Intro to Deep Learning

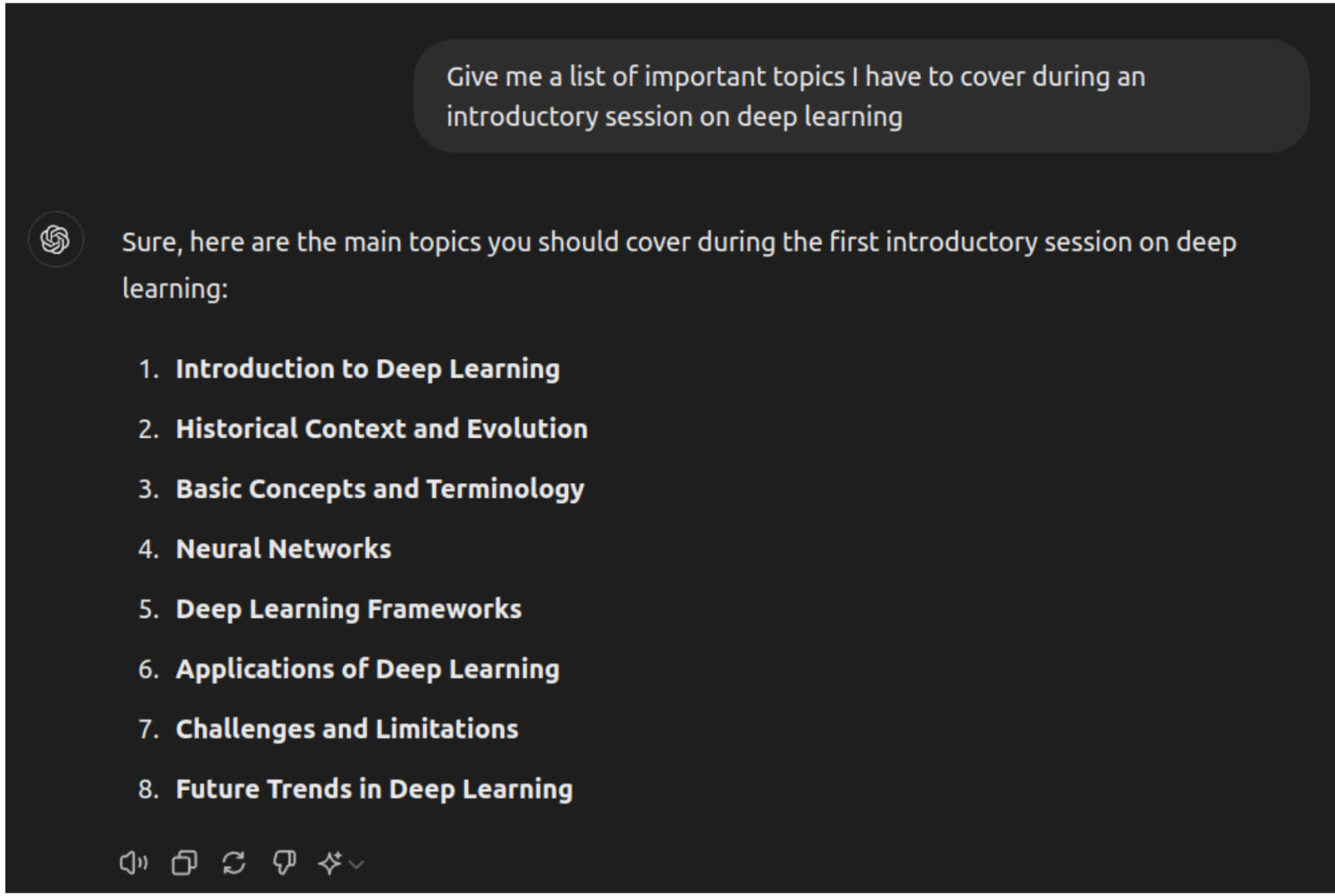
Gnana Prakash Punnavajhala

Why study Deep Learning?

https://www.youtube.com/watch?v=l82PxsKHxYc

Why study Deep Learning?

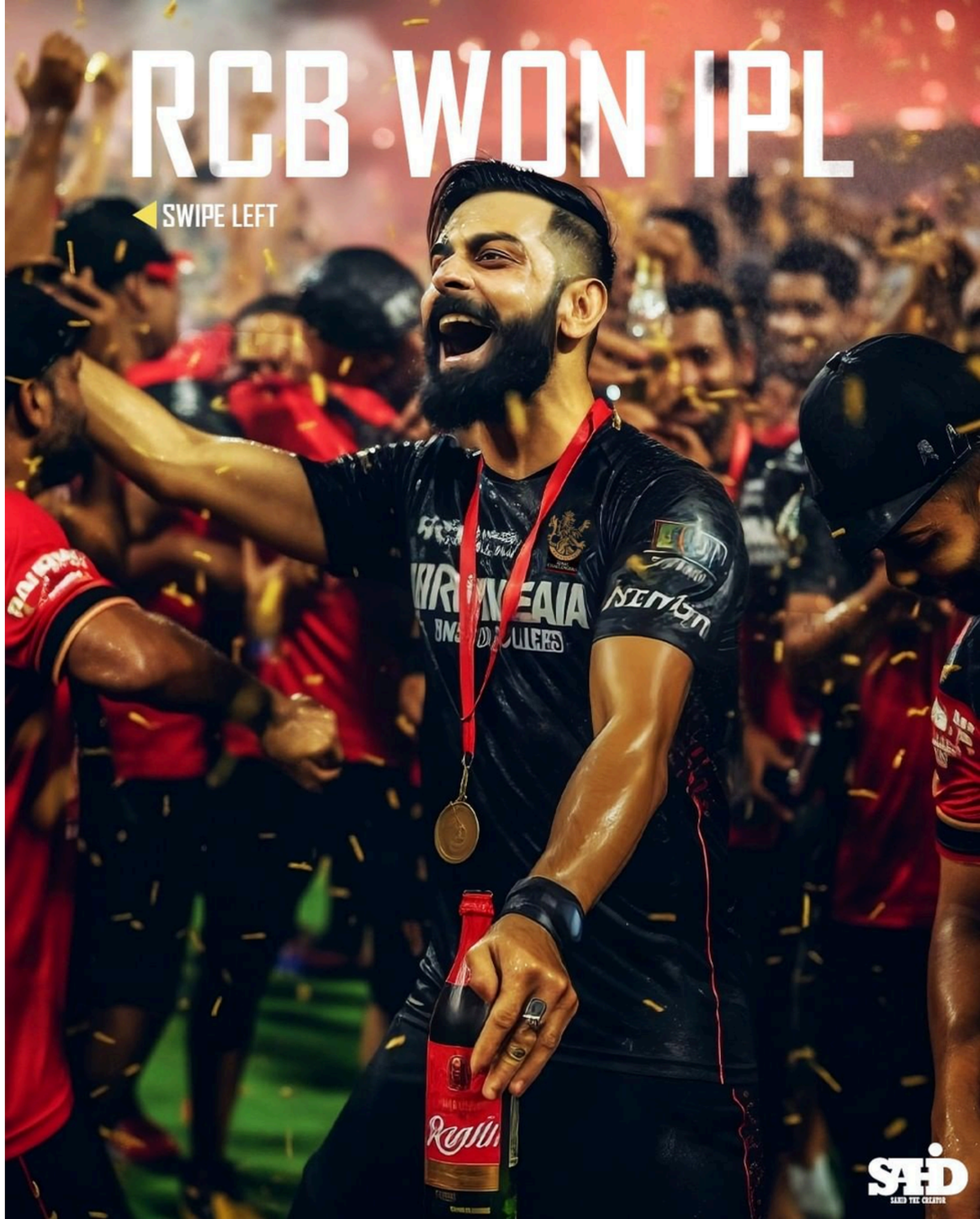
Generating Language
from
Natural Language



https://chatgpt.com/

Why study Deep Learning?

Generating Images
from
Natural Language



https://www.instagram.com/p/CsOXGGAvVwx/?img_index=1

Why study Deep Learning?

Generating Videos
from
Natural Language

https://www.youtube.com/watch?v=HK6y8DAPN_0

Why study Deep Learning?

And much more...

- **Computer Vision** - Image Recognition, Augmented Reality
- **Robotics** - Self-Driving Cars, Autonomous Robots
- **NLP** - Machine Translation, Chatbots and Virtual Assistants, Sentiment analysis
- **Healthcare** - Drug Discovery, Disease Prediction and Diagnosis
- **Entertainment** - Recommendation Systems

What is Deep Learning?

ARTIFICIAL INTELLIGENCE - Any technique that enables computers to mimic human behaviour

MACHINE LEARNING - Ability to learn without explicitly being programmed to perform a task

DEEP LEARNING - Extract patterns only from data using “neural networks”

- Teaching computers how to learn a task directly from raw data

Why is everyone talking about Deep Learning now?

- Big Data
  - Larger Datasets
  - More efficient Storage
  - Better Accessibility

IMAGENET

WIKIPEDIA
The Free Encyclopedia

LAION

- Hardware
  - Graphical Processing Units (GPUs)
  - Massively Parallelizable

NVIDIA

https://www.reddit.com/r/ChatGPT/comments/14l6drm/when_everyone_digs_for_gold_sell_shovels/

Why is everyone talking about Deep Learning now?

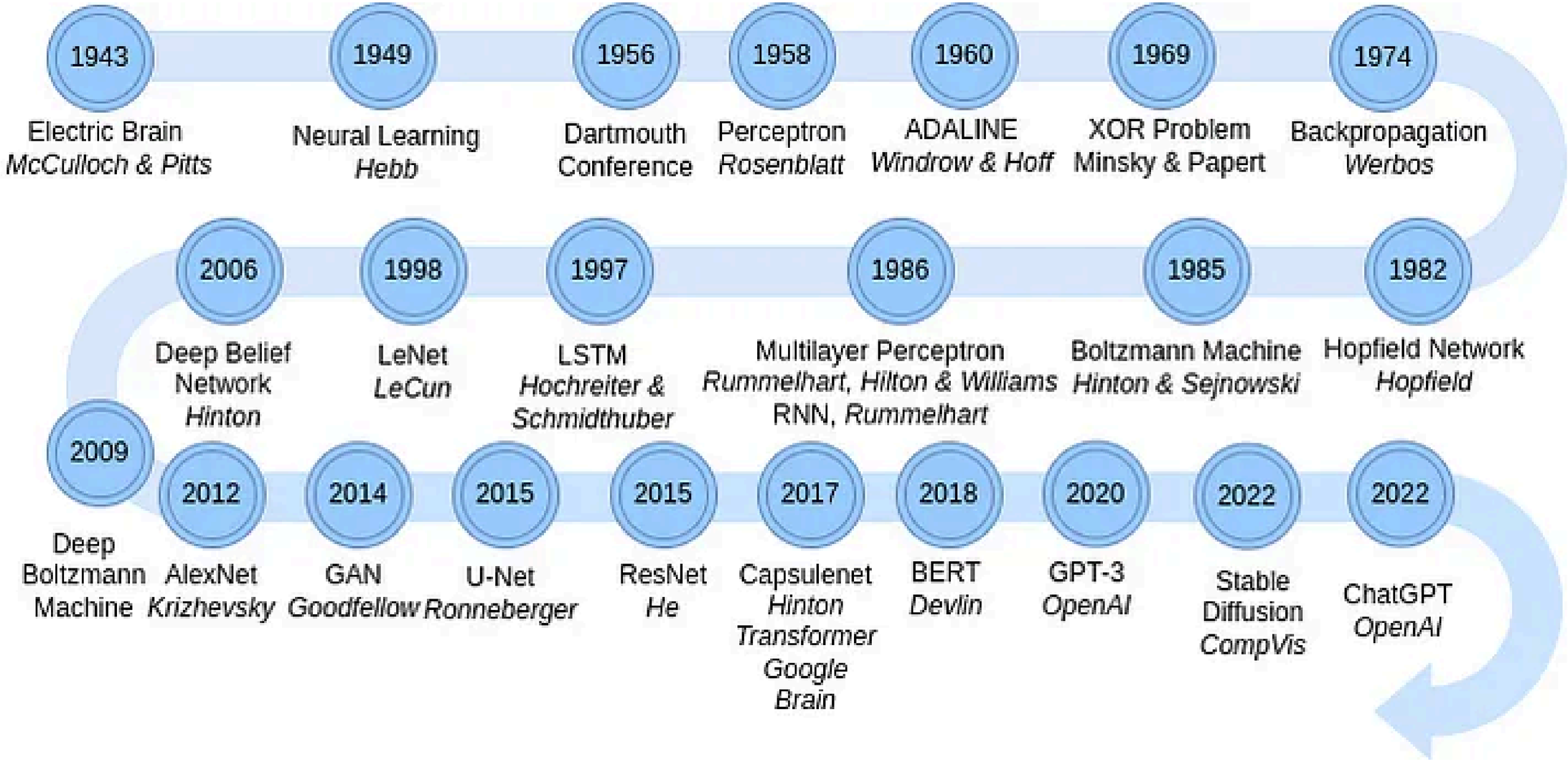
- Big Data
  - Larger Datasets
  - More efficient Storage
  - Better Accessibility

- Hardware
  - Graphical Processing Units (GPUs)
  - Massively Parallelizable

- Software
  - Efficient Packages
  - Easy Availability of models

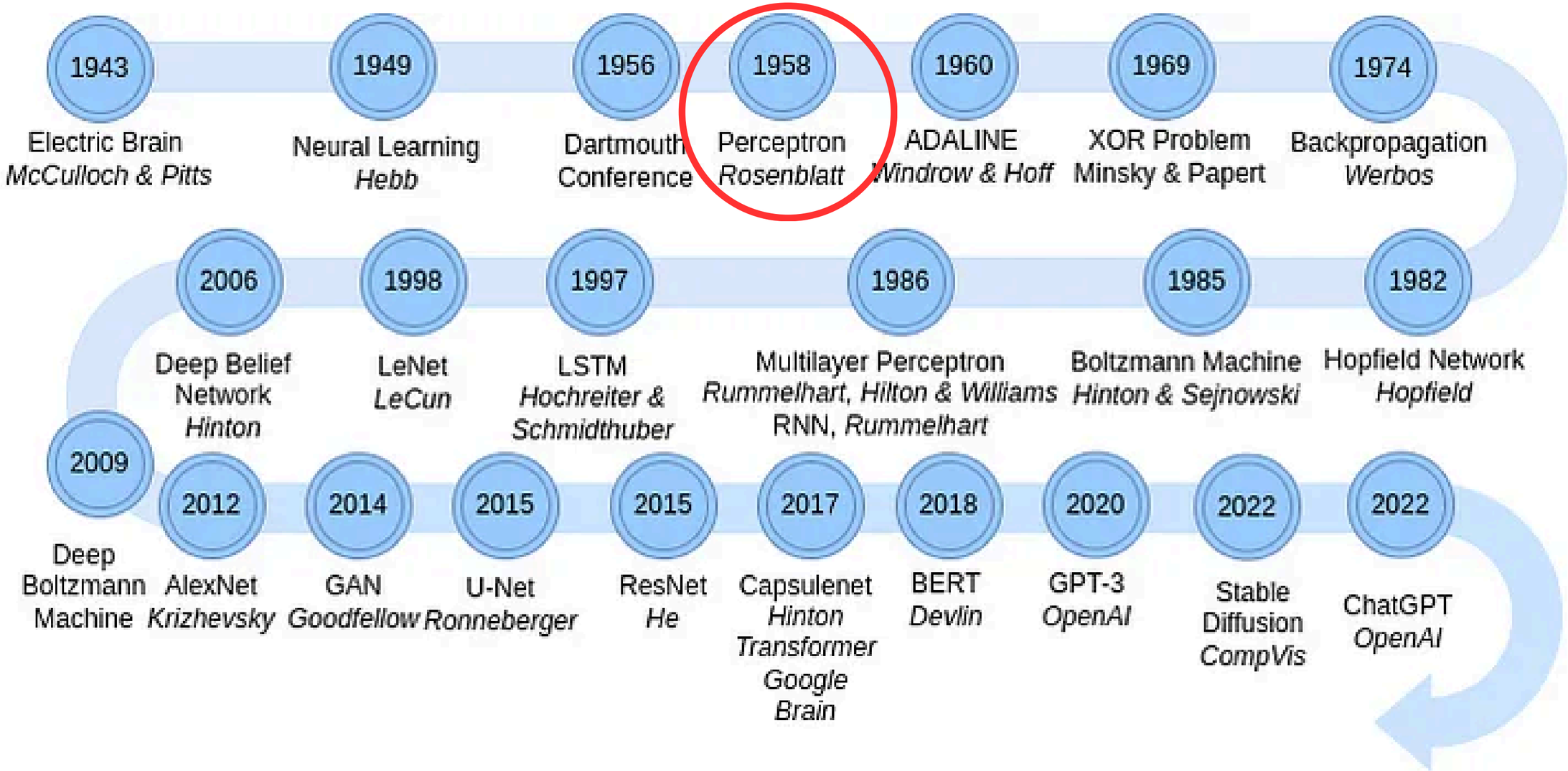
- Attention Is All You Need

Deep Learning Timeline



https://anupamakandala1111.medium.com/mastering-advanced-deep-learning-a-comprehensive-roadmap-56681af2706e

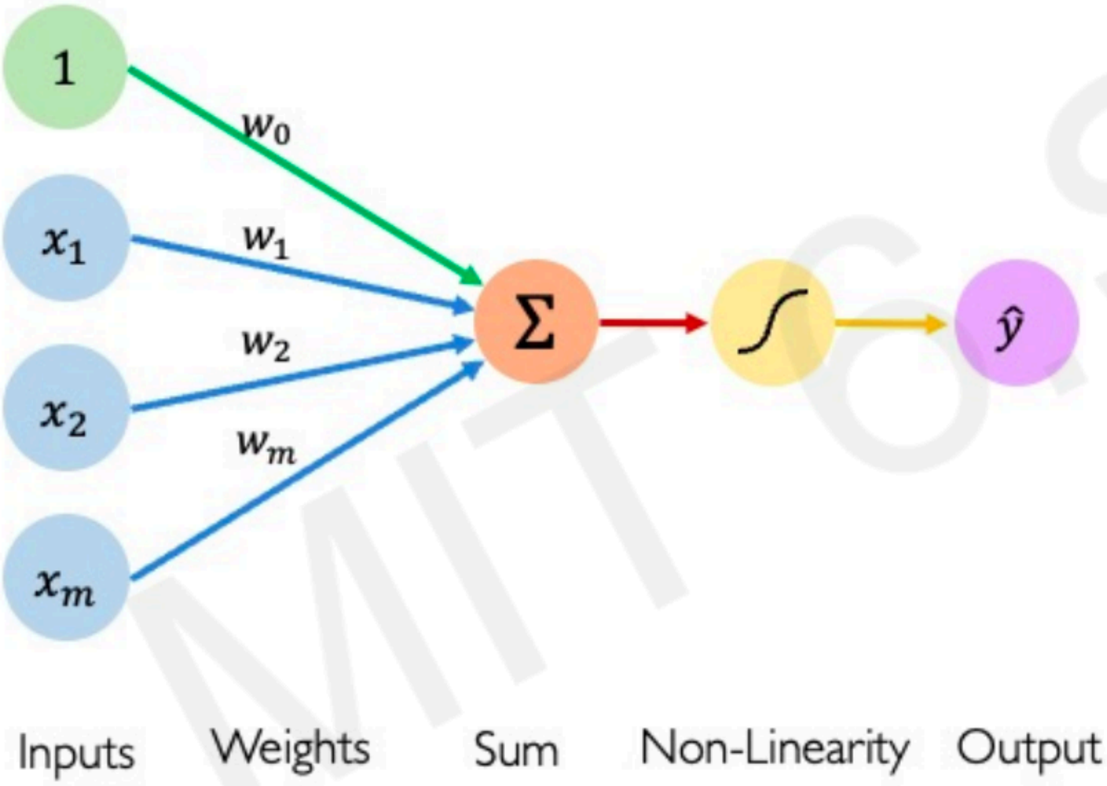
Let’s get started…



https://anupamakandala1111.medium.com/mastering-advanced-deep-learning-a-comprehensive-roadmap-56681af2706e

Perceptron - The structural building block of DL

The Perceptron: Forward Propagation



Output

Linear combination of inputs

$$\hat{y} = g\left(w_0 + \sum_{i=1}^{m} x_i\, w_i\right)$$

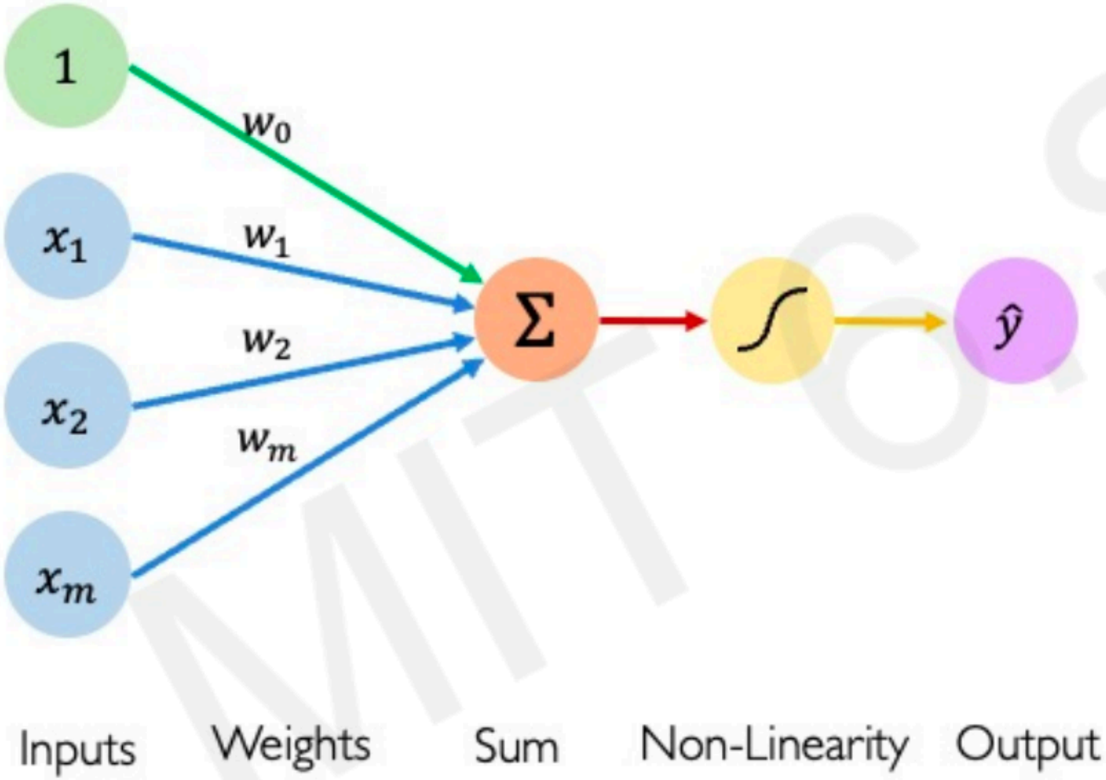
Non-linear activation function

Bias

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Perceptron - The structural building block of DL

The Perceptron: Forward Propagation

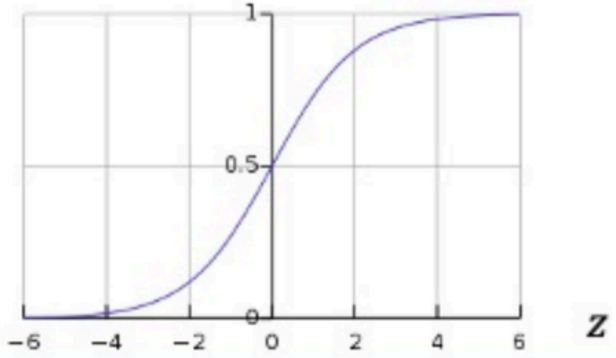


Activation Functions

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

- Example: sigmoid function

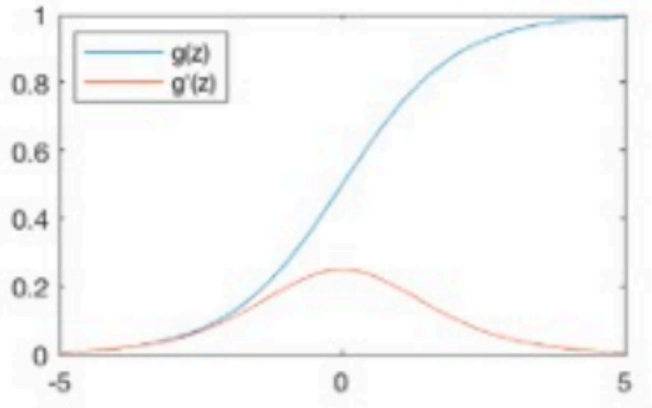
$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Common Activation Functions

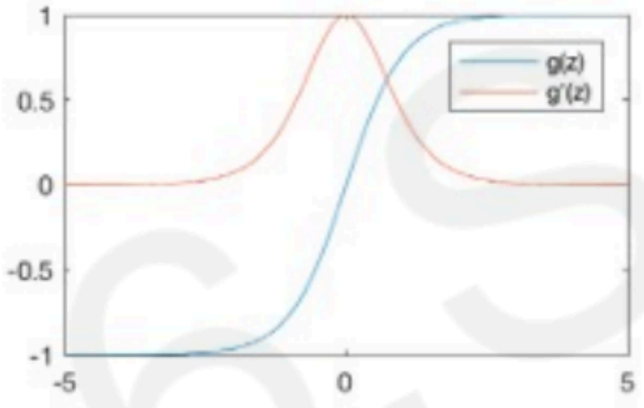
Sigmoid Function



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

$$g'(z) = g(z)(1 - g(z))$$

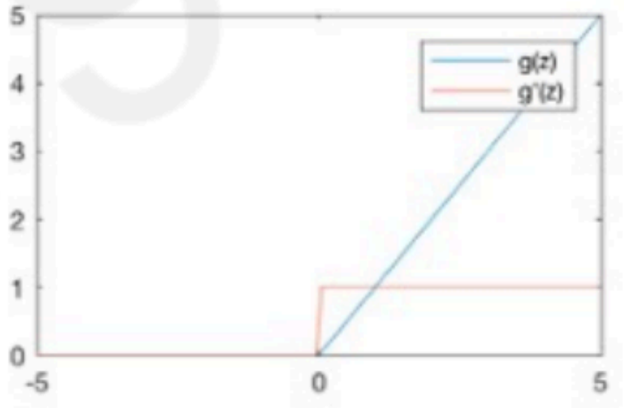
Hyperbolic Tangent



$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

Rectified Linear Unit (ReLU)

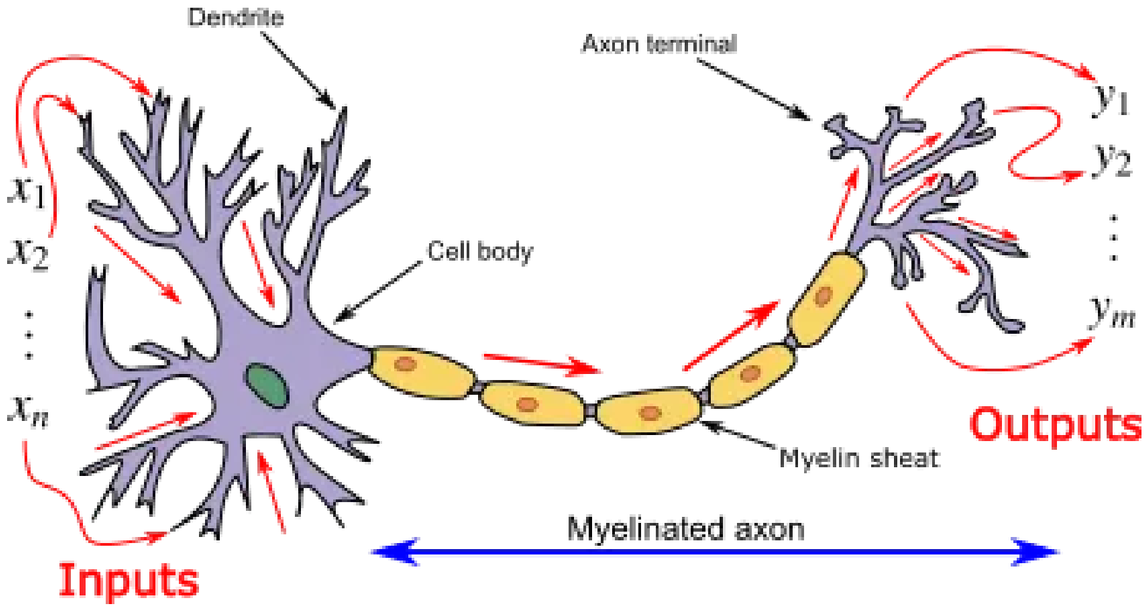


$$g(z) = \max(0, z)$$

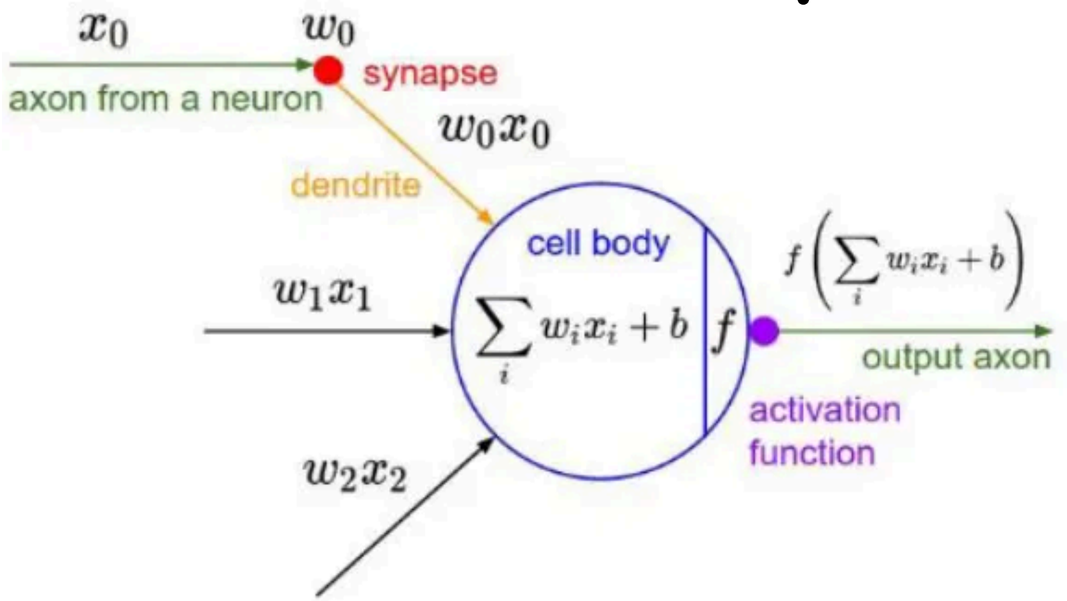
$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

How do we build Neural Networks from Perceptrons?



