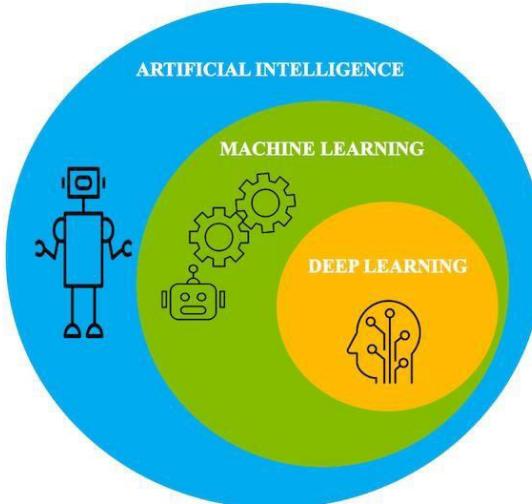


Lambton
College

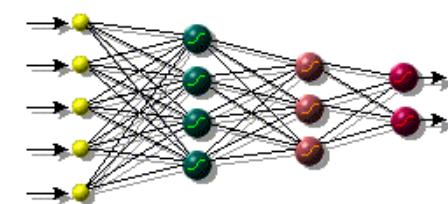


Lambton College
School of Computer Studies

AML-3104 Neural Networks
and Deep Learning

Deep Learning

AML-3104



Conventional Machine Learning

- Process natural data in raw form
- Constructing features by hand
 - Requires domain expertise
 - Difficult and time consuming



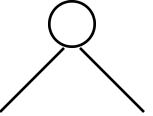
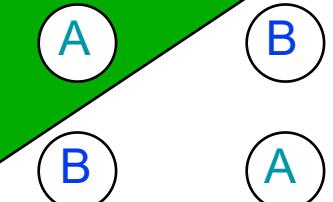
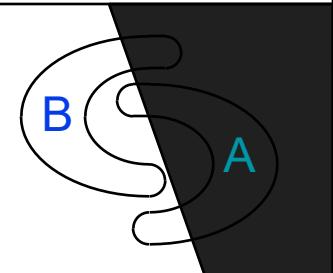
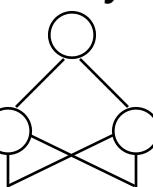
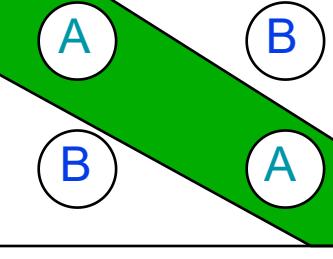
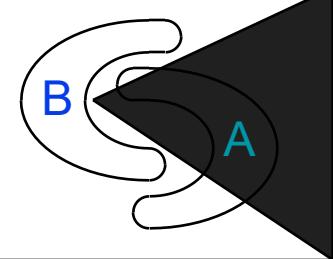
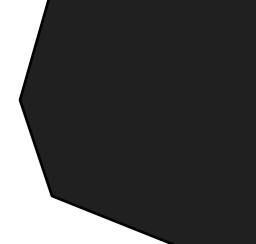
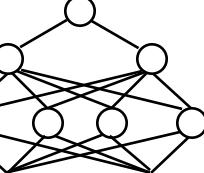
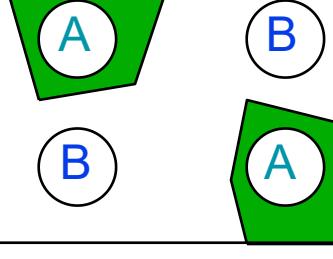
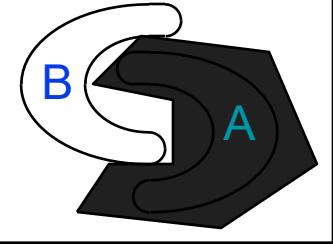
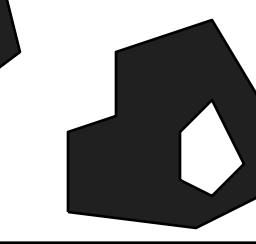
Representation Learning

Representation Learning: allows a machine to be fed with raw data and to automatically discover representations needed for detection/classification

Deep Learning: representation learning with multiple layers of representation (more than 3)

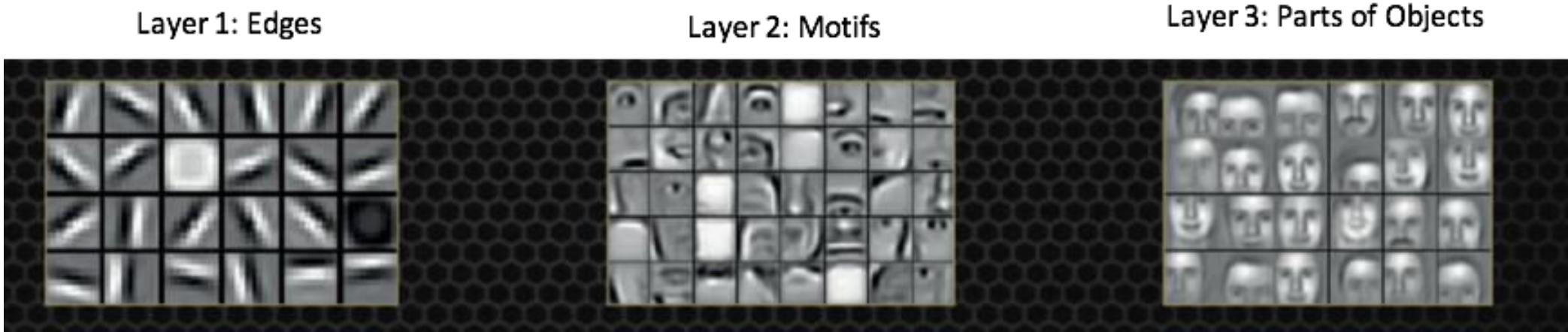
- Transformed into higher, slightly more abstract level
- Very complex functions can be learned

Different Non-Linear Separable Problems

Structure	<i>Types of Decision Regions</i>	<i>Exclusive-OR Problem</i>	<i>Classes with Meshed regions</i>	<i>Most General Region Shapes</i>
<i>Single-Layer</i> 	<i>Half Plane Bounded By Hyperplane</i>			
<i>Two-Layer</i> 	<i>Convex Open Or Closed Regions</i>			
<i>Three-Layer</i> 	<i>Arbitrary (Complexity Limited by No. of Nodes)</i>			

Deep Learning

- Layers are not handcrafted
- Features are learned from raw data via a general-purpose learning algorithm



<https://devblogs.nvidia.com/parallelforall/accelerate-machine-learning-cudnn-deep-neural-network-library/>

Applications of Deep Learning

- Domains in science, business and government
- Beat current records in image and speech recognition
- Beaten other machine-learning techniques at
 - Predicting activity of potential drug molecules
 - Analyzing particle accelerator data
 - Reconstructing brain circuits
- Produced promising results in natural language understanding
 - Topic classification
 - Sentiment analysis
 - Question answering

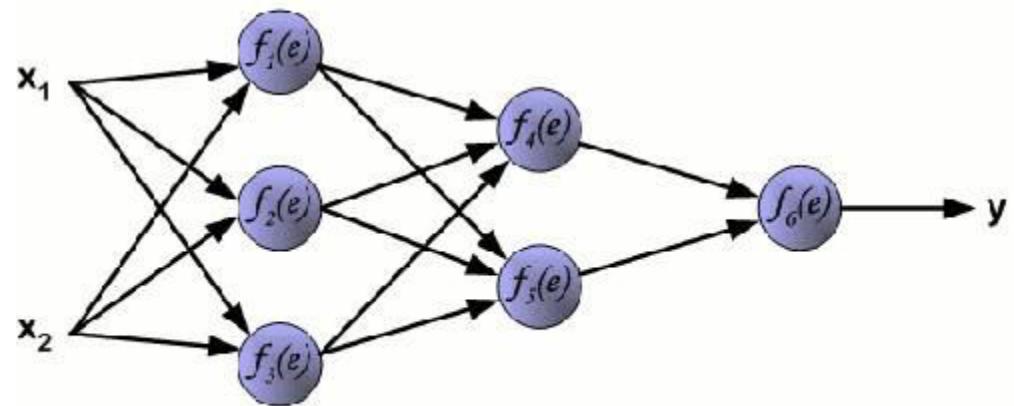
Overview

- Supervised Learning
- Backpropagation to Train Multilayer Architectures
- Convolution Neural Networks
- Image Understanding with Deep Convolution Networks
- Distributed Representation and Language Processing
- Recurrent Neural Networks
- Future of Deep Learning

Supervised Learning

- Most common form of machine learning
 - Data set
 - Labeling
 - Training on data set (tuning parameters, gradient descent)
 - Testing
- **Objective function:** measures error between output scores and the desired pattern of scores
- Modifies internal adjustable parameters (weights) to reduce error

Back-Propagation



Supervised Learning

- Objective Function → “Hilly landscape” in high dimensional space of weight values
- Computes a gradient vector
 - Indicates how much the error would increase or decrease if the weight were increased by a tiny amount
 - Negative gradient vector indicates the direction of steepest descent in this landscape
 - Taking it closer to a minimum, where the output error is low on average

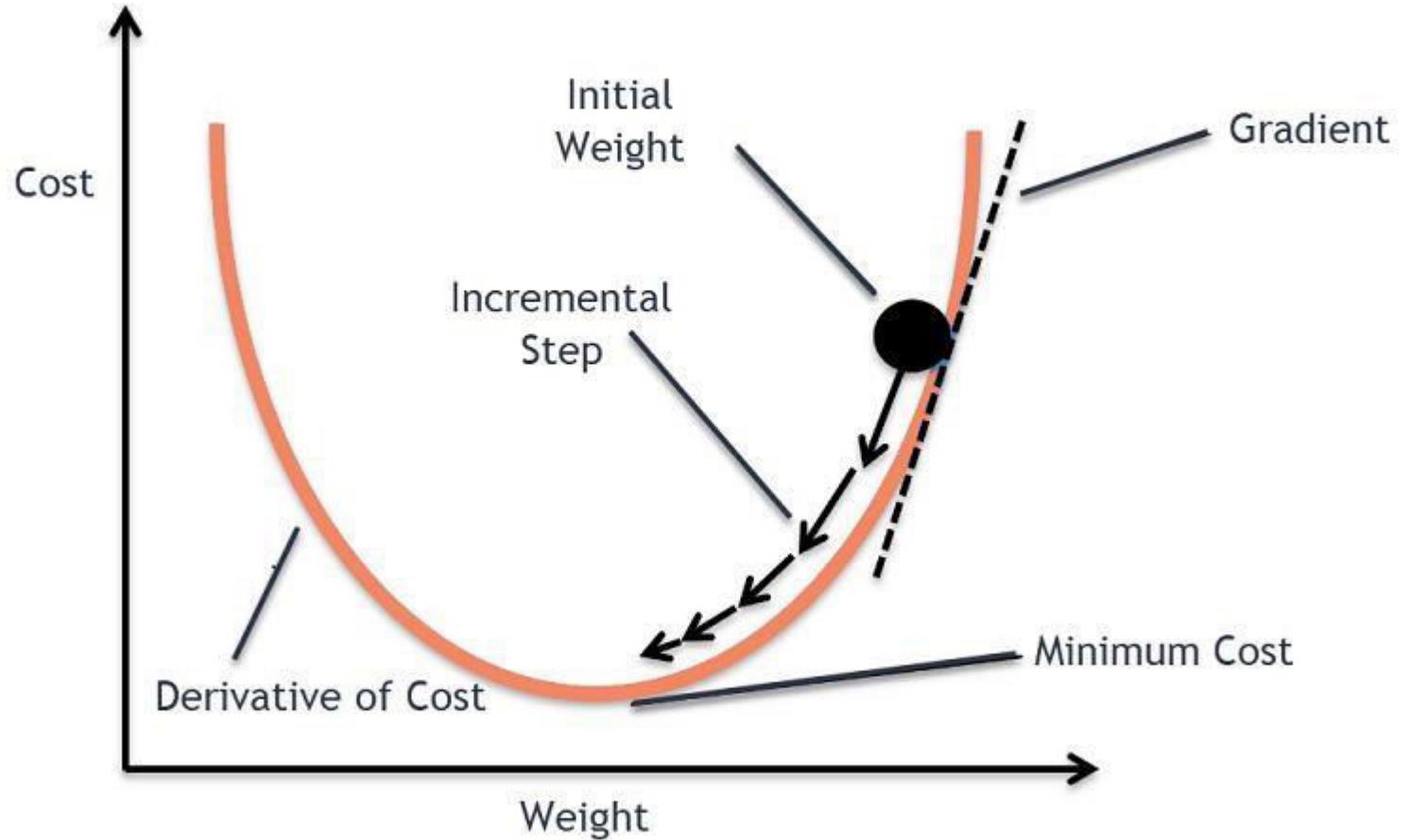
Gradient Descent Algorithm

- Gradient descent is an optimization algorithm used to find the values of parameters of a function that minimizes a cost function.
- It is an iterative algorithm.
- We use gradient descent to update the parameters of the model.

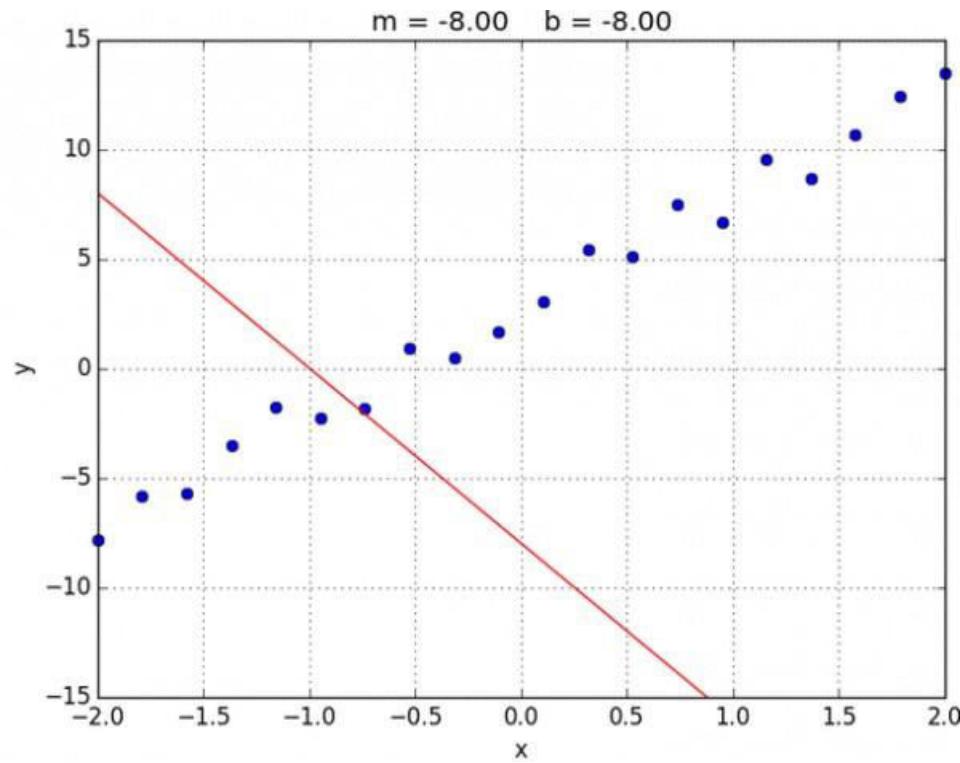
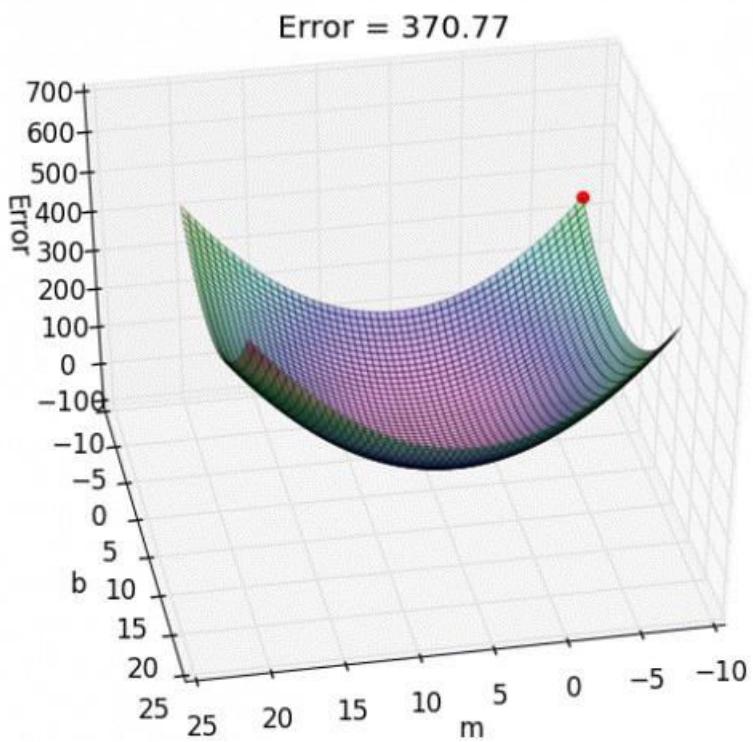
Repeat until convergence

$$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

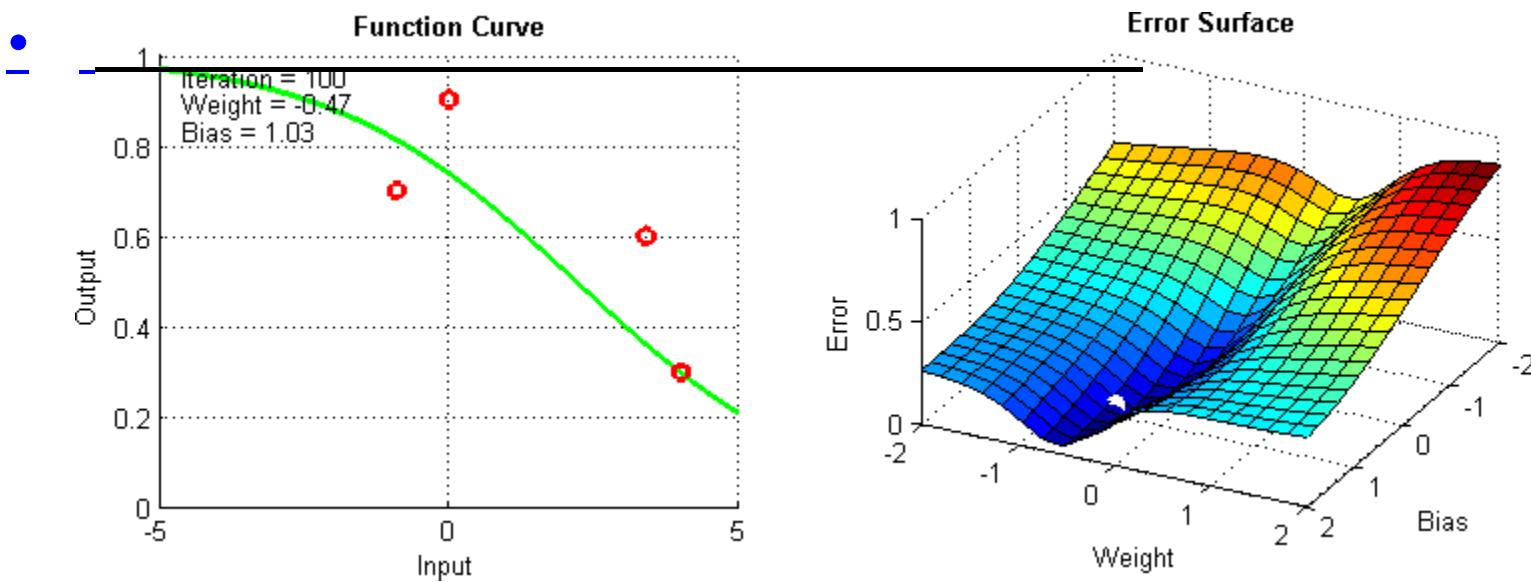
Gradient Descent



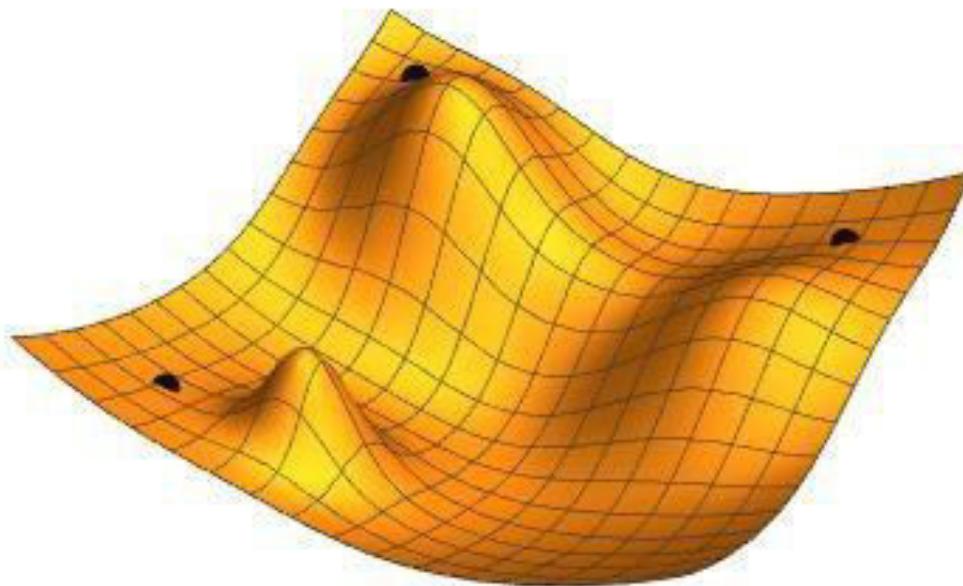
GRADIENT DESCENT



GRADIENT DESCENT



GRADIENT DESCENT



Gradient descent

- We'll update the weights
- Move in direction opposite to gradient:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$

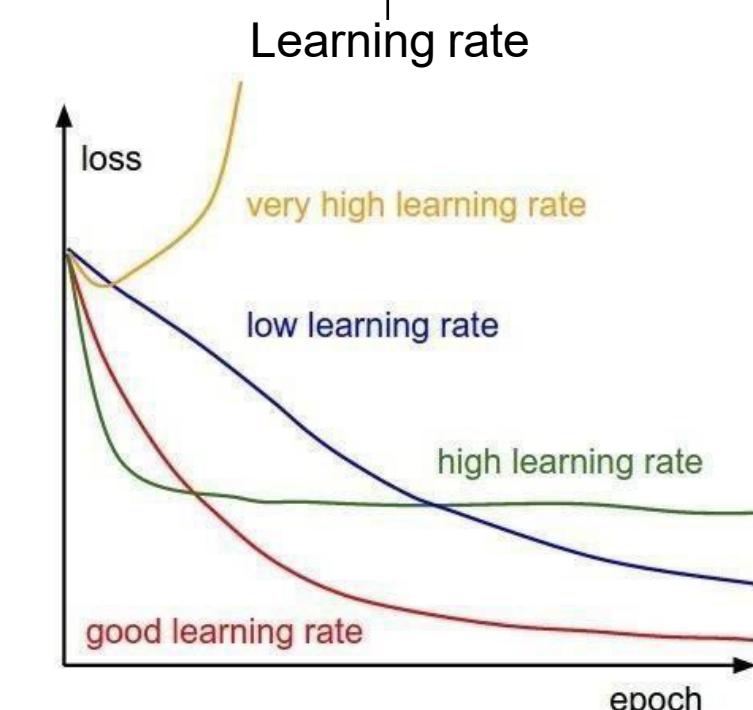


Figure from Andrej Karpathy

Gradient Descent Algorithm For Linear Regression

Cost Function

$$J(\Theta_0, \Theta_1) = \frac{1}{2m} \sum_{i=1}^m [h_\Theta(x_i) - y_i]^2$$

↑
True Value
Predicted Value

Gradient Descent

$$\Theta_j = \Theta_j - \alpha \frac{\partial}{\partial \Theta_j} J(\Theta_0, \Theta_1)$$

↑
Learning Rate

Now,

$$\begin{aligned}\frac{\partial}{\partial \Theta} J_\Theta &= \frac{\partial}{\partial \Theta} \frac{1}{2m} \sum_{i=1}^m [h_\Theta(x_i) - y_i]^2 \\ &= \frac{1}{m} \sum_{i=1}^m (h_\Theta(x_i) - y) \frac{\partial}{\partial \Theta_j} (\Theta x_i - y) \\ &= \frac{1}{m} (h_\Theta(x_i) - y) x_i\end{aligned}$$

Therefore,

$$\Theta_j := \Theta_j - \frac{\alpha}{m} \sum_{i=1}^m [(h_\Theta(x_i) - y) x_i]$$

Gradient Descent Algorithm

❑ Types of Gradient Descent:

Typically, there are three types of Gradient Descent:

- Batch Gradient Descent
- Stochastic Gradient Descent
- Mini-batch Gradient Descent

Stochastic Gradient Descent

❑ Stochastic Gradient Descent:

- The word ‘stochastic’ means a system or a process that is linked with a random probability. Hence, in Stochastic Gradient Descent, a few samples are selected randomly instead of the whole data set for each iteration.
- SGD tries to solve the main problem in Batch Gradient descent-the usage of whole training data to calculate gradients at each step.
- SGD is stochastic in nature i.e it picks up a “random” instance of training data at each step and then computes the gradient making it much faster as there is much fewer data to manipulate at a single time, unlike Batch GD.

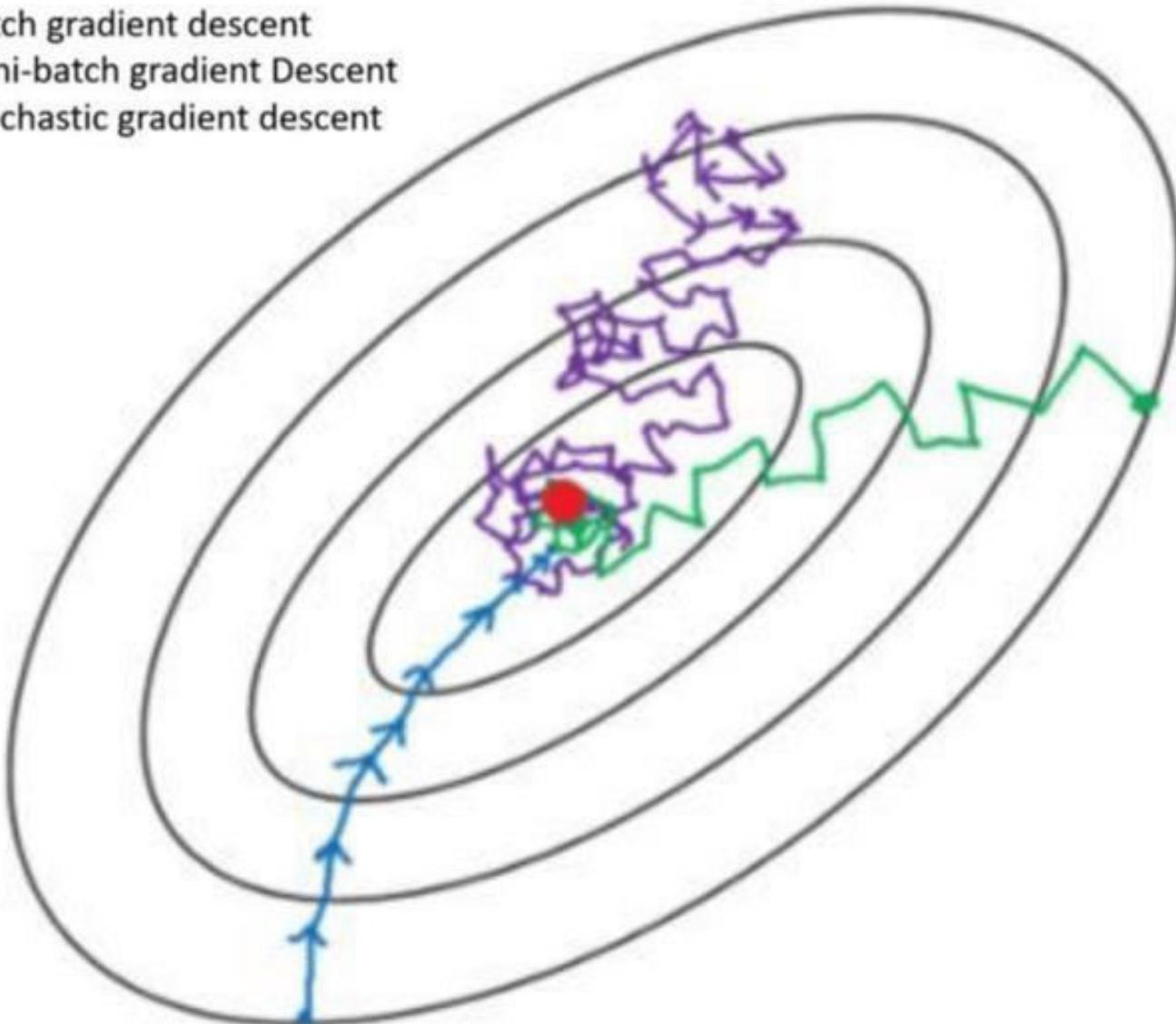
Mini-Batch Gradient Descent

- ❑ Mini-Batch Gradient Descent: Parameters are updated after computing the gradient of error with respect to a subset of the training set

Since a subset of training examples is considered, it can make quick updates in the model parameters and can also exploit the speed associated with vectorizing the code.

Gradient Descent

- Batch gradient descent
- Mini-batch gradient Descent
- Stochastic gradient descent



Stochastic Gradient Descent (SGD)

- Input vector for a few examples
- Compute the outputs and the errors
- Compute the average gradient
- Adjusting the weights
- Repeated for many small sets from the training set until the average of the objective function stops decreasing

Stochastic: small set gives a noisy estimate of the average gradient over all examples

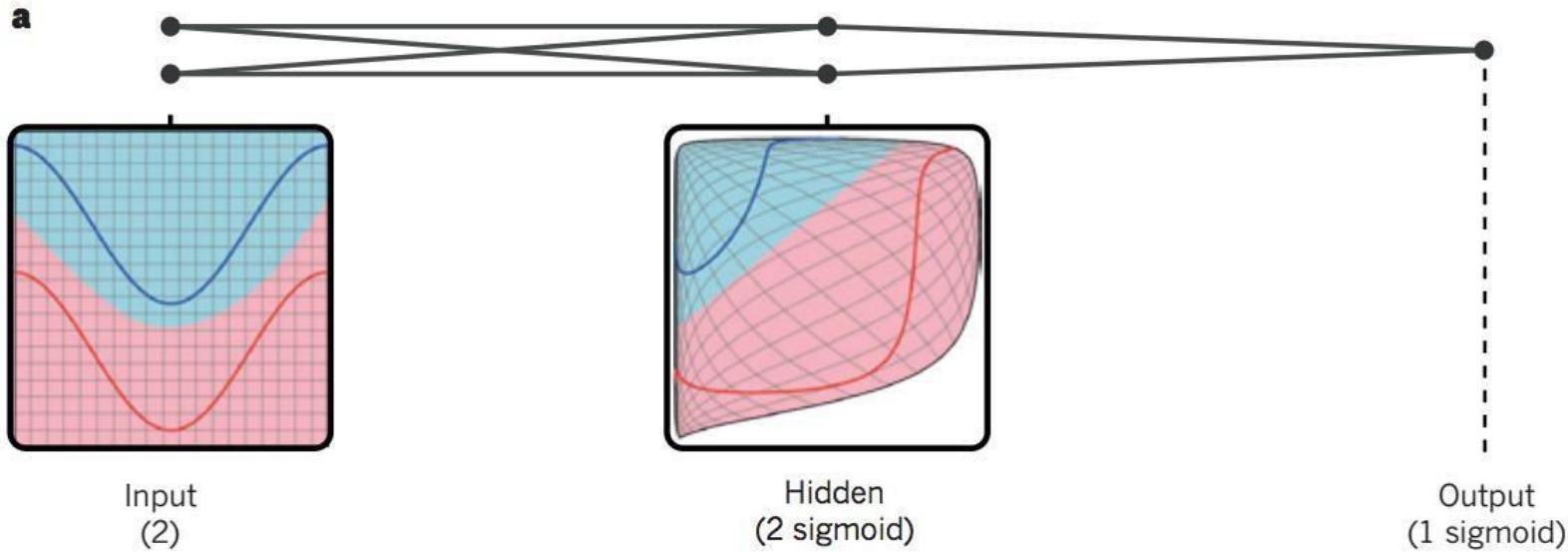
Batch Gradient Descent Vv. Stochastic Gradient Descent

S.NO.	Batch Gradient Descent	Stochastic Gradient Descent
1.	Computes gradient using the whole Training sample	Computes gradient using a single Training sample
2.	Slow and computationally expensive algorithm	Faster and less computationally expensive than Batch GD
3.	Not suggested for huge training samples.	Can be used for large training samples.
4.	Deterministic in nature.	Stochastic in nature.
5.	Gives optimal solution given sufficient time to converge.	Gives good solution but not optimal.
6.	No random shuffling of points are required.	The data sample should be in a random order, and this is why we want to shuffle the training set for every epoch.
7.	Can't escape shallow local minima easily.	SGD can escape shallow local minima more easily.
8.	Convergence is slow.	Reaches rthe convergence much faster.

Back-Propagation

- A training procedure which allows multi-layer feedforward Neural Networks to be trained;
- Can theoretically perform “any” input-output mapping;
- Can learn to solve linearly inseparable problems.

Multi Layer Neural Network

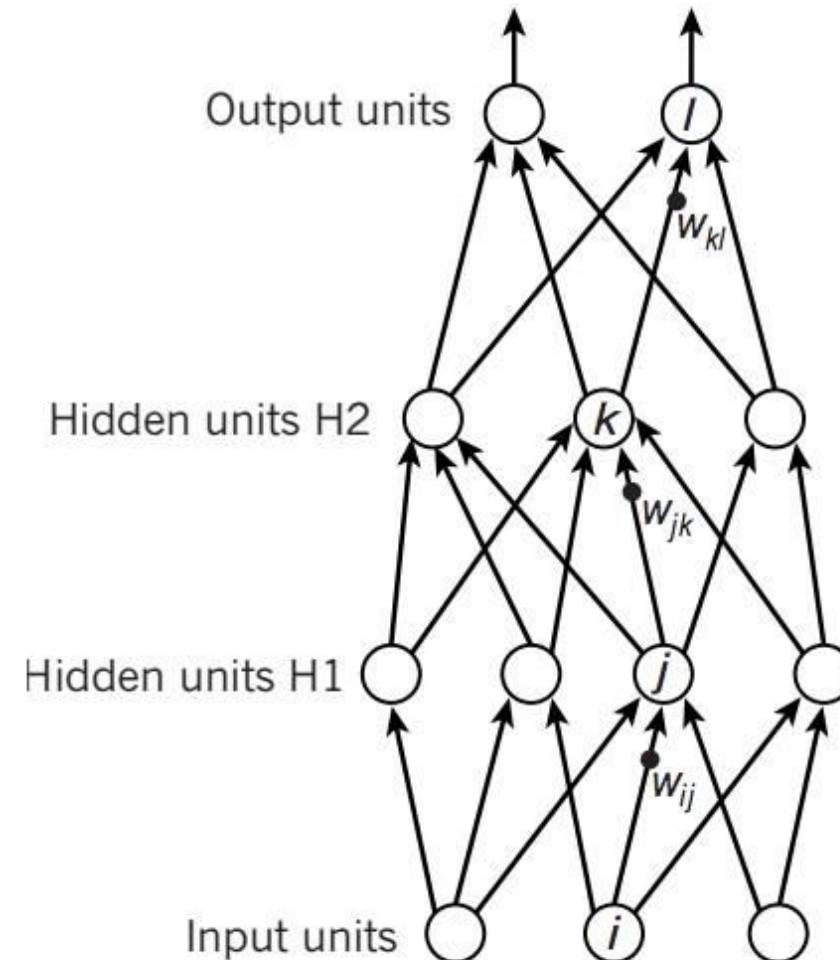


Distort the input space to make the classes of data (ex: red and blue lines) linearly separable

Illustrative example with only two input units, two hidden units and one output unit

Feed Forward

- Map a fixed-size input (Ex: an image) to a fixed-size output (Ex: a probability for each of several categories)
- A set of units compute a weighted sum of their inputs from the previous layer
- Pass the result through a nonlinear function



Backpropagation to Train Multilayer Architectures

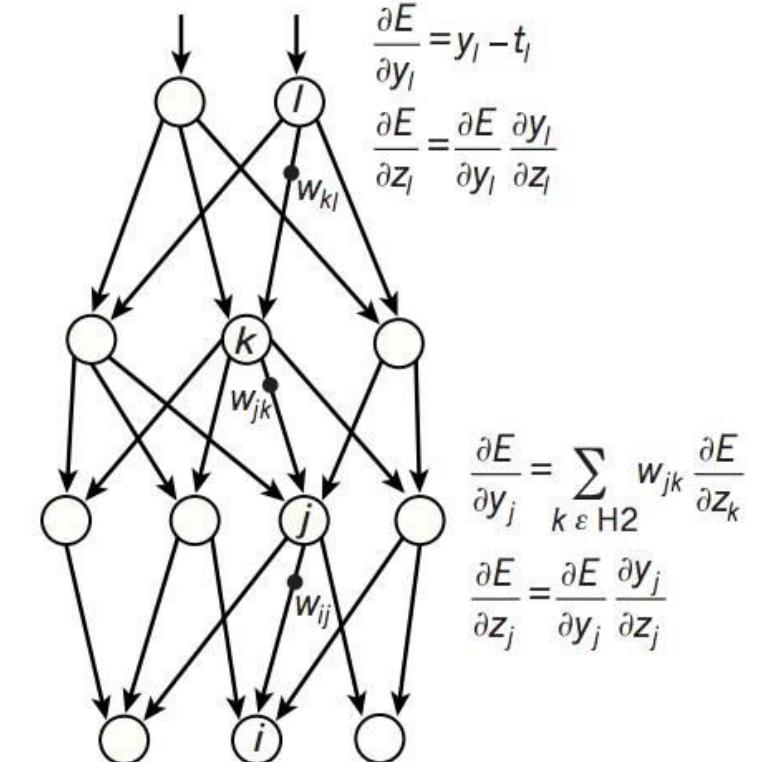
Feed Forward the input

Calculate the error function

Calculate the gradient backward

d

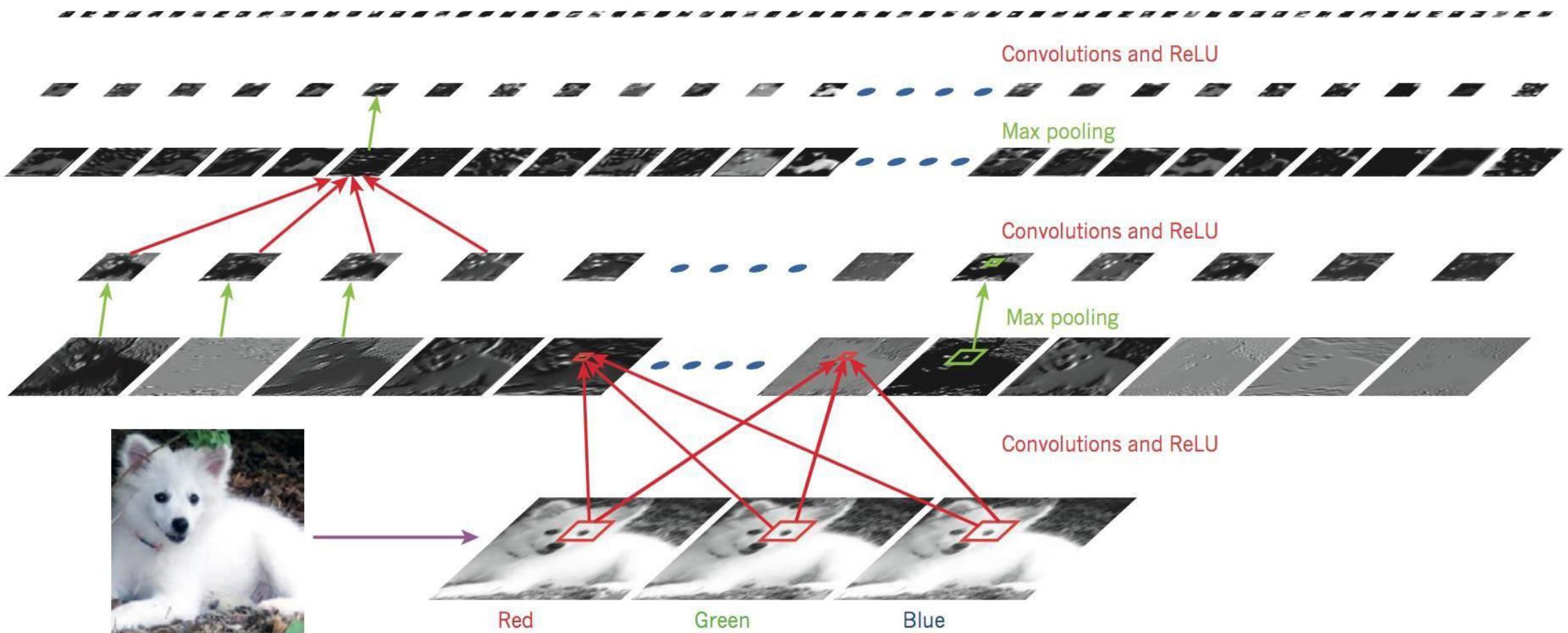
Compare outputs with correct answer to get error derivatives



Convolutional Neural Networks

- Use many different copies of the same feature detector with different positions
- Replication greatly reduces the number of features to be learned
- Enhanced generalization
- Use several different feature type, each with its own map of replicated detectors.

Image Understanding with Deep Convolutional Networks



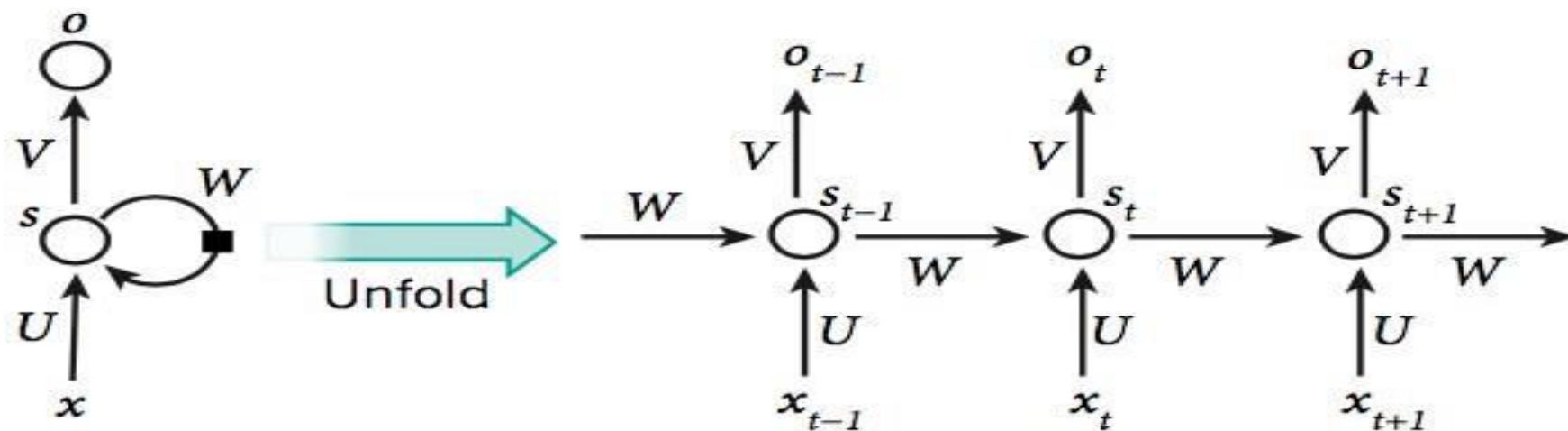
Distributed Representation and Language Processing

- Data is represented as a vector
- Each element is not mutually dependent
- Many possible combination for the same input (stochastic representation)
- Enhanced classification accuracy

Recurrent Neural Networks

- RNNs process an input sequence one element at a time
- Tasks that involve sequential input
 - speech, language
- Trained by backpropagation
 - problematic because the back propagated gradients either grow or shrink at each time step
 - over many time steps they typically explode or vanish

Recurrent Neural Networks



Future of Deep Learning

- Expect unsupervised learning to become more important
 - Human and animal learning is largely unsupervised
- Future progress in vision
 - Systems trained end-to-end
 - Combine ConvNets with RNNs that use reinforcement learning.
- Natural language
 - RNNs systems will become better when they learn strategies for selectively attending to one part at a time

Discussion

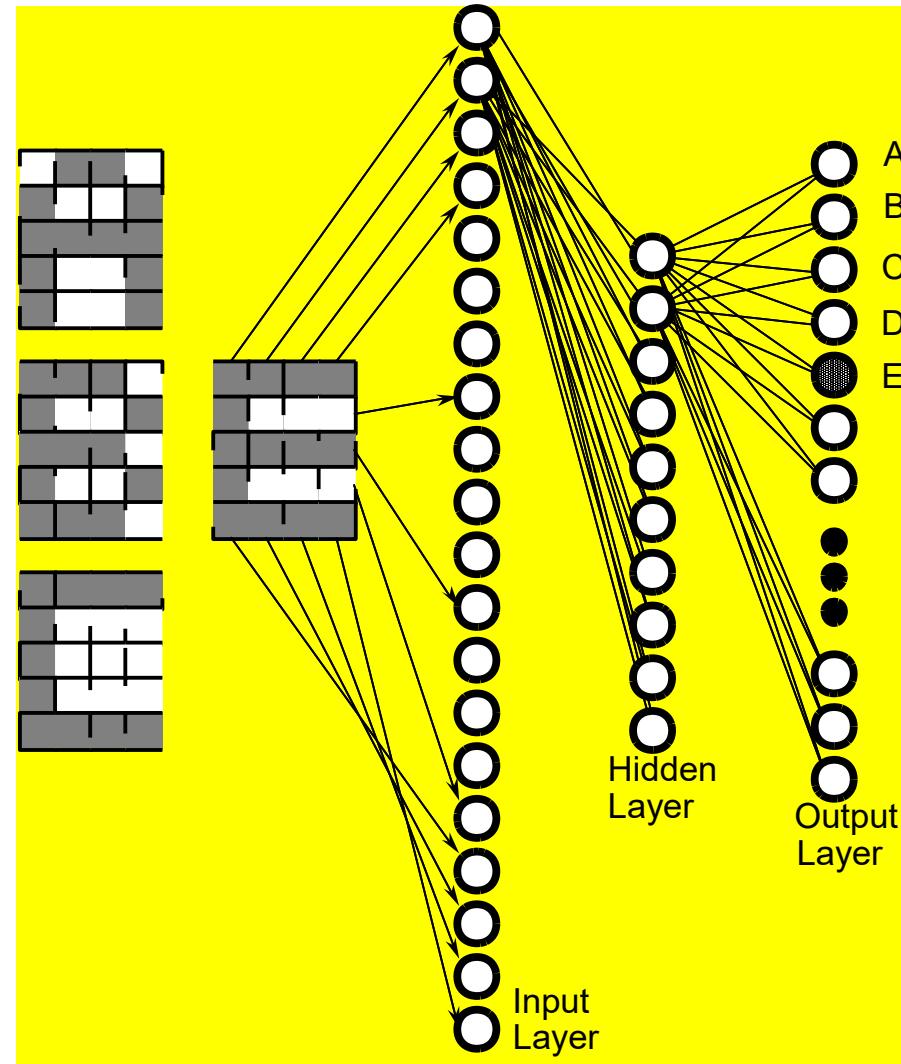
- Deep Learning has already drastically improved the state-of-the-art in
 - image recognition
 - speech recognition
 - natural language understanding
- Deep Learning requires very little engineering by hand and thus has the potential to be applied to many fields

Applications

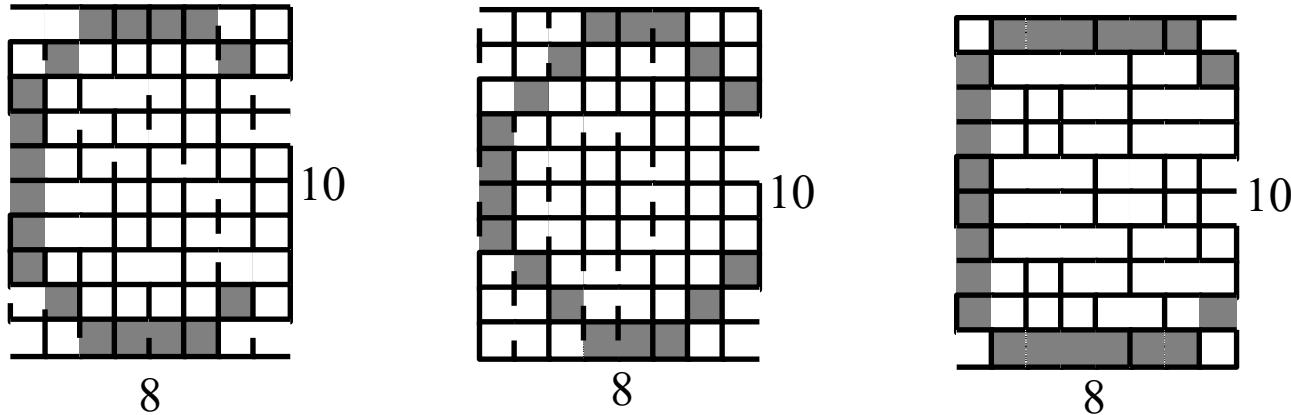
- The properties of deep neural networks define where they are useful.
 - Can learn complex mappings from inputs to outputs, based solely on samples
 - Difficult to analyse: firm predictions about neural network behaviour difficult;
 - Unsuitable for safety-critical applications.
 - Require limited understanding from trainer, who can be guided by heuristics.

Neural network for OCR

- feedforward network
- trained using Back-propagation

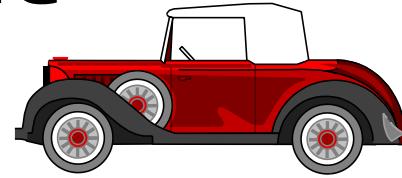


OCR for 8x10 characters



- NN are able to generalise
- learning involves generating a partitioning of the input space
- for single layer network input space must be linearly separable
- what is the dimension of this input space?
- how many points in the input space?
- this network is binary(uses binary values)
- networks may also be continuous

Engine management

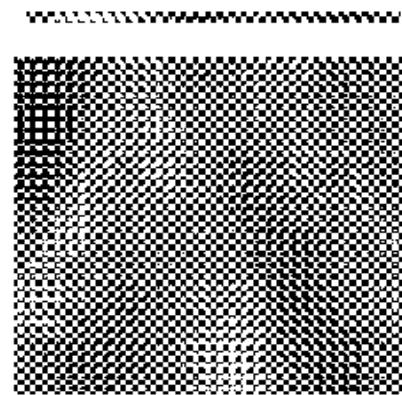
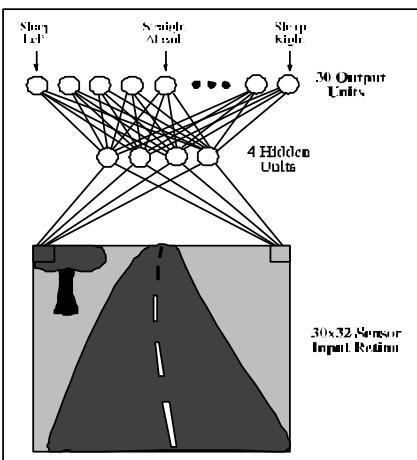


- The behaviour of a car engine is influenced by a large number of parameters
 - temperature at various points
 - fuel/air mixture
 - lubricant viscosity.
- Major companies have used neural networks to dynamically tune an engine depending on current settings.

ALVINN

Drives 70 mph on a public highway

30 outputs
for steering
4 hidden
units
30x32 pixels
as inputs

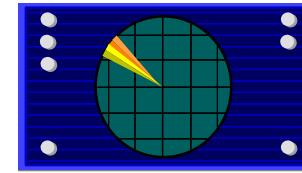


30x32 weights
into one out of
four hidden
unit

Signature recognition

- Each person's signature is different.
- There are structural similarities which are difficult to quantify.
- One company has manufactured a machine which recognizes signatures to within a high level of accuracy.
 - Considers speed in addition to gross shape.
 - Makes forgery even more difficult.

Sonar target recognition



- Distinguish mines from rocks on sea-bed
- The neural network is provided with a large number of parameters which are extracted from the sonar signal.
- The training set consists of sets of signals from rocks and mines.

Stock market prediction



- “Technical trading” refers to trading based solely on known statistical parameters; e.g. previous price
- Neural networks have been used to attempt to predict changes in prices.
- Difficult to assess success since companies using these techniques are reluctant to disclose information.

Mortgage assessment

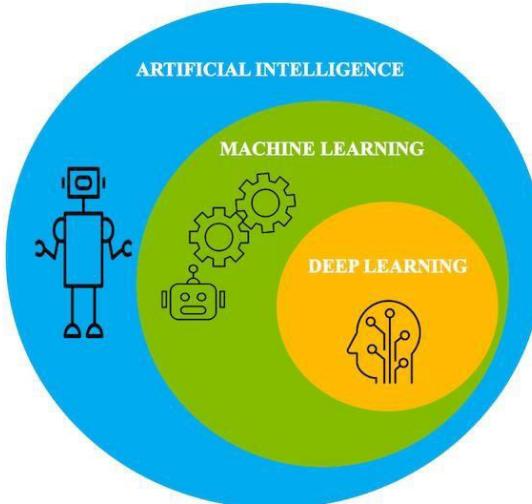


- Assess risk of lending to an individual.
- Difficult to decide on marginal cases.
- Neural networks have been trained to make decisions, based upon the opinions of expert underwriters.
- Neural network produced a 12% reduction in delinquencies compared with human experts.

References

Y. LeCun, Y. Bengio, G. Hinton (2015). Deep Learning. *Nature* 521, 436-444.

Lambton
College

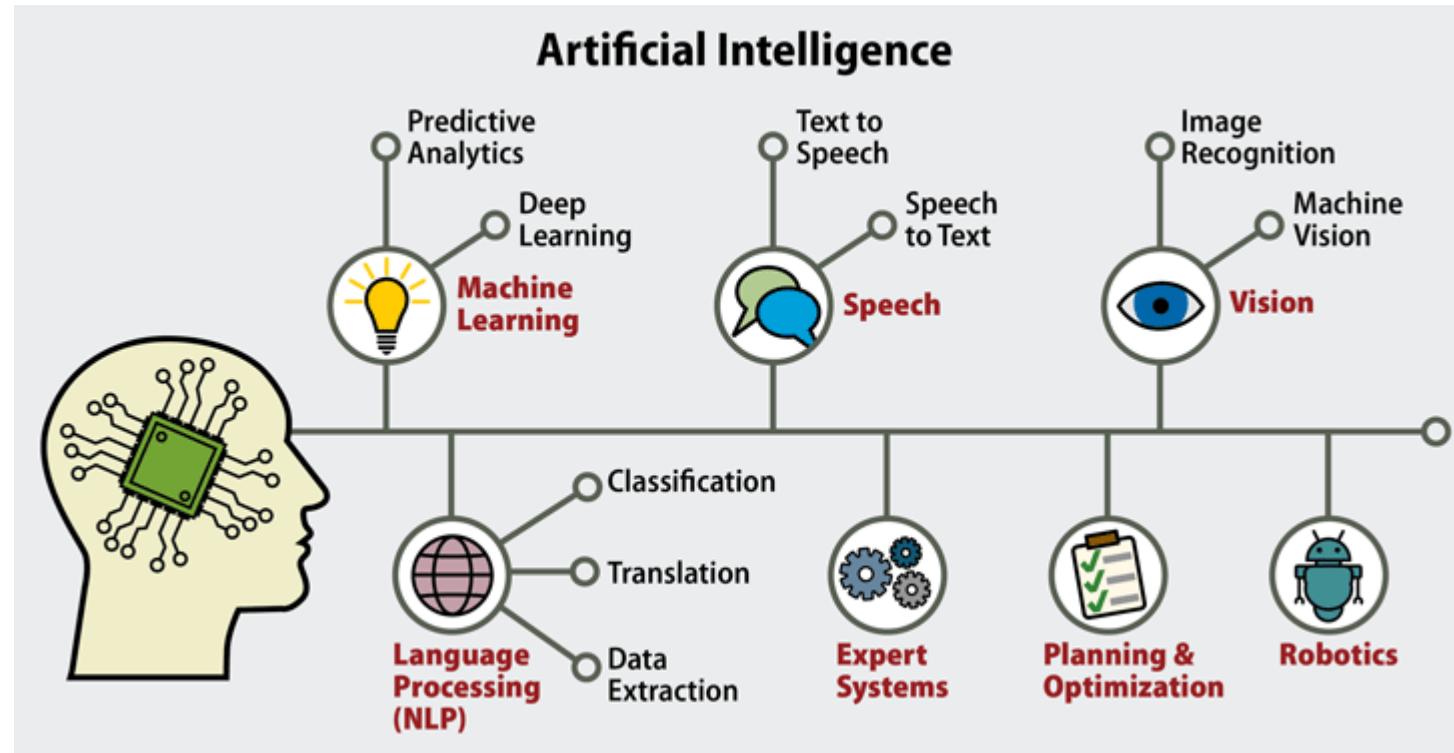


Lambton College
School of Computer Studies

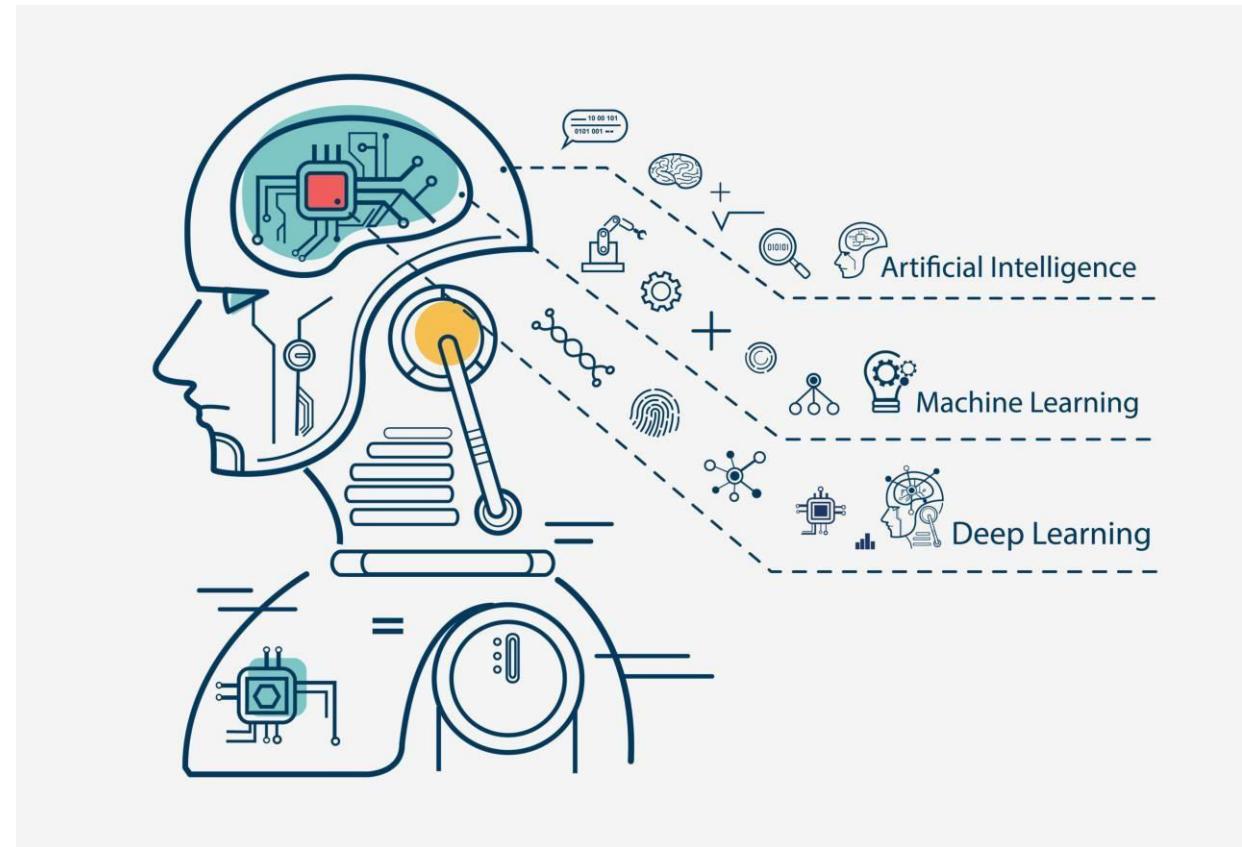
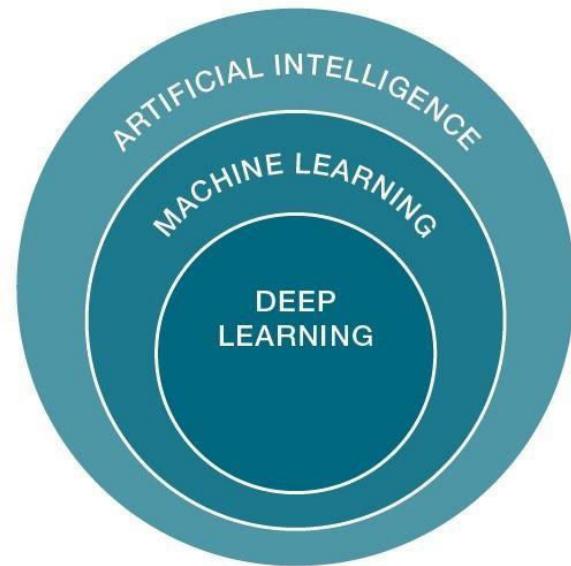
AML-3104 Neural Networks
and Deep Learning

Explore Machine Learning Algorithms

AI and ML



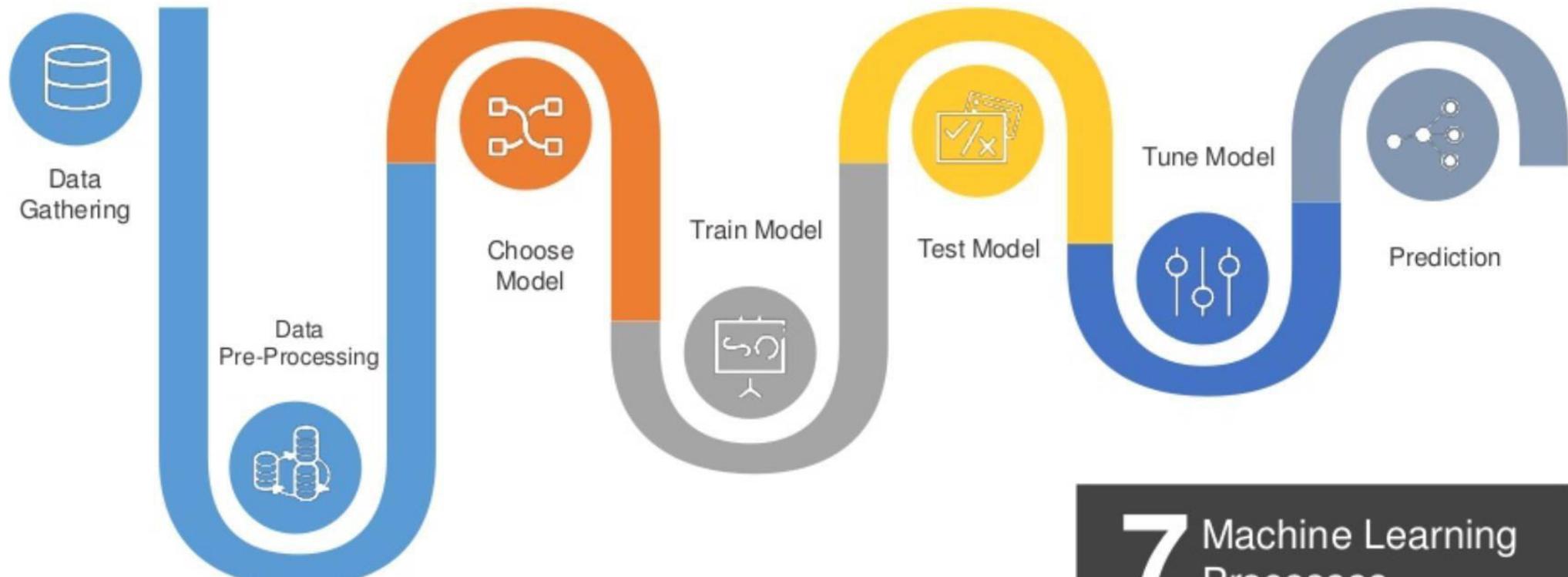
Machine Learning



ML Process

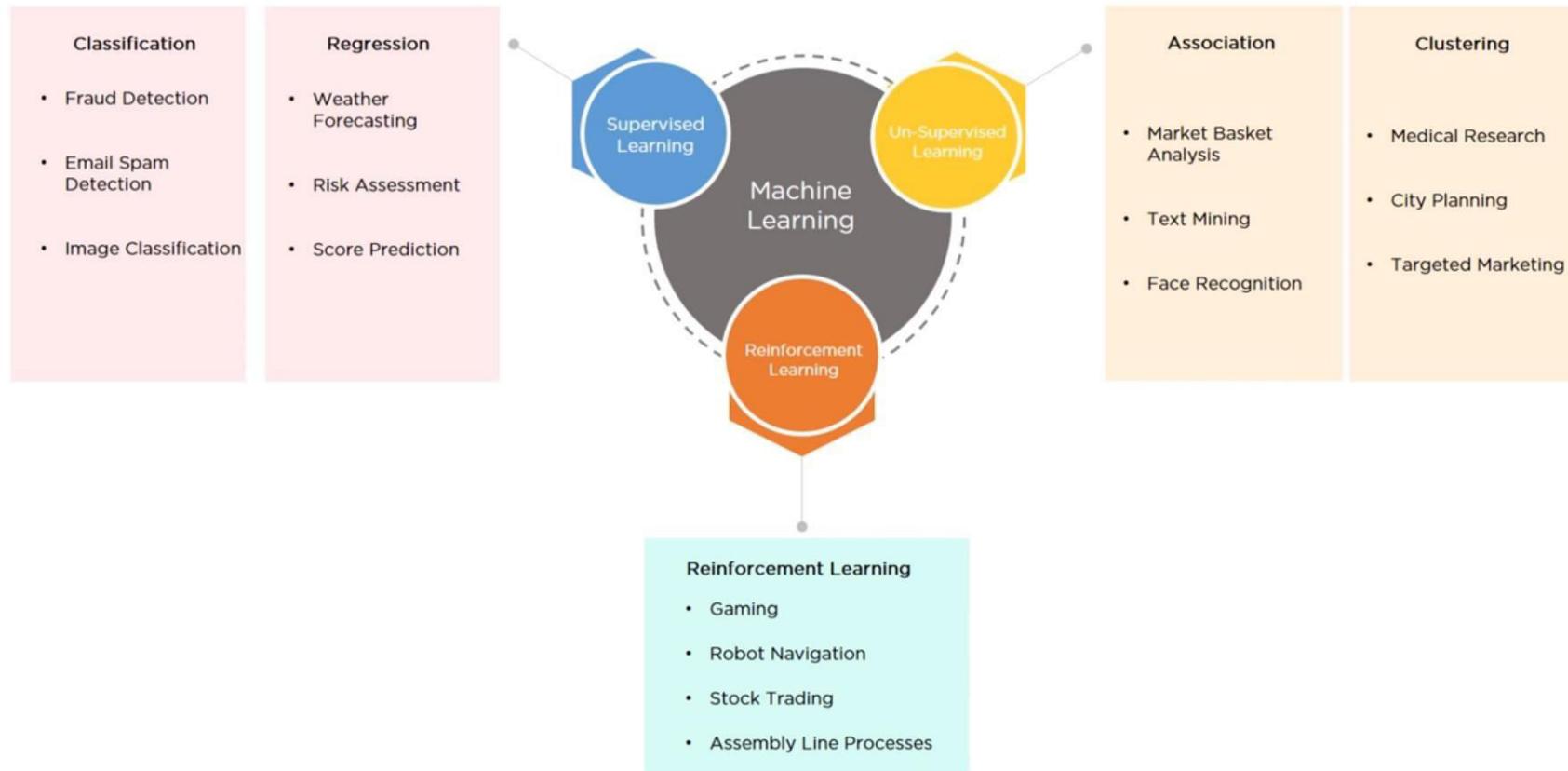


ML Process

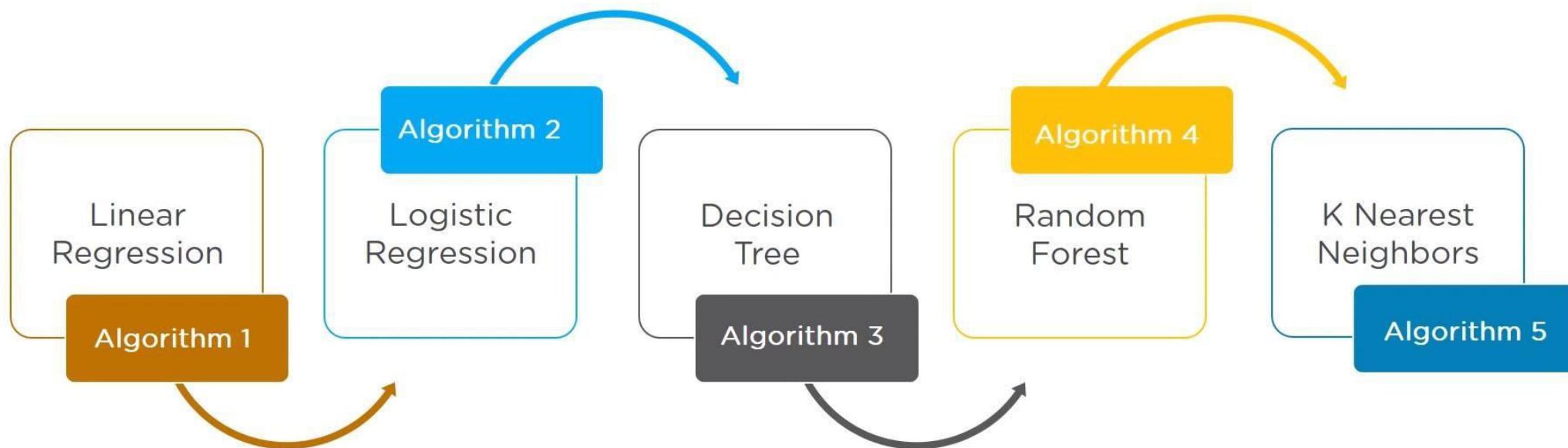


7 Machine Learning
Processes

ML Algorithm Types

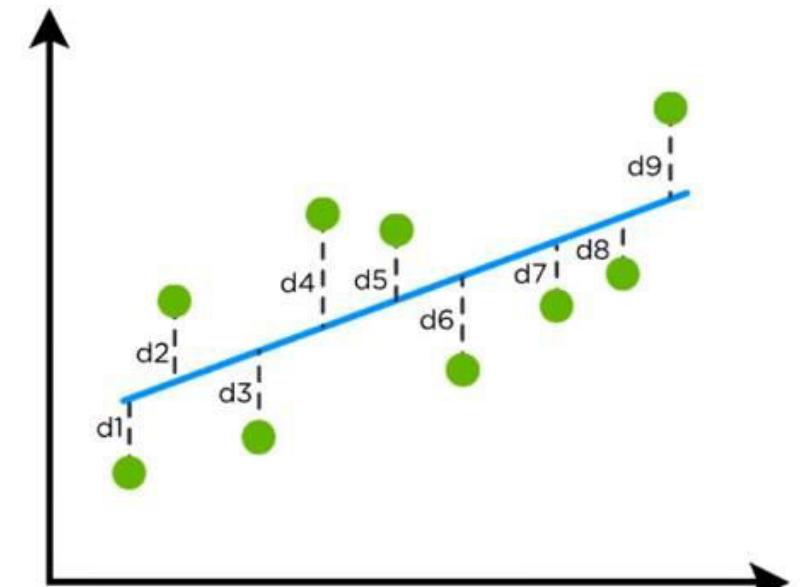
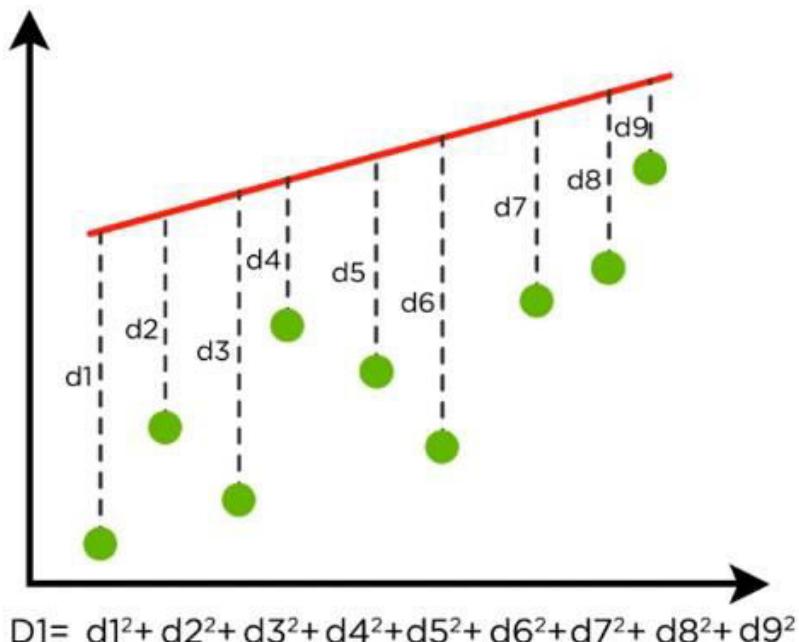


ML Algorithms



Linear Regression

Finding the best fit line: The best fit line can be found out by minimizing the distance between all the data points and the distance to the regression line. Ways to minimize this distance are sum of squared errors, sum of absolute errors etc.

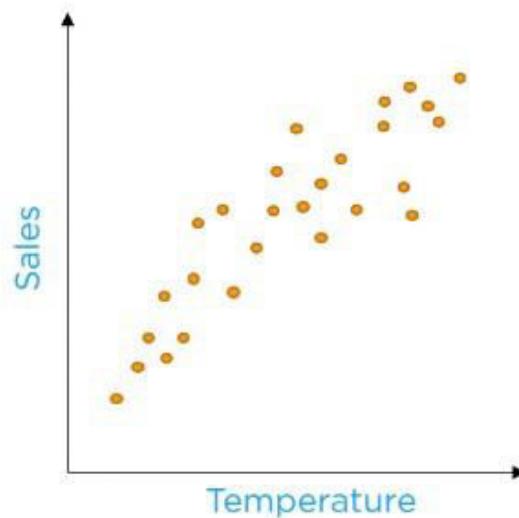


The regression line (blue) has the least value of D

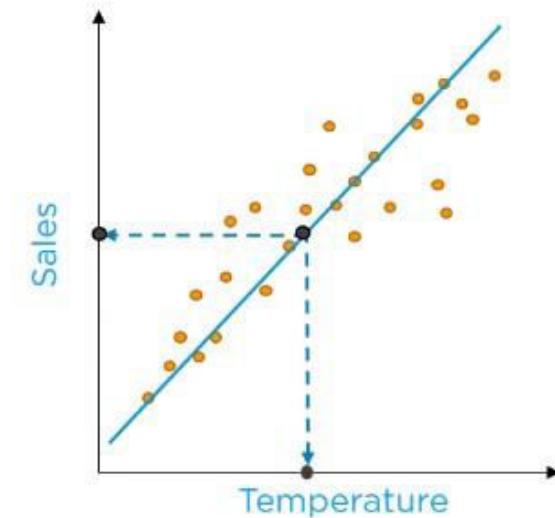
Linear Regression

Linear Regression is a linear modelling approach to find relationship between one or more independent variables (predictors) denoted as X and dependent variable (target) denoted as Y.

Predict sales of Ice Cream based on temperature:



Plotting the sales of ice cream based on temperature



Regression line to predict the sales

Logistic Regression

Logistic Regression is a Classification algorithm used to predict discrete/ categorical values

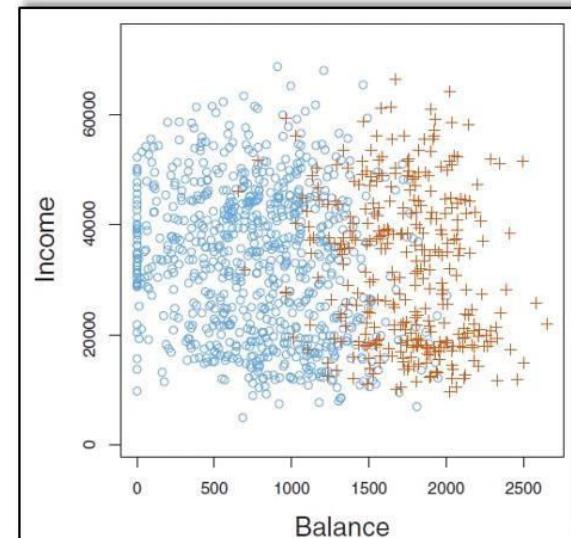
Who will default on their credit card payment?



