

W.P. Carey AI Study Group

Session #1 - Introduction to Neural Networks
1/31/2020

- Introduction
- Introduction to Neural Networks
- Training Deep Neural Networks
- Advanced Neural Architectures
- Concluding Remarks

- What?
 - Discussion of concepts, tools, and papers
 - Academic and practitioner perspectives
 - Regular cadence to meetings

- Why?
 - Increase of computational needs in industry
 - A tool for our research
 - Our students want to know it

- Who?
 - A community of WPC scholars & friends
 - Logistics: Actionable Analytics Lab & Dept. of IS
 - Point-of-Contact: Victor.Benjamin@asu.edu

- A short survey to help us organize:
 - What types of content are you most interested in?
 - What is your previous experience?
 - How often do you want to meet?

https://wpcareyschool.qualtrics.com/jfe/form/SV_ebpKijcfE3xhWhT

Introduction to Neural Networks

Xiao Liu
Dept. of Information Systems

Introduction to Neural Networks

- An artificial neural network uses a network of artificial neurons or nodes to solve learning problems
 - Used for over 50 years to simulate the brain's approach to problem-solving.
 - In the recent years, the complexity of ANNs has increased so much
- Now frequently applied to more practical problems including:
 - Speech and handwriting recognition programs: voicemail transcription services and postal mail sorting machines
 - The automation of smart devices: self-driving cars and self-piloting drones



Warren McCulloch & Walter Pitts, wrote a paper on how neurons might work; they modeled a simple neural network with electrical circuits.

Nathaniel Rochester from the IBM research laboratories led the first effort to simulate a neural network.

John von Neumann suggested imitating simple neuron functions by using telegraph relays or vacuum tubes.

STORY BY DATA

1943

1949

1950s

1956

1957

1958

Donald Hebb reinforced the concept of neurons in his book, *The Organization of Behavior*. It pointed out that neural pathways are strengthened each time they are used.

The **Dartmouth Summer Research Project** on Artificial Intelligence provided a boost to both artificial intelligence and neural networks.

Frank Rosenblatt began work on the Perceptron; the oldest neural network still in use today.

1982

1981

1969

1959

1982

John Hopfield presented a paper to the national Academy of Sciences. His approach to create useful devices; he was likeable, articulate, and charismatic.

Progress on neural network research halted due fear, unfulfilled claims, etc.

Marvin Minsky & Seymour Papert proved the Perceptron to be limited in their book, *Perceptrons*.

Bernard Widrow & Marcian Hoff of Stanford developed models they called ADALINE and MADALINE; the first neural network to be applied to a real world problem.

1982

1985

1997

1998

NOW

US-Japan Joint Conference on Cooperative/Competitive Neural Networks; Japan announced their Fifth-Generation effort resulted in US worrying about being left behind and restarted the funding in US.

American Institute of Physics began what has become an annual meeting - **Neural Networks for Computing**.

A recurrent neural network framework, LSTM was proposed by **Schmidhuber & Hochreiter**.

Yann LeCun published *Gradient-Based Learning Applied to Document Recognition*.

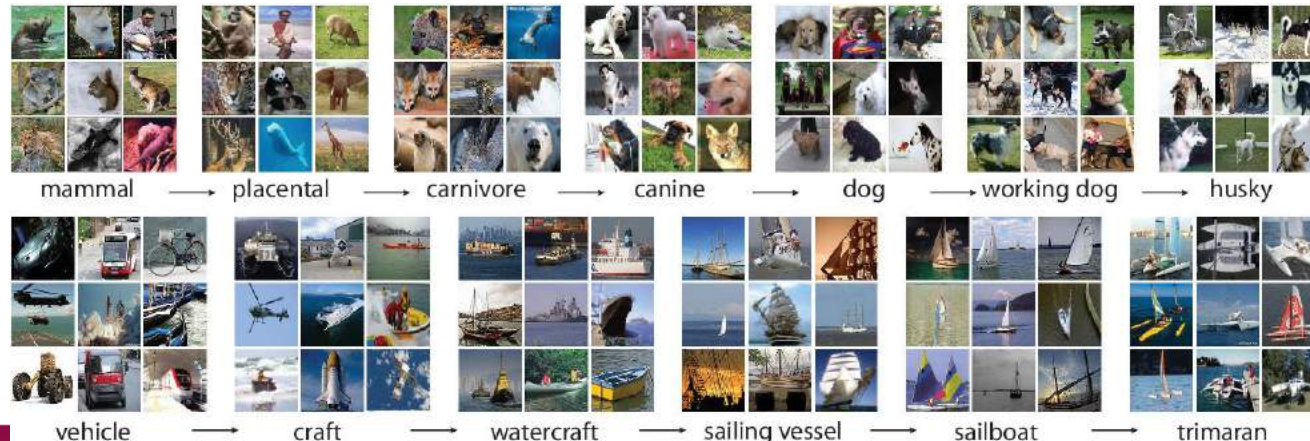
Neural networks discussions are prevalent; the future is here!

HISTORY OF NEURAL NETWORKS

1943-2019

2009 – Launch of ImageNet

- A professor and head of the Artificial Intelligence Lab at Stanford University, Fei-Fei Li launched ImageNet in 2009.
 - As of 2017, it's a very large and free database of more than 14 million (14,197,122 at last count) *labeled* images available to researchers, educators, and students.
 - Labeled data – such as these images – are needed to “train” neural nets in supervised learning.



2011 – Creation of AlexNet

- Between 2011 and 2012, Alex Krizhevsky won several international machine and deep learning competitions with his creation AlexNet, a convolutional neural network.
 - AlexNet built off and improved upon LeNet5 (built by Yann LeCun years earlier).
 - It initially contained only eight layers – five convolutional followed by three fully connected layers – and strengthened the speed and dropout using rectified linear units.
- Its success kicked off a convolutional neural network renaissance in the deep learning community.

2012 – The Cat Experiment

- Computer scientists at Google built a neural network of 16,000 computer processors and let it browse YouTube.
 - Exposed to 10 million randomly selected YouTube video thumbnails over the course of three days.
 - Despite being fed no information on distinguishing features, it began to recognize pictures of cats.
 - 81.7 percent accuracy in detecting human faces, 76.7 percent accuracy when identifying human body parts and 74.8 percent accuracy when identifying cats.
- The network recognized only about 15% of the presented objects. That said, it was yet another baby step towards genuine AI.

2014 – DeepFace

- Developed and released to the world in 2014, Facebook's deep learning system – nicknamed DeepFace
 - Trained on the largest facial dataset of four million facial images belonging to more than 4,000 identities
 - Reaches an accuracy of 97.35% on the Labeled Faces in the Wild (LFW) dataset, reducing the error of the current state of the art by more than 27%, closely approaching human-level performance ([reported to be 97.5%](#)).

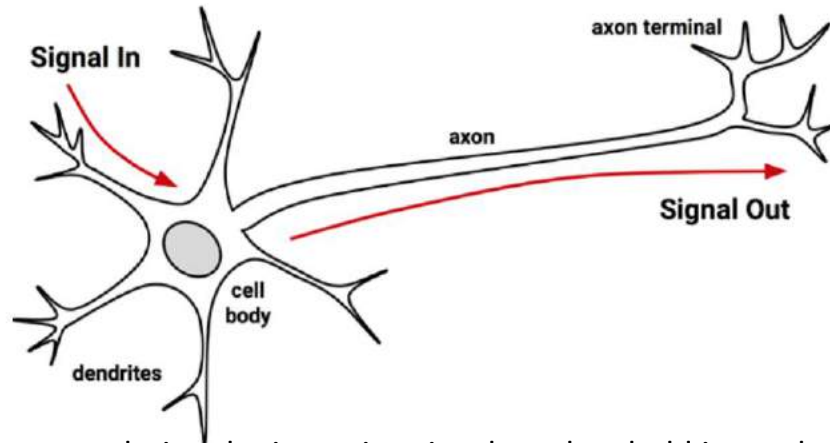
- 2016: Google's AlphaGo program beat Lee Sedol of Korea, a top-ranked international Go player.
 - Developed by DeepMind, AlphaGo uses machine learning and tree search techniques.
- 2017: AlphaGo beat the #1 ranked Go player Ke Jie in the world .

Introduction to Neural Networks

- Neural networks can be applied to many learning tasks
 - Classification
 - Numeric prediction
- Best applied to problems where
 - The input data and output data are well-defined or at least fairly simple
 - The relationship between the input and output is extremely complex.

From Biological to Artificial Neurons

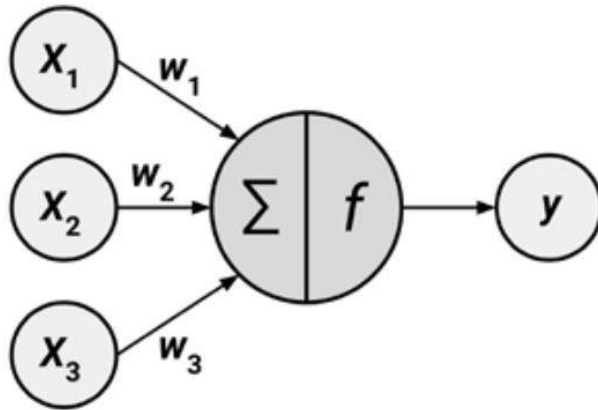
- As illustrated in the following figure, incoming signals are received by the cell's dendrites through a biochemical process.



- As the cell body begins accumulating the incoming signals, a threshold is reached at which the cell fires and the output signal is transmitted via an electrochemical process down the axon.
- At the axon's terminals, the electric signal is again processed as a chemical signal to be passed to the neighboring neurons across a tiny gap known as a synapse.

