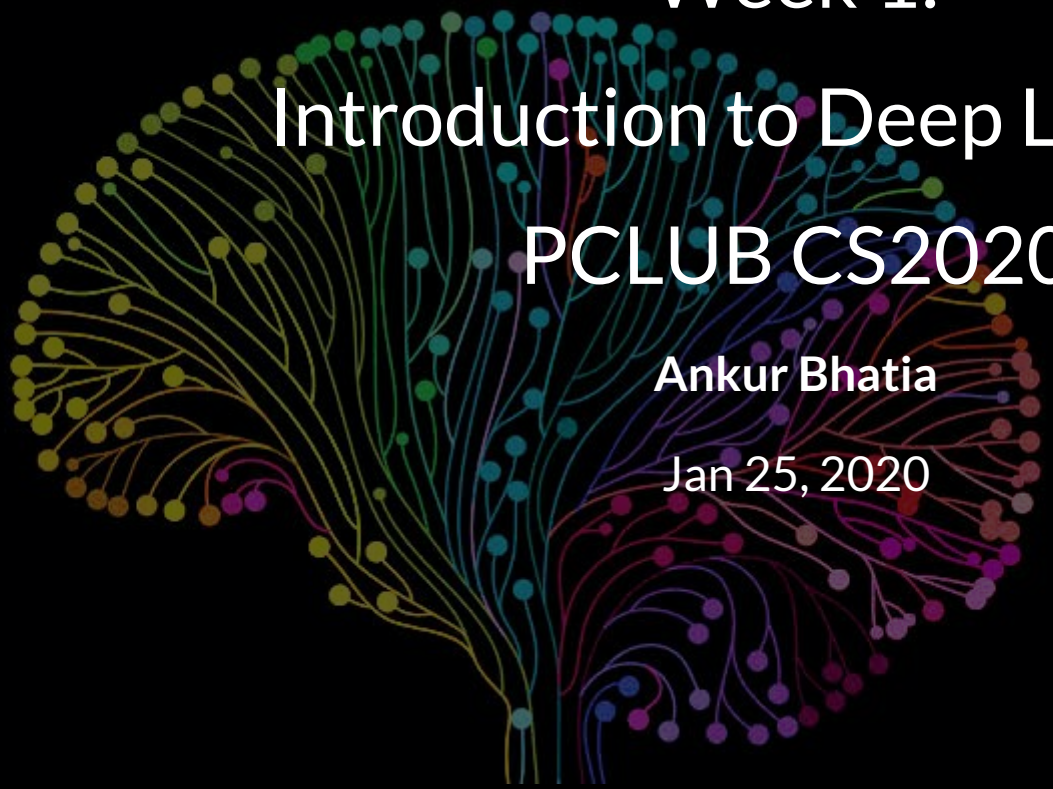
Week 1:
Introduction to Deep Learning
PCLUB CS2020

Ankur Bhatia

Jan 25, 2020



Course Prerequisites - Linear Algebra
Calculus

The Rise of Deep Learning

6.S191 Introduction to Deep Learning

introtodeeplearning.com

<p> club

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

6.S191 Introduction to Deep Learning
introtodeeplearning.com

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

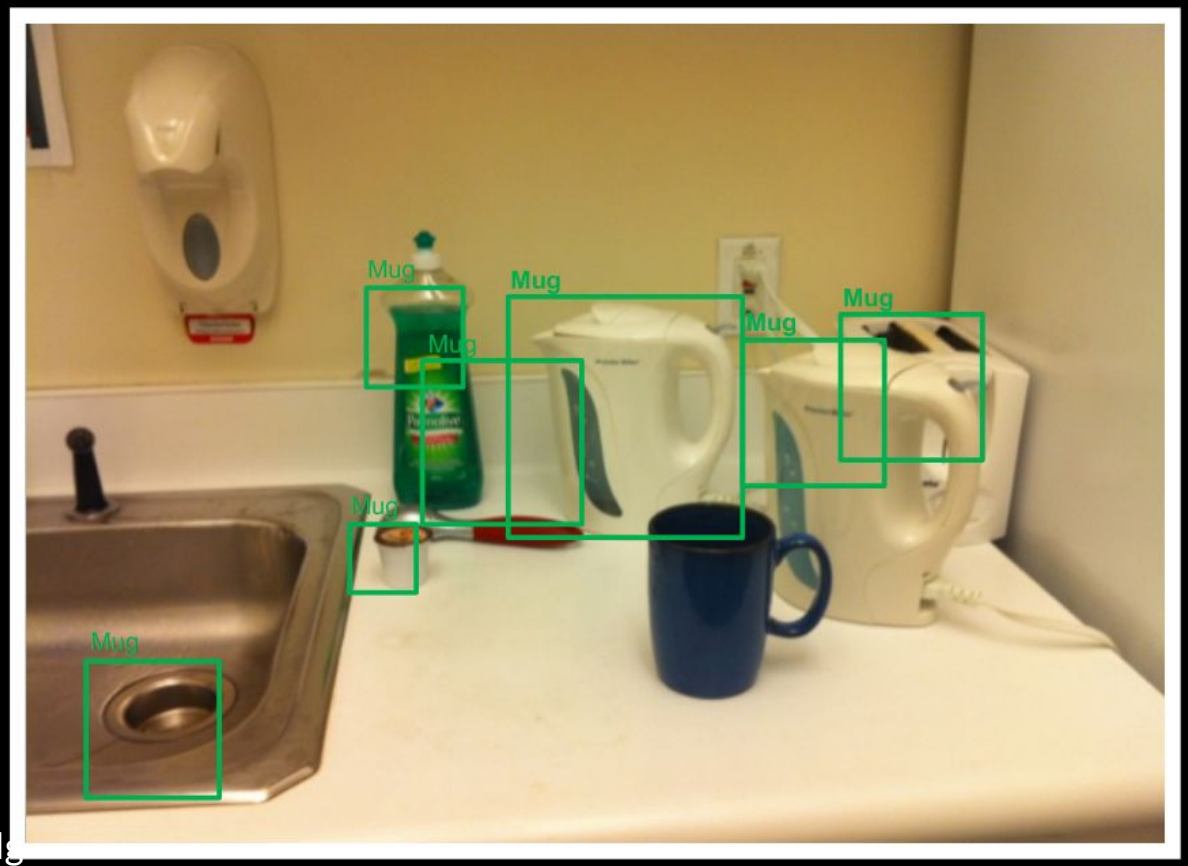
→ Web search

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

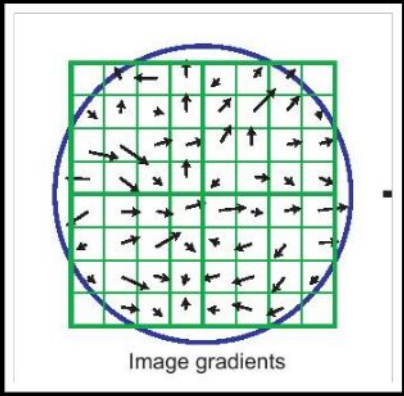
Computer vision

Feature representation

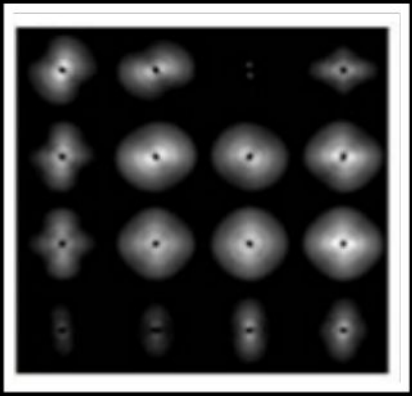
Learning algorithm

Andrew Ng

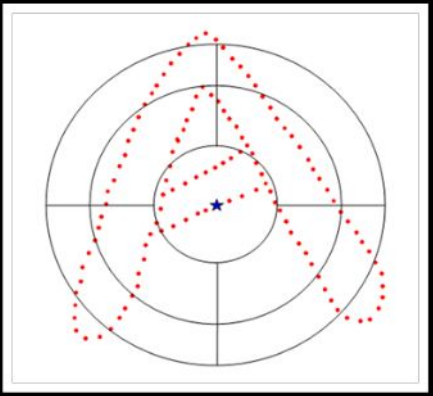
Features for vision



SIFT



GIST



Shape context

Andrew Ng

Features for machine learning

Images

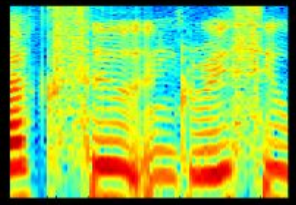
Image

Vision features

Detection

Audio

Audio

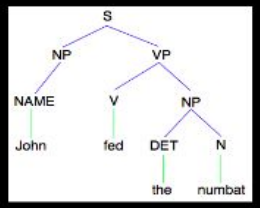


Audio features

Speaker ID

Text

Text



Text features

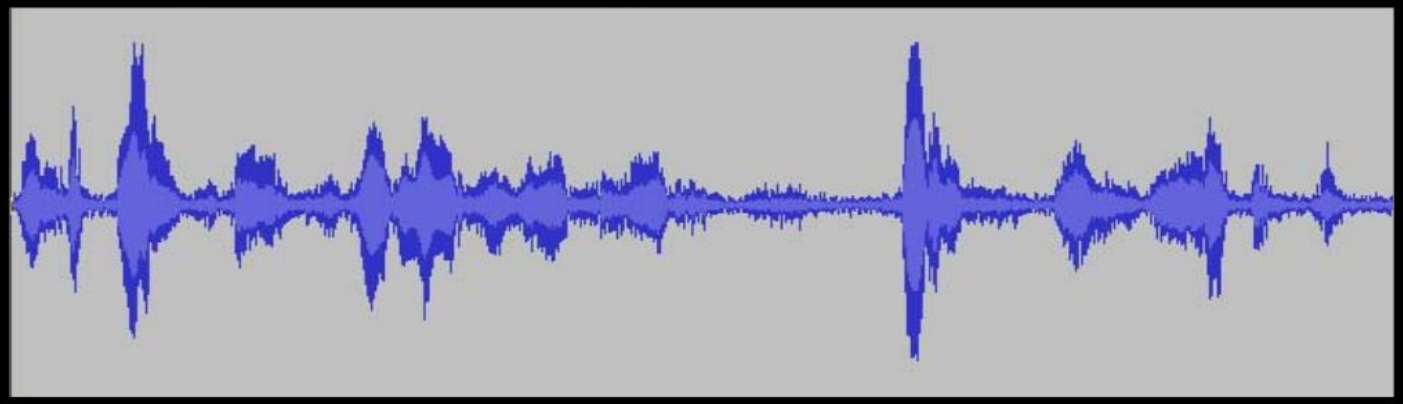
Web search

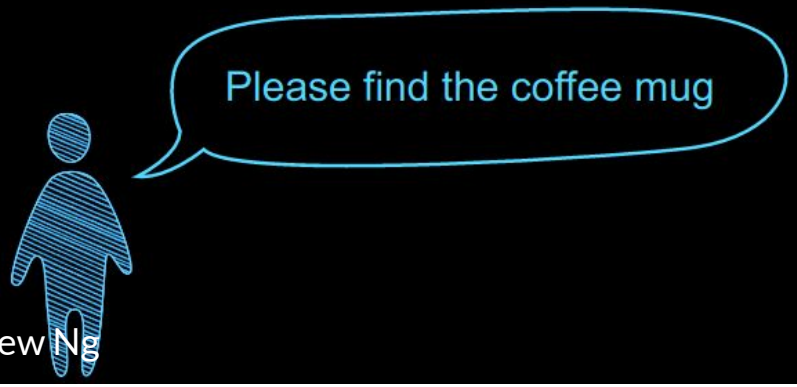
...

Andrew Ng

Why is speech recognition hard?

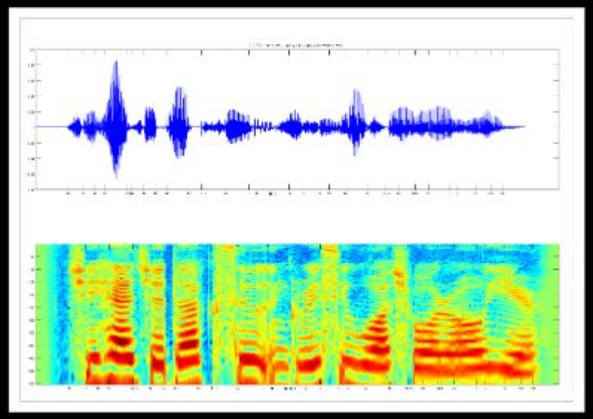
Microphone recording:



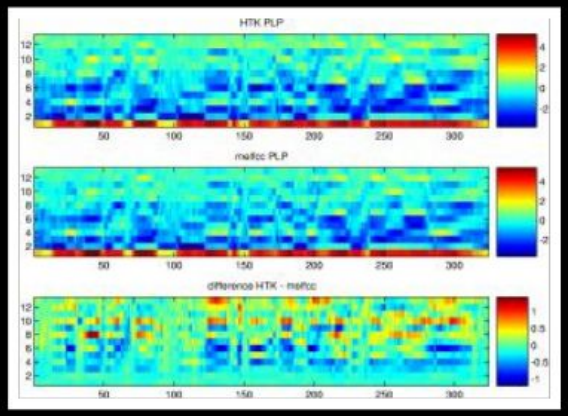


Andrew Ng

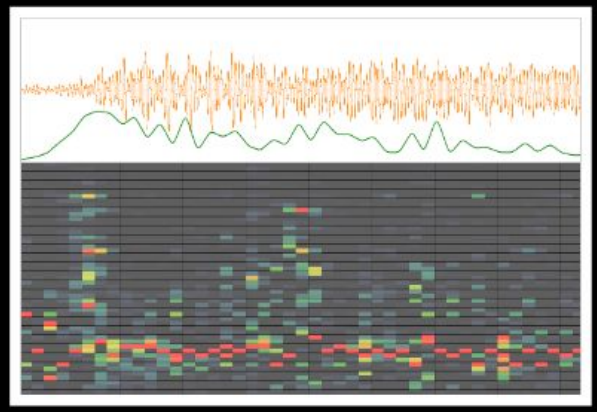
Features for audio



Spectrogram



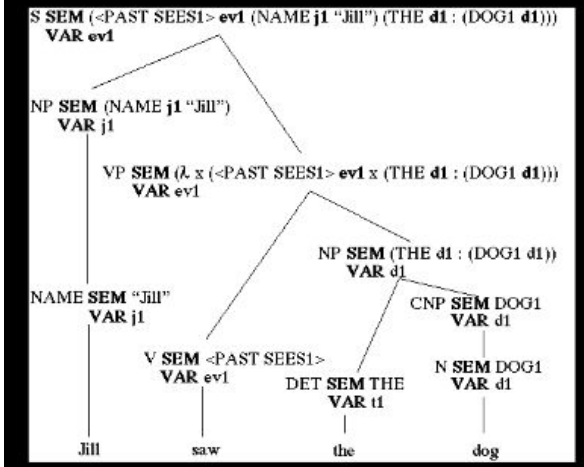
MFCC



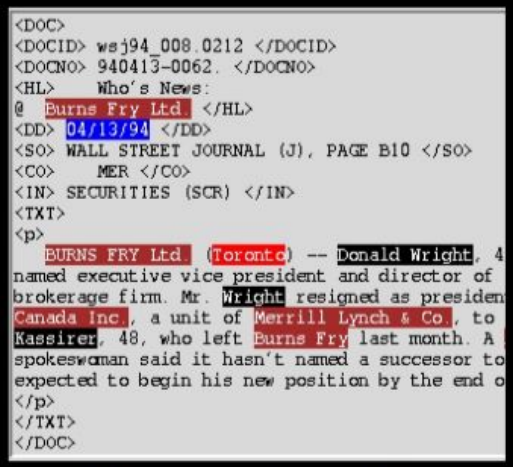
Flux

Andrew Ng

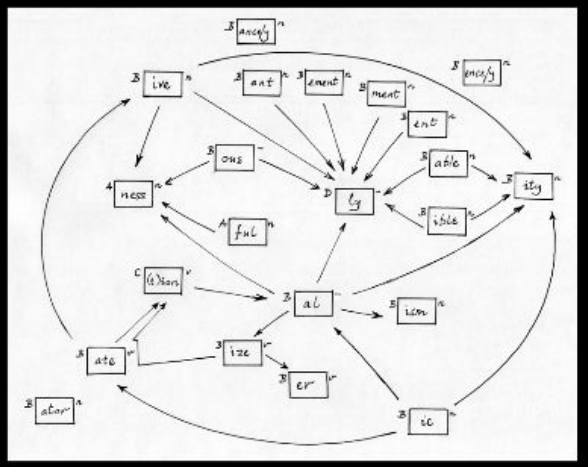
Features for text



Parser



Named entity



Stemming

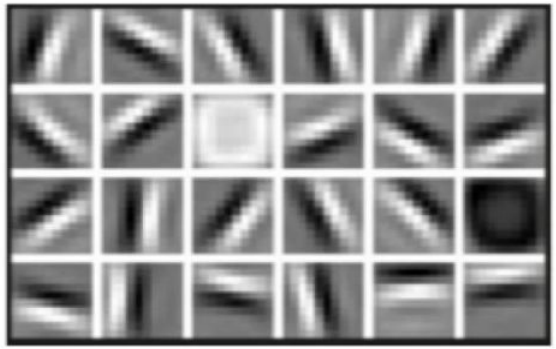
Andrew Ng

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

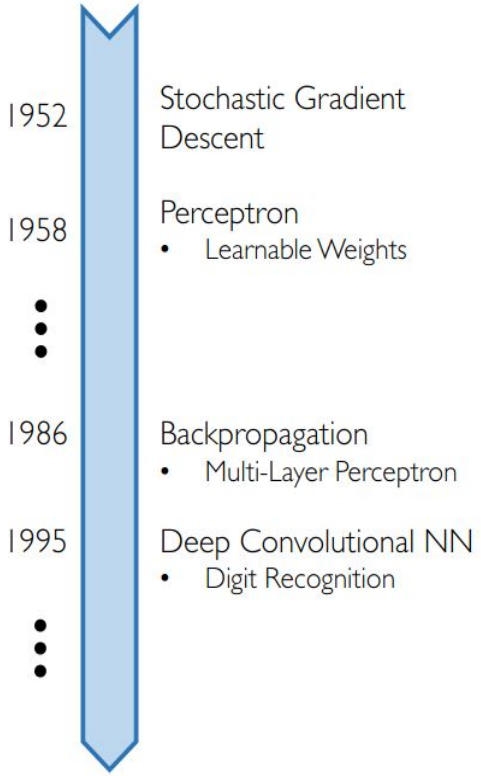
Eyes & Nose & Ears

High Level Features

Facial Structure

6.S191 Introduction to Deep Learning
introtodeeplearning.com

Why Now?



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

<p>club

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

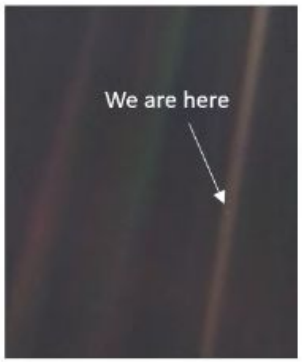
- **What is it:**
  Extract useful patterns from data.
- **How:**
  Neural network + optimization
- **How (Practical):**
  Python + TensorFlow & friends
- **Hard Part:**
  Good Questions + Good Data
- **Why now:**
  Data, hardware, community, tools, investment
- **Where do we stand?**
  Most big questions 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."

Deep Learning Growth, Celebrations, and Limitations

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

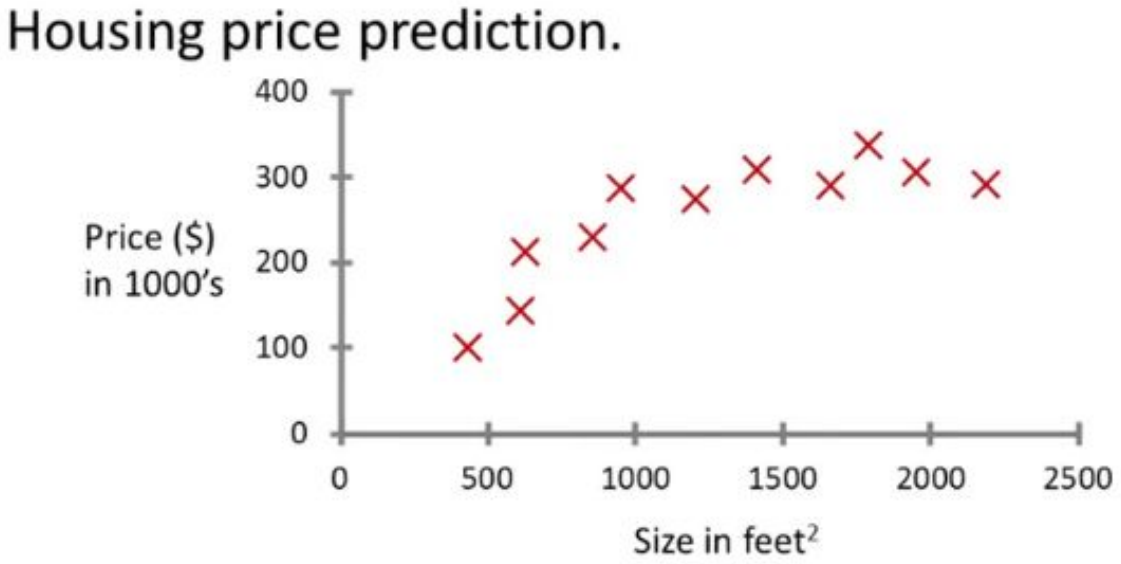
Unknown

Unknown

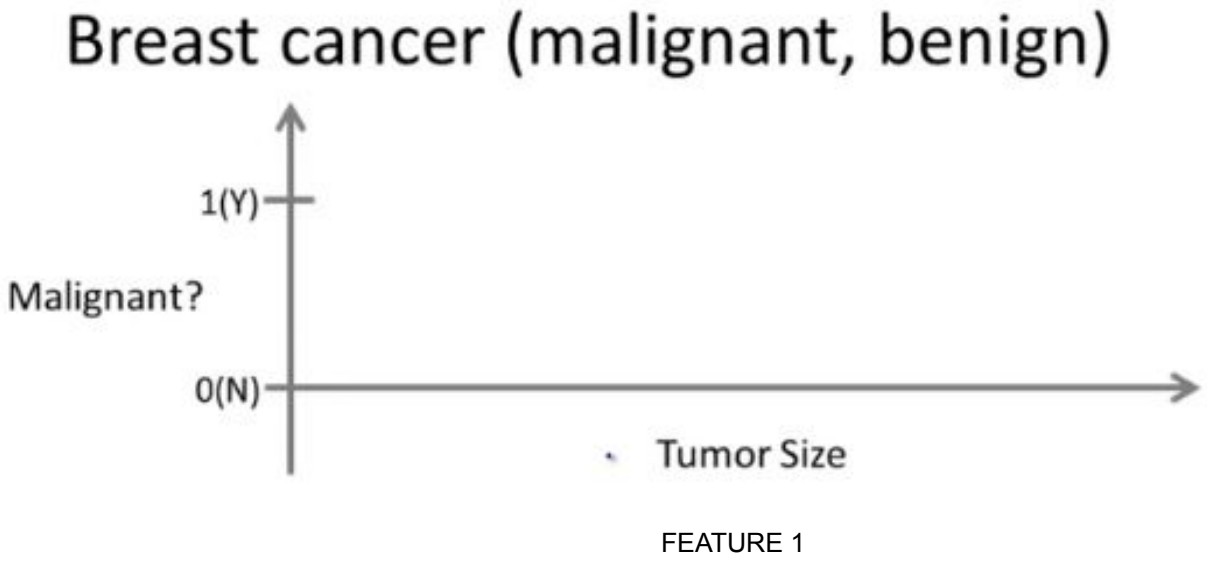
Unknown

Andrew Ng

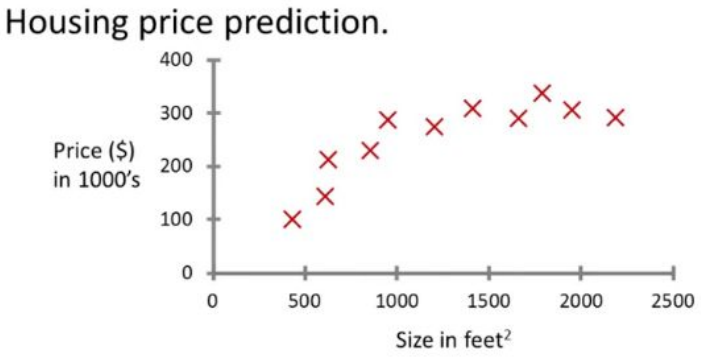
Supervised Learning
EXAMPLE 1



Supervised Learning
EXAMPLE 2



Linear Regression:



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

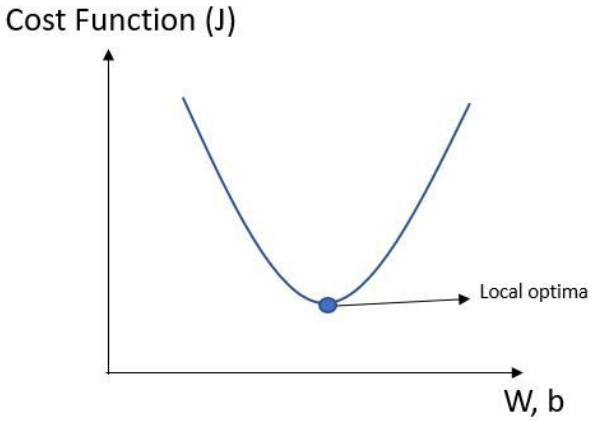
Parameters: θ_0, θ_1

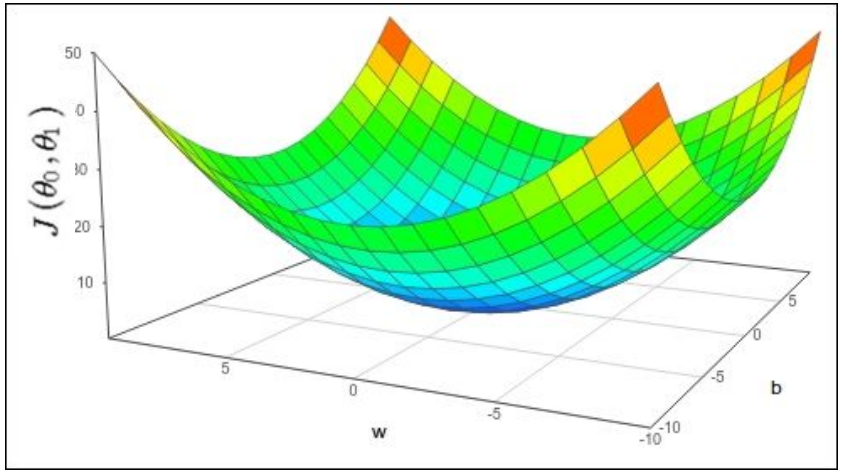
Cost Function: $J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$

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:

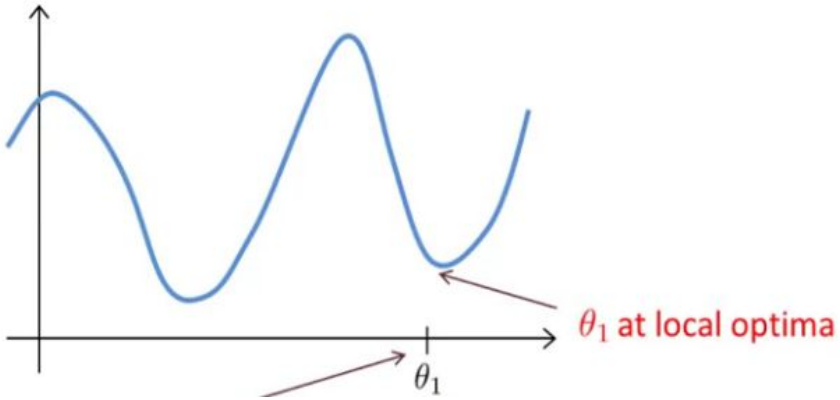


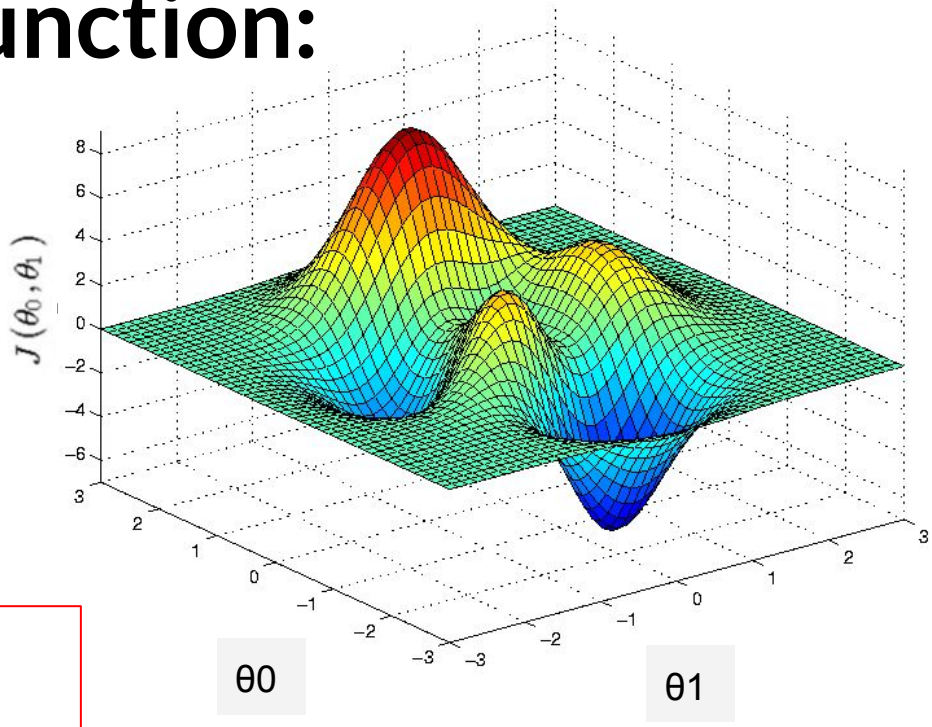


1. Notations
2. Hypothesis function and its parameters
3. Cost Function
   a. Single parameter
   b. Multiple parameters
4. Contour plots points of various h(theta0, theta1)
5. Minimize cost function

Convex Function

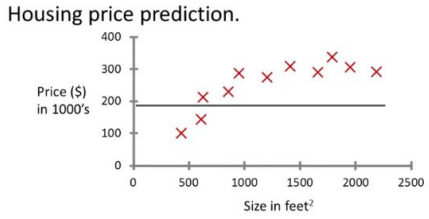
Cost Function:

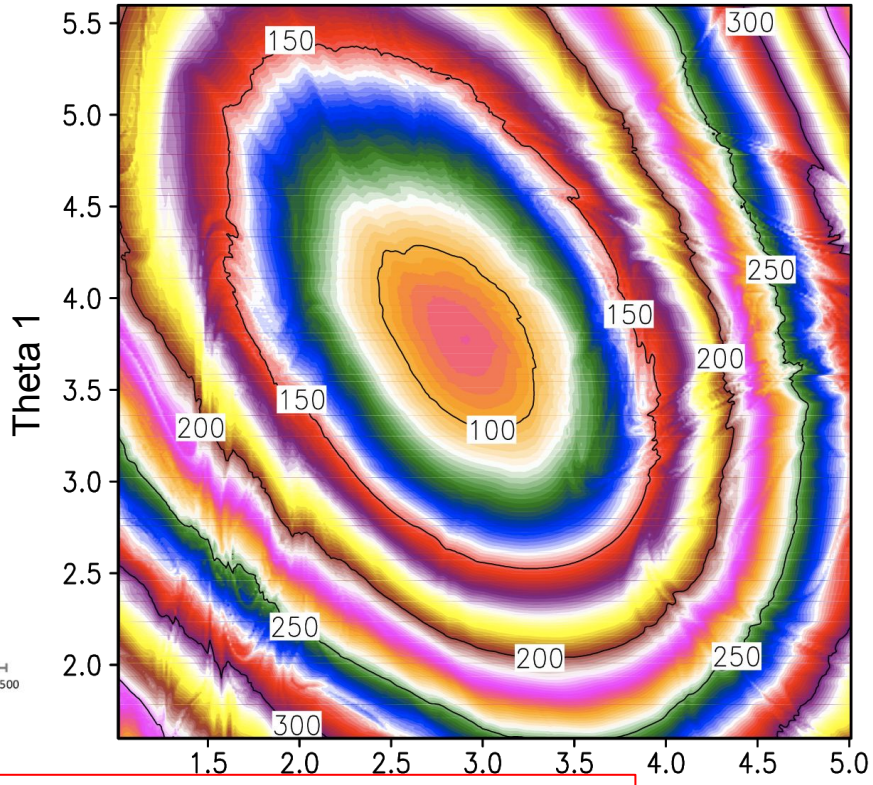




1. Notations
2. Hypothesis function and its parameters
3. Cost Function
   a. Single parameter
   b. Multiple parameters
4. Contour plots points of various h(theta0, theta1)
5. Minimize cost function

Cost Function contour:





What does a point denote in this?

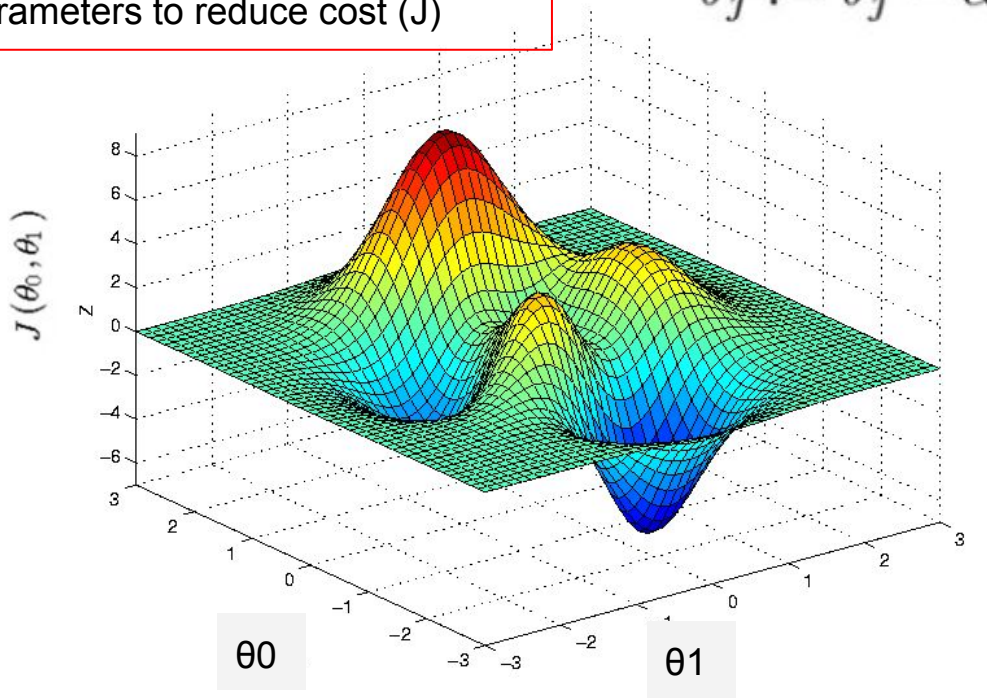
Show: Contour plots points of various h(theta0, theta1)

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.**

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

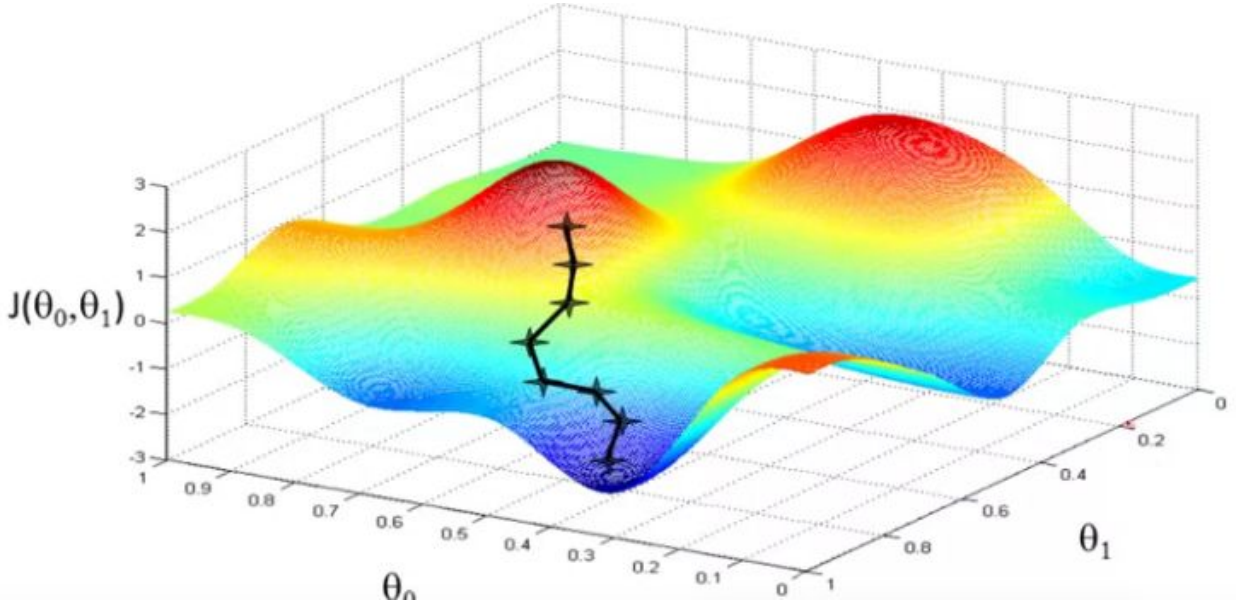
$$j = 1 : \frac{\partial}{\partial \theta_1} J(\theta_0, \theta_1)$$

$$\text{repeat until convergence } \{$$
$$\theta_0 := \theta_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)$$
$$\theta_1 := \theta_1 - \alpha \frac{1}{m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right) \cdot x^{(i)}$$
$$\}$$

Gradient Descent for Linear Reg:

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

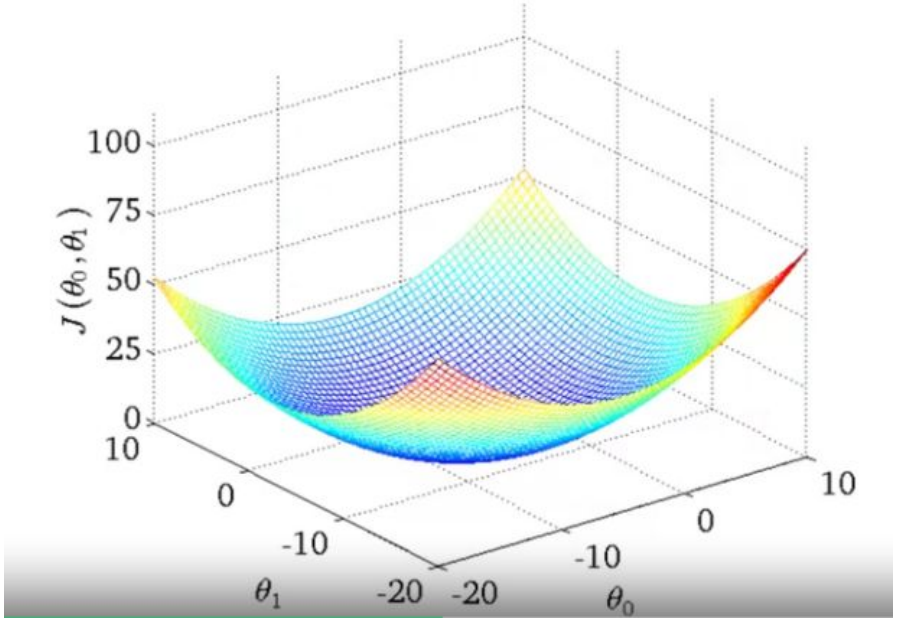
Q. What happens if algo gets stuck in a local minima?



