MACHINE
LEARNING
TOKYO

Deep Learning Workshop Series

Learning in Deep Neural Networks

CONTENT

Intro to Learning in Deep Nets

- Learning in ML
- Learning in Brain: from dendrites to axons
- Learning in Deep Networks : from input to output
    a. Input, weight, bias and output relation
- Some concepts: prediction, probability, regression and logistic labeling
- Representation learning : supervised, unsupervised
- Meta learning : learning to learn

Losses (Objective functions)

- Distance metrics analogy
- Why loss important?
- MSE (l2) (implement)
- L1 (implement)
- (categorical/binary) CrossEntropy (implement)
- Other losses (application,data based loss selection)
- GAN's loss (learned loss)
- Design custom loss (time dependent)

Activation Functions

- Activate what and why
- Tanh(implement)
- Sigmoid (implement)
- Softmax (implement)
- ReLU (implement)
- Leaky ReLU (impleme
- Design custom activat dependent)

Regularization

a. Batchnorm (instance norm, ...)
b. Dropout
c. l1/l2
d. Weight decay

Optimization (Backprop)

- ➔ What is weight updating?
- ➔ Gradient descent
- ➔ Global-local minima
- ➔ SGD , Adam , AdaGrad etc..
- ➔ Backprop in pooling (implement)
- ➔ Is it really learning? (underfitting, overfitting)

Metrics for evaluation (Simple implemen

- ➔ Accuracy
- ➔ TP, FP, FP, FN
- ➔ Precision & Recall (f1-score)
- ➔ Specificity & Sensitivity
- ➔ IoU
- ➔ mAP
- ➔ Design custom metric (time dep

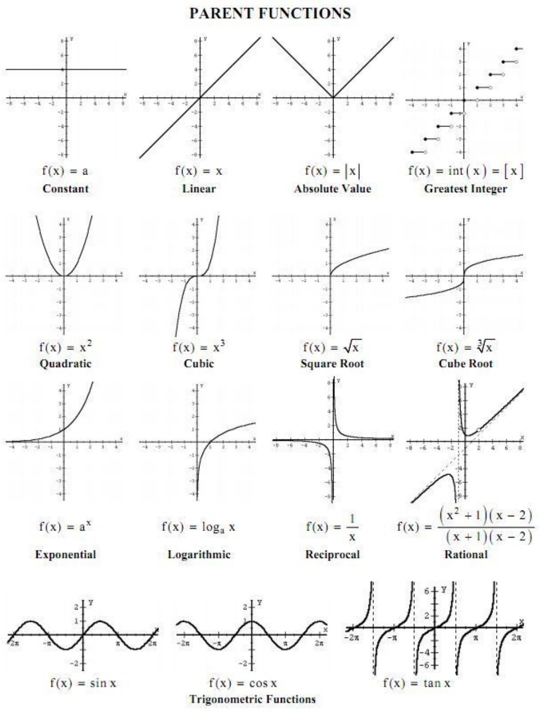
PROGRAM FLOW

11:00 - 11:30	11:30 - 11:50	11:50 - 13:00	13:00 - 13:45	13:45 - 14:15	14:15 - 15:00
INTRO	Part 1 : Intro to Learning	Part 2: Loss Functions	LUNCH	Part 3: Activation Functions	Part 4: Regularization

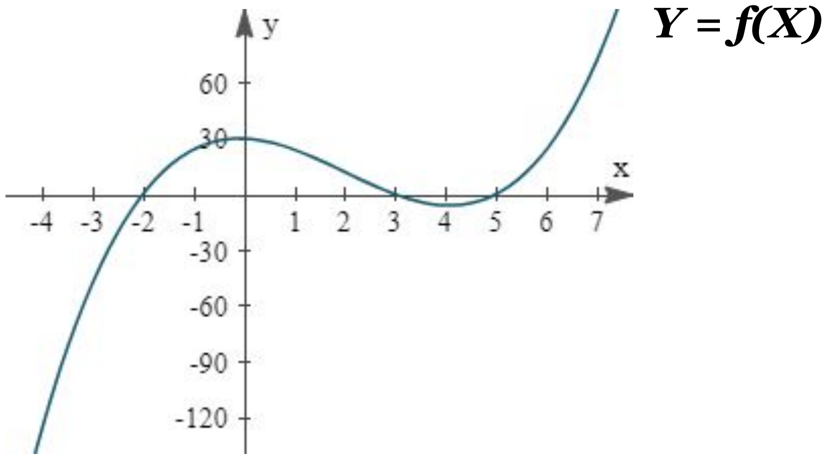
15:00 - 15:10	15:10 - 16:00	16:00 - 16:10	16:10 - 17:00	17:00 -	17:15 -
BREAK	Part 5: Optimization Process	BREAK	Part 6 : Metrics for Evaluation	CLOSING	

Part 1 : Intro to Learning in Deep Nets

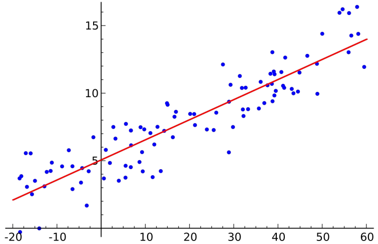
Learning in Machine Learning



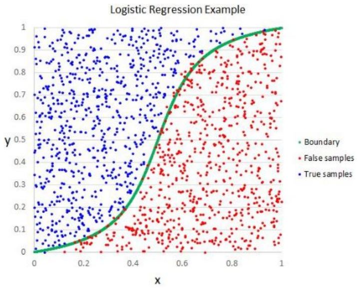
Machine learning algorithms are described as learning a target function (f) that best maps input variables (X) to an output variable (Y): ***Y = f(X)***



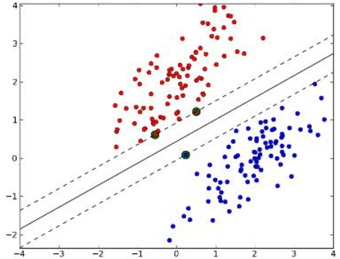
Linear Regression



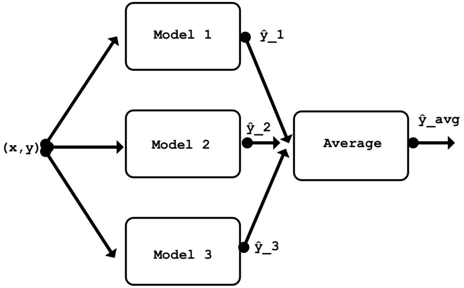
Logistic Regression



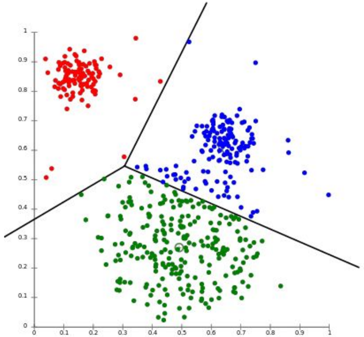
SVM



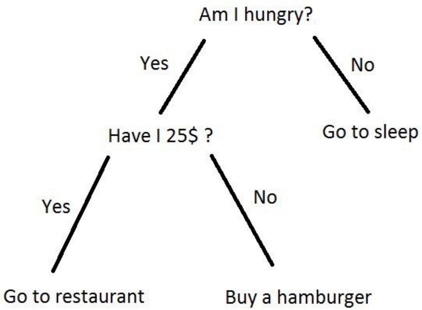
Ensemble Modeling (Adaboost)



