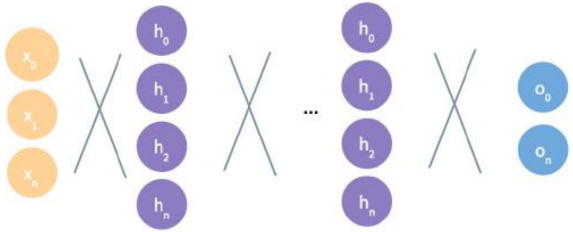
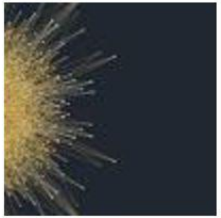
Intro to Deep Learning

Nick Locascio



2016: year of deep learning



[2016: The Year That **Deep Learning** Took Over the Internet]
WIRED - Dec 25, 2016
The project is still in the early stages, but it hints at the widespread impact of **deep learning** over past year. In 2016, this very old but newly ...

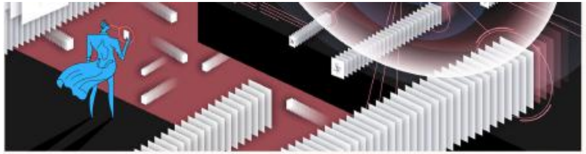


Illustration by Sandbox Studio, Chicago with Ana Kova

Deep learning takes on physics

12/06/16 | By Molly Olmstead

Can the same type of technology Facebook uses to recognize faces also recognize particles?

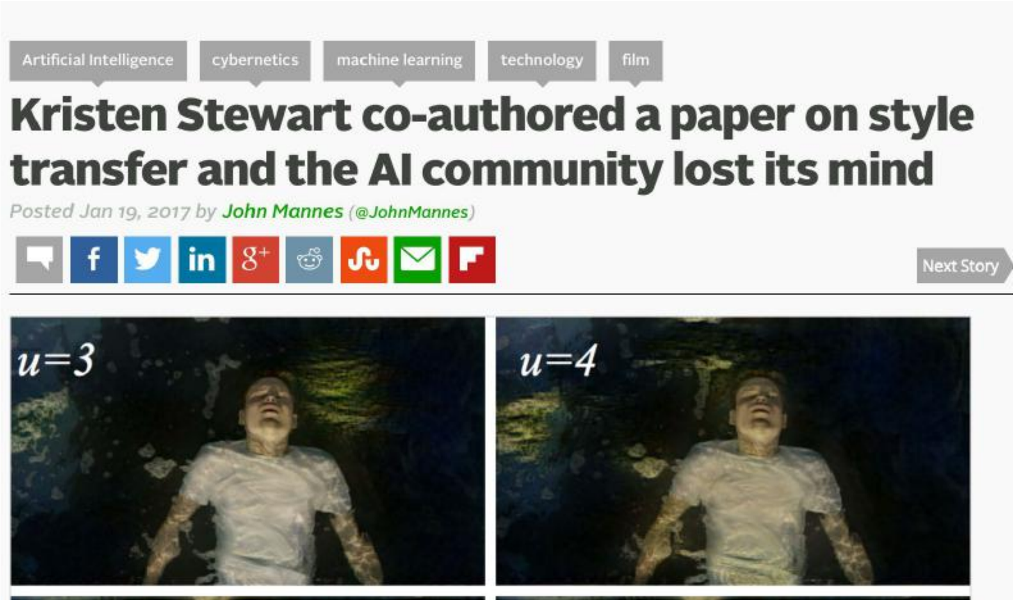
BuzzFeedNEWS

News Videos Quizzes Tasty DIY More Get Our New

This Is Why A Computer Winning At Go Is Such A Big Deal

People didn't think this would happen for at least 10 years; it's a sign of how far artificial intelligence has come.

posted on Mar. 14, 2016, at 8:46 a.m.



Artificial Intelligence cybernetics machine learning technology film

Kristen Stewart co-authored a paper on style transfer and the AI community lost its mind

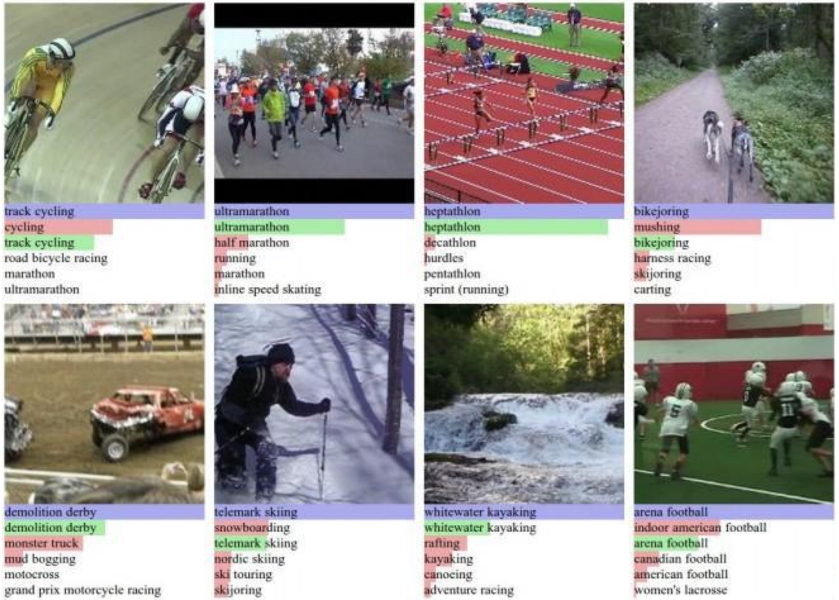
Posted Jan 19, 2017 by John Mannes (@JohnMannes)

Next Story

Deep Learning Success

- **Image Classification**

Machine Translation
Speech Recognition
Speech Synthesis
Game Playing

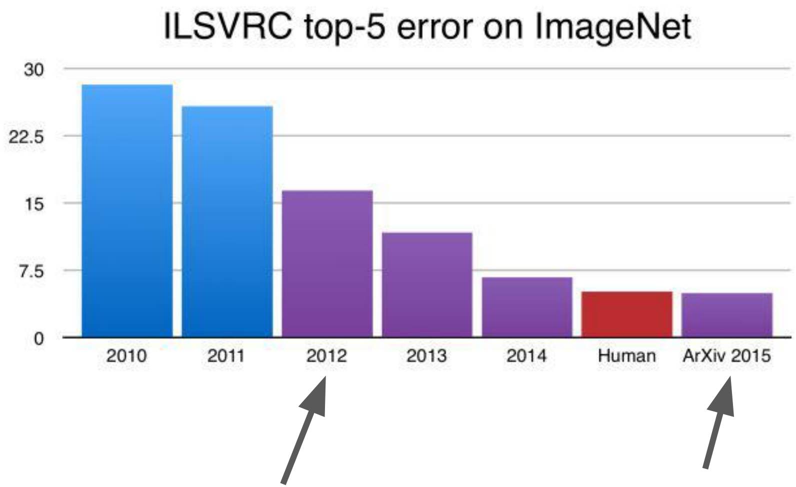


… and many, many more

Deep Learning Success

- Image Classification
Machine Translation
Speech Recognition
Speech Synthesis
Game Playing

... and many, many more

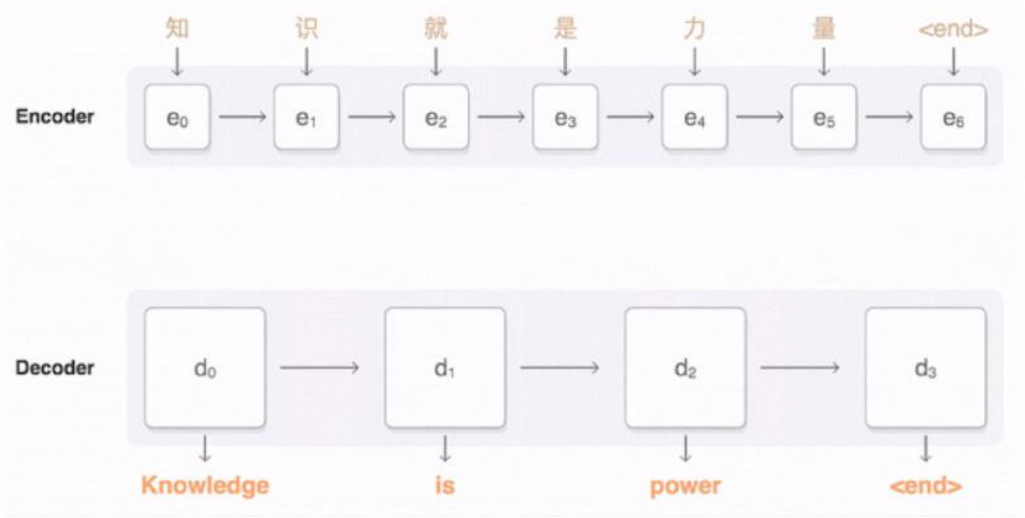


AlexNet
Krizhevsky, Sutskever, Hinton 2012

Better than humans

Deep Learning Success

Image Classification
- Machine Translation
Speech Recognition
Speech Synthesis
Game Playing



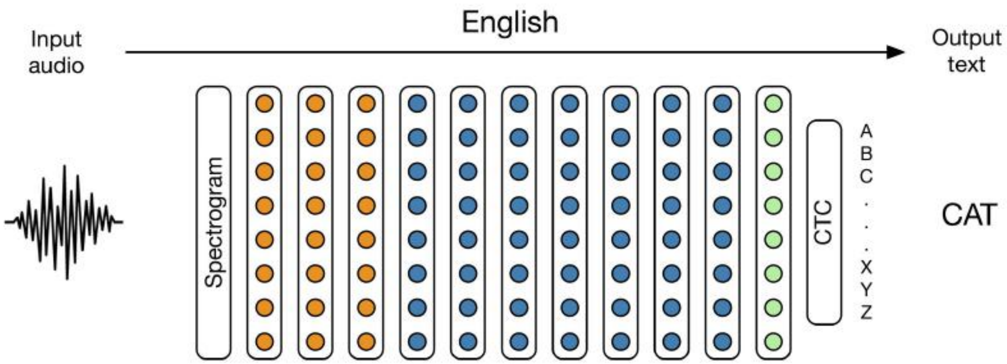
.

… and many, many more

Deep Learning Success

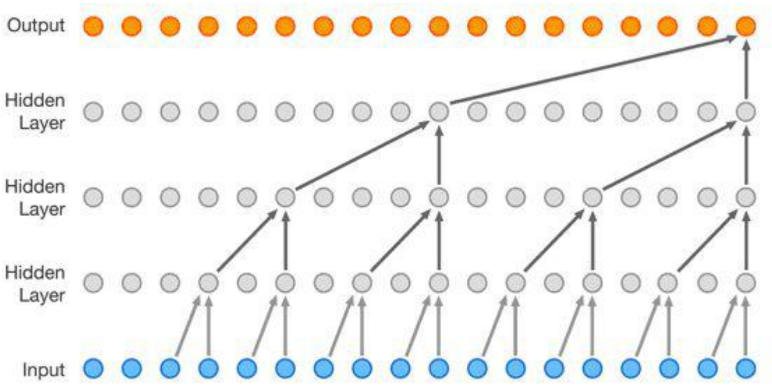
Image Classification
Machine Translation
- Speech Recognition
Speech Synthesis
Game Playing

.
… and many, many more



Deep Learning Success

Image Classification
Machine Translation
Speech Recognition
- Speech Synthesis
Game Playing



.
… and many, many more

Deep Learning Success

Image Classification
Machine Translation
Speech Recognition
Speech Synthesis
- Game Playing

.
… and many, many more

Deep Learning Success

Image Classification
Machine Translation
Speech Recognition
Speech Synthesis
- Game Playing

.
… and many, many more

6.S191 Goals

1. Fundamentals
2. Practical skills
3. Up to speed on current state of the field
4. Foster an open and collaborative deep learning community within MIT

Knowledge, intuition, know-how, and community to do deep learning research and development.

Class Information

- 1 week, 5 sessions
- P/F, 3 credits
- 2 TensorFlow Tutorials
  - In-class Monday + Tuesday
- 1 Assignment: (more info in a few slides)

Typical Schedule

● 10:30am-11:15am	**Lecture #1**
● 11:15am-12:00pm	**Lecture #2**
● 12:00pm-12:30pm	**Coffee Break**
● 12:30pm-1:30pm	**Tutorial / Proposal Time**

Assignment Information

- 1 Assignment, 2 options:
  - Present a novel deep learning research idea or application
  - **OR**
  - Write a 1-page review of a deep learning paper

Option 1: Novel Proposal

- **Proposal Presentation**
    - Groups of 3 or 4
    - Present a novel deep learning research idea or application
    - 1 slide, 1 minute
    - List of example proposals on website: introtodeeplearning.com
    - Presentations on Friday
    - Submit groups by **Wednesday 5pm** to be eligible
    - Submit slide by **Thursday 9pm** to be eligible

Option 2: Paper Review

- Write a 1-page review of a deep learning paper
    - Suggested papers listed on website introtodeeplearning.com
    - We will read + grade based on clarity of writing and technical communication of main ideas.

Class Support

- **Piazza:** https://piazza.com/class/iwmlwep2fnd5uu
- **Course Website:** introtodeeplearning.com
- **Lecture slides:** introtodeeplearning.com/schedule
- **Email us:** introtodeeplearning-staff@mit.edu
- OH by request

Staff: Lecturers

Nick Locascio
Lead Organizer

Harini Suresh
Lead Organizer

Ishaan Gulrajani
Co-Chair

Victoria Dean
Co-Chair

Lex Fridman
Co-Chair

Yo Shavit
Co-Chair

Staff: TA + Admin

Eduardo DeLeon

Jackie Xu
TA

Wengong Jin
TA

Adam Yala
TA

Harry Bleyan
TA

Tianxiao Shen
TA

Alex Lenail
TA

Rue Park
Marketing

Anish Athalye

Helen Zhou
TA

Bowen Baker
TA

Prafulla Dhariwal
TA

Alfredo Yanez
TA

Hansa Srinivasan
TA

Our Fantastic Sponsors!

