Week 14

Machine Learning and Big Data - DATA622

Fall 2023

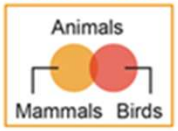
CUNY School of Professional Studies

The 5 tribes of machines learning

CUNY School of Professional Studies

5 Tribes of Machine Learning

Symbolists



Use symbols, rules, and logic to represent knowledge and draw logical inference

Favored algorithm
Rules and decision trees

Bayesians

Assess the likelihood of occurrence for probabilistic inference

Favored algorithm
Naïve Bayes or Markov

Connectionists

Recognize and generalize patterns dynamically with matrices of probabilistic, weighted neurons

Favored algorithm
Neural network

Evolutionaries

Generate variations and then assess the fitness of each for a given purpose

Favored algorithm
Genetic programs

Analogizers

Optimize a function in light of constraints ("going as high as you can while staying on the road")

Favored algorithm
Support vectors

How it started: Hubel & Wiesel

Mapping of the Visual cortex hierarchy

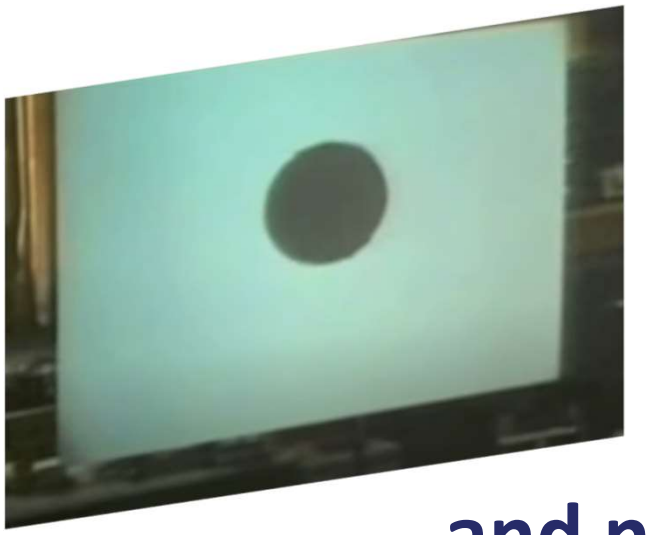
Hubel & Wiesel

David Hubel and Torsten Wiesel recorded electrical activity from individual neurons in the brains of cats. They won the Nobel Prize for Physiology & Medicine in 1981.

5

Visual cortex experiment

The two jabbed a small microelectrode into a brain cell in the visual cortex at the back of the brain of an anesthetized cat, and showed the cat various shapes.



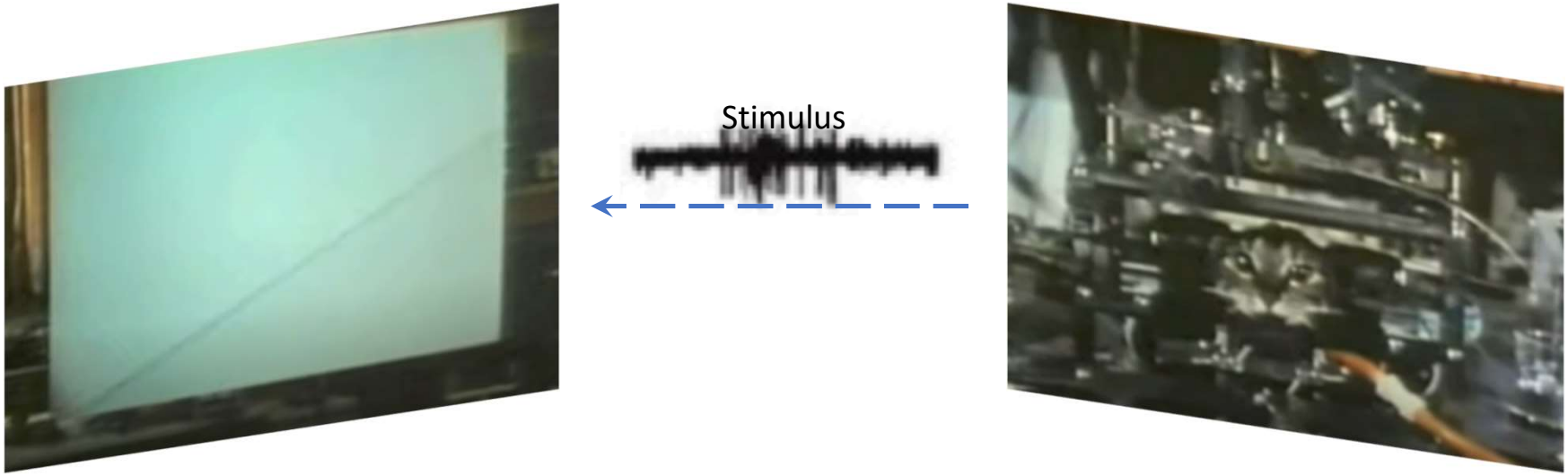
Stimulus

…and nothing happened

6

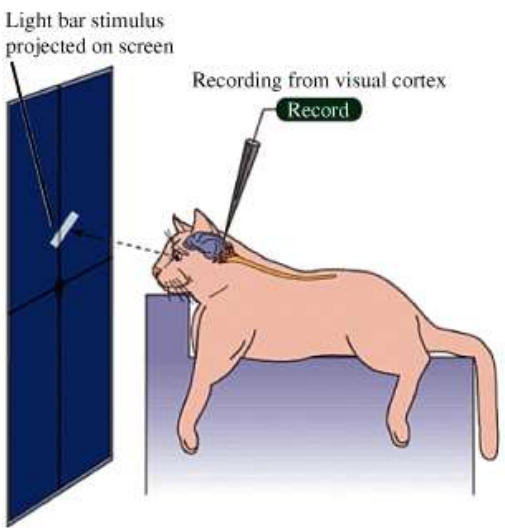
Visual cortex experiment (cont'd)

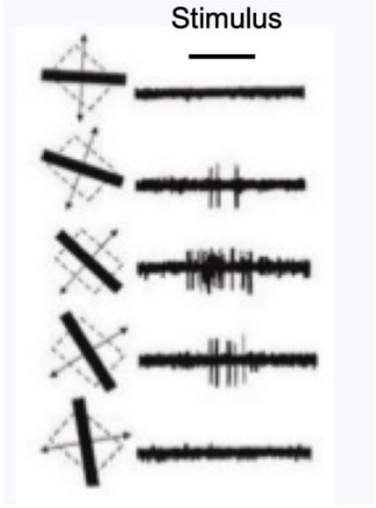
…until they changed slides in a certain way



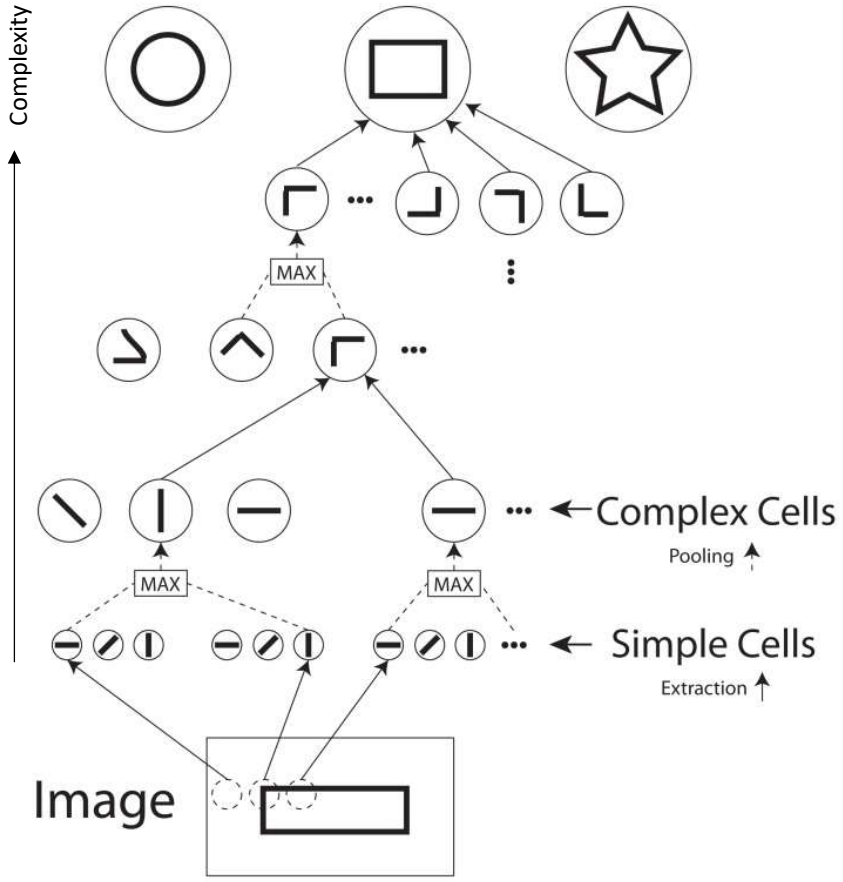
Neuron behavior (simple cells)

Neurons showed stimulus based on orientation (simple cells in visual cortex)





Cell hierarchy of the visual cortex

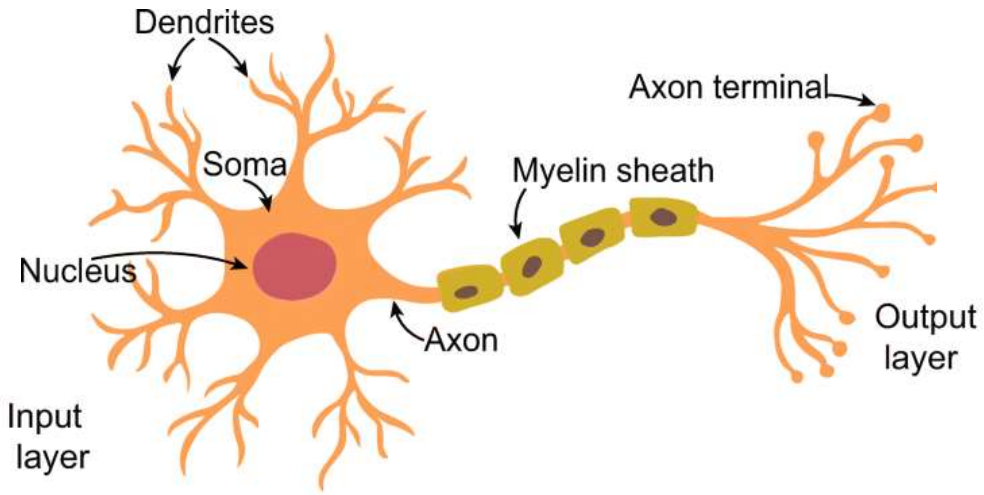


- Simple Cells: respond to points of light or bars of light in a particular orientation
- Complex cells: respond to bars of light in a particular orientation moving in a specific direction
- Hypercomplex Cells: respond to bars of light in a particular orientation, moving in a specific direction, and of a specific line length.

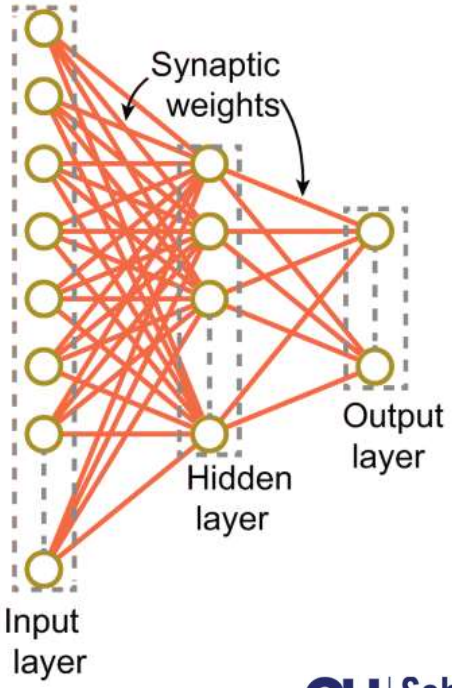
Neural networks

Neural networks were inspired by neural networks in the brain.