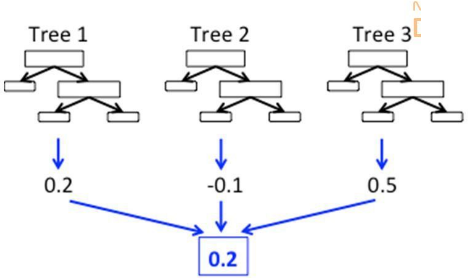
K-means



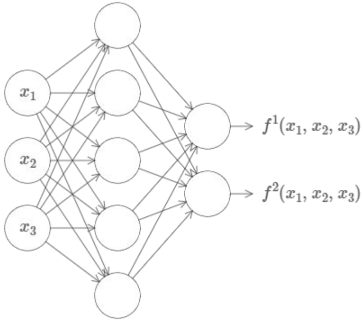
Decision tress

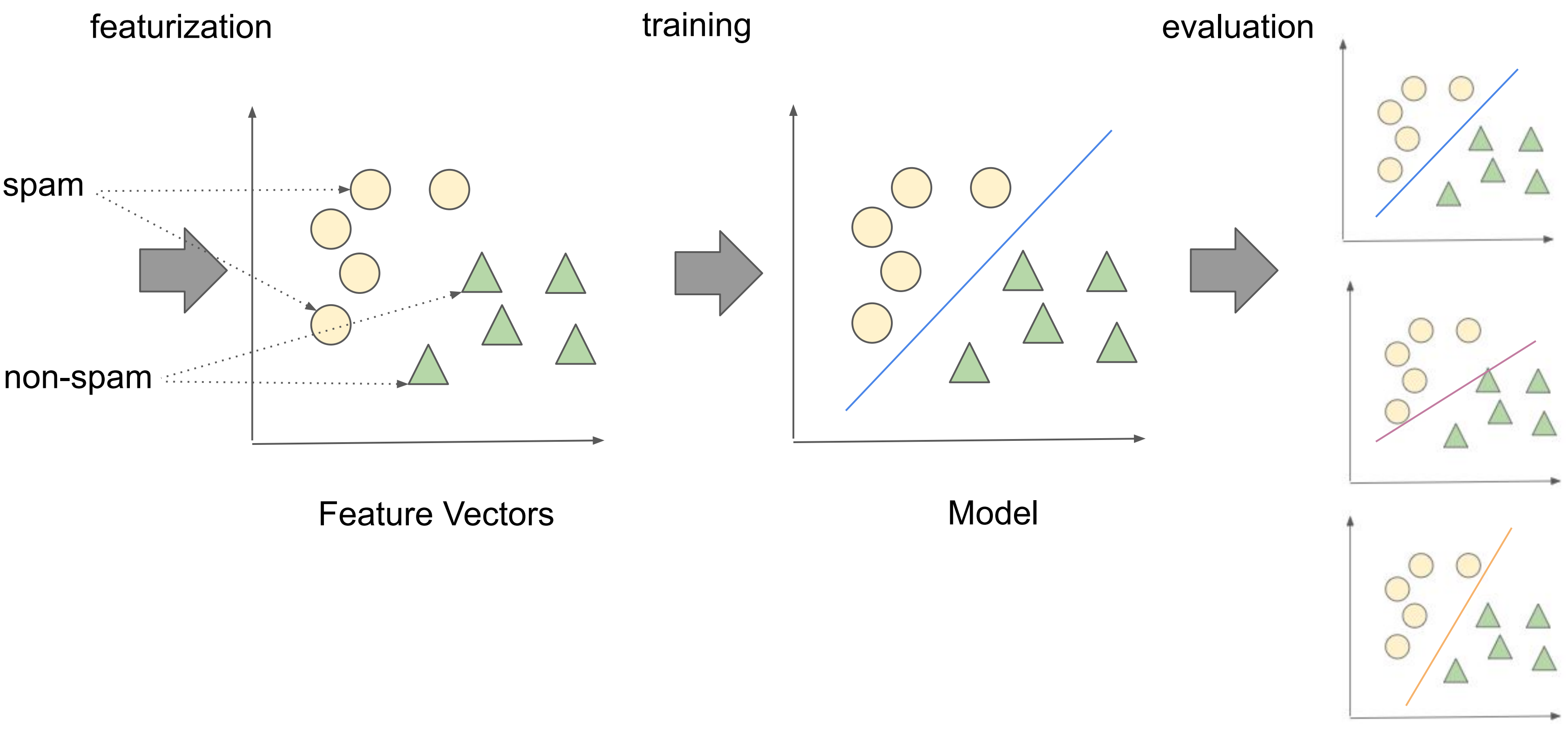


Random Forest



Neural Networks

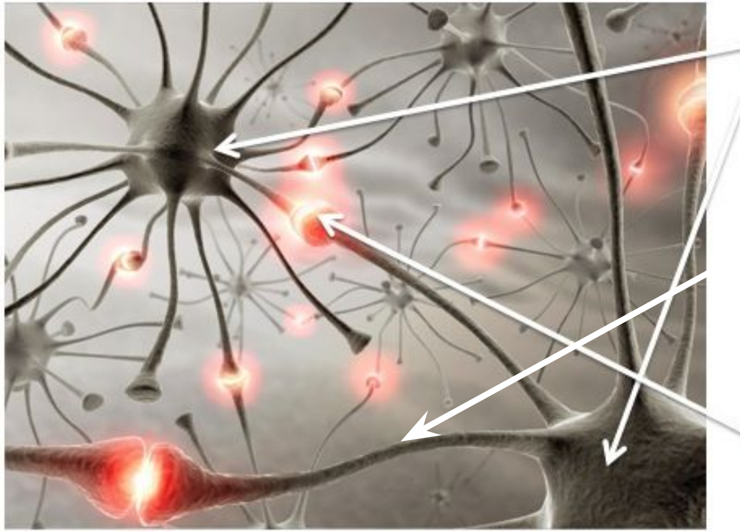




Task is fitting the optimum function to our dataset or problem

Best model

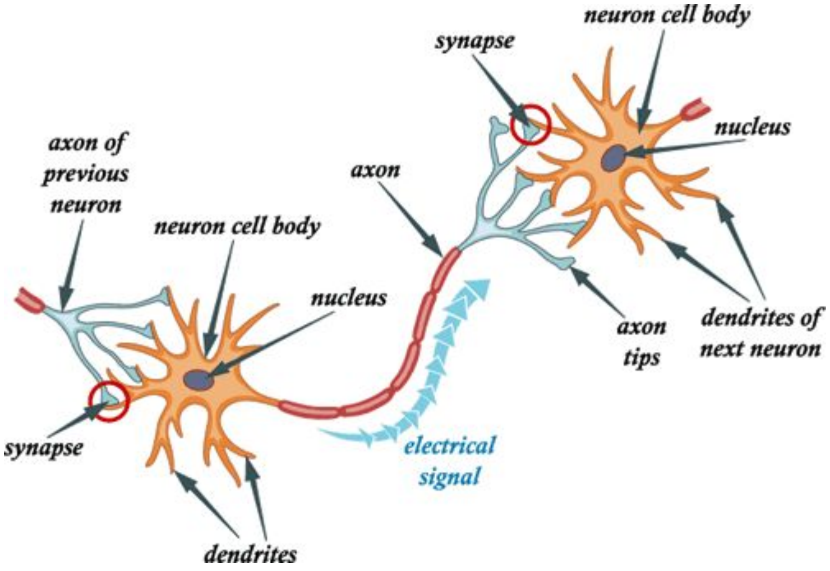
Learning in Brain

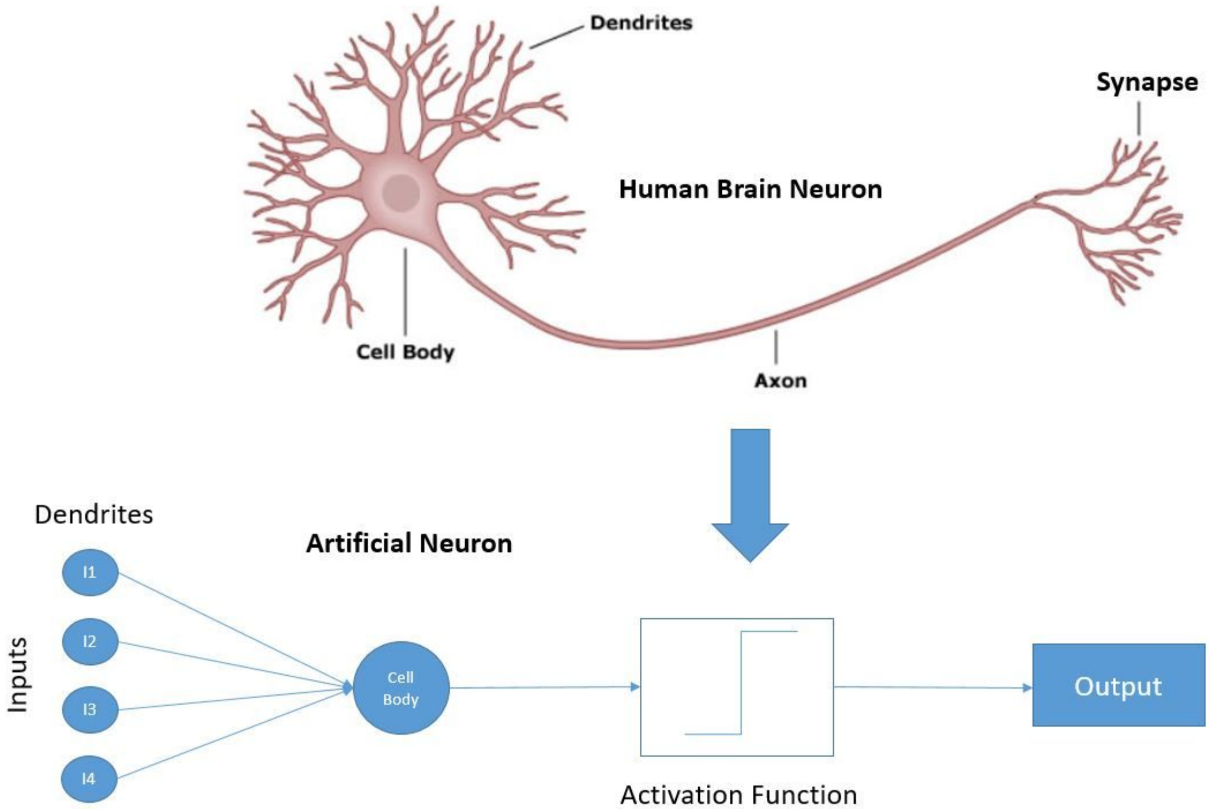


neurons

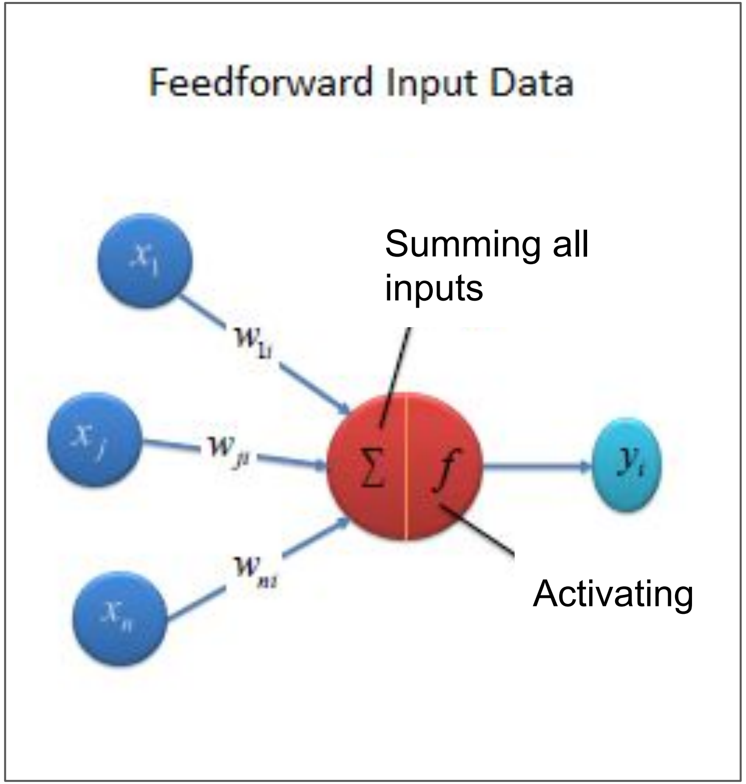
synapsis

synapsis




Dendrites
Synapse
Human Brain Neuron
Cell Body
Axon
Dendrites
Artificial Neuron
Inputs
I1
I2
I3
I4
Cell
Body
Output
Activation Function

Learning in Deep Networks



Calculating the error

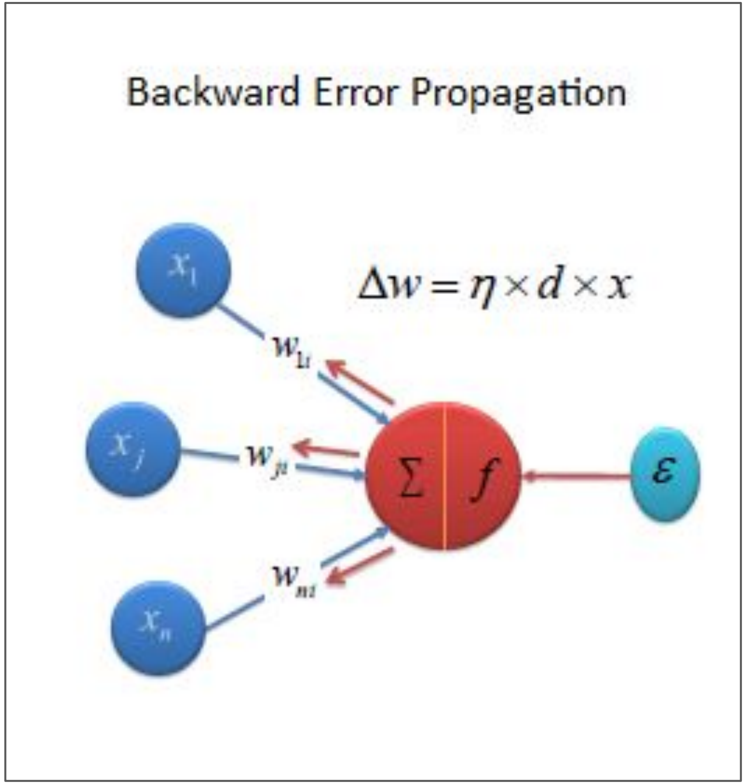
$$y_i - \hat{y}_i$$

Deriving gradients

$$\frac{\partial\,(y_i - \hat{y}_i)}{\partial w}$$

Obtaining cost

$$\Delta w = \frac{\partial\,(y_i - \hat{y}_i)}{\partial w}$$



Part 2 : Losses (Objective functions)

Why loss important?