Credit card users



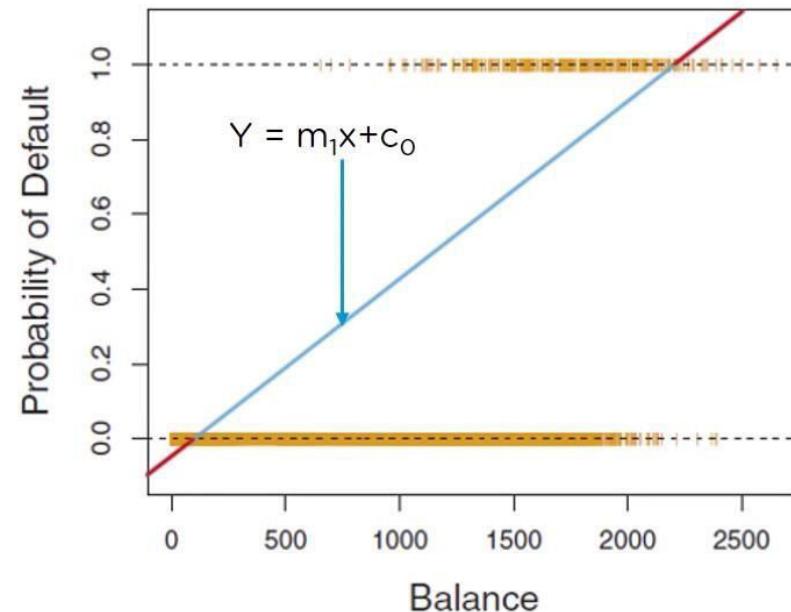
Make credit card transaction



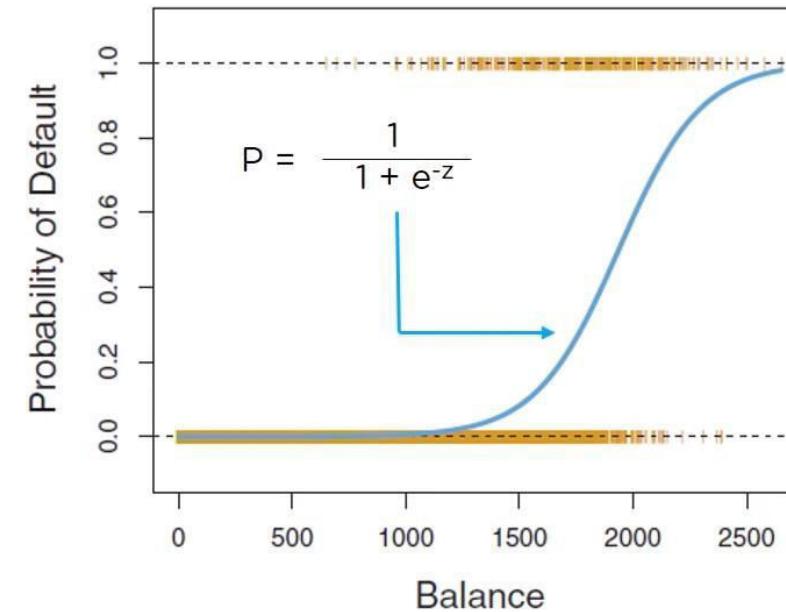
Plot of monthly credit card balance and annual income

Logistic Regression

Plotting the Logistic Regression Curve: The Logistic Regression curve is known as the Sigmoid curve (S curve)



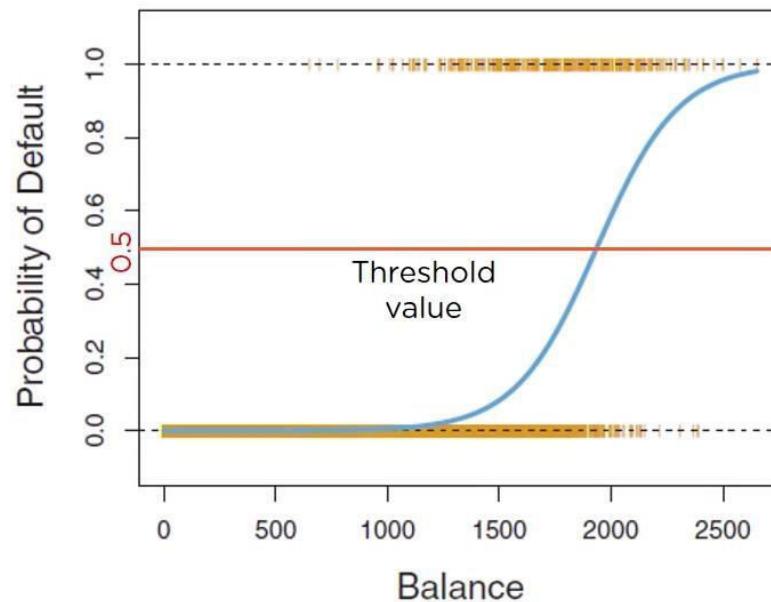
Predicted Y can exceed the range 0 and 1



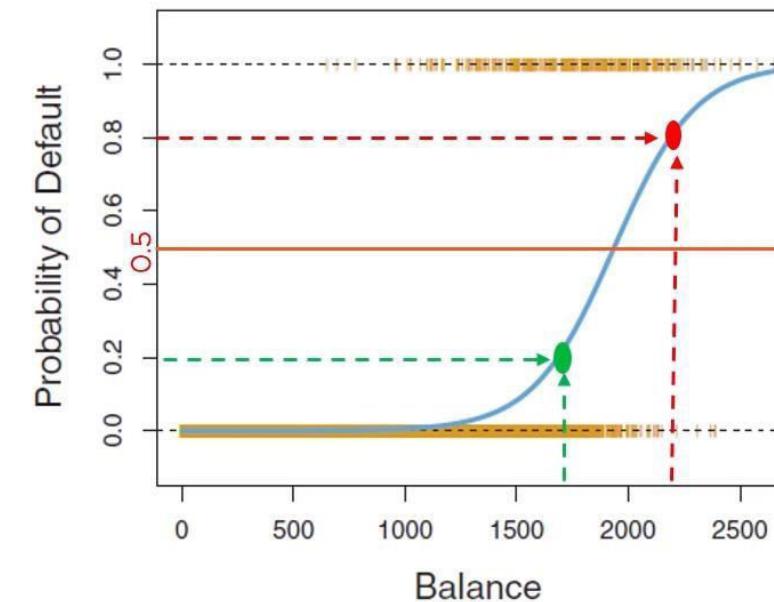
Predicted Y lies in the range 0 and 1

Logistic Regression

Plotting the Logistic Regression Curve: The Logistic Regression curve is known as the Sigmoid curve (S curve)



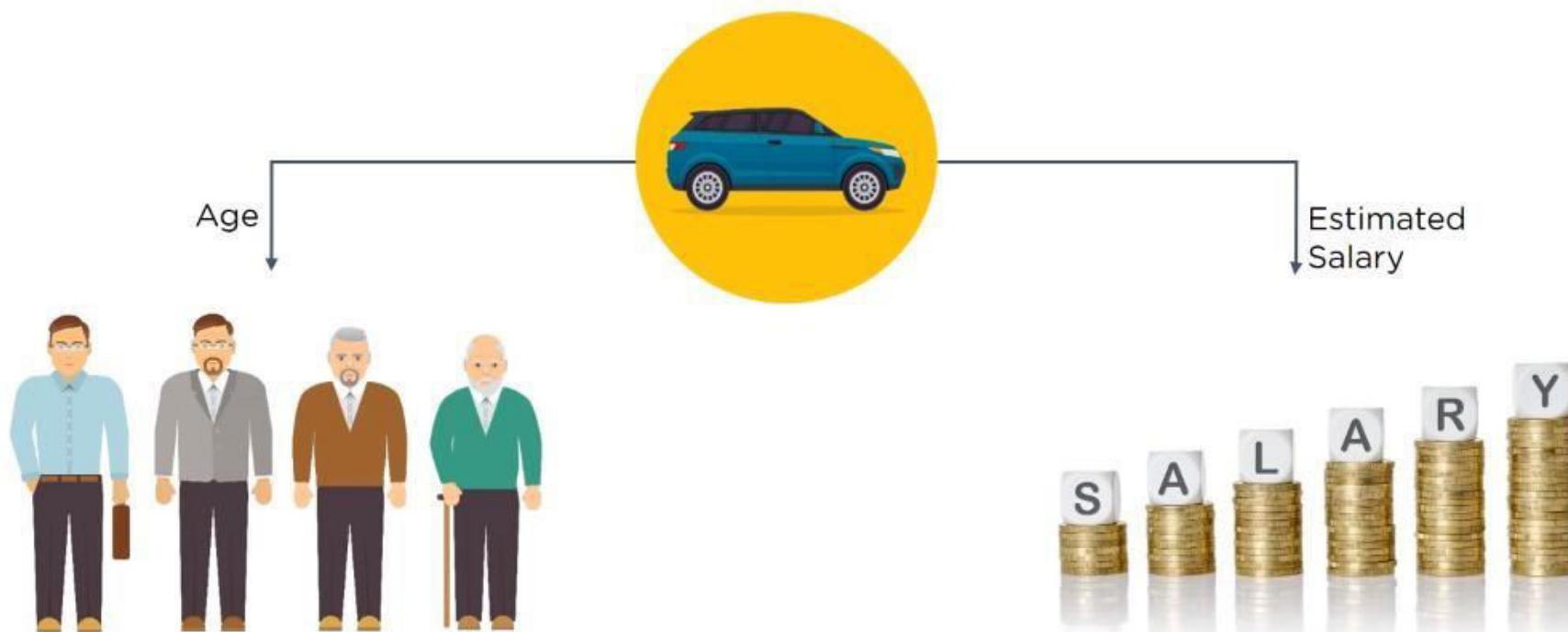
Cutoff point at 0.5, anything below it results in 0 and above is 1



Red data point will default as it is above the threshold value of 0.5 and green data point won't as it is below the threshold value

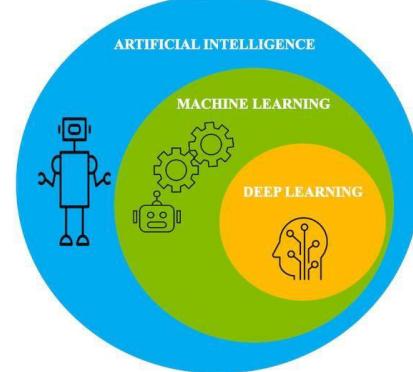
Logistic Regression

Logistic Regression: Predict if a person will buy an SUV based on their Age and Estimated Salary



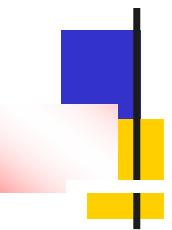
Lambton College

School of Computer Studies



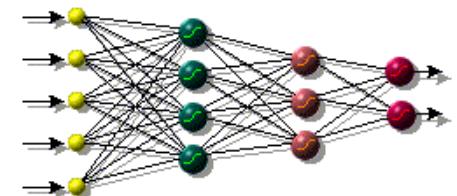
Lambton
College

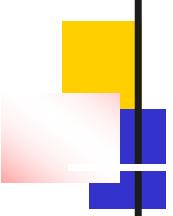
AML-3104 Neural Networks
and Deep Learning



Artificial Neural Networks

- The Brain
- Brain vs. Computers
- The Perceptron
- Multilayer networks
- Some Applications





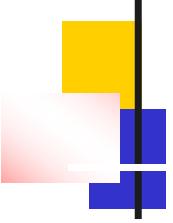
Artificial Neural Networks

- Other terms/names

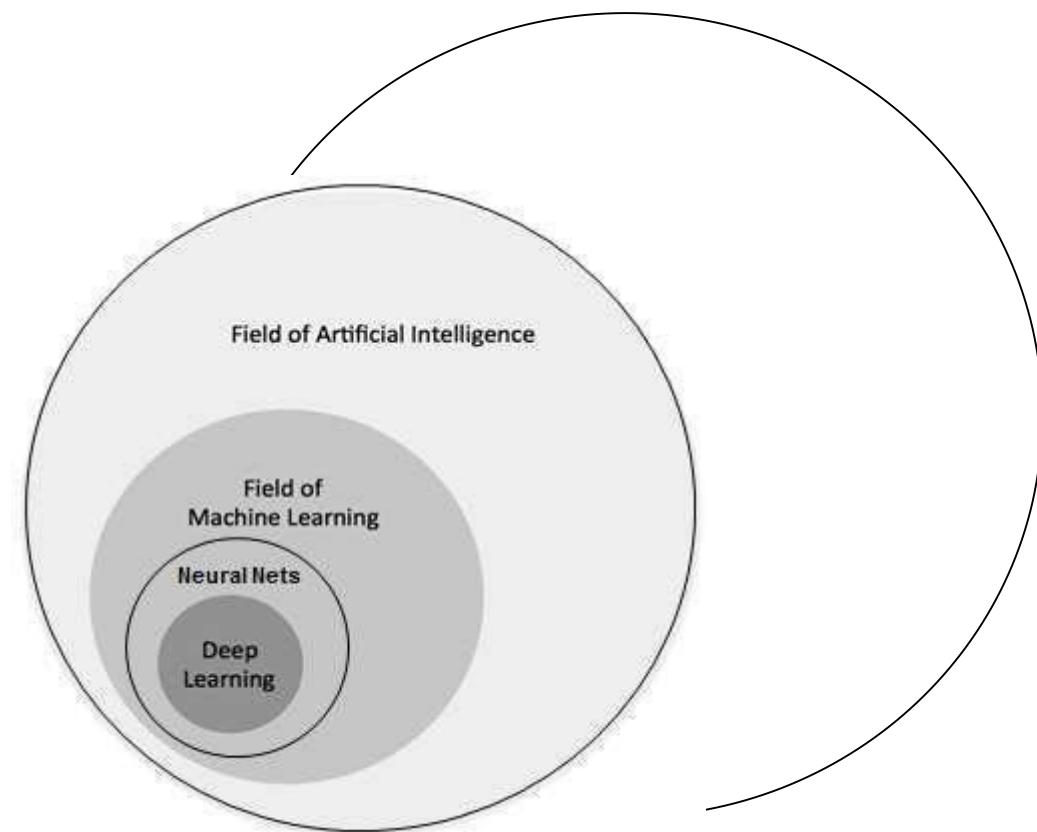
- connectionist
- parallel distributed processing
- neural computation
- adaptive networks..

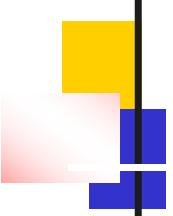
- History

- 1943-McCulloch & Pitts are generally recognised as the designers of the first neural network
- 1949-First learning rule
- 1969-Minsky & Papert - perceptron limitation - Death of ANN
- 1980's - Re-emergence of ANN - multi-layer networks



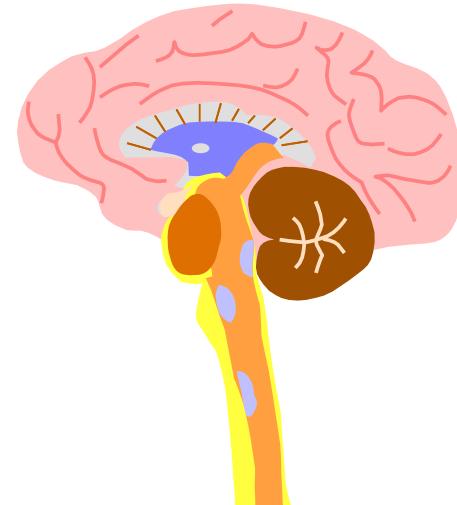
Deep Learning



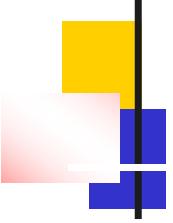


Brain and Machine

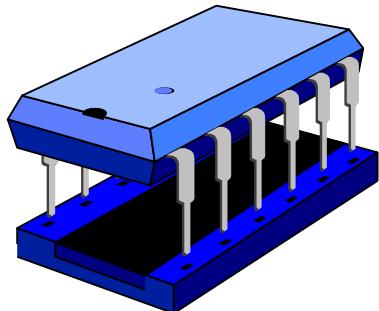
- The Brain
 - ✓ – Pattern Recognition
 - ✓ – Association
 - ✓ – Complexity
 - ✓ – Noise Tolerance



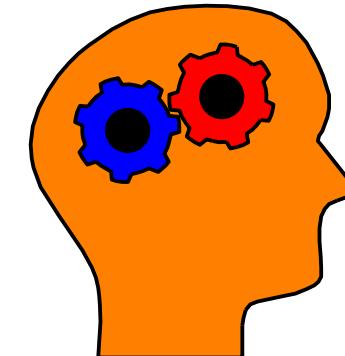
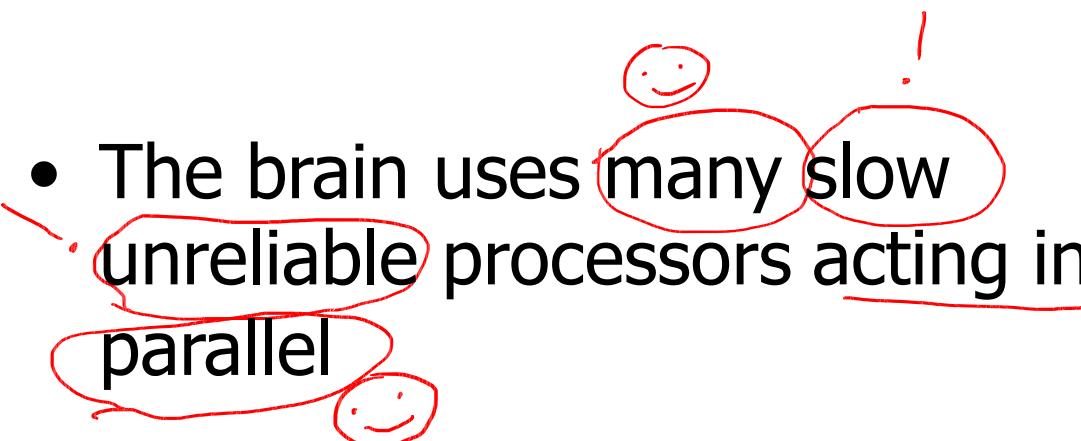
- The Machine
 - ✓ – Calculation
 - ✓ – Precision
 - ✓ – Logic

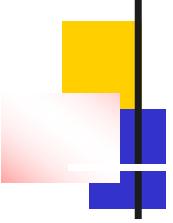


The contrast in architecture



- The Von Neumann architecture uses a single processing unit;
 - ✓ Tens of millions of operations per second
 - ✓ Absolute arithmetic precision



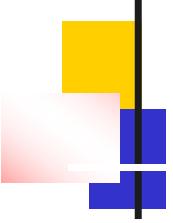


Features of the Brain

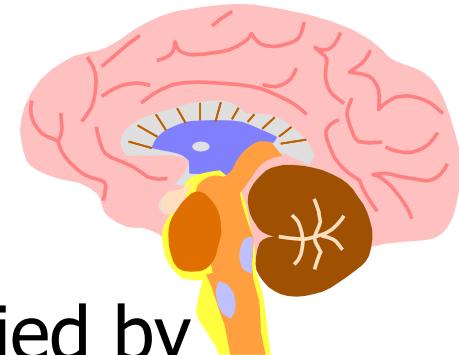


Many
• Ten billion (10^{10}) neurons

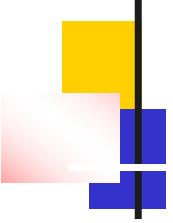
- On average, several thousand connections
- Hundreds of operations per second
- Die off frequently (never replaced)
- Compensates for problems by massive parallelism



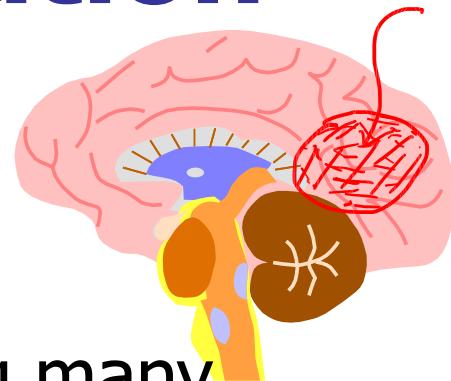
The biological inspiration



- The brain has been extensively studied by scientists.
- Vast complexity prevents all but rudimentary understanding.
- Even the behaviour of an individual neuron is extremely complex

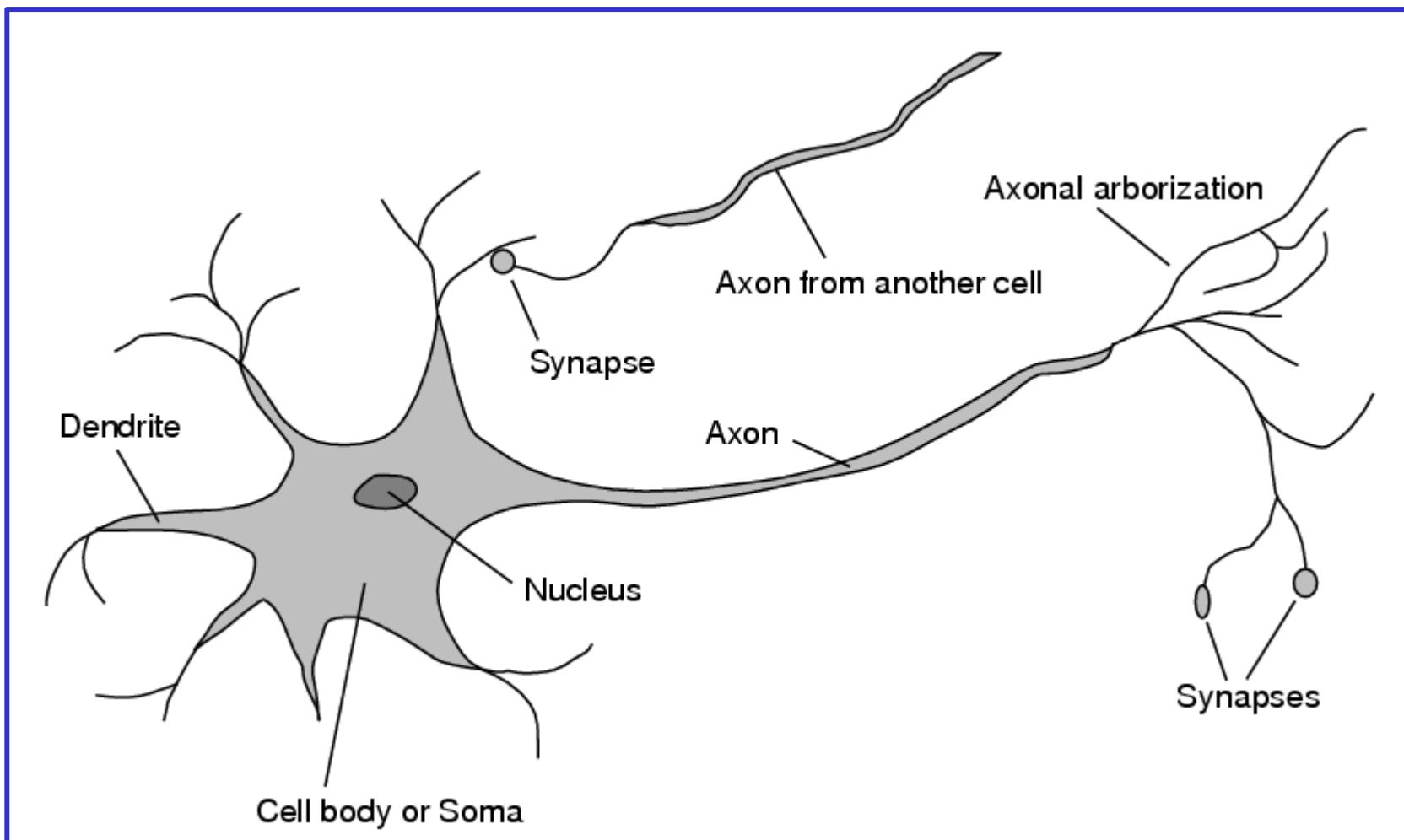


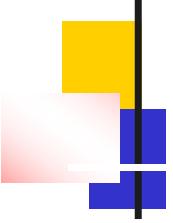
The biological inspiration



- Single “percepts” distributed among many neurons
- Localized parts of the brain are responsible for certain well-defined functions (e.g. vision, motion).

The Structure of Neurons

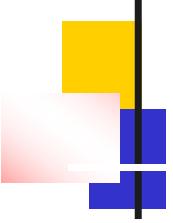




The Structure of Neurons

A neuron has a cell body, a branching **input** structure (the dendr**I**te) and a branching **output** structure (the ax**O**n)

- Axons connect to dendrites via synapses.
- Electro-chemical signals are propagated from the dendritic input, through the cell body, and down the axon to other neurons

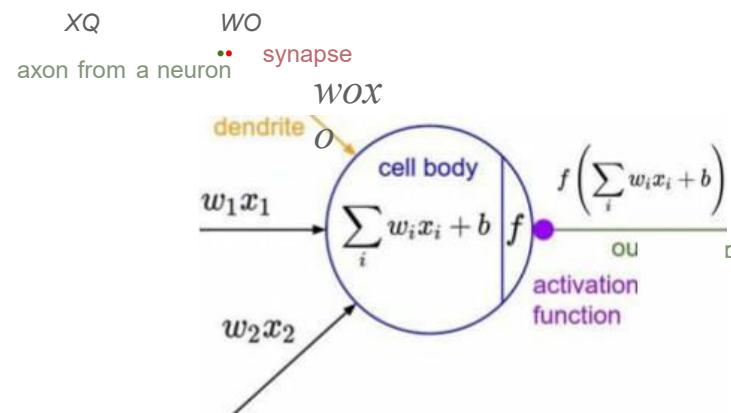
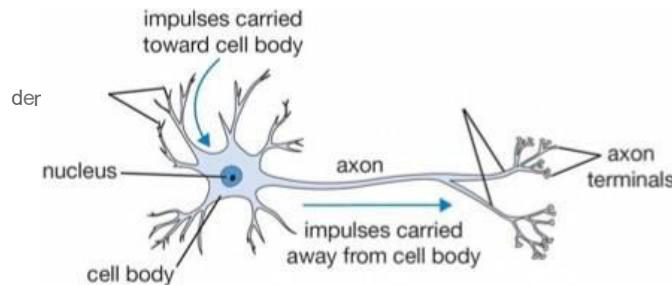


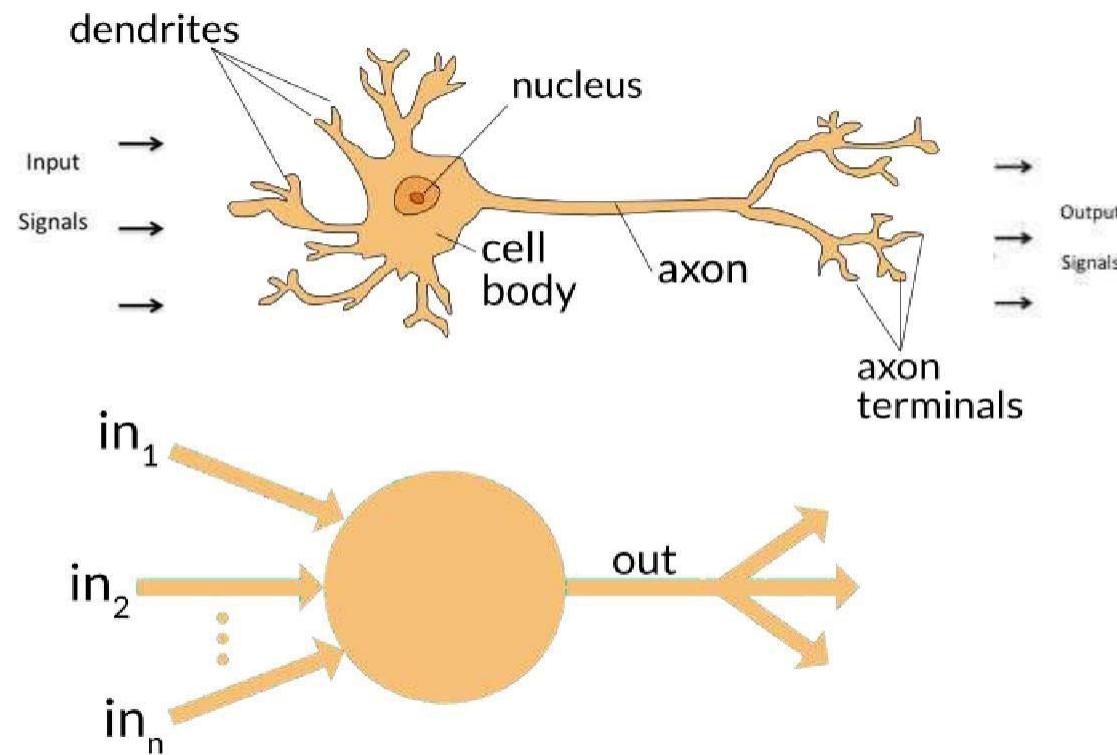
The Structure of Neurons

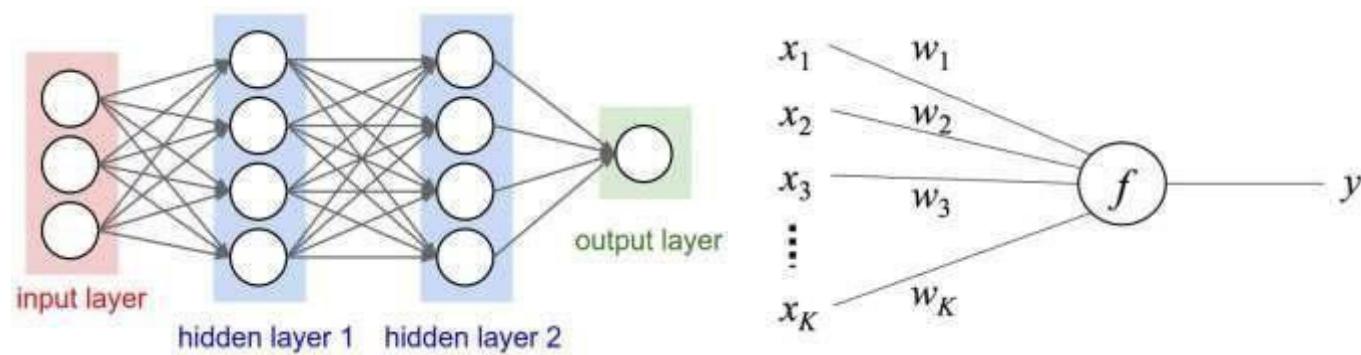
- A neuron only fires if its input signal exceeds a certain amount (the **threshold**) in a short time period.
- Synapses vary in strength
 - Good connections allowing a large signal
 - Slight connections allow only a weak signal.

Inspiration: Neuron cells

- Neurons
 - accept information from multiple inputs
 - transmit information to other neurons
- Multiply inputs by weights along edges
- Apply some function to the set of inputs at each node
- If output of function over threshold, neuron "fires"

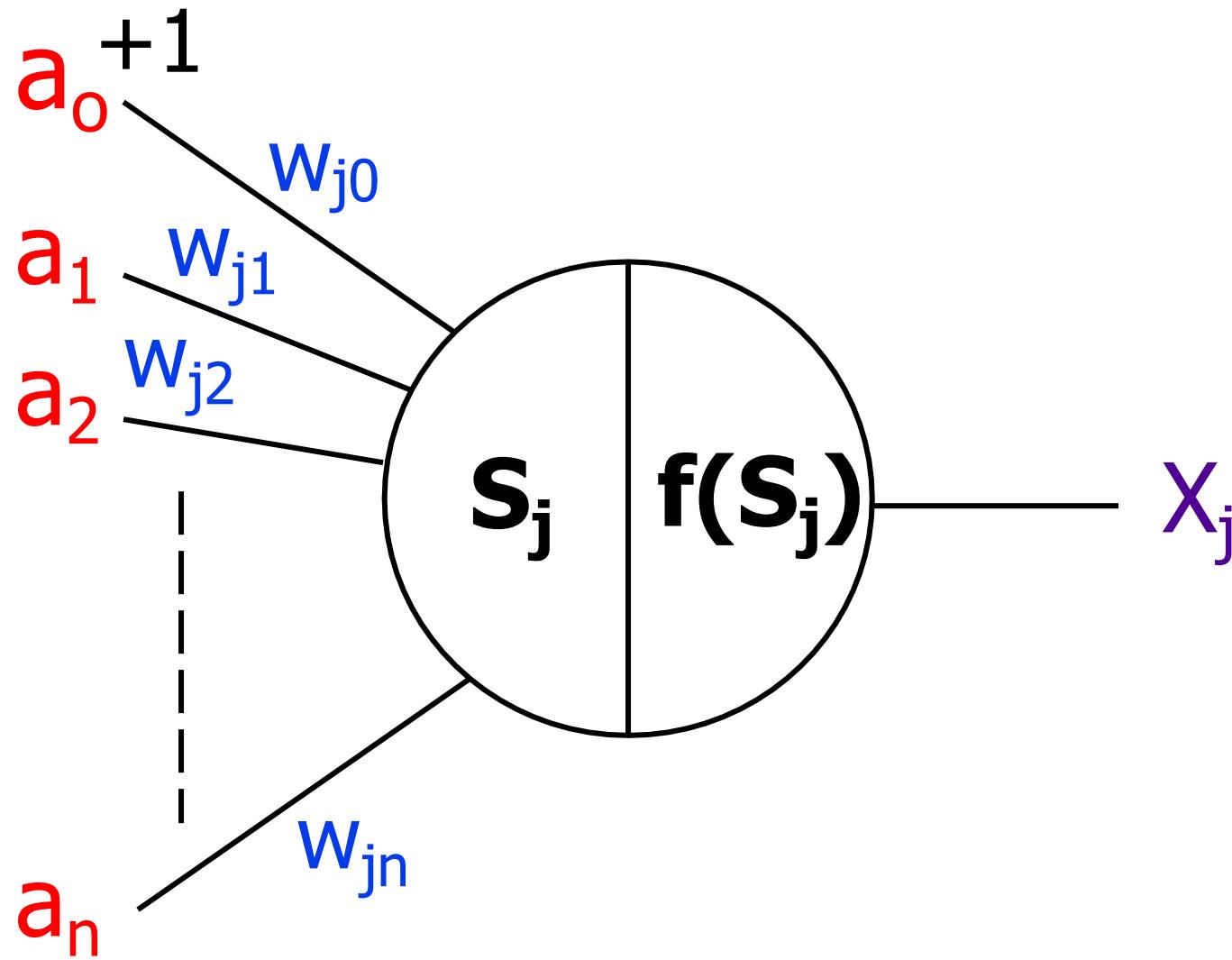




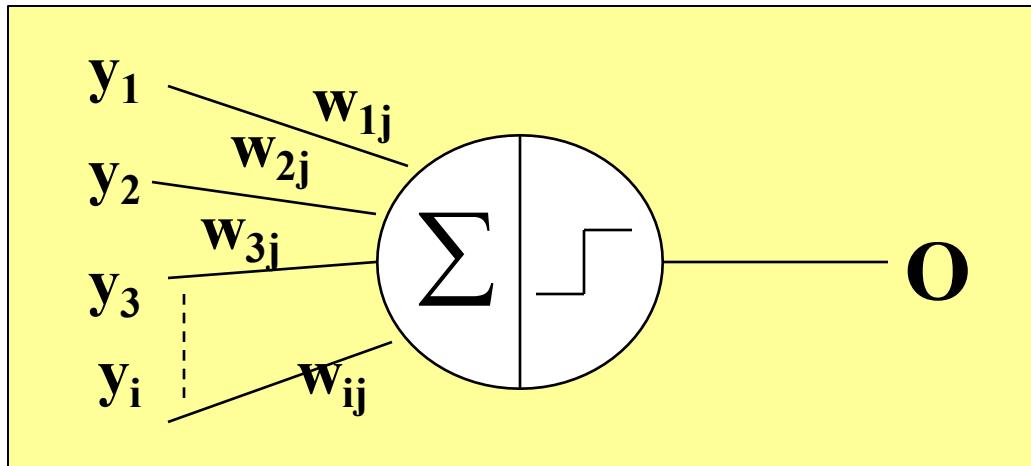




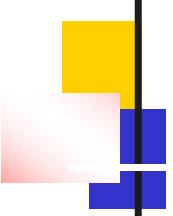
The Artificial Neuron (Perceptron)



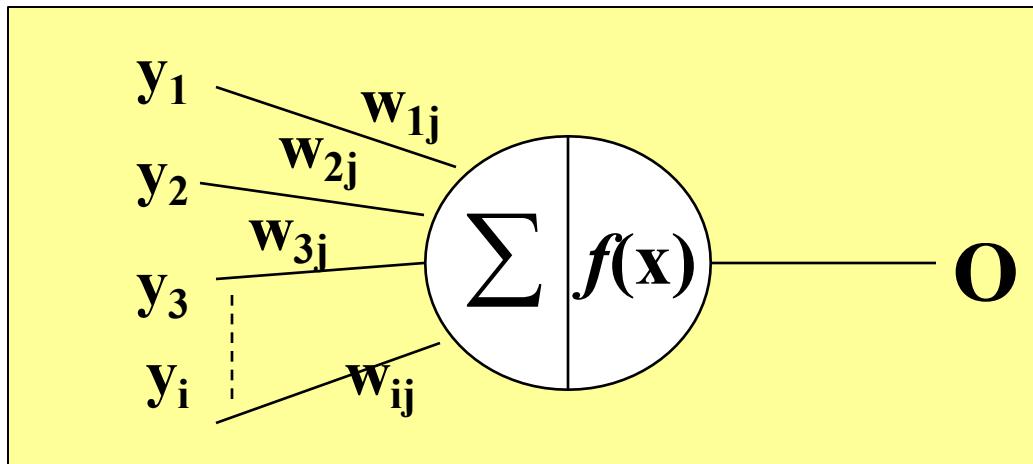
A Simple Model of a Neuron (Perceptron)



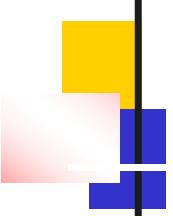
- Each neuron has a threshold value
- Each neuron has weighted inputs from other neurons
- The input signals form a weighted sum
- If the activation level exceeds the threshold, the neuron “fires”



An Artificial Neuron

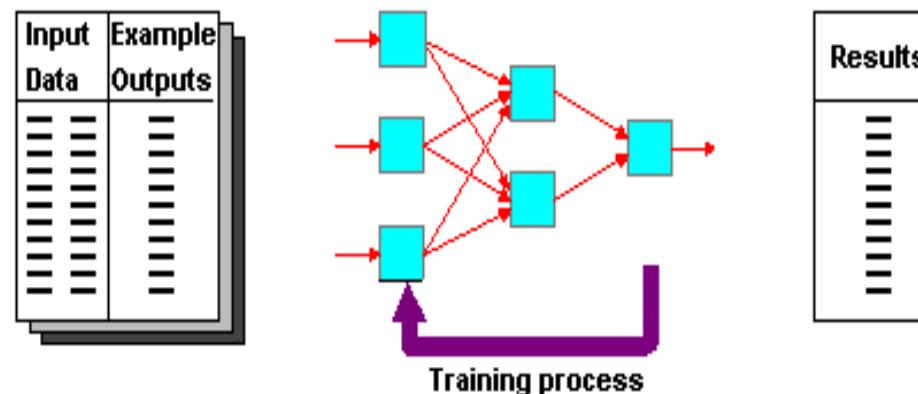


- Each hidden or output neuron has weighted input connections from each of the units in the preceding layer.
- The unit performs a weighted sum of its inputs, and subtracts its threshold value, to give its activation level.
- Activation level is passed through a sigmoid activation function to determine output.

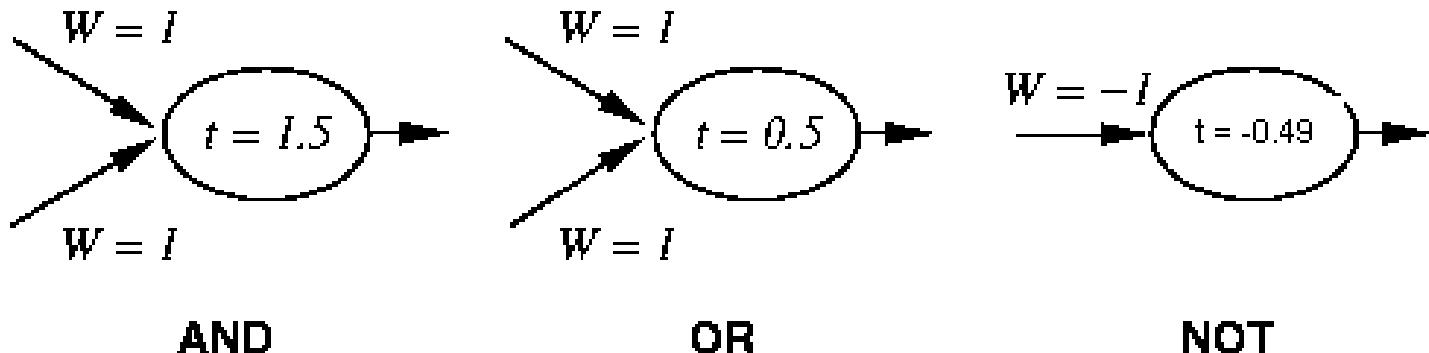


Supervised Learning

- Training and test data sets
- Training set; input & target

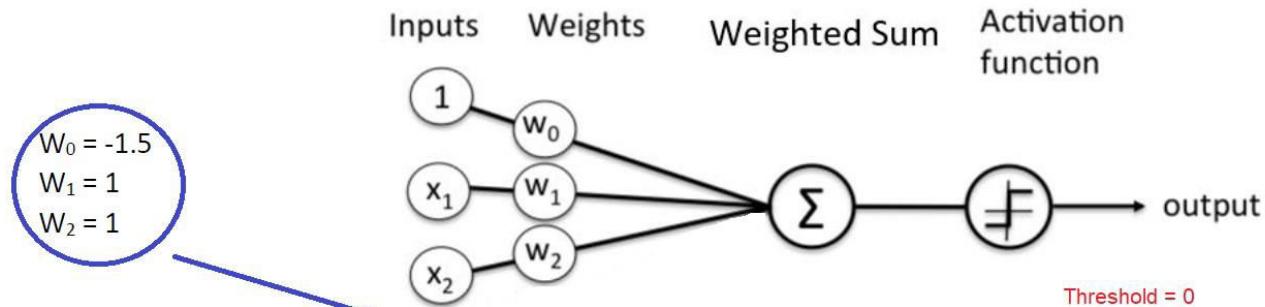


Perceptron Training



$$\text{Output} = \begin{cases} 1 & \text{if } \sum_{i=0} w_i x_i > t \\ 0 & \text{otherwise} \end{cases}$$

- Linear threshold is used.
- W - weight value
- t - threshold value



Threshold = 0

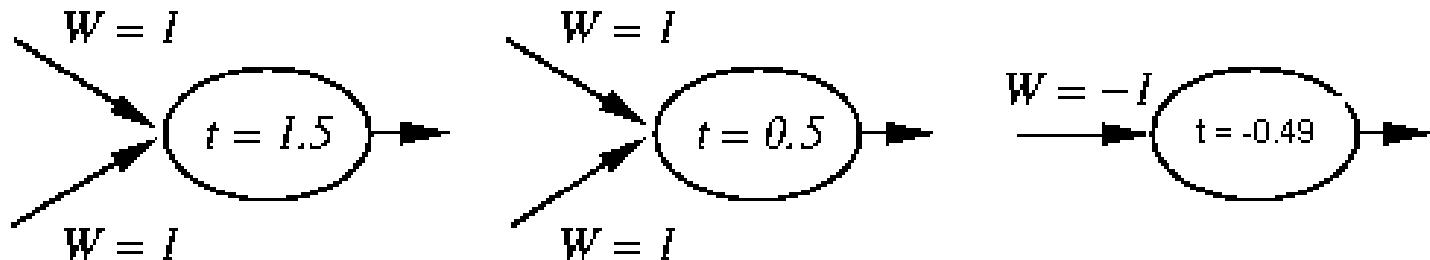
If Weighted Sum > 0, 1
If Weighted Sum < 0, 0

AND

Input-1 (X1)	Input-2 (X2)	Output
0	0	0
0	1	0
1	0	0
1	1	1

X1	X2	Weighted Sum $W_0 + X_1 W_1 + X_2 W_2 =$	Output
0	0	$-1.5 + 0 \cdot 1 + 0 \cdot 1 = -1.5$	0
0	1	$-1.5 + 0 + 1 = -0.5$	0
1	0	$-1.5 + 1 + 0 = -0.5$	0
1	1	$-1.5 + 1 + 1 = 0.5$	1

Simple network



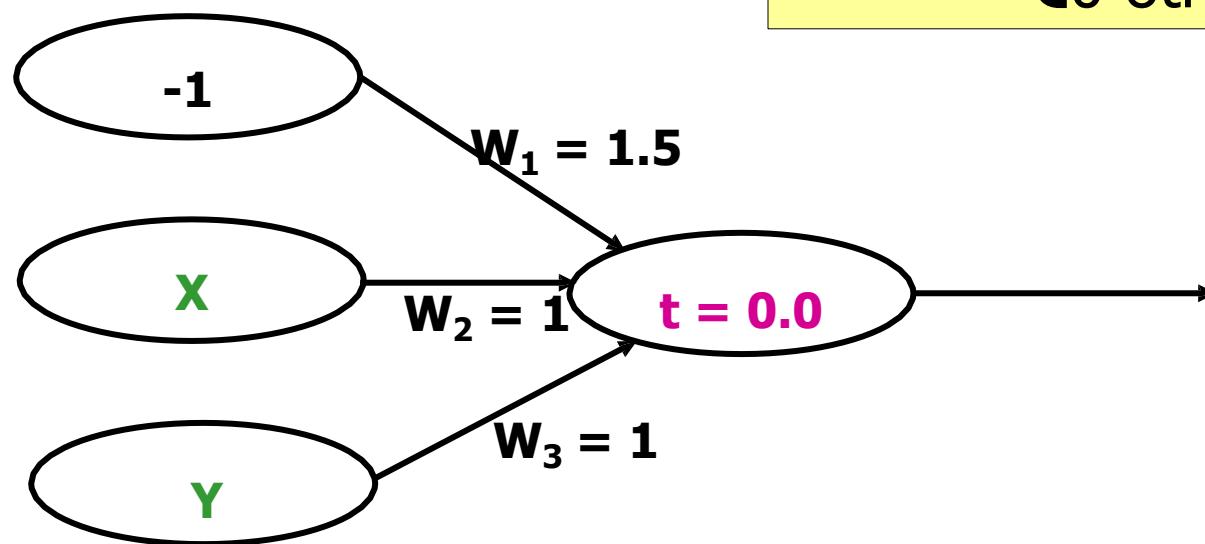
AND

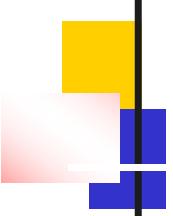
OR

NOT

AND with a Biased input

$$\text{output} = \begin{cases} 1 & \text{if } \sum_{i=0} w_i x_i > t \\ 0 & \text{otherwise} \end{cases}$$





Learning algorithm

While epoch produces an error

Present network with next inputs from epoch

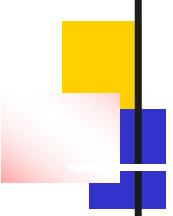
$$\text{Error} = T - O$$

If Error $\neq 0$ then

$$W_j = W_j + LR * I_j * \text{Error}$$

End If

End While

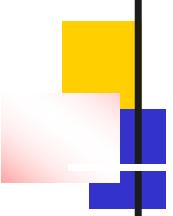


Learning algorithm

Epoch: Presentation of the entire training set to the neural network.

In the case of the AND function an epoch consists of four sets of inputs being presented to the network (i.e. [0,0], [0,1], [1,0], [1,1])

Error: The error value is the amount by which the value output by the network differs from the target value. For example, if we required the network to output 0 and it output a 1, then Error = -1



Learning algorithm

Target Value, T : When we are training a network we not only present it with the input but also with a value that we require the network to produce. For example, if we present the network with [1,1] for the AND function the target value will be 1

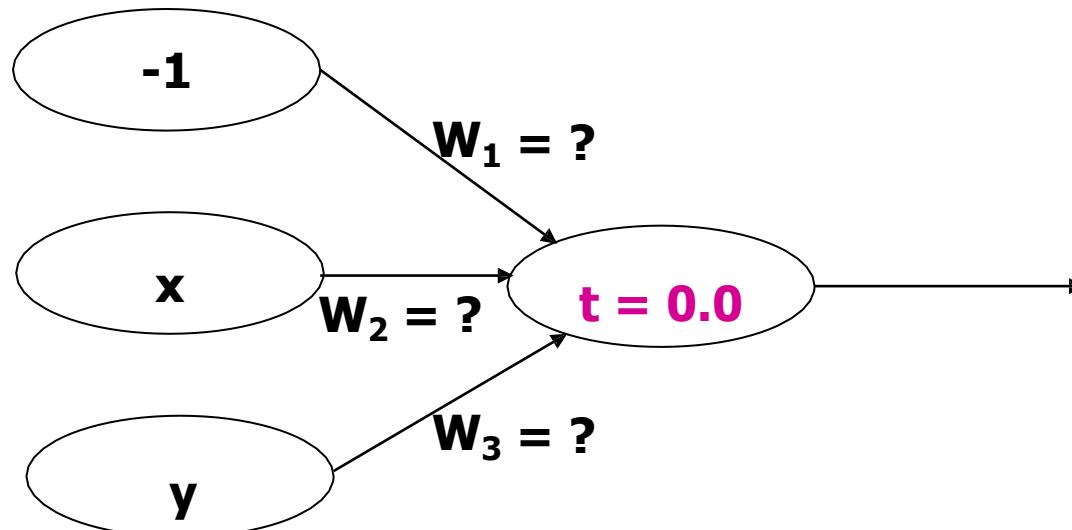
Output , Q : The output value from the neuron

I_j: Inputs being presented to the neuron

W_j : Weight from input neuron (I_j) to the output neuron

LR: The learning rate. This dictates how quickly the network converges. It is set by a matter of experimentation. It is typically 0.1

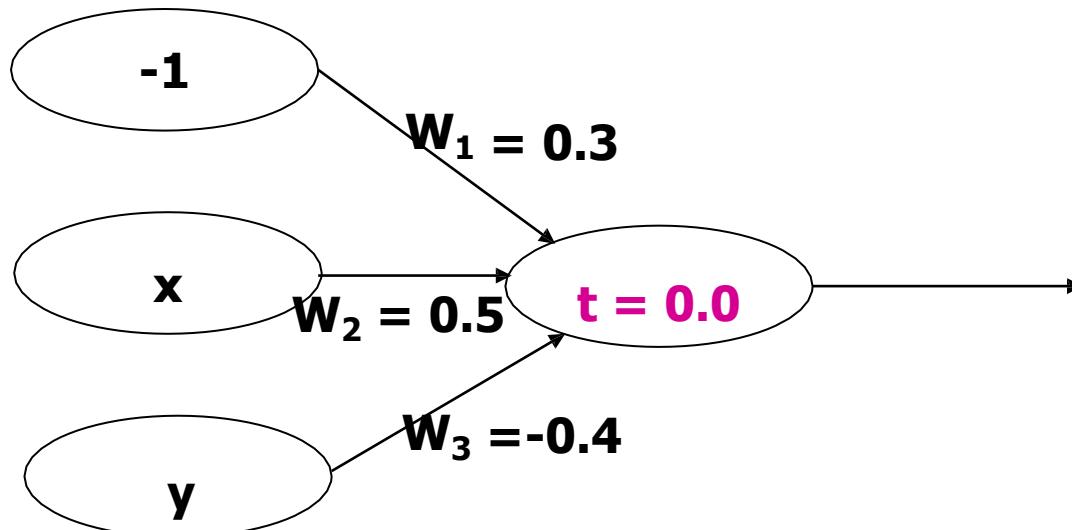
Training Perceptrons



For AND		
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

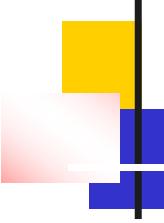
- What are the weight values?
- Initialize with random weight values

Training Perceptrons



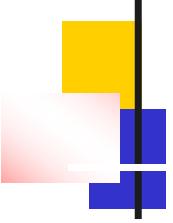
For AND		
A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1

I ₁	I ₂	I ₃	Summation	Output
-1	0	0	(-1*0.3) + (0*0.5) + (0*-0.4) = -0.3	0
-1	0	1	(-1*0.3) + (0*0.5) + (1*-0.4) = -0.7	0
-1	1	0	(-1*0.3) + (1*0.5) + (0*-0.4) = 0.2	1
-1	1	1	(-1*0.3) + (1*0.5) + (1*-0.4) = -0.2	0



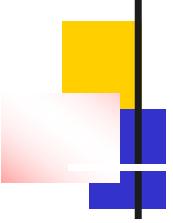
Learning in Neural Networks

- Learn values of weights from I/O pairs
- Start with random weights
- Load training example's input
- Observe computed input
- Modify weights to reduce difference
- Iterate over all training examples
- Terminate when weights stop changing OR when error is very small

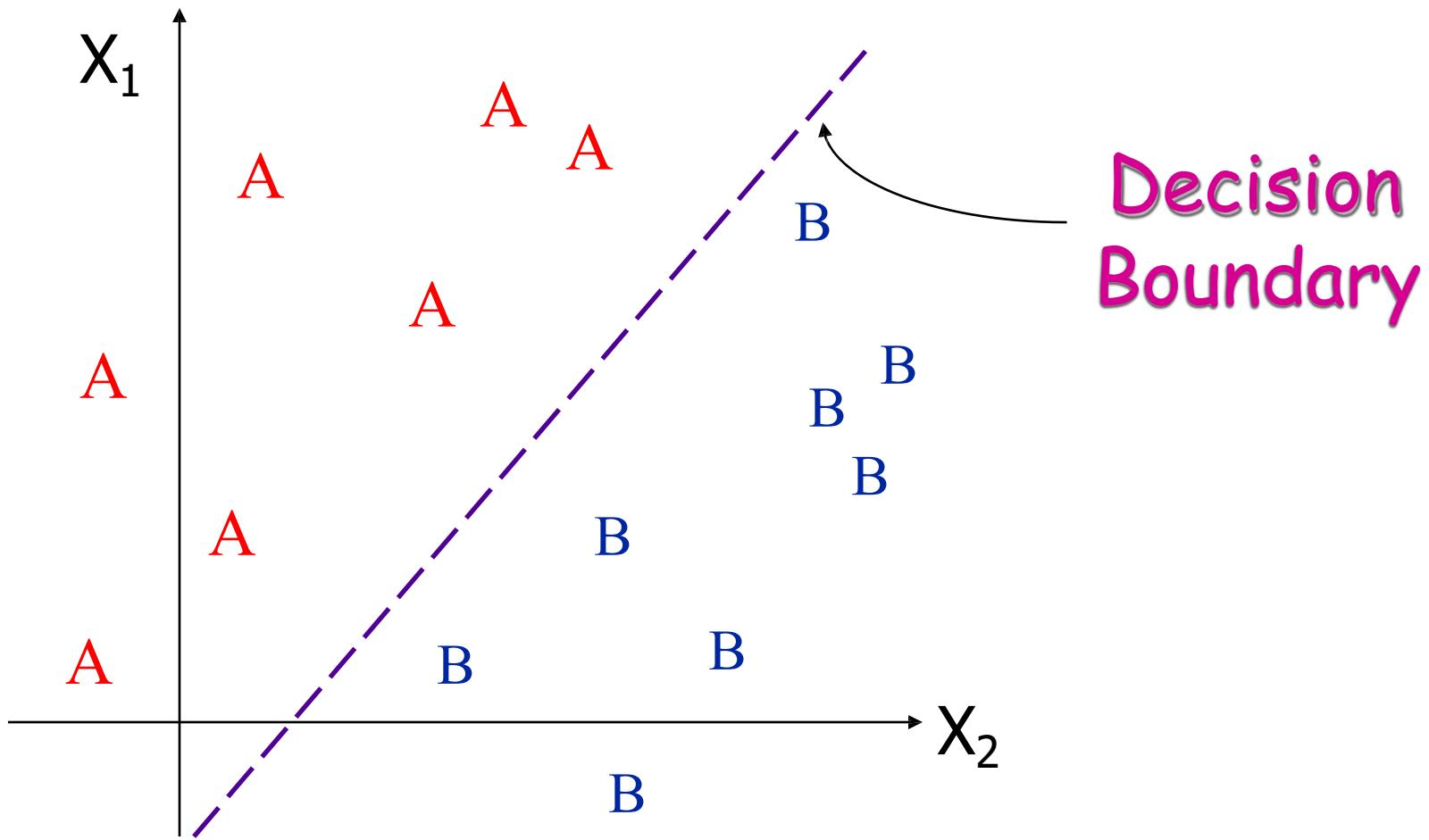


Decision boundaries

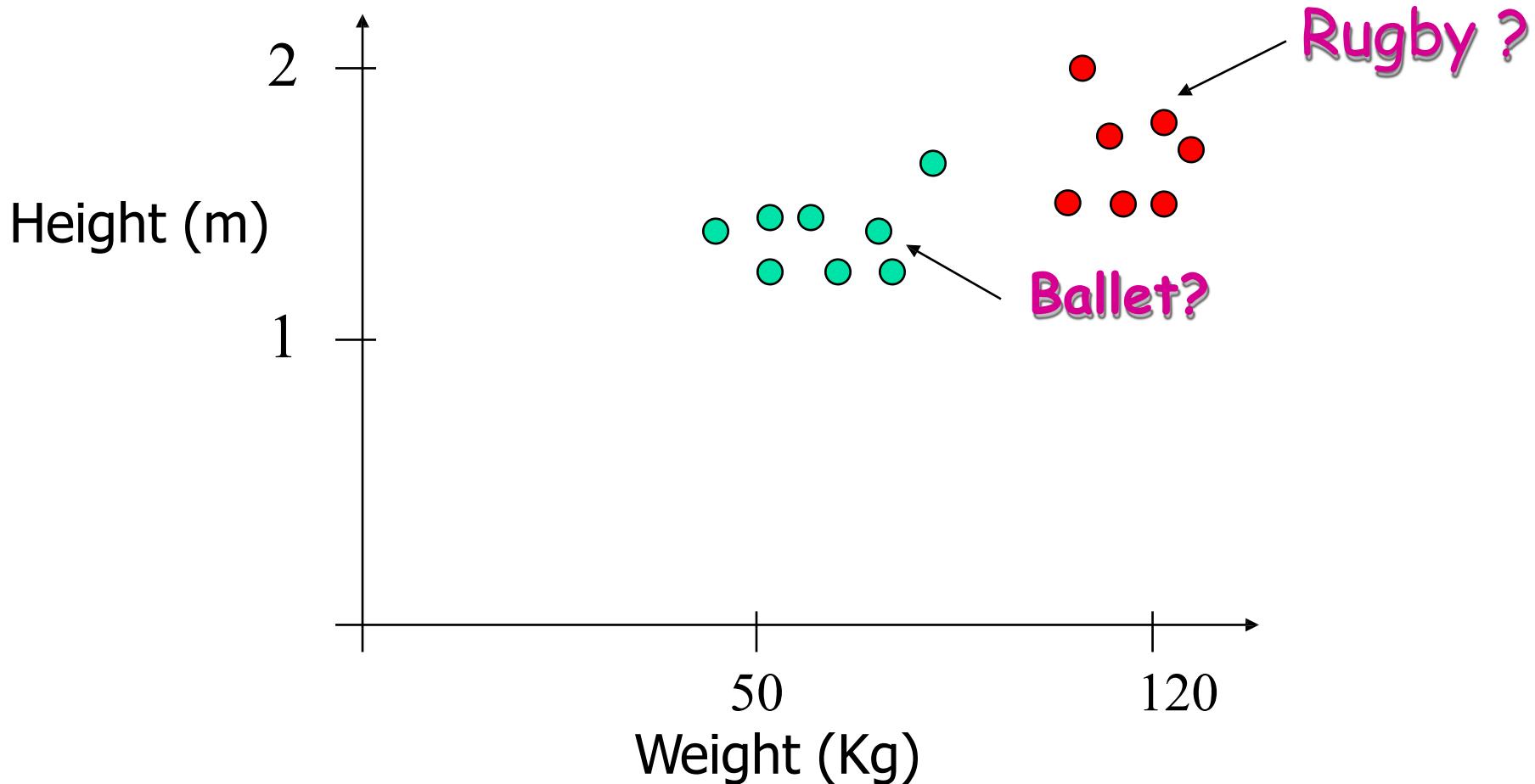
- In simple cases, divide feature space by drawing a hyperplane across it.
- Known as a **decision boundary**.
- **Discriminant function**: returns different values on opposite sides. (straight line)
- Problems which can be thus classified are **linearly separable**.

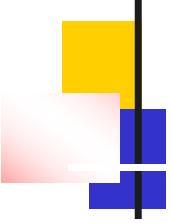


Linear Separability



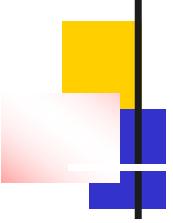
Rugby players & Ballet dancers





Hyperplane partitions

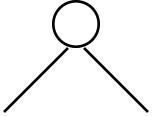
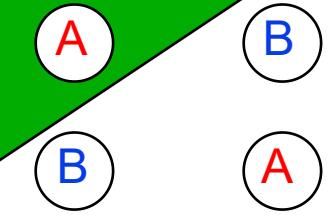
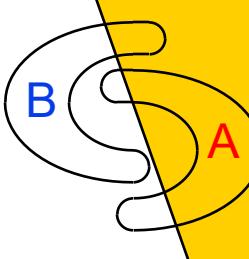
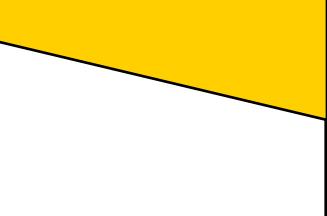
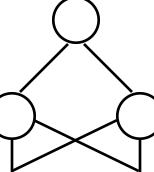
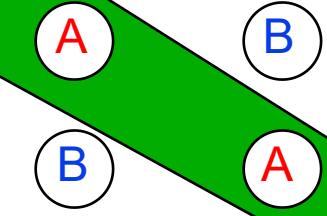
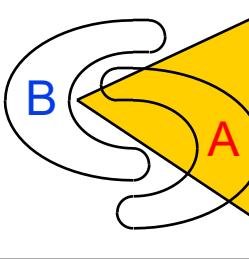
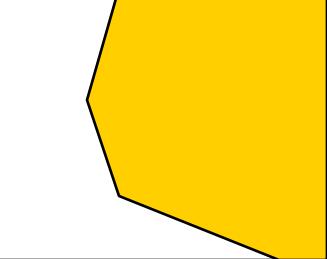
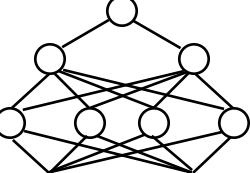
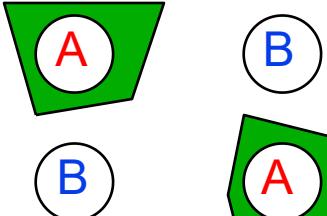
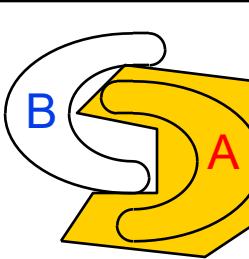
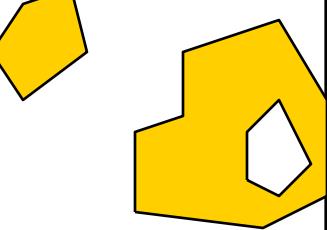
- A single Perceptron (i.e. output unit) with connections from each input can perform, and **learn**, a linear separation.
- Perceptrons have a step function activation.

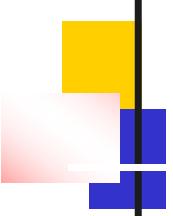


Hyperplane partitions

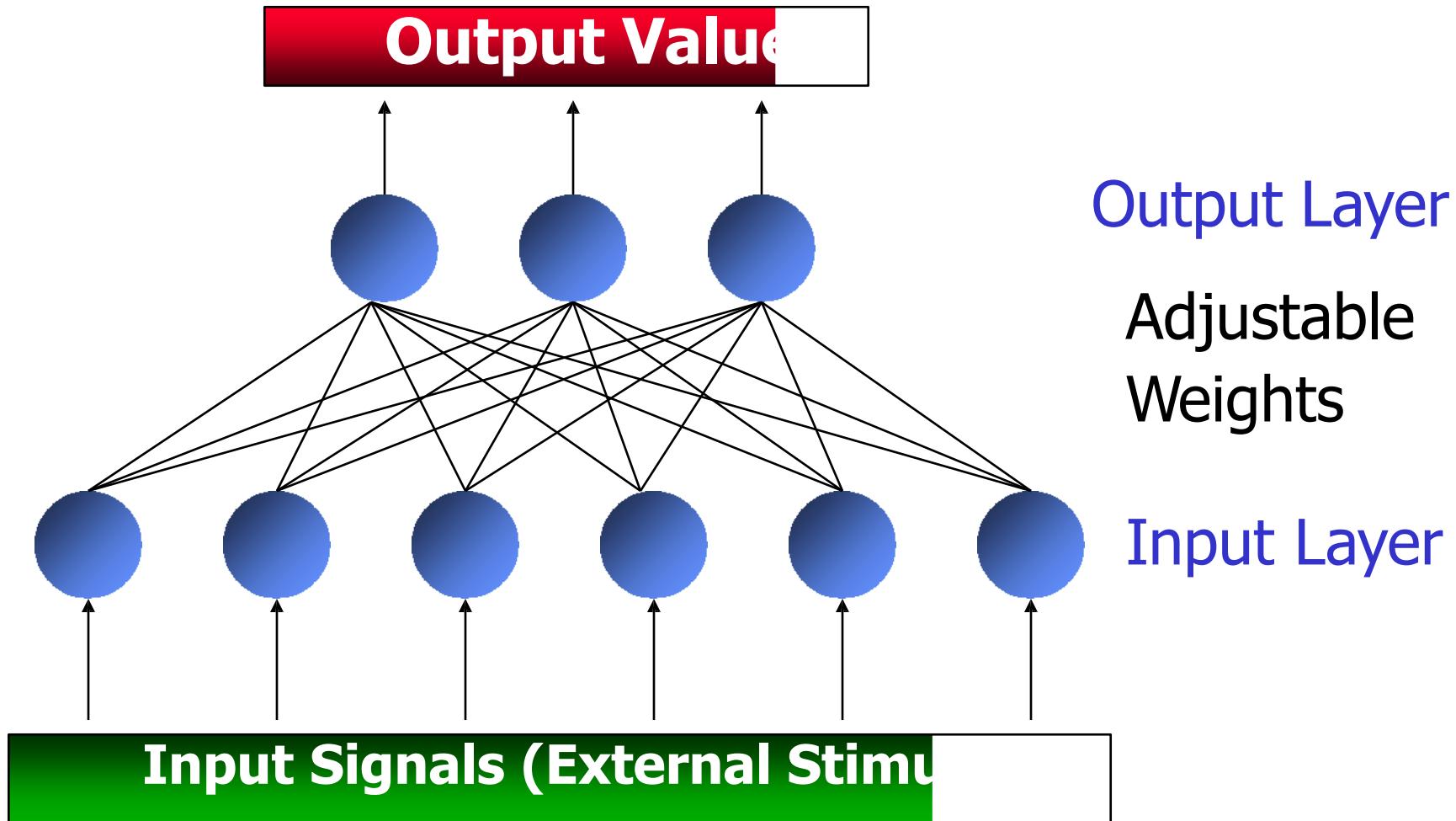
- An extra layer models a convex hull
 - “An area with no dents in it”
 - Perceptron models, but can’t learn
 - Sigmoid function learning of convex hulls
 - Two layers add convex hulls together
 - Sufficient to classify anything “sane”.
- In theory, further layers add nothing
- In practice, extra layers may be better

Different Non-Linearly Separable Problems

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyperplane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			



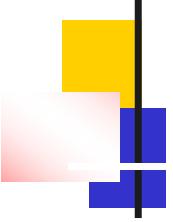
Multilayer Perceptron (MLP)



Output Layer

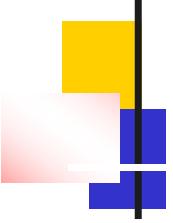
Adjustable
Weights

Input Layer



Types of Layers

- The input layer.
 - Introduces input values into the network.
 - No activation function or other processing.
- The hidden layer(s).
 - Perform classification of features
 - Two hidden layers are sufficient to solve any problem
 - Features imply more layers may be better
- The output layer.
 - Functionally just like the hidden layers
 - Outputs are passed on to the world outside the neural network.