Neurons in the brain



Neural networks

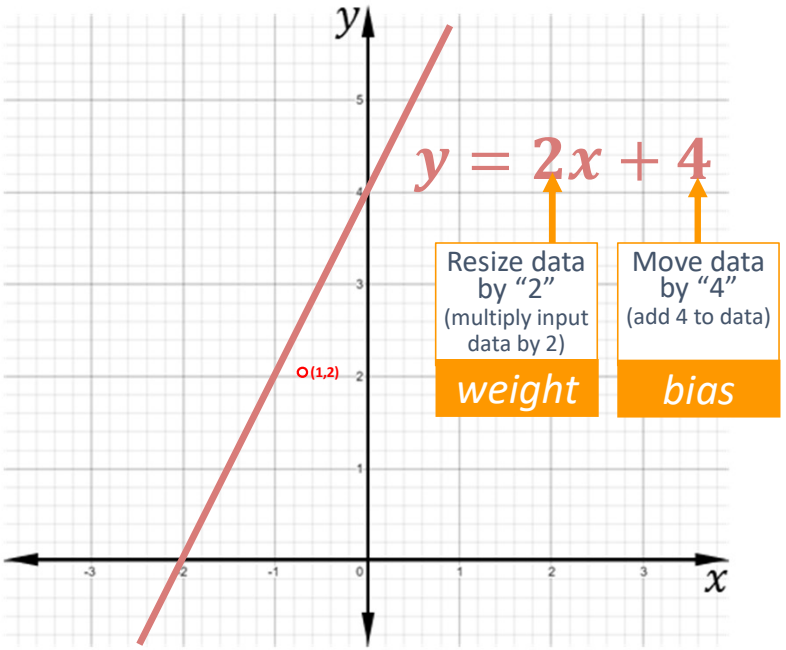


Transformation

Linear & non-linear transformation

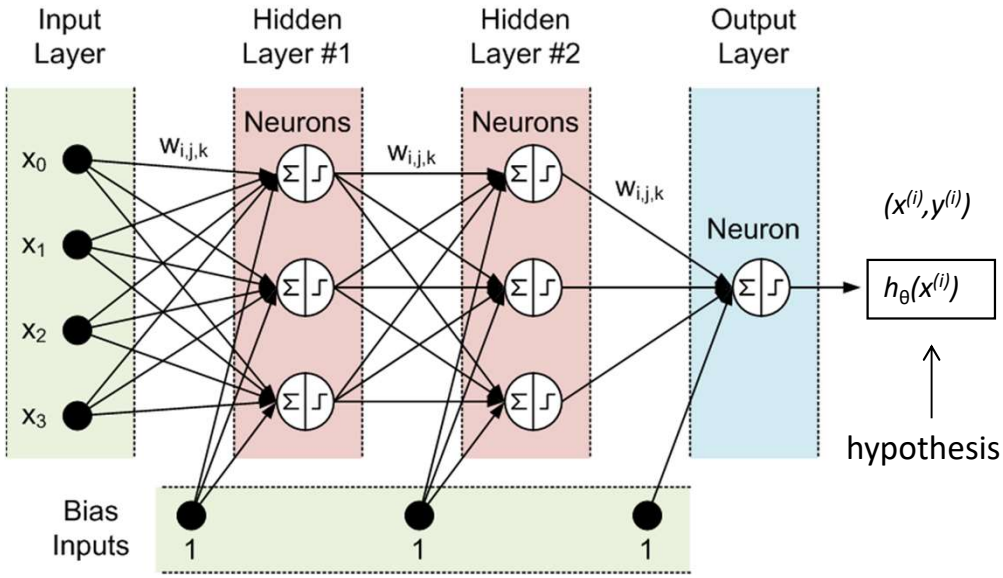
CUNY School of Professional Studies

Linear Transformations

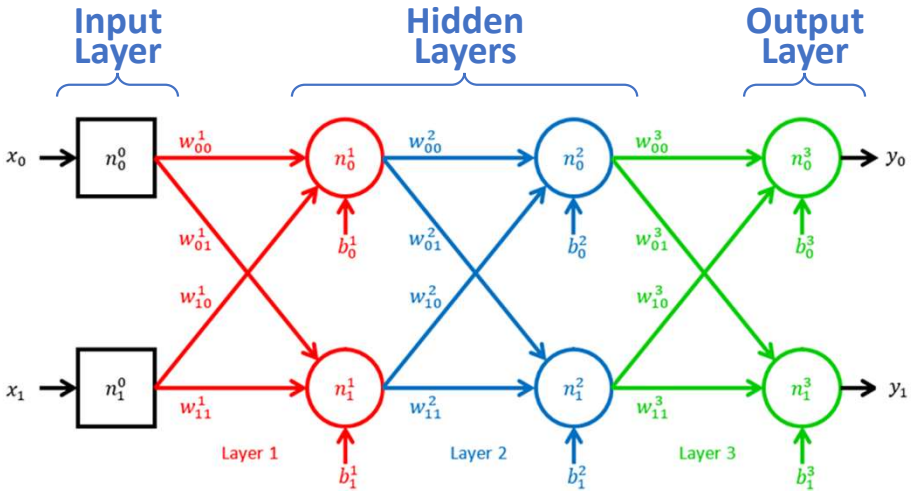


12

Simple Neural Network



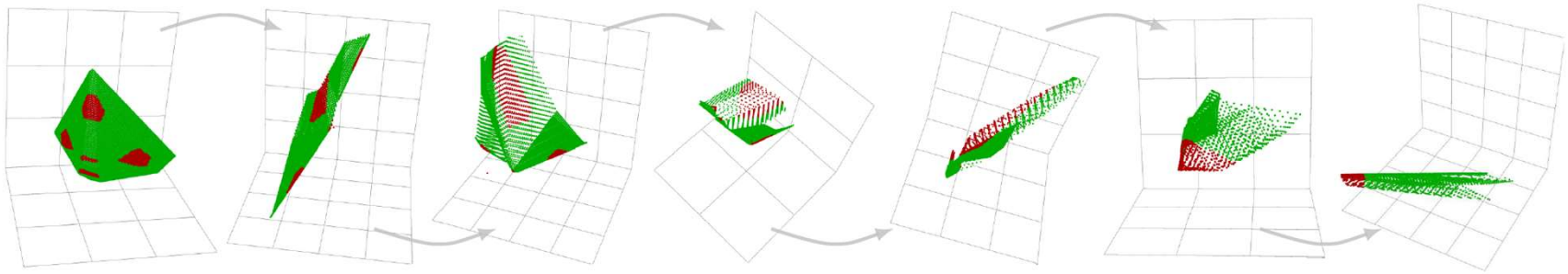
Simple Neural Network



$$\begin{pmatrix} y_0 \\ y_1 \end{pmatrix} = \begin{pmatrix} a\left(\begin{matrix} w_{00}^3 a(w_{00}^2 a(w_{00}^1 x_0 + w_{10}^1 x_1 + b_0^1) + w_{10}^2 a(w_{01}^1 x_0 + w_{11}^1 x_1 + b_1^1) + b_0^2) + \\ w_{10}^3 a(w_{01}^2 a(w_{00}^1 x_0 + w_{10}^1 x_1 + b_0^1) + w_{11}^2 a(w_{01}^1 x_0 + w_{11}^1 x_1 + b_1^1) + b_1^2) + b_0^3 \end{matrix}\right) \\ a\left(\begin{matrix} w_{01}^3 a(w_{00}^2 a(w_{00}^1 x_0 + w_{10}^1 x_1 + b_0^1) + w_{10}^2 a(w_{01}^1 x_0 + w_{11}^1 x_1 + b_1^1) + b_0^2) + \\ w_{11}^3 a(w_{01}^2 a(w_{00}^1 x_0 + w_{10}^1 x_1 + b_0^1) + w_{11}^2 a(w_{01}^1 x_0 + w_{11}^1 x_1 + b_1^1) + b_1^2) + b_1^3 \end{matrix}\right) \end{pmatrix}$$

14

Multi-stage Transformation



Source: “Topology of deep neural networks”, Authors: Gregory Naitzat, Andrey Zhitnikov, Lek-Heng Lim. Apr 13, 2020, https://arxiv.org/abs/2004.06093

15

What does each layer do?

- Linear Transformation
  - Linear transformations using weights and biases
  - Makes data shape larger/smaller (weights) and adds an offset (bias)
  - Resize & move data
- Non-linear transformation
  - Reshape & transform data
  - Morph data (Activation Functions)
- Dimension changes/reduction
  - From layer to layer the dimensions can be changed

16

Demo

https://cs.stanford.edu/people/karpathy/convnetjs/

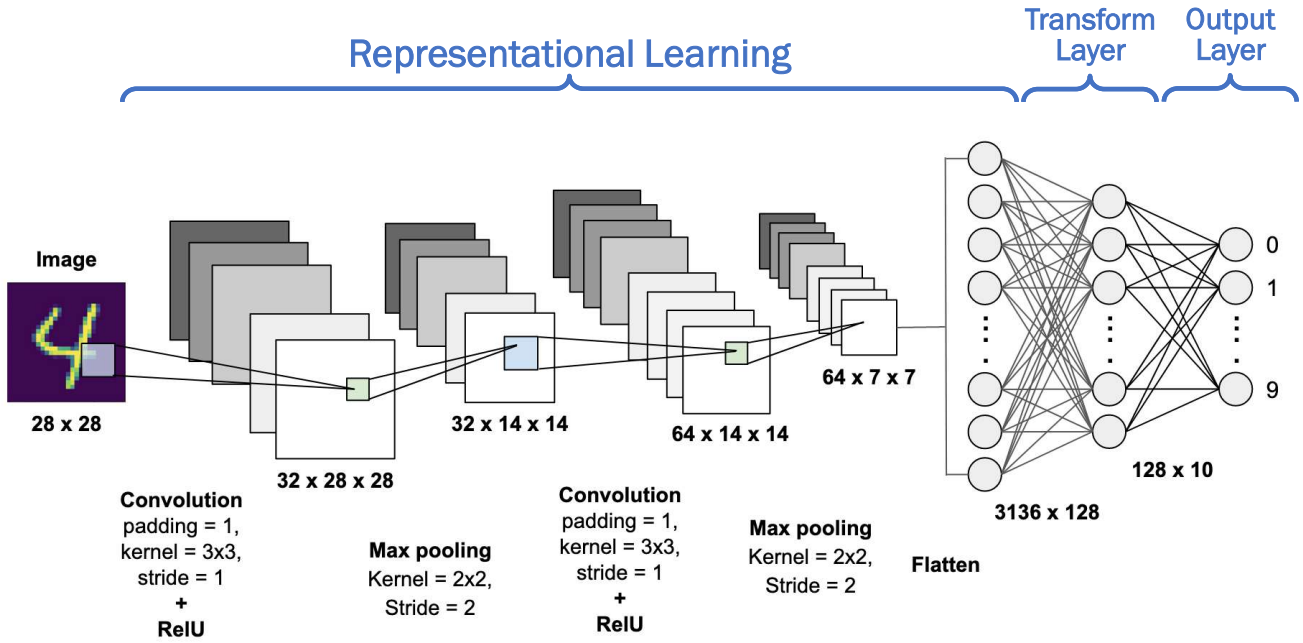
CUNY School of Professional Studies

Why do you need so many layers?

Feature Representation

CUNY School of Professional Studies

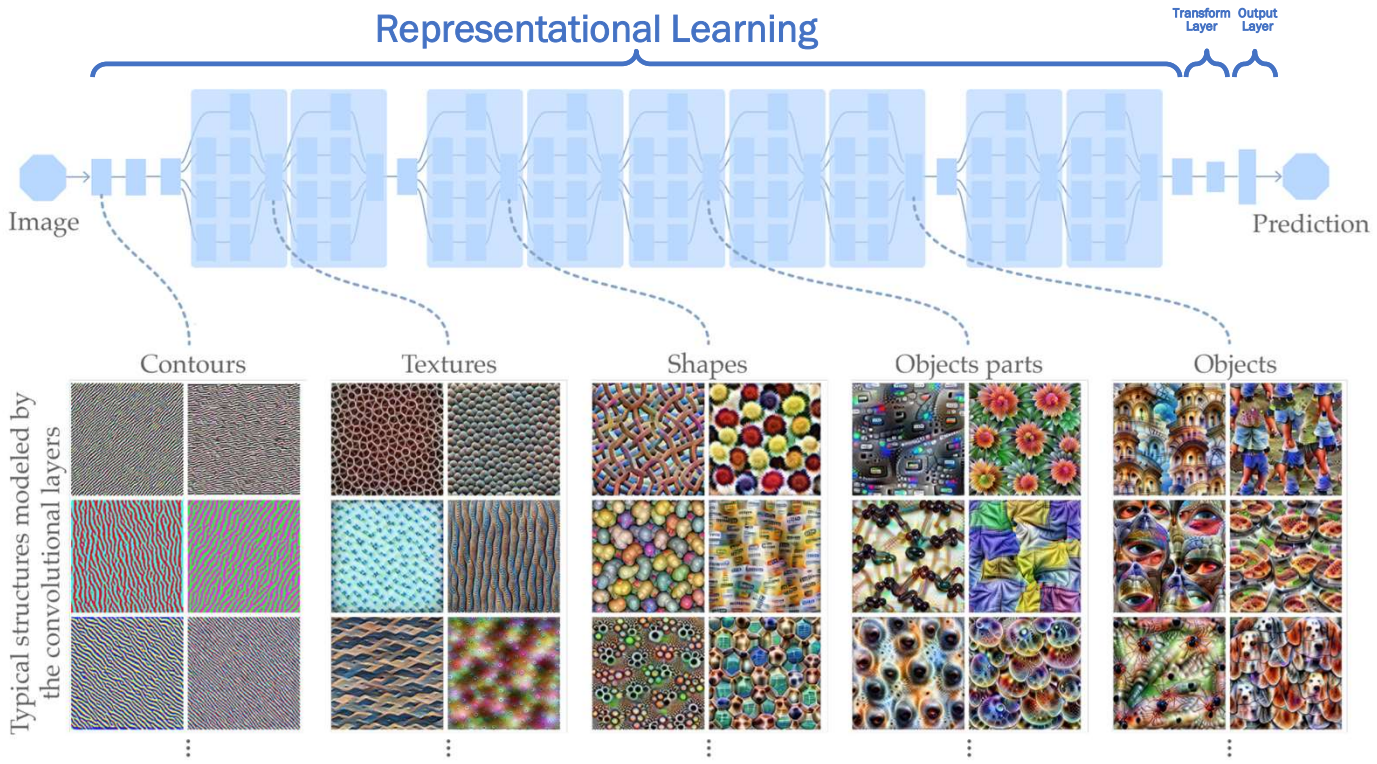
Feature Representation



ConvNet=Convolutional Neural Network, used for images.

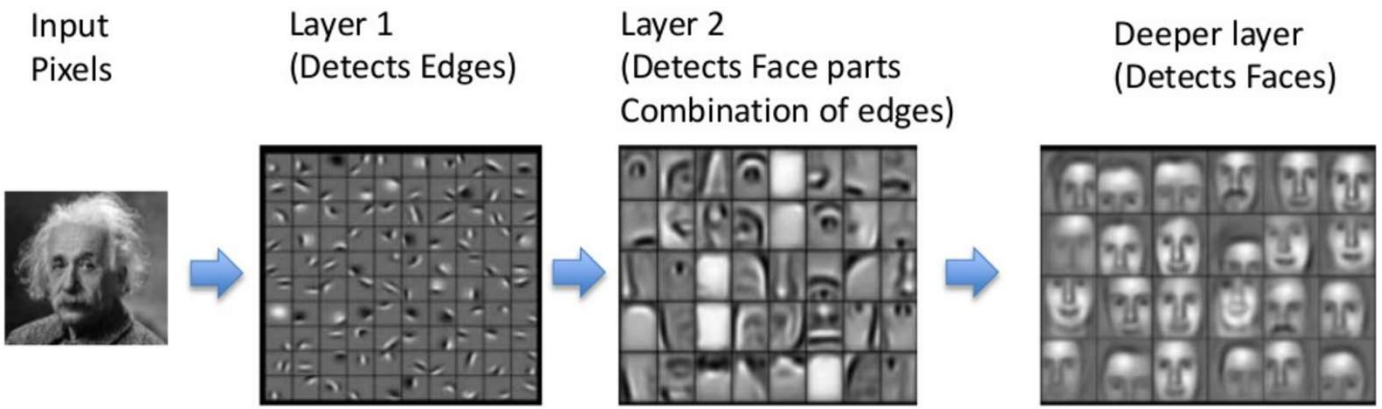
19

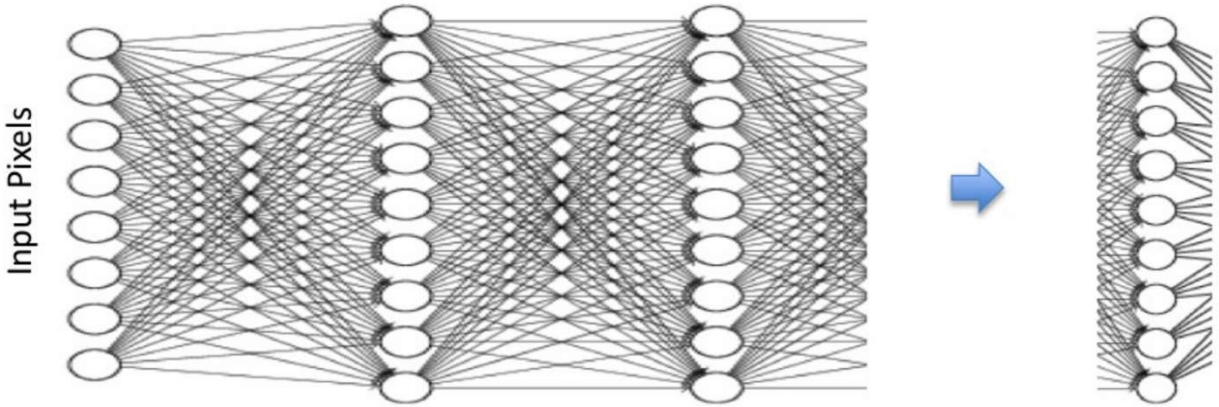
Feature Representation



Source: TLL
Source: Improving Latent Representations of ConvNets for Visual Understanding, Thomas Robert