Gradient Descent for Linear Reg:

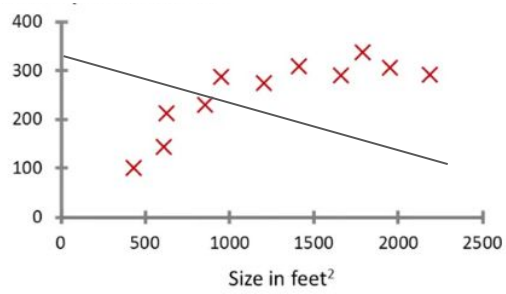
Task: Minimize J(θ0, θ1) = 1/2m (Σ(h(x) - y)^2)

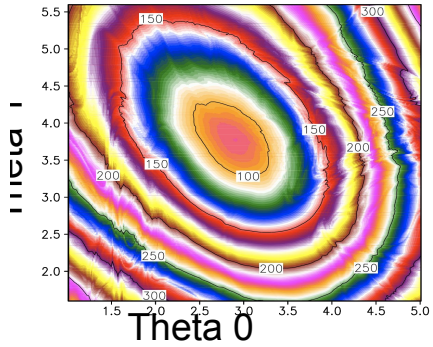
Q. What happens if algo gets stuck in a local minima?
Ans. For Linear Reg. , we have a bowl type cost function (Convex function).
Optimization problem is convex here.

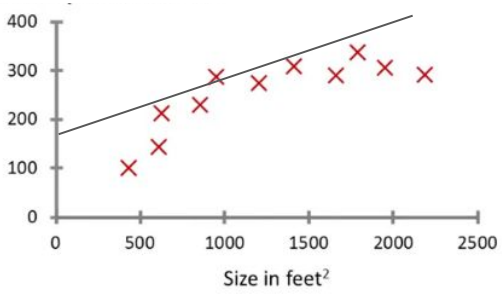


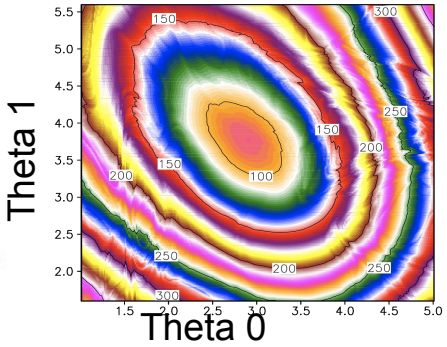
Gradient Descent for Linear Reg:

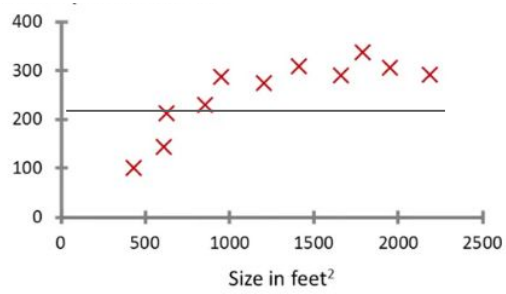
Task: Minimize J(θ0, θ1) = 1/2m (Σ(h(x) - y)^2)

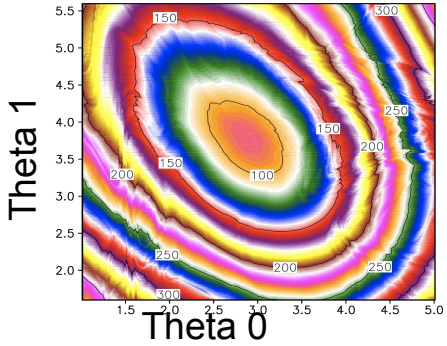


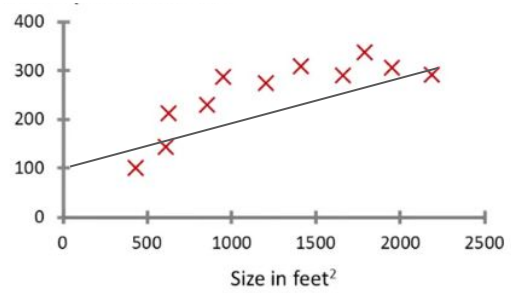


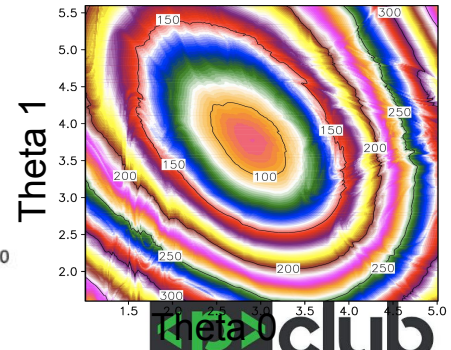




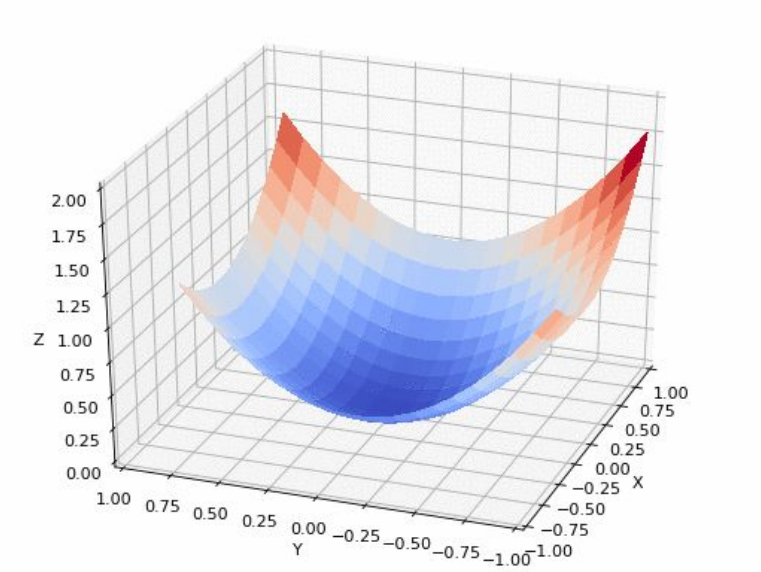


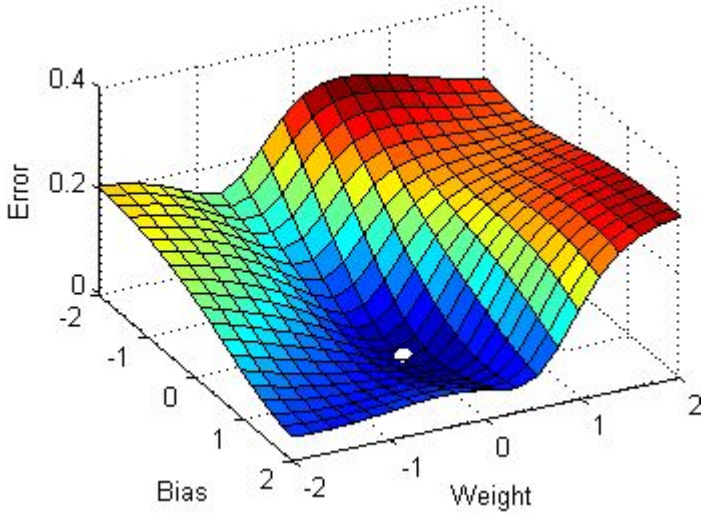




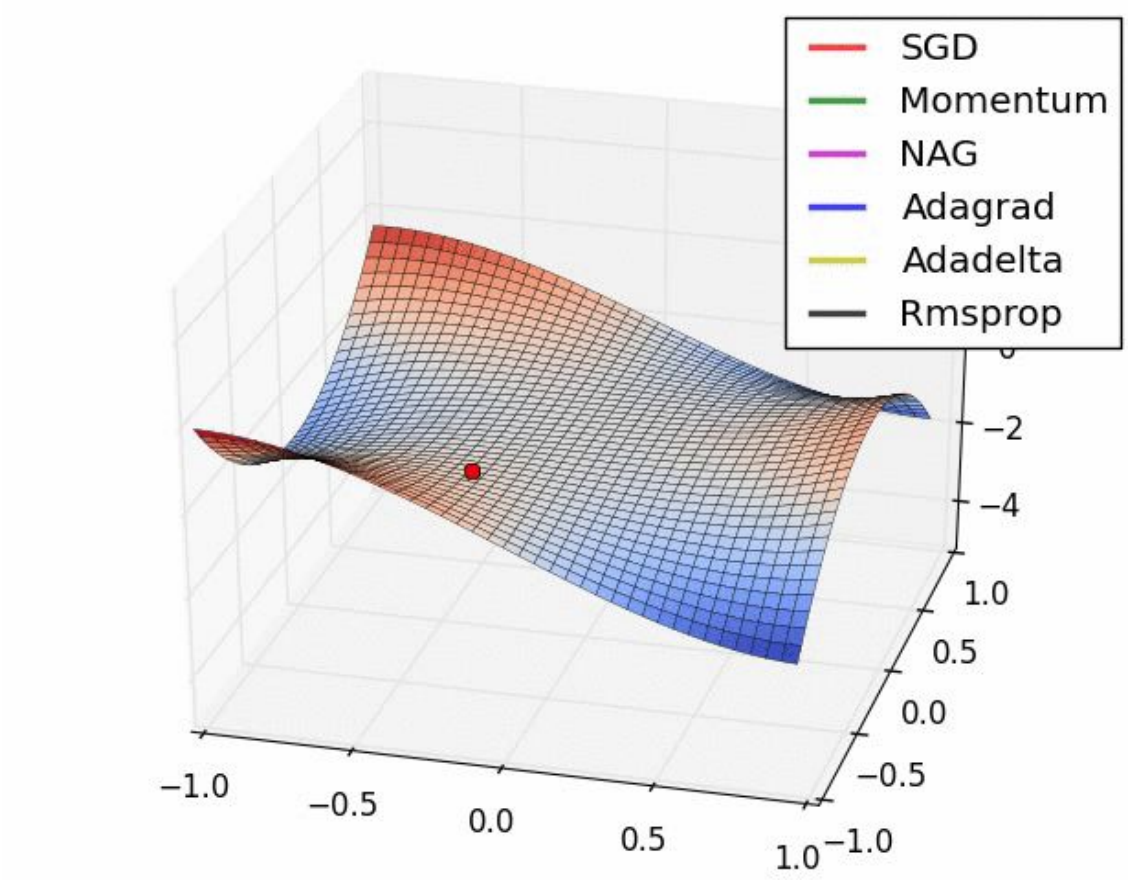


Gradient Descent for Linear Reg:



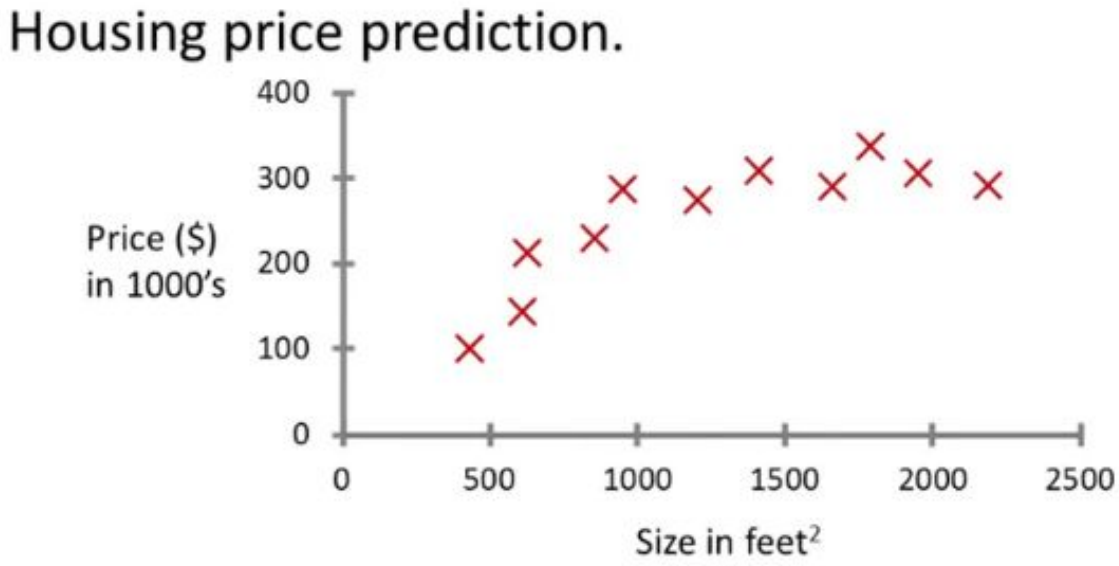


Various Descent's:

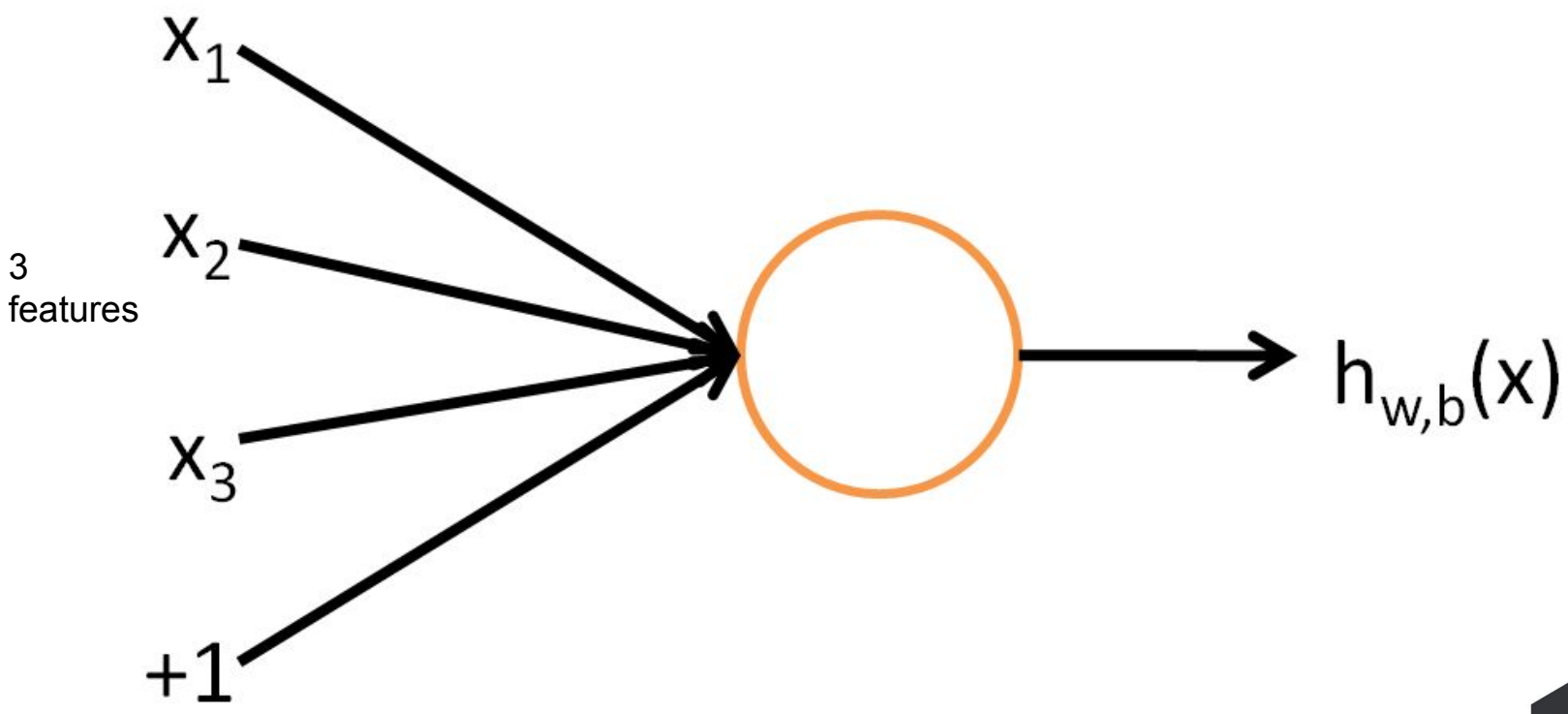


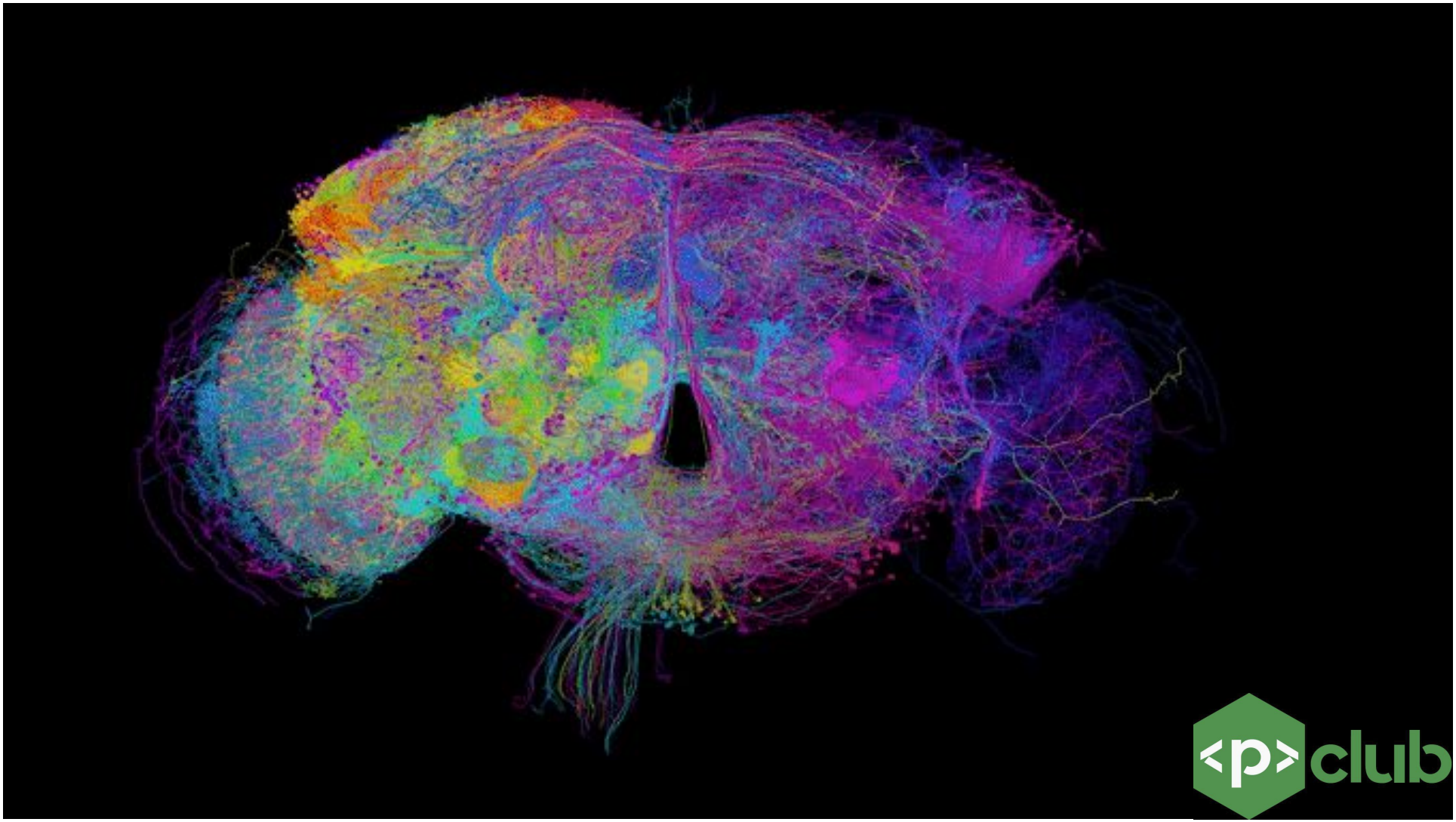
Now Let’s Dive a bit more Deep.

What is a Neural Network?



What is a Neural Network?

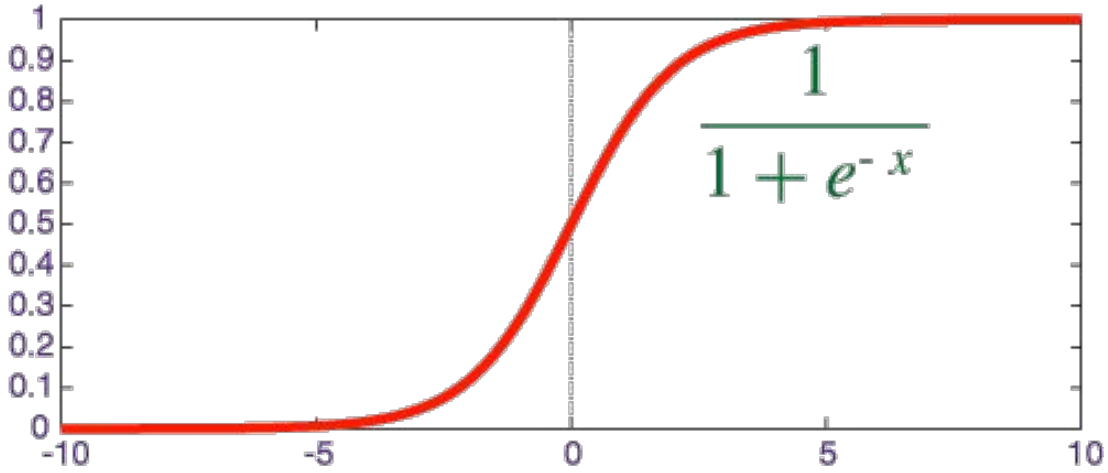



<p> club

Binary Classification problem:

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

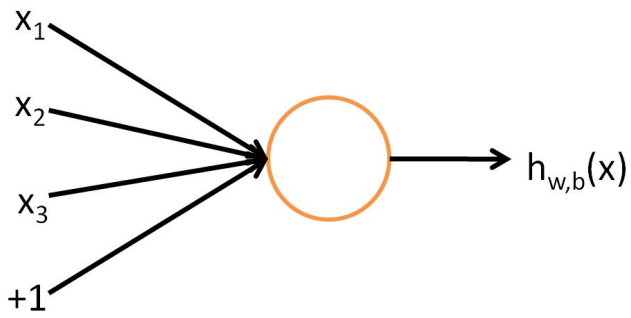
Understanding Logistic Regression problem:



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:

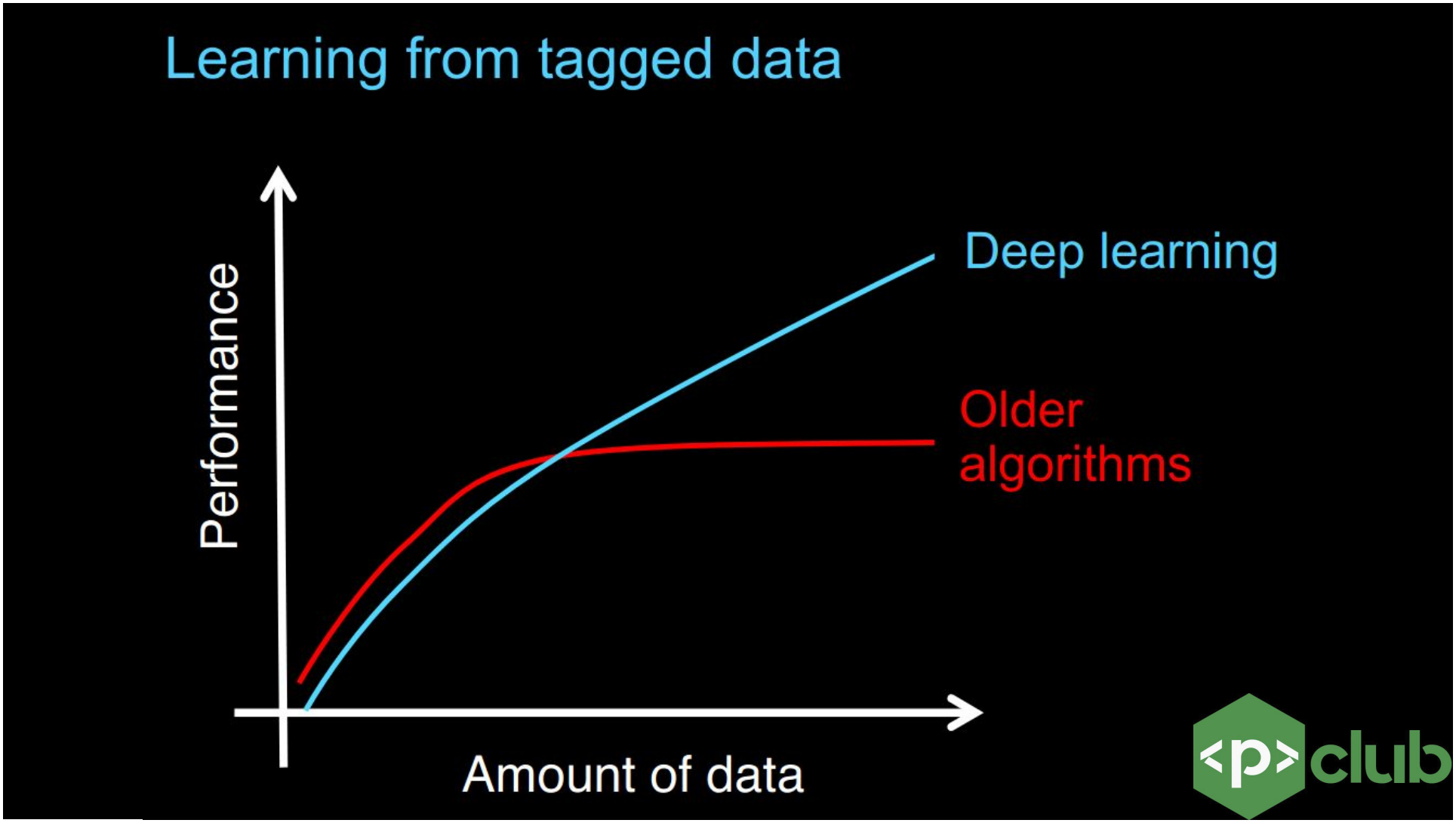
On 1 Training example

Logistic Regression Gradient Descent:

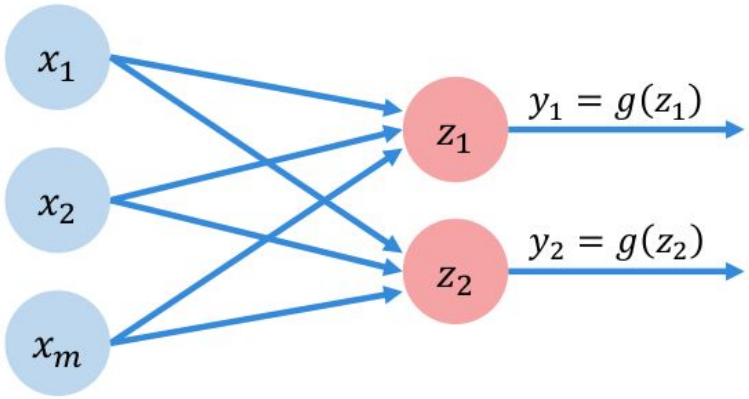
Backprop:

On m Training example

Now Let’s Dive Deeeeeeeeep in?

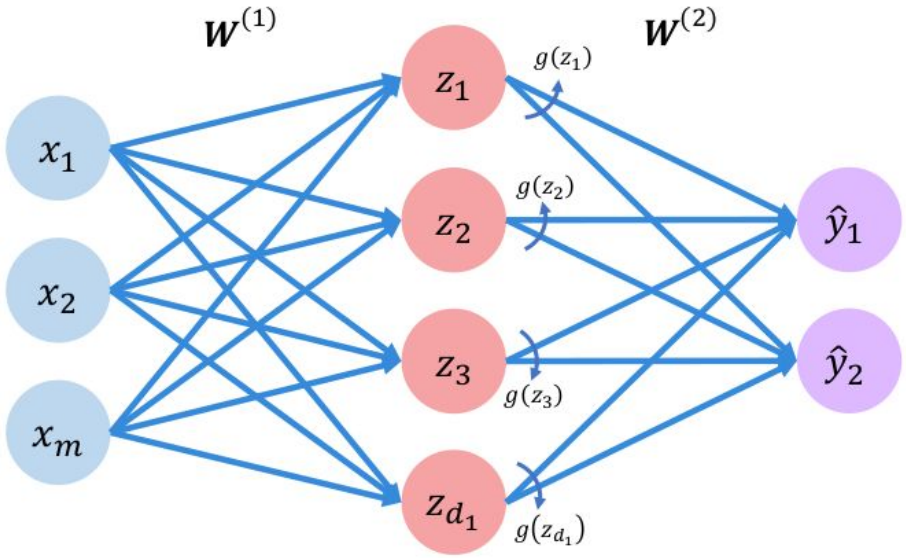

Learning from tagged data
Performance
Deep learning
Older
algorithms
Amount of data
<p> club

Multi Output Perceptron



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

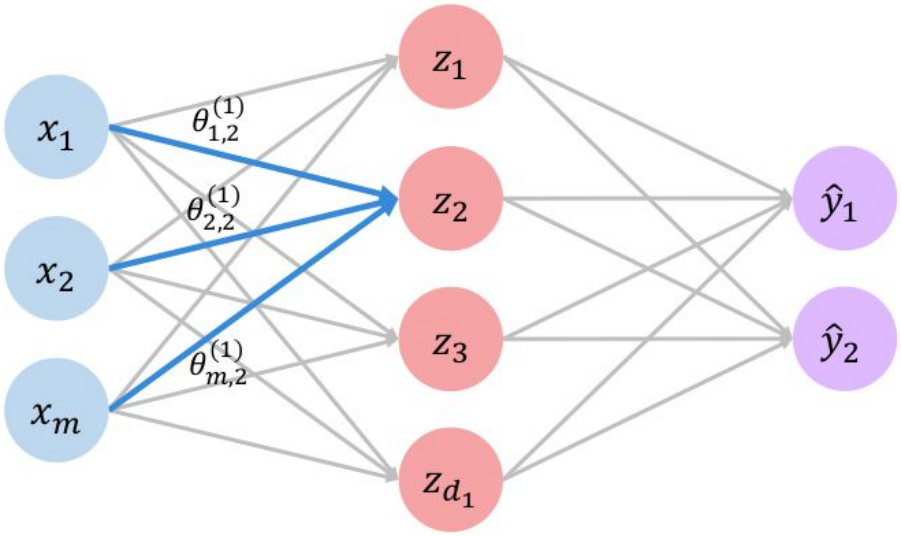
Single Layer Neural Network



Inputs Hidden Final Output

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

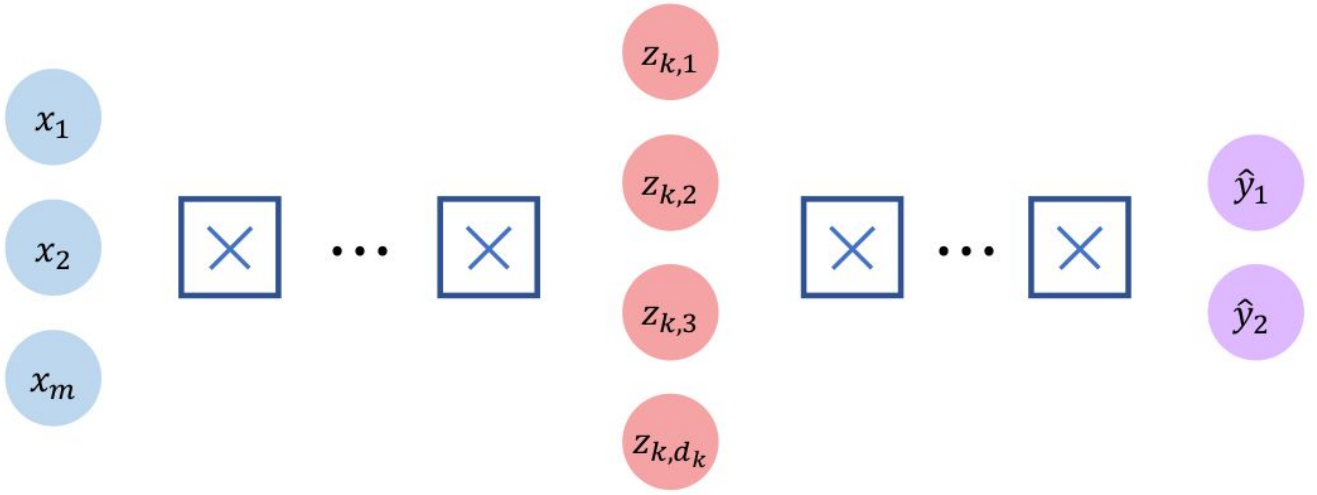
Single Layer Neural Network



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

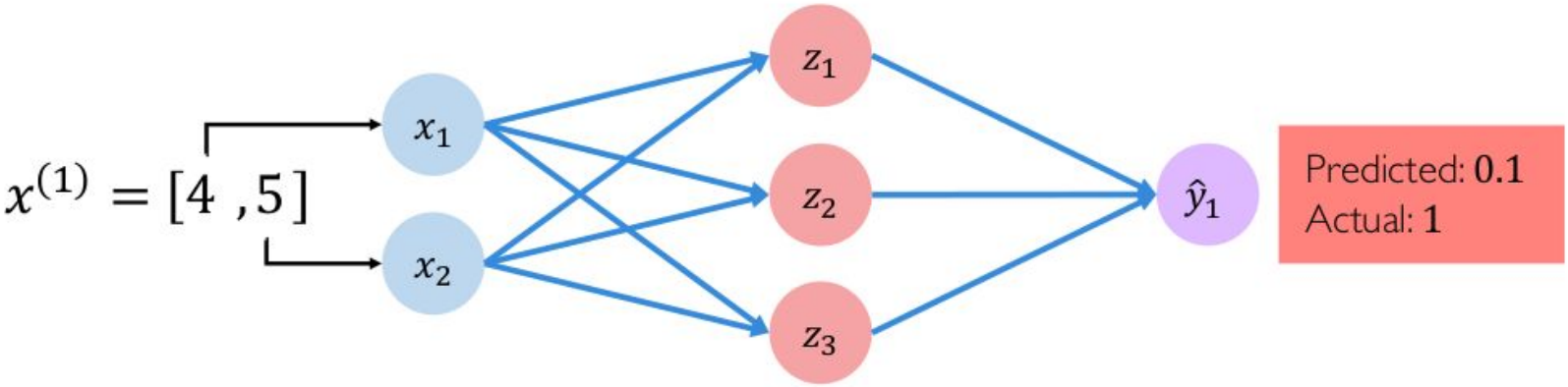
$$= w_{0,2}^{(1)} + x_1 \, w_{1,2}^{(1)} + x_2 \, w_{2,2}^{(1)} + x_m \, w_{m,2}^{(1)}$$

Deep Neural Network



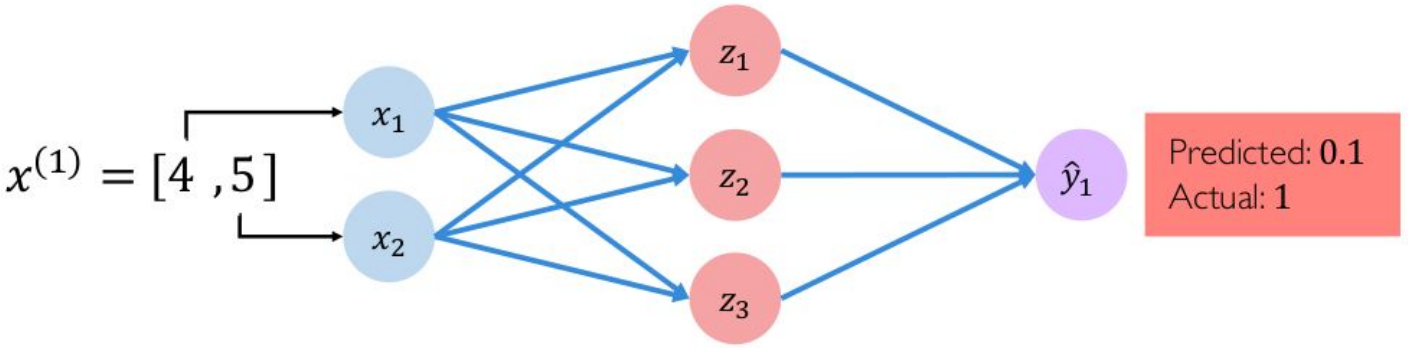
$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{d_{k-1}} g(z_{k-1,j})\, w_{j,i}^{(k)}$$

Example Problem: Will I pass this class?



