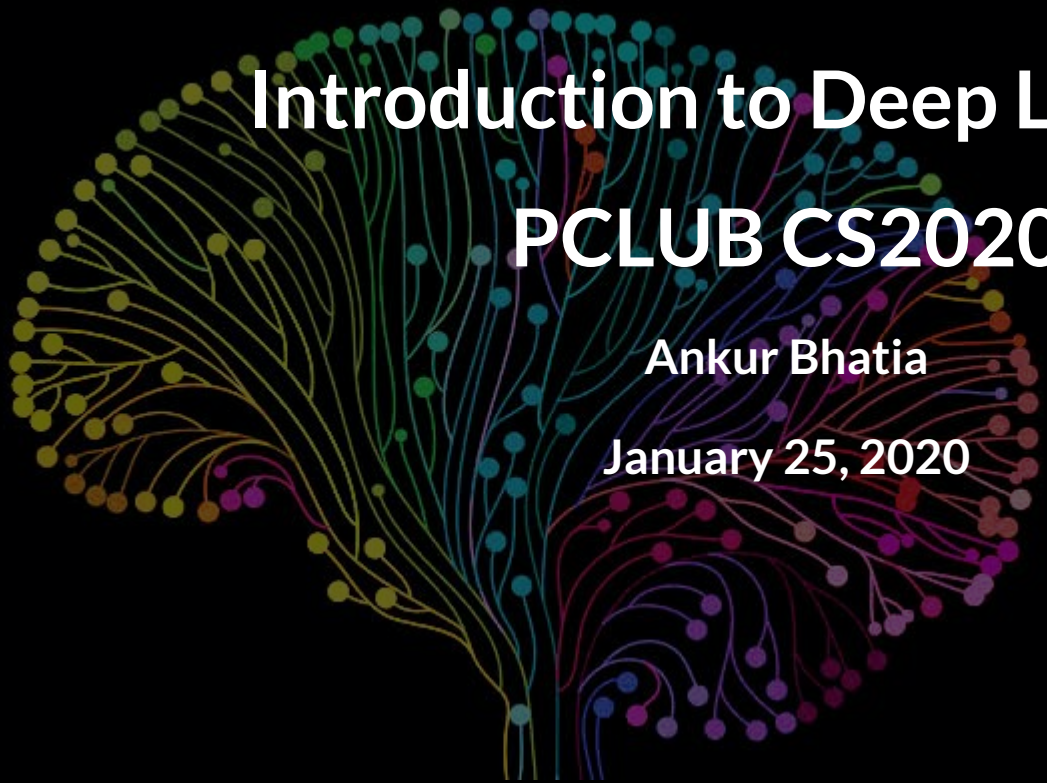
Week 1:

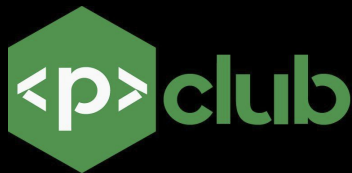
Introduction to Deep Learning

PCLUB CS2020

Ankur Bhatia

January 25, 2020





Course Prerequisites - Linear Algebra
Calculus

The Rise of Deep Learning

6.S191 Introduction to Deep Learning

introtodeeplearning.com

What is Deep Learning?

Artificial Intelligence

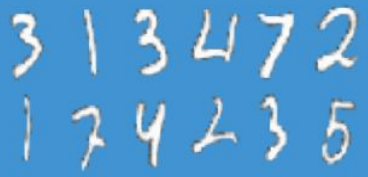
Any technique that enables computers to mimic human behavior

Machine Learning

Ability to learn without explicitly being programmed

Deep Learning

Extract patterns from data using neural networks



Course Outcomes:

1. FINAL CLASS PROJECT
2. PAPER/ IDEA PRESENTATION
3. PROJECTS SHOWCASE ON PCLUB WEBSITE.

Class support

DL- Wiki (Telegram Group)
bit.ly/dlwikipclub

Why Deep Learning and Why Now?

Things we want to do with data

Images

→ Label image

Audio

→ Speech recognition

Text

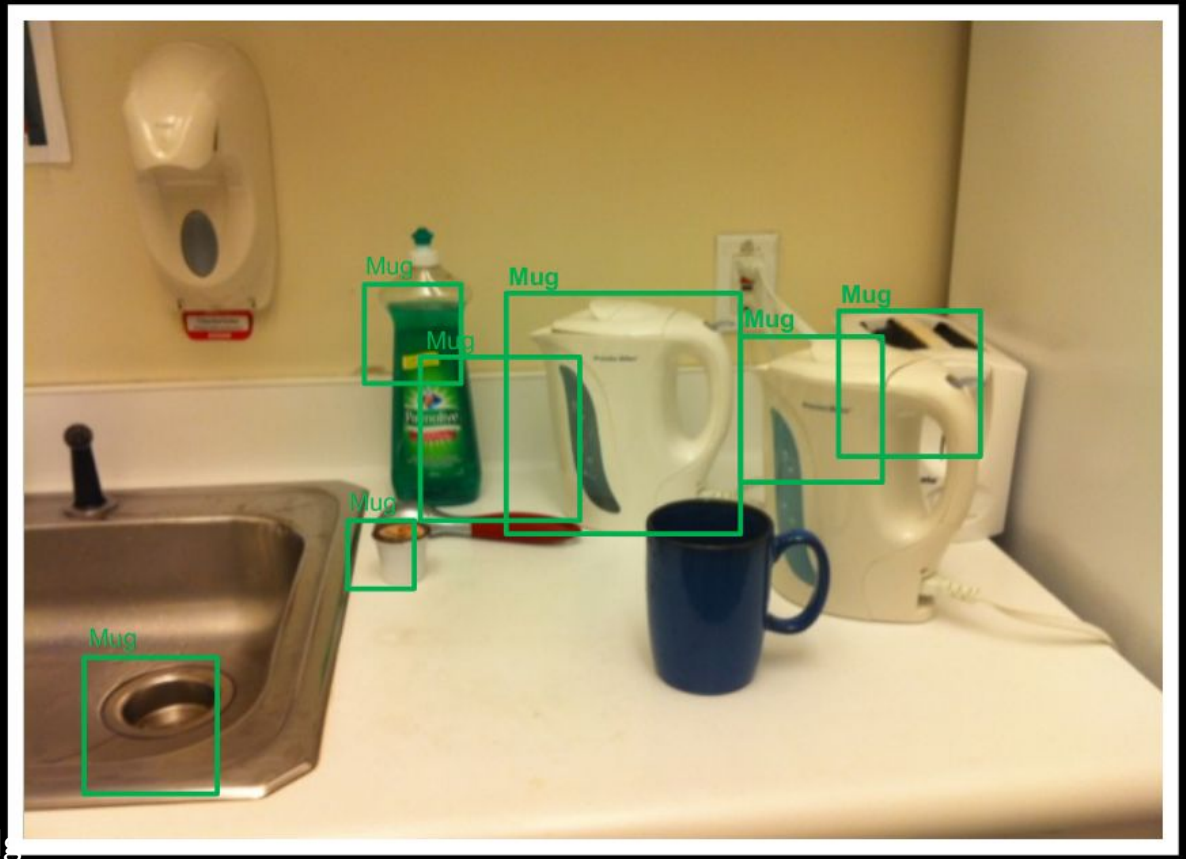
→ Web search

Andrew Ng

Computer vision: Identify coffee mug

Andrew Ng

Computer vision: Identify coffee mug



Andrew Ng

Why is computer vision hard?

The camera sees :

194	210	201	212	199	213	215	195	178	158	182	209
180	189	190	221	209	205	191	167	147	115	129	163
114	126	140	188	176	165	152	140	170	106	78	88
87	103	115	154	143	142	149	153	173	101	57	57
102	112	106	131	122	138	152	147	128	84	58	66
94	95	79	104	105	124	129	113	107	87	69	67
68	71	69	98	89	92	98	95	89	88	76	67
41	56	68	99	63	45	60	82	58	76	75	65
20	43	69	75	56	41	51	73	55	70	63	44
50	50	57	69	75	75	73	74	53	68	59	37
72	59	53	66	84	92	84	74	57	72	63	42
67	61	58	65	75	78	76	73	59	75	69	50

Andrew Ng

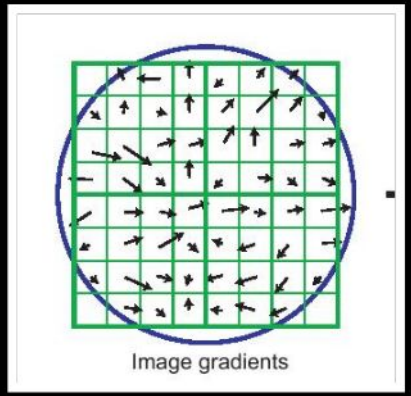
Computer vision

Feature representation

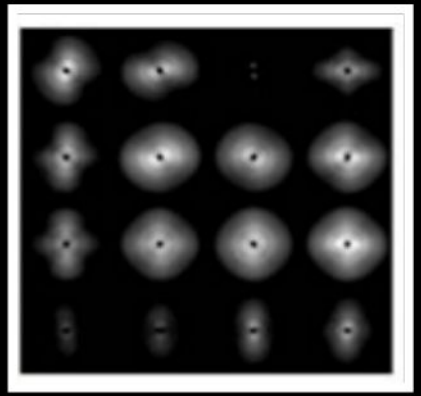
Learning algorithm

Andrew Ng

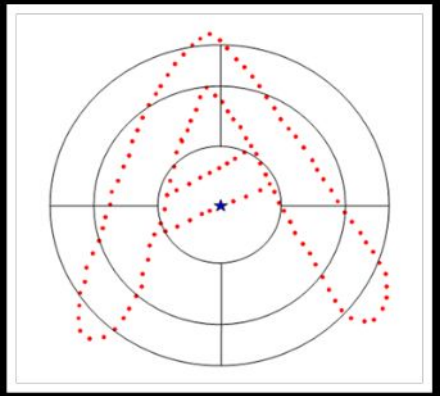
Features for vision



SIFT



GIST



Shape context

Features for machine learning

Images

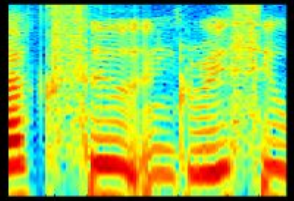
Image

Vision features

Detection

Audio

Audio

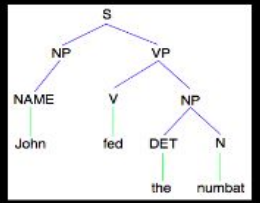


Audio features

Speaker ID

Text

Text



Text features

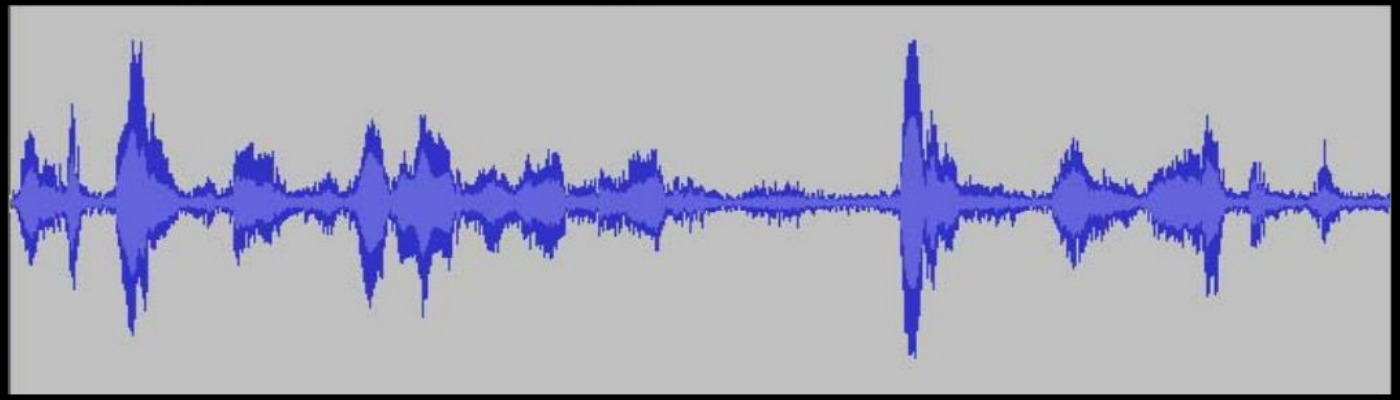
Web search

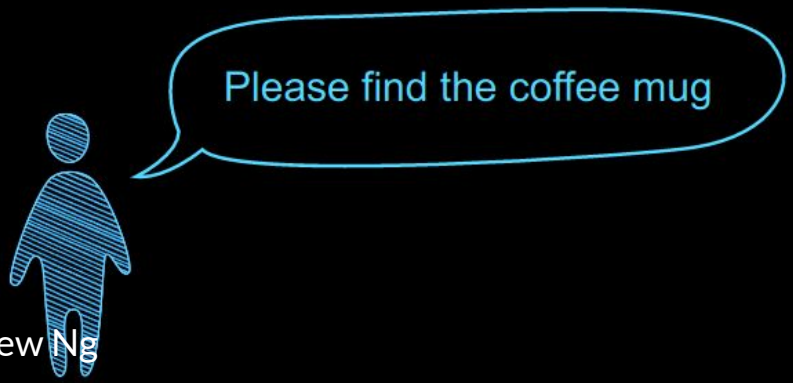
…

Andrew Ng

Why is speech recognition hard?

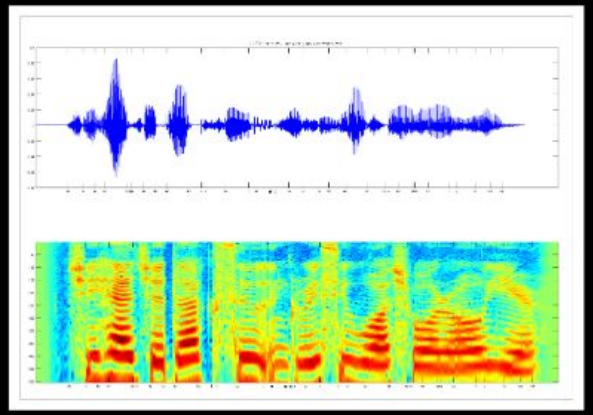
Microphone recording:



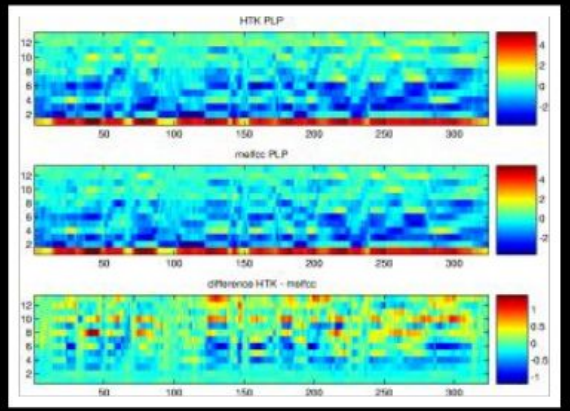


Andrew Ng

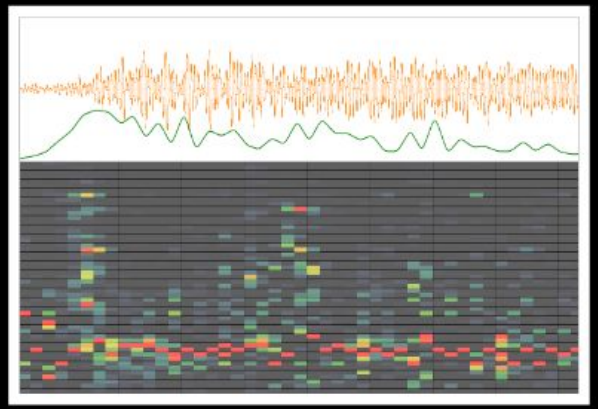
Features for audio



Spectrogram



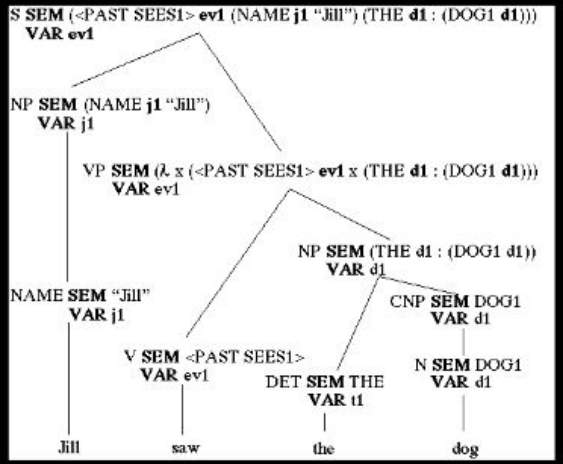
MFCC



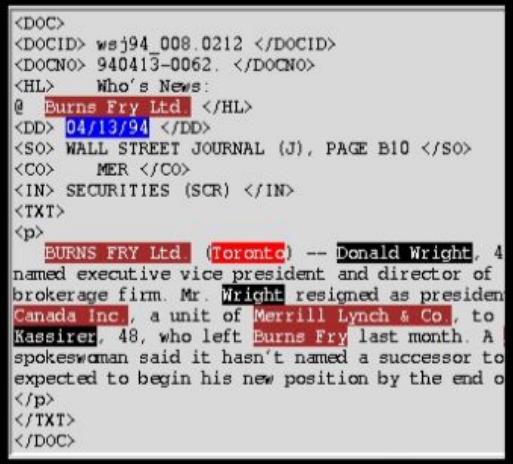
Flux

Andrew Ng

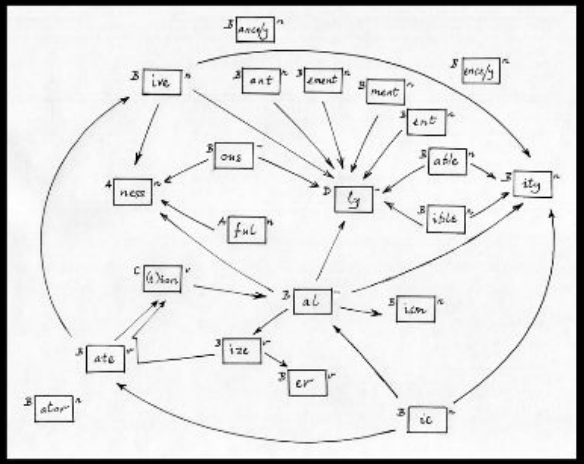
Features for text



Parser



Named entity



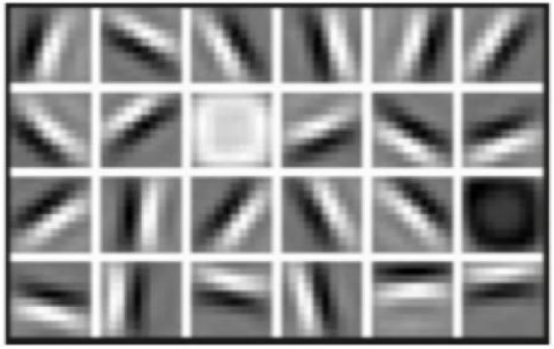
Stemming

Why Deep Learning?

