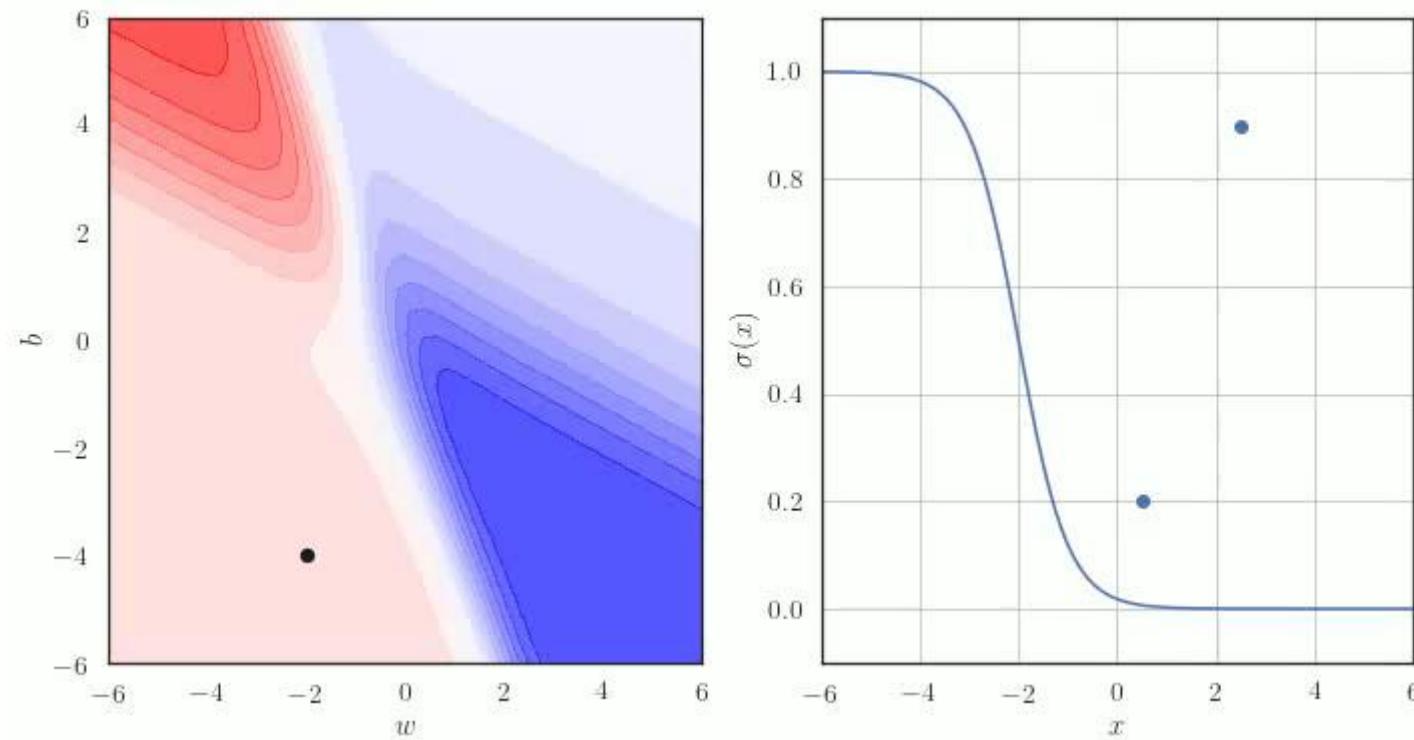
⋮

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_t = \gamma^{t-1} \cdot \eta \nabla w_1 + \gamma^{t-2} \cdot \eta \nabla w_1 + \dots + \eta \nabla w_t$$

Update = An Exponential weighted average of gradients (more weightage to recent updates and less weightage to old updates)

Gradient descent and its variants

Momentum-based Gradient Descent



Gradient descent and its variants

Momentum-based Gradient Descent

Momentum-based GD oscillates around minima before eventually reaching it

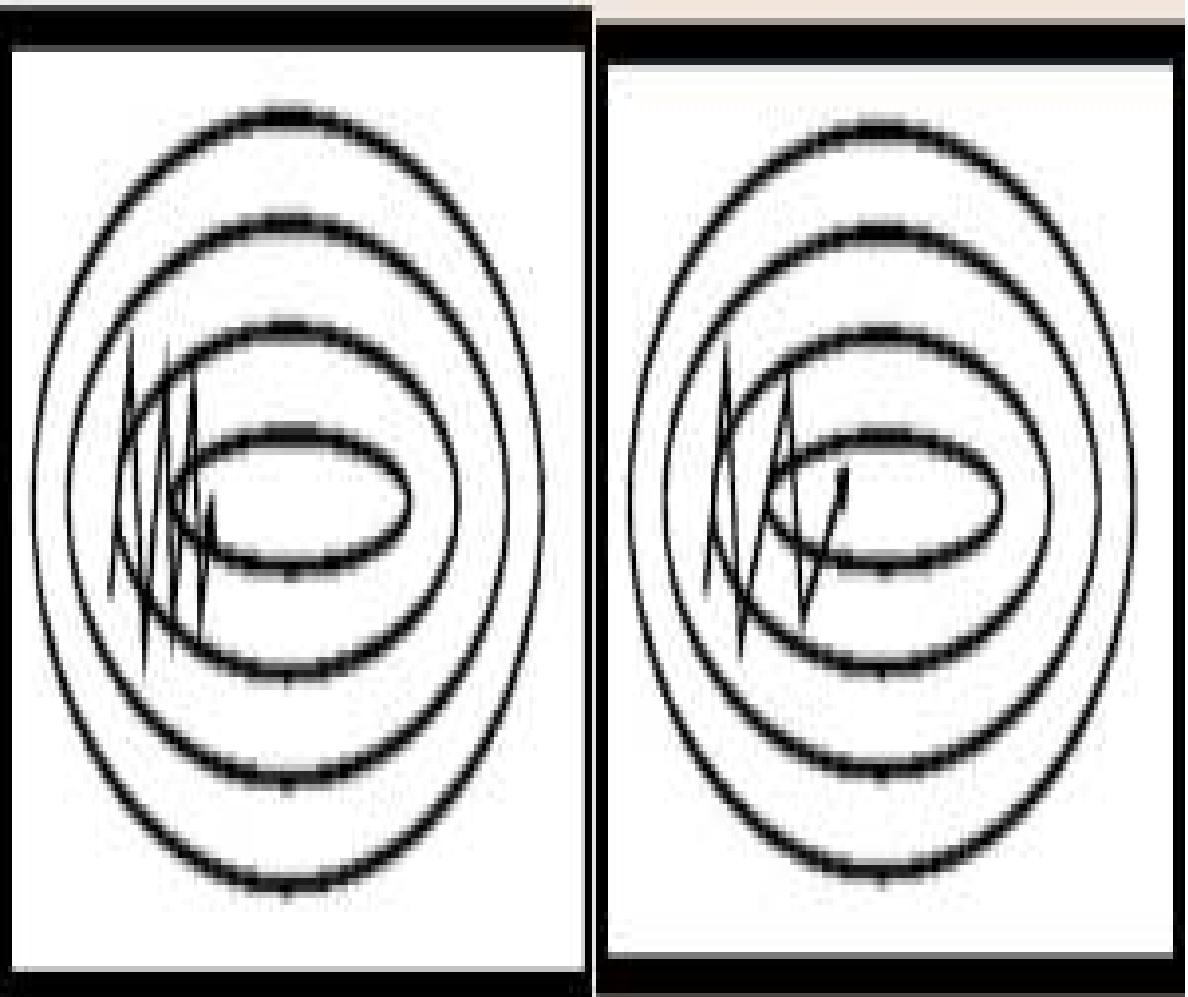
Even then, it converges faster than vanilla GD!

- After 100 iterations, momentum-based GD has error of 0.00001, whereas vanilla GD is still at error of 0.36

Nonetheless, it is wasting time in oscillations; how can we reduce oscillations?

Stochastic GD & SGD with Momentum DL

- stochastic means randomness on which the algorithm is based upon.
- Instead of taking the whole dataset for each iteration, randomly select the batches of data
- The path taken is full of noise as compared to the gradient descent algorithm.
- Uses a higher number of iterations to reach the local minima, thereby the overall computation time increases.
- The computation cost is still less than that of the gradient descent optimizer.
- If the data is enormous and computational time is an essential factor, SGD should be preferred over batch gradient descent algorithm.
- **Stochastic Gradient Descent with Momentum Deep Learning Optimizer**
 - momentum helps in faster convergence of the loss function.



SGD with Momentum Optimizer

- GD takes small , regular steps down the slope so algorithm takes more time to reach the bottom
- adding a fraction of the previous update to the current update will make the process a bit faster.
- Hyperparameter β - Momentum
 - To simulate a friction mechanism and prevent momentum from becoming too large
- Also rolls past local minima
- learning rate should be decreased with a high momentum term.

$$1. \quad \mathbf{m} \leftarrow \beta \mathbf{m} - \eta \nabla_{\theta} J(\theta)$$

$$2. \quad \theta \leftarrow \theta + \mathbf{m}$$

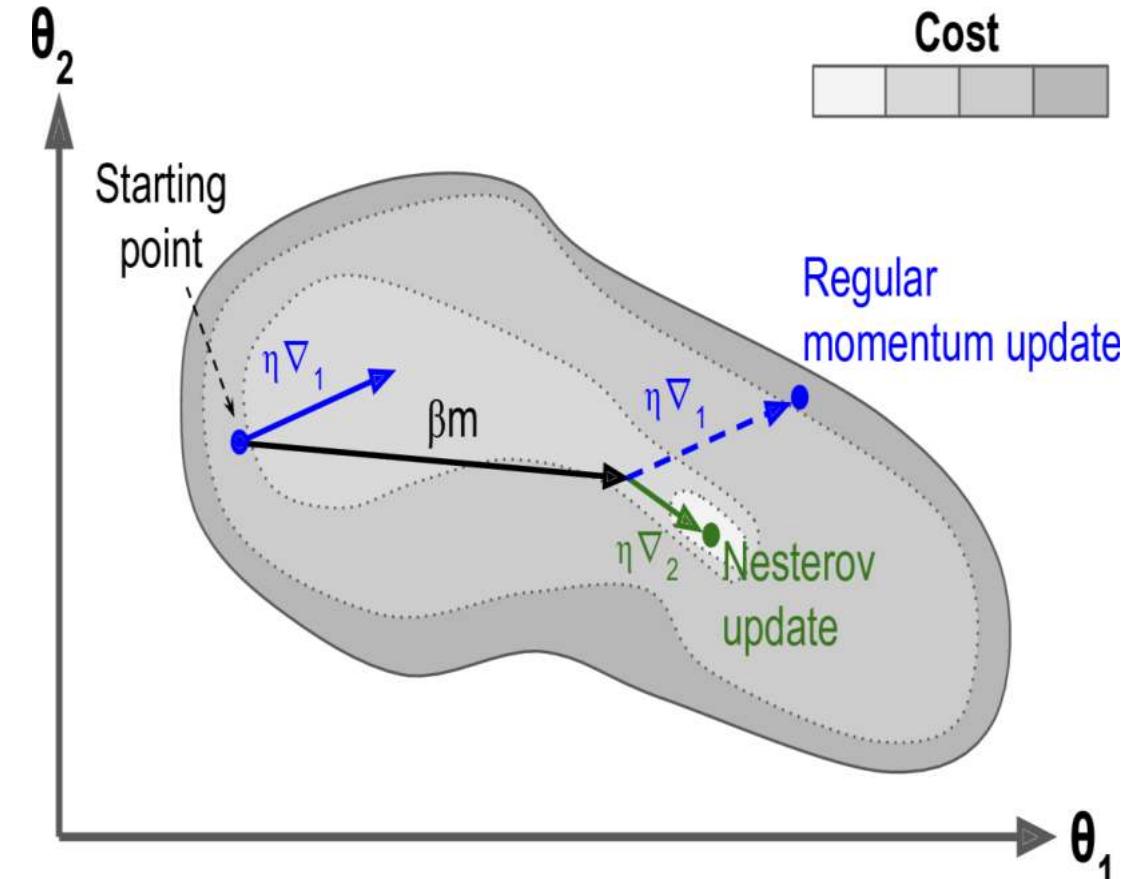
- β - Momentum
- set between 0 (high friction) and 1 (low friction)
- Typically 0.9

```
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9)
```

SGD with Nesterov Momentum Optimization

- Yurii Nesterov in 1983
- to measure the gradient of the cost function not at the local position but slightly ahead in the direction of the momentum
- the momentum vector will be pointing in the right direction (i.e., toward the optimum)
- it will be slightly more accurate to use the gradient measured a bit farther in that direction rather than using the gradient at the original position

1. $\mathbf{m} \leftarrow \beta\mathbf{m} - \eta\nabla_{\theta}J(\theta + \beta\mathbf{m})$
2. $\theta \leftarrow \theta + \mathbf{m}$



```
optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9, nesterov=True)
```

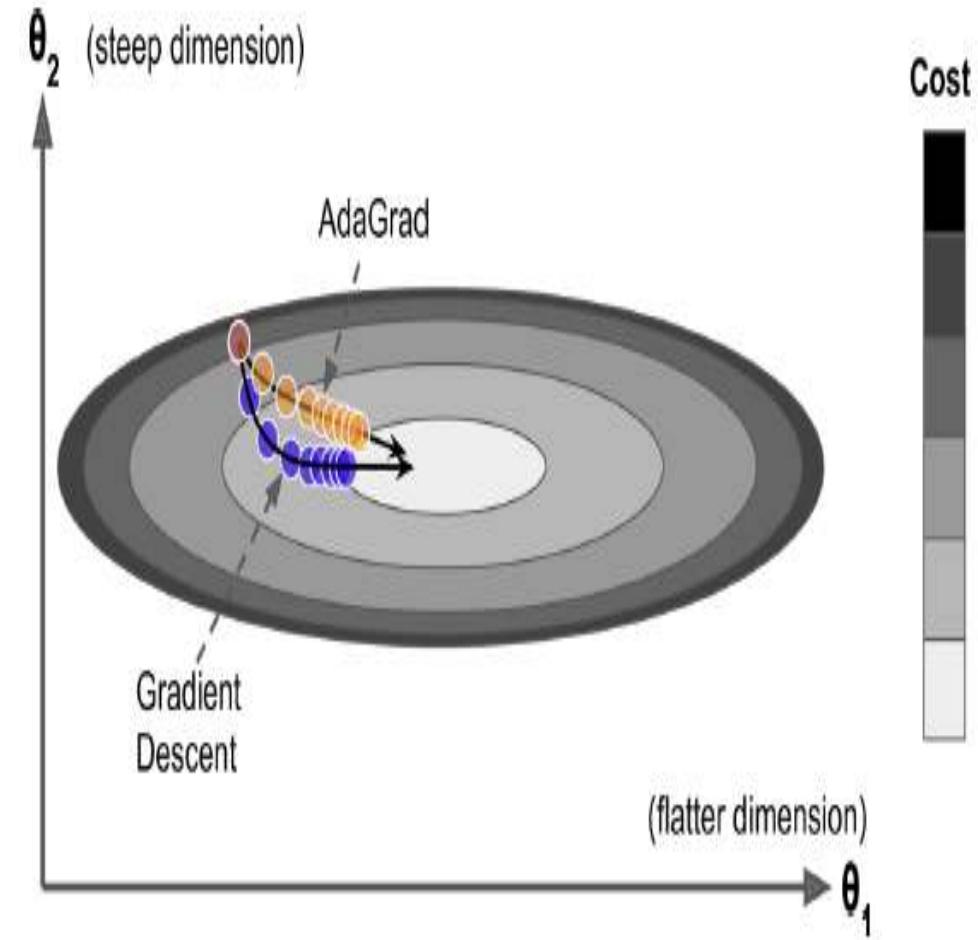
Adagrad (Adaptive Gradient Descent) Deep Learning Optimizer

- Adaptive Learning Rate
- Scaling down the gradient vector along the steepest dimension
- If the cost function is steep along the i th dimension, then s will get larger and larger at each iteration
- No need to modify the learning rate manually
- more reliable than gradient descent algorithms, and it reaches convergence at a higher speed.

$$s \leftarrow s + \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$$

$$\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta) \odot \sqrt{s + \epsilon}$$

- Disadvantage
 - it decreases the learning rate aggressively and monotonically.
 - Due to small learning rates, the model eventually becomes unable to acquire more knowledge, and hence the accuracy of the model is compromised.