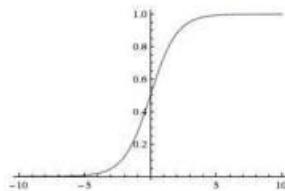
Activation functions

- Transforms neuron's input into output.
- Features of activation functions:
 - A squashing effect is required
 - Prevents accelerating growth of activation levels through the network.
 - Simple and easy to calculate

Activation functions

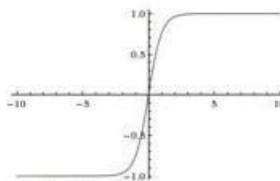
Sigmoid

$$\sigma(x) = 1/(1 + e^{-x})$$

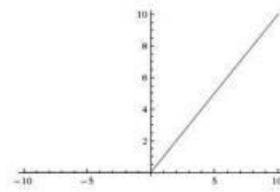


tanh $\tanh(x)$

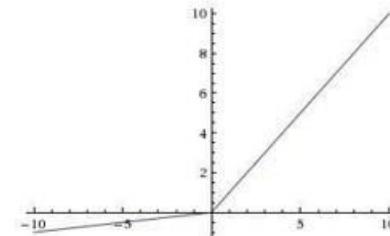
$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



ReLU $\max(0, x)$



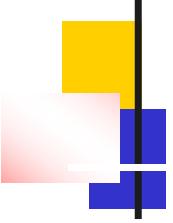
Leaky ReLU
 $\max(0.1x, x)$



Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha (\exp(x) - 1) & \text{if } x \leq 0 \end{cases}$$

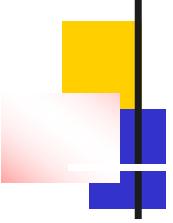


Standard activation functions

- The hard-limiting threshold function
 - Corresponds to the biological paradigm
 - either fires or not
- Sigmoid functions ('S'-shaped curves)
 - The logistic function
 - The hyperbolic tangent (symmetrical)
 - Both functions have a simple differential
 - Only the shape is important

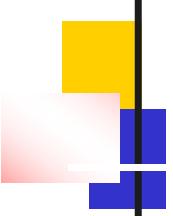


$$\phi(x) = \frac{1}{1 + e^{-ax}}$$



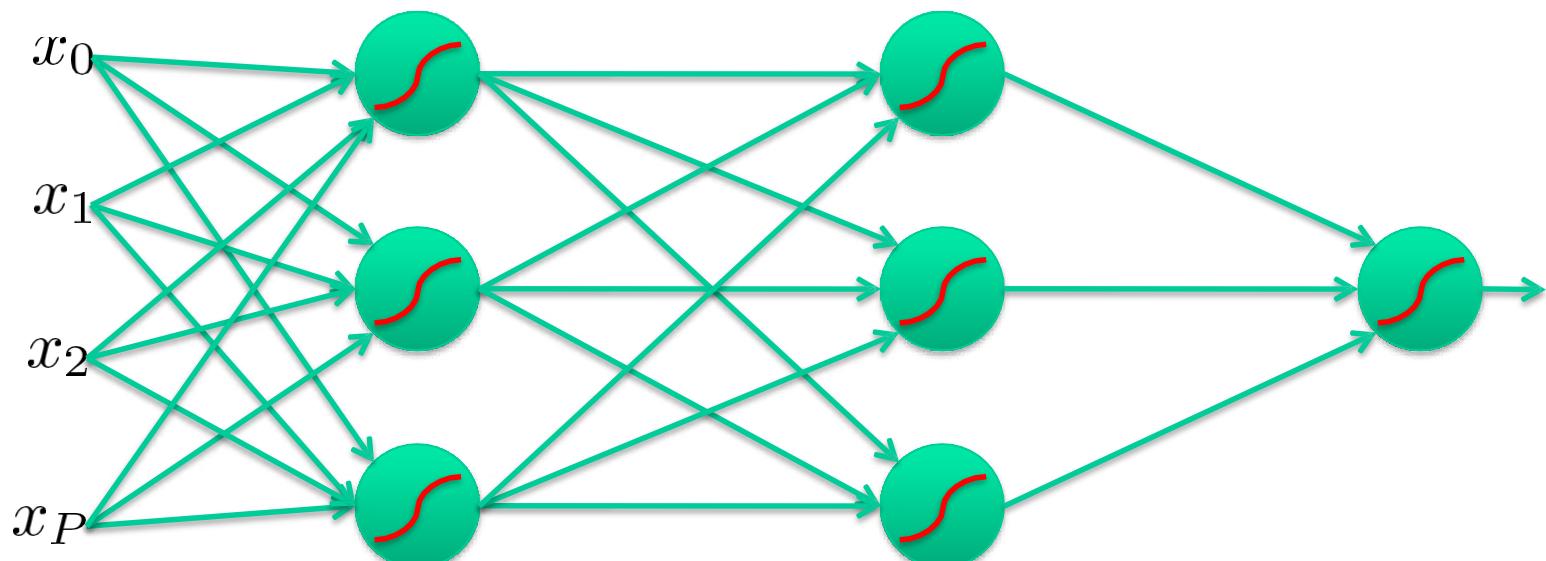
Training Algorithms

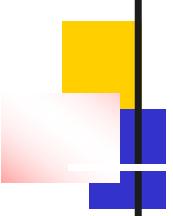
- Adjust neural network weights to map inputs to outputs.
- Use a set of sample patterns where the desired output (given the inputs presented) is known.
- The purpose is to learn to generalize
 - Recognize features which are common to good and bad exemplars



Training Algorithms

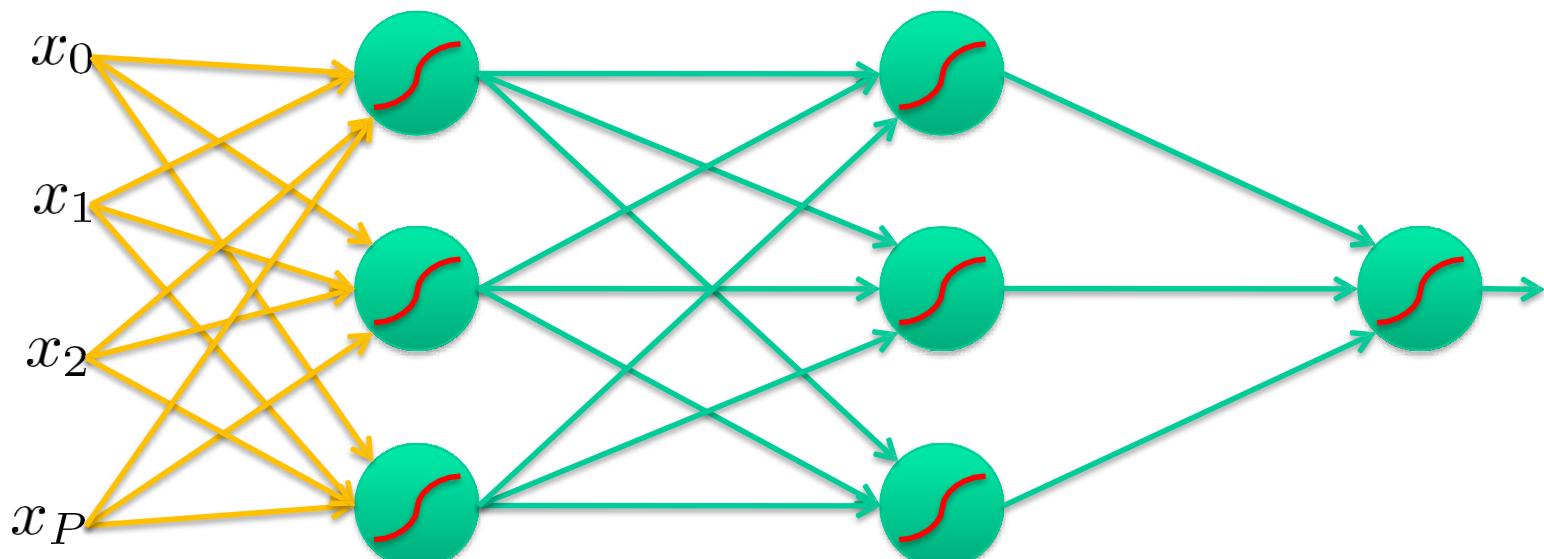
- Cascade neurons together
- Output from one layer is the input to the next
- Each layer has its own sets of weights

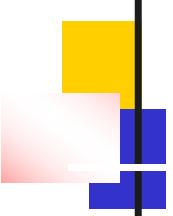




Training Algorithms

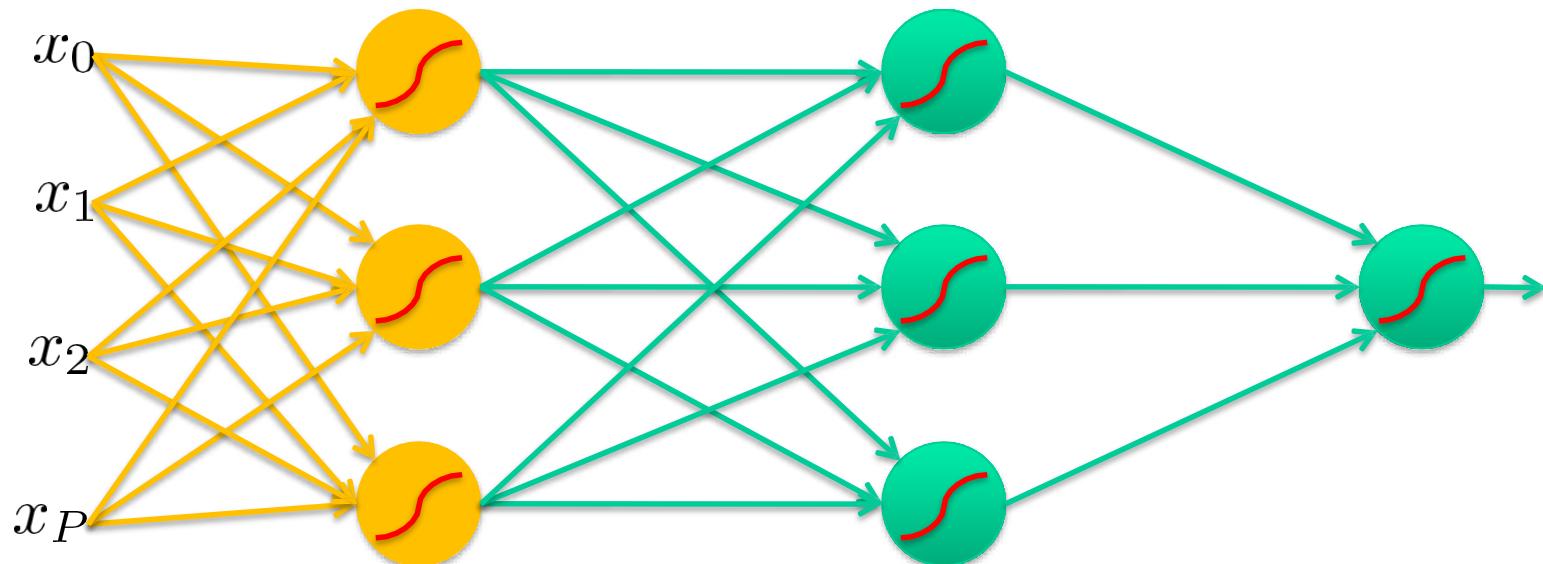
- Predictions are fed forward through the network to classify

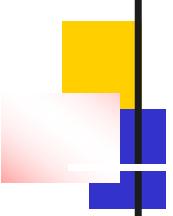




Training Algorithms

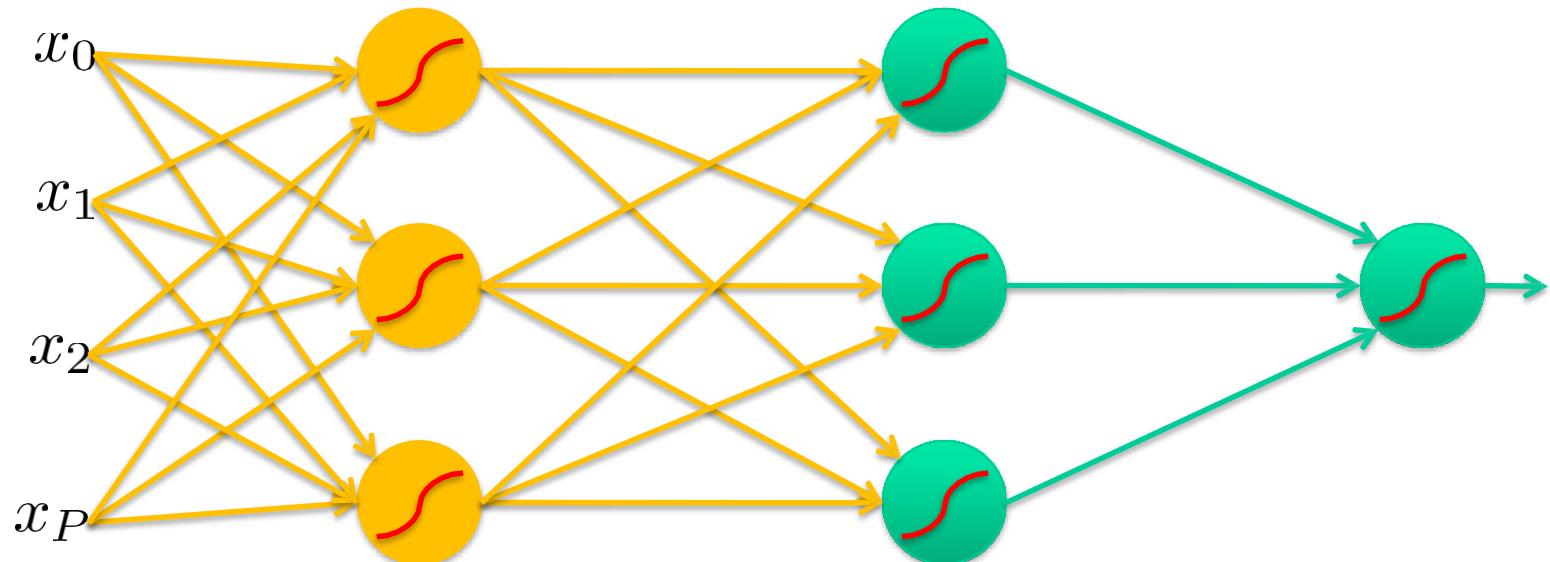
- Predictions are fed forward through the network to classify

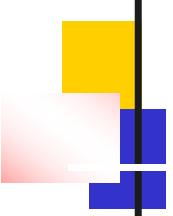




Training Algorithms

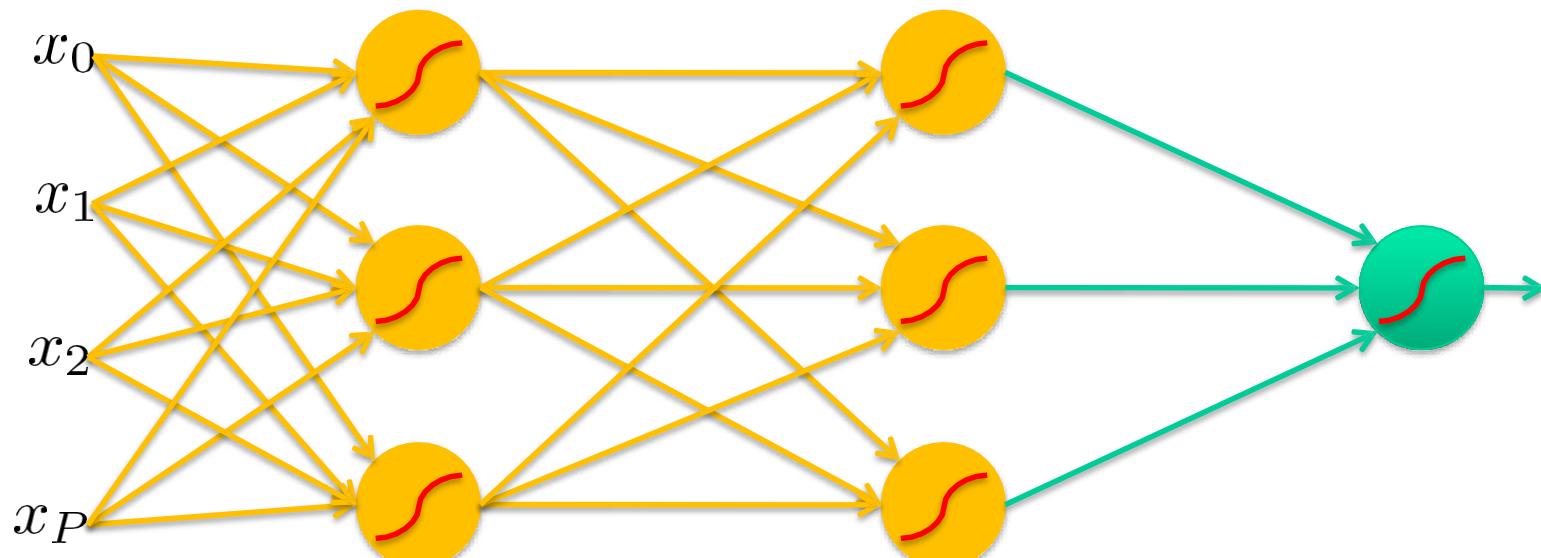
- Predictions are fed forward through the network to classify

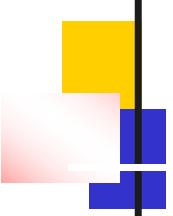




Training Algorithms

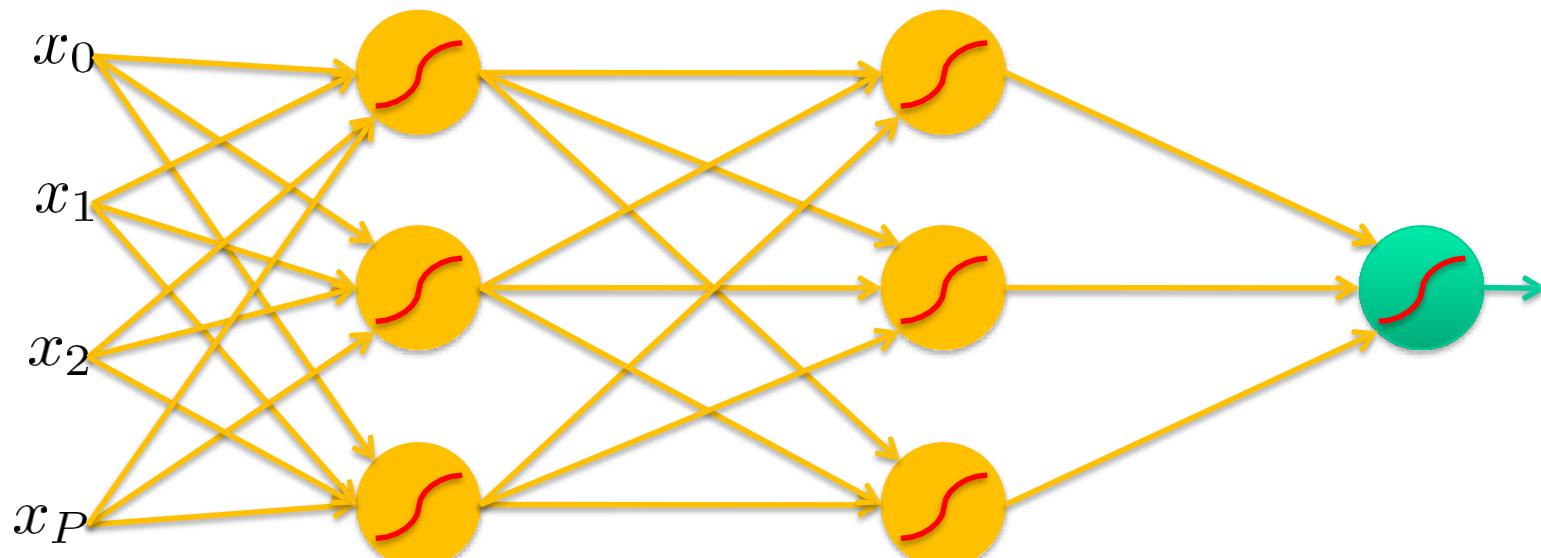
- Predictions are fed forward through the network to classify

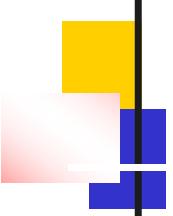




Training Algorithms

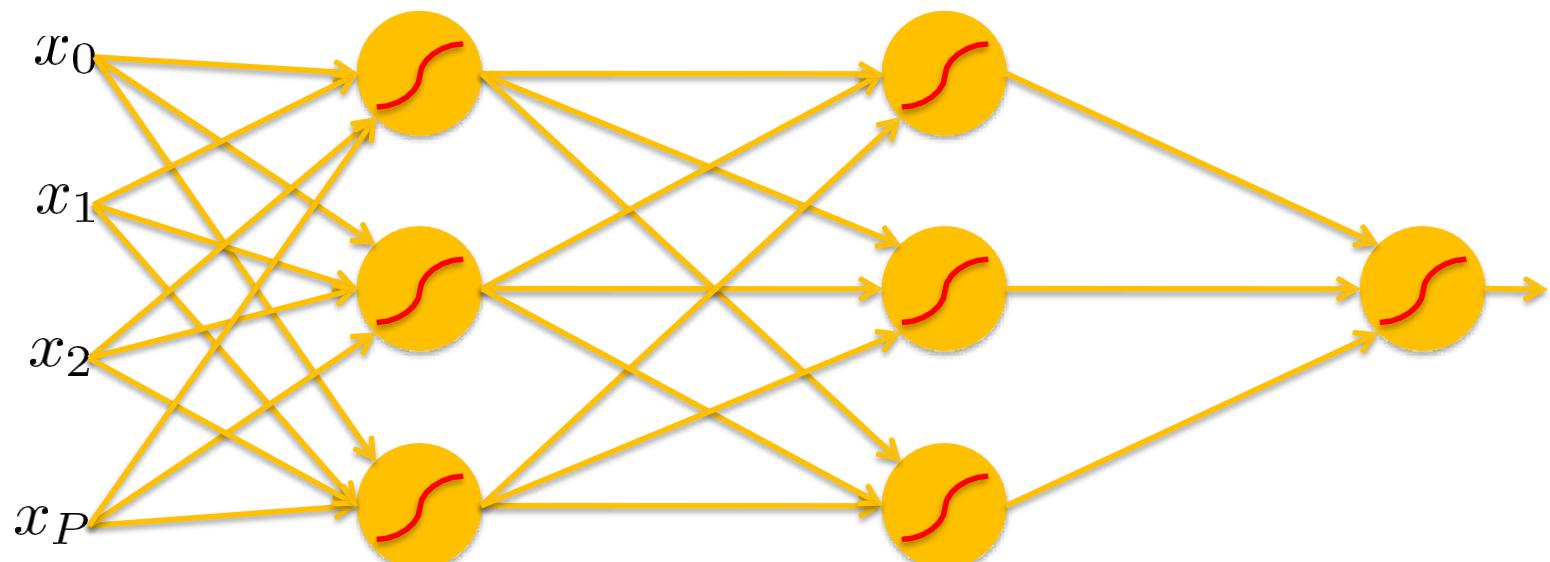
- Predictions are fed forward through the network to classify





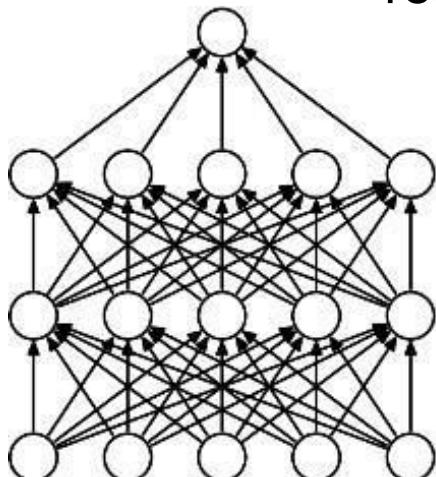
Training Algorithms

- Predictions are fed forward through the network to classify

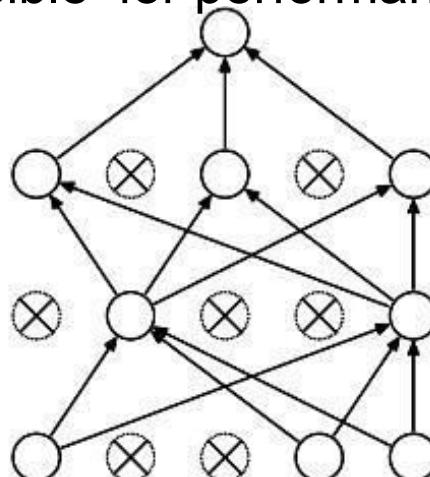


Regularization

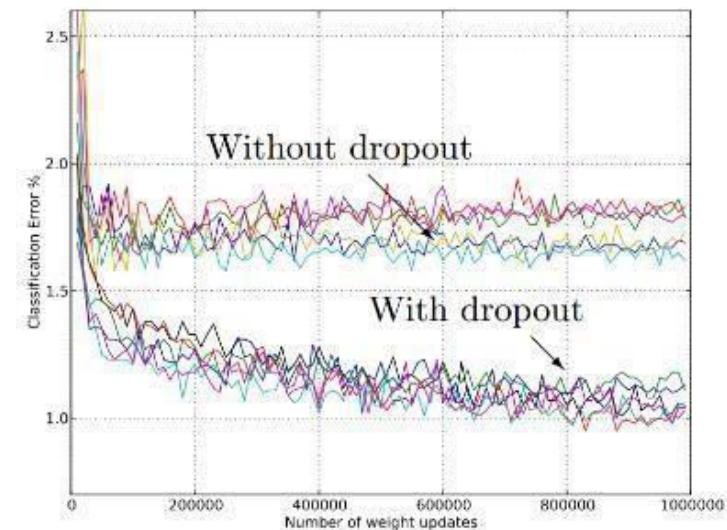
- L1, L2 regularization (*weight decay*)
- Dropout
 - Randomly turn off some neurons
 - Allows individual neurons to independently be responsible for performance



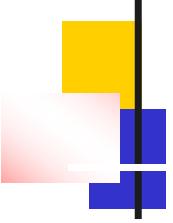
(a) Standard Neural Net



(b) After applying dropout.



Dropout: A simple way to prevent neural networks from overfitting [[Srivastava JMLR 2014](#)]

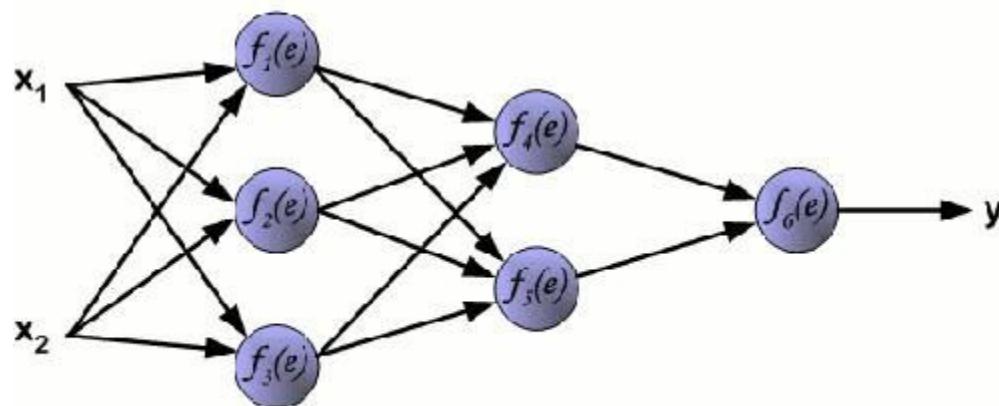


Back-Propagation

- A training procedure which allows multi-layer feedforward Neural Networks to be trained;
- Can theoretically perform “any” input-output mapping;
- Can learn to solve linearly inseparable problems.

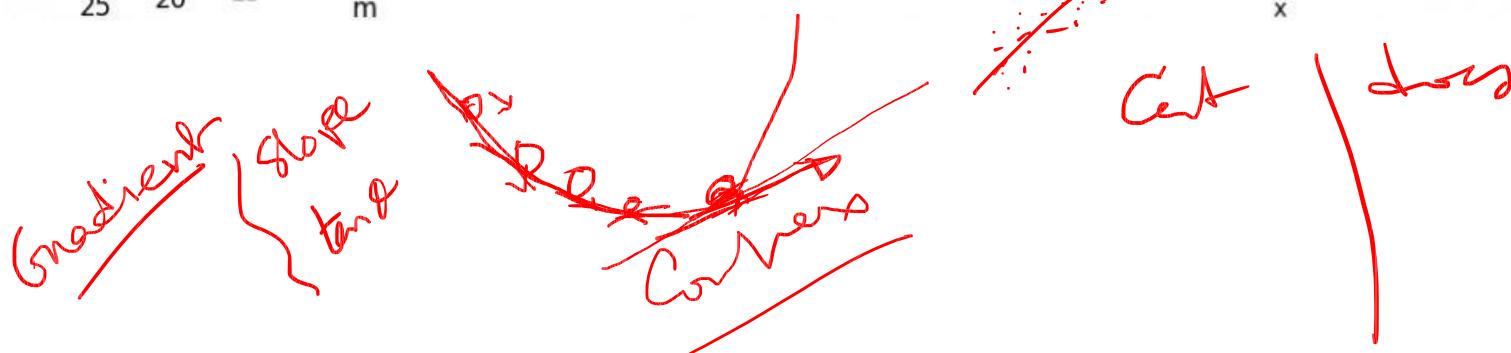
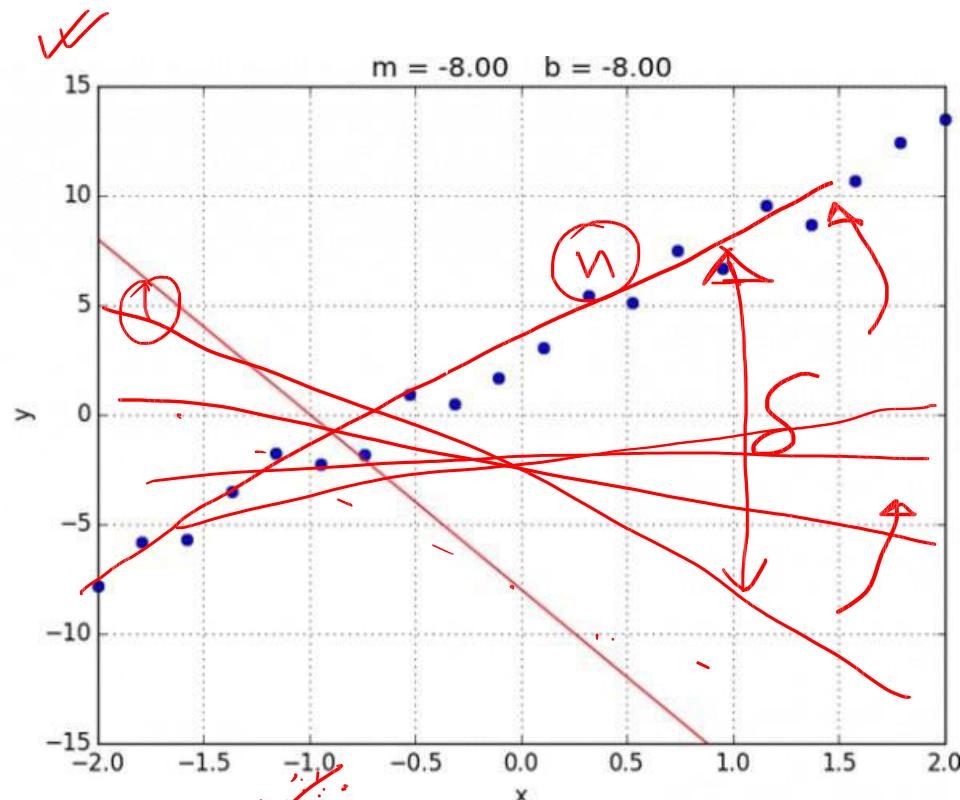
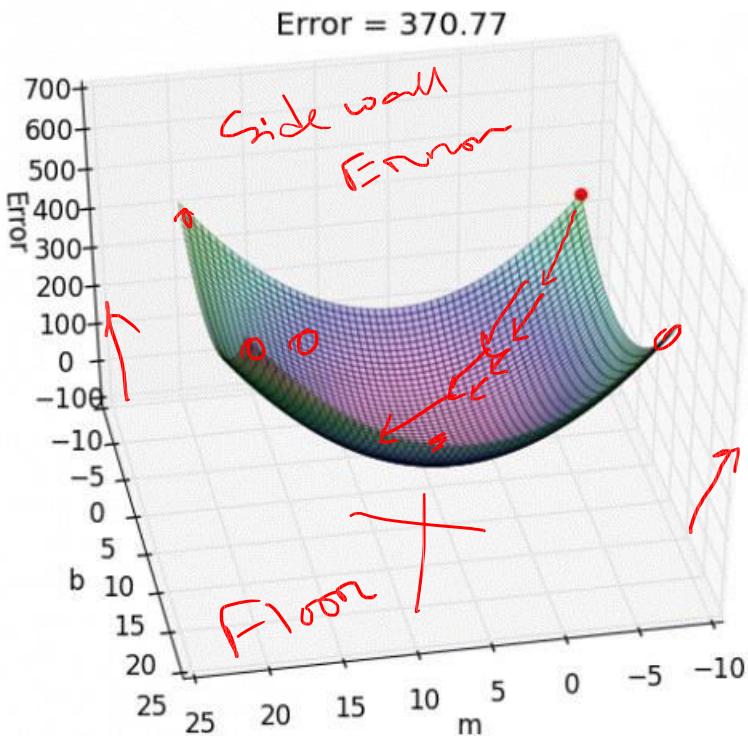
Back-Propagation

Prediction



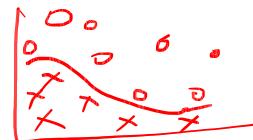
Learning

GRADIENT DESCENT

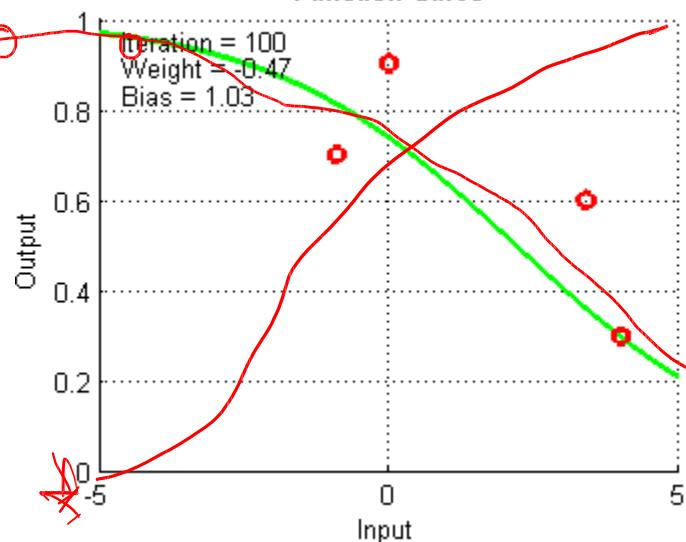


GRADIENT DESCENT

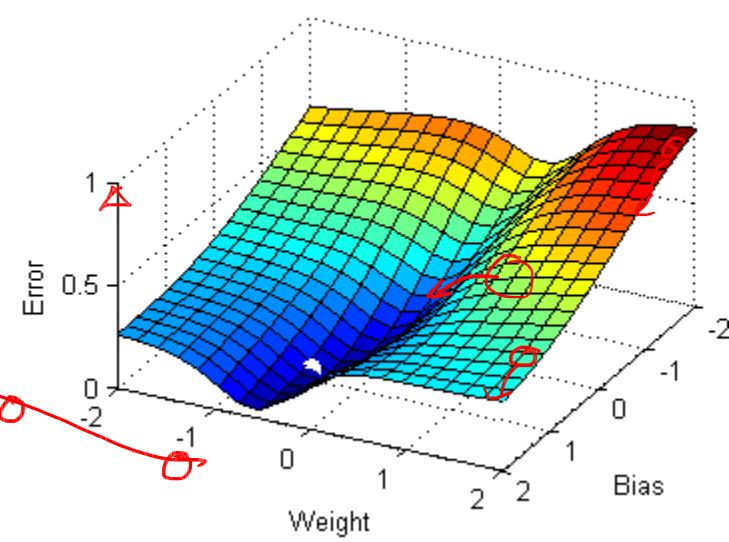
non-linear



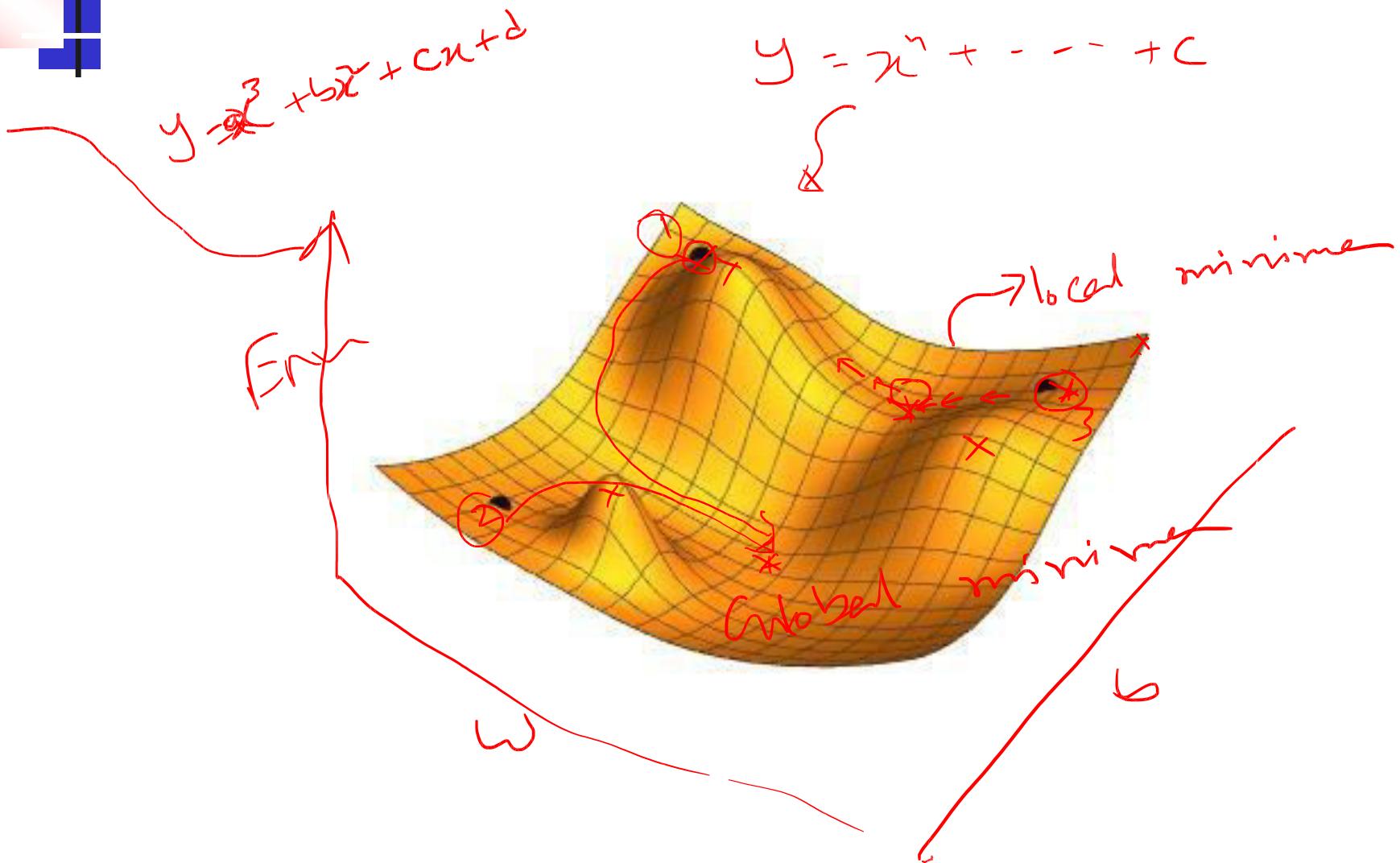
Function Curve



Error Surface



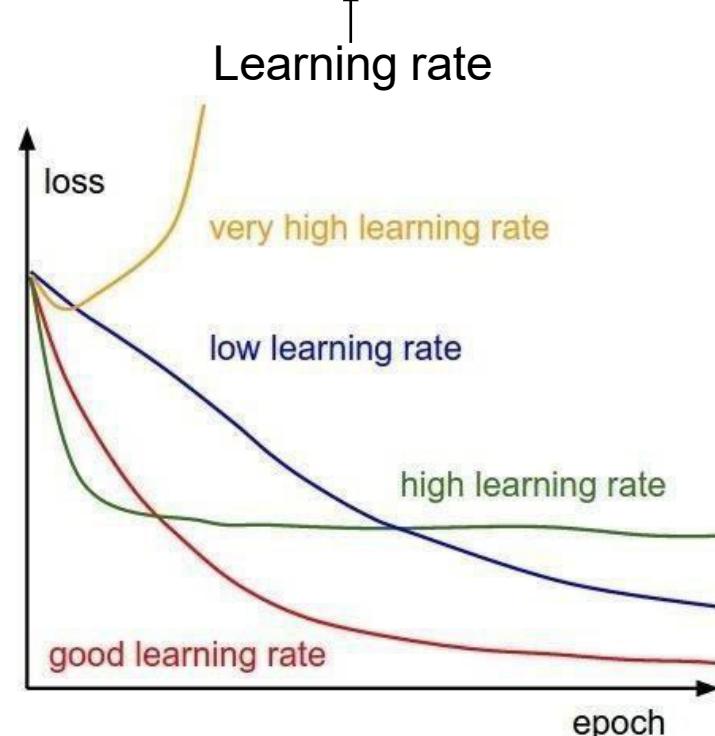
GRADIENT DESCENT



Gradient descent

- We'll update the weights
 - Move in direction opposite to gradient:

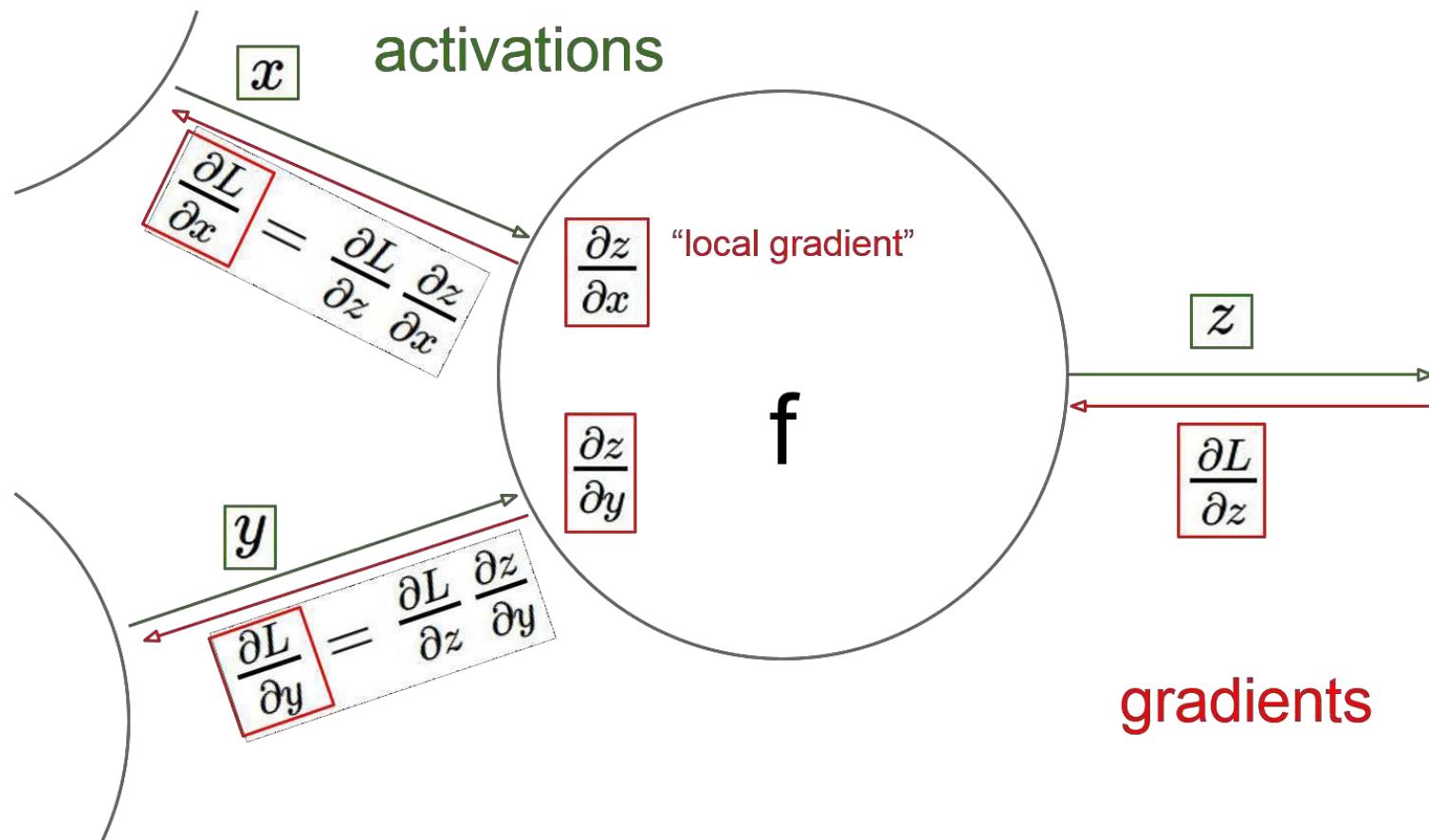
$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)})$$



Gradient descent in multi-layer nets

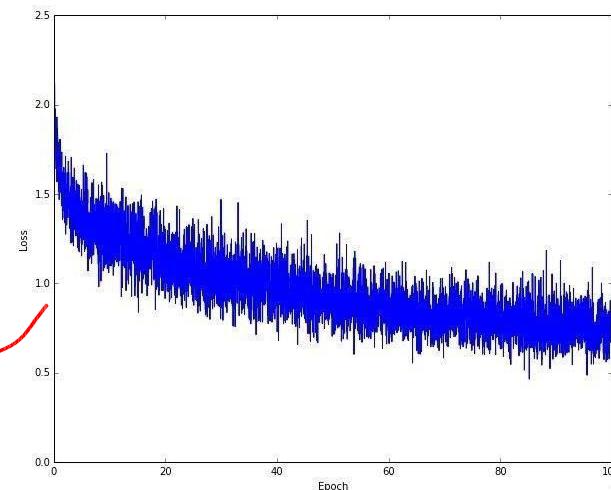
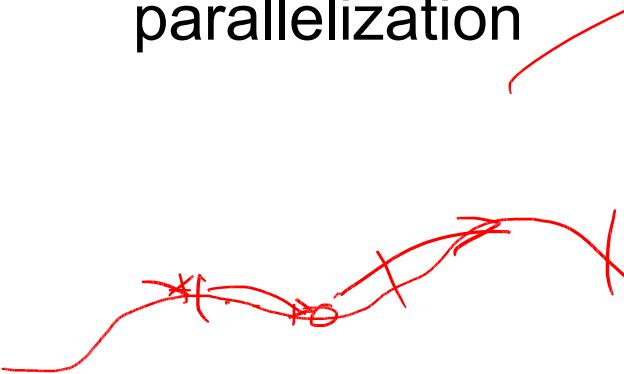
How to update the weights at all layers?

- Answer: backpropagation of error from higher layers to lower layers



Mini-batch gradient descent

- Rather than compute the gradient from the loss for all training examples, could only use some of the data for each gradient update
- We cycle through all the training examples multiple times; each time we've cycled through all of them once is called an 'epoch'
- Allows faster training (e.g. on GPUs), parallelization



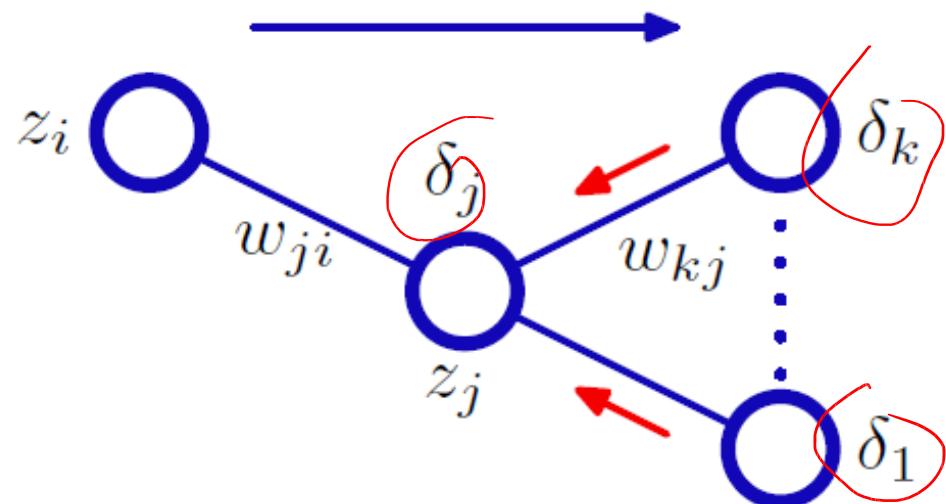
Backpropagation: Error

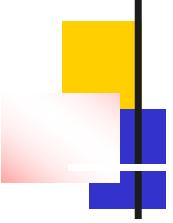
- For output units (e.g. identity output, least squares loss): $\delta_k = y_k - t_k$
- For hidden units:

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k} \frac{\partial a_k}{\partial a_j}$$

- Backprop formula:

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k$$

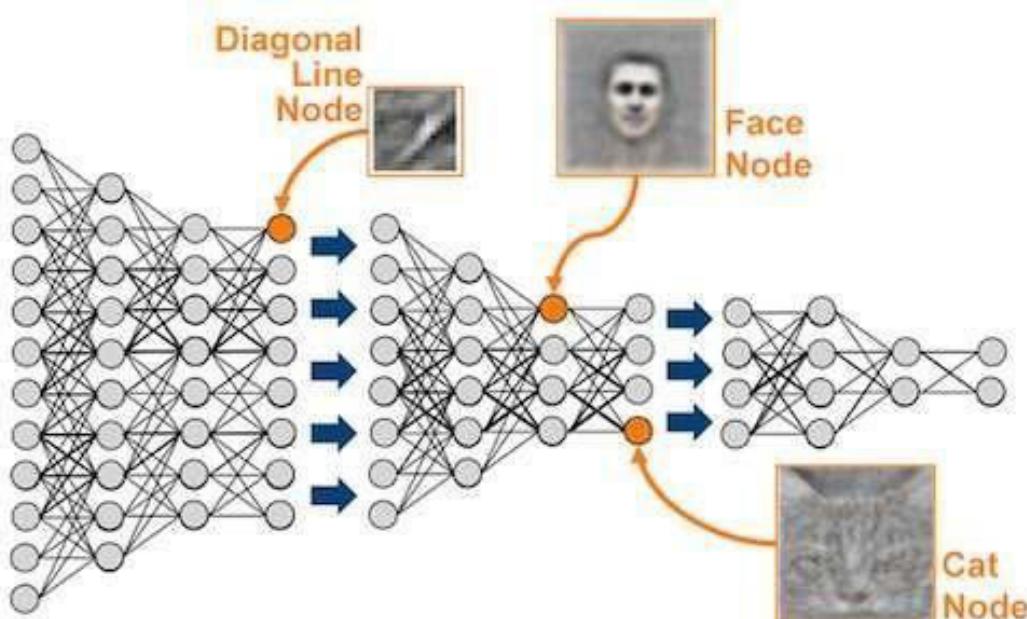
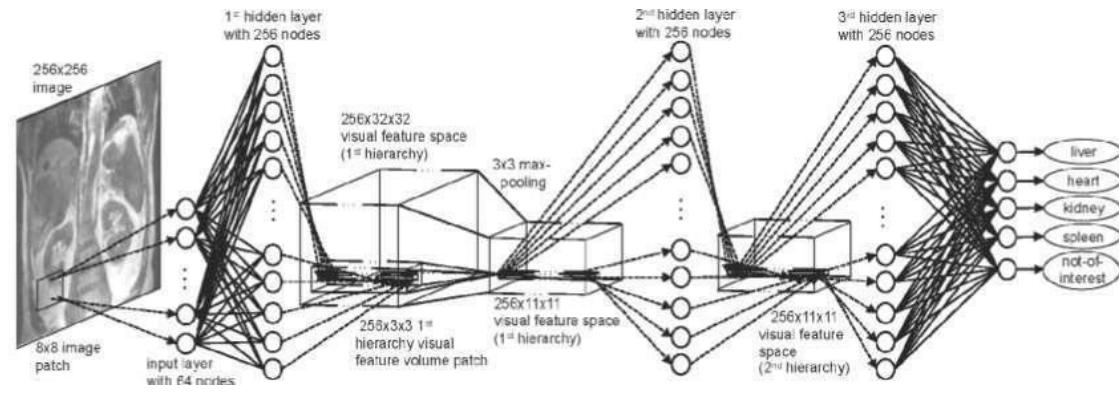


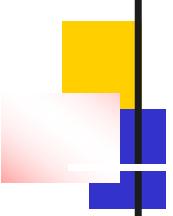


Applications

- The properties of neural networks define where they are useful.
 - Can learn complex mappings from inputs to outputs, based solely on samples
 - Difficult to analyse: firm predictions about neural network behaviour difficult;
 - Unsuitable for safety-critical applications.
 - Require limited understanding from trainer, who can be guided by heuristics.

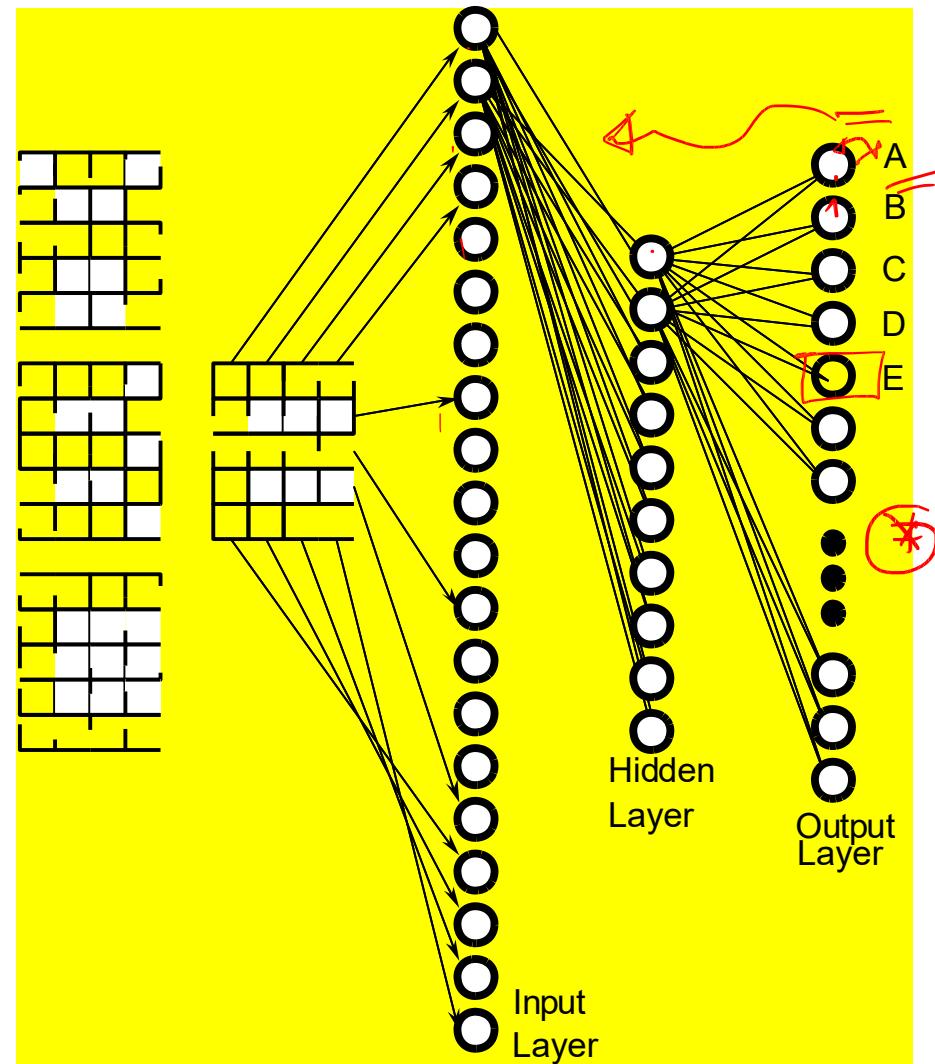
Image Classification

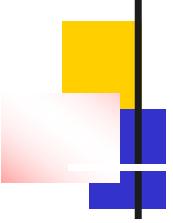




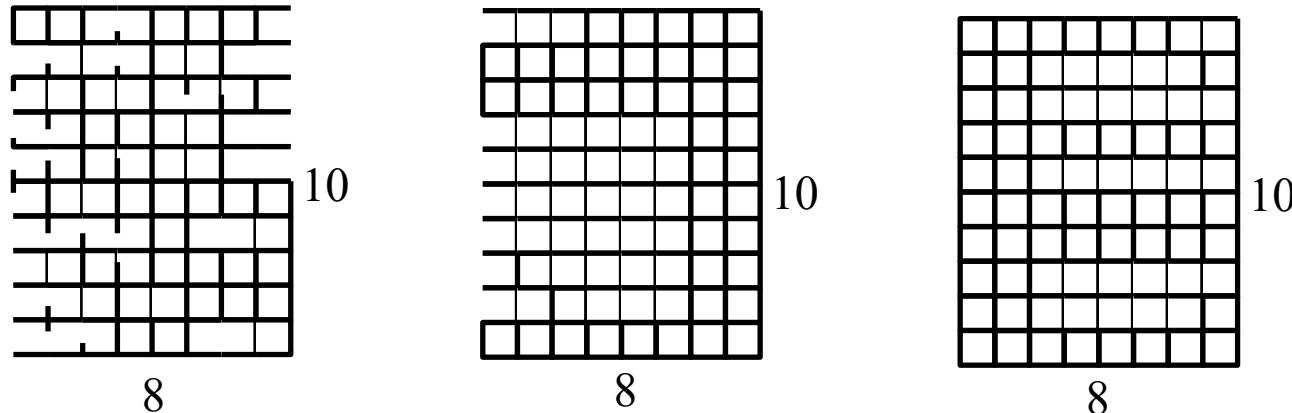
Neural network for OCR

- feedforward network
- trained using Back-propagation

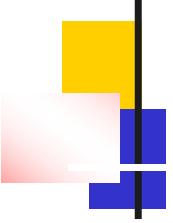




OCR for 8x10 characters



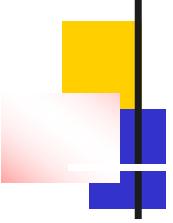
- NN are able to generalise
- learning involves generating a partitioning of the input space
- for single layer network input space must be linearly separable
- what is the dimension of this input space?
- how many points in the input space?
- this network is binary(uses binary values)
- networks may also be continuous



Engine management



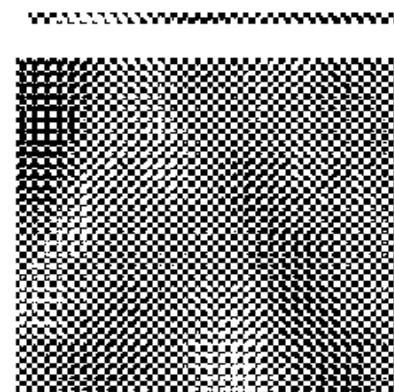
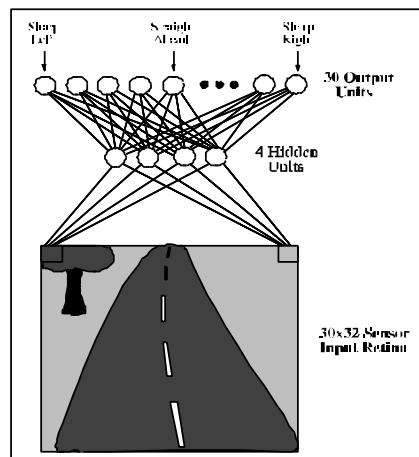
- The behaviour of a car engine is influenced by a large number of parameters
 - temperature at various points
 - fuel/air mixture
 - lubricant viscosity.
- Major companies have used neural networks to dynamically tune an engine depending on current settings.



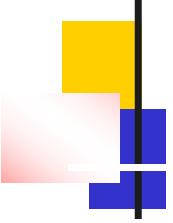
ALVINN

Drives 70 mph on a public highway

30 outputs
for steering
4 hidden
units
30x32 pixels
as inputs

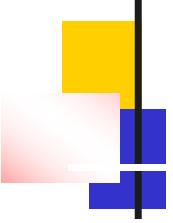


30x32 weights
into one out of
four hidden
unit

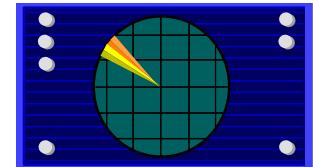


Signature recognition

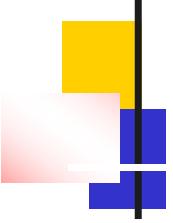
- Each person's signature is different.
- There are structural similarities which are difficult to quantify.
- One company has manufactured a machine which recognizes signatures to within a high level of accuracy.
 - Considers speed in addition to gross shape.
 - Makes forgery even more difficult.



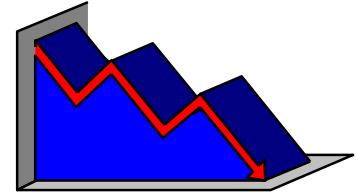
Sonar target recognition



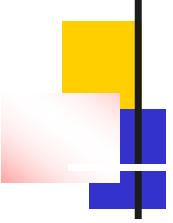
- Distinguish mines from rocks on sea-bed
- The neural network is provided with a large number of parameters which are extracted from the sonar signal.
- The training set consists of sets of signals from rocks and mines.



Stock market prediction



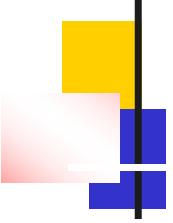
- “Technical trading” refers to trading based solely on known statistical parameters; e.g. previous price
- Neural networks have been used to attempt to predict changes in prices.
- Difficult to assess success since companies using these techniques are reluctant to disclose information.



Mortgage assessment

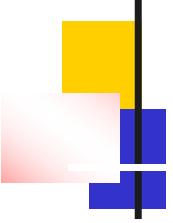


- Assess risk of lending to an individual.
- Difficult to decide on marginal cases.
- Neural networks have been trained to make decisions, based upon the opinions of expert underwriters.
- Neural network produced a 12% reduction in delinquencies compared with human experts.



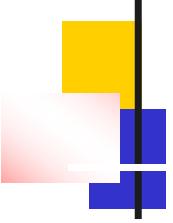
Neural Network Problems

- Many Parameters to be set
- Overfitting
- long training times
- ...



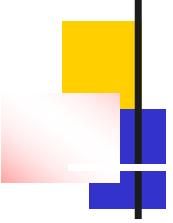
Parameter setting

- Number of layers
- Number of neurons
 - too many neurons, require more training time
- Learning rate
 - from experience, value should be small ~ 0.1
- Momentum term
- ..



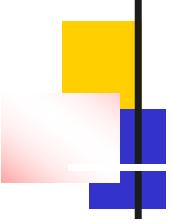
Over-fitting

- With sufficient nodes can classify any training set exactly
- May have poor generalisation ability.
- Cross-validation with some patterns
 - Typically 30% of training patterns
 - Validation set error is checked each epoch
 - Stop training if validation error goes up



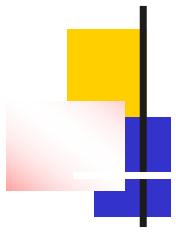
Training time

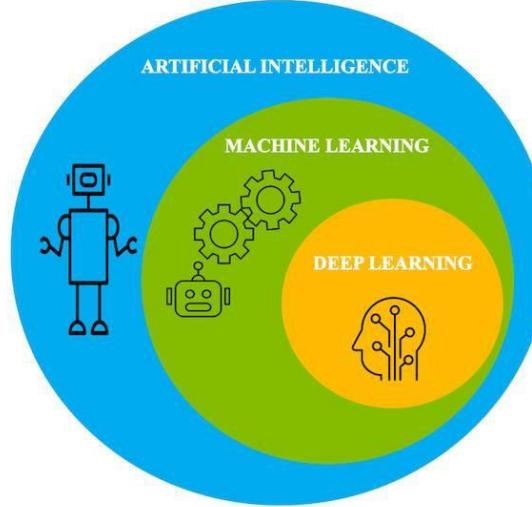
- How many epochs of training?
 - Stop if the error fails to improve (has reached a minimum)
 - Stop if the rate of improvement drops below a certain level
 - Stop if the error reaches an acceptable level
 - Stop when a certain number of epochs have passed



References

- <https://www.slideserve.com/phelan-cortez/artificial-neural-networks>
- <https://media.giphy.com/media/4LiMmbAcvgTQs/giphy.gif>
- https://miro.medium.com/max/1400/1*E-5K5rHxCRTPrSWF60XLWw.gif
- <https://media.giphy.com/media/O9rcZVmRcEGqI/giphy.gif>
- <https://media.giphy.com/media/4LiMmbAcvgTQs/giphy.gif>





Lambton College
School of Computer Studies

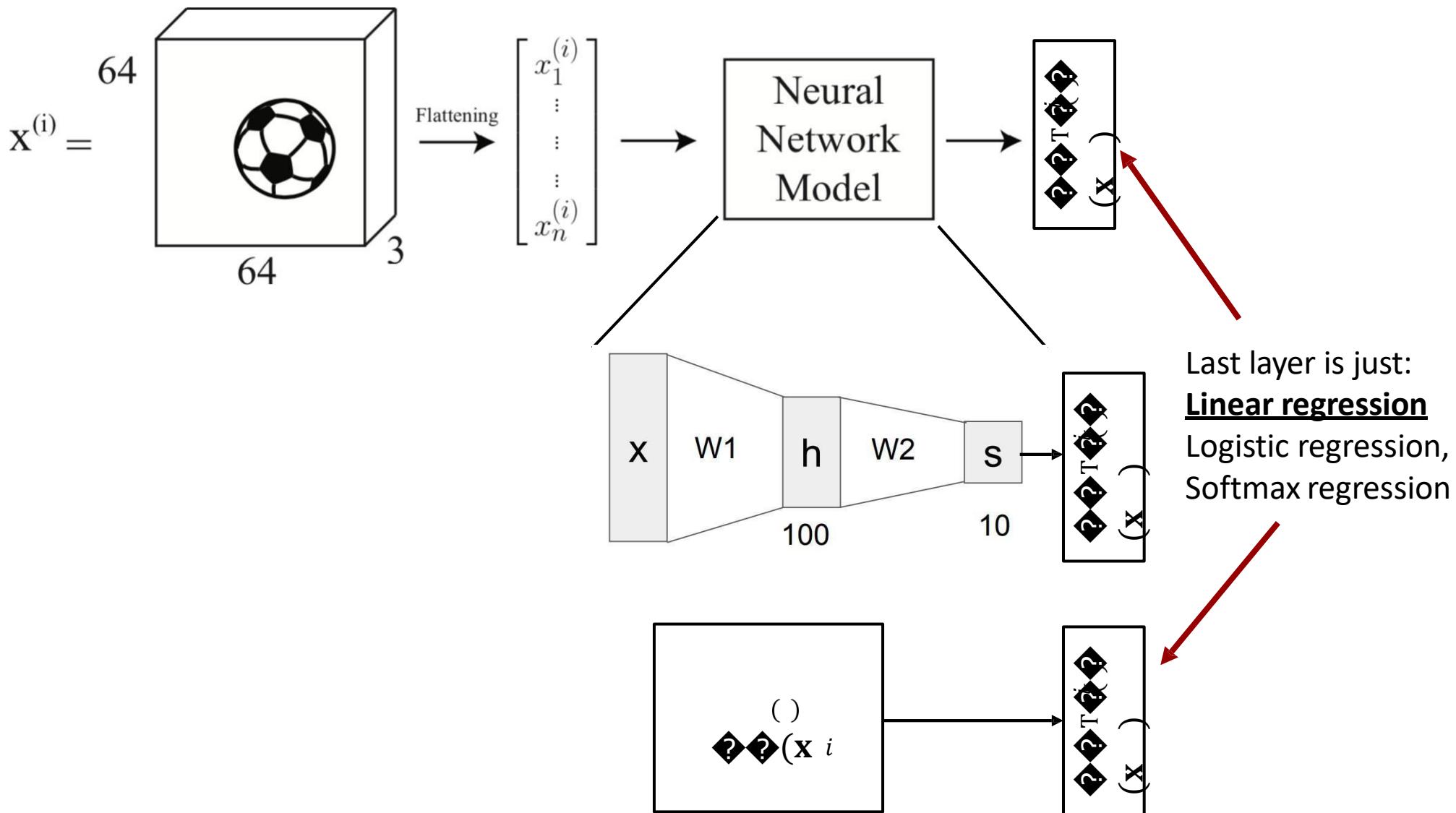
Lambton
College

AML-3104 Neural Networks
and Deep Learning

Deep Neural Networks

Convolutional Neural Networks

What we have learned so far?



What we have learned so far?

We must apply nonlinear activation.

For hidden layers, often

- ReLU: $a(x) = \max\{0,$
- Hyperbolic tangent: $a(x) = \tanh(\text{?})$

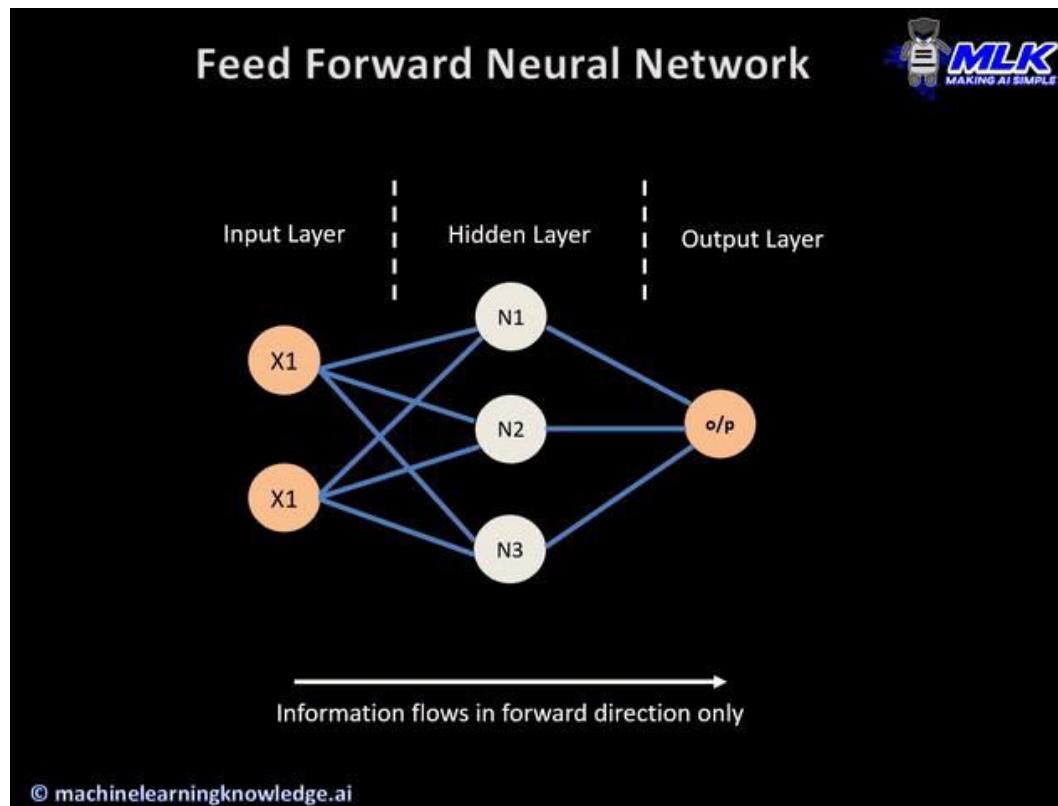
For output layers, often

- Linear (identity): $a(x) = x$
- Sigmoid: $a(x) = \frac{1}{1 + \exp(-\text{?})}$
- Softmax: $= \frac{\exp(x_i)}{\sum_j \exp(x_j)}$
 $\text{?}(\text{?})$

What we have learned so far?

We can learn the parameters using backpropagation:

- Forward pass: calculates the output of the NN and loss
- Backward pass: updates the parameters of the NN



How do we learn?

Batch Gradient Descent:

- Expensive to compute gradient for large dataset
- Computational and space complexity high

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta)$$

Stochastic Gradient Descent:

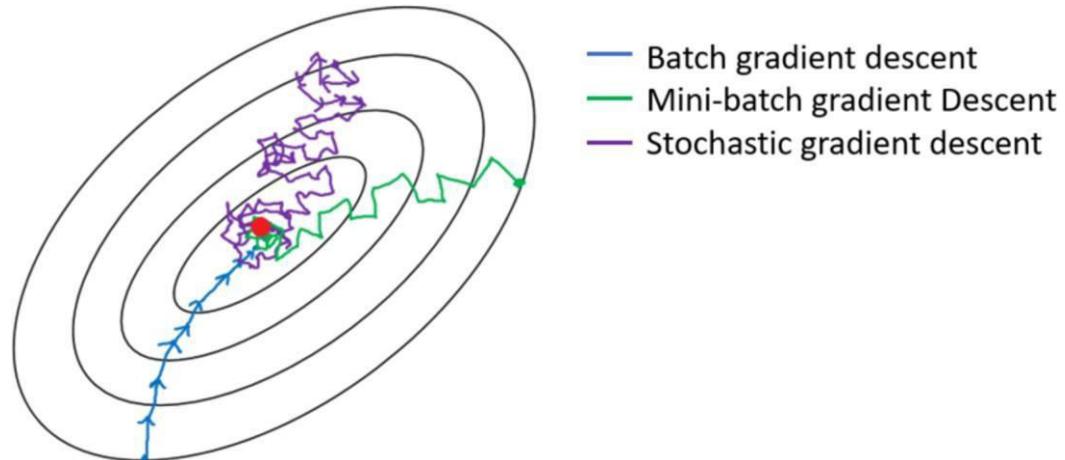
- Lots of random motion (slow to converge)

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)})$$

Mini-batch Gradient Descent (Mini-batch of size n):

- Hybrid between the two, still stochastic but less random

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$



How do we learn?

Stochastic Gradient Descent:

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i)}; y^{(i)})$$

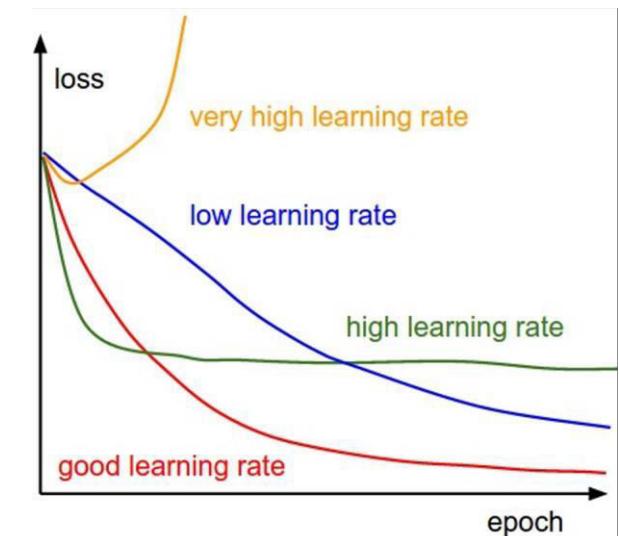
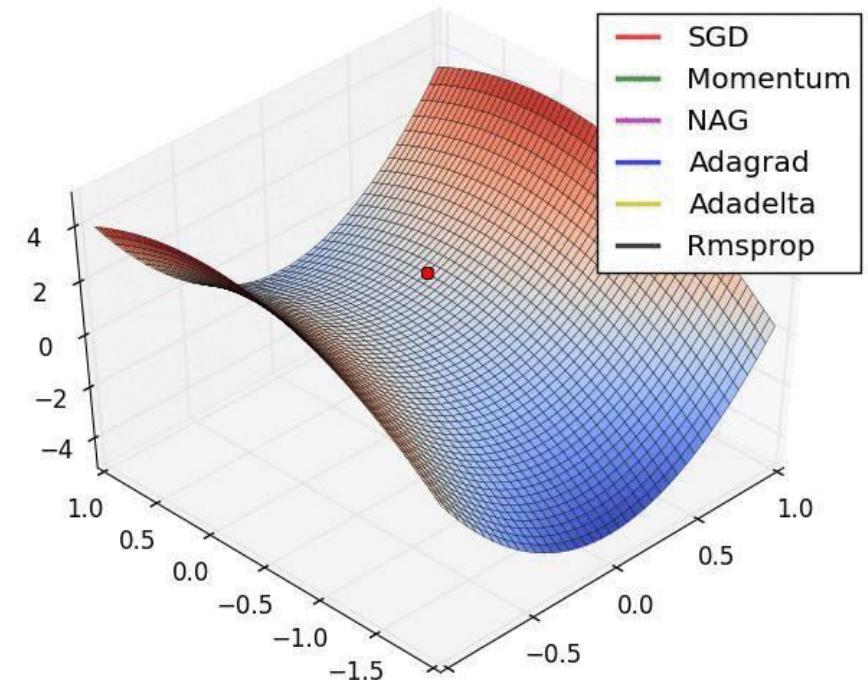
Mini-batch Gradient Descent (Mini-batch of size n):

$$\theta = \theta - \eta \cdot \nabla_{\theta} J(\theta; x^{(i:i+n)}; y^{(i:i+n)})$$

There are better variants of (mini-batch) SGD:

- Momentum: Gradient + Momentum
- Nesterov: Momentum + Gradients
- Adagrad: Normalize with sum of sq
- RMSprop: Normalize with moving avg of sum of squares
- **ADAM: RMSprop + momentum (most popular)**

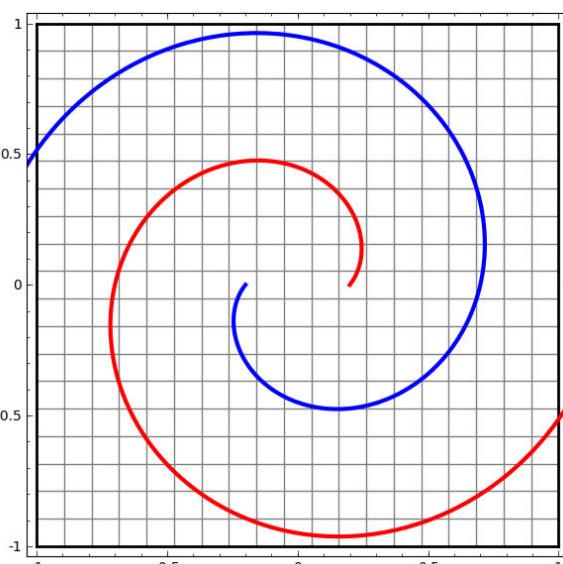
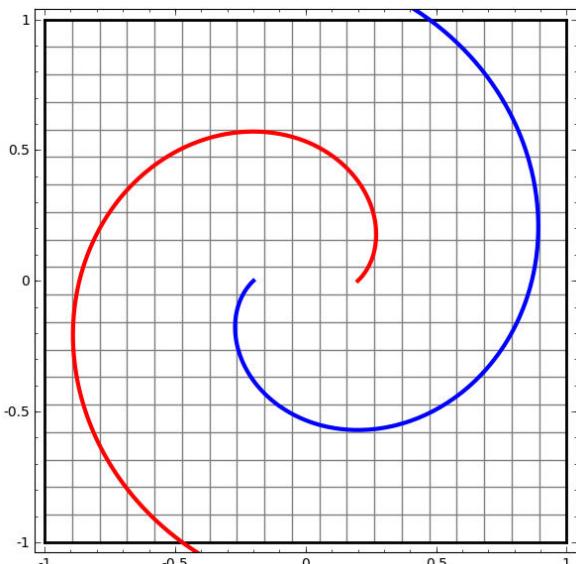
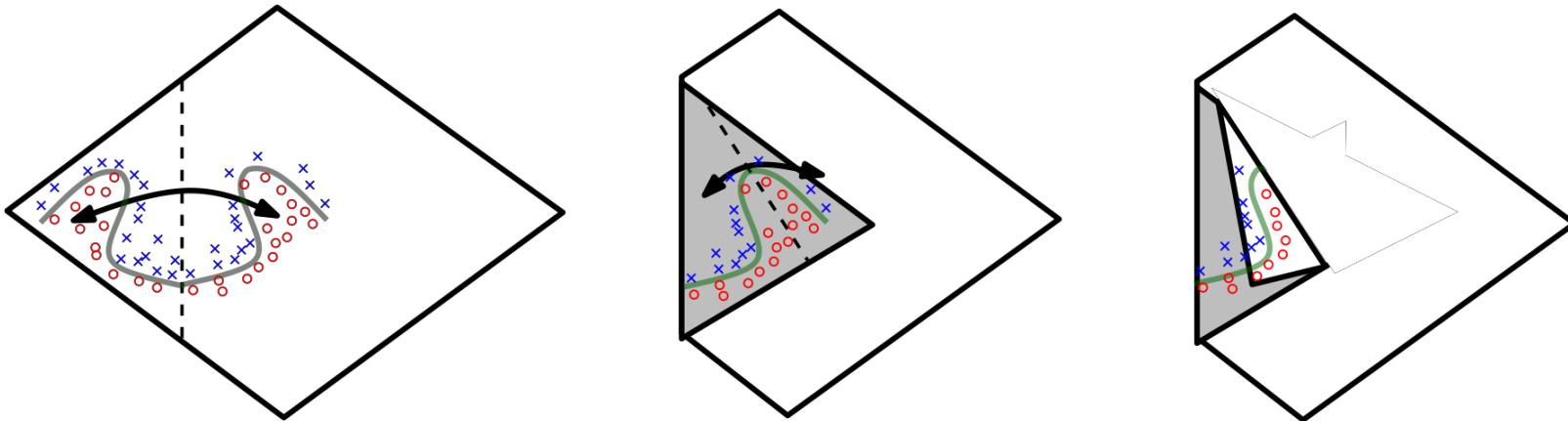
See this link for more details: <https://ruder.io/optimizing-gradient-descent/>



Universal Approximator Theorem

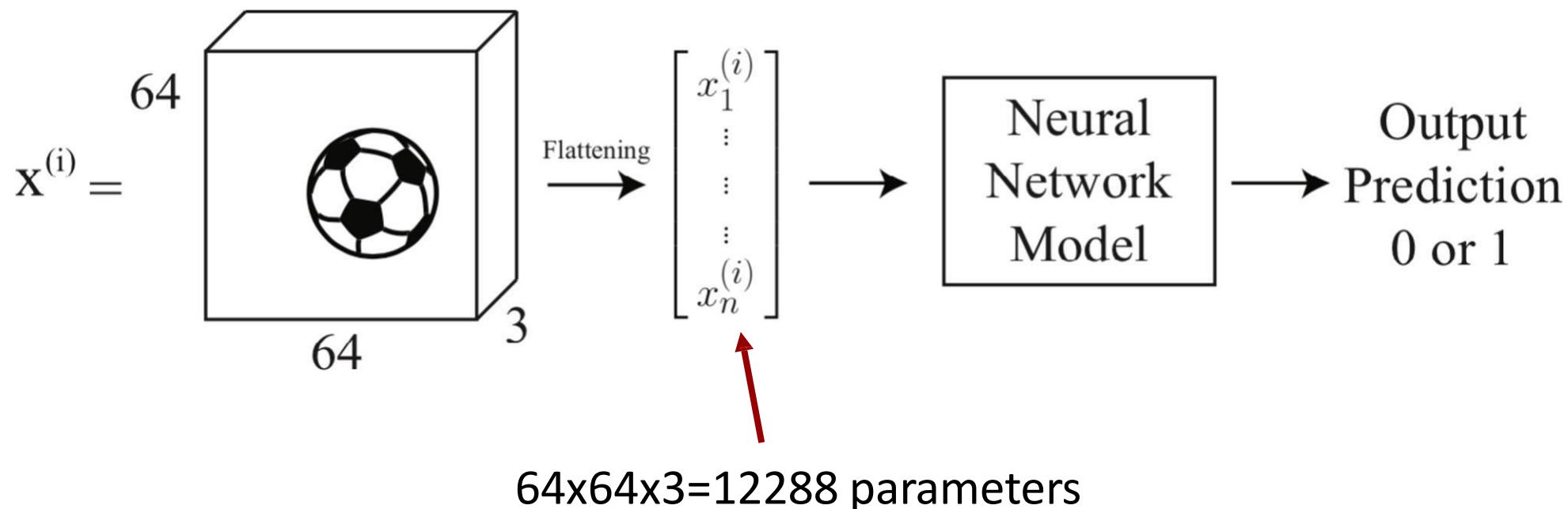
- One/two hidden layer is enough to *represent* (not *learn*) an approximation of any function to an arbitrary degree of accuracy
- So why use deep neural nets?
 - Shallow net may need (exponentially) more width to improve accuracy
 - Shallow net may overfit more

Exponential Representation Advantage of Depth



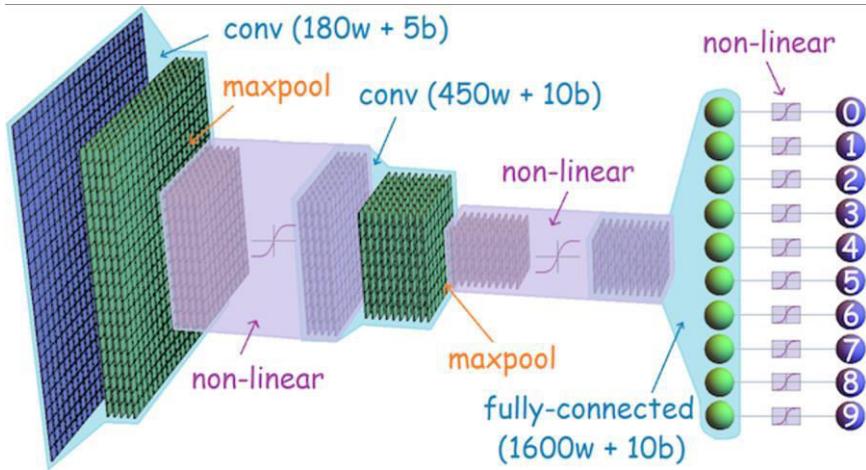
Limitations of Multi-Layer Perceptron (MLP)

- Does not exploit the order and local relations in the data!
- Large number of parameters in deep networks
- May not generalize well

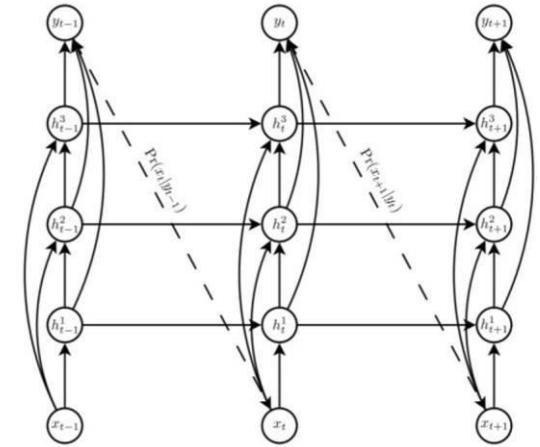


What are some specialized deep learning models?