Hand engineered features are time consuming, brittle and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features

Eyes & Nose & Ears

High Level Features

Facial Structure

Why Now?

1952 Stochastic Gradient Descent

1958 Perceptron
- Learnable Weights

1986 Backpropagation
- Multi-Layer Perceptron

1995 Deep Convolutional NN
- Digit Recognition

Neural Networks date back decades, so why the resurgence?

1. Big Data
- Larger Datasets
- Easier Collection & Storage

IMAGENET

WIKIPEDIA
The Free Encyclopedia

2. Hardware
- Graphics Processing Units (GPUs)
- Massively Parallelizable

3. Software
- Improved Techniques
- New Models
- Toolboxes

TensorFlow

6.S191 Introduction to Deep Learning
introtodeeplearning.com

Deep Learning Applications?

Everyone knows it.

Let’s see one state of the art one:

https://ai.googleblog.com/2018/05/duplex-ai-system-for-natural-conversation.html

Deep Learning in **One** Slide

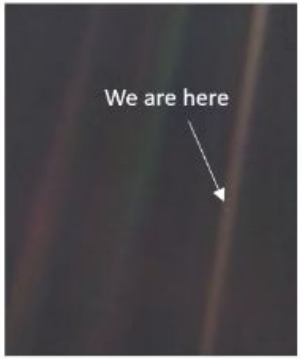
- **What is it:**
  Extract useful patterns from data.
- **How:**
  Neural network + optimization
- **How (Practical):**
  Python + TensorFlow & friends
- **Hard Part:**
  Good Questions + Good Data
- **Why now:**
  Data, hardware, community, tools, investment
- **Where do we stand?**
  Most big questions of intelligence have not been answered nor properly formulated

Exciting progress:

- Face recognition
- Image classification
- Speech recognition
- Text-to-speech generation
- Handwriting transcription
- Machine translation
- Medical diagnosis
- Cars: drivable area, lane keeping
- Digital assistants
- Ads, search, social recommendations
- Game playing with deep RL

Deep Learning
What now in 2020?

Deep Learning & AI in Context of Human History



Perspective:

- Universe created
  13.8 billion years ago
- Earth created
  4.54 billion years ago
- Modern humans
  300,000 years ago
- Civilization
  12,000 years ago
- Written record
  5,000 years ago

1700s and beyond: Industrial revolution, steam engine, mechanized factory systems, machine tools

Artificial Intelligence in Context of Human History

Perspective:

- Universe created
  13.8 billion years ago
- Earth created
  4.54 billion years ago
- Modern humans
  300,000 years ago
- Civilization
  12,000 years ago
- Written record
  5,000 years ago

Dreams, mathematical foundations, and engineering in reality.

Alan Turing, 1951: "It seems probable that once the machine thinking method had started, it would not take long to outstrip our feeble powers. They would be able to converse with each other to sharpen their wits. At some stage therefore, we should have to expect the machines to take control."

- 1943: Neural networks
- 1957-62: Perceptron
- 1970-86: Backpropagation, RBM, RNN
- 1979-98: CNN, MNIST, LSTM, Bidirectional RNN
- 2006: “Deep Learning”, DBN
- 2009: ImageNet + AlexNet
- 2014: GANs
- 2016-17: AlphaGo, AlphaZero
- 2017: 2017-19: Transformers

* Dates are for perspective and not as definitive historical record of invention or credit

Turing Award for Deep Learning

- Yann LeCun
- Geoffrey Hinton
- Yoshua Bengio

Turing Award given for:

- “The conceptual and engineering breakthroughs that have made deep neural networks a critical component of computing.”

Hopes for 2020

- **Less Hype & Less Anti-Hype:** Less tweets on how there is too much hype in AI and more solid research in AI.
- **Hybrid Research:** Less contentious, counter-productive debates, more open-minded interdisciplinary collaboration.
- **Research topics:**
  - Reasoning
  - Active learning and life-long learning
  - Multi-modal and multi-task learning
  - Open-domain conversation
  - Applications: medical, autonomous vehicles
  - Algorithmic ethics
  - Robotics

Now Let’s Dive in?

Tagged vs. untagged data

Coffee mug

Coffee mug

Coffee mug

Coffee mug

Coffee mug

Coffee mug

Andrew Ng

Untagged data (unsupervised learning)

Unknown

Unknown

Unknown

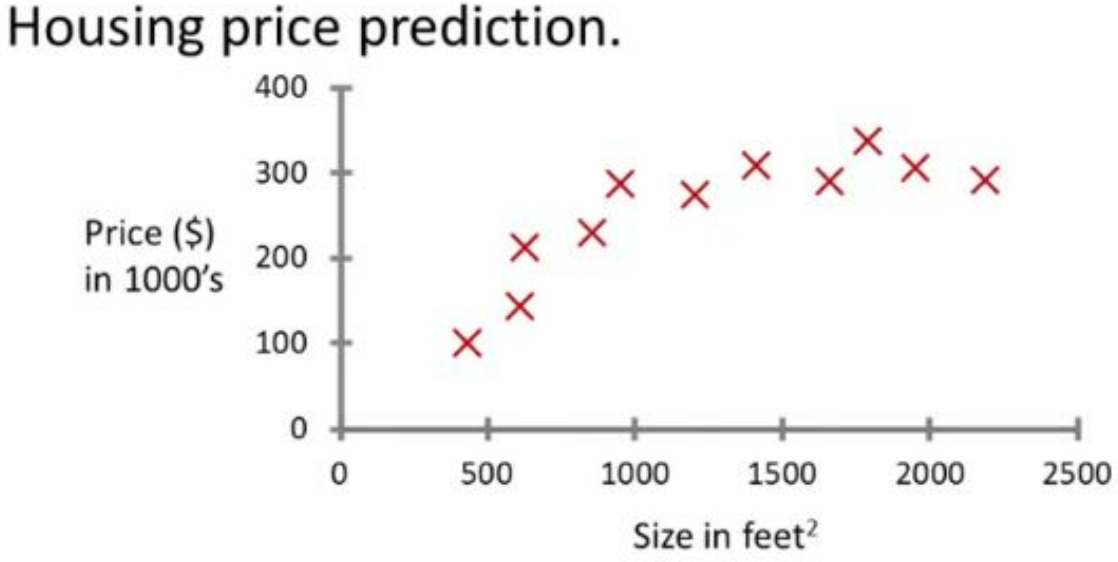
Unknown

Unknown

Unknown

Andrew Ng

Supervised Learning
EXAMPLE 1

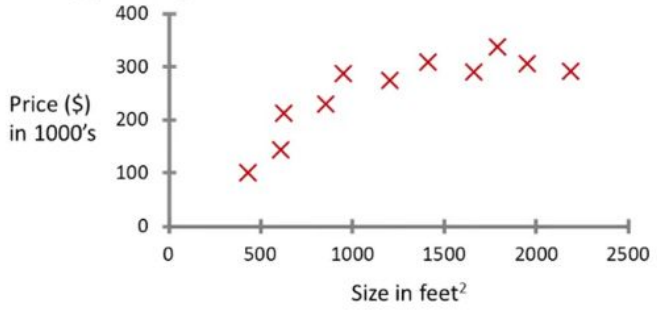


Supervised Learning
EXAMPLE 2

FEATURE 1

Linear Regression:

Housing price prediction.



Hypothesis: $h_\theta(x) = \theta_0 + \theta_1 x$

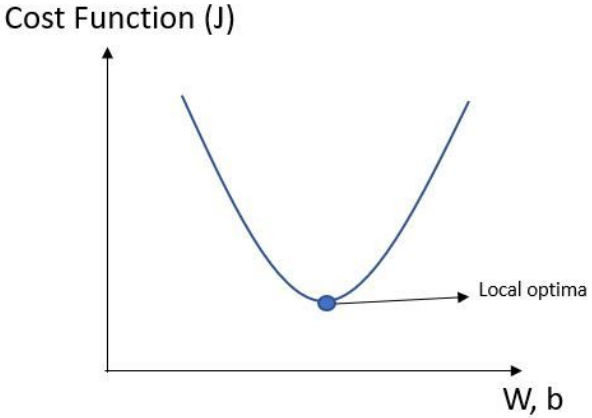
Parameters: θ_0, θ_1

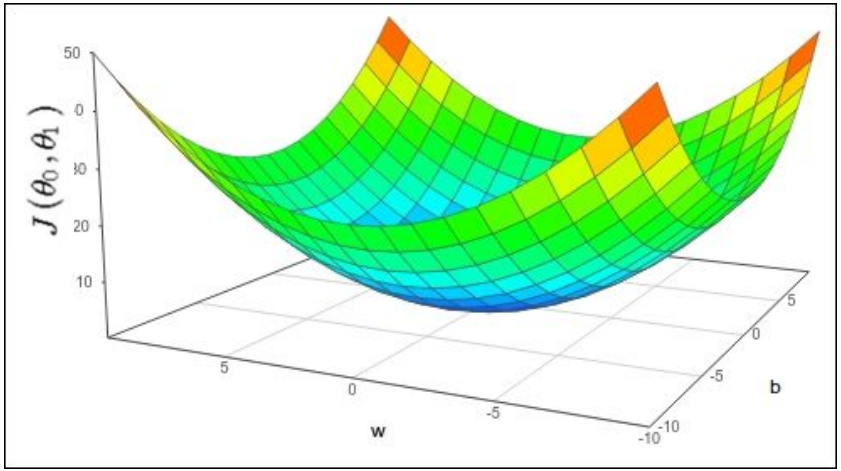
Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

Goal: $\underset{\theta_0, \theta_1}{\text{minimize}}\ J(\theta_0, \theta_1)$

1. Notations
2. Hypothesis function and its parameters
3. Cost Function
   a. Single parameter
   b. Multiple parameters
4. Contour plots points of various h(theta0, theta1)
5. Minimize cost function

Cost Function:

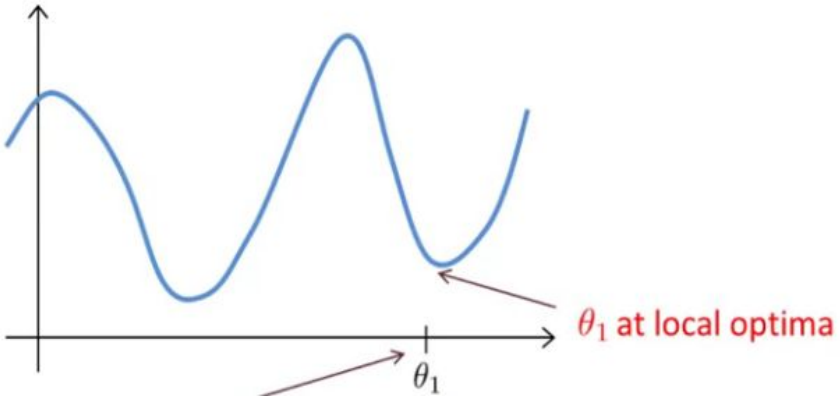


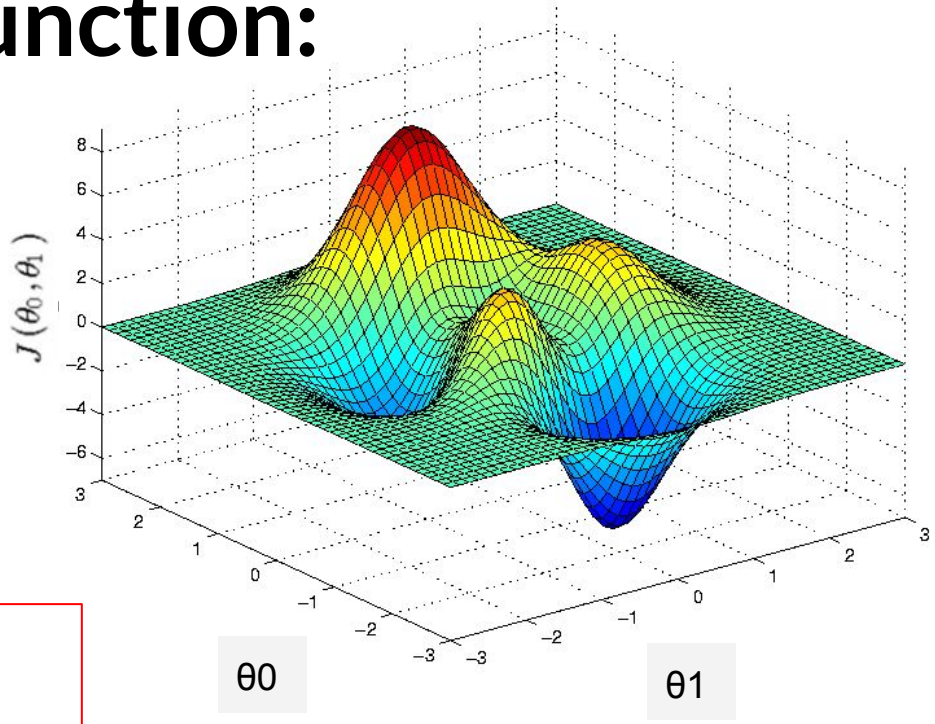


Convex Function

1. Notations
2. Hypothesis function and its parameters
3. Cost Function
   a. Single parameter
   b. Multiple parameters
4. Contour plots points of various h(theta0, theta1)
5. Minimize cost function

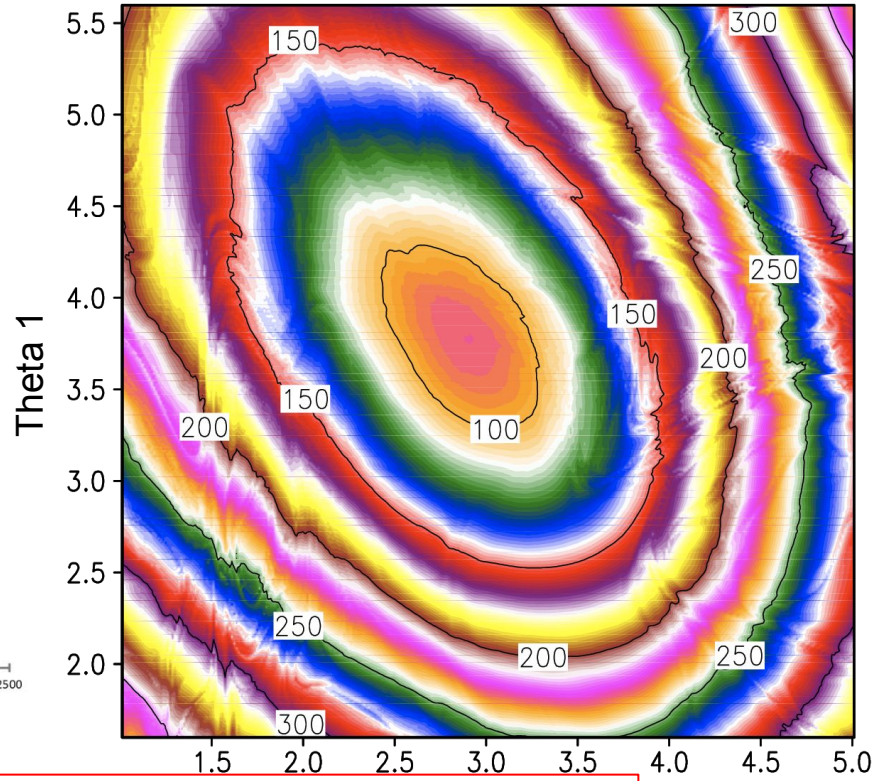
Cost Function:



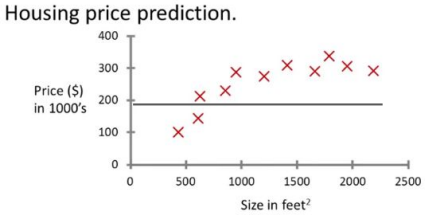


1. Notations
2. Hypothesis function and its parameters
3. Cost Function
   a. Single parameter
   b. Multiple parameters
4. Contour plots points of various h(theta0, theta1)
5. Minimize cost function

Cost Function contour:



What does a point denote in this?