Quantifying Loss

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



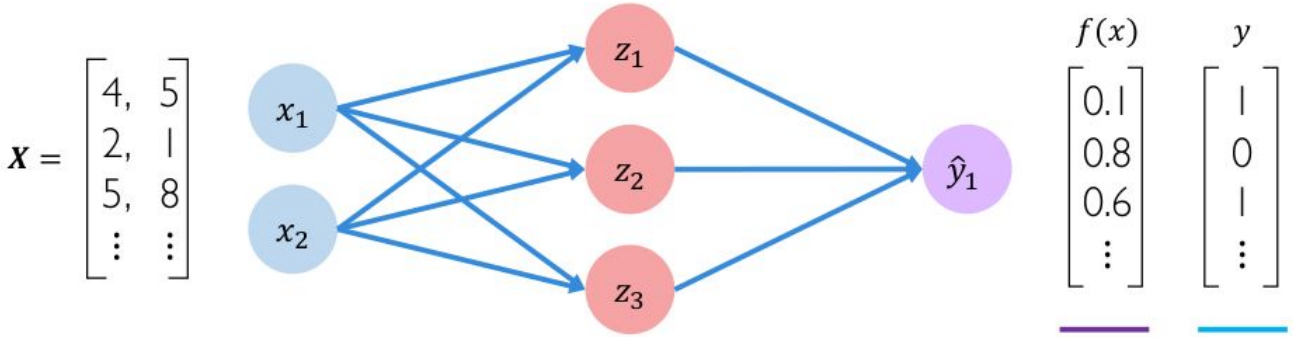
$$\mathcal{L}\left(f\left(x^{(i)}; \boldsymbol{W}\right), y^{(i)}\right)$$

Predicted Actual

Empirical Loss

(Cost Function)

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



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

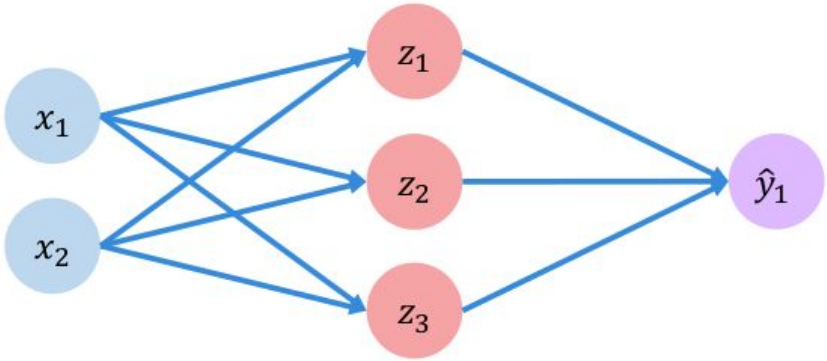
$$J(W) = \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}\left(f\left(x^{(i)}; W\right), y^{(i)}\right)$$

Predicted Actual

Binary Cross Entropy Loss

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

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



$f(x)$

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix}$$

y

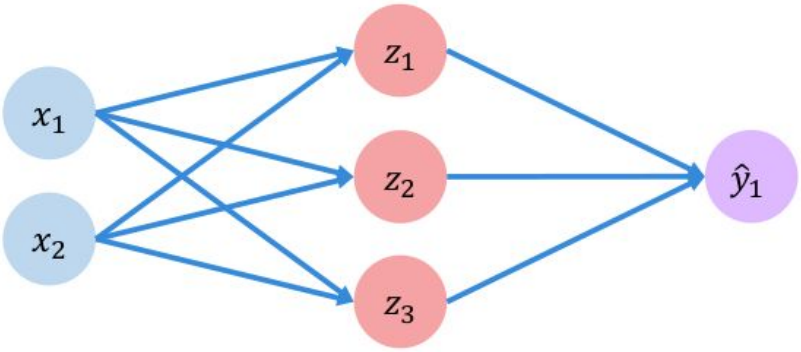
$$\begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

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

Mean Squared Error Loss

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

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



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

Actual Predicted

$f(x)$ y

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

Final Grades (percentage)

Loss Optimization

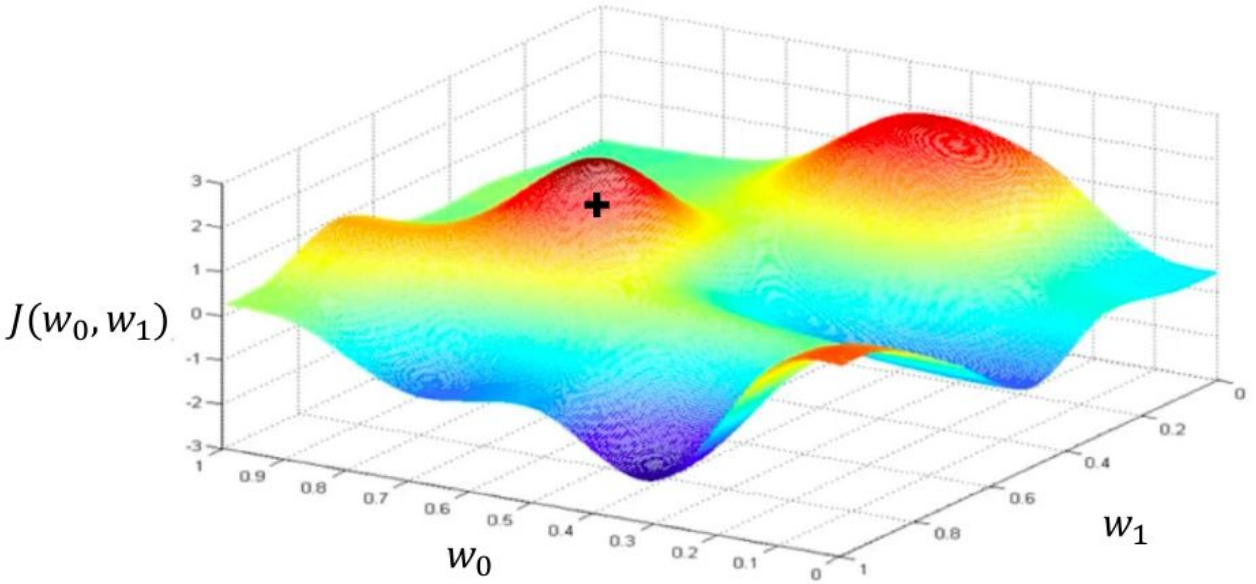
We want to find the network weights that ***achieve the lowest loss***

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

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

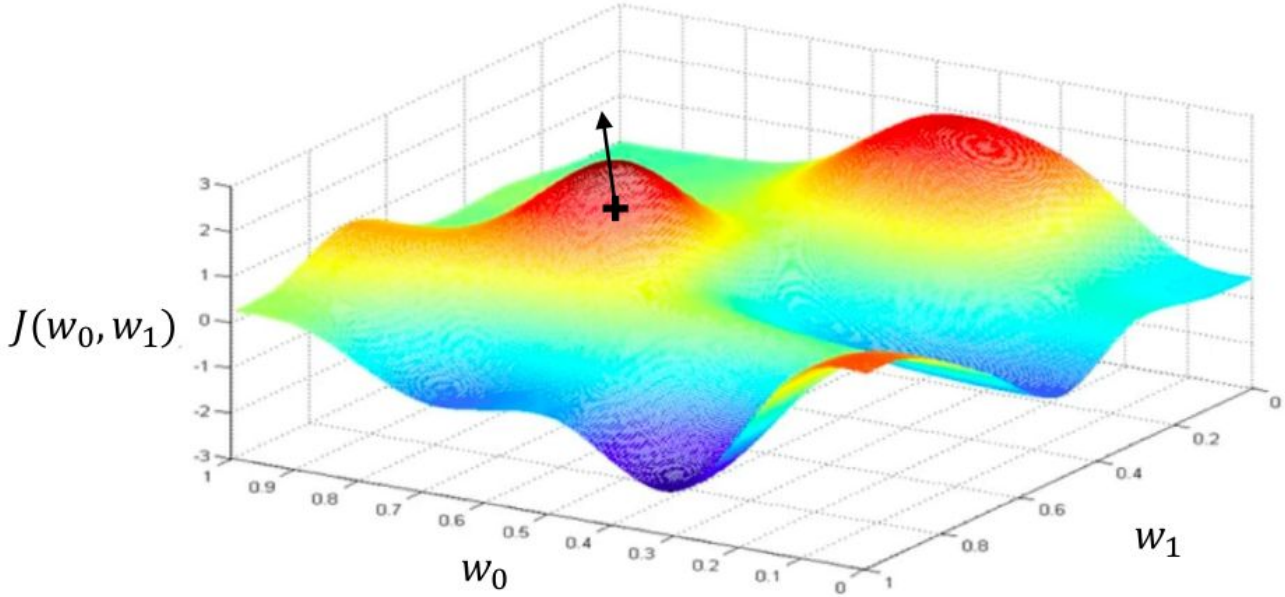
Loss Optimization

Randomly pick an initial (w_0, w_1)



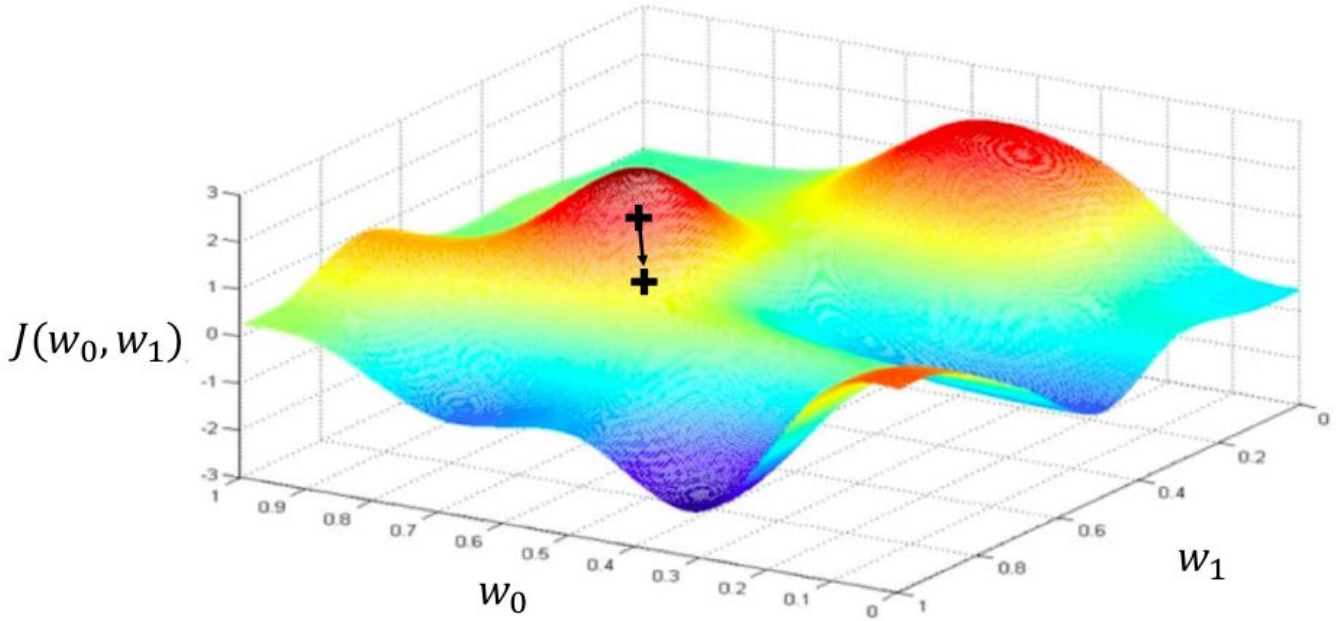
Loss Optimization

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



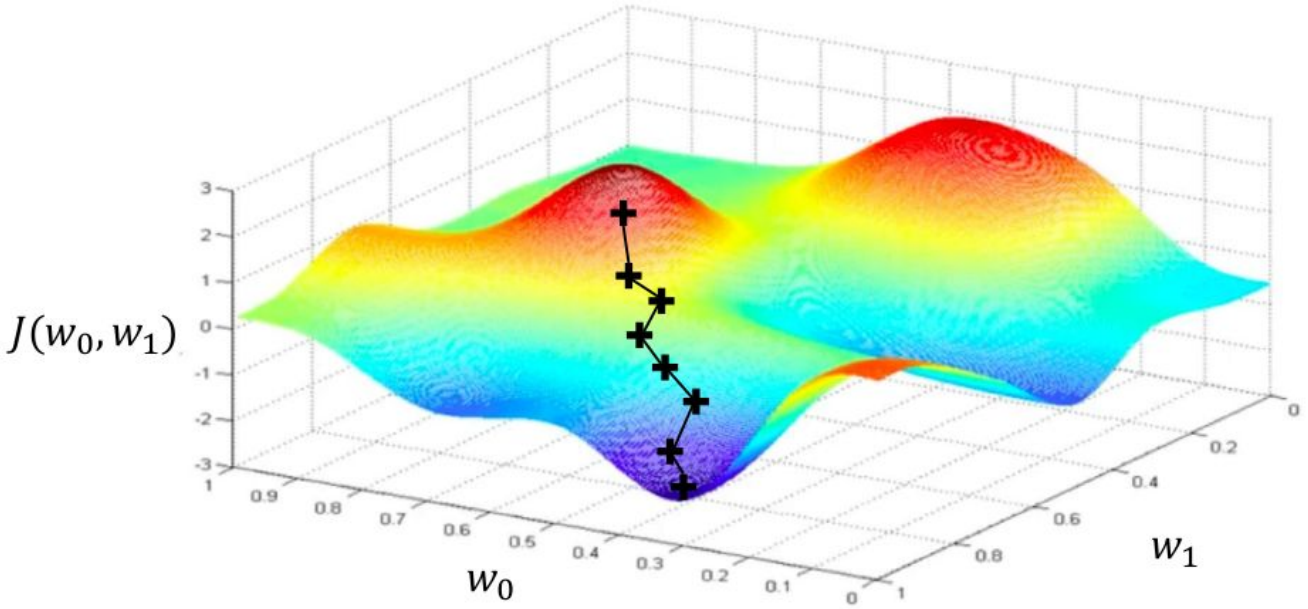
Loss Optimization

Take small step in opposite direction of gradient



Gradient Descent

Repeat until convergence



Gradient Descent

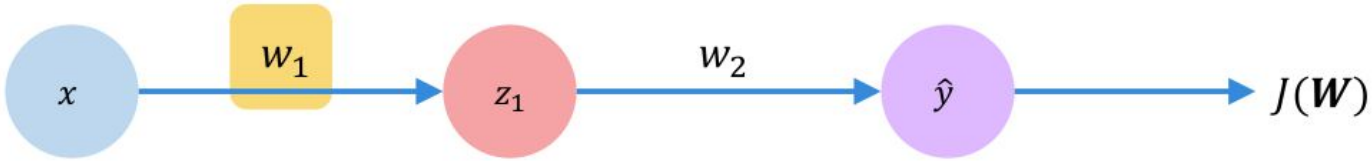
Algorithm

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

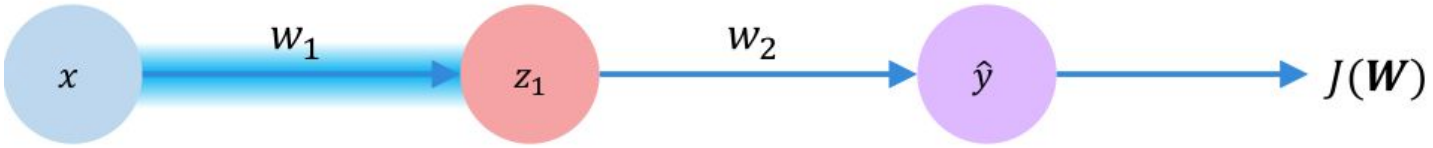


$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial w_1}$$

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

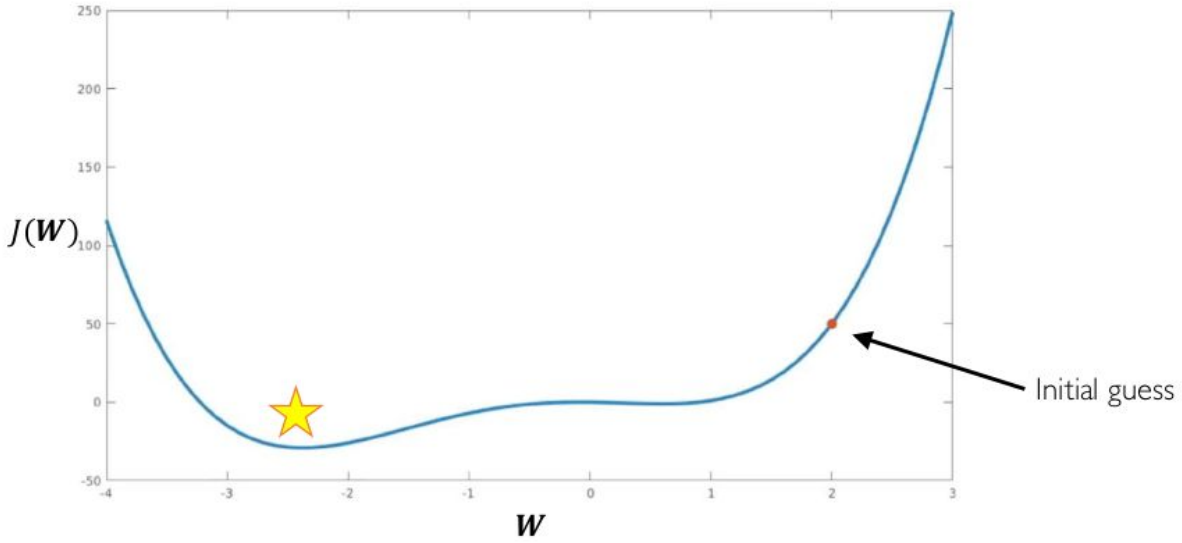
$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$$

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:

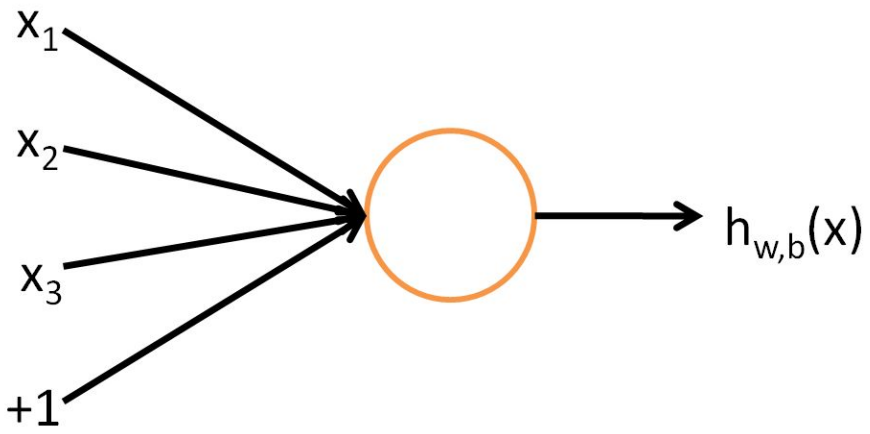
Do something smarter!
Design an adaptive learning rate that "adapts" to the landscape

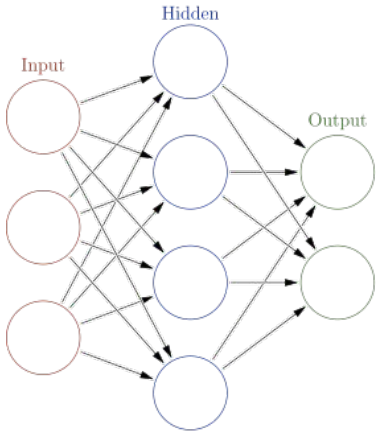
Adaptive Learning Rates

- 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?



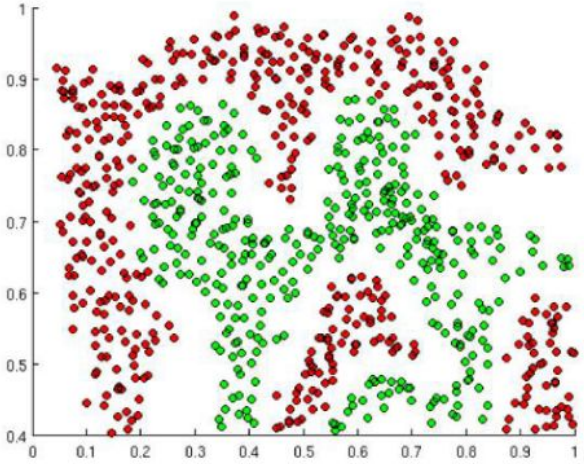


Linear hidden layer is useless.

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*

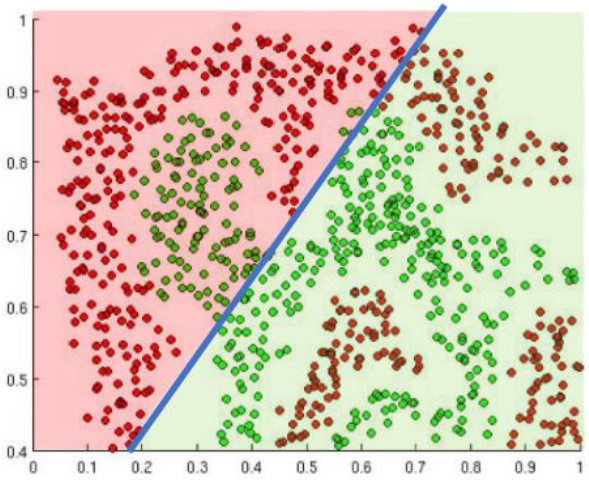


What if we wanted to build a Neural Network to distinguish green vs red points?

6.S191 Introduction to Deep Learning
introtodeeplearning.com

Importance of Activation Functions

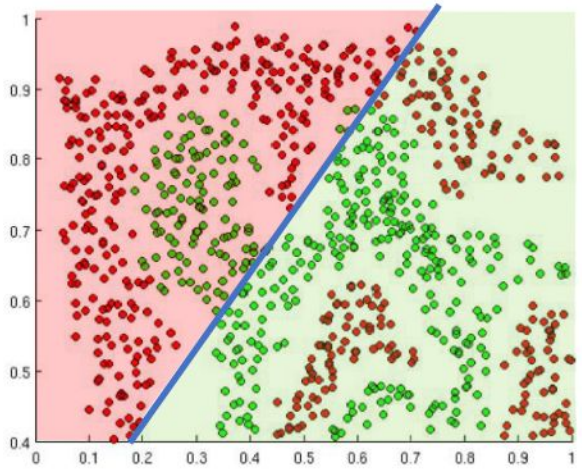
*The purpose of activation functions is to **introduce non-linearities** into the network*



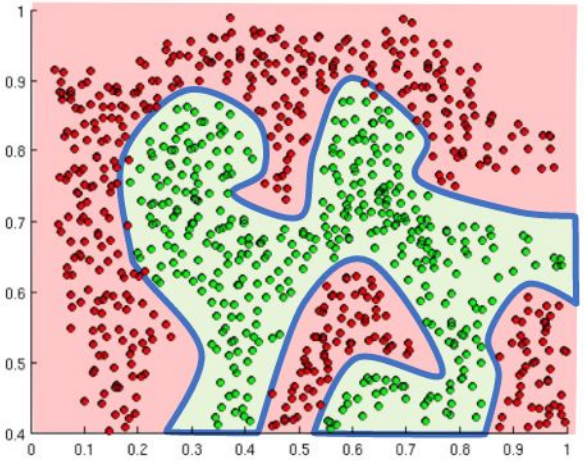
Linear Activation functions produce linear decisions no matter the network size

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

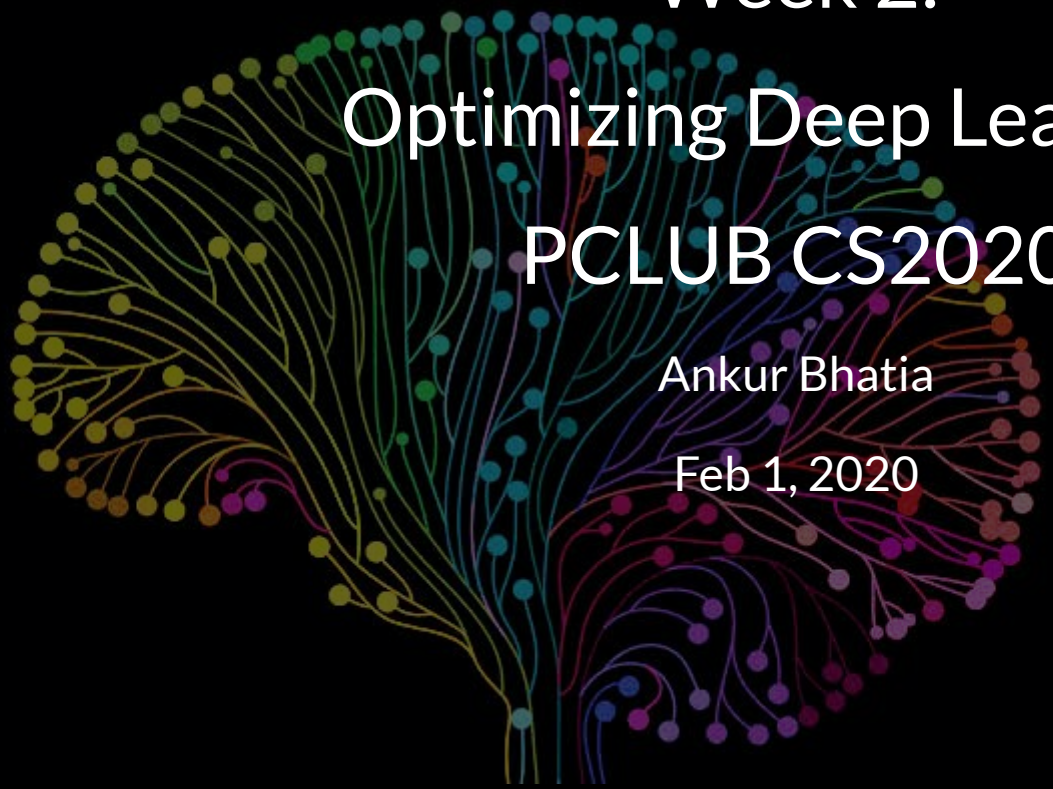


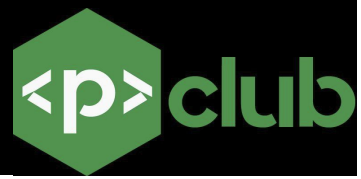
Non-linearities allow us to approxi arbitrarily complex functions

Week 2:
Optimizing Deep Learning
PCLUB CS2020

Ankur Bhatia

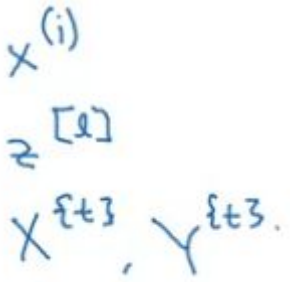
Feb 1, 2020





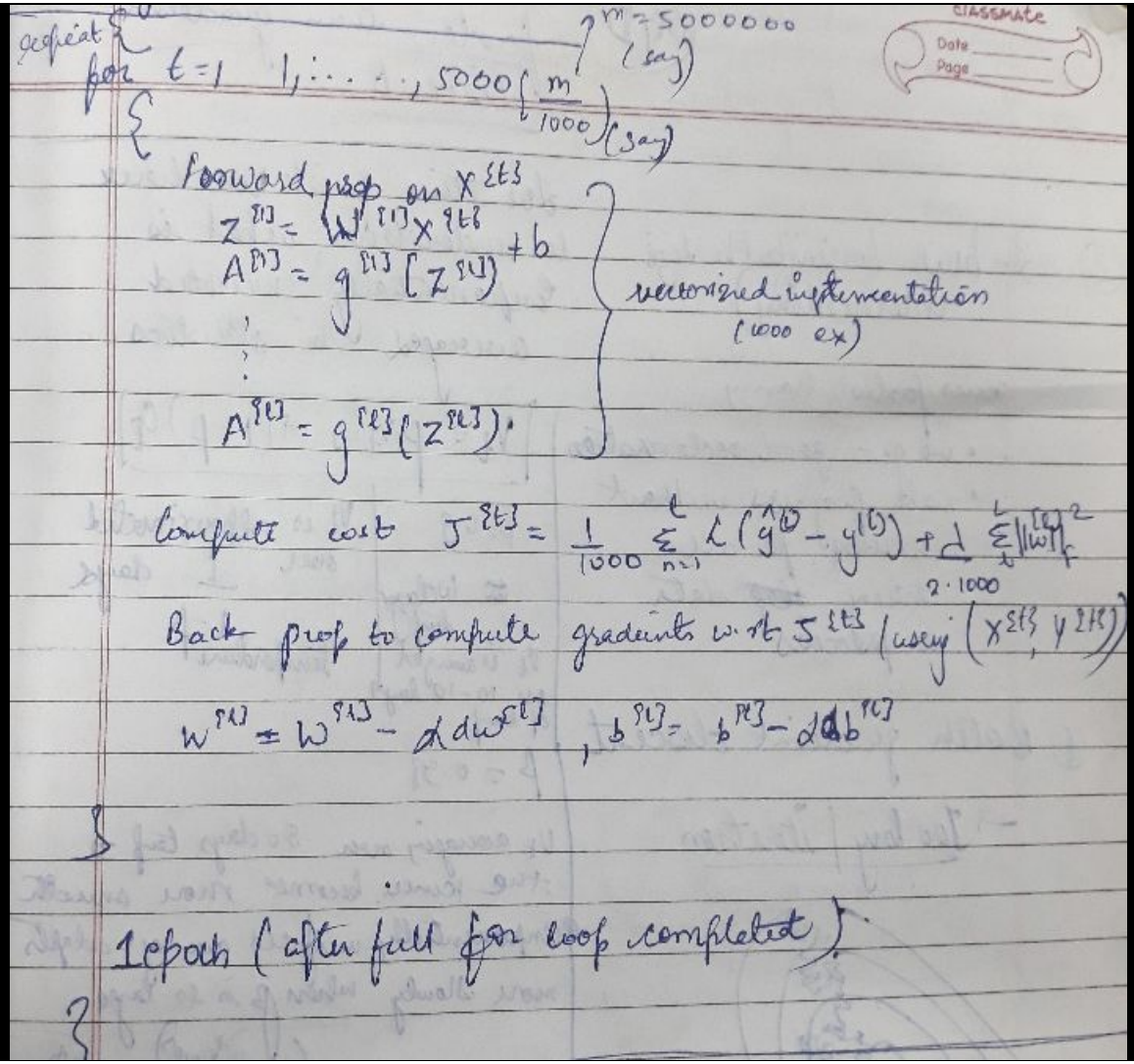
Optimizing NNs.

Batch (Vanilla) vs Mini Batch Gradient Descent

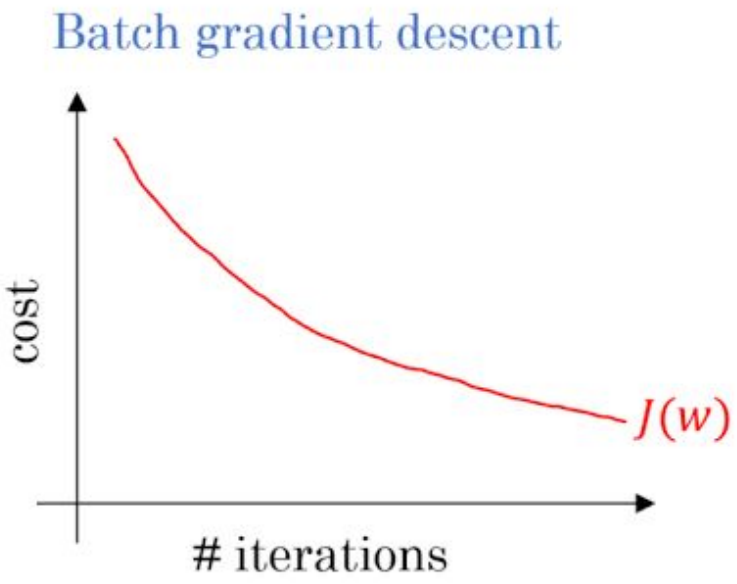


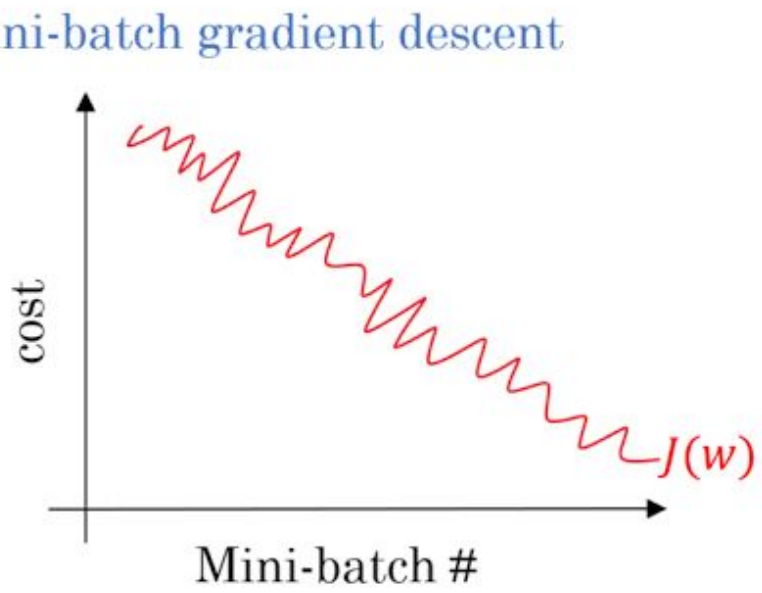
1. ith training example.
2. L - layer
3. mini-batch

Mini Batch Gradient Descent.



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'.