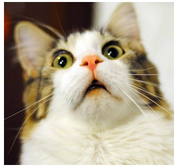
- We learn from our mistakes or experiences
- Every human or human-made system need to optimize itself
- To optimize something, we are going to need an experience or a **quantity**

<u>Problem:</u> Optimization

A quick way

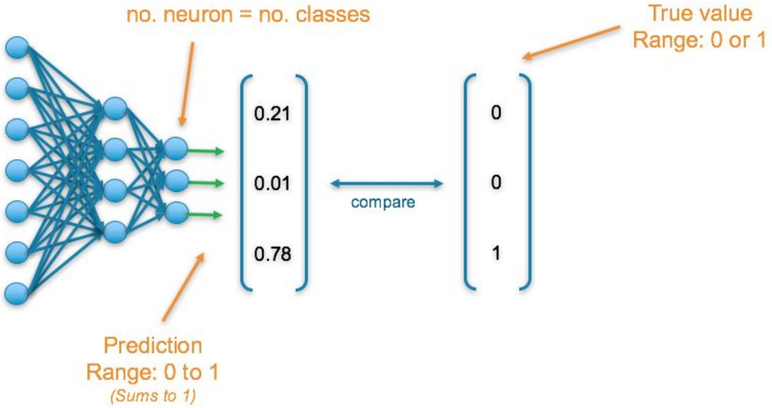
A wise way

Expected vs Observed



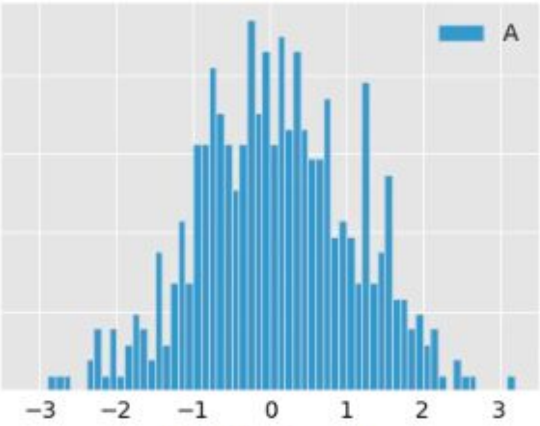
Distance metrics analogy

When we develop a model, we aim to map model's inputs to predictions by adjusting parameters so that our predictions to get closer and closer to true values

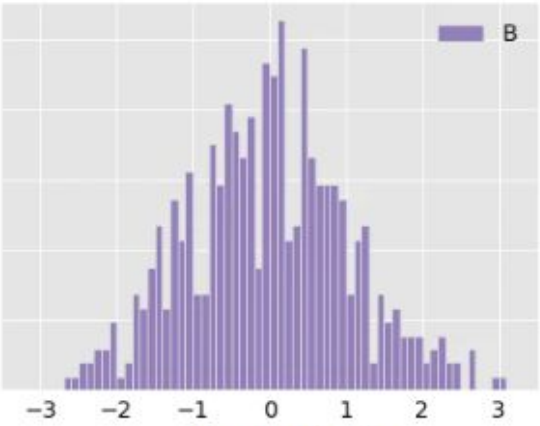


What exactly **'get closer to'** means and how measure **the difference between** predictions and true values ?

Visual inspection is a good heuristic to use , but not enough



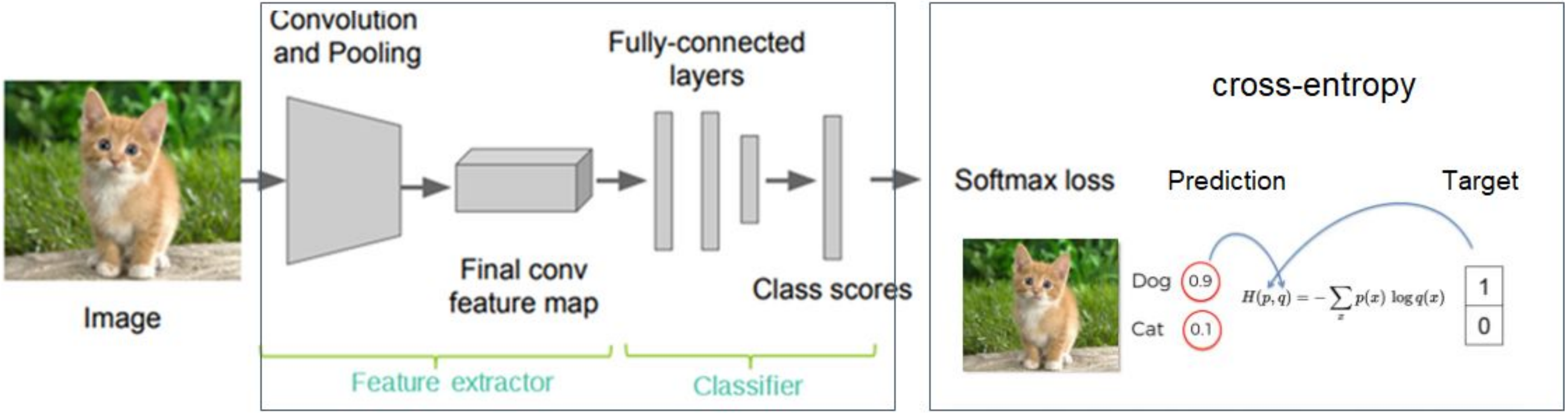
distribution A



distribution B

How ‘Quantitatively’ similar these two distributions ?

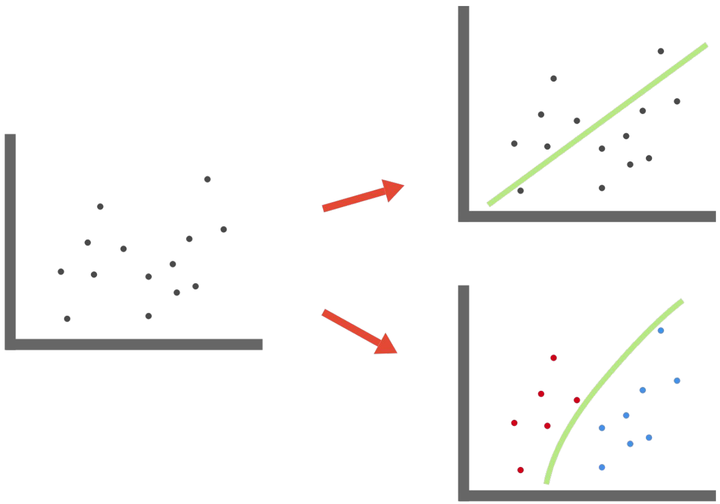
In machine learning , if feature extraction is an important step , the distance metric would be the second



Measure the distance (divergence)

Important Concepts:

- **Classification:** The goal is to predict discrete values, e.g. {1,0}, {True, False}, {spam, not spam}.
- **Regression:** The goal is to predict continuous values, e.g. home prices.



Common Metrics (Losses)

Prediction vs Target

Regression

L1 Loss

$$MAE = \frac{\sum_{i=1}^{n} | y_i - \hat{y}_i |}{n}$$

L2 Loss

$$MSE = \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{n}$$

Classification

Cross-Entropy

$$-(y\log(p) + (1-y)\log(1-p))$$

Hinge Loss

$$\sum_{i=0}^{n-1} \max(0, 1 - y_i(w^T x_i + w_0))$$

Other losses (application,data based loss selection)