Feature Representation





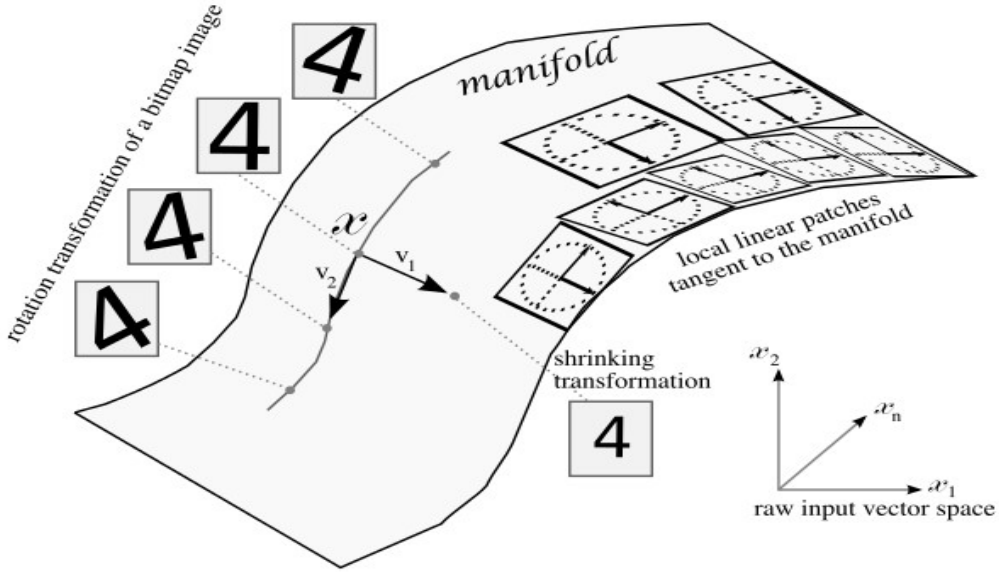
Manifold Hypothesis

Solution space of data

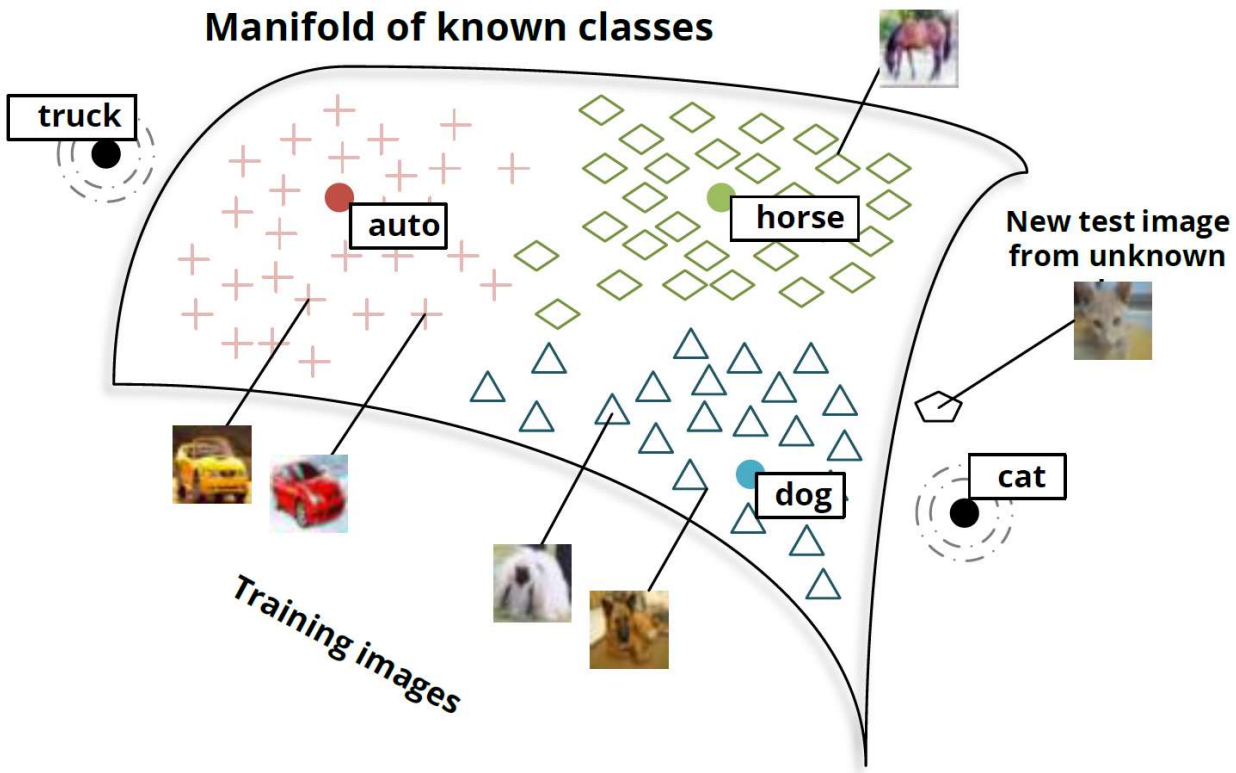
CUNY School of Professional Studies

Manifold Hypothesis

- Manifold Hypothesis states that natural data forms lower dimensional manifolds in its embedding space.
- There are both theoretical and experimental reasons to suspect that the Manifold Hypothesis is true



Another view of manifolds

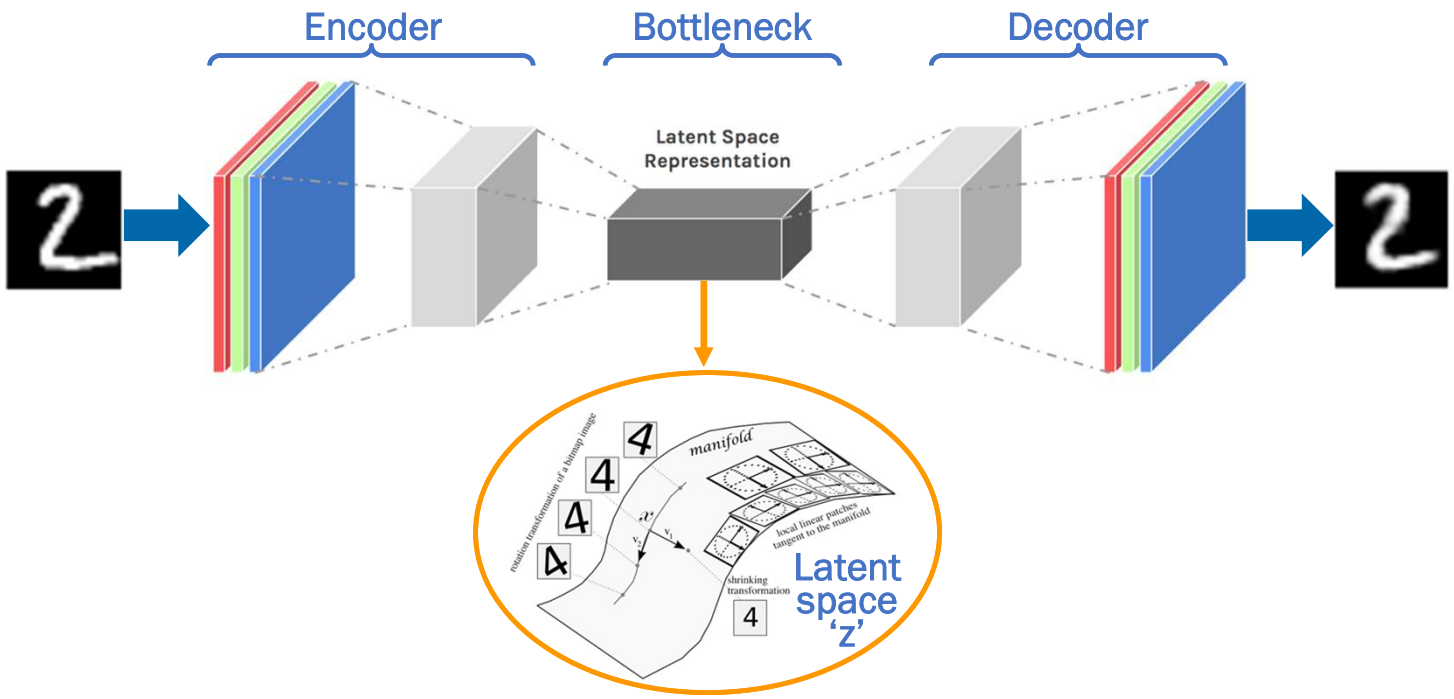


Demo

http://taylordenouden.com/VAE-Latent-Space-Explorer/

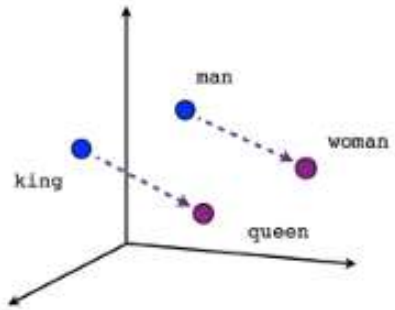
CUNY School of Professional Studies

Auto-encoders

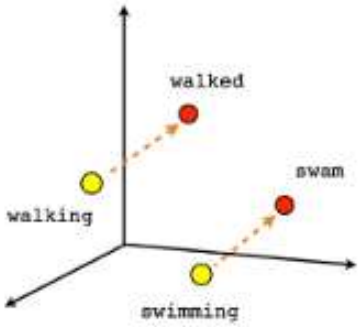


Source: Hackernoon Latent Space Visualization

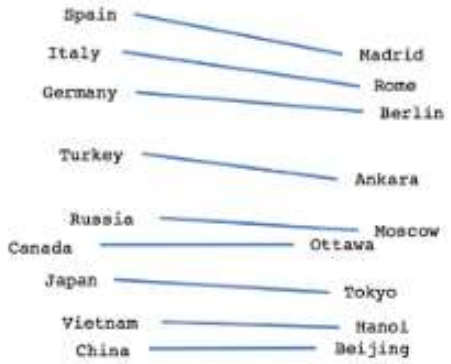
Latent-space vectors



Male-Female

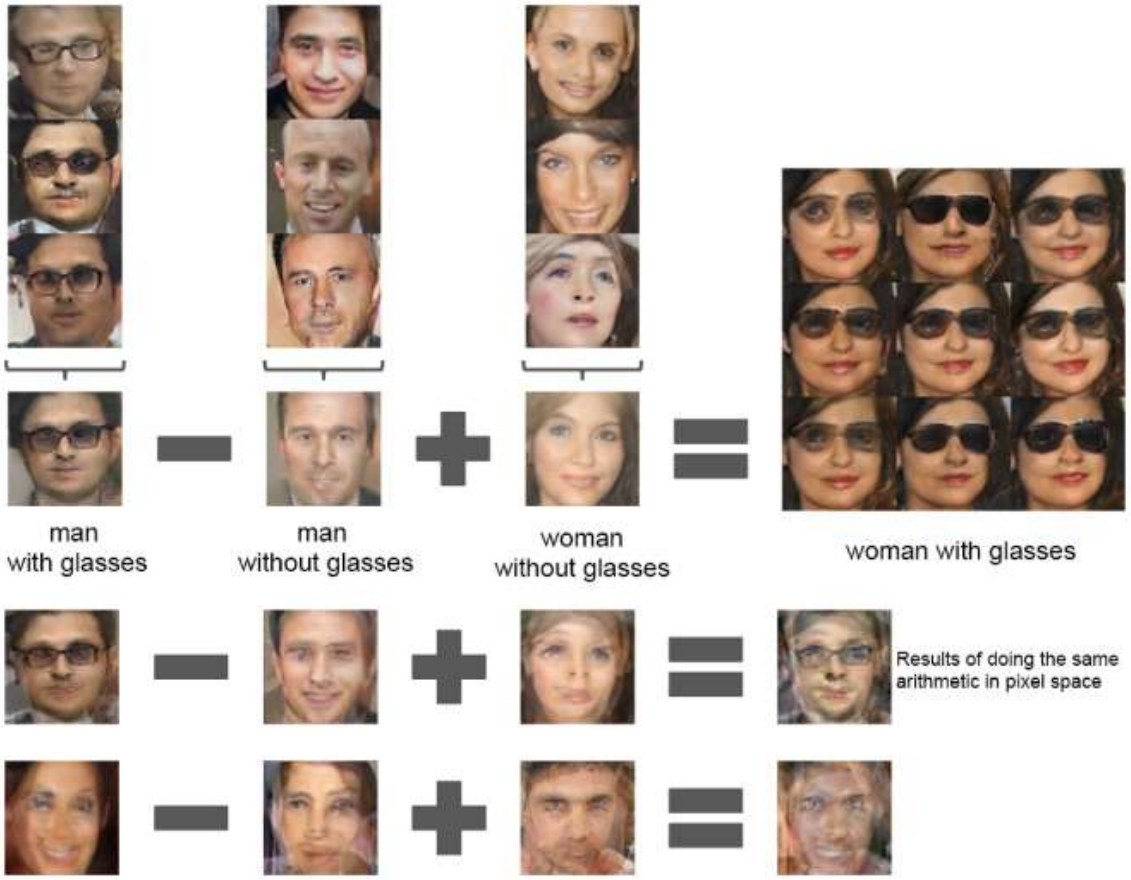


Verb tense



Country-Capital

Arithmetic with latest space vectors



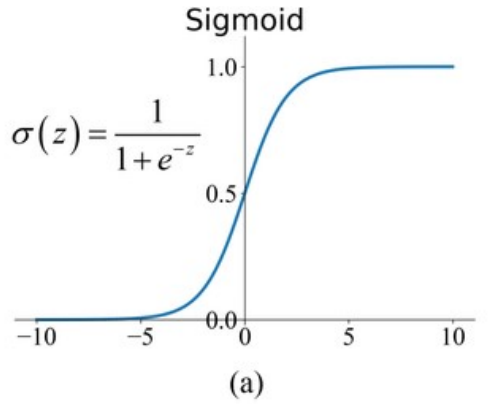
Activation Functions

What do we do about non-linear decision boundaries?