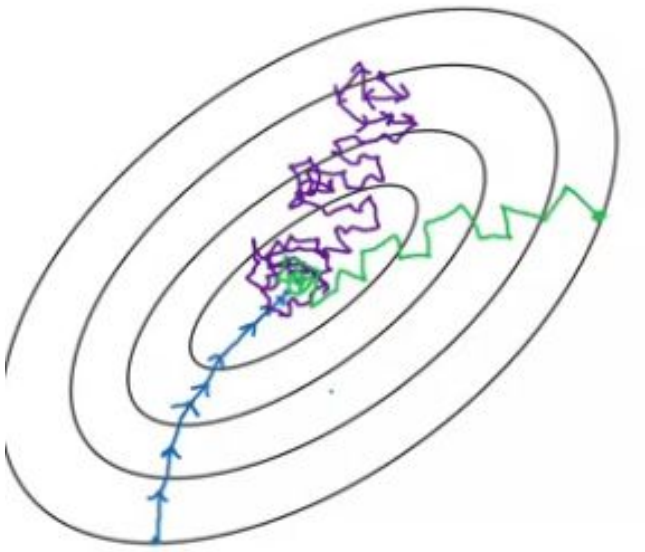




One Mini batch may be easy and other may be difficult to learn.
Parameter to choose - Size of minibatch

Mini Batch Gradient Descents

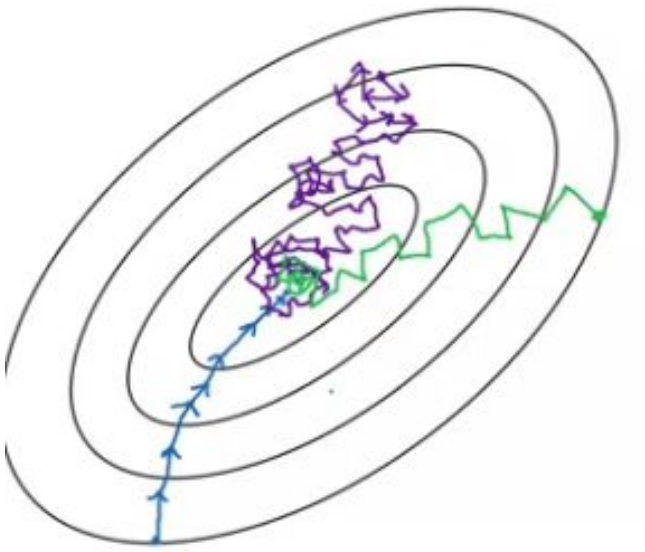
If m = 1; Stochastic Gradient Descent => Easy to compute but very noisy.
If m = m; Batch Gradient Descent



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

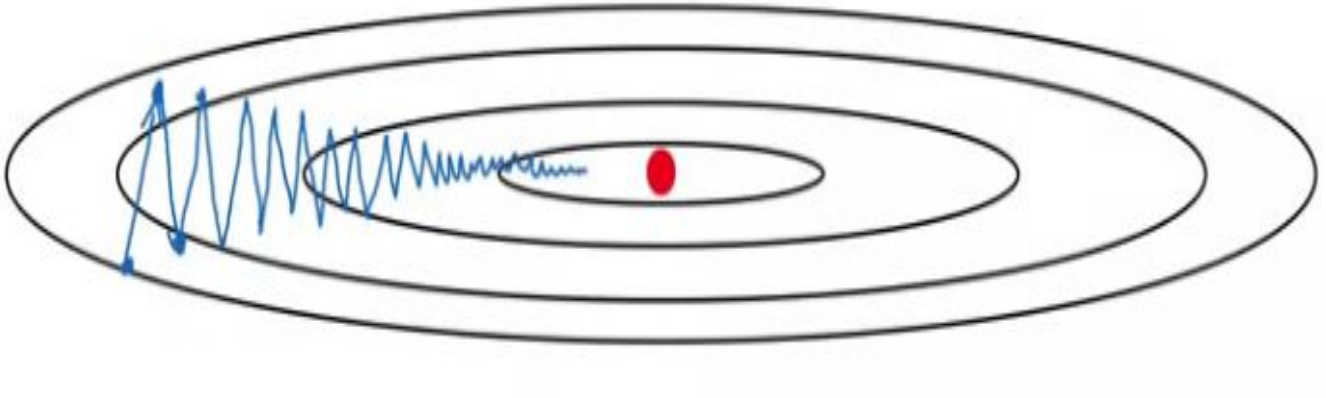


1. For small Trainx - m = m;
2. General mini batch sizes = 64, 128, 256…
3. mini - batch should fit in CPU - GPU m/m.

Optimization Algorithms faster than GD.

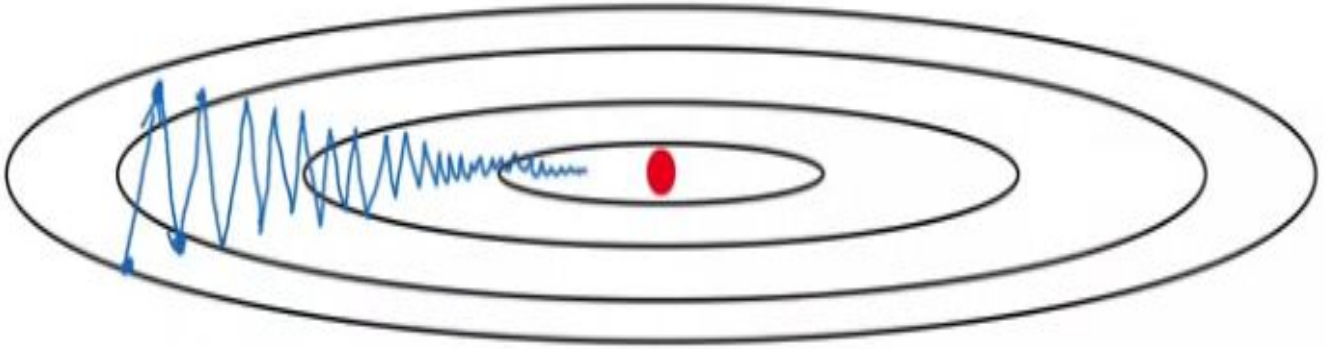
1. Gradient Descent with momentum

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:

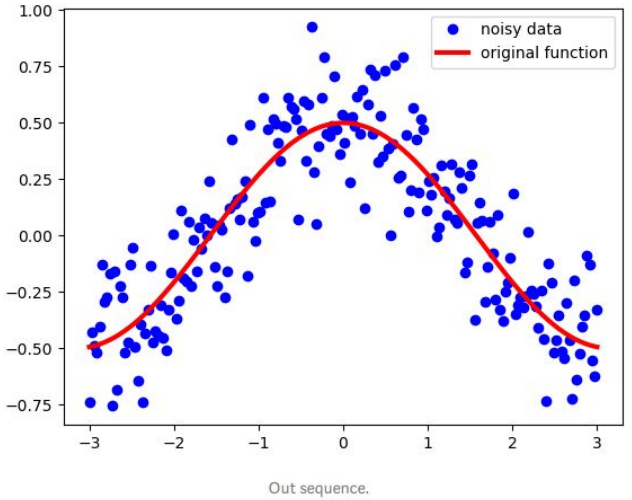
Idea is to use exponentially weighted averages across the horizontal and vertical at each time step to average out and make the next steps smooth.

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:

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

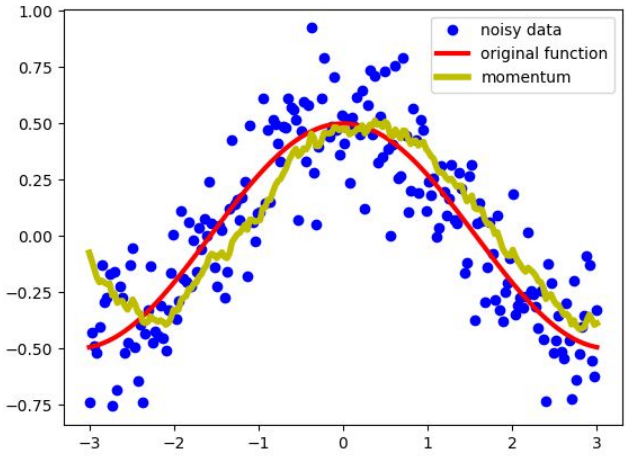


1. Gradient Descent with momentum : Exponentially weighted averages

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

Exponentially weighted averages:

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

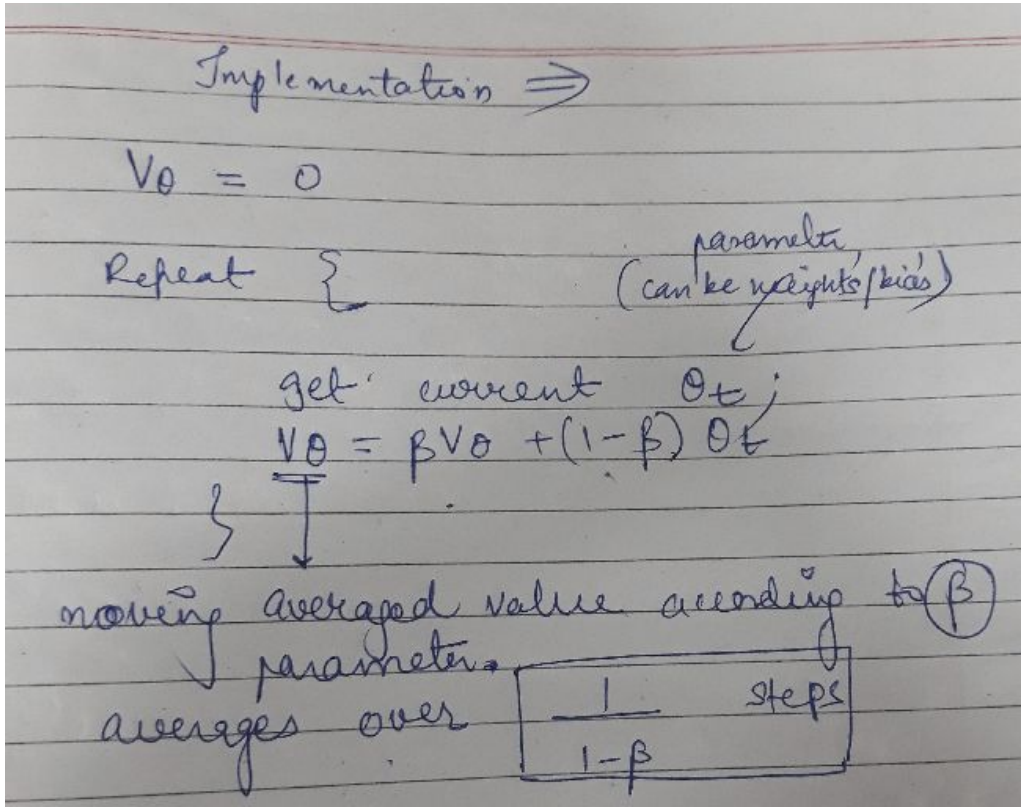


momentum — data from exponentially weighed averages.

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

<p> club

1. **Gradient Descent with momentum : Exponentially weighted averages**



Optimization Algorithms faster than GD.

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

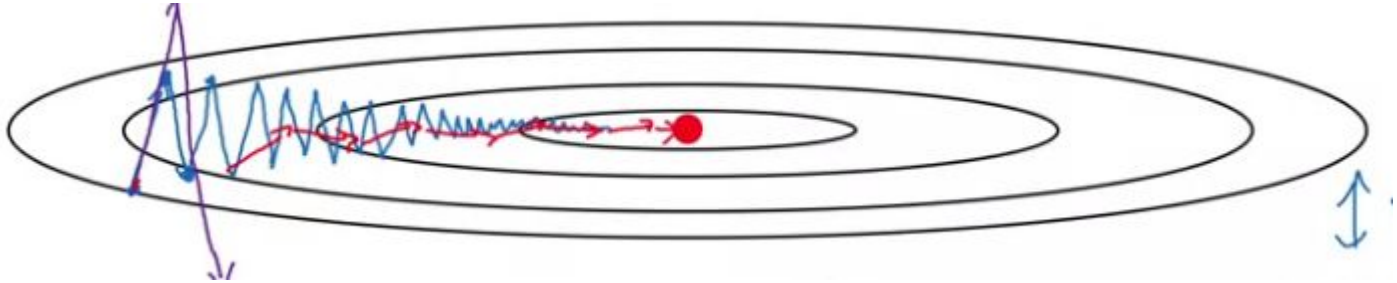
$$v_{dW} = \beta v_{dW} + (1 - \beta) dW$$

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

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

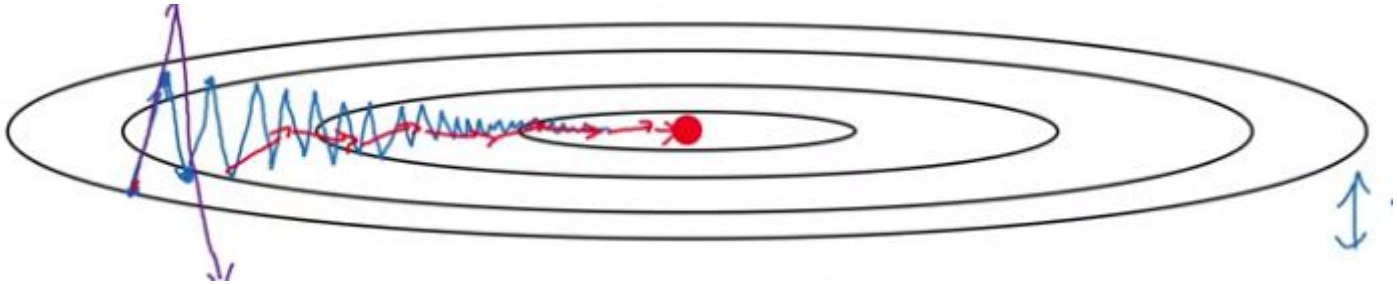
Hyperparameters: α, β $\beta = 0.9$

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

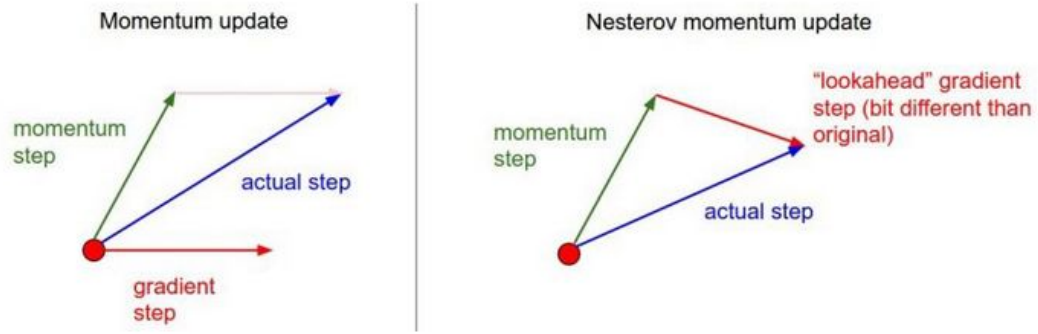
momentum

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

So after using Momentum, we can also increase the learning rate a bit because the algorithm automatically tries to maintain the updates.

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.

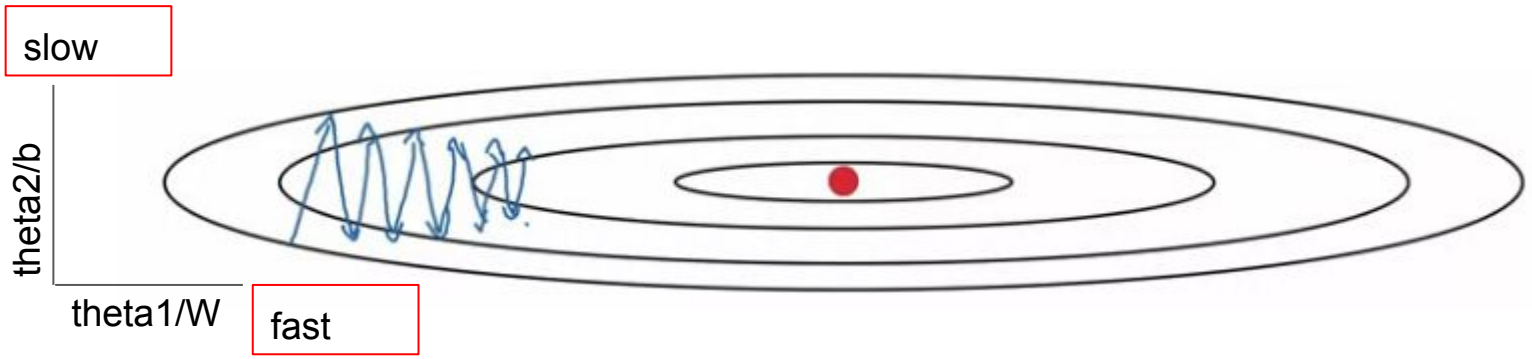


Source (Stanford CS231n class)

Optimization Algorithms faster than GD.

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

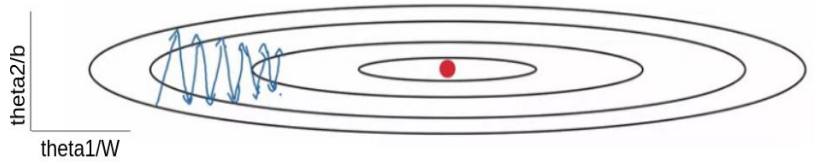
$$v_{dw} = \beta \cdot v_{dw} + (1 - \beta) \cdot dw^2$$ ← Elementwise

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

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

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

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

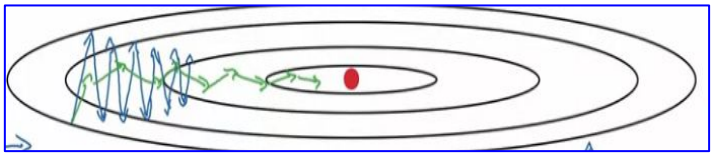
$$v_{db} = \beta \cdot v_{dw} + (1 - \beta) \cdot db^2$$

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:

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

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

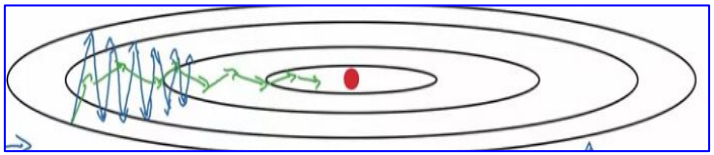
$$W = W - \alpha \cdot \frac{dw}{\sqrt{v_{dw}} + \epsilon}$$

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

Vdw to be small

Vdb to be large

Or mentioned Sdw in place of Vdw



- **Why called RMS?**
- **Why is epsilon used? -** In adam paper - epsilon mentioned 10^-8 so that algo doesn't divide by 0.

2. Gradient Descent with Root Mean Square Propagation:

Dimensions will be large.

W1, W2, W3, ……., b1, b2, b3, …….

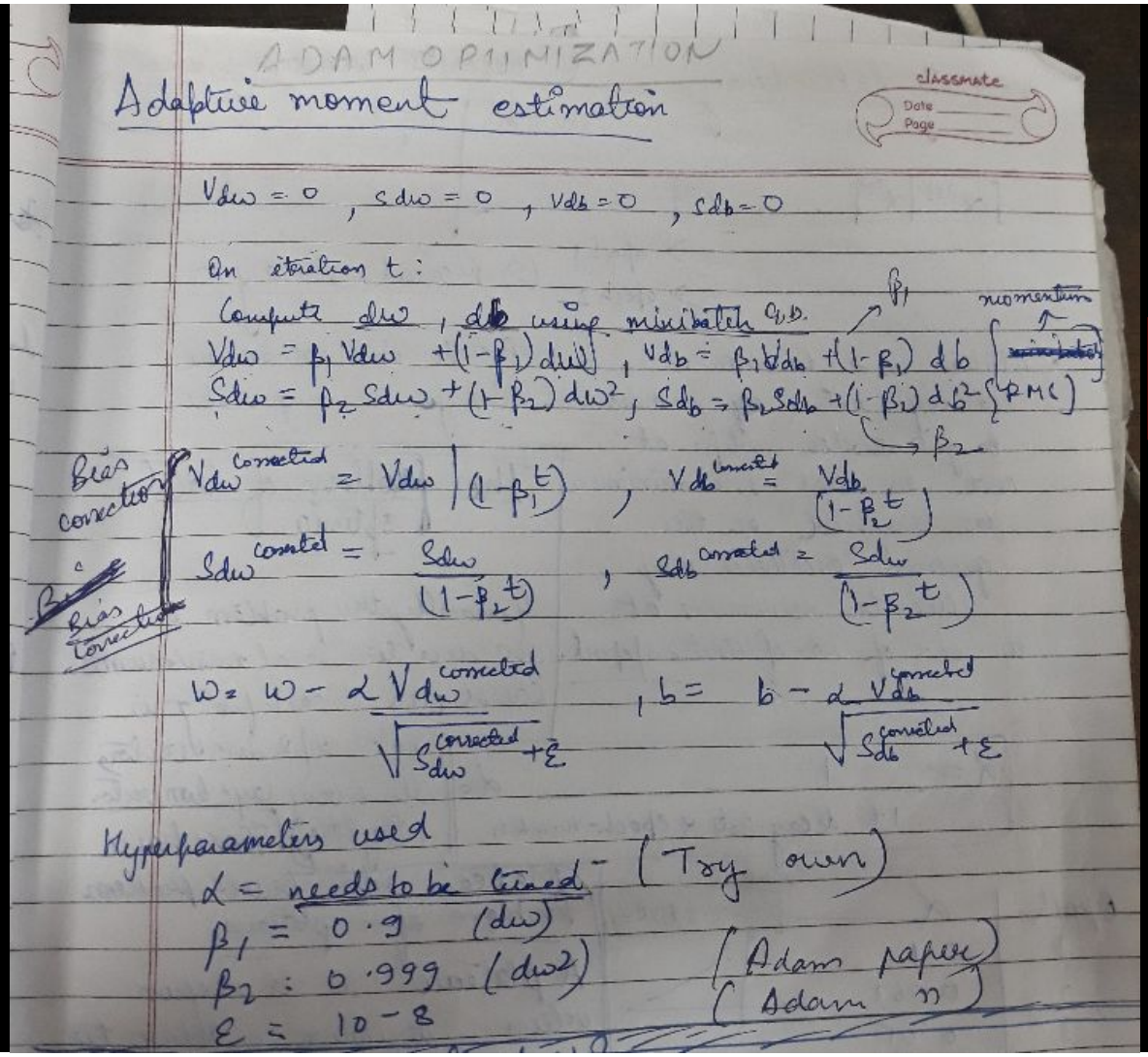
So damping out in vertical direction than in horizontal direction would actually mean - say damping out (or slowing the updates) in w1, w2, b5, w4 …… direction and fastening the updates in say w3, w5, b1, b2,…. direction.

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 FORSTOCHASTICOPTIMIZATION - ICLR 2015

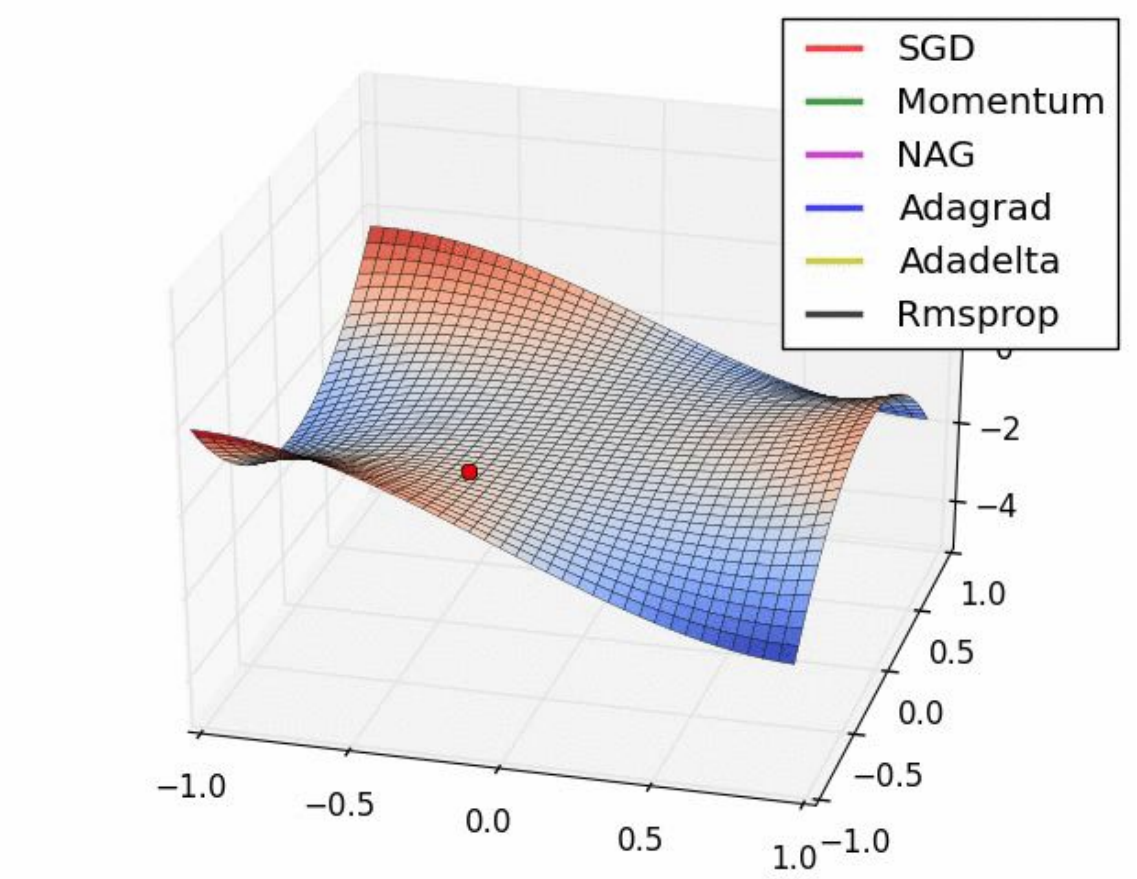
Diederik P. Kingma* , Jimmy Lei Ba* https://arxiv.org/pdf/1412.6980.pdf

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

Applied Deep learning in one domain do not appies in other

ITERATIVE PROCESS

1. Depends on amount of data.
2. GPU/CPU
3. No. of input features
4. Type of data

Still we cannot choose the hyperparameters well.

Applied Deep learning in one domain do not appies in other

ITERATIVE PROCESS

1. Setting up data well in train, test and dev.
2. Splitting percentages.

Splitting up of Train/Test/Dev sets

1. Smaller sets. - 100, 10^3, 10^4
2. Larger sets (Big Data) - > 10^6

Distribution of Train and Test Sets.

Examples

1. **Effects of online vs offline data.**
2. **Effects of user mischief.**

Dev and Test sets should be from same distributions.

Dev set / cross validation set / development set

Test set not necessary. /Call test set dev set.

Bias and Variance

BIAS:

Bias is disproportionate weight *in favor of* or *against* an idea or thing, usually in a way that is closed-minded, prejudicial, or unfair. Biases can be innate or learned. People may develop biases for or against an individual, a group, or a belief.[1] In science and engineering, a bias is a systematic error. Statistical bias results from an unfair sampling of a population, or from an estimation process that does not give accurate results on average.[2]

Bias and Variance

VARIANCE:

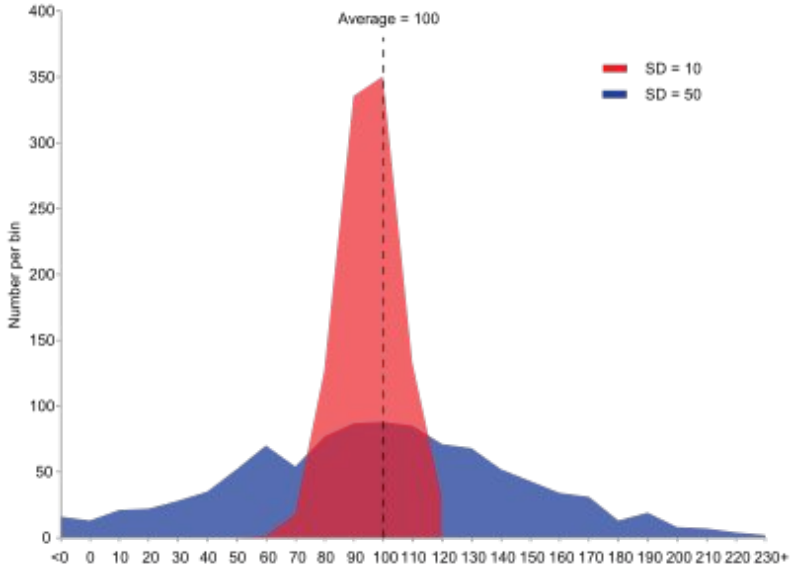
In probability theory and statistics, **variance** is the expectation of the squared deviation of a random variable from its mean. Informally, it measures how far a set of (random) numbers are spread out from their average value. Variance has a central role in statistics, where some ideas that use it include descriptive statistics, statistical inference, hypothesis testing, goodness of fit, and Monte

Bias and Variance

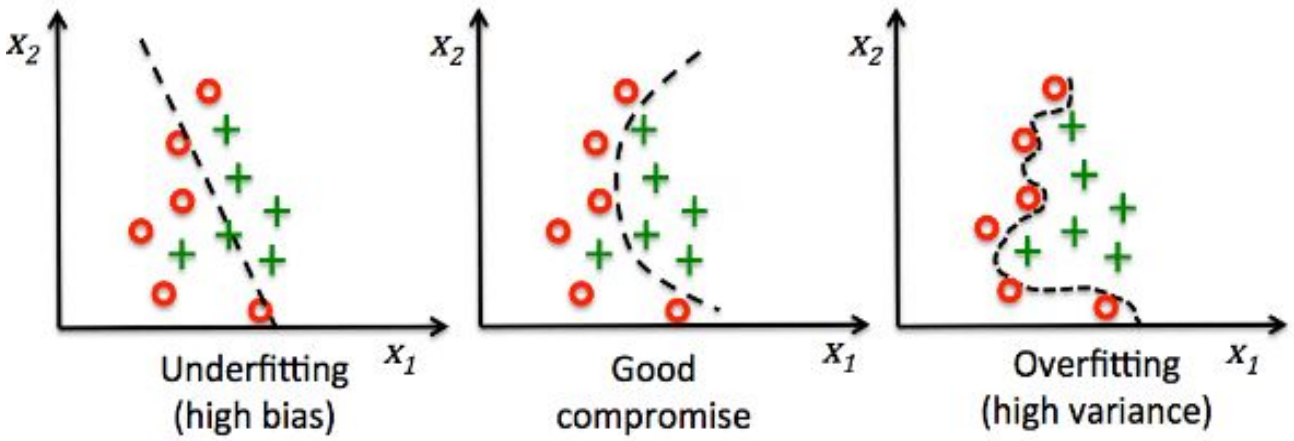
VARIANCE:

The variance of a random variable X is the expected value of the squared deviation from the mean of X, $\mu = \mathrm{E}[X]$:

$$\operatorname{Var}(X) = \mathrm{E}\left[(X - \mu)^2\right].$$



Bias and Variance



Logistic Reg. is not
a good fit.

Bias and Variance

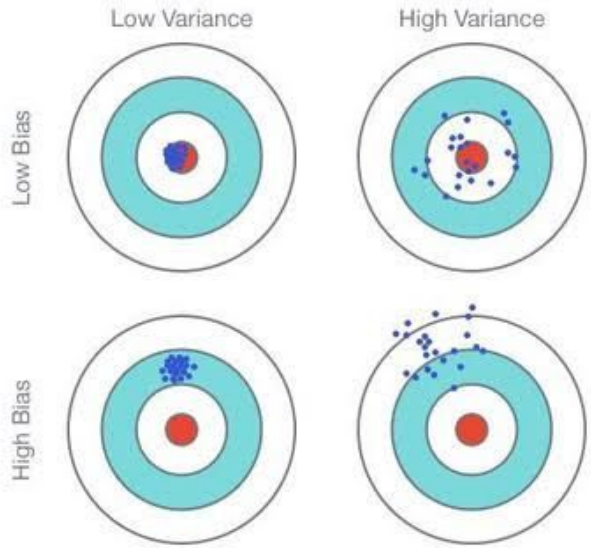
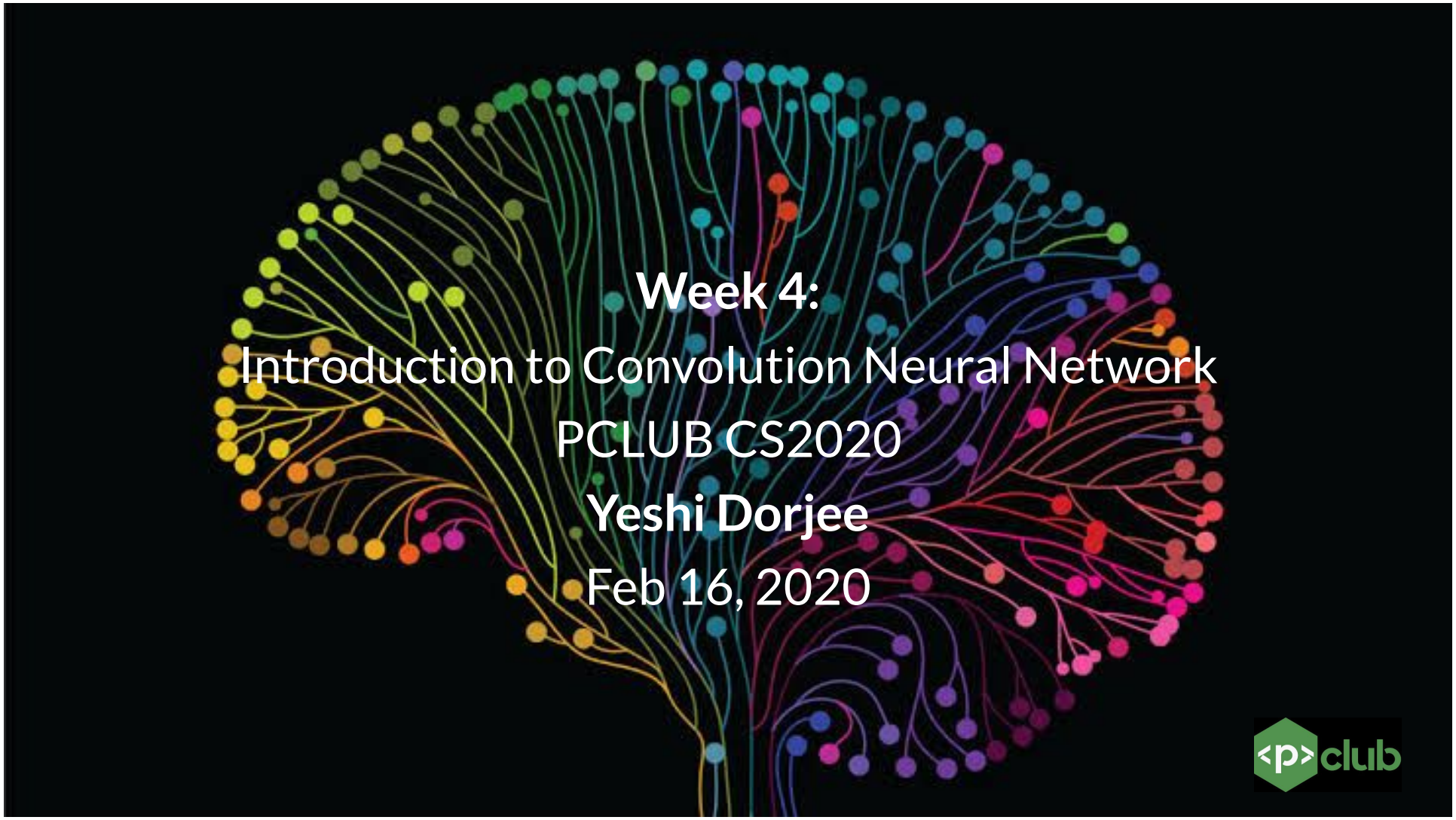
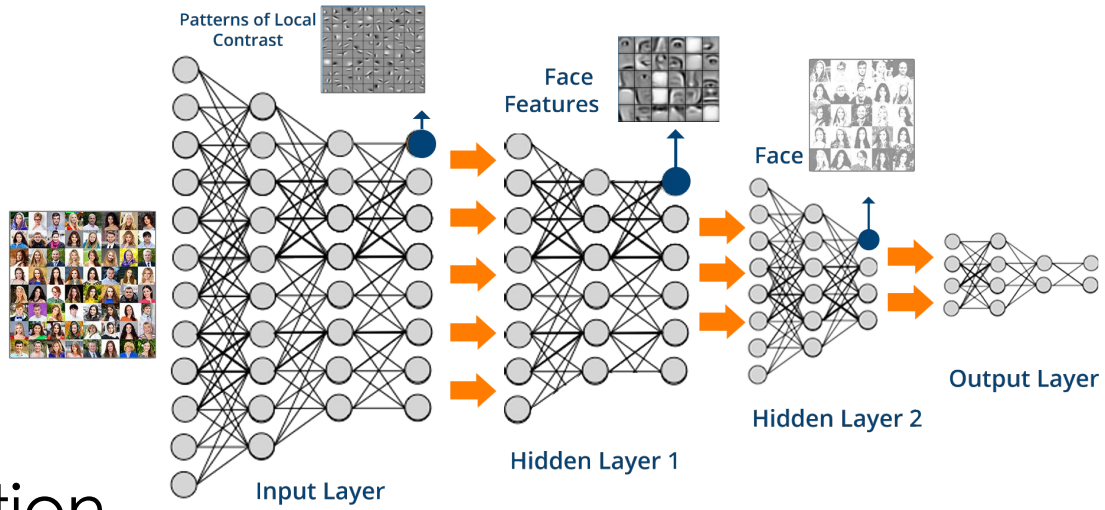


Fig. 1: Graphical Illustration of bias-variance trade-off , Source: Scott Fortmann-Roe., Understanding Bias-Variance Trade-off