Minkowski Loss

$$\left(\sum_{i=1}^{k} (|x_i - y_i|)^q \right)^{1/q}$$

Mean Absolute Deviation Loss

$$L(z) = |1 - z|$$

Checbychev Loss

$$d(x, y) = \max_{i=1}^{m} |x_i - y_i|$$

The Squared Hinge

$$L(z) = \max(0, (1 - x)^2)$$

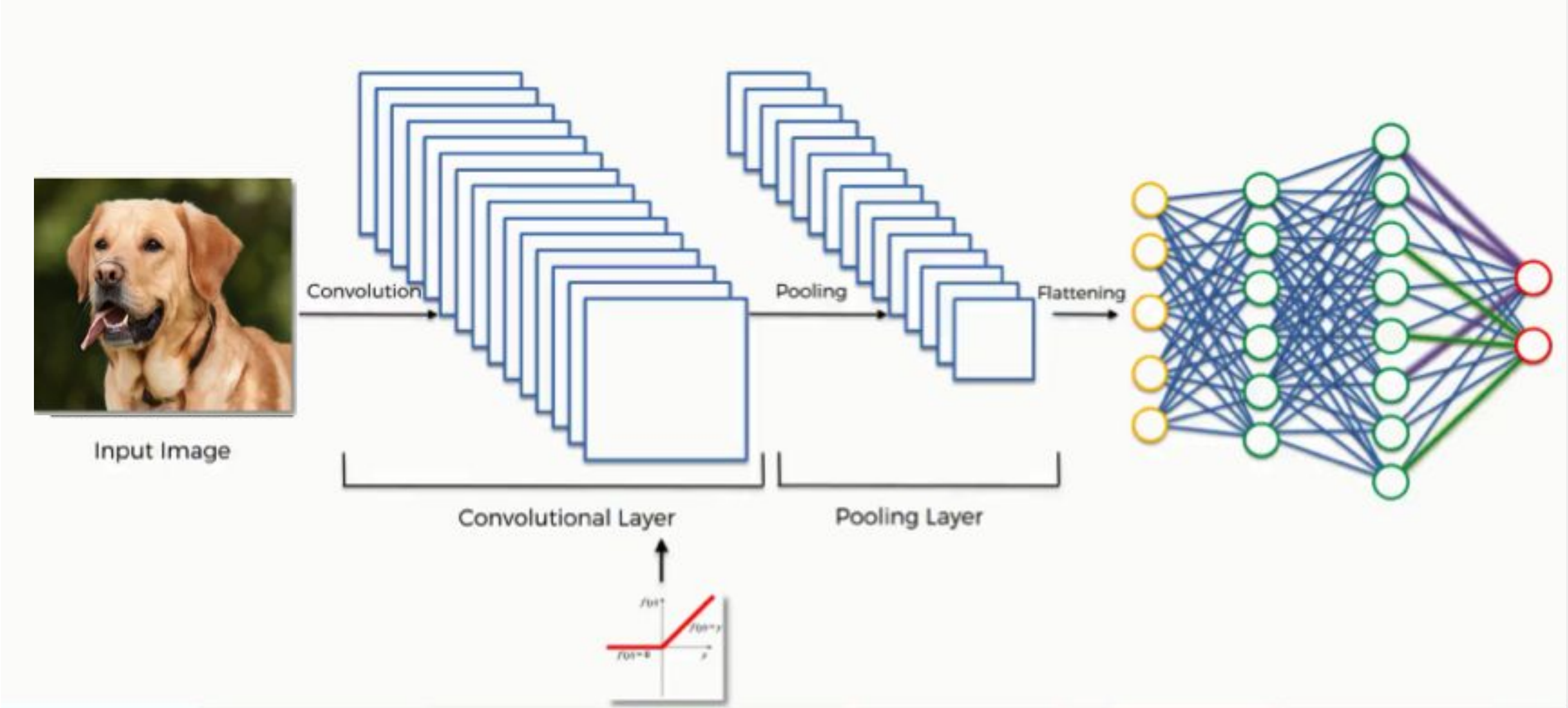
Hamming Loss

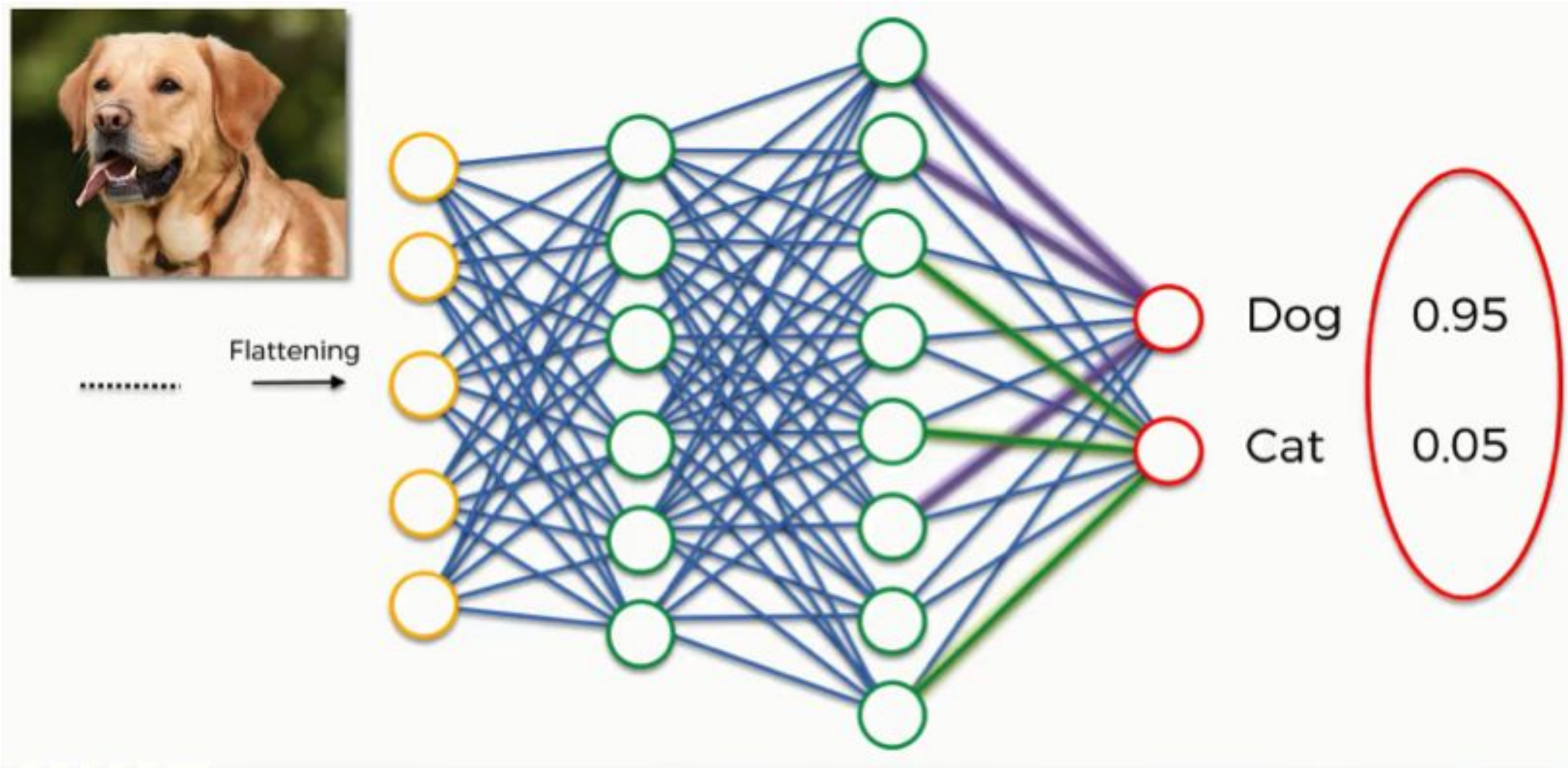
$$\sum_{i=1}^{k} |x_i - y_i|$$

AdaBoost Loss

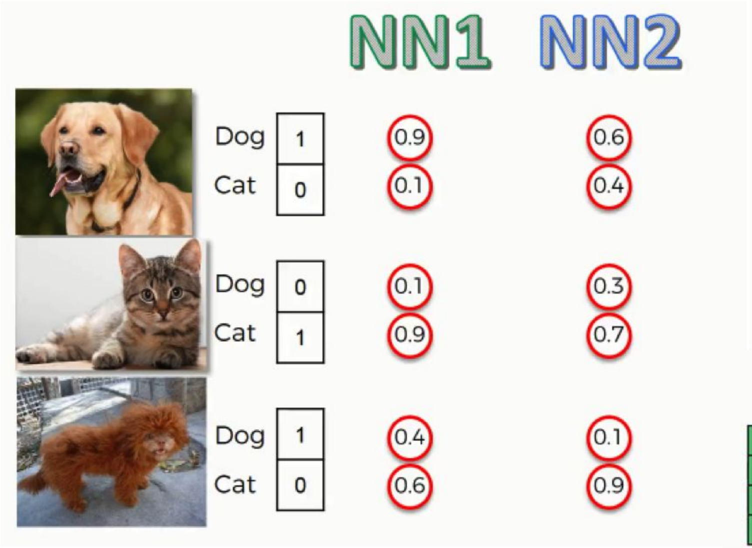
$$L(z) = \exp(-x)$$

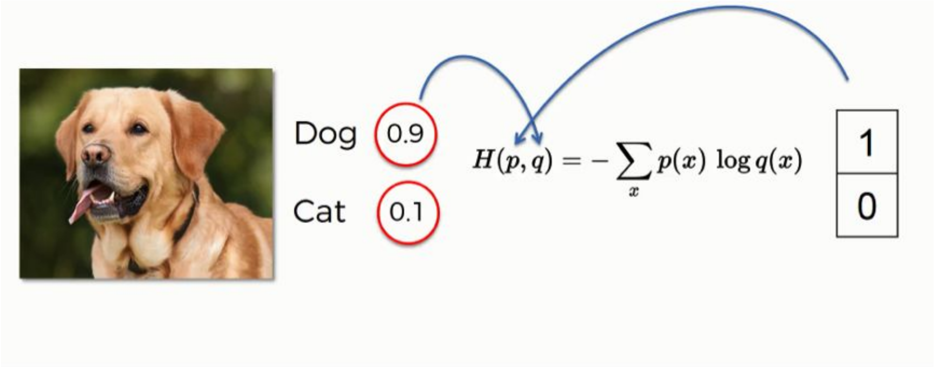
Loss in Deep Networks





1
0





NN1

Row	Dog^	Cat^	Dog	Cat
#1	0.9	0.1	1	0
#2	0.1	0.9	0	1
#3	0.4	0.6	1	0

NN2

Row	Dog^	Cat^	Dog	Cat
#1	0.6	0.4	1	0
#2	0.3	0.7	0	1
#3	0.1	0.9	1	0

	NN1	NN2
Classification Error	1/3 = 0.33	1/3 = 0.33
Mean Squared Error	0.25	0.71
Cross-Entropy	0.38	1.06

Figures : SEBASTIAN MONCADA , SuperDatascience

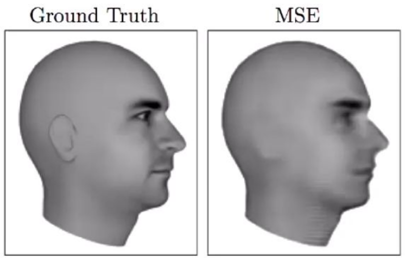
GAN's loss (learned loss)

Explicit losses :

L1
L2
Cross-entropy
KL divergence
...

These losses are good enough for regression, classification, segmentation etc..

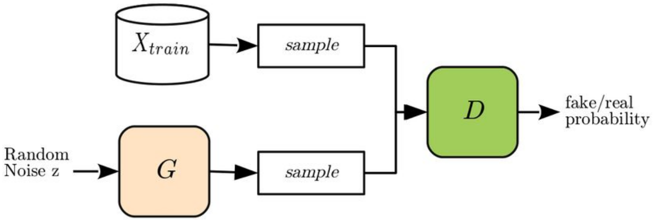
but in case the outputs have a multi-modal distribution, these losses break down



Implicit loss :

So what happens if you replace this explicit loss function with a NN model too?.

Generator's loss being Discriminator network itself !

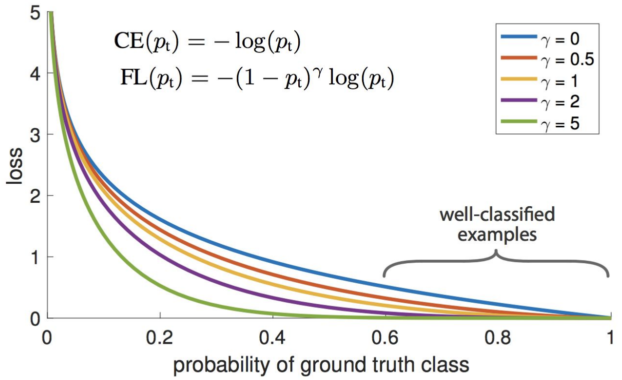


GAN's loss is adversarially learned

Design custom loss (time dependent)

There can be many types of losses based on application :

Ex : Focal Loss for objectness score improvement in object detection



So, let's design our own custom loss !