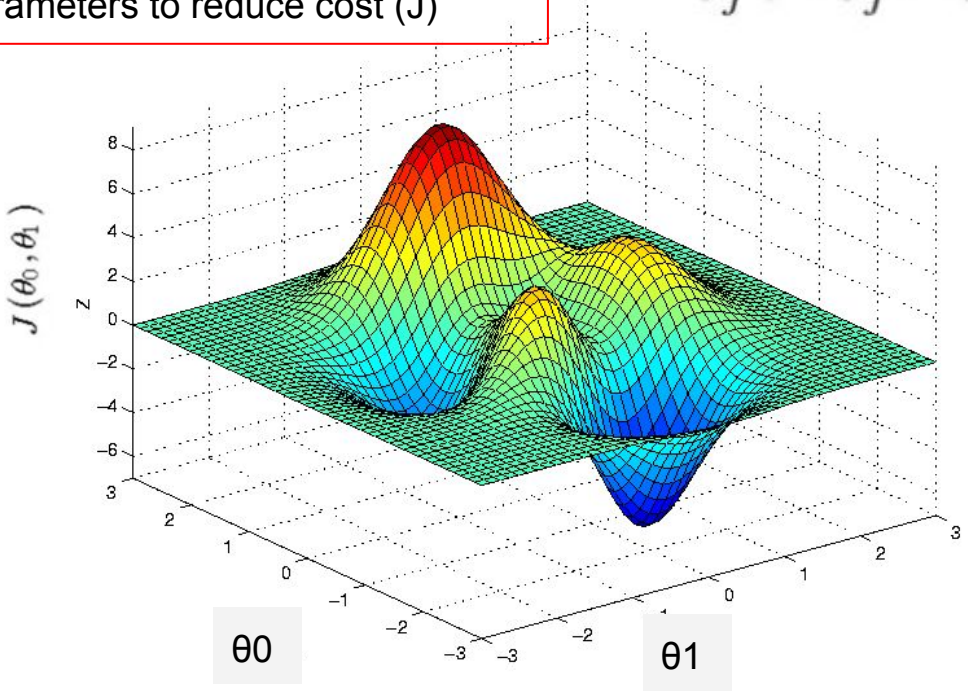
Show: Contour plots points of various h(theta0, theta1)

Gradient Descent “Batch”:

Task: Minimize J(θ0, θ1, θ2, ……., θn)

1. Start with some initial parameter value θ1, θ2
2. Change the parameters to reduce cost (J)

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$



Gradient Descent for Linear Reg:

Task: Minimize J(θ0, θ1) = 1/2m (Σ(h(x) - y)^2)

1. **Compute the derivative of cost function wrt. each of the parameters.**
2. **Put into the gradient descent algorithm.**
3. **Stop when get reached the convergence.**

$$j = 0 : \frac{\partial}{\partial \theta_0} J(\theta_0, \theta_1)$$

$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\text{repeat until convergence } \{$$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$

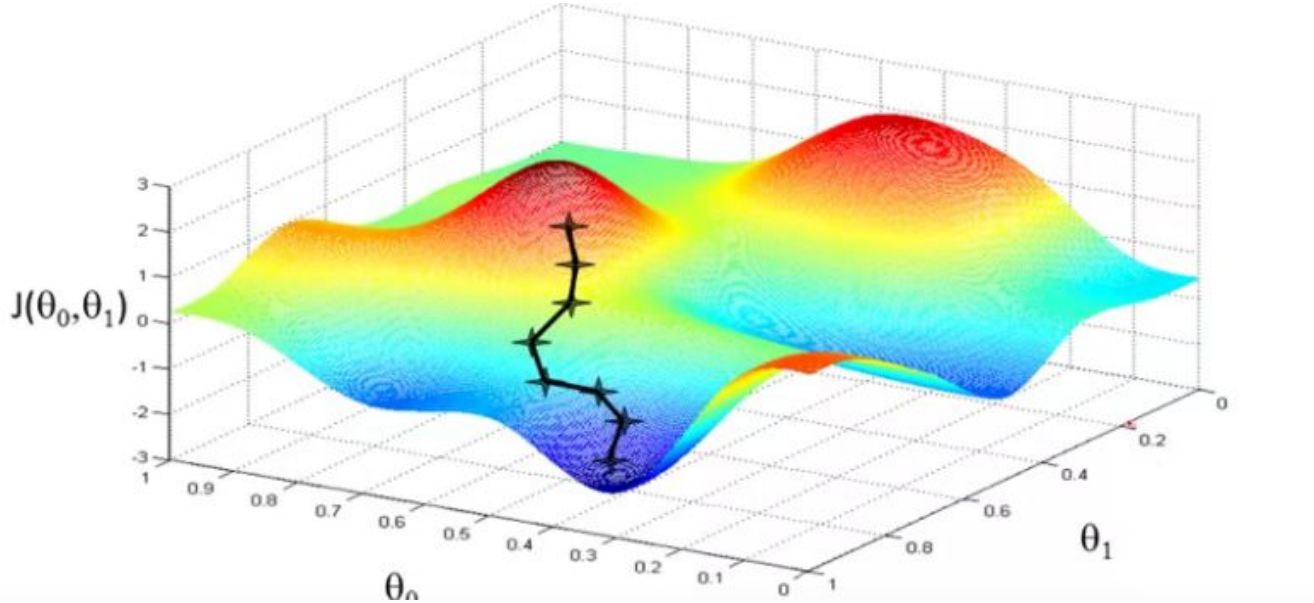
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$

$$\}$$

Gradient Descent for Linear Reg:

Task: Minimize J(θ0, θ1) = 1/2m (Σ(h(x) - y)^2)

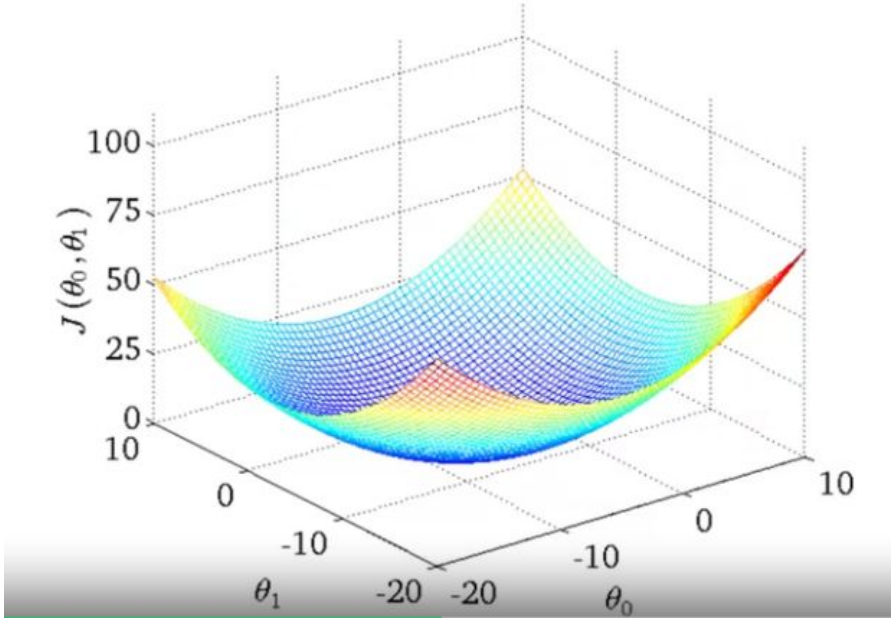
Q. What happens if algo gets stuck in a local minima?



Gradient Descent for Linear Reg:

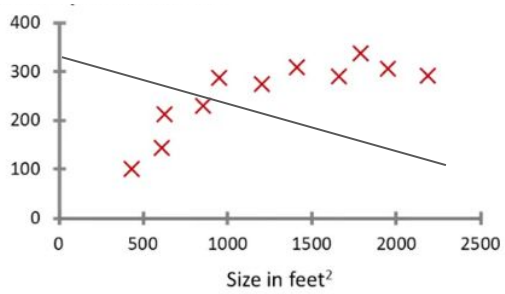
Task: Minimize J(θ0, θ1) = 1/2m (Σ(h(x) - y)^2)

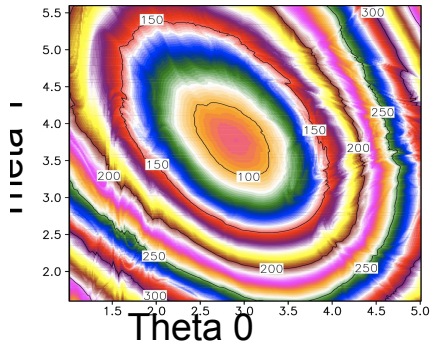
Q. What happens if algo gets stuck in a local minima?
Ans. For Linear Reg. , we have a bowl type cost function (Convex function).
Optimization problem is convex here.

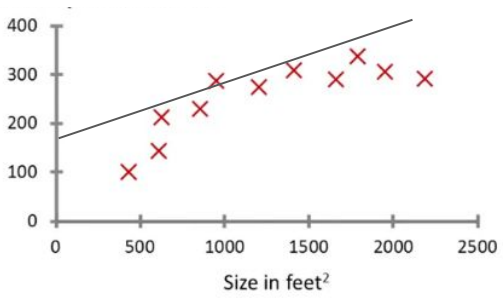


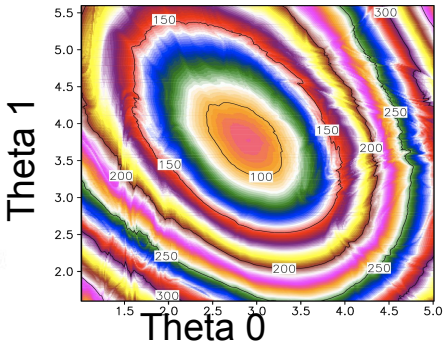
Gradient Descent for Linear Reg:

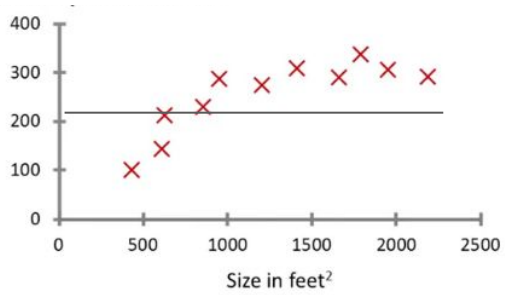
Task: Minimize J(θ0, θ1) = 1/2m (Σ(h(x) - y)^2)

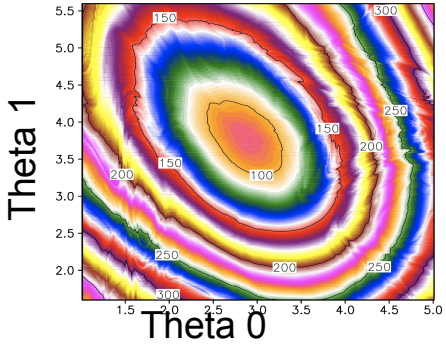


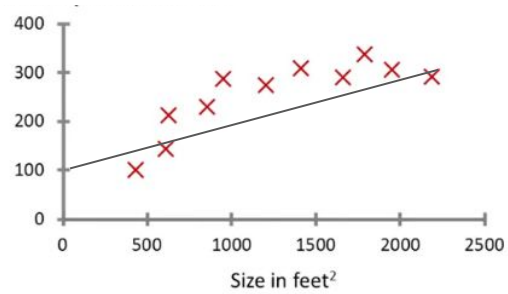


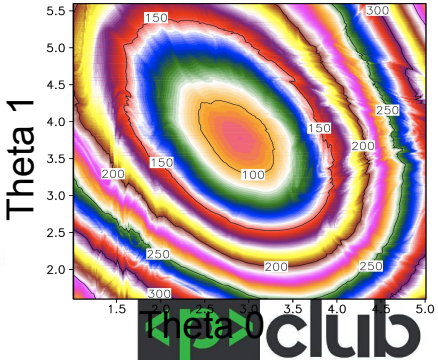




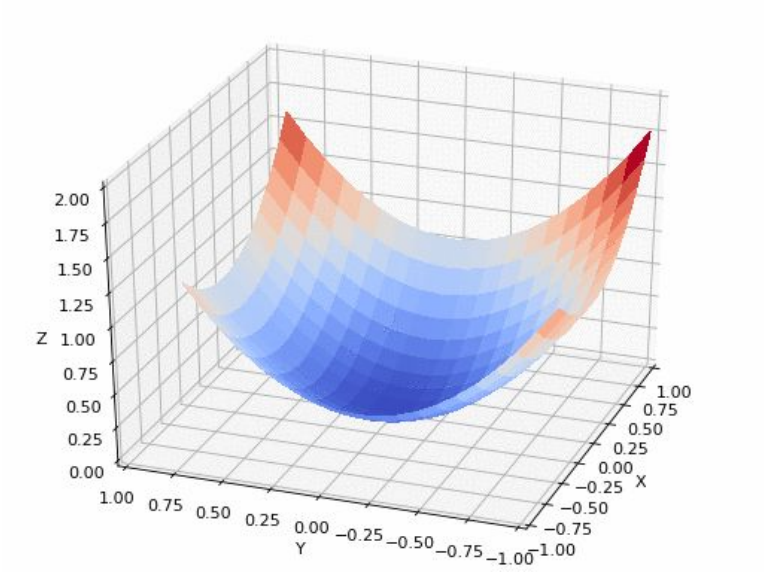


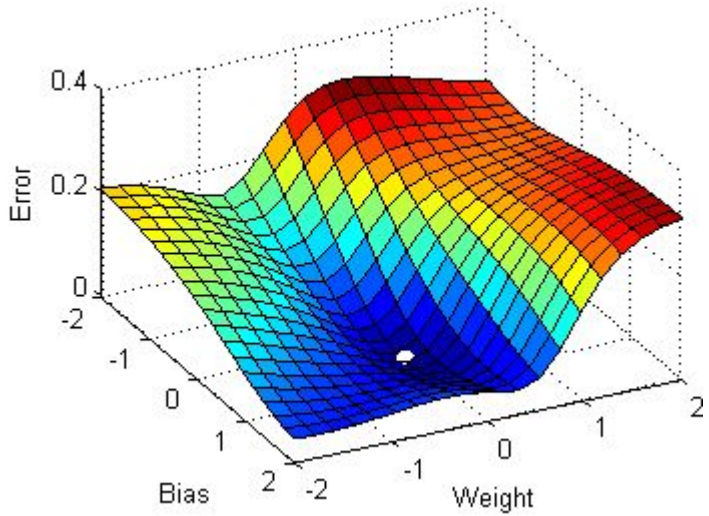




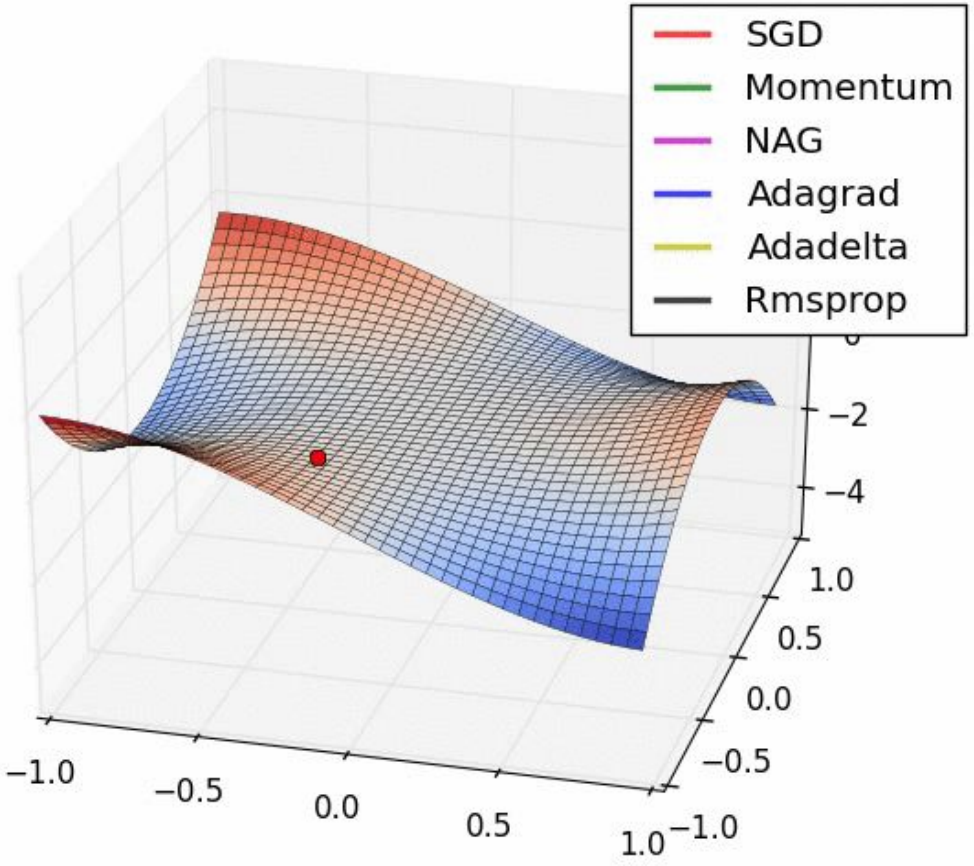


Gradient Descent for Linear Reg:



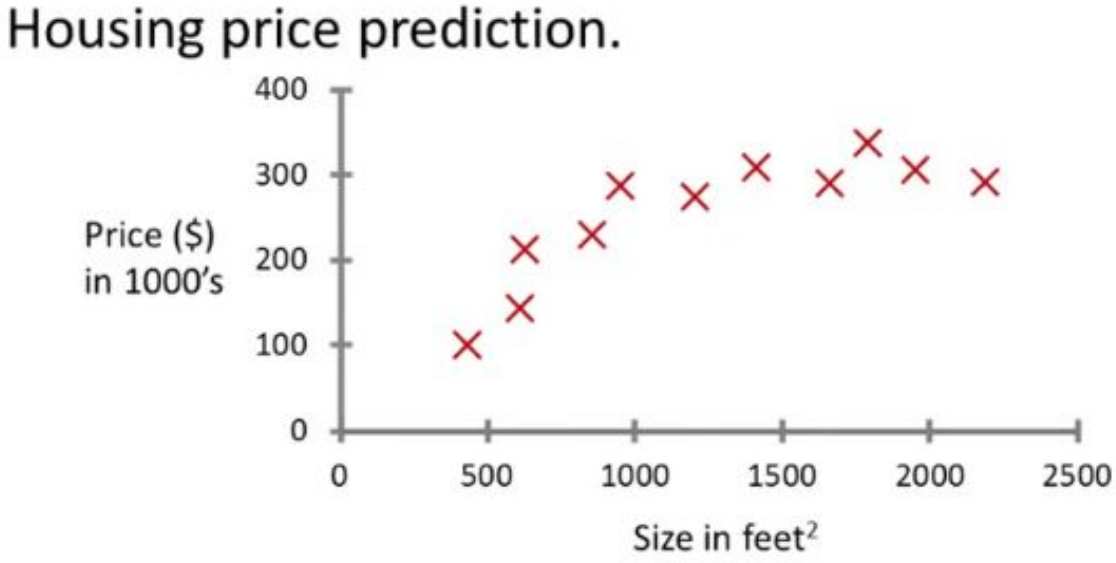


Various Descent's:

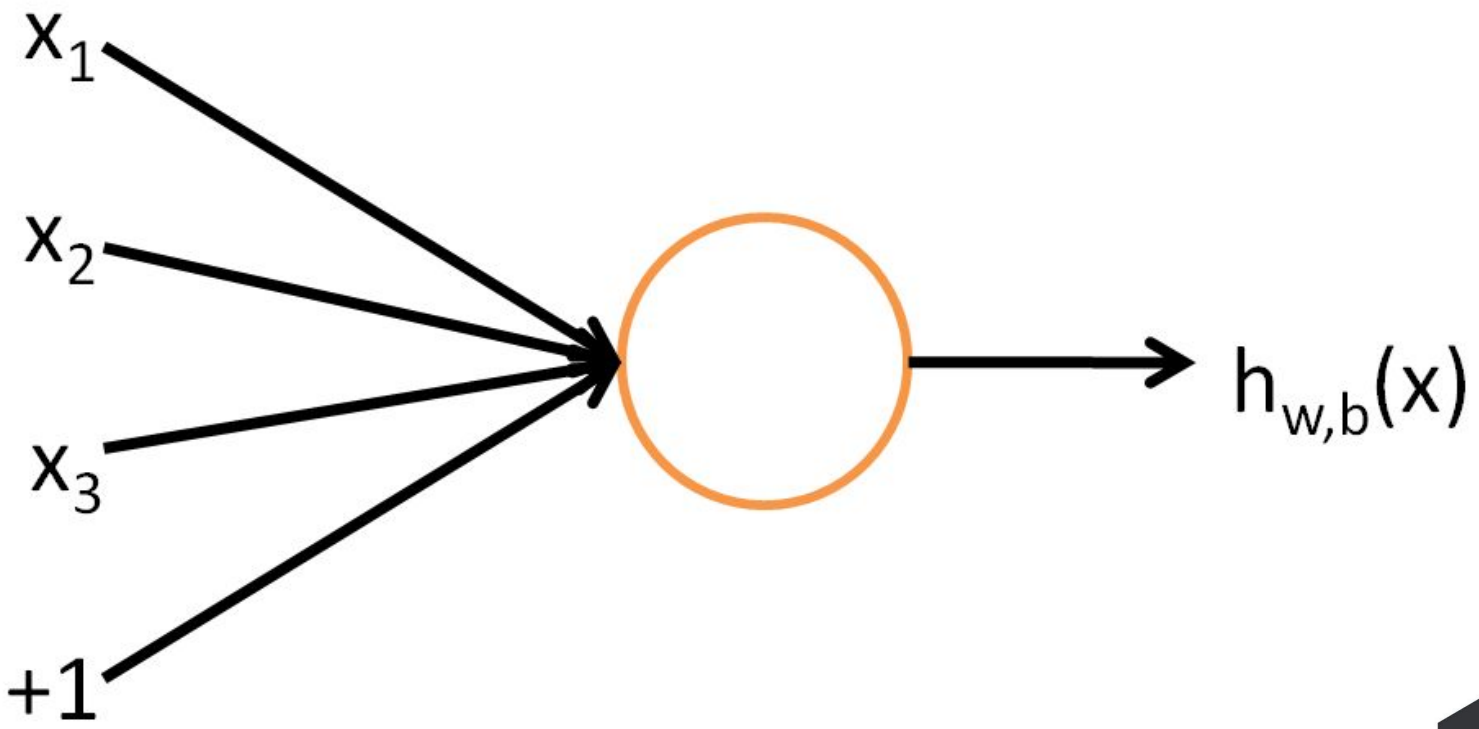


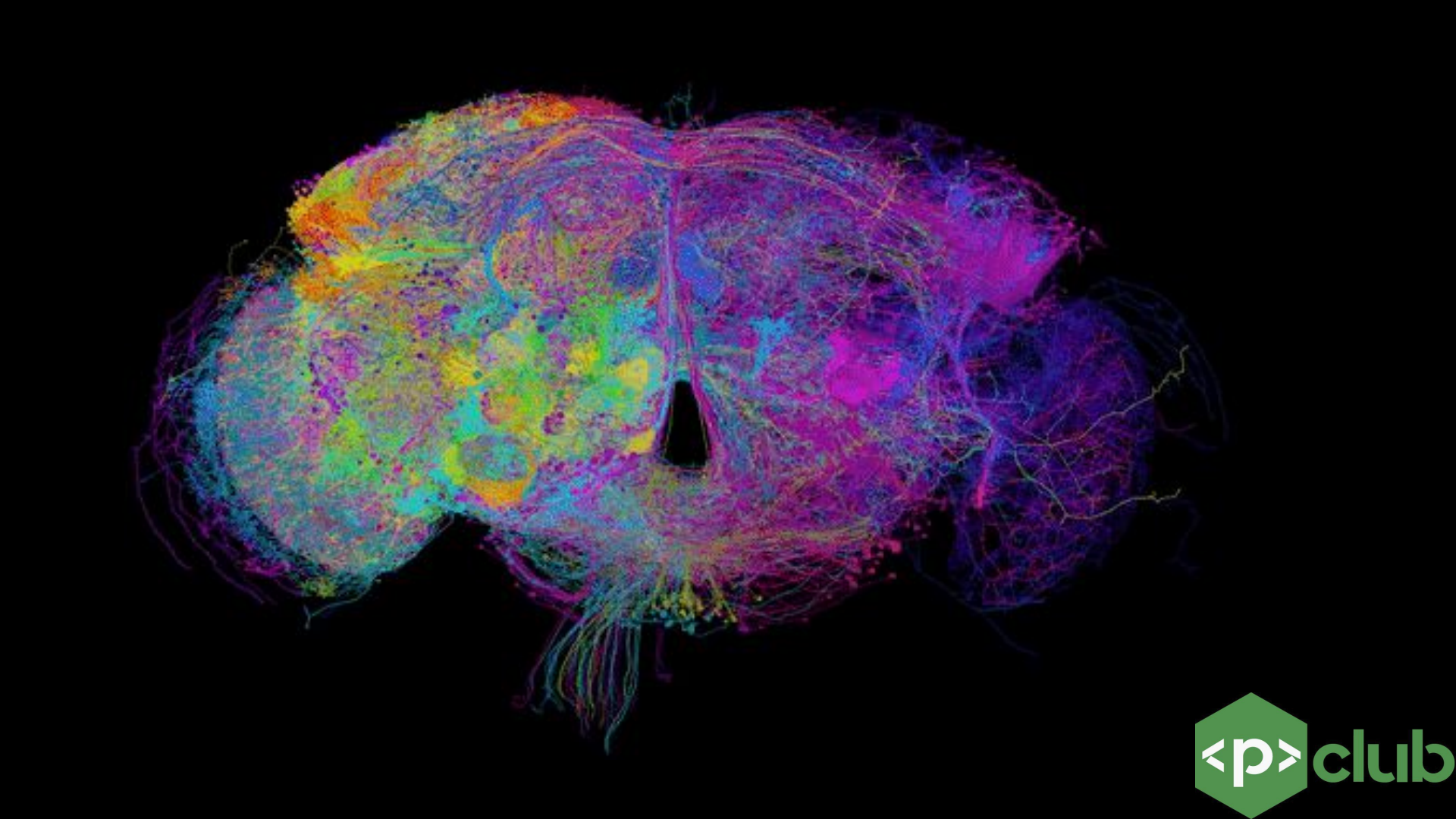
Now Let’s Dive a bit more Deep.

What is a Neural Network?



What is a Neural Network?

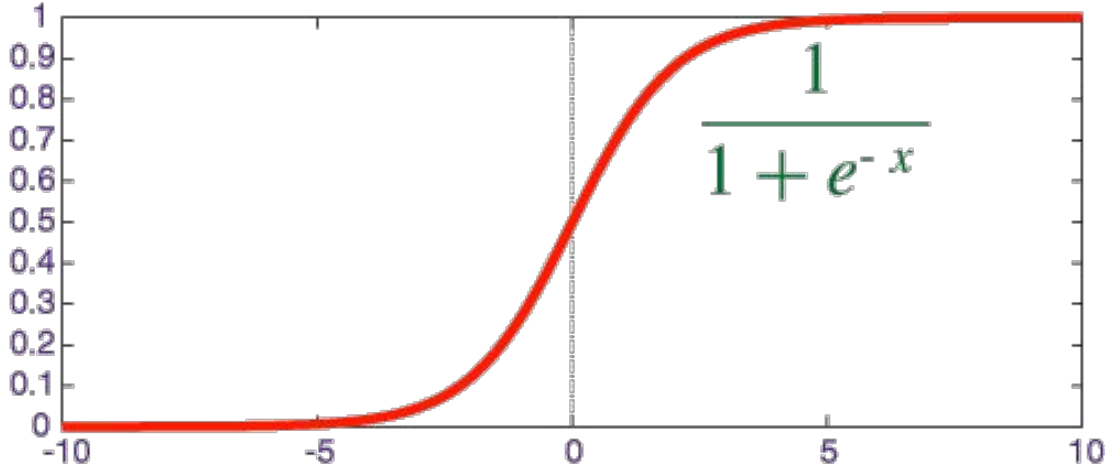



<p>club

Binary Classification problem:

Email - Spam/Not spam
Ad - Clicked/Not clicked

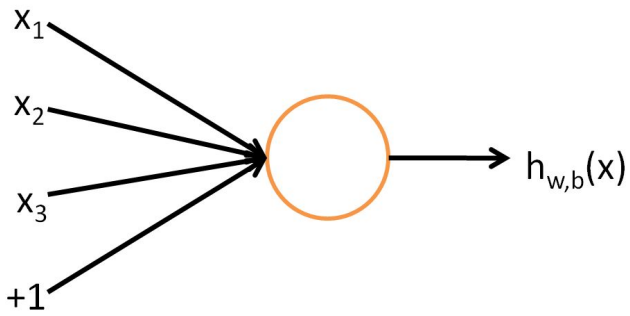
Understanding Logistic Regression problem:



Used in Supervised Learning
When outputs are 0/1 (BC)

Logistic Regression loss function:

Log Loss / Binary Cross Entropy Loss



$$\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

Gradient Descent may not find a global optimum - (yhat - y)^2

Logistic Regression Cost function:

$$J(w,b) = \frac{1}{m}\sum_{i=1}^{m}\mathcal{L}(\hat{y}^{(i)}, y^{(i)}) = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)})$$

This cost function allows a convex optimization.

Logistic Regression Gradient Descent:

Backprop:

On 1 Training example

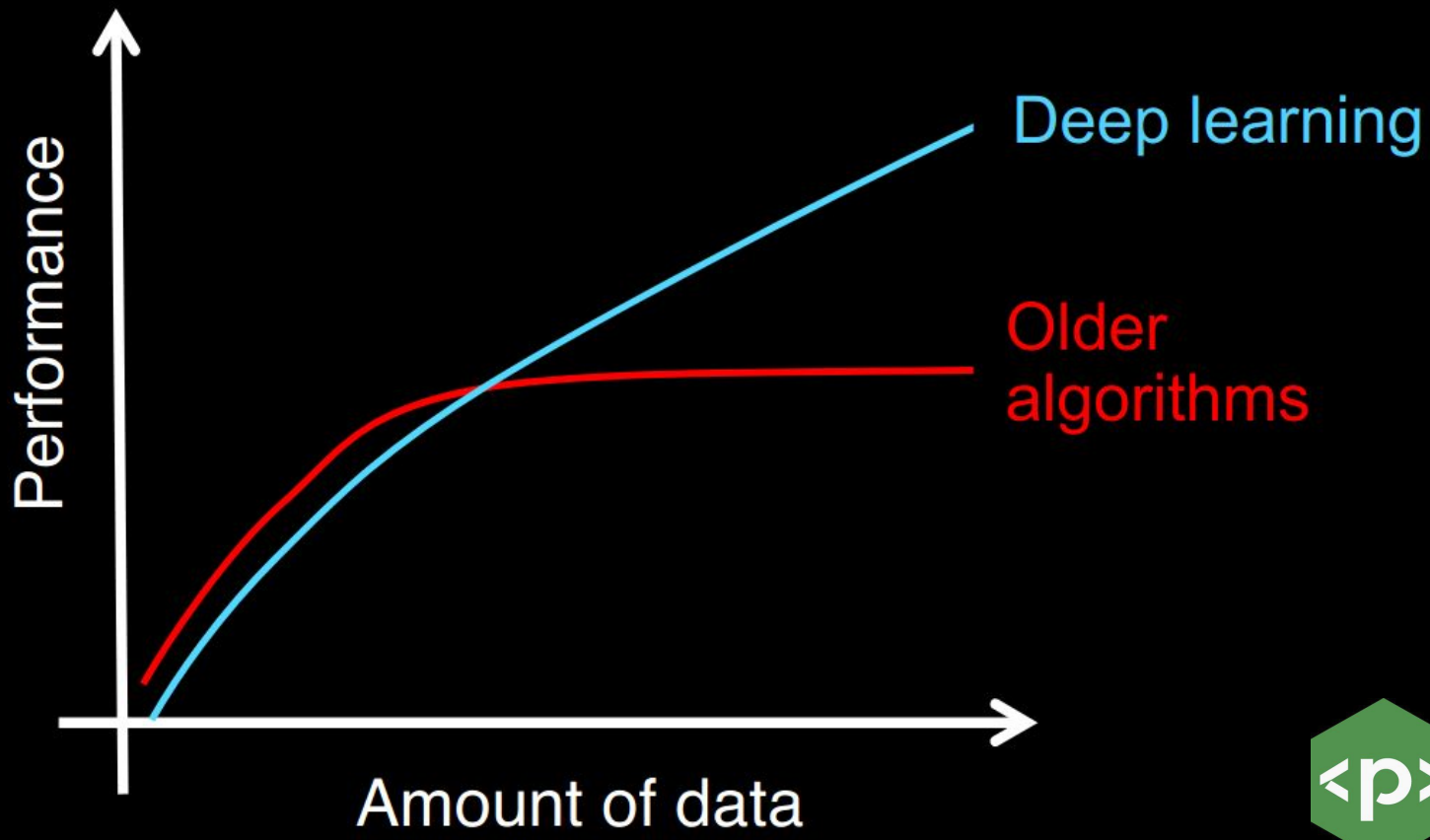
Logistic Regression Gradient Descent:

Backprop:

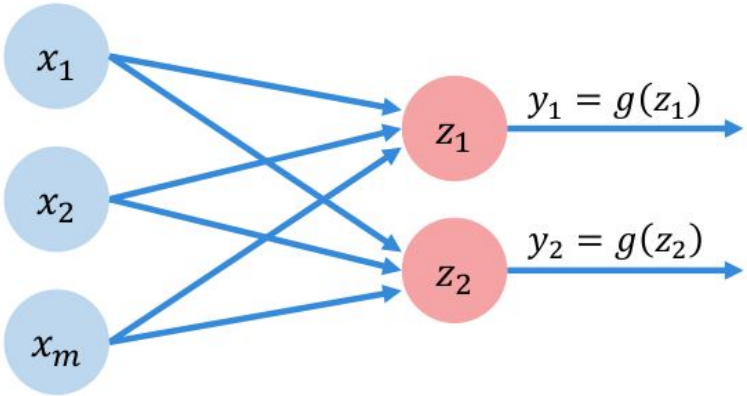
On m Training example

Now Let’s Dive Deeeeeeeeep in?