Google

NVIDIA®

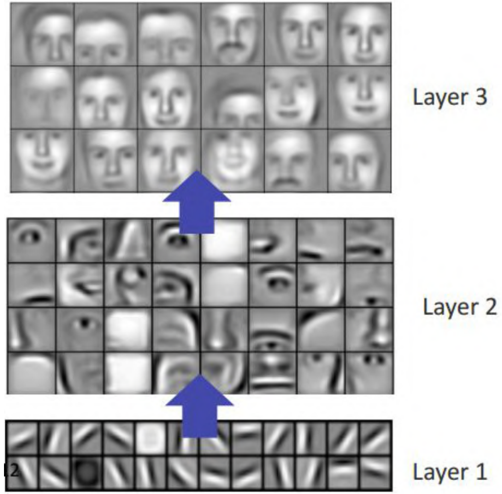
IBM Watson

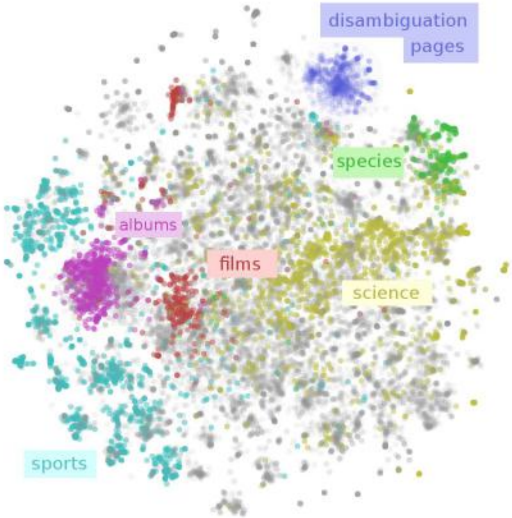
amazon alexa

Why Deep Learning and why now?

Why Deep Learning?

- Hand-Engineered Features vs. Learned features



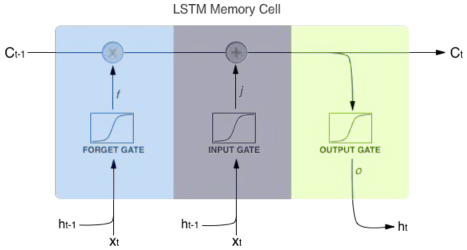


Why Now?

1. Large Datasets
2. GPU Hardware Advances + Price Decreases
3. Improved Techniques

Inception 7a

[1]Going Deeper with Convolutions, [C. Szegedy et al, CVPR 2015]