Interactive Implementation

REFERENCES

[1] Network In Network
Min Lin, Qiang Chen, Shuicheng Yan
https://arxiv.org/pdf/1312.4400v3.pdf

[2] Network in Networks and 1x1 Convolutions
Andrew Ng
https://www.coursera.org/lecture/convolutional-neural-networks/networks-in-networks-and-1x1-convolutions-ZTb8x

[3] One by One [1x1] Convolution - counter-intuitively useful
Aaditya Prakash
https://iamaaditya.github.io/2016/03/one-by-one-convolution/

[4] Deep Learning series: Convolutional Neural Networks
Mike Cavaioni
https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524

[5] Going Deeper with Convolutions
Christian Szegedy et.al.
http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf

[6] A Simple Guide to the Versions of the Inception Network
Bharath Raj
https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

Part 3 : Activation Functions

Snippet Implementation

REFERENCES

[1] Network In Network
Min Lin, Qiang Chen, Shuicheng Yan
https://arxiv.org/pdf/1312.4400v3.pdf

[2] Network in Networks and 1x1 Convolutions
Andrew Ng
https://www.coursera.org/lecture/convolutional-neural-networks/networks-in-networks-and-1x1-convolutions-ZTb8x

[3] One by One [1x1] Convolution - counter-intuitively useful
Aaditya Prakash
https://iamaaditya.github.io/2016/03/one-by-one-convolution/

[4] Deep Learning series: Convolutional Neural Networks
Mike Cavaioni
https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524

[5] Going Deeper with Convolutions
Christian Szegedy et.al.
http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf

[6] A Simple Guide to the Versions of the Inception Network
Bharath Raj
https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

Part 4 : Regularization

Snippet Implementation

REFERENCES

[1] Network In Network
Min Lin, Qiang Chen, Shuicheng Yan
https://arxiv.org/pdf/1312.4400v3.pdf

[2] Network in Networks and 1x1 Convolutions
Andrew Ng
https://www.coursera.org/lecture/convolutional-neural-networks/networks-in-networks-and-1x1-convolutions-ZTb8x

[3] One by One [1x1] Convolution - counter-intuitively useful
Aaditya Prakash
https://iamaaditya.github.io/2016/03/one-by-one-convolution/

[4] Deep Learning series: Convolutional Neural Networks
Mike Cavaioni
https://medium.com/machine-learning-bites/deeplearning-series-convolutional-neural-networks-a9c2f2ee1524

[5] Going Deeper with Convolutions
Christian Szegedy et.al.
http://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf

[6] A Simple Guide to the Versions of the Inception Network
Bharath Raj
https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202

Part 5 : Optimization (Backprop)

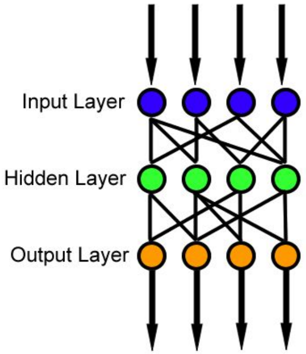
Neural Network Training Schema

Weights initialization

- uniform
- Gaussian
- Xavier

Forward pass

- Transformations



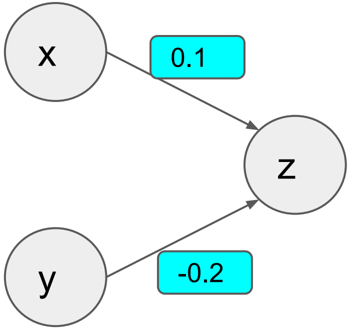
Backward pass

- Update the weights (with particular policy)
- (so that) predicted output to be closer to the target output
- (means) minimizing the error of the network
- (need) error calculation & weight update policy/rule

Weight Updates

Let's demonstrate "OR" gate with single layer perceptron (Neural Network)

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1



- 4 data samples
- 1 epoch = 4 iteration (w/ batch size of 1)
- Loss function => L2
- Optimization => SGD

- It is impossible to solve "OR" gate problem w/ single layer (w/ 2 neurons)
- "OR" gate is nonlinear function ⇒ some non-linearity is needed
  - Multi-layer
  - Activations
- But let's try single layer NN w/o any activation to see the weight update process

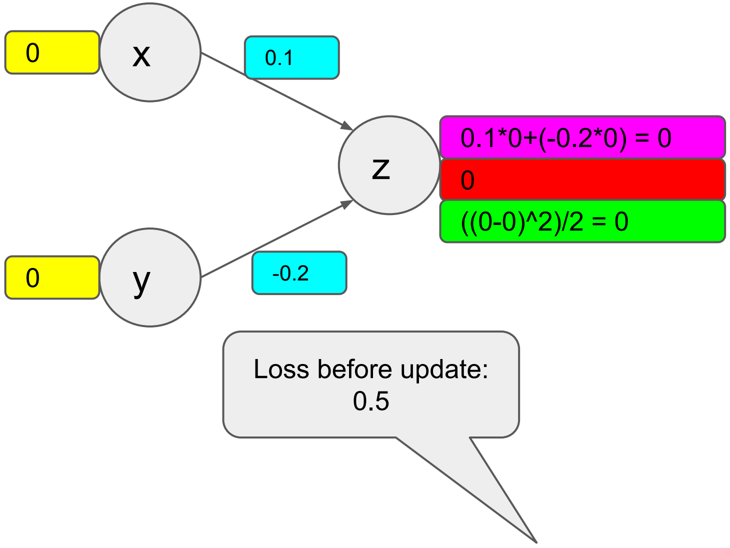
Weight Updates

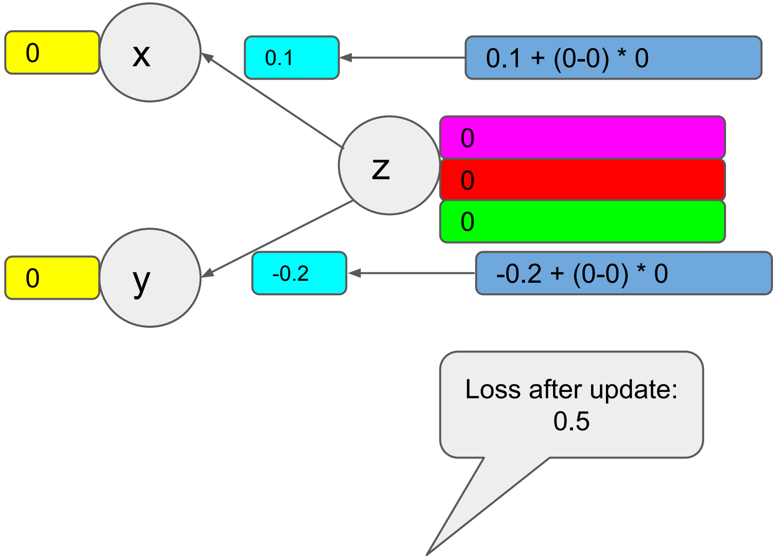
"OR" gate with single layer perceptron (Neural Network)

x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

$$w_j = w_j + \Delta w_j$$

$$\Delta w_j = (\text{target} - \text{predicted}) * x_j$$





- Target output
- Weights
- Predicted output
- Data samples
- Loss (L2)

Weight Updates

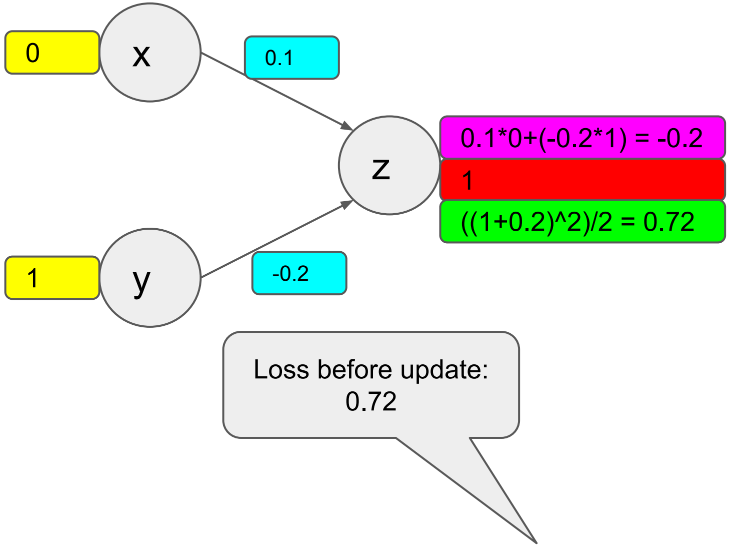
"OR" gate with single layer perceptron (Neural Network)

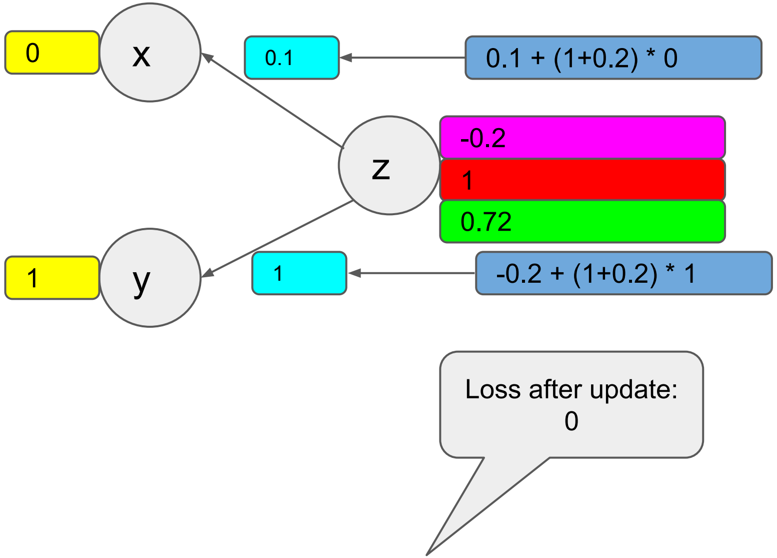
x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

Loss: (1/2)*(target - predicted)^2

$w_j = w_j + \Delta w_j$

$\Delta w_j = (\text{target - predicted}) * x_j$





Target output

Weights

Predicted output

Data samples

Loss (L2)

Weight Updates

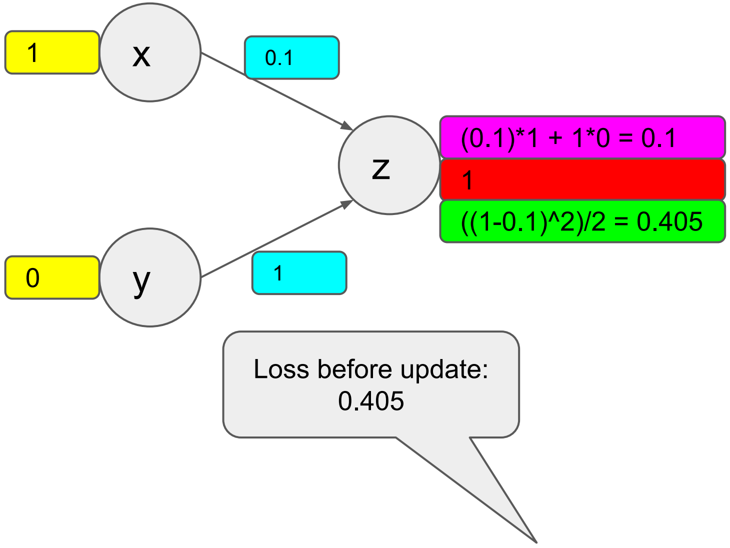
“OR” gate with single layer perceptron (Neural Network)