RMS Prop(Root Mean Square) Deep Learning Optimizer

- The problem with the gradients some are small while others may be huge
- Defining a single learning rate might not be the best idea.
- accumulating only the gradients from the most recent iterations (as opposed to all the gradients since the beginning of training).

$$s \leftarrow \beta s + (1 - \beta) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta)$$

$$\theta \leftarrow \theta - \eta \nabla_{\theta} J(\theta) \oslash \sqrt{s + \epsilon}$$

`optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9)`

- Works better than Adagrad , was popular until ADAM

Adam Deep Learning Optimizer

- is derived from adaptive moment estimation.
- inherit the features of both Adagrad and RMS prop algorithms.
 - like Momentum optimization keeps track of an exponentially decaying average of past gradients,
 - like RMSProp it keeps track of an exponentially decaying average of past squared gradients
 - β_1 and β_2 represent the decay rate of the average of the gradients.
- Advantages
 - Is straightforward to implement
 - faster running time
 - low memory requirements, and requires less tuning
- Disadvantages
 - Focusses on faster computation time, whereas SGD focus on data points.
 - Therefore SGD generalize the data in a better manner at the cost of low computation speed.

$$\begin{aligned}\mathbf{m} &\leftarrow \beta_1 \mathbf{m} - (1 - \beta_1) \nabla_{\theta} J(\theta) \\ \mathbf{s} &\leftarrow \beta_2 \mathbf{s} + (1 - \beta_2) \nabla_{\theta} J(\theta) \otimes \nabla_{\theta} J(\theta) \\ \widehat{\mathbf{m}} &\leftarrow \frac{\mathbf{m}}{1 - \beta_1^t} \\ \widehat{\mathbf{s}} &\leftarrow \frac{\mathbf{s}}{1 - \beta_2^t} \\ \theta &\leftarrow \theta + \eta \widehat{\mathbf{m}} \oslash \sqrt{\widehat{\mathbf{s}}} + \epsilon\end{aligned}$$

t represents iteration

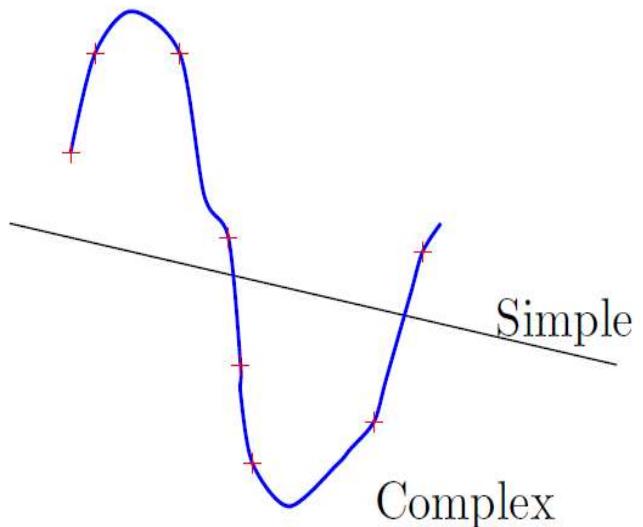
$\beta_1 = 0.9$

$\beta_2 = 0.999$

Smoothing term ϵ is $= 10^{-7}$

```
optimizer = keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999)
```

Curve Fitting – True Function is Sinusoidal



The points were drawn from a sinusoidal function (the true $f(x)$)

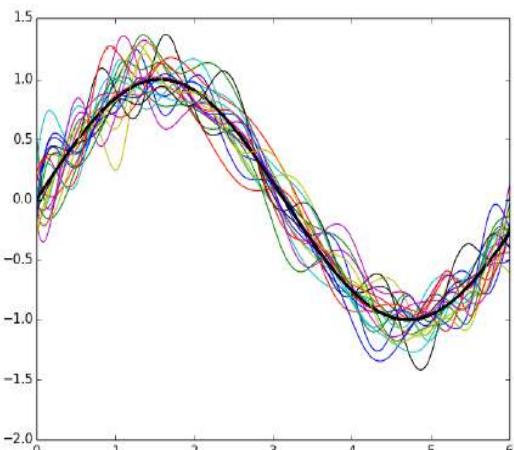
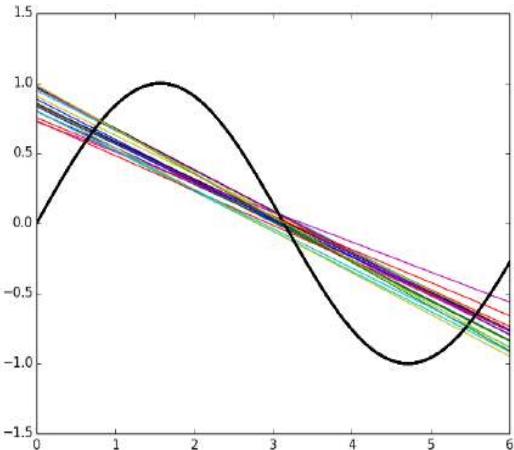
- Let us consider the problem of fitting a curve through a given set of points
- We consider two models :

$$\begin{array}{ll} \text{Simple} \\ (\text{degree:1}) & y = \hat{f}(x) = w_1 x + w_0 \end{array}$$

$$\begin{array}{ll} \text{Complex} \\ (\text{degree:25}) & y = \hat{f}(x) = \sum_{i=1}^{25} w_i x^i + w_0 \end{array}$$

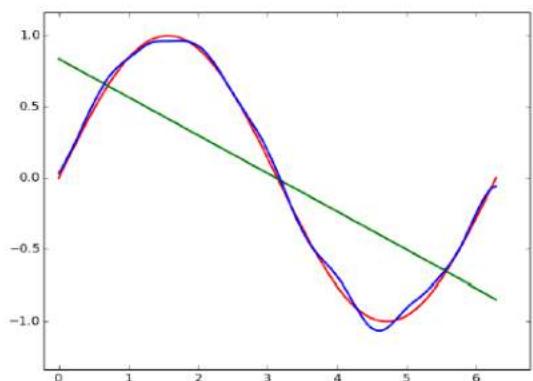
- Note that in both cases we are making an assumption about how y is related to x . We have no idea about the true relation $f(x)$

Curve Fitting – True Function is Sinusoidal



- Simple models trained on different samples of the data do not differ much from each other
- However they are very far from the true sinusoidal curve (under fitting)
- On the other hand, complex models trained on different samples of the data are very different from each other (high variance)

Bias



Green Line: Average value of $\hat{f}(x)$ for the simple model

Blue Curve: Average value of $\hat{f}(x)$ for the complex model

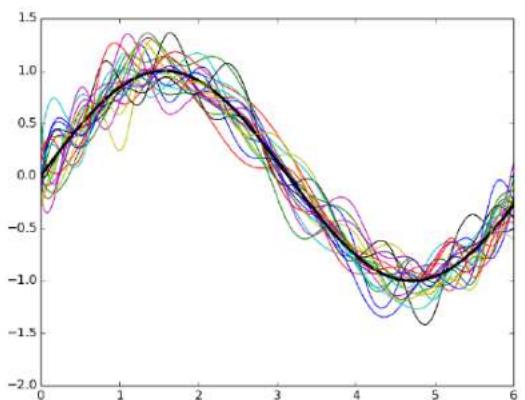
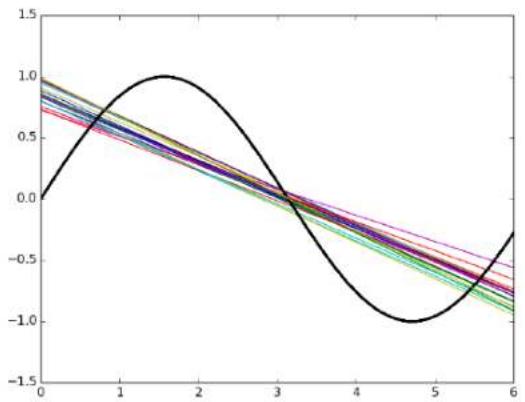
Red Curve: True model ($f(x)$)

- Let $f(x)$ be the true model (sinusoidal in this case) and $\hat{f}(x)$ be our estimate of the model (simple or complex, in this case) then,

$$\text{Bias } (\hat{f}(x)) = E[\hat{f}(x)] - f(x)$$

- $E[\hat{f}(x)]$ is the average (or expected) value of the model
- We can see that for the simple model the average value (green line) is very far from the true value $f(x)$ (sinusoidal function)
- Mathematically, this means that the simple model has a high bias
- On the other hand, the complex model has a low bias

Variance



- We now define,

Variance $(\hat{f}(x)) = E[(\hat{f}(x) - E[\hat{f}(x)])^2]$
(Standard definition from statistics)

- Roughly speaking it tells us how much the different $\hat{f}(x)$'s (trained on different samples of the data) differ from each other
- It is clear that the simple model has a low variance whereas the complex model has a high variance

Mean Square Error

- We can show that

$$\begin{aligned} E[(y - \hat{f}(x))^2] &= \text{Bias}^2 \\ &\quad + \text{Variance} \\ &\quad + \sigma^2 \text{ (irreducible error)} \end{aligned}$$

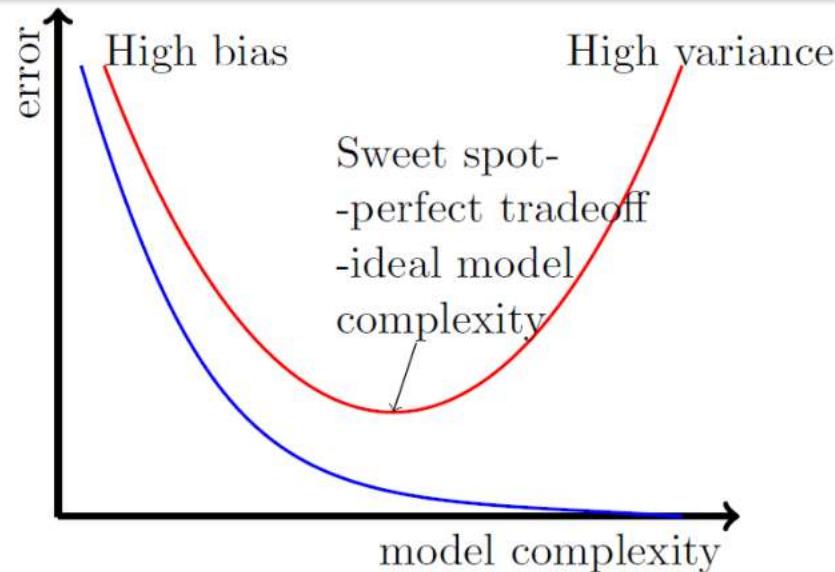
- Consider a new point (x, y) which was not seen during training

- If we use the model $\hat{f}(x)$ to predict the value of y then the mean square error is given by

$$E[(y - \hat{f}(x))^2]$$

(average square error in predicting y for many such unseen points)

Train vs Test Error



- The parameters of $\hat{f}(x)$ (all w_i 's) are trained using a training set $\{(x_i, y_i)\}_{i=1}^n$
- However, at test time we are interested in evaluating the model on a validation (unseen) set which was not used for training
- This gives rise to the following two entities of interest:
 $train_{err}$ (say, mean square error)
 $test_{err}$ (say, mean square error)
- Typically these errors exhibit the trend shown in the adjacent figure