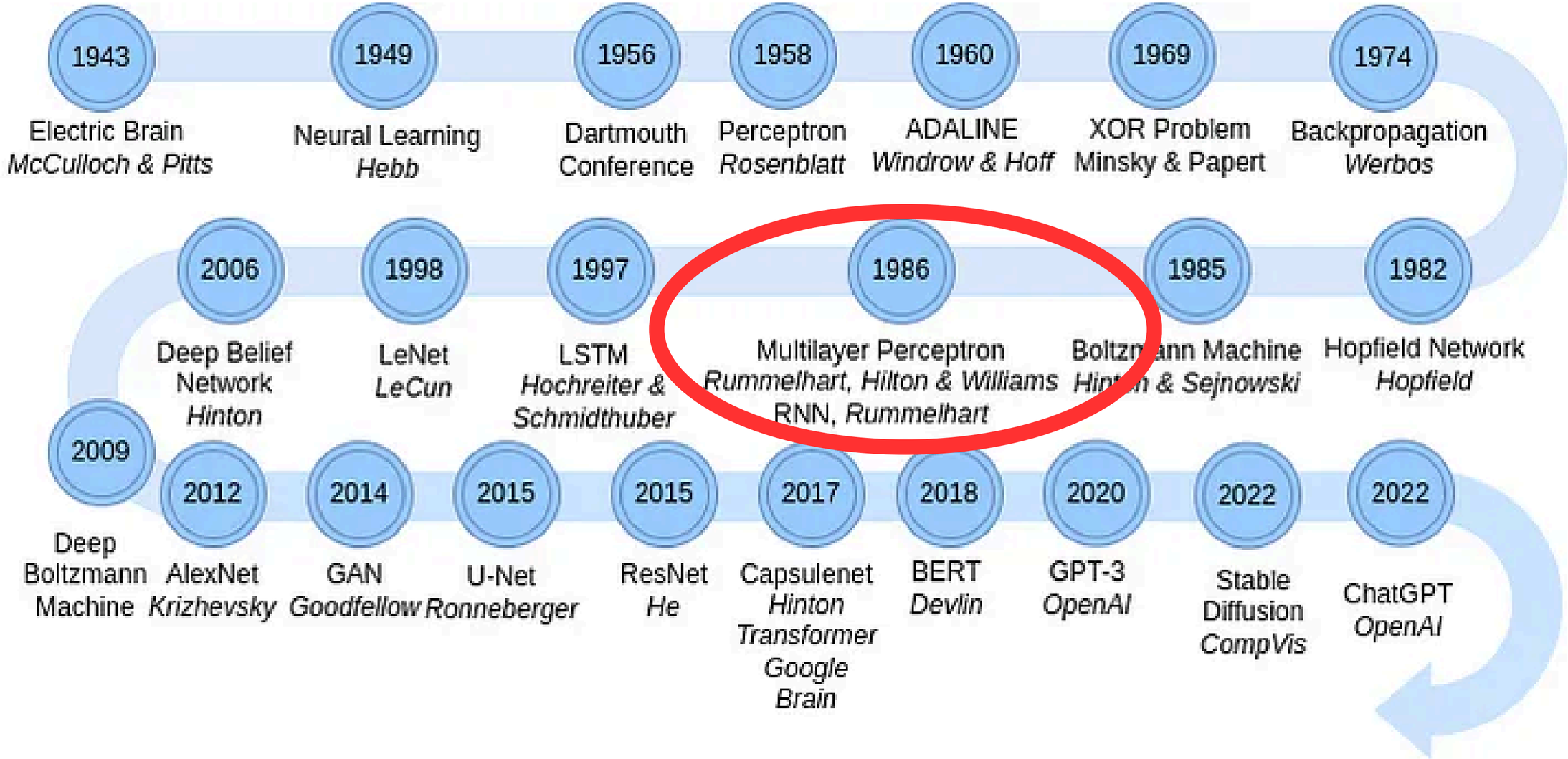
https://medium.com/@parthbs/the-perceptron-70d6aaca158b



https://medium.com/analytics-vidhya/activation-functions-all-you-need-to-know-355a850d025e

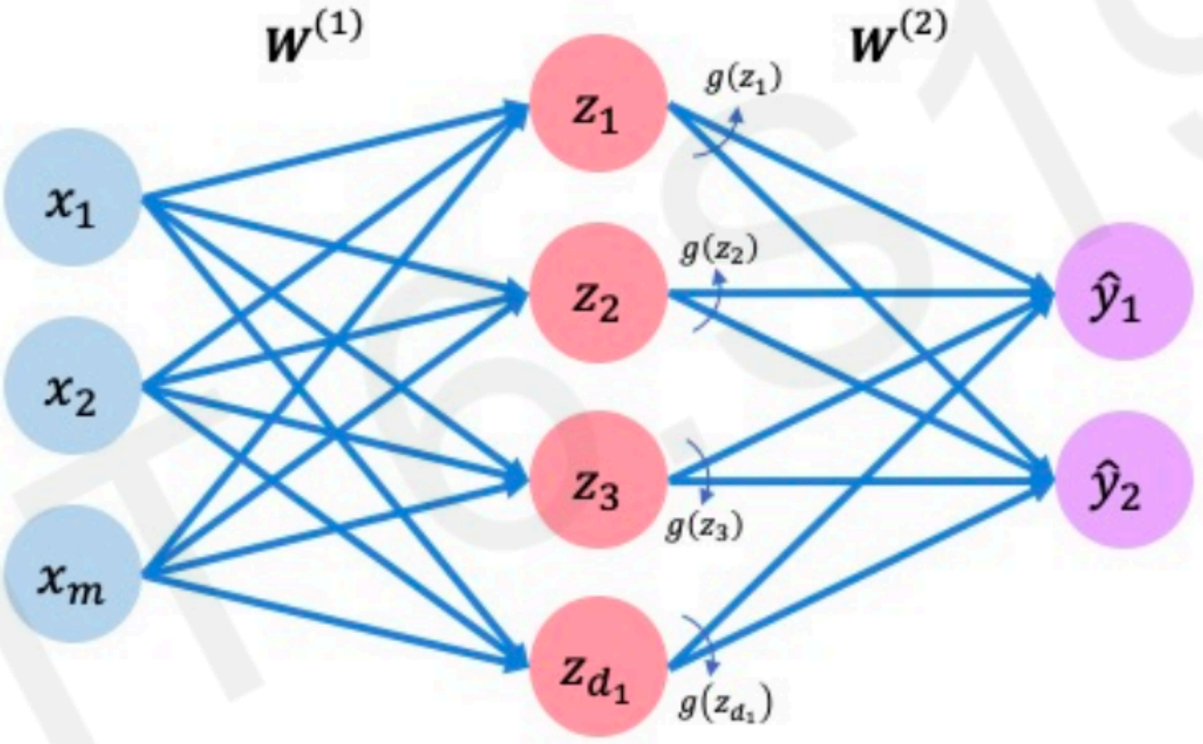
- Each perceptron is treated as a single neuron cell with multiple inputs and multiple outputs
- We chain many such neurons (or perceptrons, in our case) to get neural networks
- Note: Apart from the structural similarity, the functioning of neural networks is in no way connected to the way in which our brain functions

Neural Networks - Multi Layer Perceptrons (MLPs)



https://anupamakandala1111.medium.com/mastering-advanced-deep-learning-a-comprehensive-roadmap-56681af2706e

Single Layer Neural Network



Inputs Hidden Final Output

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

What is the need for activation functions?

- Hidden layer output

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

- Final output without activation

$$\hat{y}_i = w_{0,i}^{(2)} + \sum_{j=1}^{d_1} z_j w_{j,i}^{(2)}$$

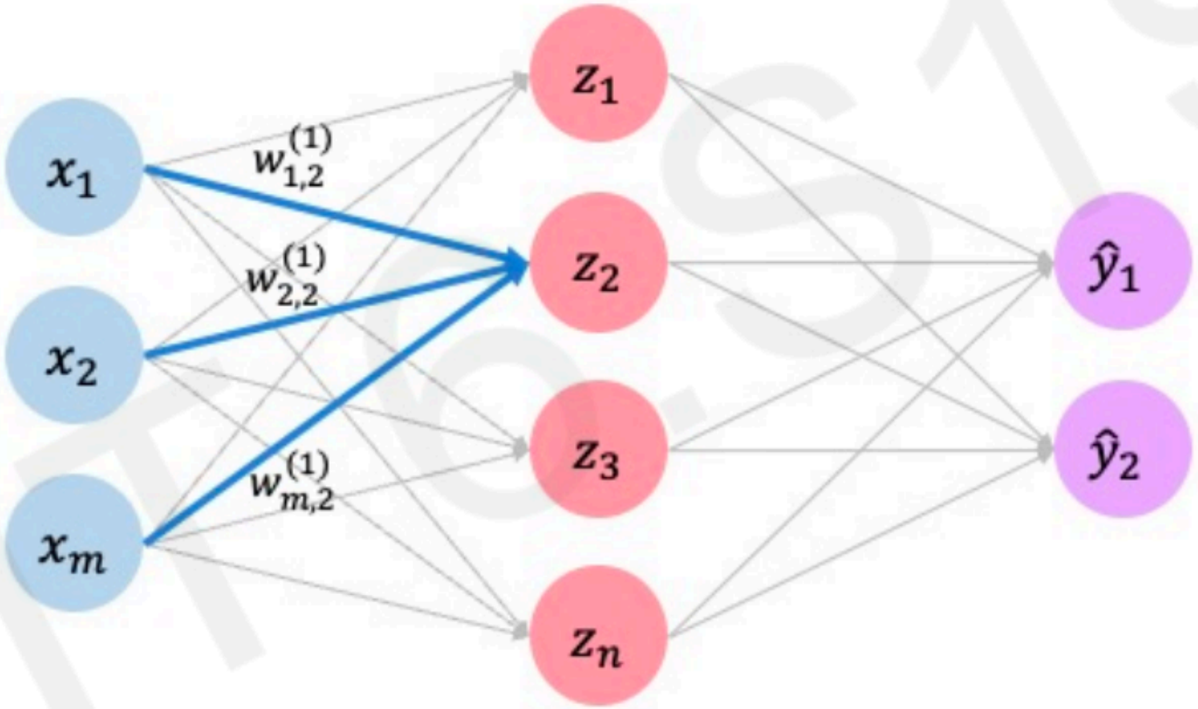
$$\hat{y}_i = w_{0,i}^{(2)} + \sum_{j=1}^{d_1} \left(w_{0,j}^{(1)} + \sum_{k=1}^{m} x_k w_{k,j}^{(1)} \right) w_{j,i}^{(2)}$$

$$\hat{y}_i = \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} w_{0,j}^{(1)} \cdot w_{j,i}^{(2)} \right) + \sum_{k=1}^{m} x_k \left(\sum_{j=1}^{d_1} w_{k,j}^{(1)} \cdot w_{j,i}^{(2)} \right)$$

$$\hat{y}_i = w'_{0,i} + \sum_{k=1}^{m} x_k w'_{k,i}$$

- Deep networks can be reduced to a single layer network!
- Network cannot handle non-linear data!

Single Layer Neural Network

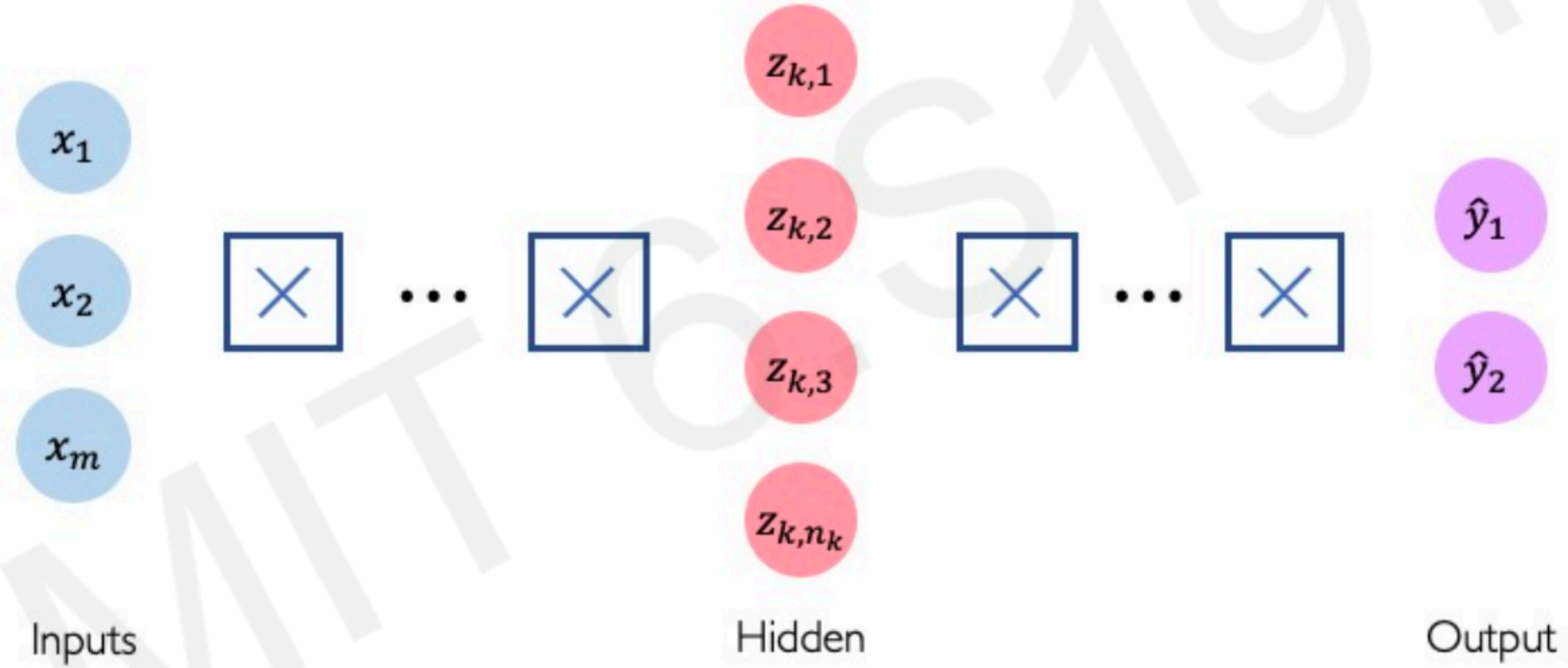


$$z_2 = w_{0,2}^{(1)} + \sum_{j=1}^{m} x_j \, w_{j,2}^{(1)}$$

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Deep Neural Network



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

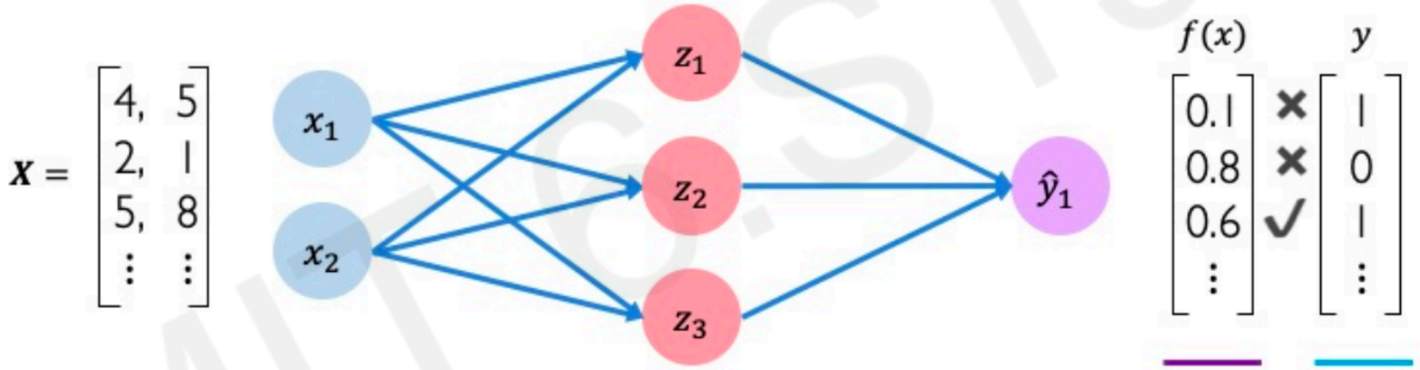
http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Putting the model to use

- Great! We know how to construct the model. Now what?
- How do we quantify the model's performance? Loss functions!

Empirical Loss

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



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

$f(x)$ y

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \begin{matrix} \times \\ \times \\ \checkmark \\ \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

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

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

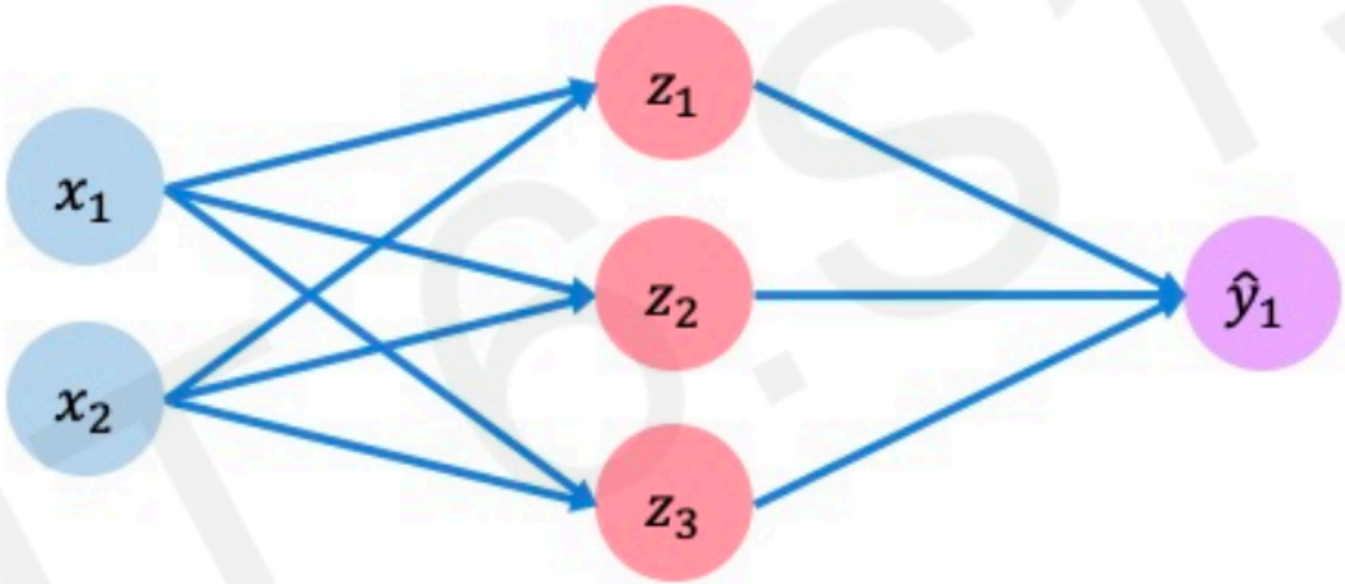
Predicted Actual

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Binary Cross Entropy Loss

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

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



$f(x)$ y

$$\begin{bmatrix} 0.1 \\ 0.8 \\ 0.6 \\ \vdots \end{bmatrix} \begin{matrix} \times \\ \times \\ \checkmark \\ \end{matrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ \vdots \end{bmatrix}$$

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

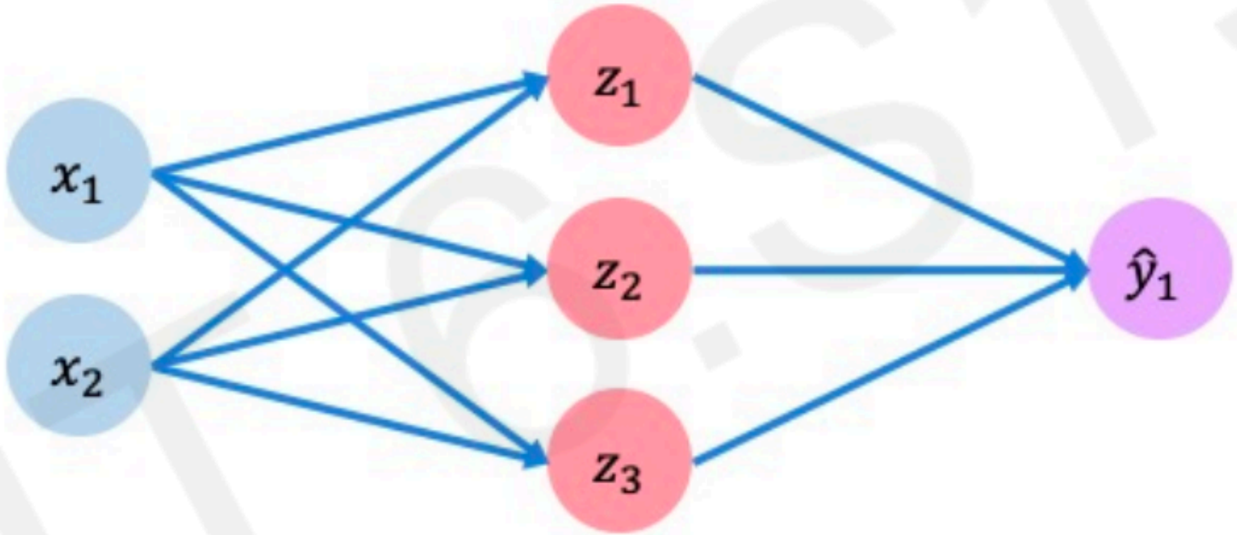
Actual Predicted Actual Predicted

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Mean Squared Error Loss

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

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



$f(x)$ y

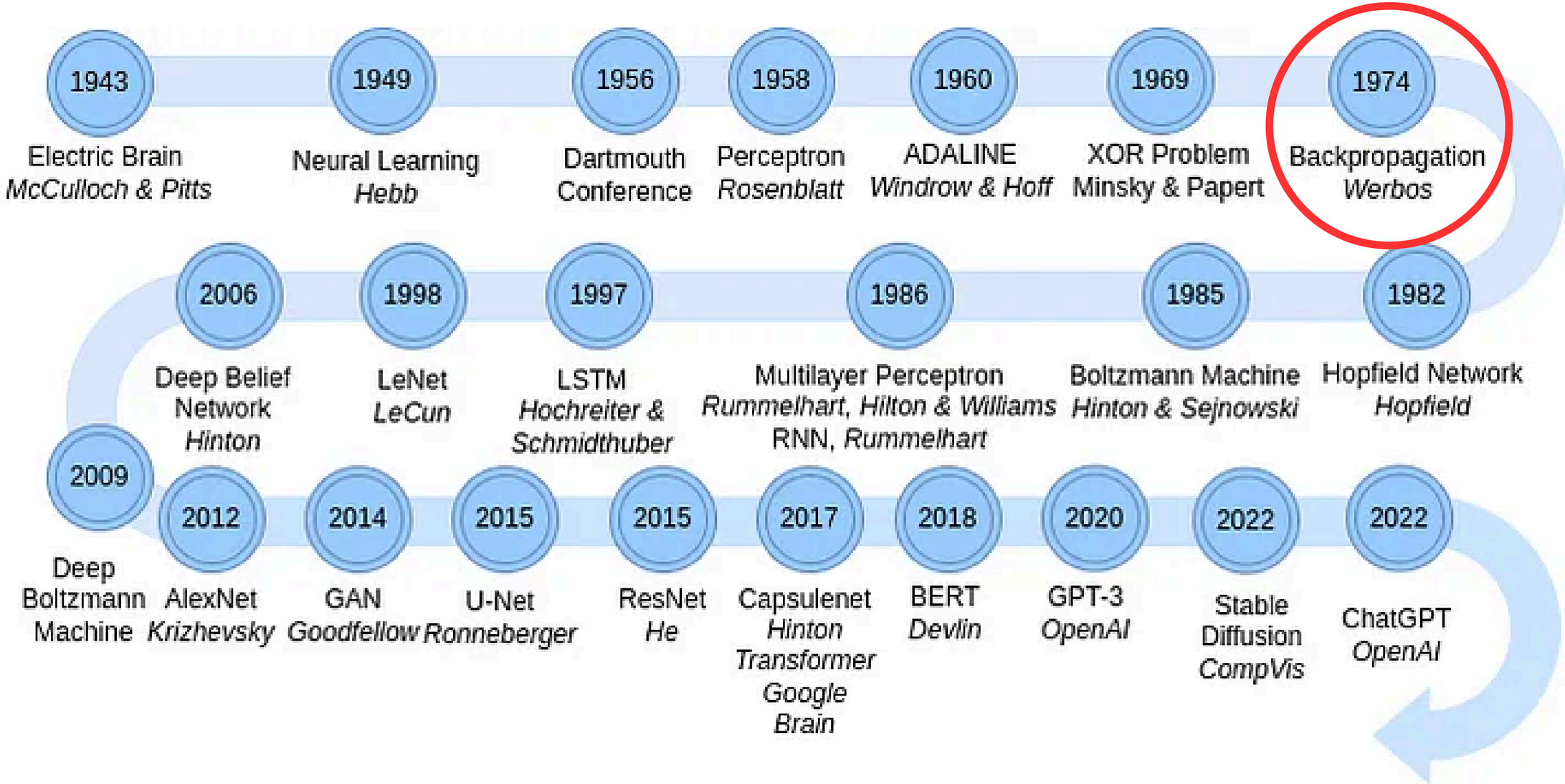
$$\begin{bmatrix} 30 \\ 80 \\ 85 \\ \vdots \end{bmatrix} \begin{matrix} \times \\ \times \\ \checkmark \\ \end{matrix} \begin{bmatrix} 90 \\ 20 \\ 95 \\ \vdots \end{bmatrix}$$