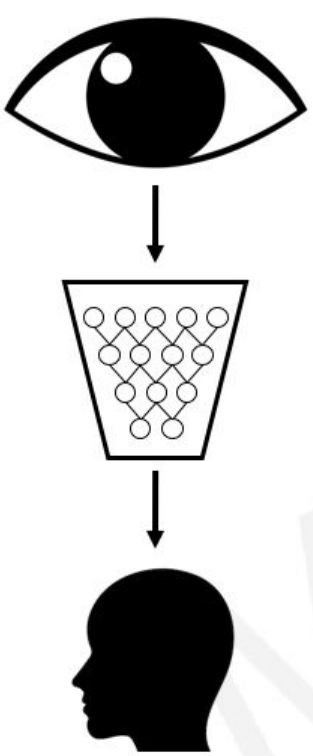

Week 4:
Introduction to Convolution Neural Network
PCLUB CS2020
Yeshi Dorjee
Feb 16, 2020
<p> club

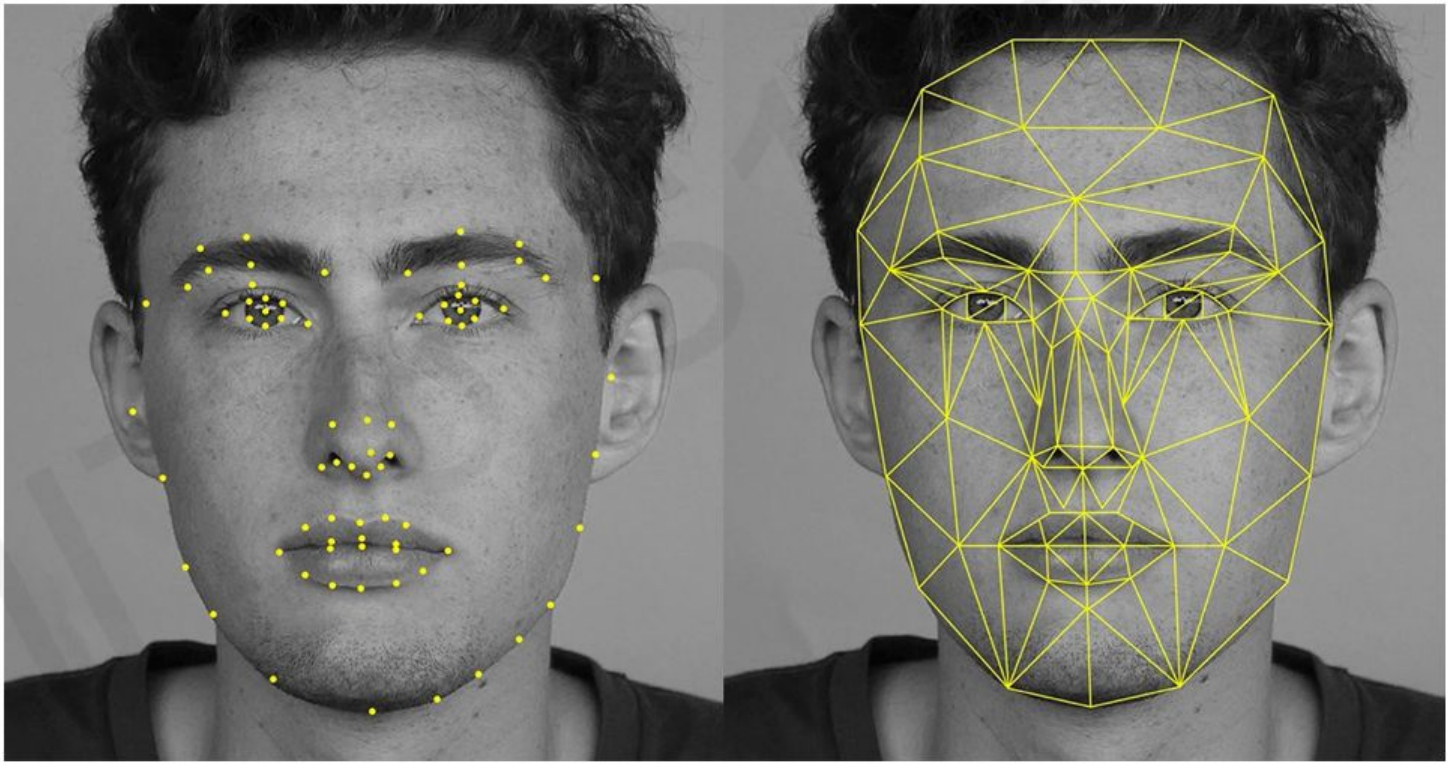
Overview

- Introduction
- Convolution Layer
- Pooling
- Image Augmentation
- Classical Network(Case Study)
- ResNets

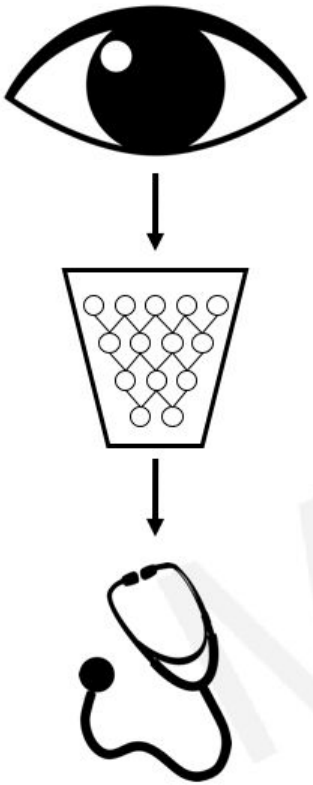


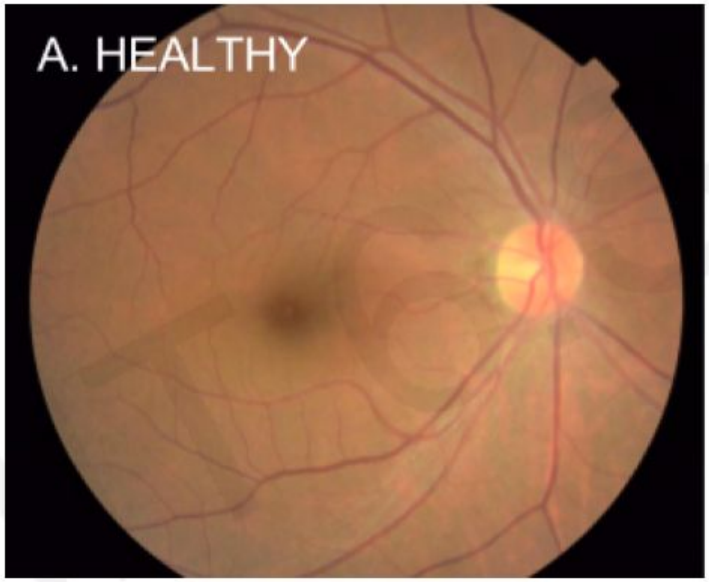
Impact: Facial Detection & Recognition

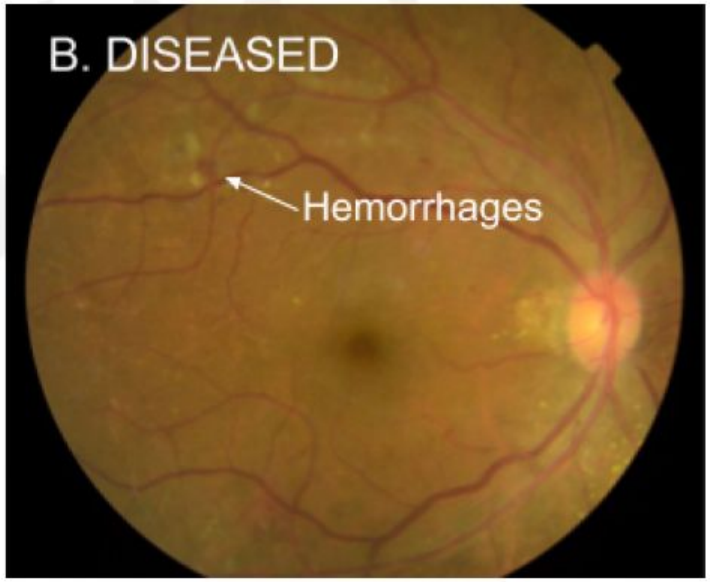




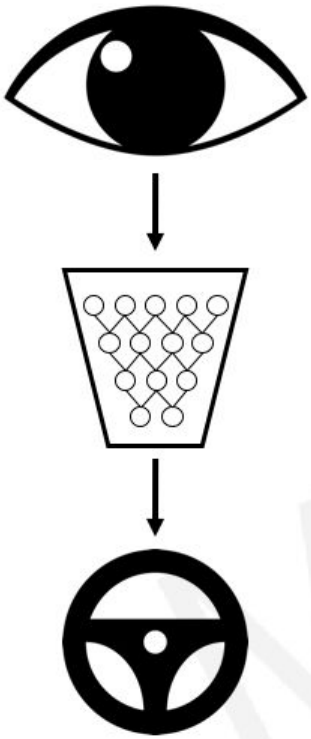
Impact: Medicine, Biology, Healthcare

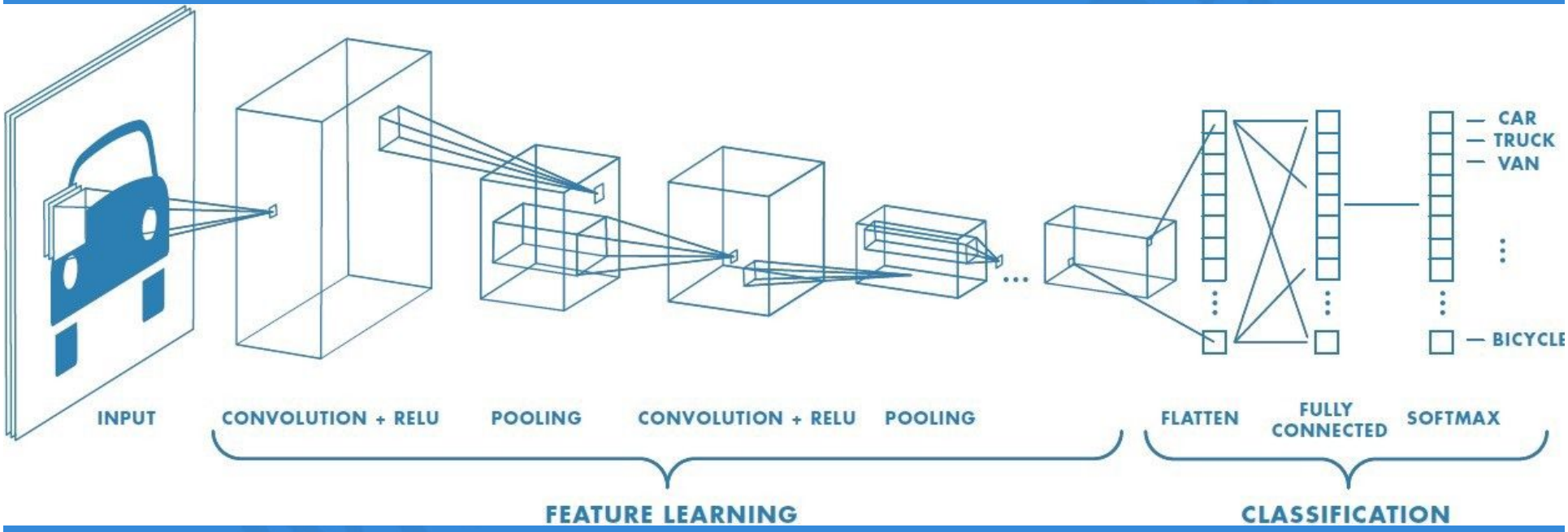




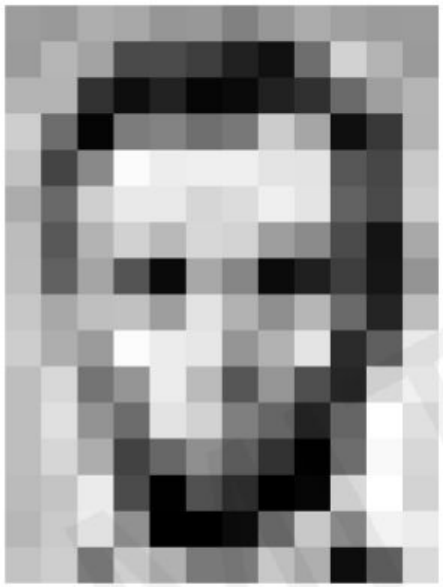


Impact: Self-Driving Cars

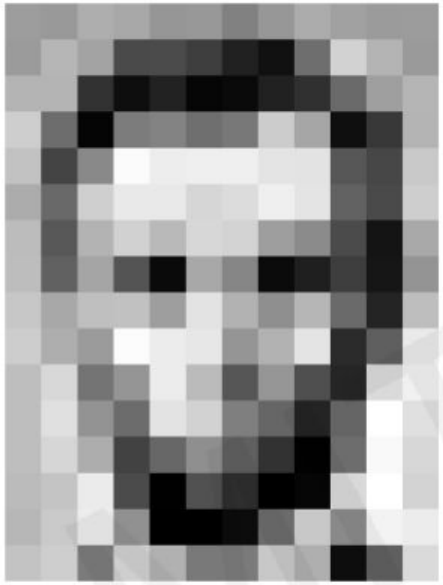



CAR
TRUCK
VAN
BICYCLE
INPUT
CONVOLUTION + RELU
POOLING
CONVOLUTION + RELU
POOLING
FLATTEN
FULLY CONNECTED
SOFTMAX
FEATURE LEARNING
CLASSIFICATION

Images are Numbers

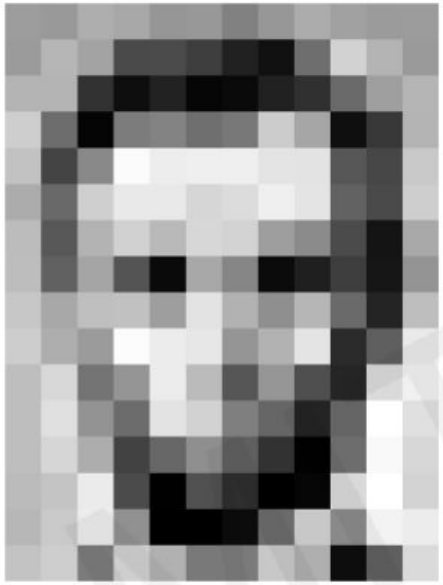


Images are Numbers



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	85	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	95	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

Images are Numbers



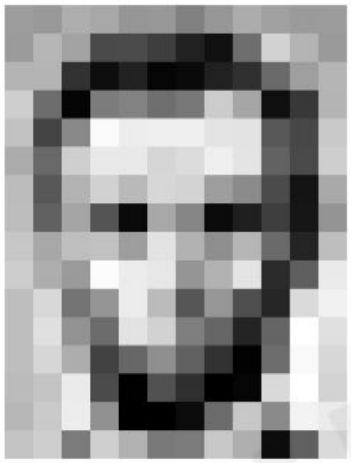
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

What the computer sees

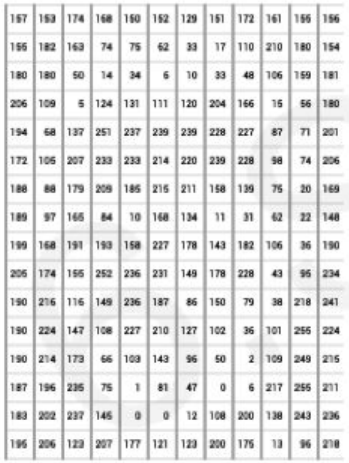
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

An image is just a matrix of numbers [0,255]!
i.e., 1080x1080x3 for an RGB image

Tasks in Computer Vision

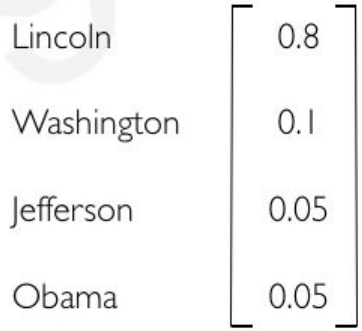


Input Image



Pixel Representation

classification



- **Regression**: output variable takes continuous value
- **Classification**: output variable takes class label. Can produce probability of belonging to a particular class

High Level Feature Detection

Let's identify key features in each image category

Nose,
Eyes,
Mouth

Wheels,
License Plate,
Headlights

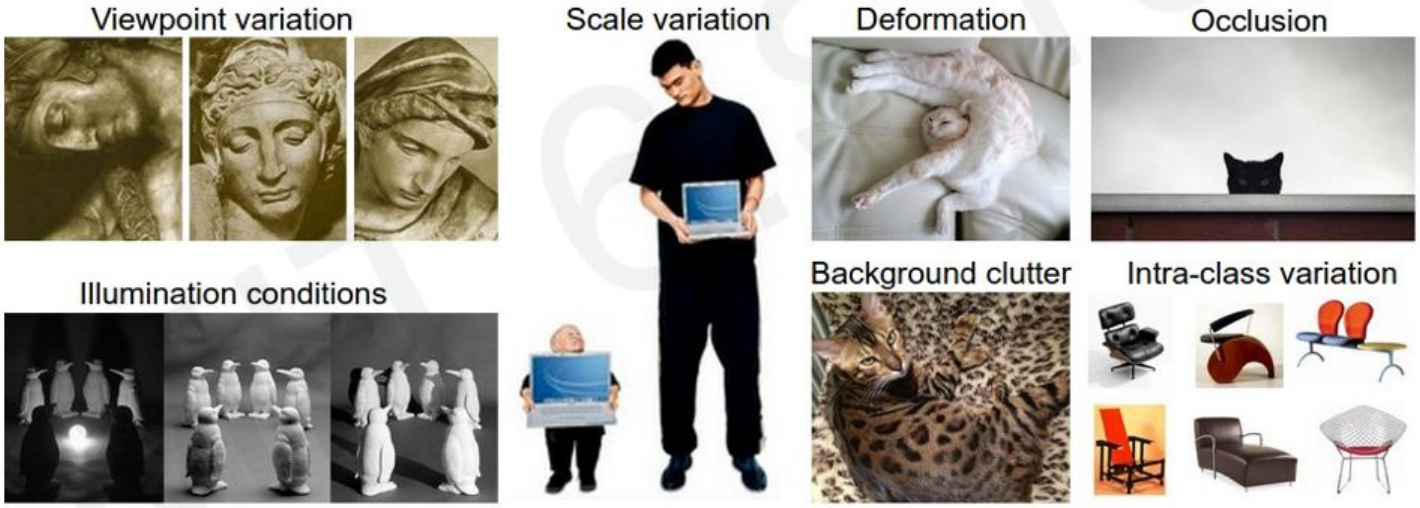
Door,
Windows,
Steps

Manual Feature Extraction

Problems?

Manual Feature Extraction

Domain knowledge → Define features → Detect features to classify



Manual Feature Extraction

Domain knowledge → Define features → Detect features to classify

Viewpoint variation

Illumination conditions

Scale variation

Deformation

Occlusion

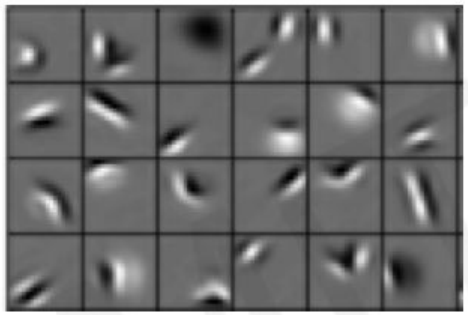
Background clutter

Intra-class variation

Learning Feature Representations

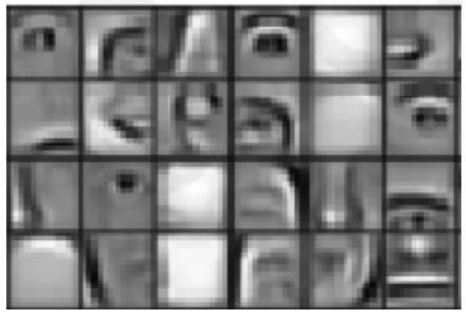
Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features



Edges, dark spots

Mid level features



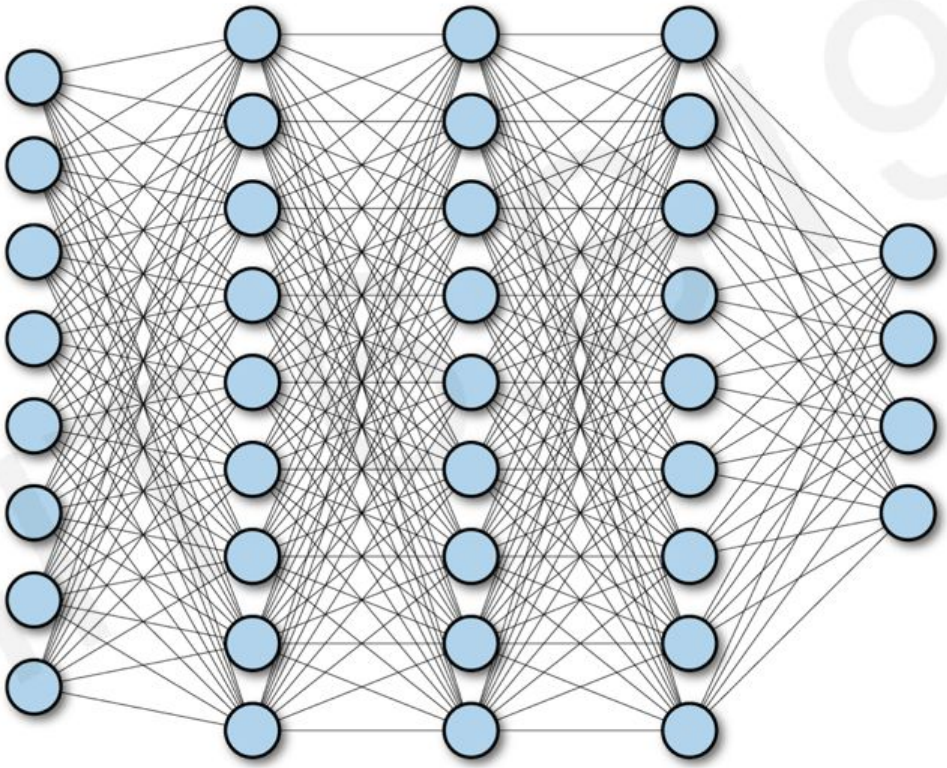
Eyes, ears, nose

High level features

Facial structure

Learning Visual Features

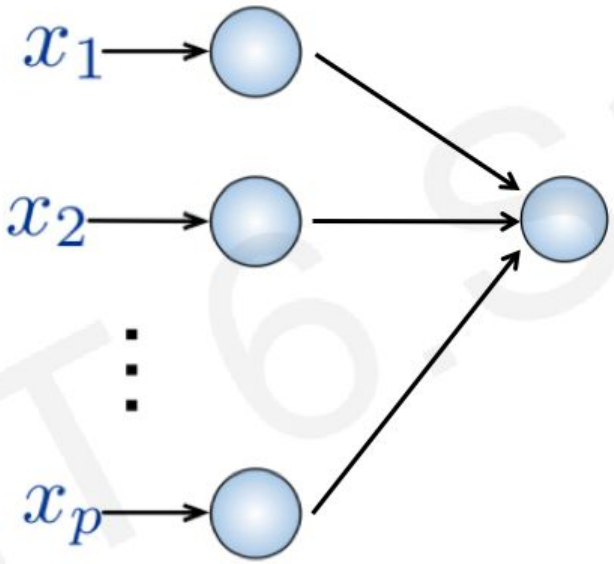
Fully Connected Neural Network



Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values



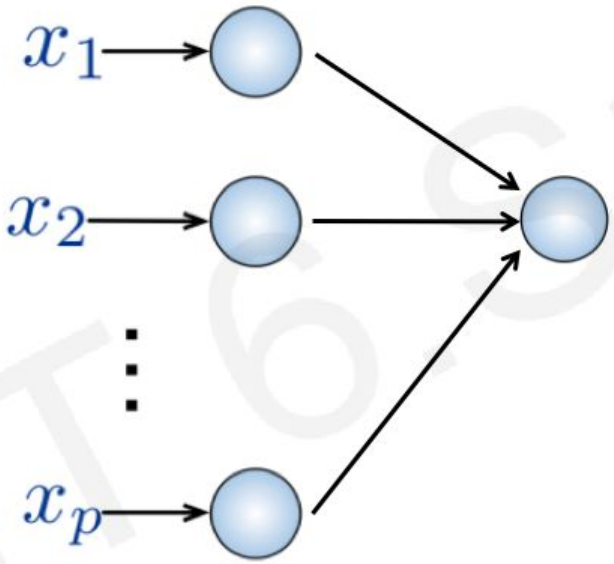
Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values



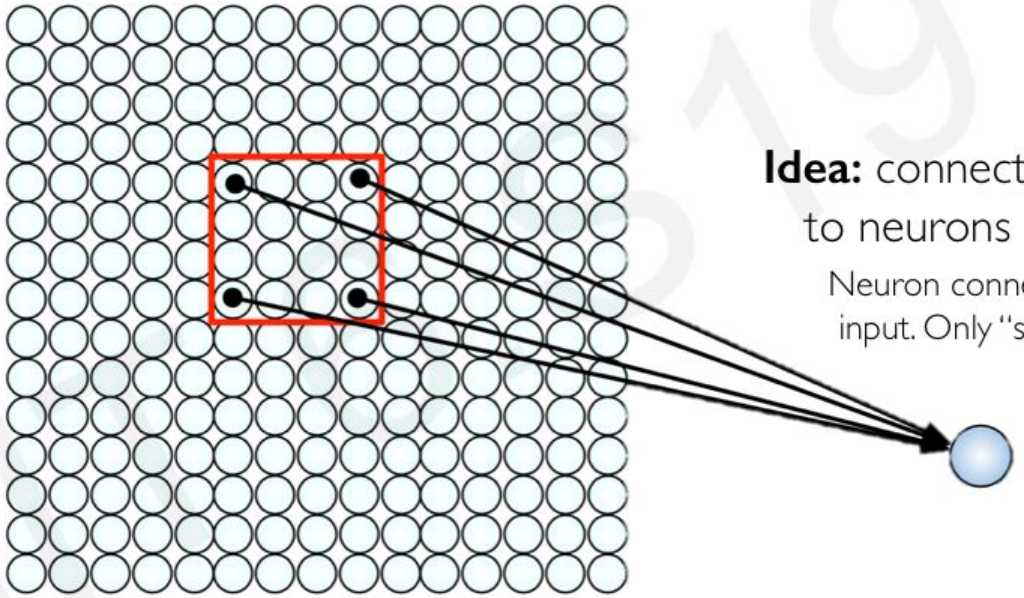
Fully Connected:

- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

How can we use **spatial structure** in the input to inform the architecture of the network?

Using Spatial Structure

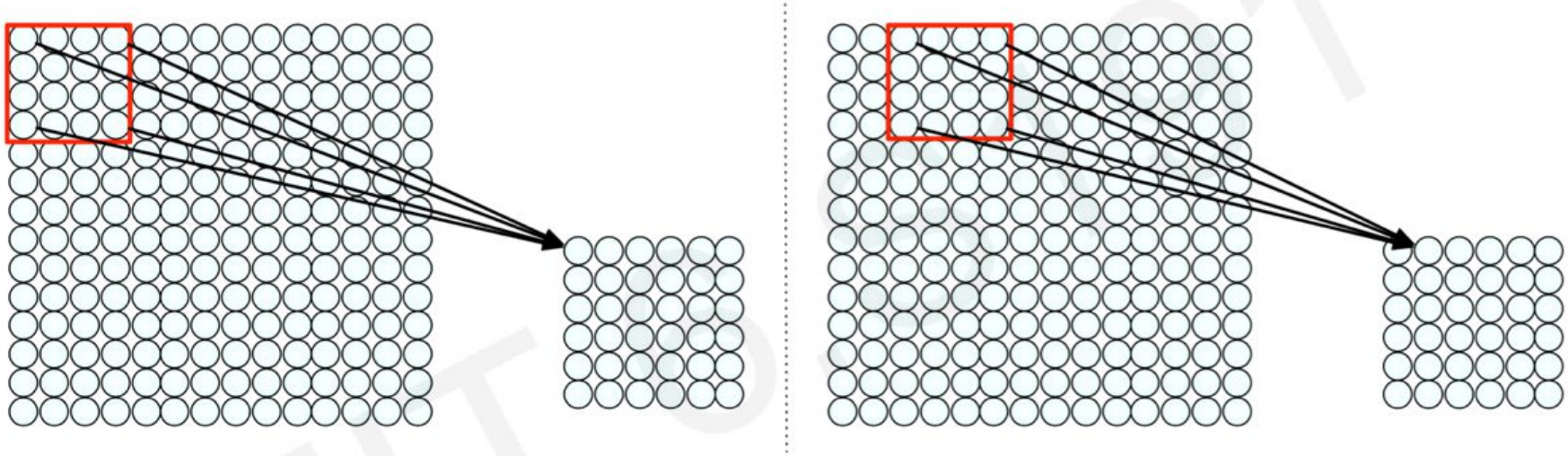
Input: 2D image.
Array of pixel values



Idea: connect patches of input to neurons in hidden layer.

Neuron connected to region of input. Only "sees" these values.

Using Spatial Structure



Connect patch in input layer to a single neuron in subsequent layer.
Use a sliding window to define connections.
How can we **weight** *the patch to detect particular features?*

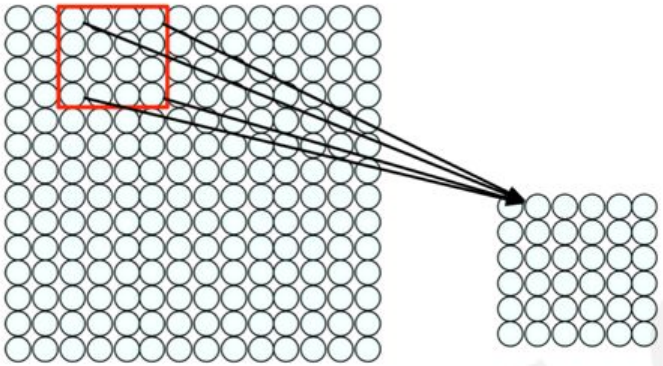
Applying Filters to Extract Features

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) Spatially **share** parameters of each filter
(features that matter in one part of the input should matter elsewhere)

Feature Extraction with Convolution



- Filter of size 4x4 : 16 different weights
- Apply this same filter to 4x4 patches in input
- Shift by 2 pixels for next patch

This "patchy" operation is **convolution**

1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

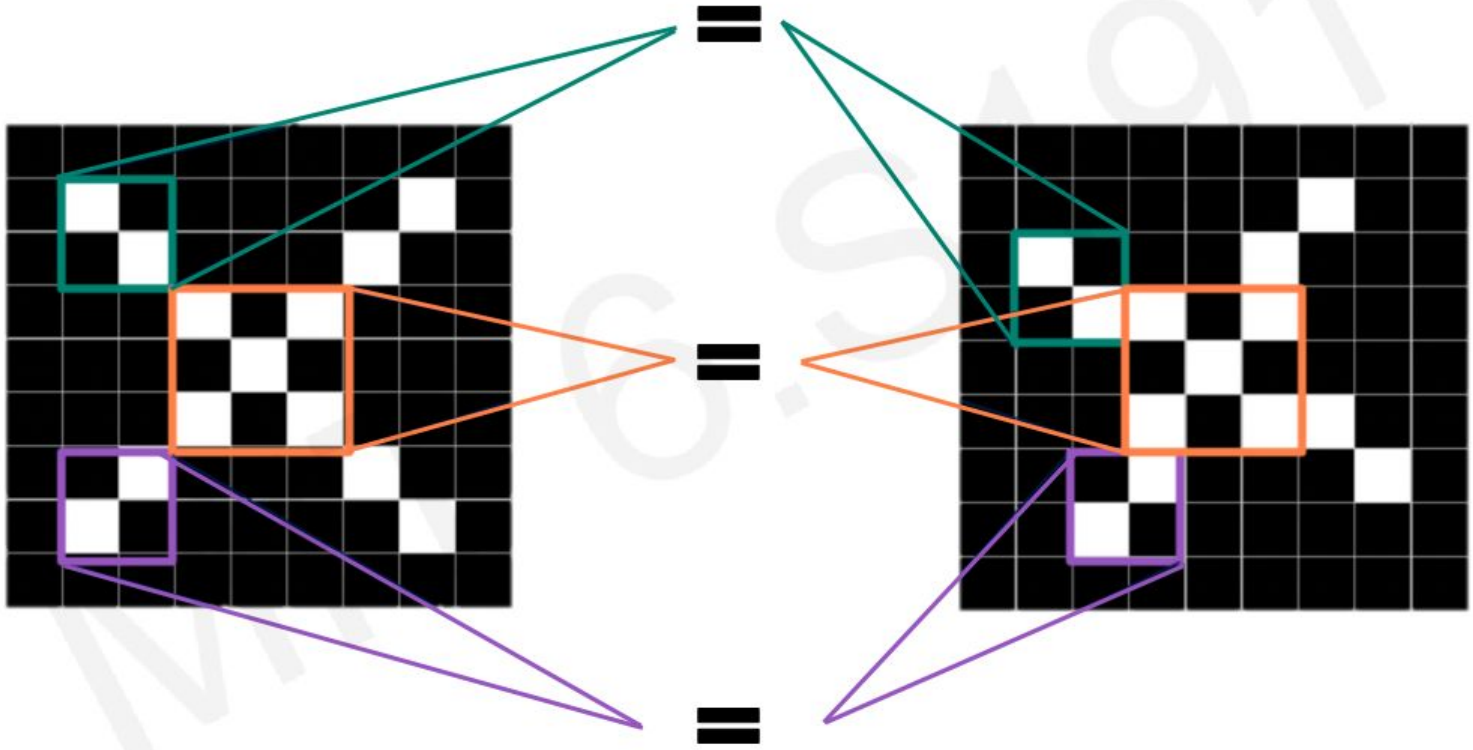
Feature Extraction and Convolution

A Case Study

X or X?

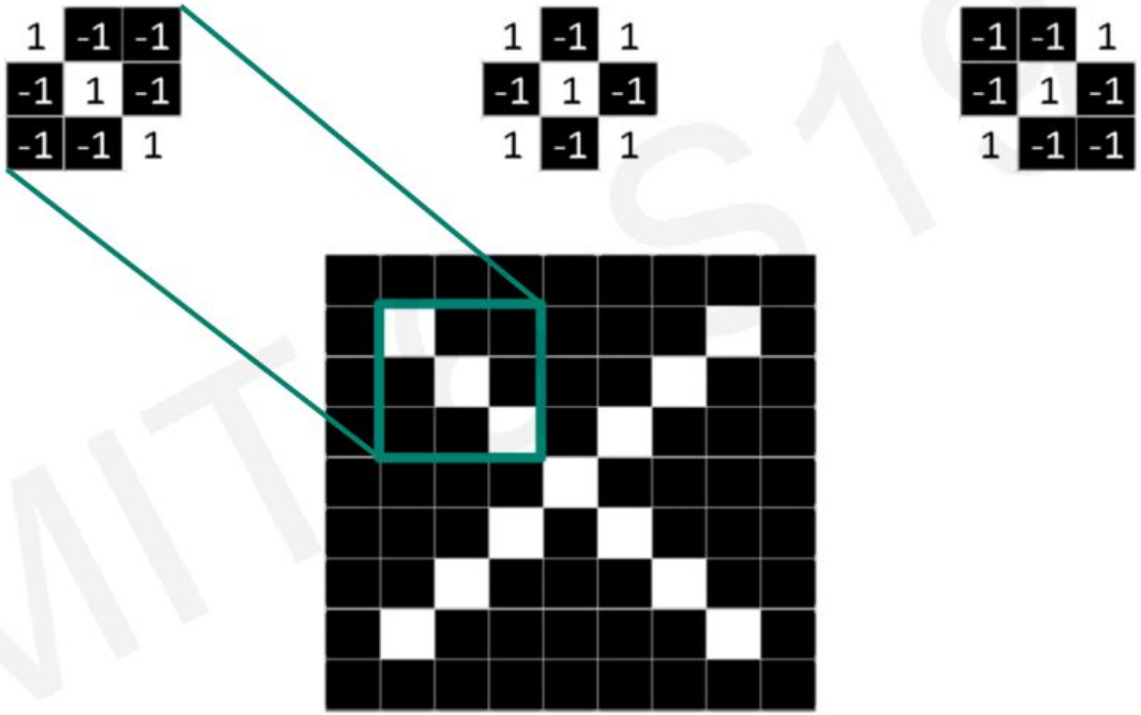
Image is represented as matrix of pixel values… and computers are literal!
We want to be able to classify an X as an X even if it's shifted, shrunk, rotated, deformed.