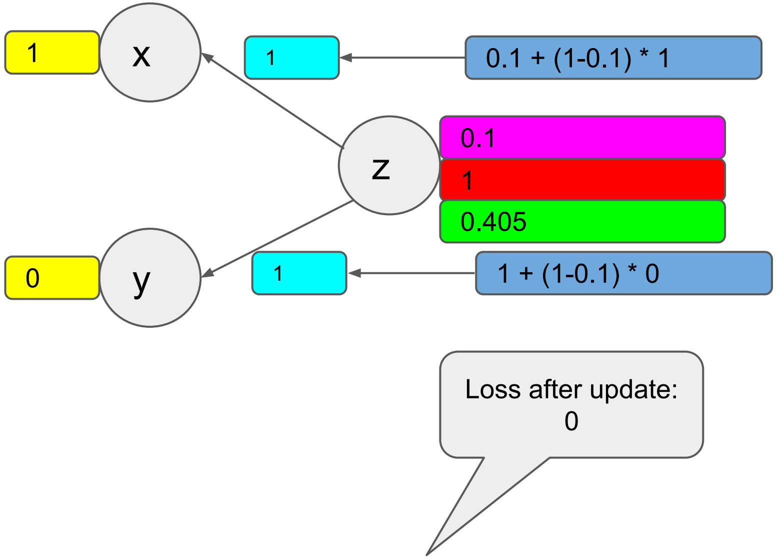
x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

Loss: (1/2)*(target - predicted)^2

$w_j = w_j + \Delta w_j$

$\Delta w_j = (\text{target - predicted}) * x_j$





Target output

Weights

Predicted output

Data samples

Loss (L2)

Weight Updates

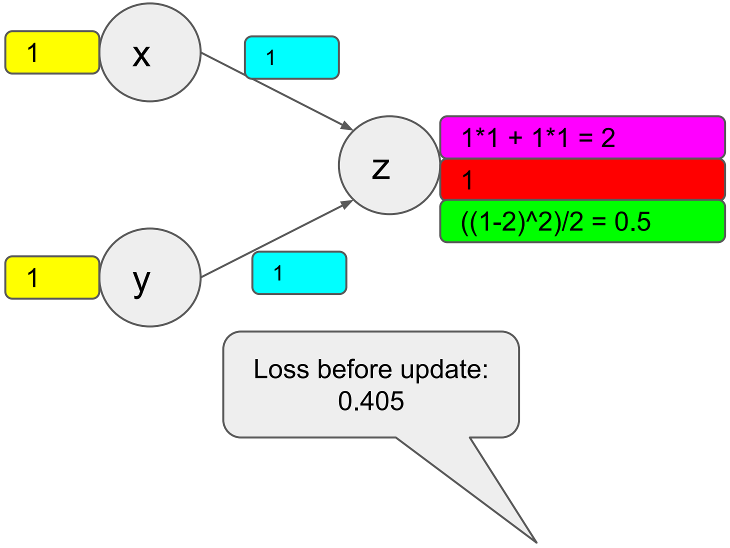
“OR” gate with single layer perceptron (Neural Network)

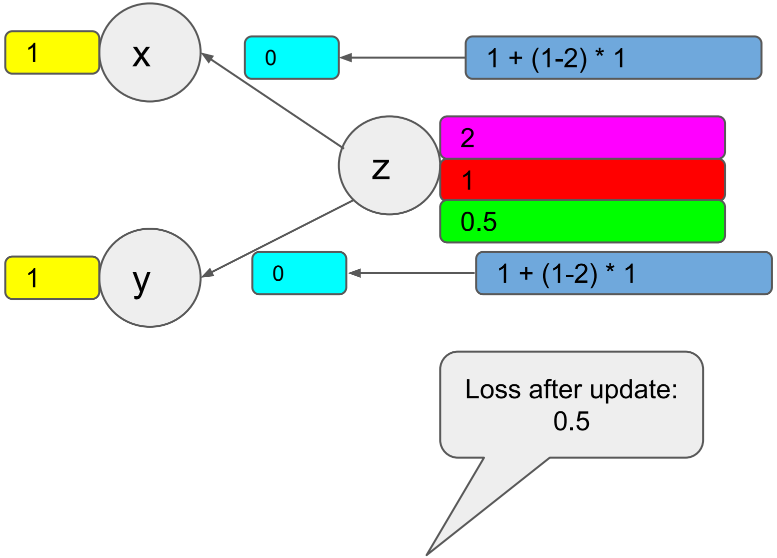
x	y	z
0	0	0
0	1	1
1	0	1
1	1	1

Loss: (1/2)*(target - predicted)^2

$w_j = w_j + \Delta w_j$

$\Delta w_j = (\text{target - predicted}) * x_j$





Target output

Weights

Predicted output

Data samples

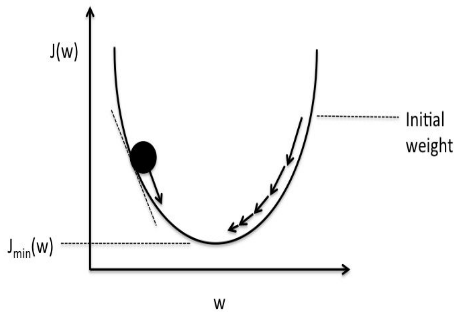
Loss (L2)

Weight Updates

Keynotes

Loss function is crucial:
Varies for different tasks
Weight initialization is important:
To converge faster
Activations:
To add non-linearity
Data pre-processing:
Normalization: mean & variance

learning rate:
Smaller → slower
Larger → faster



Weight update strategy:
Optimization algorithm

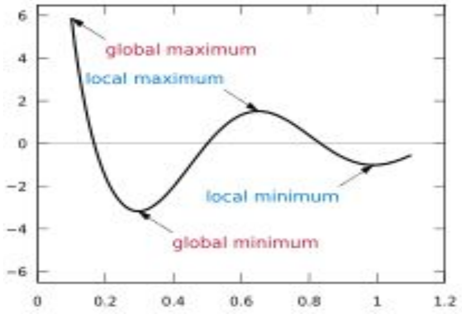
- Gradient Descent
- SGD
- Momentum
- AdaGrad
- Adam

Global/Local Minima: Loss function

Univariate function:

"univariate & non-linear function"

Input: x
Output: $f(x) = cos(3\pi x)/x$

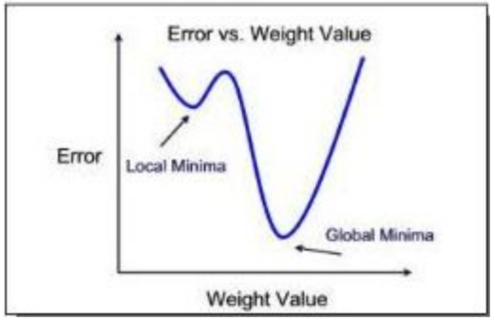


$y = f(x)$
$e = g(w)$

(Deep) NN Loss function:

"multivariate & non-linear function in hyperdimensional space"

Inputs: weights of the network
Output: $e=f(w)$ error/loss



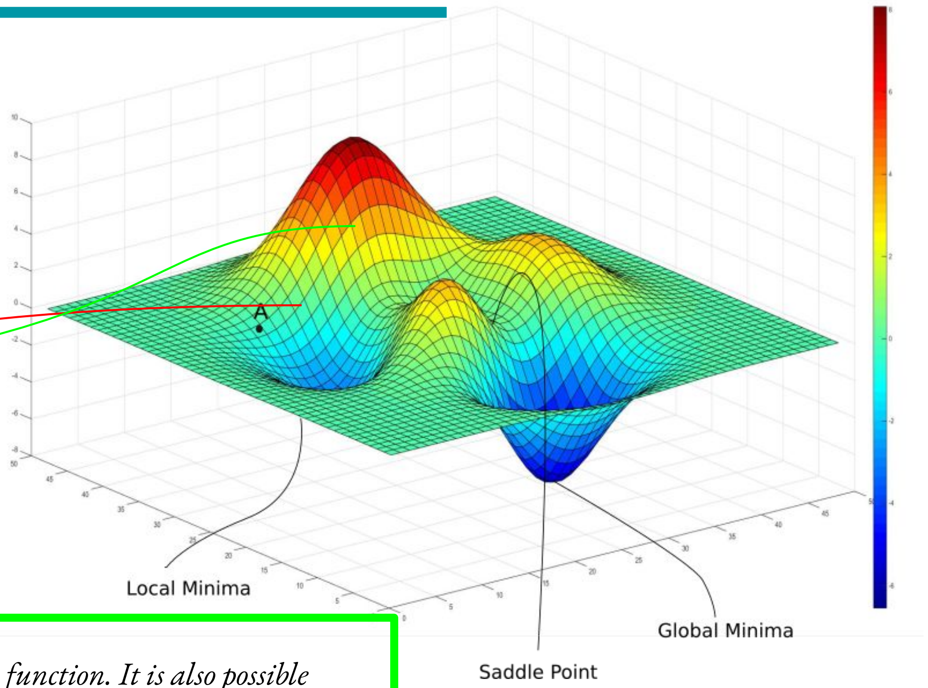
Global/Local Minima: Loss function

Loss function in Deep NNs:
- is not in two dimensional space
- is in hyperdimensional space

(in the figure it is shown in 3D)

Deep NN Optimization:
- random weight initialization
- the goal is to find the global minima
- but it is still okay to converge to local minima in deep learning

Quote from Deep Learning Book:



"There can be only one global minimum or multiple global minima of the function. It is also possible for there to be local minima that are not globally optimal. In the context of deep learning, we optimize functions that may have many local minima that are not optimal and many saddle points surrounded by very flat regions. All of this makes optimization difficult, especially when the input to the function is multidimensional. We therefore usually settle for finding a value off that is very low but not necessarily minimal in any formal sense."

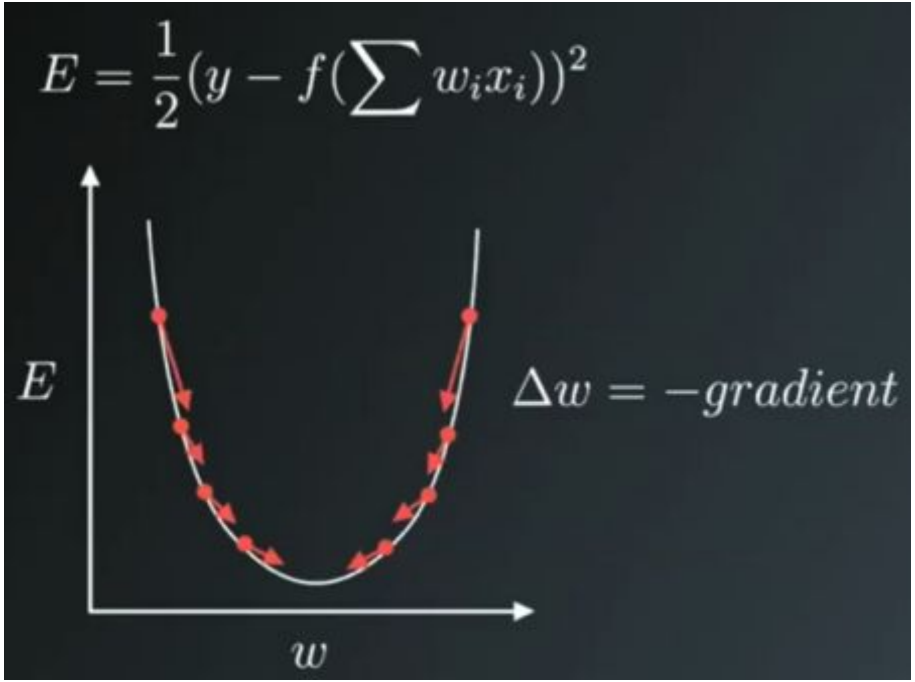
Optimization Algorithms

"There is no one clear optimization algorithm that outperforms the others - it primarily depends on user familiarity of hyperparameter tuning"

Gradient Descent:

$$\theta = \theta - \eta \cdot \nabla J(\theta)$$

→ η: learning rate
→ J(θ): loss function
→ ∇J(θ): gradient of loss function
→ update and tune model parameters in the direction that the loss function minimizes



Optimization Algorithms

"There is no one clear optimization algorithm that outperforms the others - it primarily depends on user familiarity of hyperparameter tuning"

Batch Gradient Descent:

Repeat until convergence {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^{m} \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}.$$

}

Stochastic Gradient Descent:

"stochastic": samples are randomly selected

Loop {

for i=1 to m, {

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)})\right) x_j^{(i)} \qquad \text{(for every } j\text{)}.$$

}

}

Optimization Algorithms

"There is no one clear optimization algorithm that outperforms the others - it primarily depends on user familiarity of hyperparameter tuning"

Batch Gradient Descent:

→ run through all training samples
→ then update the weights
→ slower convergence
→ preferred when training set is small

Stochastic Gradient Descent:

→ calculate an error for single sample
→ update the weights using that error
→ iterate over all training samples
→ faster convergence
→ preferred when training set is large
→ updates have high variance
→ This is "good thing" - helps to discover new and possibly better local minima
→ frequent loss fluctuations...