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

Actual Predicted

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Training the Model



https://anupamakandala1111.medium.com/mastering-advanced-deep-learning-a-comprehensive-roadmap-56681af2706e

Loss Optimization

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

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

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

Remember:

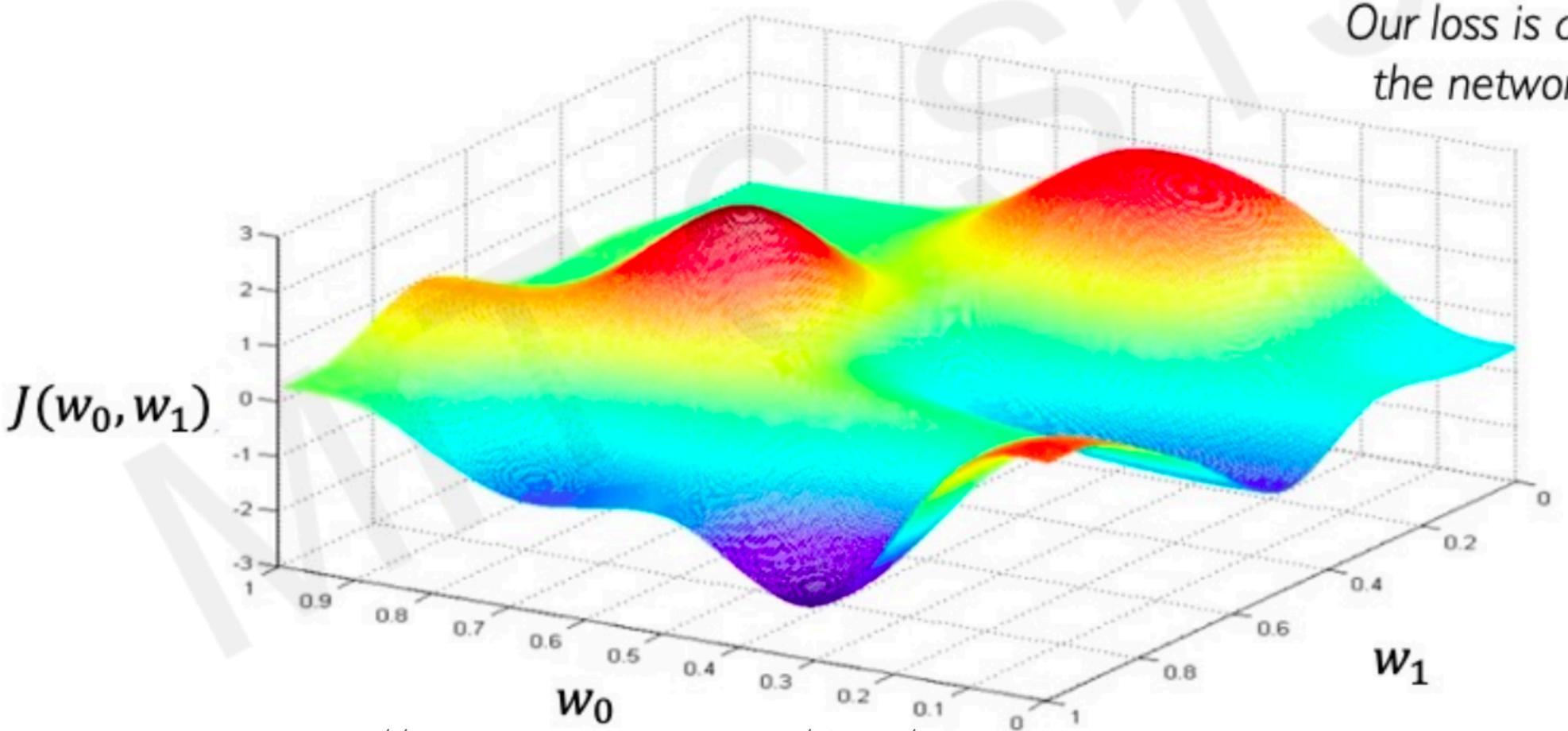
$$\boldsymbol{W} = \left\{\boldsymbol{W}^{(0)}, \boldsymbol{W}^{(1)}, \dots\right\}$$

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Loss Optimization

$$W^* = \underset{W}{\operatorname{argmin}} J(W)$$

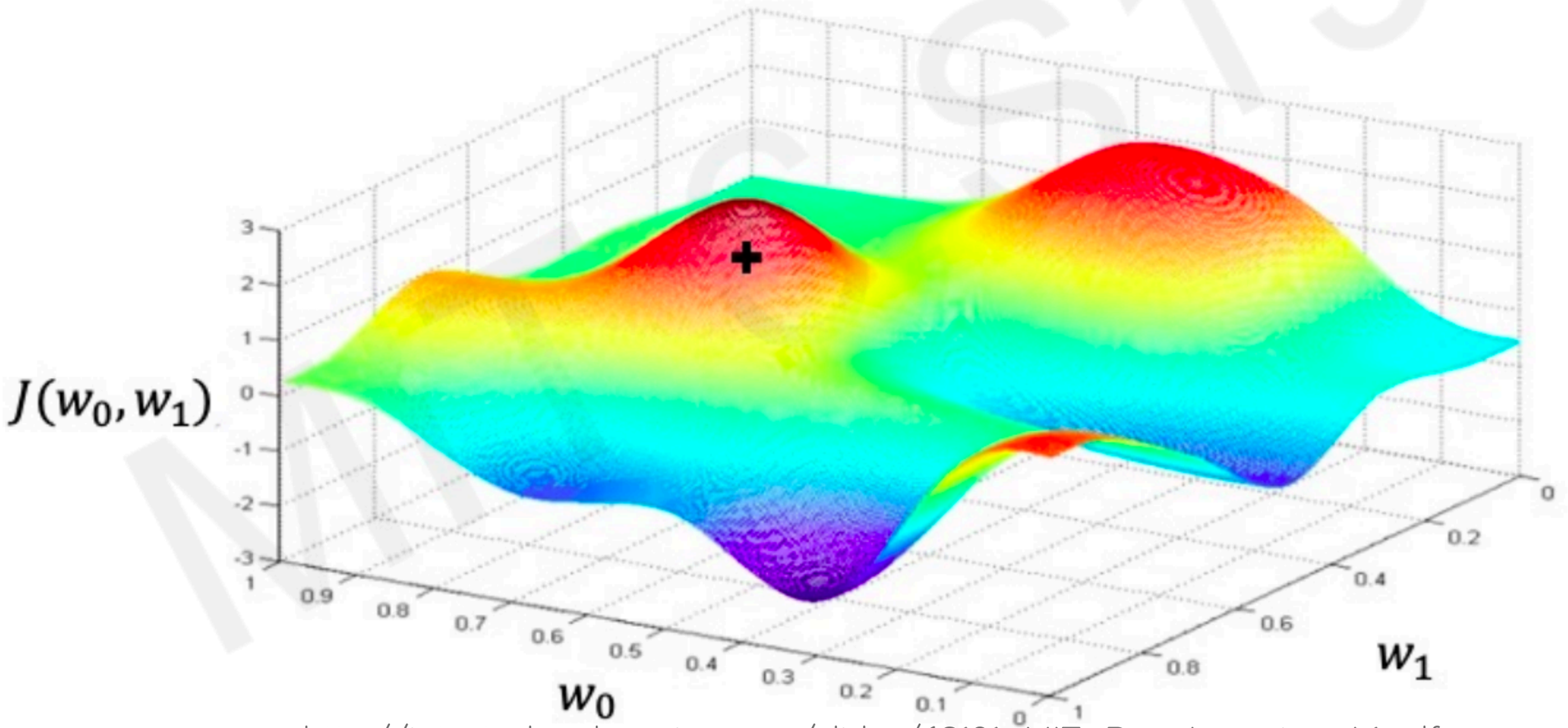
Remember:
Our loss is a function of the network weights!



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Loss Optimization

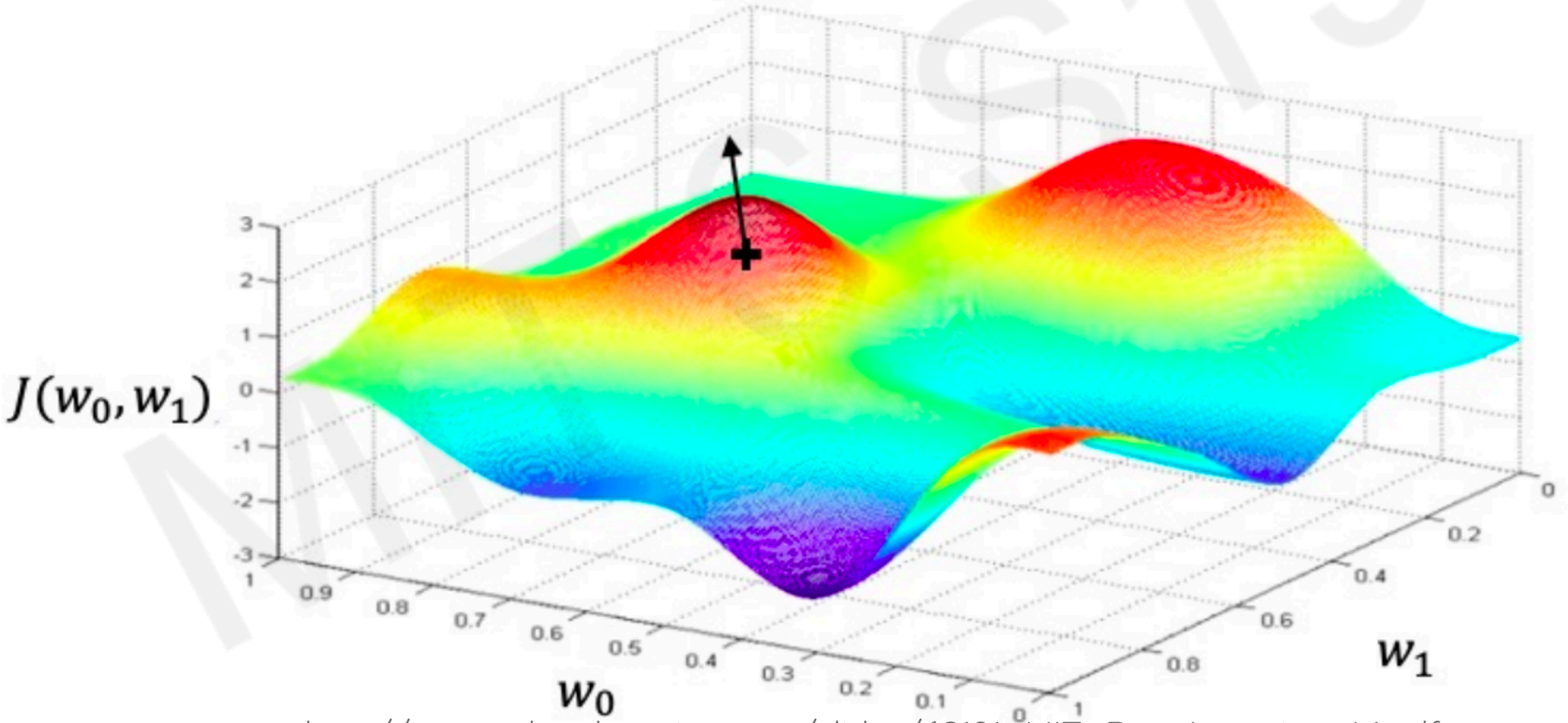
Randomly pick an initial (w_0, w_1)



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Loss Optimization

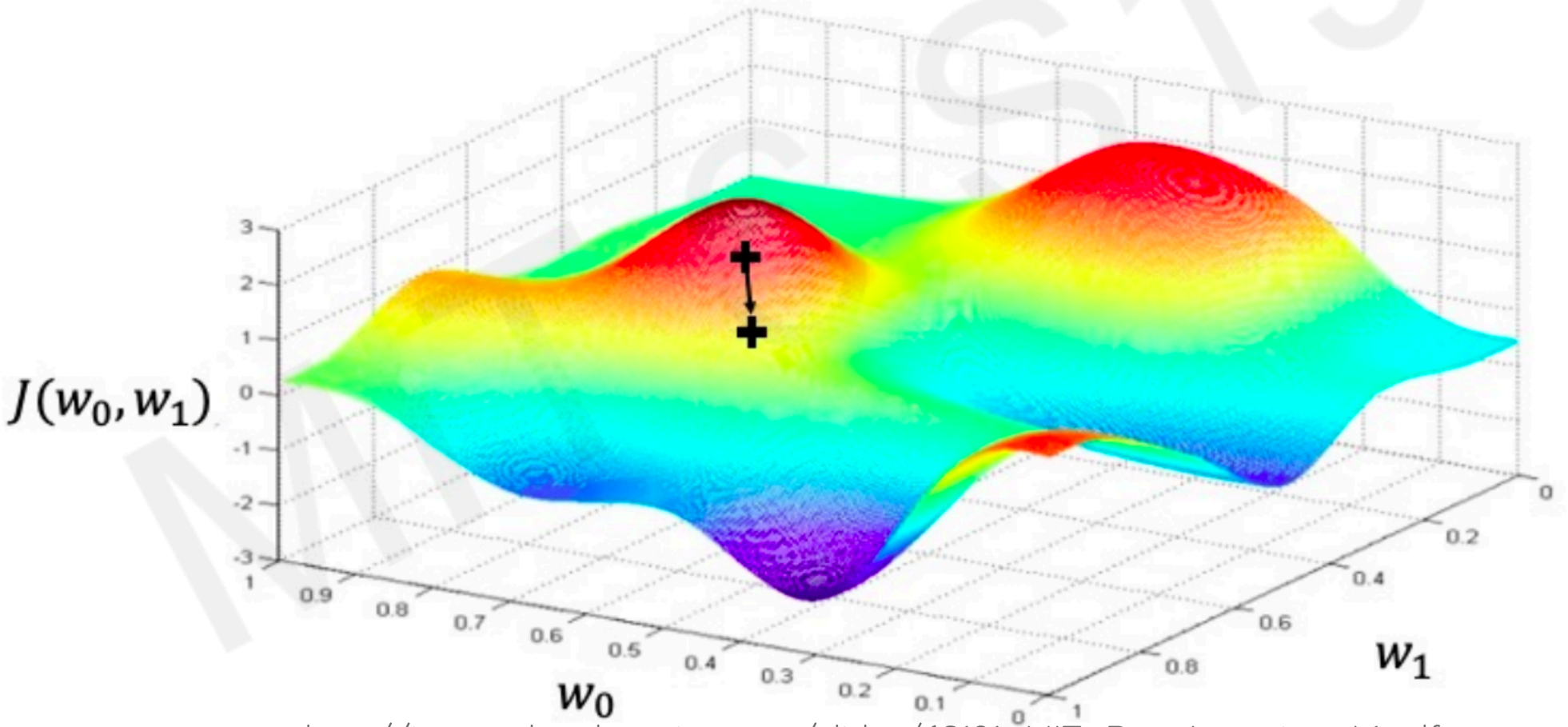
Compute gradient, $\frac{\partial J(W)}{\partial W}$



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Loss Optimization

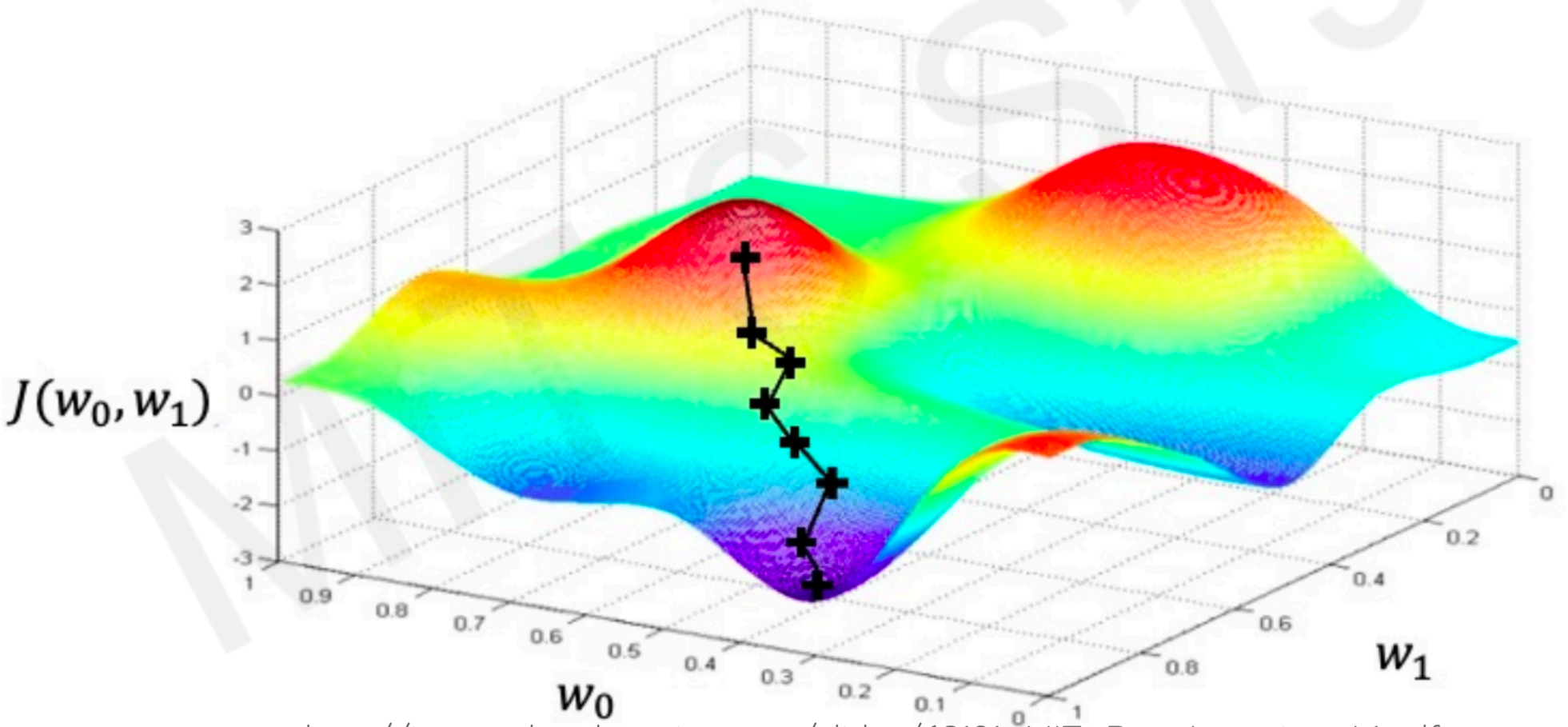
Take small step in opposite direction of gradient



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Gradient Descent

Repeat until convergence



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

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

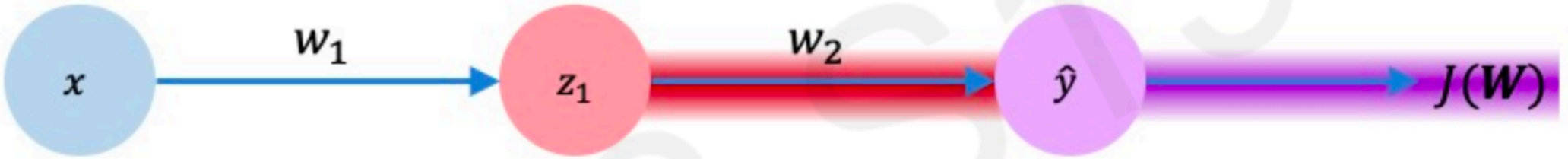
http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Computing Gradients: Backpropagation

How does a small change in one weight (ex. $\boldsymbol{w_2}$) affect the final loss $\boldsymbol{J(W)}$?

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

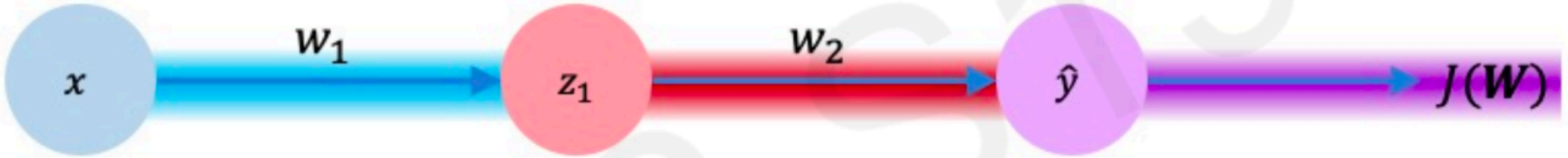
Computing Gradients: Backpropagation



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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Computing Gradients: Backpropagation



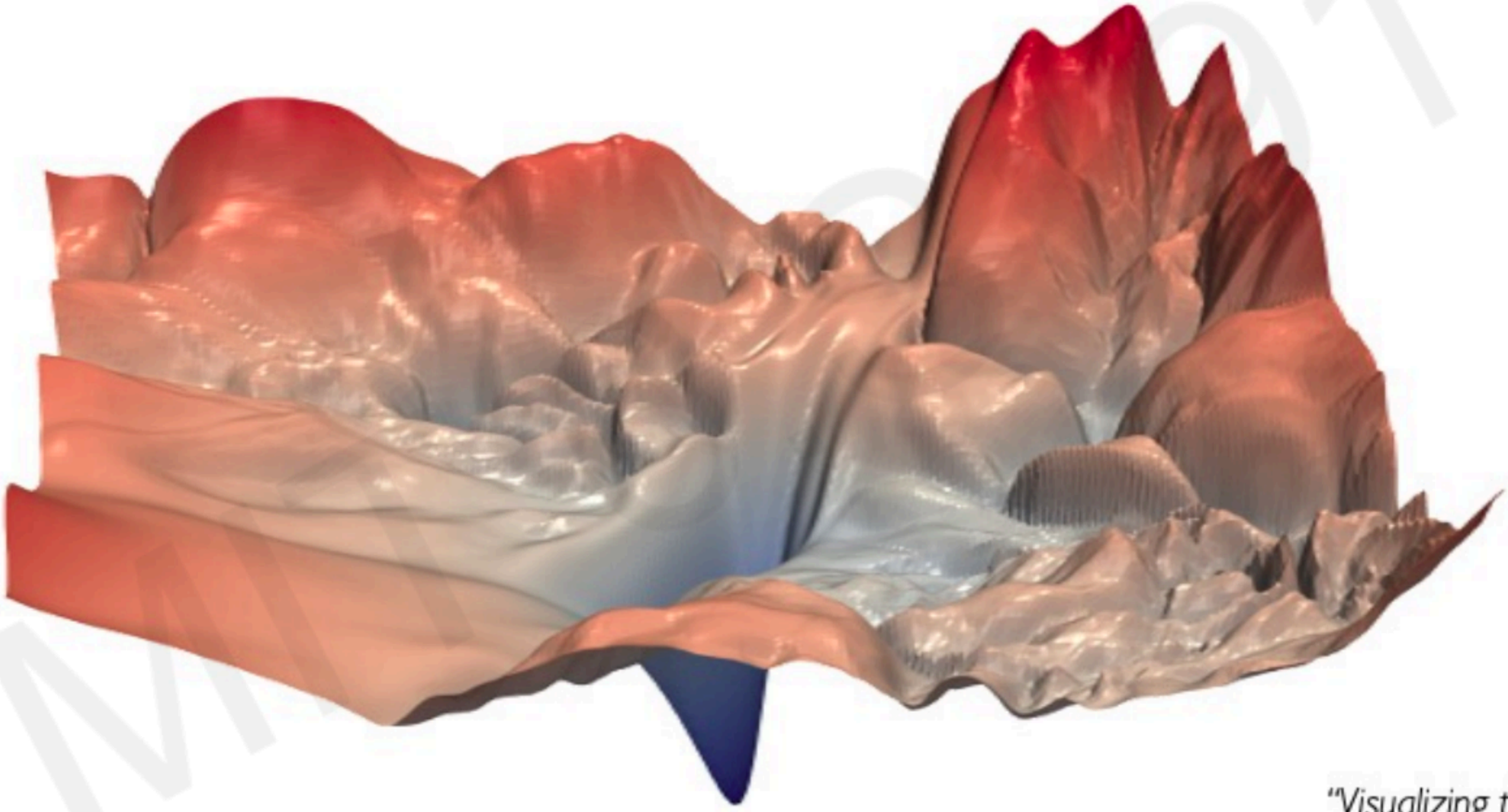
$$\frac{\partial J(\boldsymbol{W})}{\partial w_1} = \frac{\partial J(\boldsymbol{W})}{\partial \hat{y}} * \frac{\partial \hat{y}}{\partial z_1} * \frac{\partial z_1}{\partial w_1}$$

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Questions Break

Training Neural Networks is Difficult

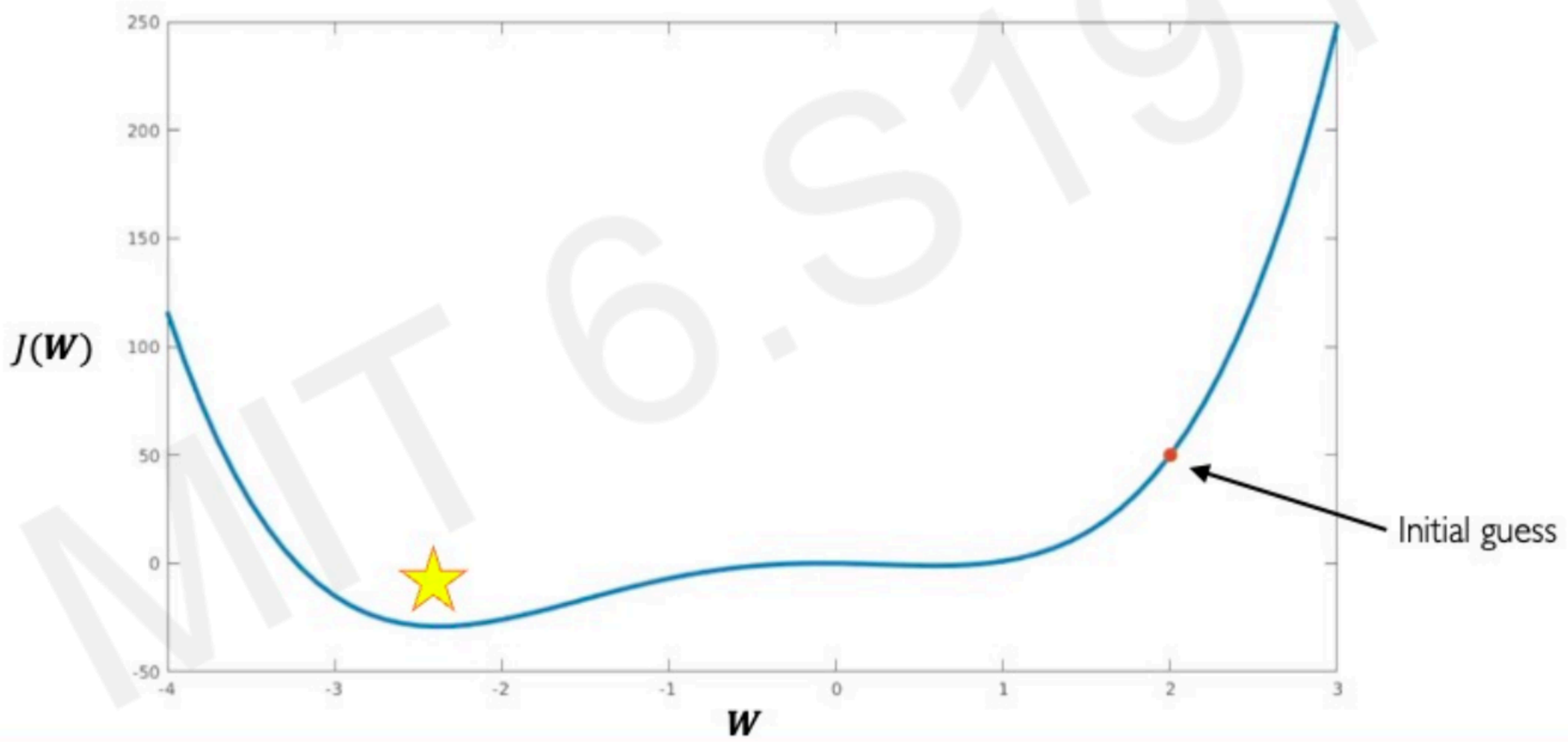


"Visualizing the loss landscape of neural nets". Dec 2017.