Features of X

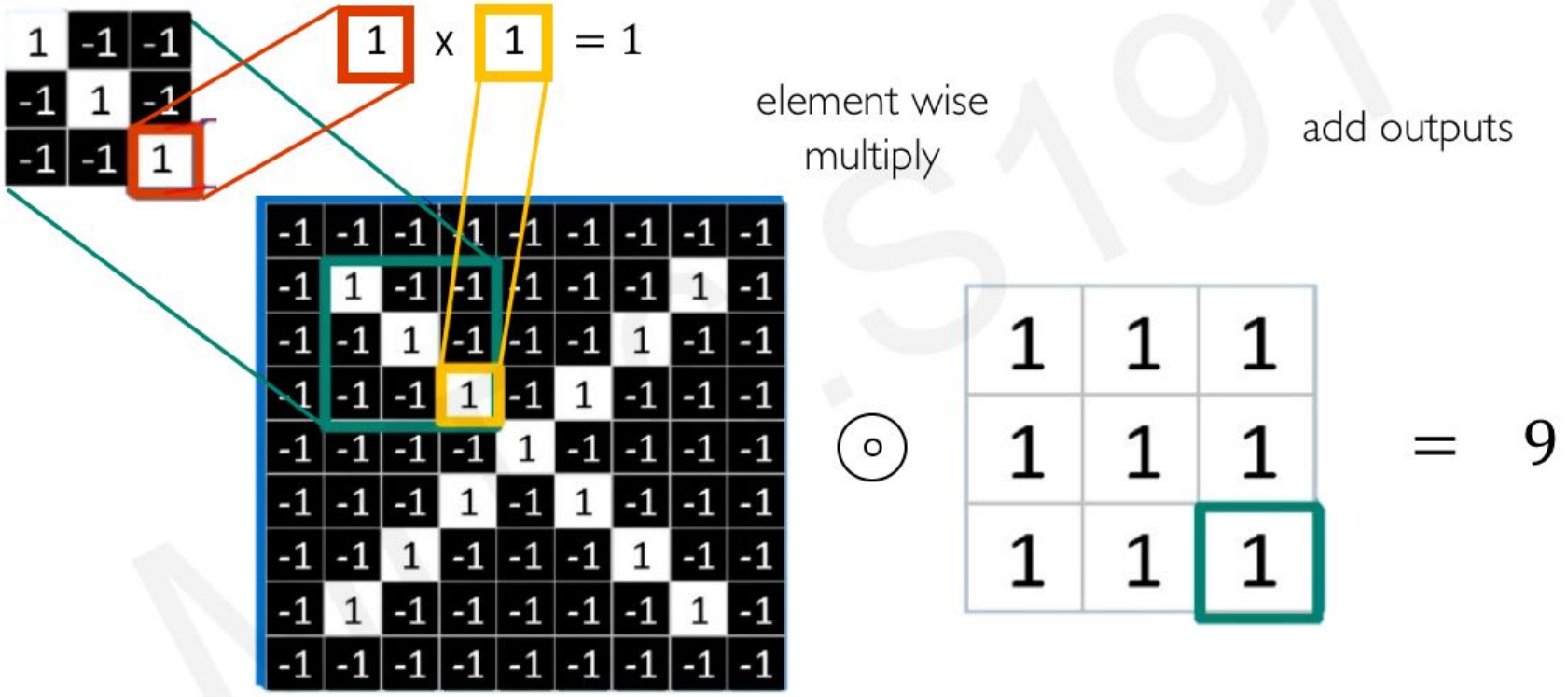


Filters to Detect X Features

filters

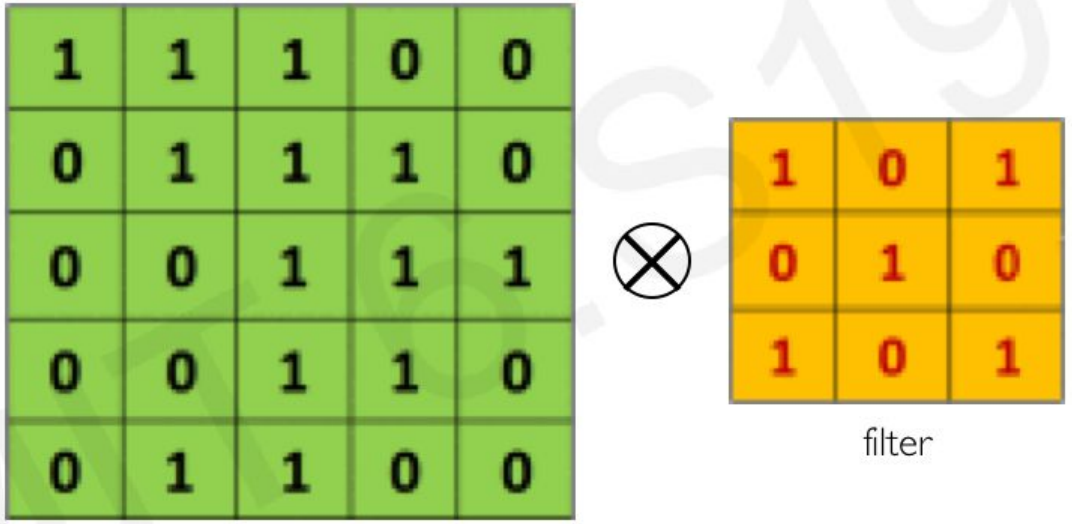


The Convolution Operation



The Convolution Operation

Suppose we want to compute the convolution of a 5x5 image and a 3x3 filter:



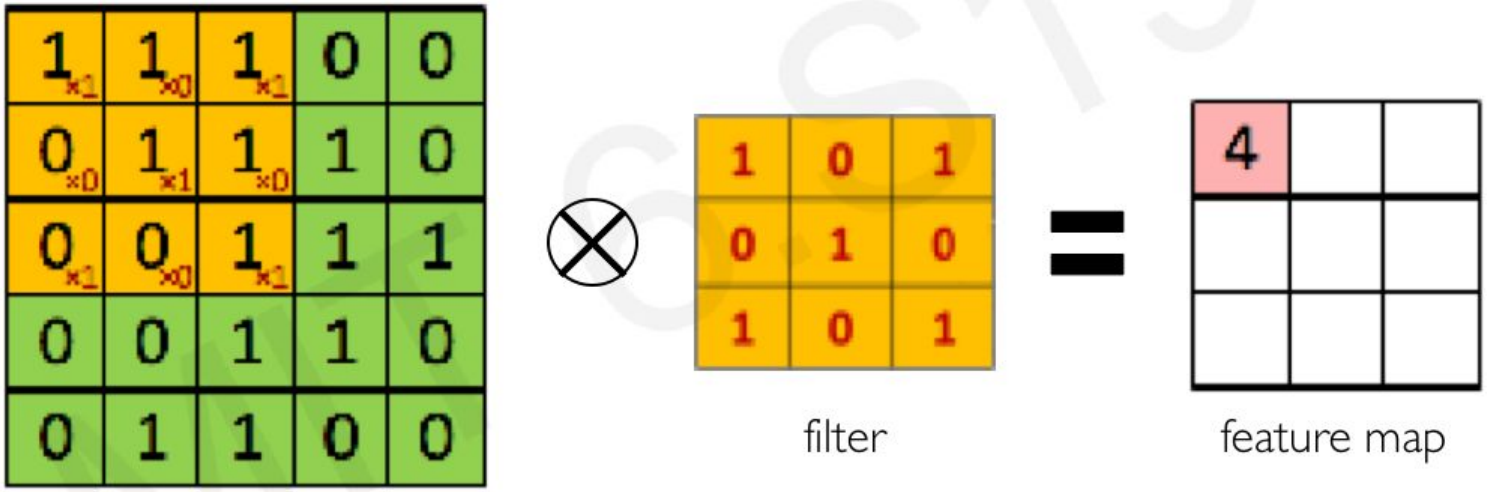
image

filter

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs…

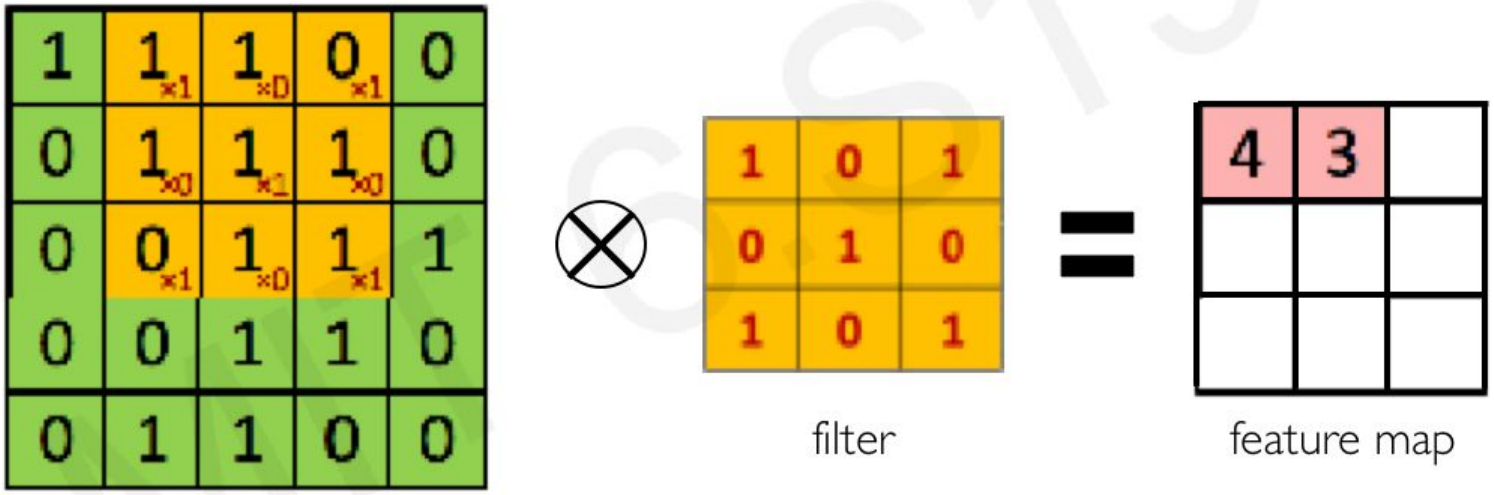
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



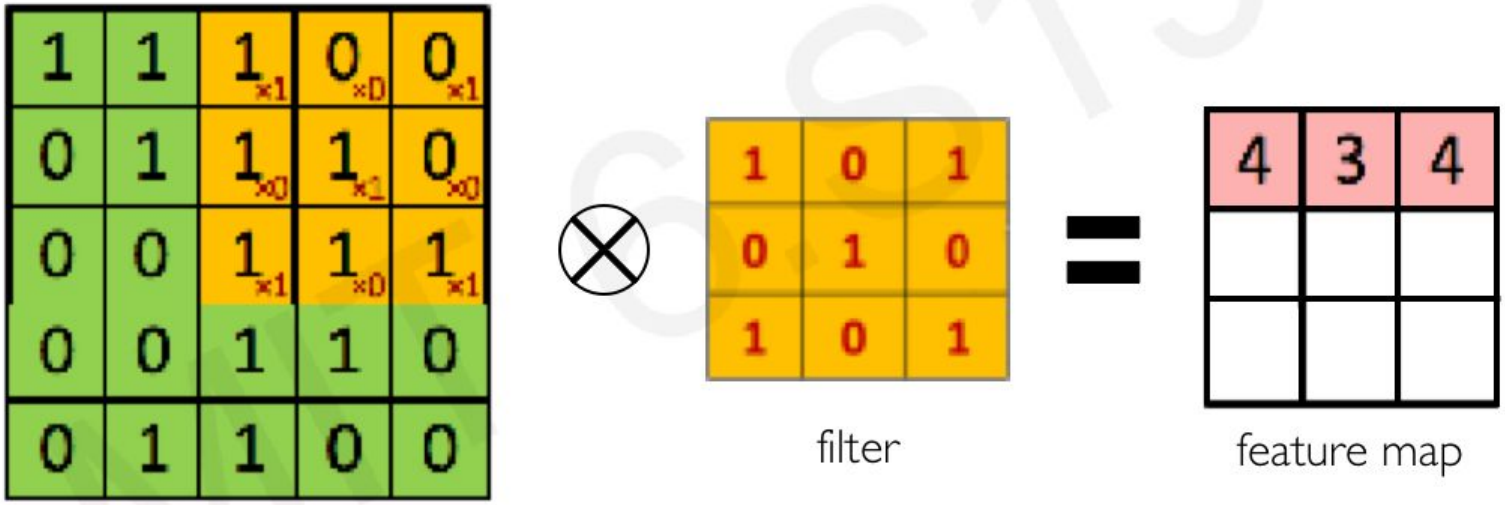
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



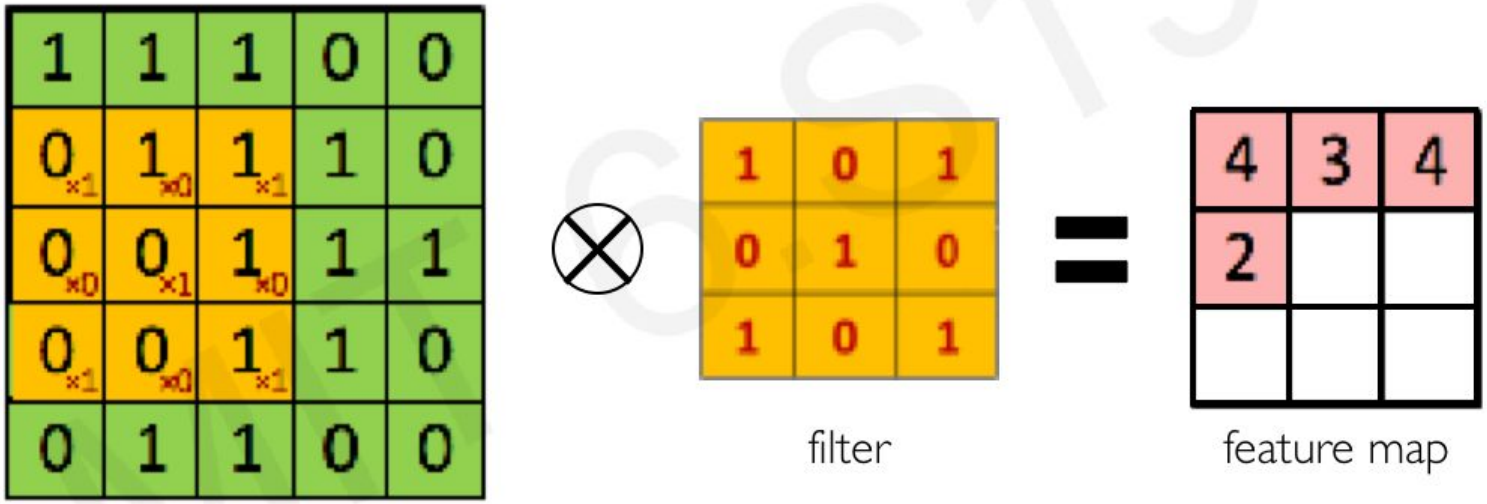
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



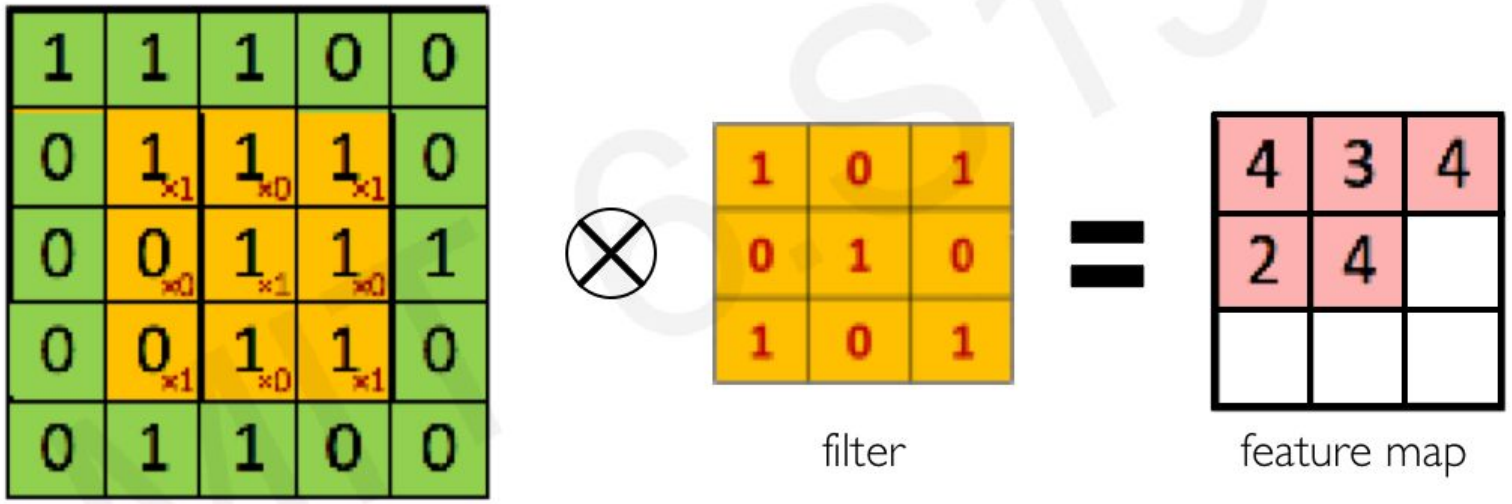
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



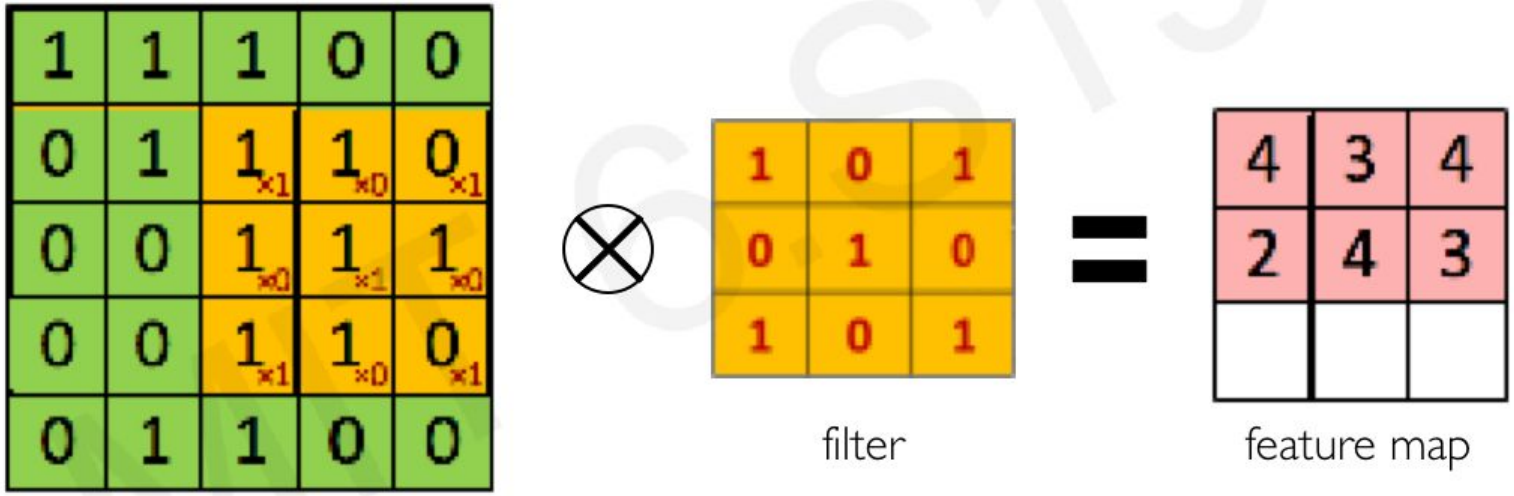
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



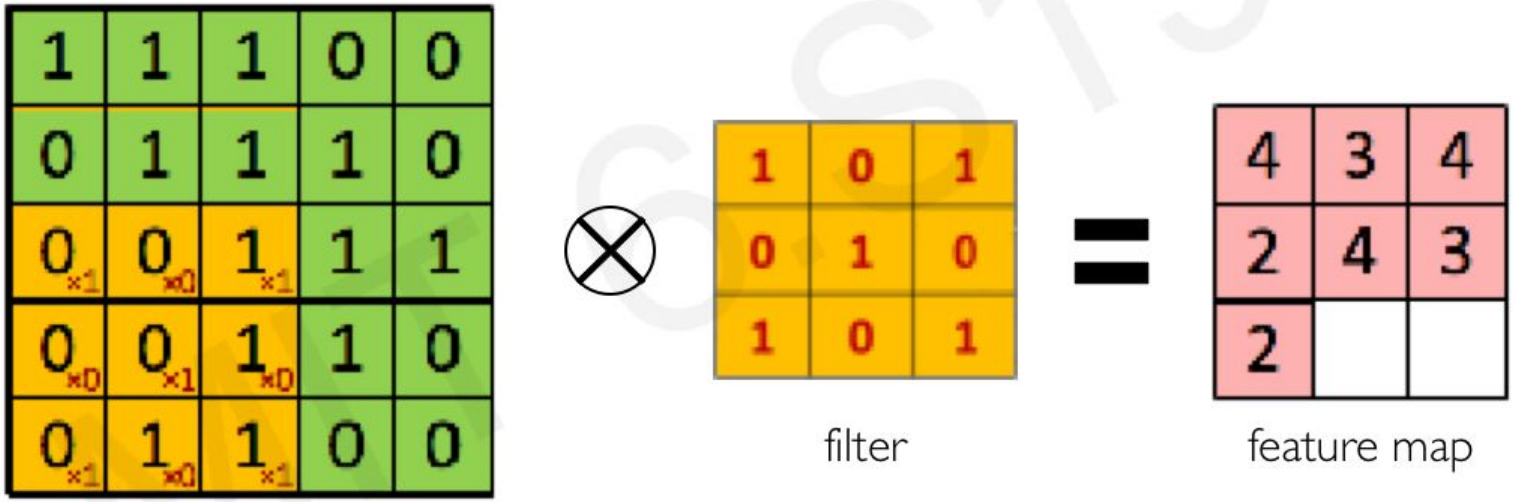
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



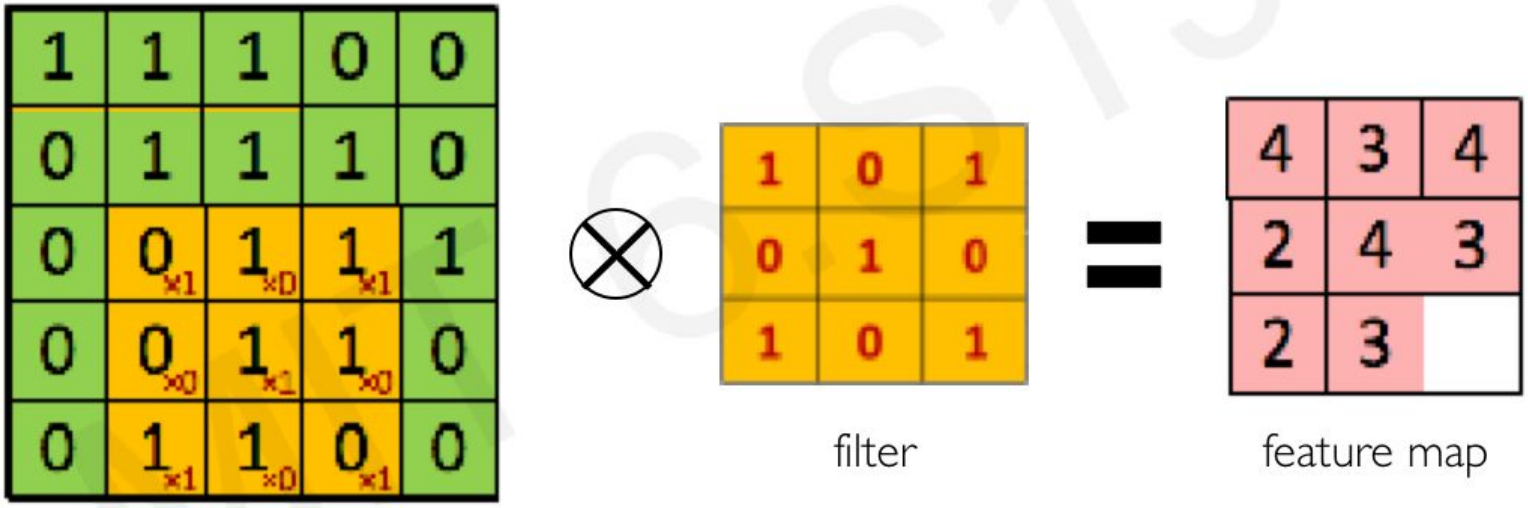
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



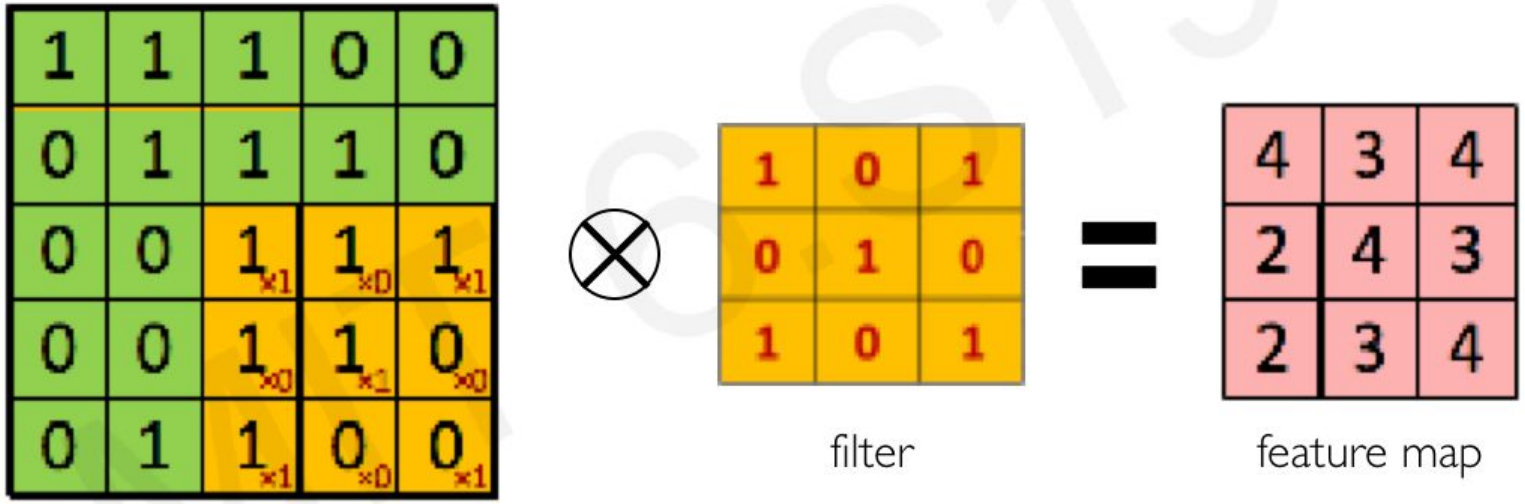
The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



The Convolution Operation

We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



Producing Feature Maps

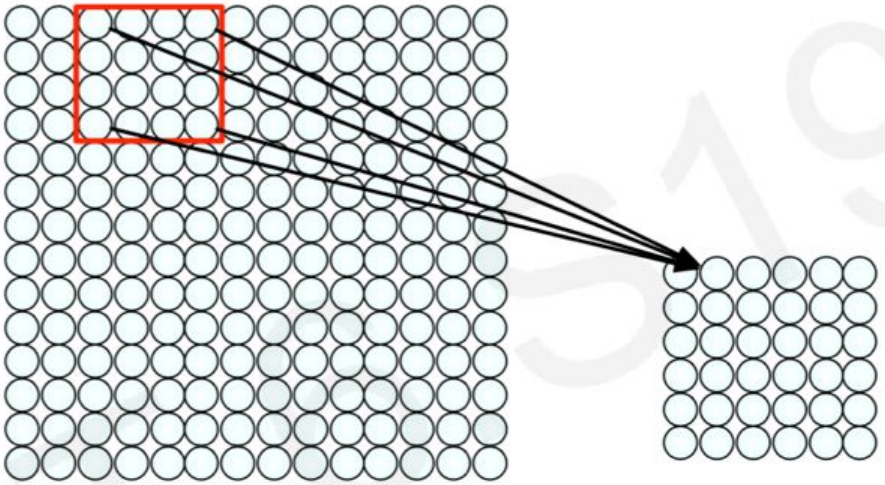
Original

Sharpen

Edge Detect

"Strong" Edge Detect

Feature Extraction with Convolution



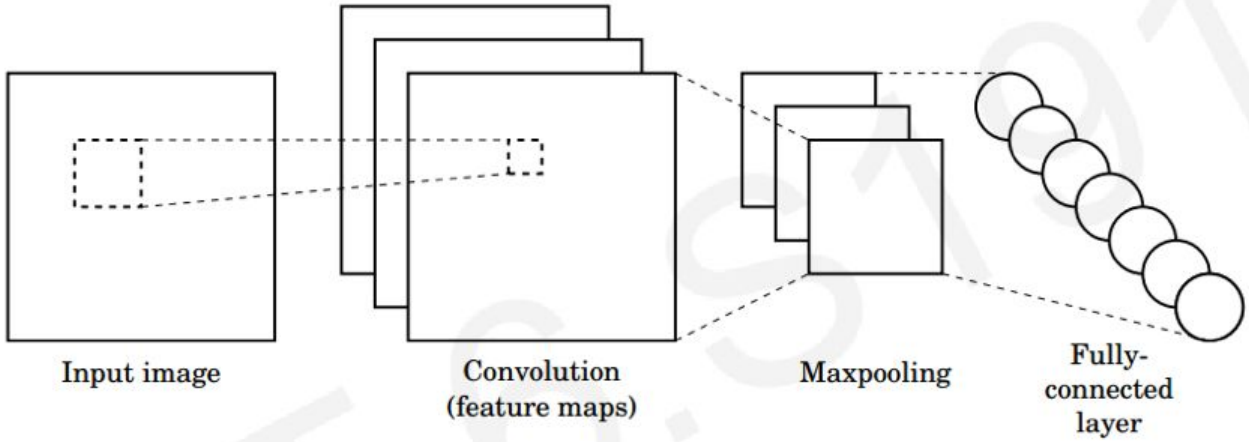
1) Apply a set of weights – a filter – to extract **local features**

2) Use **multiple filters** to extract different features

3) **Spatially share** parameters of each filter

Convolutional Neural Networks (CNNs)

CNNs for Classification



1. **Convolution**: Apply filters to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

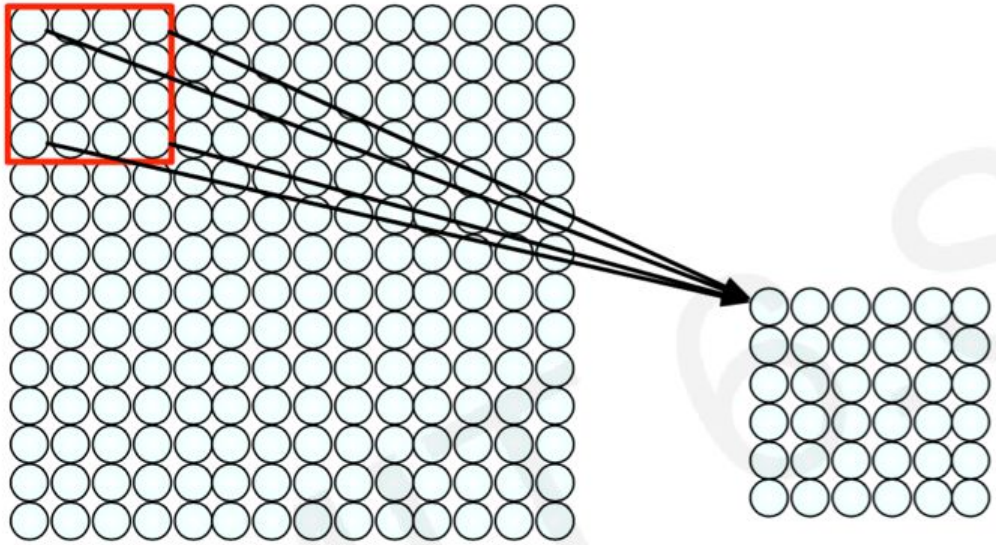
`tf.keras.layers.Conv2D`

`tf.keras.activations.*`

`tf.keras.layers.MaxPool2D`

Train model with image data.
Learn weights of filters in convolutional layers.