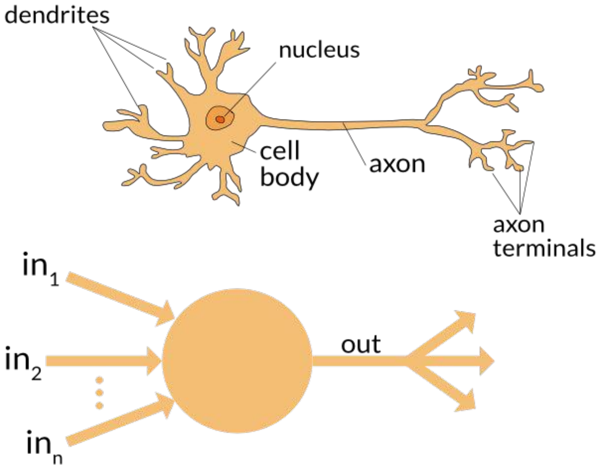


WIKIPEDIA

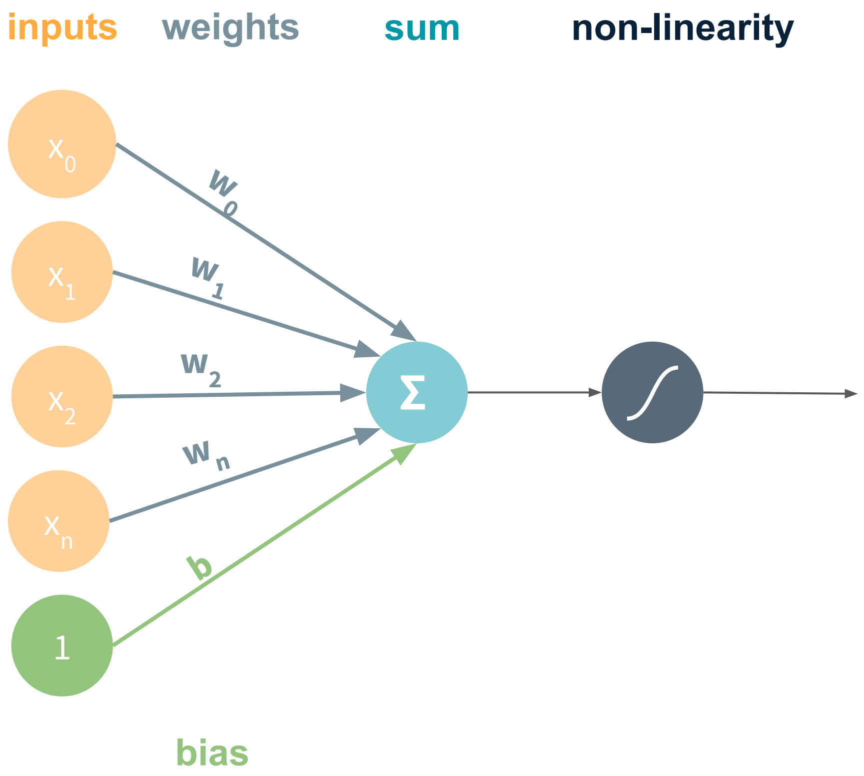
Fundamentals of Deep Learning

The Perceptron

1. Invented in 1954 by Frank Rosenblatt
2. Inspired by neurobiology



The Perceptron

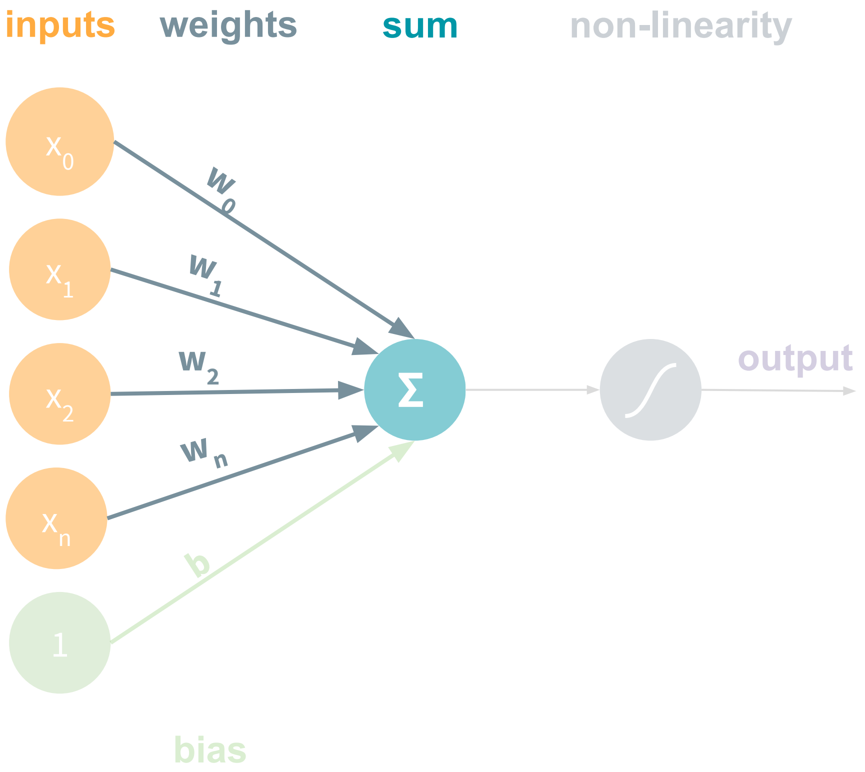


Perceptron Forward Pass

$$output =$$

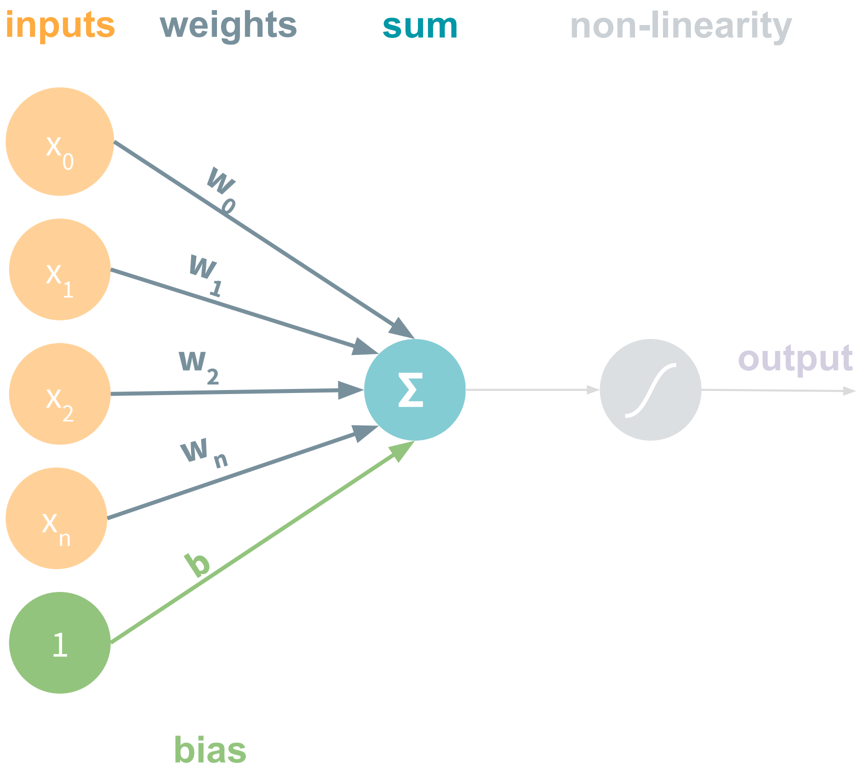
Perceptron Forward Pass

$$output = \sum_{i=0}^{N} x_i * w_i$$



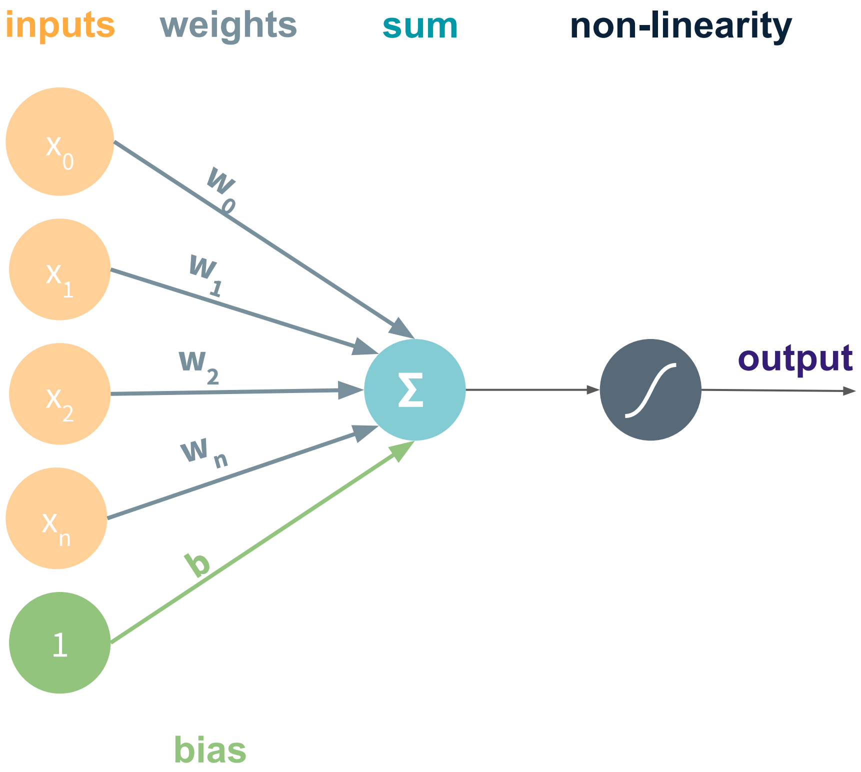
Perceptron Forward Pass

$$output = \left(\sum_{i=0}^{N} x_i * w_i\right) + b$$



Perceptron Forward Pass

$$output = g((\sum_{i=0}^{N} x_i * w_i) + b)$$

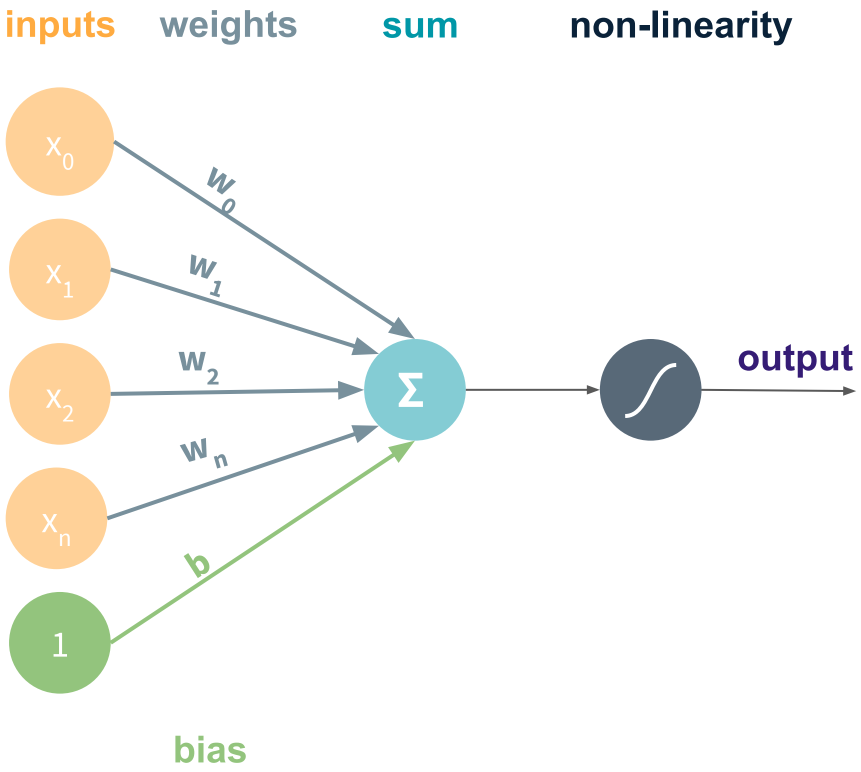


Perceptron Forward Pass

$$output = g(XW + b)$$

$$X = x_0, x_1, \ldots x_n$$

$$W = w_0, w_1, \ldots w_n$$



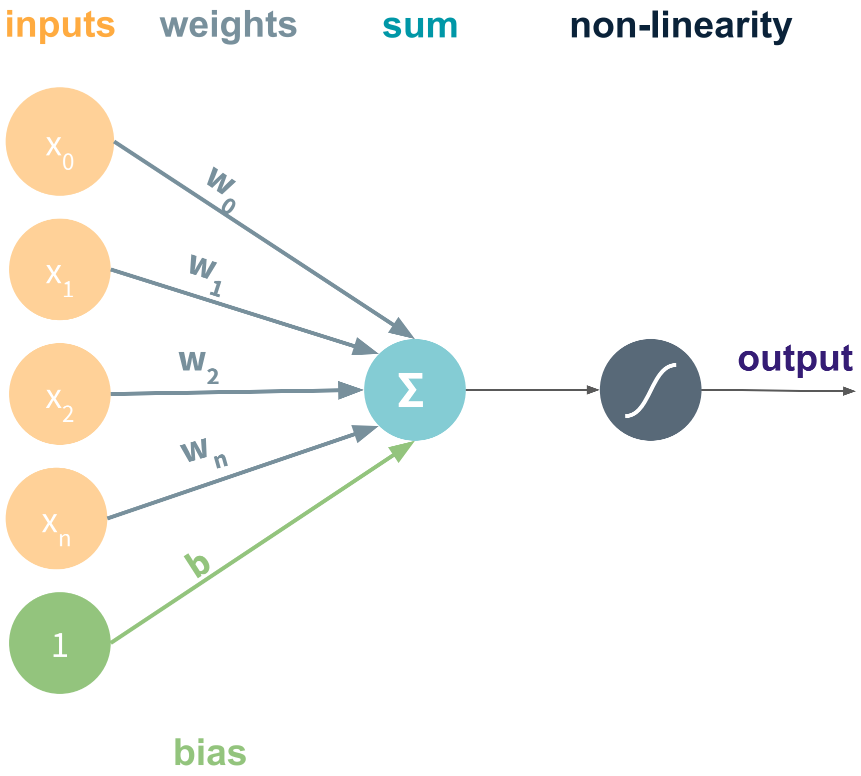
Perceptron Forward Pass

Activation Function

$$output = \boxed{g}(XW + b)$$

$$X = x_0, x_1, ...x_n$$

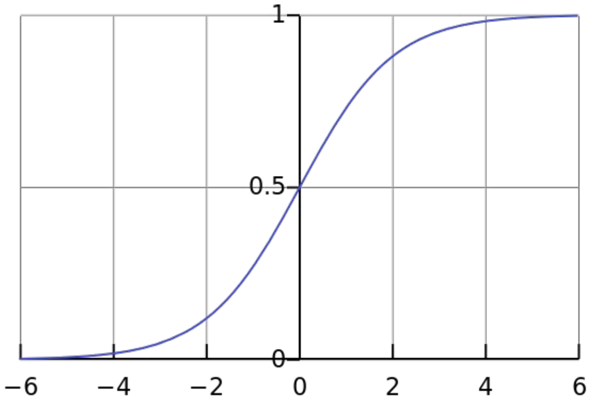
$$W = w_0, w_1, ...w_n$$

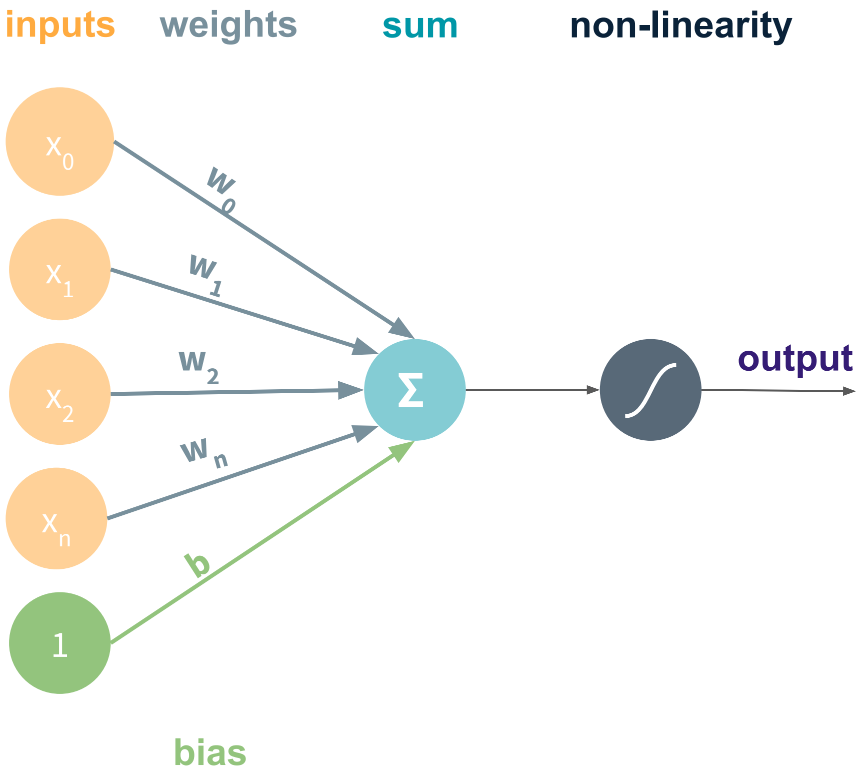


Sigmoid Activation

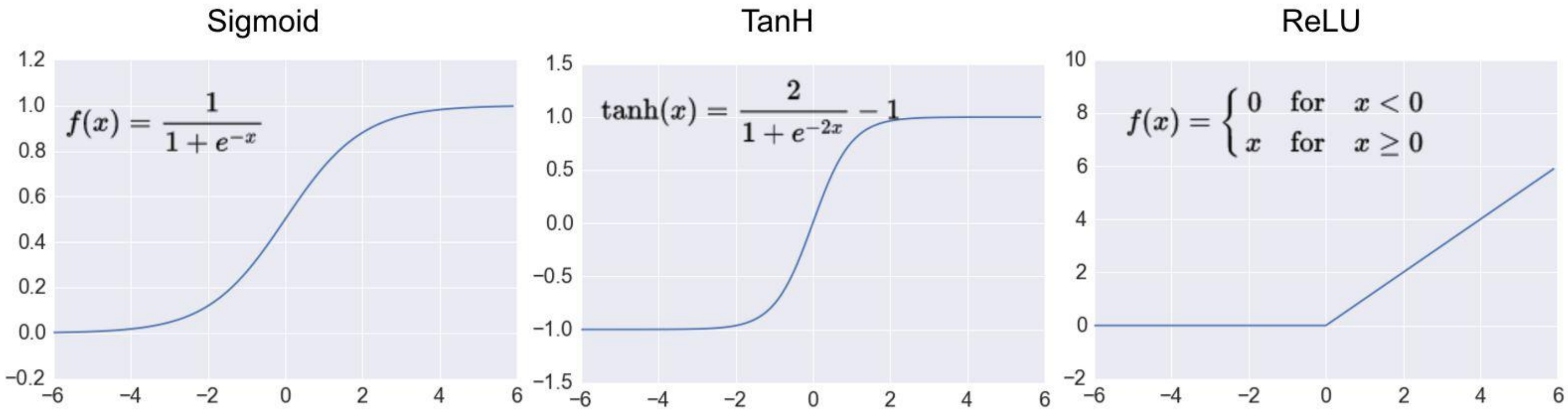
$$output = g(XW + b)$$

$$g(a) = \sigma(a) = \frac{1}{1+e^{-a}}$$



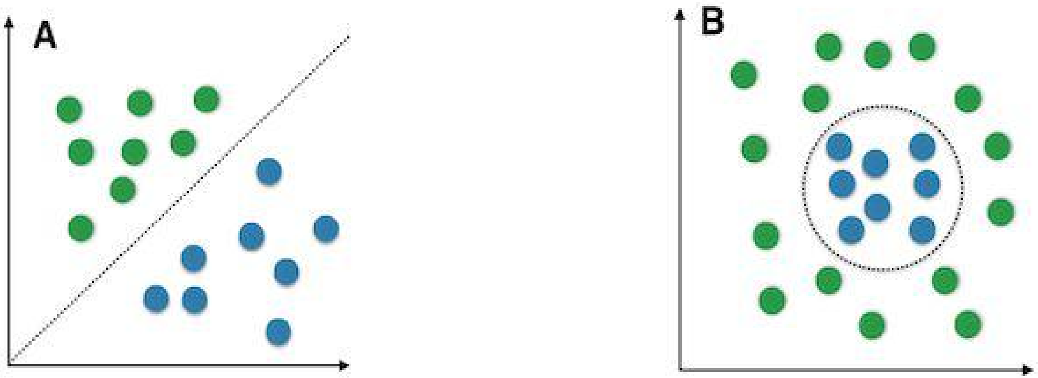


Common Activation Functions



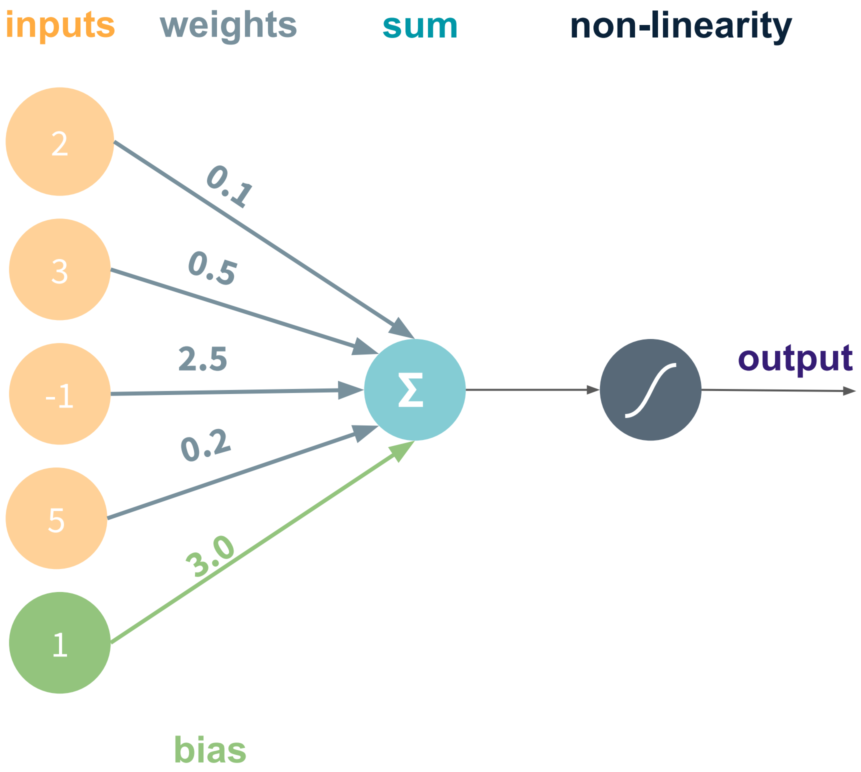
Importance of Activation Functions

- Activation functions add non-linearity to our network's function
- Most real-world problems + data are **non-linear**



Perceptron Forward Pass

$$output = g(XW + b)$$



Perceptron Forward Pass

$$output = g($$

(2*0.1) +

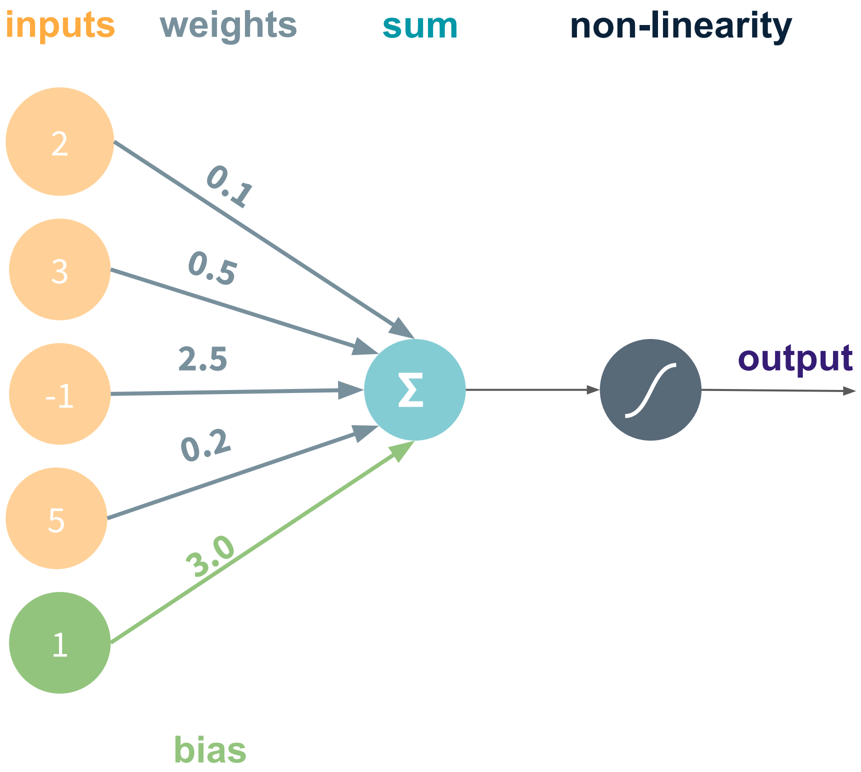
(3*0.5) +

(-1*2.5) +

(5*0.2) +

(1*3.0)

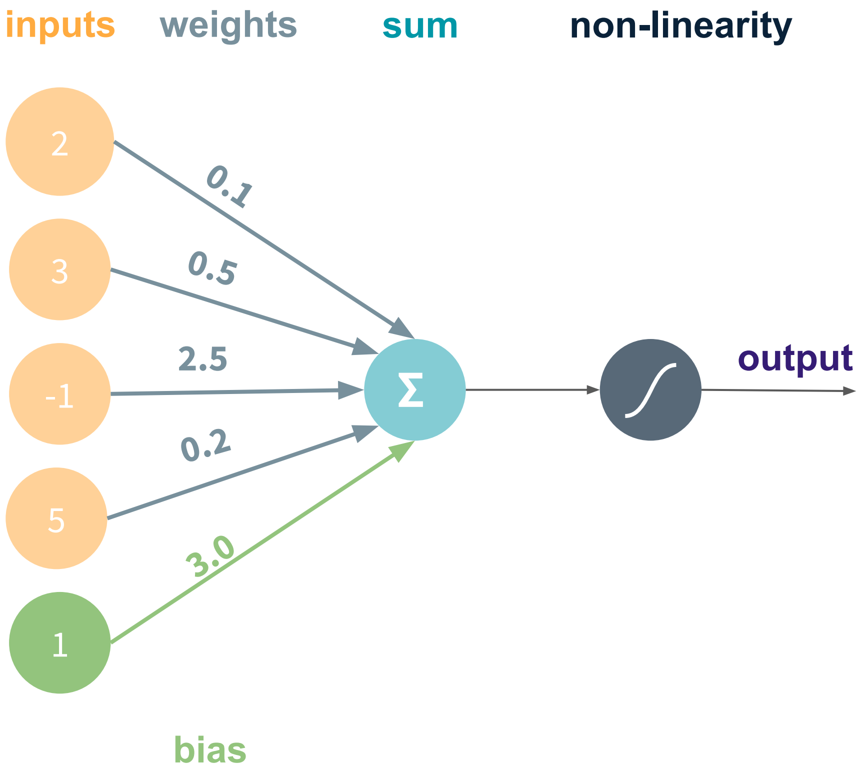
)