Train vs Test Error

- Let there be n training points and m test (validation) points

$$train_{err} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}(x_i))^2$$

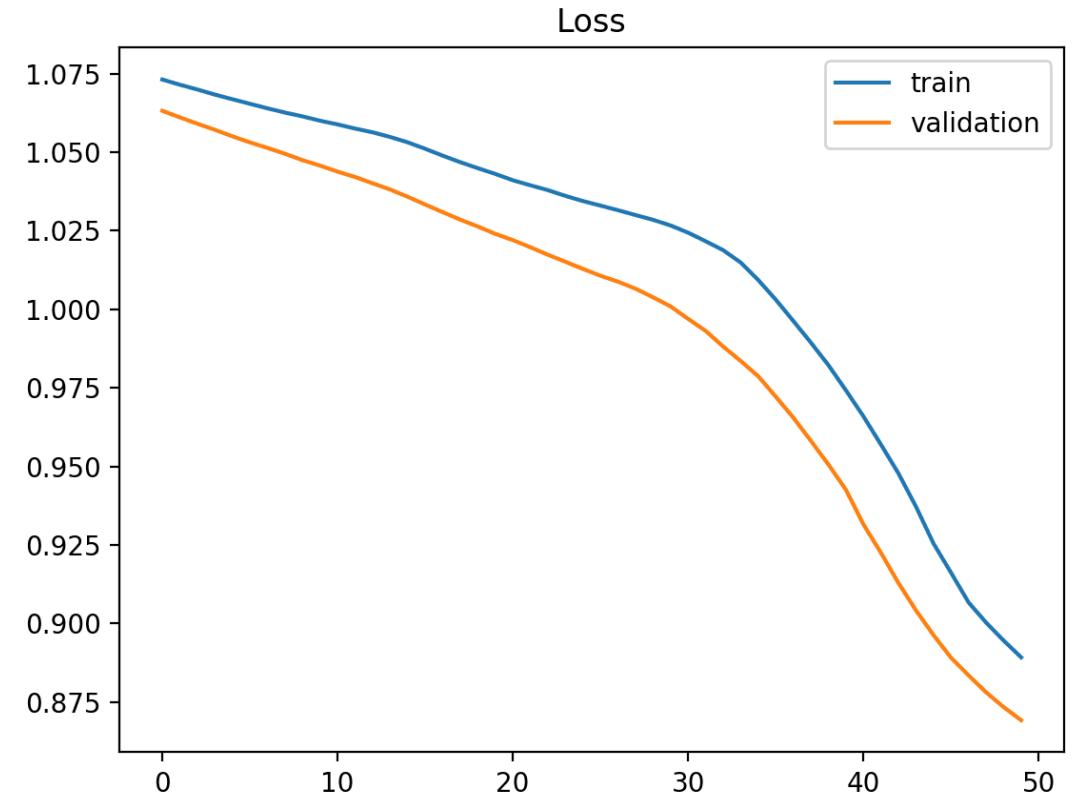
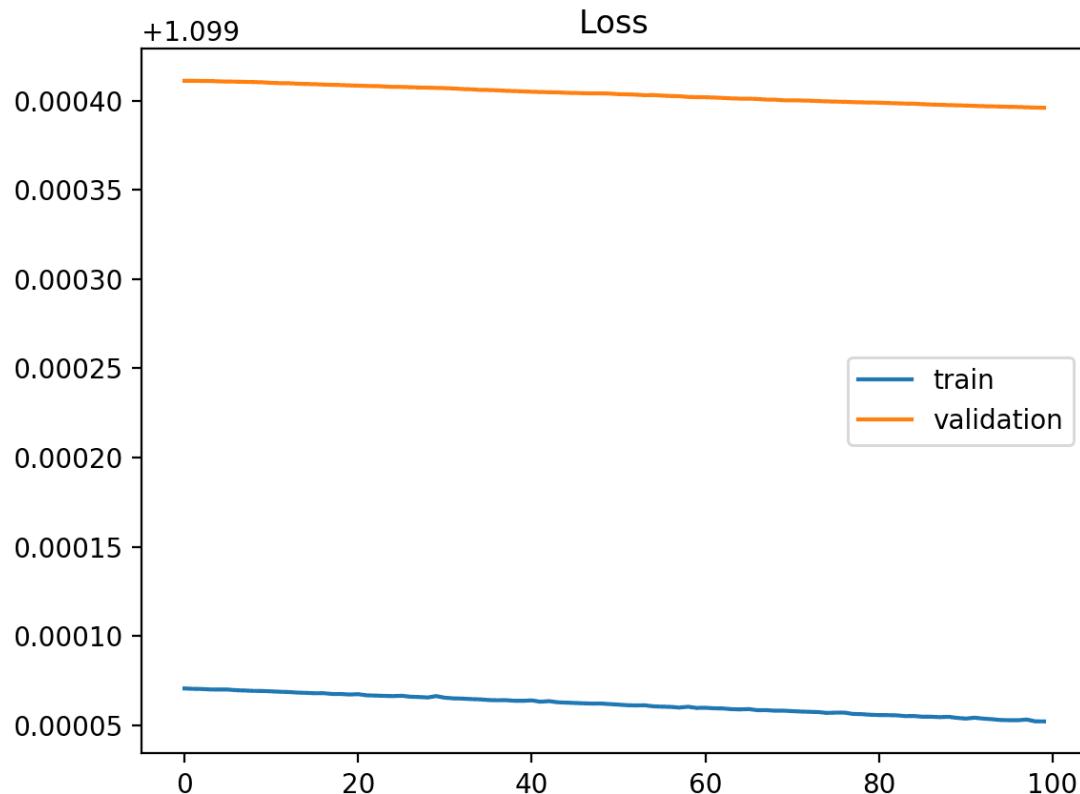
$$test_{err} = \frac{1}{m} \sum_{i=n+1}^{n+m} (y_i - \hat{f}(x_i))^2$$

- As the model complexity increases $train_{err}$ becomes overly optimistic and gives us a wrong picture of how close \hat{f} is to f
- The validation error gives the real picture of how close \hat{f} is to f

Learning Curves

- Line plot of learning (y-axis) over experience (x-axis)
- The metric used to evaluate learning could be
- **Optimization Learning Curves:**
 - calculated on the metric by which the parameters of the model are being optimized, e.g. loss.
 - Minimizing, such as loss or error
- **Performance Learning Curves:**
 - calculated on the metric by which the model will be evaluated and selected, e.g. accuracy.
 - Maximizing metric , such as classification accuracy
- **Train Learning Curve:**
 - calculated from the training dataset that gives an idea of how well the model is learning.
- **Validation Learning Curve:**
 - calculated from a hold-out validation dataset that gives an idea of how well the model is generalizing.

Underfit Learning Curves

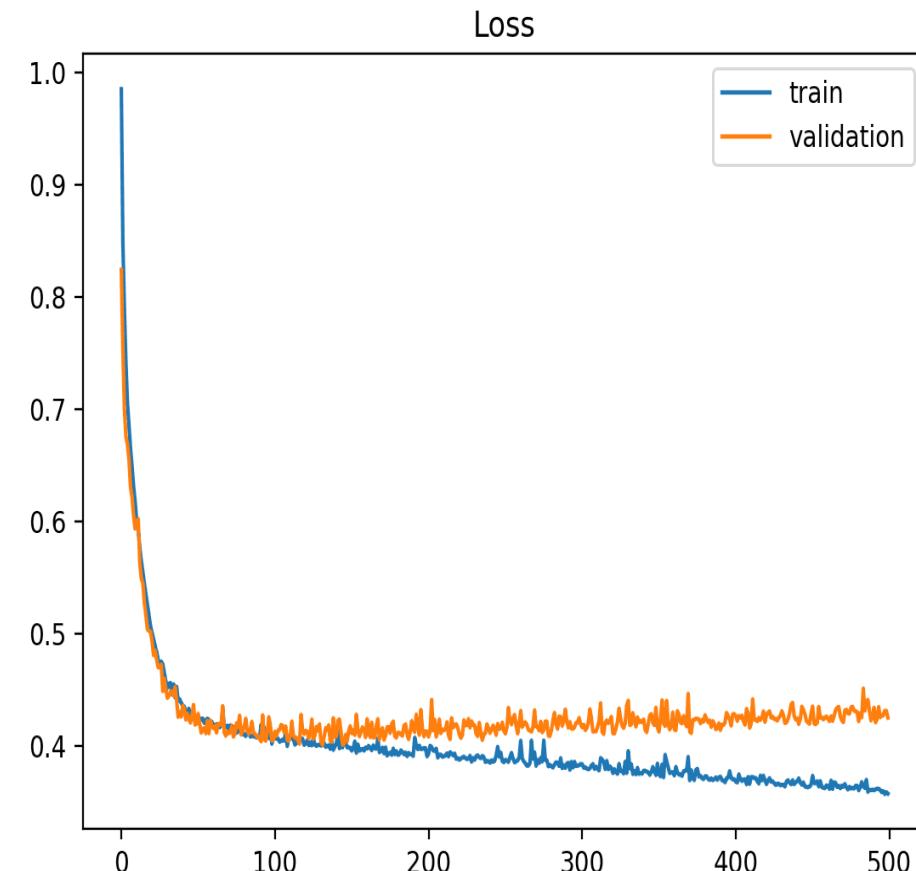


A plot of learning curves shows underfitting if:

- The training loss remains flat regardless of training.
- The training loss continues to decrease until the end of training.

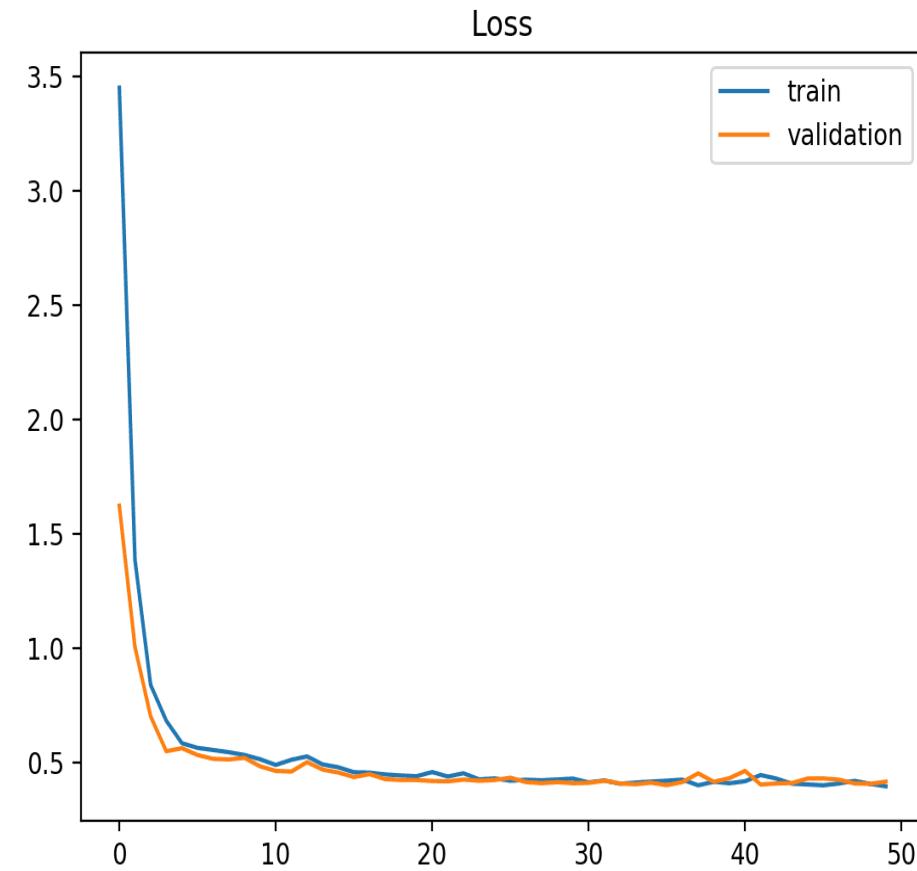
Overfitting Curves

- Overfitting
 - Model specialized on training data, it is not able to generalize to new data
 - Results in increase in generalization error.
 - generalization error can be measured by the performance of the model on the validation dataset.
- A plot of learning curves shows overfitting if:
 - The plot of training loss continues to decrease with experience.
 - The plot of validation loss decreases to a point and begins increasing again.
 - The inflection point in validation loss may be the point at which training could be halted as experience after that point shows the dynamics of overfitting.



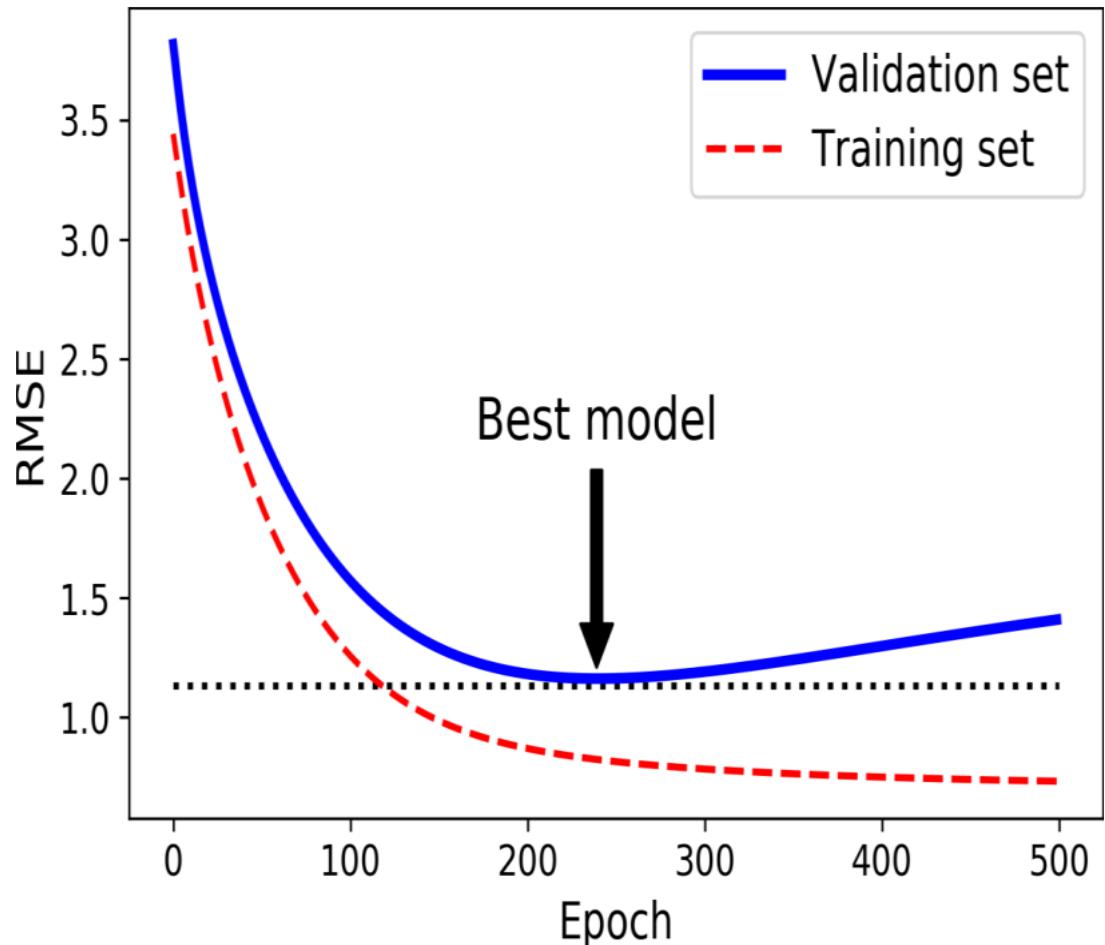
Good Fit Learning Curves

- A good fit is the goal of the learning algorithm and exists between an overfit and underfit model.
- A plot of learning curves shows a good fit if:
 - Plot of training loss decreases to a point of stability
 - Plot of validation loss decreases to a point of stability and has a small gap with the training loss.
- Loss of the model will almost always be lower on the training than the validation dataset.
- We should expect some gap between the train and validation loss learning curves.
- This gap is referred to as the “generalization gap.”