Learning from tagged data

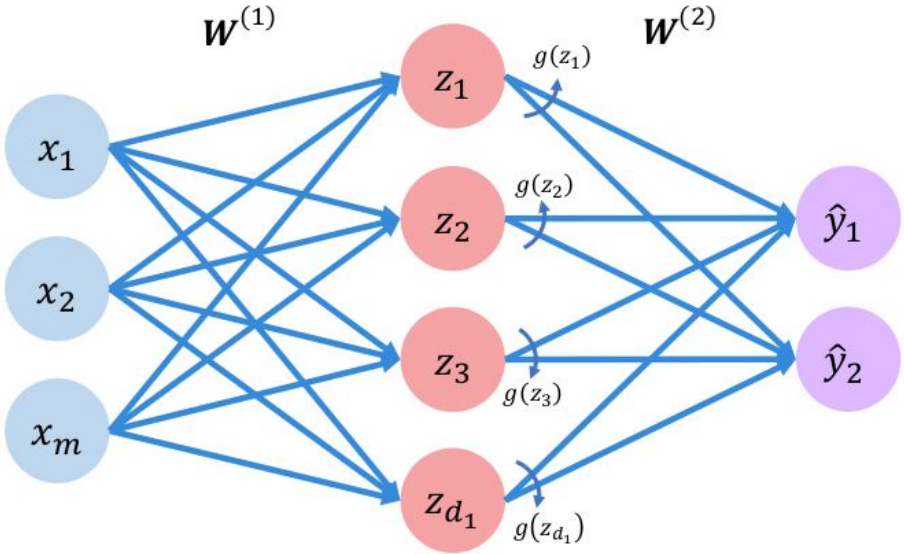


Multi Output Perceptron



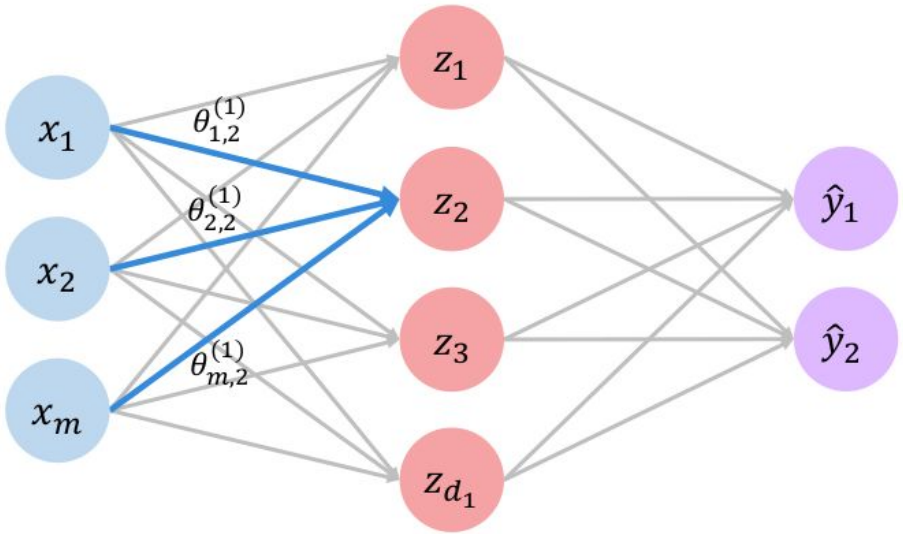
$$z_i = w_{0,i} + \sum_{j=1}^{m} x_j \, w_{j,i}$$

Single Layer Neural Network



$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,i}^{(1)} \qquad \hat{y}_i = g\left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j \, w_{j,i}^{(2)} \right)$$

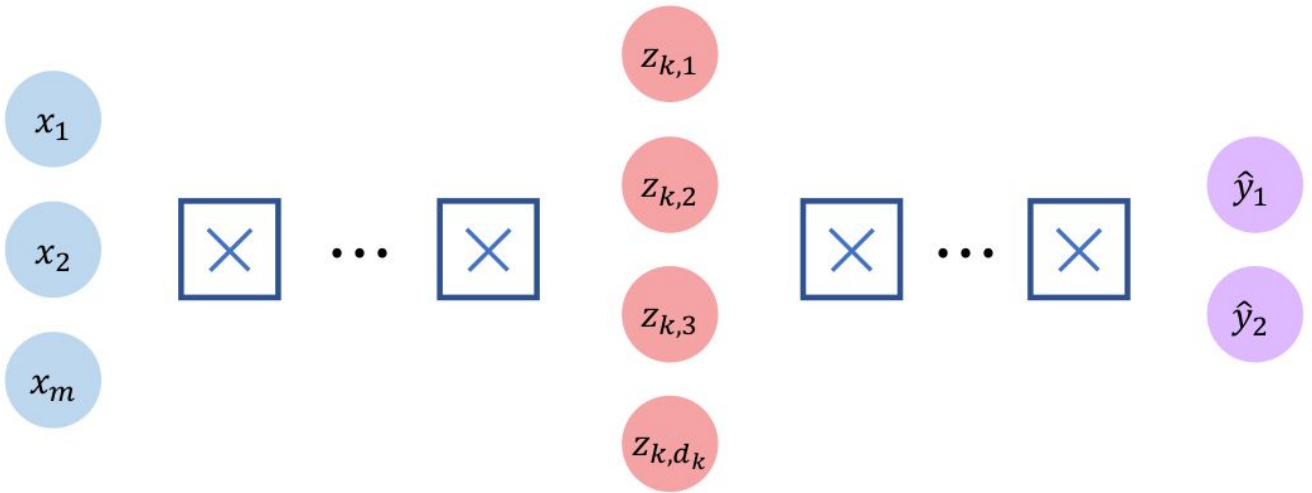
Single Layer Neural Network



$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

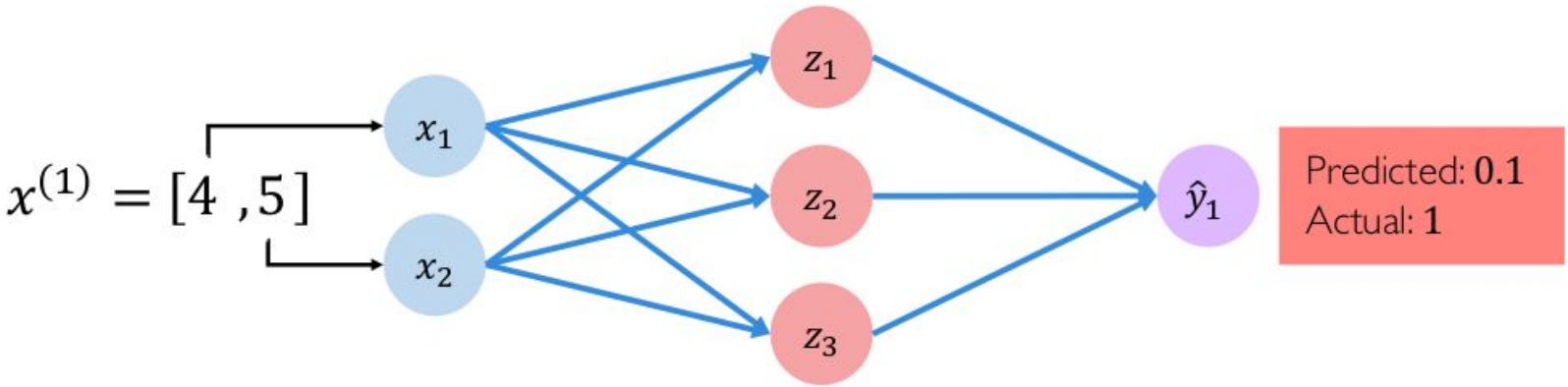
$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

Deep Neural Network



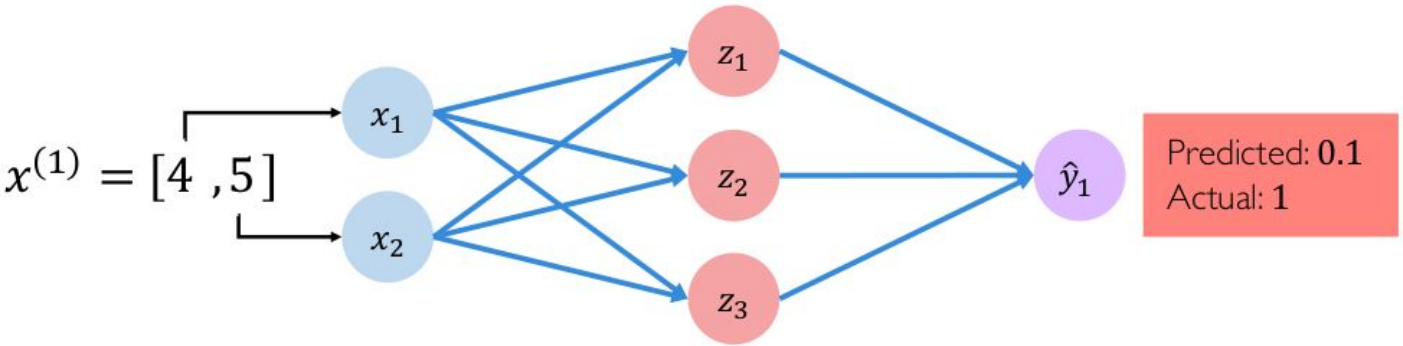
$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Example Problem: Will I pass this class?



Quantifying Loss

*The **loss** of our network measures the cost incurred from incorrect predictions*



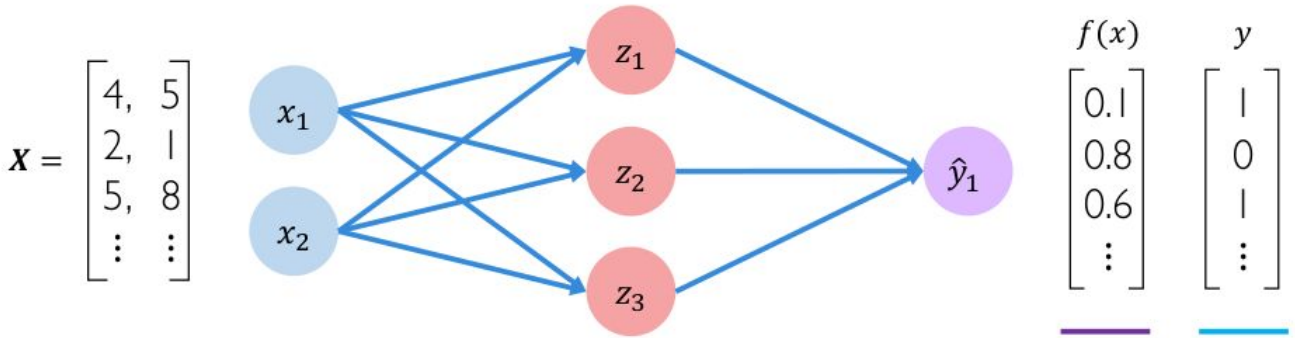
$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Empirical Loss

(Cost Function)

*The **empirical loss** measures the total loss over our entire dataset*



$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

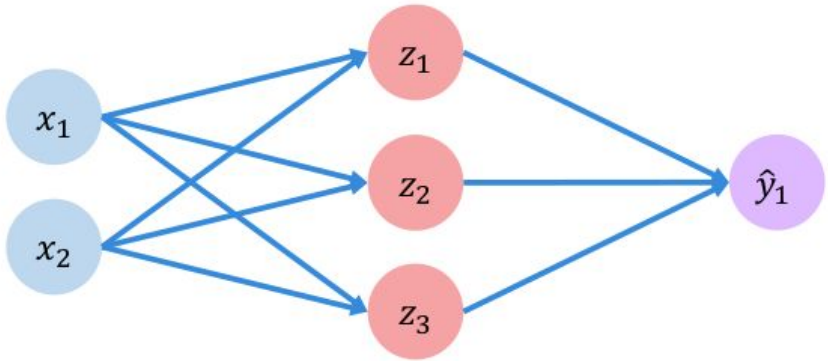
Predicted Actual

Also known as:
- Objective function
- Cost function
- Empirical Risk

Binary Cross Entropy Loss

Cross entropy loss *can be used with models that output a probability between 0 and 1*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



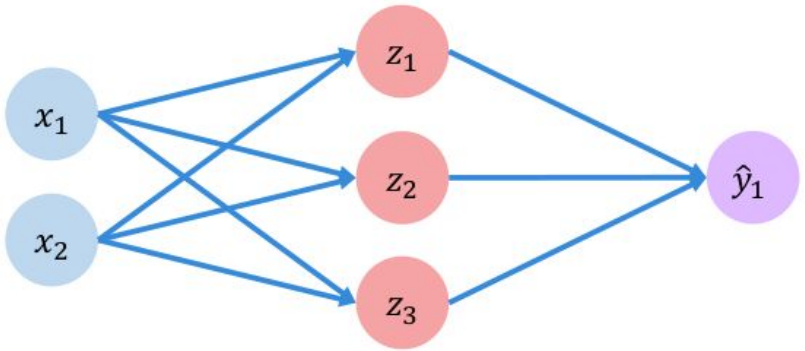
$$f(x) = \begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \qquad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

$$\boldsymbol{J(W)} = \frac{1}{n}\sum_{i=1}^{n} \underbrace{y^{(i)}}_{\text{Actual}} \log\Big(\underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}\Big) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log\Big(1 - \underbrace{f\big(x^{(i)}; \boldsymbol{W}\big)}_{\text{Predicted}}\Big)$$

Mean Squared Error Loss

Mean squared error loss *can be used with regression models that output continuous real numbers*

$$\boldsymbol{X} = \begin{bmatrix} 4, & 5 \\ 2, & 1 \\ 5, & 8 \\ \vdots & \vdots \end{bmatrix}$$



$f(x)$ y

$$\begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} \quad \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix}$$

Final Grades (percentage)

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - f\left(x^{(i)}; \boldsymbol{W}\right)\right)^2$$

Actual Predicted

Loss Optimization

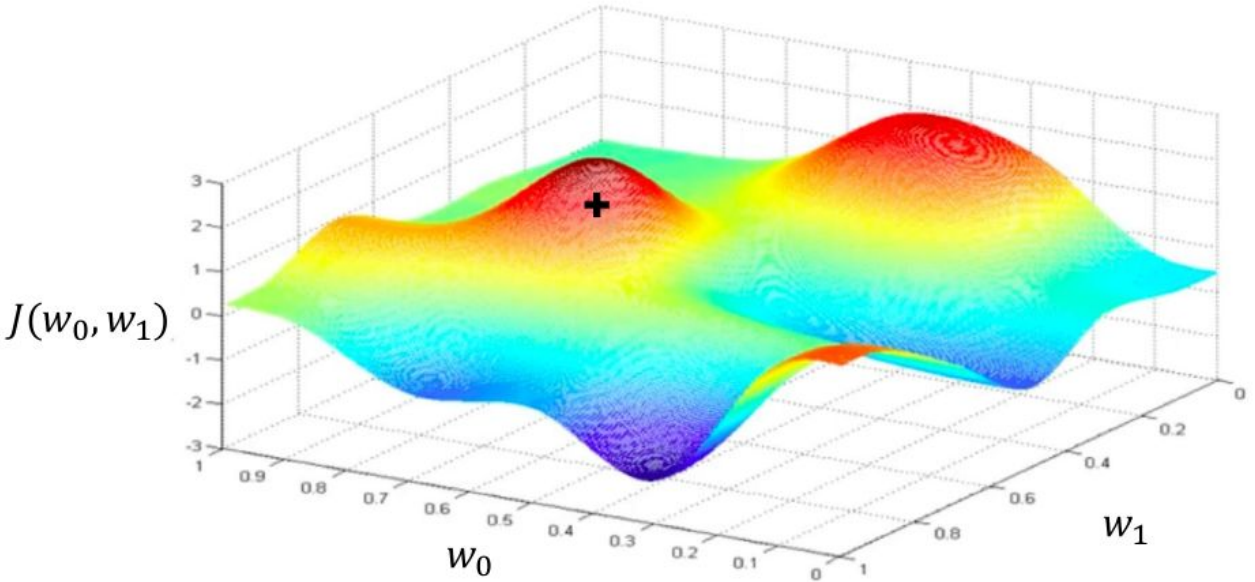
We want to find the network weights that ***achieve the lowest loss***

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\text{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

$$\boldsymbol{W}^* = \underset{\boldsymbol{W}}{\text{argmin}}\, J(\boldsymbol{W})$$

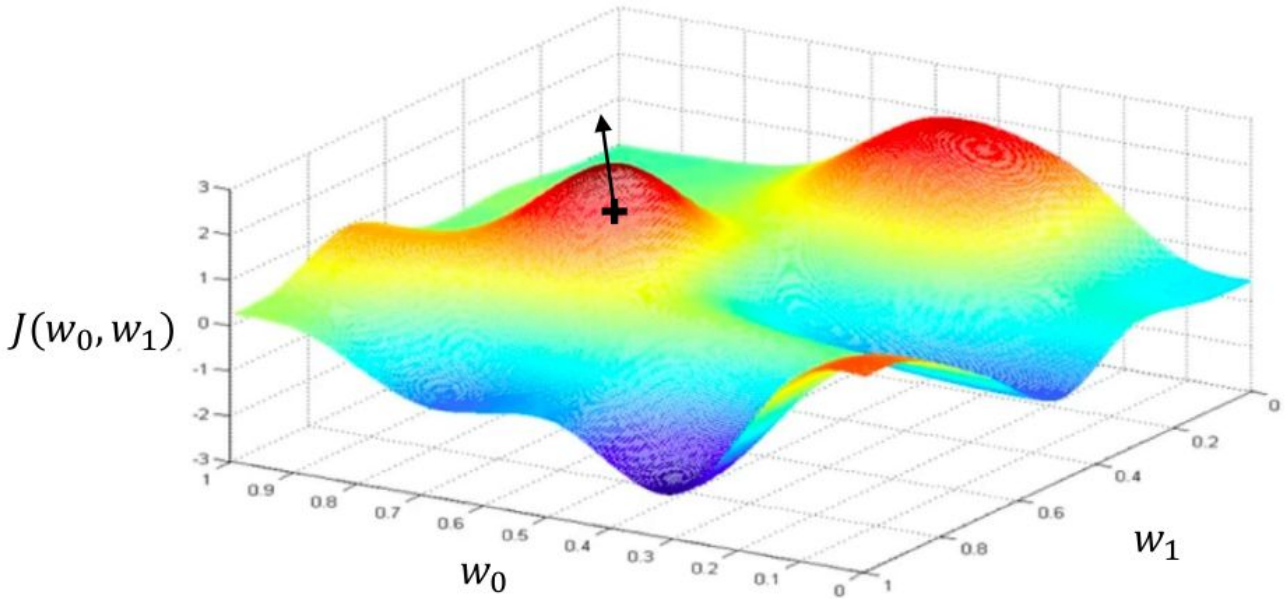
Loss Optimization

Randomly pick an initial (w_0, w_1)