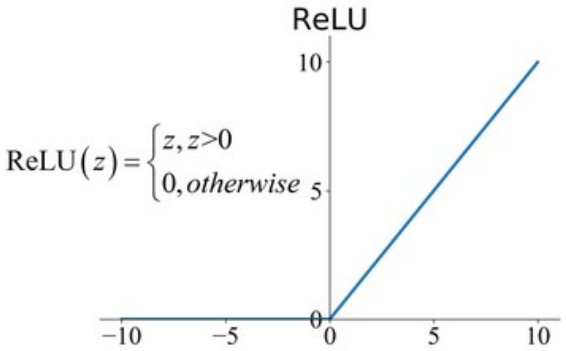
CUNY School of Professional Studies

Activation Functions

- Activation functions provide non-linear transformation (“reshape” data)
- Sigmoid is typically used for classification e.g. to predict the probability, as an output.
- Rectified Linear Unit (ReLU) is the ‘go-to’ choice for most purposes.
- There are a lot of choices of activation functions – these are only two of many – but they must always be non-linear.



(a)

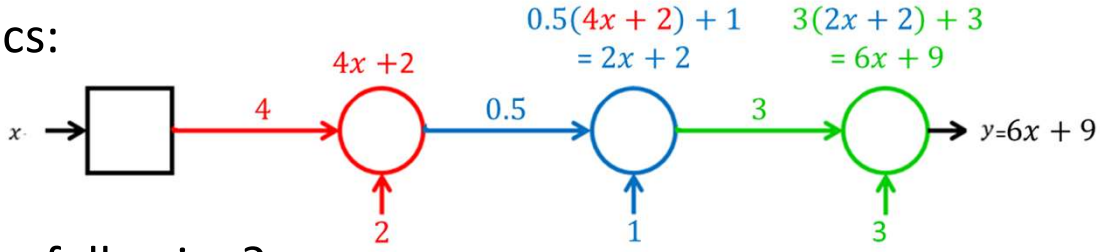


(c)

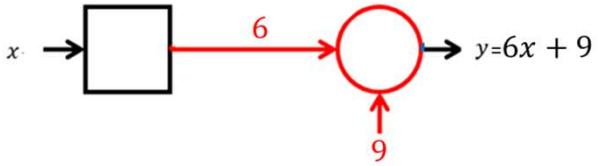
Do I really need activation functions?

Let's consider a simple neural network:

Let's expand the mathematics:



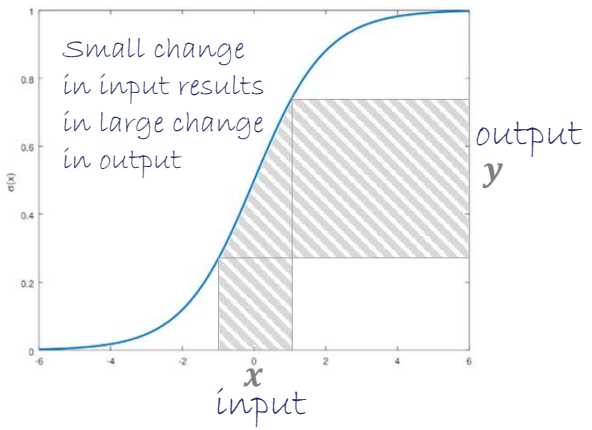
But isn't that the same as the following?

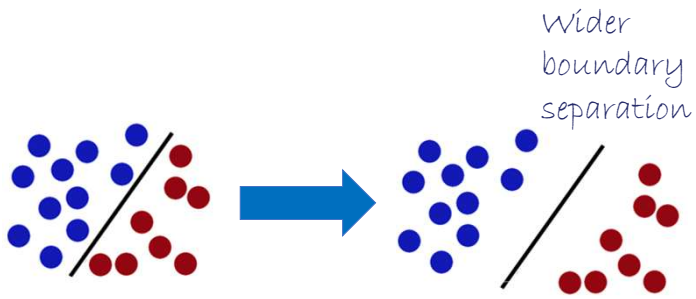


Takeaway: Without a non-linear activation the neural network will collapse

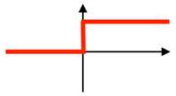
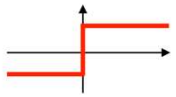
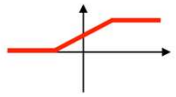
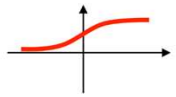
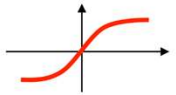
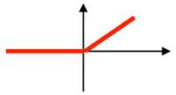
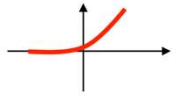
Sigmoid Activation Function

- Useful for output layer for classification problems
- A small change in input causes large change in output, at decision boundary
- Separates classes away from decision boundary
- Softmax performs a similar function for multi-class classification (3 or more classes)





More Activation Functions to choose from

Activation function	Equation	Example	1D Graph
Unit step (Heaviside)	$\phi(z) = \begin{cases} 0, & z < 0, \\ 0.5, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Sign (Signum)	$\phi(z) = \begin{cases} -1, & z < 0, \\ 0, & z = 0, \\ 1, & z > 0, \end{cases}$	Perceptron variant	
Piece-wise linear	$\phi(z) = \begin{cases} 1, & z \geq \frac{1}{2}, \\ z + \frac{1}{2}, & -\frac{1}{2} < z < \frac{1}{2}, \\ 0, & z \leq -\frac{1}{2}, \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression, Multi-layer NN	
Hyperbolic tangent	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$	Multi-layer Neural Networks	
Rectifier, ReLU (Rectified Linear Unit)	$\phi(z) = max(0, z)$	Multi-layer Neural Networks	
Rectifier, softplus	$\phi(z) = \ln(1 + e^{z})$	Multi-layer Neural Networks	

Universal Approximation Theorem

What makes neural networks so powerful?

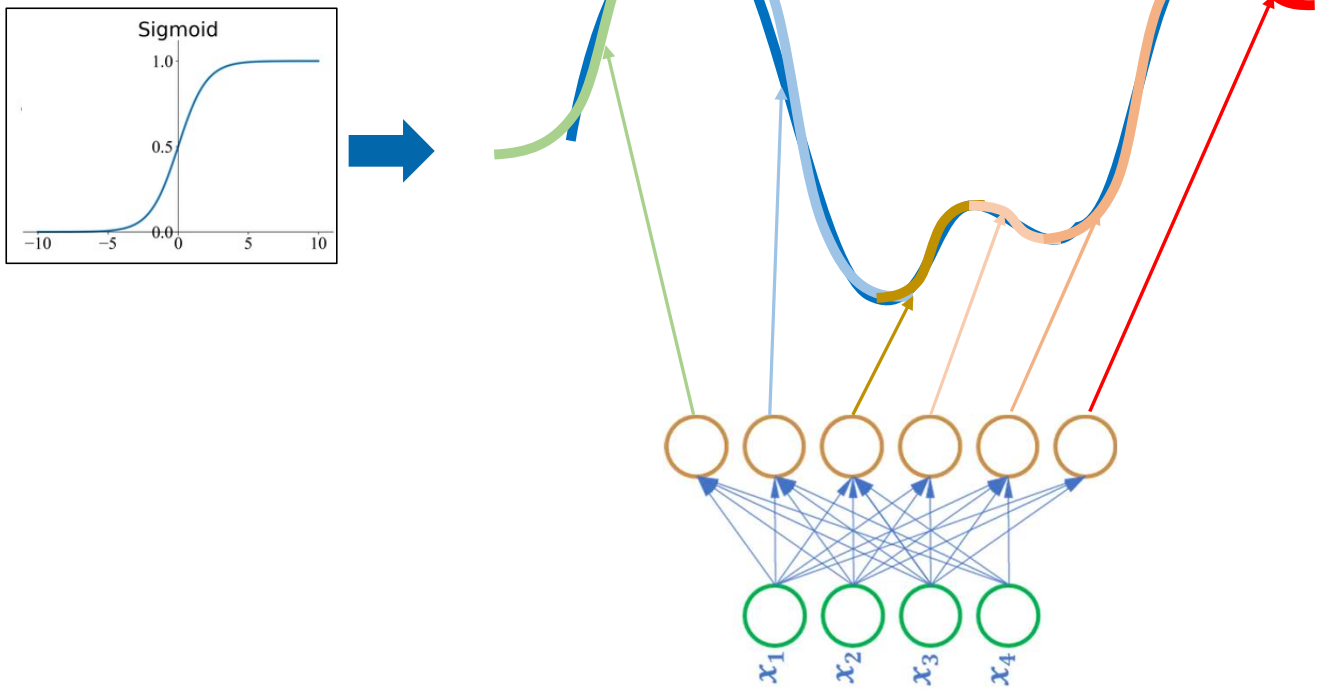
CUNY School of Professional Studies

Universal Approximation Theorem

- A neural network with only 1 hidden layer can approximate any continuous function for inputs within a specific range
- Accuracy of approximation can be controlled by number of layers & neurons
- If the function is not continuous (jumps around or has large gaps) a neural network won't be able to approximate it.

Num Hidden Layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.
>2	Additional layers can learn complex representations (sort of automatic feature engineering) for layer layers.

Universal Approximation Theorem



Note: Simplified diagram – there is also cross-connectivity contribution.