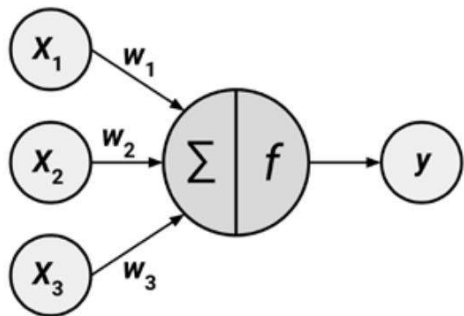
From Biological to Artificial Neurons

- The model of a single artificial neuron can be understood in terms very similar to the biological model.



$$y(x) = f \left(\sum_{i=1}^n w_i x_i \right)$$

From Biological to Artificial Neurons



$$y(x) = f \left(\sum_{i=1}^n w_i x_i \right)$$

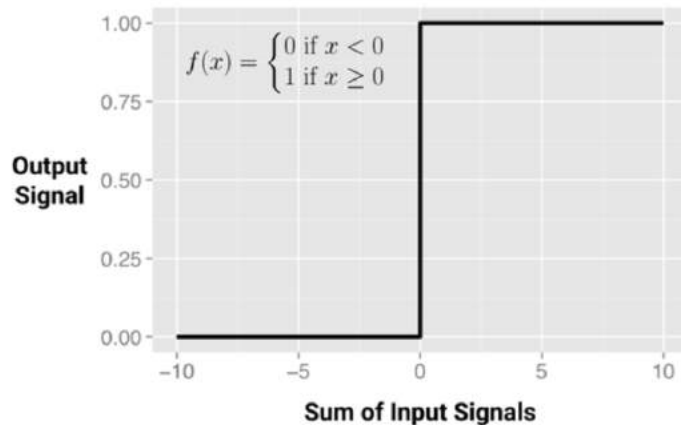
- A relationship between the input signals received by the dendrites (x variables), and the output signal (y variable).
- Each dendrite's signal is weighted (w values) according to its importance.
- The input signals are summed by the cell body and the signal is passed on according to an activation function denoted by f.
- The w weights allow each of the n inputs (denoted by x_i) to contribute a greater or lesser amount to the sum of input signals.
- The net total is used by the activation function $f(x)$, and the resulting signal, $y(x)$, is the output axon.

Understanding Neural Networks

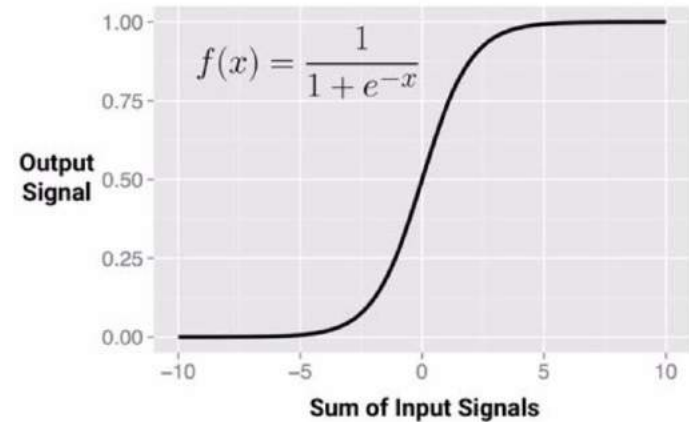
- Neuron: building block to construct complex models of data.
- Characteristics of neural networks:
 - An **activation function**, which transforms a neuron's combined input signals into a single output signal to be broadcasted further in the network
 - A **network topology** (or architecture), which describes the number of neurons in the model as well as the number of layers and manner in which they are connected
 - The **training algorithm** that specifies how connection weights are set in order to inhibit or excite neurons in proportion to the input signal

Activation Function

- The activation function is the mechanism by which the artificial neuron processes incoming information and passes it throughout the network.

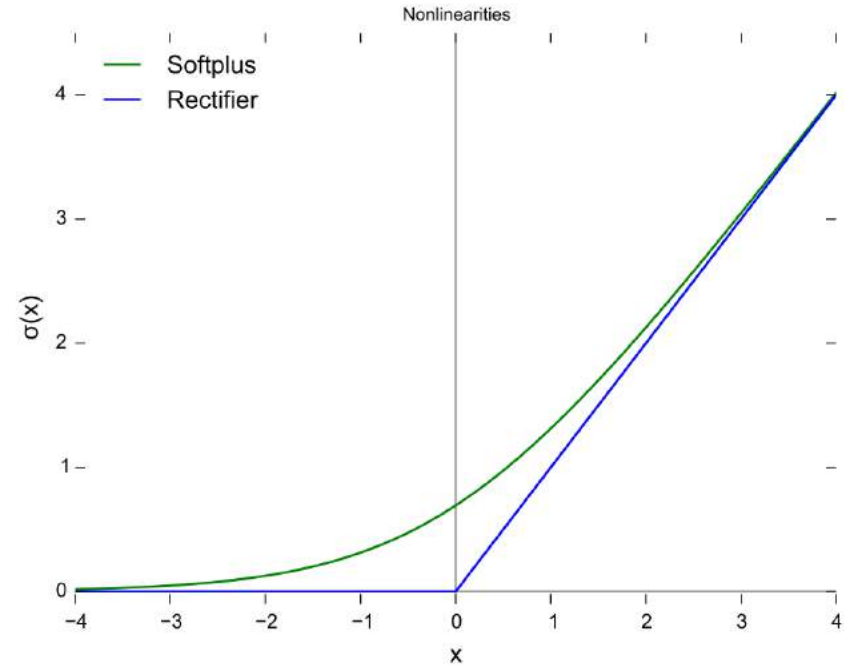
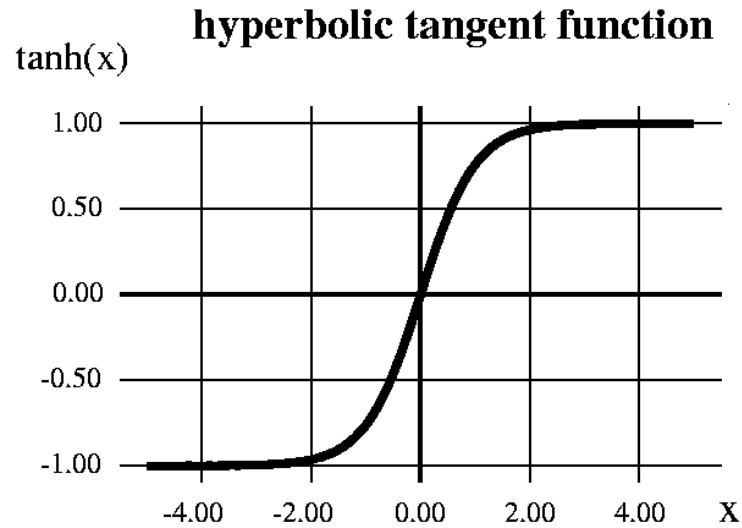


Threshold (unit step) activation function



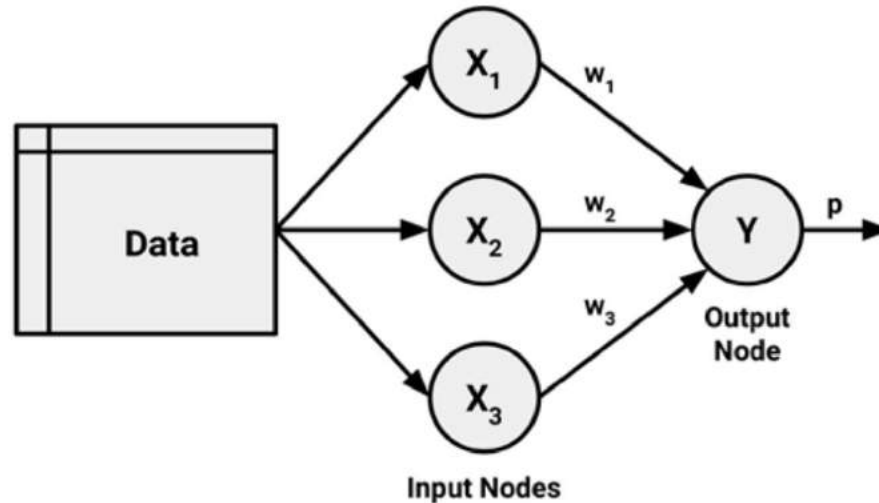
Sigmoid activation function

Activation Function

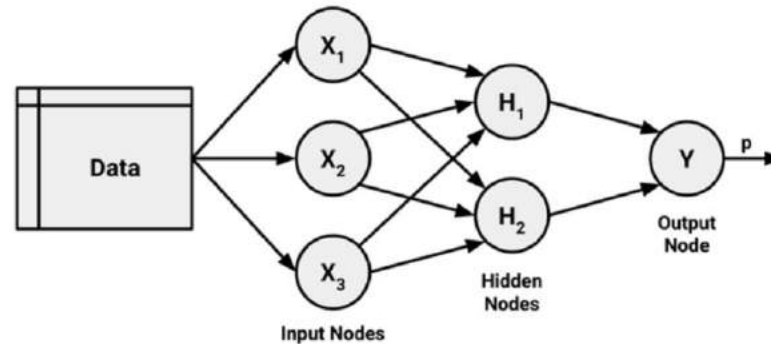


- The ability of a neural network to learn is rooted in its topology, or the patterns and structures of interconnected neurons.
- Although there are countless forms of network architectures, they can be differentiated by three key characteristics:
 - The number of layers
 - Whether information in the network is allowed to travel backward
 - The number of nodes within each layer of the network

- To define topology, we need a terminology that distinguishes artificial neurons based on their position in the network.