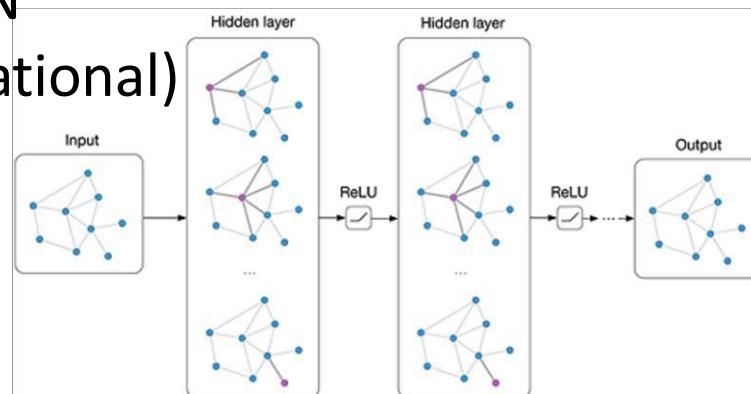
Convolutional
NN (Images)



Recurrent or
Transformer NNs
(Time Series)

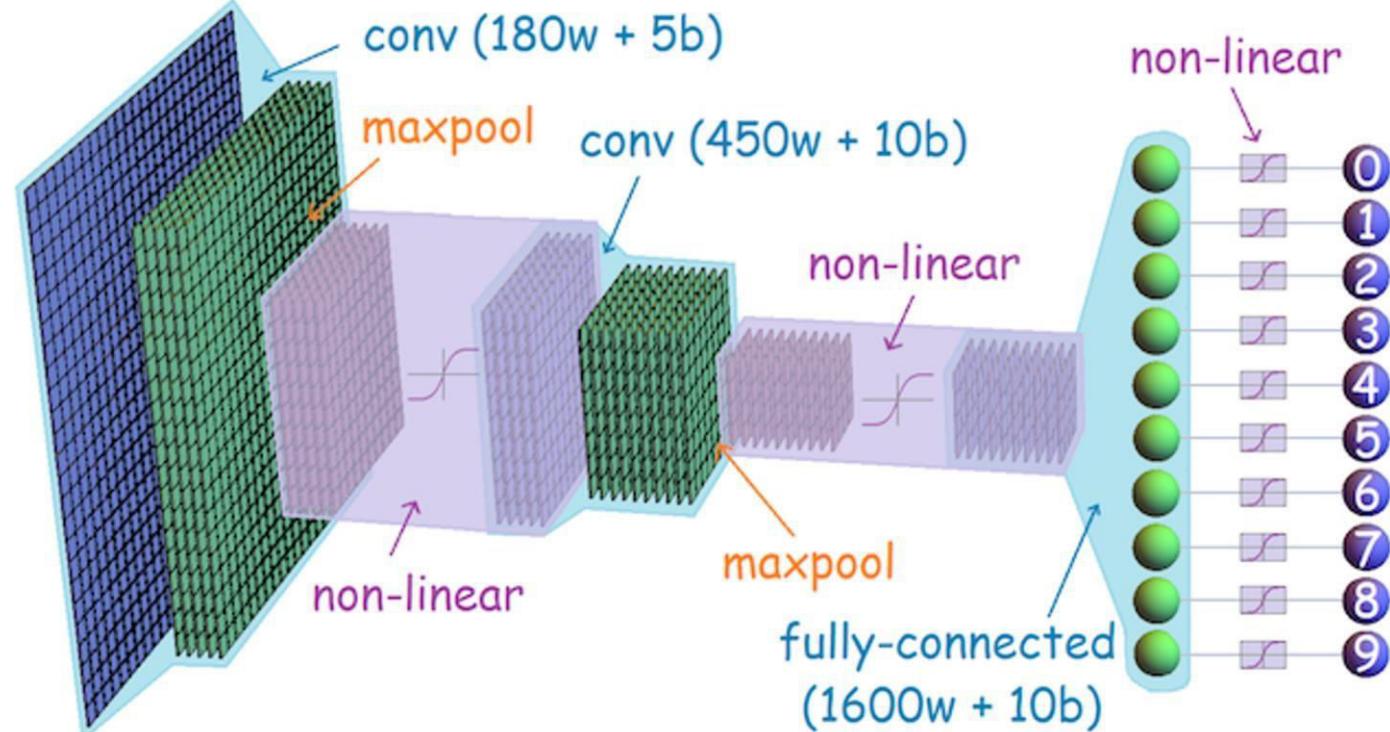


Graph NN
(Networks/Relational)



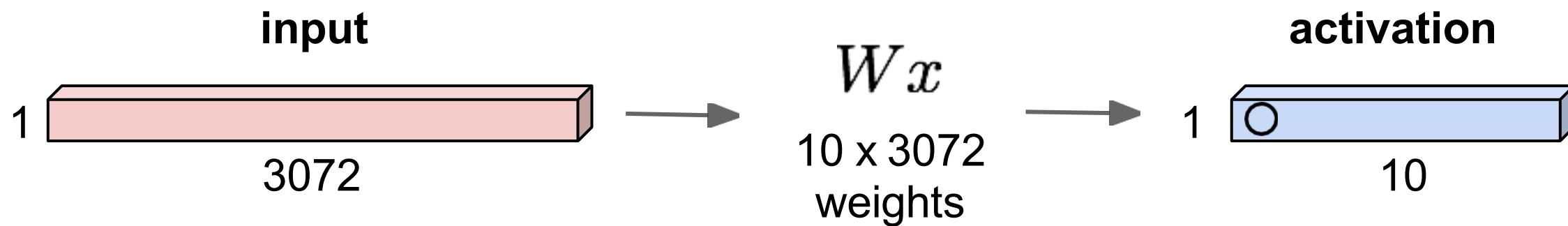
What are some specialized deep learning models?

Convolutional Neural Network



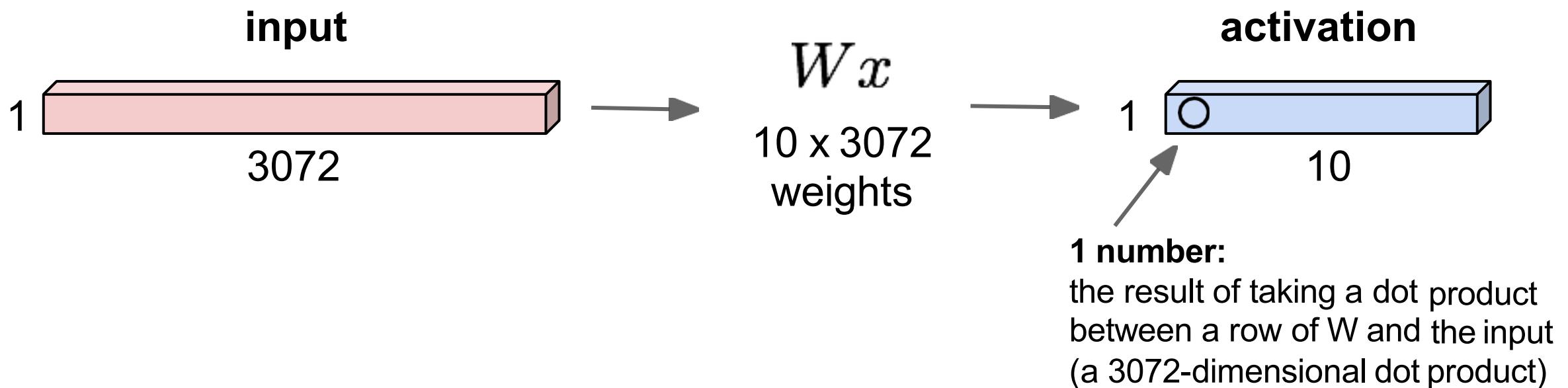
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



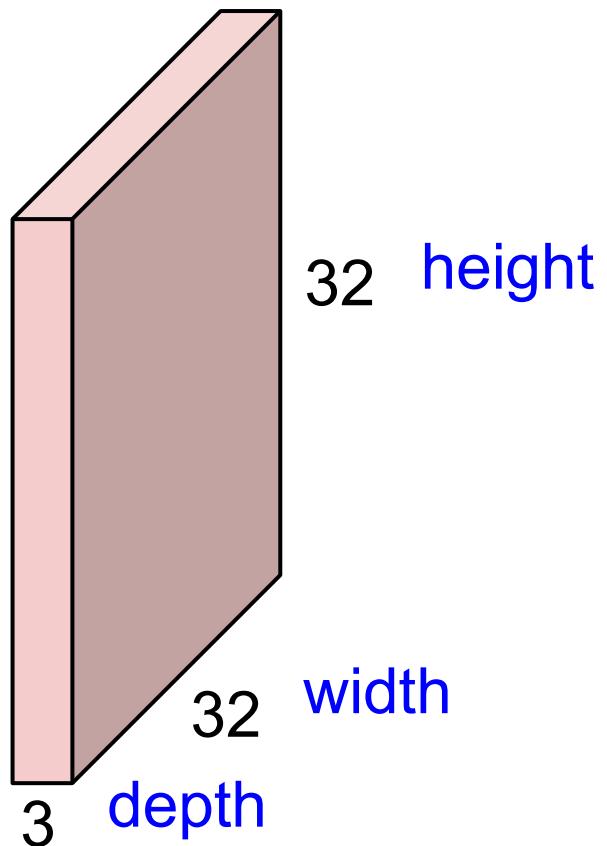
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1



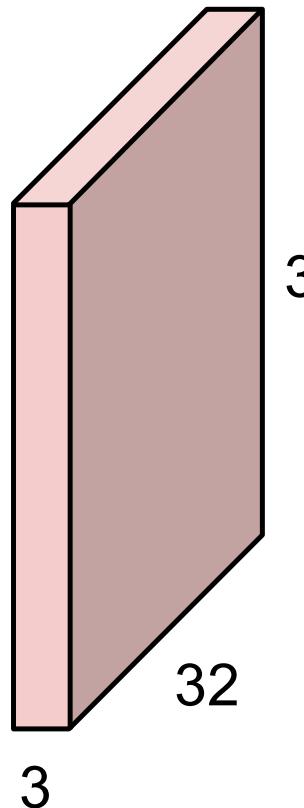
Convolution Layer

32x32x3 image -> preserve spatial structure



Convolution Layer

32x32x3 image



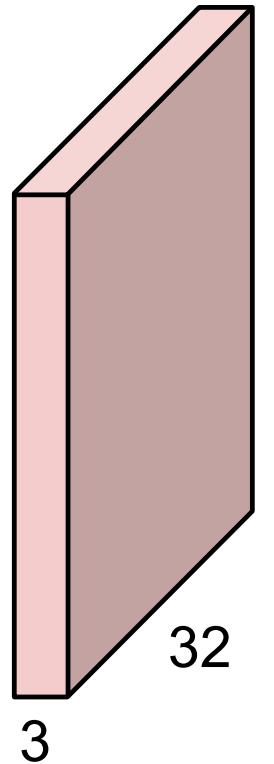
5x5x3 filter



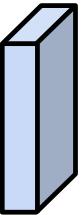
Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

Convolution Layer

32x32x3 image



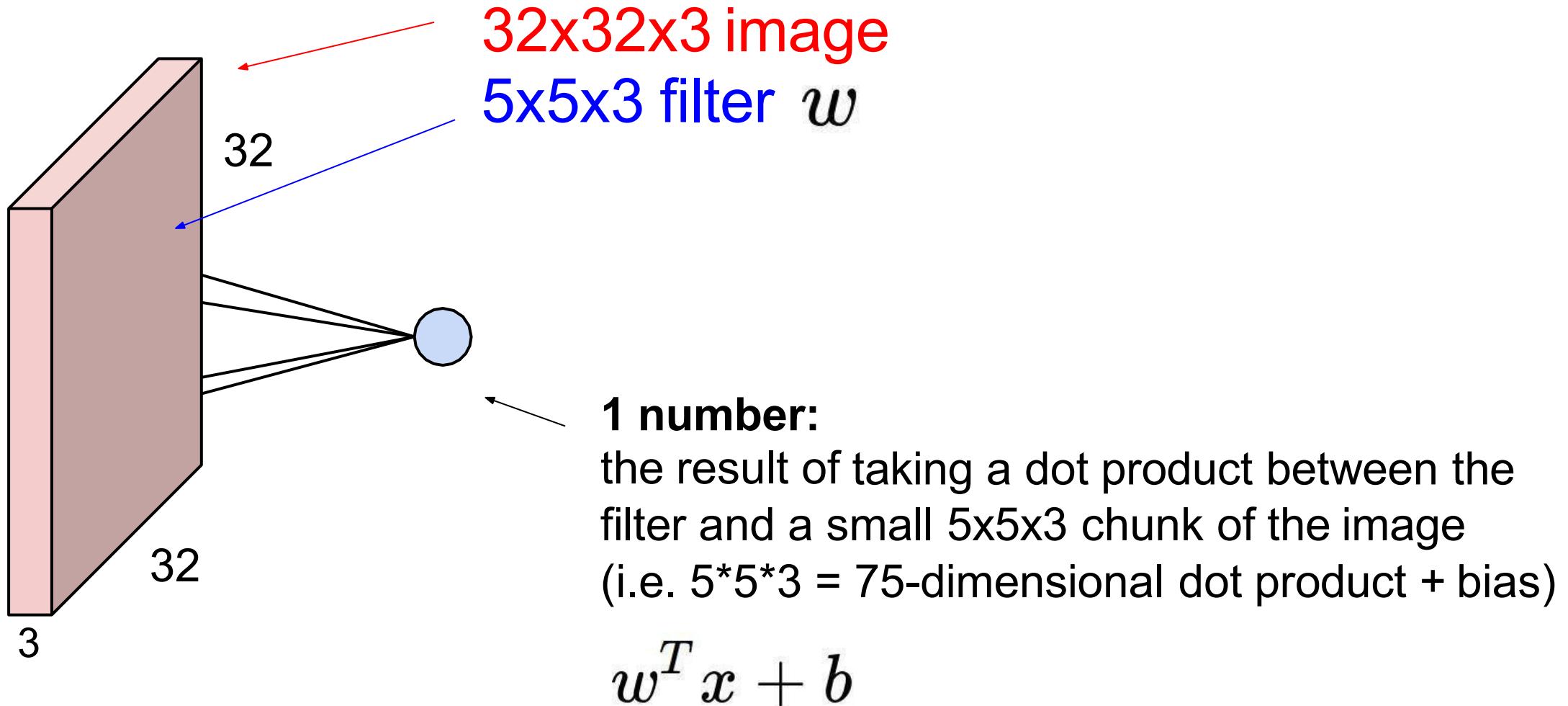
5x5x3 filter



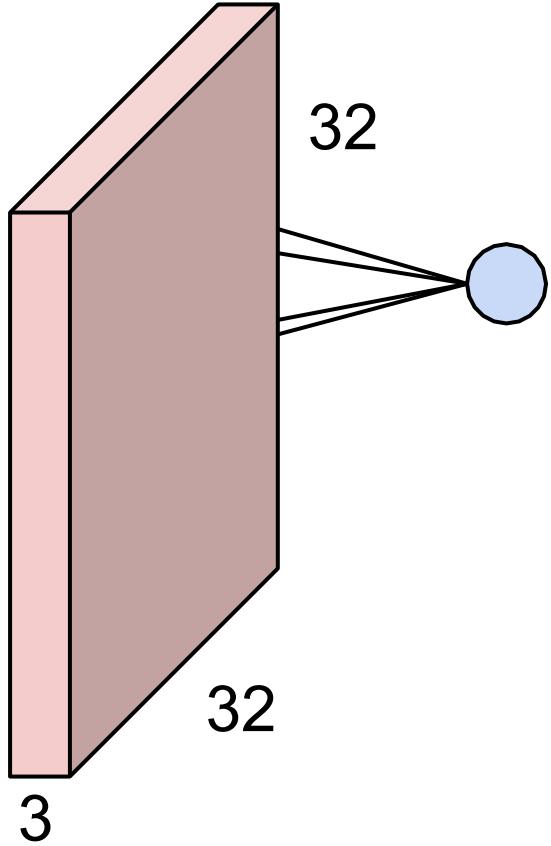
Filters always extend the full depth of the input volume

Convolve the filter with the image
i.e. “slide over the image spatially,
computing dot products”

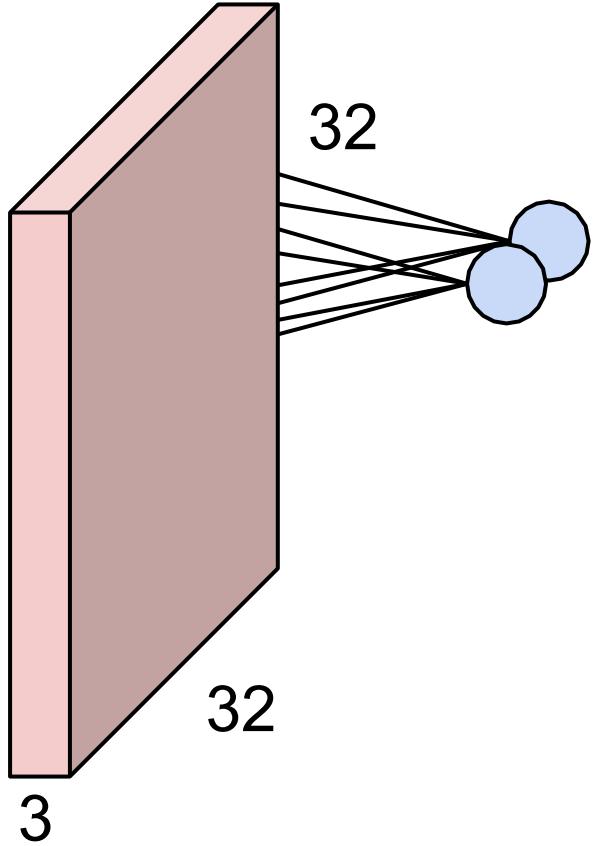
Convolution Layer



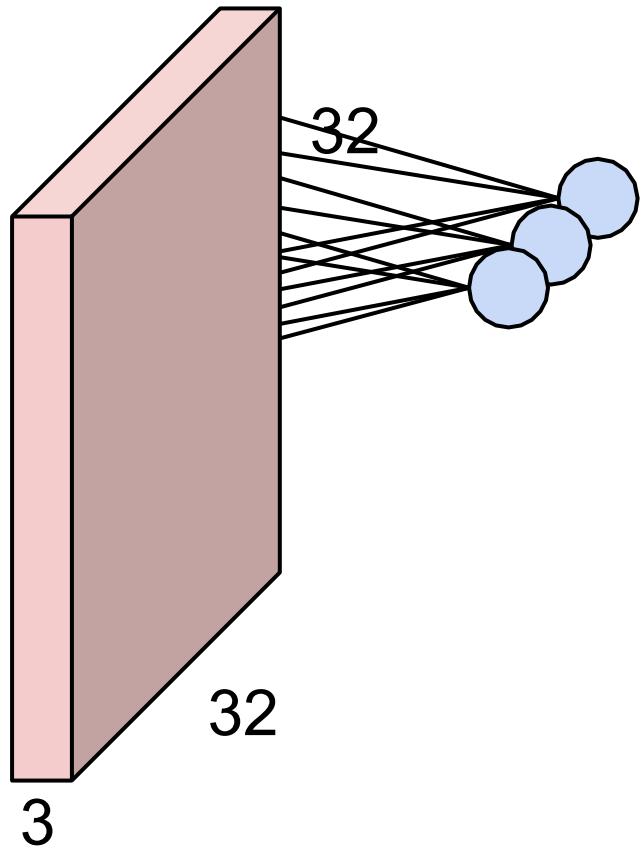
Convolution Layer



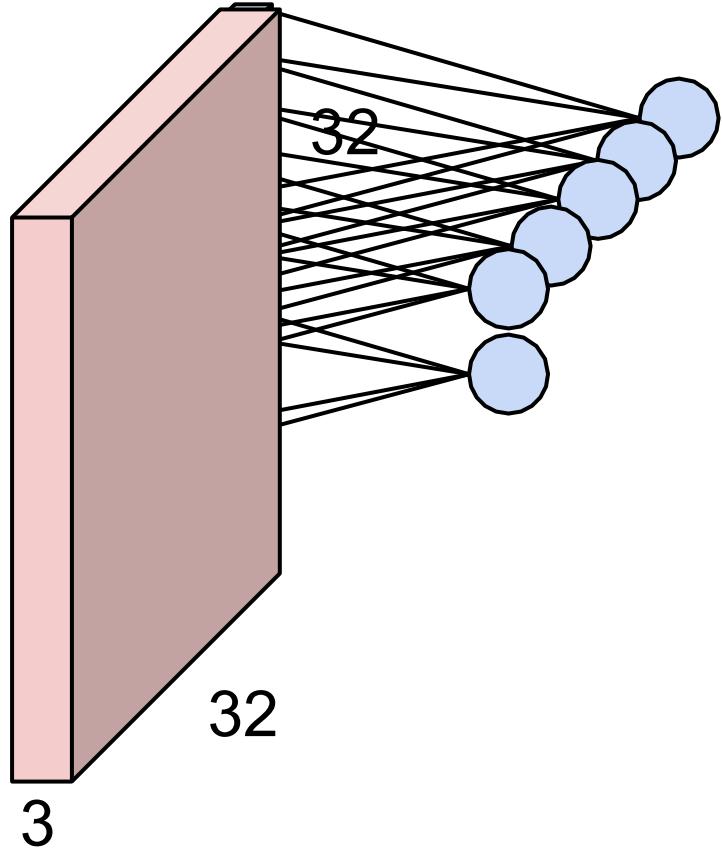
Convolution Layer



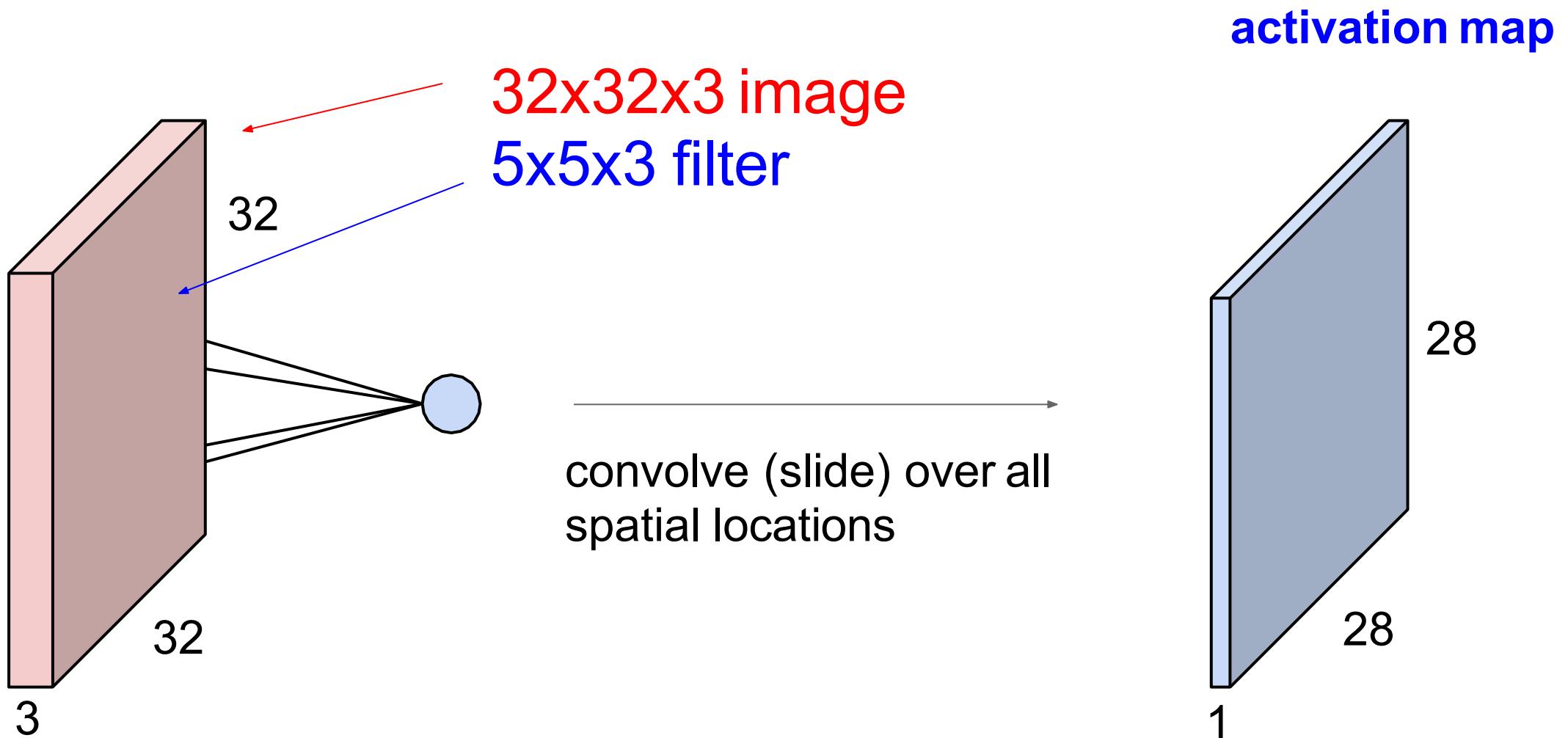
Convolution Layer



Convolution Layer



Convolution Layer

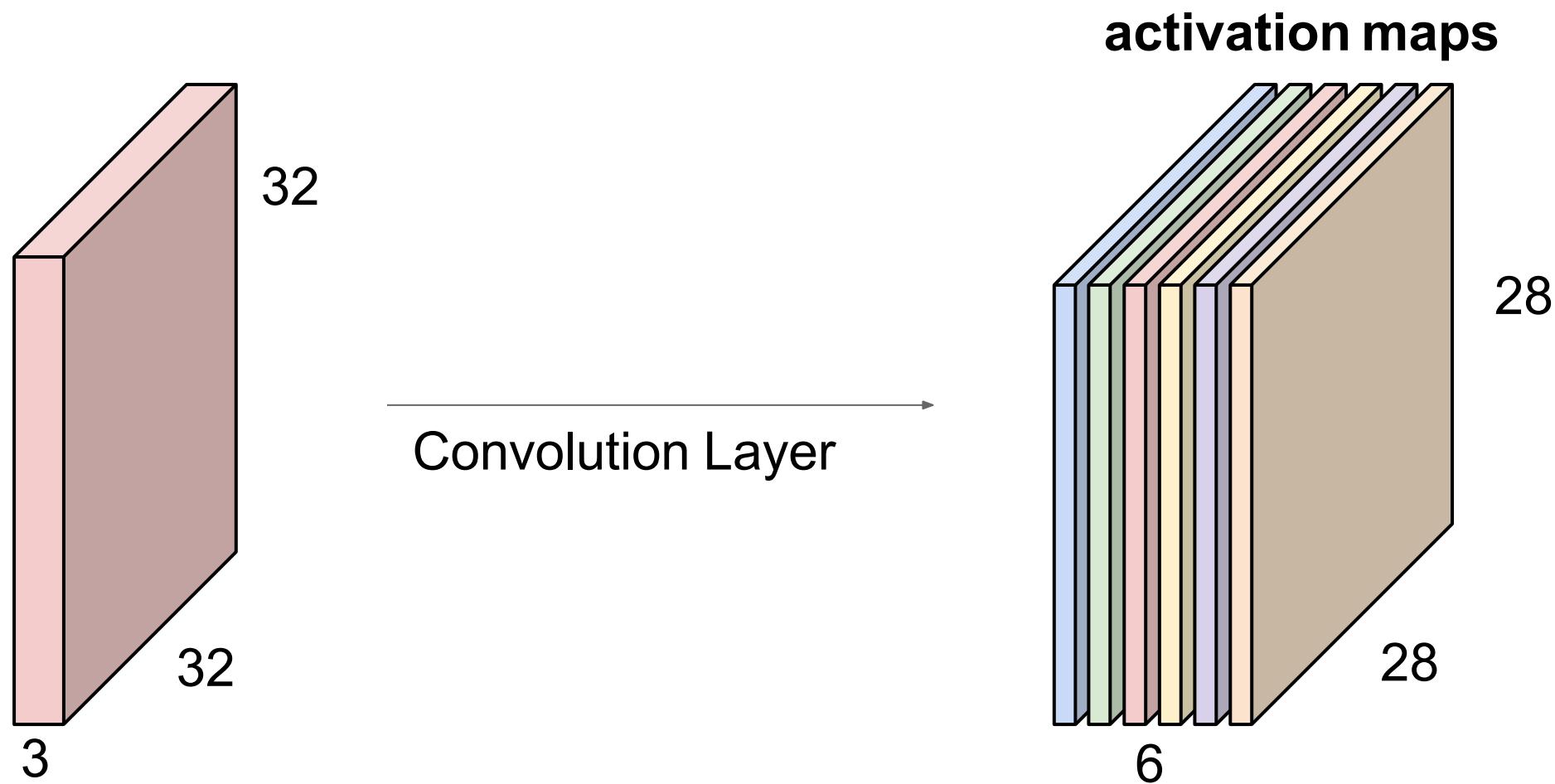


Convolution Layer

consider a second, green filter

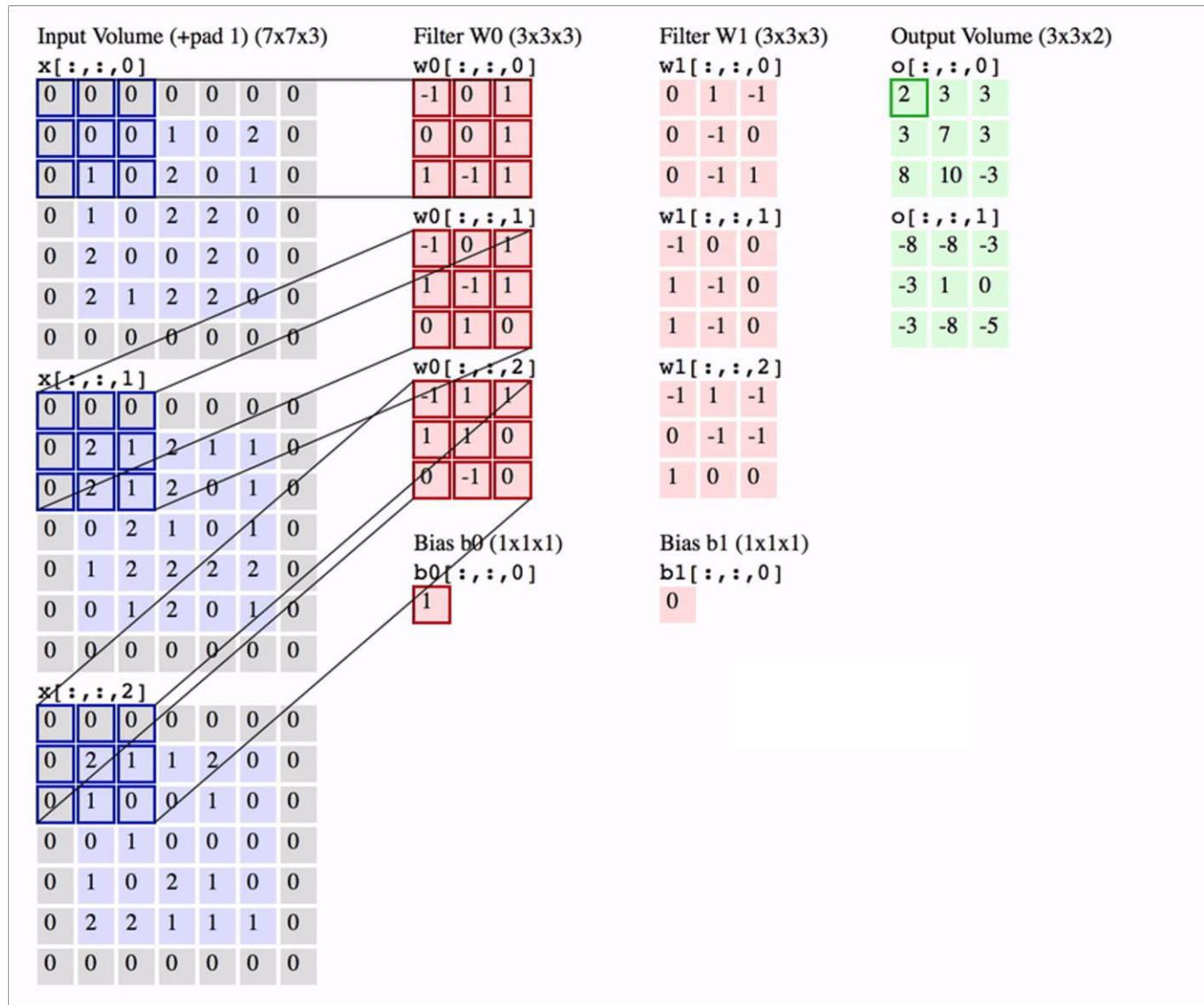


For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

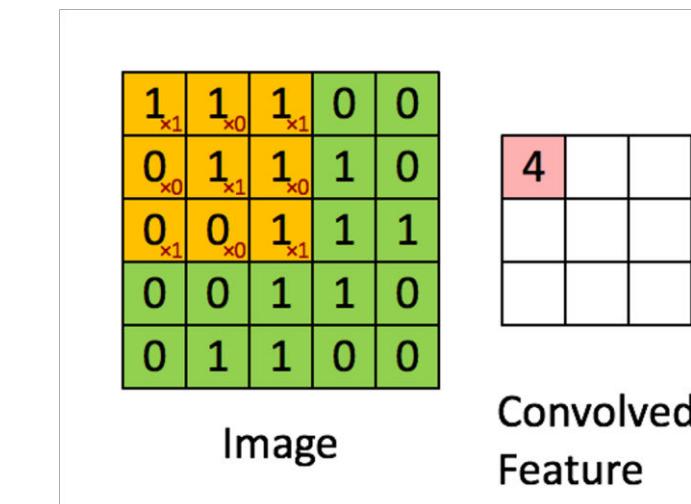


We stack these up to get a “new image” of size 28x28x6!

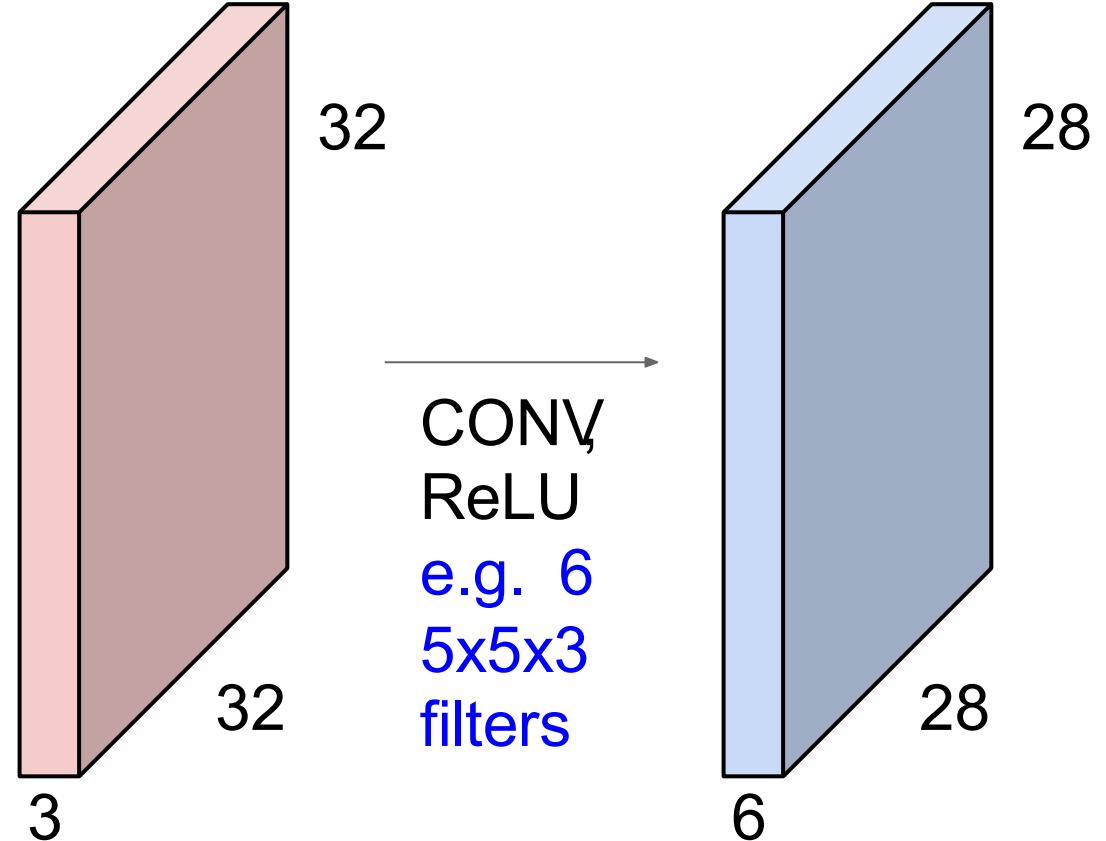
Another View



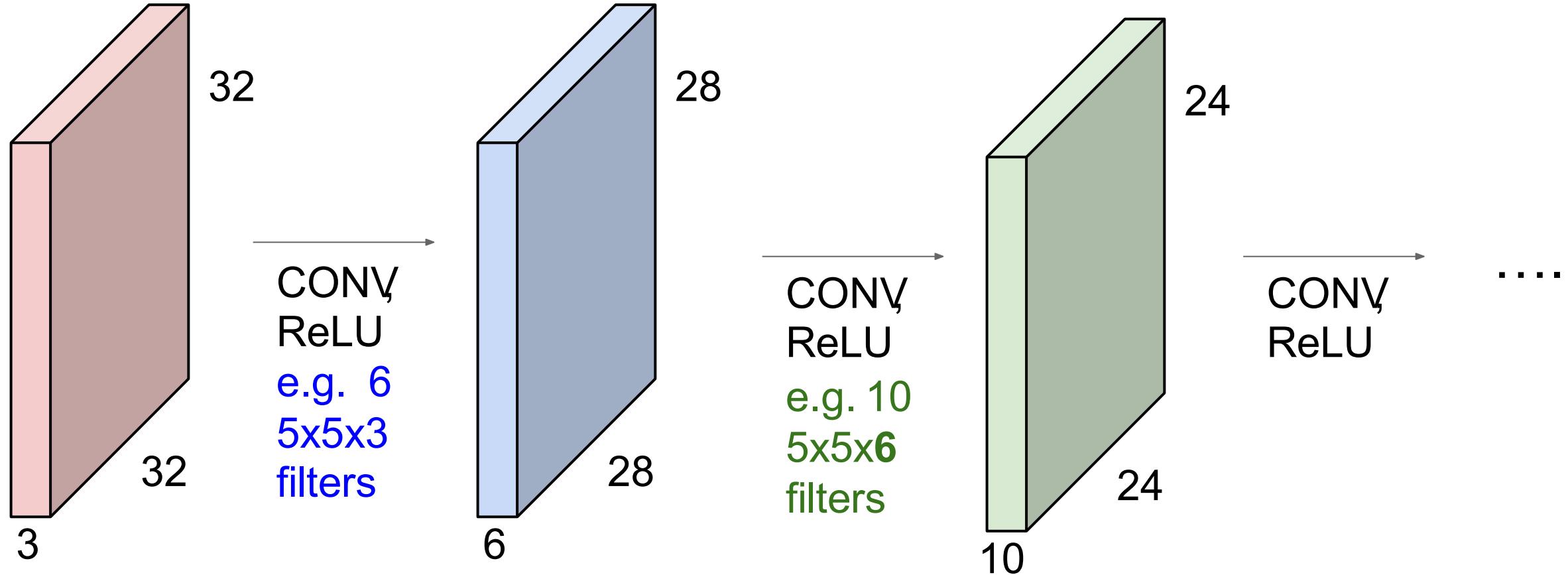
An example



Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions



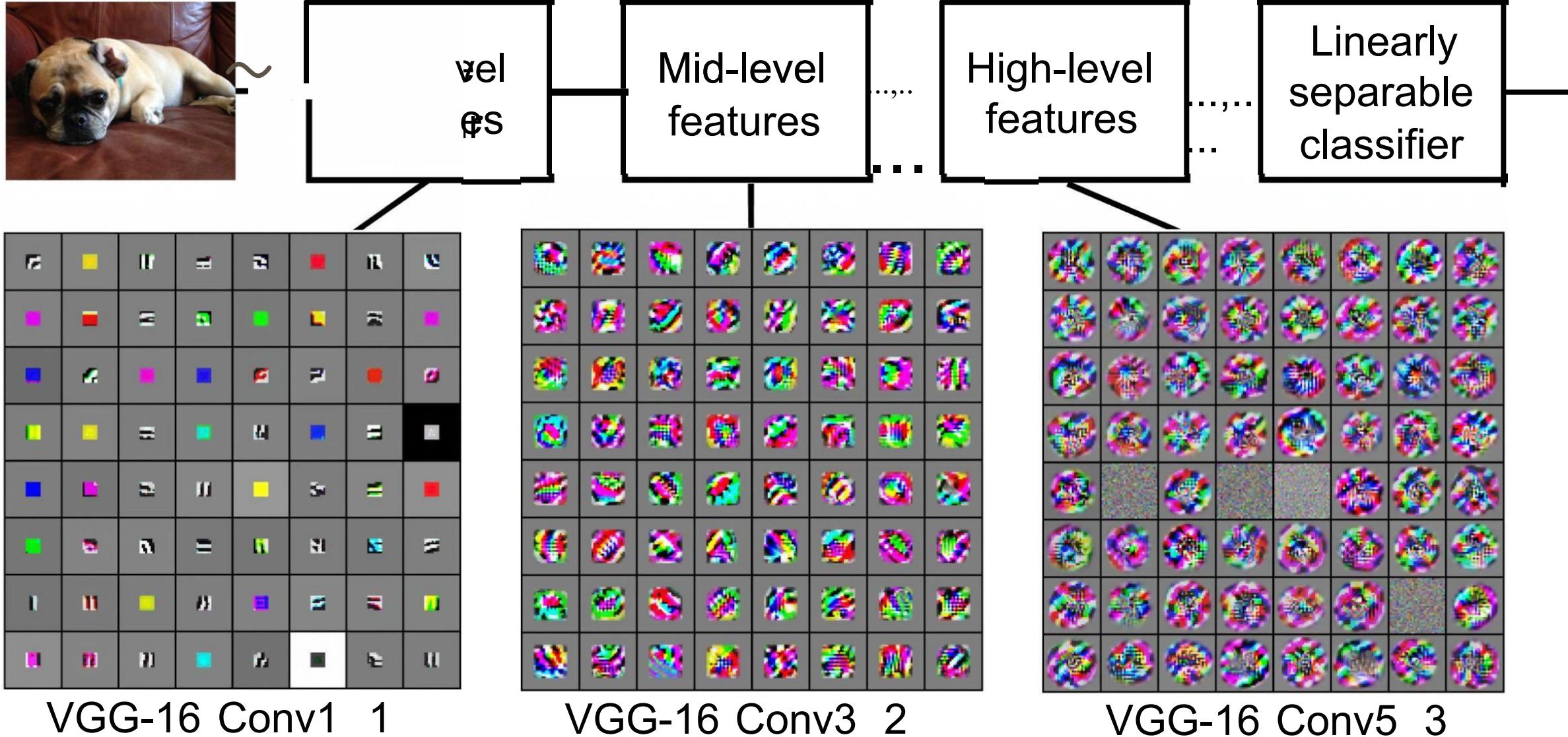
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

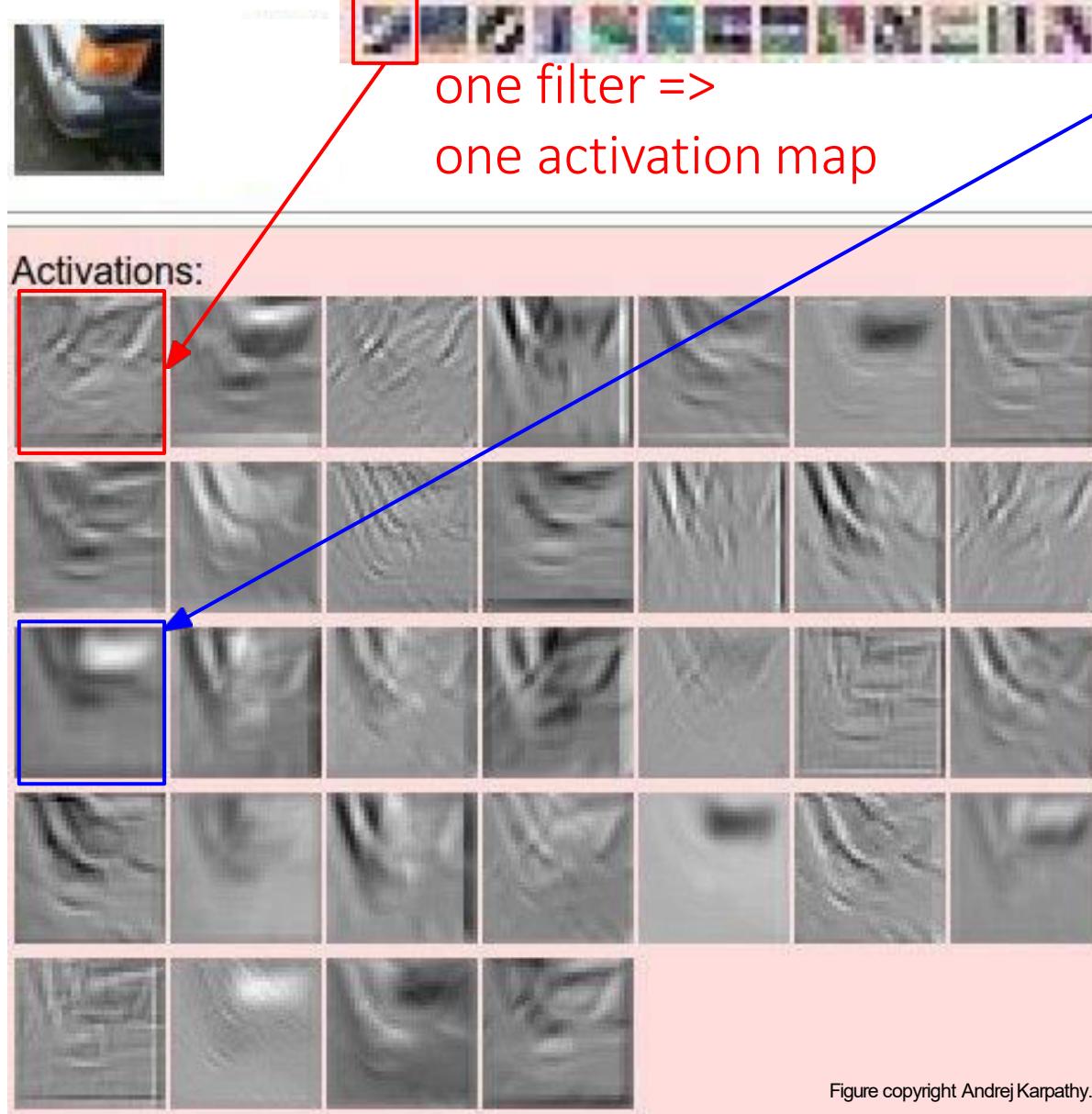


Preview

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman2014].





example 5x5 filters
(32 total)

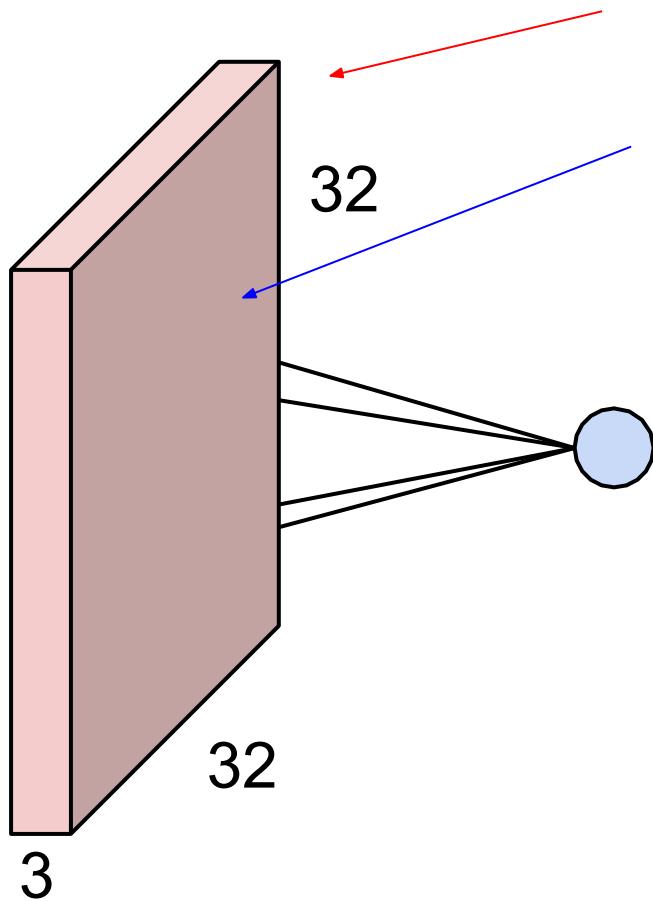
We call the layer convolutional
because it is related to convolution
of two signals:

$$f[x,y] * g[x,y] = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2]$$



elementwise multiplication and sum of
a filter and the signal (image)

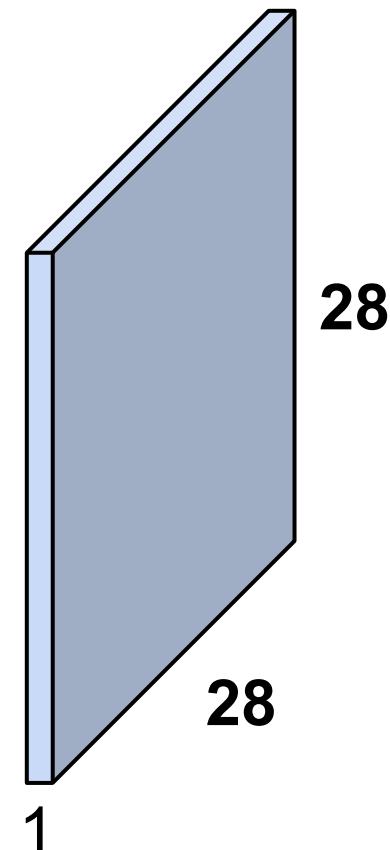
A closer look at spatial dimensions:



32x32x3 image
5x5x3 filter

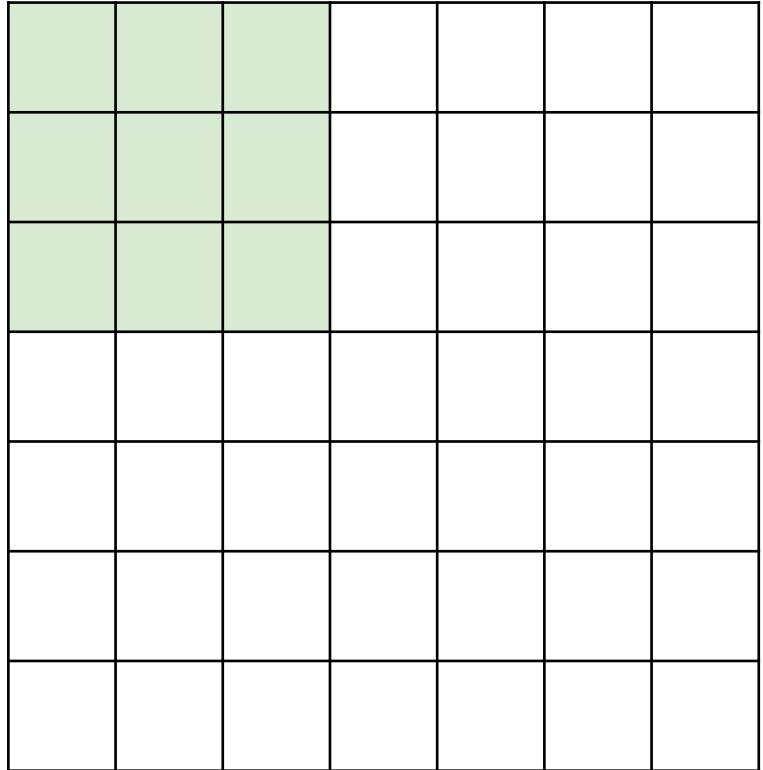
convolve (slide) over all
spatial locations

activation map



A closer look at spatial dimensions:

7



7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

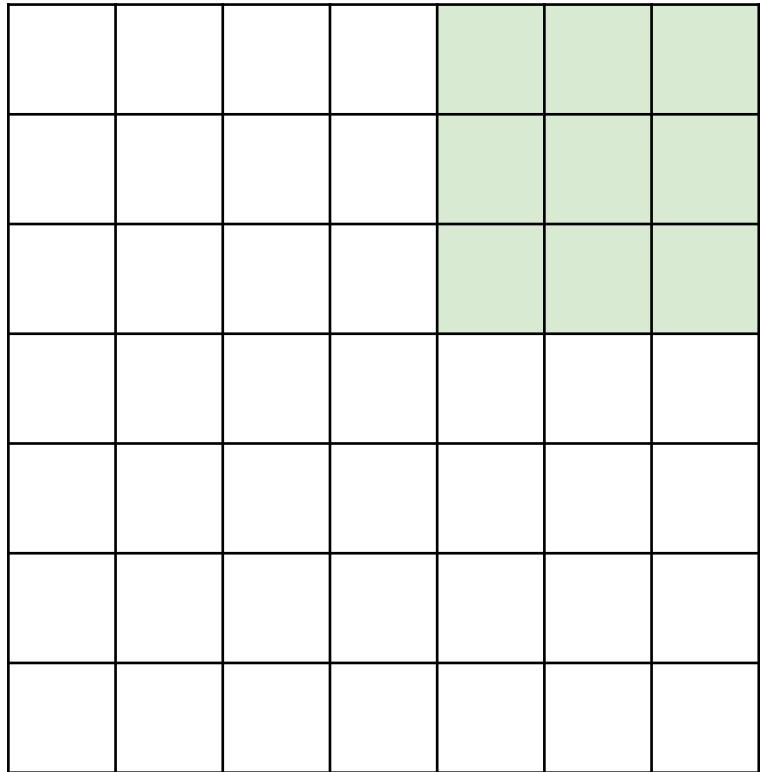
7

7x7 input (spatially)
assume 3x3 filter

7

A closer look at spatial dimensions:

7

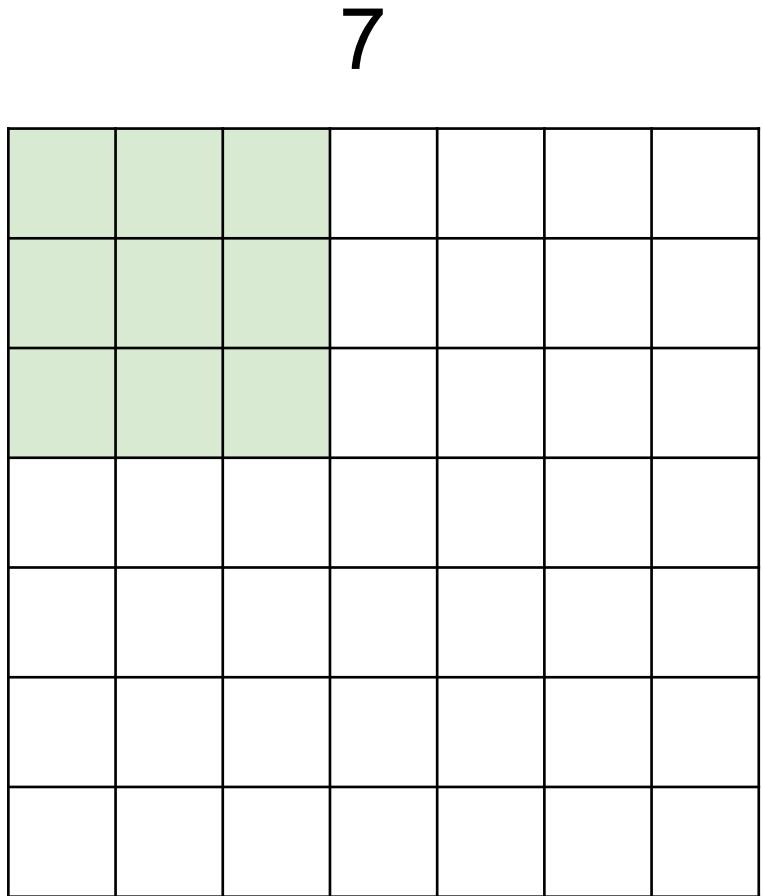


7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

7

A closer look at spatial dimensions:



7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

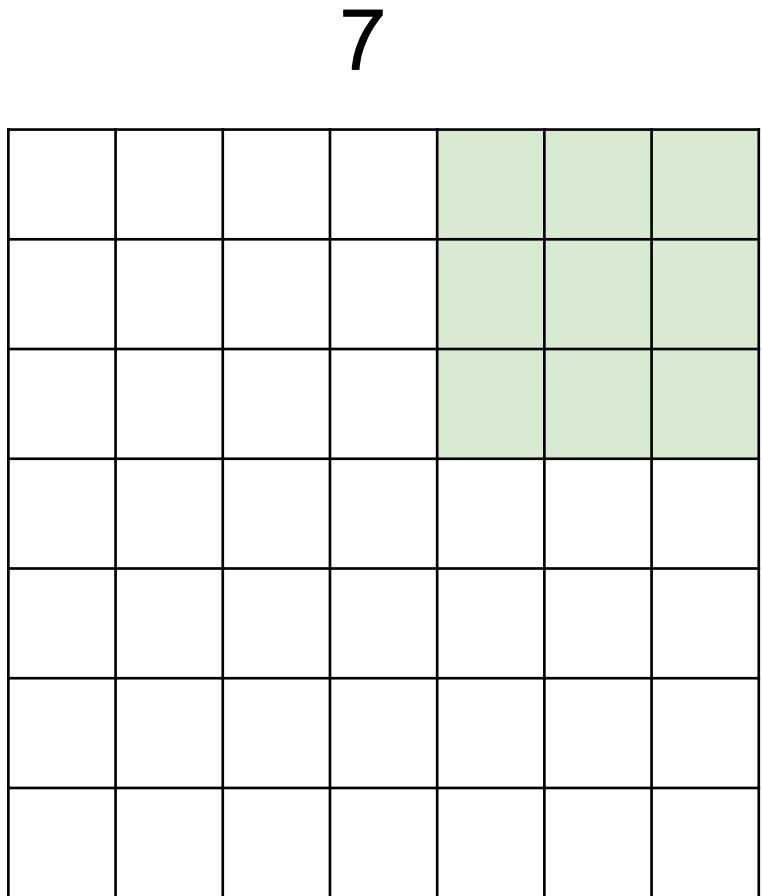
A closer look at spatial dimensions:

7

7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

A closer look at spatial dimensions:

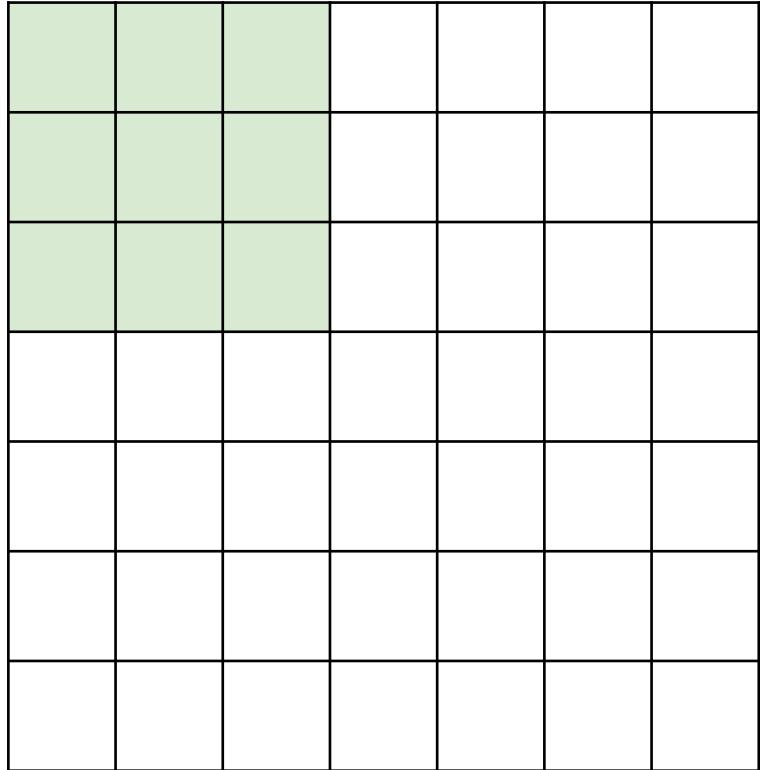


7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

A closer look at spatial dimensions:

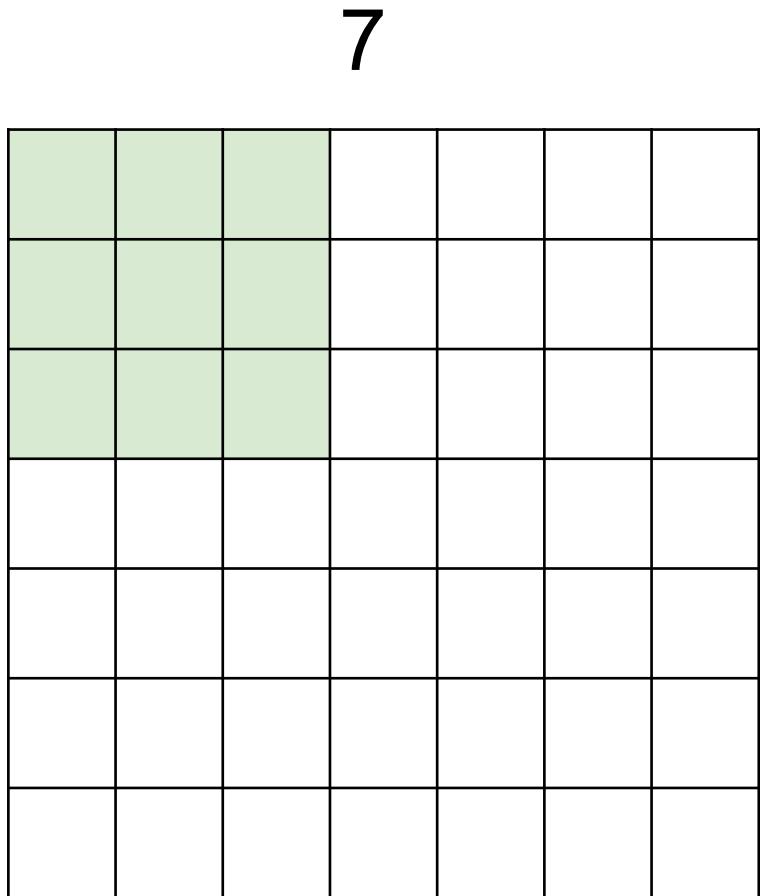
7



7

7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

A closer look at spatial dimensions:

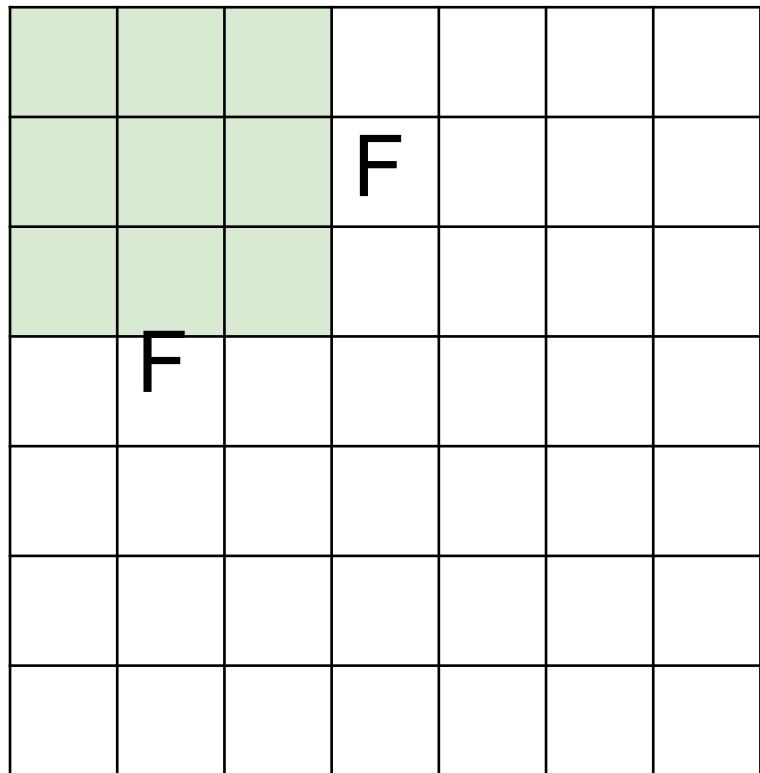


7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

7

doesn't fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

N



N

Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:

stride 1 => $(7 - 3)/1 + 1 = 5$

stride 2 => $(7 - 3)/2 + 1 = 3$

stride 3 => $(7 - 3)/3 + 1 = 2.33 :\backslash$

In practice: Common to zero pad the border

0	0	0	0	0	0				
0									
0									
0									
0									

e.g. input 7x7

3x3 filter, applied with stride 1

pad with 1 pixel border => what is the output?

(recall:)

$$(N - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0				
0									
0									
0									
0									

e.g. input 7x7

3x3 filter, applied with stride 1

pad with 1 pixel border => what is the output?

7x7 output!

(recall:)

$$(N + 2P - F) / \text{stride} + 1$$

In practice: Common to zero pad the border

0	0	0	0	0	0				
0									
0									
0									
0									

e.g. input 7x7

3x3 filter, applied with **stride 1**

pad with 1 pixel border => what is the output?

7x7 output!

in general, common to see CONV layers with stride 1, filters of size $F \times F$ and zero-padding with $(F-1)/2$. (will preserve size spatially)

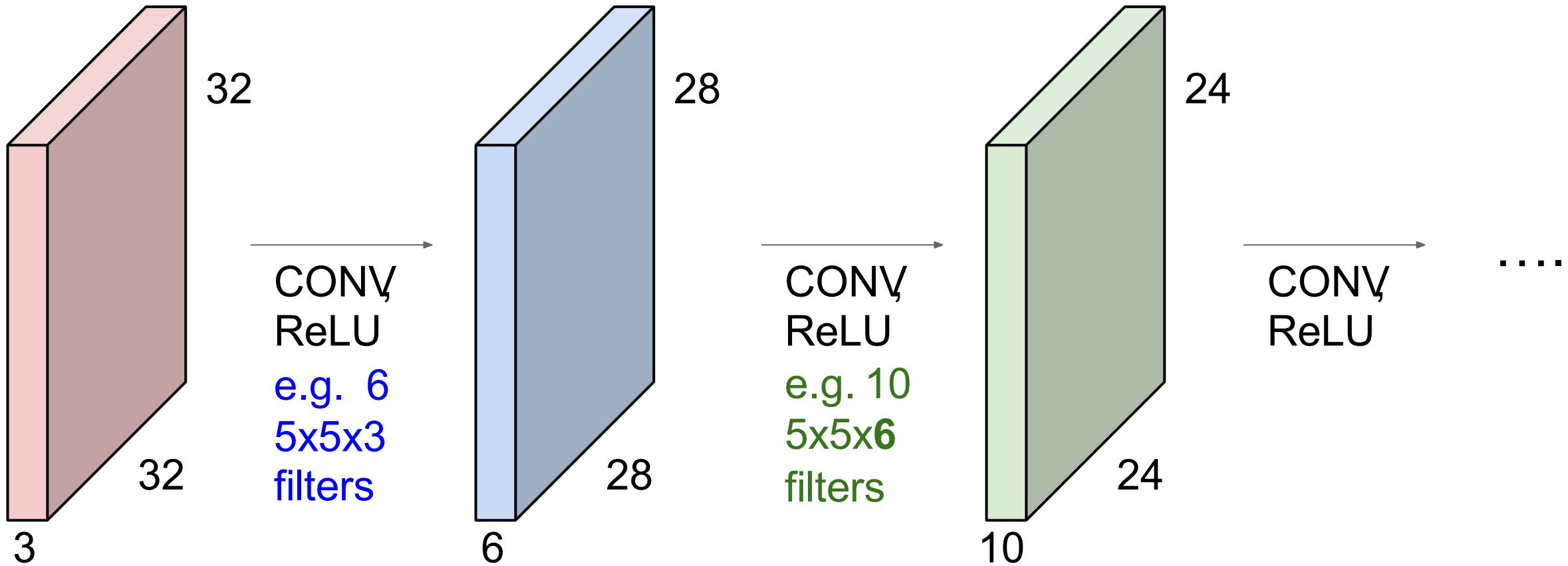
e.g. $F = 3 \Rightarrow$ zero pad with 1

$F = 5 \Rightarrow$ zero pad with 2

$F = 7 \Rightarrow$ zero pad with 3

Remember back to...

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially!
(32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

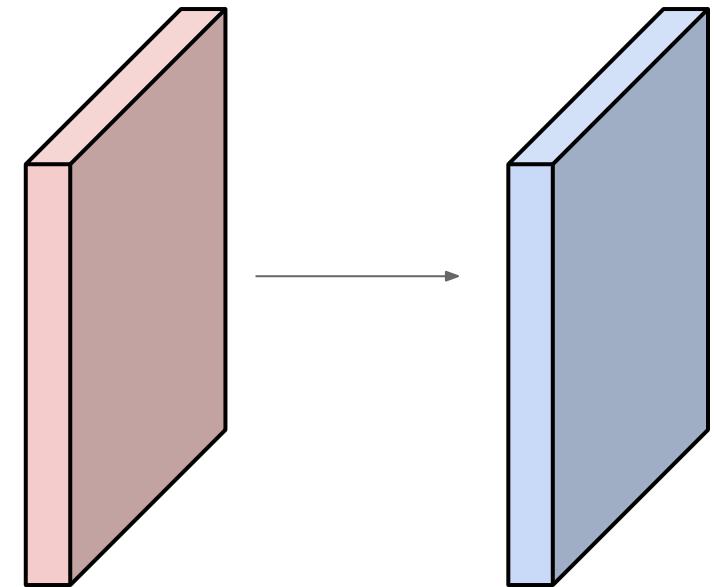


Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

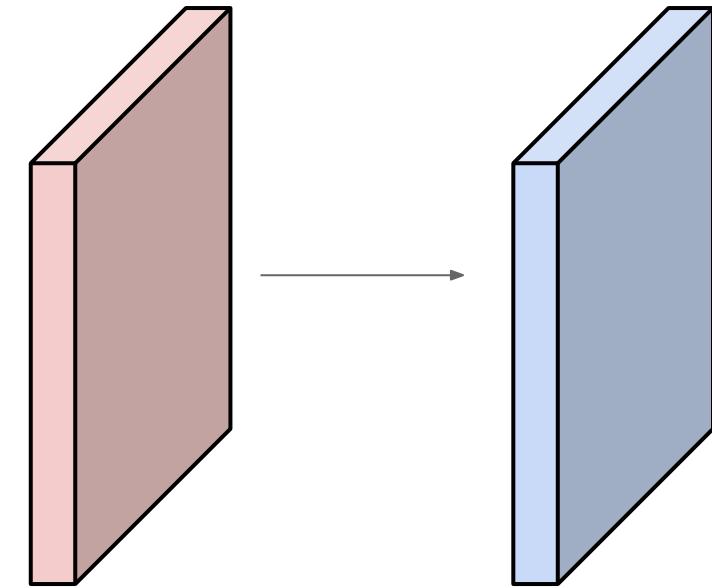
Output volume size: ?



Examples time:

Input volume: **32x32x3**

10 **5x5** filters with stride **1**, pad **2**



Output volume size:

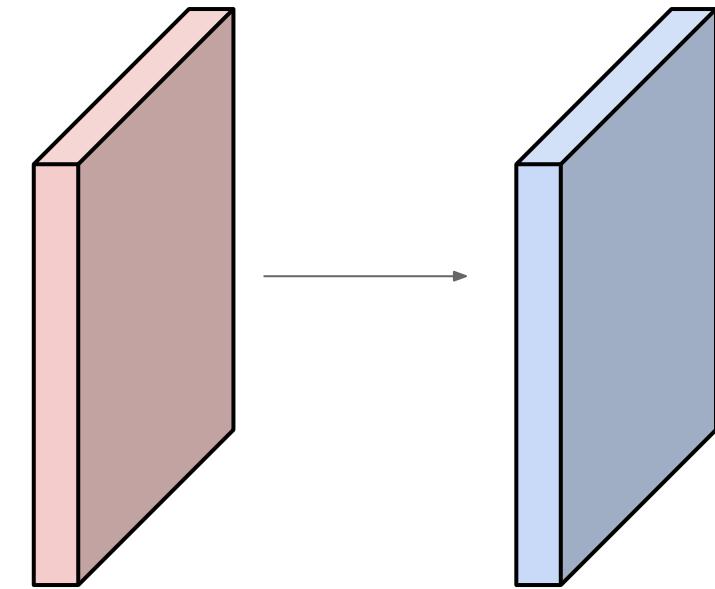
$(32+2*2-5)/1+1 = 32$ spatially, so

32x32x10

Examples time:

Input volume: **32x32x3**

10 5x5 filters with stride 1, pad 2

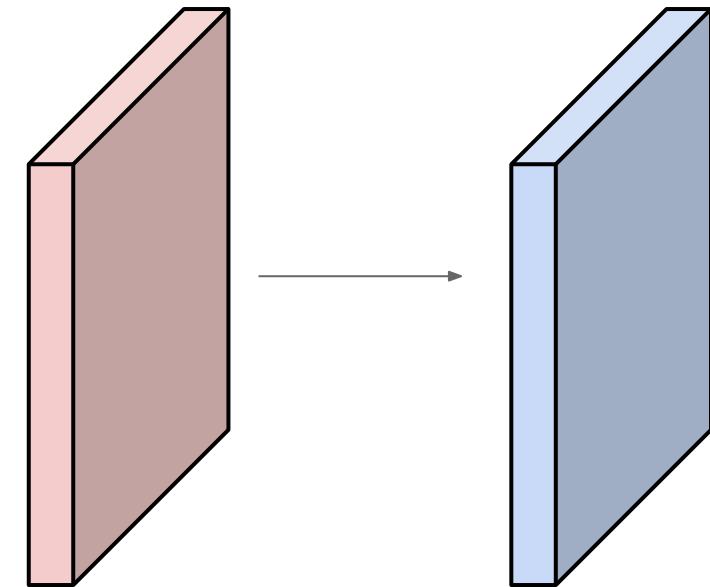


Number of parameters in this layer?