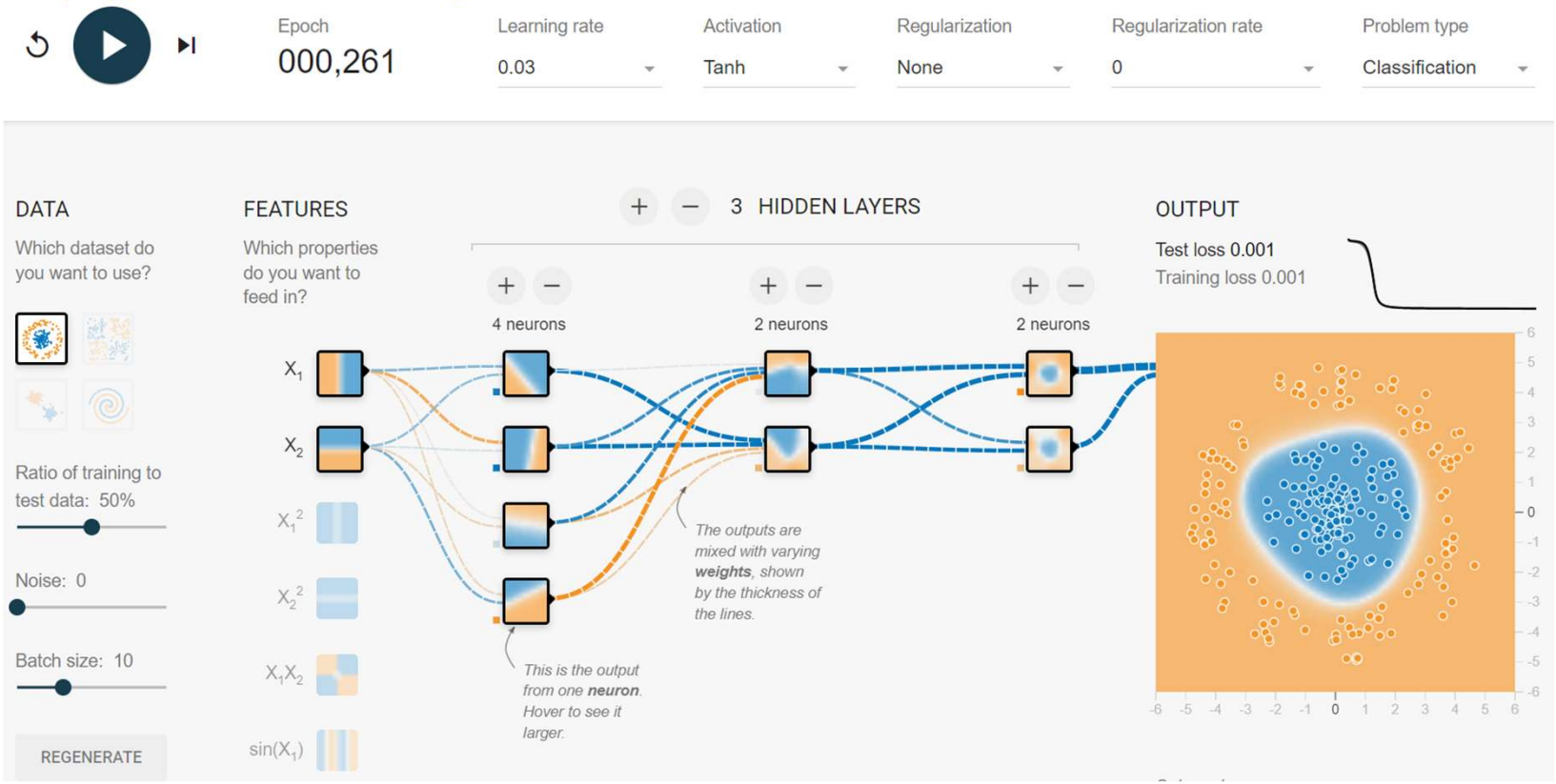
Demo

https://playground.tensorflow.org/

CUNY School of Professional Studies

Neural network demo

https://playground.tensorflow.org/

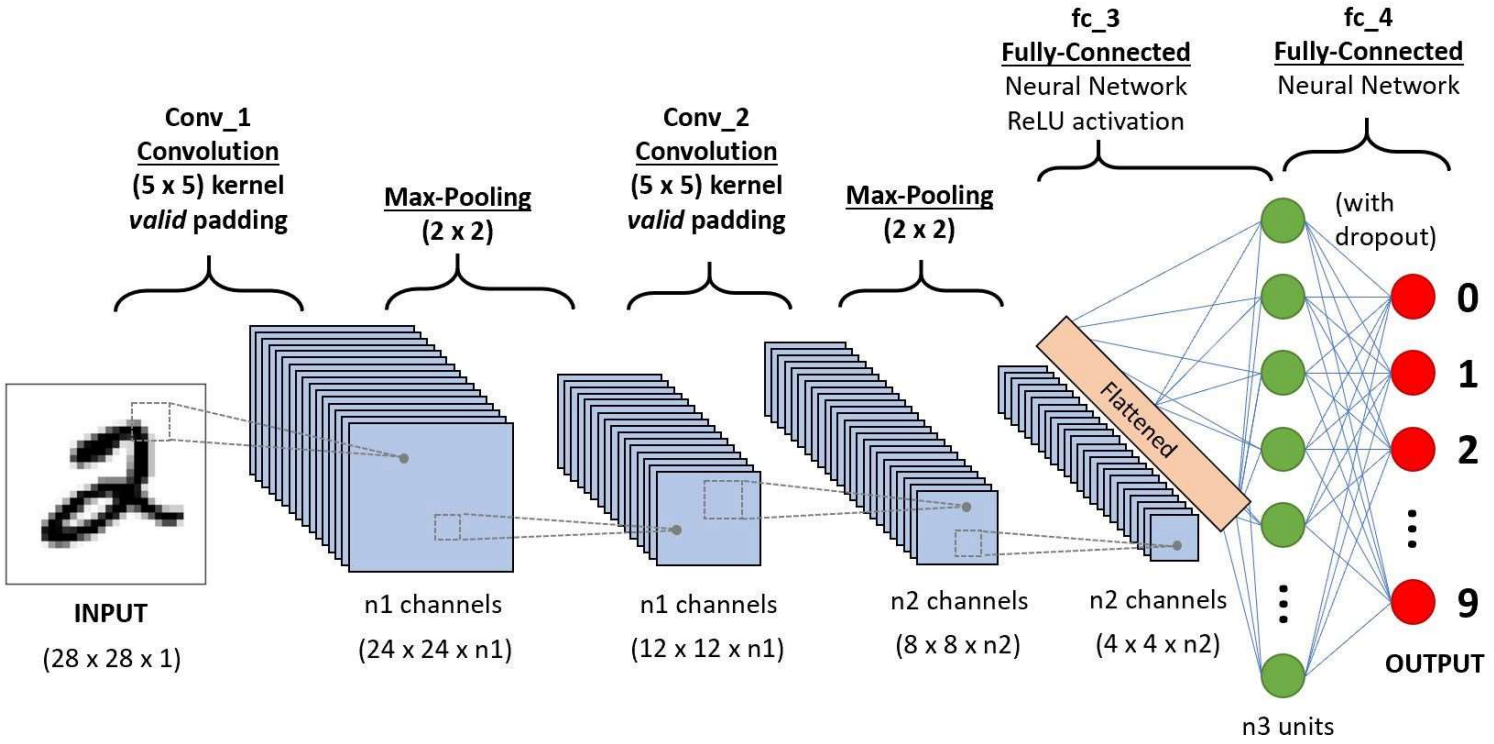


Convolutional Neural Networks

Image processing (inspired by nature)

CUNY School of Professional Studies

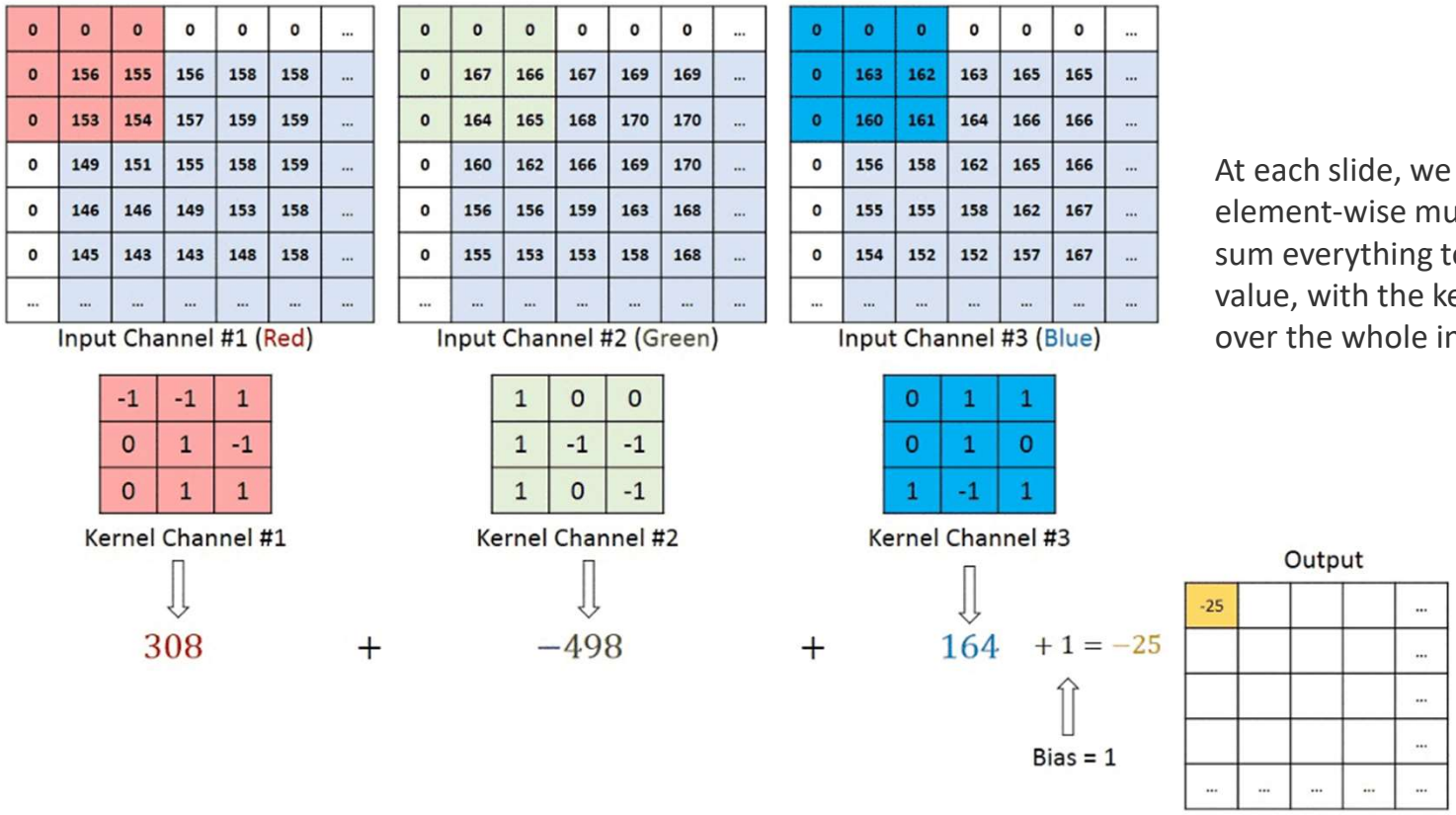
Convolutional Neural Networks (CNN)



Source: Jonas Bokstaller

CUNY School of Professional Studies

Convolutions



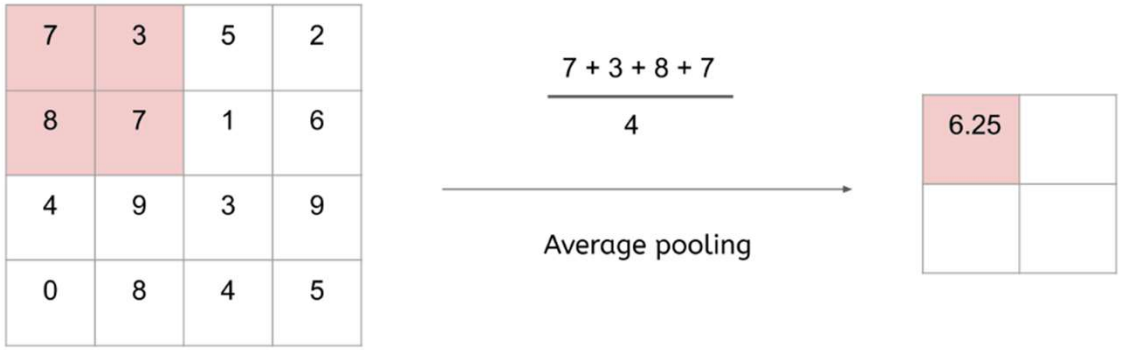
At each slide, we perform an element-wise multiplication and sum everything to get a single value, with the kernel is sliding over the whole input.

Source: https://medium.datadriveninvestor.com/convolutional-neural-networks-3b241a5da51e

Pooling

Max Pooling

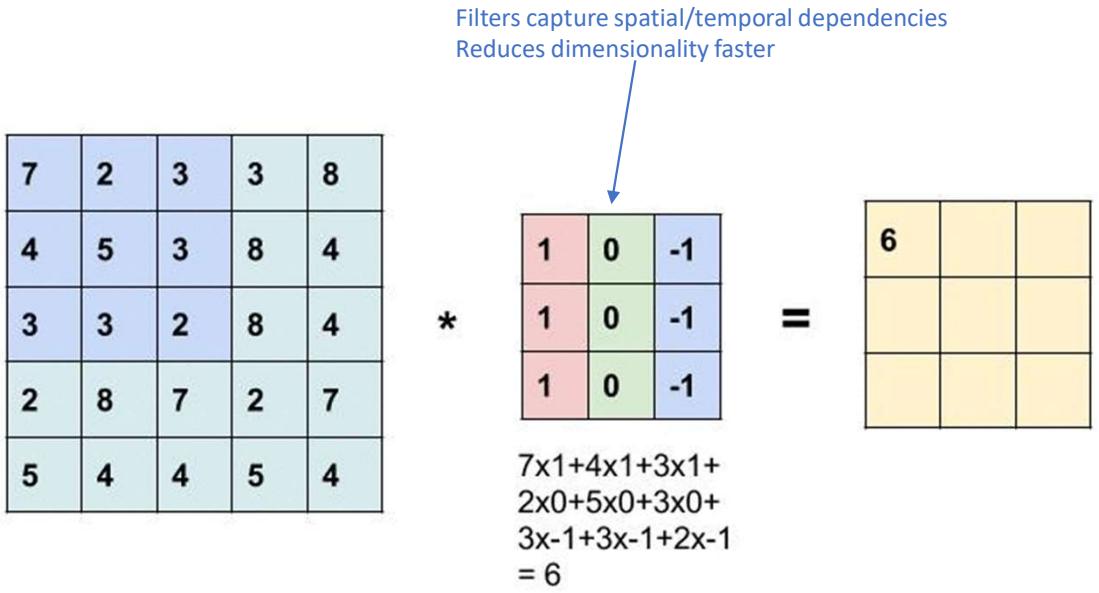
Average pooling



Source: https://www.geeksforgeeks.org/cnn-introduction-to-pooling-layer/

The goal of pooling is to reduce the height and width of our image but not the number of channels by using a stride > 1

Convolutions



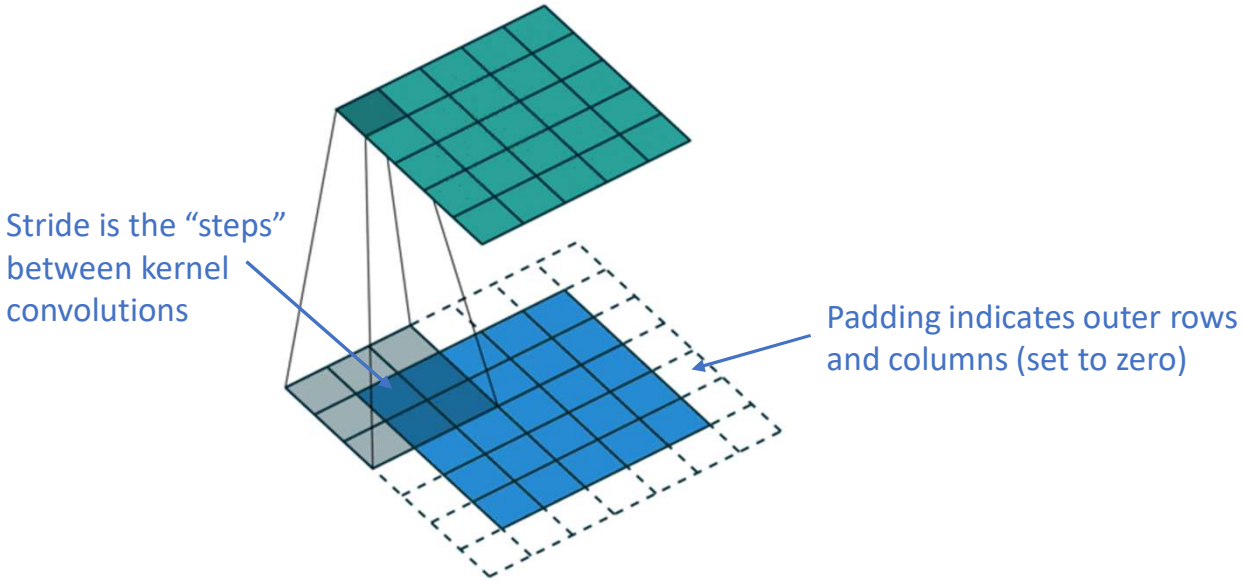
CUNY School of Professional Studies

Some examples of kernels

Original	*Gaussian Blur*	*Sharpen*	*Edge Detection*
$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$
			

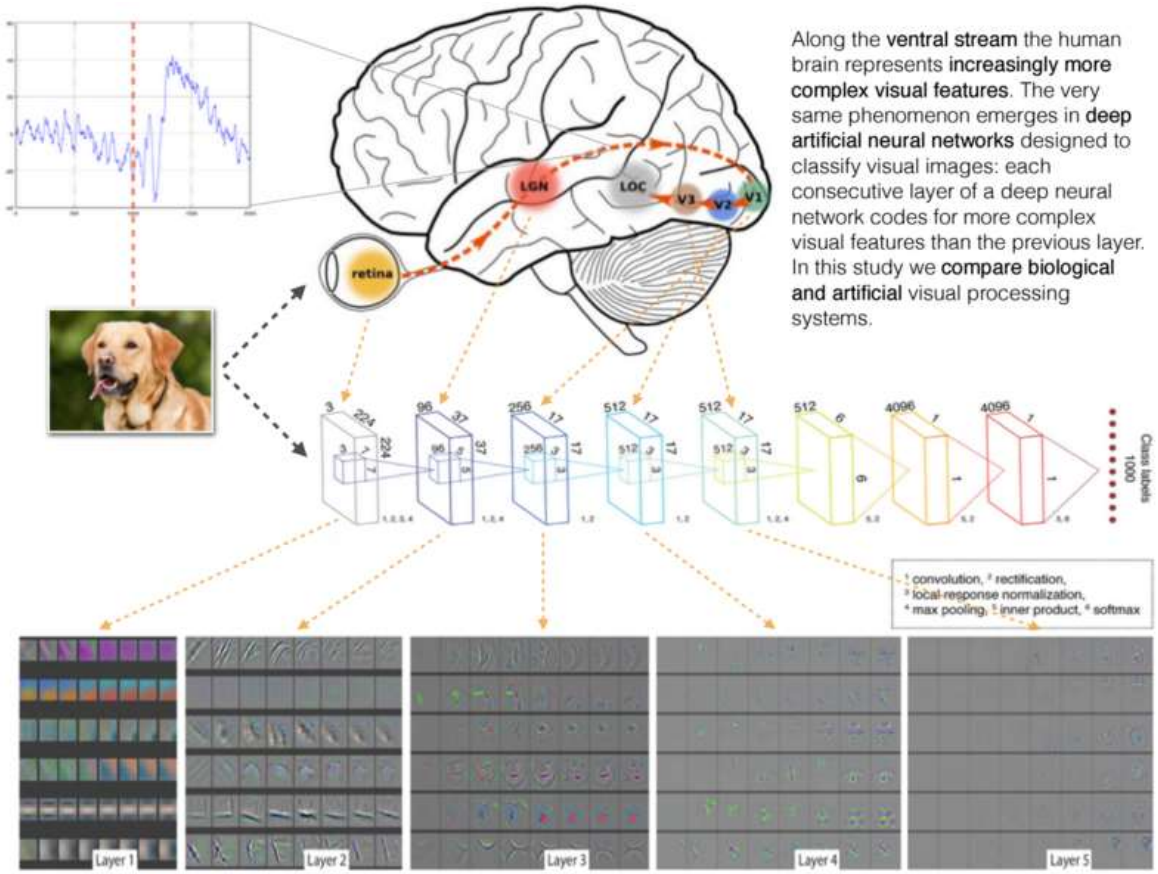
Source: https://medium.com/analytics-vidhya/understanding-convolution-operations-in-cnn-1914045816d4