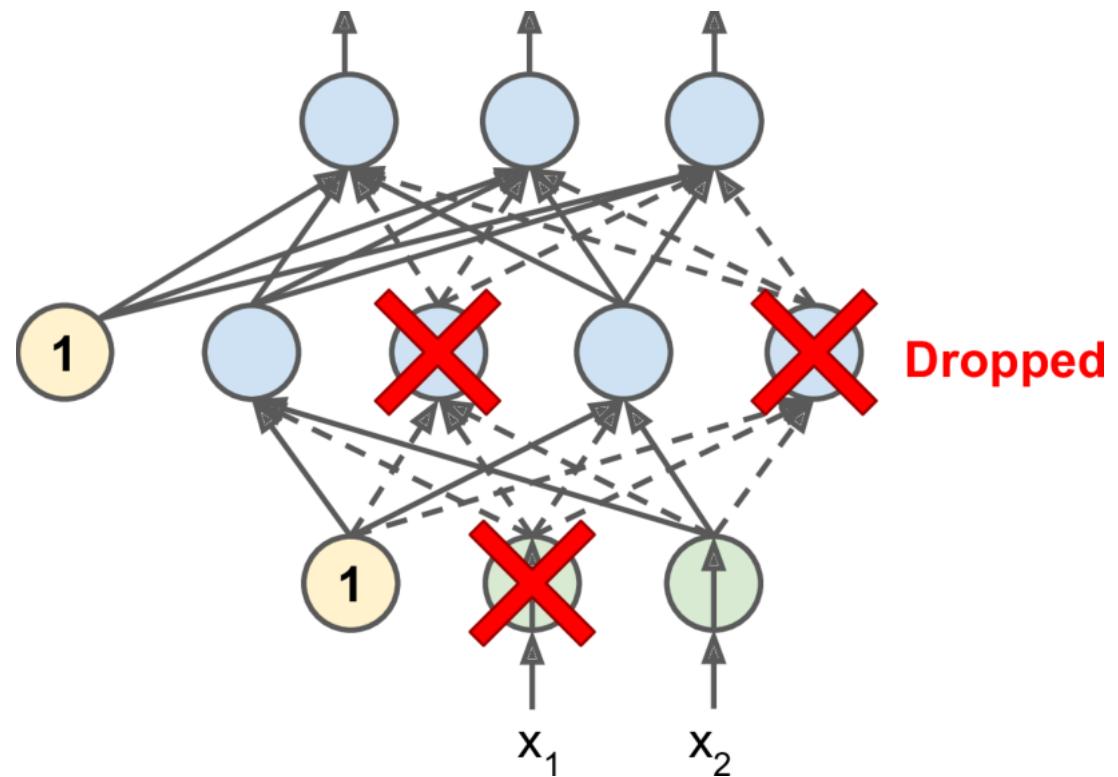
Avoiding Overfitting

- With so many parameters, DNN can fit complex datasets.
- But also prone to overfitting the training set.
- Early Stopping
 - stop training as soon as the validation error reaches a minimum
 - With Stochastic and Mini-batch Gradient Descent, the curves are not so smooth, and it may be hard to know whether you have reached the minimum or not.
 - Stop only after the validation error has been above the minimum for some time



Avoiding overfitting

- Dropout
 - Proposed by Geoffrey Hinton in 2012
 - at every training step, every neuron has a probability p of being temporarily “dropped out,”
 - it will be entirely ignored during this training step, but it may be active during the next step
 - p is called the dropout rate, usually 50%.
 - After training, neurons don’t get dropped
 - If $p = 50\%$
 - during testing a neuron will be connected to twice as many input neurons as it was (on average) during training.
 - multiply each input connection weight by the keep probability ($1 - p$) after training
 - Alternatively, divide each neuron’s output by the keep probability during training



Overfitting using Regularization

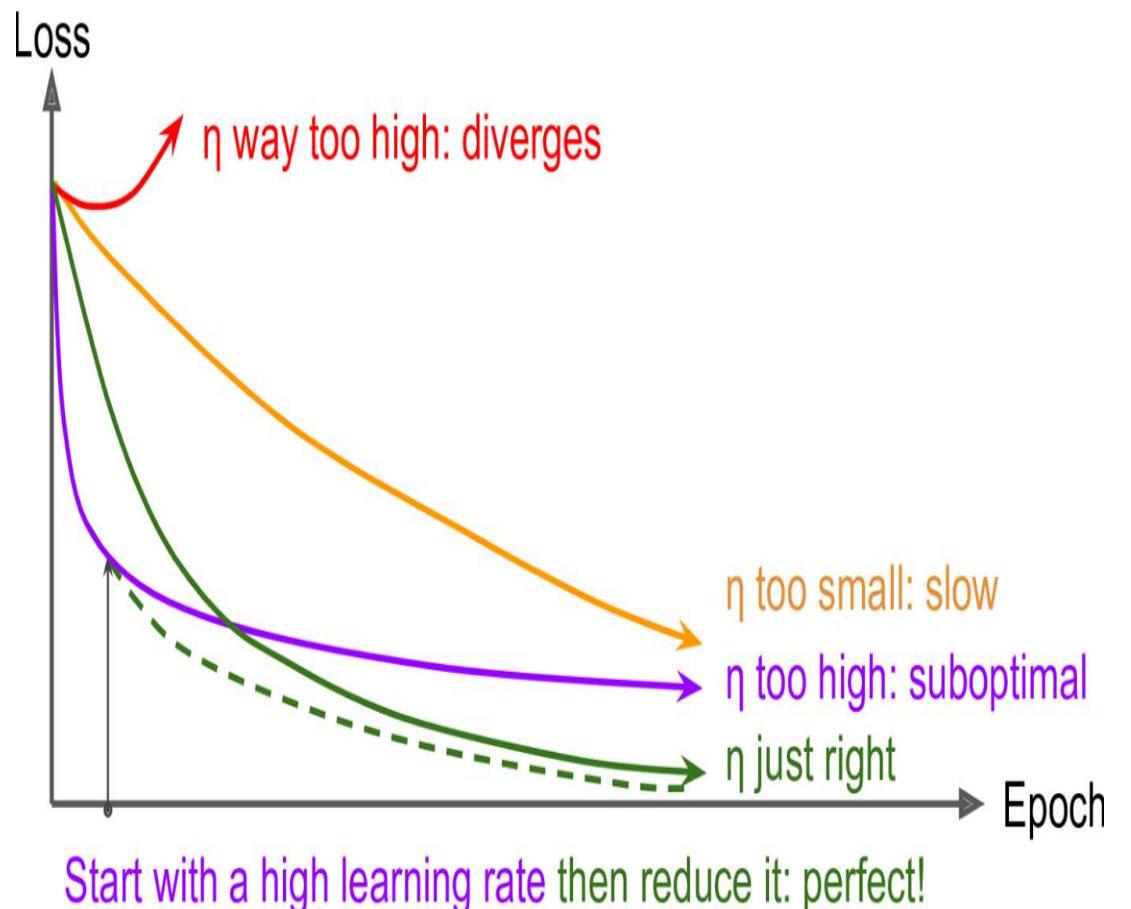
- L1, l2 regularization
 - Regularization can be used to constrain the NN weights
 - Lasso – (l1) least absolute shrinkage and selection operator
 - adds the “absolute value of magnitude” of the coefficient as a penalty term to the loss function.
 - Ridge regression (l2)
 - adds the “squared magnitude” of the coefficient as the penalty term to the loss function.
 - Use l1(), l2(), l1_l2() function
 - which returns a regularizer that will compute the regularization loss at each step during training
 - Regularization loss is added to final loss
- Max-Norm Normalization
 - It **constrains** the weights w of the incoming connections

$$\cdot (\mathbf{w} \leftarrow \mathbf{w} \frac{r}{\|\mathbf{w}\|_2}).$$

- r is max-norm hyper parameter and $\|\cdot\|_2$ is the ℓ_2 norm
- Reducing r increases the amount of regularization and helps reduce overfitting
- Can also help alleviate the unstable gradients problem if we are not using Batch normalization

Constant learning rate not ideal

- Better to start with a high learning rate
- then reduce it once it stops making fast progress
- can reach a good solution faster
- Learning Schedule strategies can be applied
 - Power Scheduling
 - Exponential Scheduling
 - Piecewise Constant Scheduling
 - Performance Scheduling



Learning Rate Scheduling

- *Power scheduling*
 - Learning rate set to a function of the iteration number ‘t’
 - $t: \eta(t) = \eta_0 / (1 + t/k)^c$
 - The initial learning rate η_0 , the power c (typically set to 1)
 - The learning rate drops at each step, and after s steps it is down to $\eta_0 / 2$ and so on
 - schedule first drops quickly, then more and more slowly
 - `optimizer = keras.optimizers.SGD(lr=0.01, decay=1e-4)`
 - *The decay is the number of steps it takes to divide the learning rate by one more unit,*
 - *Keras assumes that c is equal to 1.*
- *Exponential scheduling*
 - Set the learning rate to: $\eta(t) = \eta_0 0.1^{t/s}$
 - learning rate will gradually drop by a factor of 10 every s steps.

Learning Rate Scheduling

- *Piecewise constant scheduling*
 - Constant learning rate for a number of epochs
 - e.g., $\eta_0 = 0.1$ for 5 epochs
 - then a smaller learning rate for another number of epochs
 - e.g., $\eta_1 = 0.001$ for 50 epochs and so on
- *Performance scheduling*
 - Measure the validation error every N steps (just like for early stopping)
 - reduce the learning rate by a factor of λ when the error stops dropping

How about optimizing GD by varying learning rate ?

- If learning rate is too small, it takes long time to converge.
If learning rate is too large, the gradients explode.

Some techniques for choosing learning rate:

- **Linear Search**

Tune learning rate [Try different values on a log scale: 0.0001, 0.001, 0.01, 0.1. 1.0]

Run a few epochs with each of these and figure out a learning rate which works best

Now do a finer search around this value [for example, if the best learning rate was 0.1 then now try some values around it: 0.05, 0.2, 0.3]

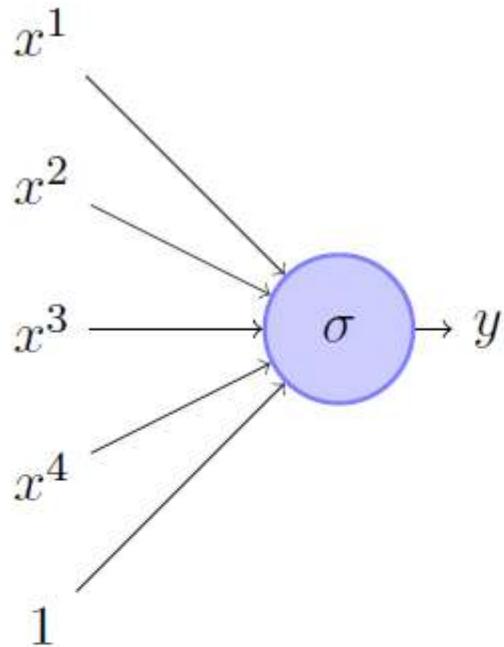
GD with adaptive learning rate

- **Motivation:** Can we have a different learning rate for each parameter which takes care of the frequency of features ?
- **Intuition:** Decay the learning rate for parameters in proportion to their update history.
 - For sparse features, accumulated update history is small
 - For dense features, accumulated update history is large

Make learning rate inversely proportional to the update history i.e, if the feature has been updated fewer times, give it a larger learning rate and vice versa

Let's consider an example

GD with adaptive learning rate



$$y = f(x) = \frac{1}{1+e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

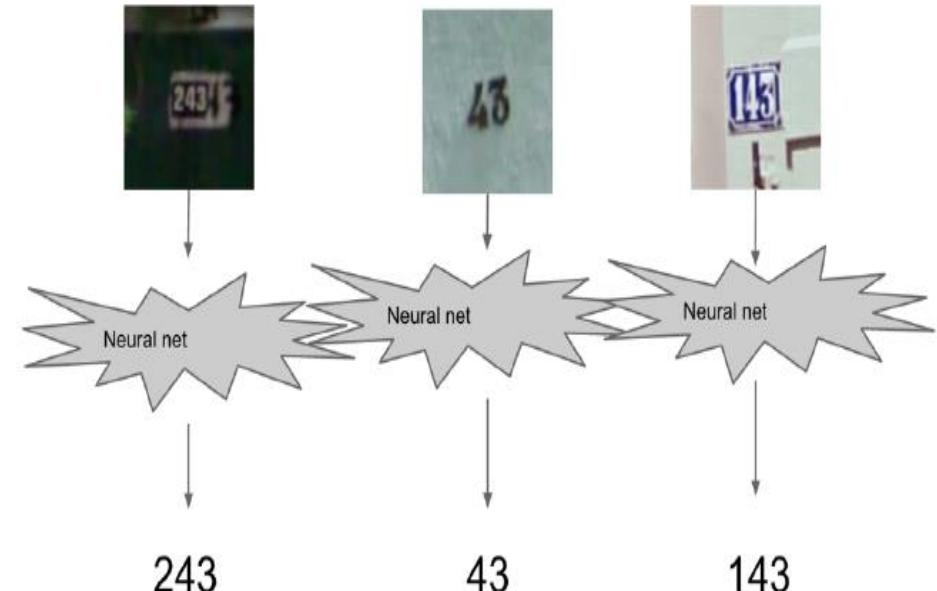
$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point (\mathbf{x}, y) ...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2 \dots$ so on
- If there are n points, we can just sum the gradients over all the n points to get the total gradient
- What happens if the feature x^2 is very sparse? (*i.e.*, if its value is 0 for most inputs)
- ∇w^2 will be 0 for most inputs (see formula) and hence w^2 will not get enough updates
- If x^2 happens to be sparse as well as important we would want to take the updates to w^2 more seriously
- Can we have a different learning rate for each parameter which takes care of the frequency of features ?

Practical Methodology

- ML practitioner needs to know
 - how to choose an algorithm for a particular application
 - how to monitor and respond to feedback obtained from experiments
 - in order to improve a machine learning system

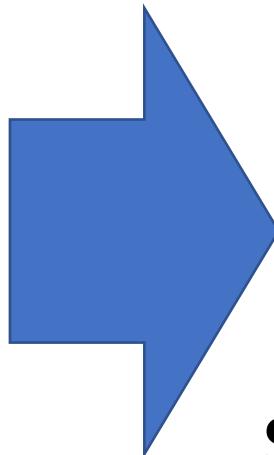
Example: Street View Address
Number Transcription



(Goodfellow et al, 2014)

Practical Design Process

- **PHASE 1**
 - ✓ Define the problem statement.
 - ✓ Meta data of Data set
- **In real world problems**
 - If training data is limited, determine the value of reducing error against the cost of collecting more data
- **In scientific problems**
 - Benchmarked data sets to be used
- ✓ Exploratory Analysis
- ✓ Preprocessing pipeline specific to data
- ✓ Define Project Objectives



Step 1- Define Project Objectives

- Deep or not?
- Use needs to define metric based goals

Step 1 - Problem Statement – Deep or Not?

- Lots of noise, little structure -> not deep
- Little noise, complex structure -> deep
- Good shallow baseline:
 - *Use what you know*
 - Logistic regression, SVM, boosted tree are all good

Step – 1 Performance Metrics

- Performance Metrics
 - Bayes error defines the minimum error rate , even if there is infinite training data and can recover the true probability distribution
- For example
 - Precision is the fraction of detections reported by the model that were correct
 - Recall is the fraction of true events that were detected.
 - Visualization such as PR curve
- ML model can quantify how confident it should be about a decision
 - if a wrong decision can be harmful , a human operator can occasionally take over
- Coverage
 - is the fraction of examples for which the ML system is able to produce a response
- For example
 - Street View task, the goal for the project was to reach human-level transcription accuracy while maintaining 95% coverage.
 - Human-level performance on this task is 98% accuracy.