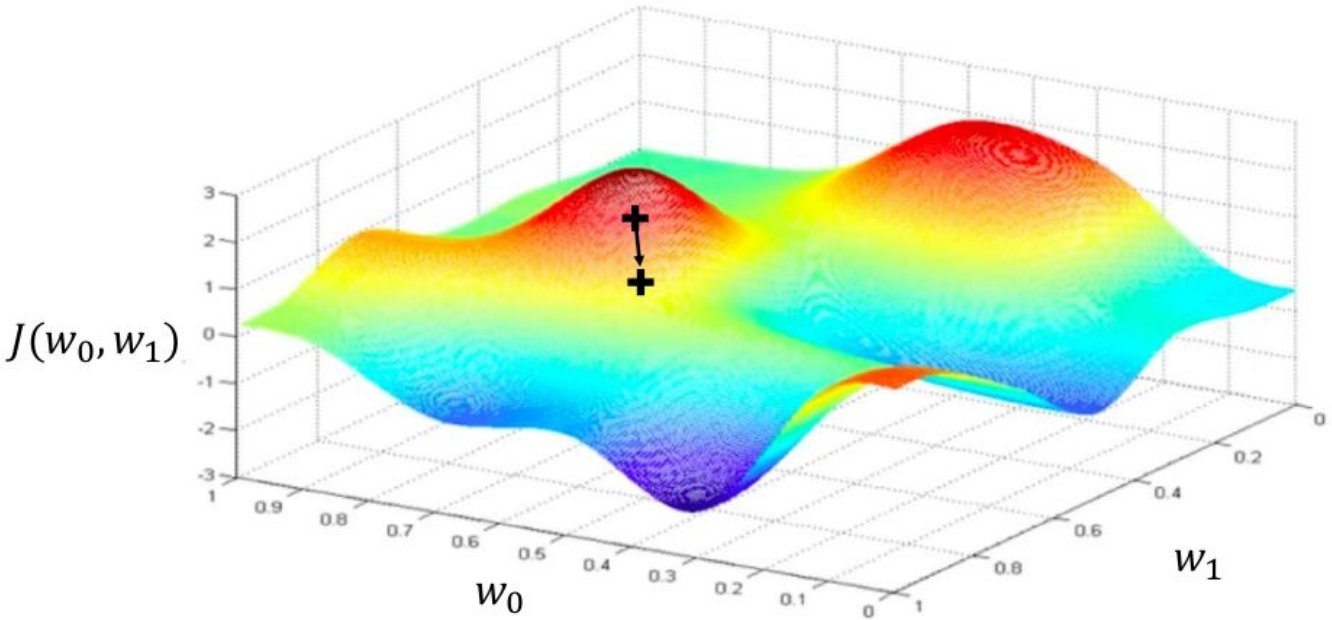
Loss Optimization

Compute gradient, $\frac{\partial J(W)}{\partial W}$



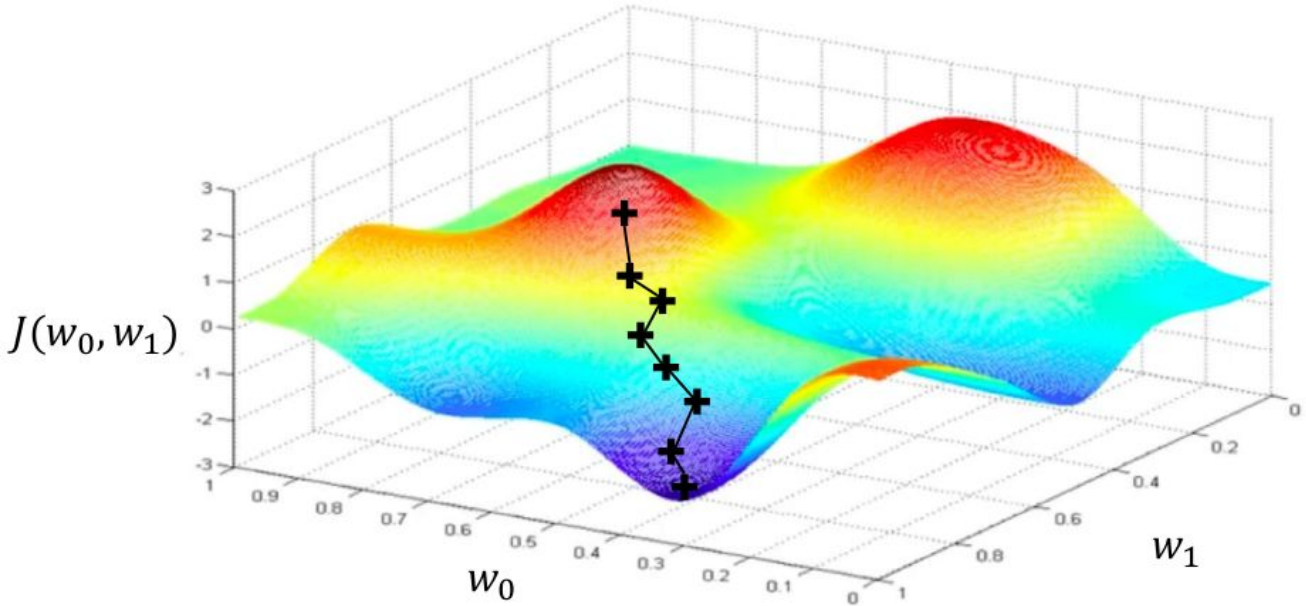
Loss Optimization

Take small step in opposite direction of gradient



Gradient Descent

Repeat until convergence

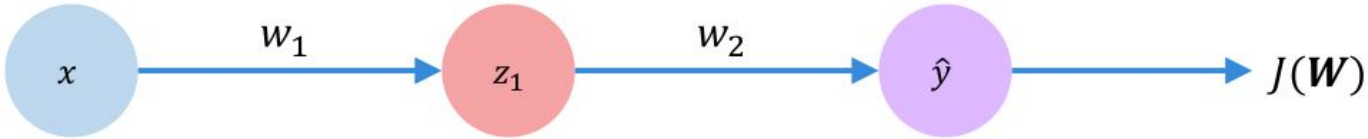


Gradient Descent

Algorithm

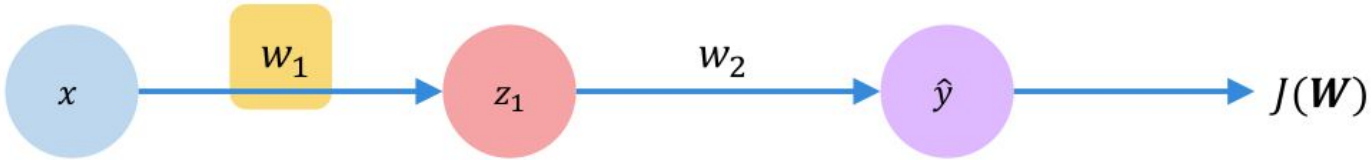
1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Return weights

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_2} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_2}$$

Computing Gradients: Backpropagation



$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

Apply chain rule!

Apply chain rule!

Computing Gradients: Backpropagation

$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

Repeat this for ***every weight in the network*** *using gradients from later layers*

Loss Functions Can Be Difficult to Optimize

Remember:

Optimization through gradient descent

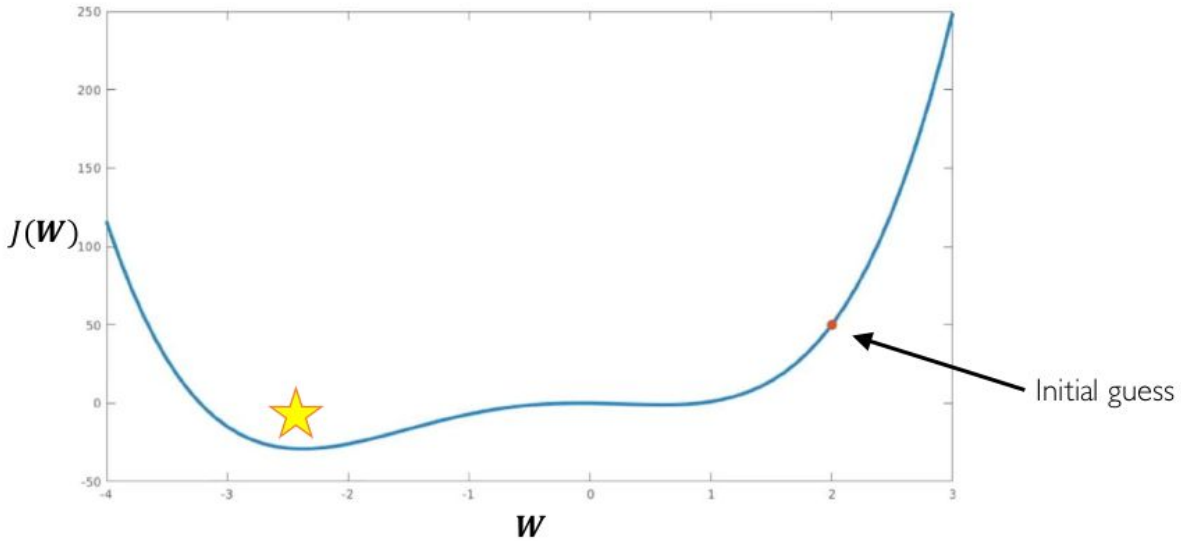
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

How can we set the learning rate?

Setting the Learning Rate

Large learning rates *overshoot, become unstable and diverge*

Small learning rate *converges slowly and gets stuck in false local minima*



Stable learning rates *converge smoothly and avoid local minima*

How to deal with this?

Idea 1:

Try lots of different learning rates and see what works "just right"

Idea 2:

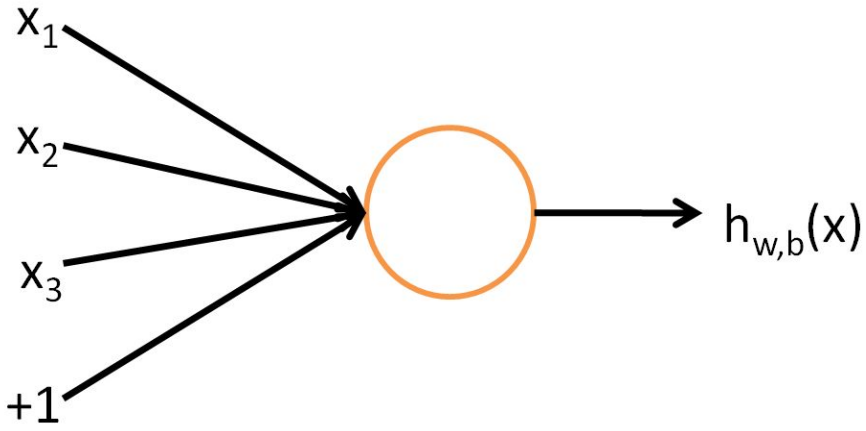
Do something smarter!
Design an adaptive learning rate that "adapts" to the landscape

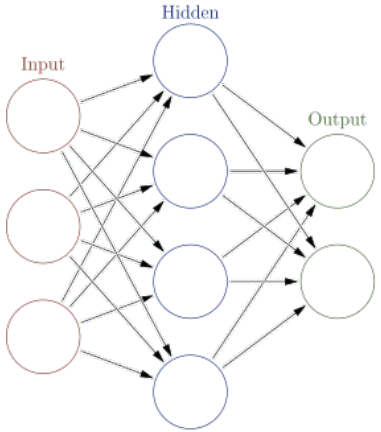
Adaptive Learning Rates

- Learning rates are 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...

That’s it for today. But just a simple intuition.

Why do one need non linear activation function?



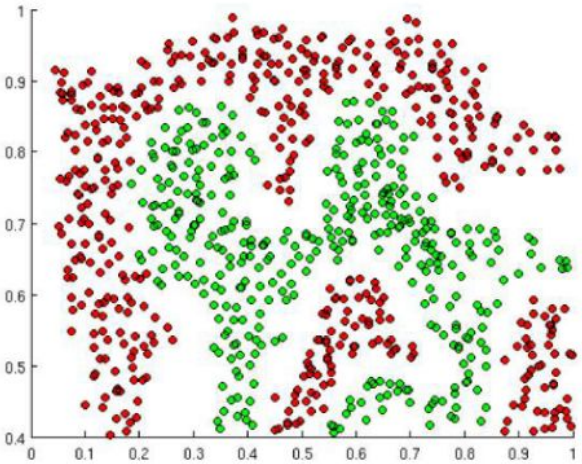


Linear hidden layer is useless.

We actually need an AF to include non-linearities.

Importance of Activation Functions

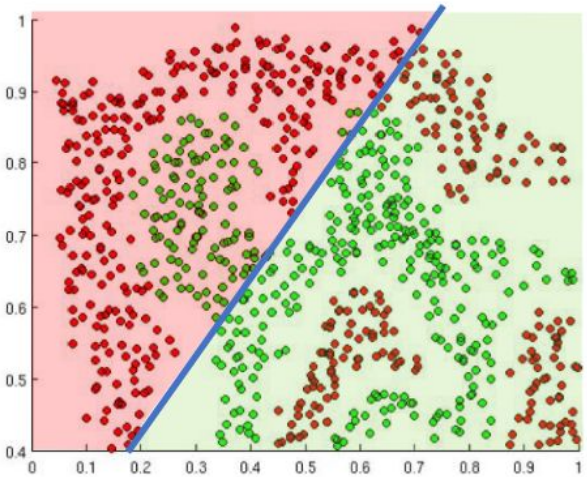
*The purpose of activation functions is to **introduce non-linearities** into the network*



What if we wanted to build a Neural Network to distinguish green vs red points?

Importance of Activation Functions

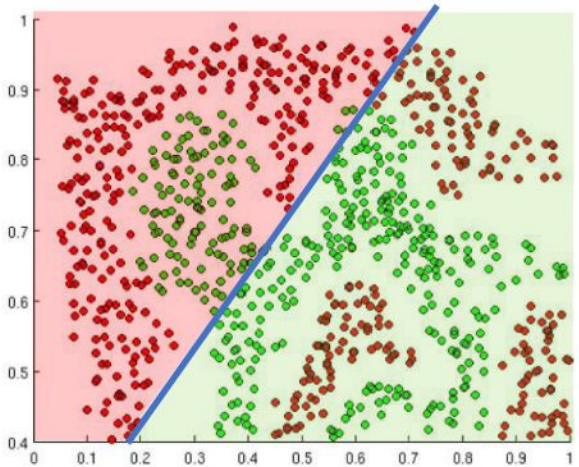
The purpose of activation functions is to ***introduce non-linearities*** *into the network*



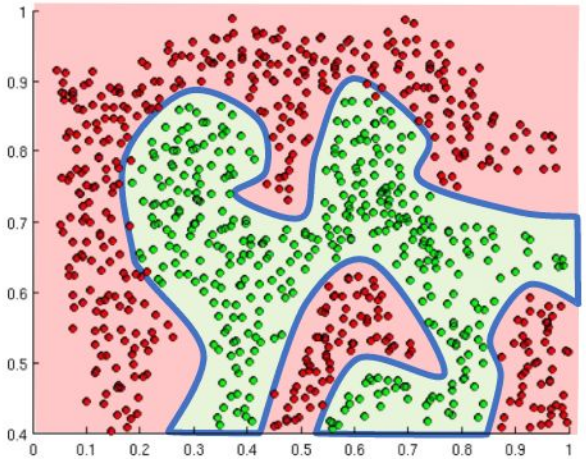
Linear Activation functions produce linear decisions no matter the network size

Importance of Activation Functions

*The purpose of activation functions is to **introduce non-linearities** into the network*



Linear Activation functions produce linear decisions no matter the network size



Non-linearities allow us to approxi arbitrarily complex functions

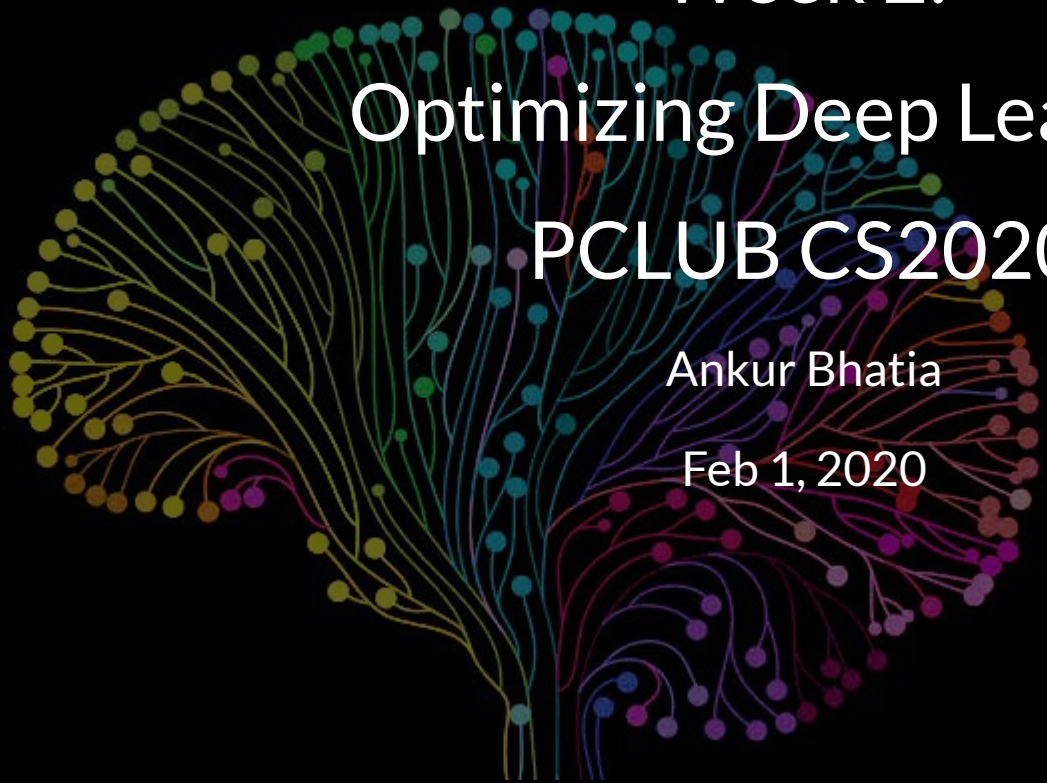
Week 2:

Optimizing Deep Learning

PCLUB CS2020

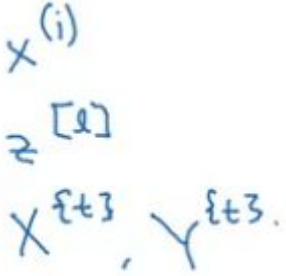
Ankur Bhatia

Feb 1, 2020



Optimizing NNs.