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Setting the Learning Rate

- Small learning rate converges slowly and gets stuck in false local minima
- Large learning rates overshoot, become unstable and diverge
- Stable learning rates converge smoothly and avoid local minima



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

How to deal with this?

- One approach is to try various learning rates and see what works “just right”
- A smarter way would be to adjust the learning rate so that it “adapts” to the landscape
- The learning rate can be made smaller/larger based on:
  - How large the gradient is
  - How fast the model is converging
  - Magnitudes of weights
  - and so on..

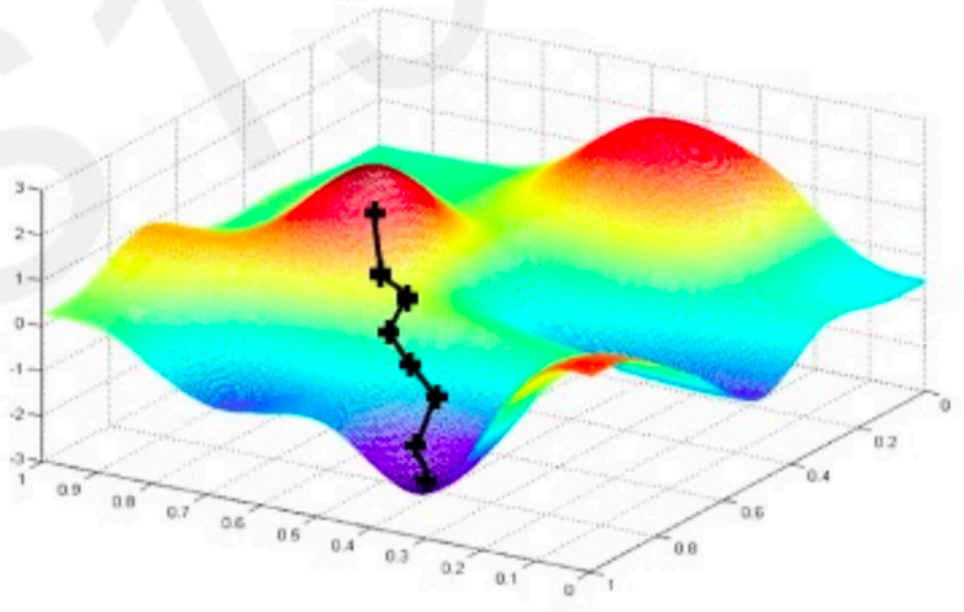
Commonly used Gradient Descent algorithms

- SGD
- Adam
- Adadelta
- Adagrad
- RMSProp

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



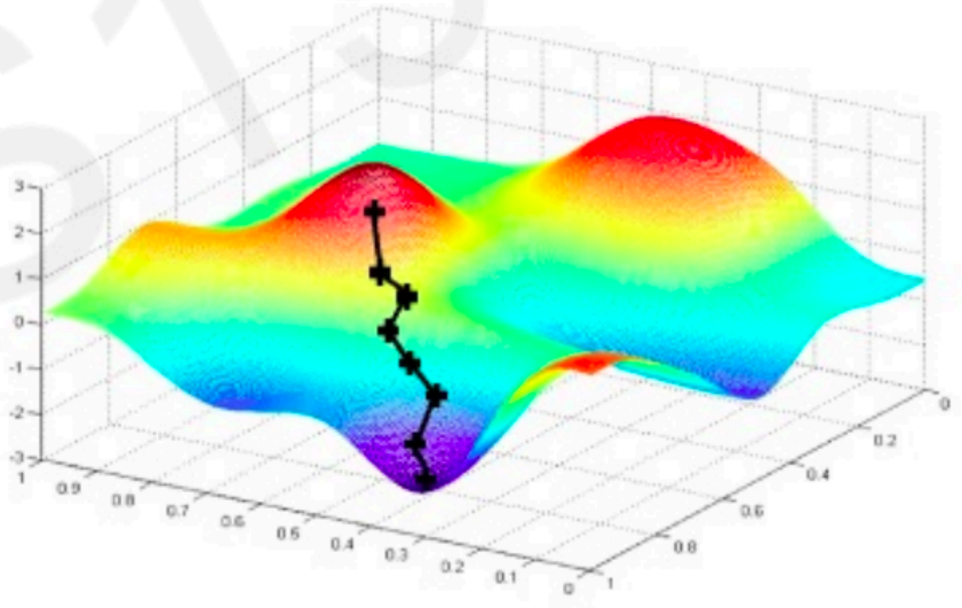
Can be very **computationally intensive** to compute!

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Stochastic Gradient Descent

Algorithm

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



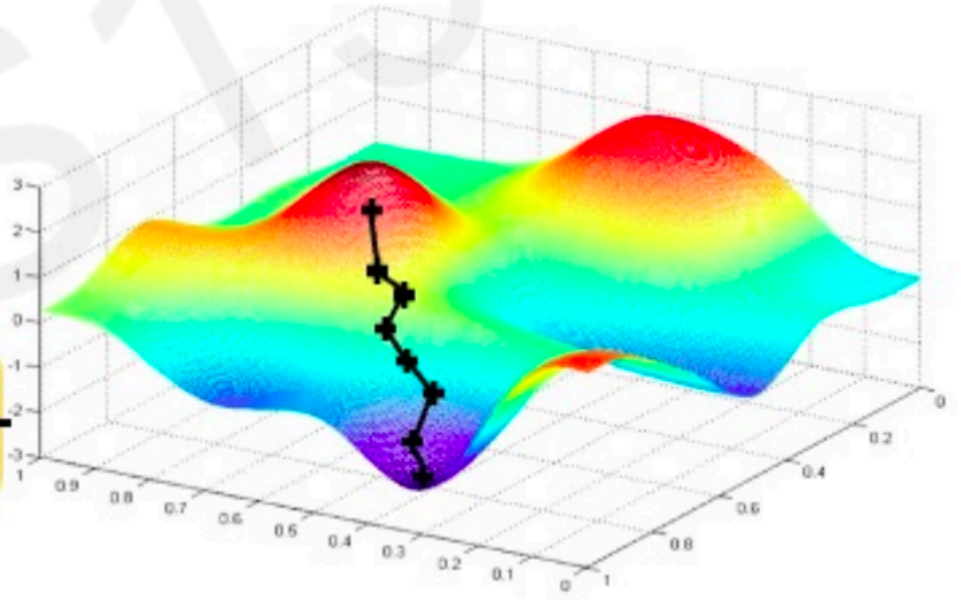
Easy to compute but
very noisy (stochastic)!

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B} \sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. Update weights, $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
6. Return weights



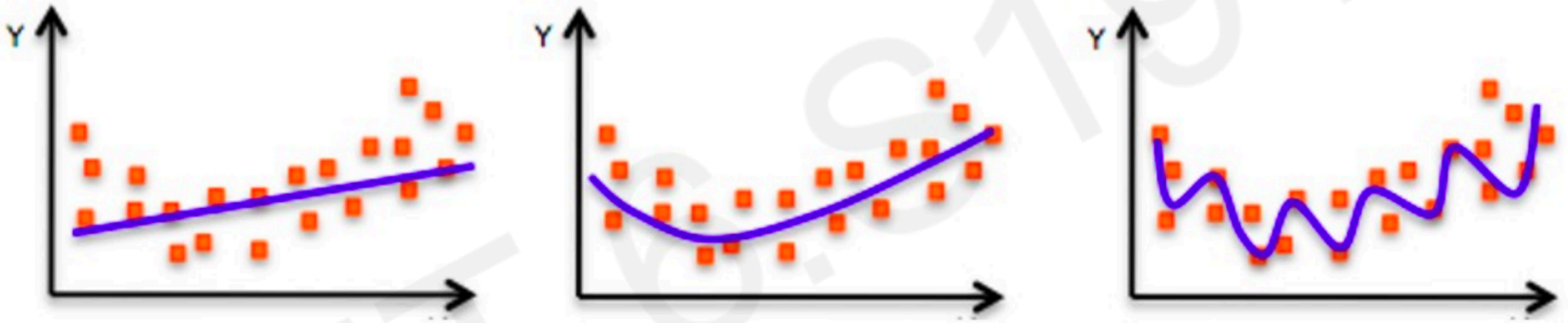
Fast to compute and a much better estimate of the true gradient!

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Advantages of Mini-Batches

- Enables more accurate estimation of gradient compared to SGD
  - Smoother convergence - Why?
    - A single sample can also be an outlier and can throw off the descent process
    - This is avoided by the mini batch which ensures that most of the contribution to the gradient computation comes from inliers
  - Allows for higher learning rates
- Enables parallelization of computation
  - Each sample in the mini batch doesn't influence the computations of other samples
- Gives significant performance gains on GPUs that especially benefit from parallel computations

The Problem of Overfitting



Underfitting
Model does not have capacity to fully learn the data

Ideal fit

Overfitting
Too complex, extra parameters, does not generalize well

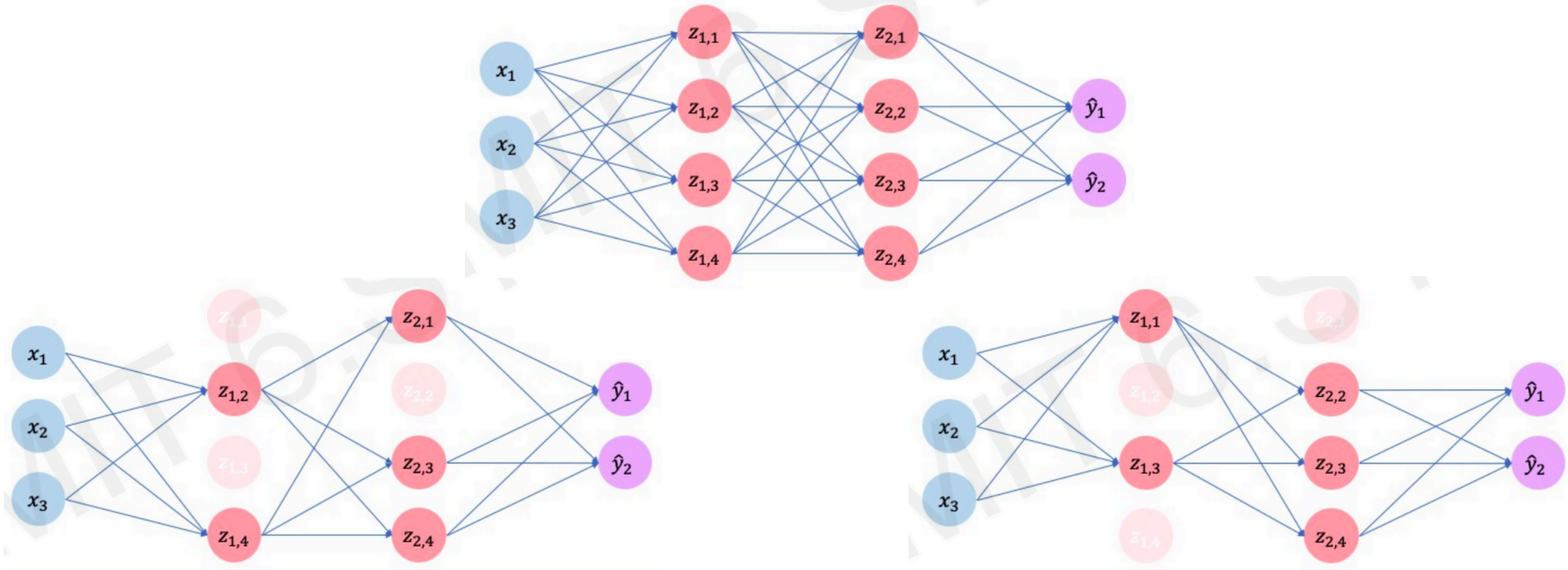
http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Regularization

- It is a technique that constrains our optimization problem to discourage complex models
- We need it to improve generalization of our model on unseen data and not let it learn the patterns of only the provided training data
- Commonly used methods of regularization are:
  - Dropout
  - Early stopping

Dropout

- During training, randomly set activation of some neurons to 0
- Forces the network to not rely on any single node



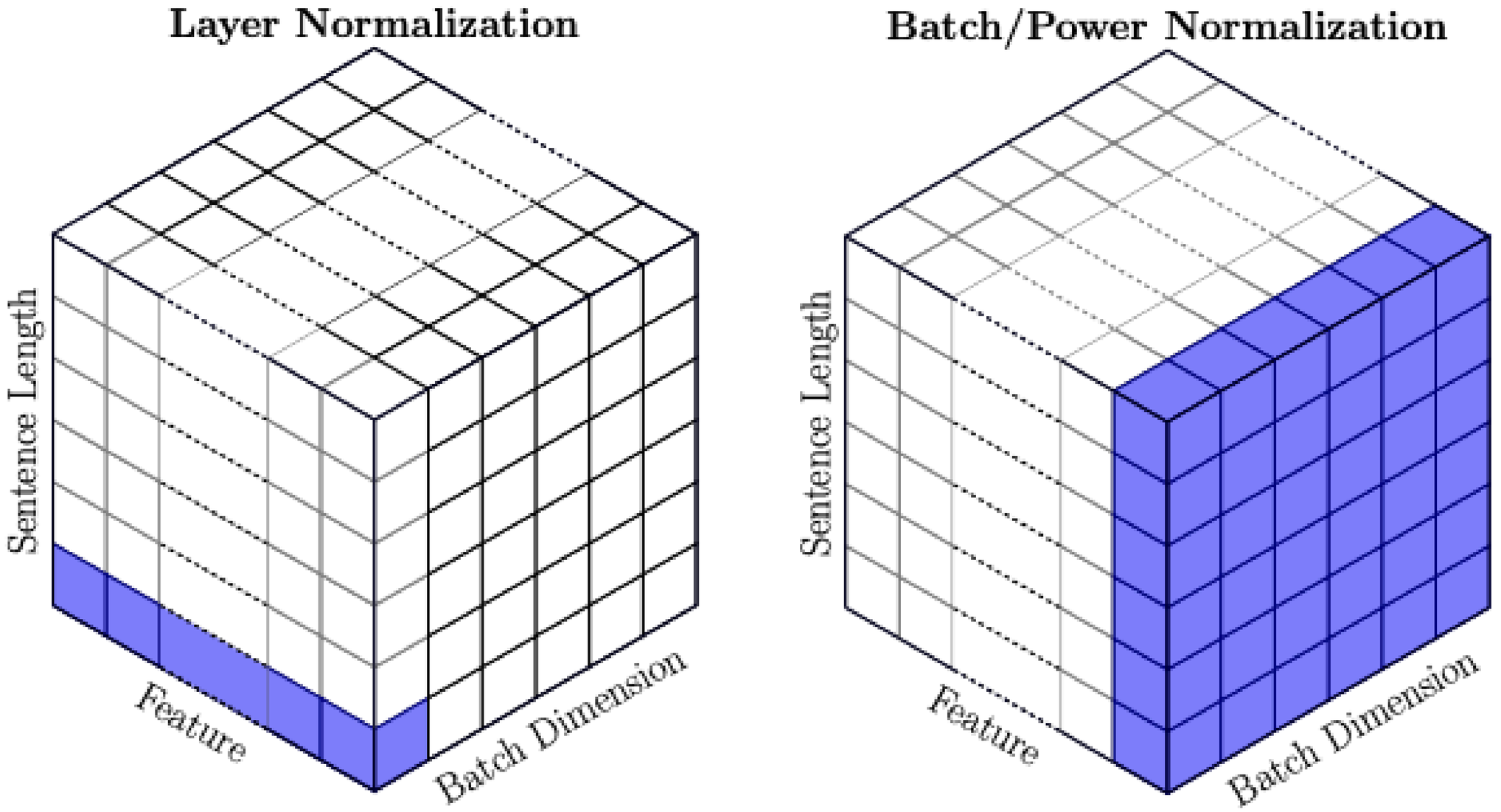
http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Early stopping

- Stop training before the model leans towards overfitting

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Normalization for stable training

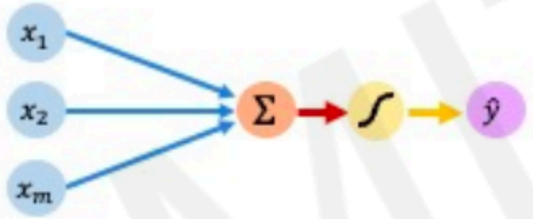


https://stats.stackexchange.com/a/505349

Core Foundation Review

The Perceptron

- Structural building blocks
- Nonlinear activation functions

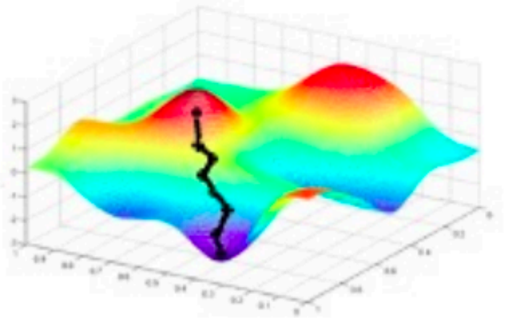


Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation

Training in Practice

- Adaptive learning
- Batching
- Regularization
- Normalization

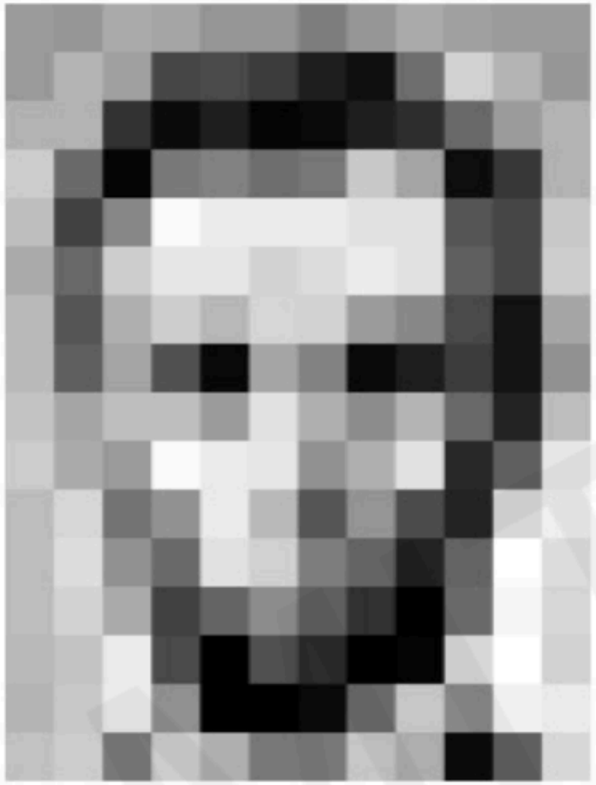


http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L1.pdf

Questions Break

Convolutional Neural Networks (CNNs)

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

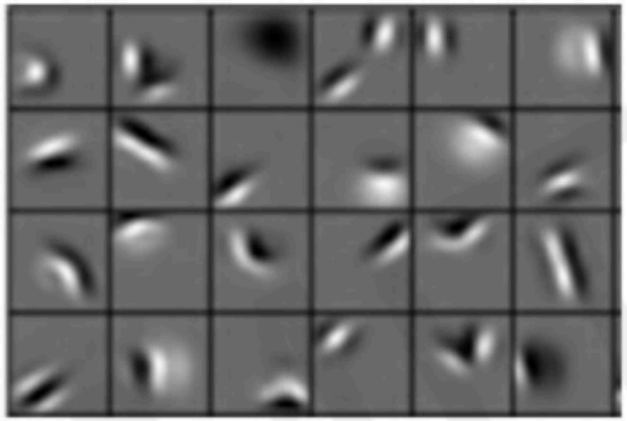
Manual Feature Extraction

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

Learning Feature Representations

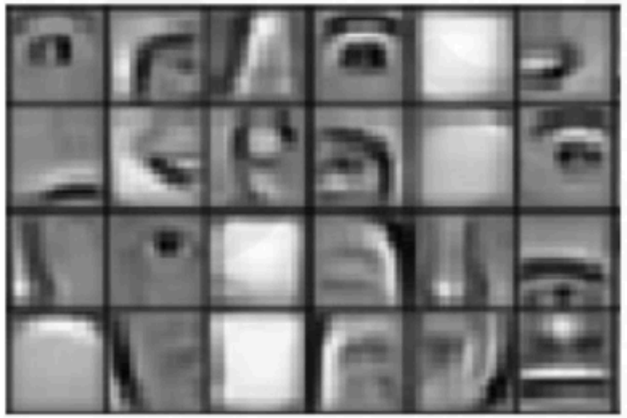
Can we learn a **hierarchy of features** directly from the data instead of hand engineering?

Low level features



Edges, dark spots

Mid level features



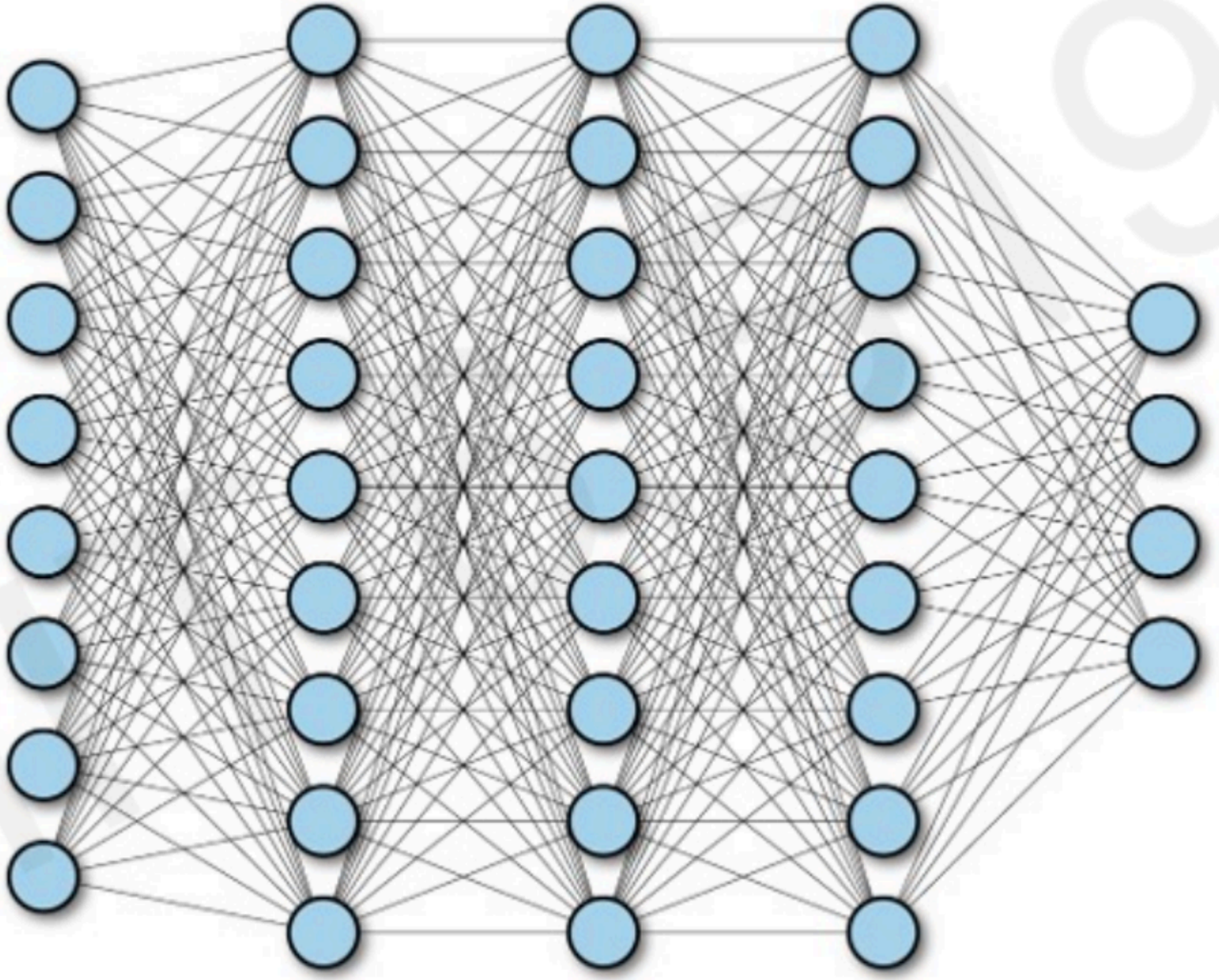
Eyes, ears, nose

High level features

Facial structure

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

Fully Connected Neural Network

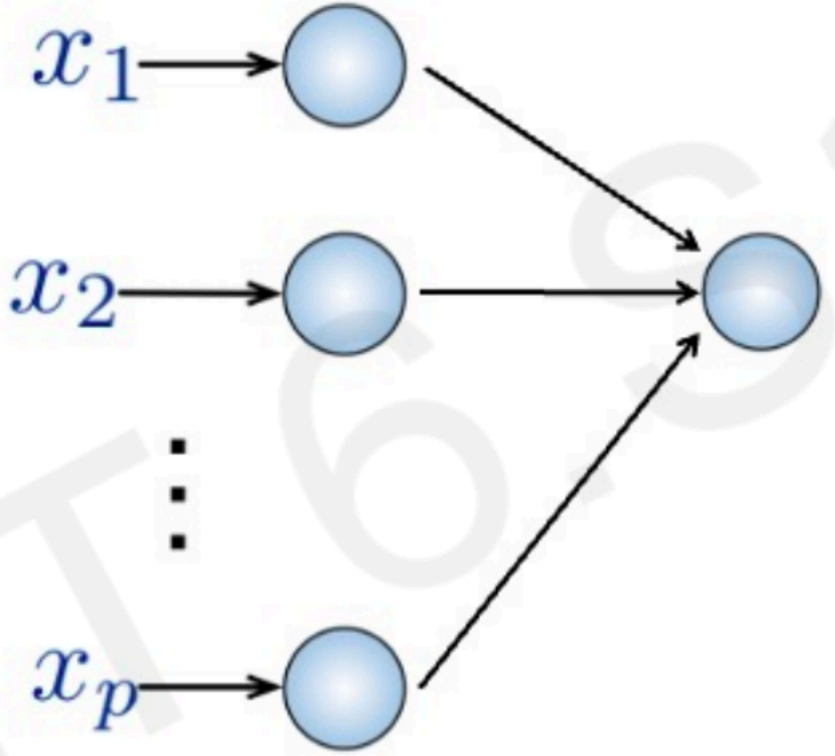


http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

Fully Connected Neural Network

Input:

- 2D image
- Vector of pixel values



Fully Connected:

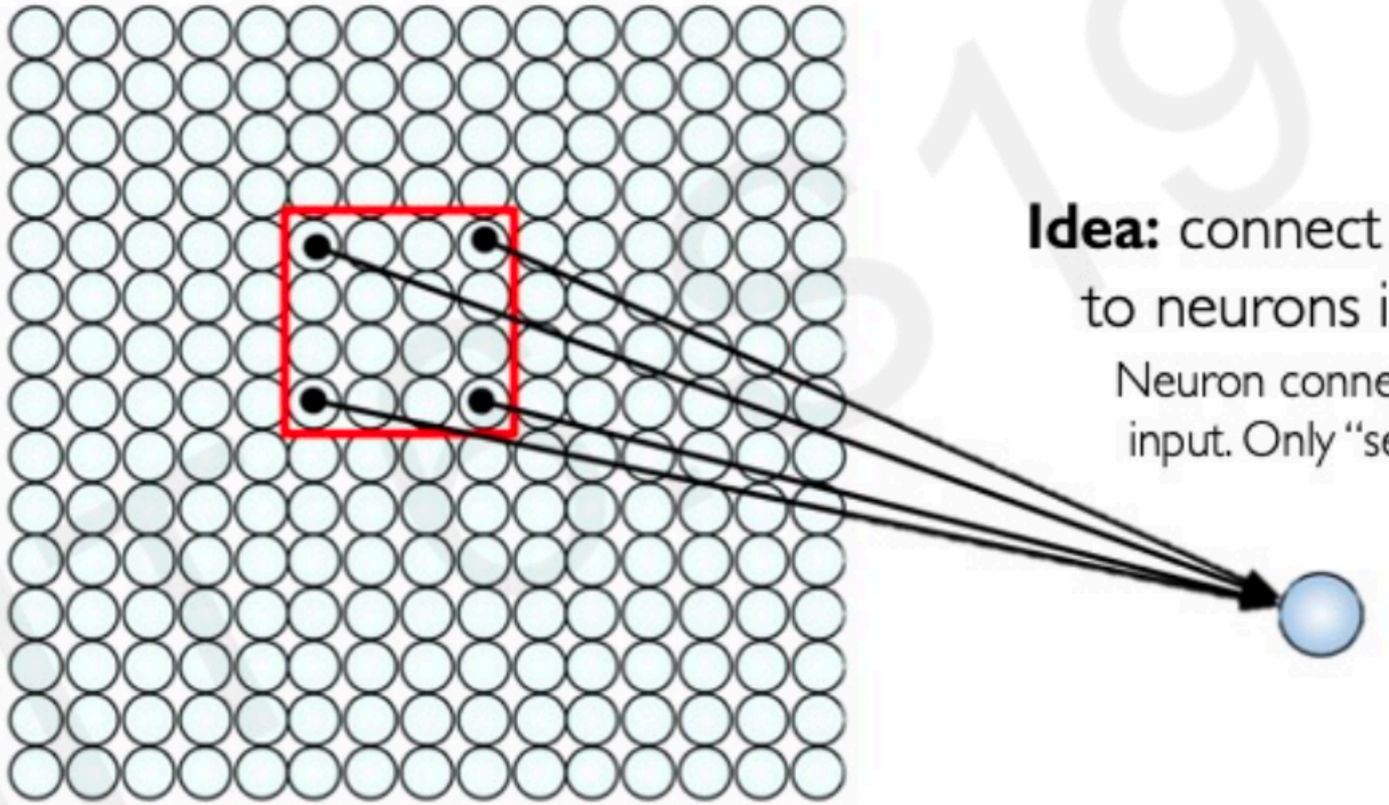
- Connect neuron in hidden layer to all neurons in input layer
- No spatial information!
- And many, many parameters!