Practical Design Process

PHASE 2 .1

- ✓ Literature review to identify models to be implemented
- ✓ Pros, Cons of each model
- ✓ Shortlist models for implementation
- ✓ Define baseline model



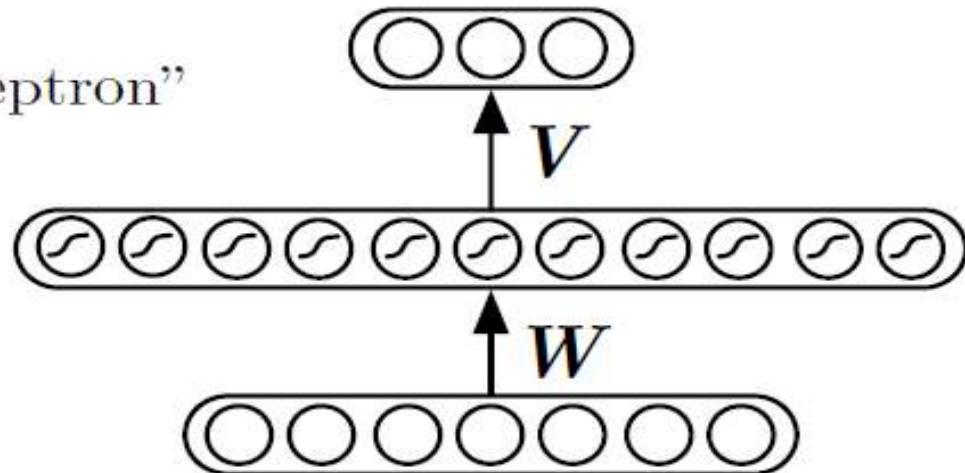
Can begin with simple statistical model like

- logistic regression
 - Decision Tree
 - Shallow Networks
 - SVM
-
- No structure -> fully connected
 - Spatial structure -> convolutional
 - Sequential structure -> recurrent

Step 2 - Default Baseline Models

Fully Connected Baseline

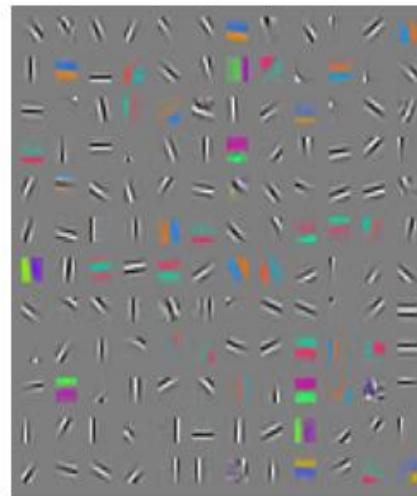
- 2-3 hidden layer feed-forward neural network
 - AKA “multilayer perceptron”
 - Rectified linear units
 - Batch normalization
 - Adam
 - Maybe dropout



Step 2 - Default Baseline Models

Convolutional Network Baseline

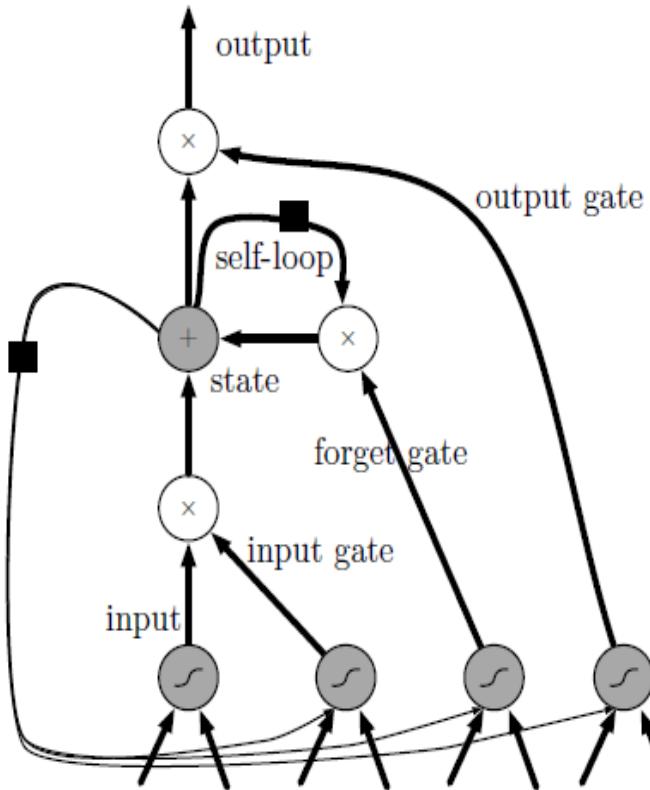
- Download a pretrained network
- Or copy-paste an architecture from a related task
 - Or:
 - Deep residual network
 - Batch normalization
 - Adam



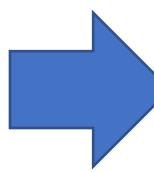
Step 2 - Default Baseline Models

Recurrent Network Baseline

- LSTM
- SGD
- Gradient clipping
- High forget gate bias

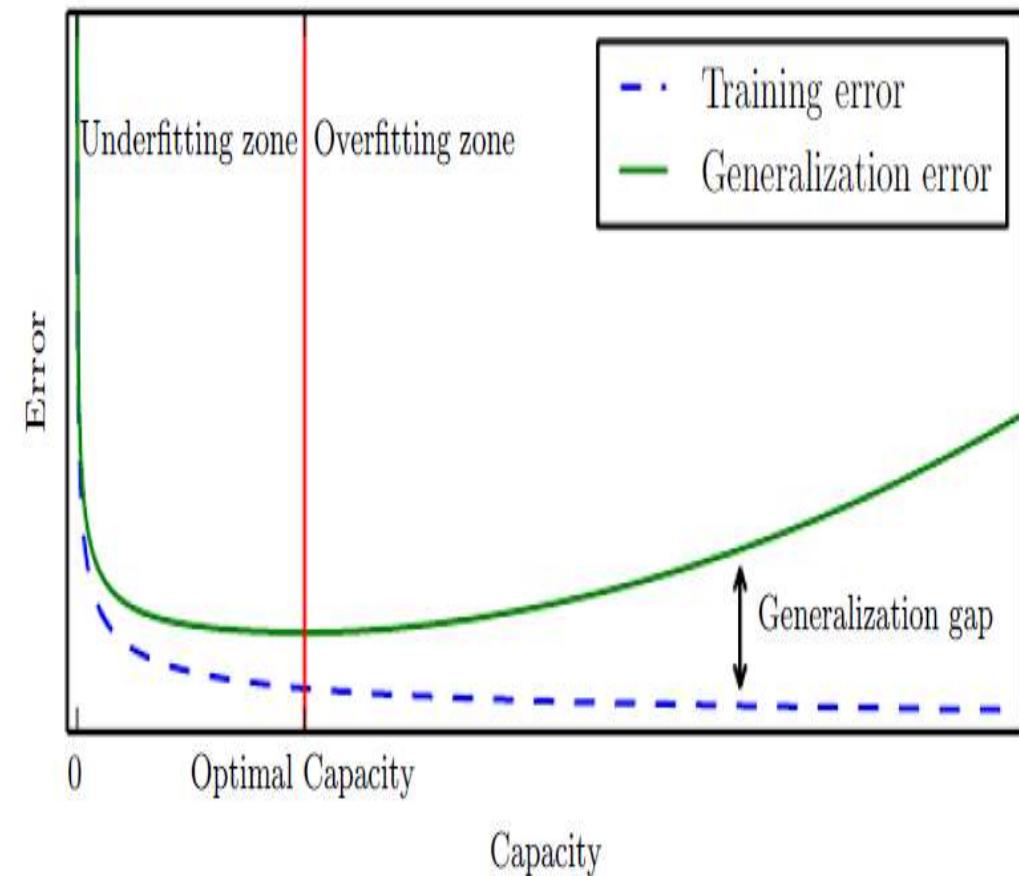


Practical Design Process

- **PHASE 2.2**
 - ✓ Define working end-to-end pipeline
 - ✓ Determine your goals—what error metric to use, and target value.
 - ✓ Diagnose performance and optimization curves
 - Measure train and test error
 - Overfitting vs Underfitting
- 
- Based on findings whether to gather new data?
 - adjust hyperparameters?
(learning rate, number of layers etc)
 - Hyperparameter tuning can be Manual or Automatic
 - or change architecture?

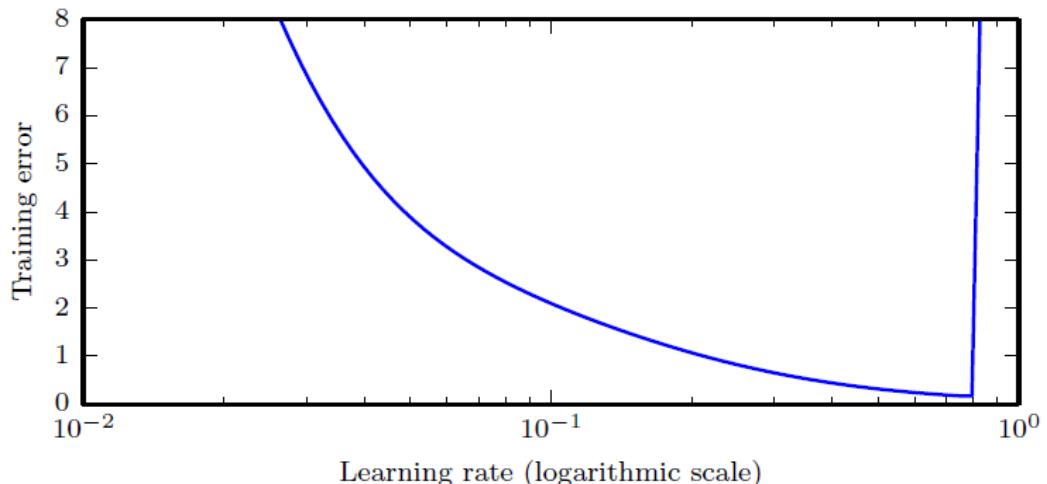
Step 3- Data Driven Refinement

- Manual Hyperparameter Tuning
 - Need to understand the relationship between hyperparameters, training error, generalization error and computational resources (memory & runtime)
- Goal of manual Hyperparameter tuning
 - to find the lowest generalization error subject to some runtime and memory budget
 - to adjust the effective capacity of the model to match the complexity of the task
 - Effective capacity is constrained by
 - ✓ the representational capacity of the model
 - ✓ ability of the learning algorithm to successfully minimize the cost function used to train the model
 - ✓ the degree to which the cost function and training procedure regularize the model.



Practical Design Process

- PHASE 3
 - Deployment in app/cloud
 - Tabulation and visualization of results in terms of performance and accuracy , roc /prc etc
 - Result analysis
 - Comment on accuracy, performance
 - Reasoning about hyperparameters
 - Conclusion



Hyperparameter	Increases capacity when...	Reason	Caveats
Number of hidden units	increased	Increasing the number of hidden units increases the representational capacity of the model.	Increasing the number of hidden units increases both the time and memory cost of essentially every operation on the model.

References

- Ian Goodfellow, Yoshua Bengio and Aaron Courville, “Deep Learning”, MIT Press 2016
- NPTEL Notes from CS6910 Deep Learning , Mitesh Khapra,
- Coursera Notes from Neural Networks and Deep Learning , Andrew NG
- Aurelien Geron, “Hands-On Machine Learning with Scikit-Learn , Keras & Tensorflow, OReilly Publications
- Jiawei Han and Micheline Kamber, “Data Mining Concepts And Techniques”, 3rd Edition, Morgan Kauffmann

Generative Adversarial Networks

DSE 3151 DEEP LEARNING

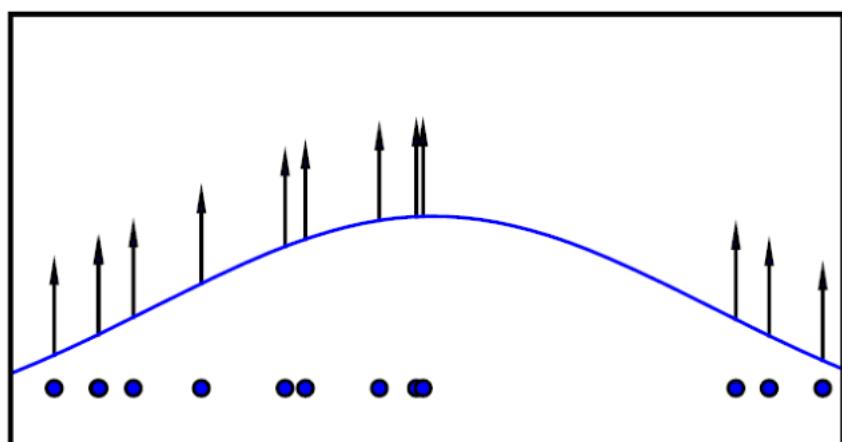
Rohini Rao & Abhilash K Pai

Department of Data Science and Computer Applications

MIT Manipal

Generative Models : Overview

Maximum Likelihood Estimation

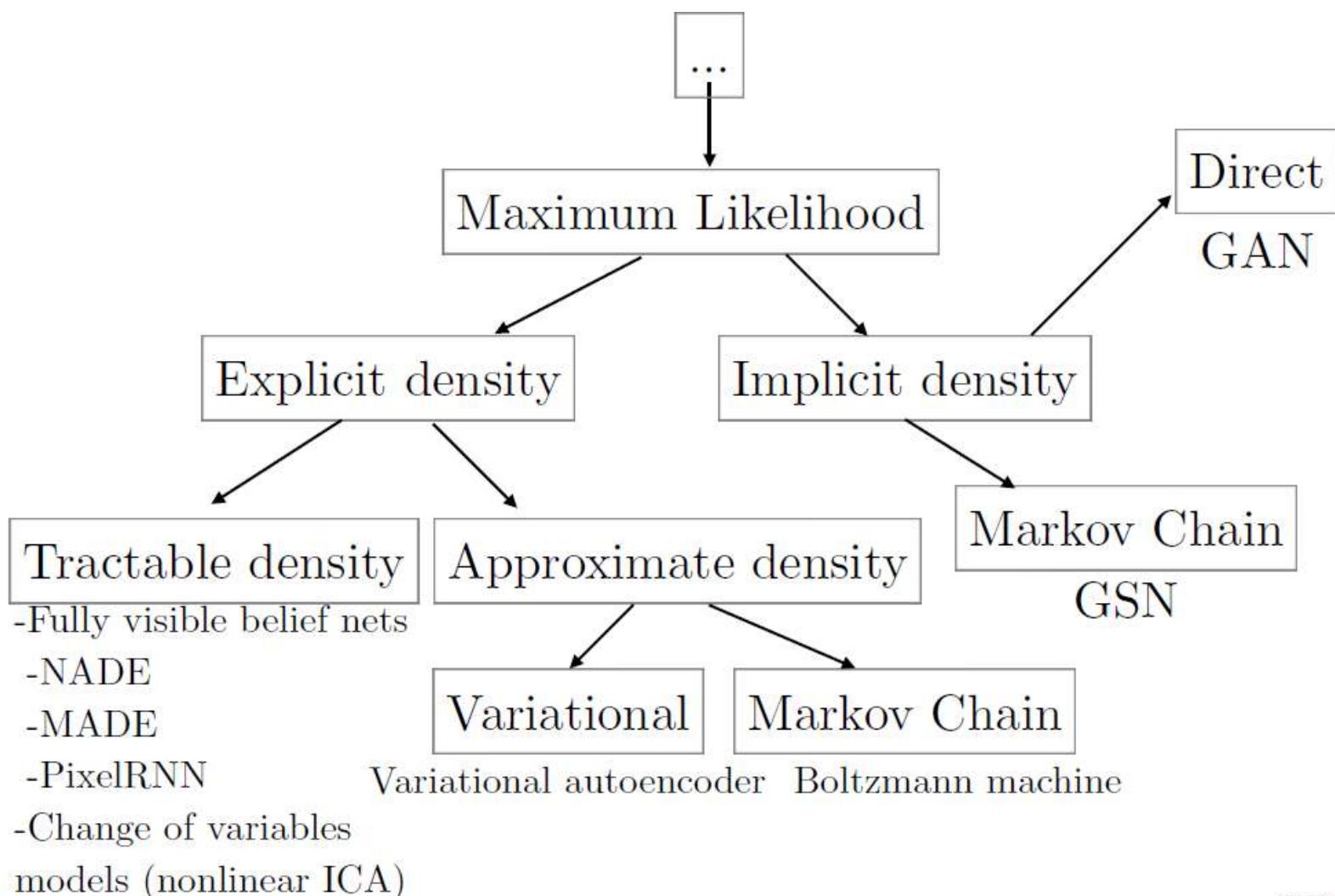


$$\theta^* = \arg \max_{\theta} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(x | \theta)$$