Batch (Vanilla) vs Mini Batch Gradient Descent



1. ith training example.
2. L - layer
3. mini-batch

Mini Batch Gradient Descent.

repeat {

for $t = 1, \ldots, 5000$ { $\frac{m}{1000}$ } ($m = 5000000$ (say)) (say)

{

forward prop on $X^{\{t\}}$

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

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

$\vdots$

$A^{[l]} = g^{[l]}(Z^{[l]})$.

} vectorized implementation (1000 ex)

Compute cost $J^{\{t\}} = \frac{1}{1000} \sum_{n=1}^{L} \mathcal{L}(\hat{y}^{(i)}, y^{(i)}) + \frac{\lambda}{2 \cdot 1000} \sum_{k} \|w^{[l]}\|_F^2$

Back prop to compute gradients w.r.t $J^{\{t\}}$ (using $(X^{\{t\}}, Y^{\{t\}})$)

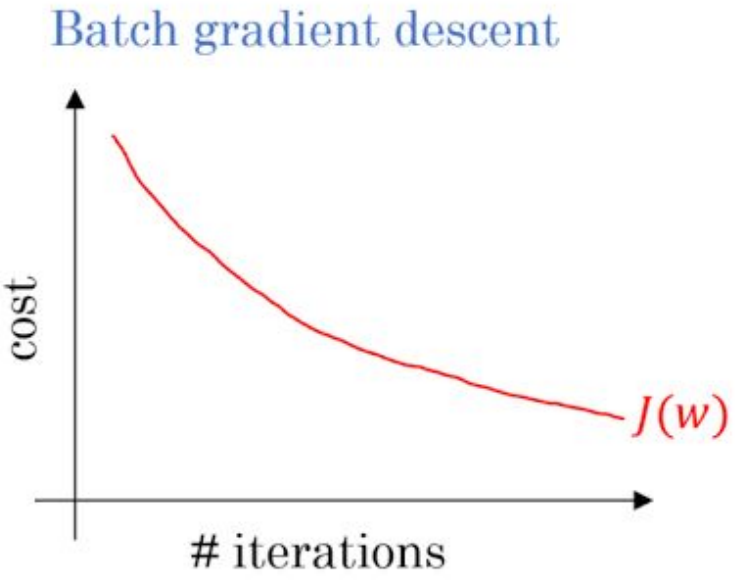
$w^{[l]} = w^{[l]} - \alpha dw^{[l]}$, $b^{[l]} = b^{[l]} - \alpha db^{[l]}$

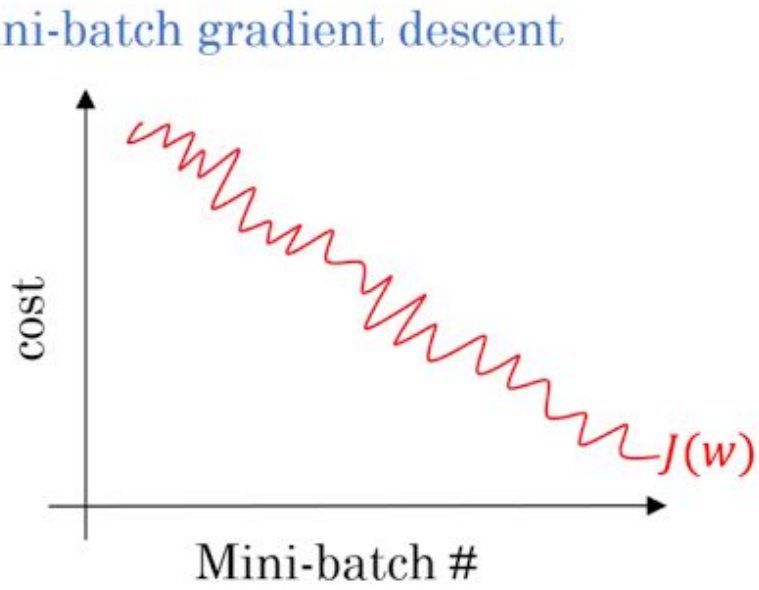
}

1 epoch (after full for loop completed).

}

Instead, we're estimating it on a small batch. Which means we're not always going in the optimal direction, because our derivatives are 'noisy'.

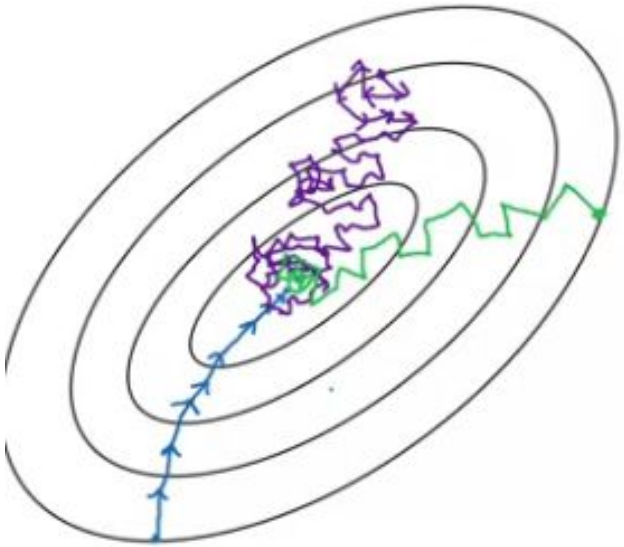




One Mini batch may be easy and other may be difficult to learn.
Parameter to choose - Size of minibatch

Mini Batch Gradient Descents

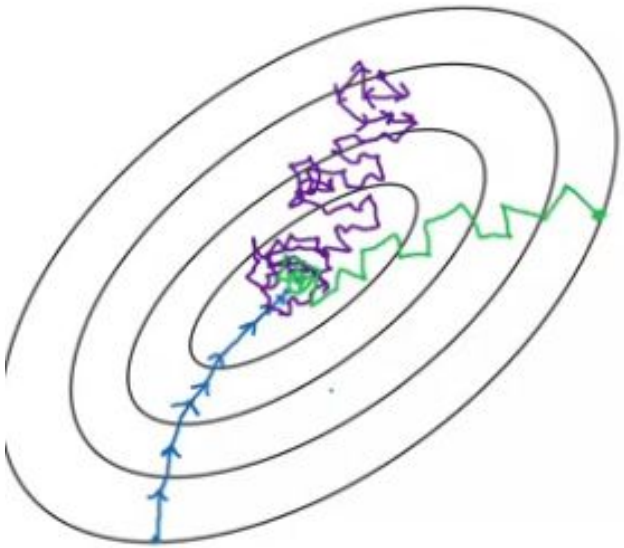
If m = 1; Stochastic Gradient Descent => Easy to compute but very noisy.
If m = m; Batch Gradient Descent



1. So What's the size to choose?.(bet 1 - m)
2. Stochastic GD loses speed of Vectorization.
3. In between mini-batch size gives:
   a. Fast learning
   b. Get good vectorization.
   c. Makes progress without all data to process.
4. Batch-GD takes too long per iteration.

Mini Batch Gradient Descents

If m = 1; Stochastic Gradient Descent
If m = m; Batch Gradient Descent

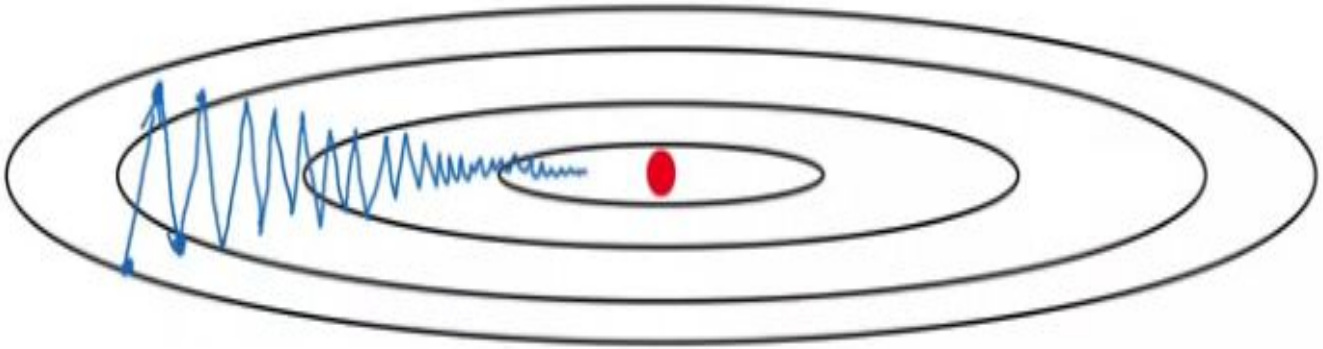


1. For small Trainx - m = m;
2. General mini batch sizes = 64, 128, 256…
3. mini - batch should fit in CPU - GPU m/m.

Optimization Algorithms faster than GD.

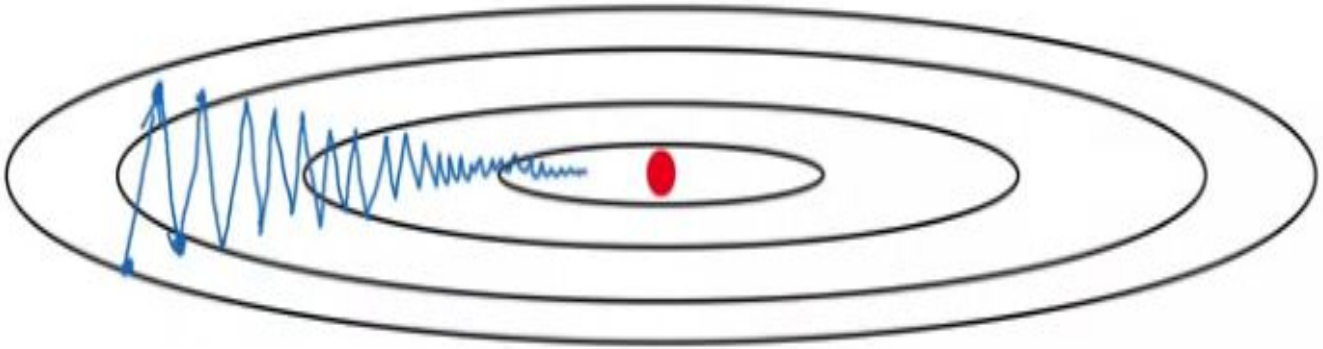
1. Gradient Descent with momentum

1. GD (either batch-minibatch) takes a lot of steps to reach minima - oscillates (these osc. Slows GD).
2. That is why can't use larger LR. (overshoot, diverge) (So LR large - X)



1. GD (either batch-minibatch) takes a lot of steps to reach minima - oscillates (these osc. Slows GD).
2. That is why can't use larger LR. (overshoot, diverge) (So LR large - X)

90 degree steps
to ellipse.



1. Slower learning in vertical.
2. Faster learning in horizontal.

Optimization Algorithms faster than GD.

1. Gradient Descent with momentum

1. GD (either batch-minibatch) takes a lot of steps to reach minima - oscillates (these osc. Slows GD).
2. That is why can't use larger LR. (overshoot, diverge) (So LR large - X)
3. What we do then?
4. Is running average a good option? Yes maybe!
5. A hyper-parameter beta (for the sequence of points) will be controlling the extent of rememberance average.
6. Eg - beta = 0.9 implies averaging over the last 10 gradients to get the next gradient. (it averages over 1/(1-beta)

1. Gradient Descent with momentum : Exponentially weighted averages

Exponentially weighted averages:

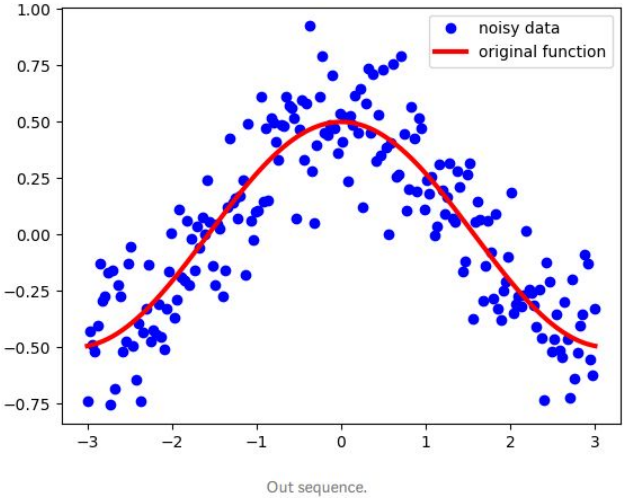
Idea is to use exponentially weighted averages across the horizontal and vertical at each time step to average out and make the next steps smooth.

1. Gradient Descent with momentum : Exponentially weighted averages

$$V_t = \beta V_{t-1} + (1-\beta) S_t$$
$$\beta \in [0, 1]$$

Exponentially weighted averages:

$$v_0 = 0$$
$$v_1 = \beta v_0 + (1-\beta)\,\theta_1$$
$$v_2 = \beta v_1 + (1-\beta)\,\theta_2$$
$$v_3 = \beta v_2 + (1-\beta)\,\theta_3$$
$$\dots$$

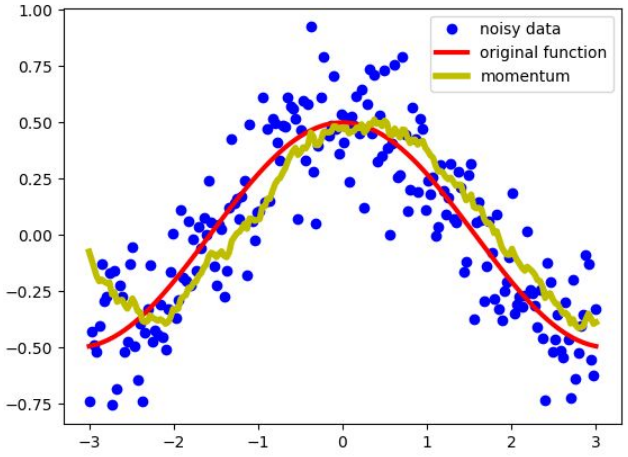


1. Gradient Descent with momentum : Exponentially weighted averages

$$V_t = \beta V_{t-1} + (1-\beta) S_t$$
$$\beta \in [0, 1]$$

Exponentially weighted averages:

$$v_0 = 0$$
$$v_1 = \beta v_0 + (1-\beta)\,\theta_1$$
$$v_2 = \beta v_1 + (1-\beta)\,\theta_2$$
$$v_3 = \beta v_2 + (1-\beta)\,\theta_3$$
$$\dots$$

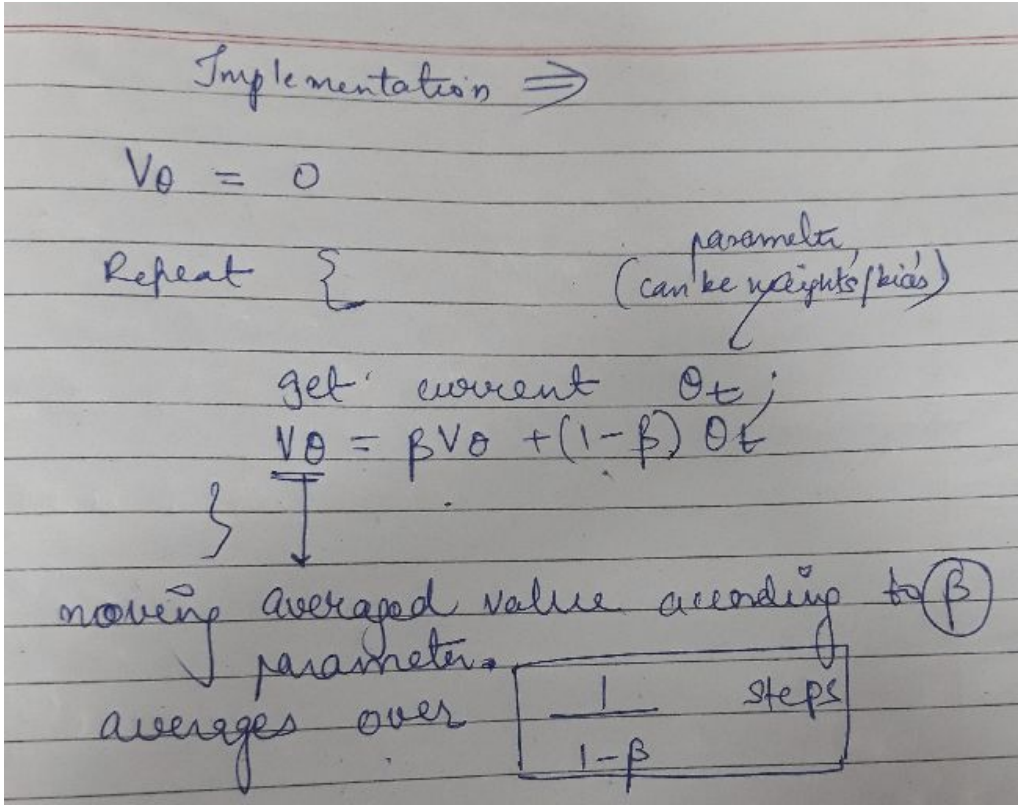


momentum — data from exponentially weighed averages.

https://towardsdatascience.com/stochastic-gradient-descent-with-momentum-a84097641a5d

<p>club

1. Gradient Descent with momentum : Exponentially weighted averages



Gradient Descent with momentum

Initializations:
1. Vdw = 0
2. Vdw = 0

Dimensions are same as dW and db

Beta = 0.9 is a common choice.