Padding and stride



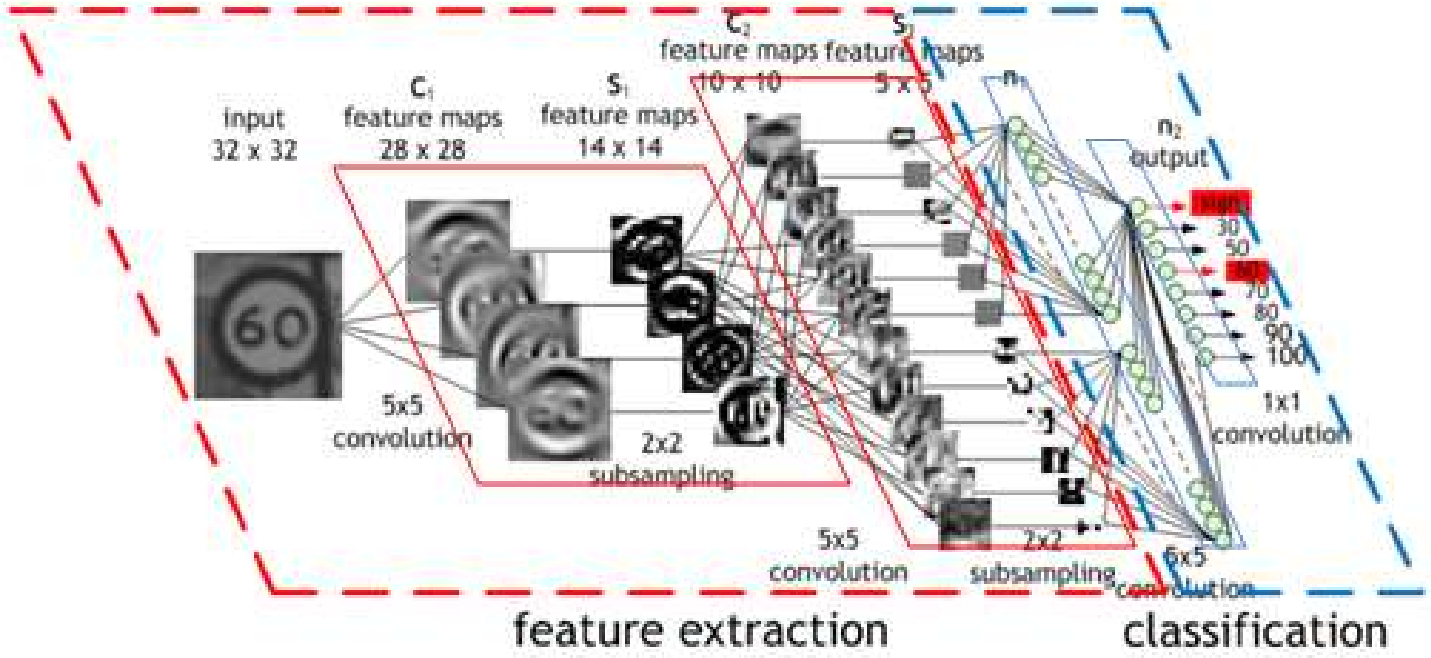
Source: https://medium.datadriveninvestor.com/convolutional-neural-networks-3b241a5da51e

The Brain's visual cortex



CUNY School of Professional Studies

CNN example

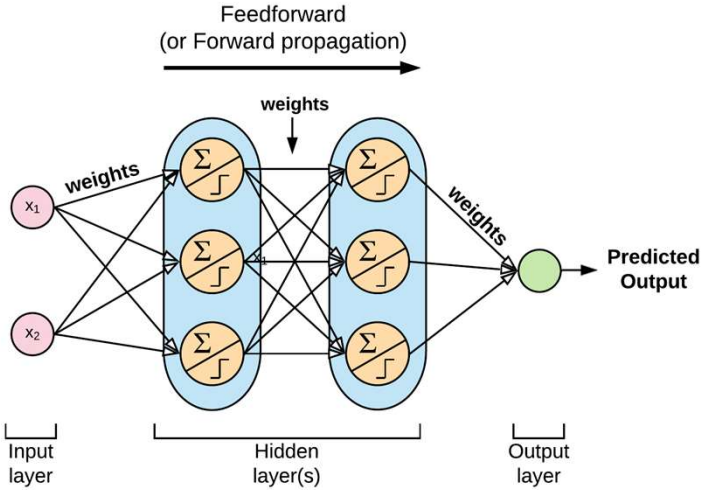


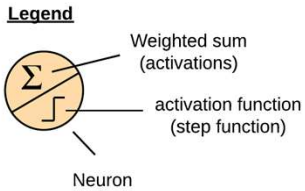
Feedforward

How neural networks make predictions

CUNY School of Professional Studies

Feedforward



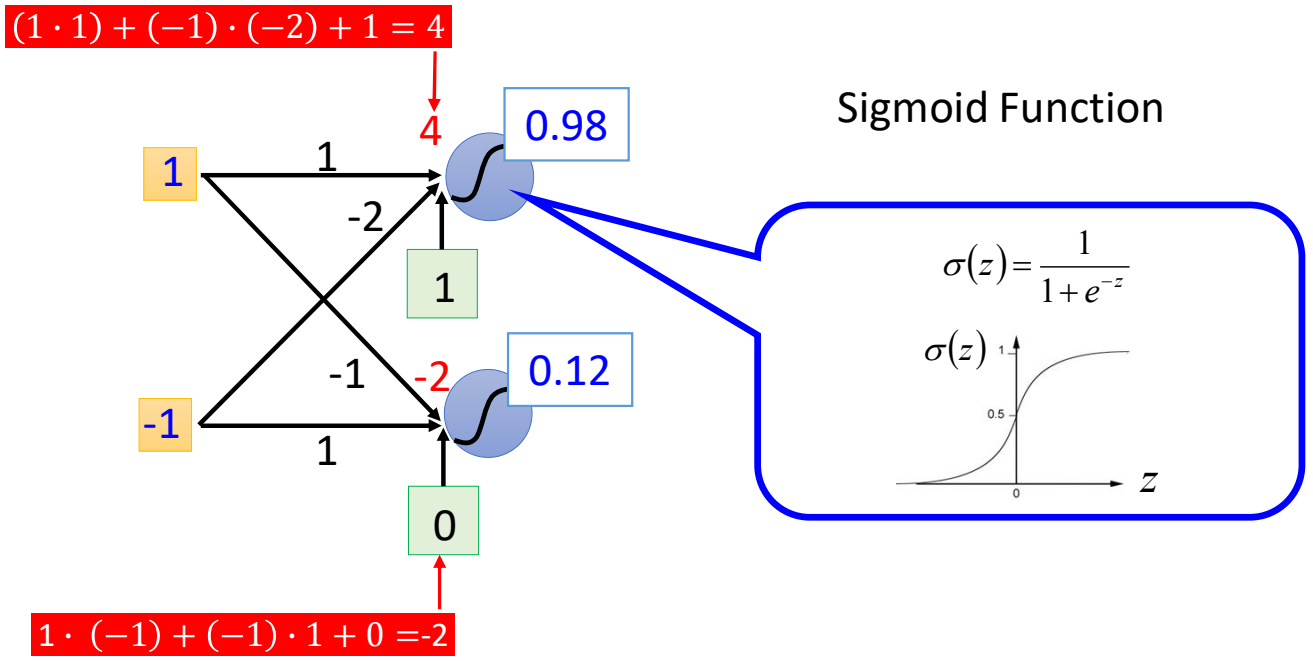


Source: https://ekababisong.org/gcp-ml-seminar/deep-learning/

49

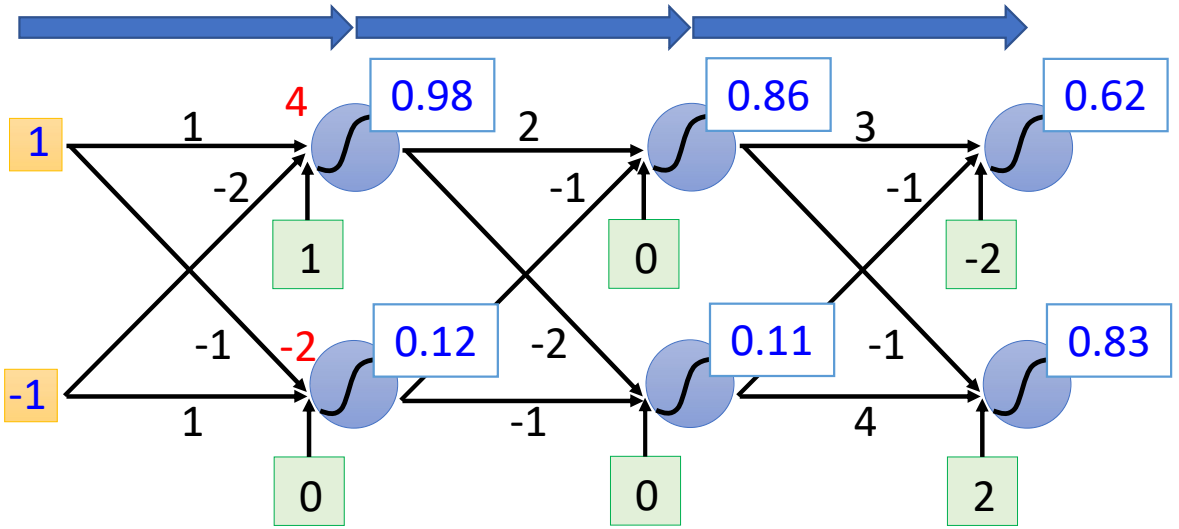
Feedforward

- A simple network, toy example



Source: Hung-yi Lee – Deep Learning Tutorial

Feedforward



$f: R^2 \rightarrow R^2$

$$f\left(\begin{bmatrix}1\\-1\end{bmatrix}\right) = \begin{bmatrix}0.62\\0.83\end{bmatrix}$$

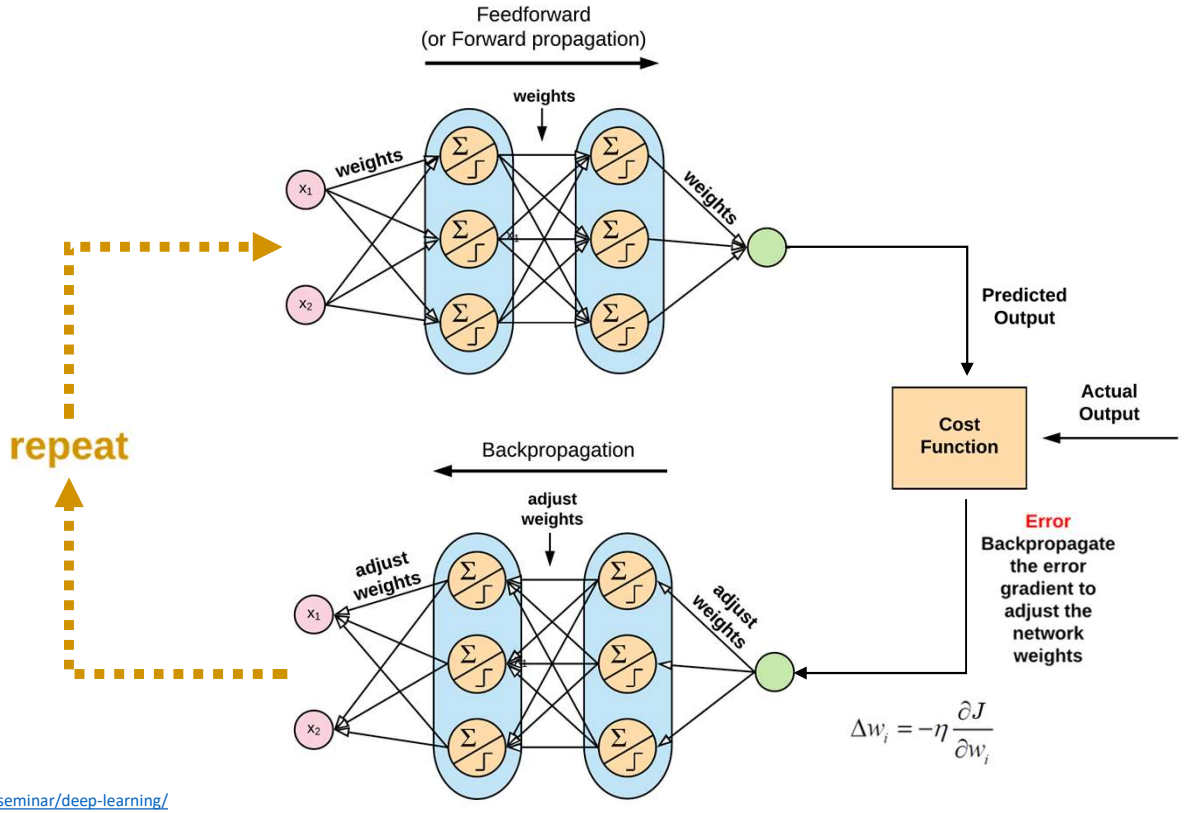
Source: Hung-yi Lee – Deep Learning Tutorial