Examples time:

Input volume: **32x32x3**

10 **5x5** filters with stride 1, pad 2



Number of parameters in this layer?

each filter has $5*5*3 + 1 = 76$ params

(+1 for bias)

$$\Rightarrow 76 * 10 = 760$$

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of $W_2 \times H_2 \times K$

where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Convolution layer: summary

Let's assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters **K**
- The filter size **F**
- The stride **S**
- The zero padding **P**

This will produce an output of $W_2 \times H_2 \times K$

where:

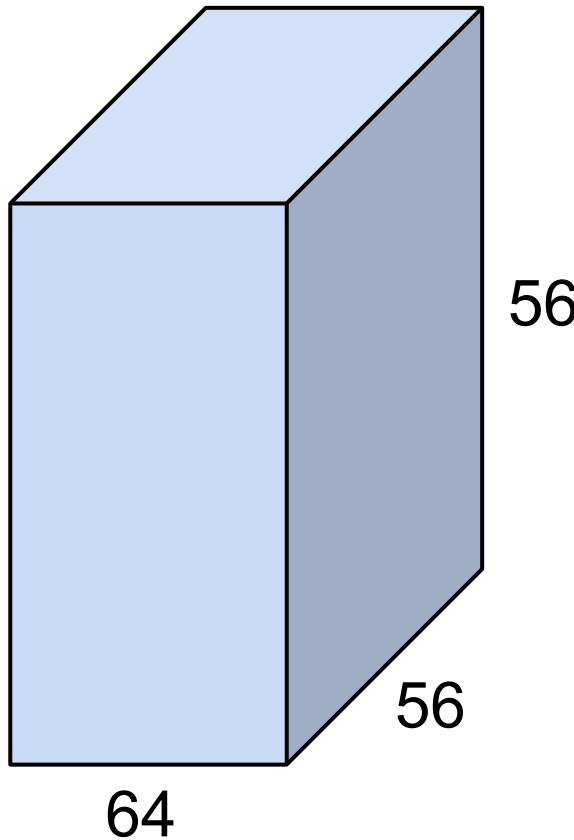
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: F^2CK and K biases

Common settings:

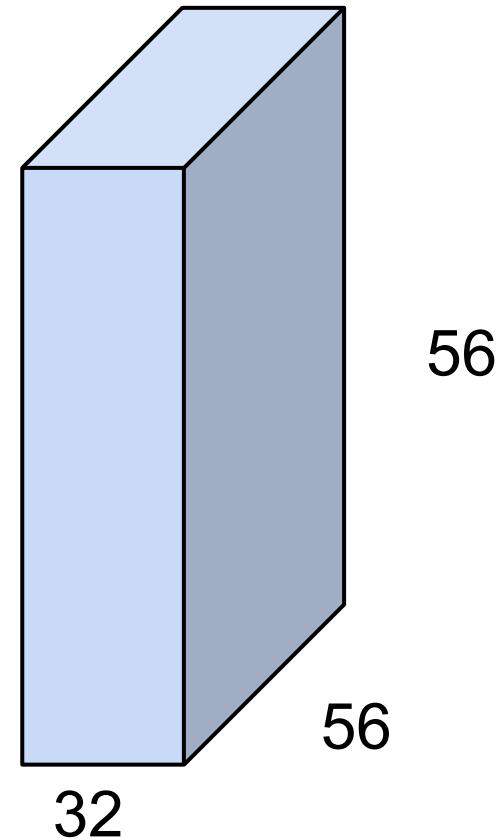
- K** = (powers of 2, e.g. 32, 64, 128, 512)
- $F = 3, S = 1, P = 1$
 - $F = 5, S = 1, P = 2$
 - $F = 5, S = 2, P = ?$ (whatever fits)
 - $F = 1, S = 1, P = 0$

(btw, 1x1 convolution layers make perfect sense)

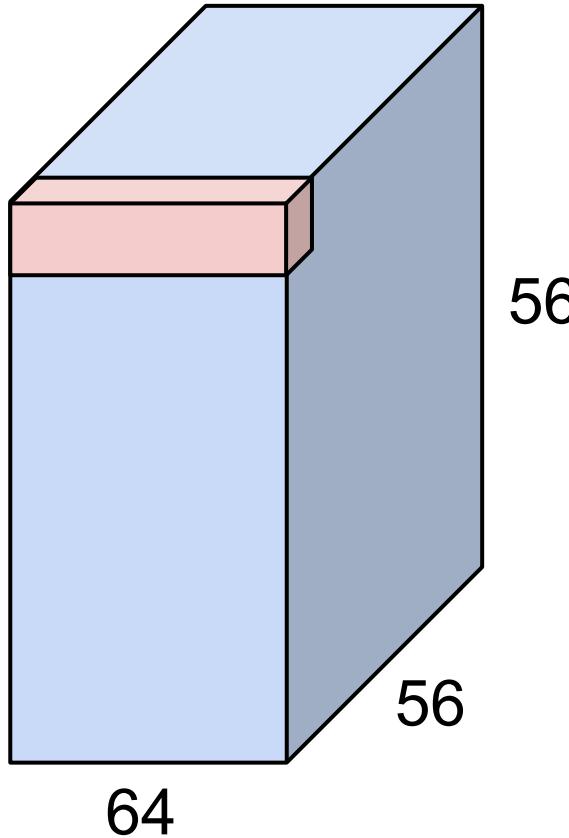


1x1 CONV
with 32 filters

→
(each filter has size
1x1x64, and performs a
64-dimensional dot
product)

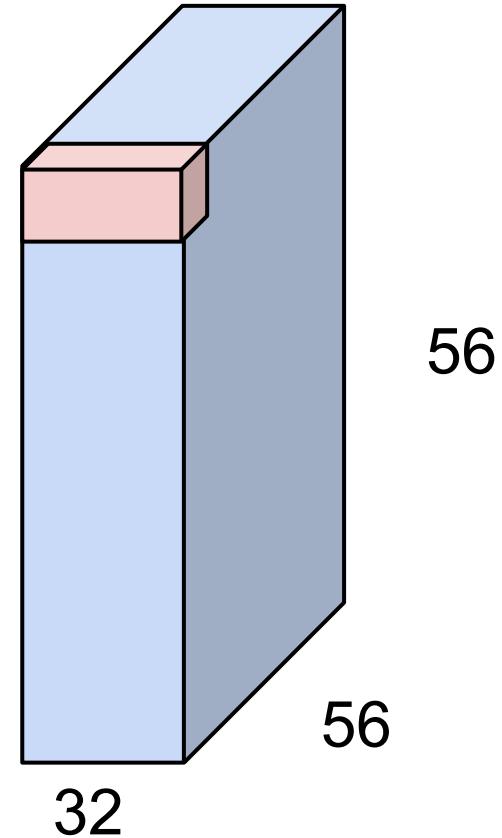


(btw, 1x1 convolution layers make perfect sense)



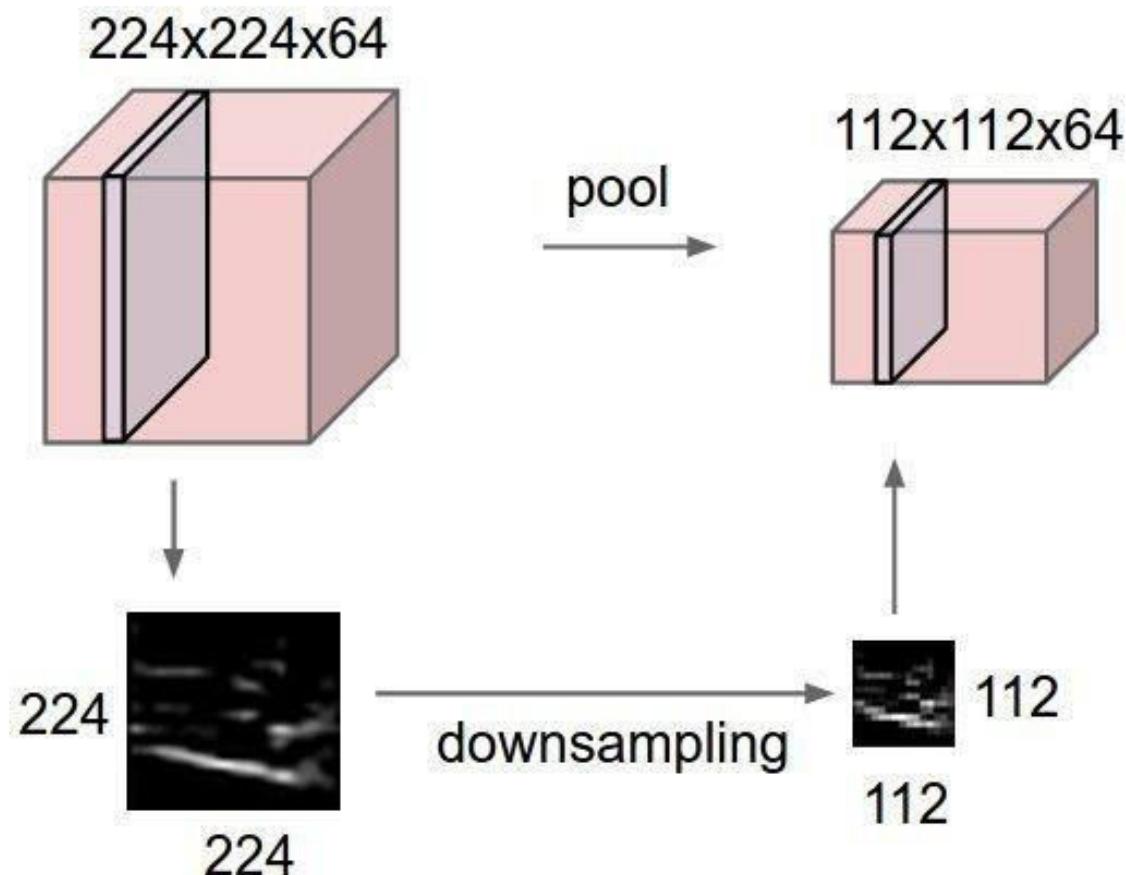
1x1 CONV
with 32 filters

(each filter has size
1x1x64, and performs a
64-dimensional dot
product)



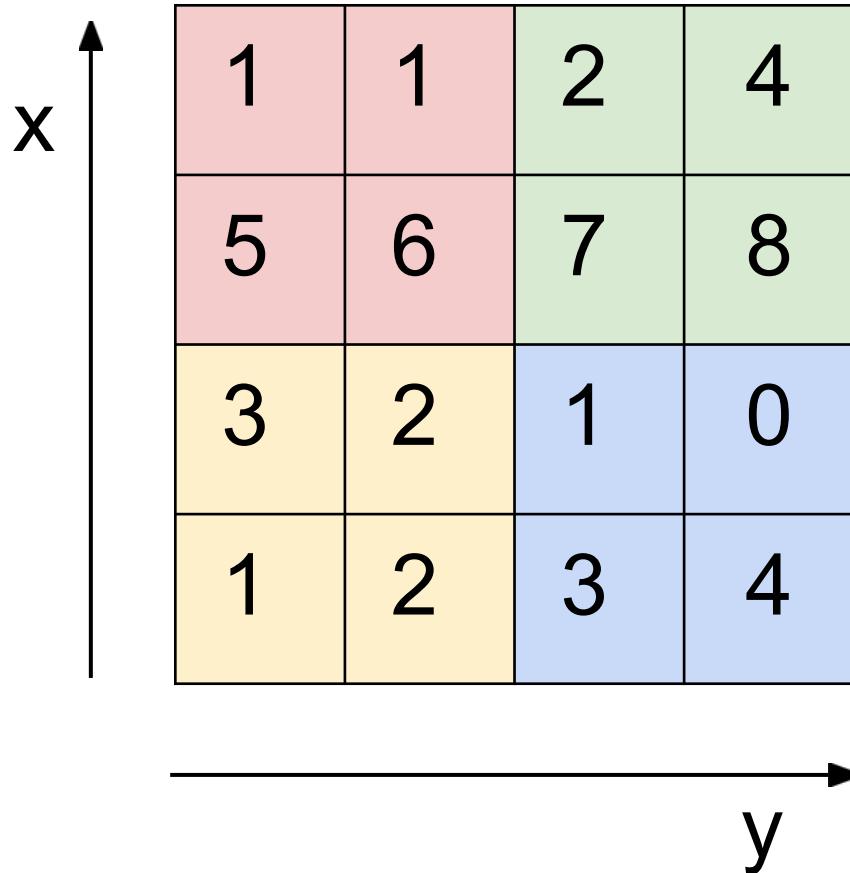
Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:



MAX POOLING

Single depth slice

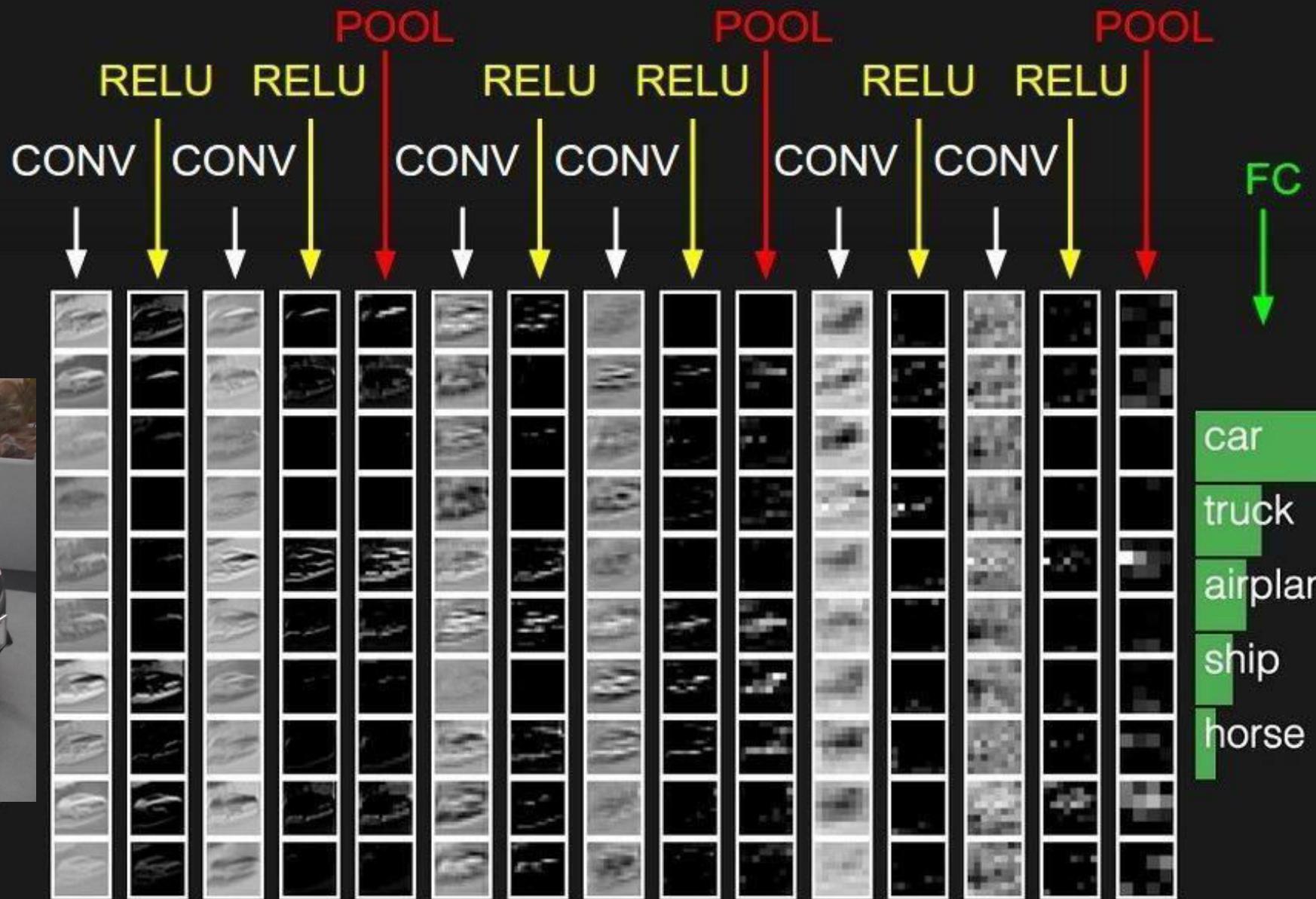


max pool with 2x2 filters
and stride 2

A 2x2 grid representing the output of the max pooling operation. It contains four cells: top-left (6) is pink, top-right (8) is light green, bottom-left (3) is yellow, and bottom-right (4) is blue. An arrow points from the input grid to this output grid.

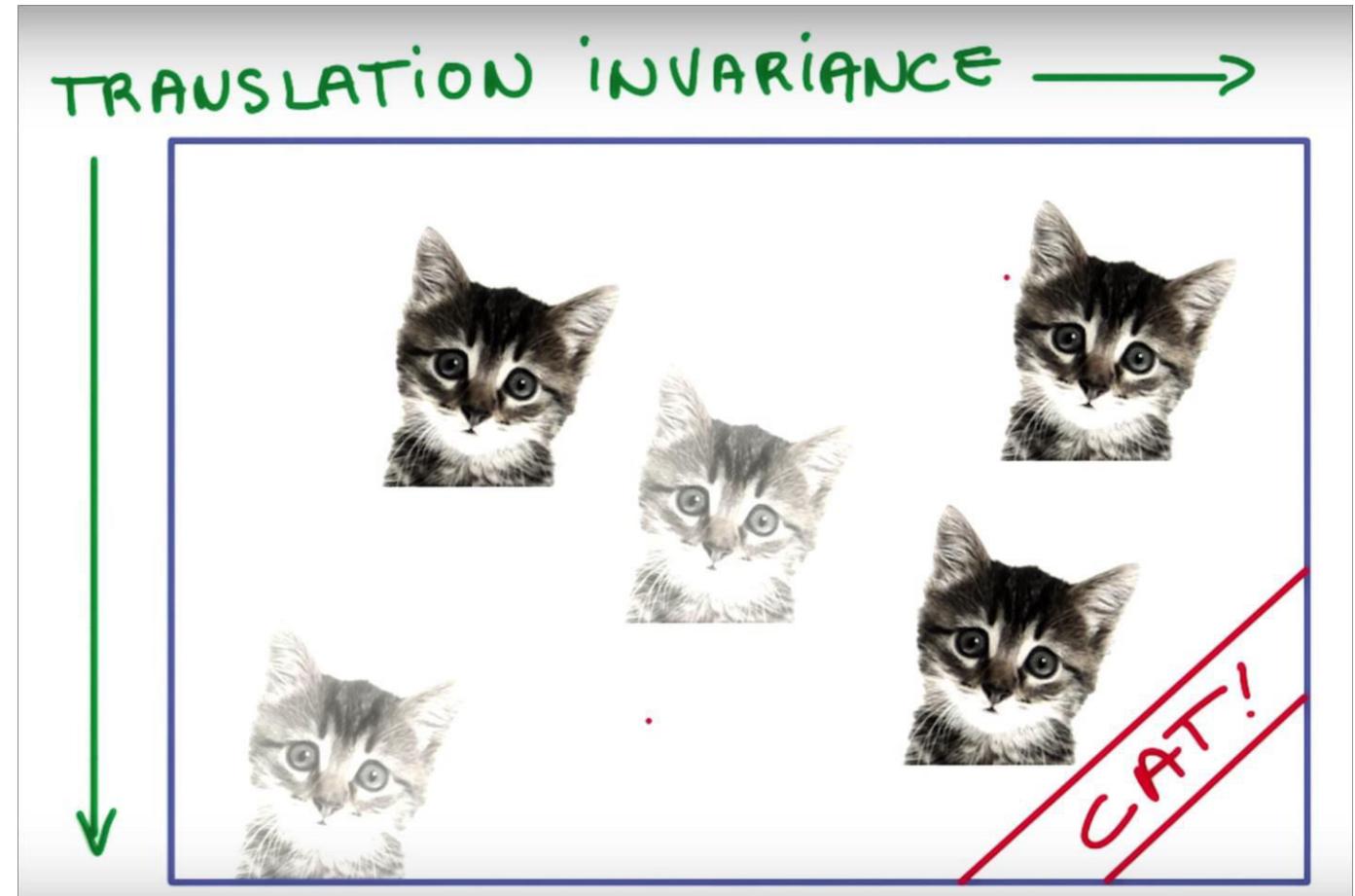
6	8
3	4

preview:



Properties of CNN: Translational invariance

Since we are training filters to detect cats and the moving these filters over the data, a differently positioned cat will also get detected by the same set of filters.



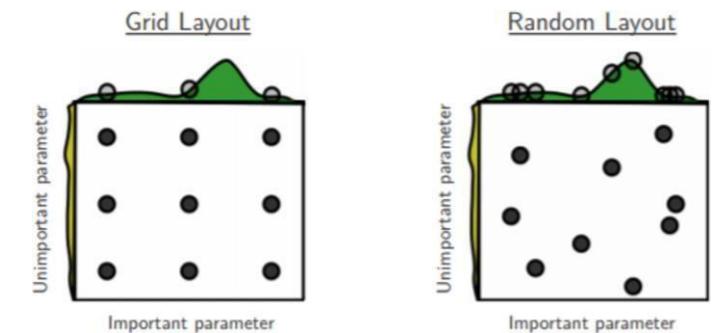
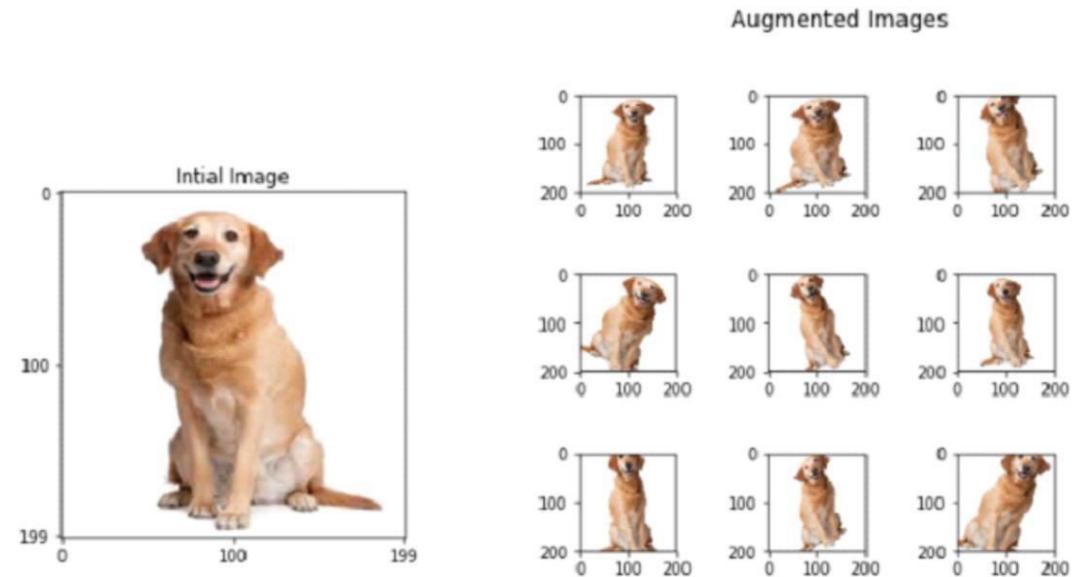
Tricks for training

Data augmentation if your data set is smaller. This helps the network generalize more.

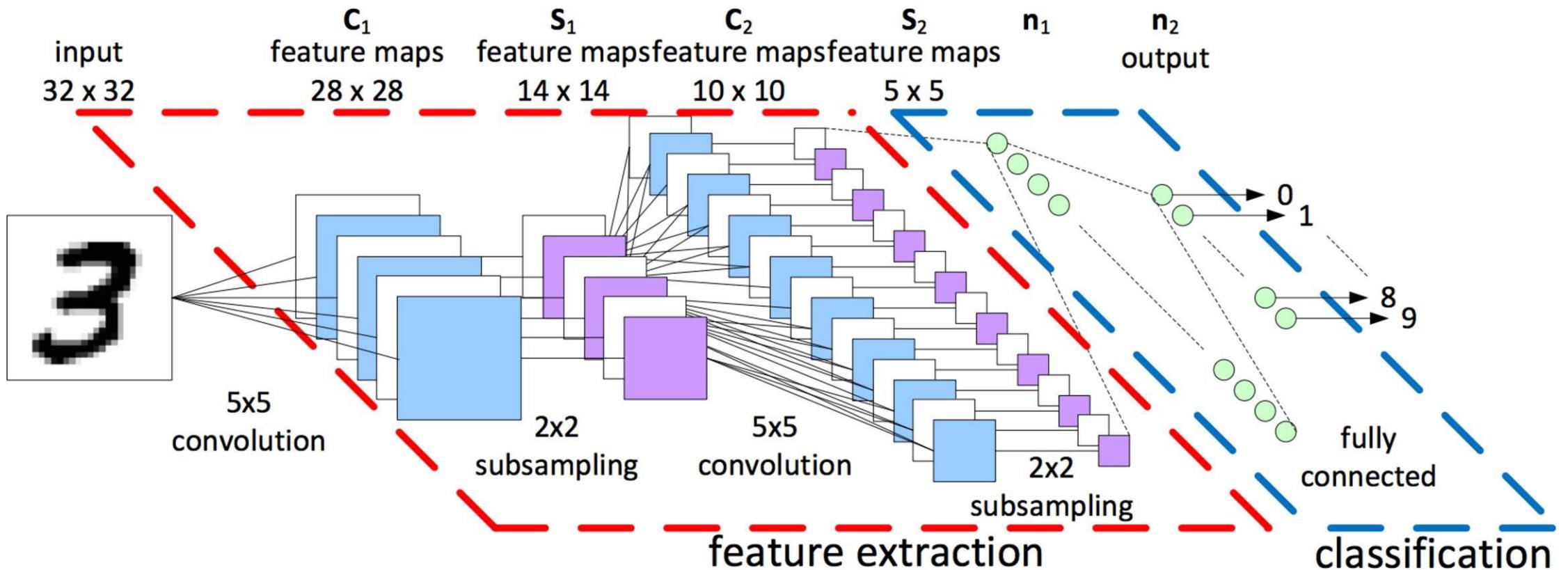
Early stopping if validation loss stops improving.

Tune hyperparameters:

- Random search
- Grid search



CNN: Recap



Let convolutions extract features and let normal cnn's decide on them.

Summary

- ConvNets stack CONV,POOL,FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like

$[(\text{CONV-RELU})^*N - \text{POOL?}]^*M - (\text{FC-RELU})^*K, \text{SOFTMAX}$

where N is usually up to ~5, M is large, $0 \leq K \leq 2$.

- but recent advances such as ResNet/GoogLeNet have challenged this paradigm



Lambton
College

In Toronto



CESTAR COLLEGE
of Business, Health & Technology

Deep Sequence Modeling

Ava Soleimany

MIT 6.S191

January 27, 2020



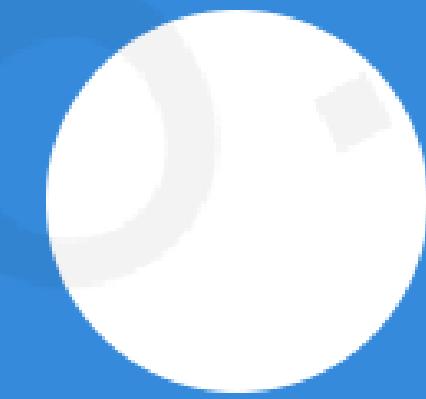
6.S191 Introduction to Deep Learning

introtodeeplearning.com

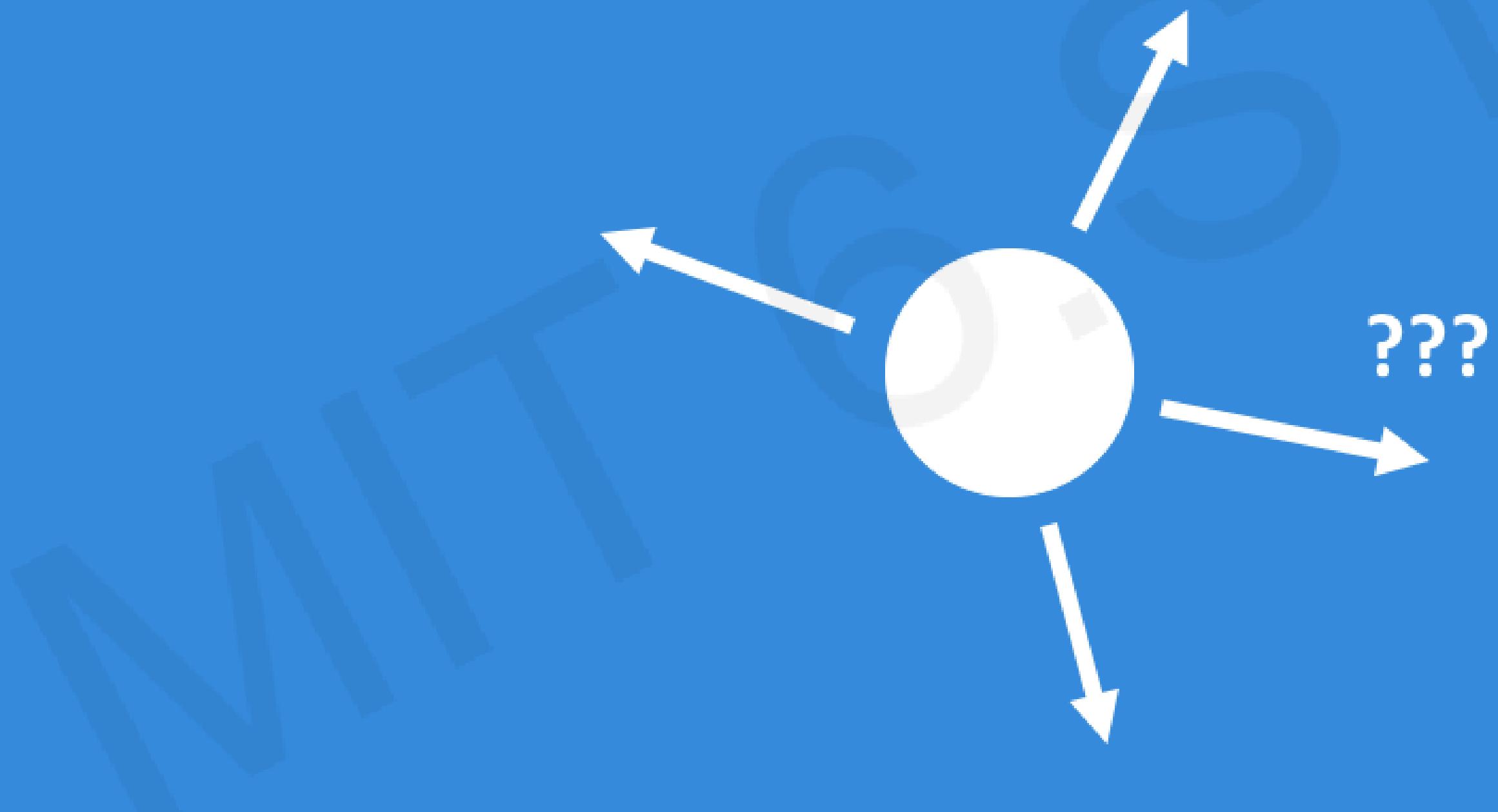
@MITDeepLearning



Given an image of a ball,
can you predict where it will go next?



Given an image of a ball,
can you predict where it will go next?



Given an image of a ball,
can you predict where it will go next?



Given an image of a ball,
can you predict where it will go next?



A Sequence Modeling Problem: Predict the Next Word

A Sequence Modeling Problem: Predict the Next Word

“This morning I took my cat for a walk.”

A Sequence Modeling Problem: Predict the Next Word

“This morning I took my cat for a walk.”

given these words

A Sequence Modeling Problem: Predict the Next Word

“This morning I took my cat for a walk.”

given these words

predict the
next word

Idea #1: Use a Fixed Window

“This morning I took my cat for a walk.”

given these
two words

predict the
next word

Idea #1: Use a Fixed Window

“This morning I took my cat for a walk.”

given these
two words predict the
next word

One-hot feature encoding: tells us what each word is

[1 0 0 0 0 0 1 0 0 0]

for a



prediction

Problem #1: Can't Model Long-Term Dependencies

“France is where I grew up, but I now live in Boston. I speak fluent ____.”



We need information from **the distant past** to accurately predict the correct word.

Idea #2: Use Entire Sequence as Set of Counts

“This morning I took my cat for a”



“bag of words”

[0 1 0 0 1 0 0 ... 0 0 1 1 0 0 0 1]



prediction

Problem #2: Counts Don't Preserve Order



The food was good, not bad at all.

vs.

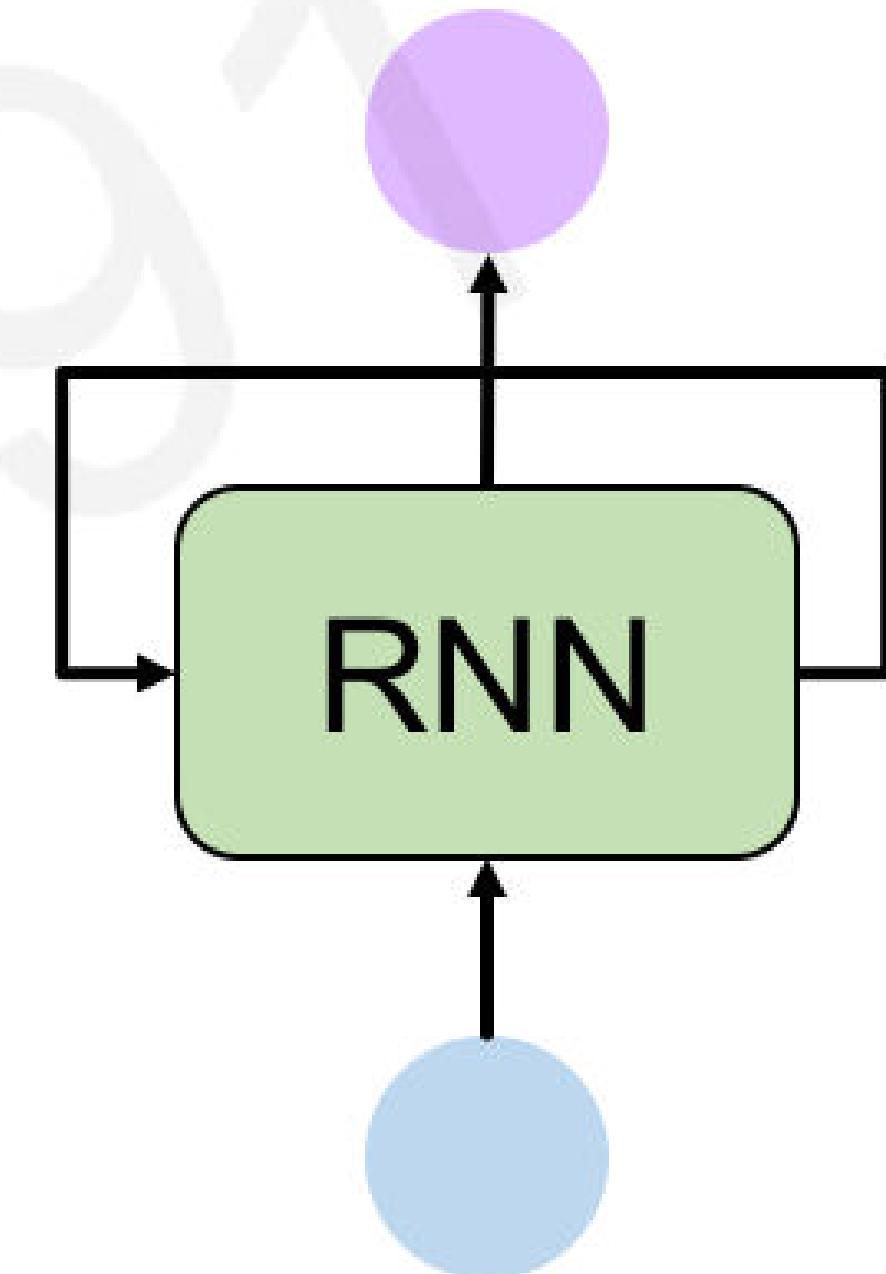
The food was bad, not good at all.



Sequence Modeling: Design Criteria

To model sequences, we need to:

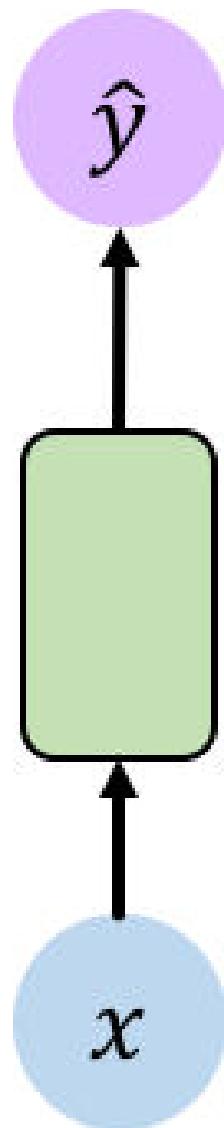
1. Handle **variable-length** sequences
2. Track **long-term** dependencies
3. Maintain information about **order**
4. **Share parameters** across the sequence



Today: **Recurrent Neural Networks (RNNs)** as
an approach to sequence modeling problems

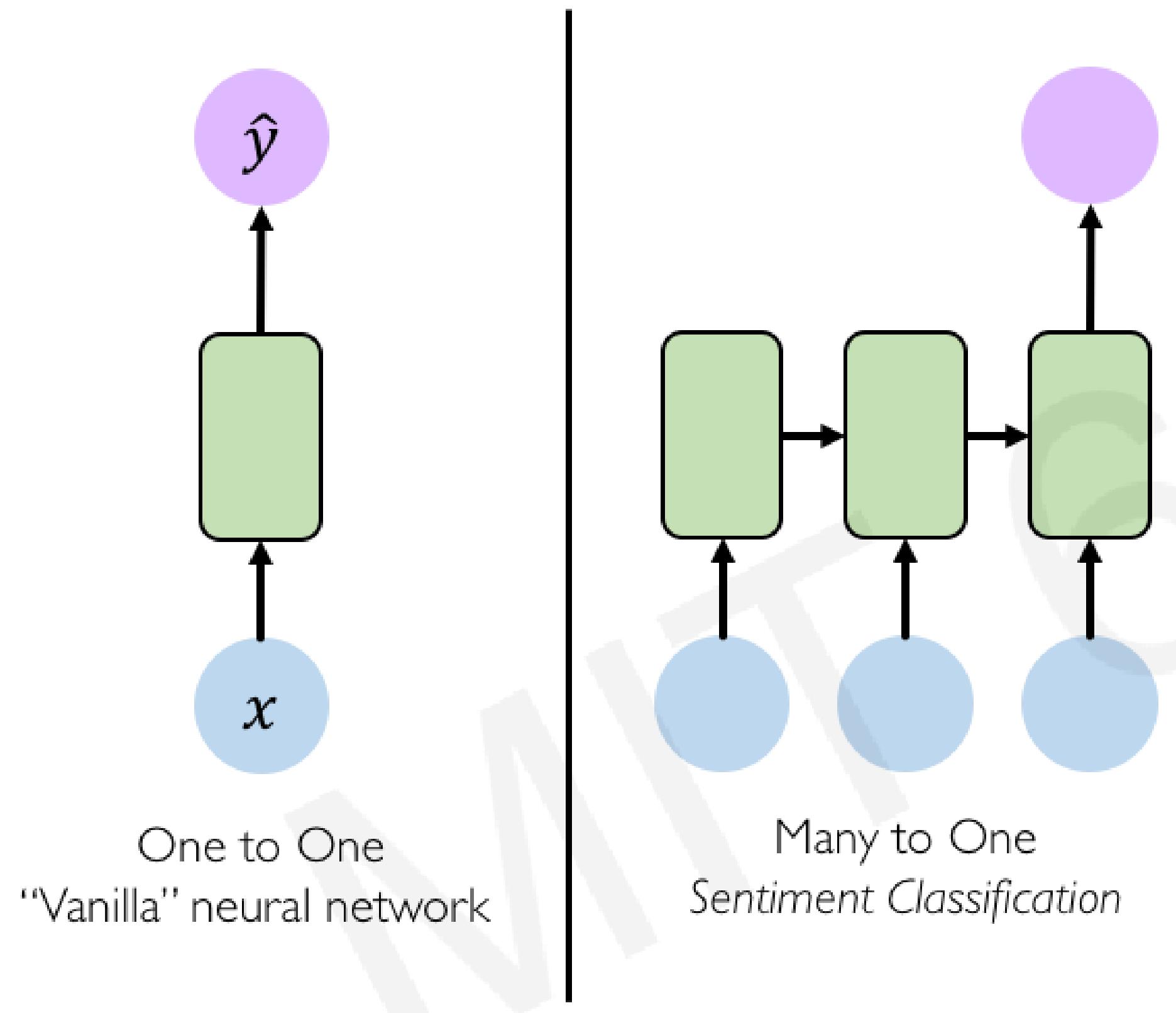
Recurrent Neural Networks (RNNs)

Standard Feed-Forward Neural Network

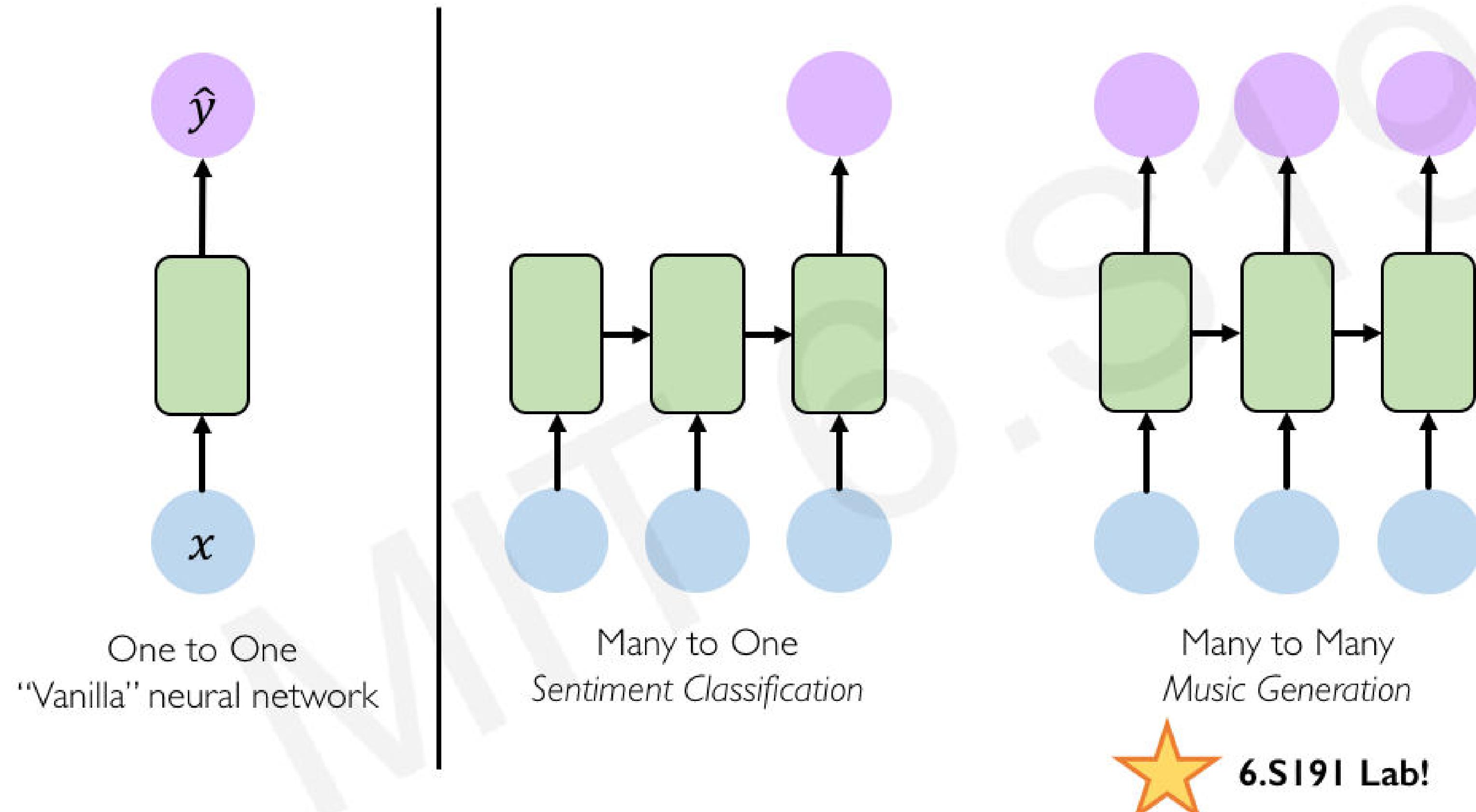


One to One
"Vanilla" neural network

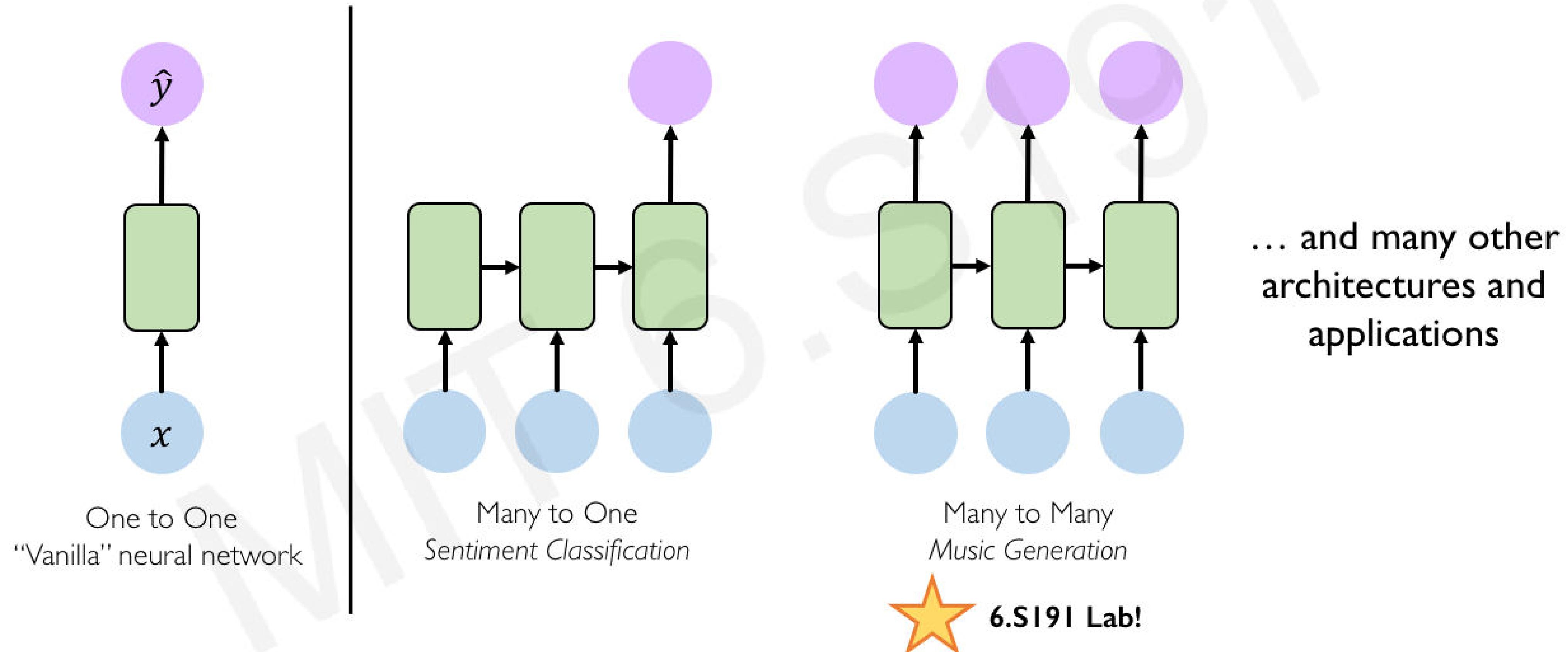
Recurrent Neural Networks for Sequence Modeling



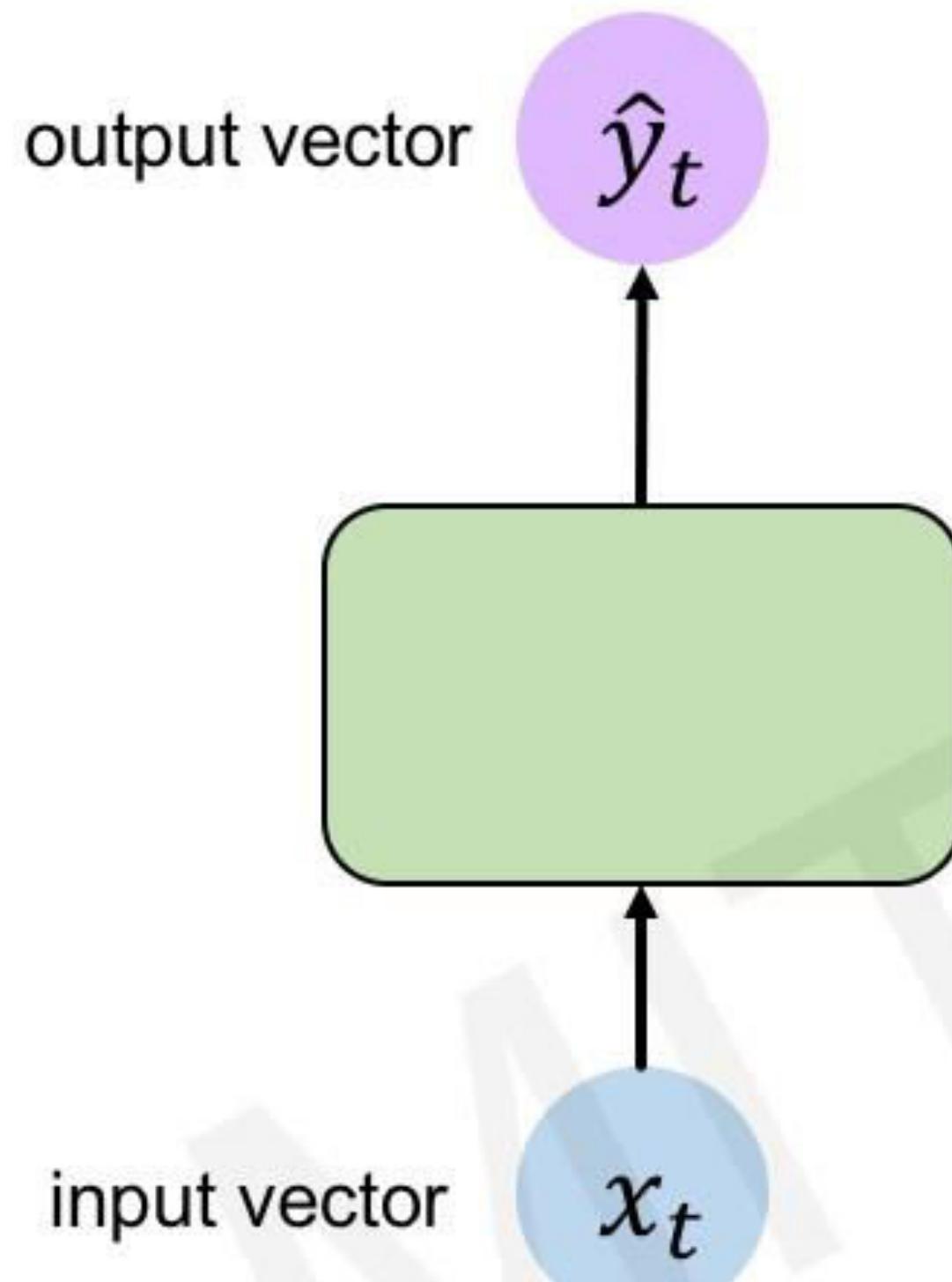
Recurrent Neural Networks for Sequence Modeling



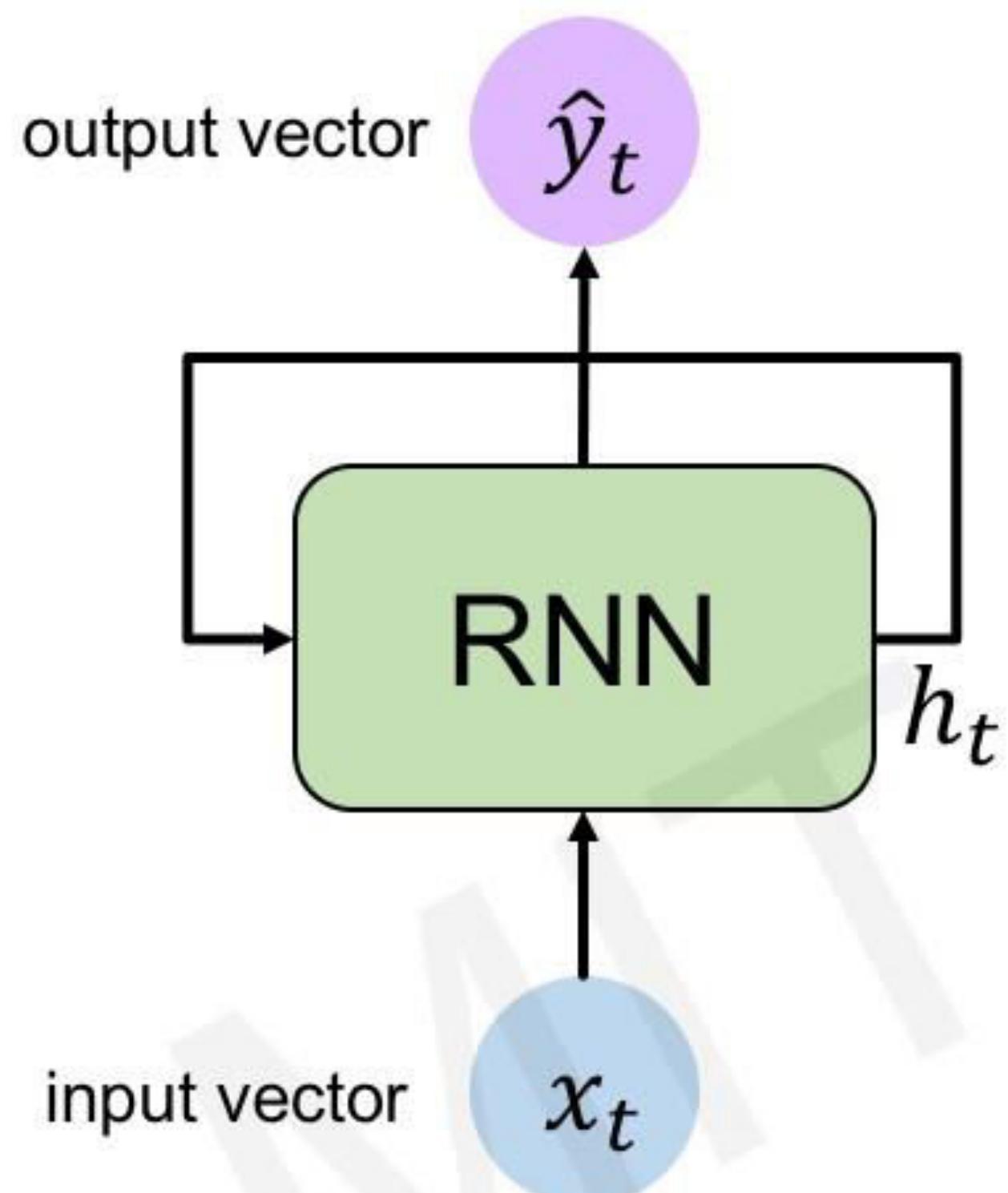
Recurrent Neural Networks for Sequence Modeling



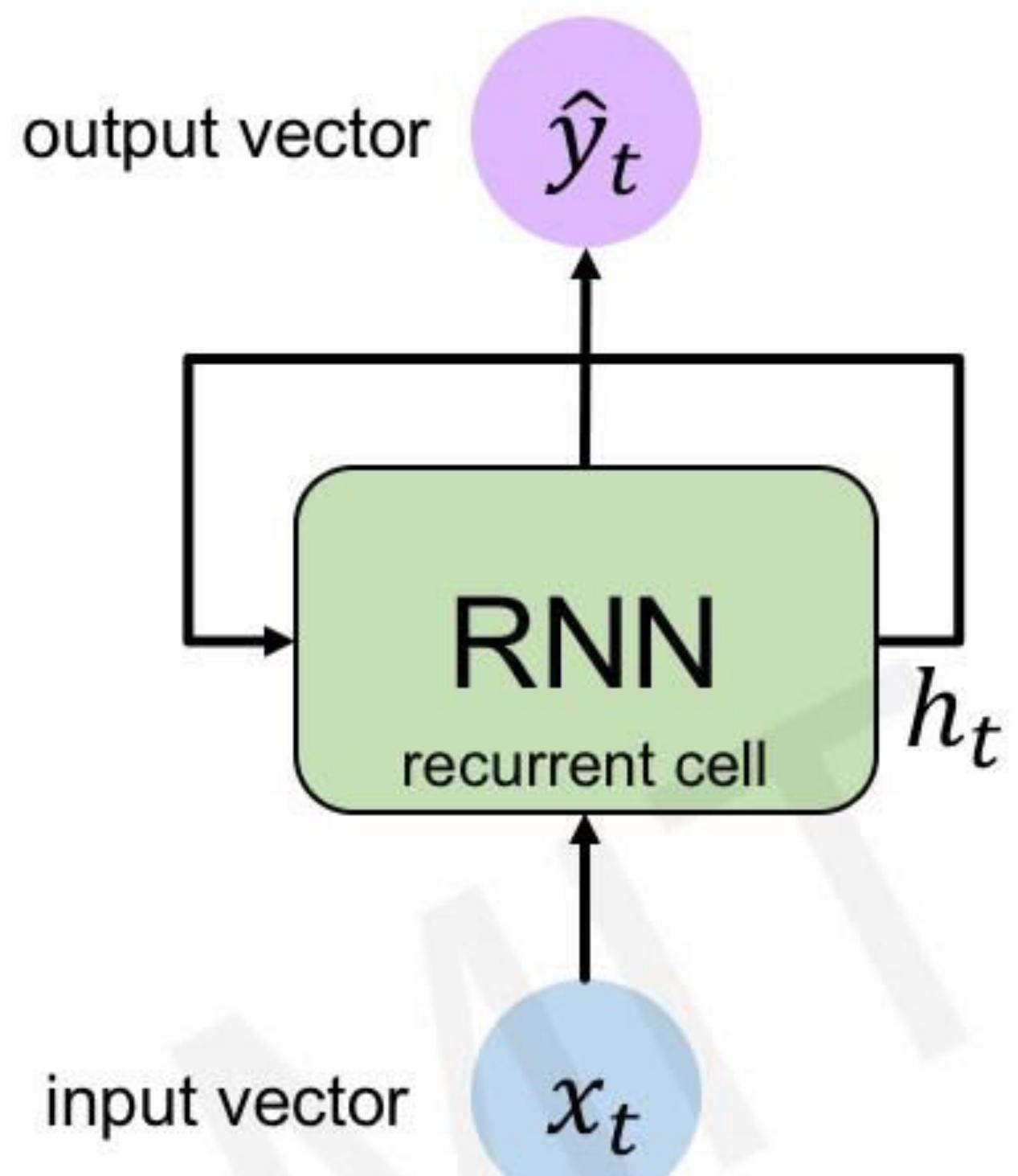
Standard “Vanilla” Neural Network



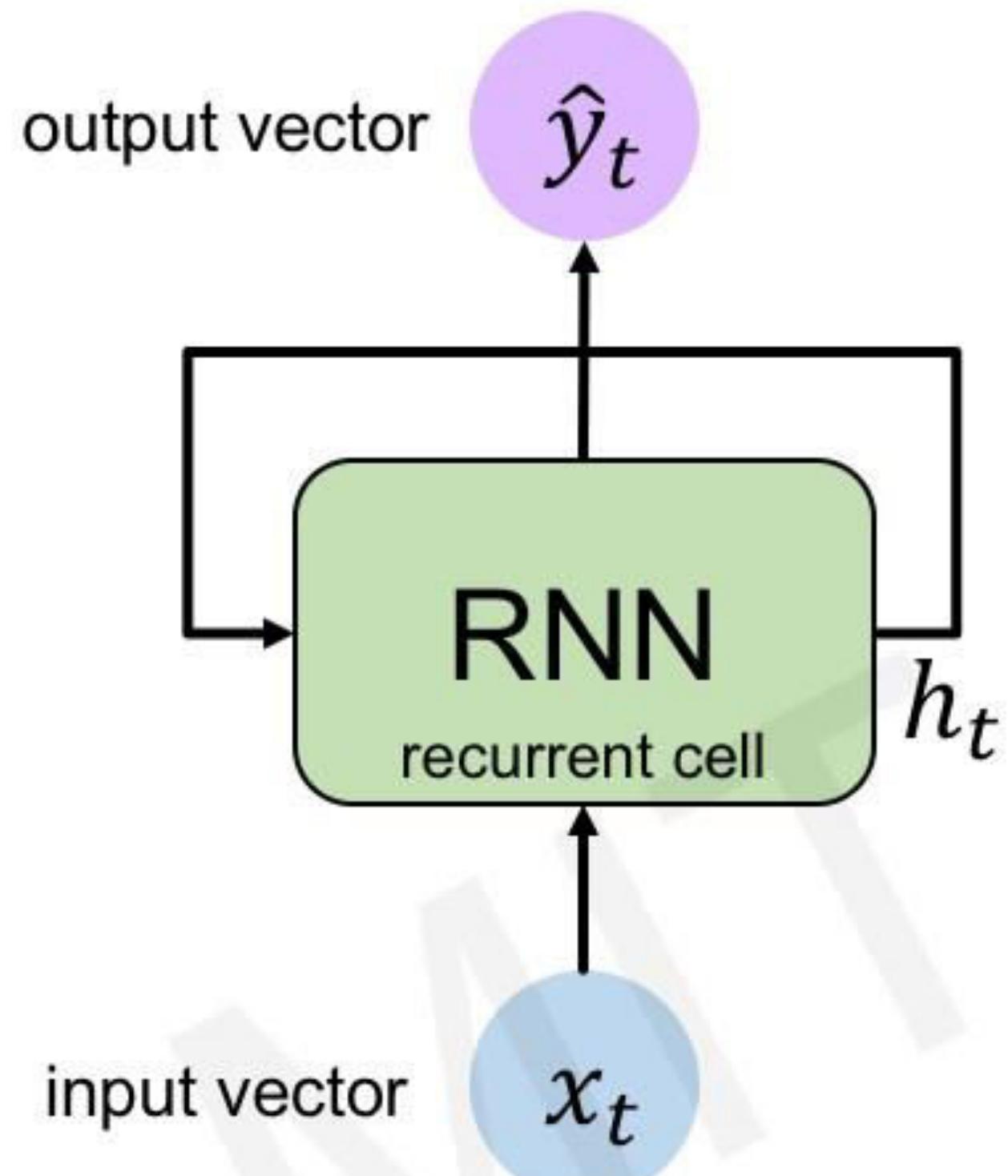
Recurrent Neural Network (RNN)



Recurrent Neural Network (RNN)

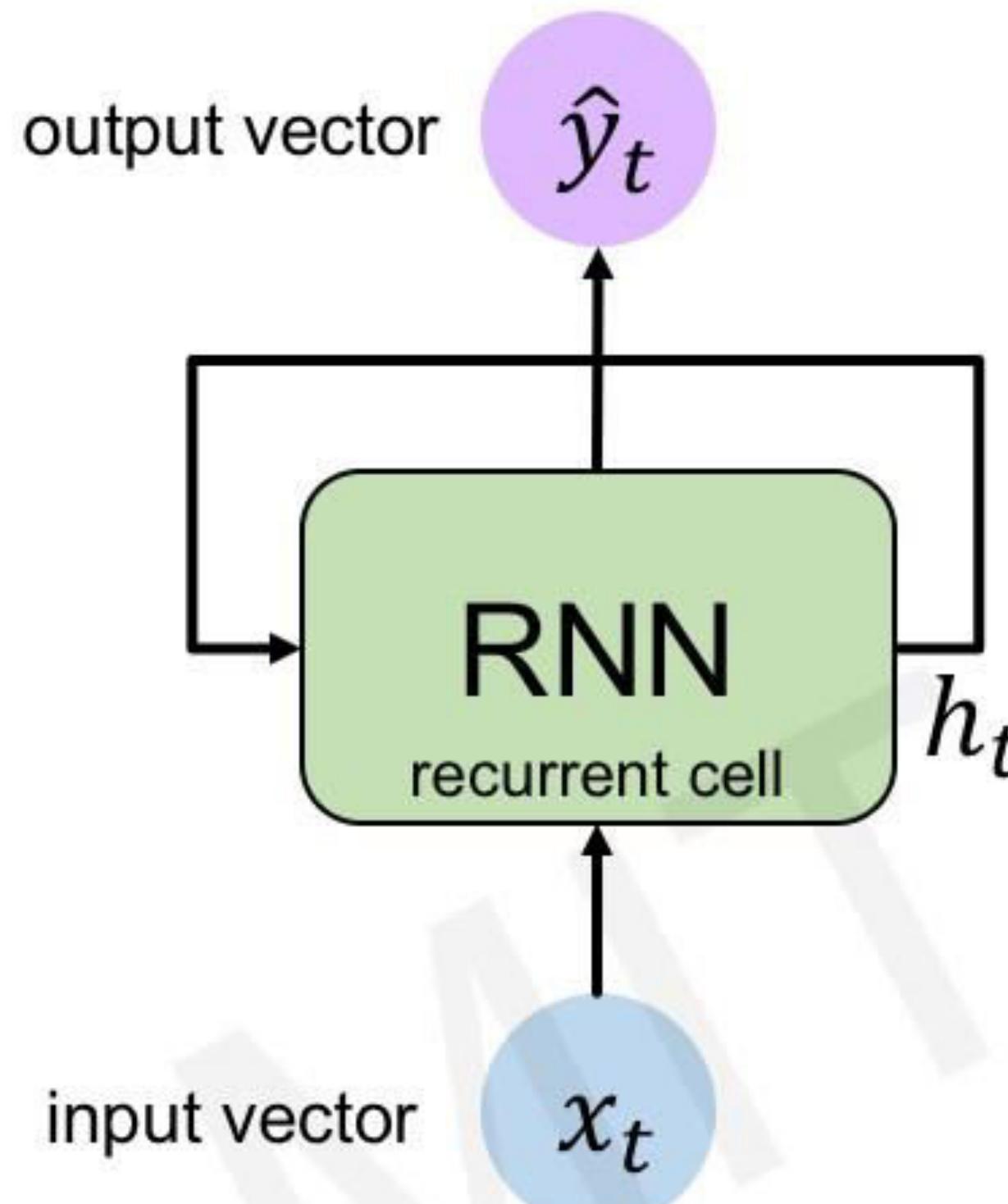


Recurrent Neural Network (RNN)



Apply a **recurrence relation** at every time step to process a sequence:

Recurrent Neural Network (RNN)

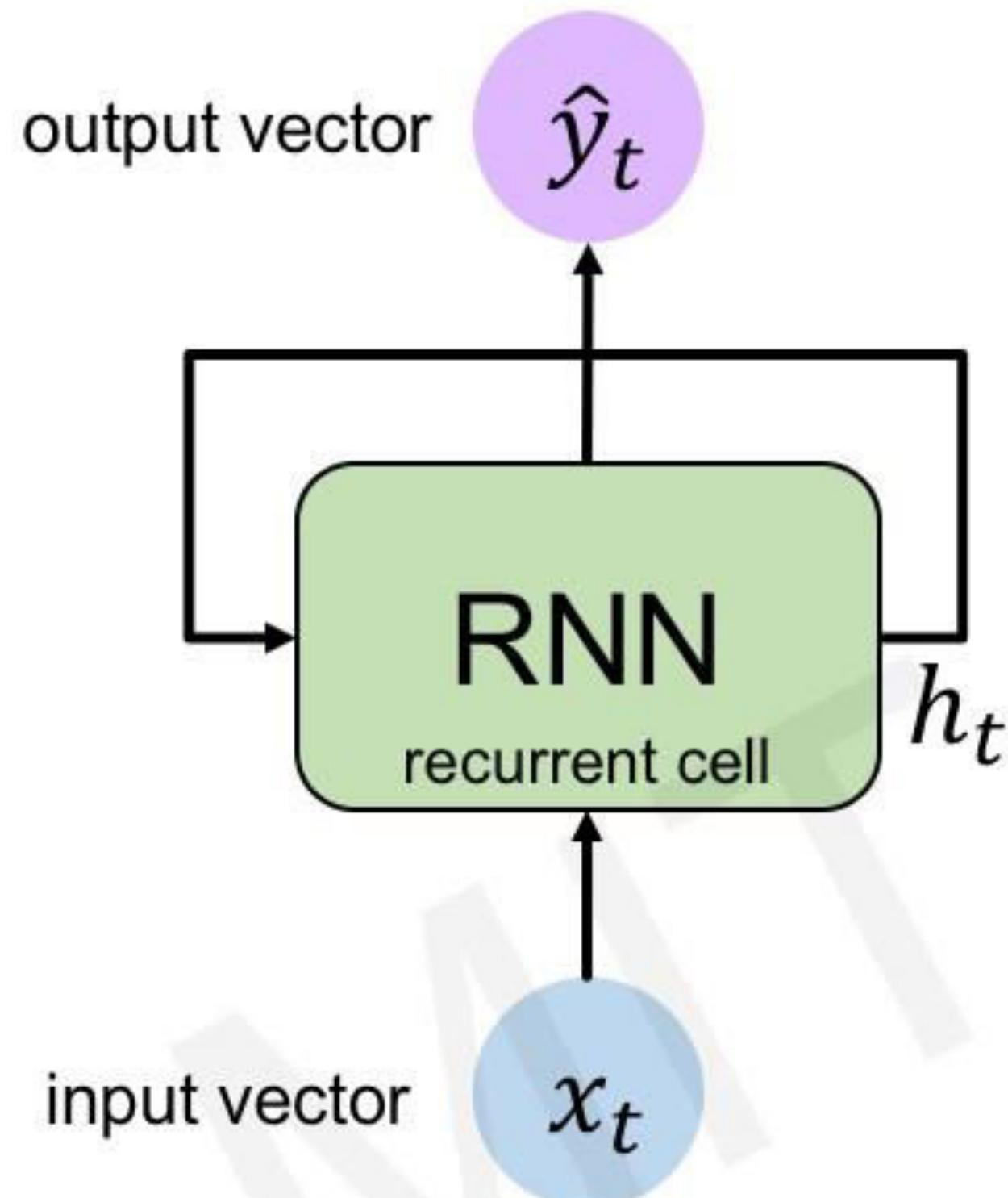


Apply a **recurrence relation** at every time step to process a sequence:

$$h_t = f_W(h_{t-1}, x_t)$$

cell state function old state input vector at
 parameterized by W time step t

Recurrent Neural Network (RNN)



Apply a **recurrence relation** at every time step to process a sequence:

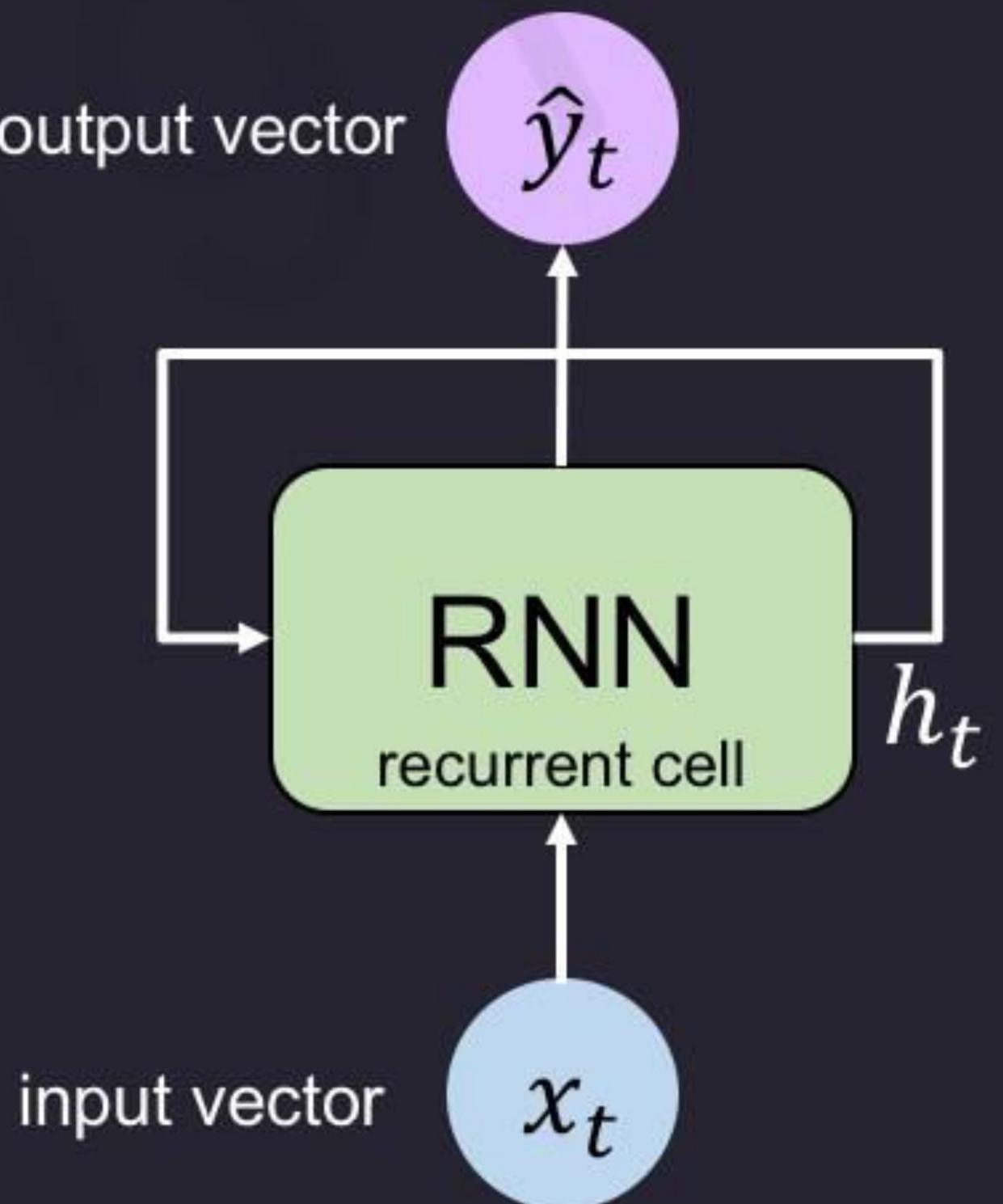
$$h_t = f_W(h_{t-1}, x_t)$$

cell state function old state input vector at
 parameterized by W time step t

Note: the same function and set of parameters are used at every time step

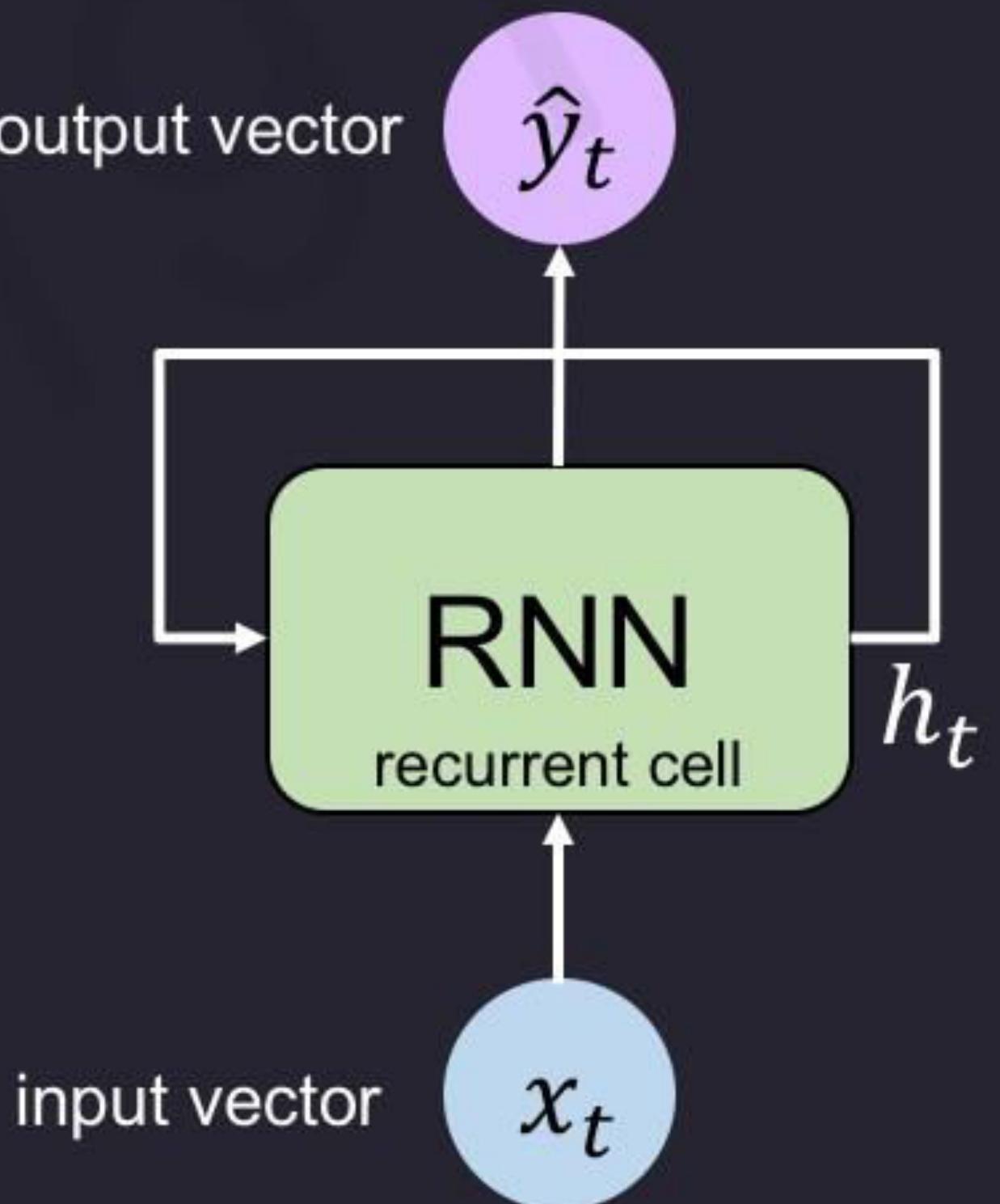
RNN Intuition

```
my_rnn = RNN()  
hidden_state = [0, 0, 0, 0]  
  
sentence = ["I", "love", "recurrent", "neural"]  
  
for word in sentence:  
    prediction, hidden_state = my_rnn(word, hidden_state)  
  
next_word_prediction = prediction  
# >>> "networks!"
```