On iteration t:

Compute dW, db on the current mini-batch

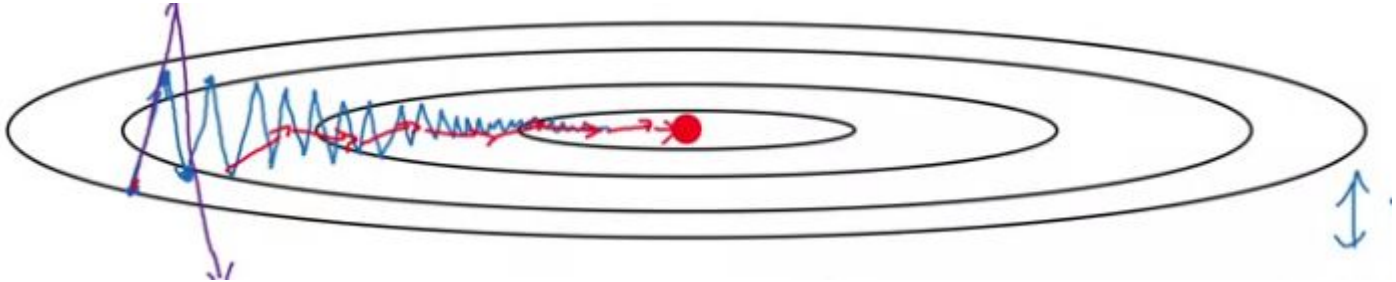
$$v_{dW} = \beta v_{dW} + (1 - \beta) dW$$

$$v_{db} = \beta v_{db} + (1 - \beta) db$$

$$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$$

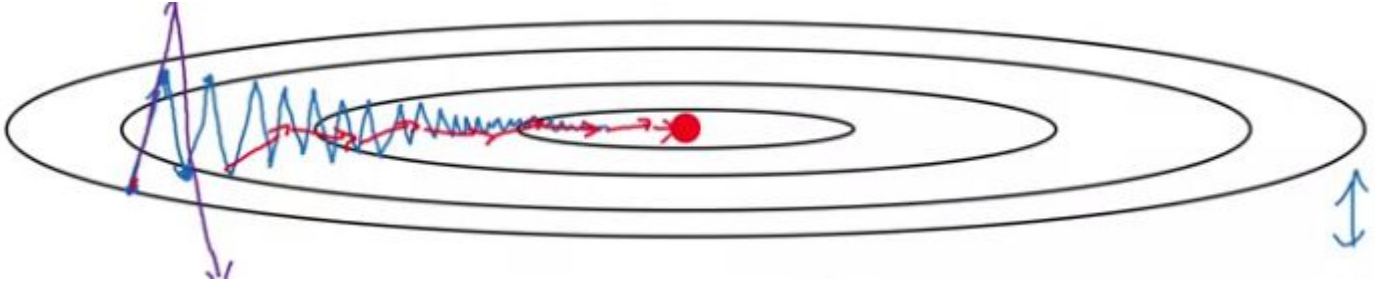
Hyperparameters: α, β $\beta = 0.9$

Gradient Descent with momentum



After Applying Momentum:
1.Moves more quickly to the horizontal direction.
2.**Oscillations are damped. (on using momentum)**
3.Takes more straight forward path.

1. Vertical average will almost be 0.
2. Horizontal average, all derivatives in horizontal direction, will be still big.



Now, Consider the intuition by: ball rolling

On iteration t:

Compute dW, db on the current mini-batch

$$v_{dW} = \beta v_{dW} + (1-\beta)dW$$

$$v_{db} = \beta v_{db} + (1-\beta)db$$

$$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$$

Hyperparameters: α, β $\beta = 0.9$

friction

velocity

momentum

1. beta = 0.9 implies averaging over the last 10 gradients to get the next gradient. (it averages over 1/(1-beta)

Remember: Dimensions of Vdb, Vdw?

Now, Consider the intuition by: ball rolling

On iteration t:

Compute dW, db on the current mini-batch

$$v_{dW} = \beta v_{dW} + (1-\beta) dW$$

$$v_{db} = \beta v_{db} + (1-\beta) db$$

$$W = W - \alpha v_{dW}, \quad b = b - \alpha v_{db}$$

Hyperparameters: α, β $\beta = 0.9$

friction

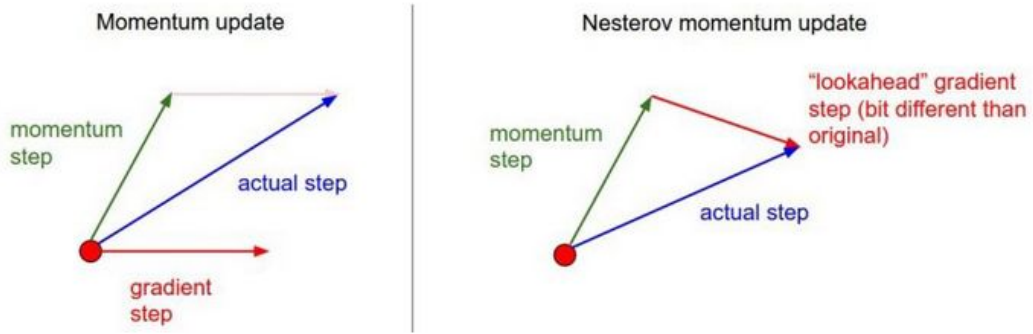
velocity

momentum

1. beta = 0.9 implies averaging over the last 10 gradients to get the next gradient. (it averages over 1/(1-beta)

Nesterov accelerated gradient (NAG)

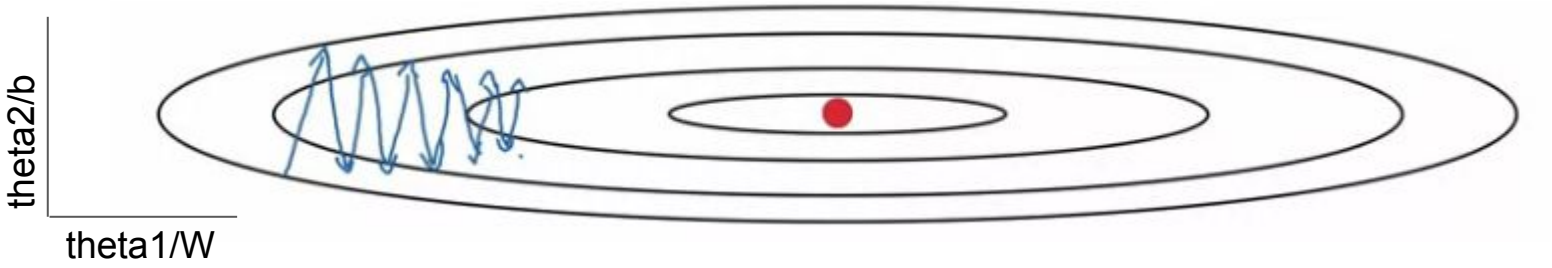
Nesterov Momentum is a slightly different version of the momentum update that has recently been gaining popularity. In this version we're first looking at a point where current momentum is pointing to and computing gradients from that point. It becomes much clearer when you look at the picture.



Source (Stanford CS231n class)

2. Understanding Gradient Descent with RMS Prop in for 2D.

Root Mean Square Propagation



2. Gradient Descent with Root Mean Square Propagation:

Or mentioned Sdw in place of Vdw

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

Exponential weighted average of squares.

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

2. Gradient Descent with Root Mean Square Propagation:

Or mentioned Sdw in place of Vdw

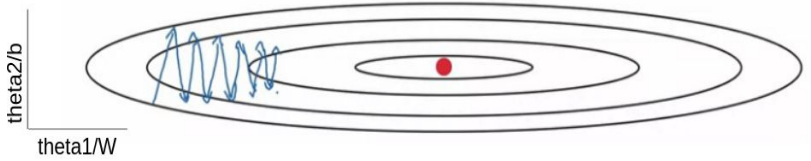
$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$ ← Elementwise

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

2. Gradient Descent with Root Mean Square Propagation:



Or mentioned Sdw in place of Vdw

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

Sdw to be small

$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

Sdb to be large

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$

2. Gradient Descent with Root Mean Square Propagation:

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

Vdw to be small

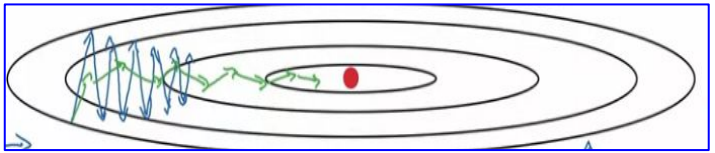
$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

Vdb to be large

Or mentioned Sdw in place of Vdw

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$



- Slopes in b direction is very large compared to w direction.=> large db and small dw => Vdb large and Vdw small
- Also can use larger value of learning rate.

2. Gradient Descent with Root Mean Square Propagation:

$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$

Vdw to be small

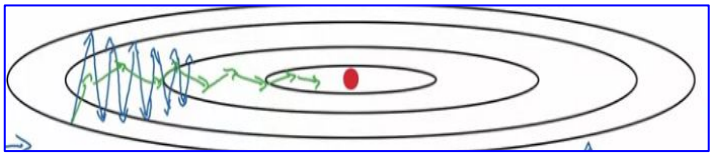
$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

Vdb to be large

Or mentioned Sdw in place of Vdw

$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

$$b = b - \alpha \cdot \frac{db}{\sqrt{v_{db}} + \epsilon}$$



- **Why called RMS?**
- **Why is epsilon used? -** In adam paper - epsilon mentioned 10^-8 so that algo doesn't divide by .

2. Gradient Descent with Root Mean Square Propagation:

Dimensions will be large.

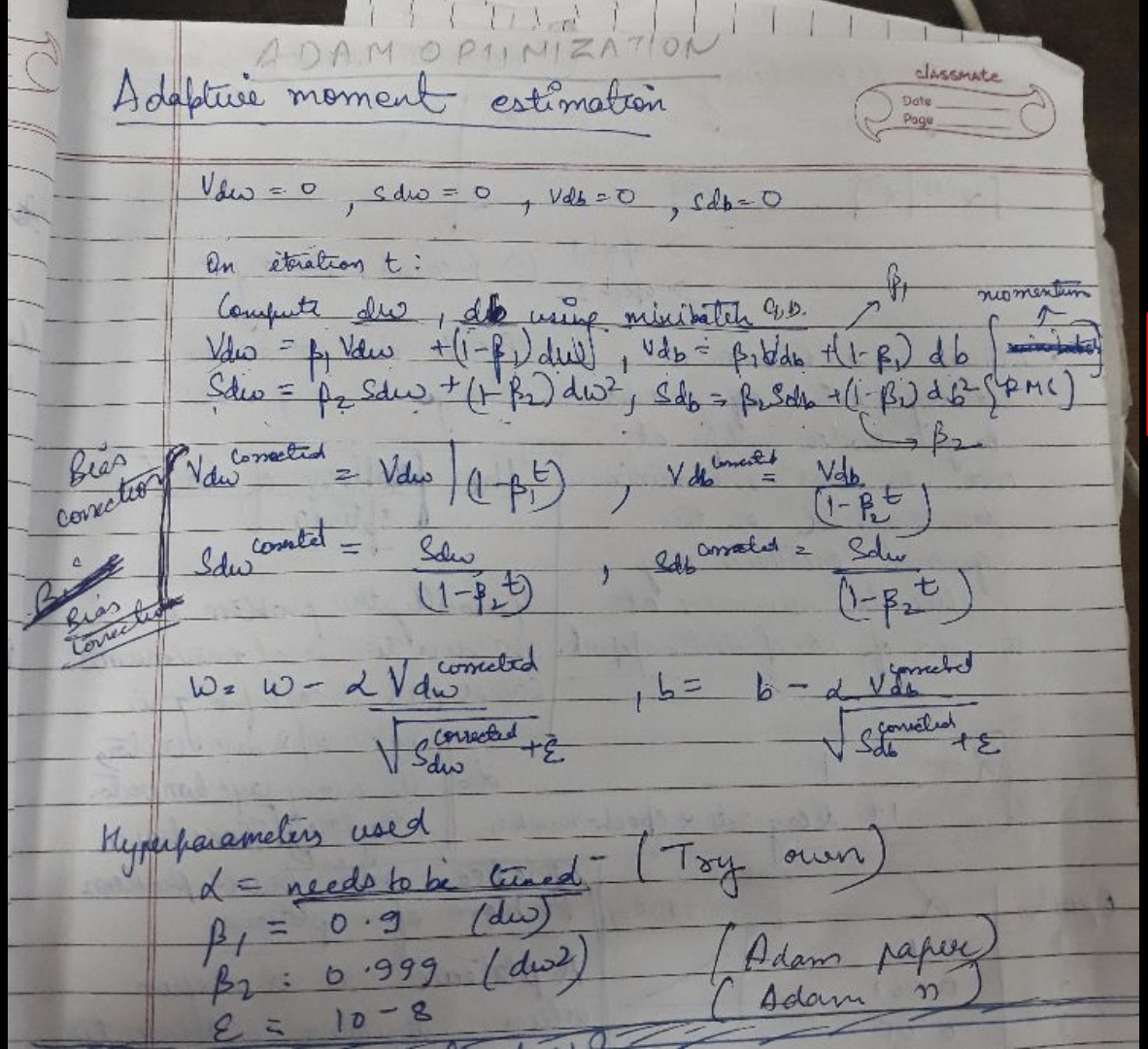
W1, W2, W3, ……., b1, b2, b3, …….

Optimization Algorithms faster than GD.

3. Momentum + RMSProp = Adam

Adaptive Moment Estimation

It is seen that this algorithm generalized over lots of deep learning architectures.
Worked well on many problems.



Beta1 - 1st moment
Beta2 - 2nd moment

Exponential weighted average of squares.

Adaptive Moment Estimation

ADAM: A METHOD FOR STOCHASTIC OPTIMIZATION - ICLR 2015

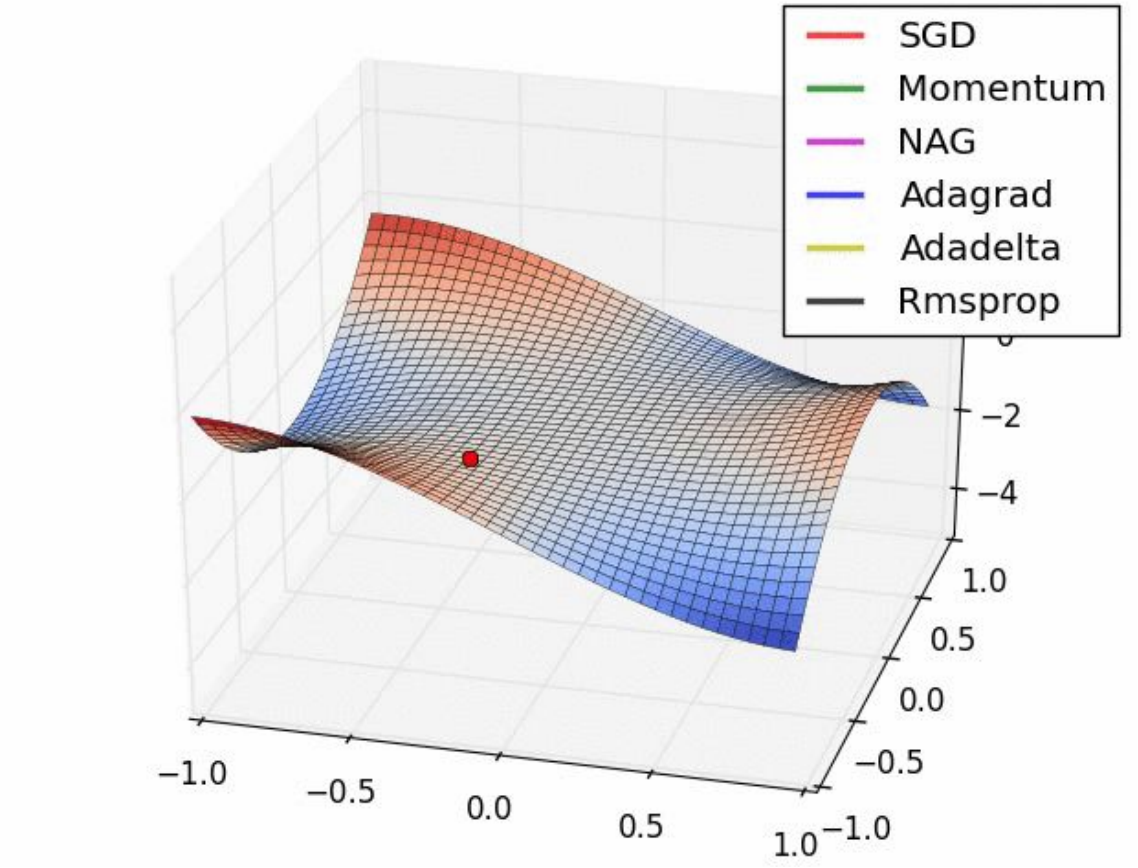
Diederik P. Kingma* , Jimmy Lei Ba* https://arxiv.org/pdf/1412.6980.pdf

Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

Require: α: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters θ
Require: θ_0: Initial parameter vector
$m_0 \leftarrow 0$ (Initialize 1st moment vector)
$v_0 \leftarrow 0$ (Initialize 2nd moment vector)
$t \leftarrow 0$ (Initialize timestep)
while θ_t not converged **do**
$t \leftarrow t + 1$
$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)
$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
$\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
$\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
end while
return θ_t (Resulting parameters)

Hyperparameters

Various Descent's:



Nesterov accelerated gradient