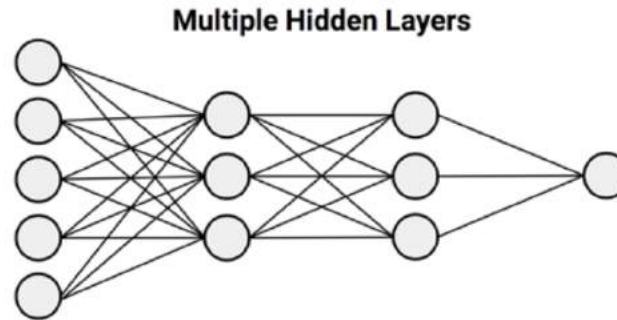
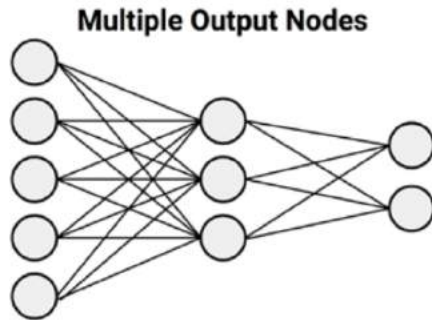


- As you might expect, an obvious way to create more complex networks is by adding additional layers.

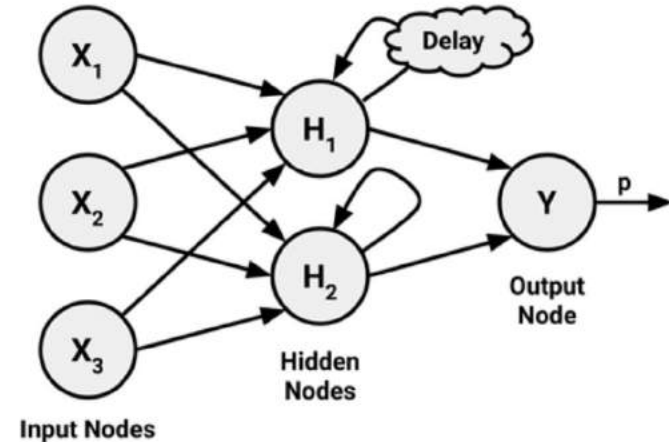


- The multilayer feedforward network, sometimes called the Multilayer Perceptron (MLP).
- A multilayer network adds one or more hidden layers that process the signals from the input nodes prior to it reaching the output node.
- Most multilayer networks are fully connected, which means that every node in one layer is connected to every node in the next layer, but this is not required.

- You may have noticed that in the prior examples, arrowheads were used to indicate signals traveling in only one direction.
- Networks in which the input signal is fed continuously in one direction from connection to connection until it reaches the output layer are called **feedforward** networks.



- In contrast, a **recurrent network** (or **feedback network**) allows signals to travel in both directions using loops.
 - This property, which more closely mirrors how a biological neural network works, allows extremely complex patterns to be learned.
 - The addition of a short-term memory, or delay, increases the power of recurrent networks immensely.



- Neural networks can also vary in complexity by the number of nodes in each layer.
 - The number of input nodes is predetermined by the number of features in the input data.
 - The number of output nodes is predetermined by the number of outcomes to be modeled or the number of class levels in the outcome.
 - The number of hidden nodes is left to the user to decide prior to training the model.

- There is no reliable rule to determine the number of neurons in the hidden layer.
 - The appropriate number depends on the number of input nodes, the amount of training data, the amount of noisy data, and the complexity of the learning task, among many other factors.
 - The best practice is to use the fewest nodes that result in adequate performance in the test dataset.

Training Deep Neural Networks

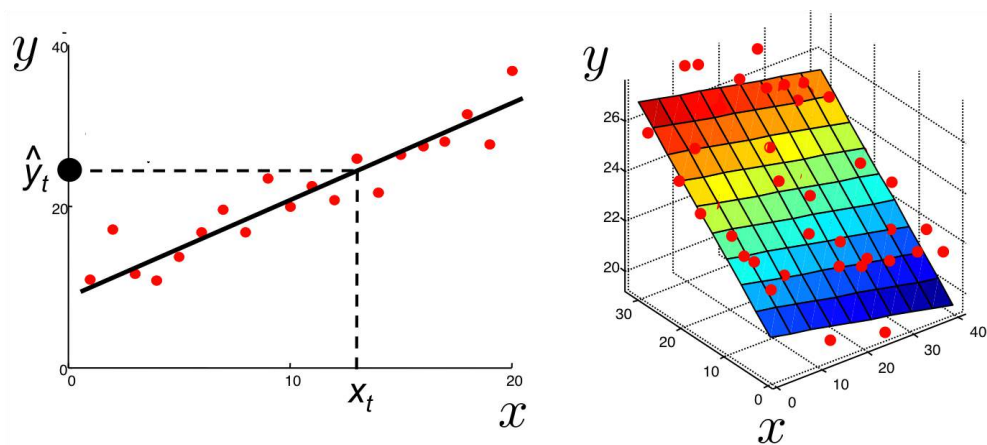
Donghyuk Shin
Dept. of Information Systems

Linear Regression

- Given samples $(x_i, y_i)_{i=1, \dots, N}$, predict y_t given a new test point x_t
- Goal is to estimate \hat{y}_t by a linear function of given data x_t :

$$\hat{y}_t = w_0 + w_1 x_{t,1} + w_2 x_{t,2} + \dots + w_d x_{t,d} = \mathbf{w}^T \mathbf{x}_t$$

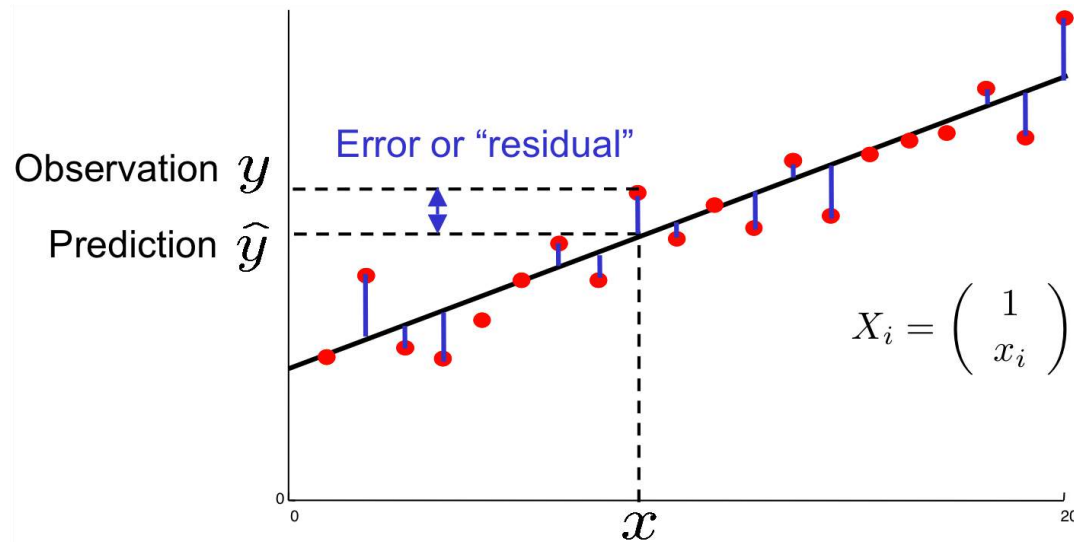
where \mathbf{w} is the **parameter** to be estimated



Least Squares

- Loss function: sum squared error

$$J(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}_i - y_i)^2$$



Normal Equations

- Minimize the sum squared error $J(\mathbf{w})$

$$\begin{aligned} J(\mathbf{w}) &= \frac{1}{2} \sum_{i=1}^N (\mathbf{w}^T \mathbf{x}_i - y_i)^2 = \frac{1}{2} (\mathbf{X}\mathbf{w} - \mathbf{y})^T (\mathbf{X}\mathbf{w} - \mathbf{y}) \\ &= \frac{1}{2} (\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{y}^T \mathbf{X} \mathbf{w} + \mathbf{y}^T \mathbf{y}) \end{aligned}$$

- Derivative: $\frac{\partial}{\partial \mathbf{w}} J(\mathbf{w}) = \mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{X}^T \mathbf{y}$
- Setting the derivative equal to zero gives the **normal equations**:

$$\begin{aligned} \mathbf{X}^T \mathbf{X} \mathbf{w} &= \mathbf{X}^T \mathbf{y} \\ \mathbf{w} &= (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y} \end{aligned}$$