Ian Goodfellow, NeurIPS'16

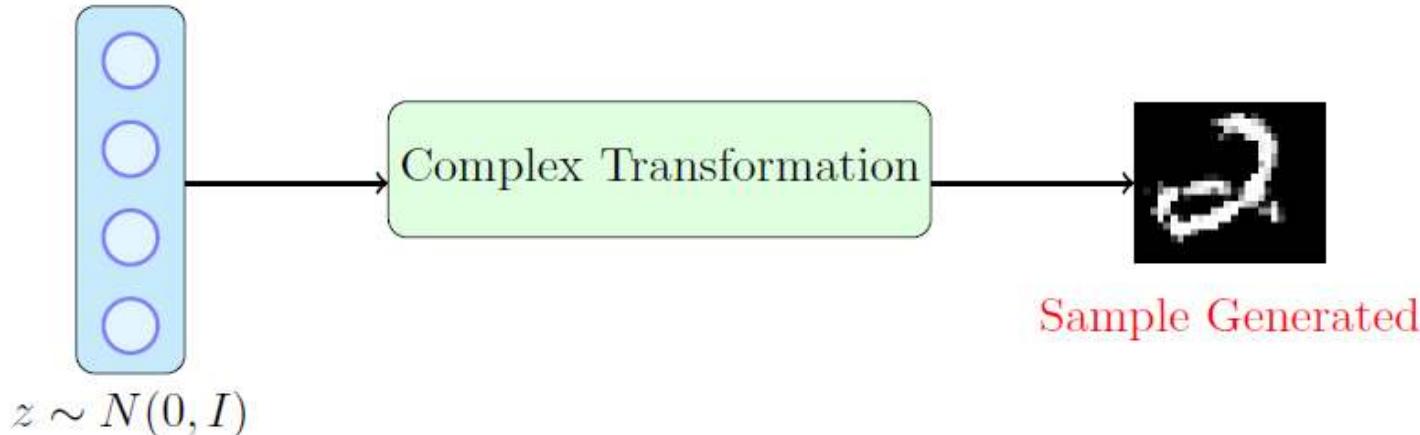
- Most of the generative models perform maximum likelihood estimation (MLE), where we have a density function ($p_{\text{model}}(x)$) that the model describes (x is a vector representing the input)
- Specifically, ($p_{\text{model}}(x | \Theta)$) is a distribution controlled by the parameters Θ that describes exactly where the data concentrates and where it is spread more sparsely.
- MLE aims to measure the log probability that the above density function assigns to all the training data points and adjust the parameters Θ to increase this probability.

Taxonomy of Generative Models



(Goodfellow 2016)

Generative Adversarial Networks: Overview

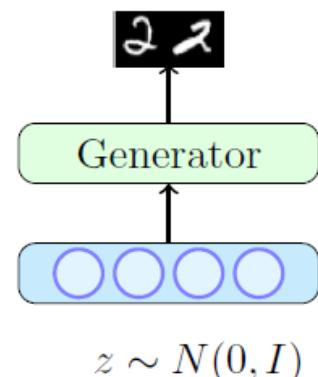


- What can we use for such a complex transformation?
 - A Neural Network
- How do you train such a neural network?
 - Using a two-player game.
- Basic ideas of GANs is similar to Variational autoencoders (VAEs) where we sample from a simple tractable distribution and then learn a complex transformation on it so that the o/p looks as if it came from the training distribution.
- However, GANs start with a d-dimensional vector and generate a n-dimensional vector (usually, $d < n$) as compared to VAEs which start from n-dimensional i/p and produce o/p of same dimension.

Generative Adversarial Networks: Overview

GANs consists of two networks that involve in an adversarial game:

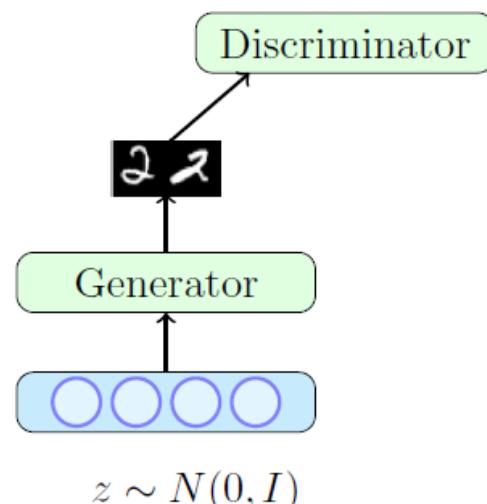
1. **Generator:** A neural network that takes as **input random noise** and transforms it into a sample from the model distribution.
2. **Discriminator:** A neural network that distinguishes between output data point (**Fake**) from the Generator and training data samples (**Real**)



Generative Adversarial Networks: Overview

GANs consists of two networks that involve in an adversarial game:

1. **Generator:** A neural network that takes as **input random noise** and transforms it into a sample from the model distribution.
2. **Discriminator:** A neural network that distinguishes between output data point (**Fake**) from the Generator and training data samples (**Real**)

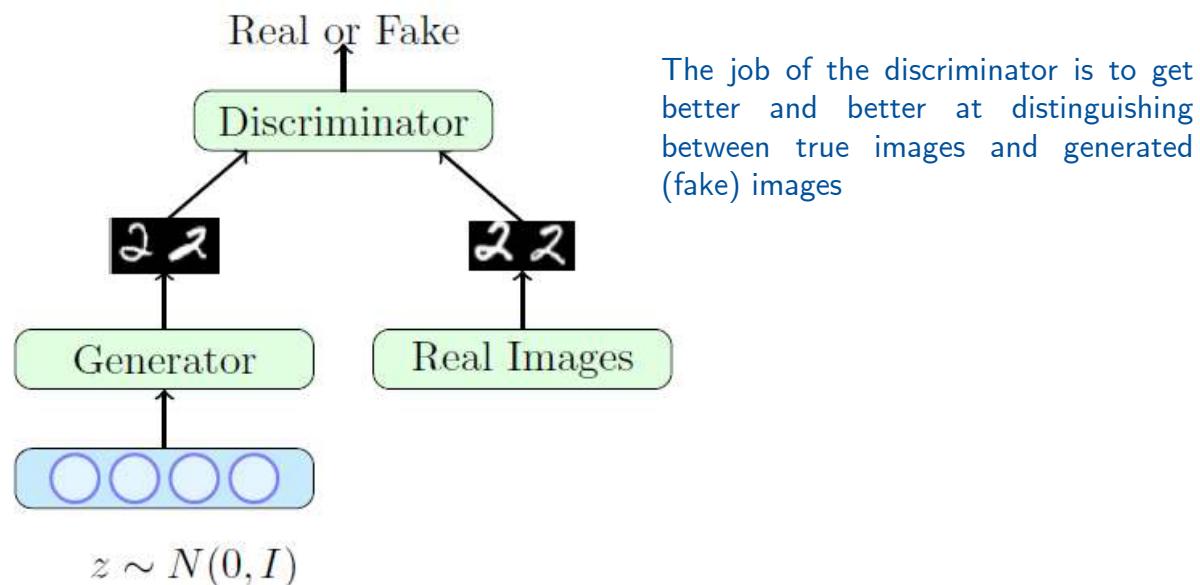


Generative Adversarial Networks: Overview

GANs consists of two networks that involve in an adversarial game:

1. **Generator:** A neural network that takes as input random noise and transforms it into a sample from the model distribution.
2. **Discriminator:** A neural network that distinguishes between output data point (**Fake**) from the Generator and training data samples (**Real**)

The job of the generator is to produce images which look so natural that the discriminator thinks that the images came from the real data distribution.

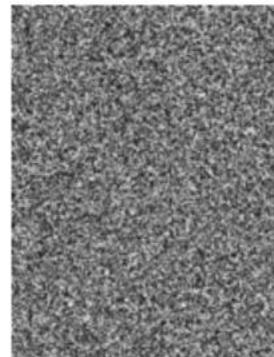


Generative Adversarial Networks: Overview

The generator mostly starts by generating a noisy image (as it takes random latent vectors as inputs).



Generator



Discriminator

Nope!

[A Friendly Introduction to Generative Adversarial Networks \(GANs\) - YouTube](#)

Generative Adversarial Networks: Overview

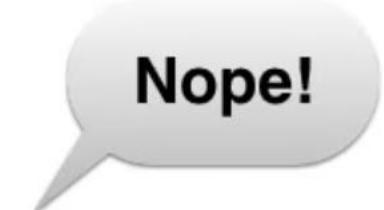
The generator output keeps improving during training.



Generator



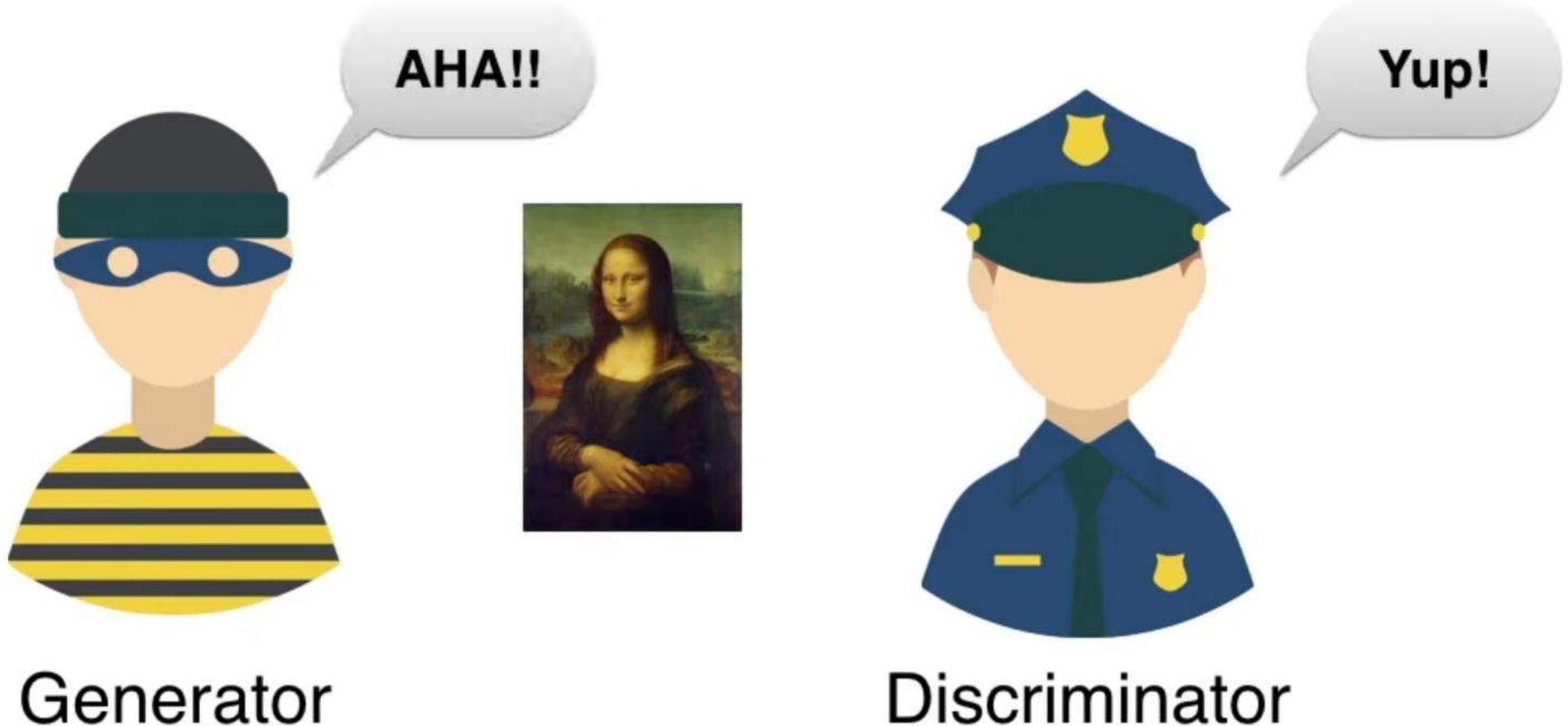
Discriminator



[A Friendly Introduction to Generative Adversarial Networks \(GANs\) - YouTube](#)

Generative Adversarial Networks: Overview

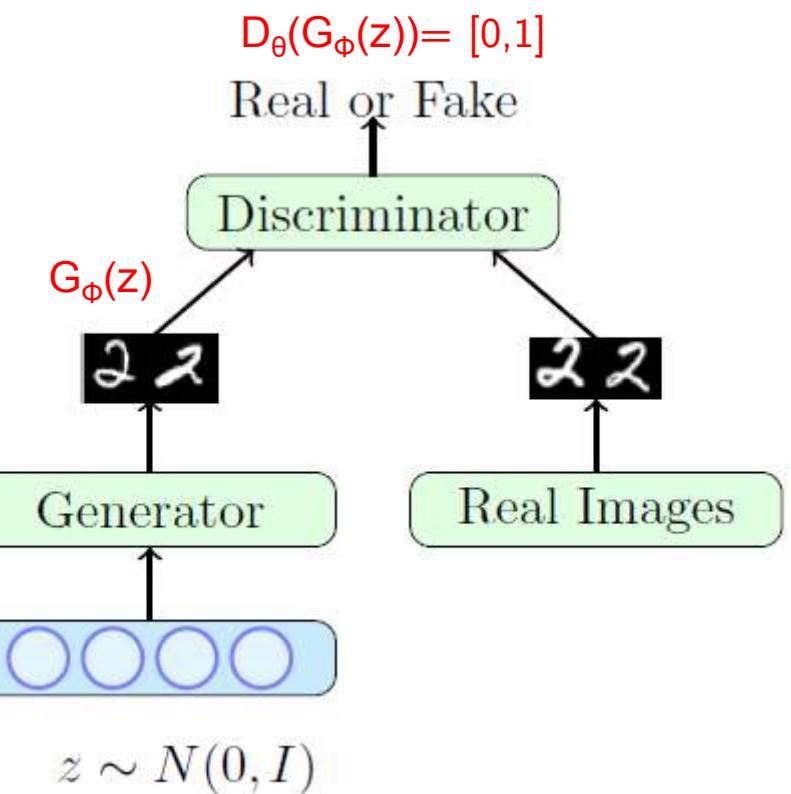
Equilibrium is reached when the generator finally succeeds to fool the discriminator.