Convolutional Layers: Local Connectivity

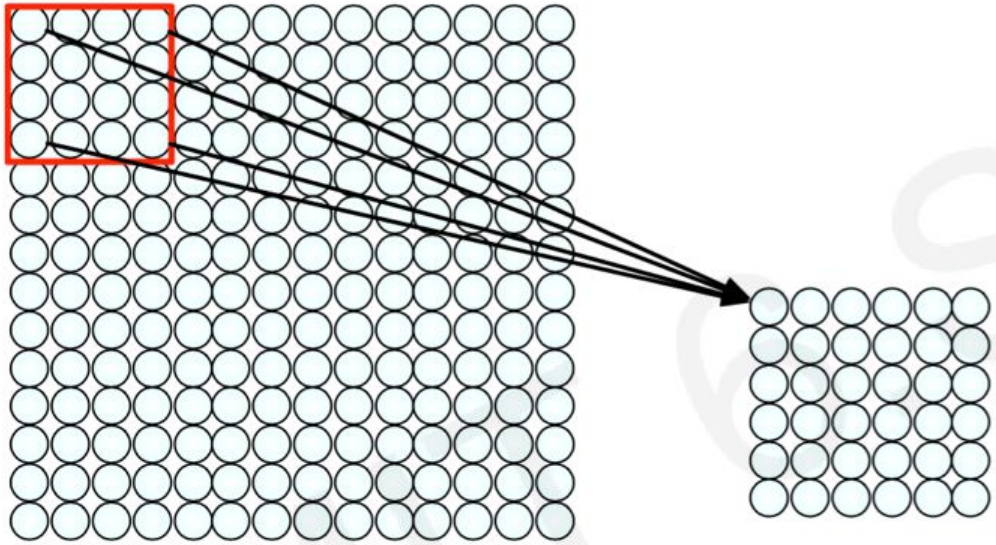


`tf.keras.layers.Conv2D`

For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

Convolutional Layers: Local Connectivity



`tf.keras.layers.Conv2D`

For a neuron in hidden layer:

- Take inputs from patch
- Compute weighted sum
- Apply bias

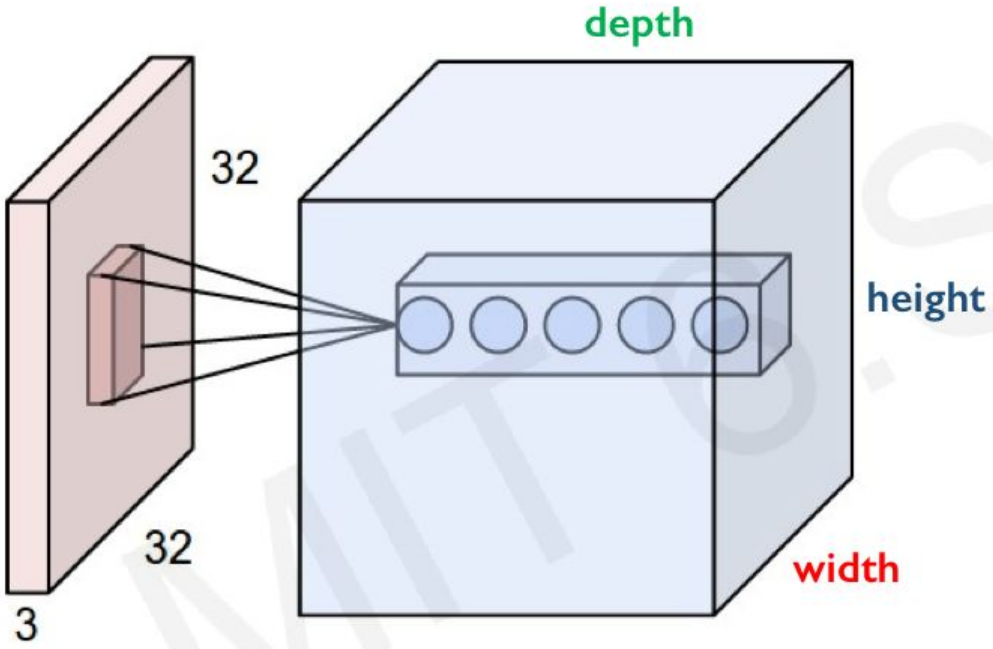
4x4 filter: matrix of weights w_{ij}

$$\sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij}\, x_{i+p,j+q} + b$$

for neuron (p,q) in hidden layer

1) applying a window of weights
2) computing linear combinations
3) activating with non-linear function

CNNs: Spatial Arrangement of Output Volume



Layer Dimensions:

$h \; x \; w \; x \; d$

where h and w are spatial dimensions
d (depth) = number of filters

Stride:

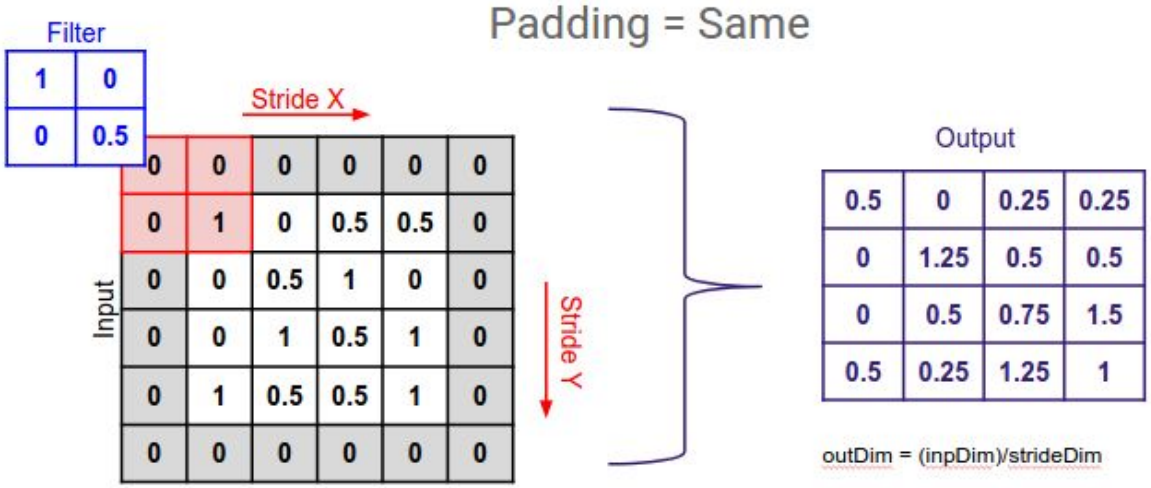
Filter step size

Receptive Field:

Locations in input image that a node is path connected to

```
tf.keras.layers.Conv2D( filters=d, kernel_size=(h,w), strides=s )
```

Padding



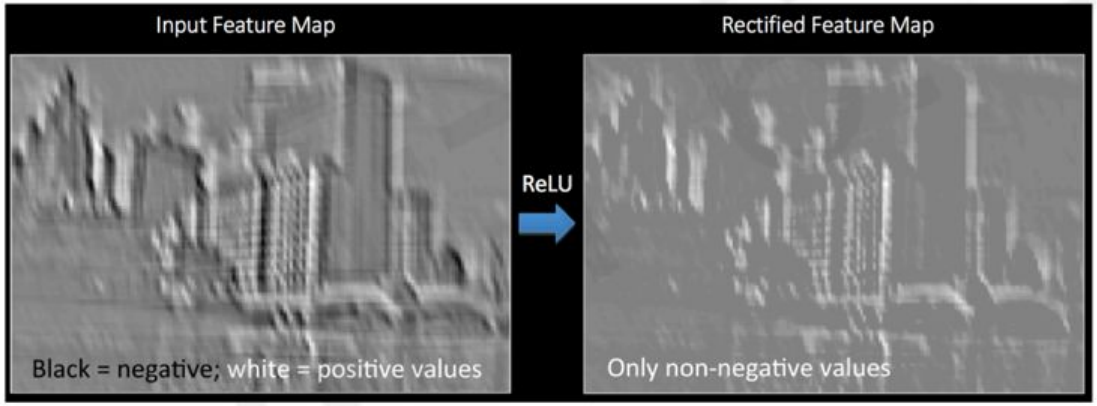
$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

n_{in}: number of input features
n_{out}: number of output features
k: convolution kernel size
p: convolution padding size
s: convolution stride size

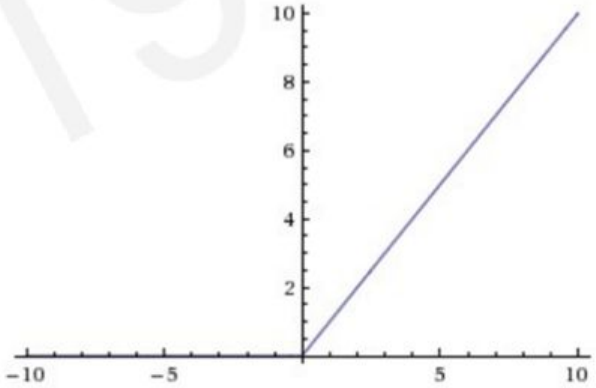
<p>club

Introducing Non-Linearity

- Apply after every convolution operation (i.e., after convolutional layers)
- ReLU: pixel-by-pixel operation that replaces all negative values by zero. **Non-linear operation!**



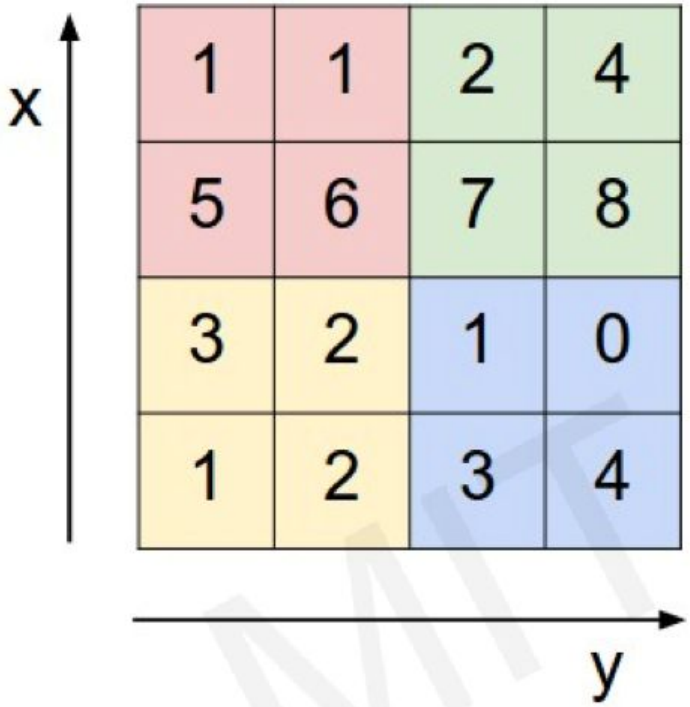
Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

`tf.keras.layers.ReLU`

Pooling



max pool with 2x2 filters and stride 2

```
tf.keras.layers.MaxPool2D(
    pool_size=(2,2),
    strides=2
)
```

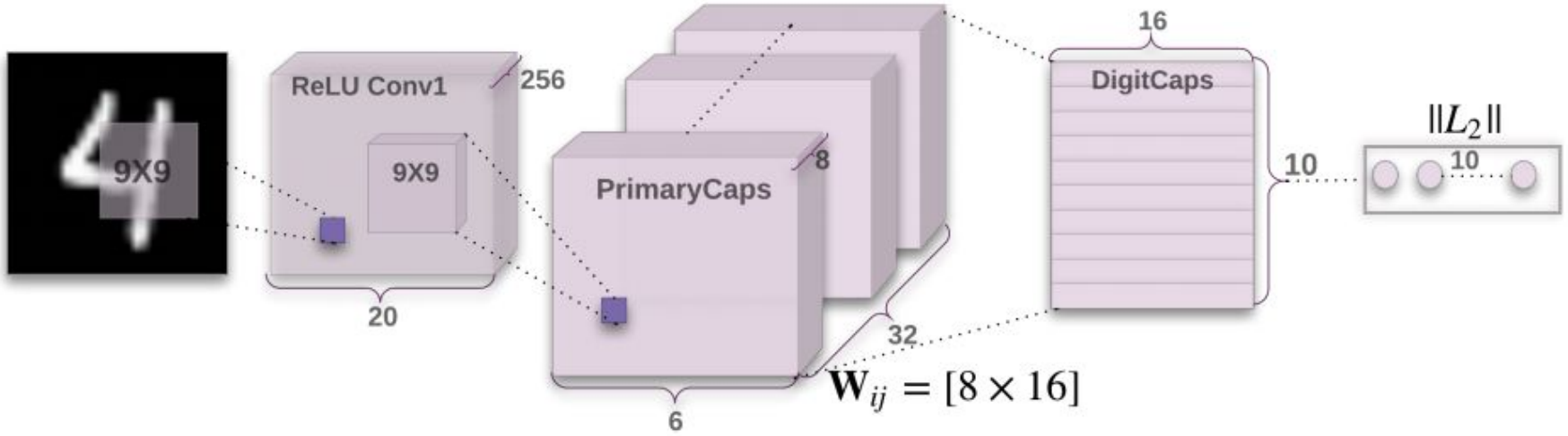
6	8
3	4

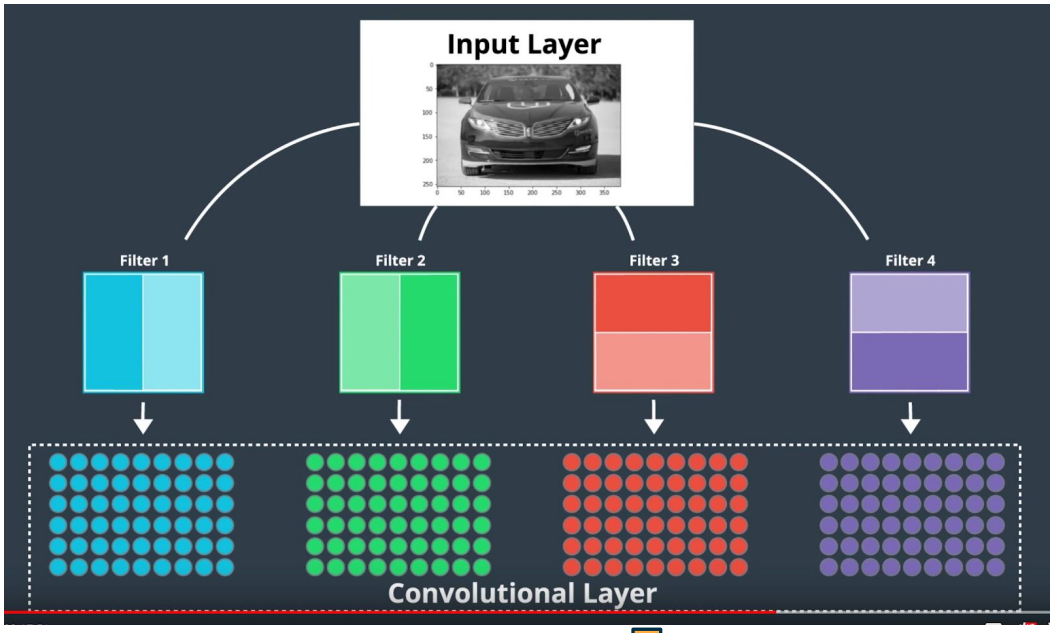
1) Reduced dimensionality
2) Spatial invariance

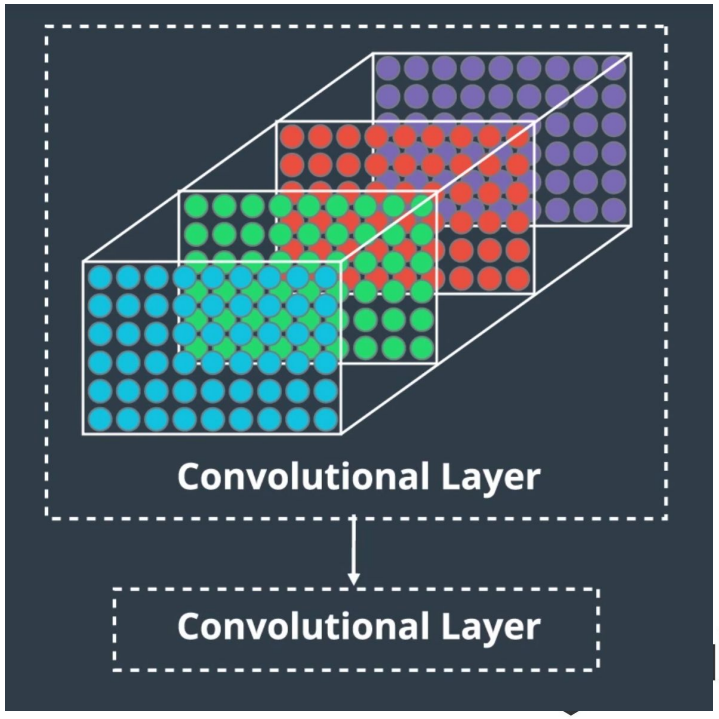
How else can we downsample and preserve spatial invariance?

Paper by
(Dynamic Routing Between Capsules)
https://arxiv.org/abs/1710.09829

The Capsule Network




Input Layer
Filter 1
Filter 2
Filter 3
Filter 4
Convolutional Layer


Convolutional Layer
Convolutional Layer

Layer *l* which is a convolution layer

filter size $= f^{[l]}$

padding $= p^{[l]}$

stride $= s^{[l]}$

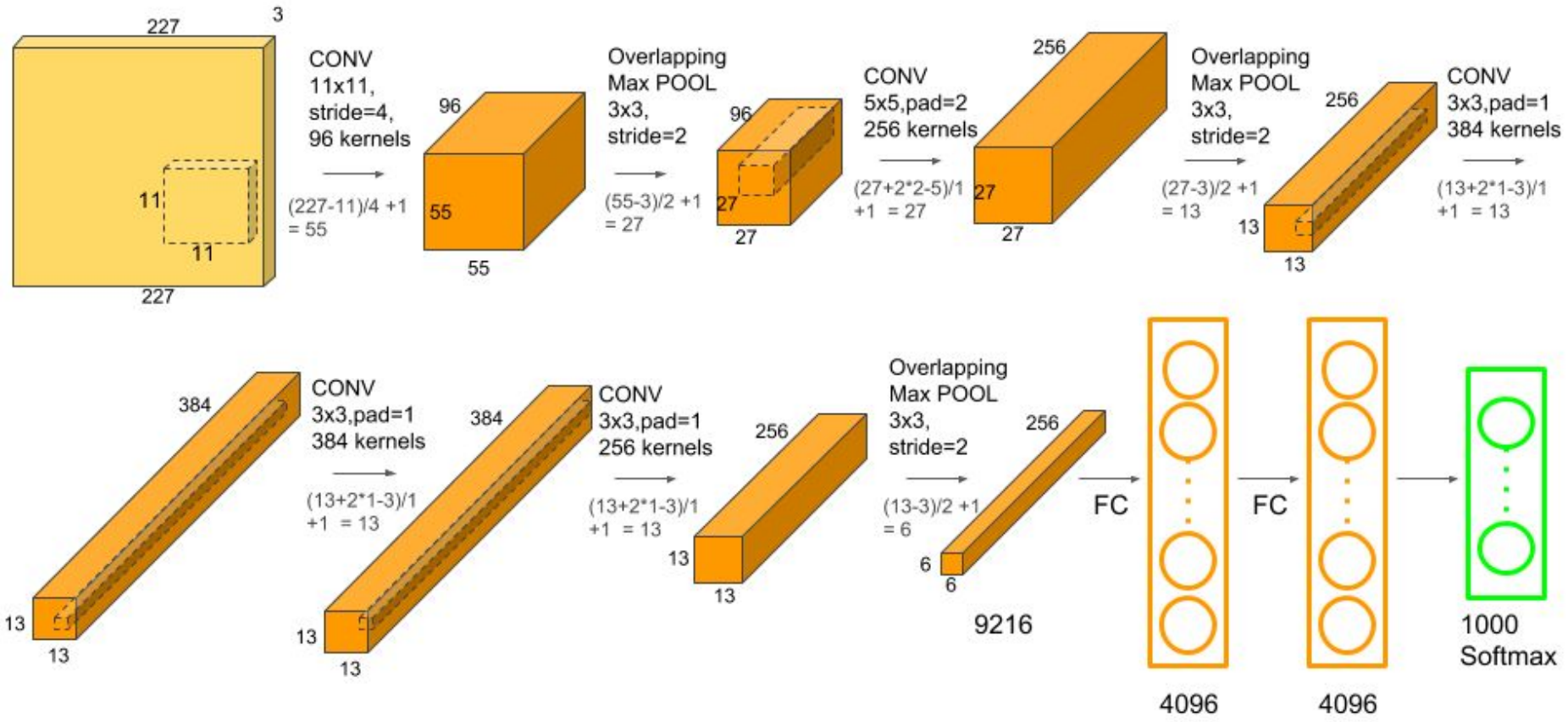
input $= n_h^{[l-1]} \times n_w^{[l-1]} \times n_c^{[l-1]}$

each filter $= f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$

weights $= f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$

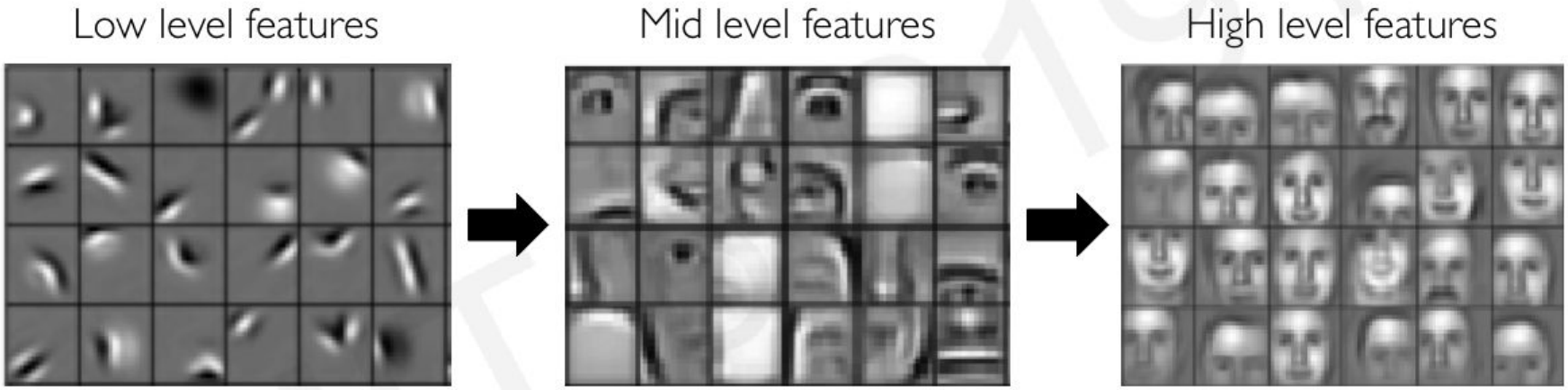
output $= n_h^{[l]} \times n_w^{[l]} \times n_c^{[l]}$

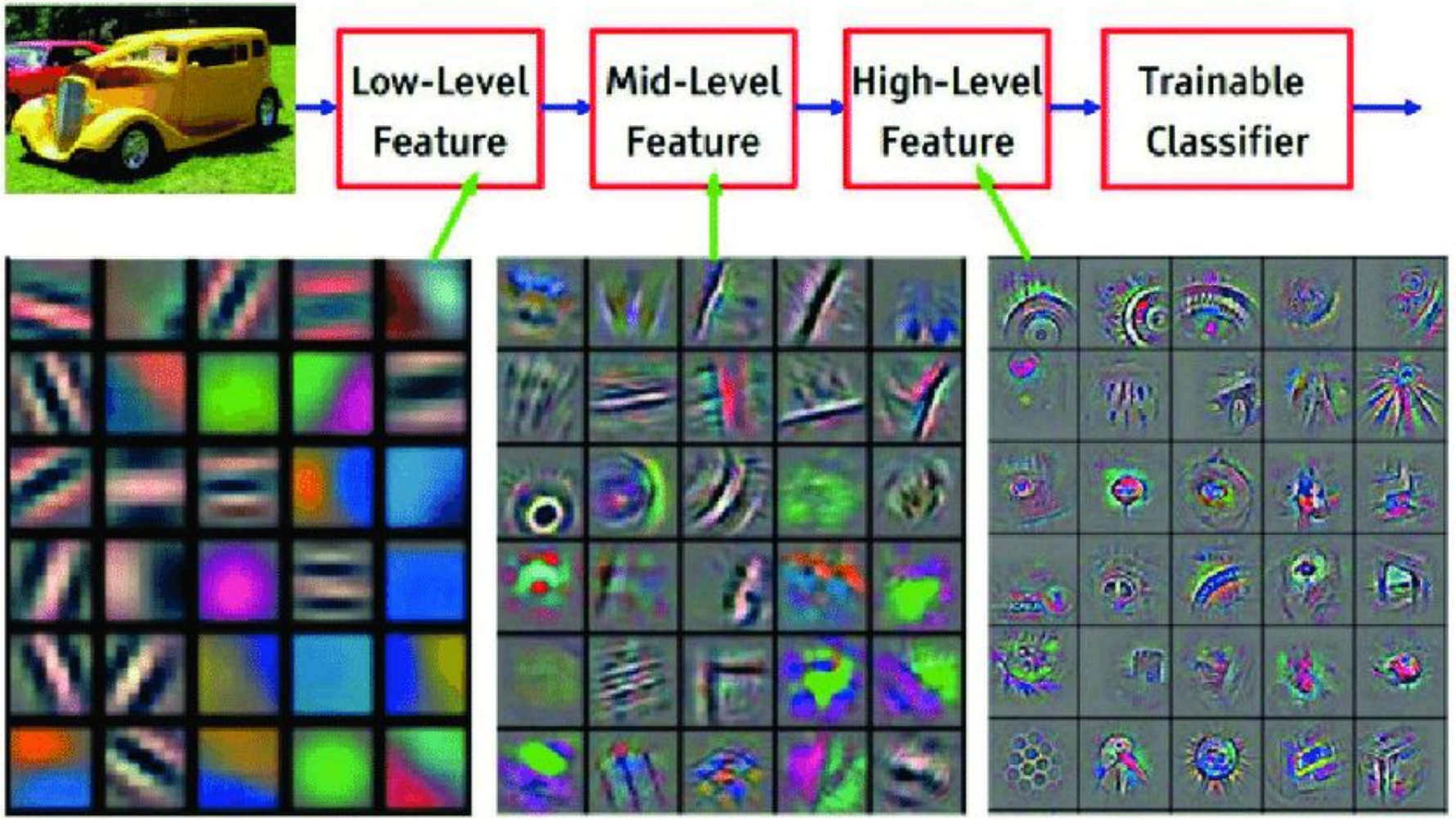
bias $= 1 \times 1 \times 1 \times n_c^{[l]}$



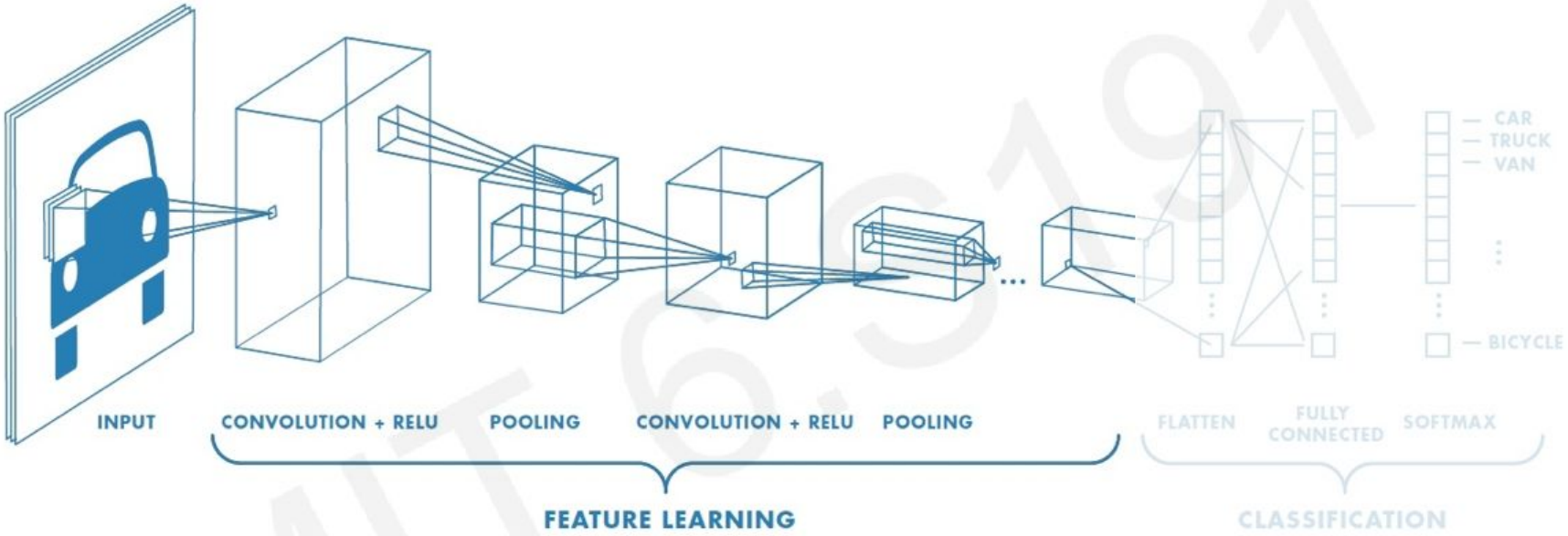
Height and Weight Decrease Where Channel

Representation Learning in Deep CNNs



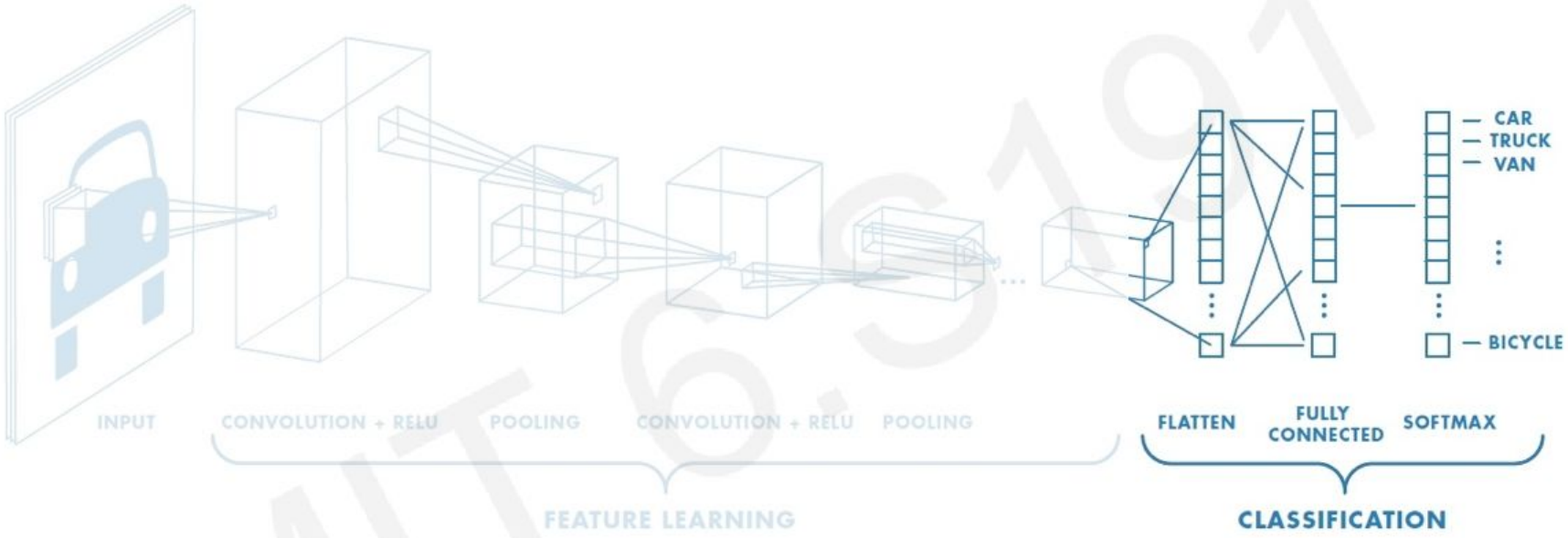

Low-Level Feature
Mid-Level Feature
High-Level Feature
Trainable Classifier

CNNs for Classification: Feature Learning



1. Learn features in input image through **convolution**
2. Introduce **non-linearity** through activation function (real-world data is non-linear!)
3. Reduce dimensionality and preserve spatial invariance with **pooling**

CNNs for Classification: Class Probabilities

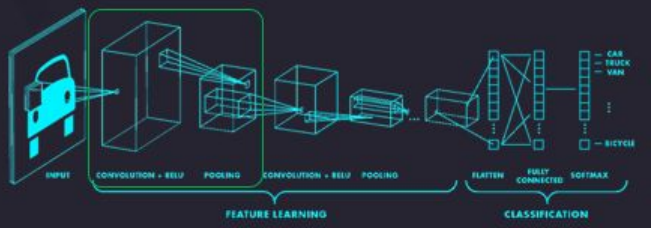


- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

$$\text{softmax}(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

Putting it all together

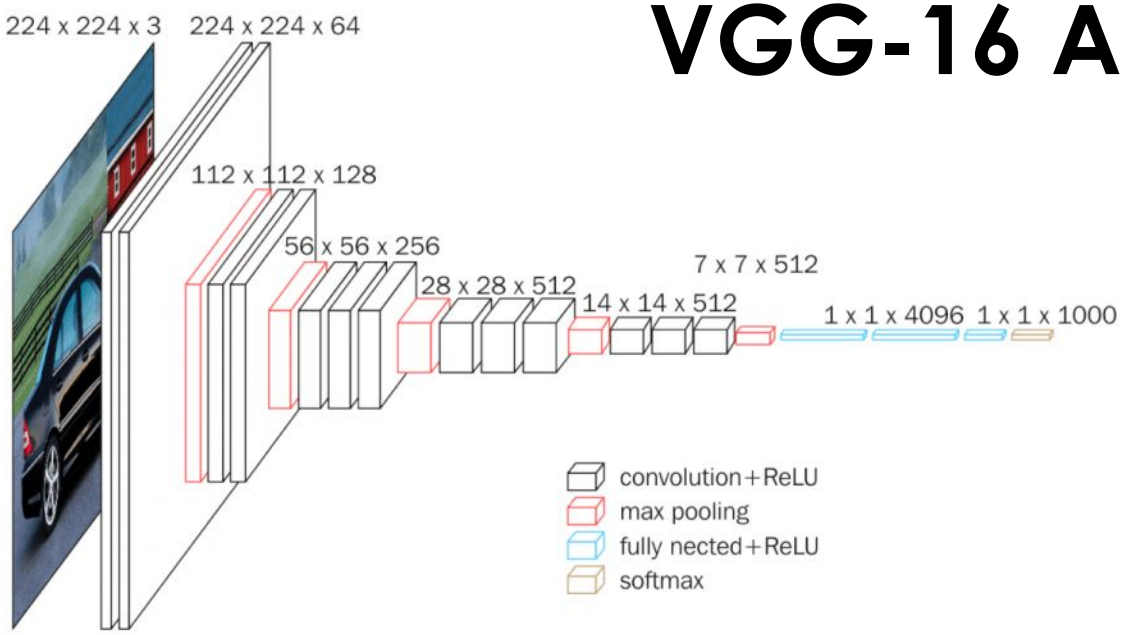
```
import tensorflow as tf

def generate_model():
    model = tf.keras.Sequential([
        # first convolutional layer
        tf.keras.layers.Conv2D(32, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # second convolutional layer
        tf.keras.layers.Conv2D(64, filter_size=3, activation='relu'),
        tf.keras.layers.MaxPool2D(pool_size=2, strides=2),

        # fully connected classifier
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(1024, activation='relu'),
        tf.keras.layers.Dense(10, activation='softmax')  # 10 outputs
    ])
    return model
```

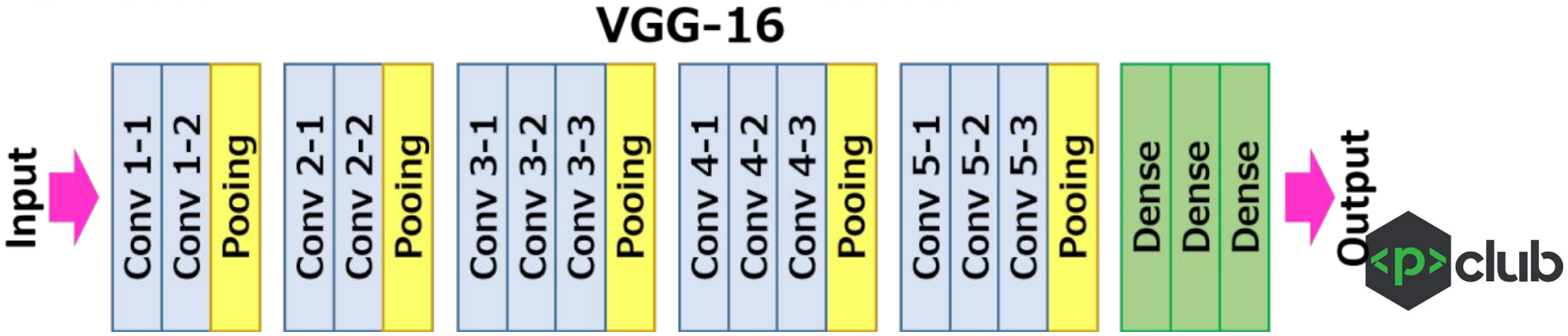


Classical Networks

- LeNet-5
- Alex Net
- VGG-16/19

VGG-16 Architecture

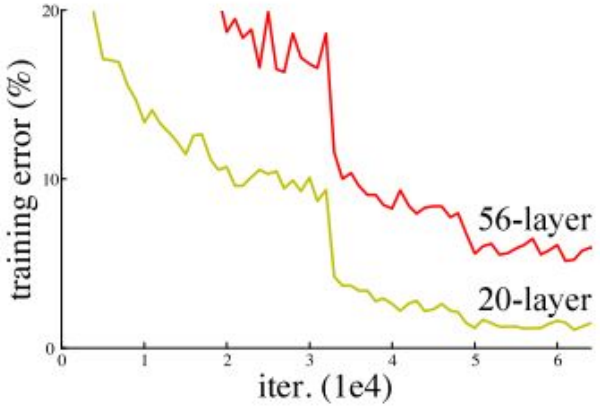


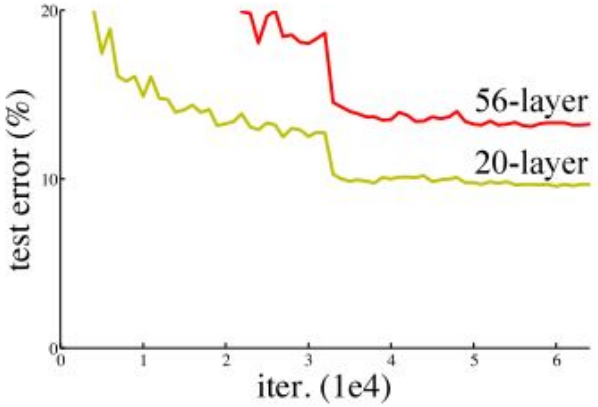


Researchers observed that it makes sense to affirm that "*the deeper the better*" when it comes to convolutional neural networks. This makes sense, since the models should be more capable (their flexibility to adapt to any space increase because they have a bigger parameter space to explore).

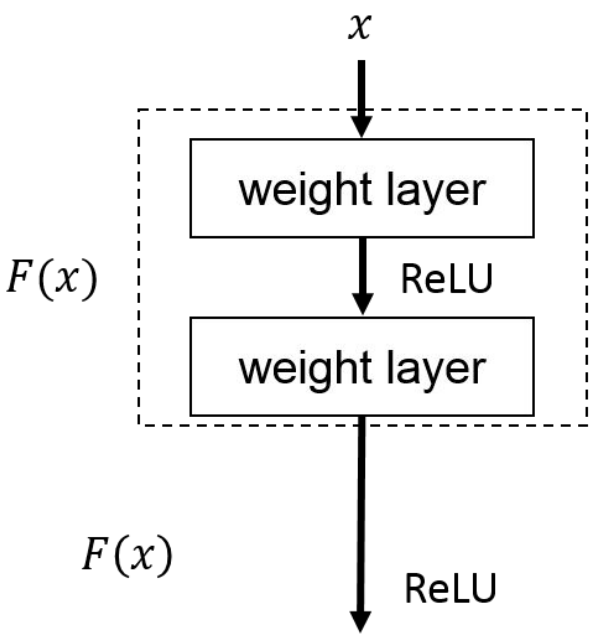
However, it has been noticed that after some depth, the performance degrades.

This was one of the bottlenecks of VGG. They couldn't go as deep as wanted, because they started to lose generalization capability.