Perceptron Forward Pass

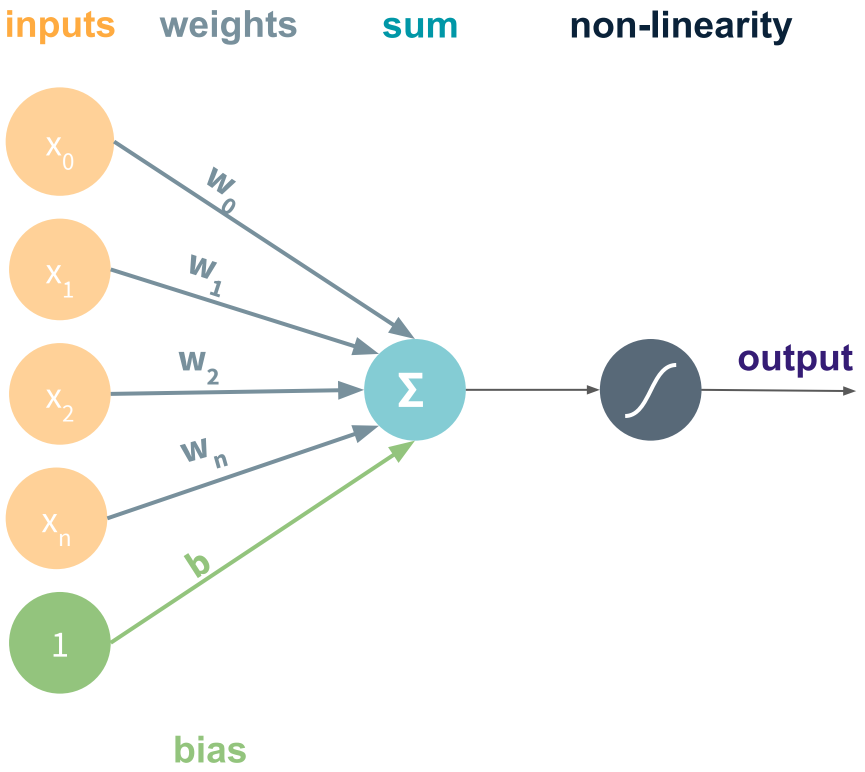
$$output = g(3.2) = \sigma(3.2)$$

$$= \frac{1}{(1 + e^{-3.2})} = 0.96$$

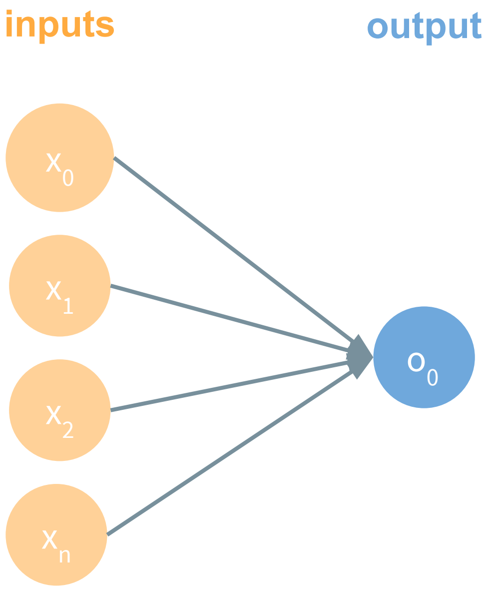


How do we build neural networks with perceptrons?

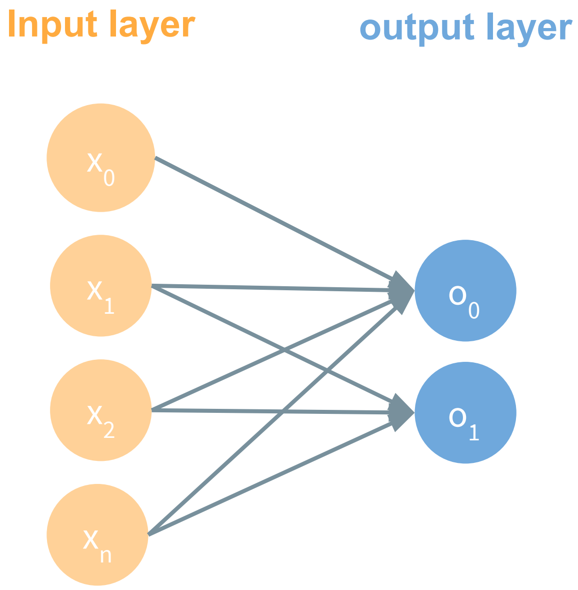
Perceptron Diagram Simplified



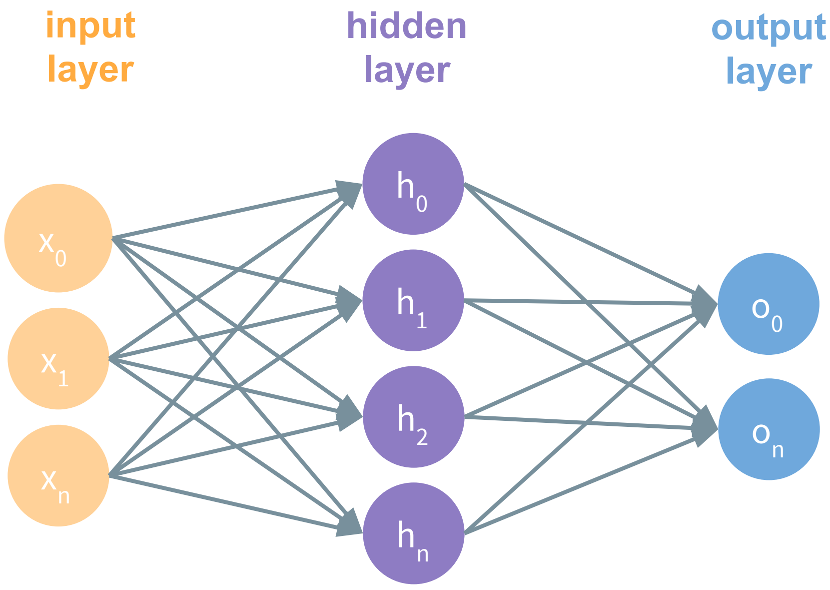
Perceptron Diagram Simplified



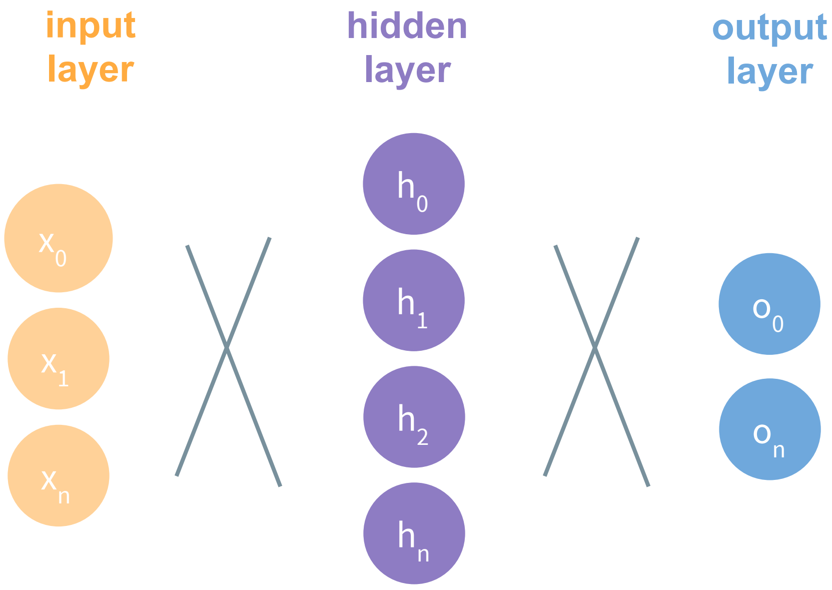
Multi-Output Perceptron



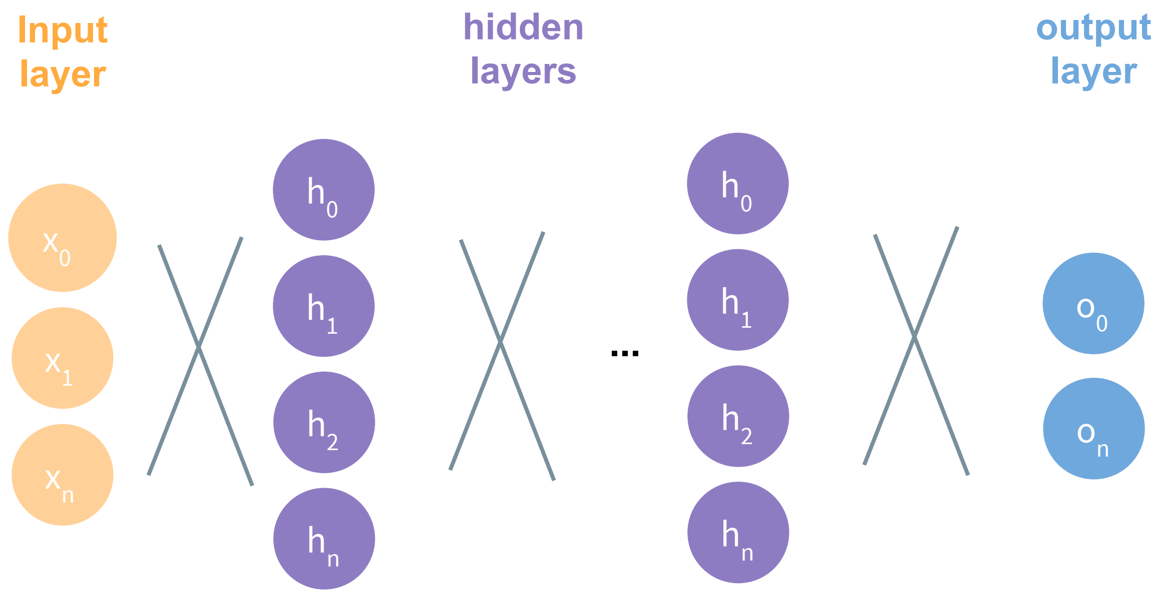
Multi-Layer Perceptron (MLP)



Multi-Layer Perceptron (MLP)



Deep Neural Network



Applying Neural Networks

Example Problem: Will my Flight be Delayed?

Example Problem: Will my Flight be Delayed?

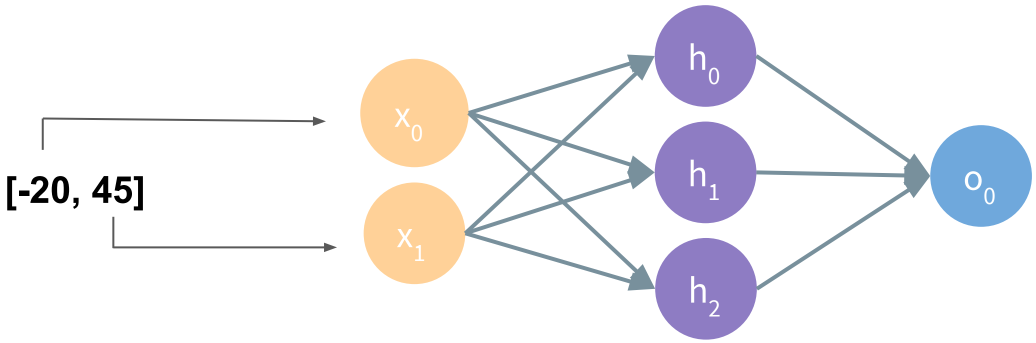
Temperature: -20 F

Wind Speed: 45 mph

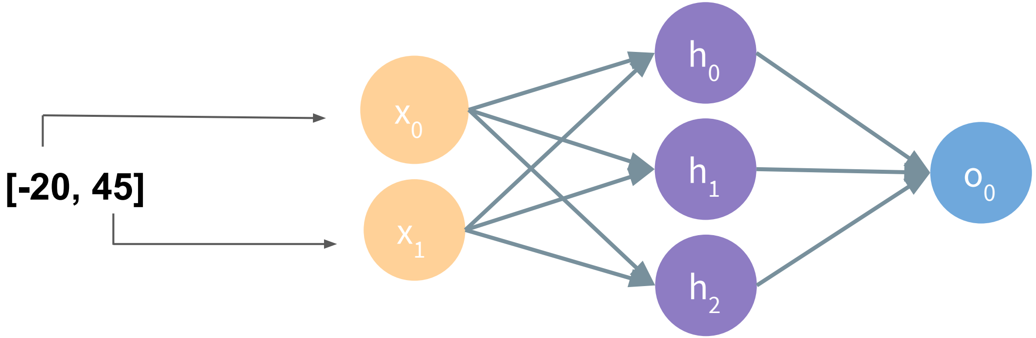
Example Problem: Will my Flight be Delayed?

[-20, 45]

Example Problem: Will my Flight be Delayed?

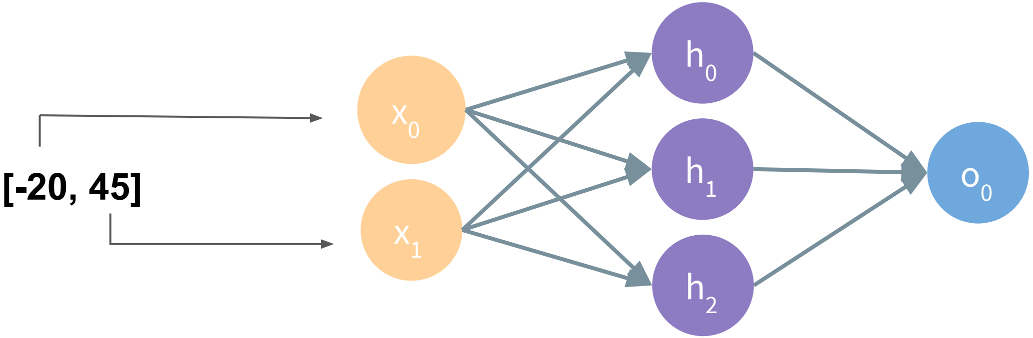


Example Problem: Will my Flight be Delayed?



Predicted: 0.05

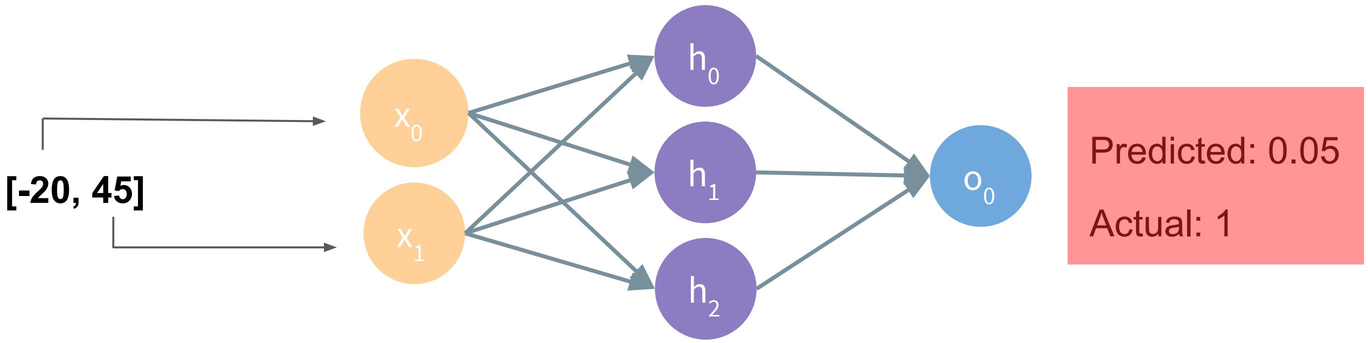
Example Problem: Will my Flight be Delayed?