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

Using Spatial Structure

Input: 2D image.
Array of pixel values

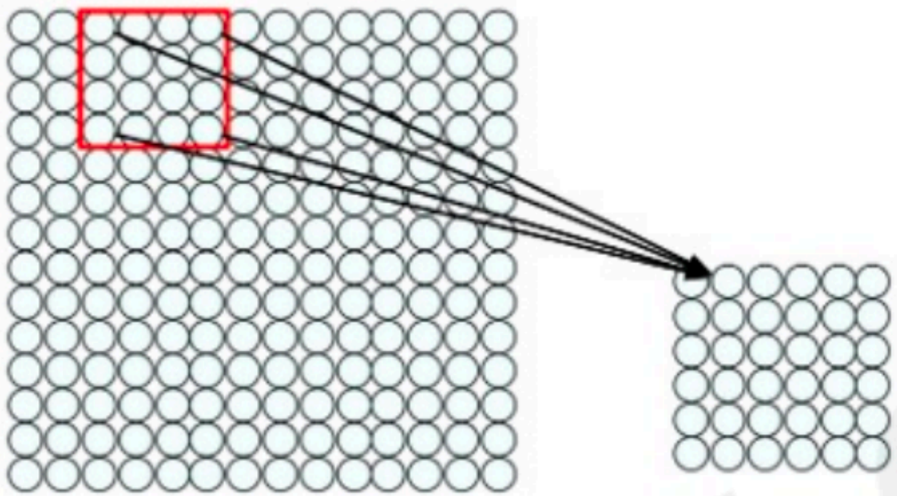


Idea: connect patches of input to neurons in hidden layer.

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

The Convolution Operation

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

image

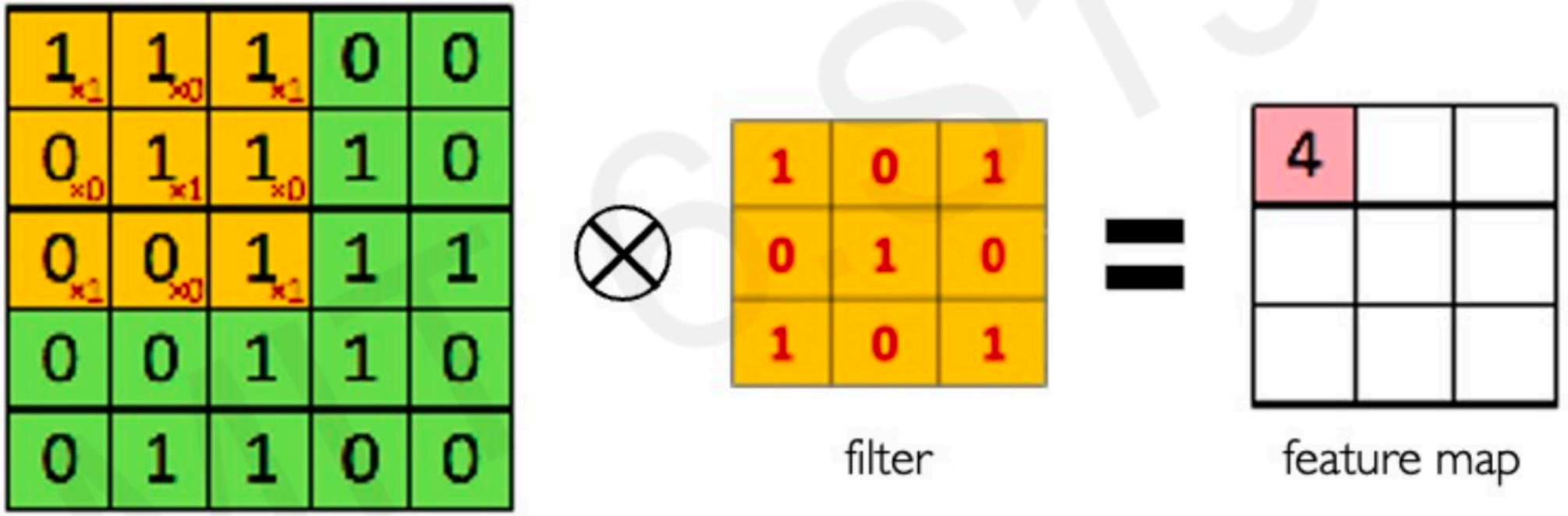
filter

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

The Convolution Operation

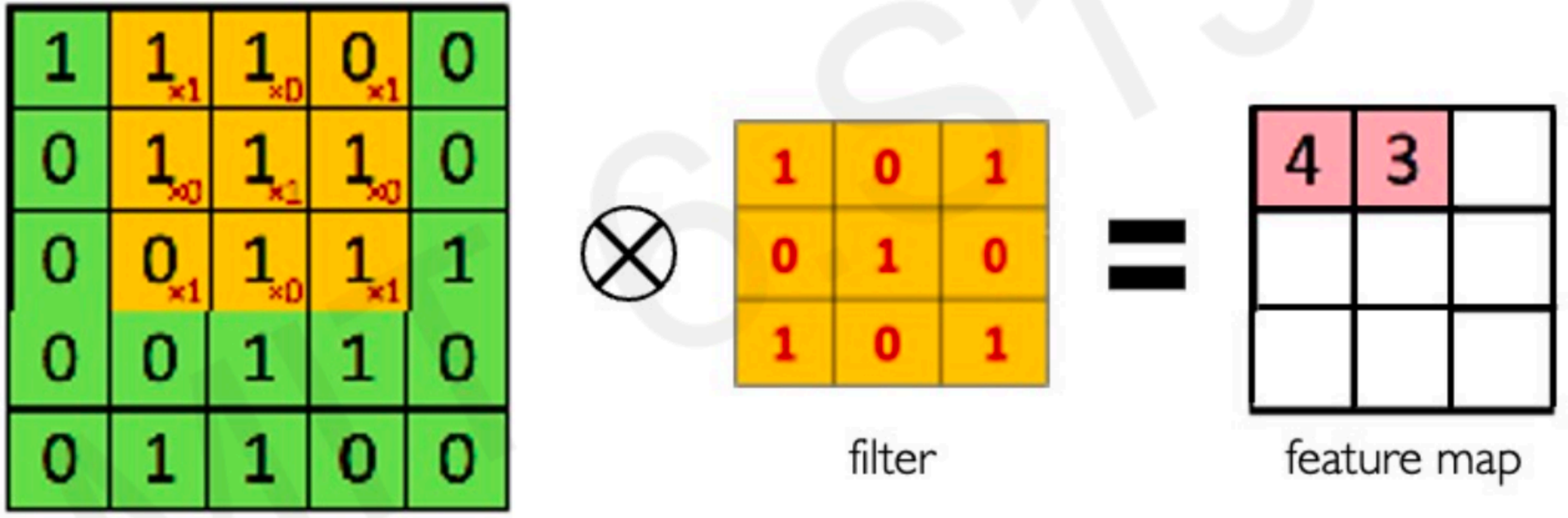
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

The Convolution Operation

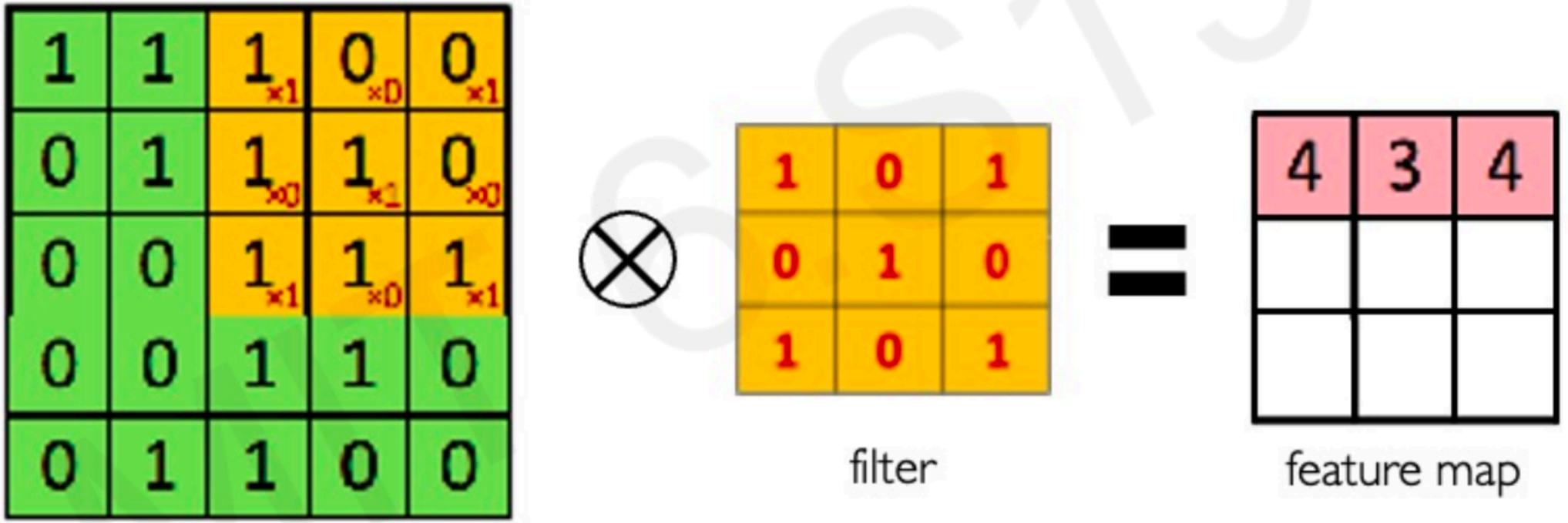
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

The Convolution Operation

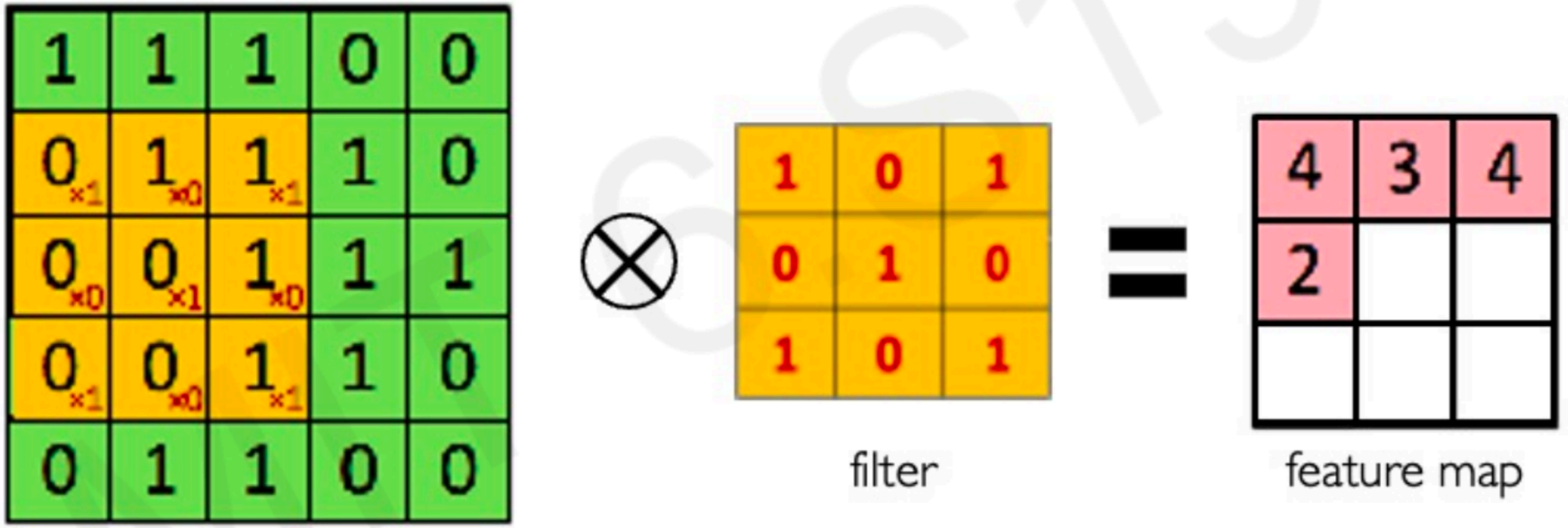
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

The Convolution Operation

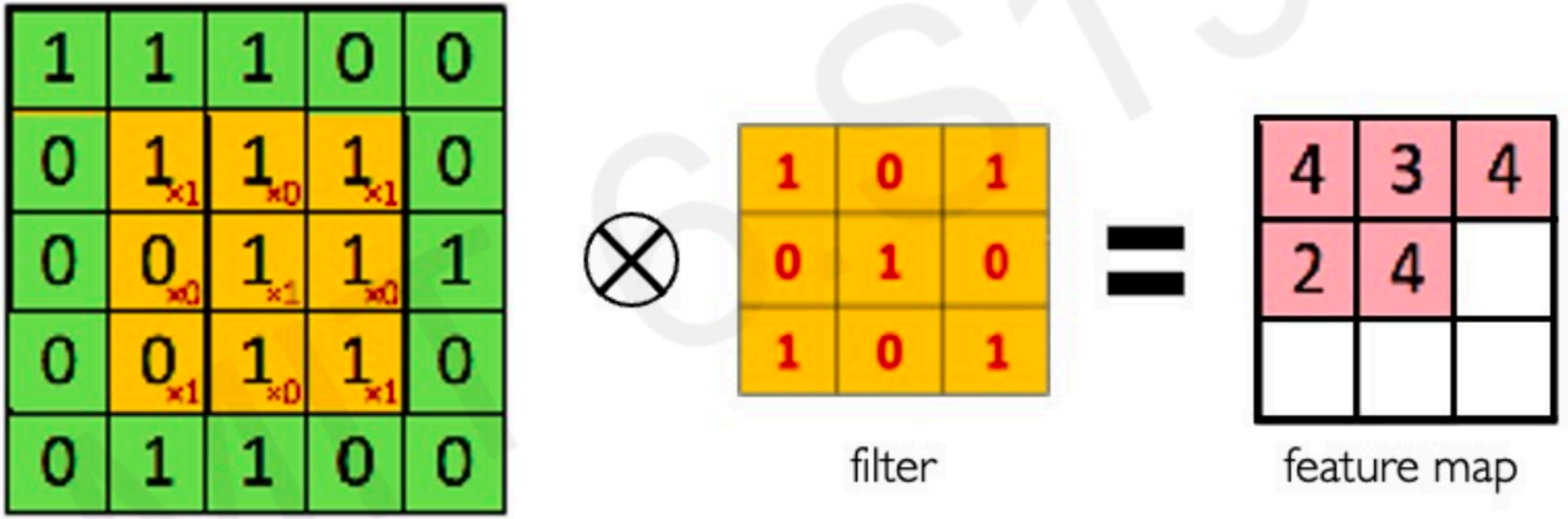
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

The Convolution Operation

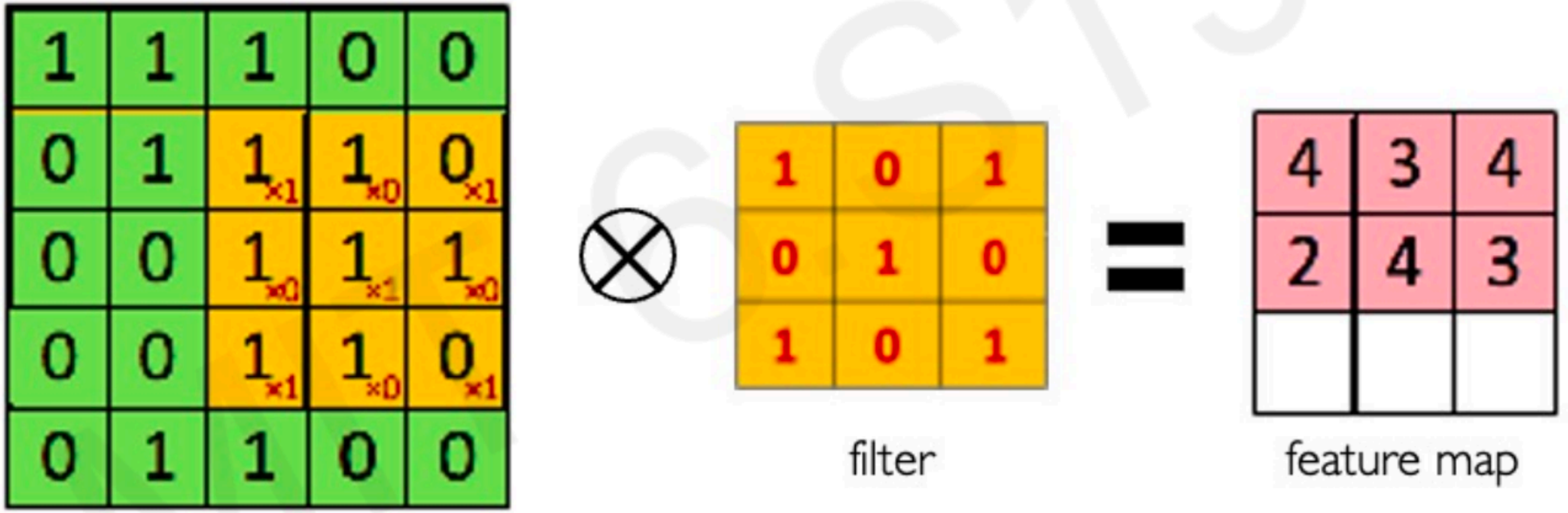
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

The Convolution Operation

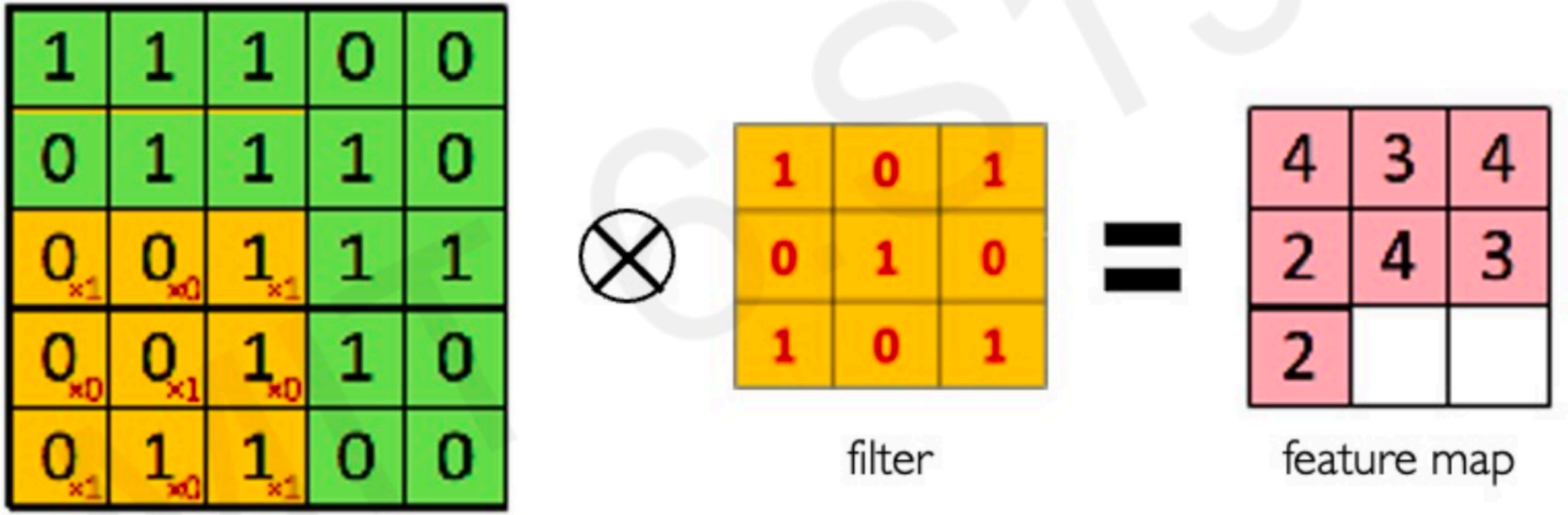
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

The Convolution Operation

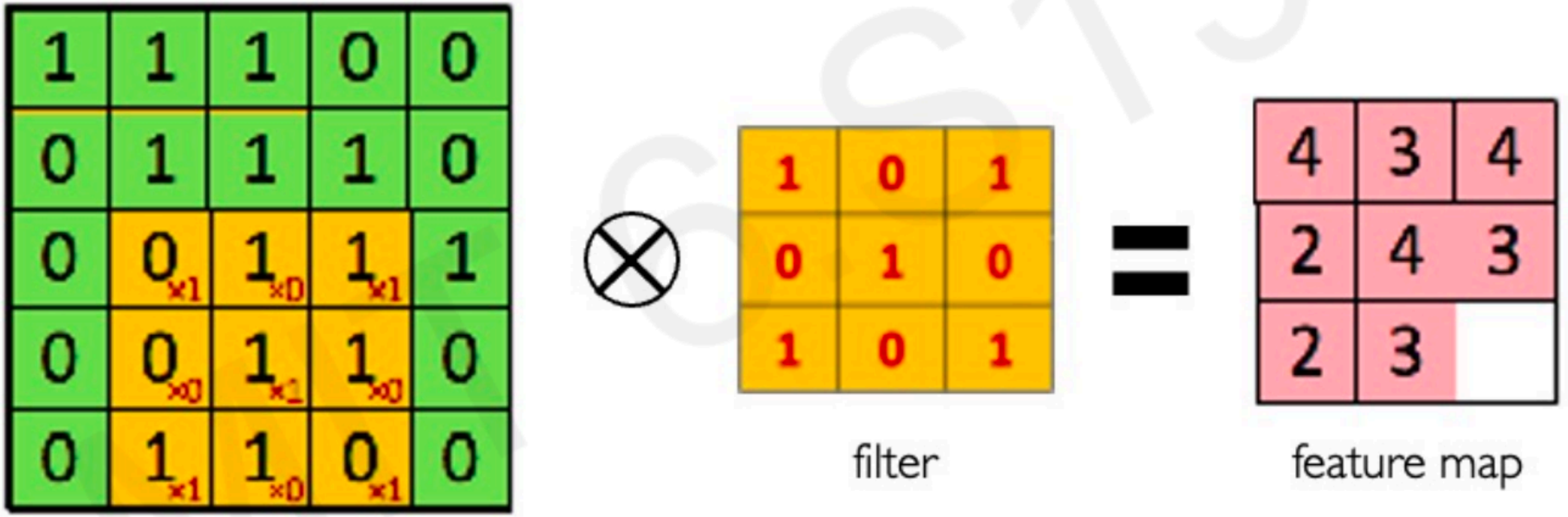
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

The Convolution Operation

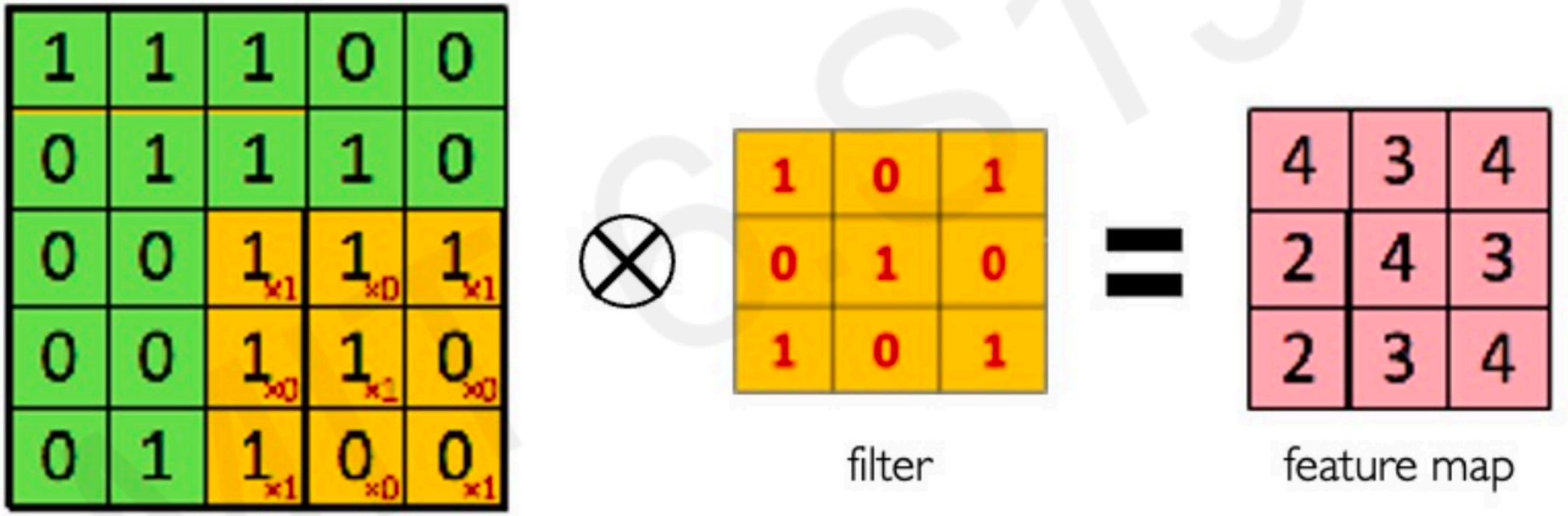
We slide the 3x3 filter over the input image, element-wise multiply, and add the outputs:



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

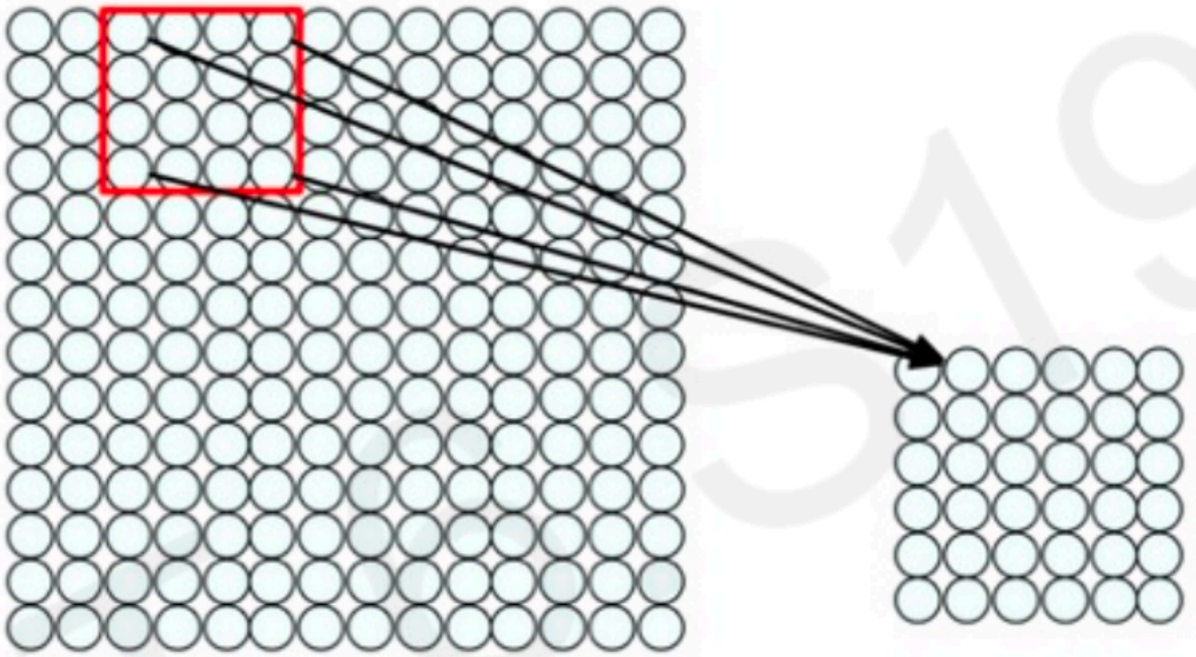
The Convolution Operation

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



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

Feature Extraction with Convolution



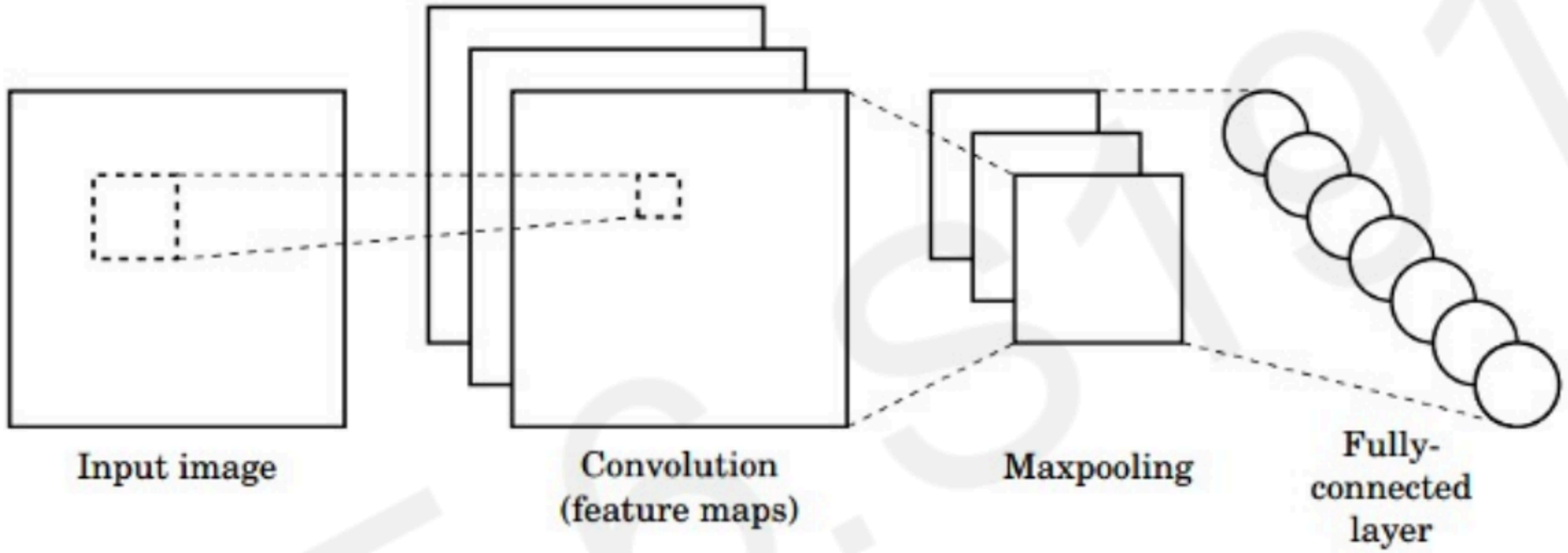
1) Apply a set of weights – a filter – to extract **local features**

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

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

CNNs for Classification

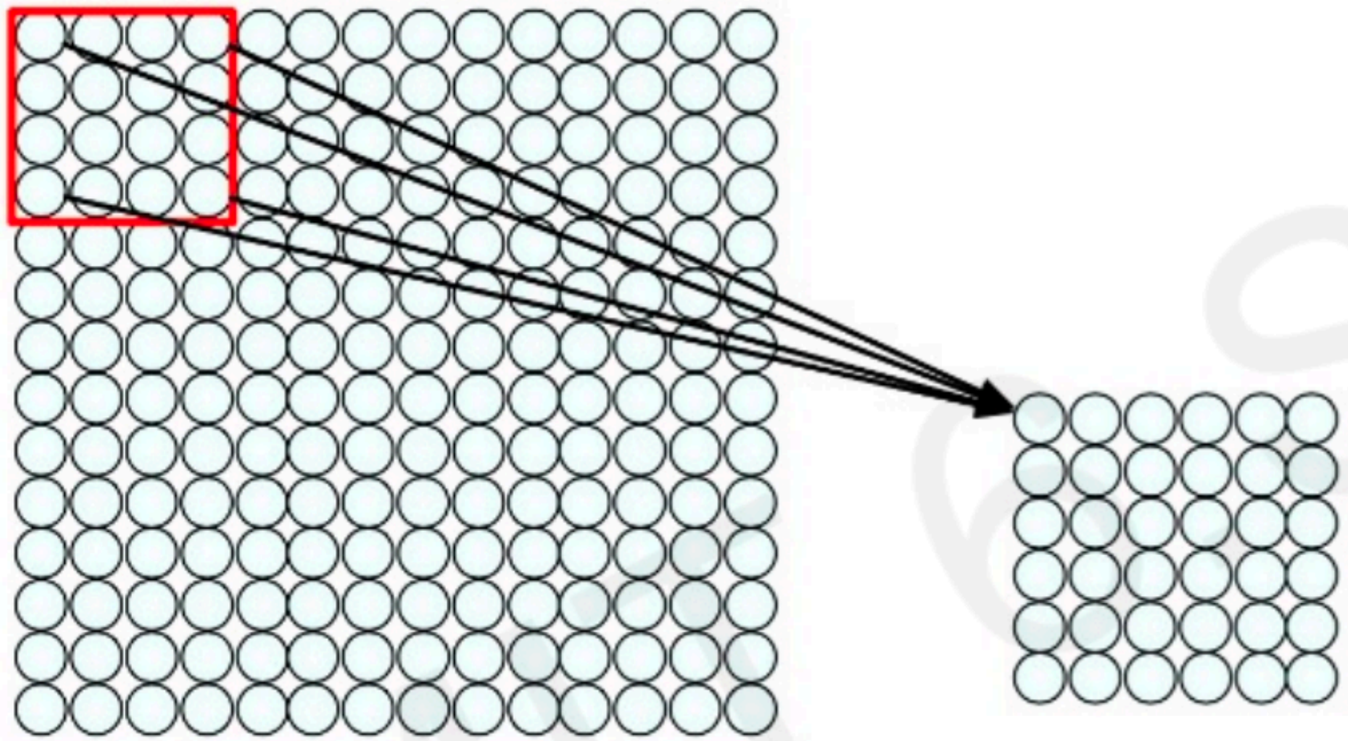


1. **Convolution**: Apply filters to generate feature maps.
2. **Non-linearity**: Often ReLU.
3. **Pooling**: Downsampling operation on each feature map.

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

Convolutional Layers: Local Connectivity



For a neuron in hidden layer:

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

4x4 filter: matrix of weights w_{ij}

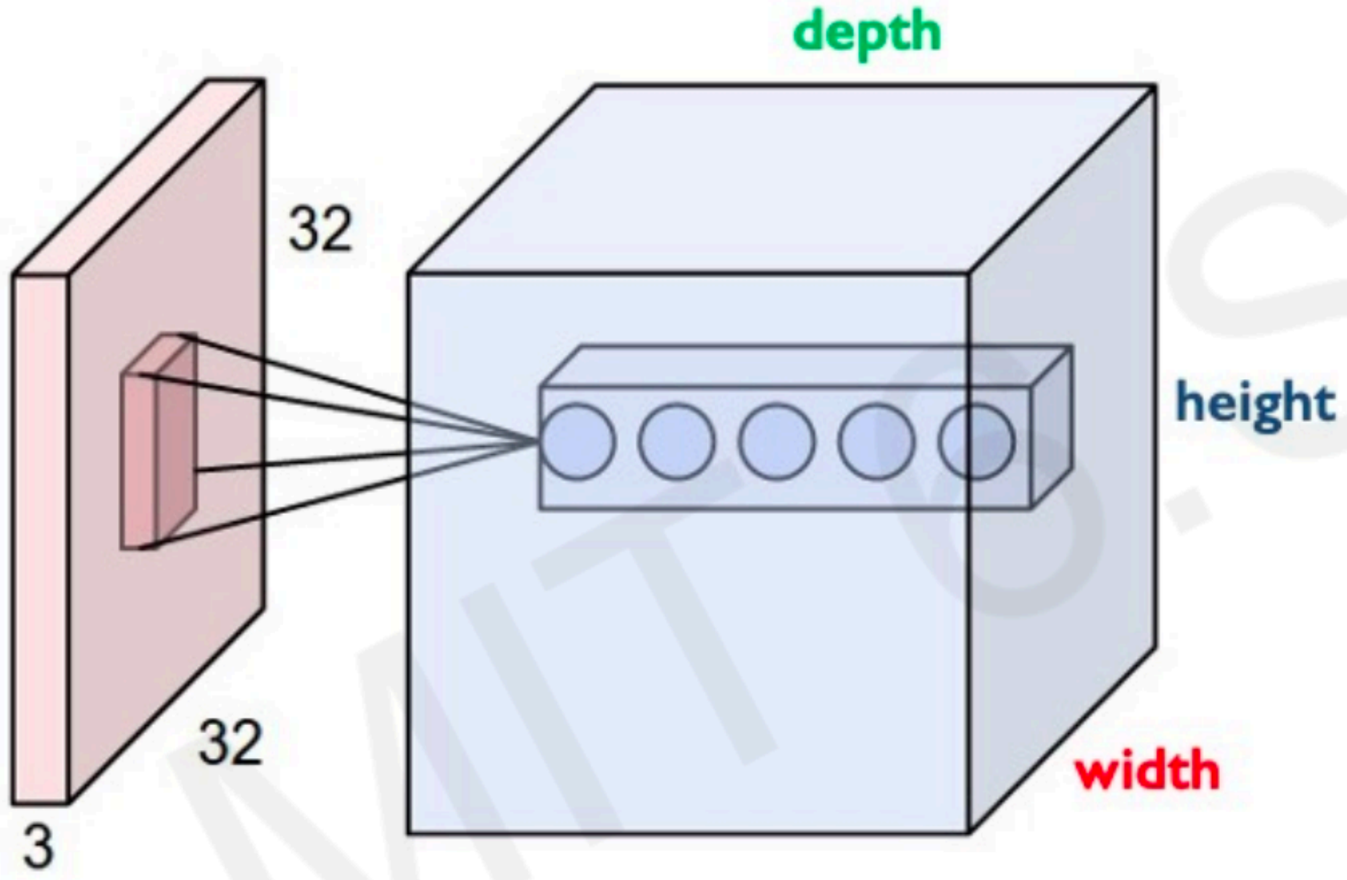
$$\sum_{i=1}^{4}\sum_{j=1}^{4} w_{ij}\, x_{i+p,j+q} + b$$

for neuron (p,q) in hidden layer

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

CNNs: Spatial Arrangement of Output Volume



Layer Dimensions:

$h \times w \times d$

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

Stride:

Filter step size

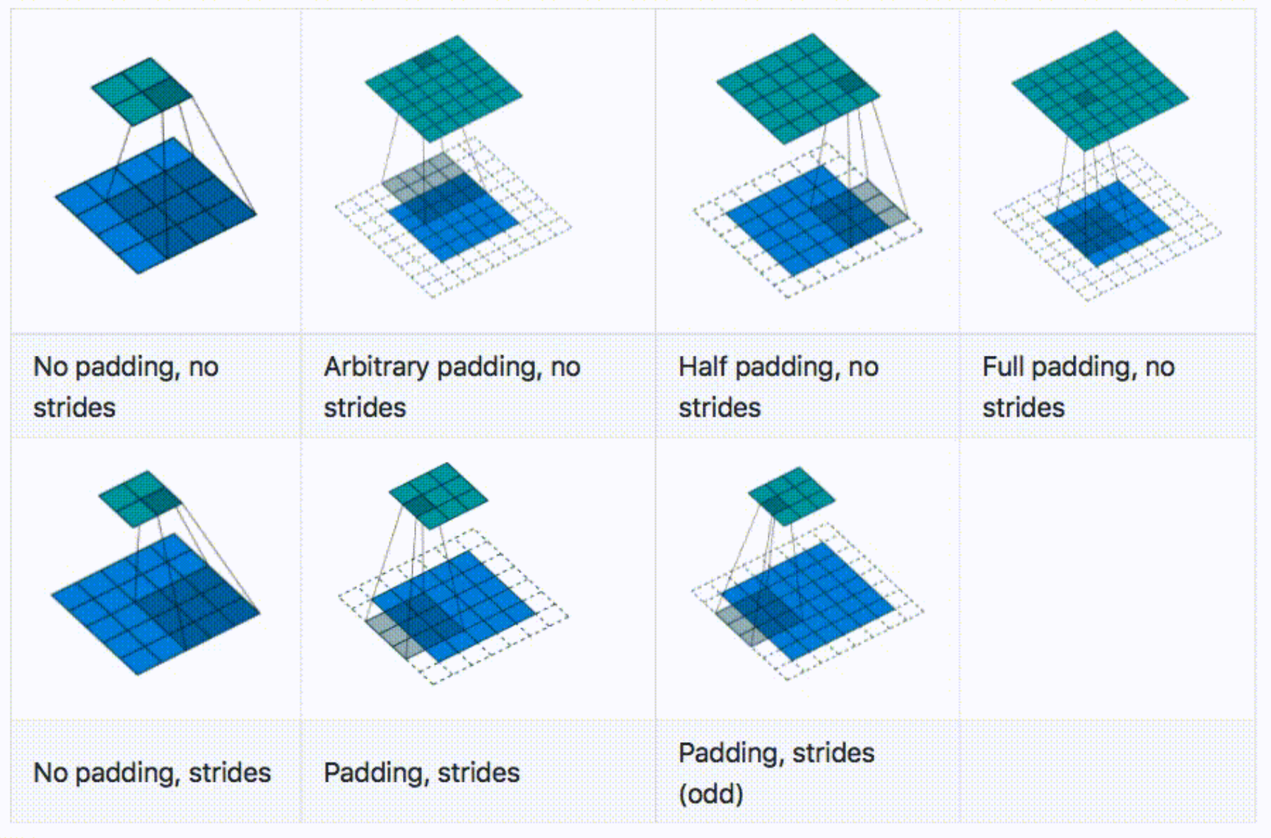
Receptive Field:

Locations in input image that a node is path connected to

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

Padding and Stride

- Padding “adds” zeros around the feature map to reduce the amount by which the feature map shrinks after the convolution operation
- Stride is the number of pixels by which the filter/kernel shifts after each convolution operation



https://github.com/vdumoulin/conv_arithmetic

Output feature dimension

- Consider that we have an input feature map with dimensions W x W x D
- We also have D' filters/kernels each with dimensions F x F, padding P and stride S
- What will be the dimension of the output feature map W_out?

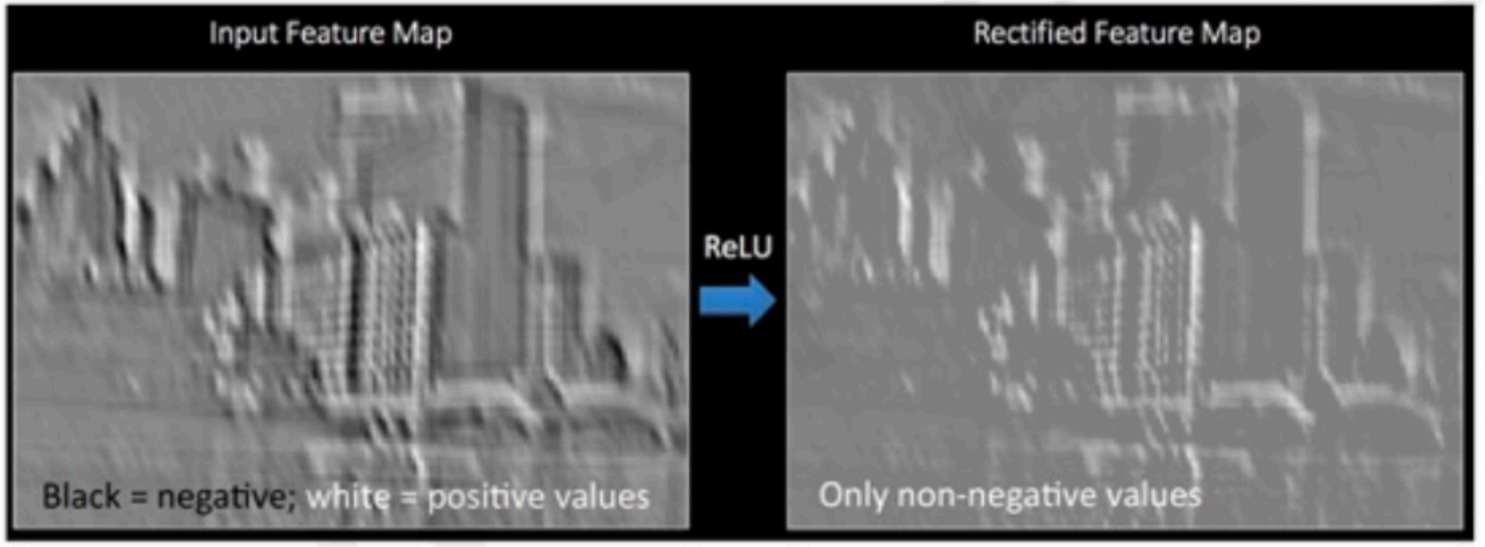
$$W_{out} = \frac{W - F + 2P}{S} + 1$$

https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939

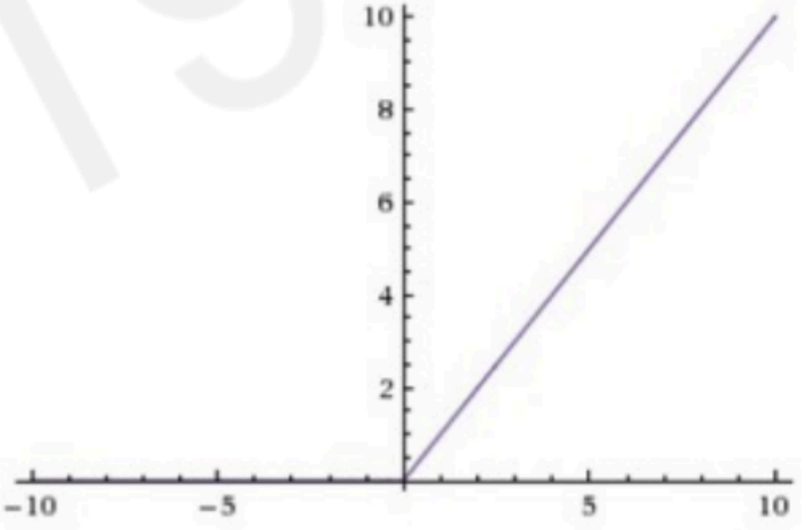
- Additionally, since there are D' filters that are being applied, the final output dimension is W_out x W_out x D'

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!**



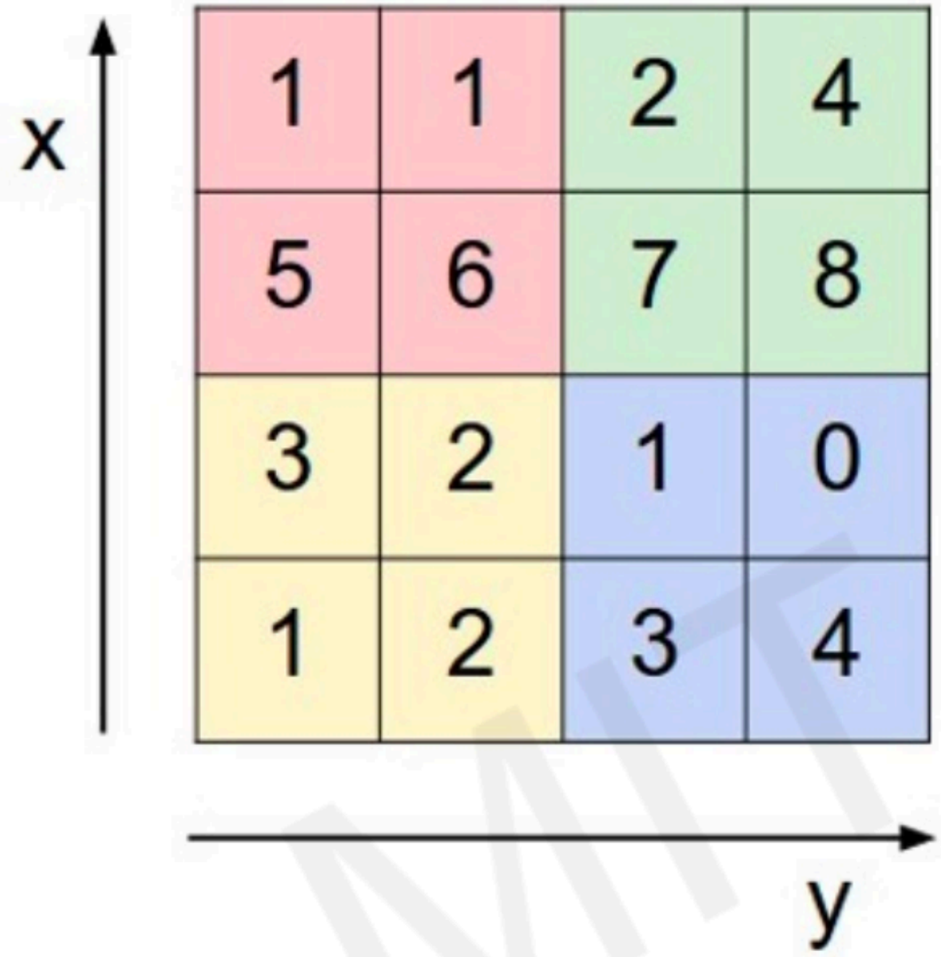
Rectified Linear Unit (ReLU)



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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

Pooling



max pool with 2x2 filters
and stride 2

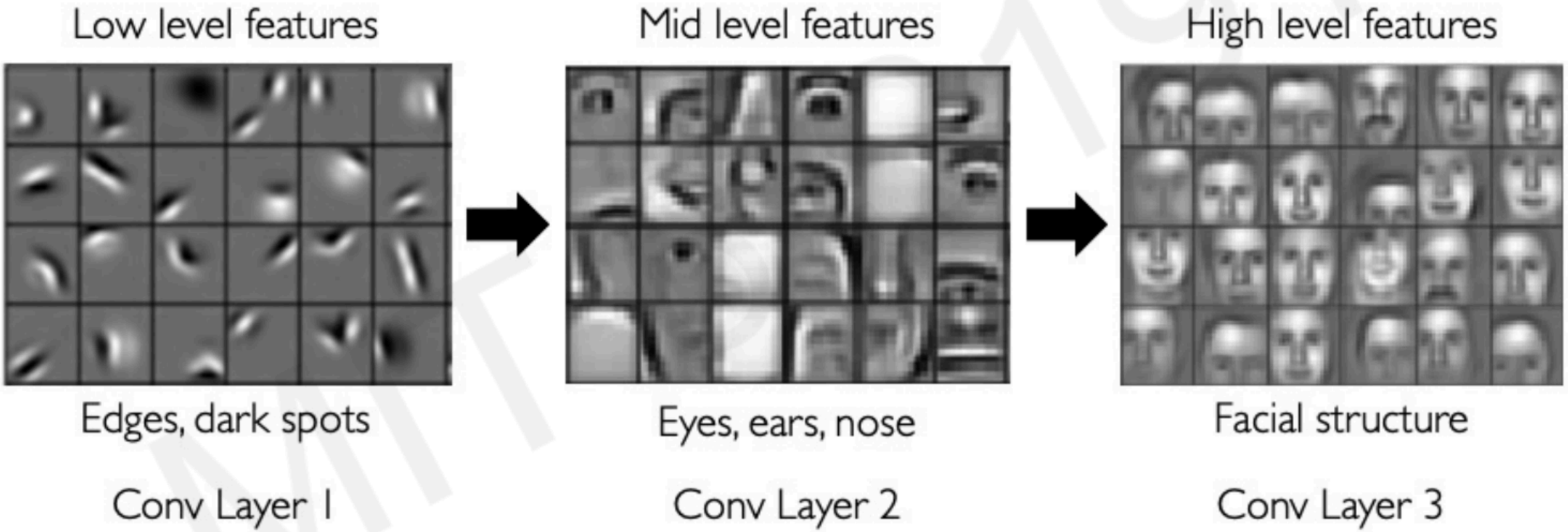
1) Reduced dimensionality
2) Spatial invariance

How else can we downsample and preserve spatial invariance?

Average Pooling!