Optimization Algorithms

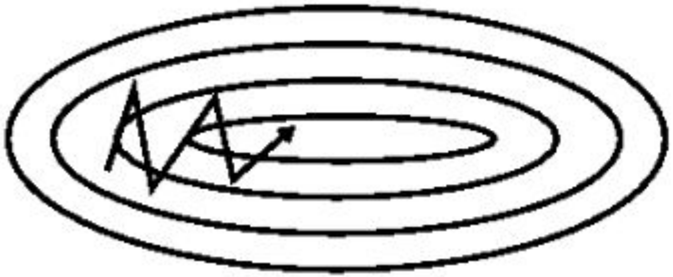
"There is no one clear optimization algorithm that outperforms the others - it primarily depends on user familiarity of hyperparameter tuning"

Challenges in SGD:

→ high variance oscillations makes it hard to reach convergence

SGD with Momentum:

→ method that helps accelerate SGD in the relevant direction and dampens oscillations



Optimization Algorithms

"There is no one clear optimization algorithm that outperforms the others - it primarily depends on user familiarity of hyperparameter tuning"

Momentum:

→ adds a fraction "γ" of the last update step
→ "γ" is usually selected as "0.9"

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

Analogy for the momentum:

→ Push a ball down a hill. The ball accumulates momentum becoming faster and faster on the way...

→ Similarly, the momentum term increases for dimensions whose gradients point in the same directions
→ reduces updates for dimensions whose gradients change direction.

→ we gain faster convergence and reduced oscillation

Optimization Algorithms

"There is no one clear optimization algorithm that outperforms the others - it primarily depends on user familiarity of hyperparameter tuning"

AdaGrad:

→ adapts the learning rate to the parameters
→ smaller updates for parameters associated with frequently occurring features
→ larger updates for parameters associated with infrequent features
→ well-suited for dealing with sparse data
→ most implementations use "lr=0.01" and leave as it is
→ **issue:** learning rate always decreases that causes to "no learning" after some point...

RMSProp:

→ divide the learning rate for a weight by a running average of the magnitudes of recent gradients for that weighs

Optimization Algorithms

"There is no one clear optimization algorithm that outperforms the others - it primarily depends on user familiarity of hyperparameter tuning"

Adam (Adaptive Moment Estimation):

→ is another method that computes adaptive learning rates for each parameter

→ uses second order momentums as well

→ behaves like a heavy ball with friction, thus prefers flat minima in the error surface

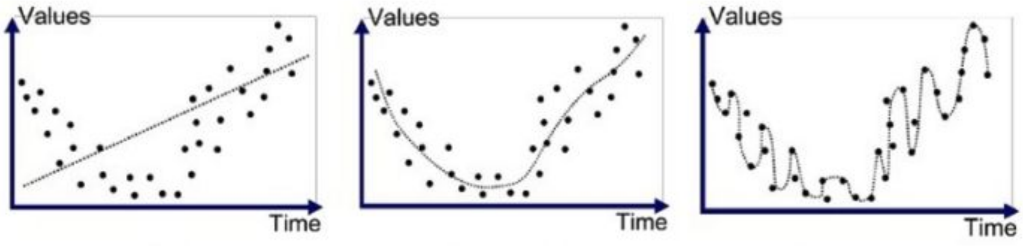
→ Adam works well in practice and outperforms other Adaptive techniques

Which Algorithms to use:

→ for sparse datasets use one of the adaptive learning-rate methodsy (you won't need to tune the learning rate)

→ if you want fast convergence and train Deep NN model, then use Adam or any other adaptive learning rate methods

Underfitting Vs. Overfitting



Underfitted | Good Fit/Robust | Overfitted

Underfitting:
poor performance on the training data and poor generalization to other data

→ solution: increase model's complexity

Overfitting:
good performance on the training data, poor generalization to other data

solution:
→ use regularization
→ use BN (Batch Normalization)
→ use dropout

Snippet Implementation

REFERENCES

[1] https://hmkcode.github.io/ai/backpropagation-step-by-step/

[2] https://sebastianraschka.com/Articles/2015_singlelayer_neurons.html

[3] https://academo.org/demos/3d-surface-plotter/

[4] http://www.cs.cornell.edu/boom/2004sp/projectarch/appofneuralnetworkcrystallography/NeuralNetworkAlgorithms.htm

[5] https://www.quora.com/How-do-I-overcome-a-local-minimum-problem-in-neural-networks

[6] https://jeffmacaluso.github.io/post/DeepLearningRulesOfThumb/

[7]https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f

[8] http://ruder.io/optimizing-gradient-descent/

[9] https://towardsdatascience.com/preventing-deep-neural-network-from-overfitting-953458db800

[10] https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms

Part 6 : Metrics for evaluation

Performance metrics for Deep NN

→ After (during) training our deep neural network model, we need to evaluate whether it is performing "good" or not.

Accuracy

→ this performance metrics would help us to know whether the model underfits/overfits or not

"There is no single magic performance metric for all the tasks"

Sensitivity

mAP

Recall

IoU

Precision

Specificity

TP	FP
TN	FN

TP, TN, FP, FN

→ TP: True Positive
→ TN: True Negative
→ FP: False Positive
→ FN: False Negatives
→ "True" & "False" are about ground true data
→ "Positive" & "Negative" are about prediction

TP, TN, FP, FN are used in the most of the metrics:

→ accuracy
→ recall
→ precision
→ sensitivity
→ specificity
→ f1-score

TP
correctly identified

FP
incorrectly identified

TN
correctly rejected

FN
incorrectly rejected

Accuracy

→ Mostly, used as a performance metric in classification problems
→ (number of correct predictions) / (number of all samples)
→ (TP+TN) / (TP+TN+FP+FN) == (TP+TN) / (number of all samples)

GT	1	0	1	1	0	1	0
Predictions	0	0	0	0	0	0	0

TP = 0

TN = 3

Accuracy = 43%

FP = 0

FN = 4

GT	1	0	1	1	0	1	0
Predictions	1	1	1	1	1	1	1

TP = 4

TN = 0

Accuracy = 57%

FP = 3

FN = 0

Accuracy

→ Mostly, used as a performance metric in classification problems
→ (number of correct predictions) / (number of all samples)
→ (TP+TN) / (TP+TN+FP+FN) == (TP+TN) / (number of all samples)

GT	1	0	1	1	0	1	0
Predictions	1	0	0	0	1	0	1

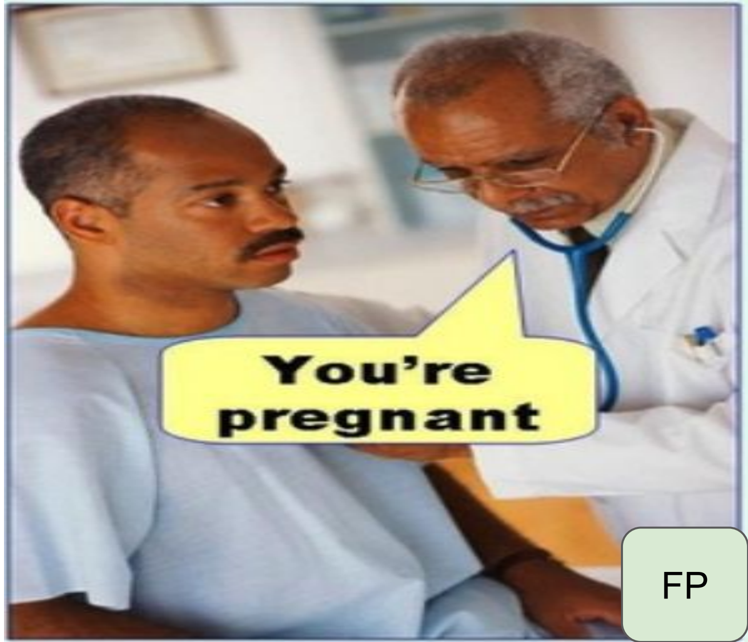
TP = 1
TN = 1
Accuracy = 28%
FP = 2
FN = 3

GT	1	0	1	1	0	1	0
Predictions	1	0	1	1	0	0	0

TP = 3
TN = 3
Accuracy = 86%
FP = 0
FN = 1

Which is worse?