Predicted: 0.05

Actual: 1

Quantifying Loss

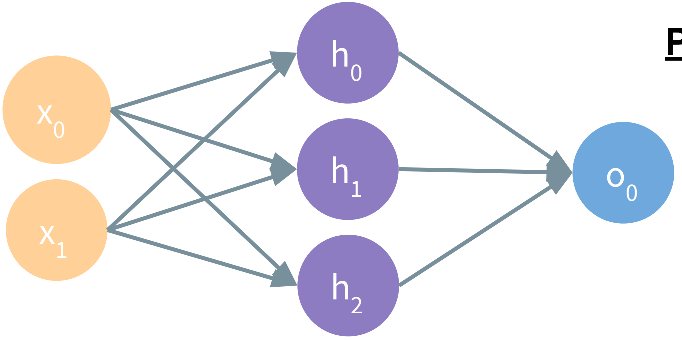


$$loss(f(x^{(i)};\theta), y^{(i)})$$

Total Loss

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



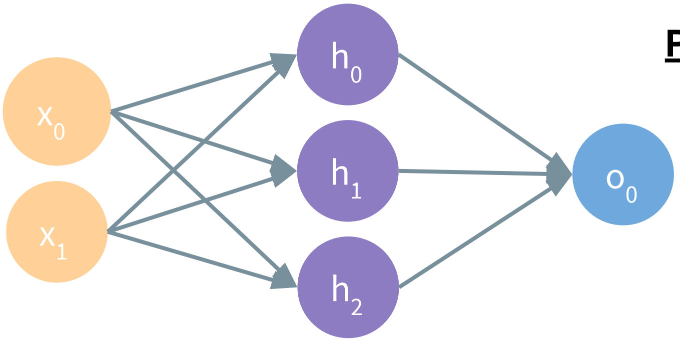
Predicted	**Actual**
[	[
0.05	1
0.02	0
0.96	1
0.35	1
]	]

$$\text{total loss} := J(\theta) = \frac{1}{N}\sum_{i}^{N} loss(\underbrace{f(x^{(i)};\theta)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

Total Loss

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



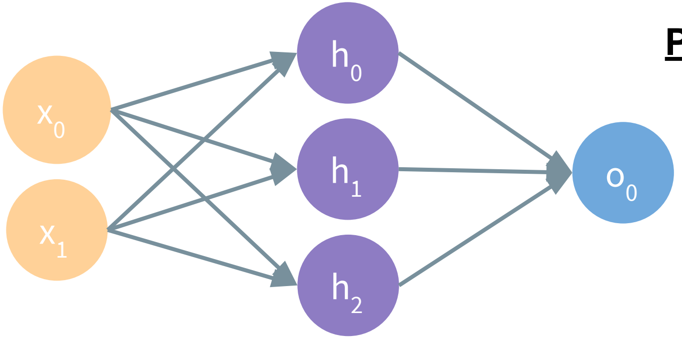
Predicted	**Actual**
[	[
0.05	1
0.02	0
0.96	1
0.35	1
]	]

$$\text{total loss} := J(\theta) = \frac{1}{N}\sum_{i}^{N} loss(\underbrace{f(x^{(i)};\theta)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

Binary Cross Entropy Loss

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



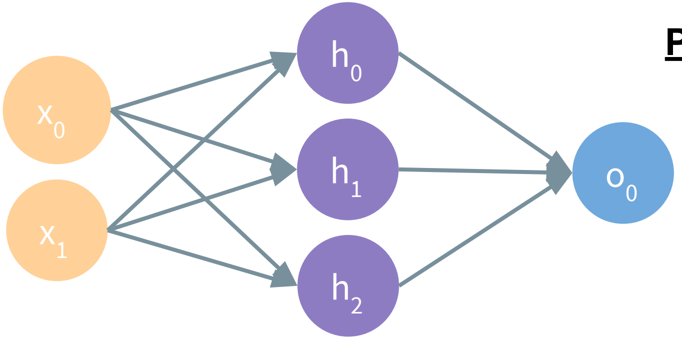
Predicted	**Actual**
[	[
0.05	1
0.02	0
0.96	1
0.35	1
]	]

$$\text{cross\_entropy}(\theta) = \frac{1}{N}\sum_{i}^{N} \underbrace{y^{(i)}}_{\text{Actual}} log(\underbrace{f(x^{(i)};\theta)}_{\text{Predicted}}) + (1 - \underbrace{y^{(i)}}_{\text{Actual}})log(1 - \underbrace{f(x^{(i)};\theta)}_{\text{Predicted}}))$$

Mean Squared Error (MSE) Loss

Input

[
[-20, 45],
[80, 0],
[4, 15],
[45, 60],
]



Predicted	**Actual**
[	[
10	40
45	42
100	110
15	55
]	]

$$\mathrm{MSE}(\theta) = \frac{1}{N}\sum_{i}^{N}(\underbrace{f(x^{(i)};\theta)}_{\text{Predicted}} - \underbrace{y^{(i)}}_{\text{Actual}})^2$$

Training Neural Networks

Training Neural Networks: Objective

$$arg_{\theta}\ \min \frac{1}{N}\sum_{i}^{N} \text{loss}(f(x^{(i)};\theta), y^{(i)})$$

Training Neural Networks: Objective

$$\underbrace{arg_{\theta}\ \min \frac{1}{N}\sum_{i}^{N} \text{loss}(f(x^{(i)};\theta), y^{(i)})}_{J(\theta)}$$

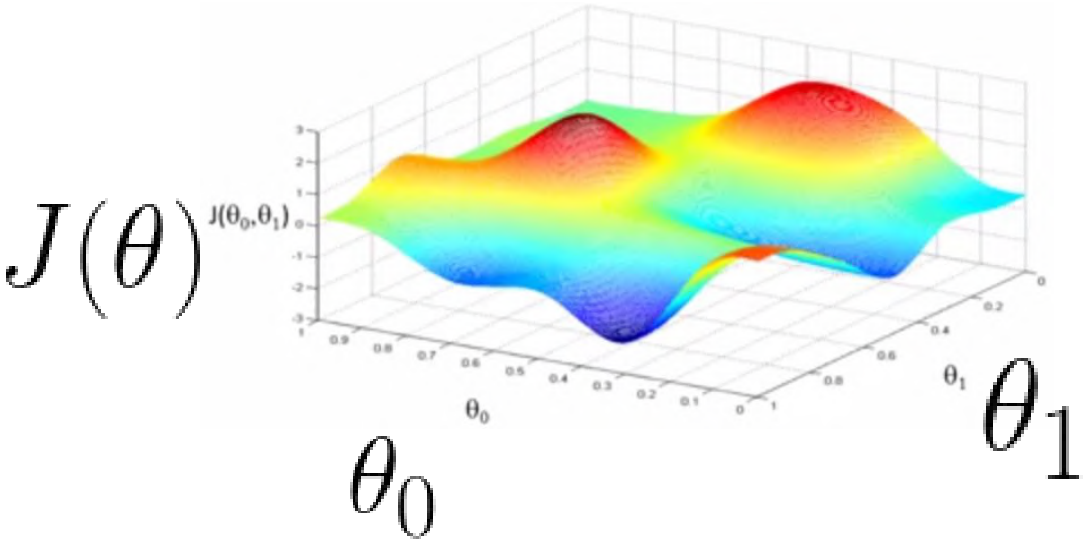
loss function

Training Neural Networks: Objective

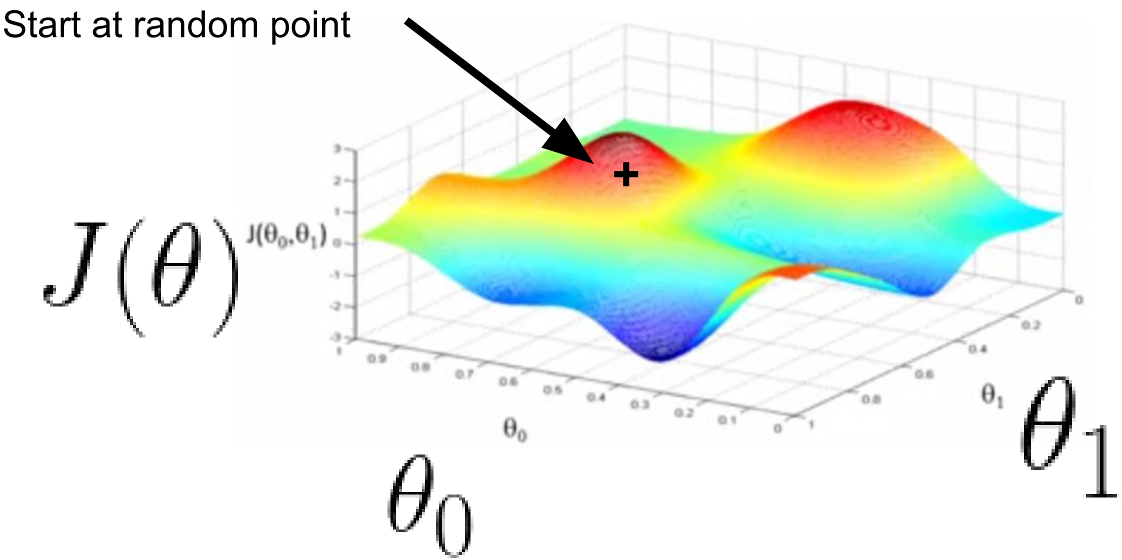
$$\underbrace{arg_{\theta}\ \min \frac{1}{N}\sum_{i}^{N} \text{loss}(f(x^{(i)};\theta), y^{(i)})}_{J(\theta)}$$

$$\theta = W_1, W_2 \ldots W_n$$

Loss is a **function** of the model's parameters



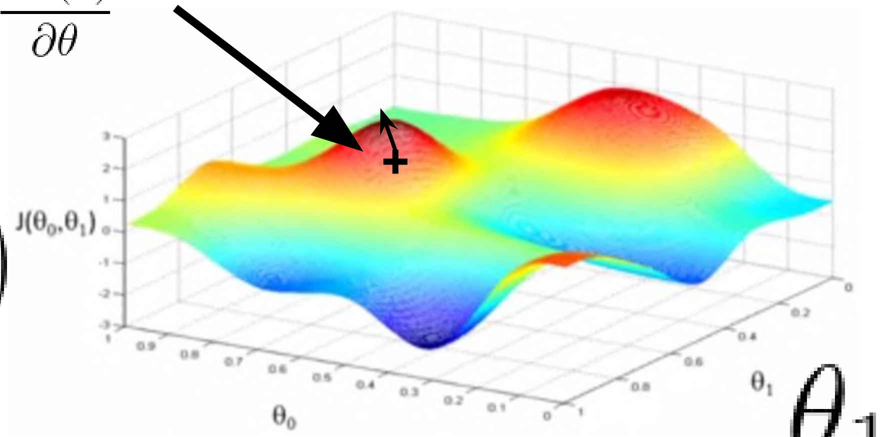
How to minimize loss?



How to minimize loss?

Compute: $\frac{\partial J(\theta)}{\partial \theta}$

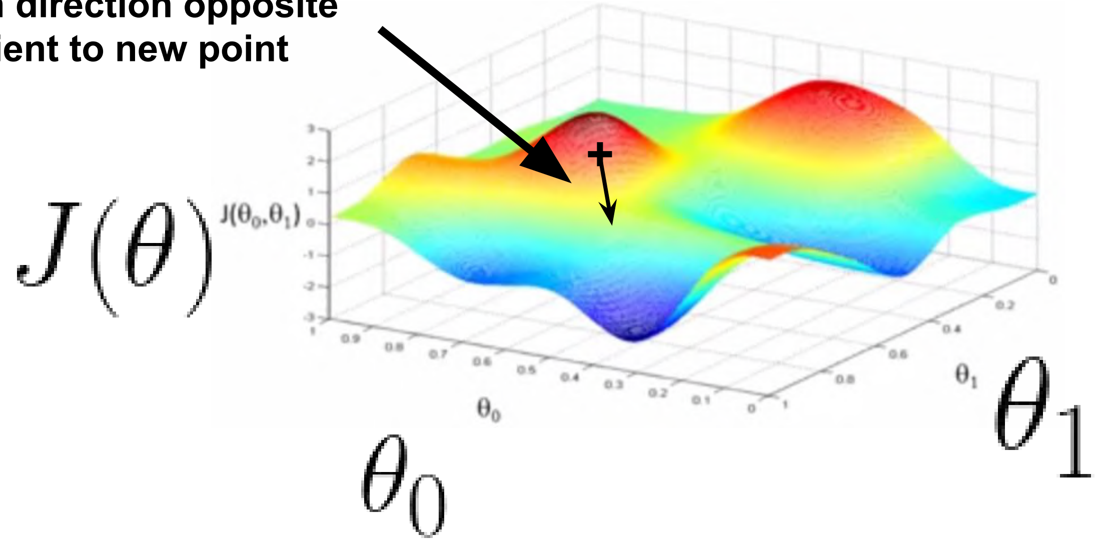
$J(\theta)$



θ_0 θ_1

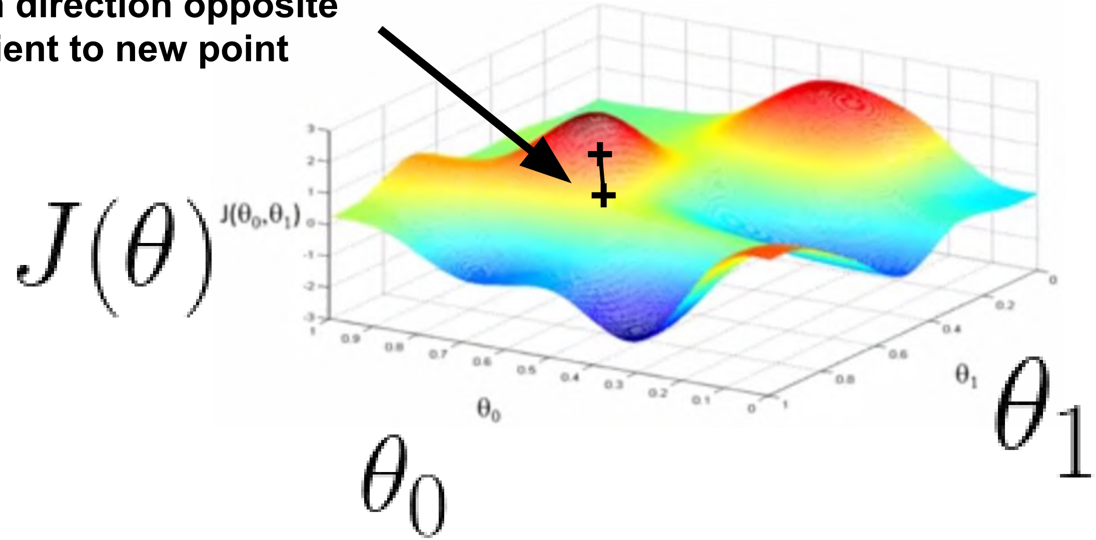
How to minimize loss?