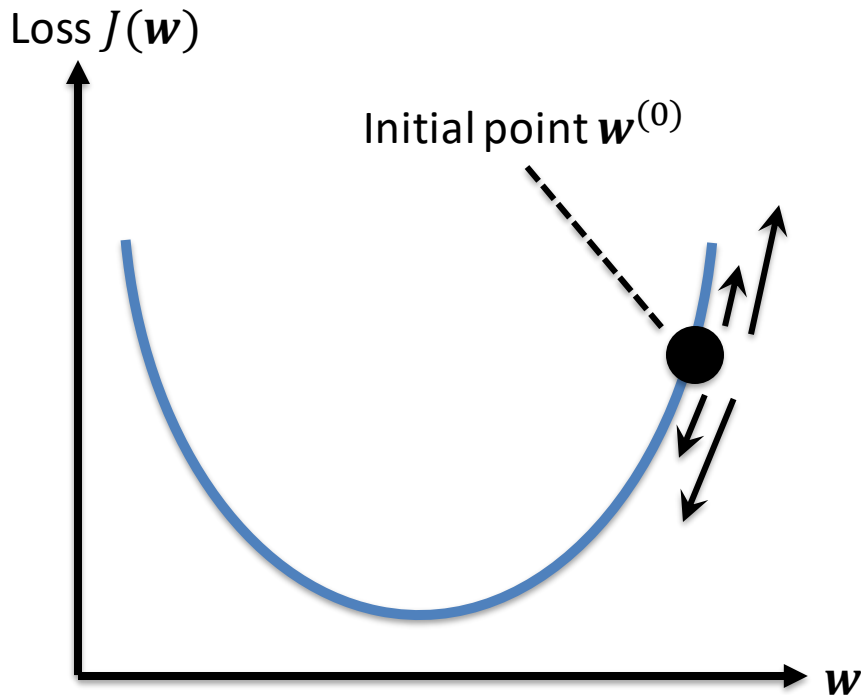
Computing \mathbf{w}

- Large-scale data?
 - Computing $\mathbf{w} = (X^T X)^{-1} X^T \mathbf{y}$ is expensive (matrix inverse): $O(d^3)$
 - Storing $X^T X$ can be prohibitive: $O(d^2)$
- Closed form solution?
 - Deep neural networks?
- Alternatives? \rightarrow Iterative methods
 - Gradient Descent

Gradient Descent

Start somewhere: $w^{(0)}$

- Which way to go?
- How big a step to take?



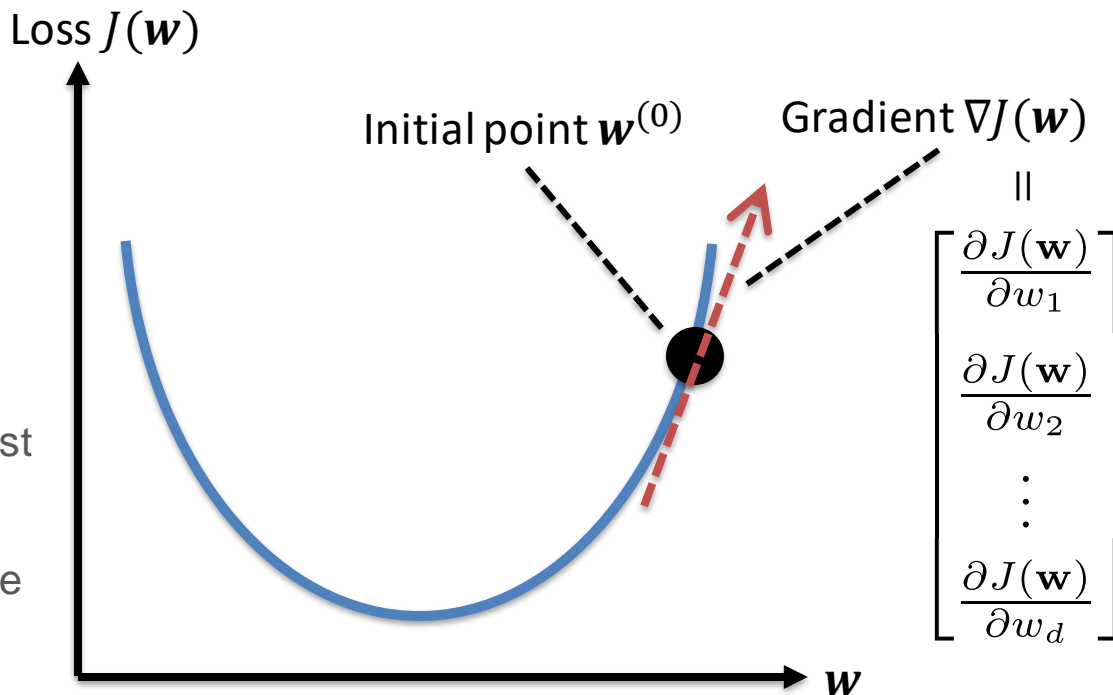
Gradient Descent

Start somewhere: $\mathbf{w}^{(0)}$

- Which way to go?
- How big a step to take?

→ Gradient

- Points in the direction of steepest increase (ascent) of $J(\mathbf{w})$
- Magnitude is the rate of increase



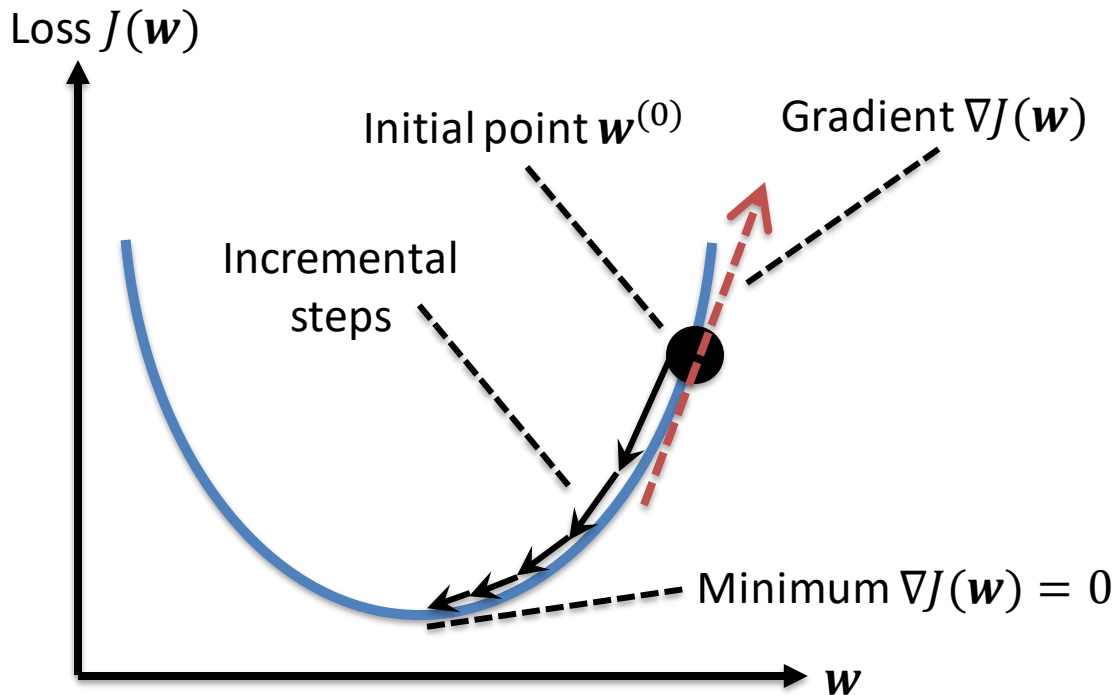
Gradient Descent

Start somewhere: $\mathbf{w}^{(0)}$

1. Compute gradient $\nabla J(\mathbf{w}^{(t)})$
2. Descend according to $\nabla J(\mathbf{w}^{(t)})$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla J(\mathbf{w}^{(t)})$$

Repeat until convergence



Gradient Descent – Neural Networks

Start somewhere: $\mathbf{w}^{(0)}$

1. Compute gradient $\nabla J(\mathbf{w}^{(t)})$  Back-propagation

Algorithm to compute the gradient of $J(\mathbf{w})$

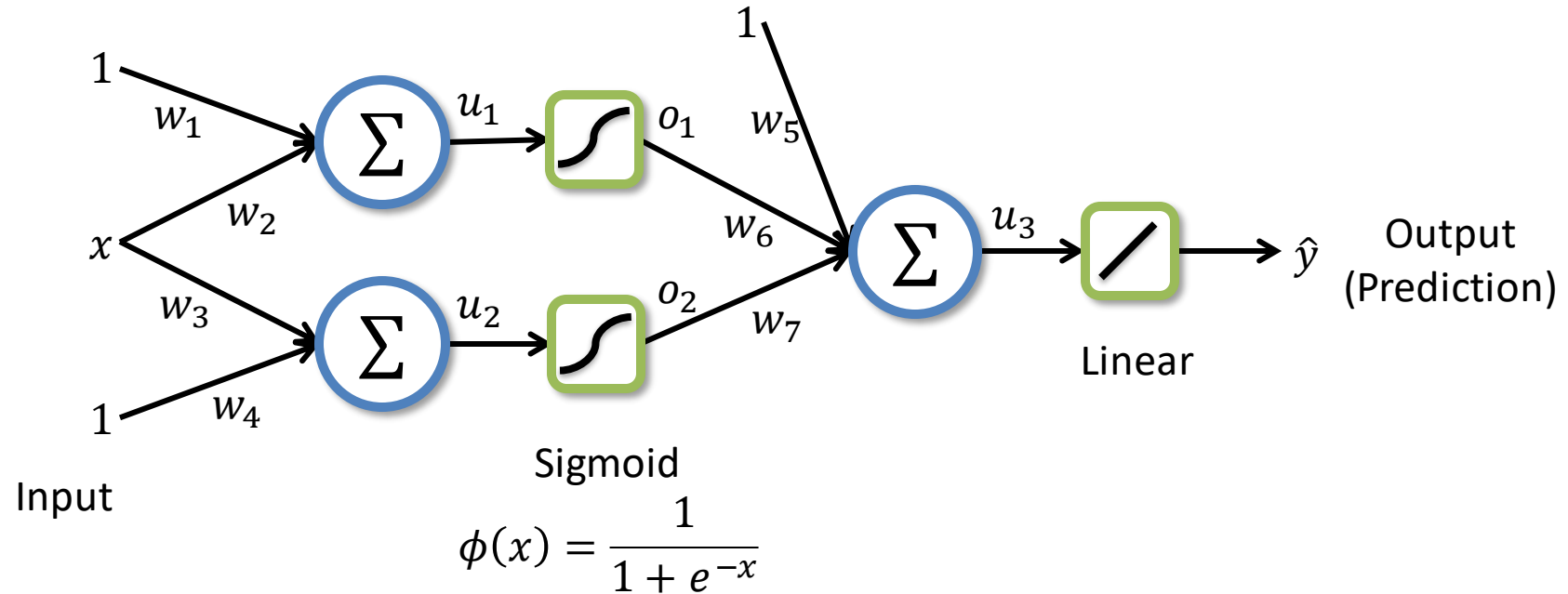
2. Descend according to $\nabla J(\mathbf{w}^{(t)})$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla J(\mathbf{w}^{(t)})$$