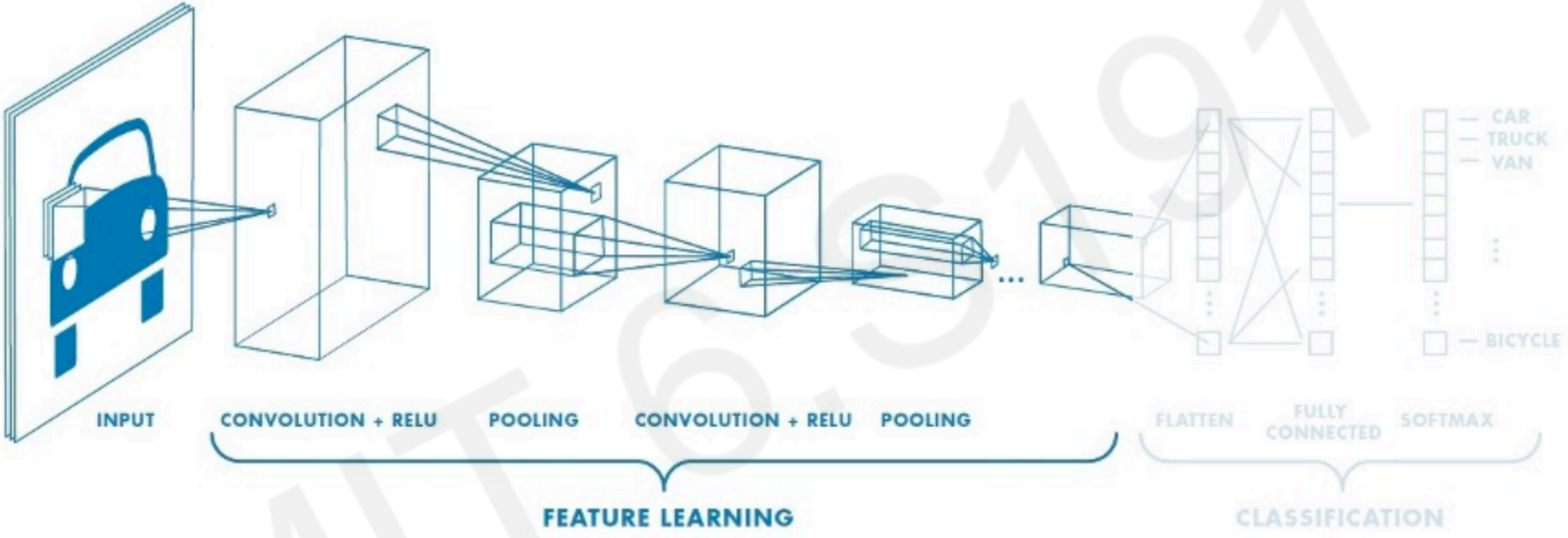
http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

Representation Learning in Deep CNNs



http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

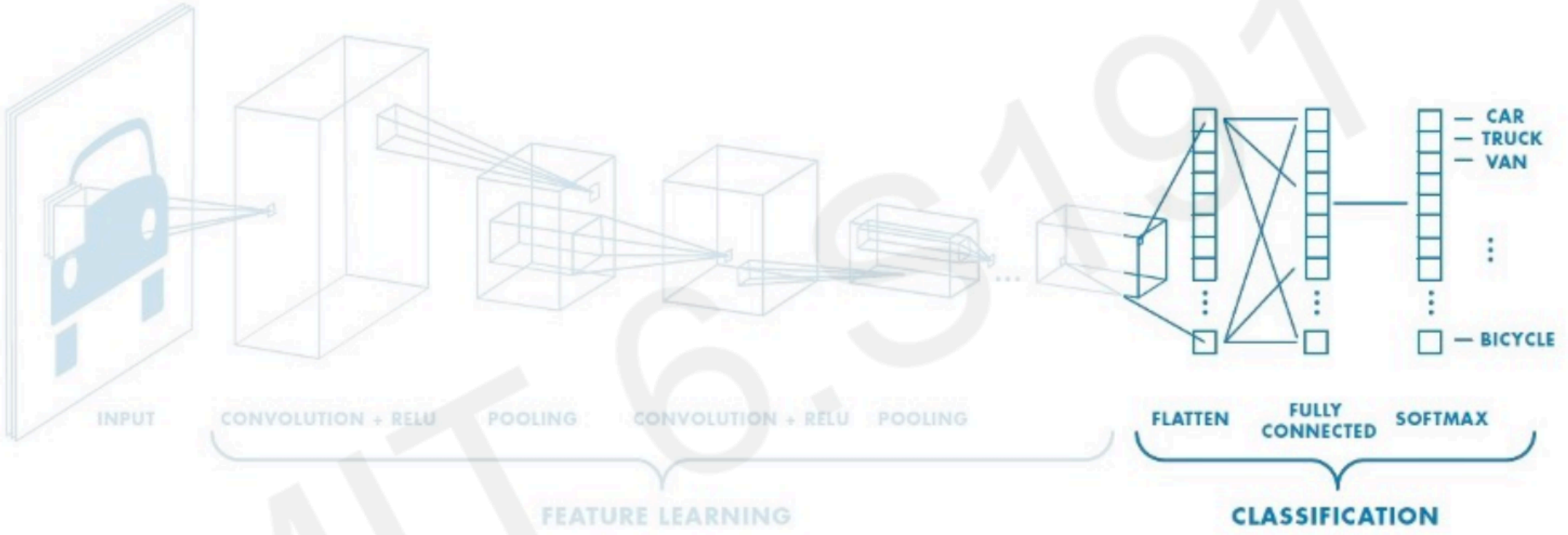
CNNs for Classification: Feature Learning



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**

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

CNNs for Classification: Class Probabilities



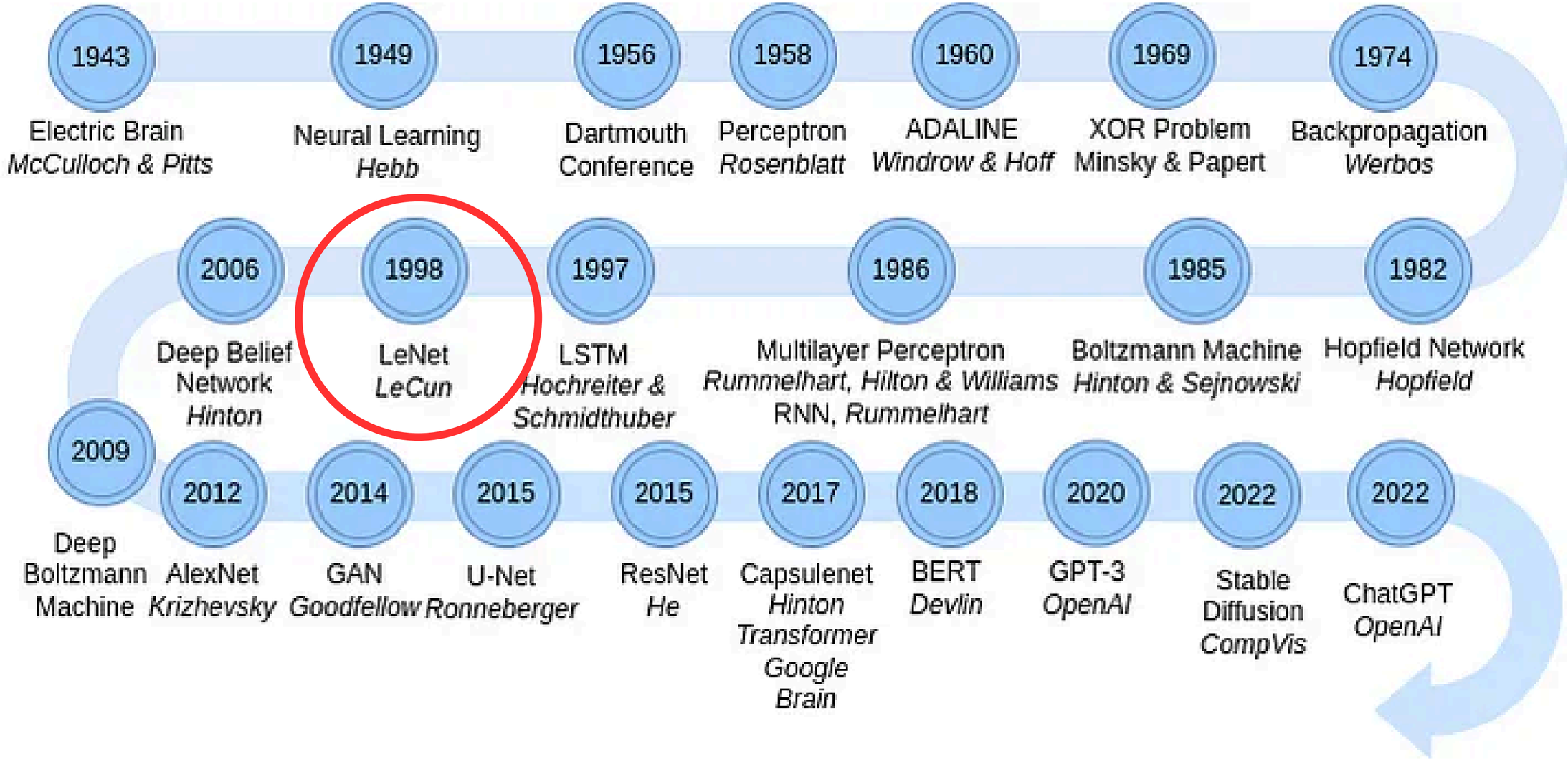
- CONV and POOL layers output high-level features of input
- Fully connected layer uses these features for classifying input image
- Express output as **probability** of image belonging to a particular class

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

http://introtodeeplearning.com/slides/6S191_MIT_DeepLearning_L3.pdf

Brief history of various CNN architectures

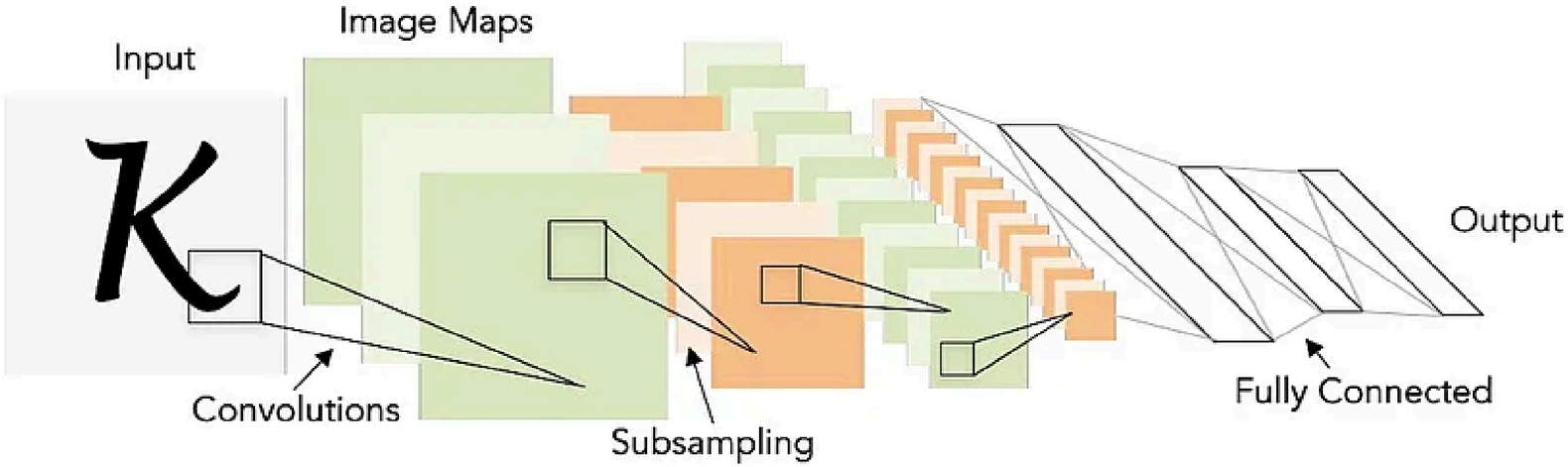
LeNet-5 (1989-1998)



https://anupamakandala1111.medium.com/mastering-advanced-deep-learning-a-comprehensive-roadmap-56681af2706e

LeNet-5 (1989-1998)

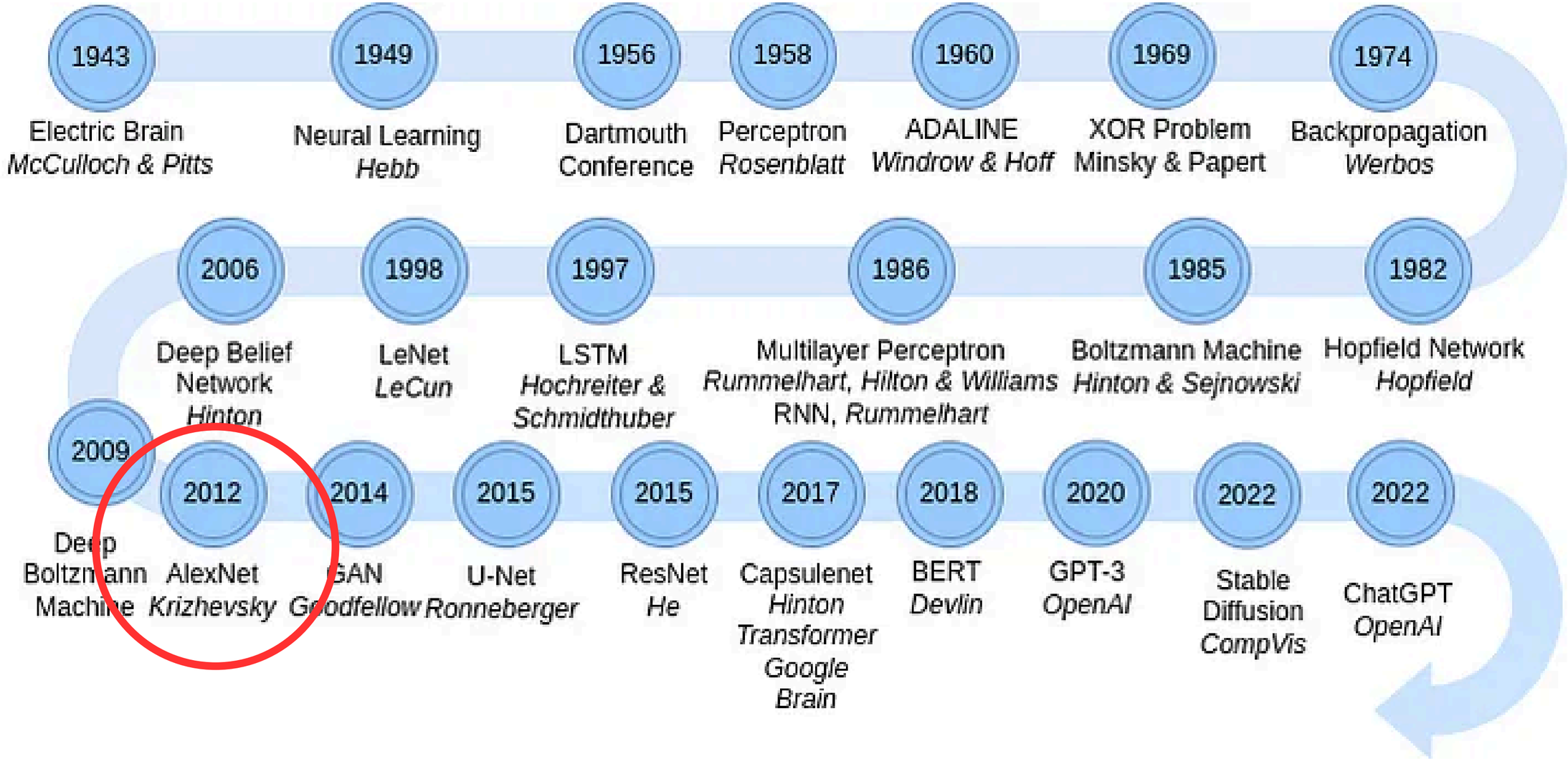
- Was mostly developed for the hadnwritten digits recognition task



https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405568597

- Overall architecture was [CONV-POOL-CONV-POOL-FC-FC]
- Used 5x5 conv filters with stride 1
- Pooling layers of dimension 2x2 with stride 1
- 60K params

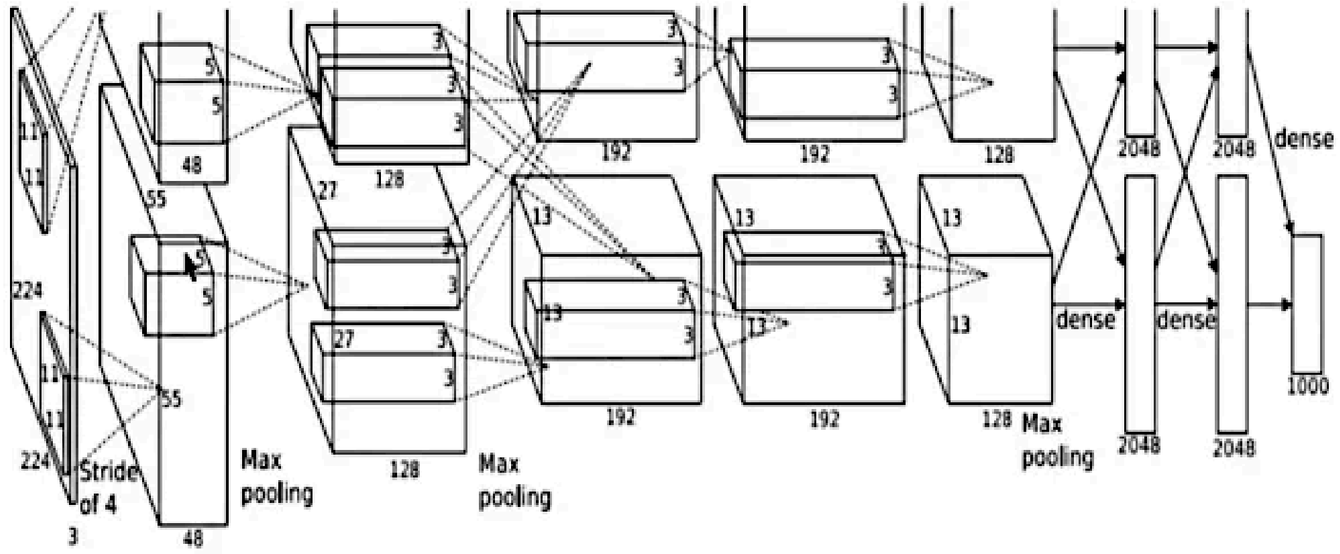
AlexNet (2012)

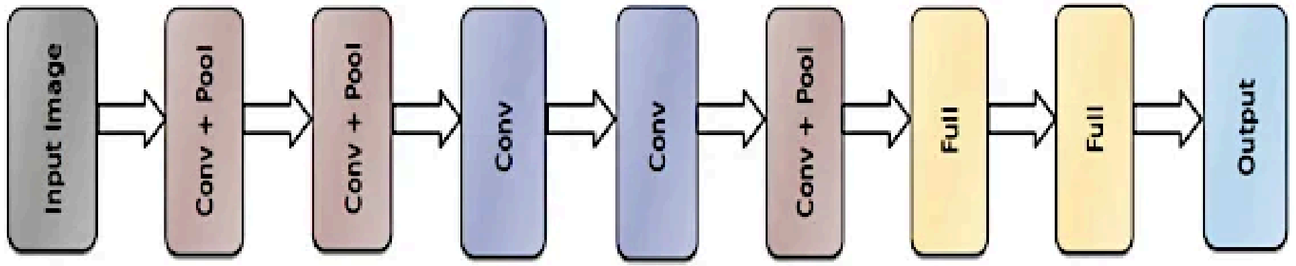


https://anupamakandala1111.medium.com/mastering-advanced-deep-learning-a-comprehensive-roadmap-56681af2706e

AlexNet (2012)

- Was the first CNN-based model that took part in the ImageNet challenge
- Significantly outperformed traditional methods by a margin of 9.4%
- Partitioned the model into two halves and ran them in parallel on one GPU each due to memory constraints!
- Total 8 layers with 5 conv layers and 3 fully connected layers
- 60M parameters
- Conv layers perform 95% of computations while contributing to only 5% of params





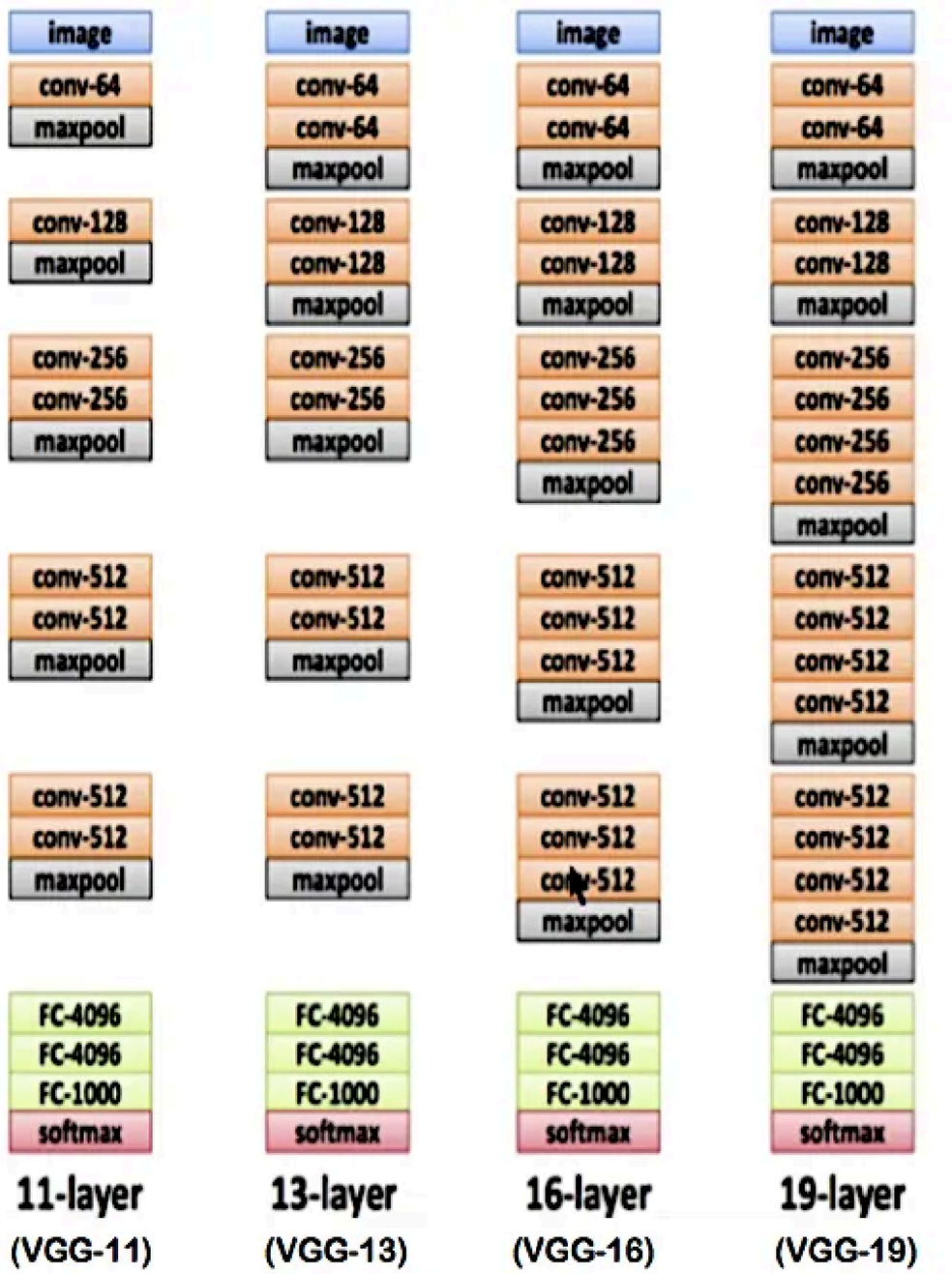
https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405568597

VGGNet (2014)

- Main contribution was that adding depth to the model gave performance boost due to the increase in non-linearity from activations with increase in the number of layers

- But the difference between VGG-16 and VGG-19 was minimal, which showed saturation with depth

- VGG-16 consisted of 13 conv and 3 fc layers with a whopping 138M parameters with a memory overhead of 48.6MB as compared to a meagre 1.9MB of AlexNet

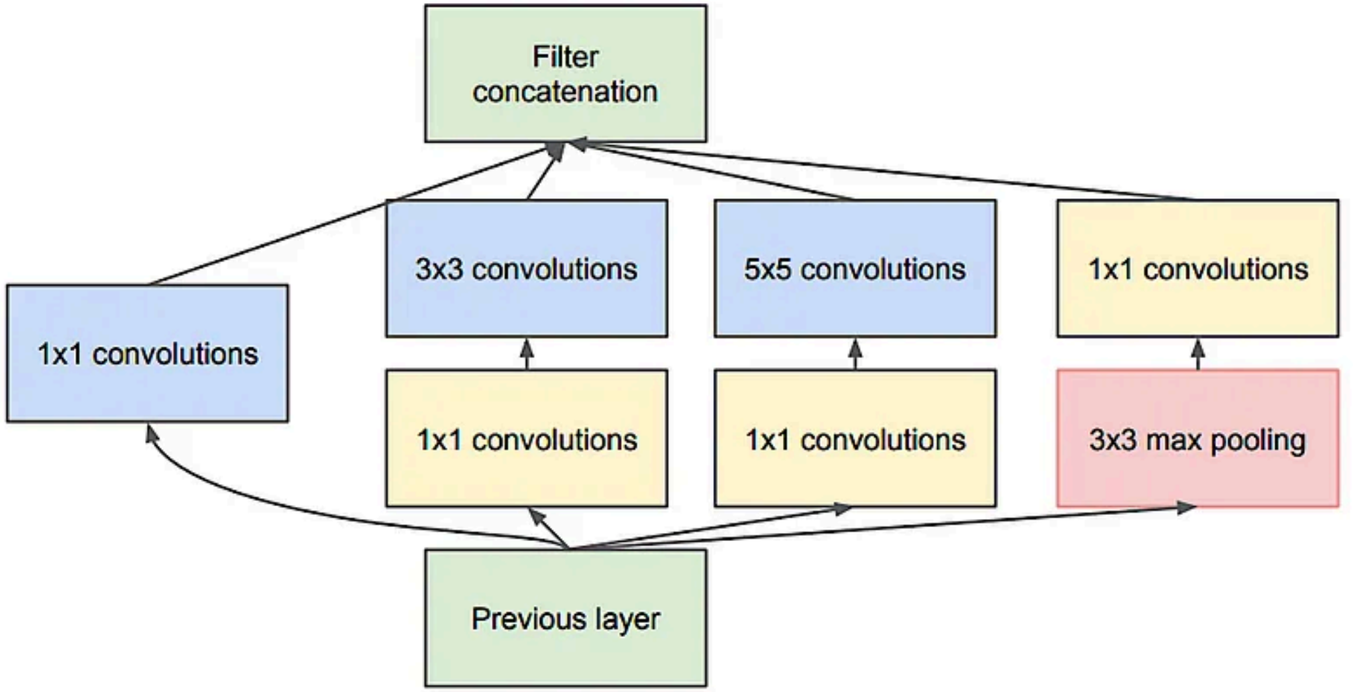


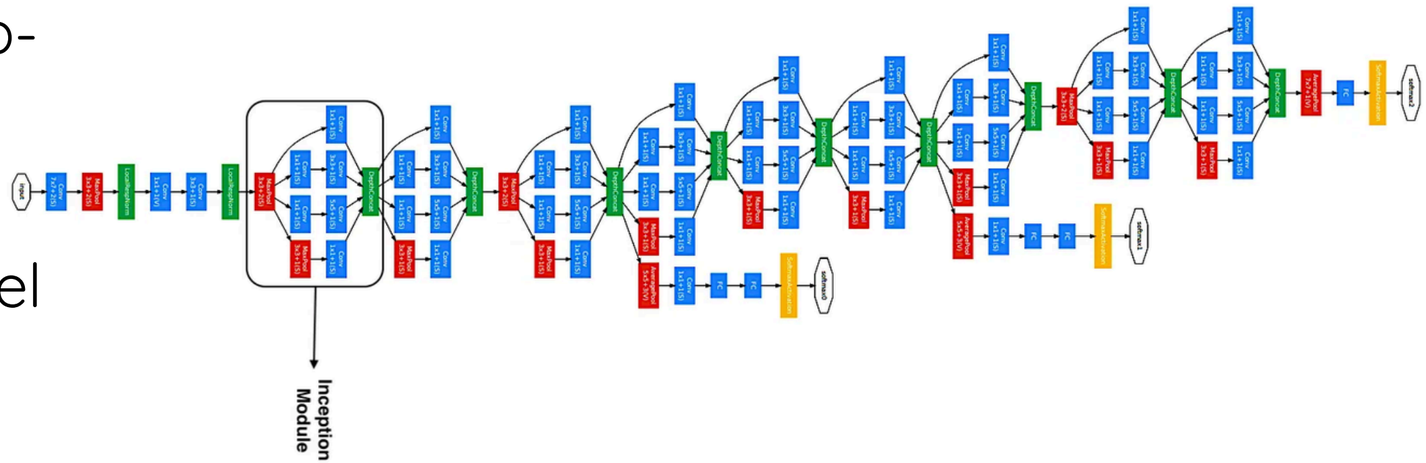
https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405568597