Move in direction opposite
of gradient to new point



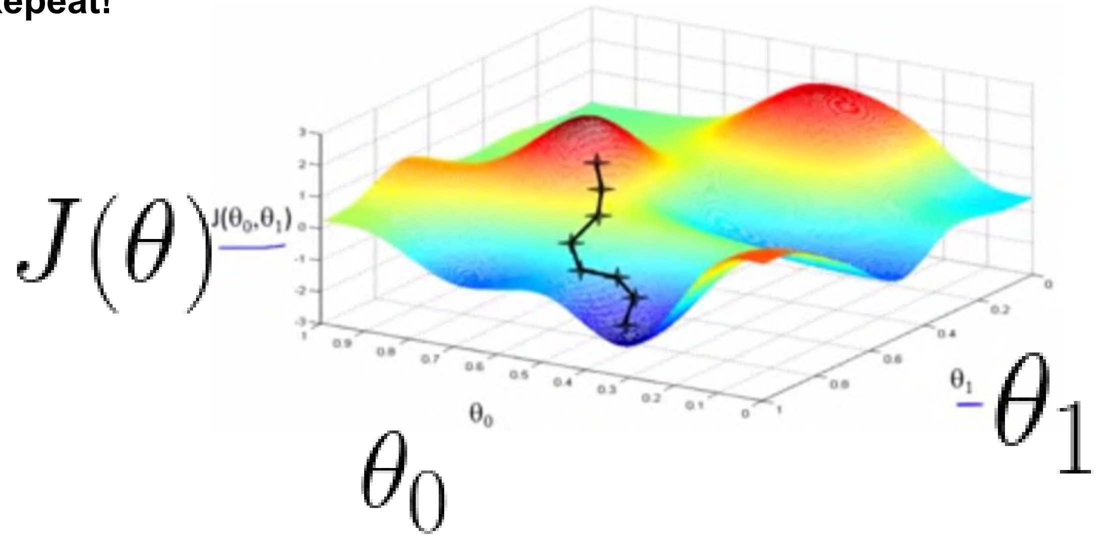
How to minimize loss?

Move in direction opposite of gradient to new point



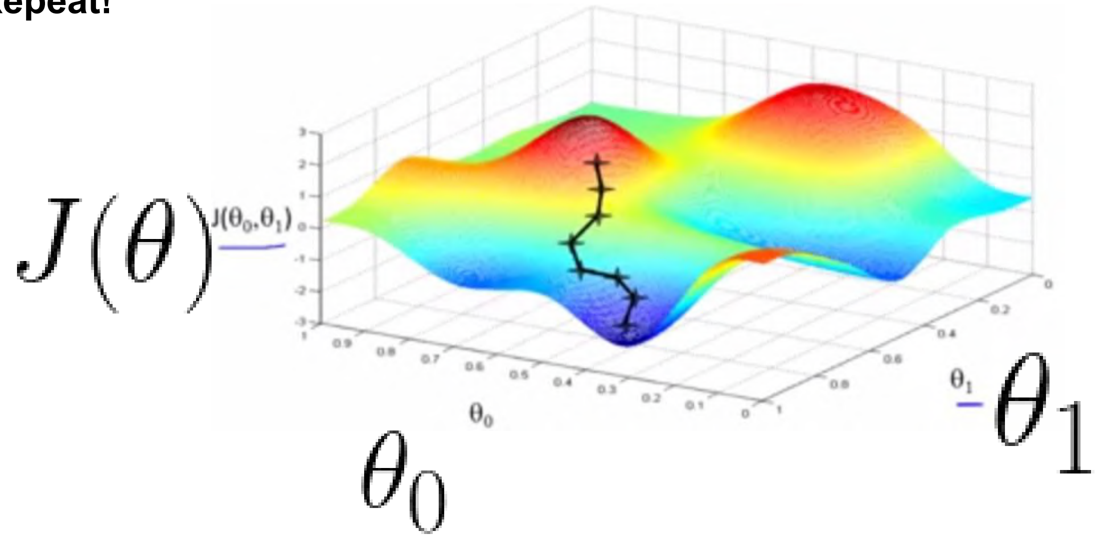
How to minimize loss?

Repeat!



This is called Stochastic Gradient Descent (SGD)

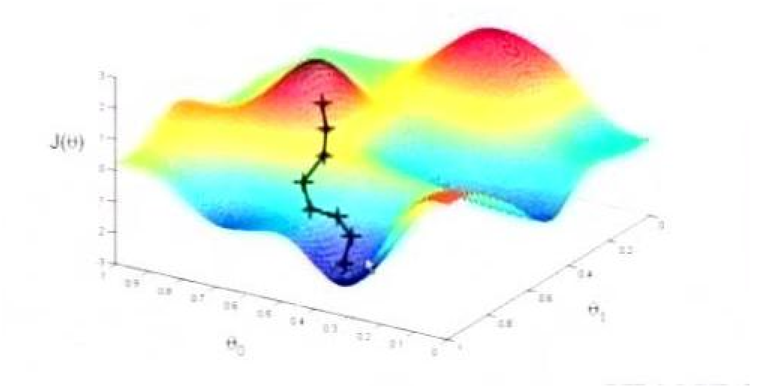
Repeat!



Stochastic Gradient Descent (SGD)

- Initialize θ randomly
- For N Epochs
    - For each training example (x, y):
        - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
        - Update θ with update rule:

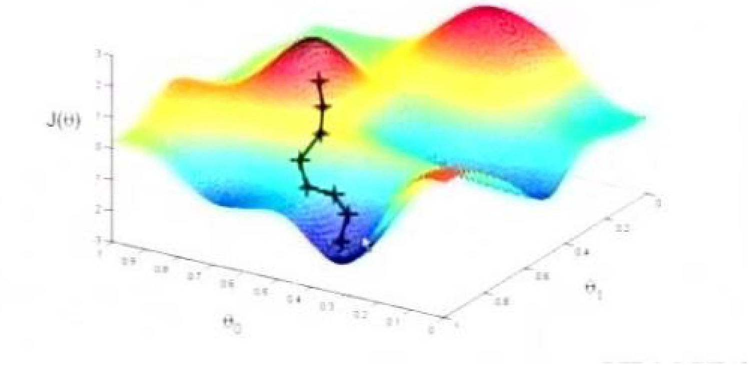
$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$



Stochastic Gradient Descent (SGD)

- Initialize θ randomly
- For N Epochs
  - For each training example (x, y):
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
    - Update θ with update rule:

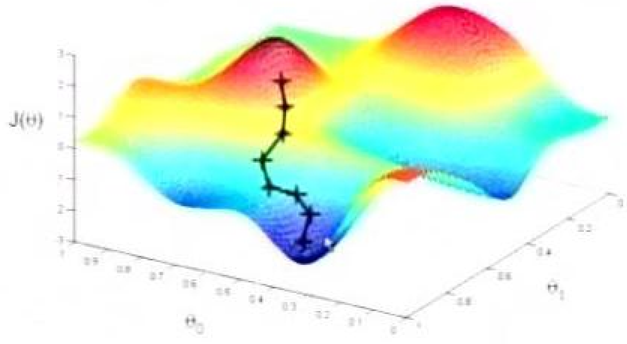
$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$



Stochastic Gradient Descent (SGD)

- Initialize θ randomly
- For N Epochs
  - For each training example (x, y):
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
    - Update θ with update rule:

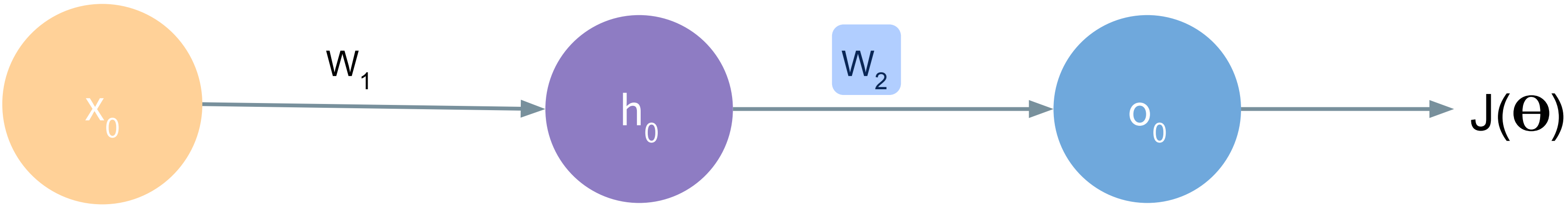
$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$



- How to Compute Gradient?

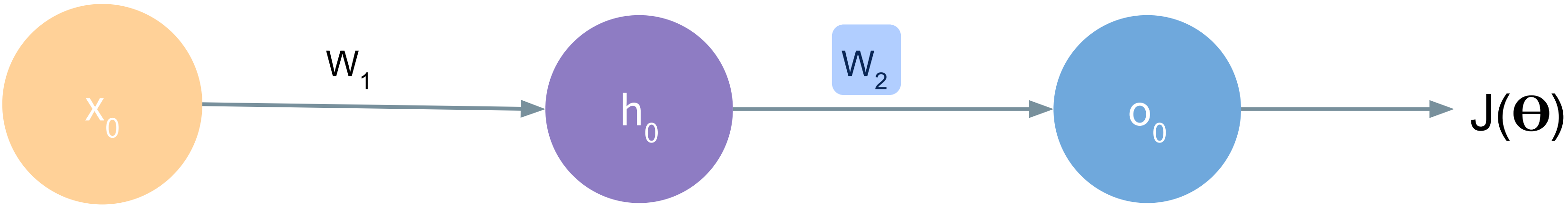
Calculating the Gradient: Backpropagation

Calculating the Gradient: Backpropagation



$$\frac{\partial J(\theta)}{\partial W_2} =$$

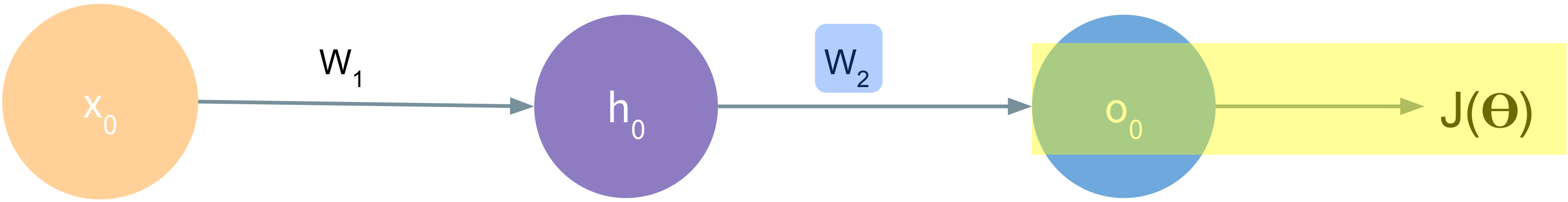
Calculating the Gradient: Backpropagation



Apply the chain rule

$$\frac{\partial J(\theta)}{\partial W_2} =$$

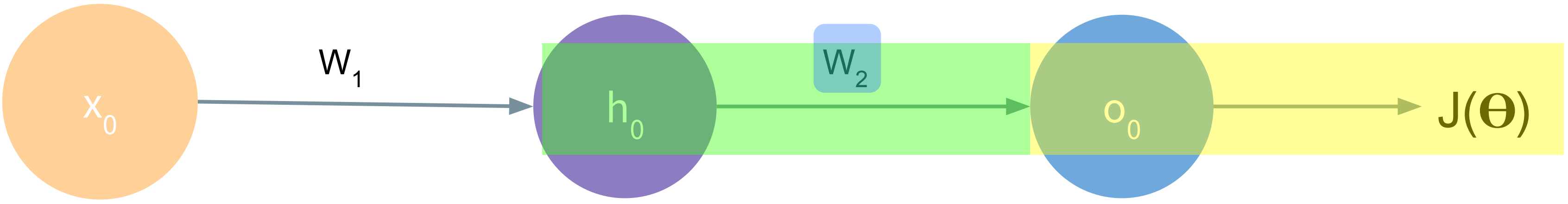
Calculating the Gradient: Backpropagation



Apply the chain rule

$$\frac{\partial J(\theta)}{\partial W_2} = \frac{\partial J(\theta)}{\partial o_0}$$

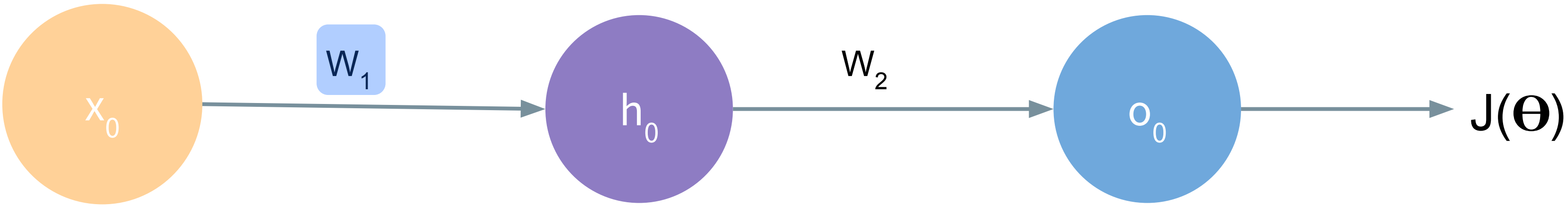
Calculating the Gradient: Backpropagation



Apply the chain rule

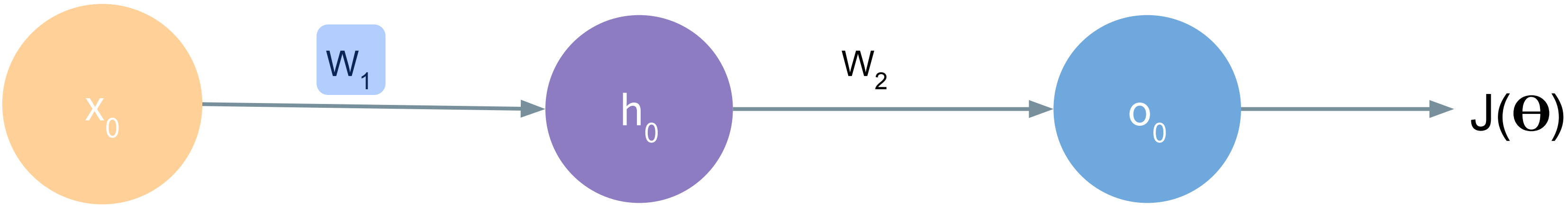
$$\frac{\partial J(\theta)}{\partial W_2} = \frac{\partial J(\theta)}{\partial o_0} * \frac{\partial o_0}{\partial W_2}$$