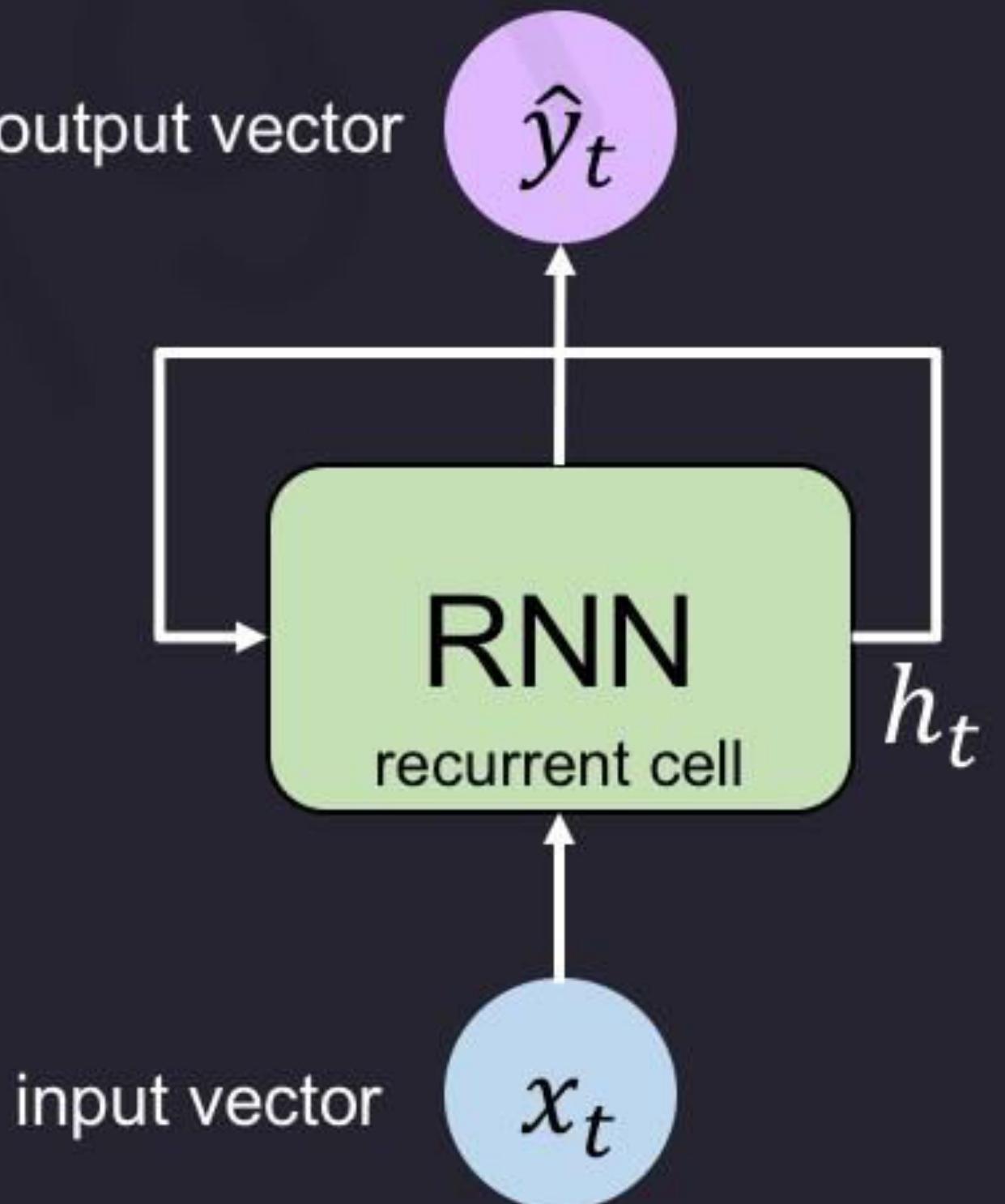
RNN Intuition

```
my_rnn = RNN()  
hidden_state = [0, 0, 0, 0]  
  
sentence = ["I", "love", "recurrent", "neural"]  
  
for word in sentence:  
    prediction, hidden_state = my_rnn(word, hidden_state)  
  
next_word_prediction = prediction  
# >>> "networks!"
```

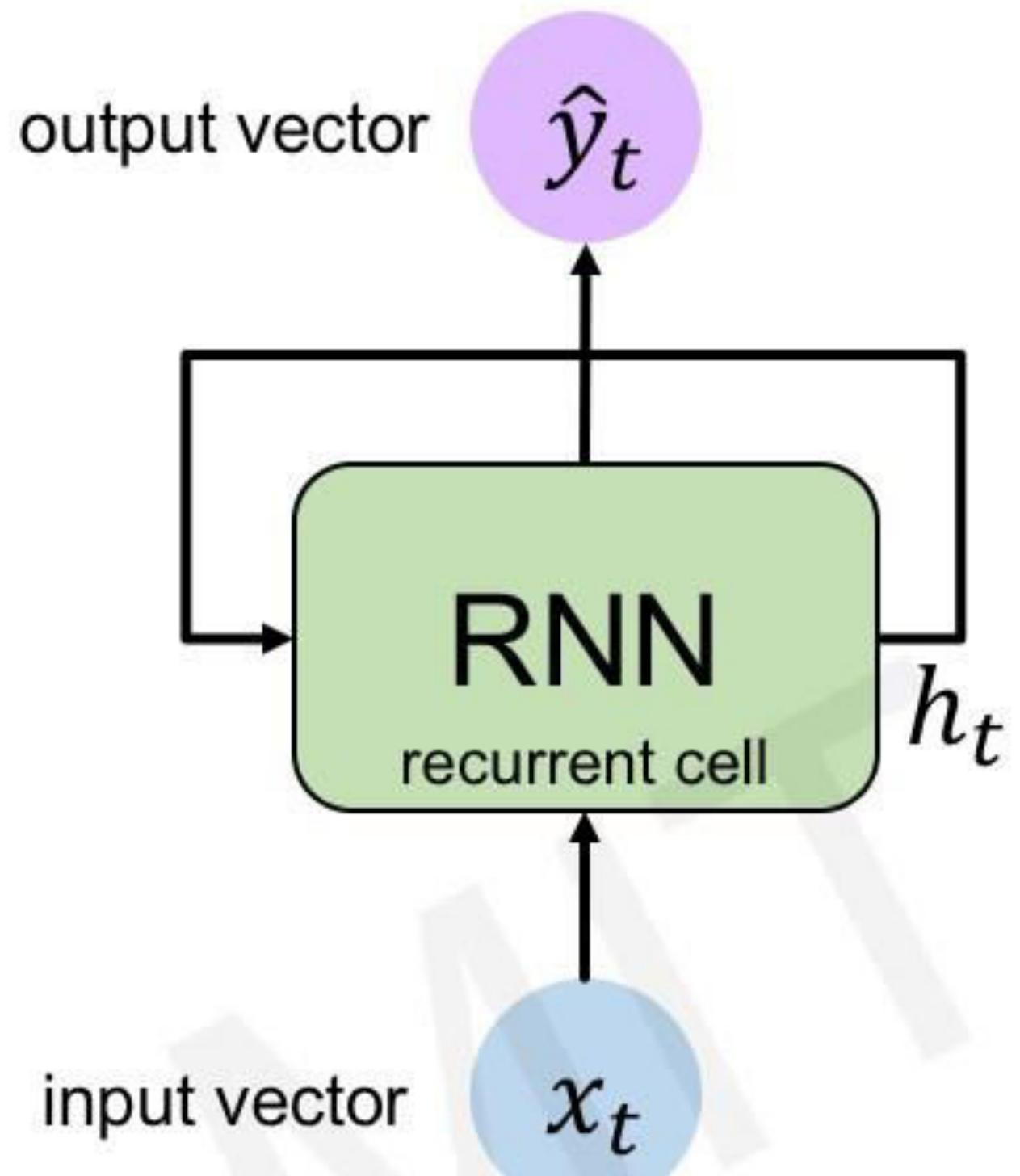


RNN Intuition

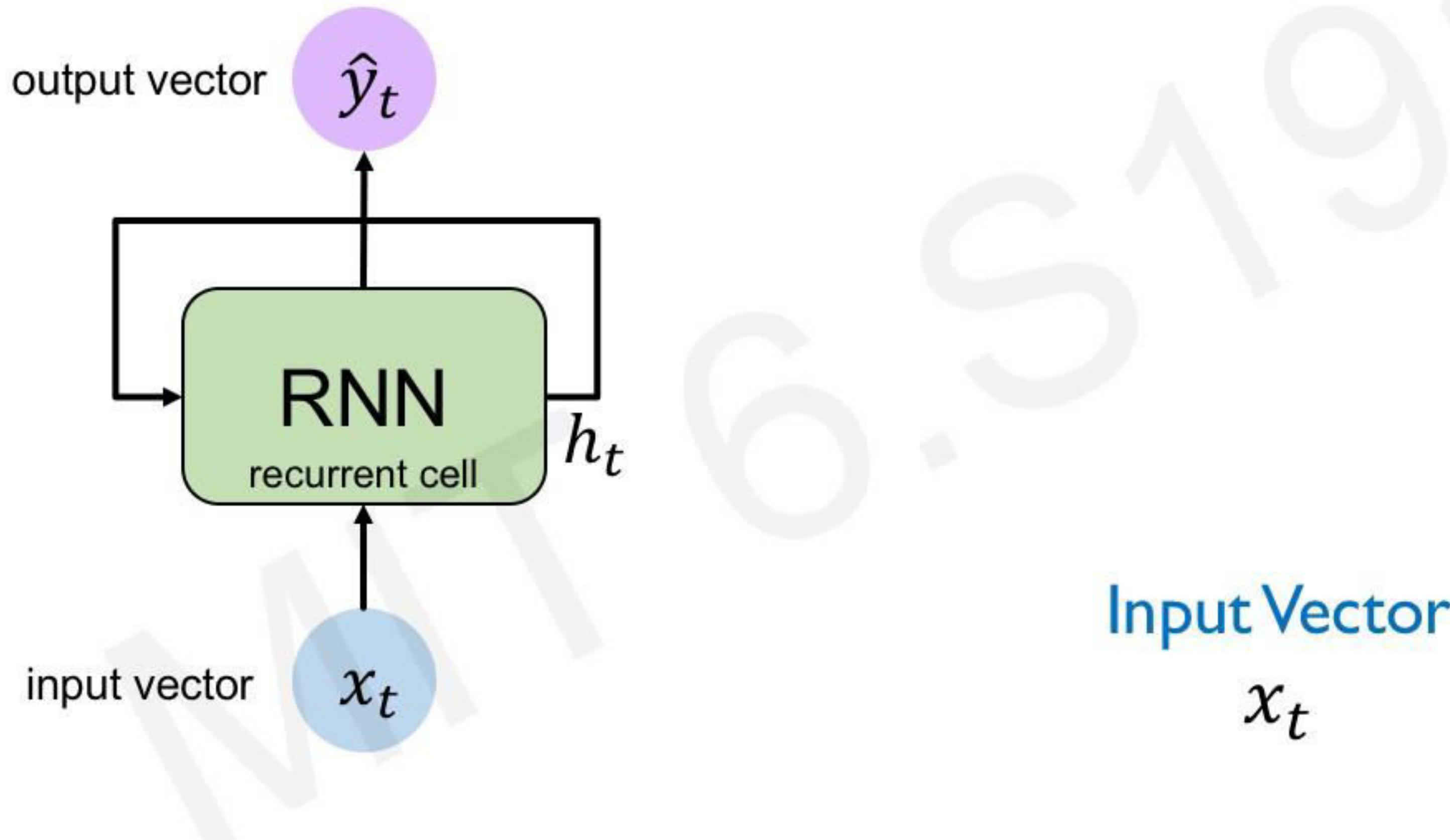
```
my_rnn = RNN()  
hidden_state = [0, 0, 0, 0]  
  
sentence = ["I", "love", "recurrent", "neural"]  
  
for word in sentence:  
    prediction, hidden_state = my_rnn(word, hidden_state)  
  
    next_word_prediction = prediction  
    # >>> "networks!"
```



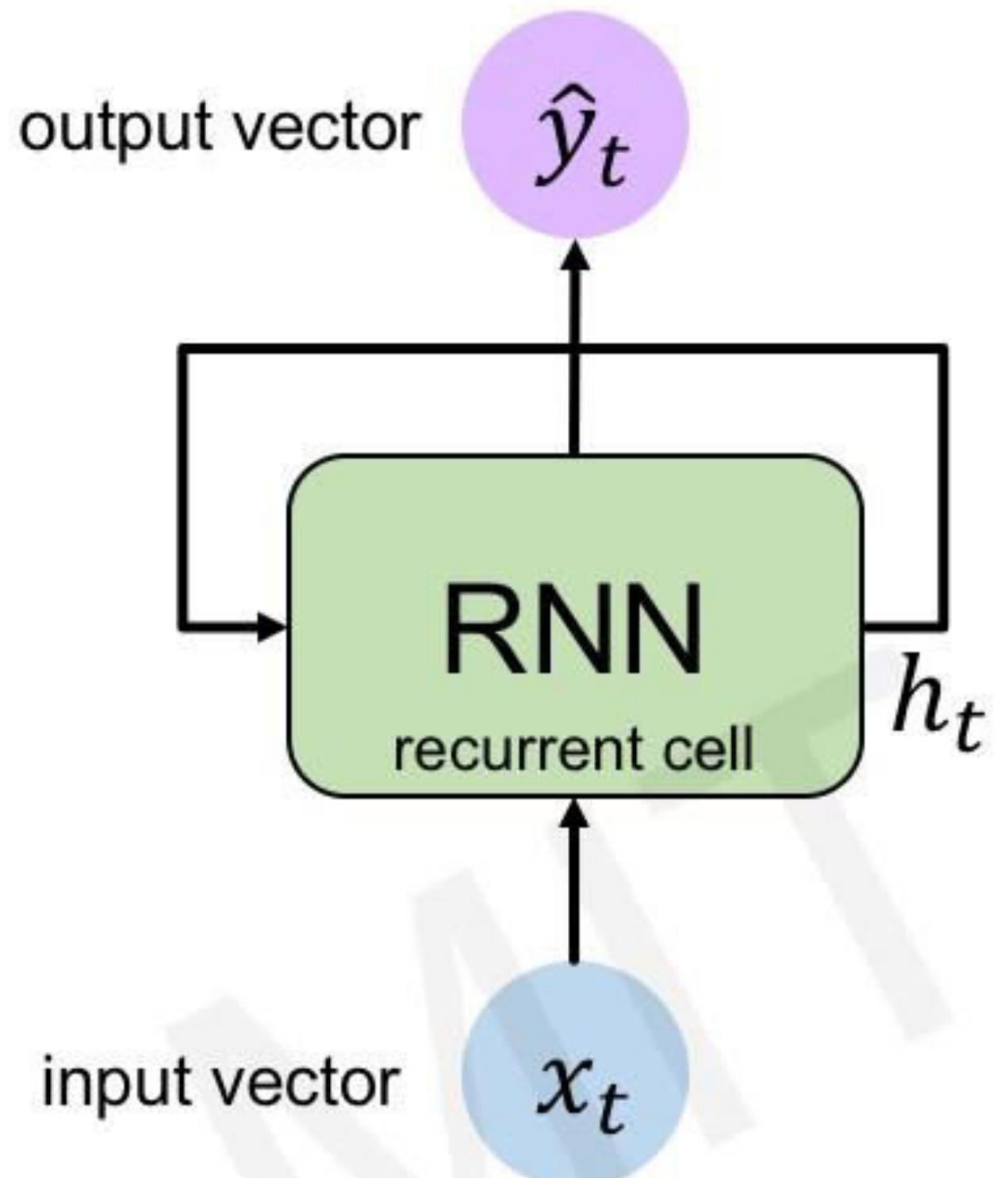
RNN State Update and Output



RNN State Update and Output



RNN State Update and Output



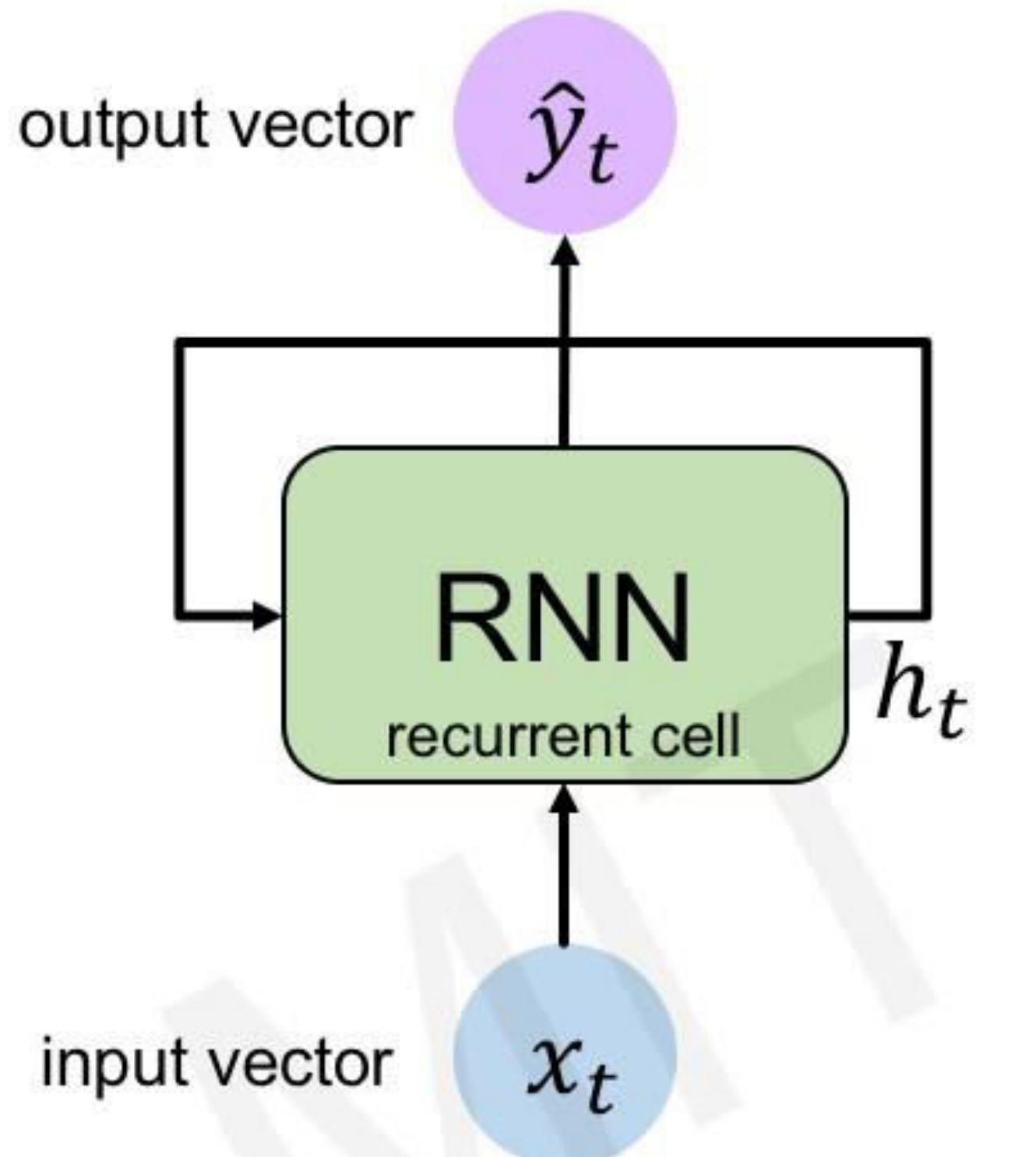
Update Hidden State

$$h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t)$$

Input Vector

x_t

RNN State Update and Output



Output Vector

$$\hat{y}_t = W_{hy}^T h_t$$

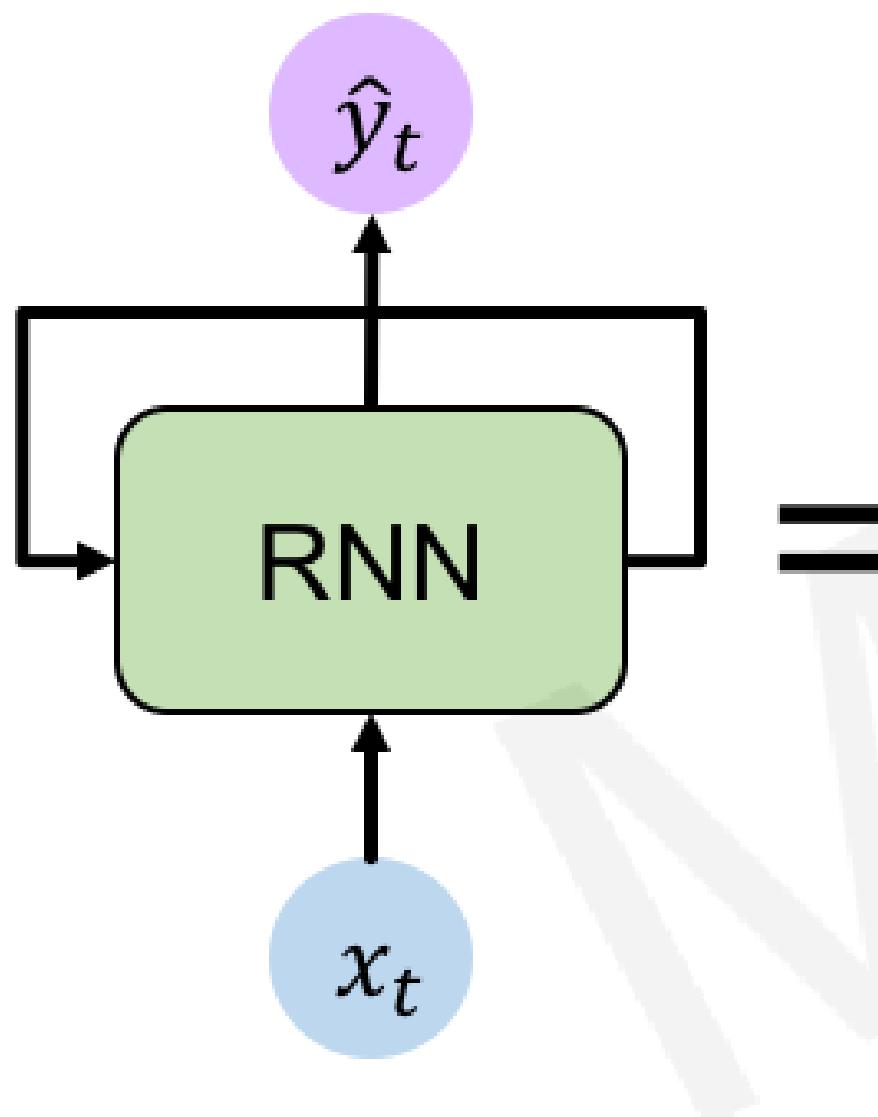
Update Hidden State

$$h_t = \tanh(W_{hh}^T h_{t-1} + W_{xh}^T x_t)$$

Input Vector

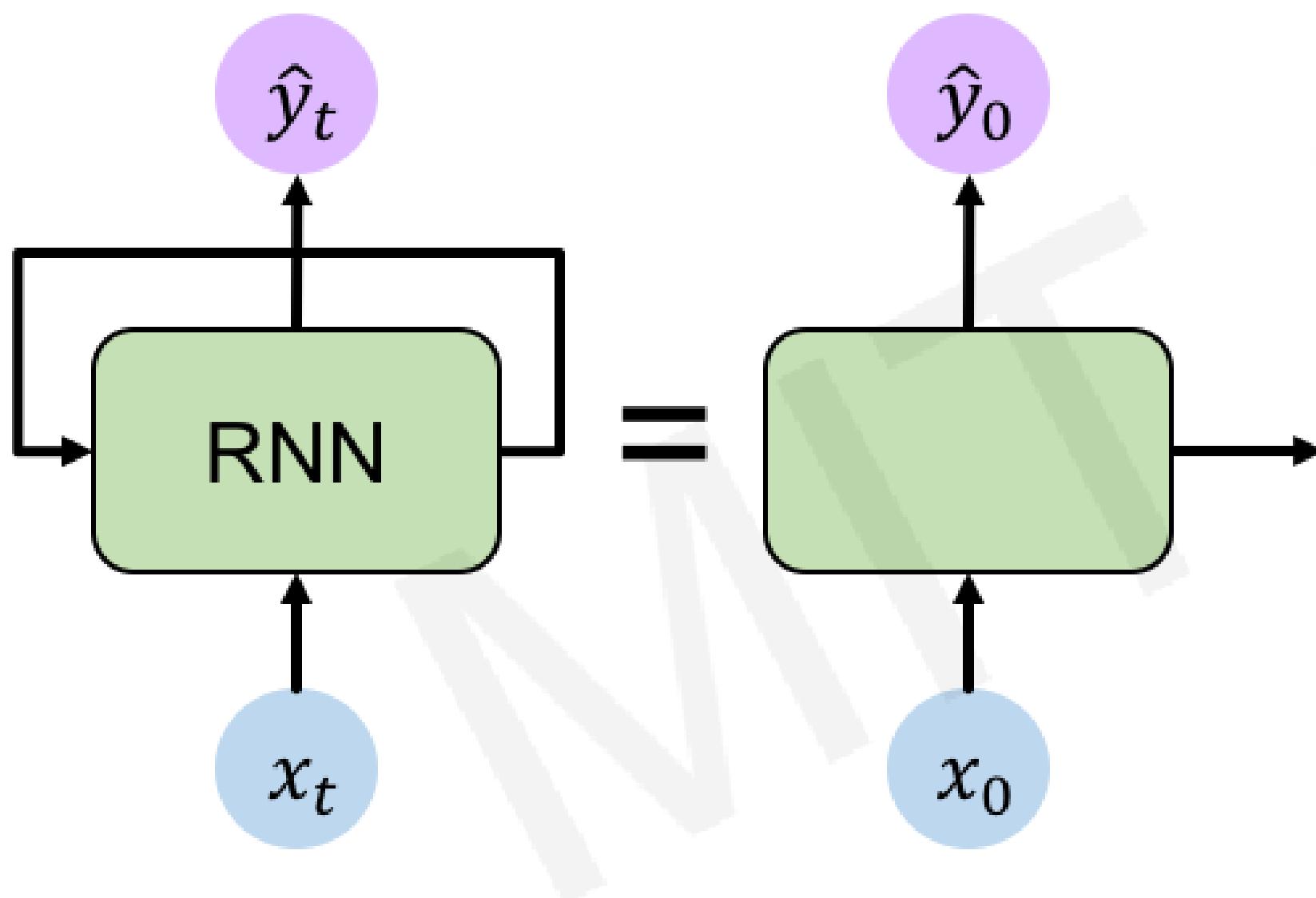
$$x_t$$

RNNs: Computational Graph Across Time

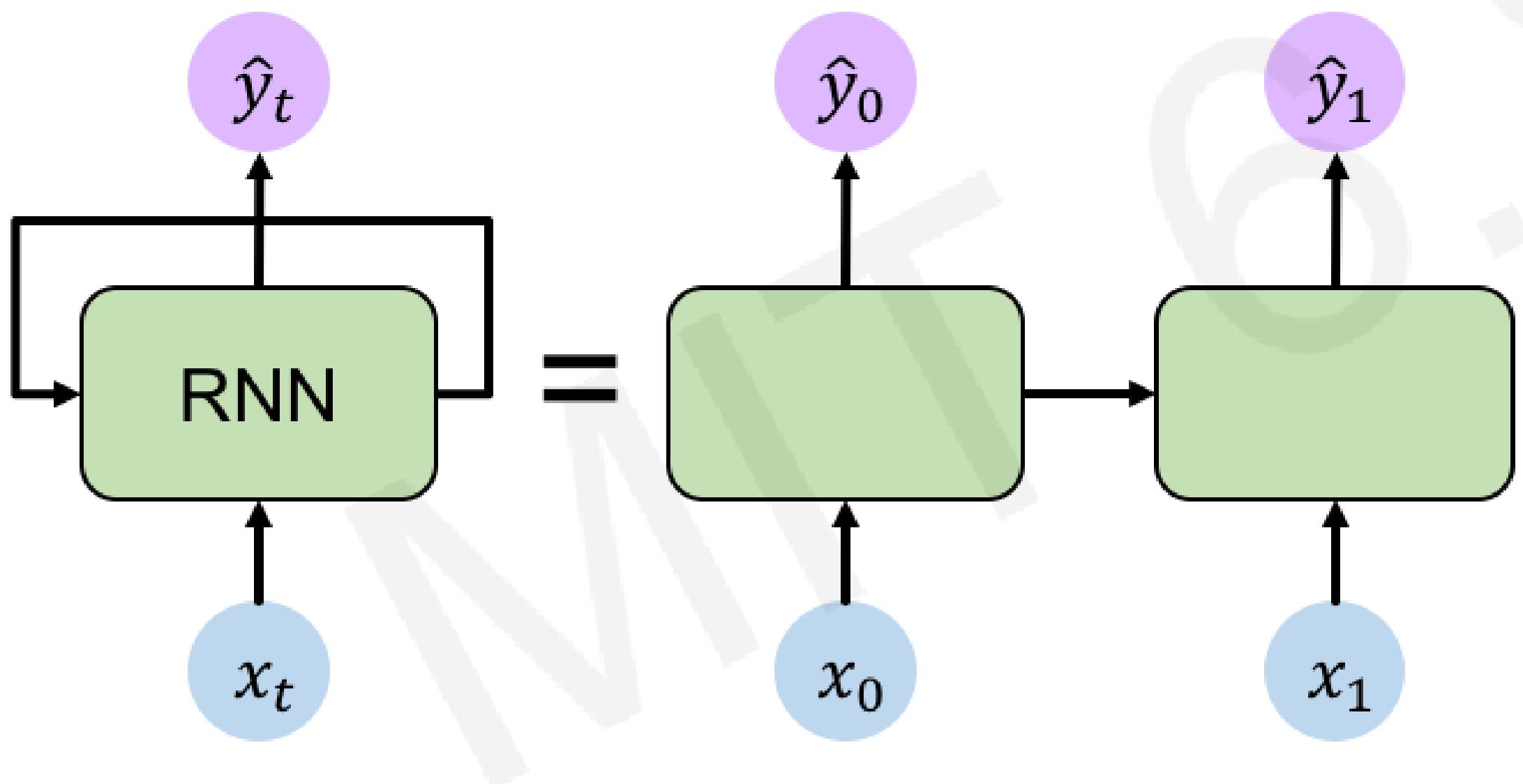


= Represent as computational graph unrolled across time

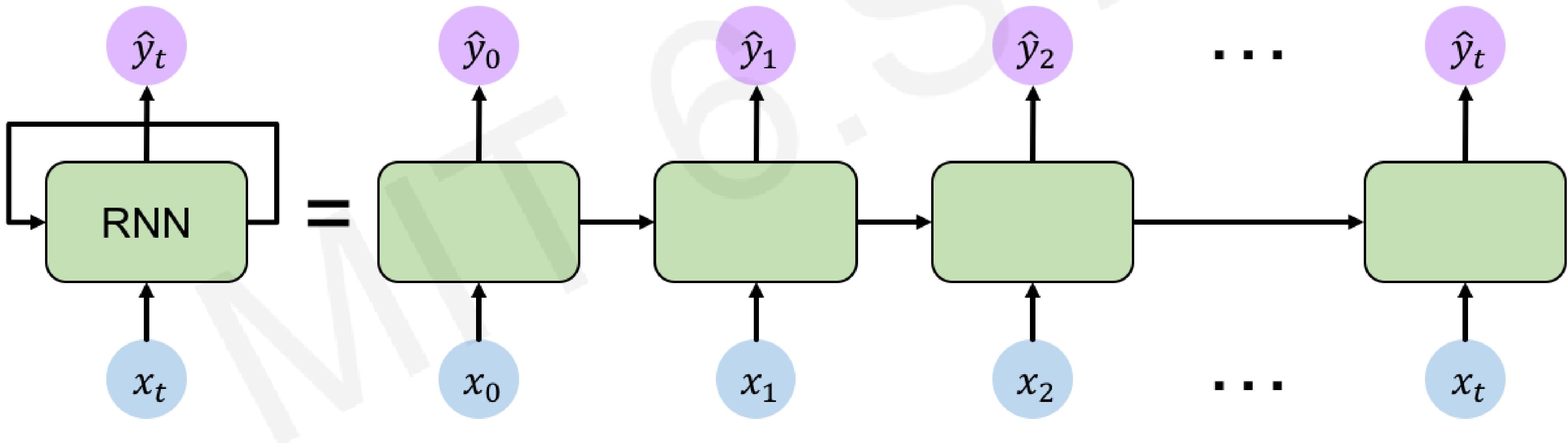
RNNs: Computational Graph Across Time



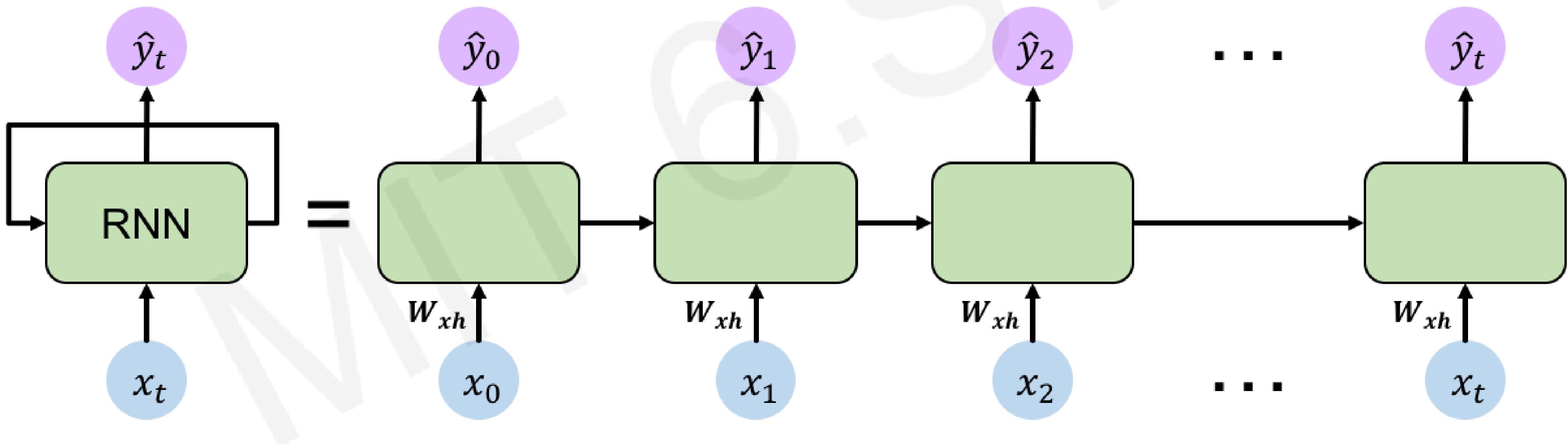
RNNs: Computational Graph Across Time



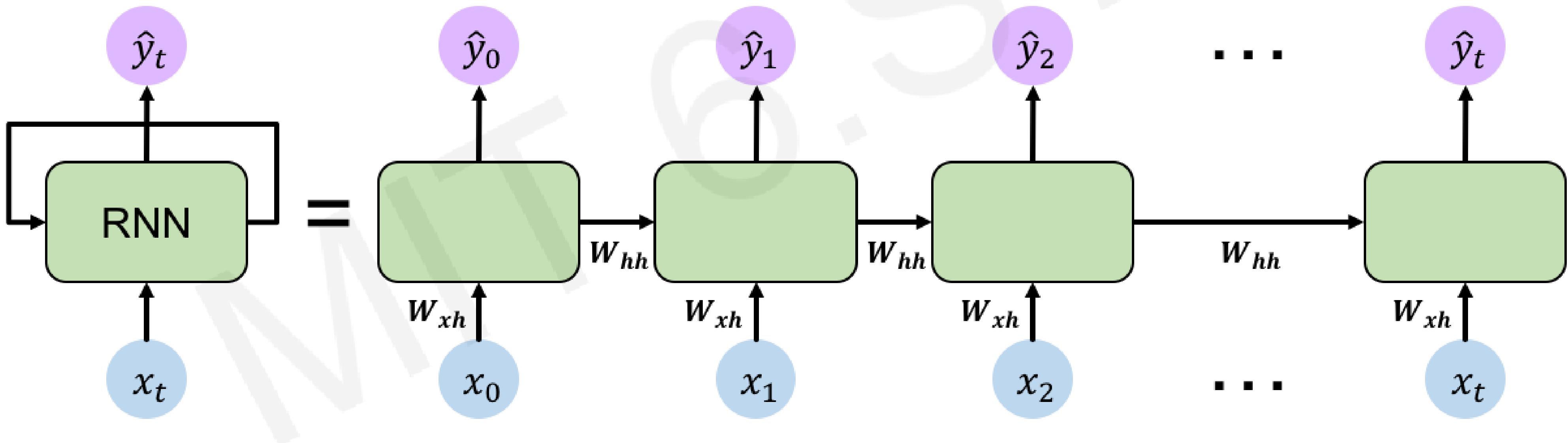
RNNs: Computational Graph Across Time



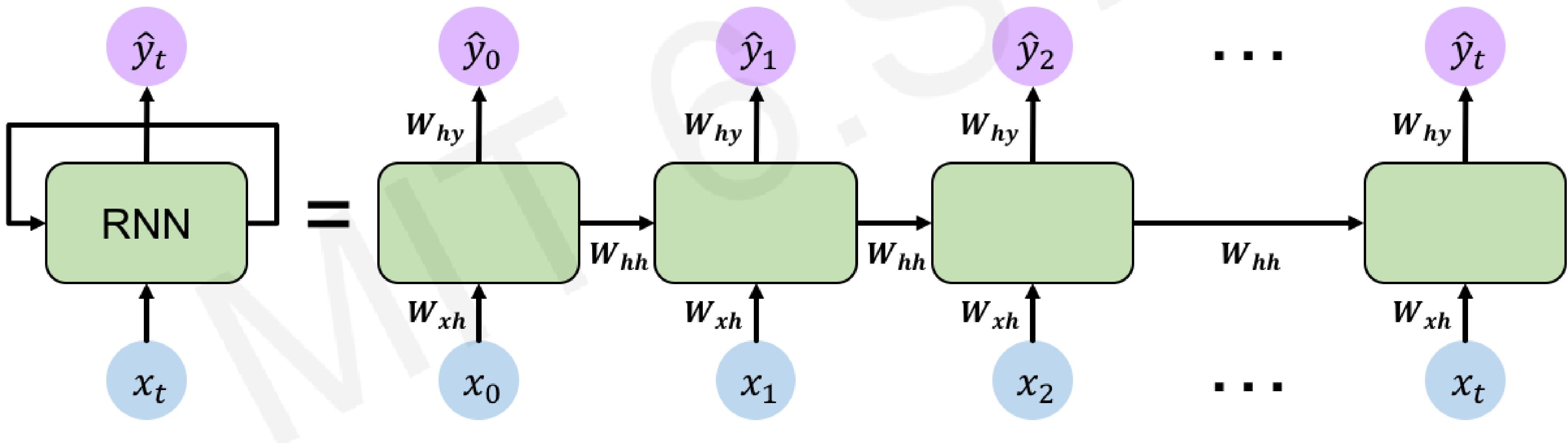
RNNs: Computational Graph Across Time



RNNs: Computational Graph Across Time

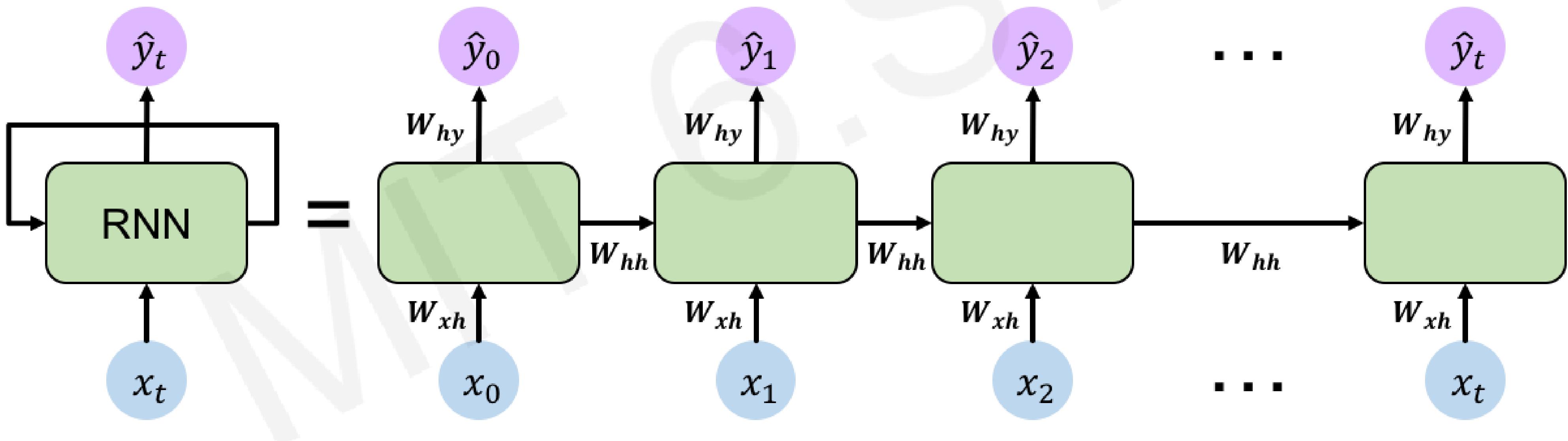


RNNs: Computational Graph Across Time



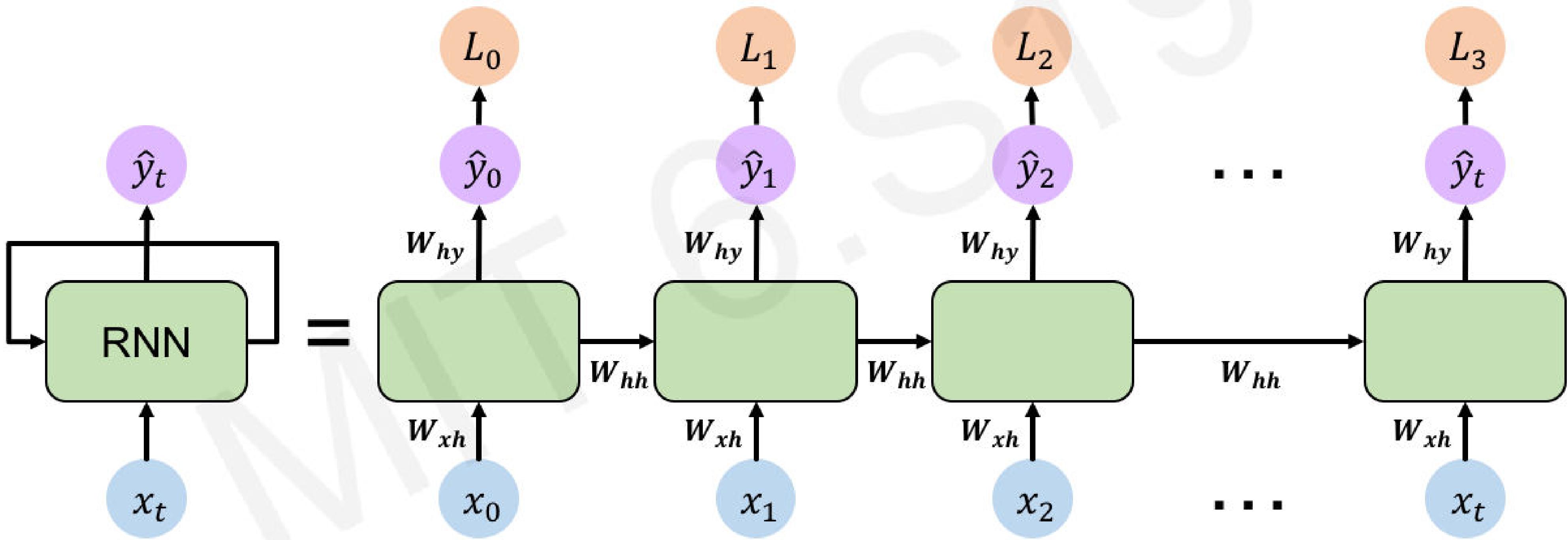
RNNs: Computational Graph Across Time

Re-use the **same weight matrices** at every time step

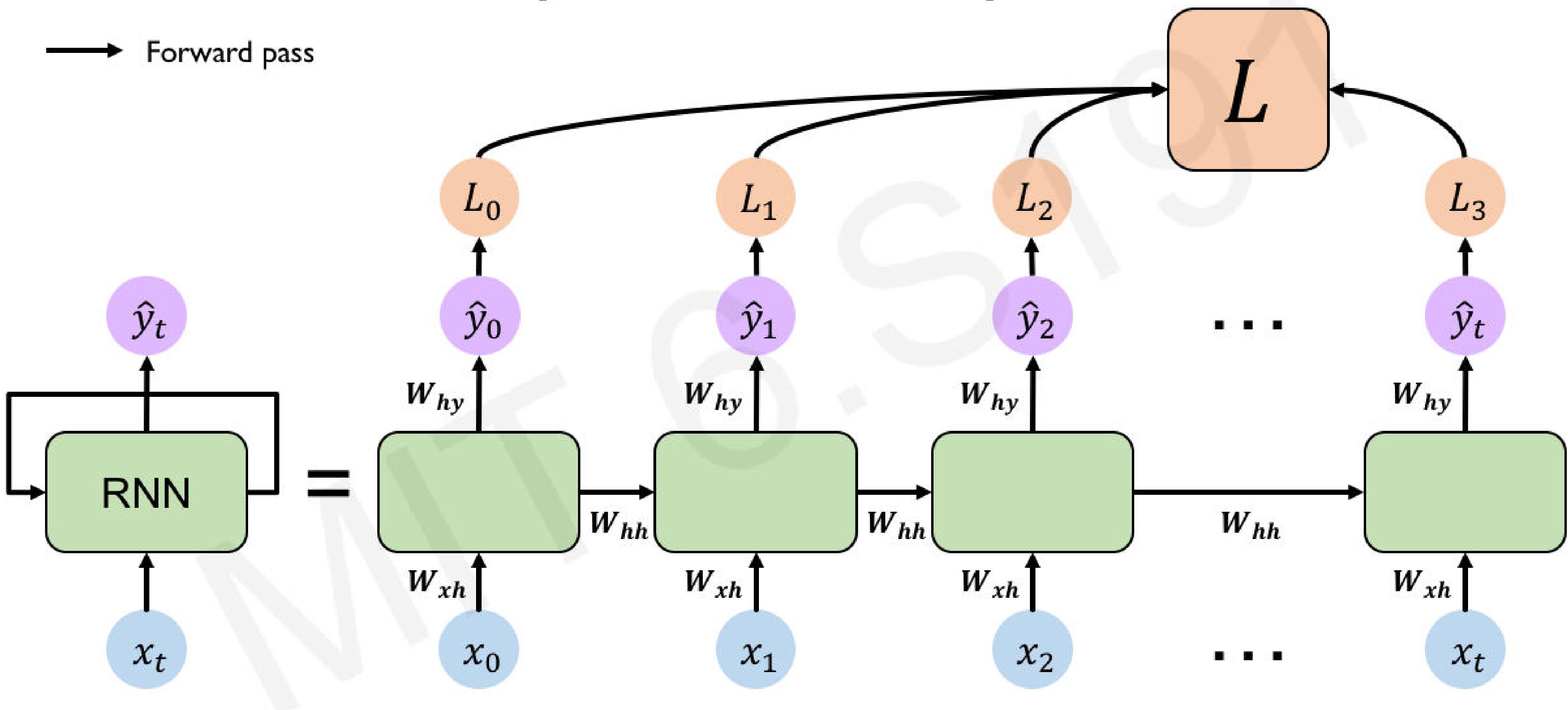


RNNs: Computational Graph Across Time

→ Forward pass



RNNs: Computational Graph Across Time



RNNs from Scratch

```

class MyRNNCell(tf.keras.layers.Layer):
    def __init__(self, rnn_units, input_dim, output_dim):
        super(MyRNNCell, self).__init__()

        # Initialize weight matrices
        self.W_xh = self.add_weight([rnn_units, input_dim])
        self.W_hh = self.add_weight([rnn_units, rnn_units])
        self.W_hy = self.add_weight([output_dim, rnn_units])

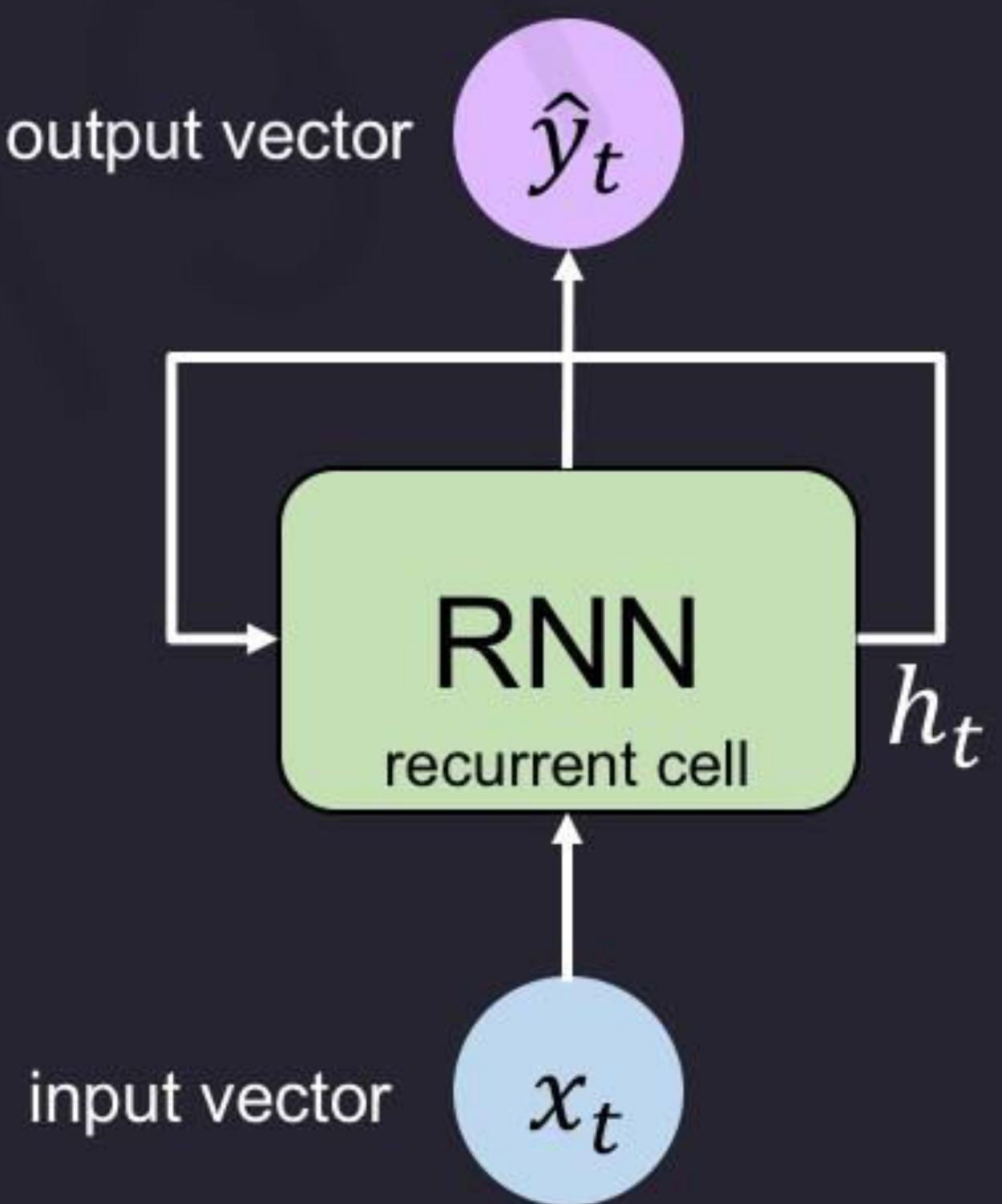
        # Initialize hidden state to zeros
        self.h = tf.zeros([rnn_units, 1])

    def call(self, x):
        # Update the hidden state
        self.h = tf.math.tanh( self.W_hh * self.h + self.W_xh * x )

        # Compute the output
        output = self.W_hy * self.h

        # Return the current output and hidden state
        return output, self.h

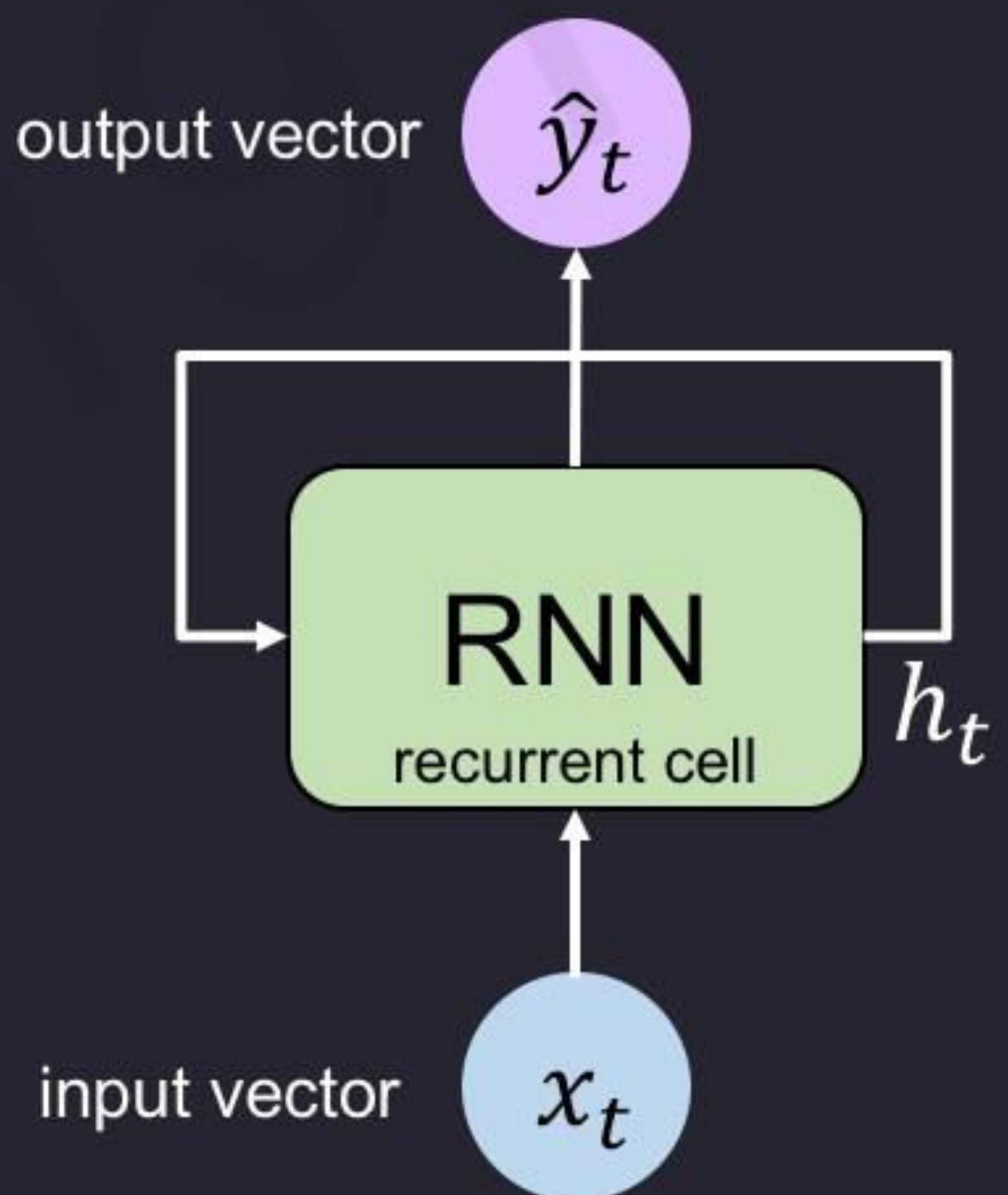
```



RNN Implementation in TensorFlow

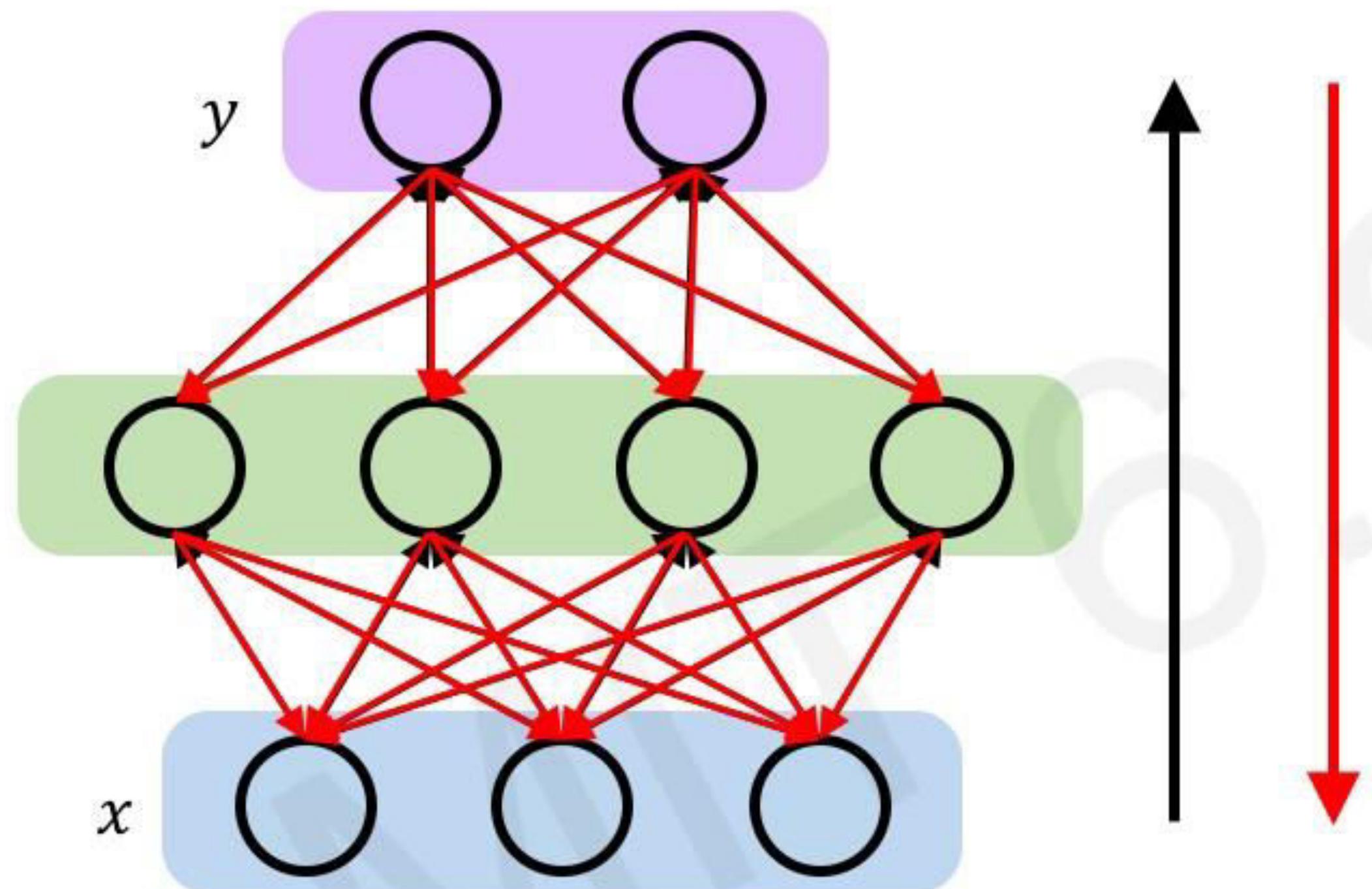


```
tf.keras.layers.SimpleRNN(rnn_units)
```



Backpropagation Through Time (BPTT)

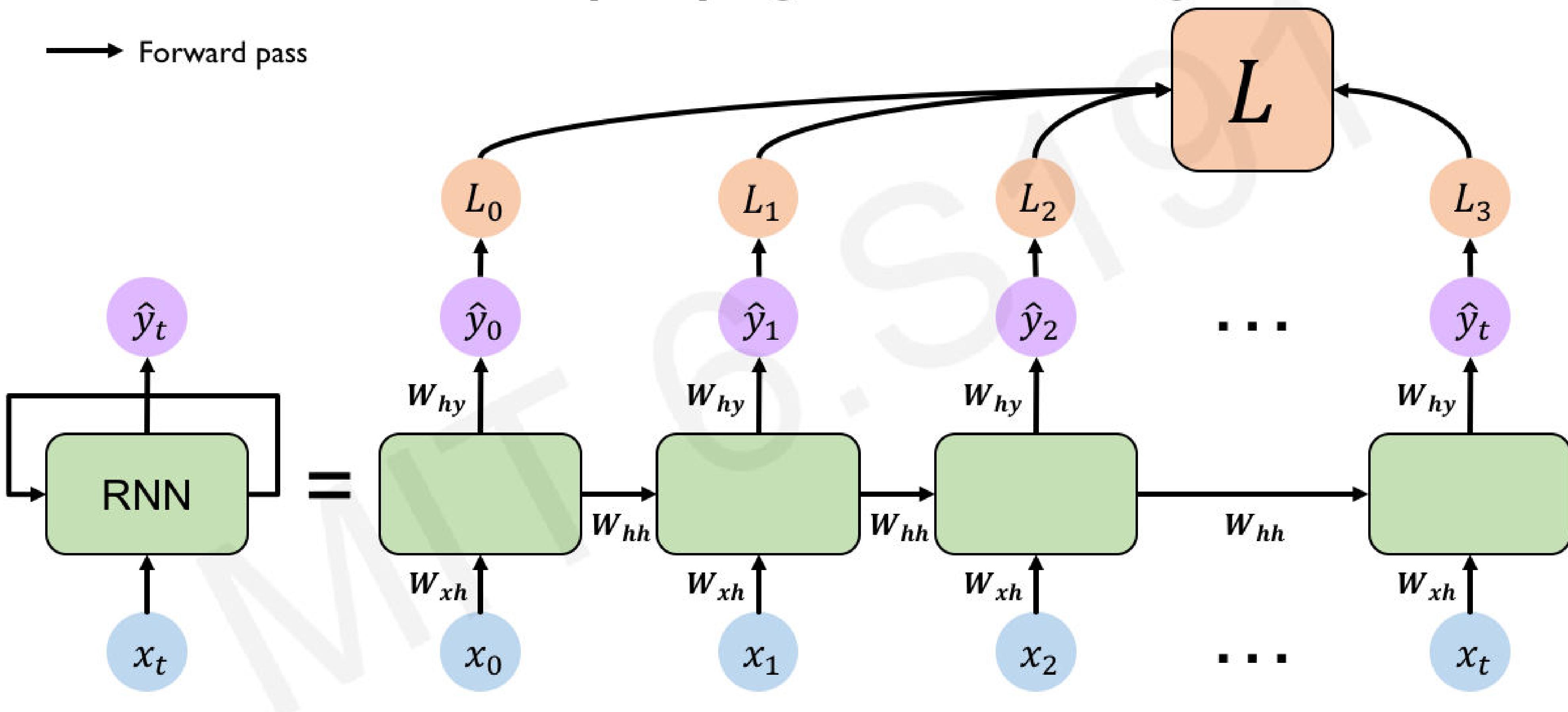
Recall: Backpropagation in Feed Forward Models



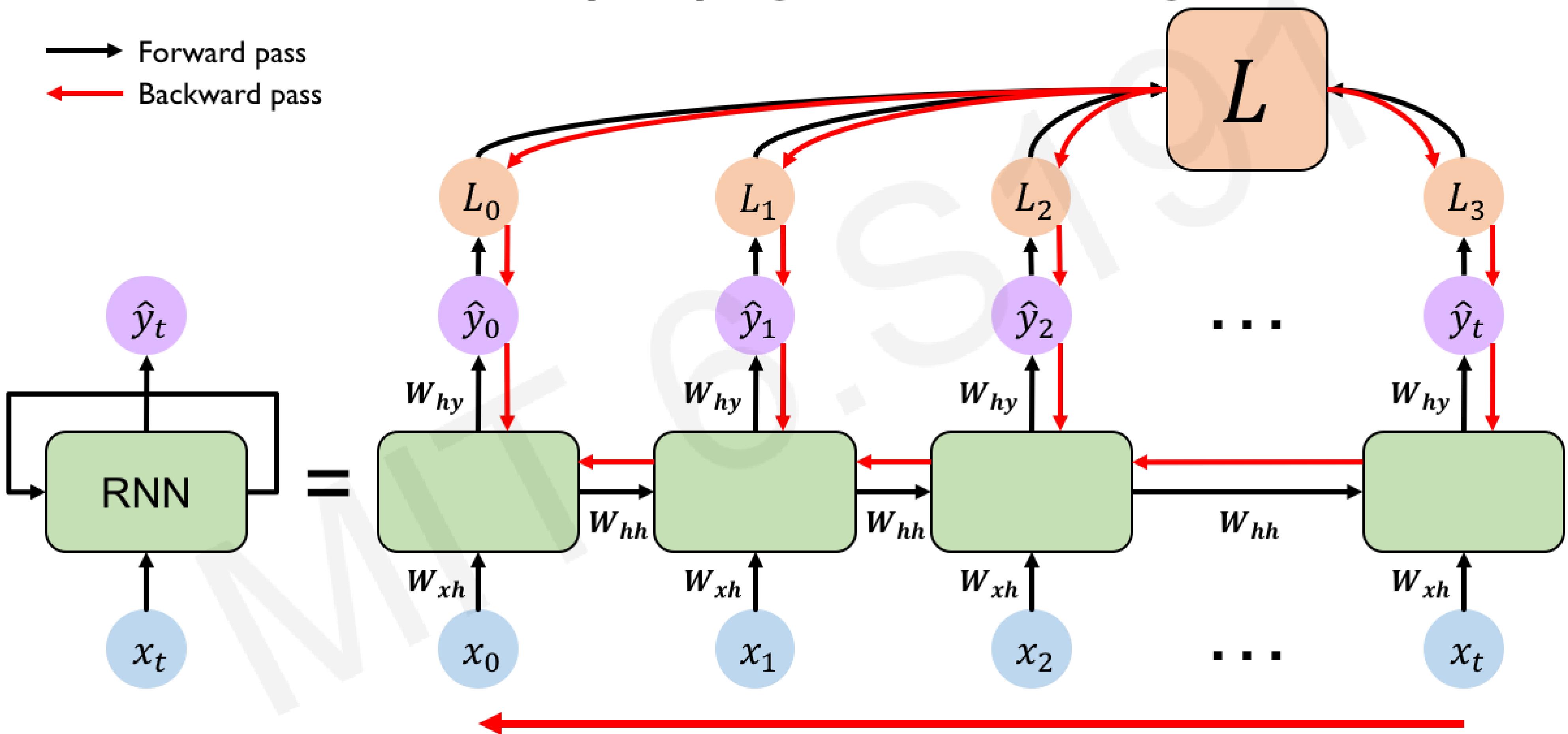
Backpropagation algorithm:

1. Take the derivative (gradient) of the loss with respect to each parameter
2. Shift parameters in order to minimize loss

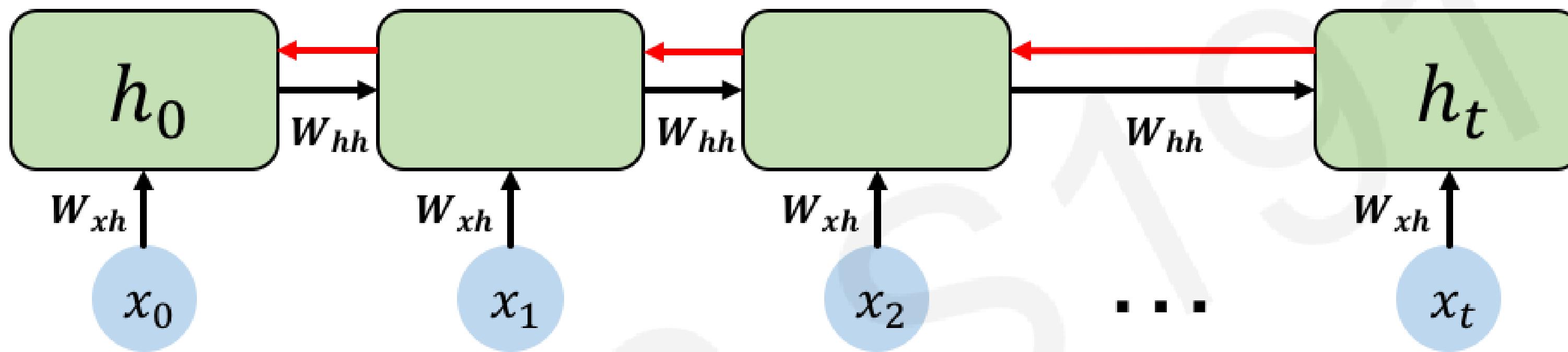
RNNs: Backpropagation Through Time



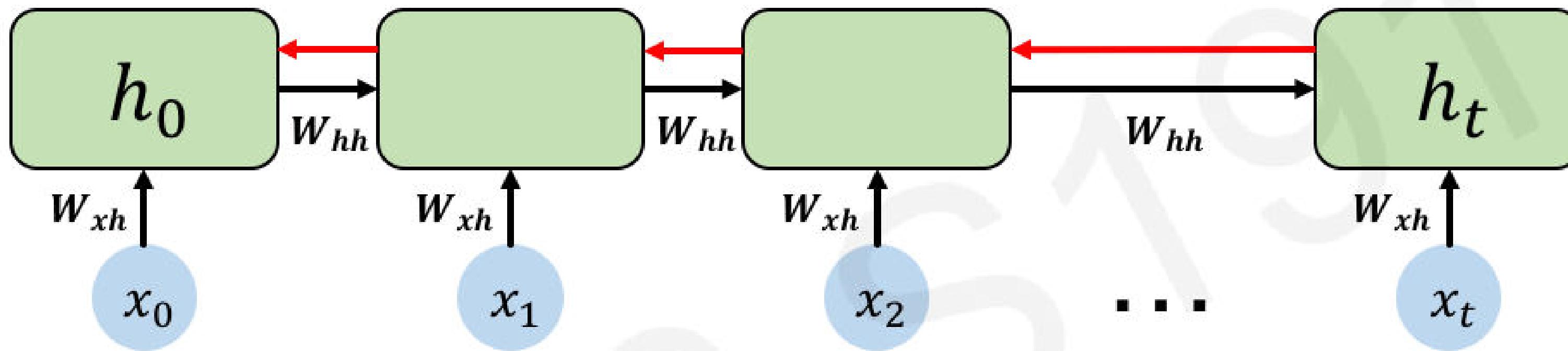
RNNs: Backpropagation Through Time



Standard RNN Gradient Flow

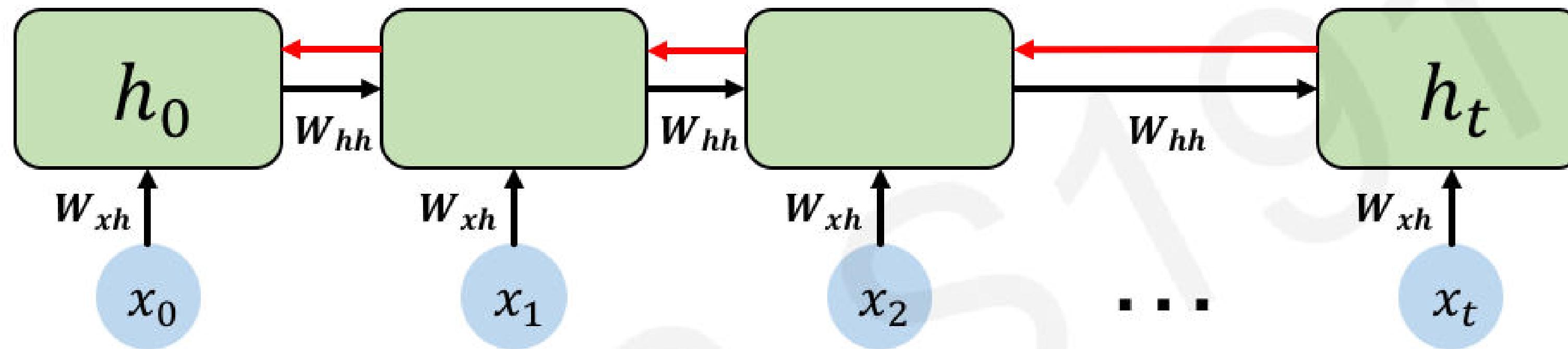


Standard RNN Gradient Flow



Computing the gradient wrt h_0 involves many factors of W_{hh} + repeated gradient computation!

Standard RNN Gradient Flow: Exploding Gradients

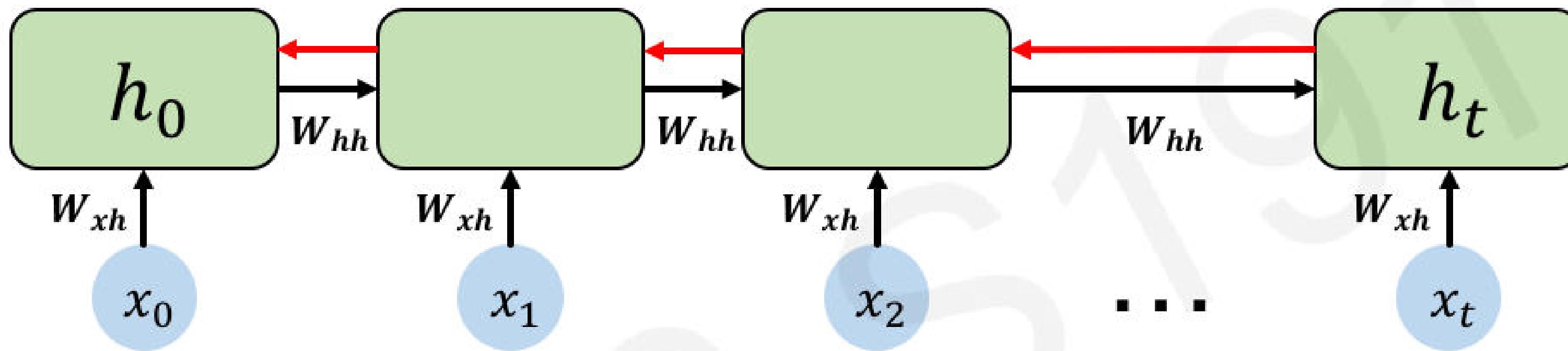


Computing the gradient wrt h_0 involves many factors of W_{hh} + repeated gradient computation!

Many values > 1 :
exploding gradients

Gradient clipping to
scale big gradients

Standard RNN Gradient Flow: Vanishing Gradients



Computing the gradient wrt h_0 involves many factors of W_{hh} + repeated gradient computation!

Many values > 1 :
exploding gradients

Gradient clipping to
scale big gradients

Many values < 1 :
vanishing gradients

1. Activation function
2. Weight initialization
3. Network architecture

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?



Massachusetts
Institute of
Technology

The Problem of Long-Term Dependencies

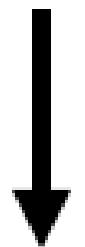
Why are vanishing gradients a problem?

Multiply many **small numbers** together

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together

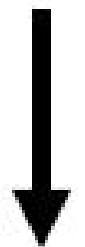


Errors due to further back time steps
have smaller and smaller gradients

The Problem of Long-Term Dependencies

Why are vanishing gradients a problem?

Multiply many **small numbers** together



Errors due to further back time steps
have smaller and smaller gradients



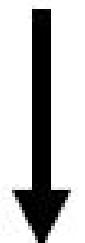
Bias parameters to capture short-term
dependencies

The Problem of Long-Term Dependencies

"The clouds are in the ___"

Why are vanishing gradients a problem?

Multiply many **small numbers** together



Errors due to further back time steps
have smaller and smaller gradients



Bias parameters to capture short-term
dependencies

The Problem of Long-Term Dependencies

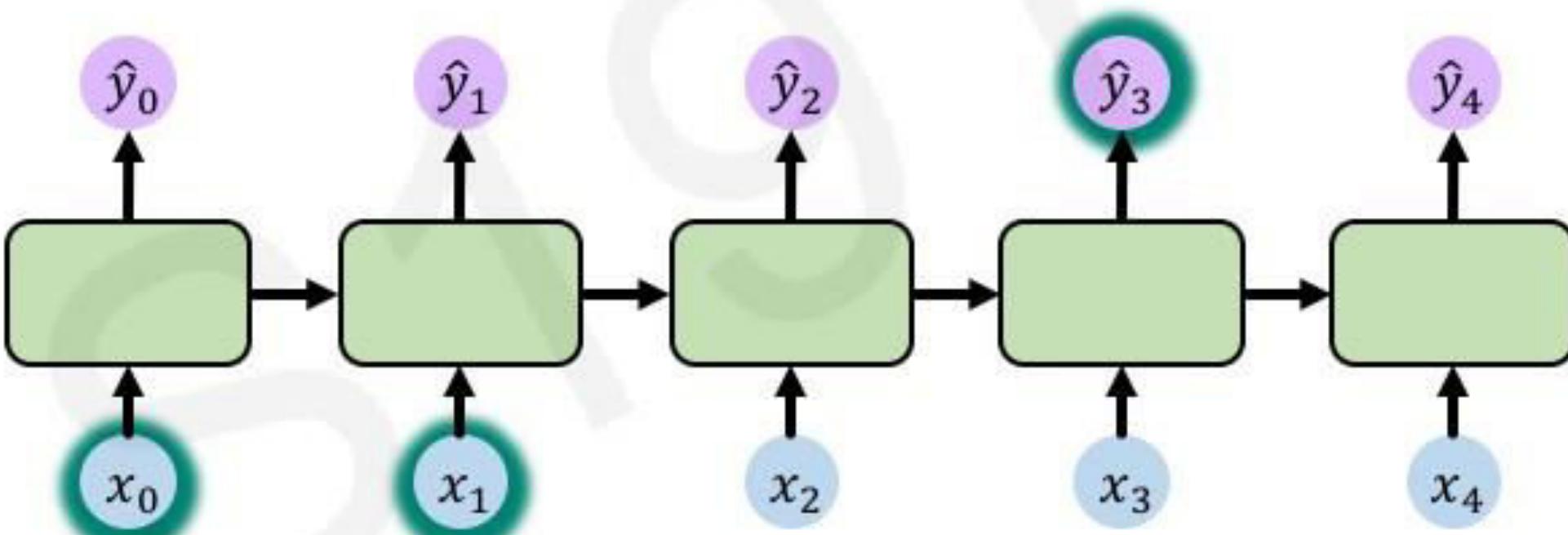
Why are vanishing gradients a problem?