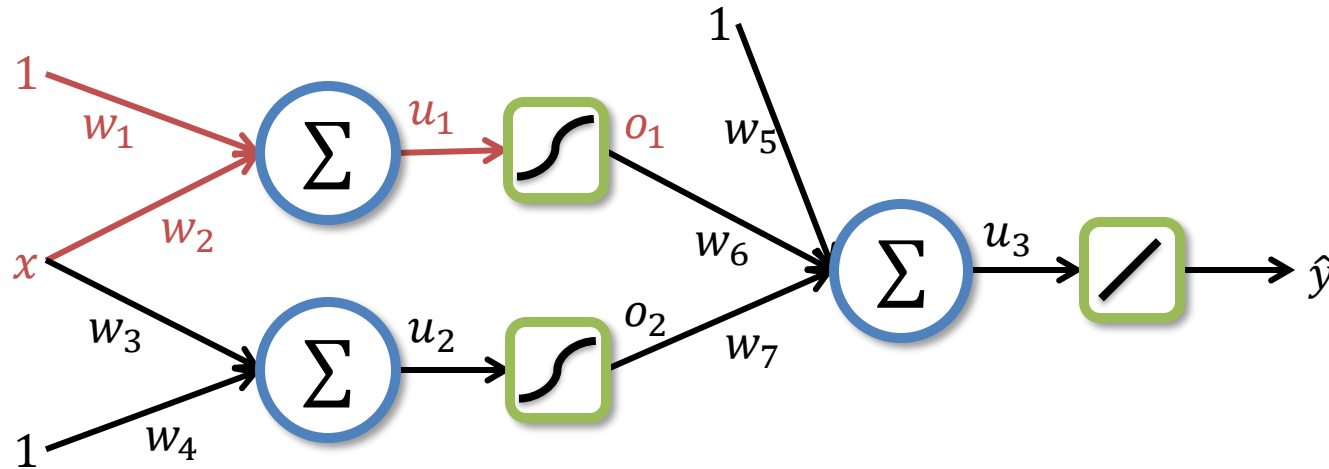
Repeat until convergence

$$\nabla J(\mathbf{w}) = \begin{bmatrix} \frac{\partial J(\mathbf{w})}{\partial w_1} \\ \frac{\partial J(\mathbf{w})}{\partial w_2} \\ \vdots \\ \frac{\partial J(\mathbf{w})}{\partial w_d} \end{bmatrix}$$