[A Friendly Introduction to Generative Adversarial Networks \(GANs\) - YouTube](#)

Objective function of Generator

- Let G_Φ be the generator and D_θ be the discriminator (Φ and θ are the parameters of G and D, respectively)
- We have a neural network-based generator which takes as input a noise vector $z \sim N(0; I)$ and produces $G_\Phi(z) = X$
- We have a neural network-based discriminator which could take as input a real X or a generated $X = G_\Phi(z)$ and classify the input as real/fake
- Given an image generated by the generator as $G_\Phi(z)$ the discriminator assigns a score $D_\theta(G_\Phi(z))$ to it
- This score will be between 0 and 1 and will tell us the probability of the image being real or fake
- For a given z , the generator would want to maximize $\log D_\theta(G_\Phi(z))$ (log likelihood) or minimize $\log(1 - D_\theta(G_\Phi(z)))$



Objective function of Generator

This is just for a single z and the generator would like to do this for all possible values of z ,

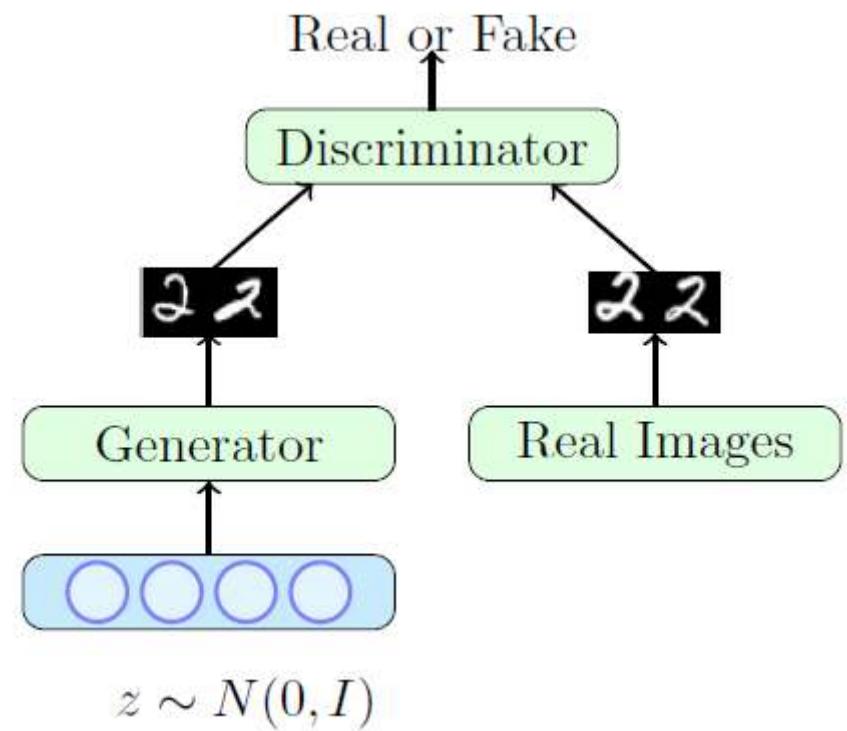
For example, if z was discrete and drawn from a uniform distribution (*i.e.*, $p(z) = \frac{1}{N} \forall z$) then the generator's objective function would be

$$\min_{\phi} \sum_{i=1}^N \frac{1}{N} \log(1 - D_{\theta}(G_{\phi}(z)))$$

However, in our case, z is continuous and not uniform ($z \sim N(0, I)$) so the equivalent objective function would be

$$\min_{\phi} \int p(z) \log(1 - D_{\theta}(G_{\phi}(z)))$$

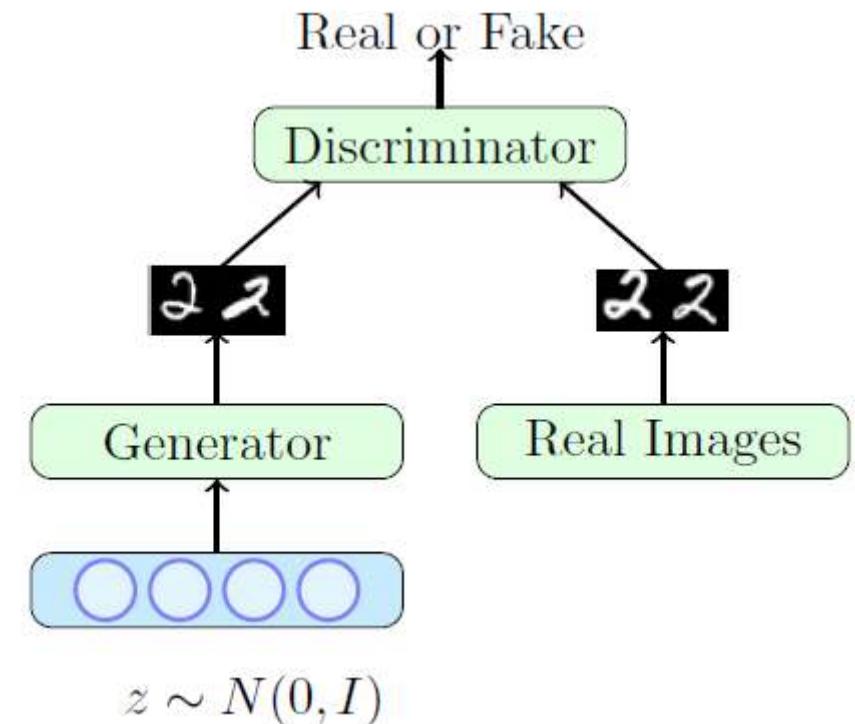
$$\boxed{\min_{\phi} E_{z \sim p(z)} [\log(1 - D_{\theta}(G_{\phi}(z)))]}$$



Objective function of Discriminator

- The task of the discriminator is to assign a high score to real images and a low score to fake images
- And it should do this for all possible real images and all possible fake images
- In other words, it should try to maximize the following objective function:

$$\max_{\theta} E_{x \sim p_{data}} [\log D_{\theta}(x)] + E_{z \sim p(z)} [\log(1 - D_{\theta}(G_{\phi}(z)))]$$



Generative Adversarial Networks: MiniMax formulation

Objective function:

$$\min_G \max_D V(\theta_D, \Phi_G) = \mathbb{E}_{x \sim P_{data}} \log D(x) + \mathbb{E}_{z \sim P(z)} \log(1 - D(G(z)))$$

Maximizes the obj. fn.
w.r.t to the Discriminator
network parameters

Minimizes the obj. fn.
w.r.t to the Generator
network parameters

The discriminator wants to maximize the second term whereas the generator wants to minimize it (hence it is a two-player game)

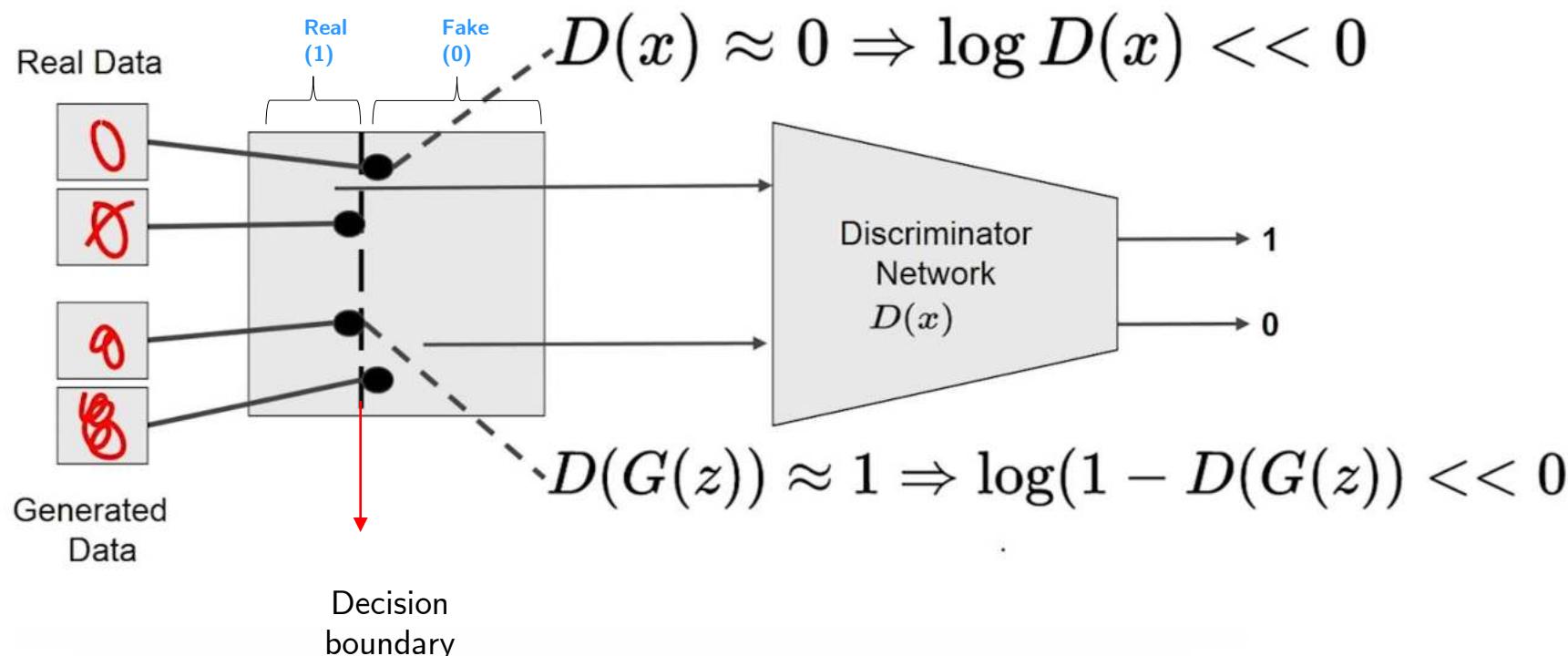
Training the Discriminator Network

- Before training (when discriminator is not performing optimally- cannot clearly distinguish real and fake data)

$$\max_D [\mathbb{E}_{x \sim P_{data}(x)} \log(D(x)) + \mathbb{E}_{z \sim P_z(z)} \log(1 - D(G(z)))]$$

$D(x)$ should be 1

$D(G(z))$ should be 0



[Generative Adversarial Networks \(GAN\) - YouTube](#)

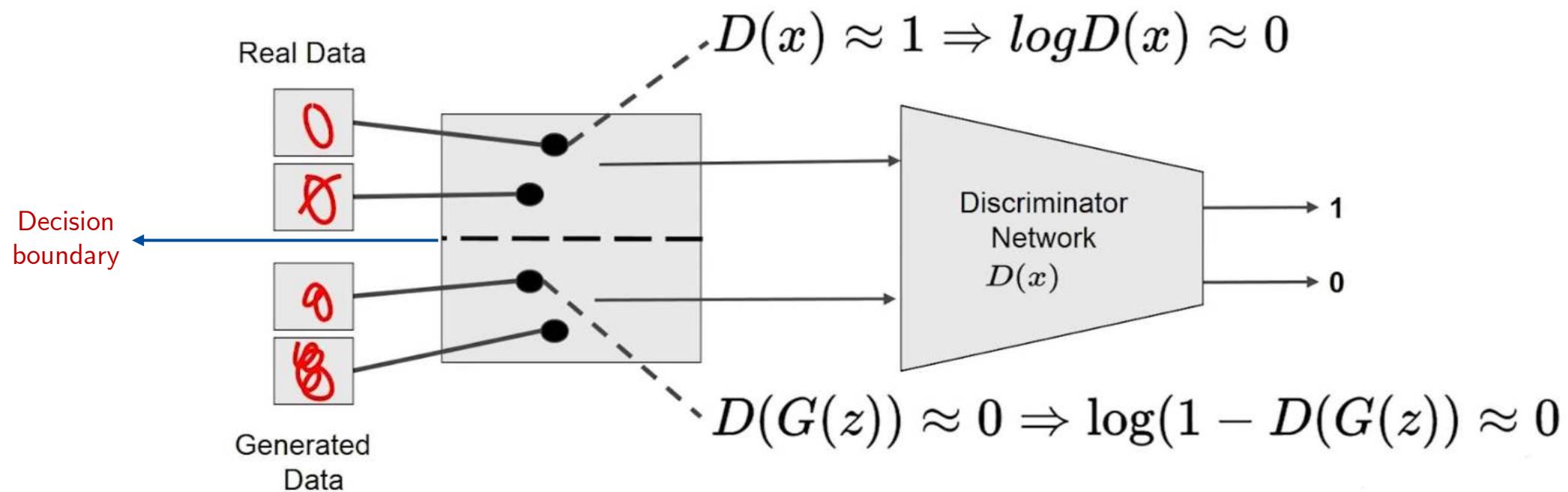
Training the Discriminator Network

- After training (when discriminator is performing optimally – clearly distinguishes real and fake data)

$$\max_D [\mathbb{E}_{x \sim P_{data}(x)} \log(D(x)) + \mathbb{E}_{z \sim P_z(z)} \log(1 - D(G(z)))]$$

$D(x)$ should be 1

$D(G(z))$ should be 0



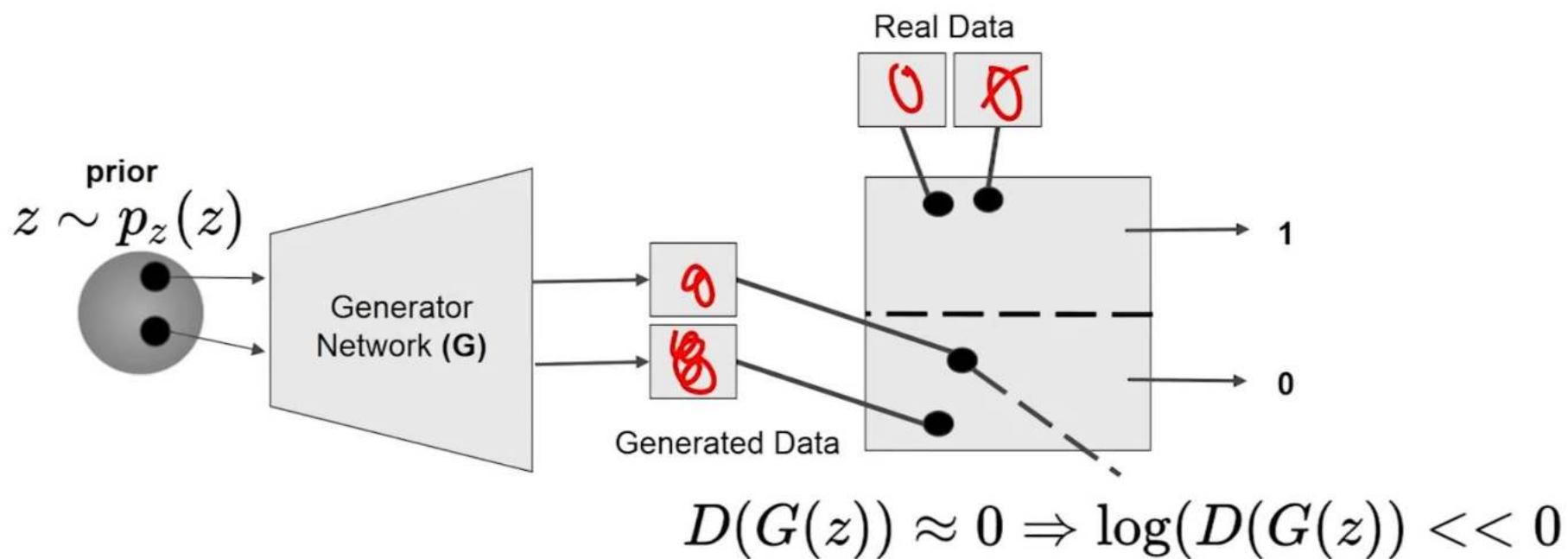
[Generative Adversarial Networks \(GAN\) - YouTube](#)

Training the Generator Network

- Before training

$$\max_G [\mathbb{E}_{z \sim P_z(z)} \log(D(G(z)))] \rightarrow D(G(z)) \text{ should be 1}$$

Tries to maximize the error in D.
That is, it incorrectly classifies $D(G(z))$ as 1 instead of 0