Multiply many **small numbers** together

Errors due to further back time steps
have smaller and smaller gradients

Bias parameters to capture short-term
dependencies

"The clouds are in the ___"



The Problem of Long-Term Dependencies

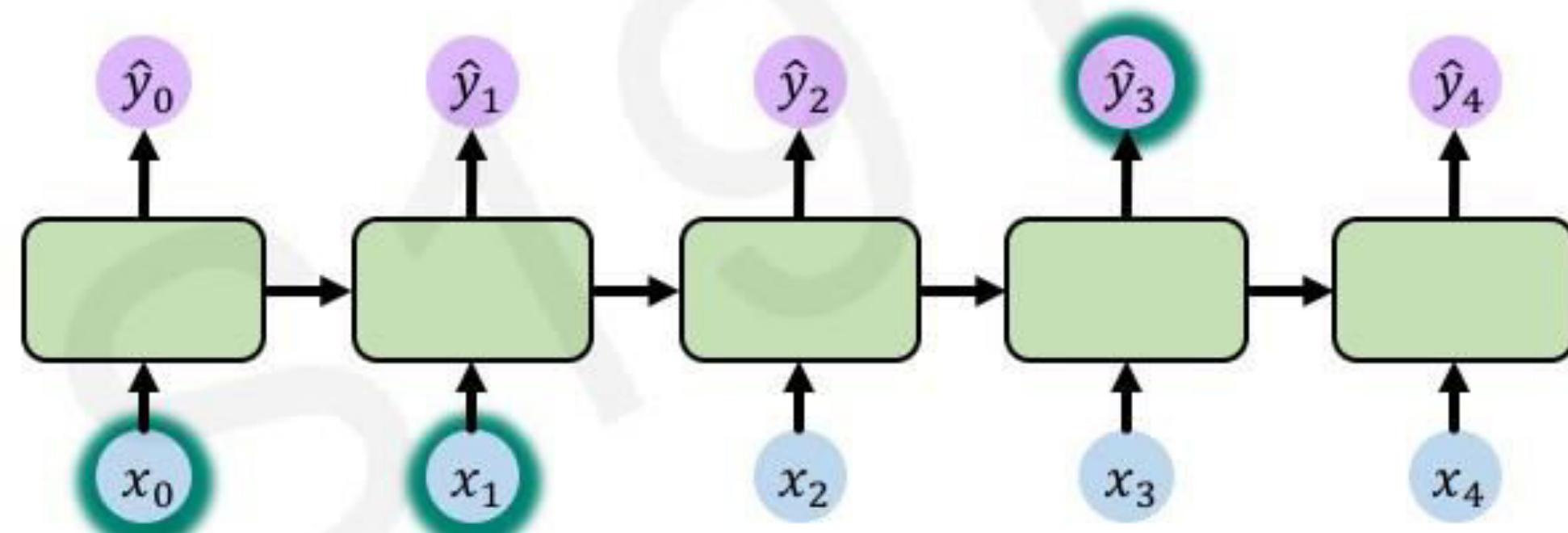
Why are vanishing gradients a problem?

Multiply many **small numbers** together

Errors due to further back time steps
have smaller and smaller gradients

Bias parameters to capture short-term
dependencies

"The clouds are in the ___"



"I grew up in France, ... and I speak fluent ___ "

The Problem of Long-Term Dependencies

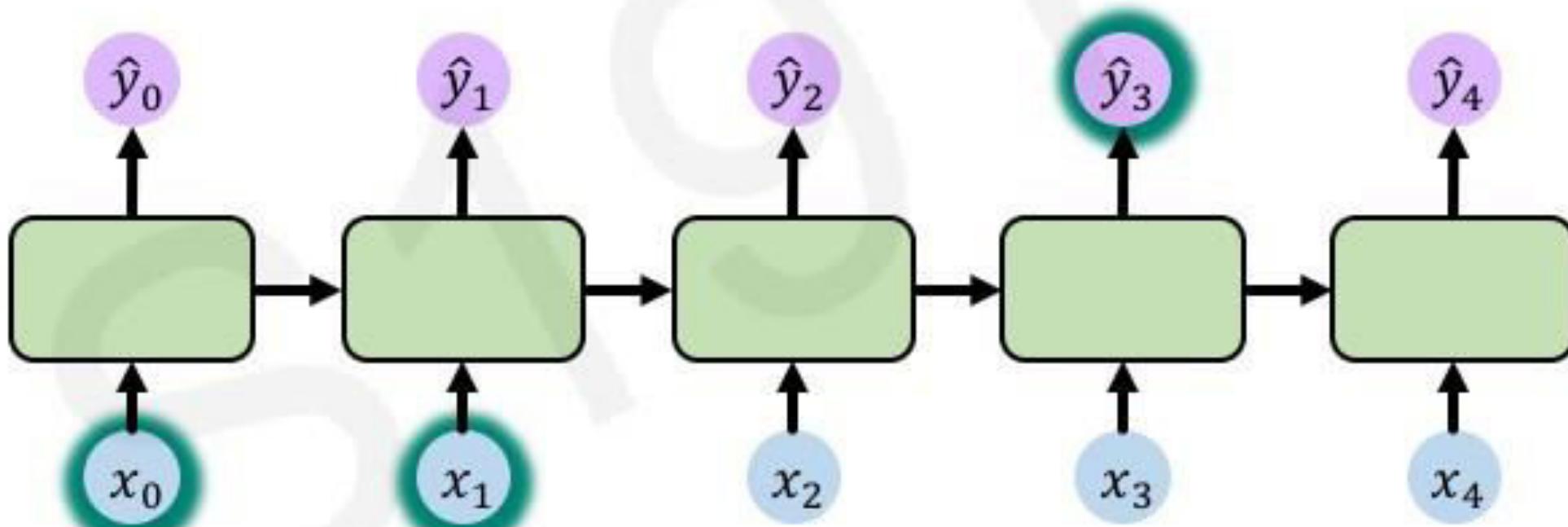
Why are vanishing gradients a problem?

Multiply many **small numbers** together

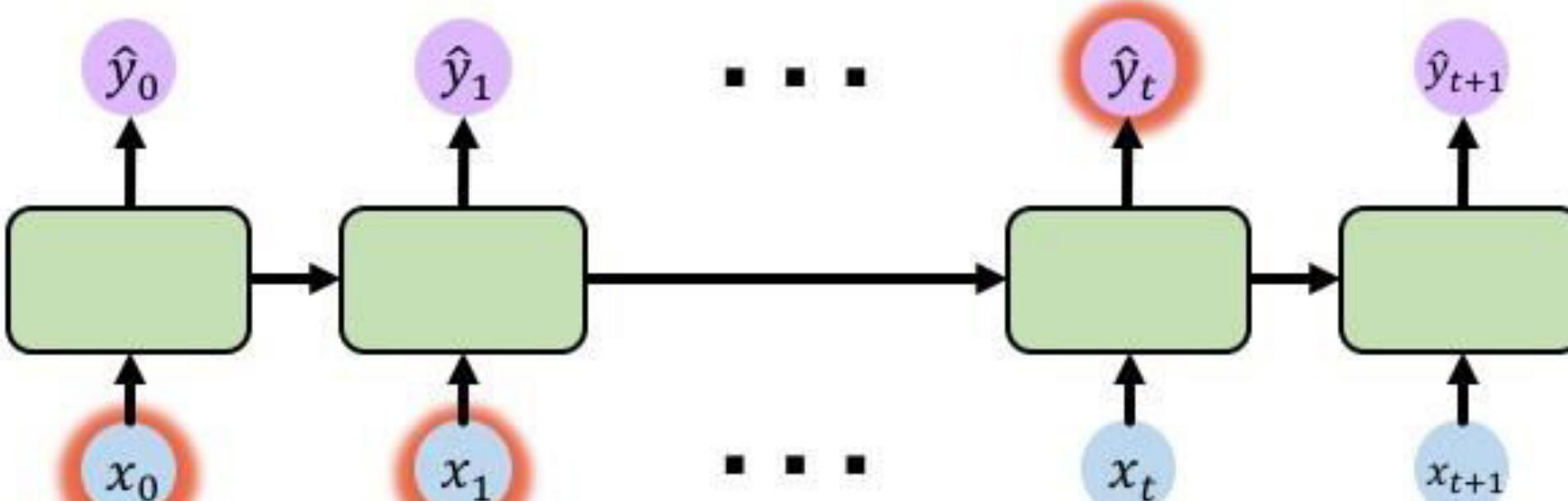
Errors due to further back time steps
have smaller and smaller gradients

Bias parameters to capture short-term
dependencies

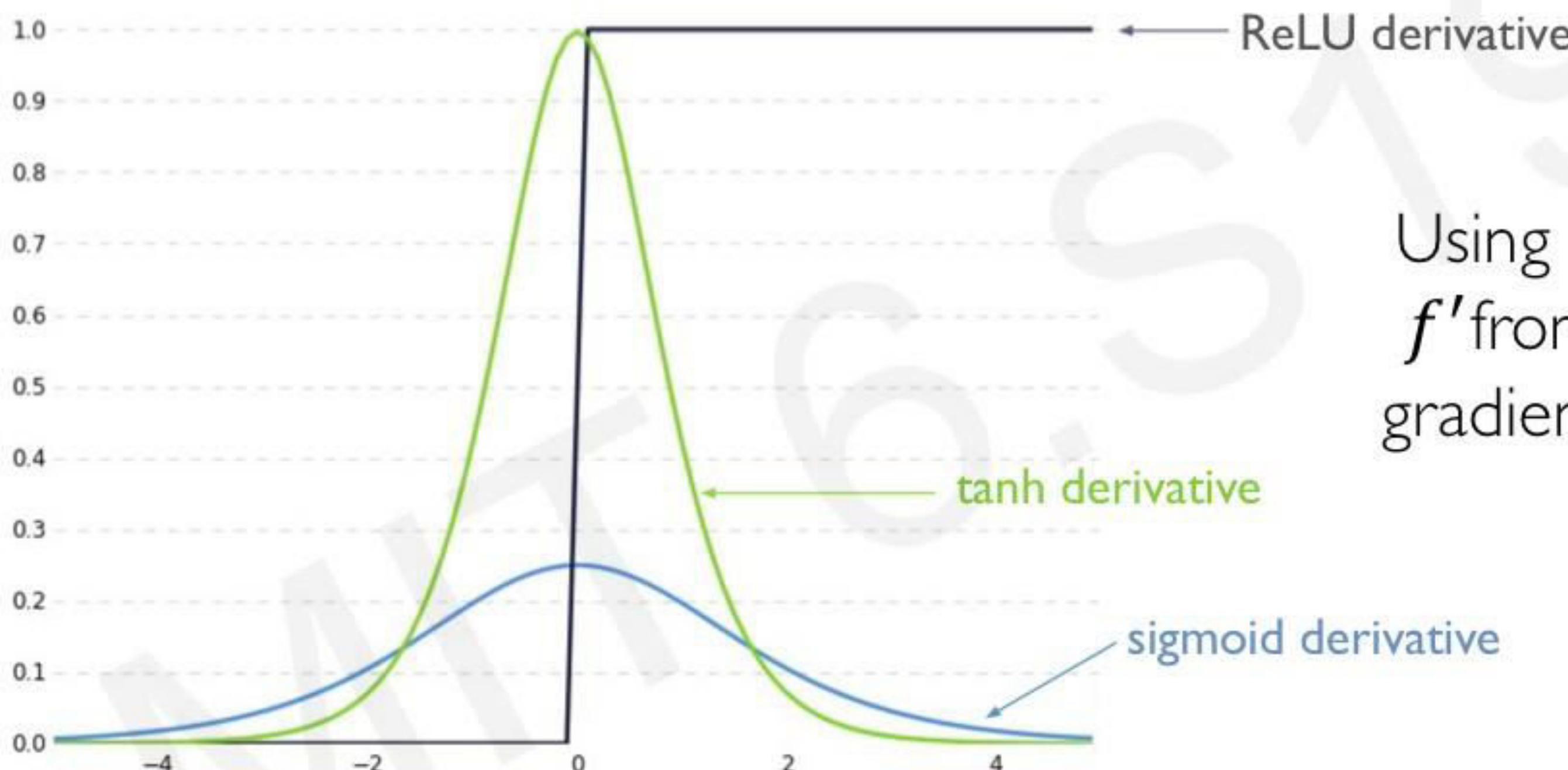
"The clouds are in the ___"



"I grew up in France, ... and I speak fluent ___ "



Trick #1: Activation Functions



Using ReLU prevents
 f' from shrinking the
gradients when $x > 0$

Trick #2: Parameter Initialization

Initialize **weights** to identity matrix

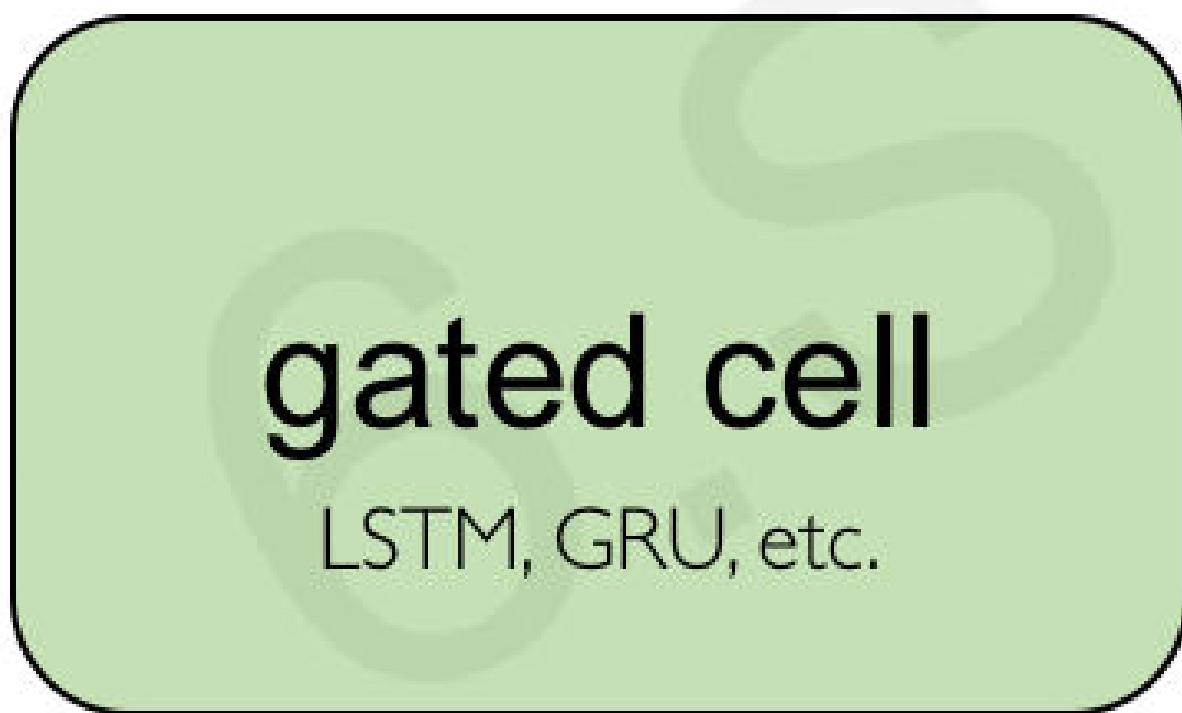
Initialize **biases** to zero

$$I_n = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \end{pmatrix}$$

This helps prevent the weights from shrinking to zero.

Solution #3: Gated Cells

Idea: use a more **complex recurrent unit with gates** to control what information is passed through

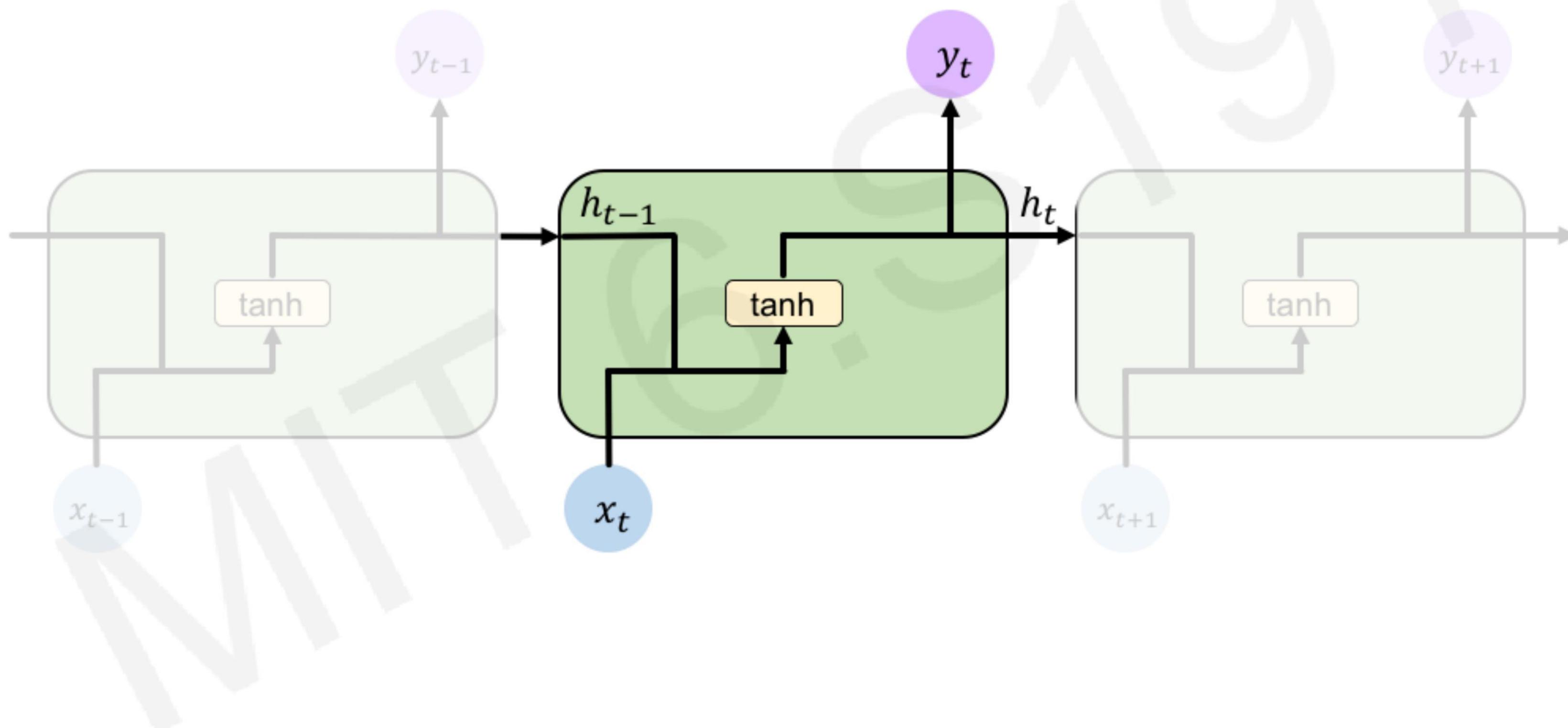


Long Short Term Memory (LSTMs) networks rely on a gated cell to track information throughout many time steps.

Long Short Term Memory (LSTM) Networks

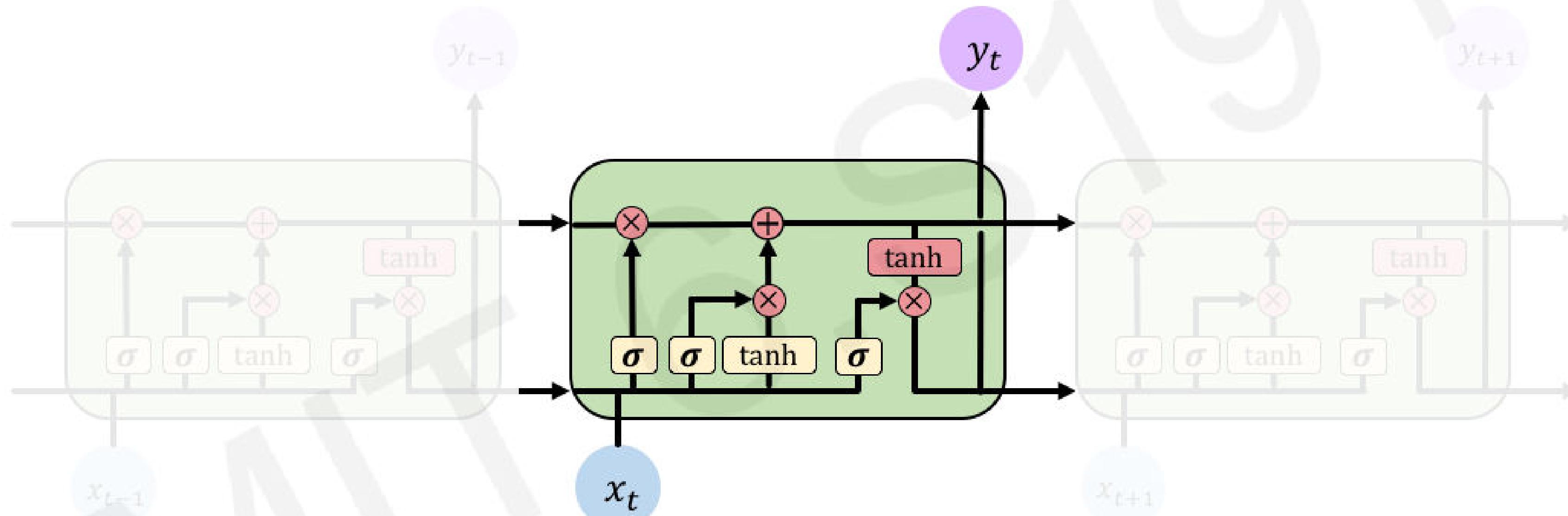
Standard RNN

In a standard RNN, repeating modules contain a **simple computation node**



Long Short Term Memory (LSTMs)

LSTM modules contain **computational blocks** that control information flow

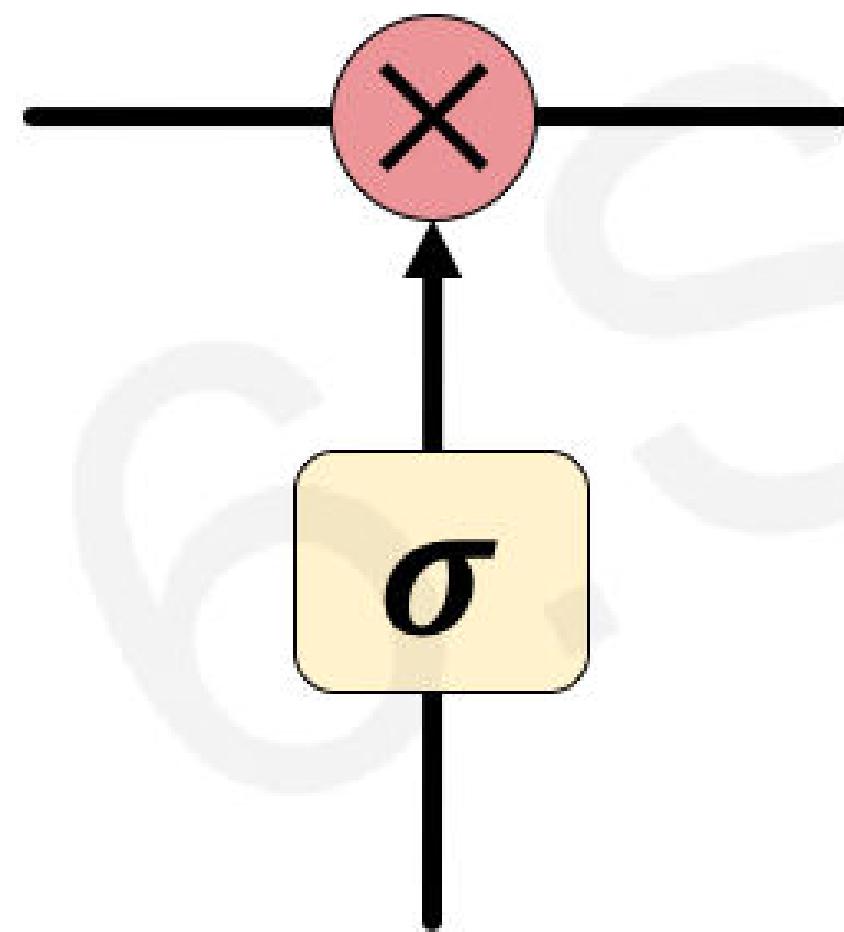


LSTM cells are able to track information throughout many timesteps

 `tf.keras.layers.LSTM(num_units)`

Long Short Term Memory (LSTMs)

Information is **added** or **removed** through structures called **gates**

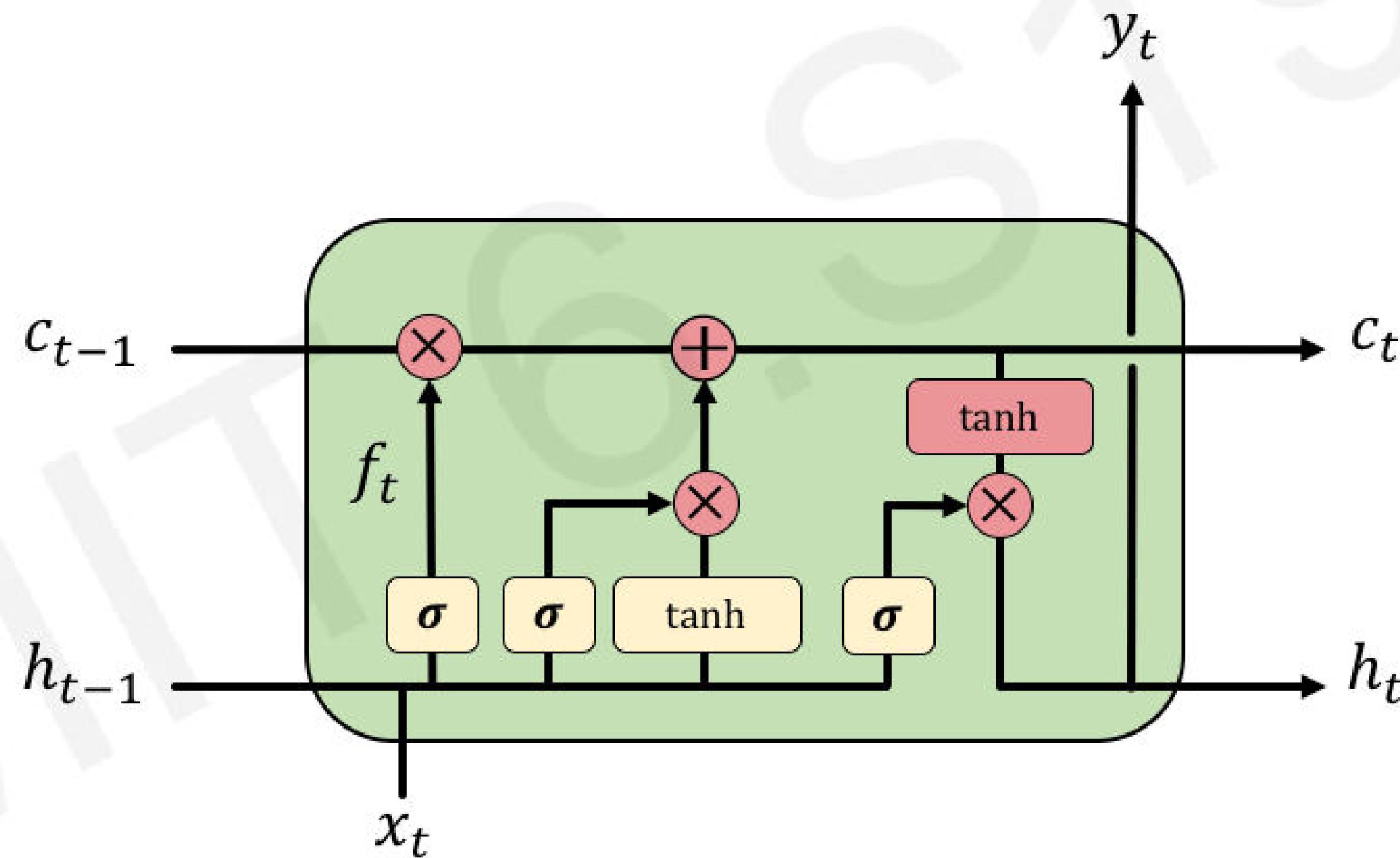


Gates optionally let information through, for example via a sigmoid neural net layer and pointwise multiplication

Long Short Term Memory (LSTMs)

How do LSTMs work?

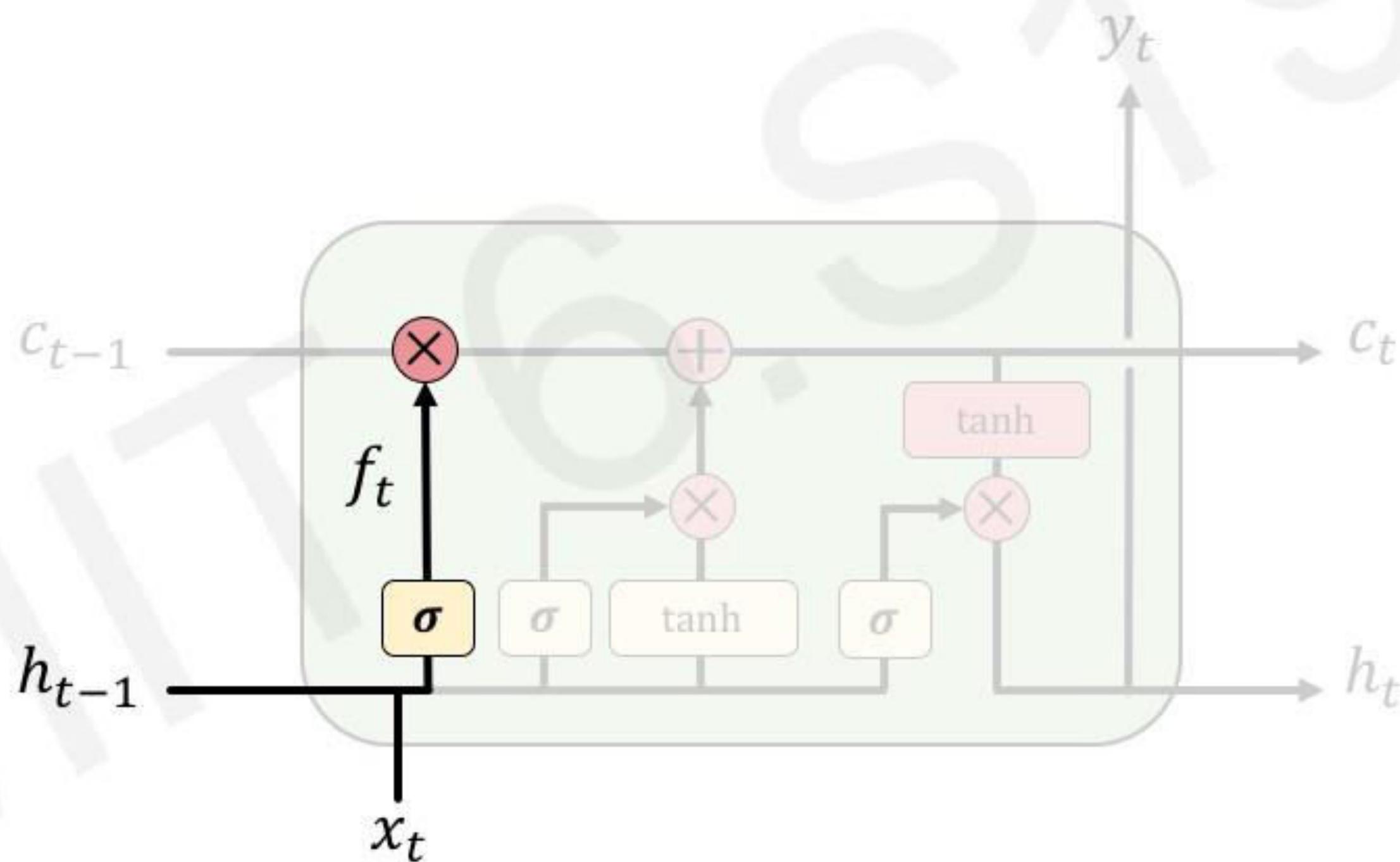
- 1) Forget
- 2) Store
- 3) Update
- 4) Output



Long Short Term Memory (LSTMs)

- 1) Forget
- 2) Store
- 3) Update
- 4) Output

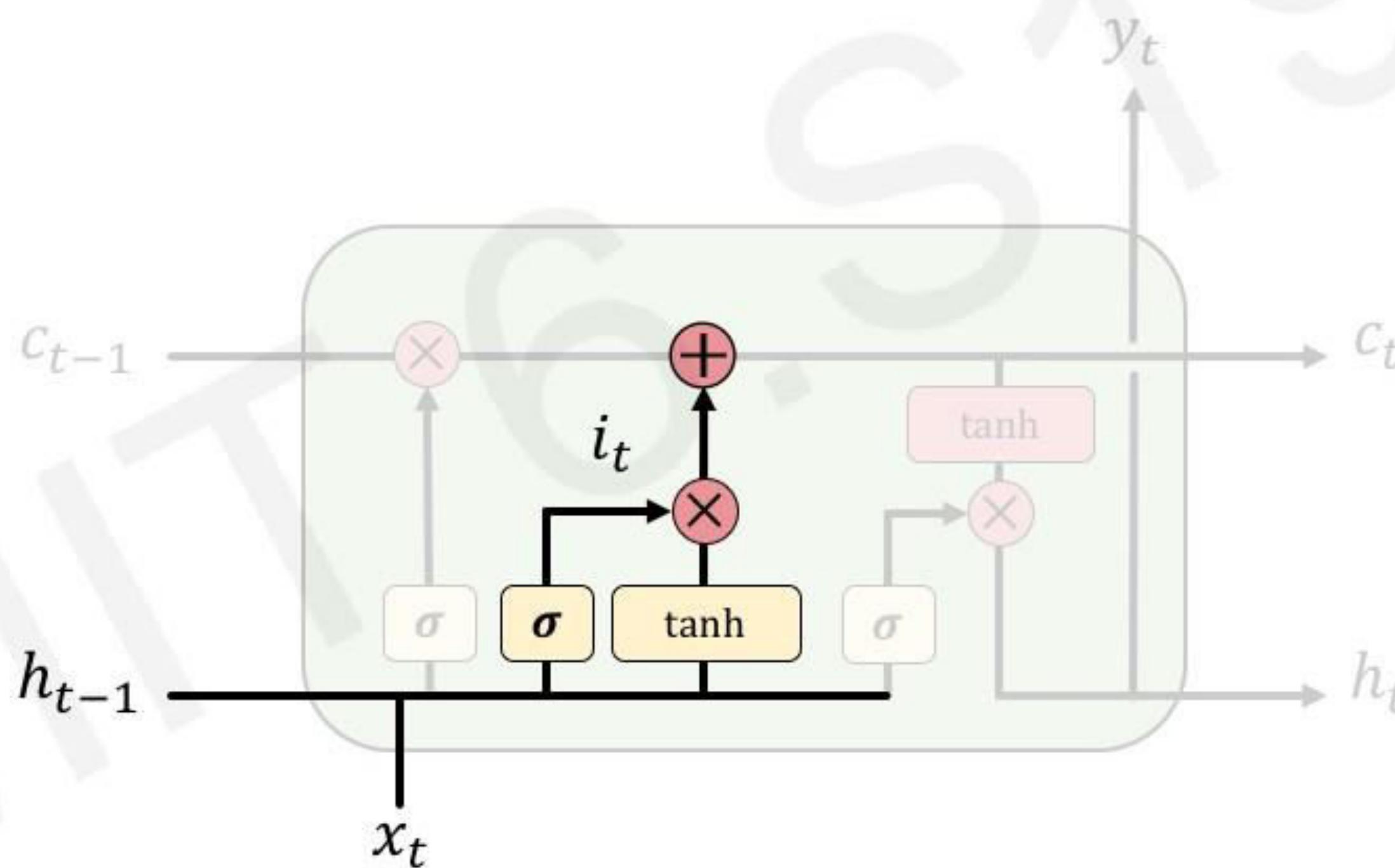
LSTMs **forget irrelevant** parts of the previous state



Long Short Term Memory (LSTMs)

- 1) Forget
- 2) Store**
- 3) Update
- 4) Output

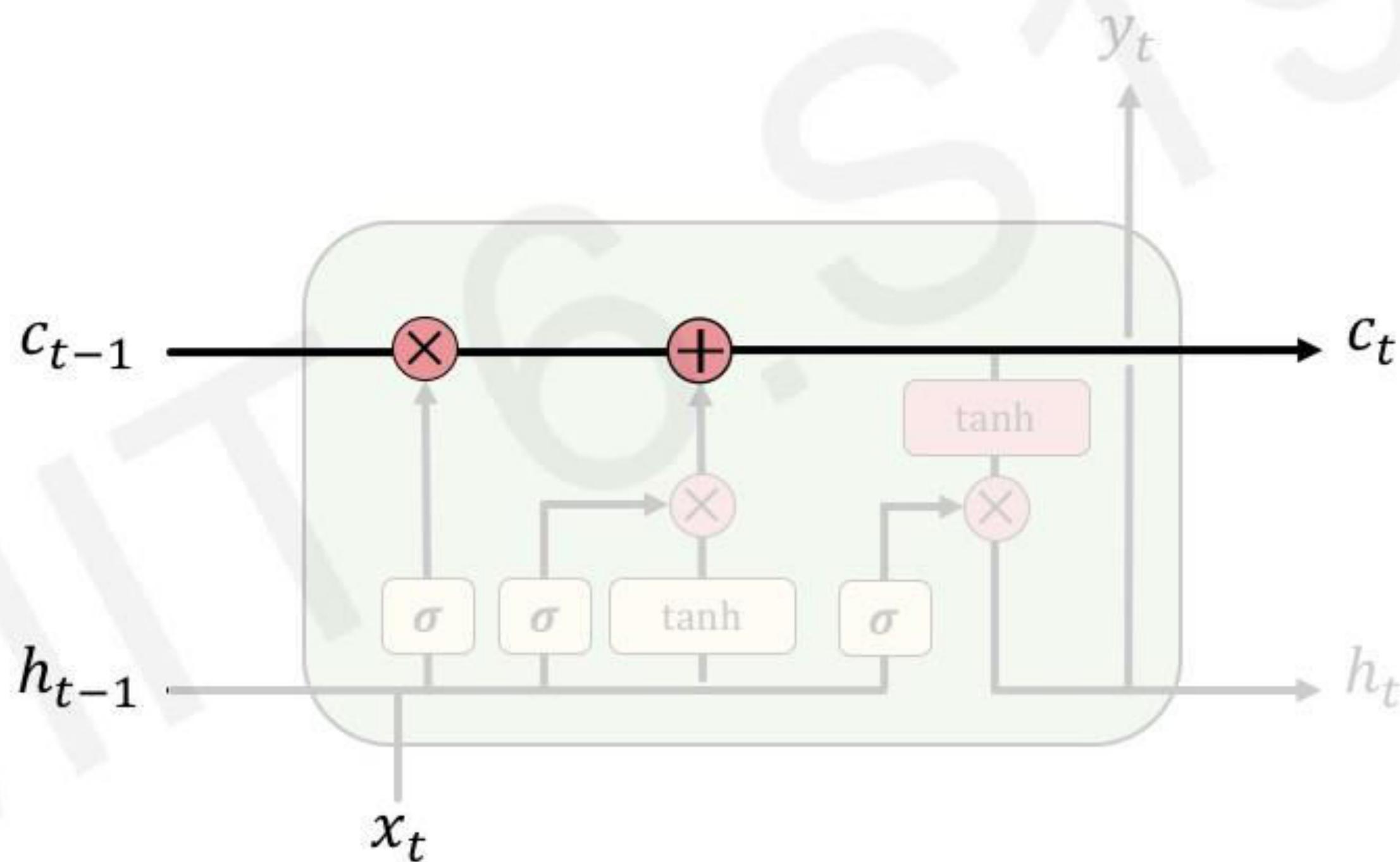
LSTMs **store relevant** new information into the cell state



Long Short Term Memory (LSTMs)

- 1) Forget
- 2) Store
- 3) Update**
- 4) Output

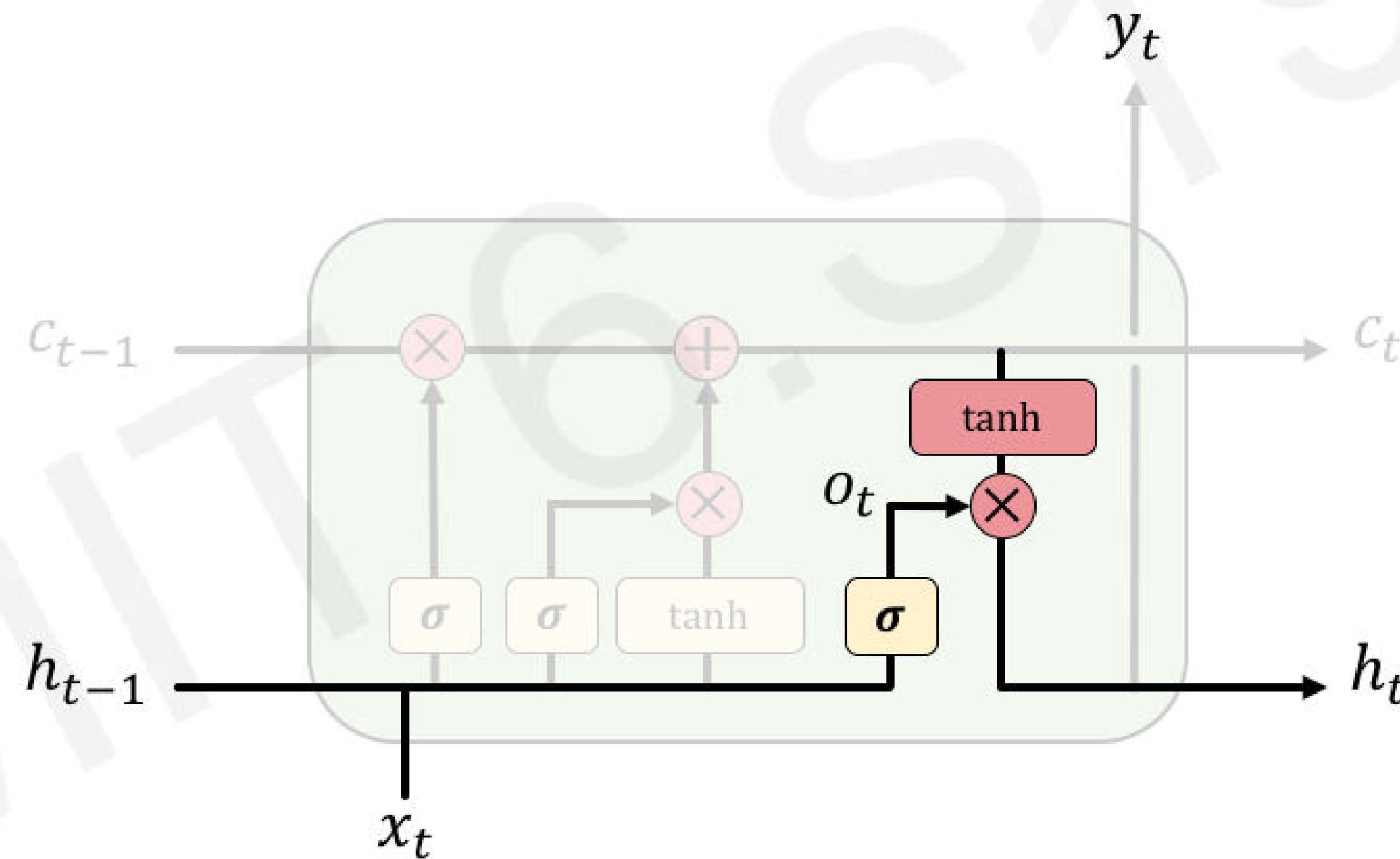
LSTMs **selectively update** cell state values



Long Short Term Memory (LSTMs)

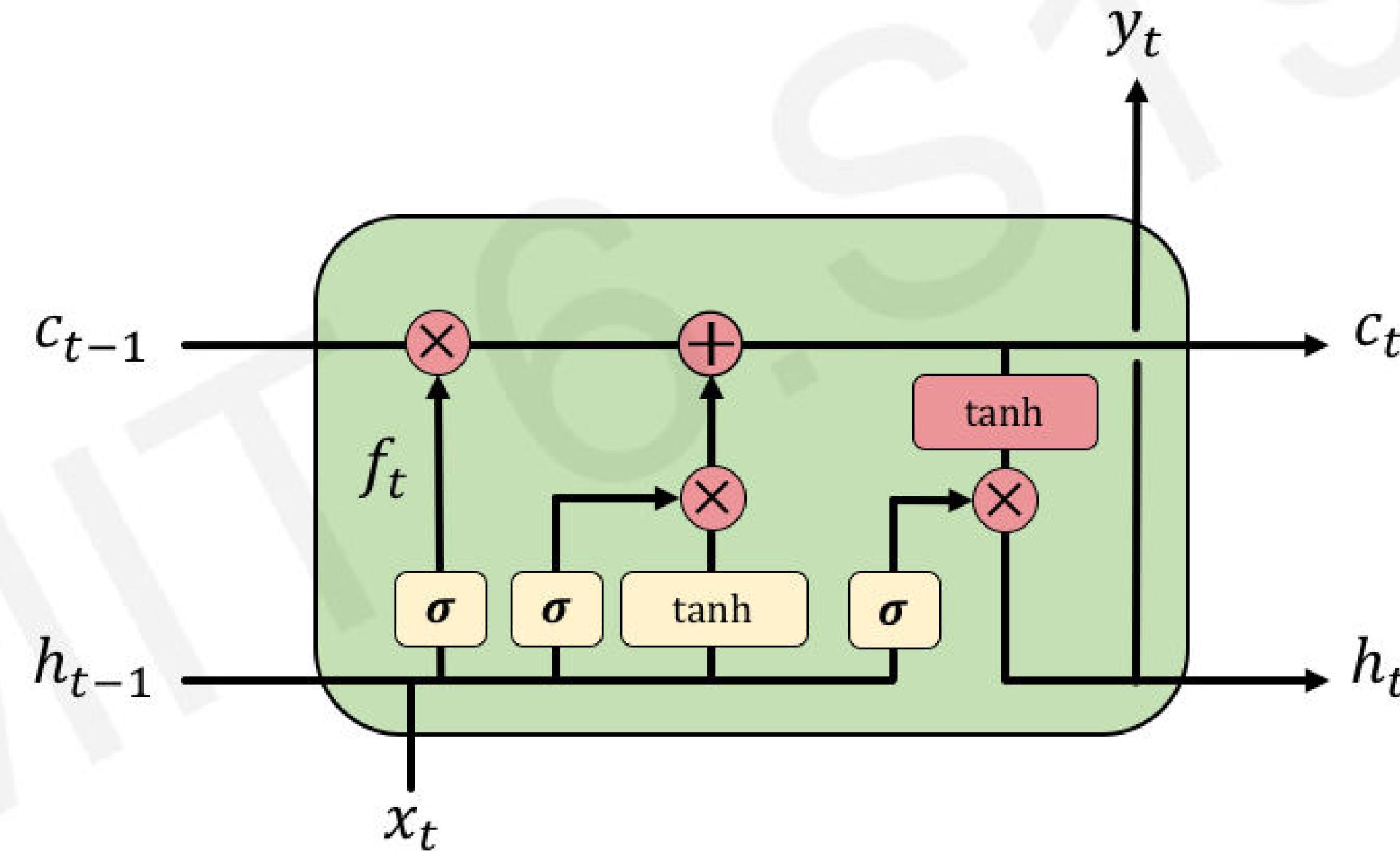
- 1) Forget
- 2) Store
- 3) Update
- 4) Output**

The **output gate** controls what information is sent to the next time step



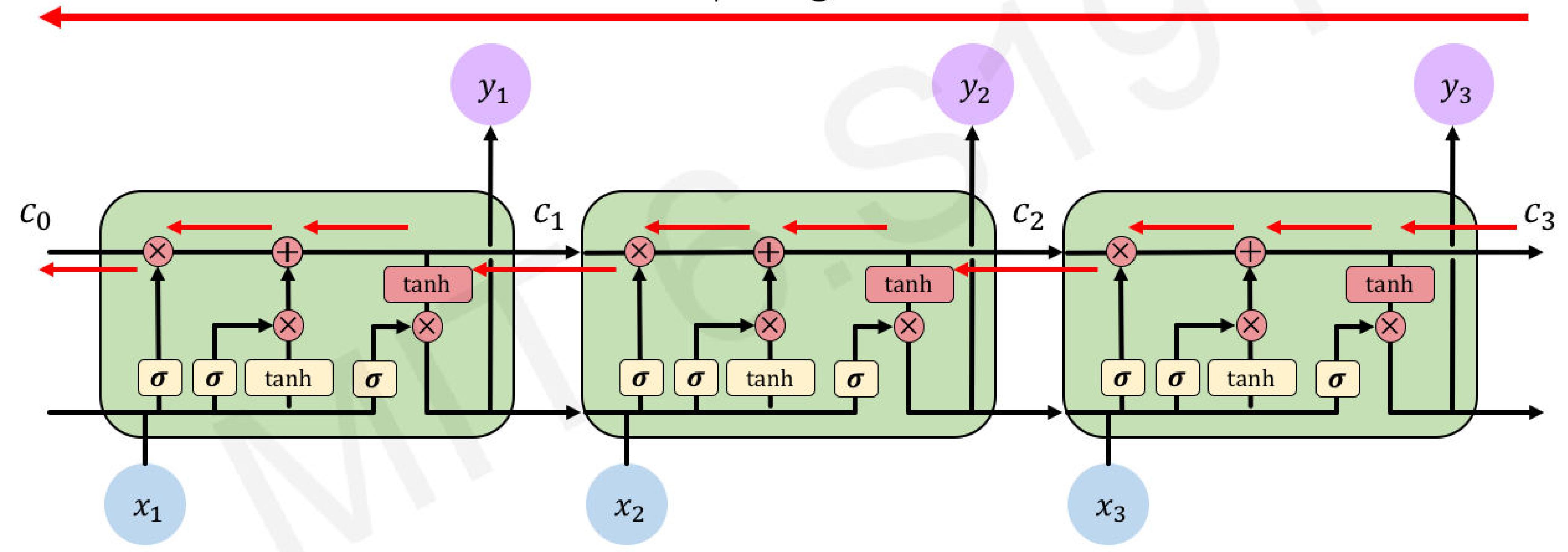
Long Short Term Memory (LSTMs)

- 1) Forget
- 2) Store
- 3) Update
- 4) Output



LSTM Gradient Flow

Uninterrupted gradient flow!

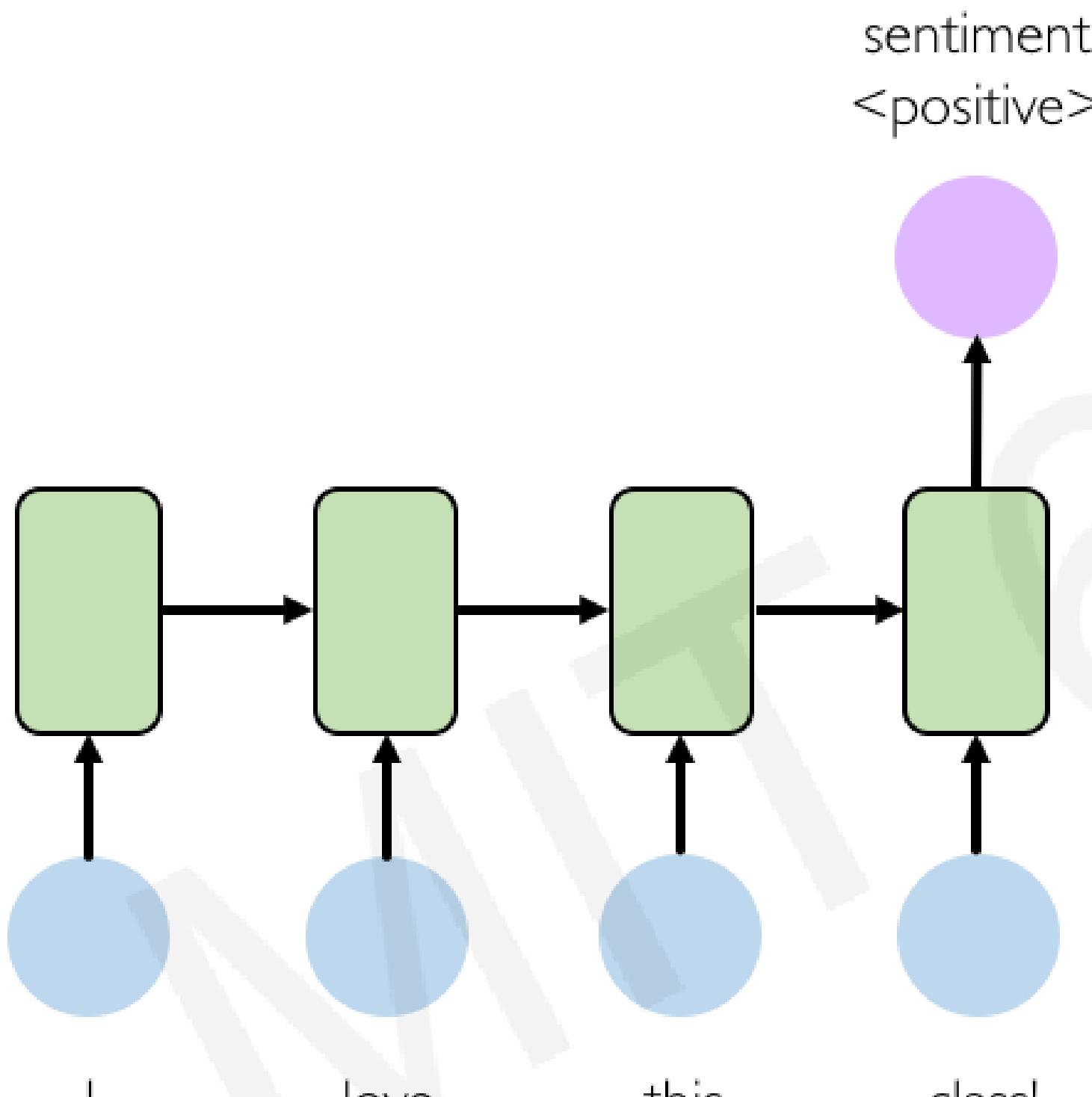


LSTMs: Key Concepts

1. Maintain a **separate cell state** from what is outputted
2. Use **gates** to control the **flow of information**
 - **Forget** gate gets rid of irrelevant information
 - **Store** relevant information from current input
 - Selectively **update** cell state
 - **Output** gate returns a filtered version of the cell state
3. Backpropagation through time with **uninterrupted gradient flow**

RNN Applications

Example Task: Sentiment Classification



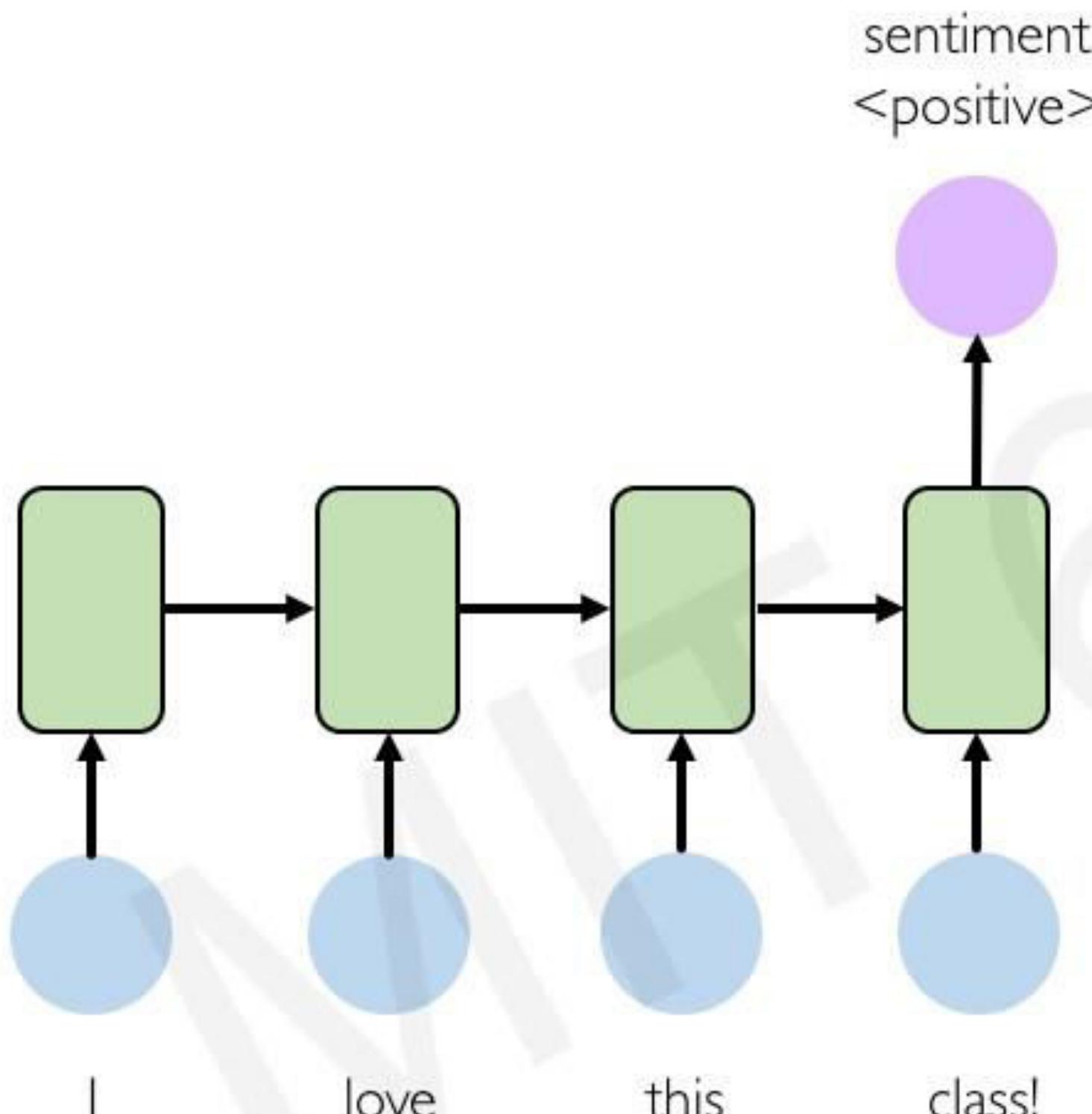
Input: sequence of words

Output: probability of having positive sentiment



```
loss = tf.nn.softmax_cross_entropy_with_logits(y, predicted)
```

Example Task: Sentiment Classification



Tweet sentiment classification



Ivar Hagendoorn
@IvarHagendoorn

Follow



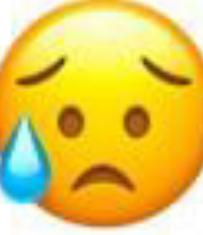
The @MIT Introduction to #DeepLearning is definitely one of the best courses of its kind currently available online introtodeeplearning.com

12:45 PM - 12 Feb 2018



Angels-Cave
@AngelsCave

Follow

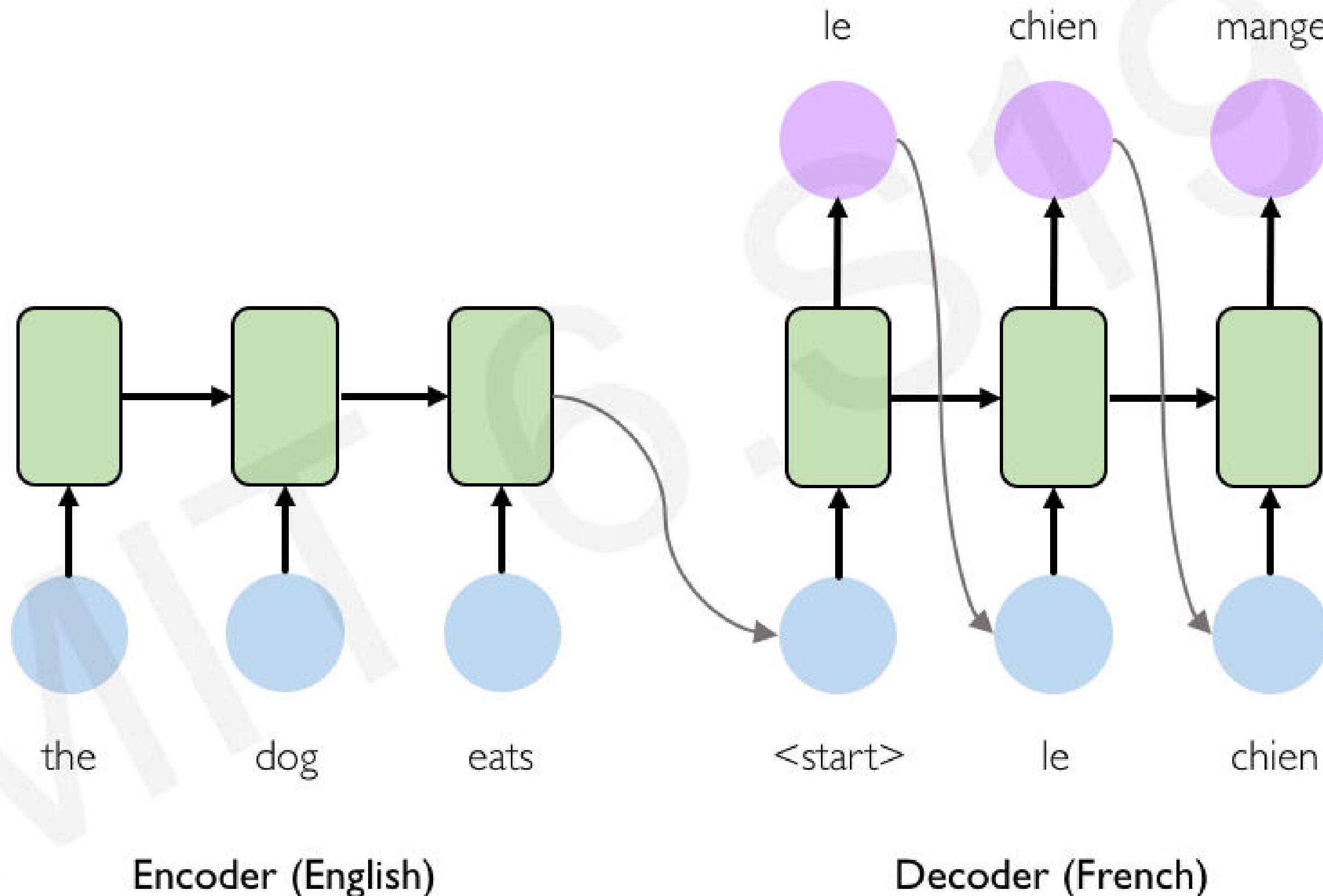


Replying to @Kazuki2048

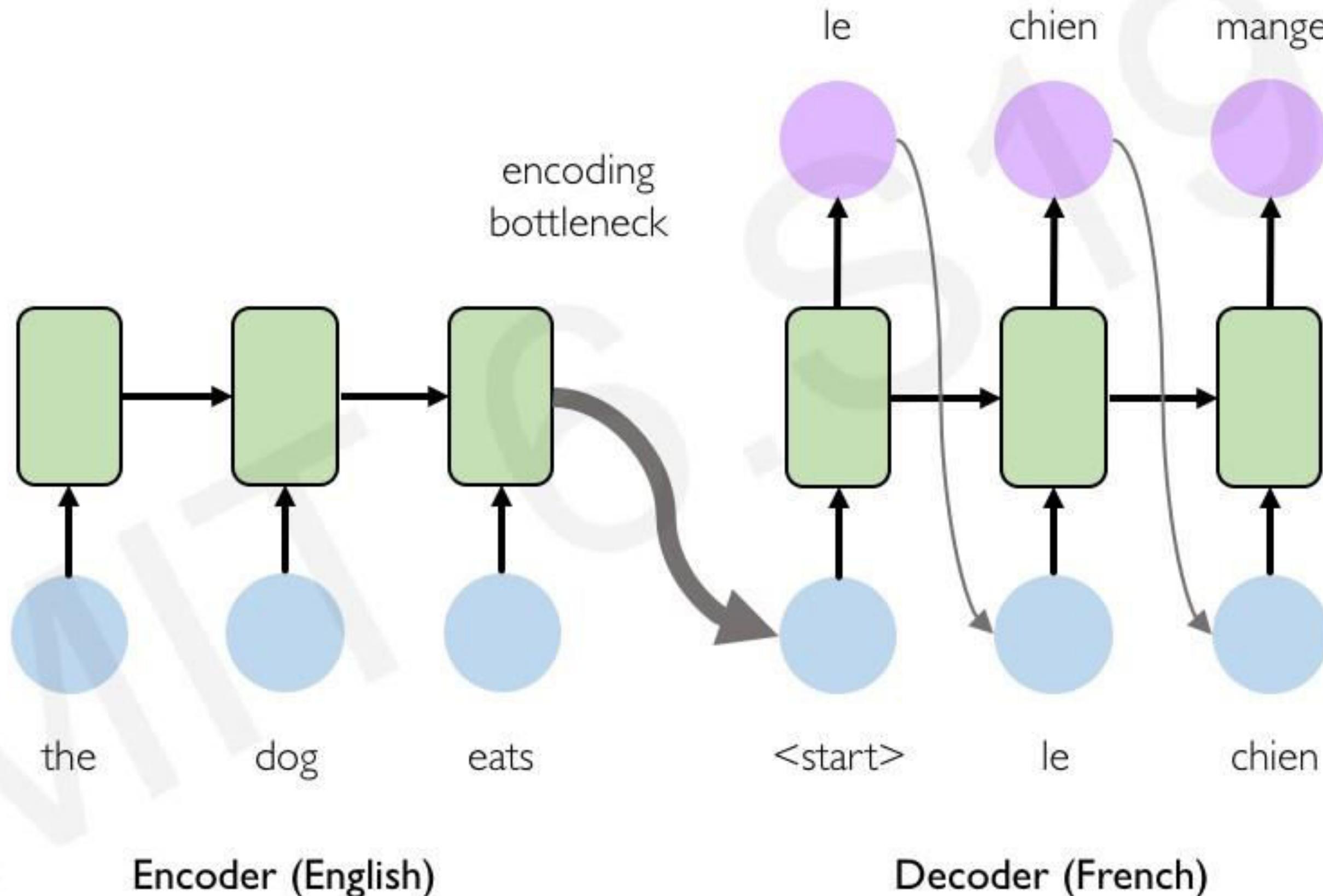
I wouldn't mind a bit of snow right now. We haven't had any in my bit of the Midlands this winter! :(

2:19 AM - 25 Jan 2019

Example Task: Machine Translation

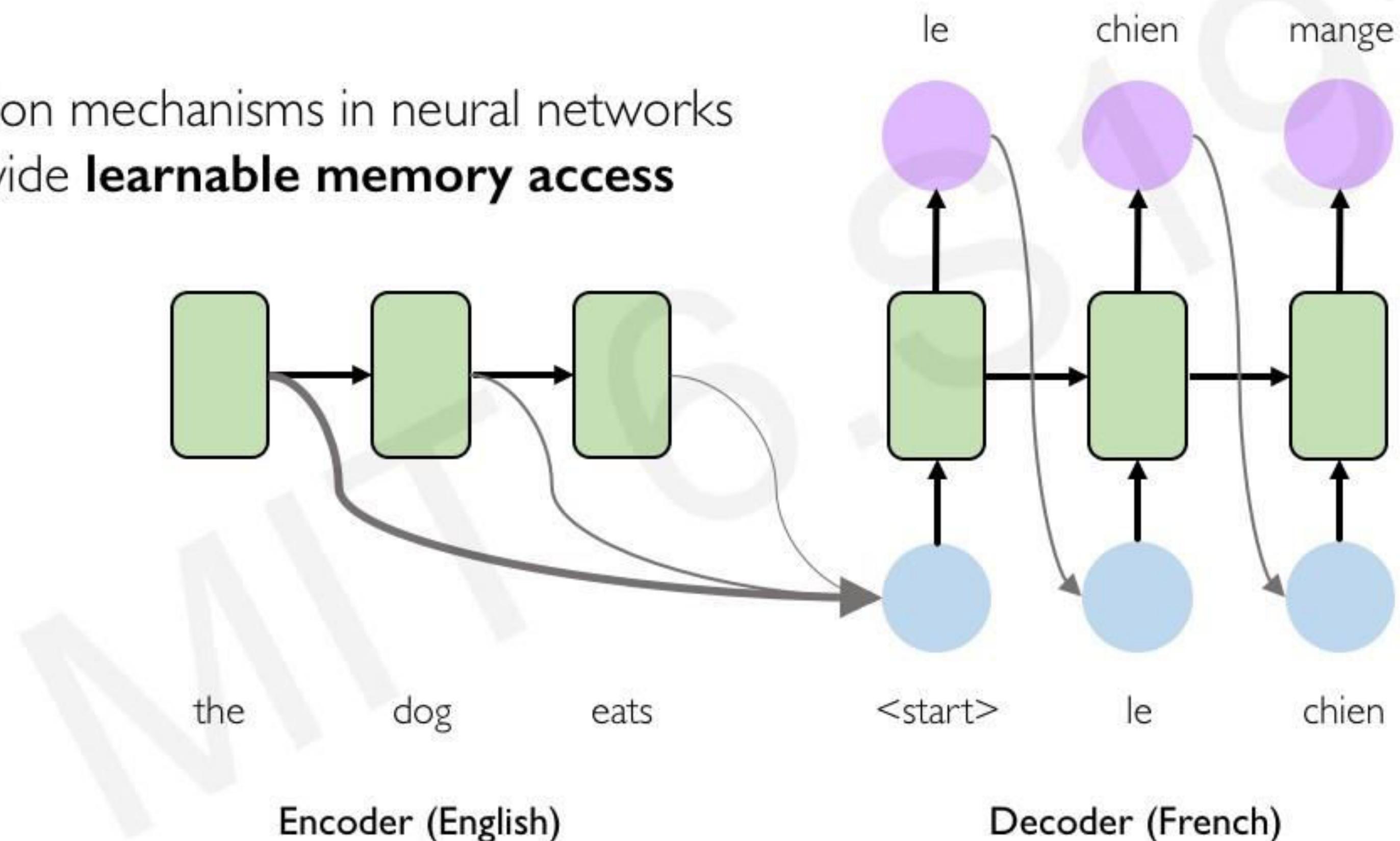


Example Task: Machine Translation



Attention Mechanisms

Attention mechanisms in neural networks provide **learnable memory access**

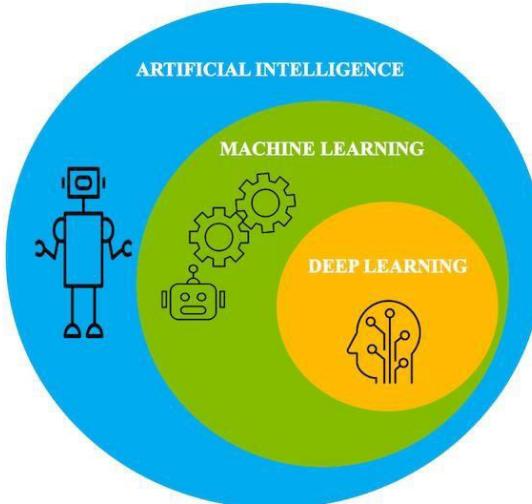


Deep Learning for Sequence Modeling: Summary

1. RNNs are well suited for **sequence modeling** tasks
2. Model sequences via a **recurrence relation**
3. Training RNNs with **backpropagation through time**
4. Gated cells like **LSTMs** let us model **long-term dependencies**
5. Models for **music generation**, classification, machine translation, and more



Lambton
College



Lambton College
School of Computer Studies

AML-3104 Neural Networks
and Deep Learning

How to get in touch?

Send email to

- Expect response within 24 hours

Please

- Include **[AML-3104_#]** at the beginning into the subject line
- Send email from your college account (college policy)

Who are you?

Required Materials

- Neural Networks and Deep Learning: A Textbook

Author: Charu C. Aggarwal

ISBN-13: 978-3319944623

Publisher: Springer

Published: 26 August 2018

- Deep Learning

Author: Ian Goodfellow, Yoshua Bengio and Aaron Courville

ISBN-13: 978-0262035613

Publisher: The MIT Press

Published: 18 November 2016

Learning Outcomes

- Explore Neural networks.
- Evaluate the Applications of Neural Networks.
- Demonstrate Machine Learning Basics.
- Explore Machine Learning Algorithms.
- Basic Architecture of Neural Networks.
- Training of a Neural Network
- Evaluate Deep Neural Network
- Recommend Deep Neural Network Implementation

Review Policies

- Late Assignments/exams

Evaluation

- Evaluation methods

Tests (40%):

- Midterm Test (20%)
- Final Exam (20%)

Assignments and Projects (60%):

- Assignment (20%)
- Final Term Project (40%)

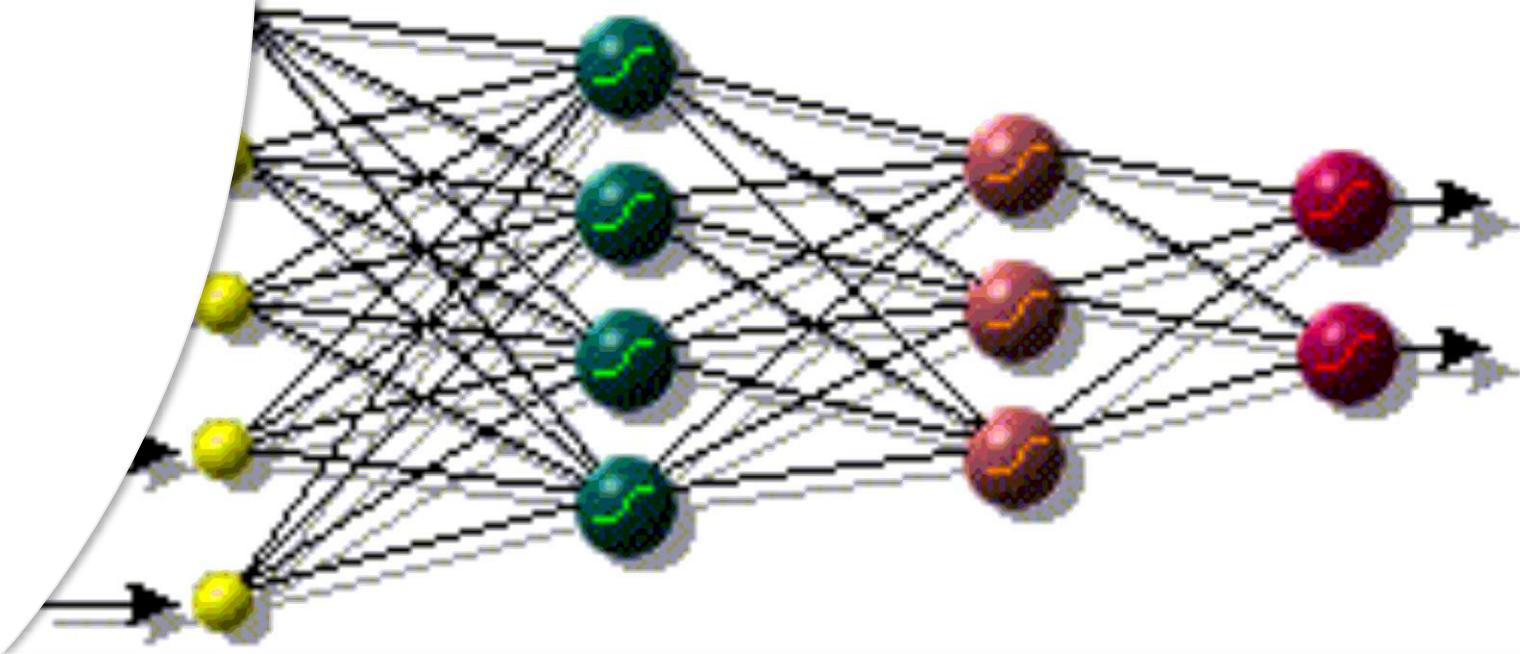
- Marking scheme

Mark(%)	Grade	Mark(%)	Grade
94-100	A+	67-69	C+
87-93	A	63-66	C
80-86	A-	60-62	C-
77-79	B+	50-59	D
73-76	B	0-49	F
70-72	B-		



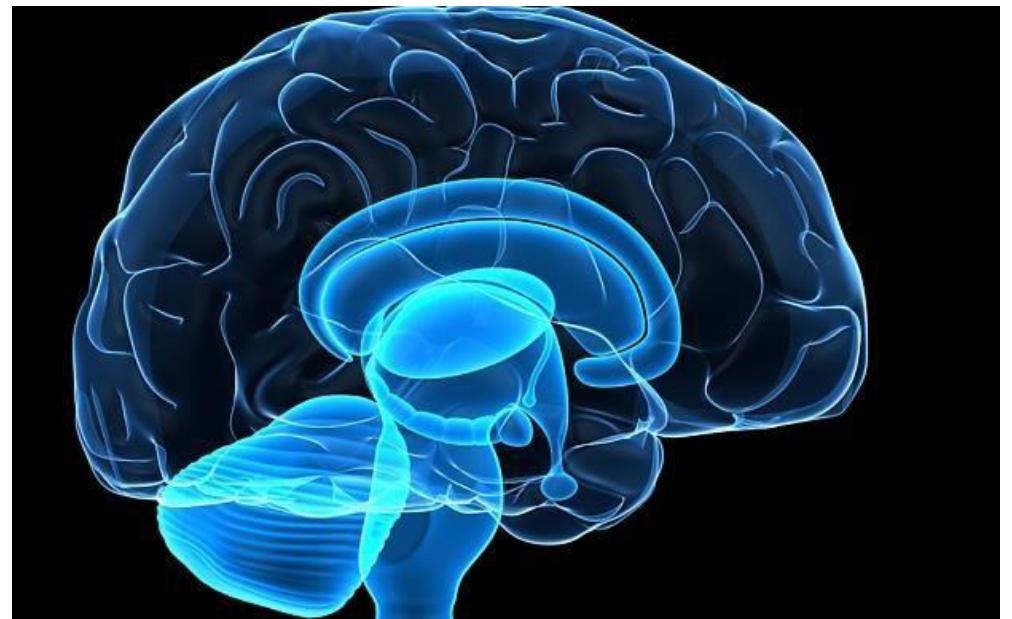
Any questions so far?
Any comments?