Example Neural Network

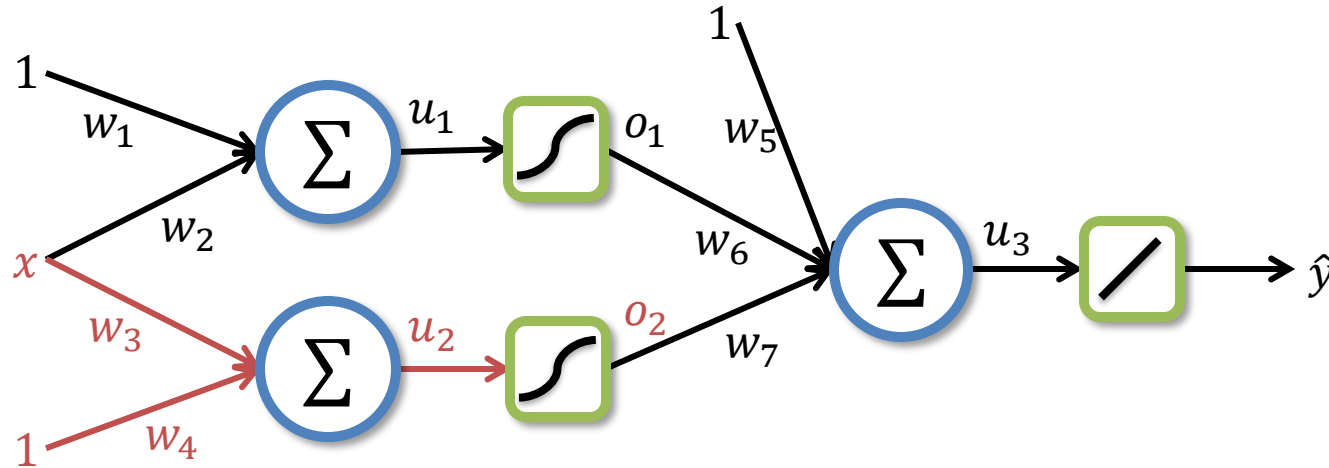


Feed Forward



$$o_1 = \frac{1}{1+e^{-u_1}}, \text{ where } u_1 = w_1 + w_2x$$

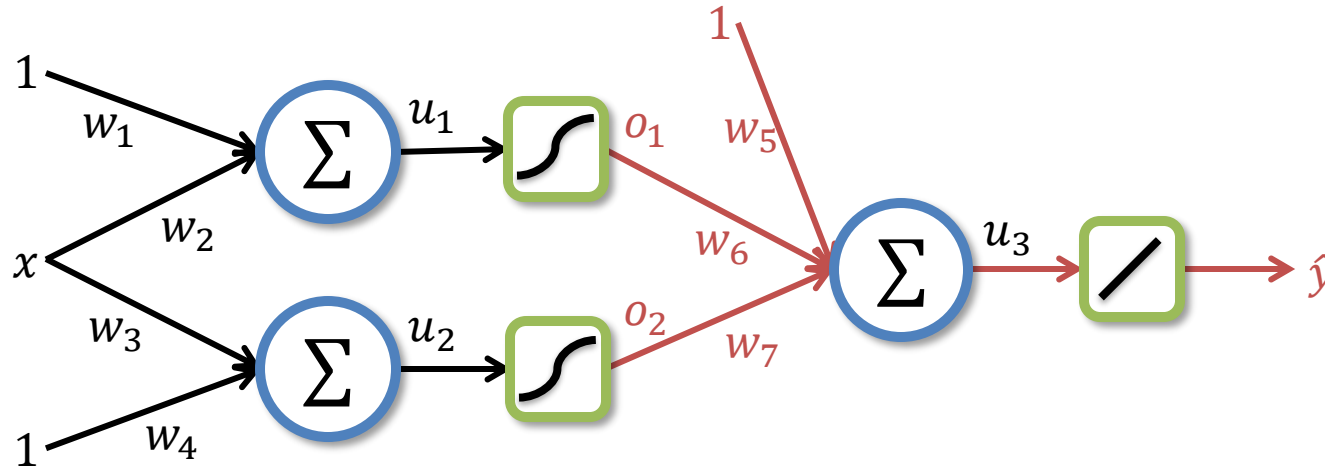
Feed Forward



$$o_1 = \frac{1}{1+e^{-u_1}}, \text{ where } u_1 = w_1 + w_2x$$

$$o_2 = \frac{1}{1+e^{-u_2}}, \text{ where } u_2 = w_4 + w_3x$$

Feed Forward

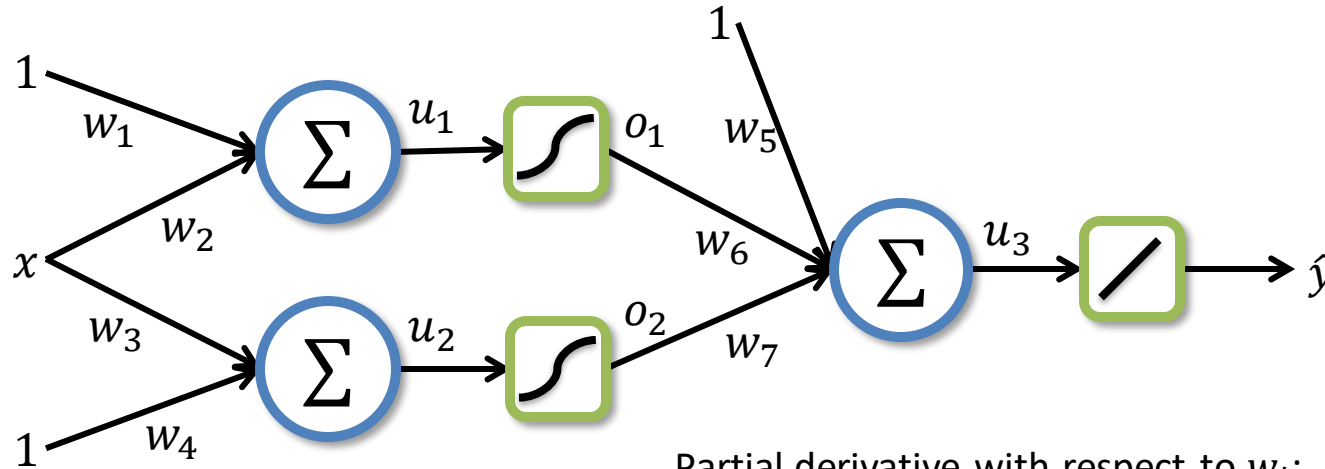


$$o_1 = \frac{1}{1+e^{-u_1}}, \text{ where } u_1 = w_1 + w_2x$$

$$o_2 = \frac{1}{1+e^{-u_2}}, \text{ where } u_2 = w_4 + w_3x$$

$$\hat{y} = w_5 + w_6o_1 + w_7o_2$$

Back-propagation

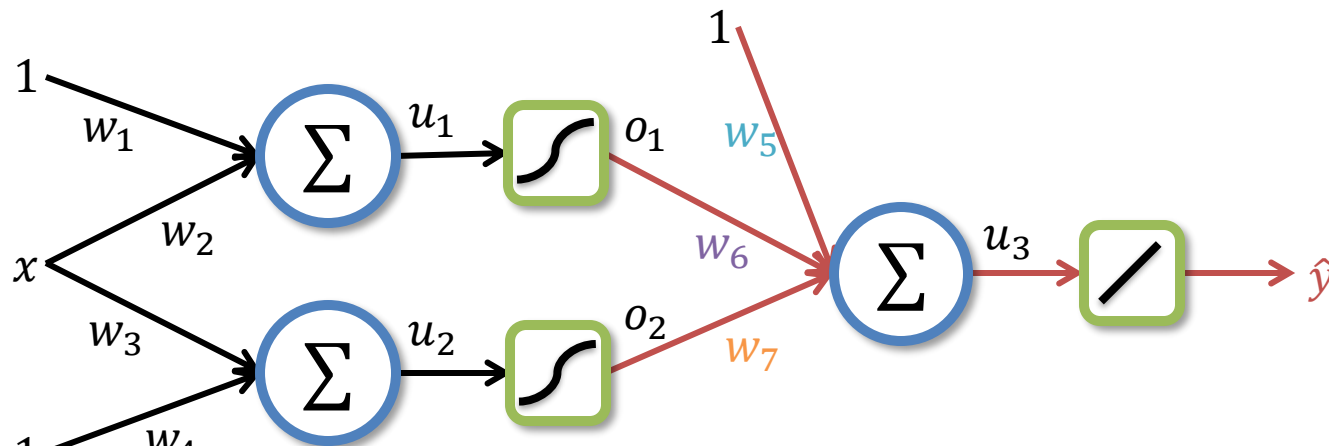


Loss: $J(\mathbf{w}) = \frac{1}{2} (y - \hat{y}(x, \mathbf{w}))^2$

Partial derivative with respect to w_i :

$$\frac{\partial J(\mathbf{w})}{\partial w_i} = -(y - \hat{y}) \frac{\partial \hat{y}}{\partial w_i}$$

Back-propagation



Loss: $J(\mathbf{w}) = \frac{1}{2} (y - \hat{y}(x, \mathbf{w}))^2$

Partial derivative with respect to w_i :

$$\frac{\partial J(\mathbf{w})}{\partial w_i} = -(y - \hat{y}) \frac{\partial \hat{y}}{\partial w_i}$$

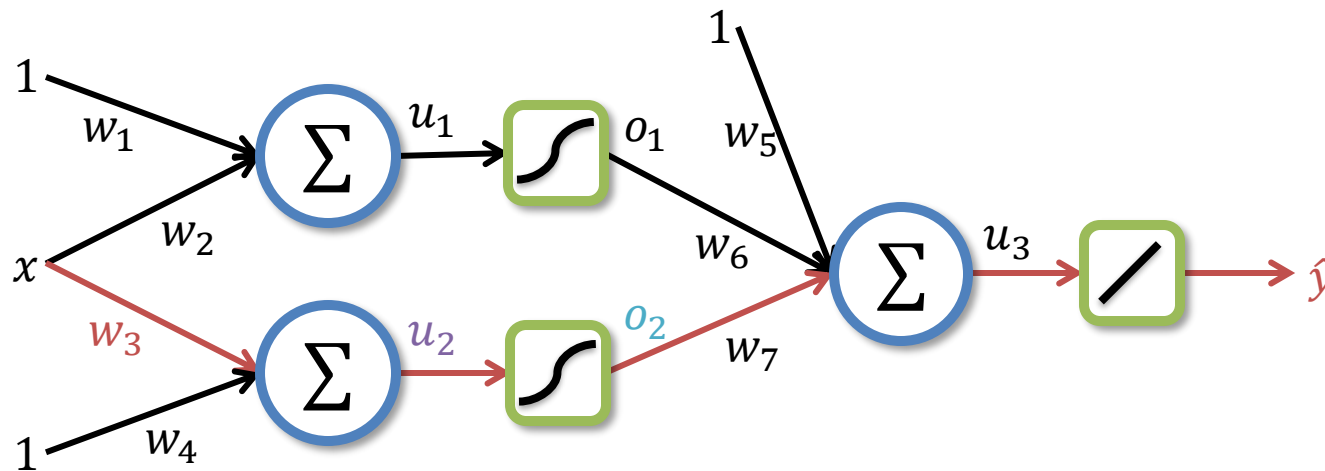
$$\hat{y} = w_5 + w_6 o_1 + w_7 o_2$$

$$\frac{\partial \hat{y}}{\partial w_5} = 1$$

$$\frac{\partial \hat{y}}{\partial w_6} = o_1$$

$$\frac{\partial \hat{y}}{\partial w_7} = o_2$$

Back-propagation

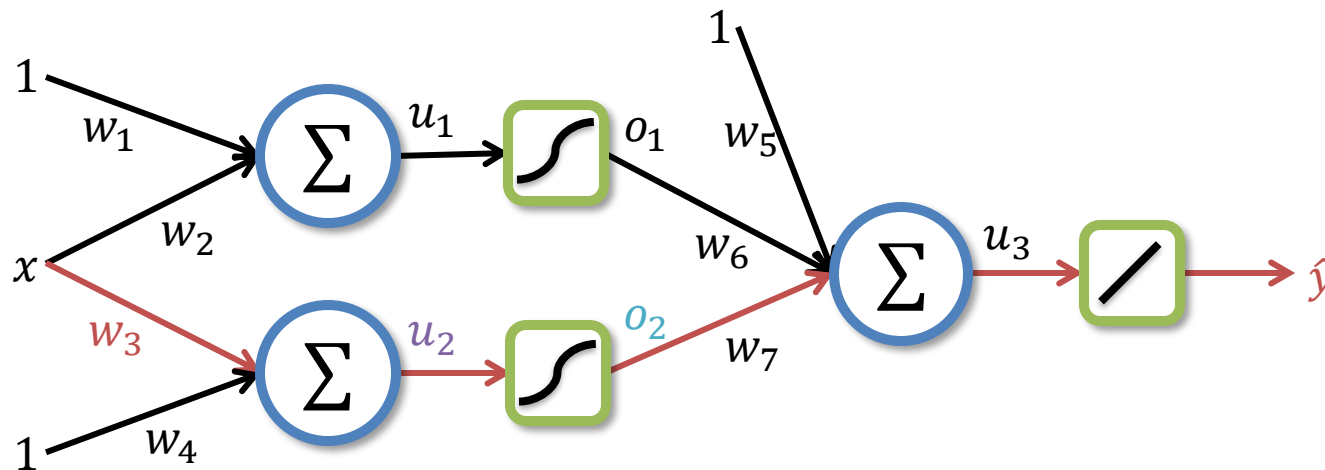


$$\frac{\partial \hat{y}}{\partial w_3} = ?$$

$$\hat{y} = w_5 + w_6 o_1 + w_7 o_2$$

$$o_2 = \frac{1}{1+e^{-u_2}}, \text{ where } u_2 = w_4 + w_3 x$$

Back-propagation

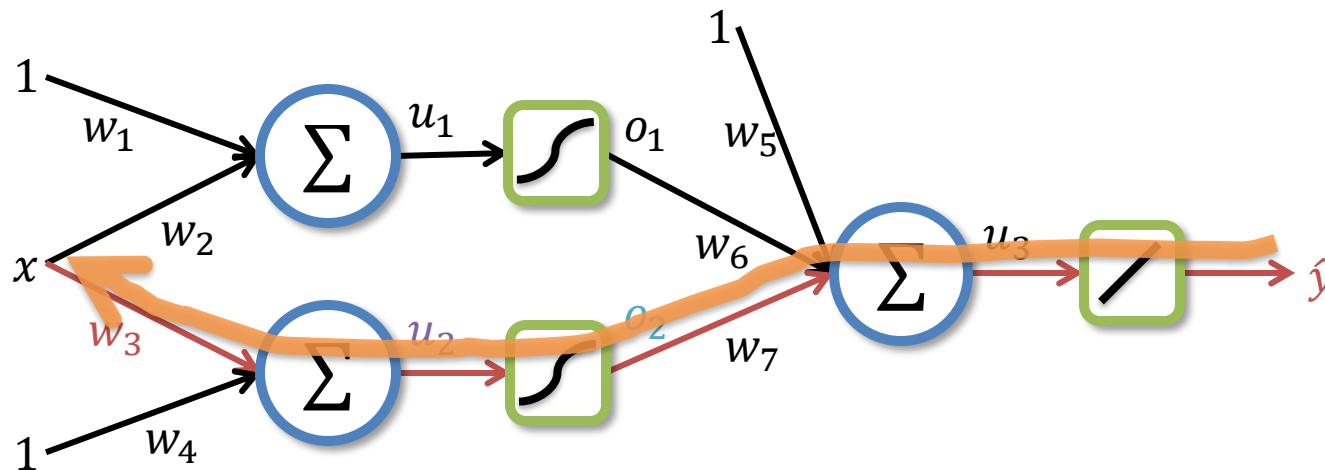


$$\hat{y} = w_5 + w_6 o_1 + w_7 o_2$$

$$\frac{\partial \hat{y}}{\partial w_3} = \frac{\partial \hat{y}}{\partial o_2} \frac{\partial o_2}{\partial u_2} \frac{\partial u_2}{\partial w_3} \quad \text{Chain rule!}$$