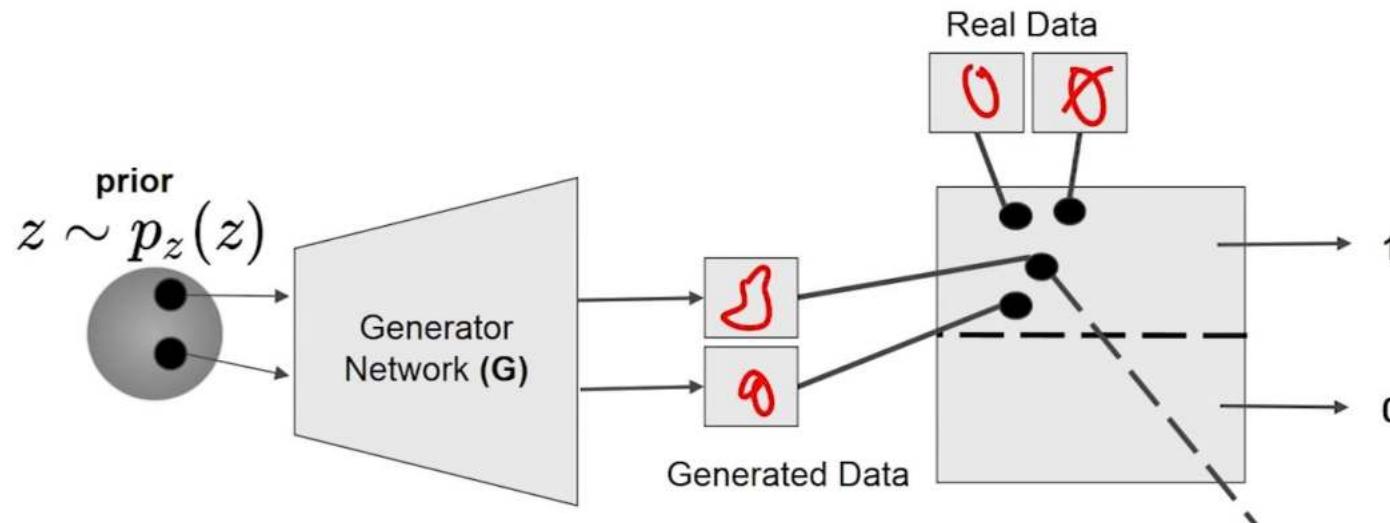
[Generative Adversarial Networks \(GAN\) - YouTube](#)

Training the Generator Network

- After training

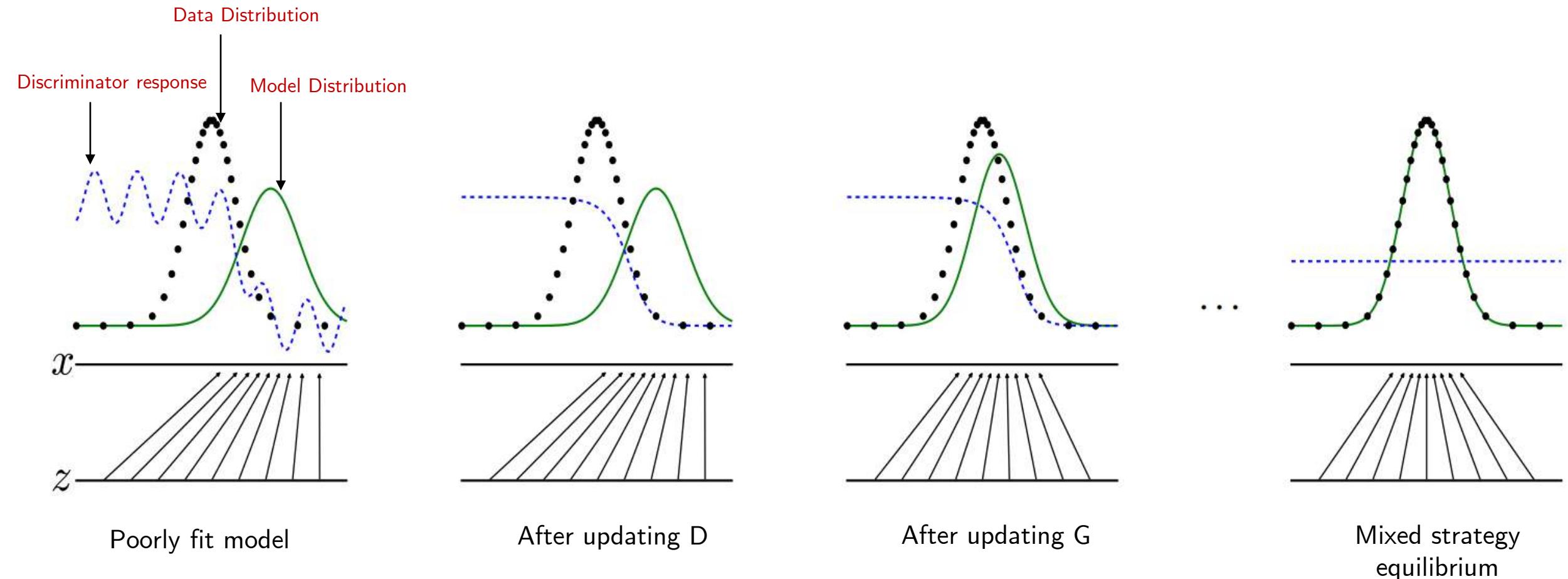
$$\max_G [\mathbb{E}_{z \sim P_z(z)} \log(D(G(z)))]$$

$D(G(z))$ should be 1



$$D(G(z)) \approx 1 \Rightarrow \log(D(G(z))) \approx 0$$

Training the Generator Network



[Generative Adversarial Networks \(GAN\) - YouTube](#)

Training the GANs

```
1: procedure GAN TRAINING
2:   for number of training iterations do
3:     for k steps do
4:       • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ 
5:       • Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{data}(\mathbf{x})$ 
6:       • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta} \frac{1}{m} \sum_{i=1}^m \left[ \log D_{\theta} \left( x^{(i)} \right) + \log \left( 1 - D_{\theta} \left( G_{\phi} \left( z^{(i)} \right) \right) \right) \right]$$

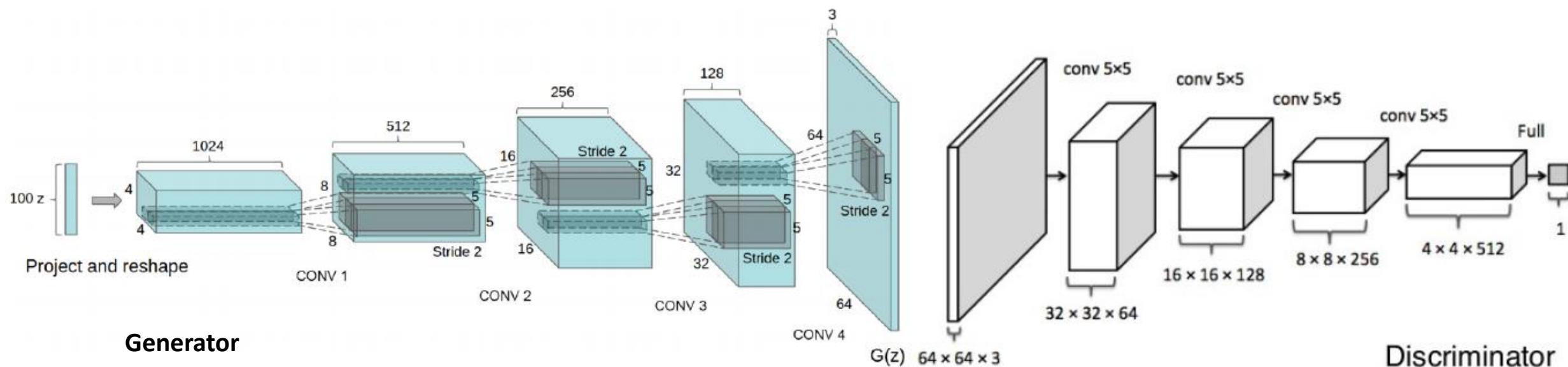
7:     end for
8:     • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ 
9:     • Update the generator by ascending its stochastic gradient

$$\nabla_{\phi} \frac{1}{m} \sum_{i=1}^m \left[ \log \left( D_{\theta} \left( G_{\phi} \left( z^{(i)} \right) \right) \right) \right]$$

10:   end for
11: end procedure
```

Deep Convolutional GANs

For discriminator, any CNN based classifier with 1 class (real) at the output can be used (e.g. VGG, ResNet, etc.)



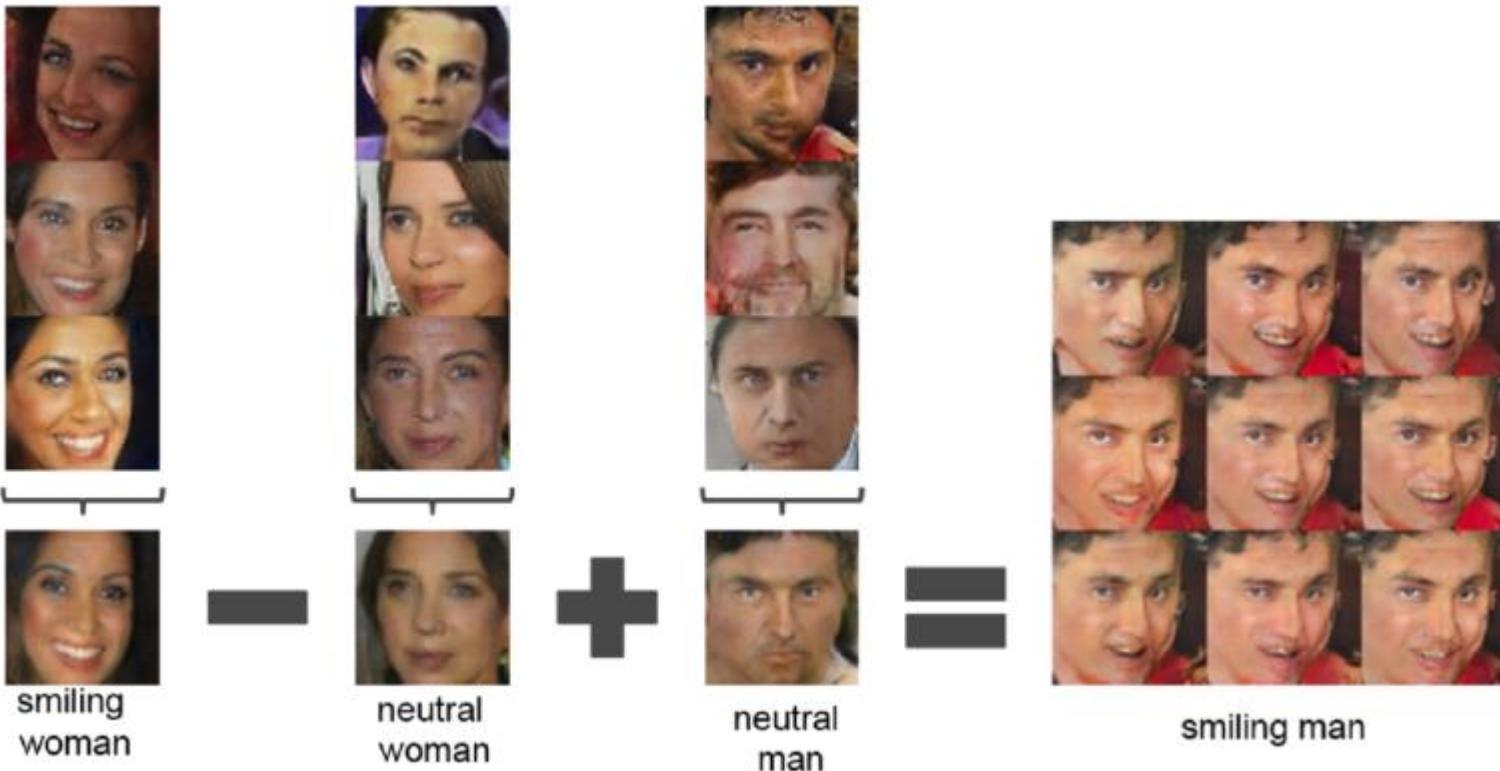
Radford et al. Unsupervised Representational Learning with Deep Convolutional GANs, ICLR 2016

Deep Convolutional GANs

Architecture guidelines for stable Deep Convolutional GANs

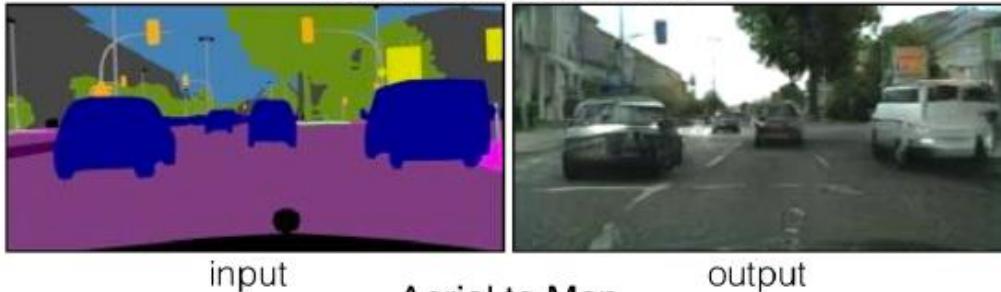
- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses tanh.
- Use LeakyReLU activation in the discriminator for all layers

GANs: Applications



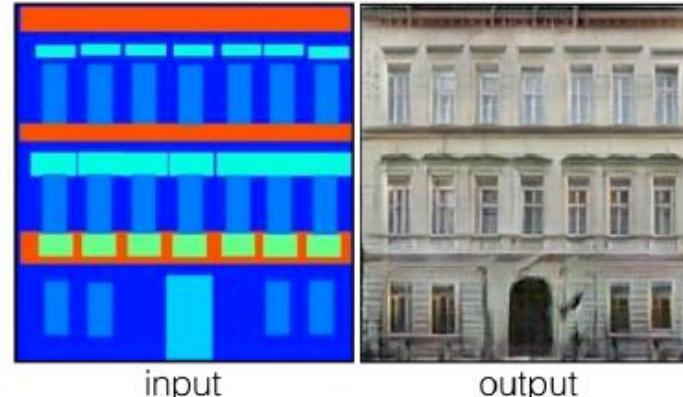
GANs: Applications

Labels to Street Scene



input

Labels to Facade



input

BW to Color



input

output

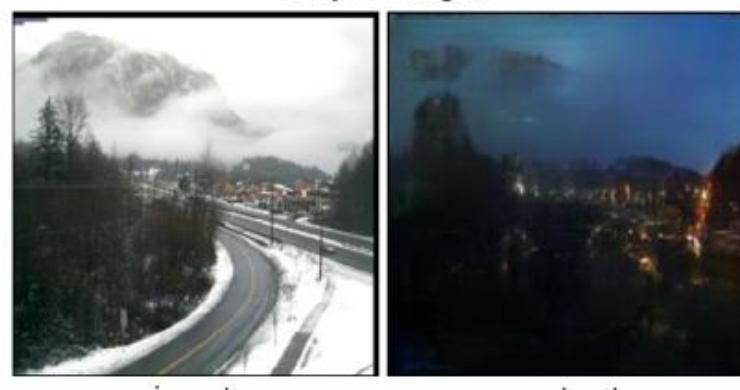
Aerial to Map



input

output

Day to Night



input

output

Edges to Photo



input

output

Philip et al. Image-to-Image Translation with Conditional Adversarial Networks, CVPR 2017