Neural Networks



Human Brain

- Consist of special cells
 - Neurons

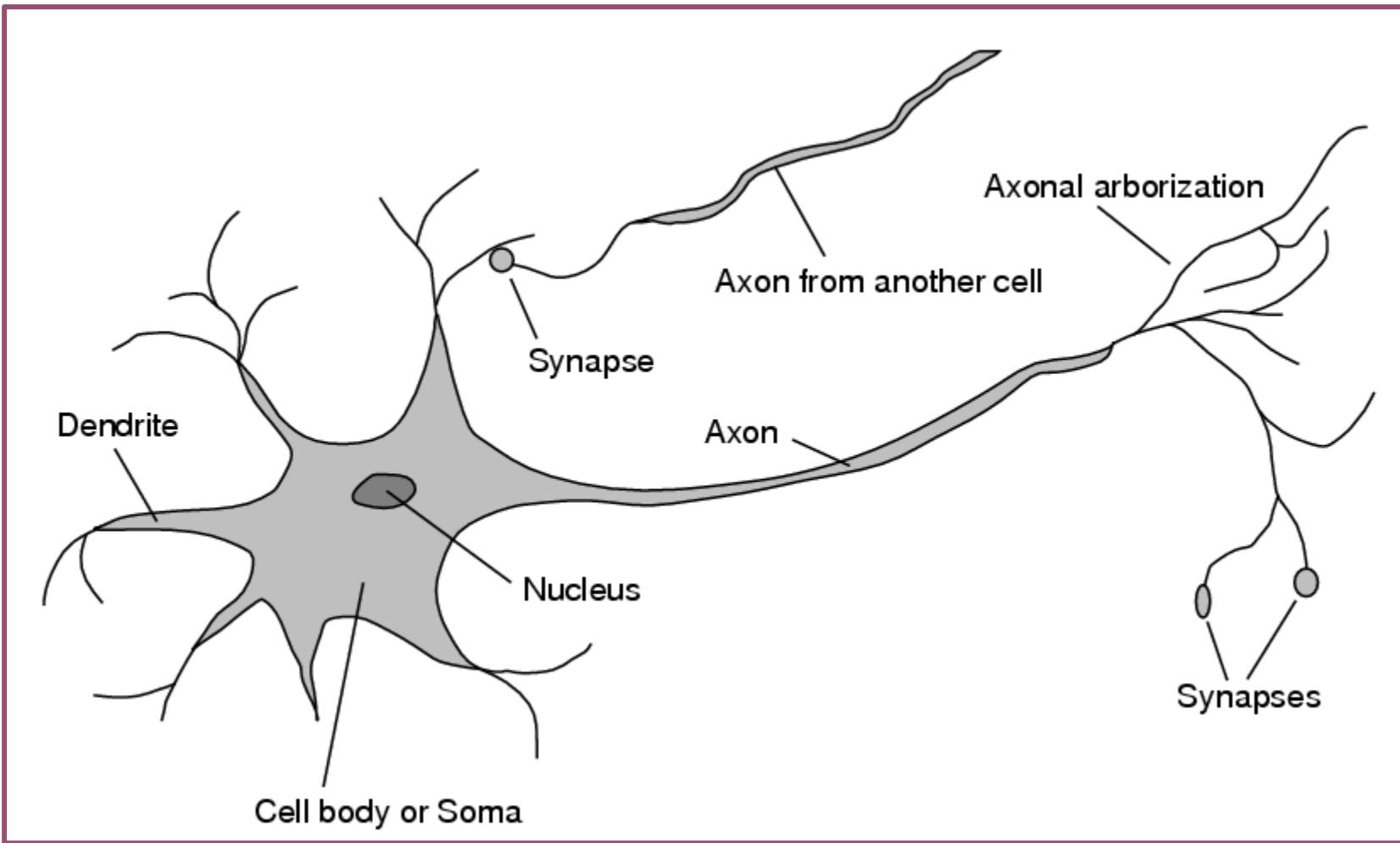


Features of the Brain

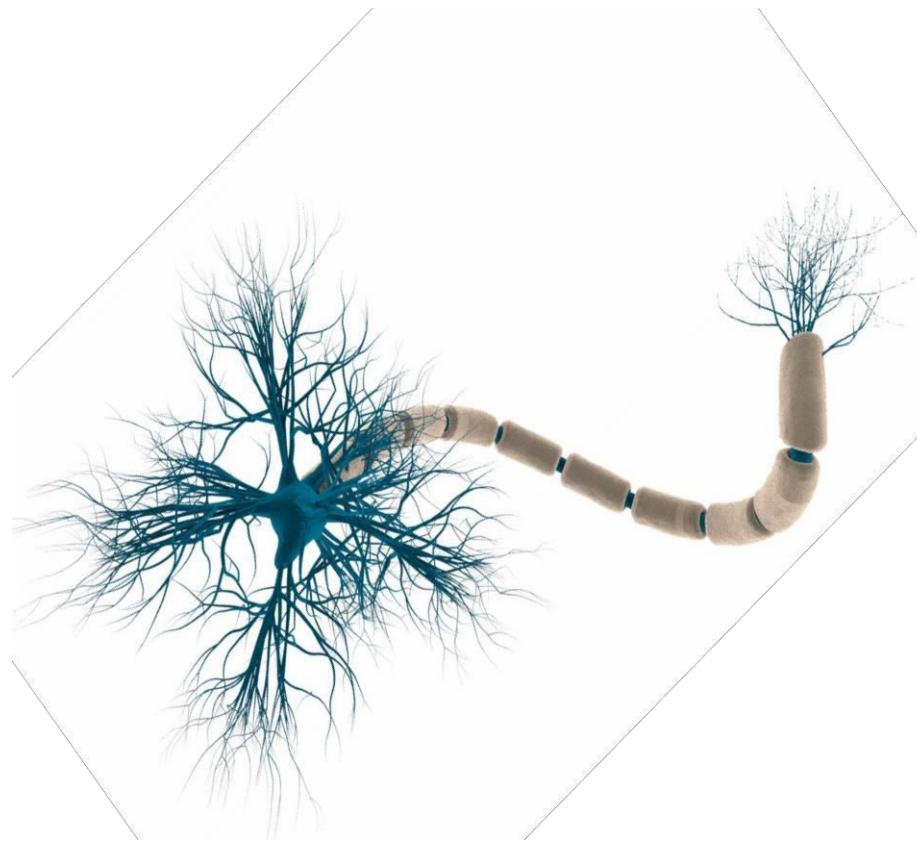
- Ten billion (10^{10}) neurons
- On average, several thousand connections
- Hundreds of operations per second
- Die off frequently (never replaced)
- Compensates for problems by massive parallelism



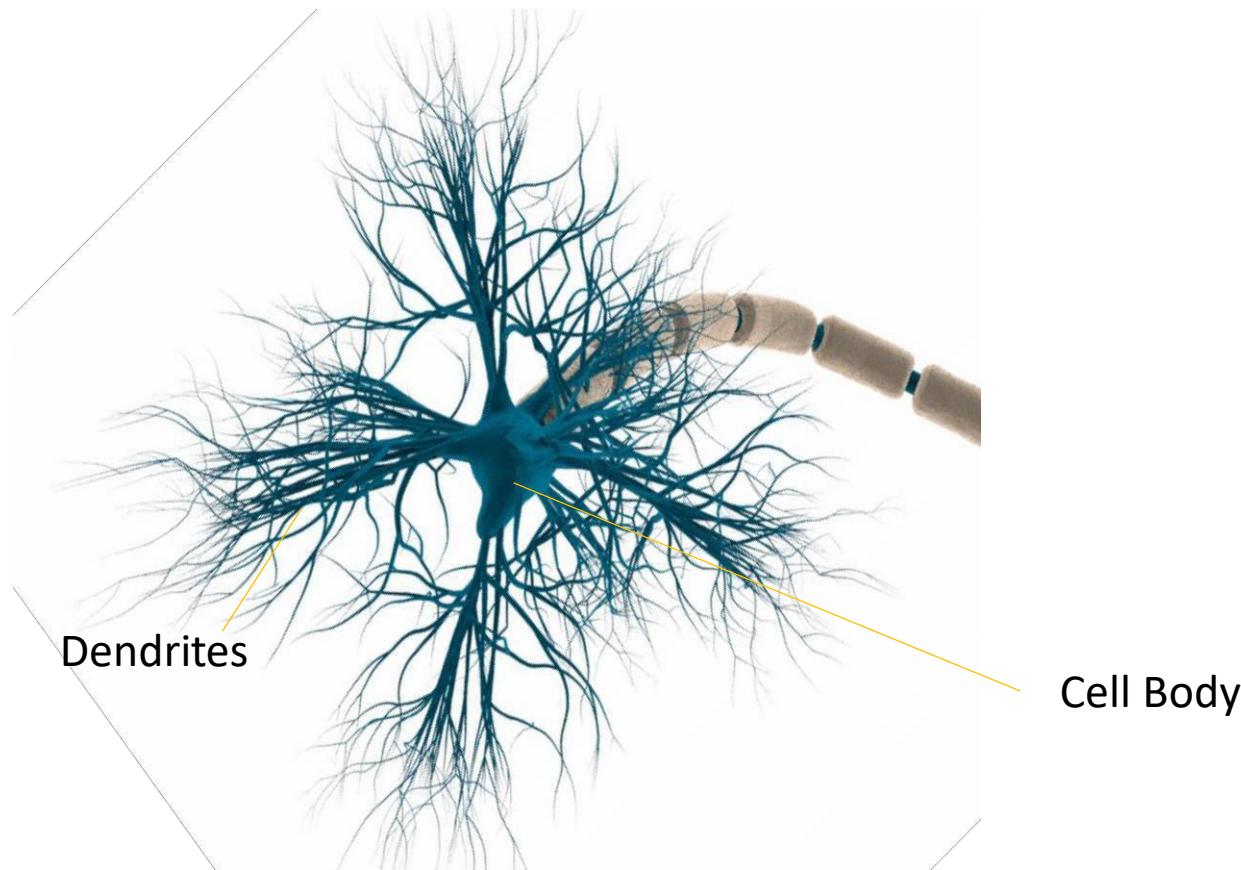
Structure of Neuron



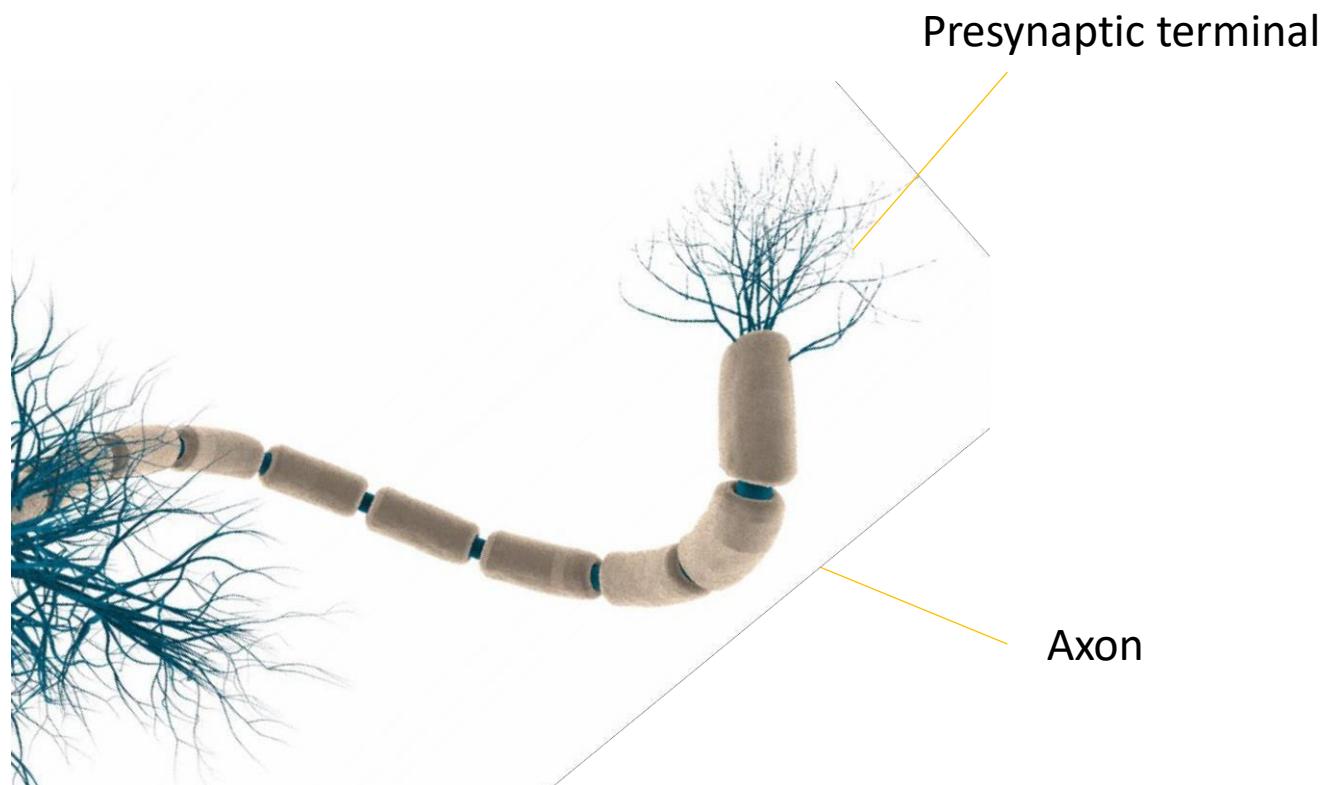
Neuron



Neuron Details



Neuron Details



Structure of Neuron

A neuron has a cell body, a branching **input** structure (the dendr**I**te) and a branching **output** structure (the ax**O**n)

- Axons connect to dendrites via synapses.
- Electro-chemical signals are propagated from the dendritic input, through the cell body, and down the axon to other neurons

Structure of Neuron

- A neuron only fires if its input signal exceeds a certain amount (the **threshold**) in a short time period.
- Synapses vary in strength
 - Good connections allowing a large signal
 - Slight connections allow only a weak signal.

Artificial Neural Networks



Image: https://miro.medium.com/max/1400/1*jJs3ZVz8xMEjvThtFY4gDA.jpeg

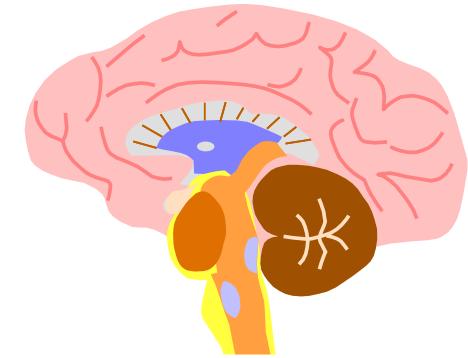
Artificial Neural Networks

❑ History

- 1943-McCulloch & Pitts are generally recognised as the designers of the first neural network
- 1949-First learning rule
- 1969-Minsky & Papert - perceptron limitation - Death of ANN
- 1980's - Re-emergence of ANN - multi-layer networks

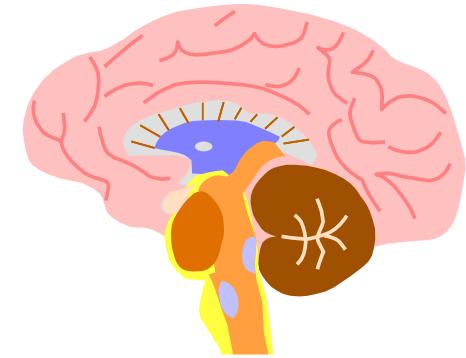
The Biological Inspiration

- The brain has been extensively studied by scientists.
- Vast complexity prevents all but rudimentary understanding.
- Even the behaviour of an individual neuron is extremely complex

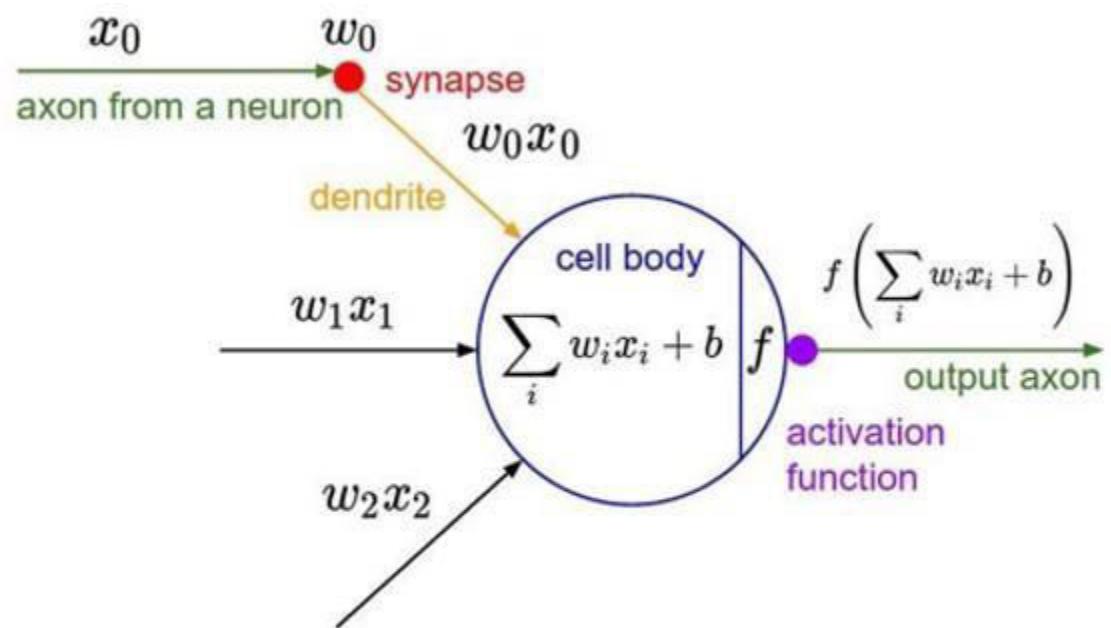
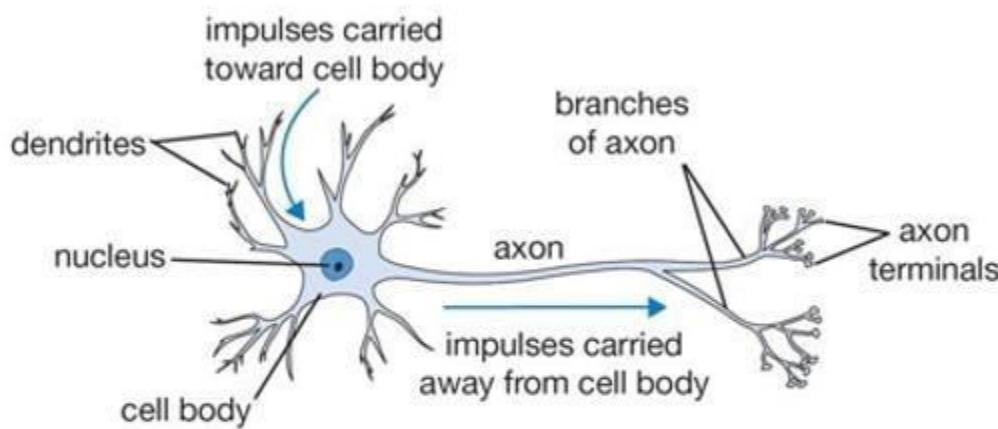


The Biological Inspiration

- Single “percepts” distributed among many neurons
- Localized parts of the brain are responsible for certain well-defined functions (e.g. vision, motion).



Inspiration: Neuron Cells

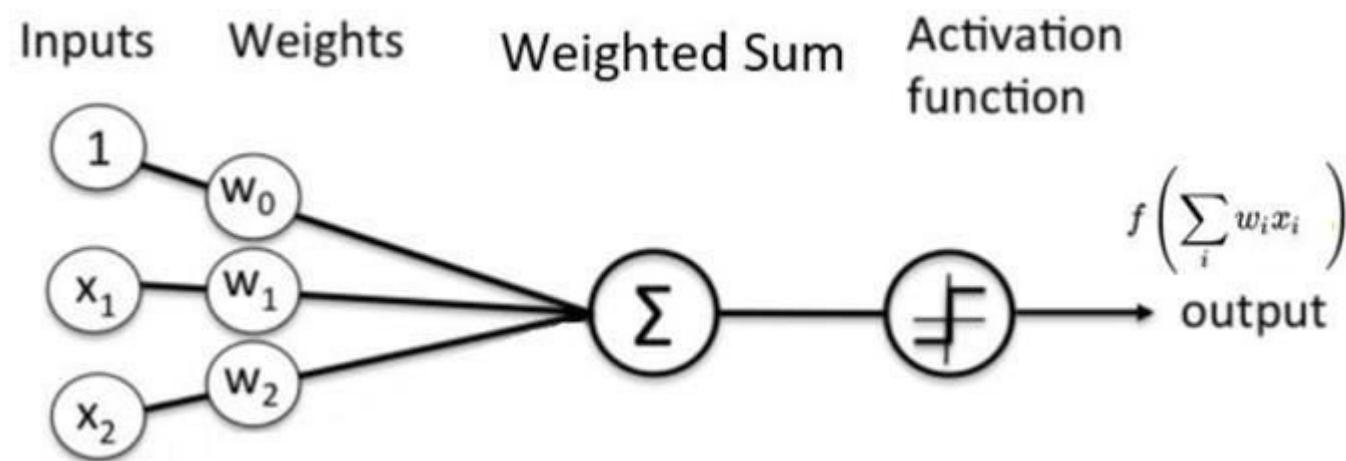


Text: HKUST, figures: Andrej Karpathy

Inspiration: Neuron Cells

- Neurons
 - Accept information from multiple inputs.
 - Transmit information to other neurons.
- Multiply inputs by weights along edges
- Apply some function to set the set of inputs at each node
- If output of function over threshold, neuron “**fires**”

Inspiration: Neuron Cells

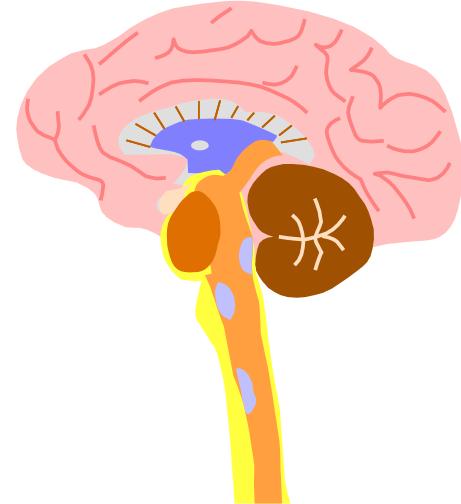


Human versus Computers



Human Brain

- The Brain
 - Pattern Recognition
 - Association
 - Complexity
 - Noise Tolerance

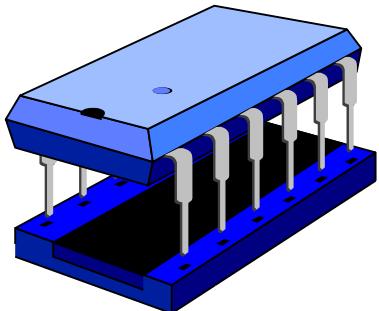


Computing Machines

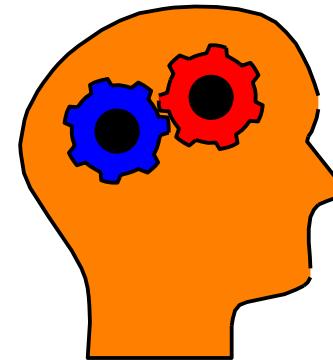


- The Machine
 - Calculation
 - Precision
 - Logic

The Contrast in Architecture



- A single processing unit;
 - Tens of millions of operations per second
 - Absolute arithmetic precision
- The brain uses many slow unreliable processors acting in parallel



Linear Algebra



Linear Algebra

- Linear algebra is the branch of mathematics concerning linear equations such as

$$a_1 x_1 + \cdots + a_n x_n = b$$

and their representations through matrices and vector spaces.

- It allows modeling many natural phenomena, and efficiently computing with such models.

Linear Algebra Topics

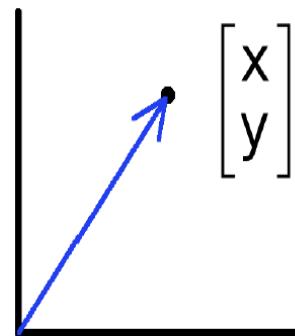
- Vector
- Matrix
- Matrix operations
- Identity matrix
- Diagonal matrix
- Transpose of a matrix
- Symmetric matrix
- Norm

Scalar, Vectors and Matrices

Scalar	Vector	Matrix	Tensor
1	$\begin{bmatrix} 1 \\ 2 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 & [3 & 2] \\ 1 & 7 & [5 & 4] \end{bmatrix}$

Scalar, Vectors and Matrices

- **Scalar:** Is a single number that only has magnitude.
Example: Temperature
- **Vector:** Is an array of numbers. It can represent an offset in 2D or 3D space. Points are examples of vectors from origin.

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_n \end{bmatrix}$$
A 2D Cartesian coordinate system is shown with a vertical y-axis and a horizontal x-axis. A blue vector arrow originates from the origin (0,0) and points into the first quadrant. The tip of the vector is marked with a small black dot. To the right of the vector, its components are given as a column vector: $\begin{bmatrix} x \\ y \end{bmatrix}$. The letter 'x' is in blue, matching the vector, while 'y' is in black.

Scalar, Vectors and Matrices

Vectors and Matrices

- Vectors and matrices are just collections of ordered numbers that represent something:
 - movements in space,
 - scaling factors,
 - Word counts,
 - movie ratings,
 - pixel brightness's etc.

Scalar, Vectors and Matrices

- **Matrix:** Is a two-dimensional array of numbers with size m (rows) by n (columns). A matrix is called square matrix if m=n.

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & & & & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

Matrix Operations

- **Matrix Addition:** Works for matrix of equal dimensions and with scalar

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} a+1 & b+2 \\ c+3 & d+4 \end{bmatrix}$$

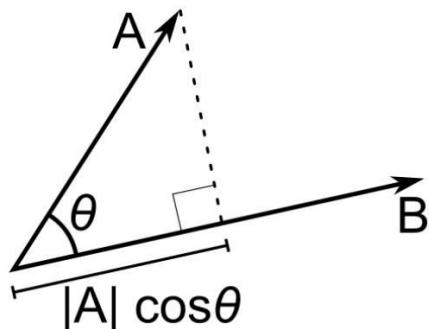
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} + 7 = \begin{bmatrix} a+7 & b+7 \\ c+7 & d+7 \end{bmatrix}$$

Matrix Operations

- **Scaling:** Multiply each element with the scalar

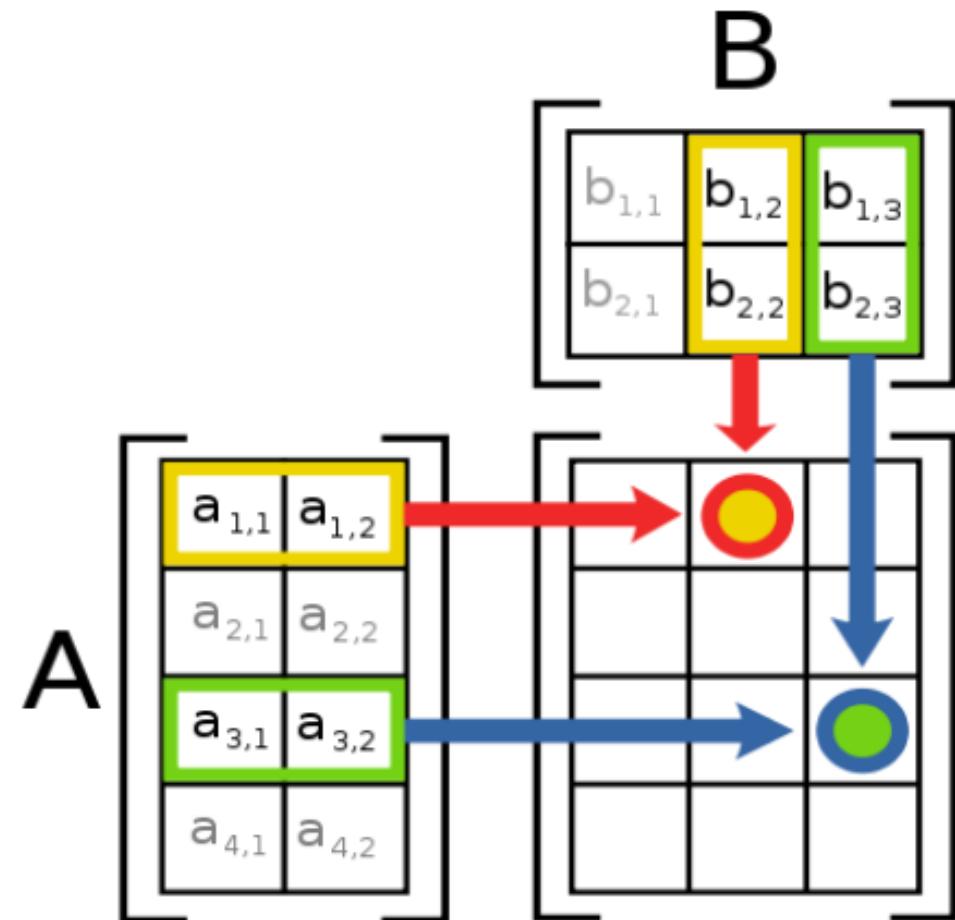
$$\begin{bmatrix} a & b \\ c & d \end{bmatrix} \times 3 = \begin{bmatrix} 3a & 3b \\ 3c & 3d \end{bmatrix}$$

- **Inner product (dot product) of vectors:** If B is a unit vector, then $A \cdot B$ gives the length of A which lies in the direction of B (projection)



Matrix Operations

- **Multiplication:** Each entry in the result is (that row of A) dot product with (that column of B).



Credit: Professor Fei Fei Li

Matrix Operations

- **Multiplication example:**

Each entry of the matrix product is made by taking the dot product of the corresponding row in the left matrix, with the corresponding column in the right one.

$$\begin{matrix} A & \times & B \\ \downarrow & & \searrow \\ \begin{bmatrix} 0 & 2 \\ 4 & 6 \end{bmatrix} & & \begin{bmatrix} & & 14 \\ & & \end{bmatrix} \end{matrix}$$

$$0 \cdot 3 + 2 \cdot 7 = 14$$

Matrix Operations

- **Transpose:** Transpose flip the matrix, so row and column swap positions; i.e., row 1 becomes column 1.

$$\text{Transpose } A = A^T$$

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \end{bmatrix}^T = \begin{bmatrix} 0 & 2 & 4 \\ 1 & 3 & 5 \end{bmatrix}$$

Matrix Operations

- **Identity matrix I**
 - Square matrix; 1's along diagonal, 0's elsewhere
 - $I \cdot [\text{Another matrix}] = [\text{This matrix}]$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Matrix Operations

- **Diagonal matrix**
 - Square matrix with numbers along diagonal, 0's elsewhere
 - A diagonal . [Another matrix] = Scales the rows of that matrix

$$\begin{bmatrix} 3 & 0 & 0 \\ 0 & 7 & 0 \\ 0 & 0 & 2.5 \end{bmatrix}$$

Matrix Operations

- **Symmetric matrix**
 - Special matrix with $A^T = A$

$$\begin{bmatrix} 1 & 2 & 5 \\ 2 & 1 & 7 \\ 5 & 7 & 1 \end{bmatrix}$$

Matrix Operation Properties

- **Matrix Addition:** Cumulative and associative

$$A + B = B + A$$

$$A + (B + C) = (A + B) + C$$

- **Matrix Multiplication:** Associative and distributive; not cumulative

$$A(B * C) = (A * B)C$$

$$A(B + C) = A * B + A * C$$

$$A * B \neq B * A$$

Matrix Operations [Norms]

- **L1 Norm:** Absolute-value norm

$$\|\mathbf{x}\|_1 := \sum_{i=1}^n |x_i|$$

- **L2 Norm:** Euclidean norm

$$\|\mathbf{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}$$

- **L3 Norm:** p-norm

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^n |x_i|^p \right)^{1/p}$$



Any questions so far?
Any comments?

Basic Statistics

Statistical Analysis

- Mean
- Median
- Mode
- Stdv
- Range
- IQR
- Skewness
- Kurtosis

Mean, median, and mode

- Mean, median, and mode are main **measures of central tendency** in a distribution.
- Each of these measures try to **summarize** a dataset with a **single number** to represent a typical or **center data point** from the numerical data set.
- Mean good for larger sample size without outliers (symmetric dataset)
- Median is good for dataset with extreme values (skewed dataset)

Mean, median, and mode - continues

Mean: The "average" number; found by adding all data points and dividing by the number of data points.

Example: The mean of 4, 1, and 7 is $(4+1+7)/3 = 12/3 = 4$

Median: The middle number; found by ordering all data points and picking out the one in the middle (or if there are two middle numbers, taking the mean of those two numbers).

Example: The median of 4, 1, and 7 is 4 because when the numbers are put in order (1, 4, 7), the number 4 is in the middle.

Mean, median, and mode - continues

Mode: The most frequent number—that is, the number that occurs the highest number of times.

Example: The mode of { 4, 2, 4, 3, 2, 2} is 2 because it occurs three times, which is more than any other number.

Standard Deviation

- The **Standard Deviation** is a measure of variation or dispersion of a dataset.
- It is a number used to **tell** how measurements for a group are spread out from the average (mean), or expected value.

$$s = \sqrt{\frac{1}{N-1} \sum_{i=1}^N (x_i - \bar{x})^2},$$

where $\{x_1, x_2, \dots, x_N\}$ are the observed values of the sample items, \bar{x} is the mean value of these observations, and N is the number of observations in the sample.

Standard Deviation - continues

The marks of a class of eight students: 2, 4, 4, 4, 5, 5, 7, 9.

These eight data points have the mean (average) of 5:

$$\mu = \frac{2 + 4 + 4 + 4 + 5 + 5 + 7 + 9}{8} = 5.$$

First, calculate the deviations of each data point from the mean, and **square** the result of each:

$$\begin{array}{ll} (2 - 5)^2 = (-3)^2 = 9 & (5 - 5)^2 = 0^2 = 0 \\ (4 - 5)^2 = (-1)^2 = 1 & (5 - 5)^2 = 0^2 = 0 \\ (4 - 5)^2 = (-1)^2 = 1 & (7 - 5)^2 = 2^2 = 4 \\ (4 - 5)^2 = (-1)^2 = 1 & (9 - 5)^2 = 4^2 = 16. \end{array}$$

The **variance** is the mean of these values:

$$\sigma^2 = \frac{9 + 1 + 1 + 1 + 0 + 0 + 4 + 16}{8} = 4.$$

and the **population** standard deviation is equal to the square root of the variance:

$$\sigma = \sqrt{4} = 2.$$

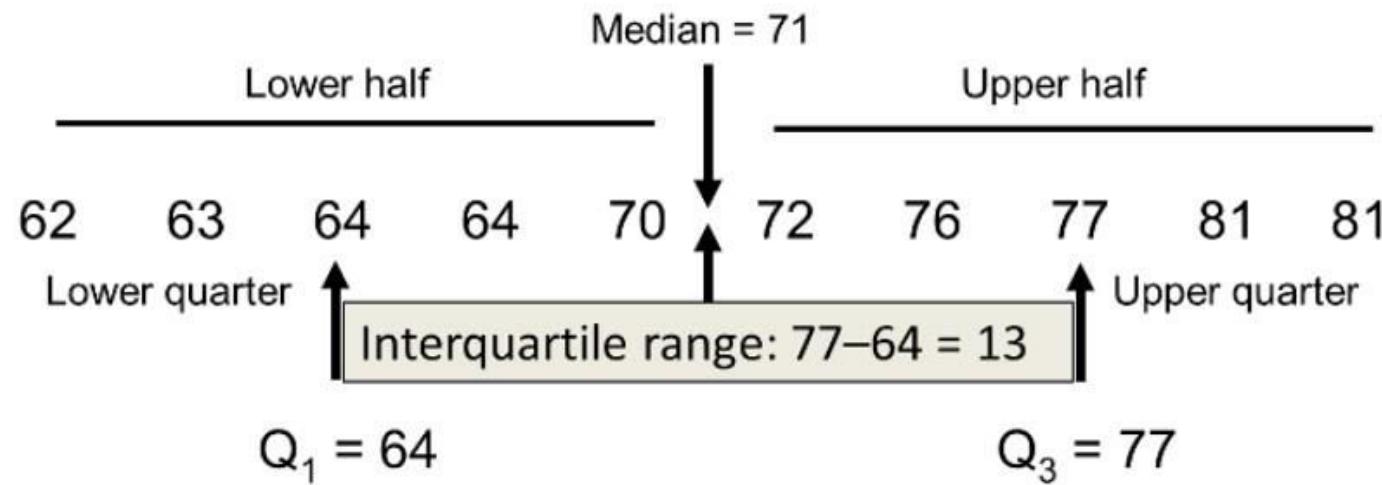
InterQuartile Range (IQR)

Interquartile Range (IQR): It is a measure of statistical dispersion, being equal to the difference between 75th and 25th percentiles, or between upper and lower quartiles.

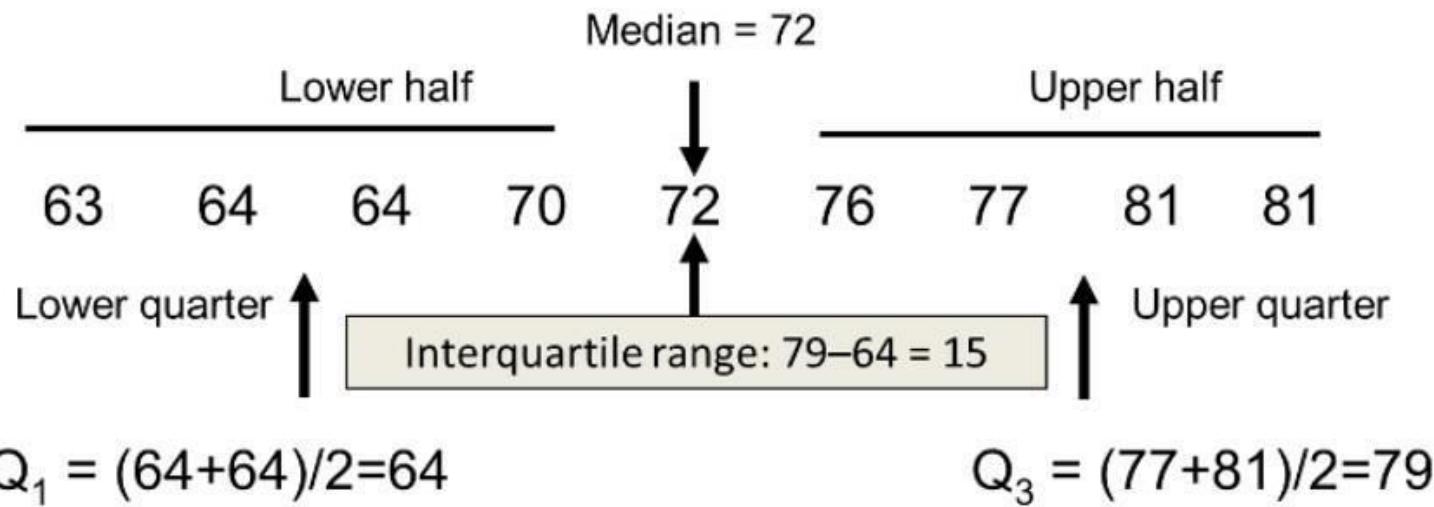
- Also called the **midspread** or **middle 50%**, or technically **H-spread**

$$\text{Interquartile Range} = Q_3 - Q_1$$

IQR- Even Sample Size

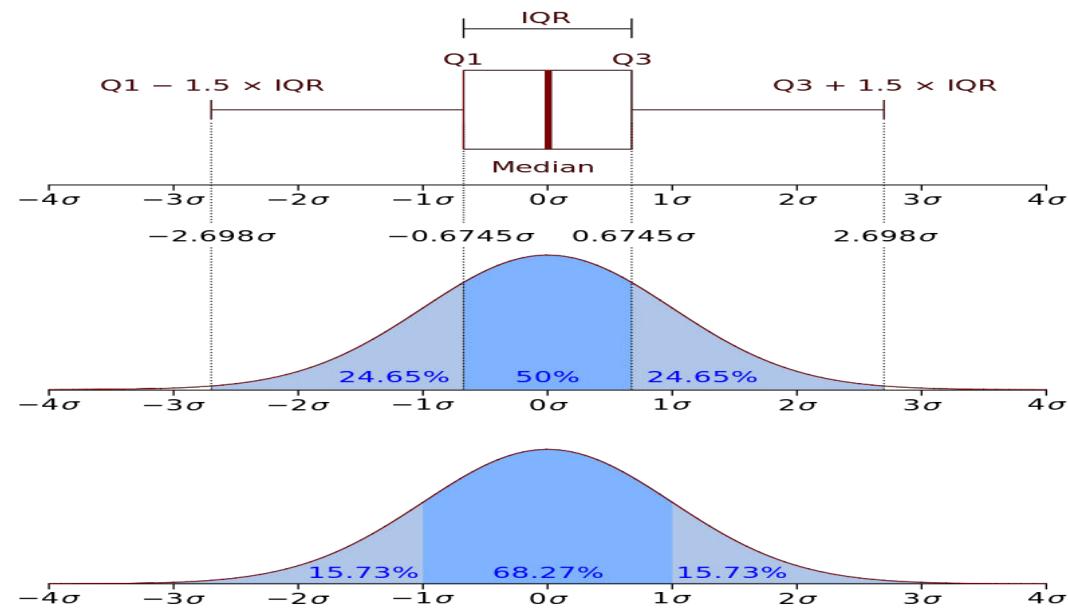


IQR- Odd Sample Size

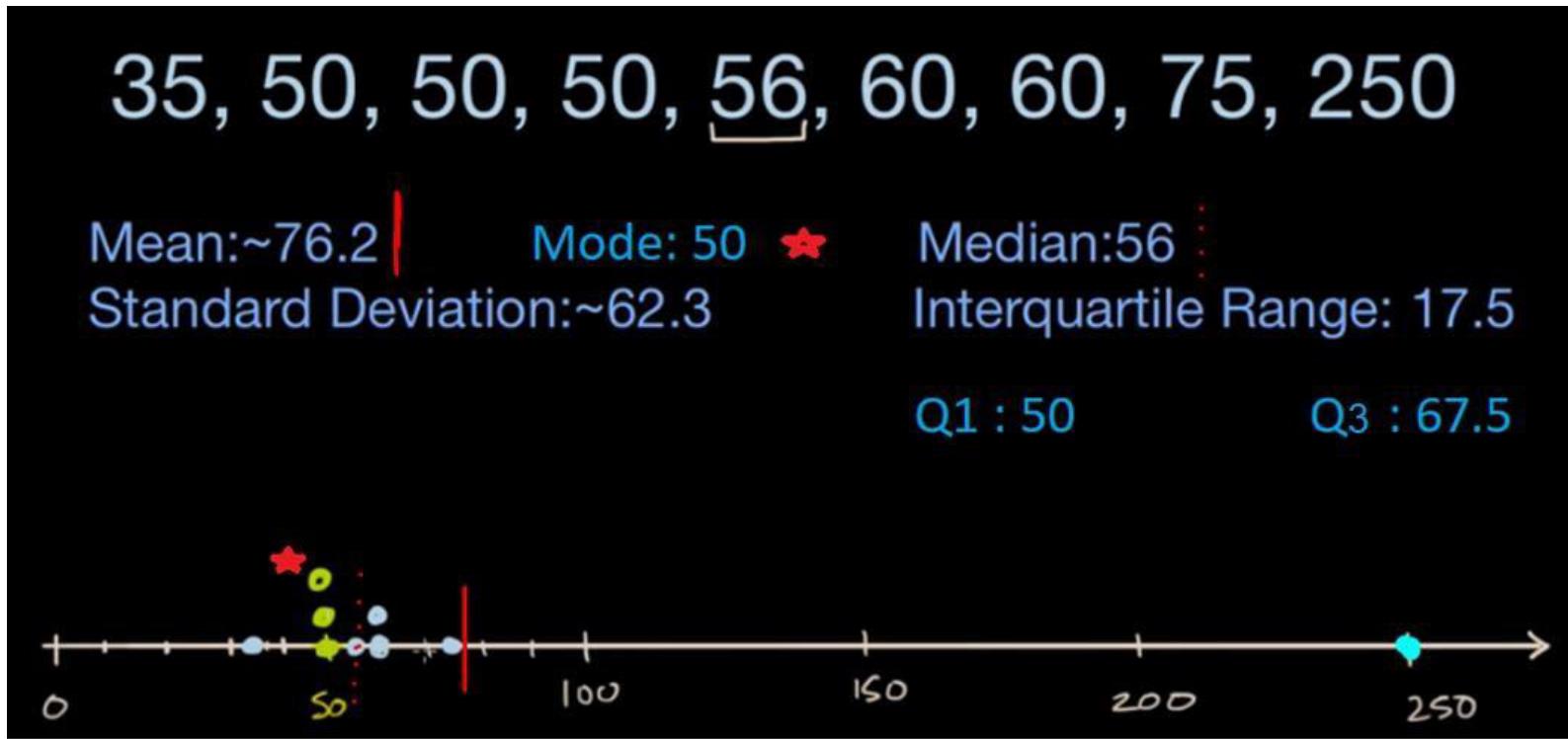


IQR - continues

- Boxplot (with an interquartile range) and a probability density function (pdf) of a Normal $N(0,\sigma^2)$ Population



Summary - Statistical Analysis

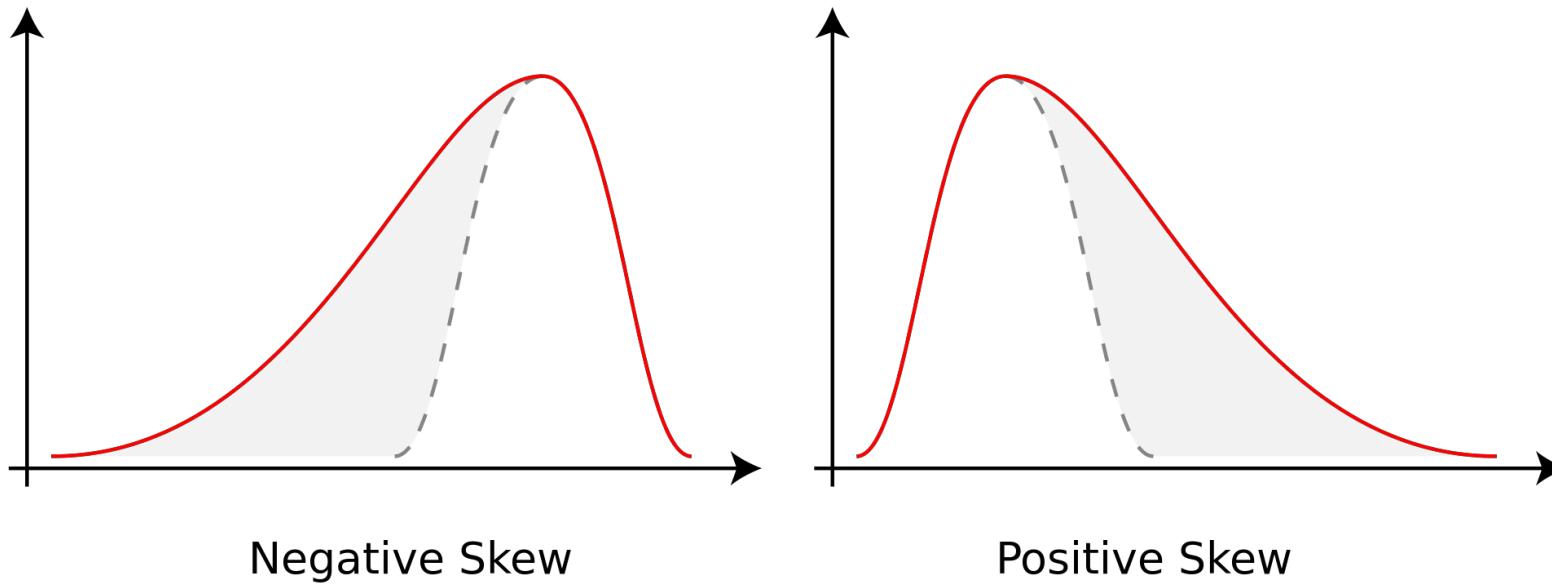


Skewness

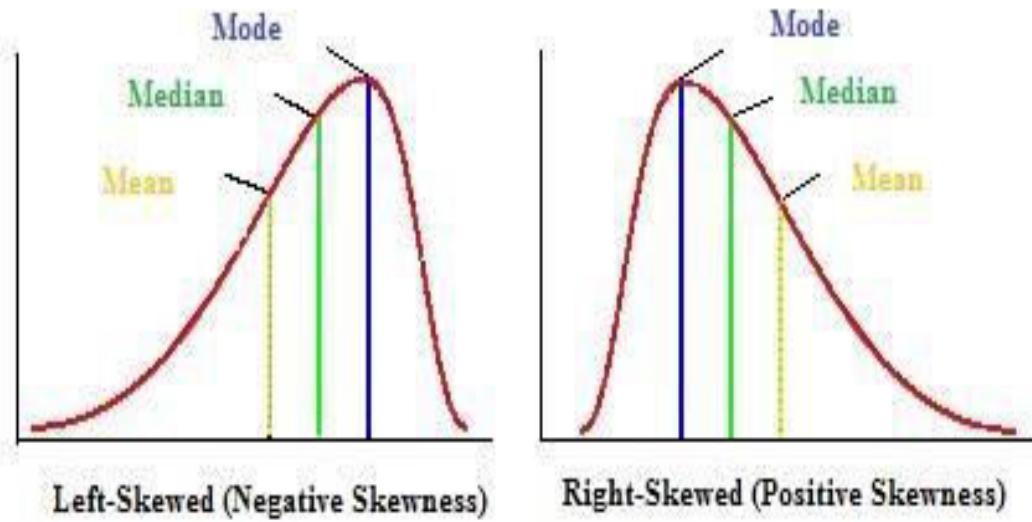
✓ The **skewness**, that reflects the asymmetry of a distribution

- Skewness measures the **asymmetry** of a distribution. A distribution is called asymmetric when one tail is longer than the other.
- The direction of skewness is given by the sign.
- The coefficient compares the sample distribution with a normal distribution. The larger the value, the larger the distribution differs from a **normal distribution**.
 - If the skewness is positive, then the distribution is **skewed to the right** while a negative skewness implies a distribution **skewed to the left**.
 - A zero skewness suggests a **perfectly symmetric** distribution.

Skewness



Skewness



Pearson's Coefficient of Skewness

1. Pearson's Coefficient of Skewness #1 uses the mode. The formula is:

$$Sk_1 = \frac{\bar{X} - Mo}{s}$$

Where \bar{X} = the mean, Mo = the mode and s = the standard deviation for the sample.

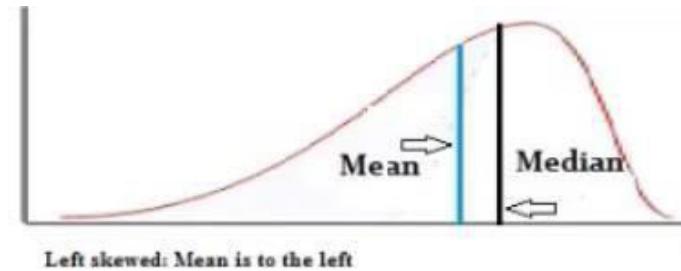
See: Pearson Mode Skewness.

2. Pearson's Coefficient of Skewness #2 uses the median. The formula is:

$$Sk_2 = \frac{3(\bar{X} - Md)}{s}$$

Where \bar{X} = the mean, Mo = the mode and s = the standard deviation for the sample.

It is generally used when you don't know the mode.



$$Sk_1 = \frac{\bar{X} - Mo}{s}$$

$$Sk_2 = \frac{3(\bar{X} - Md)}{s}$$

Mean = 70.5.
Median = 80.
Mode = 85.
Standard deviation = 19.33.

- **Pearson's Coefficient of Skewness #1 (Mode):**

Step 1: Subtract the mode from the mean: $70.5 - 85 = -14.5$.

Step 2: Divide by the standard deviation: $-14.5 / 19.33 = -0.75$.

- **Pearson's Coefficient of Skewness #2 (Median):**

Step 1: Subtract the median from the mean: $70.5 - 80 = -9.5$.

Step 2: Multiply Step 1 by 3: $-9.5(3) = -28.5$

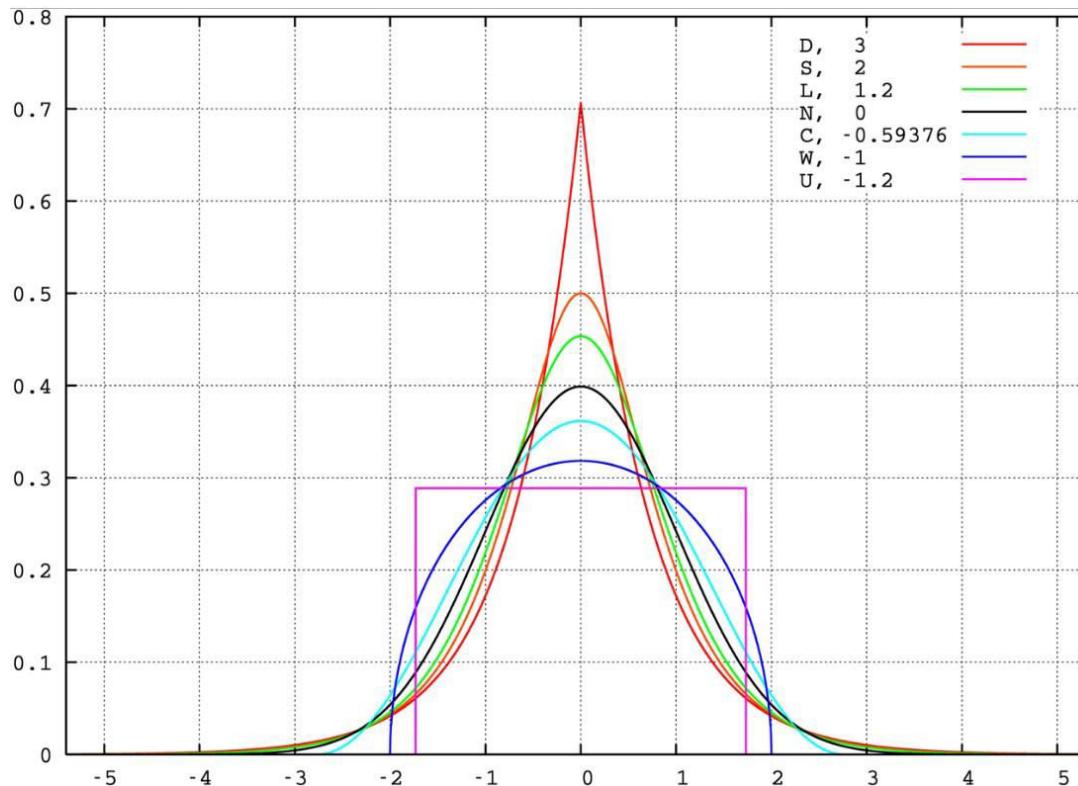
Step 2: Divide by the standard deviation: $-28.5 / 19.33 = -1.47$.

Kurtosis

- ✓ The **kurtosis**, that reflects the characteristics of the tails of a distribution.
 - Kurtosis provides information on the tails (the extremes, or outliers) of a distribution. When interpreting kurtosis, the normal distribution is used a reference.
 - A **positive kurtosis** implies a distribution with more extreme possible data values (outliers) than a normal distribution thus fatter tails (*Leptokurtic distributions*).
 - A **negative kurtosis** implies a distribution with less extreme possible data values than a normal distribution thus thinner tails (*Platykurtic distributions*).
 - Finally, distributions with **zero kurtosis** have roughly the same outlier character as a normal distribution (*Mesokurtic distributions*).

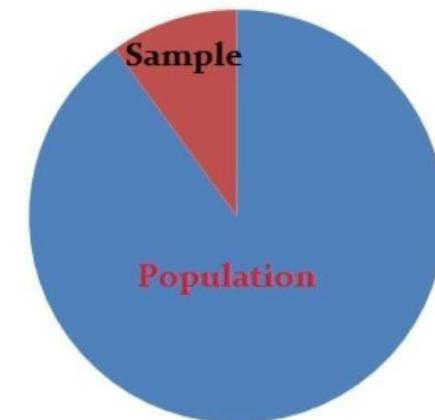
Kurtosis Example

Probability density
functions for selected
distributions with
mean 0,
variance 1 and
different excess
kurtosis



Inferential Statistics

- ✓ **Inferential statistics** makes inferences and predictions about a population based on a sample of data taken from the population in question.
- ✓ This infers properties of a population by testing **hypotheses** and deriving **estimates**.
 - Descriptive statistics describes data (for example, a chart or graph)
 - Inferential statistics allows you to make predictions (“inferences”) from that data.



Inferential Statistics – Main Areas

- ✓ **Estimating parameters:** This means taking a statistic from your sample data (*for example the sample mean*) and using it to say something about a population parameter (*i.e. the population mean*).
- ✓ **Hypothesis tests:** This is where you can use sample data to answer research questions.
For example,
 - Interested in knowing if a new cancer drug is effective. Or
 - If breakfast helps children perform better in schools.

Inferential Statistics – continues

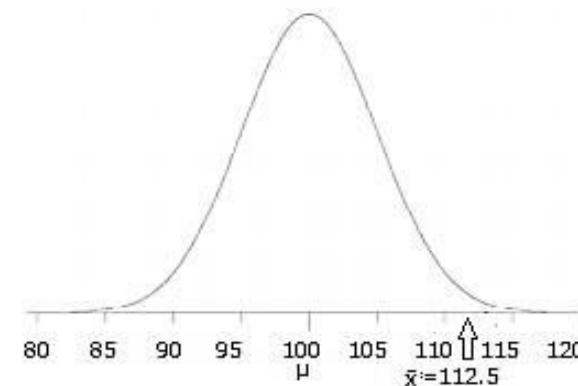
- ✓ **Regression and Correlation** analysis are statistical techniques used extensively to examine causal relationships between variables.
 - Measure the degree of relationship between two or more variables in two different but related ways.

- ✓ **Regression Analysis:** a single dependent variable, Y , is considered to be a function of one or more independent variables, X_1, X_2 , and so on. The values of both the dependent and independent variables are assumed as being ascertained in an error-free random manner.

Inferential Statistics – continues

- **Inferential statistics** you take that sample data from a small number of people and try to determine if the data can predict whether the drug will work for everyone (i.e. the population).
 - Calculating a z-score is a way to do this

Z-score is the number of standard deviations from the mean a data point is.



- It's way to show where your data would lie in a normal distribution to post-hoc

Probability

✓ **Probability** is a measure quantifying the likelihood that events will occur (simply *how likely something is to happen*).

- The higher the probability of an event, the more likely it is that the event will occur.
- Probability quantifies as a number between 0 and 1; 0 indicates impossibility and 1 indicates certainty.

Example: Probability of flipping a unbiased coin:

- There are two possible outcomes—head or tail; each with a probability of 0.5

Probability - Example

Probability
Experiment
Flip
 $P(H) = ?$

$\frac{1}{\# \text{ of equally likely possibilities}} \cdot \frac{\# \text{ of possibilities that meet by condition}}{\# \text{ of possibilities that meet by condition}}$

$$P(H) = \frac{1}{2} = \boxed{50\%}$$

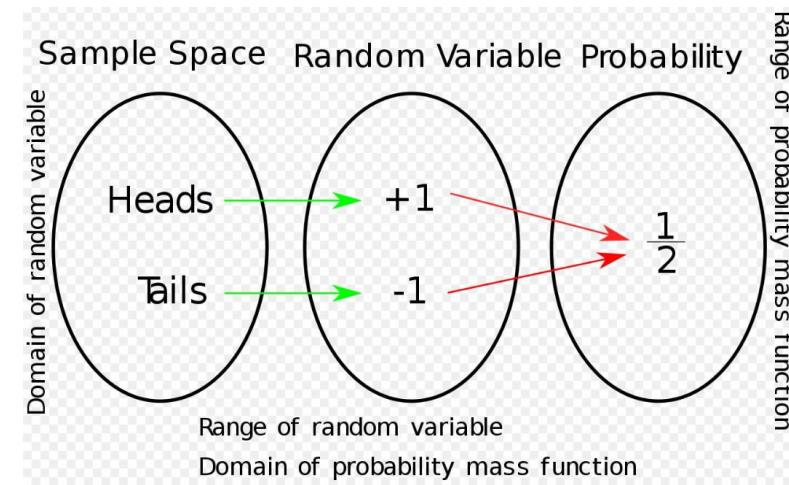
$$P(1) = \frac{1}{6}$$
$$P(1 \text{ or } 6) = \frac{2}{6} = \frac{1}{3}$$
$$P(2 \text{ and } 3) = \frac{0}{6}$$


1	-
2	-
3	-
4	-
5	-
6	-

Probability - Terms

✓ **Random variable** (random quantity, aleatory variable, or stochastic variable) is described informally as a variable whose values depend on outcomes of a random phenomenon.

- Examples of random phenomena can include the results of an experiment or survey.
- Flipping a unbiased coin experiment.



Probability Distribution

✓ **Probability distribution** is a table or an equation that links each outcome of a statistical experiment with its probability of occurrence.

Example: A desired outcome of the experiment might be the number of heads that we see in two coin flips:

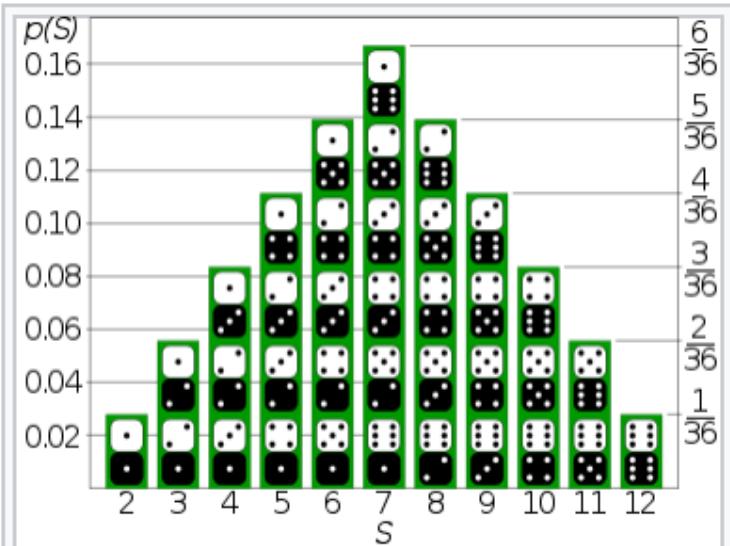
Number of heads	Probability
0	0.25
1	0.50
2	0.25

Probability Distribution

↳ table
graph
chart

① possible values of variable (X)
② probabilities for each value

Probability Distribution – Two dice flips



The probability mass function (pmf) $p(S)$ specifies the probability distribution for the sum S of counts from two dice. For example, the figure shows that $p(11) = 2/36 = 1/18$.

Σ = Sum of numbers

	1	2	3	4	5	6
1	2	3	4	5	6	7
2	3	4	5	6	7	8
3	4	5	6	7	8	9
4	5	6	7	8	9	10
5	6	7	8	9	10	11
6	7	8	9	10	11	12

1	
2	1/36
3	2/36
4	3/36
5	4/36
6	5/36
7	6/36
8	5/36
9	4/36
10	3/36
11	2/36
12	1/36

Roll two dice
 X = sum of numbers

Probability Distribution - continues

✓ **Probability distribution** are generally divided into two classes:

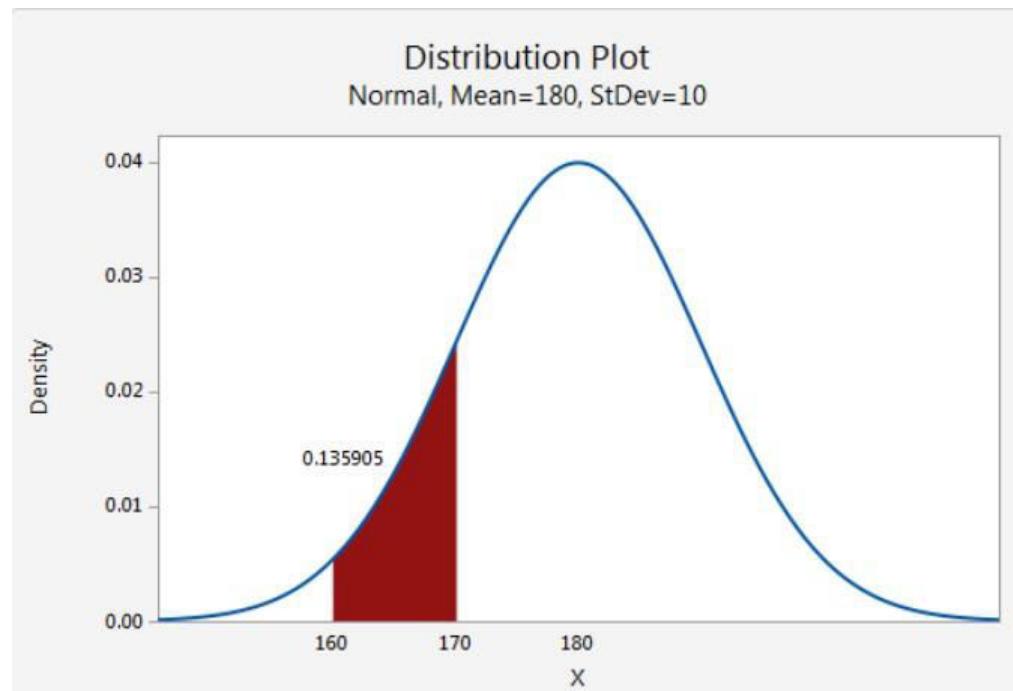
- Continuous probability distribution
- Discrete probability distribution

Continuous probability distribution

- **Continuous distribution** describes the probabilities of the possible values of a continuous random variable.
- **Continuous random variable** is a random variable with a set of possible values (known as the range) that is infinite and uncountable.
- Probabilities of continuous random variables (X) are defined as the area under the curve of its PDF.
 - Thus, only ranges of values can have a nonzero probability.
 - The probability that a continuous random variable equals some value is always zero.

Continuous probability distribution

- **Distribution plot of the weight of adult males**

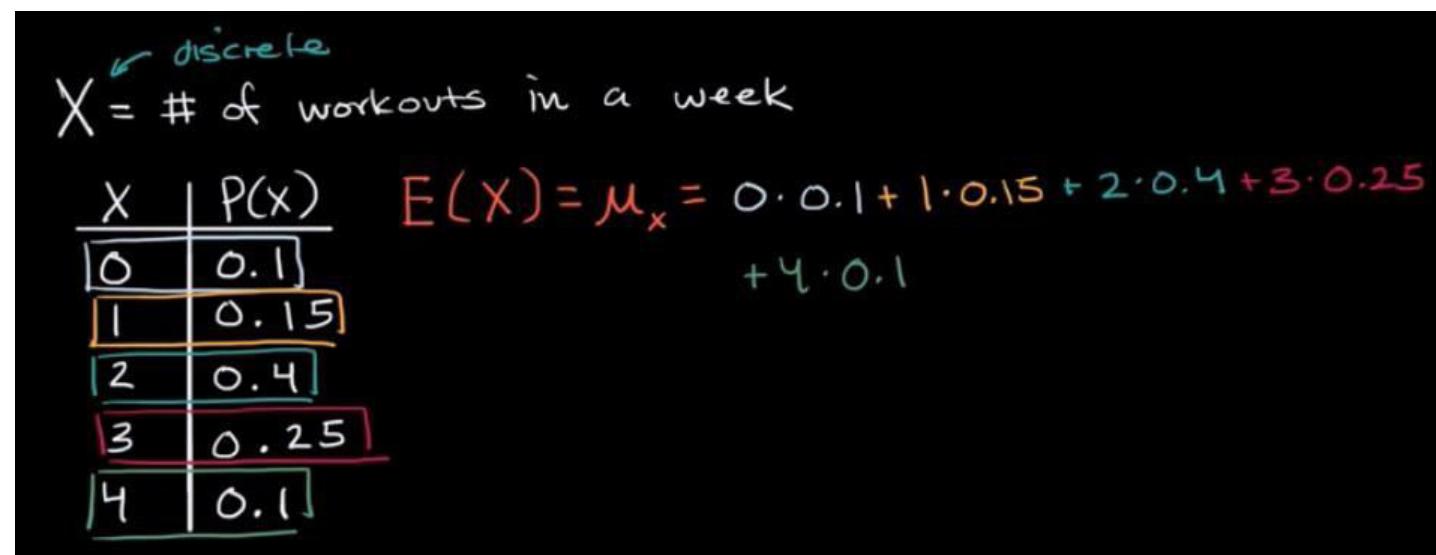


Discrete probability distribution

- **Discrete distribution** describes the probability of occurrence of each value of a discrete random variable.
- A **discrete random** variable is a random variable that has countable values, such as a list of non-negative integers.
- With a discrete probability distribution, each possible value of the discrete random variable can be associated with a non-zero probability.
- Discrete probability distribution is often presented in tabular form.

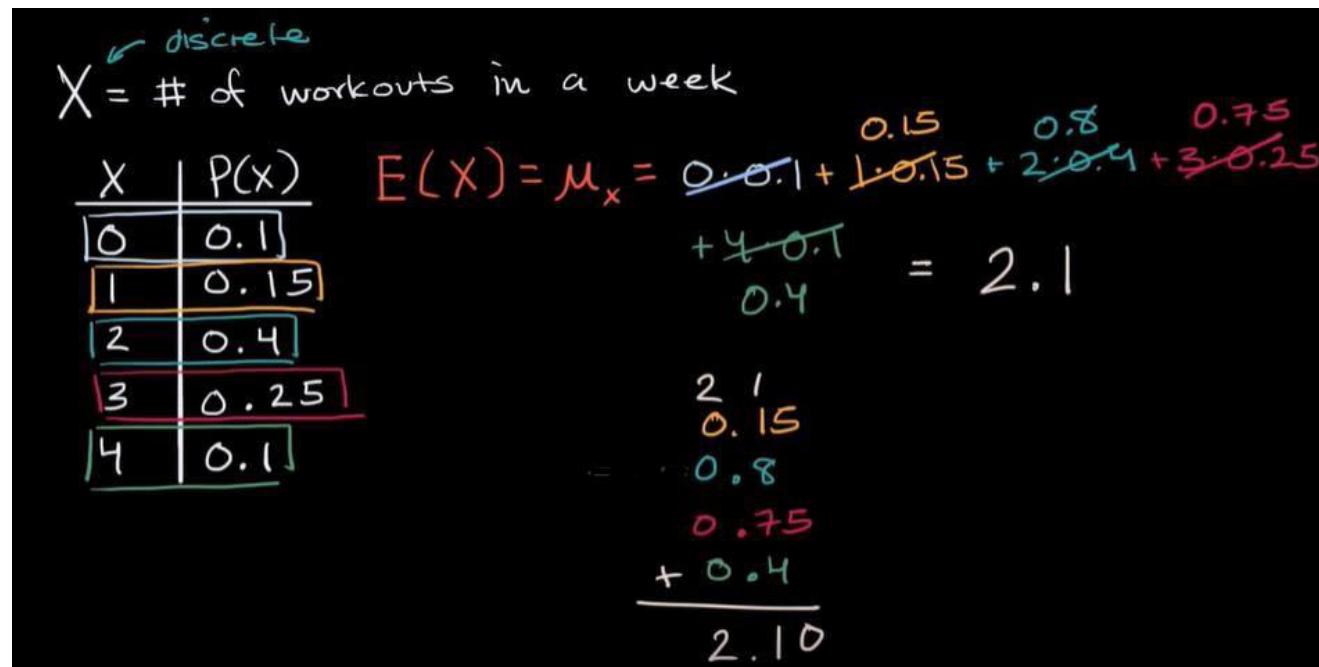
Mean (expected value)

Mean (expected value) of a discrete random variable



Mean (expected value) - continues

Mean (expected value) of a discrete random can be non-integer.



Variance and Standard deviation

Variance and standard deviation of a discrete random variable

$X \leftarrow \text{discrete}$
 $X = \# \text{ of workouts in a week}$

X	$P(X)$
0	0.1
1	0.15
2	0.4
3	0.25
4	0.1

$$E(X) = \mu_x = \frac{0 \cdot 0.1 + 1 \cdot 0.15 + 2 \cdot 0.4 + 3 \cdot 0.25 + 4 \cdot 0.1}{0.4} = 2.1$$
$$\text{Var}(X) = (0 - 2.1)^2 \cdot 0.1 + (1 - 2.1)^2 \cdot 0.15 + (2 - 2.1)^2 \cdot 0.4 + (3 - 2.1)^2 \cdot 0.25 + (4 - 2.1)^2 \cdot 0.1 = 1.19$$
$$\sigma_x = \sqrt{1.19} \approx 1.09$$

Concept of correlation

Correlation

- Correlation describe the linear relationship between two continuous variables. It indicates the trend of change in one variable with the change in values of other variable.
- Importance:
 - Correlation can help in predicting one quantity from another
 - It can indicate the presence of a causal relationship
 - Correlation is used as a foundation for many other modeling techniques

Concept of correlation

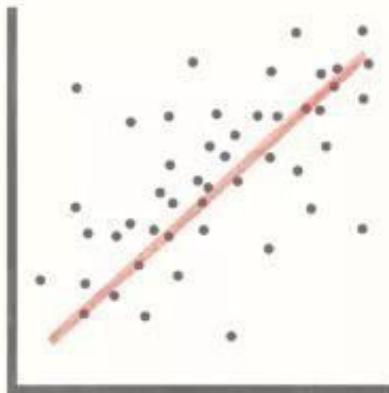
Correlation types

- **Positive Correlation:** as one variable increases so does the other.
 - Height and shoe size are an example; as one's height increases so does the shoe size.
- **Negative Correlation:** as one variable increases, the other decreases.
 - Inflation rate and purchase power are negatively correlated;
Higher rate of inflation means expensive commodities, thus lower purchase power.

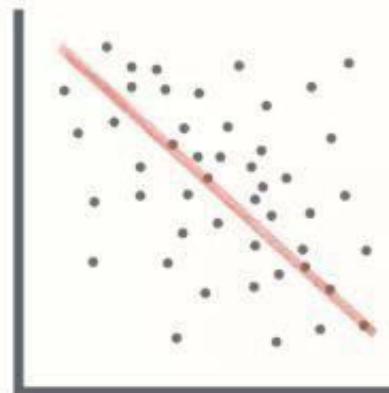
Concept of correlation

Correlation types

- **No Correlation:** there is no apparent relationship between the variables.
 - Inflation and shoe size apparently have no relationship between them.



Positive Correlation



Negative Correlation



No Correlation

Random Sampling

Random sampling is a way of selecting a sample of observations from a population in order to make inferences about the population.

- Each individual in the sample has been chosen entirely at random (by chance), and
- All have the same probability to be chosen.
- Example: Polls from voters to predict election outcome.



Random Sampling

Random sampling types

- **Simple random sampling:** It is subset of individual randomly chosen from a larger set called population.
- **Stratified random sampling:** Performed by breaking a population into key subgroups and obtaining a simple random sample from each group.
 - To ensure participation from male-female and both adult and child; the four sub-group can be adult-female, girls, boys, adult-males.
 - Then Simple random sampling can be used to take equal number of individuals from each group

Concept of correlation

Correlation types

- **Positive Correlation:** as one variable increases so does the other.
 - Height and shoe size are an example; as one's height increases so does the shoe size.
- **Negative Correlation:** as one variable increases, the other decreases.
 - Inflation rate and purchase power are negatively correlated;
Higher rate of inflation means expensive commodities, thus lower purchase power.

Installing Python and Pycharm

- <https://www.python.org/downloads/>
- <https://www.jetbrains.com/pycharm/download/#section=windows>
(download the free version)

Installing Python and Pycharm

- Install with pip <https://packaging.python.org/installing/>
 - python -m pip install -U pip setuptools(installing pip)
 - pip install -U pip (updating pip)
 - Pip install “library_name”
- Install with easy_install
 - easy_install“library_name”
- Install with wheel (.whlfiles)
 - Go to binaries for python packages <http://www.lfd.uci.edu/~gohlke/pythonlibs/>
 - Pip install wheel
 - pip install PATH+“library_name”.whl

Further Readings

- [Data-Mining Boosts Netflix's Subscriber Base](#)