Calculating the Gradient: Backpropagation



$$\frac{\partial J(\theta)}{\partial W_1} =$$

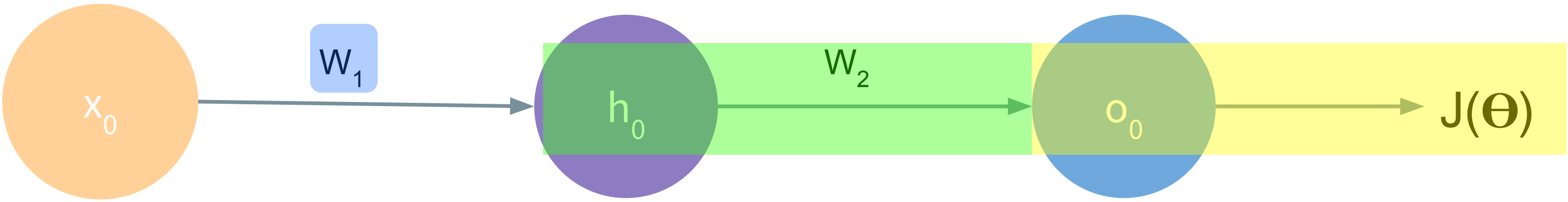
Calculating the Gradient: Backpropagation



Apply the chain rule

$$\frac{\partial J(\theta)}{\partial W_1} =$$

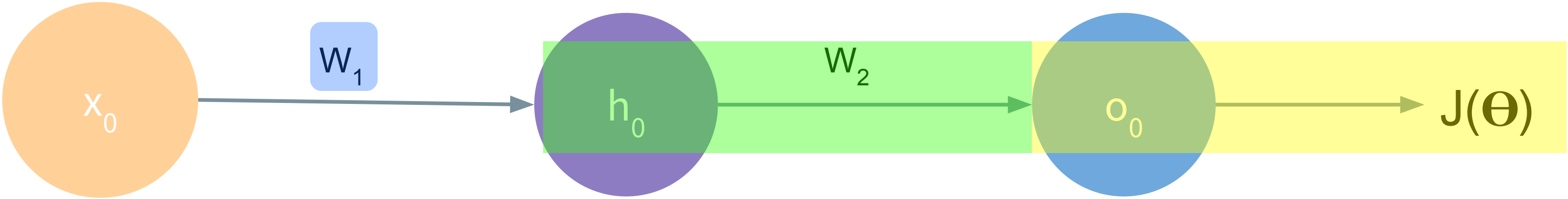
Calculating the Gradient: Backpropagation



Apply the chain rule

$$\frac{\partial J(\theta)}{\partial W_1} = \frac{\partial J(\theta)}{\partial o_0} * \frac{\partial o_0}{\partial h_0}$$

Calculating the Gradient: Backpropagation

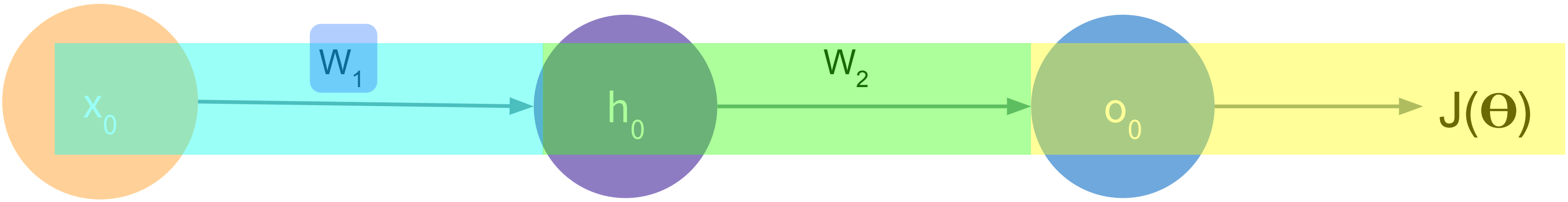


Apply the chain rule

Apply the chain rule

$$\frac{\partial J(\theta)}{\partial W_1} = \frac{\partial J(\theta)}{\partial o_0} * \frac{\partial o_0}{\partial h_0}$$

Calculating the Gradient: Backpropagation



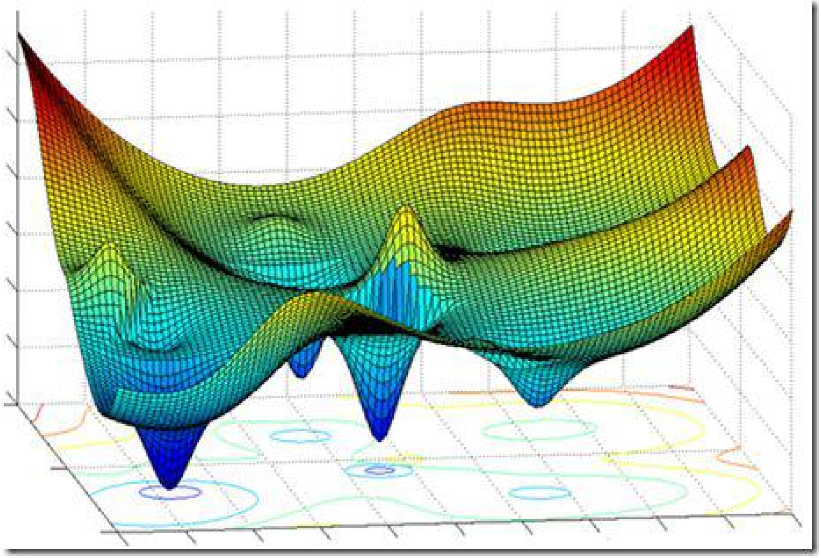
Apply the chain rule

Apply the chain rule

$$\frac{\partial J(\theta)}{\partial W_1} = \frac{\partial J(\theta)}{\partial o_0} * \frac{\partial o_0}{\partial h_0} * \frac{\partial h_0}{\partial W_1}$$

Training Neural Networks In Practice

Loss function can be difficult to optimize



Loss function can be difficult to optimize

Update Rule: $$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

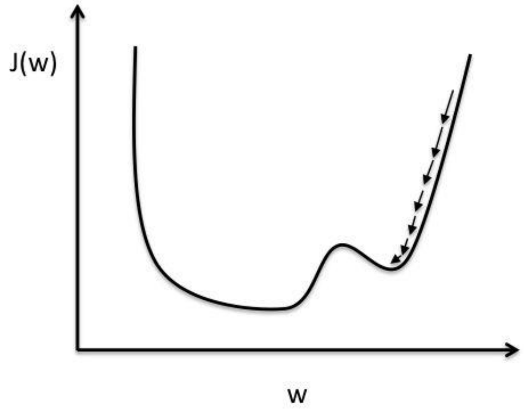
Loss function can be difficult to optimize

How to Choose Learning Rate?

Update Rule: $$\theta := \theta - \boxed{\eta} \frac{\partial J(\theta)}{\partial \theta}$$

Learning Rate & Optimization

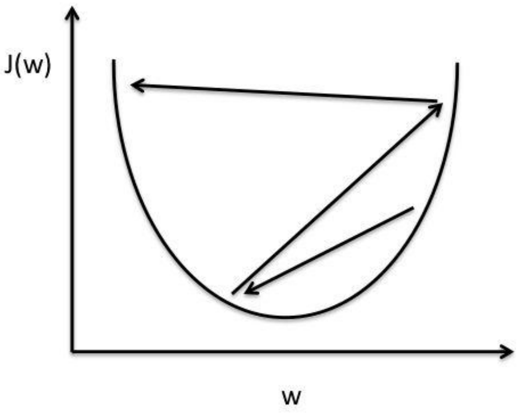
- Small Learning Rate



Small learning rate: Many iterations until convergence and trapping in local minima.

Learning Rate & Optimization

- Large learning rate



Large learning rate: Overshooting.

How to deal with this?

1. Try lots of different learning rates to see what is 'just right'

How to deal with this?

1. Try lots of different learning rates to see what is ‘just right’
2. Do something smarter

How to deal with this?

1. Try lots of different learning rates to see what is ‘just right’
2. **Do something smarter : Adaptive Learning Rate**

Adaptive Learning Rate

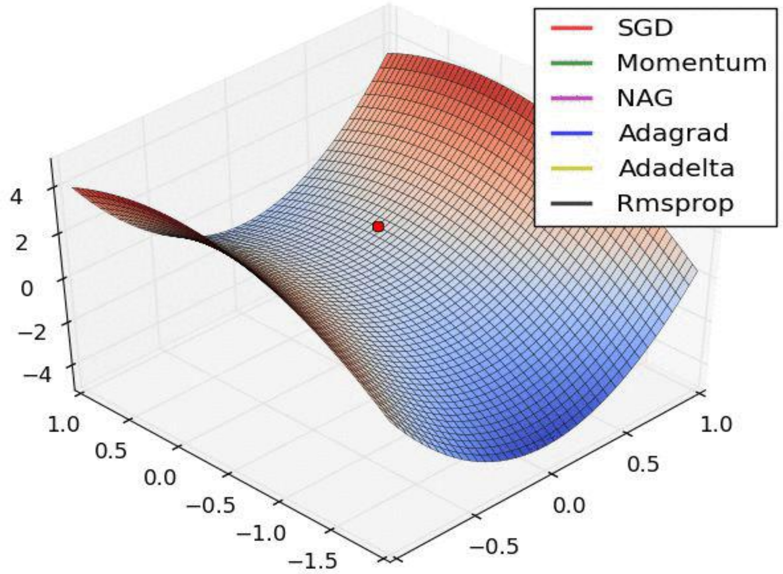
- Learning rate is no longer fixed
- Can be made larger or smaller depending on:
  - how large gradient is
  - how fast learning is happening
  - size of particular weights
  - etc

Adaptive Learning Rate Algorithms

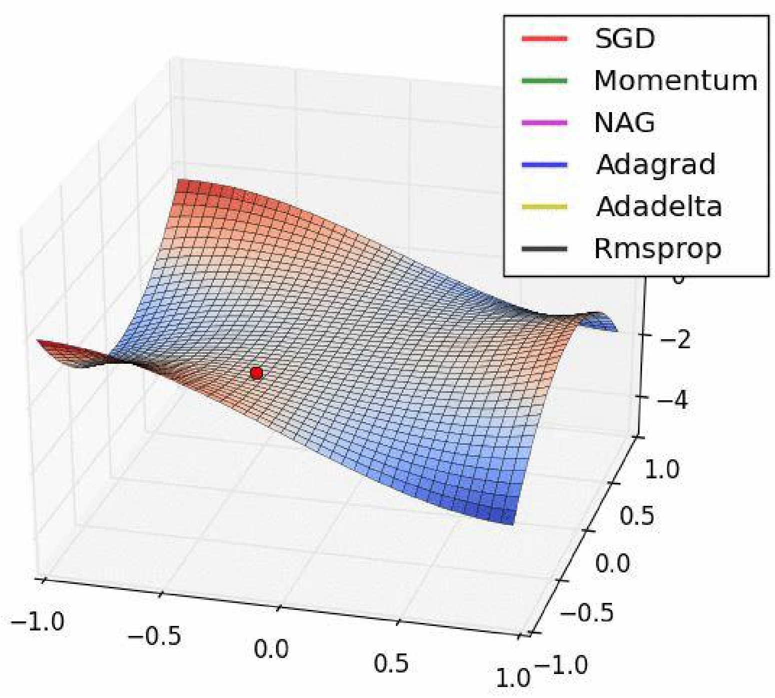
- ADAM
- Momentum
- NAG
- Adagrad
- Adadelta
- RMSProp

For details: check out http://sebastianruder.com/optimizing-gradient-descent/

Escaping Saddle Points



Escaping Saddle Points



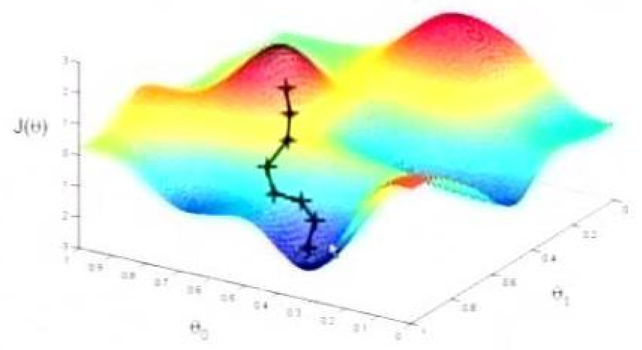
Training Neural Networks In Practice 2: MiniBatches

Why is it **Stochastic** Gradient Descent?

- Initialize θ randomly
- For N Epochs
    - For each training example (x, y):
        - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta}$
        - Update θ with update rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

Only an estimate of true gradient!

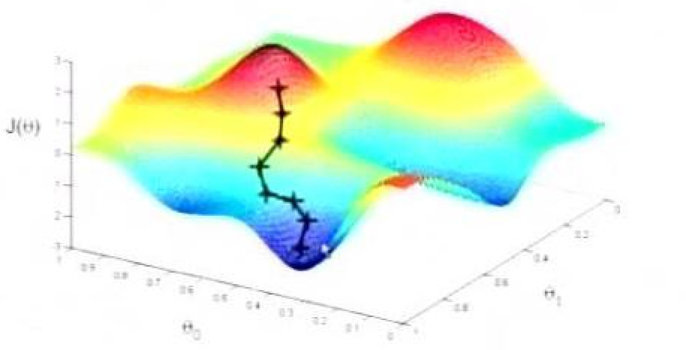


Minibatches Reduce Gradient Variance

- Initialize θ randomly
- For N Epochs
  - For each training **batch {(x0, y0), … , (x_B, y_B)}:**
    - Compute Loss Gradient: $\frac{\partial J(\theta)}{\partial \theta} = \frac{1}{B}\sum_{i}^{B} \frac{\partial J_i(\theta)}{\partial \theta}$
    - Update θ with update rule:

$$\theta := \theta - \eta \frac{\partial J(\theta)}{\partial \theta}$$

More accurate estimate!