Backpropagation

How neural networks are trained

CUNY School of Professional Studies

Backpropagation



Source: https://ekababisong.org/gcp-ml-seminar/deep-learning/

Backpropagation

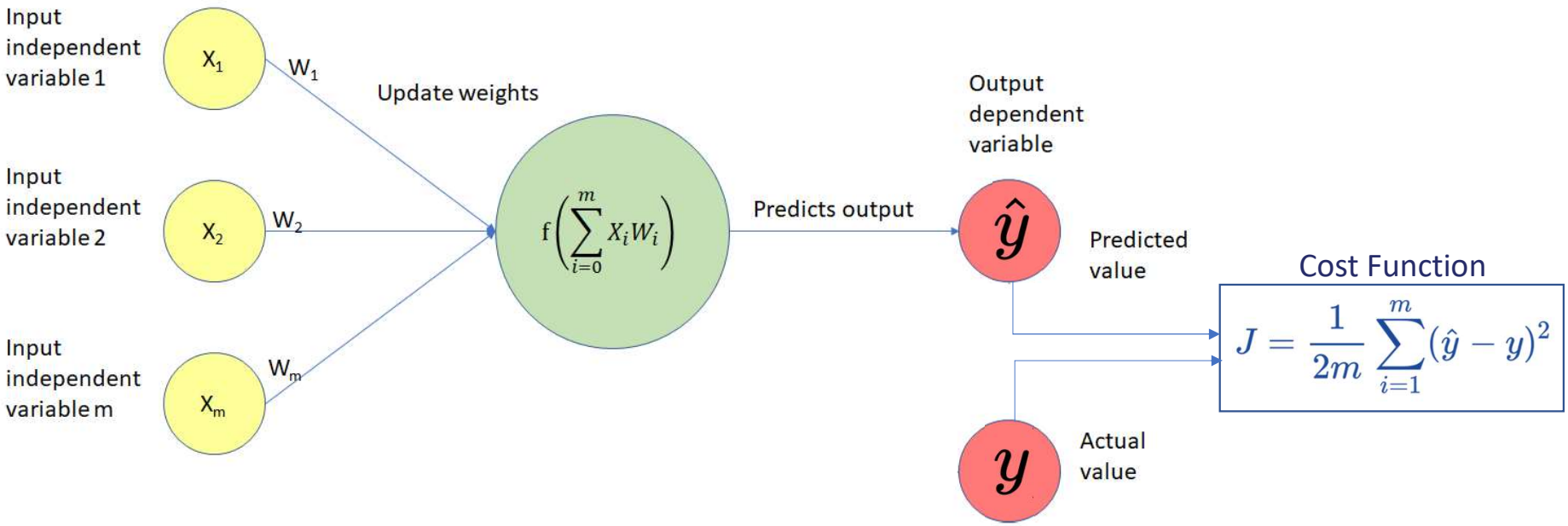
- Backpropagation is a type of gradient descent (terms used interchangeably)
- Gradient descent refers to the calculation of a gradient on each weight in the neural network for each training element.
  - Because the neural network will not output the expected value for a training element, the gradient of each weight will give you an indication about how to modify each weight to achieve the expected output.
  - If the neural network did output exactly what was expected, the gradient for each weight would be 0, indicating that no change to the weight is necessary.
- How it works:
  - Output of NN is evaluated against desired output
  - If results are not satisfactory, connection (weights) between layers are modified and process is repeated again and again until error is small enough.

Cost function

Tracking how your training is performing

CUNY School of Professional Studies

Cost Function (J)



Cost Function

- Cost functions determine how well a model performs for a given dataset
- Cost Functions measure just how wrong the model is in finding a relation between the input and output.
- Comparing other functions:
  - Accuracy functions tell you how well your model does, not how to improve it
  - Error functions measure the difference between the target and the actual values e.g. $(\hat{y} - y)$
  - Loss functions quantify the cost for a single training example e.g. $(\hat{y} - y)^2$
  - Cost functions quantify the average across the entire dataset
    e.g.
    $$J = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y} - y)^2$$
  - Loss functions quantify the impact of the error i.e. error is objective while loss is subjective e.g. we might adopt a non-symmetric loss function if we may be more negatively affected by an error in a particular direction (e.g., false positive vs. false negative)

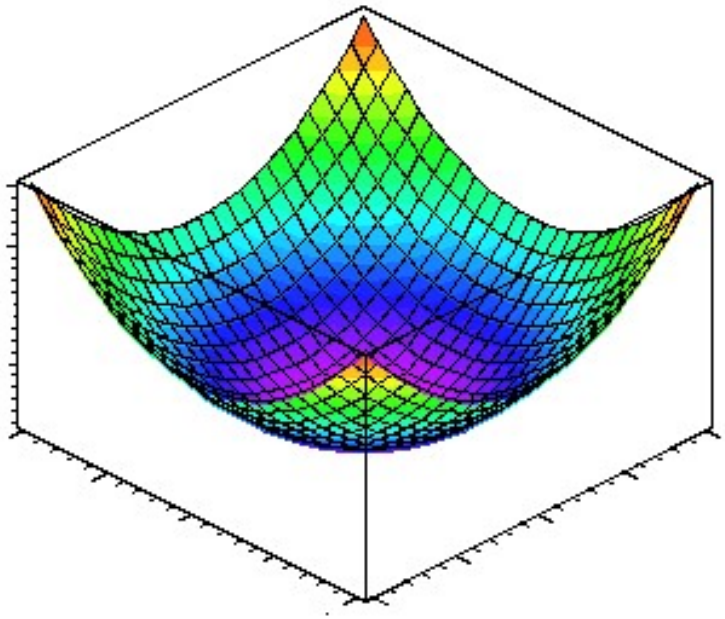
Cost Function

- MSE Cost function will always be parabolic (by definition)
  - So it has only one global minimum

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

- SSE – Similar to MSE but not an average

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



Note: The 2 we added in the denominator makes the mathematics cleaner.

Gradient Descent Rule

Gradient descent rule:

$$w_i \leftarrow w_i + \Delta w_i$$

Where:

$$\Delta w_i = -\eta \frac{\partial J}{\partial w_i}$$

η is a positive constant called the *learning rate*, and determines step size of gradient descent search

Cost Function

$$SSE = \frac{1}{2}\sum_{j=1}^{n}\left(y_j - \hat{y}_j\right)^2$$

$$SSE = \frac{1}{2}\sum_{j=1}^{n}\left(y_j - \left(\sum_{i=0}^{m}\phi\left(w_i^T \times x_{j,i}\right)\right)\right)^2$$

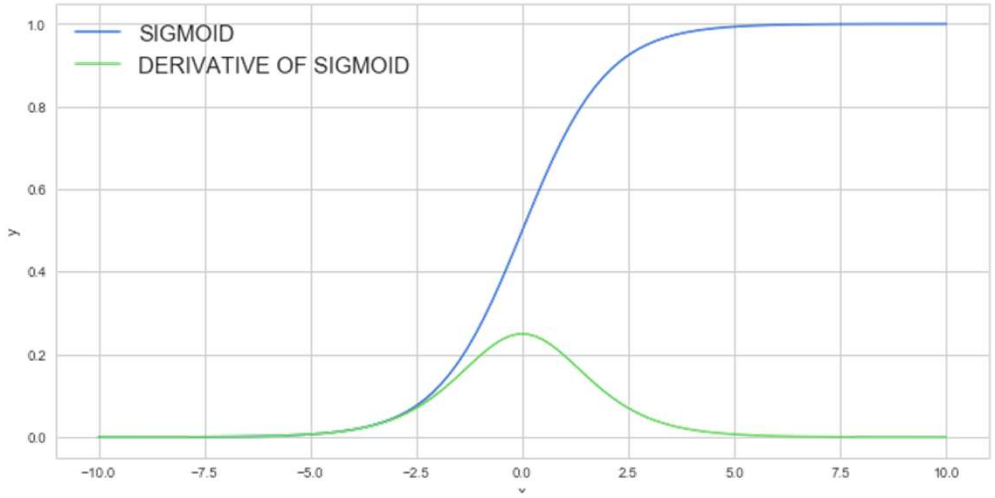
$$\frac{\partial\, SSE}{\partial\, w_i} = \sum_{j=1}^{n}\left(\underbrace{\left(y_j - \hat{y}_j\right)}_{\substack{\text{error of the}\\ \text{output of the}\\ \text{weighted sum}}} \times \underbrace{-x_{j,i}}_{\substack{\text{rate of change of}\\ \text{weighted sum}\\ \text{with respect to}\\ \text{change in } w_i}}\right)$$

Derivative of the Sigmoid Function

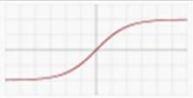
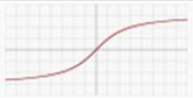
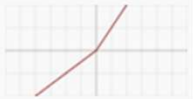
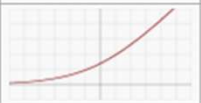
$$y = \frac{1}{1+e^{-x}}$$

$$\frac{dy}{dx} = -\frac{1}{(1+e^{-x})^2}(-e^{-x}) = \frac{e^{-x}}{(1+e^{-x})^2}$$

$$= \frac{1}{1+e^{-x}}\left(1-\frac{1}{1+e^{-x}}\right) = y(1-y)$$



Derivatives of other Activation Functions

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

Cost Function

$$SSE = \frac{1}{2}\sum_{j=1}^{n}\left(y_j - \hat{y}_j\right)^2$$

$$SSE = \frac{1}{2}\sum_{j=1}^{n}\left(y_j - \left(\sum_{i=0}^{m}\phi\left(w_i^T \times x_{j,i}\right)\right)\right)^2$$

$$\frac{\partial\, SSE}{\partial\, w_i} = \sum_{j=1}^{n}\left(\underbrace{\left(y_j - \hat{y}_j\right)}_{\substack{\text{error of the}\\ \text{output of the}\\ \text{weighted sum}}} \times \underbrace{-x_{j,i}}_{\substack{\text{rate of change of}\\ \text{weighted sum}\\ \text{with respect to}\\ \text{change in } w_i}}\right)$$

Gradient Descent Rule or Sigmoid

$$w_i^{t+1} = w_i^t + \left(\eta \times \underbrace{\sum_{j=1}^{n} \left(\left(y_j^t - \hat{y}_j^t \right) \times \underbrace{\left(\hat{y}_j^t \times \left(1 - \hat{y}_j^t \right) \right)}_{\text{Derivative of the sigmoid activation function with respect to the weighted sum}} \times x_{j,i}^t \right)}_{\text{Error gradient for } w_i} \right)$$

Cost Function

$$w_i^{t+1} = w_i^t + \left(\eta \times \underbrace{\sum_{j=1}^{n} \left(\left(y_j^t - \hat{y}_j^t \right) \times x_{j,i}^t \right)}_{\text{error gradient for } w_i} \right)$$

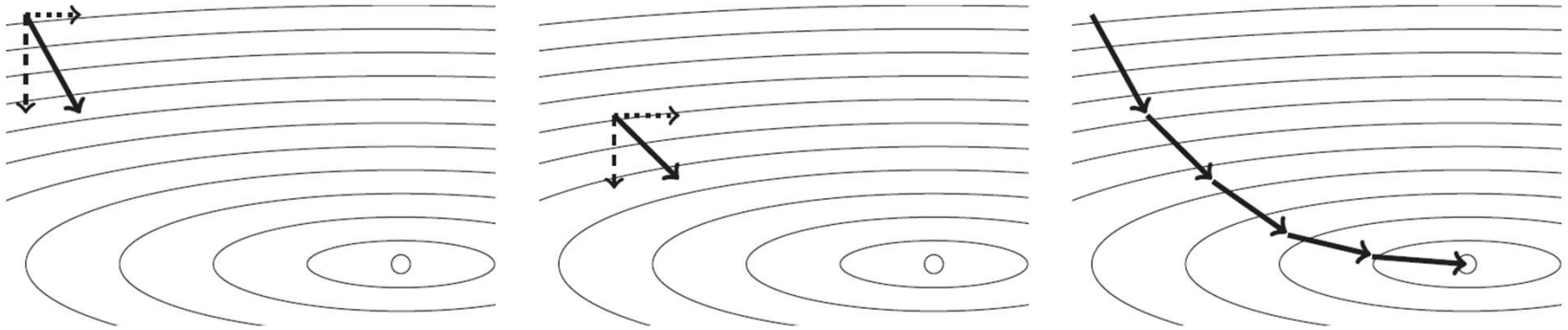
Learning Rate

You need to adjust the learning rate as you train

CUNY School of Professional Studies

Gradient descent

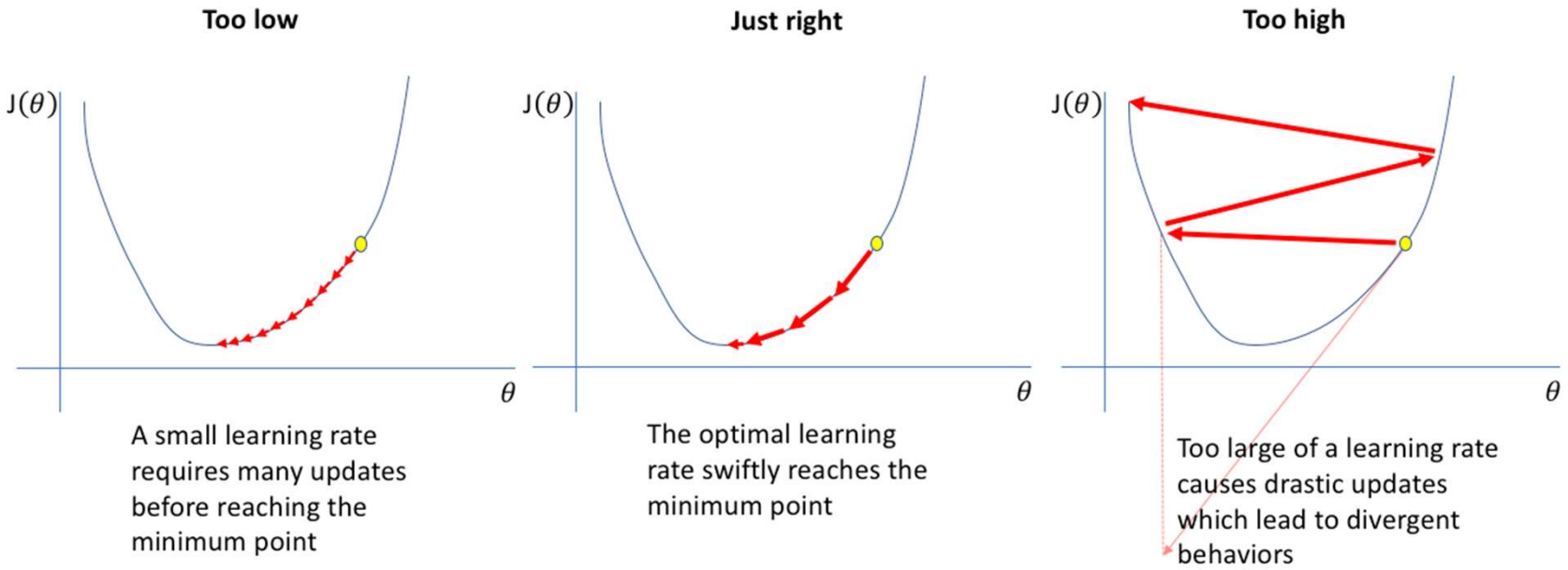
The gradient tells us the direction in which the loss has the steepest rate of increase, but it does not tell us how far we should step