$$o_2 = \frac{1}{1+e^{-u_2}}, \text{ where } u_2 = w_4 + w_3 x$$

Back-propagation

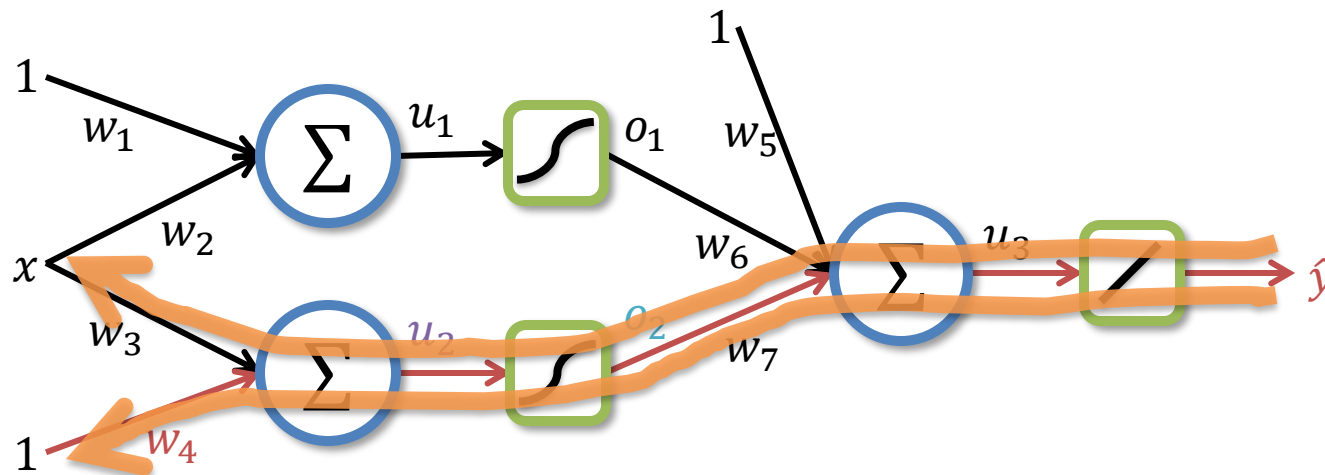


$$\hat{y} = w_5 + w_6 o_1 + w_7 o_2$$

$$o_2 = \frac{1}{1+e^{-u_2}}, \text{ where } u_2 = w_4 + w_3 x$$

$$\begin{aligned} \frac{\partial \hat{y}}{\partial w_3} &= \frac{\partial \hat{y}}{\partial o_2} \frac{\partial o_2}{\partial u_2} \frac{\partial u_2}{\partial w_3} \quad \text{Chain rule!} \\ &= w_7 o_2 (1 - o_2) x \end{aligned}$$

Back-propagation

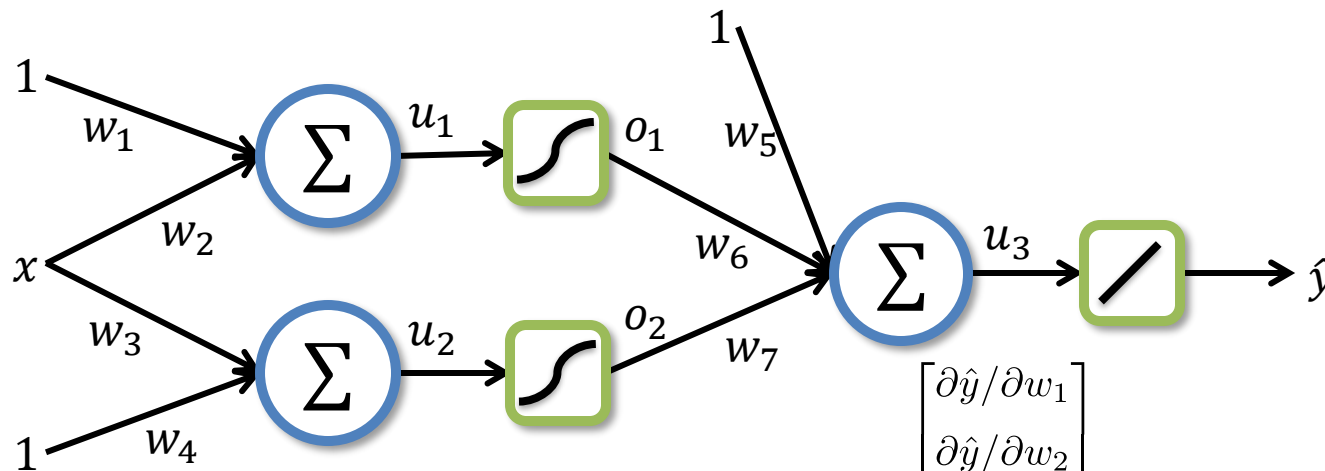


$$\hat{y} = w_5 + w_6 o_1 + w_7 o_2$$

$$o_2 = \frac{1}{1+e^{-u_2}}, \text{ where } u_2 = w_4 + w_3 x$$

$$\begin{aligned} \frac{\partial \hat{y}}{\partial w_4} &= \frac{\partial \hat{y}}{\partial o_2} \frac{\partial o_2}{\partial u_2} \frac{\partial u_2}{\partial w_4} \quad \text{Chain rule!} \\ &= w_7 o_2 (1 - o_2) 1 \end{aligned}$$

Back-propagation



Full gradient vector

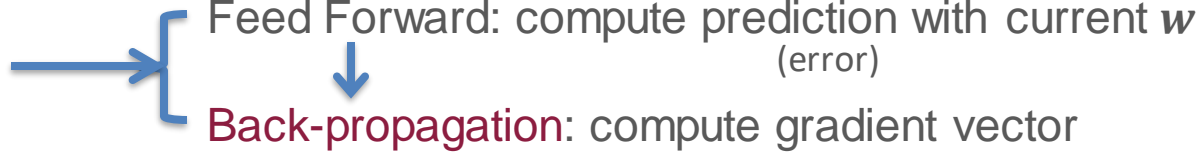
$$\text{Loss: } J(\mathbf{w}) = \frac{1}{2} (y - \hat{y}(x, \mathbf{w}))^2$$

$$\nabla J(\mathbf{w}) = -(y - \hat{y}) \begin{bmatrix} \partial \hat{y} / \partial w_1 \\ \partial \hat{y} / \partial w_2 \\ \partial \hat{y} / \partial w_3 \\ \partial \hat{y} / \partial w_4 \\ \partial \hat{y} / \partial w_5 \\ \partial \hat{y} / \partial w_6 \\ \partial \hat{y} / \partial w_7 \end{bmatrix} = -(y - \hat{y}) \begin{bmatrix} w_6 o_1 (1 - o_1) 1 \\ w_6 o_1 (1 - o_1) x \\ w_7 o_2 (1 - o_2) x \\ w_7 o_2 (1 - o_2) 1 \\ 1 \\ o_1 \\ o_2 \end{bmatrix}$$

$$\frac{\partial J(\mathbf{w})}{\partial w_i} = -(y - \hat{y}) \frac{\partial \hat{y}}{\partial w_i}$$

Gradient Descent – Neural Networks

Start somewhere: $\mathbf{w}^{(0)}$

1. Compute gradient $\nabla J(\mathbf{w}^{(t)})$ 
2. Descend according to $\nabla J(\mathbf{w}^{(t)})$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla J(\mathbf{w}^{(t)})$$

Repeat until convergence

Gradient Descent – Learning Rate

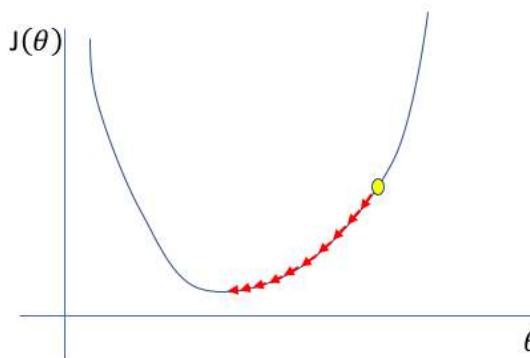
Start somewhere: $\mathbf{w}^{(0)}$

1. Compute gradient $\nabla J(\mathbf{w}^{(t)})$
2. Descend according to $\nabla J(\mathbf{w}^{(t)})$

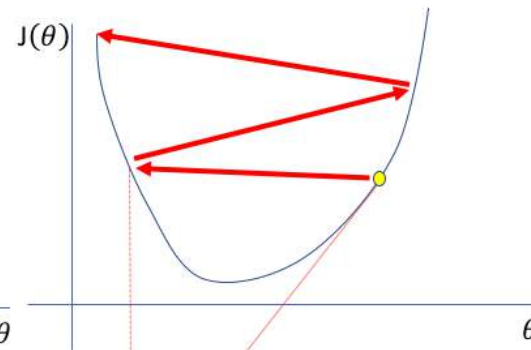
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla J(\mathbf{w}^{(t)})$$