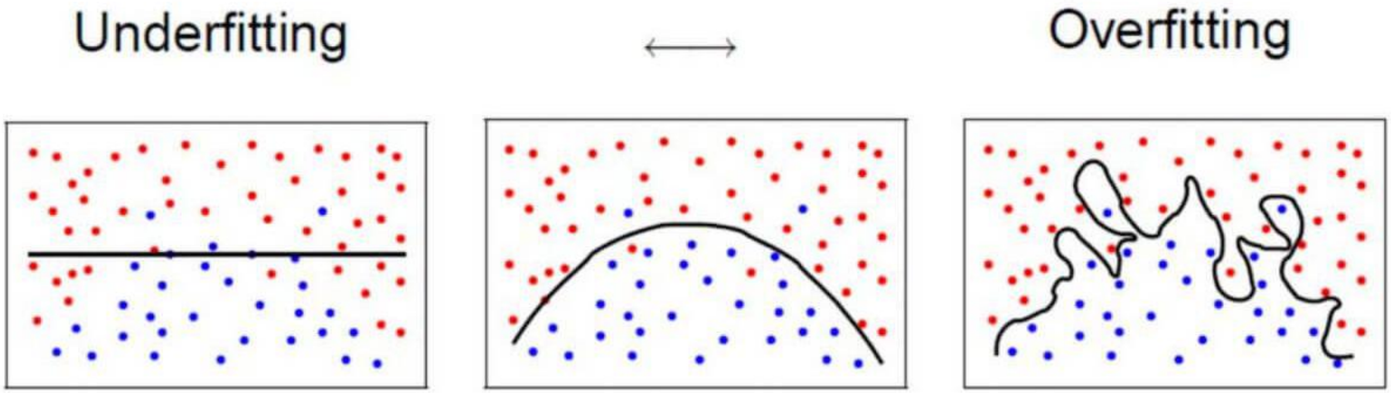


Advantages of Minibatches

- More accurate estimation of gradient
  - Smoother convergence
  - Allows for larger learning rates
- Minibatches lead to fast training!
  - Can parallelize computation + achieve significant speed increases on GPU's

Training Neural Networks In Practice 3: Fighting Overfitting

The Problem of Overfitting

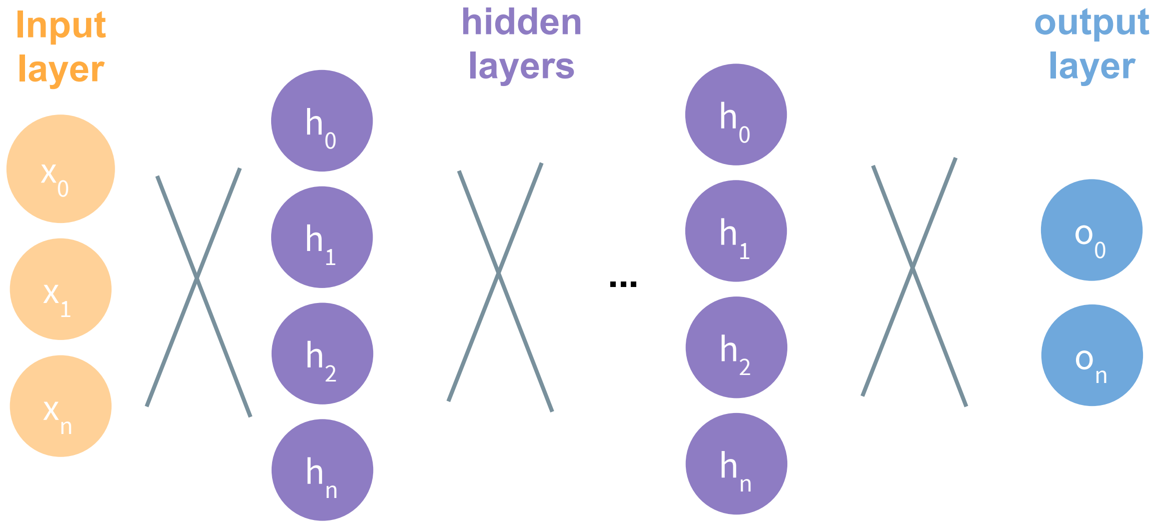


Regularization Techniques

1. Dropout
2. Early Stopping
3. Weight Regularization
4. ...many more

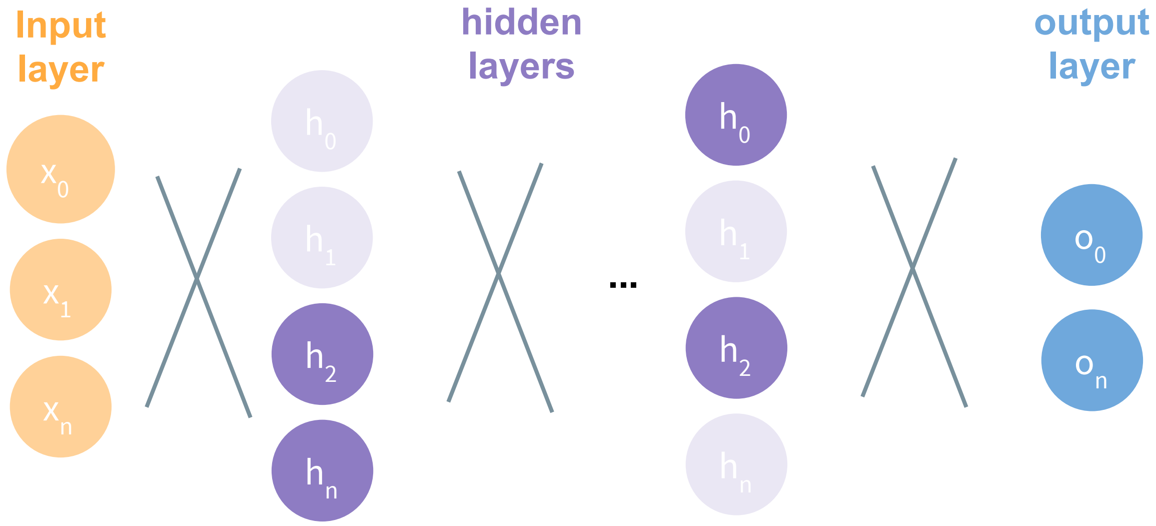
Regularization I: Dropout

- During training, randomly set some activations to 0



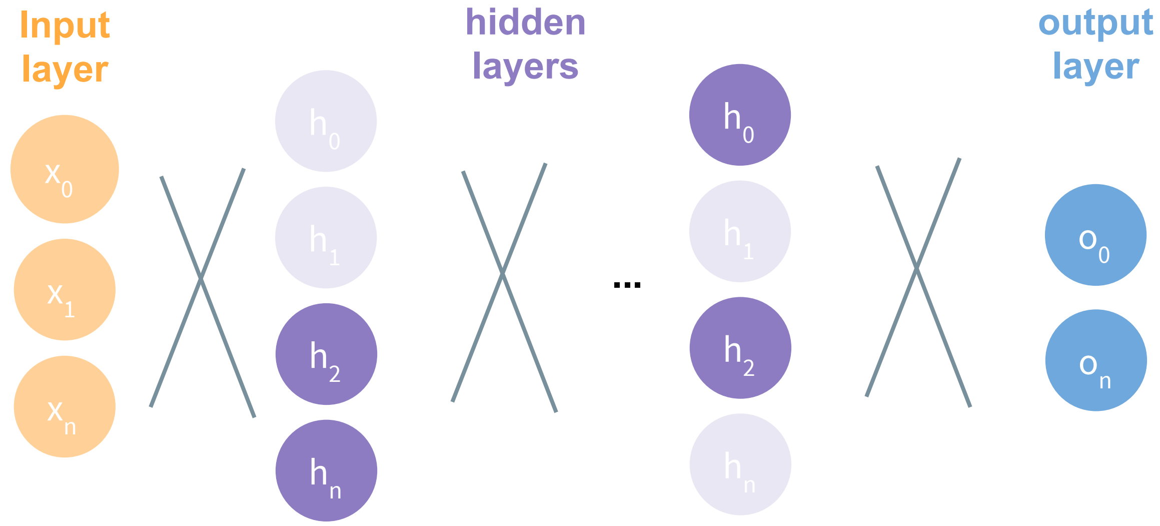
Regularization I: Dropout

- During training, randomly set some activations to 0



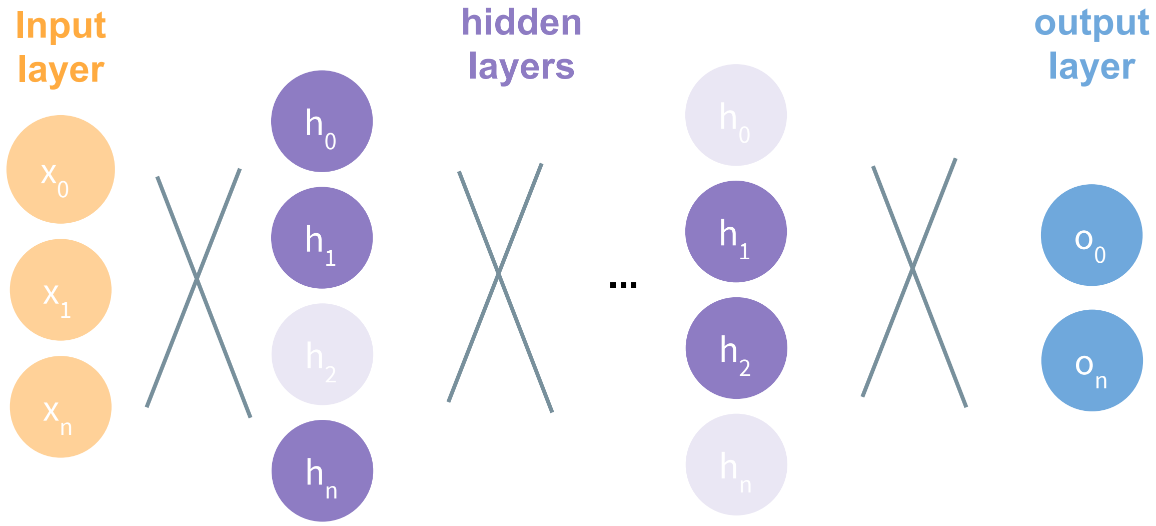
Regularization I: Dropout

- During training, randomly set some activations to 0
  - Typically ‘drop’ 50% of activations in layer
  - Forces network to not rely on any 1 node



Regularization I: Dropout

- During training, randomly set some activations to 0
  - Typically ‘drop’ 50% of activations in layer
  - Forces network to not rely on any 1 node



Regularization II: Early Stopping

- Don't give the network time to overfit
- ...
- **Epoch 15:** Train: 85% Validation: 80%
- **Epoch 16:** Train: 87% Validation: 82%
- **Epoch 17:** Train: 90% Validation: 85%
- **Epoch 18:** Train: 95% Validation: 83%
- **Epoch 19:** Train: 97% Validation: 78%
- **Epoch 20:** Train: 98% Validation: 75%

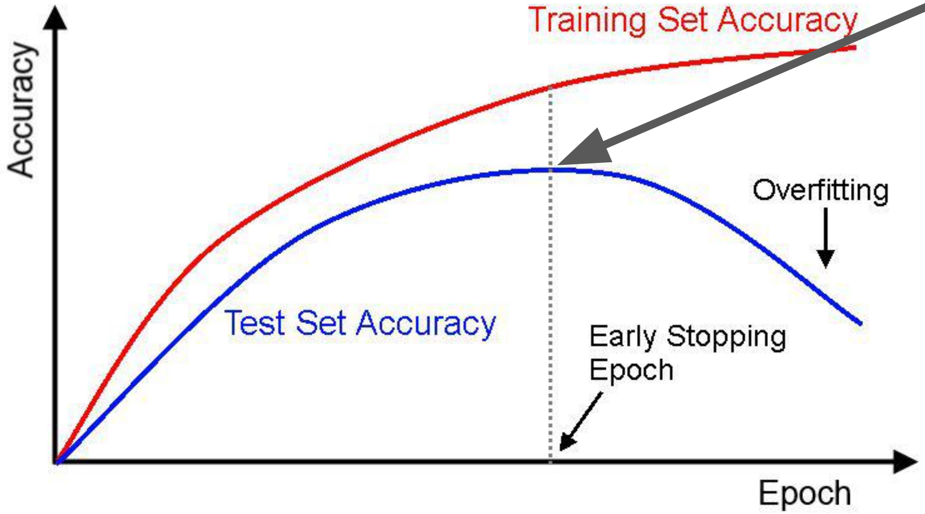
Regularization II: Early Stopping

- Don't give the network time to overfit
- ...
- **Epoch 15:** Train: 85% Validation: 80%
- **Epoch 16:** Train: 87% Validation: 82%
- **Epoch 17: Train: 90% Validation: 85%**
- **Epoch 18:** Train: 95% Validation: 83%
- **Epoch 19:** Train: 97% Validation: 78%
- **Epoch 20:** Train: 98% Validation: 75%

Stop here!

Regularization II: Early Stopping

Stop here!



Regularization III: Weight Regularization

- Large weights typically mean model is overfitting
- Add the size of the weights to our loss function
- Perform well on task + keep weights small

Regularization III: Weight Regularization

- Large weights typically mean model is overfitting
- Add the size of the weights to our loss function
- Perform well on task + keep weights small

$$\arg_\theta \min \frac{1}{T} \sum_t \mathrm{loss}(f(x^{(t)}; \theta), y^{(t)}))$$

Regularization III: Weight Regularization

- Large weights typically mean model is overfitting
- Add the size of the weights to our loss function
- Perform well on task + keep weights small

$$\arg_\theta \min \frac{1}{T} \sum_t \mathrm{loss}(f(x^{(t)};\theta), y^{(t)}) + \lambda \sum_i (\theta_i)^2$$

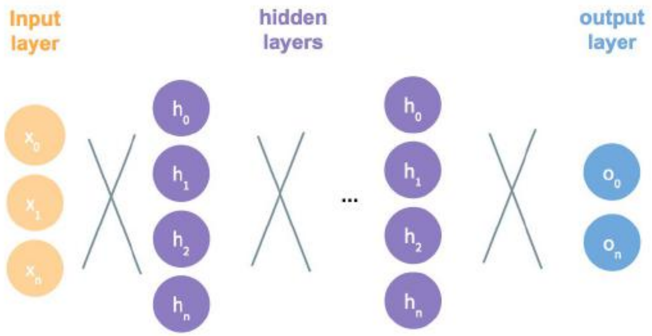
Regularization III: Weight Regularization

- Large weights typically mean model is overfitting
- Add the size of the weights to our loss function
- Perform well on task + keep weights small

$$\underbrace{\arg_\theta \min \frac{1}{T} \sum_t \operatorname{loss}(f(x^{(t)}; \theta), y^{(t)}) + \lambda \sum_i (\theta_i)^2}_{J(\theta)}$$

Core Fundamentals Review

- Perceptron Classifier
- Stacking Perceptrons to form neural networks
- How to formulate problems with neural networks
- Train neural networks with backpropagation
- Techniques for improving training of deep neural networks



Questions?