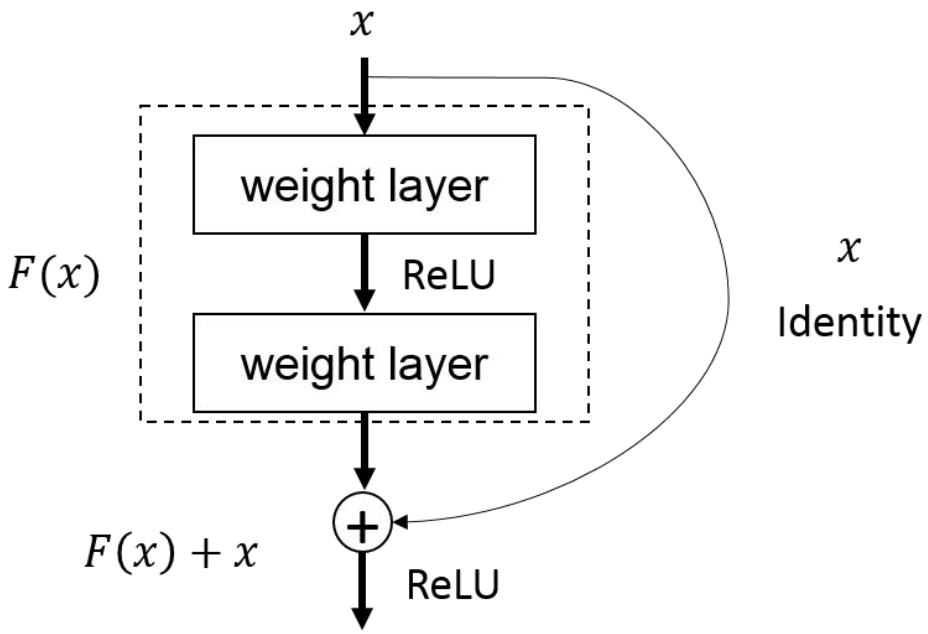


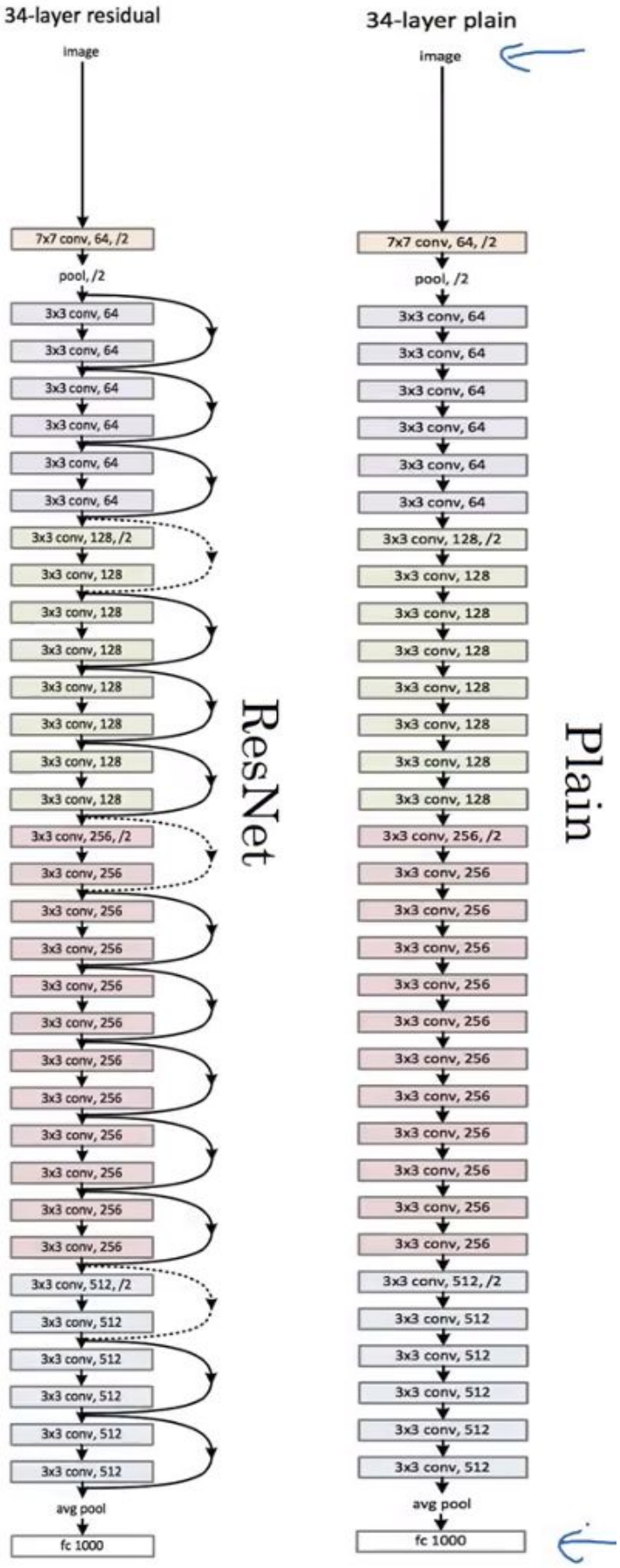


Plain network



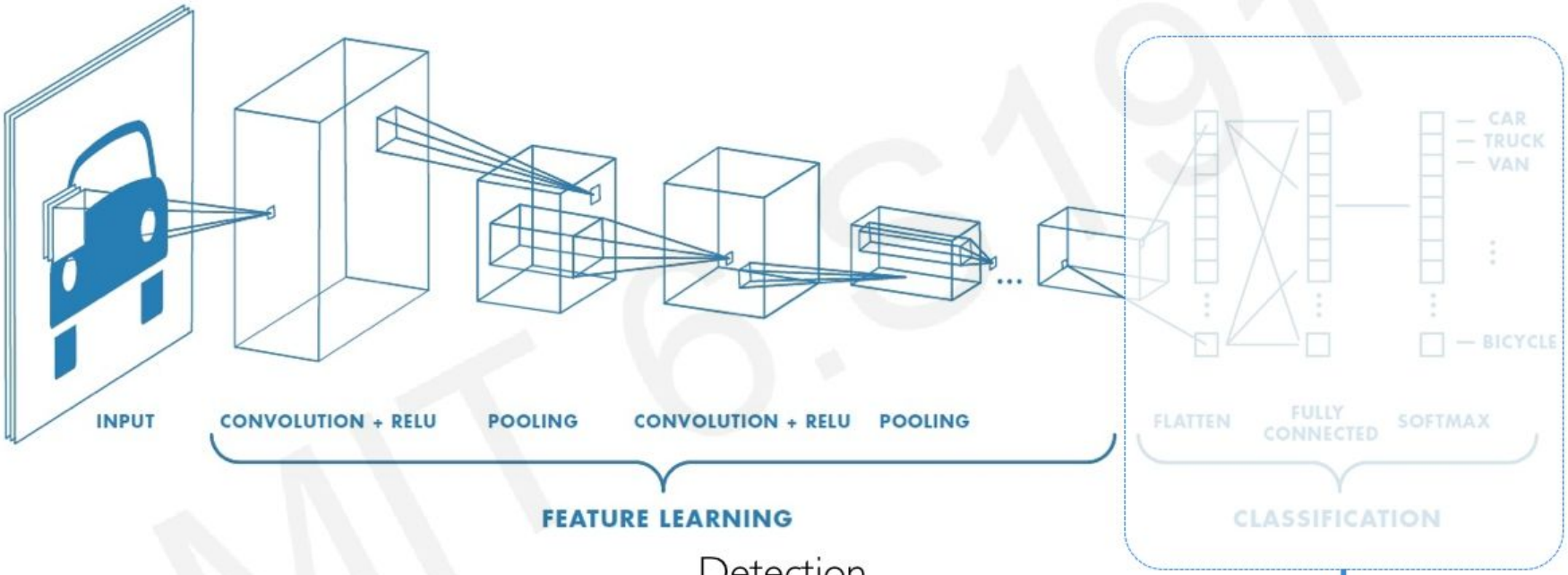
Residual network




34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
ResNet
34-layer plain
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000
Plain

An Architecture for Many Applications

An Architecture for Many Applications



Detection
Semantic segmentation
End-to-end robotic control

Deep Learning for Computer Vision: Impact

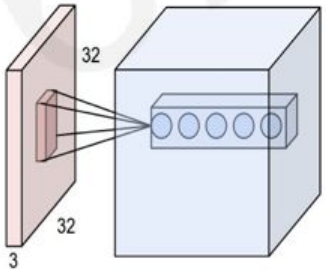
Deep Learning for Computer Vision: Summary

Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction

CNNs

- CNN architecture
- Application to classification
- ImageNet



Applications

- Segmentation, image captioning, control
- Security, medicine, robotics

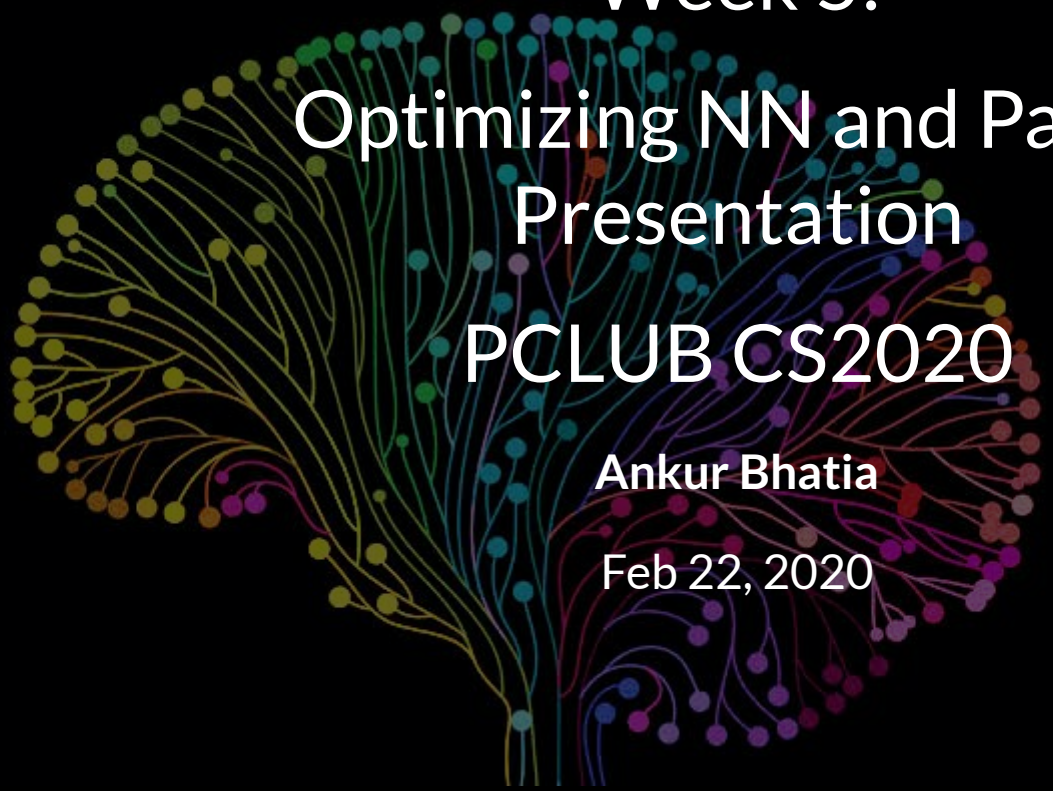
Week 5:

Optimizing NN and Paper Presentation

PCLUB CS2020

Ankur Bhatia

Feb 22, 2020



Today's Agenda:

1. Normalization and Batch Normalization
2. PAPER/ IDEA PRESENTATION

Normal Distribution

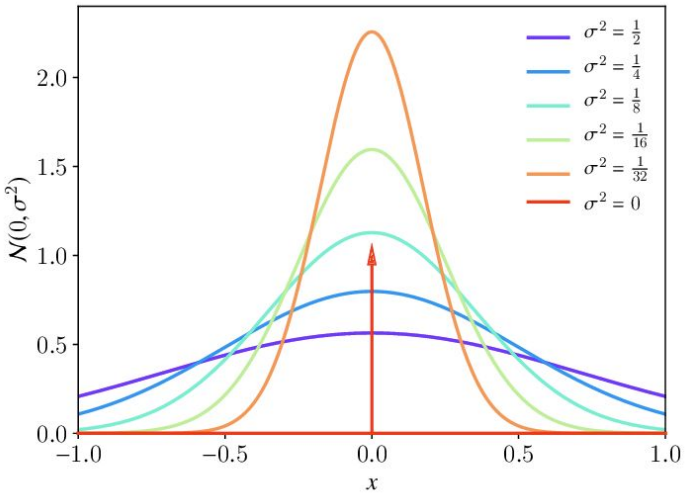
Normal distribution, also known as the **Gaussian distribution**, is a probability **distribution** that is symmetric about the mean, showing that data near the mean are more frequent in occurrence than data far from the mean.

$$\sigma = \sqrt{\frac{\sum(X-\mu)^2}{n}}$$

where,

σ = population standard deviation
$\sum$ = sum of...
μ = population mean
n = number of scores in sample.

Skewness and Kurtosis



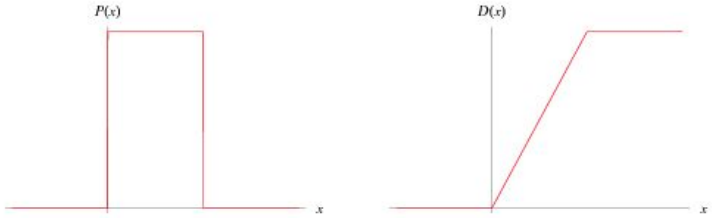
https://www.investopedia.com/terms/n/normaldistribution.asp

Uniform Distribution

Uniform Distribution

DOWNLOAD Wolfram Notebook EXPLORE THIS TOPIC IN The *MathWorld* Classroom

A uniform distribution, sometimes also known as a rectangular distribution, is a distribution that has constant probability.



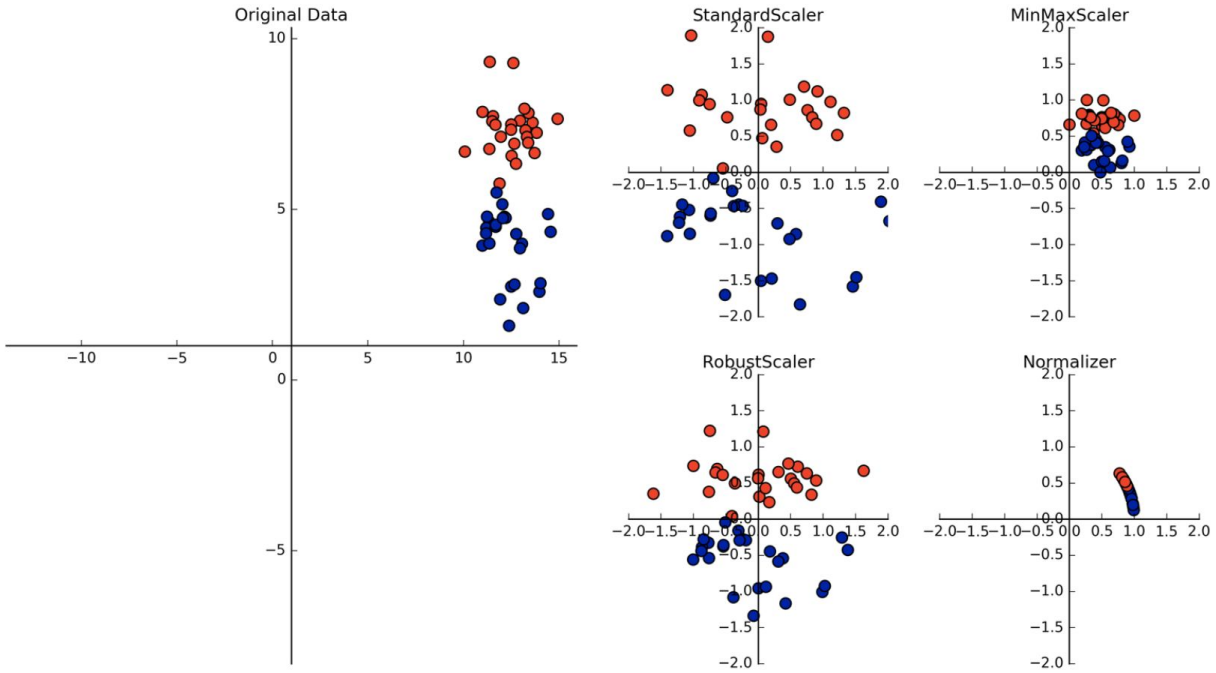
The probability density function and cumulative distribution function for a *continuous* uniform distribution on the interval $[a, b]$ are

$$P(x) = \begin{cases} 0 & \text{for } x < a \\ \dfrac{1}{b-a} & \text{for } a \le x \le b \\ 0 & \text{for } x > b \end{cases}$$

$$D(x) = \begin{cases} 0 & \text{for } x < a \\ \dfrac{x-a}{b-a} & \text{for } a \le x \le b \\ 1 & \text{for } x > b. \end{cases}$$

http://mathworld.wolfram.com/UniformDistribution.html

Normalizing Training Set



$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$$

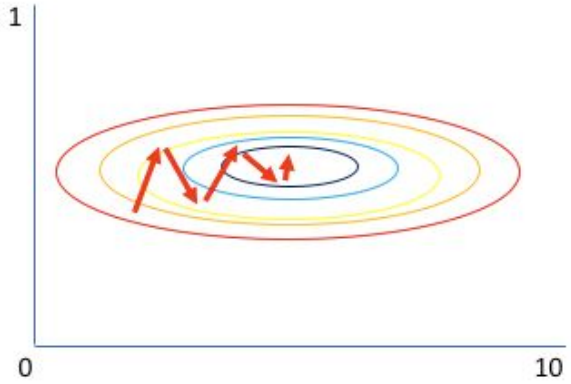
$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$$

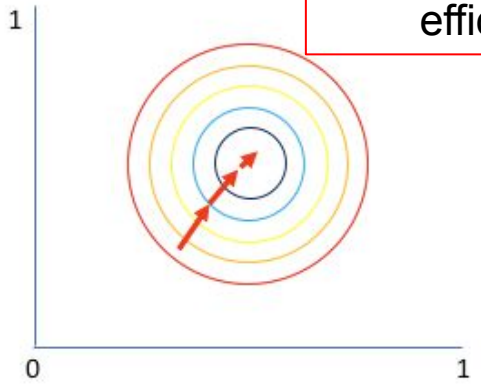
1. Use the same mean sd variance for train and test data.

Why normalize?

1. Cos func is round and easier to optimize.
2. Features on similar scale(similar variances)
3. Normalizing x1, x2, x3 …. helps you train w and b more efficiently.



Gradient of larger parameter dominates the update



Both parameters can be updated in equal proportions

If not normalized, then the parameter ratios can take very different values.
Cost func. will be an elongated bowl like.

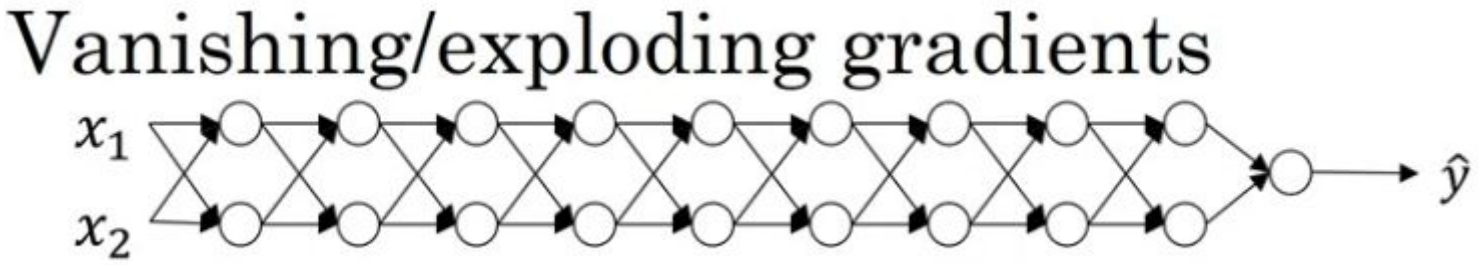
1. Easier and faster to optimize.

Random Weight Initialization

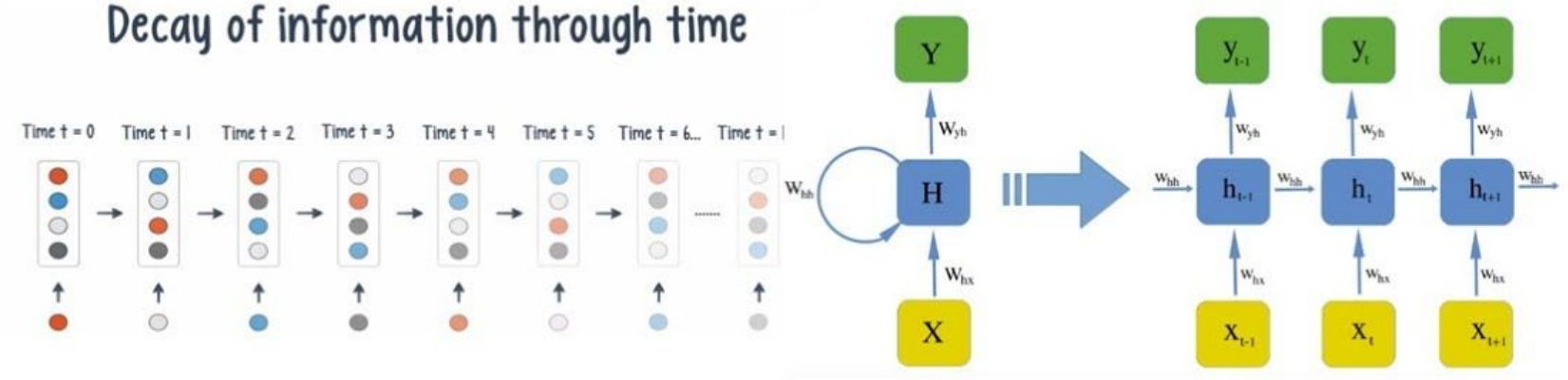
1. Vanishing and Exploding Gradients.
2. How to reduce the problem of V and E gradients?

Random Weight Initialization

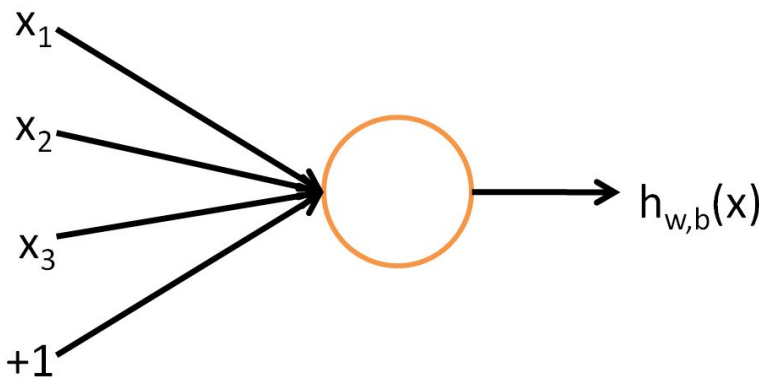
1. In Deep NN's , the derivatives become extremely small or large can cause problem in training or converging of the Neural Network.



Exploding Gradient Problem



Weight Initialization



1. $Z = W1X1 + W2X2 + W3X3 \ldots\ldots$
2. Larger the value of n, smaller the value of Wi required so that z do not explode.
3. $Variance(W_i) = 1/n$

Weight Initialization

1. Relu => $2/(n^{[l-1]})$.
2. Tanh => $1/(n^{[l-1]})$ (Xavier Initializiation)
3. => $2/(n^{[l-1]} + n^{[l]})$

In [25]:
```
def tanh(x): return torch.tanh(x)
```

In [26]:
```
x = torch.randn(512)

for i in range(100):
    a = torch.randn(512,512) * math.sqrt(1./512)
    x = tanh(a @ x)
x.mean(), x.std()
```

Out[26]:
```
(tensor(-0.0034), tensor(0.0613))
```

Gradient Checking

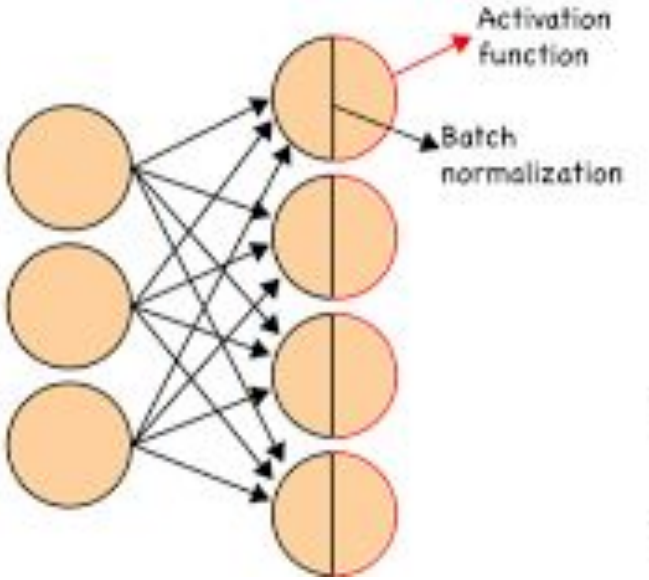
Helps in finding bugs in implementations of back propagation.

Batch Normalization

1. Makes hyperparameter search problem much easier.
2. I.e. The choice of hyperparameters is a much bigger range of hyperparameters that work well, and will also enable you to much more easily train even very deep networks.

Batch Normalization

1. Batch Normalizing after a layer l-1 will help in efficient training of the weights and biases of the layer l.
2. Normalization need not be mean = 0 and variance = 1.
3. Batch Norm generally applied on the z's.



$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$$

1. These beta and gamma [of each layer] are trainable parameters. Show over 2 layers**********
2. Any optimization algorithm can be used to train these parameters.
3. Programming Frameworks allows you just one line of code.
4. BN is applied generally on mini batches of data.
5. Show how bias parameter is irrelevant in applying batch norm. Beta (BN parameter) plays its role there then.

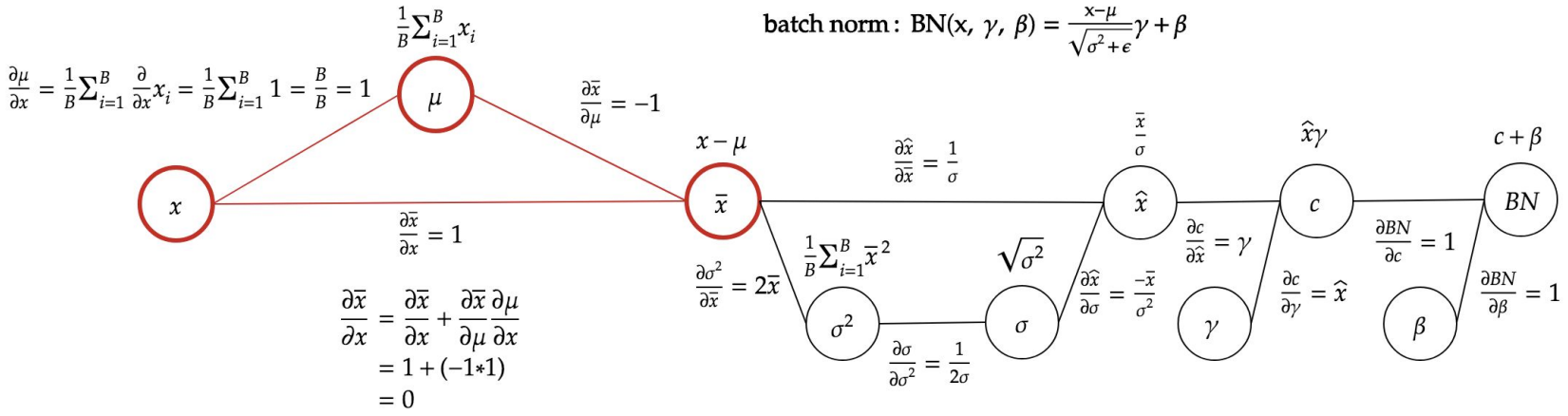
$$\boldsymbol{X} = \begin{bmatrix} X_{1,1} & X_{1,2} & \cdots & X_{1,n} \\ X_{2,1} & X_{2,2} & \cdots & X_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ X_{m,1} & X_{m,2} & \cdots & X_{m,n} \end{bmatrix} \tag{1}$$

$$\hat{\boldsymbol{X}} = \left(\boldsymbol{X} - \boldsymbol{\mu}^\top\right) \odot \hat{\boldsymbol{a}}^\top$$

$$\hat{\boldsymbol{X}} = \begin{bmatrix} \frac{X_{1,1}-\mu_1}{\sqrt{\sigma_1^2+\epsilon}} & \frac{X_{1,2}-\mu_2}{\sqrt{\sigma_2^2+\epsilon}} & \cdots & \frac{X_{1,n}-\mu_n}{\sqrt{\sigma_n^2+\epsilon}} \\ \frac{X_{2,1}-\mu_1}{\sqrt{\sigma_1^2+\epsilon}} & \frac{X_{2,2}-\mu_2}{\sqrt{\sigma_2^2+\epsilon}} & \cdots & \frac{X_{2,n}-\mu_n}{\sqrt{\sigma_n^2+\epsilon}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{X_{m,1}-\mu_1}{\sqrt{\sigma_1^2+\epsilon}} & \frac{X_{m,2}-\mu_2}{\sqrt{\sigma_2^2+\epsilon}} & \cdots & \frac{X_{m,n}-\mu_n}{\sqrt{\sigma_n^2+\epsilon}} \end{bmatrix}$$

$$\boldsymbol{Y} = \boldsymbol{\gamma}^\top \odot \hat{\boldsymbol{X}} + \boldsymbol{\beta}^\top \tag{2}$$

Back Prop with Batch Norm



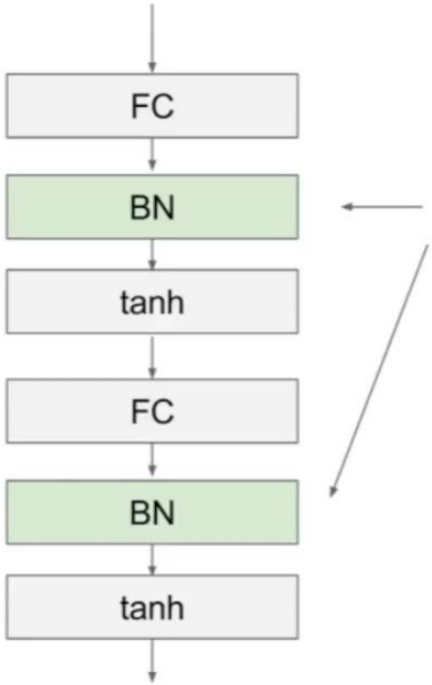
$$\frac{\partial BN}{\partial x} = \left(\frac{\partial BN}{\partial c}\frac{\partial c}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial \bar{x}} + \frac{\partial BN}{\partial c}\frac{\partial c}{\partial \hat{x}}\frac{\partial \hat{x}}{\partial \sigma}\frac{\partial \sigma}{\partial \sigma^2}\frac{\partial \sigma^2}{\partial \bar{x}}\right)\frac{\partial \bar{x}}{\partial \mu}\frac{\partial \mu}{\partial x} + \frac{\partial \bar{x}}{\partial x}$$

https://datascience.stackexchange.com/questions/30056/deriving-the-gradient-of-batch-normalization

<p> club

Batch Normalization

[Ioffe and Szegedy, 2015]



Usually inserted after Fully Connected or Convolutional layers, and before nonlinearity.

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Norm over a mini batch

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

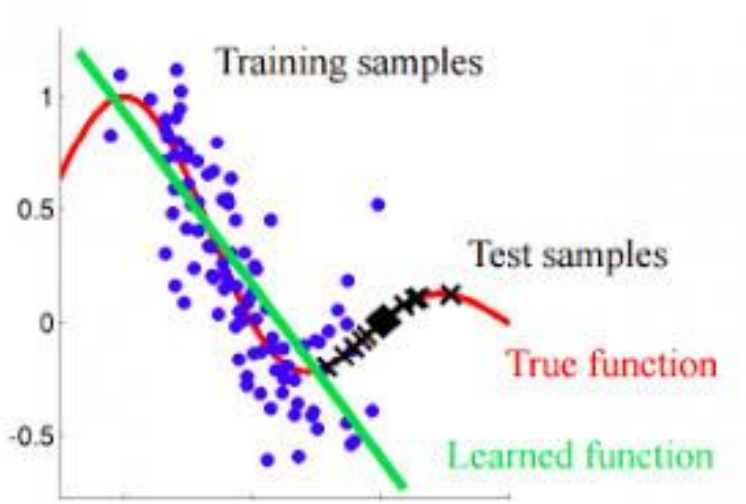
$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

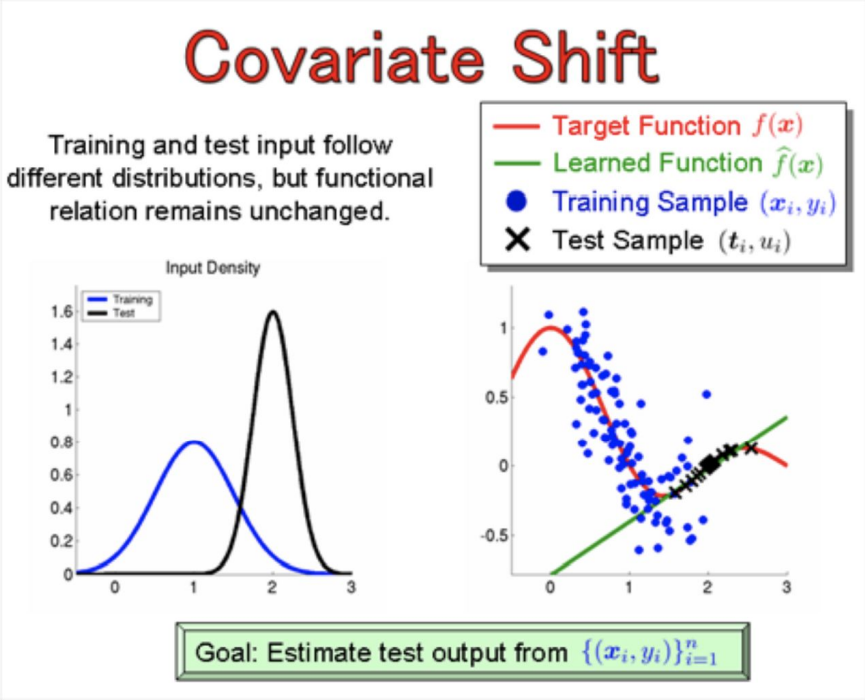
From the original batch-norm paper

<p>club

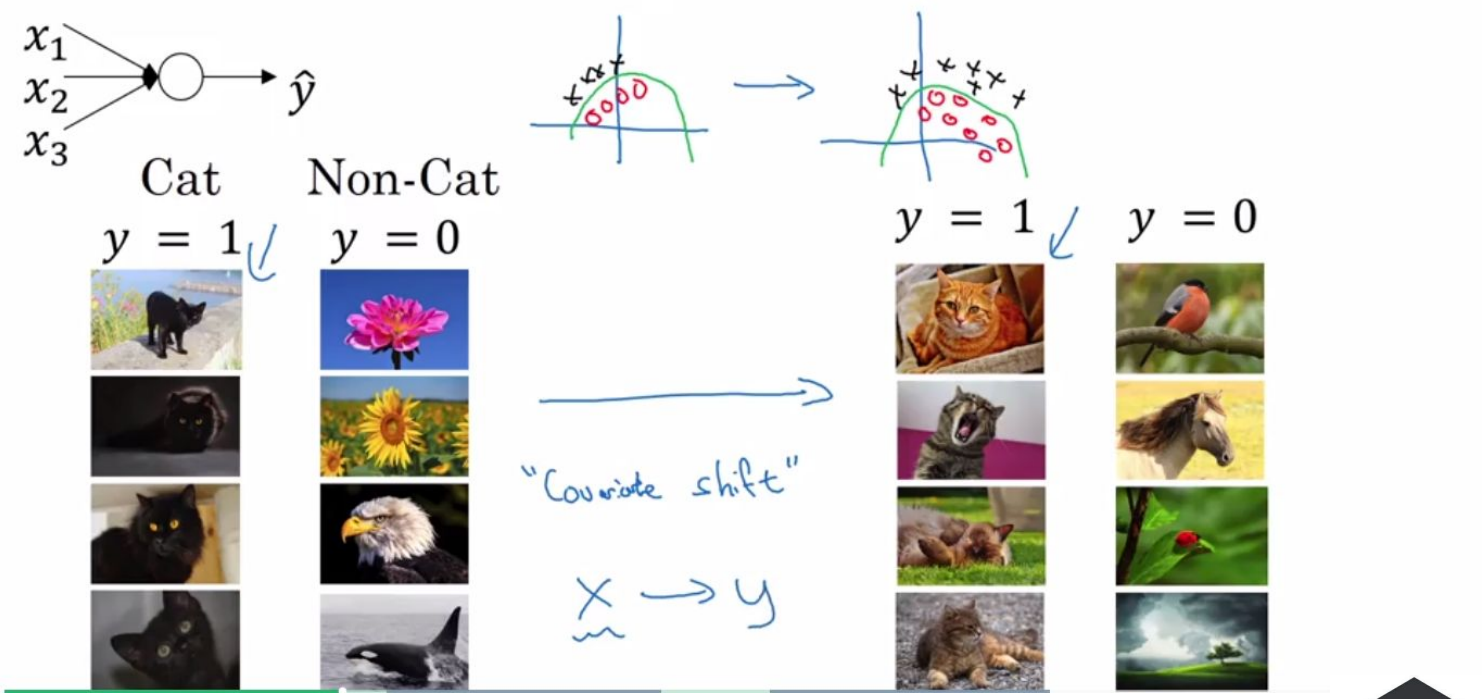
Covariate Shift



Covariate Shift



Covariate Shift



Deeplearning.ai: Why Does Batch Norm Work? (C2W3L06)

Paper Presentations