67

Learning Rate

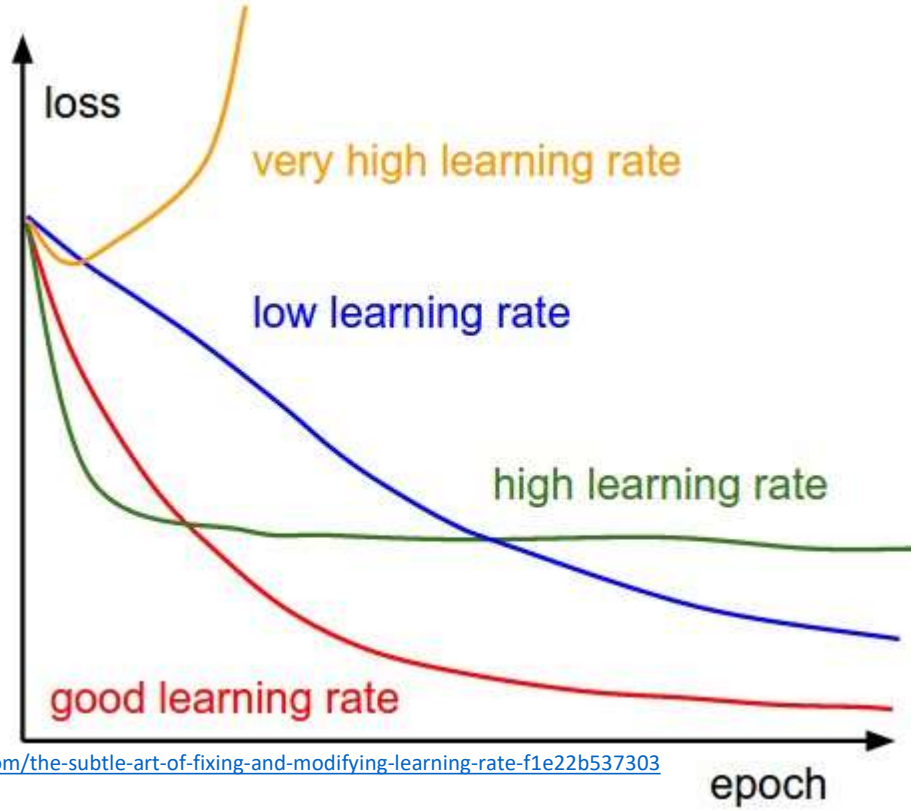
Learning rate (step size) is a hyper-parameter which defines the speed of 'learning'



Source: https://www.jeremyjordan.me/nn-learning-rate/

Impact of different learning rates

- High learning rate: loss increases or plateaus too quickly
- Low learning rate: loss decreases too slowly: too many epochs to reach a solution

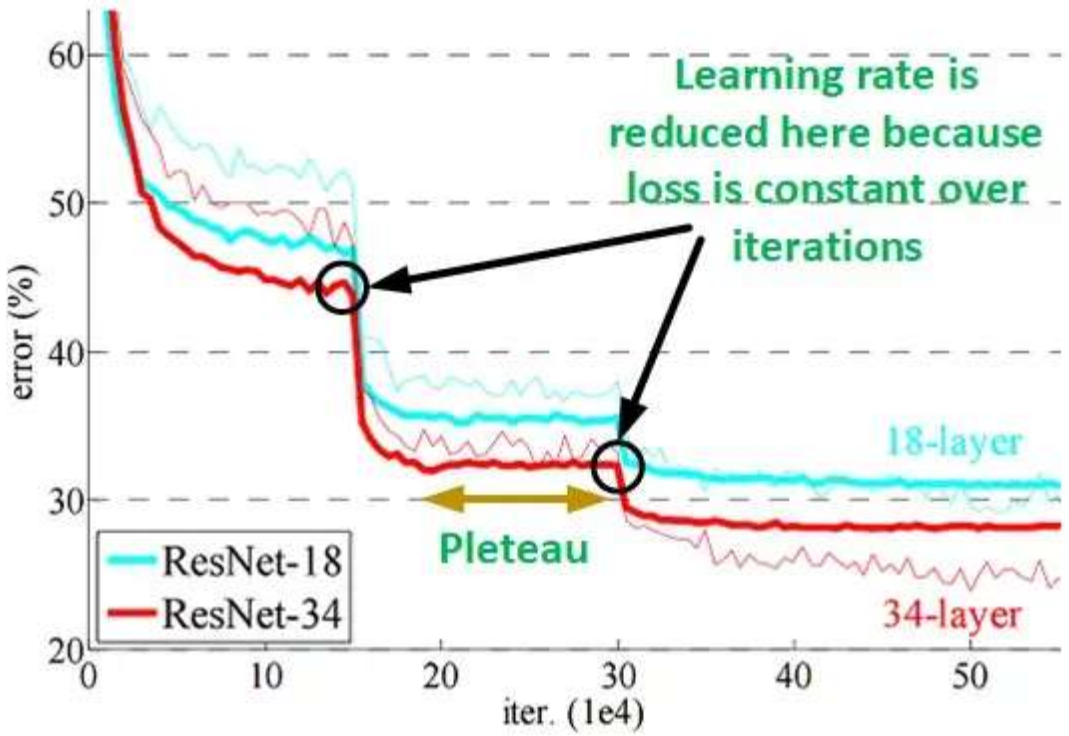


Source: https://towardsdatascience.com/the-subtle-art-of-fixing-and-modifying-learning-rate-f1e22b537303

Adjusting learning rates

- Learning rates must be decayed during training
- Approach 1: Automatic Decay
  - Reduce the learning rate by some factor every few epochs
  - Typical values: reduce the learning rate by a half every 5 epochs, or by 10 every 20 epochs
  - Exponential decay reduces the learning rate exponentially over time
  - These numbers depend heavily on the type of problem and the model
- Approach 2: Plateau-driven
  - Reduce the learning rate by a constant (e.g., by half) when the validation loss stops improving
  - In TensorFlow: tf.keras.callbacks.ReduceLROnPleateau()
  - Monitor: validation loss
  - Factor: 0.1 (i.e., divide by 10)
  - Patience: 10 (how many epochs to wait before applying it)
  - Minimum learning rate: 1e-6 (when to stop)

Adjusting learning rates



Source: https://towardsdatascience.com/the-subtle-art-of-fixing-and-modifying-learning-rate-f1e22b537303

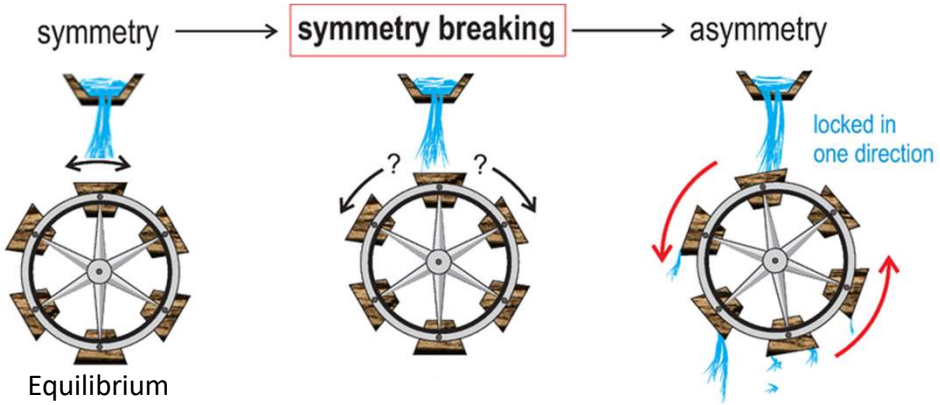
Lottery ticket hypothesis

Breaking symmetry of neural networks with random initialization

CUNY School of Professional Studies

Breaking symmetry

- Initializing weights & biases to zeroes (or ones) is problematic

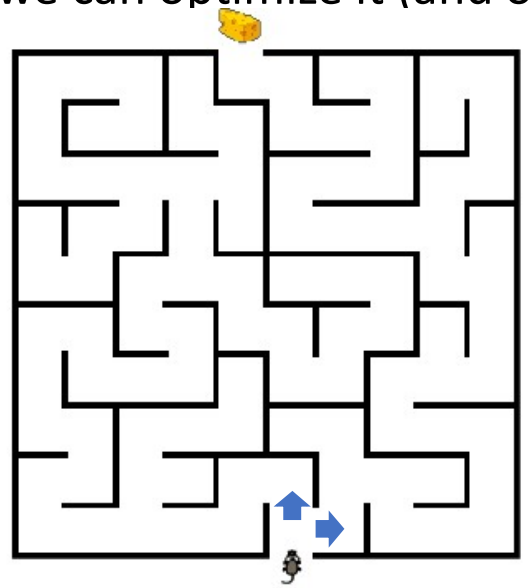


- If all weights are zeros, subsequent hidden layers will get zero signal
- If weights are all the same value (e.g. 1), then subsequent hidden layers will get exactly the same signal i.e. if all weights are initialized to 1, each unit gets signal equal to sum of inputs (and outputs sigmoid(sum(inputs))).
- No matter what was the input - if all weights are the same, all units in hidden layer will be the same too. This is why you should initialize weights randomly.

Mouse Maze

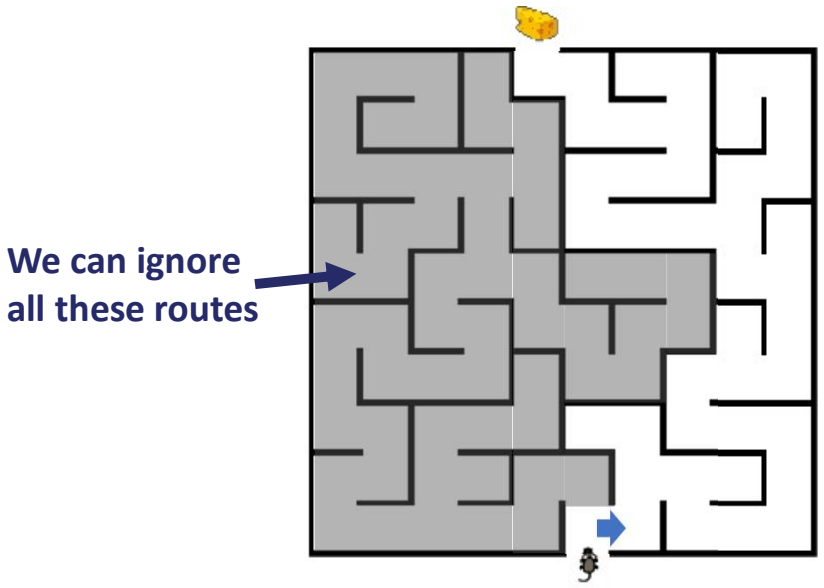
Consider a mouse trying to find a way through a maze:

1. The initial routing will be critical in terms of how quickly a solution is found
2. Once a route is found, we can optimize it (and other routes are ignored)



Mouse Maze

If we have a favorable start and found a route, then the rest of the maze is irrelevant



Lottery ticket hypothesis

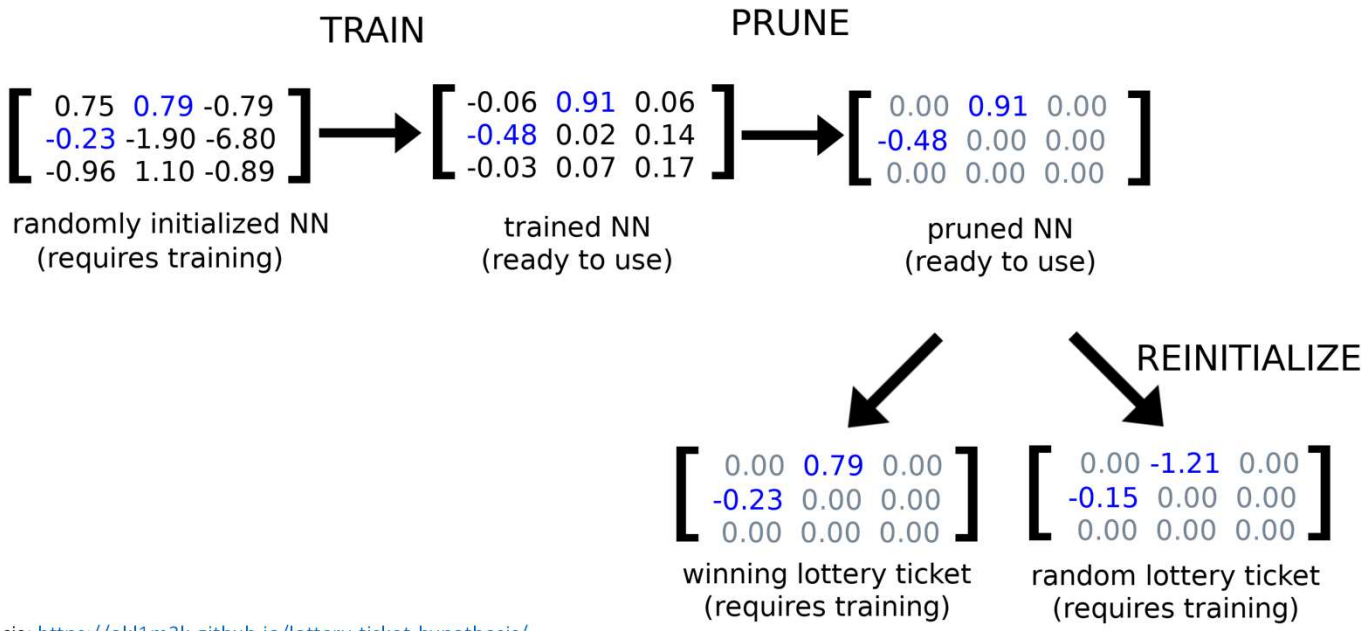
"A randomly-initialized, dense neural network contains a subnetwork that is initialized such that-when trained in isolation-it can match the test accuracy of the original network after training for at most the same number of iterations."

The subnetwork is referred to as a winning ticket (it has won the initialization lottery!)

Lottery ticket hypothesis: https://arxiv.org/pdf/1803.03635.pdf

Lottery ticket hypothesis

- A neural network is typically sparse (most of the weights are zero)
- Pruning the network doesn't change accuracy but makes the model faster
- The lottery ticket (initial optimal weights) will regenerate the optimal model



Lottery ticket hypothesis: https://okl1m3k.github.io/lottery-ticket-hypothesis/