WE NEED TO GO
DEEPER

GoogleNet (2014)

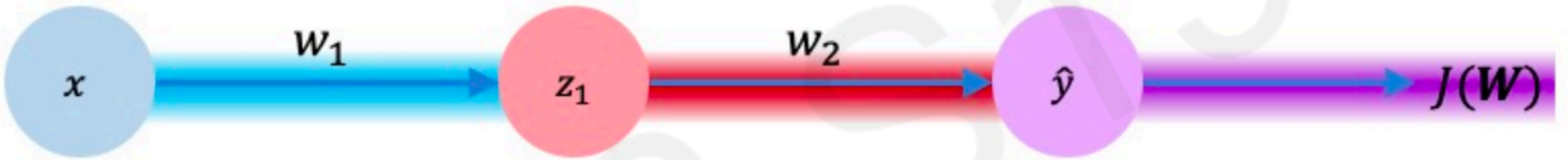
- They further increased depth upto 22 layers but with reduction in parameters by saving fc layers only for the last, resulting in just 5M params
- Number of features were increased at each level using the inception module where they were stacked from all sub-branches
- Observed that 20 layer model was performing better than 56 layer model on both train and test sets. Why?





https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405568597

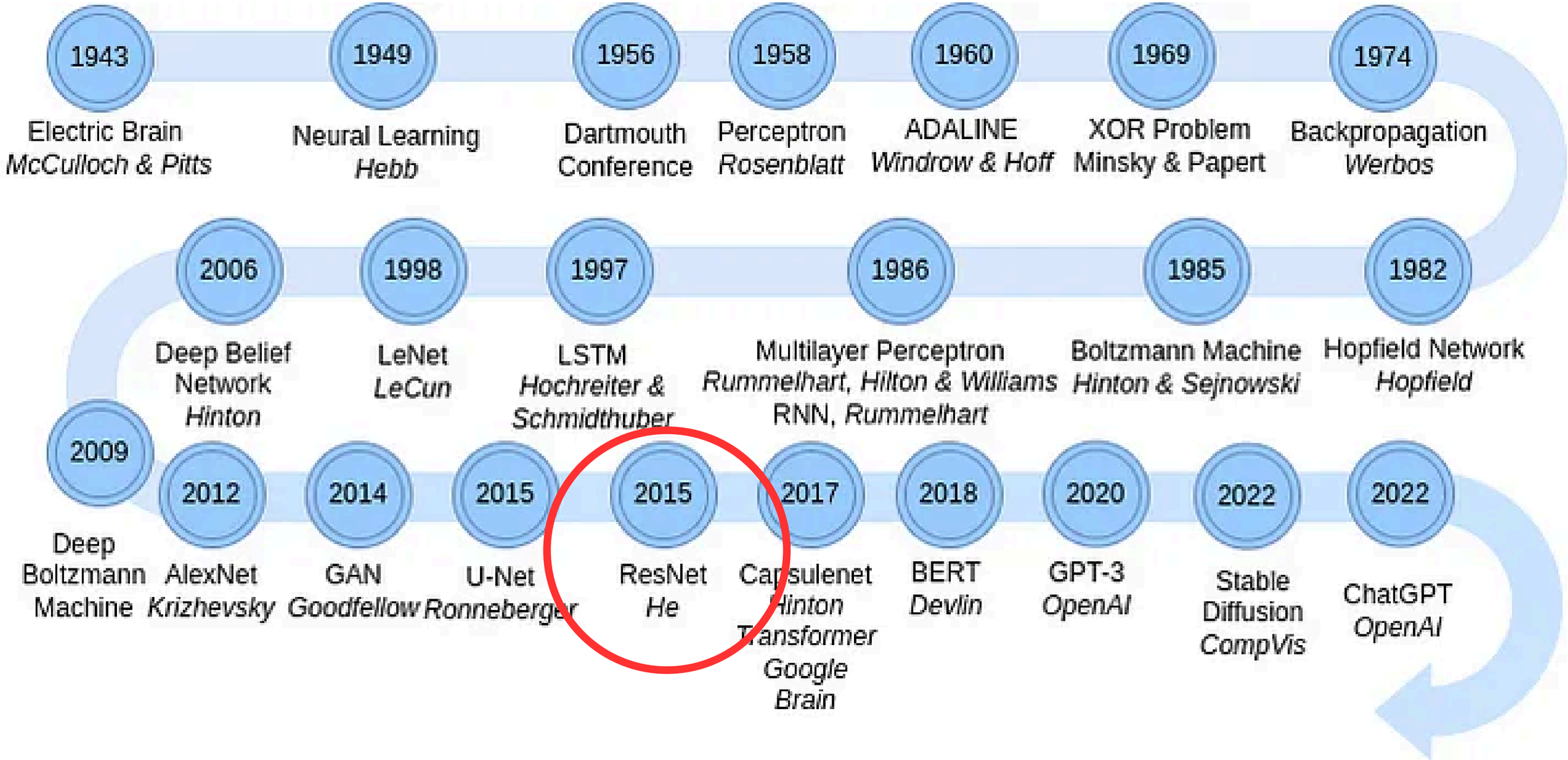
Vanishing/Exploding Gradients



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

- As we keep adding the layers, the derivatives keep on stacking, and after we reach a point, if they are less than 1, we get nearly zero values and if they are greater than 1, we get exponential blowups of gradients during backprop
- How to fix this?

ResNet (2015)



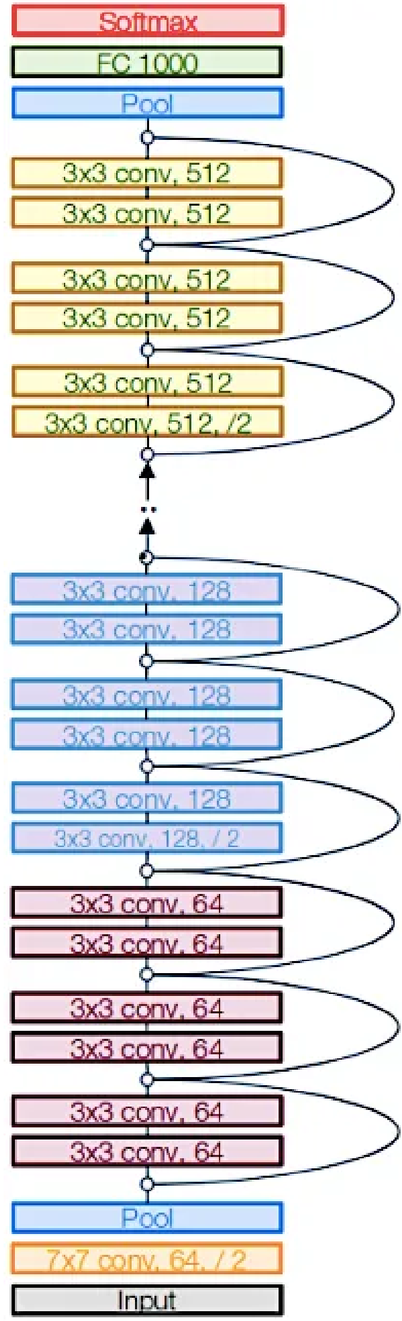
https://anupamakandala1111.medium.com/mastering-advanced-deep-learning-a-comprehensive-roadmap-56681af2706e

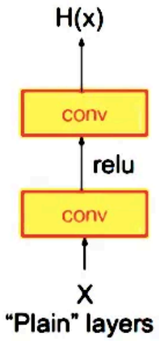
ResNet (2015)

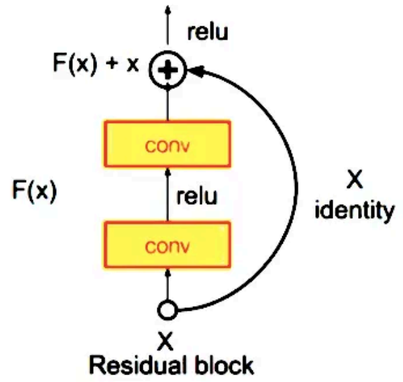
- The layers are connected with identity (skip) connections across blocks, where each block has two 3x3 conv layers

- Using this, they were able to go as deep as 152 layers

- This is because, even if the gradients for the function collapse, the identity input added to the output keep it in check and allow for smooth backprop

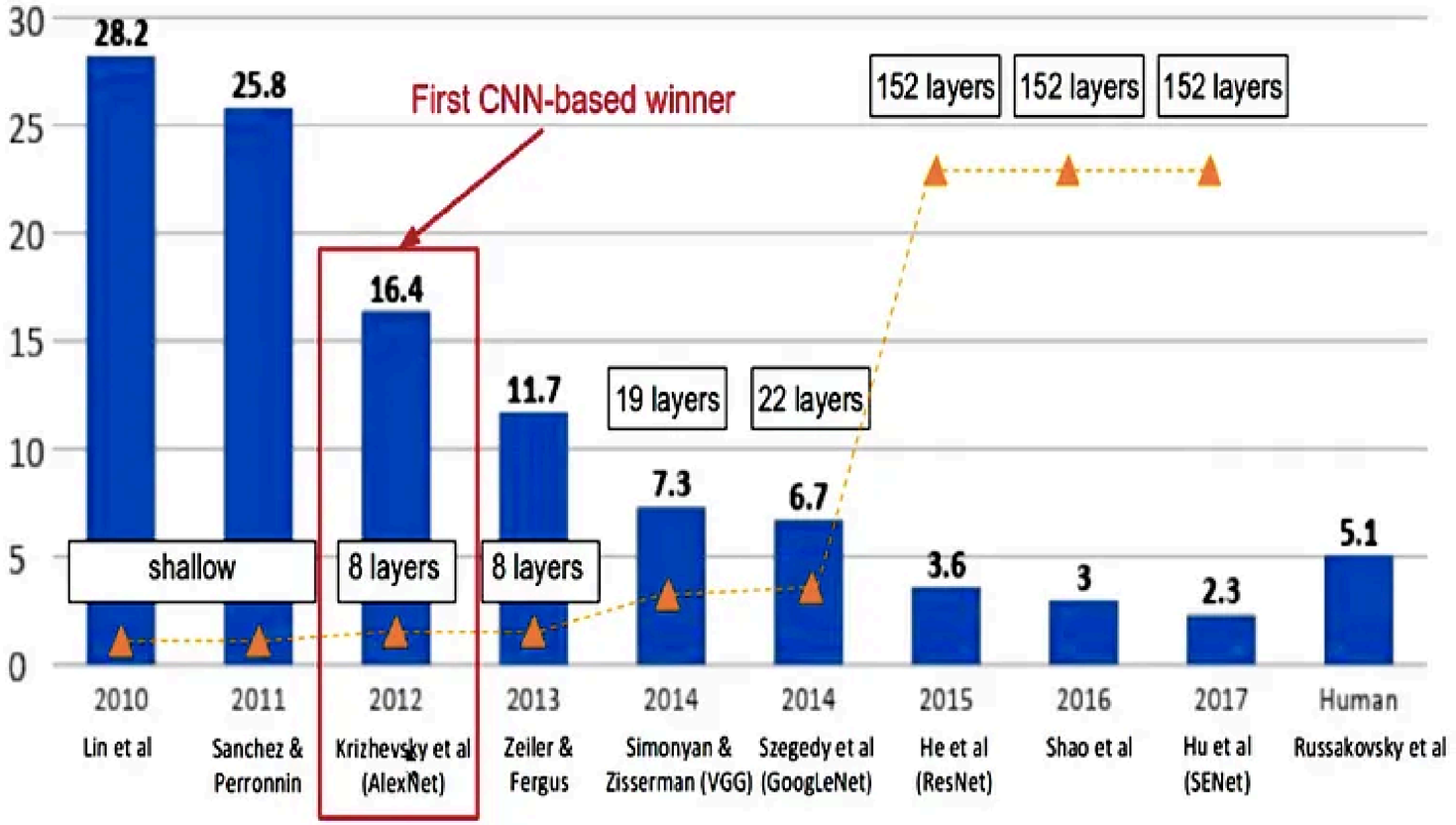






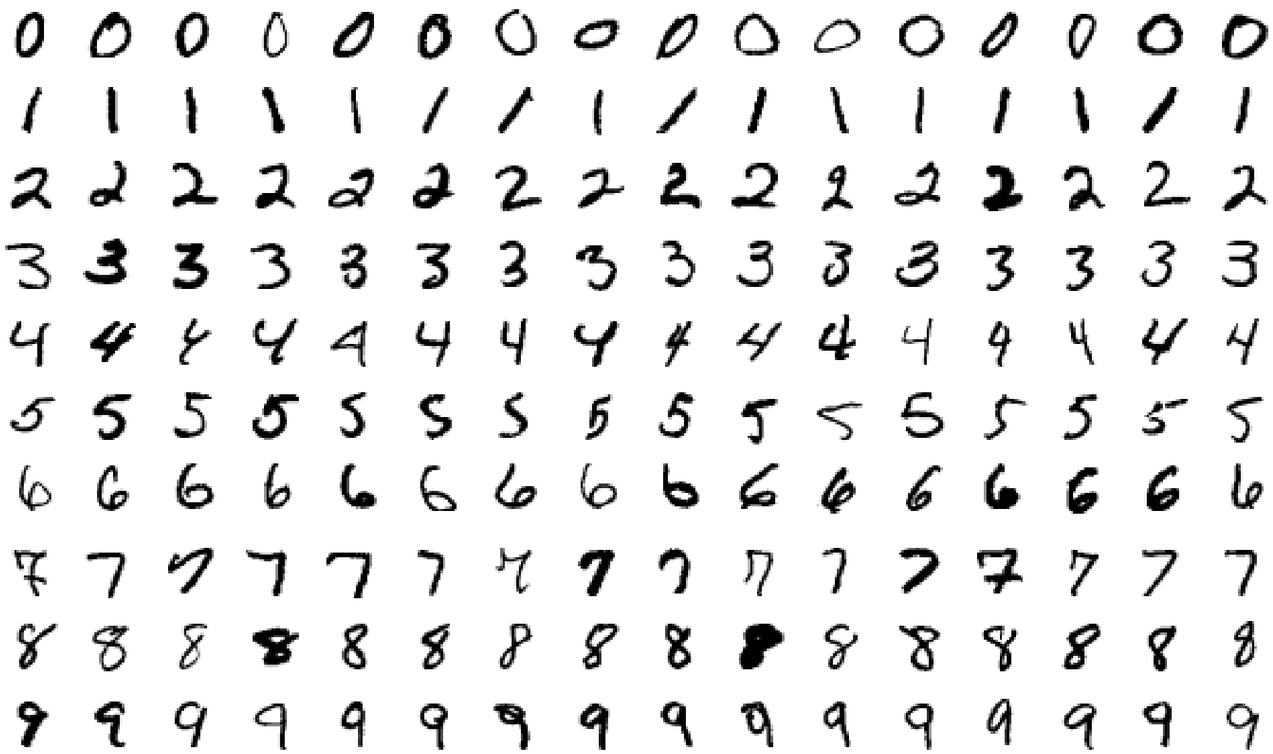
https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405568597

Winners of the ImageNet classification challenge



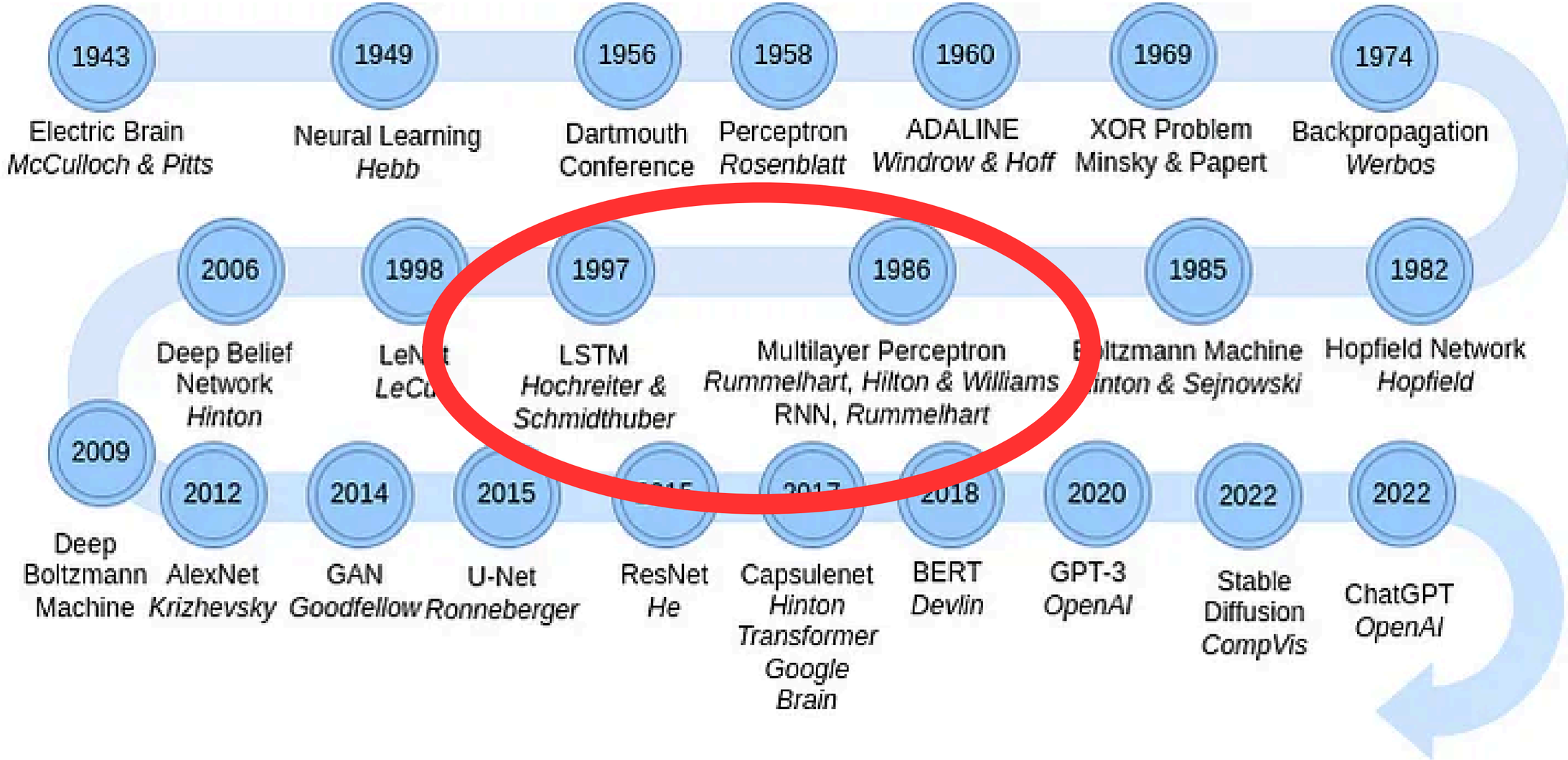
https://medium.com/appyhigh-technology-blog/convolutional-neural-networks-a-brief-history-of-their-evolution-ee3405568597

Code Demo - LeNet on MNIST dataset



https://en.wikipedia.org/wiki/MNIST_database

RNNs and LSTMs



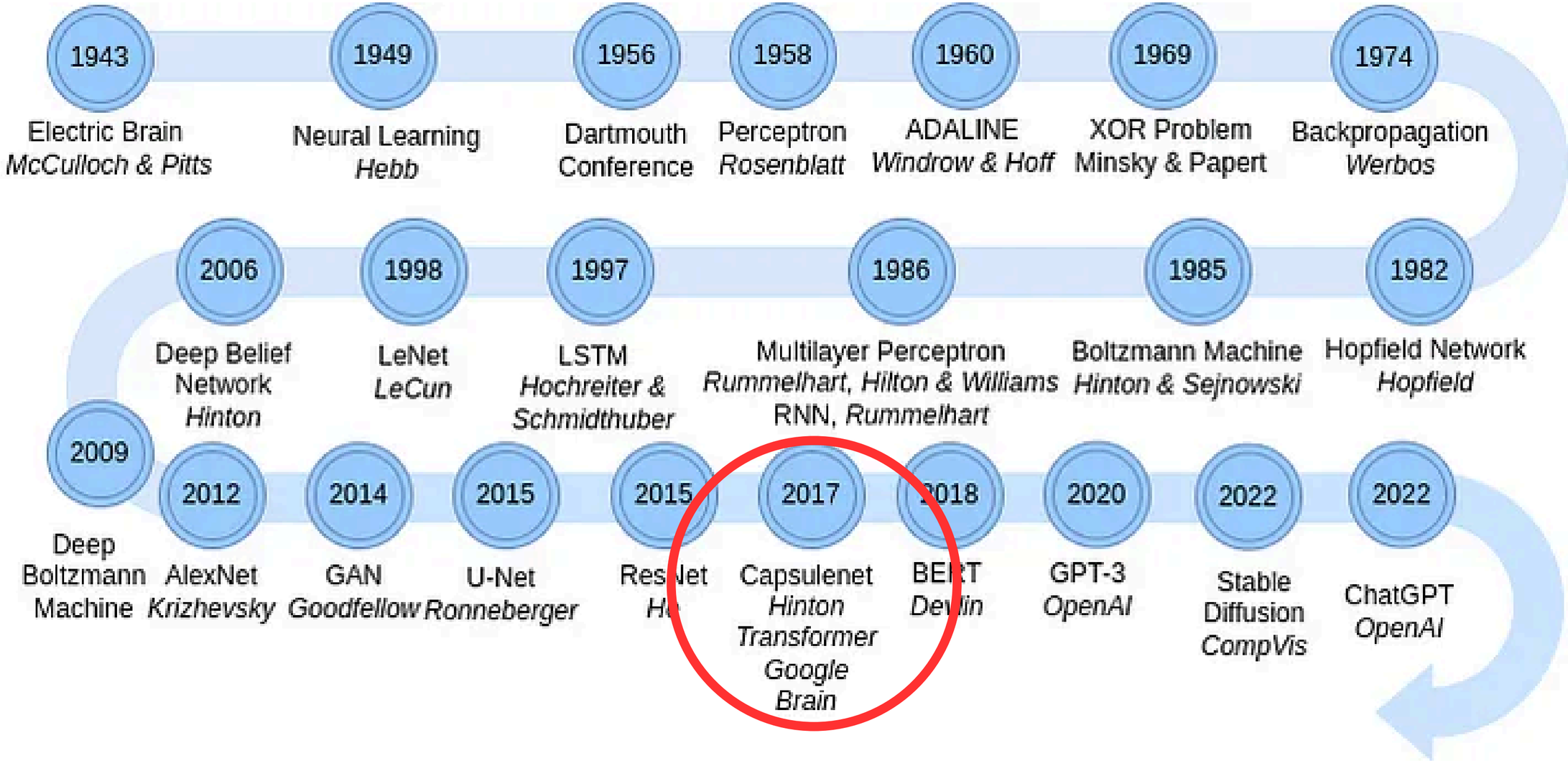
https://anupamakandala1111.medium.com/mastering-advanced-deep-learning-a-comprehensive-roadmap-56681af2706e

RNNs and LSTMs

- An excellent blog describing RNNs and LSTMs in detail

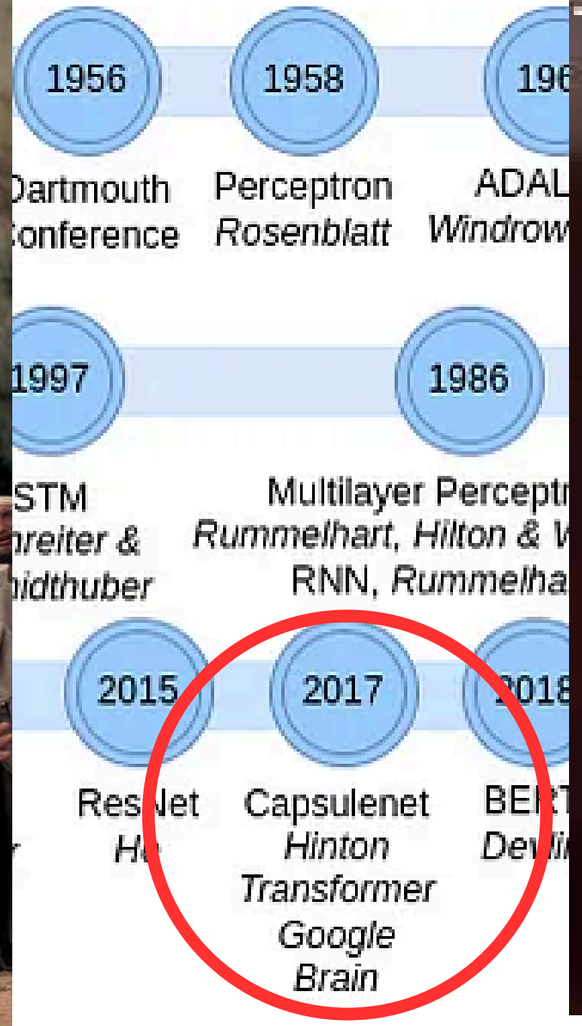
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Transformer - Attention Is All You Need



https://anupamakandala1111.medium.com/mastering-advanced-deep-learning-a-comprehensive-roadmap-56681af2706e

Transformer - Attention Is All You Need



https://anupamakandala1111.medium.com/mastering-advanced-deep-learning-a-comprehensive-roadmap-56681af2706e

Transformer

~~Transformer~~ - Not so soon...

Sources for material I used in these slides

- MIT Intro to DL
- MIT CNNs
- History of CNNs
- Understanding RNNs and LSTMs
- Timeline of Deep Learning
- Structural Similarity between Neuron and Perceptron - Neuron
- Structural Similarity between Neuron and Perceptron - Perceptron
- Types of Normalization
- Padding and Stride Visualization
- Output dimension of CNN given Padding and Stride
- Samples of MNIST Dataset

Other media

- Obama’s “speech”
- ChatGPT Text Prompt
- RCB lifting IPL trophy
- OpenAI SORA announcement video
- NVIDIA sells shovels

Further Readings

- 3 Blue 1 Brown Deep Learning Playlist - Can’t recommend enough

- CS231n RNN blog
- Andrej Karpathy’s RNN blog
- Deep Learning book RNN chapter

- Batch Norm blog
- Layer Norm blog
- Deep Learning Normalization methods

- CS231n: Deep Learning for Computer Vision
- Deep Learning Book