FN

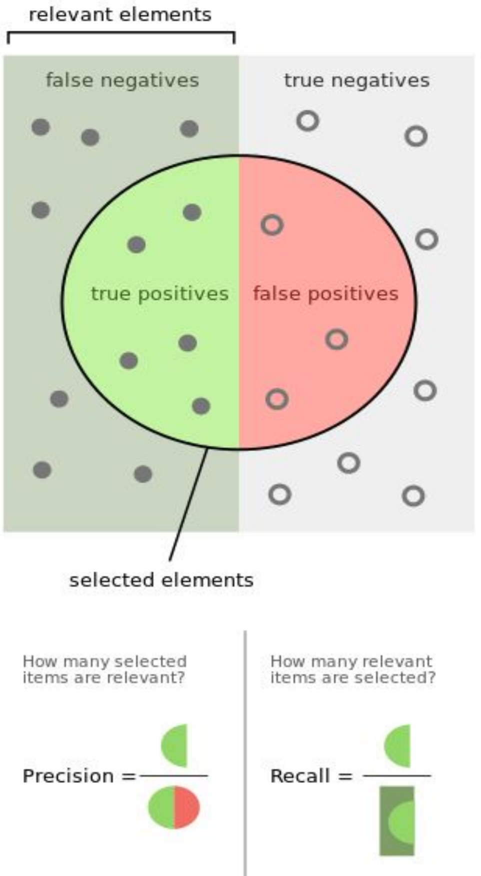


FP

Recall-Precision

→ Recall: what proportion of actual positives was predicted correctly?
(TP) / (TP+FN)
→ Precision: what proportion of positive predictions was actually correct?
(TP) / (TP+FP)

→**Note:** A model that produces no false negatives has a recall of 1.0.
If your prediction is all “1” (in binary classification) ⇒ recall = 1.0
→ **Note:** A model that produces no false positives has a precision of 1.0.
If your prediction is all “0” with at least one “1” (in binary classification)
⇒ precision = ~1.0



Recall-Precision

→ Recall: (TP) / (TP+FN)
→ Precision: (TP) / (TP+FP)

GT	1	0	1	1	0	1	0
Predictions	0	0	0	0	0	0	0

TP = 0

TN = 3

Recall = 0.0
Precision = 0.0

FP = 0

FN = 4

GT	1	0	1	1	0	1	0
Predictions	1	1	1	1	1	1	1

TP = 4

TN = 0

Recall = 1.0
Precision = 0.57

FP = 3

FN = 0

Recall-Precision

→ Recall: (TP) / (TP+FN)
→ Precision: (TP) / (TP+FP)

GT	1	0	1	1	0	1	0
Predictions	1	0	0	0	1	0	1

TP = 1
TN = 1
Recall = 0.25
Precision = 0.33
FP = 2
FN = 3

GT	1	0	1	1	0	1	0
Predictions	1	0	1	1	0	0	0

TP = 3
TN = 3
Recall = 0.75
Precision = 1.0
FP = 0
FN = 1

Object Detection Metrics

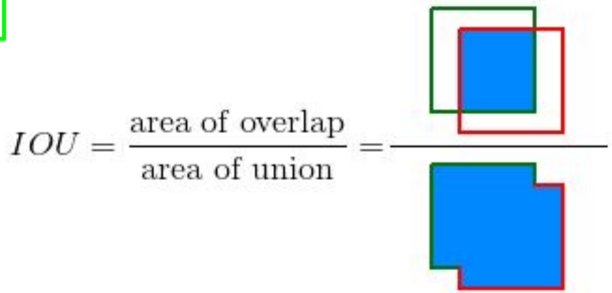
→ IoU: Intersection over union
→ Recall: mean Average Precision
→ Precision: mean Average Precision
→ mAP: mean Average Precision

IoU

→ measures the localization performance

→ define threshold to accept the detection either as True or False

→ used in NMS: to discard the multiple classes localized at the “same” location



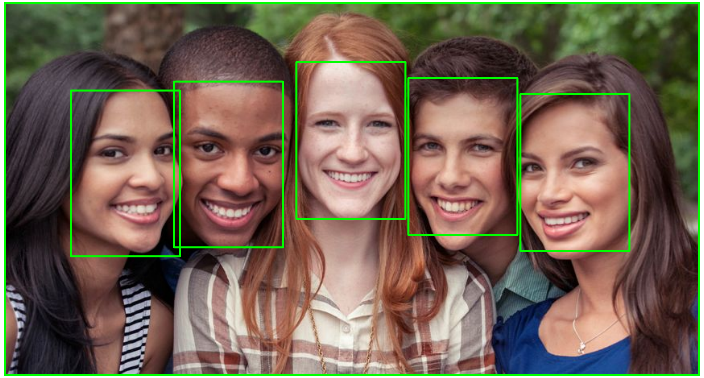
Object Detection Metrics

Recall-precision in face detection

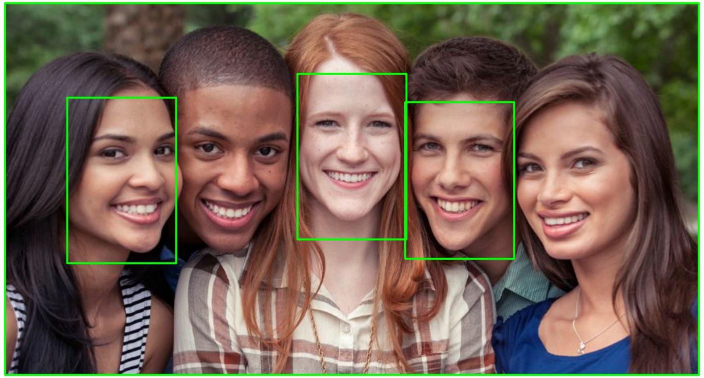
→ Recall: how many of faces did you detect?

→ Precision: how many of detected faces are really faces?

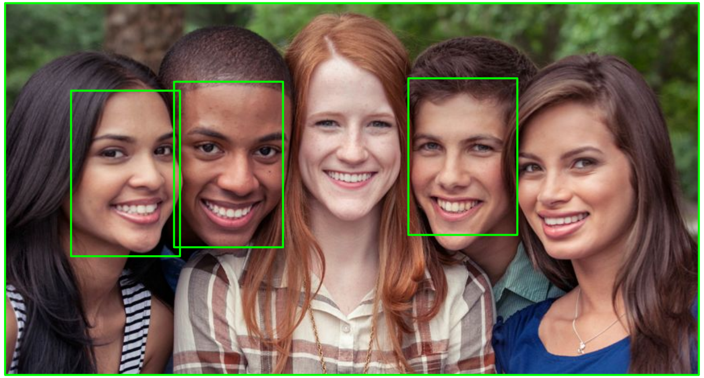
Object Detection Metrics



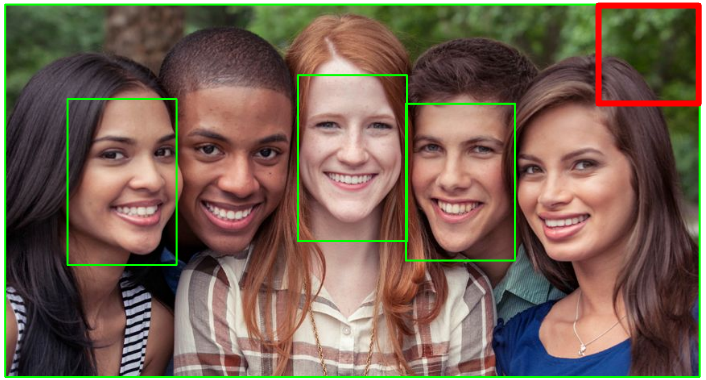
Recall: 1.0
Precision: 1.0



Recall: 0.6
Precision: 1.0



Recall: 0.6
Precision: 1.0



Recall: 0.6
Precision: 0.75

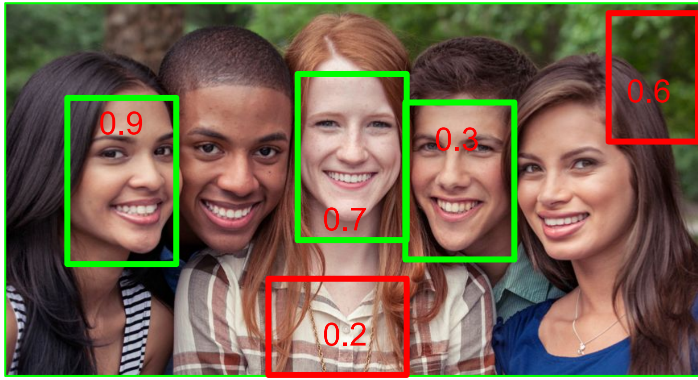
Object Detection Metrics

Average Precision (AP) & mAP

→ AP: Average of the maximum precisions at different recall values 2/ fixed IoU (IoU>0.5)

→ mAP: metric to measure the accuracy of object detectors. Mean of APs over all classes

rank	correct?	Precision	Recall
1(0.9)	true	1.0	0.2
2(0.7)	true	1.0	0.4
3(0.6)	false	0.66	0.4
4(0.3)	true	0.75	0.6
5(0.2)	false	0.6	0.6

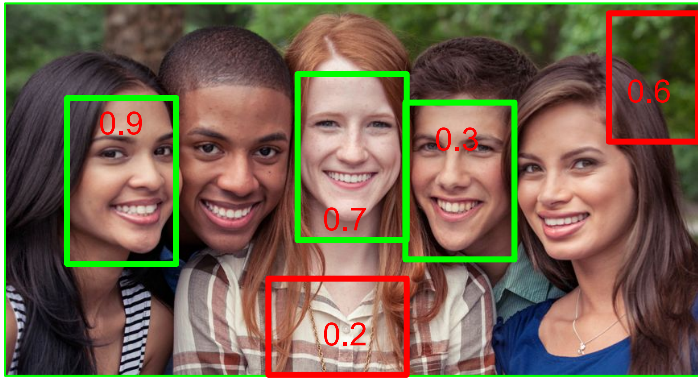


Recall: 0.6 Precision: 0.6

Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Precision	1.0	1.0	1.0	1.0	1.0	0.75	0.75	0.6	0.6	0.6	0.6

Object Detection Metrics

Recall: 0.6 Precision: 0.6



Average Precision (AP)

→ AP: Average of precision with varying confidence thresholds (w/ fixed IoU)

→ average of the maximum precisions at different recall values

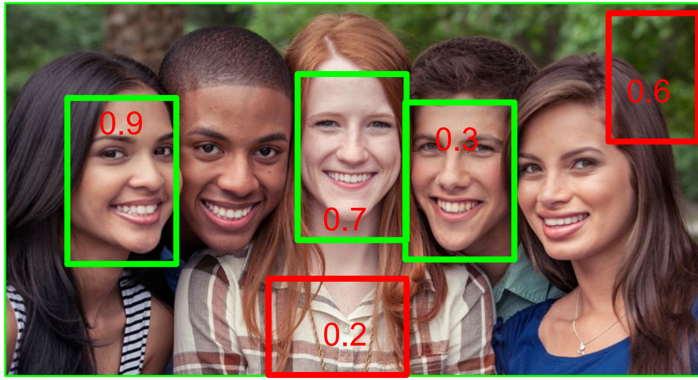
Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Precision	1.0	1.0	1.0	1.0	1.0	0.75	0.75	0.6	0.6	0.6	0.6

Average Precision (AP)

(5 x 1.0 + 0.75 x 2 + 0.6 x 4) / 11 = 8.9 / 11 = 0.81

Object Detection Metrics

Recall: 0.6 Precision: 0.6



CoCo dataset metrics (AP)

→ For COCO, AP is the average over multiple IoU.
→ AP@[.5:.95] - average AP for IoU from 0.5 to 0.95 with a step size of 0.05
Average of precision with varying confidence thresholds (w/ fixed IoU)

→ mAP@.75 - mAP with IoU=0.75 (over 80 classes)

Recall	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Precision	1.0	1.0	1.0	1.0	1.0	0.75	0.75	0.6	0.6	0.6	0.6

Mean Average Precision (AP)

(5 x 1.0 + 0.75 x 2 + 0.6 x 4) / 11 = 8.9 / 11 = 0.81

REFERENCES

[1] https://en.wikipedia.org/wiki/Precision_and_recall

[2] https://developers.google.com/machine-learning/crash-course/classification/precision-and-recall

[3] https://github.com/rafaelpadilla/Object-Detection-Metrics

[4] https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

[5] https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3