Repeat until convergence

Learning rate α



A small learning rate
requires many updates
before reaching the
minimum point

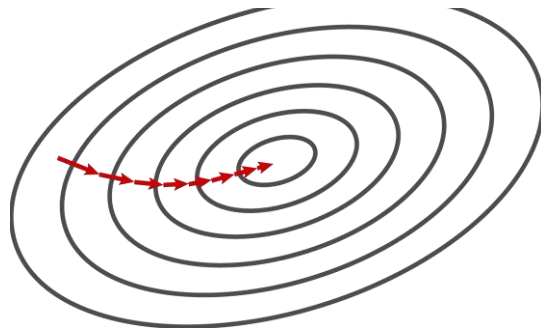


Too large of a learning rate
causes drastic updates
which lead to divergent
behaviors

Gradient Descent – Batch Size

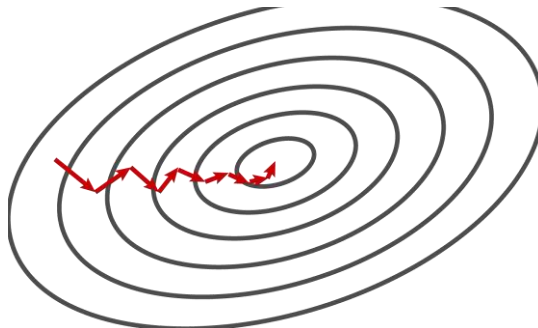
Batch Gradient Descent

- Use **entire** training data per iteration
- Needs very large memory
- Slow per iteration, but may need less iterations



Mini-batch Gradient Descent

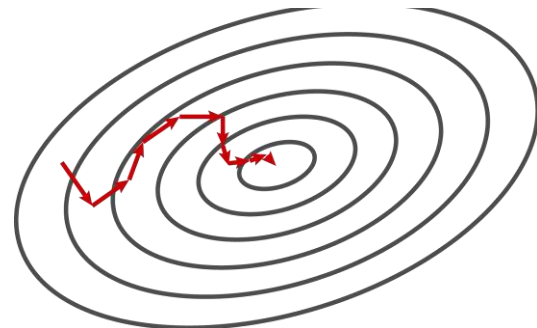
- Use **m** samples from training data per iteration
- Similar to SGD, but better utilizes GPU/multi-cores
- $m = 8, 16, 32, 64, \dots, 1024$



Stochastic Gradient Descent (SGD)

- Use **one** sample from training data per iteration
- Fast per iteration, but may require many iterations

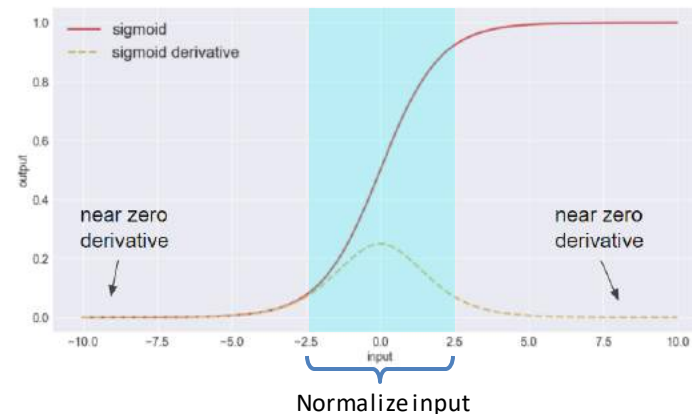
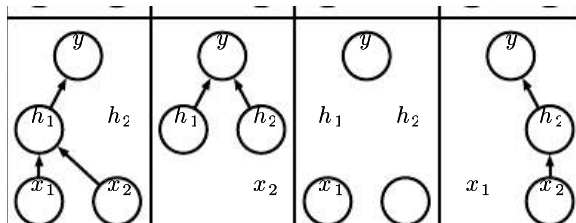
(Going over all training data = 1 epoch)



Some Issues with Deep Neural Nets

Vanishing / Exploding Gradient

- Gradients at layers close to the input can become *very, very small* or *large*
 - Initialization with small $w^{(0)}$
 - Change learning rate
 - Architectural decisions (e.g., ReLU, skip connections)
 - Appropriate regularizations
 - Gradient clipping (bounding)
 - Batch normalization
 - Dropouts

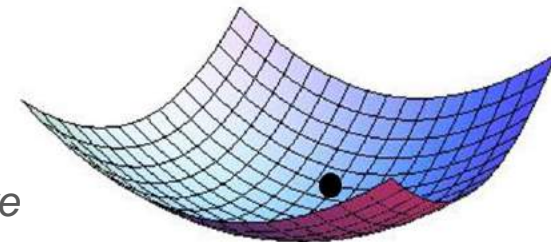


Some Issues with Deep Neural Nets

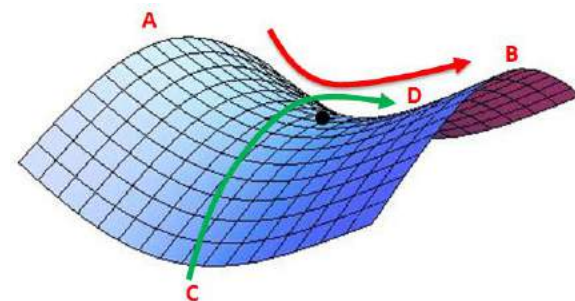
Local Minima

- Always a concern, but it turns out that local minima is less of an issue (both theoretically and experimentally)
 - Large DNNs are so high-dimensional that minima with no direction decreasing the loss function are *exponentially rare*
- *Saddle points* can be of bigger concern
 - Randomness in the gradient helps: SGD, Mini-batch
 - Advanced gradient descent algorithms:
Adagrad, RMSprop, Adam, Nadam, ...

(local) Minima



Saddle Point



Python Example with Keras

```
from keras.models import Sequential
from keras.layers import Dense
from keras import optimizers

# Define the keras model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the keras model
opt = optimizers.SGD(learning_rate=0.01)
model.compile(loss='mean_squared_error', optimizer=opt, metrics=['mse'])

# Fit the keras model on the dataset
model.fit(X, y, epochs=100, batch_size=16)

# Evaluate the keras model
mse = model.evaluate(X, y)
print('MSE: {:.4f}'.format(mse))
```

Advanced Neural Architectures

Victor Benjamin
Dept. of Information Systems

Advanced Neural Architectures

- Today we introduced foundational concepts of neural networks
- Many advanced architectures exist that can each augment the behavior of neural networks to enhance their performance for different problem contexts
- We will introduce such architectures today with intention to dedicate future study group sessions on exploring them more deeply

Recurrent Neural Networks

What they can do: Learn from temporal sequence of data

Example:

Assume the sentence “I went to Japan last summer. I ate a lot of _____ food.”

Your mind computes a probability distribution of words that can fit the blank. Among the most probable solutions is the word “Japanese.”

Recurrent neural networks can use previously learned information to boost their performance for predicting the current task. Traditional neural networks cannot do this.

How it's commonly used: Natural language processing, sequence mining

Convolutional Neural Networks

What they can do: Learn from visual data

Example:

Convolutional neural networks take inspiration from the brain structure called the visual cortex. The visual cortex possesses small regions of cells that are sensitive to specific regions of the visual field. It has been discovered that these clusters of neurons fire systematically in the presence of visual stimuli.

For a given cluster of neurons, individual neurons within that cluster will fire only in the presence of edges of a certain orientation. That is, some neurons fire only when exposed to a vertical edge, some when shown a horizontal edge or diagonal edge. This idea of neurons having specialized tasks is the basis behind CNNs.

How it's commonly used: Image mining, audio mining

Reinforcement Learning

What they can do: Maximize success of an objective that requires many steps

Example:

Imagine you have a pet mouse. You can not communicate with the mouse in human language or otherwise directly tell it what to do. You have to follow a different strategy to communicate.

To teach the mouse a trick, you create a scenario and hope the mouse responds in the desired manner. Each time the mouse responds correctly, you reward it with a cookie.

Over time, the mouse will learn what sequence of actions to execute to be rewarded with a cookie. At the same time, the mouse will learn what sequence of actions are not rewarded.

How it's commonly used: Attaining complex objectives such as navigating environments

Generative Adversarial Networks

What they can do: Generate new data that has similar characteristics to input training data

Example:

Two neural networks train together, with one being the generator and the other being the discriminator. For the sake of this example, let's consider the generator as a “counterfeiter” and the discriminator as the “police.”

The counterfeiter takes examples of real currency and continuously makes fake copies. The police detect the fake copies. Over time the counterfeiter learns how to improve their fakes based on trial-and-error of what gets caught by the police. When the police is no longer able to distinguish between what is real and what is fake, the counterfeiter has won.

Generative adversarial networks operate in the same fashion.

How it's commonly used: Image synthesis

Explainable AI

What they can do: Open the AI “black box” and become more interpretable by humans

Example:

A society where cooperation between AI and humans can occur is dependent on trust. If humans are to fully trust the capabilities of AI such as self-driving cars, personalized medicine, manufacturing, etc., they must possess the capability to understand an AI’s reasoning and logic behind its actions.

Further, explainability of AI will become critical for various facets of society to operate. Imagine a lawsuit involving some AI-enabled application; telling a court that you did something simply because your model told you to do so would be a laughable defense.

How it’s commonly used: AI transparency, explainable “human-in-the-loop” capabilities

Concluding today's session

- First ever WPCAI Study Group! Very cool 😊
- This is an awesome community, let's keep it going
- An email containing these slides and the survey will follow
- If you want to present a topic, let us know!

Thank you!

<https://research.wpcarey.asu.edu/actionable-analytics/>