15.01.2025

Computer Vision (CS7.403)

Machine Learning Recap

Ravi Kiran

Makarand Tapaswi

Center for Visual Information Technology (CVIT)

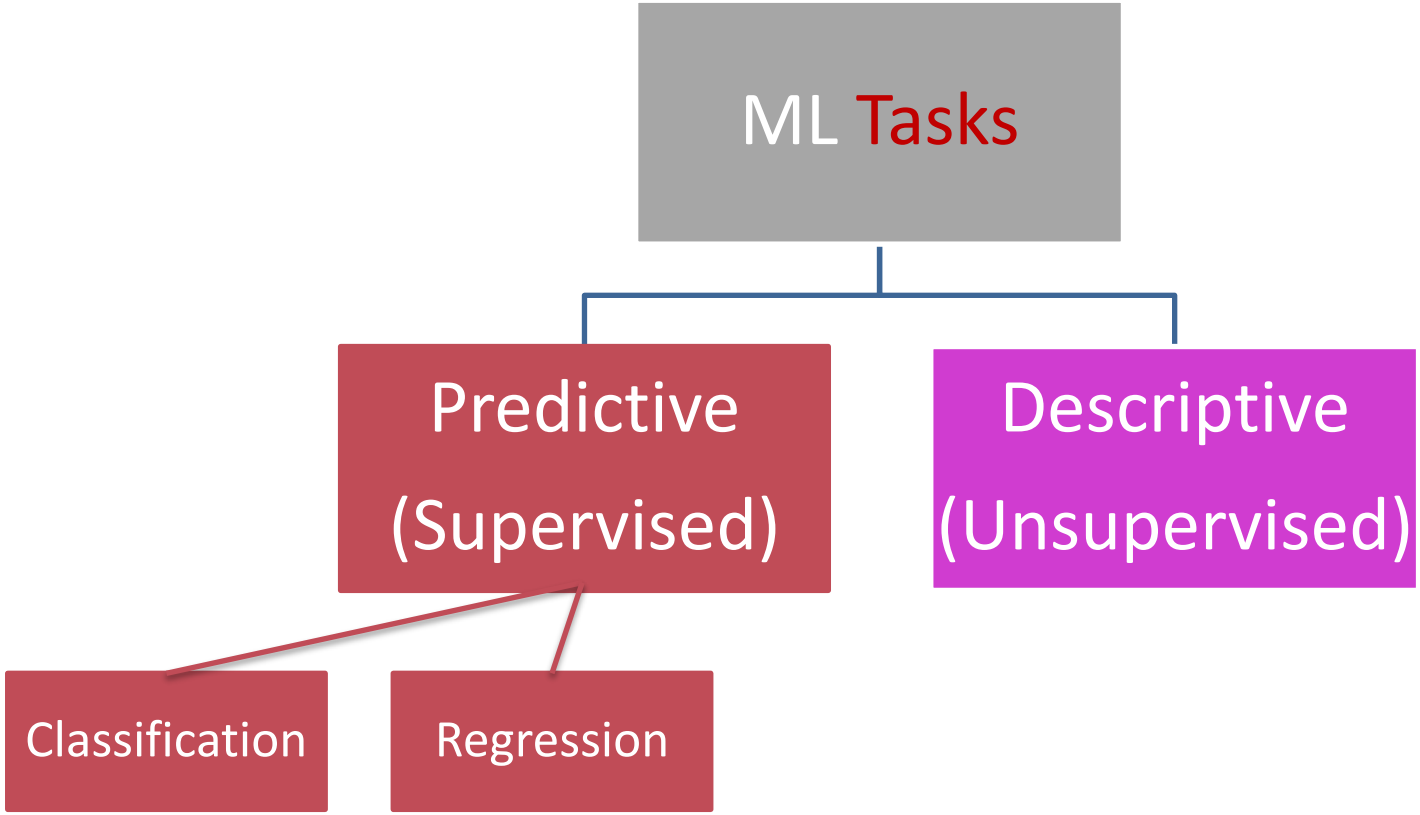
IIIT Hyderabad

Assignment Evals

Several students modified their code and reports prior to the evaluations and presented these altered versions during the assessment process.

You will get a straight 0 if you do any of the above

Some students whose code implementations were not functioning correctly included execution times, images, and data plots sourced from others in their reports.


ML Tasks
Predictive
(Supervised)
Descriptive
(Unsupervised)
Classification
Regression

ML::Tasks → Predictive → Classification

https://spectrum.ieee.org/ai-proves-trash-sorter-extraordinaire

Supervised Learning in Computer Vision

- Image Classification
  - x = raw pixels of the image, y = the main object

- There is too much info in raw data
- Relevant info is hidden
- Feature Extraction: Extract useful info (X) from raw data

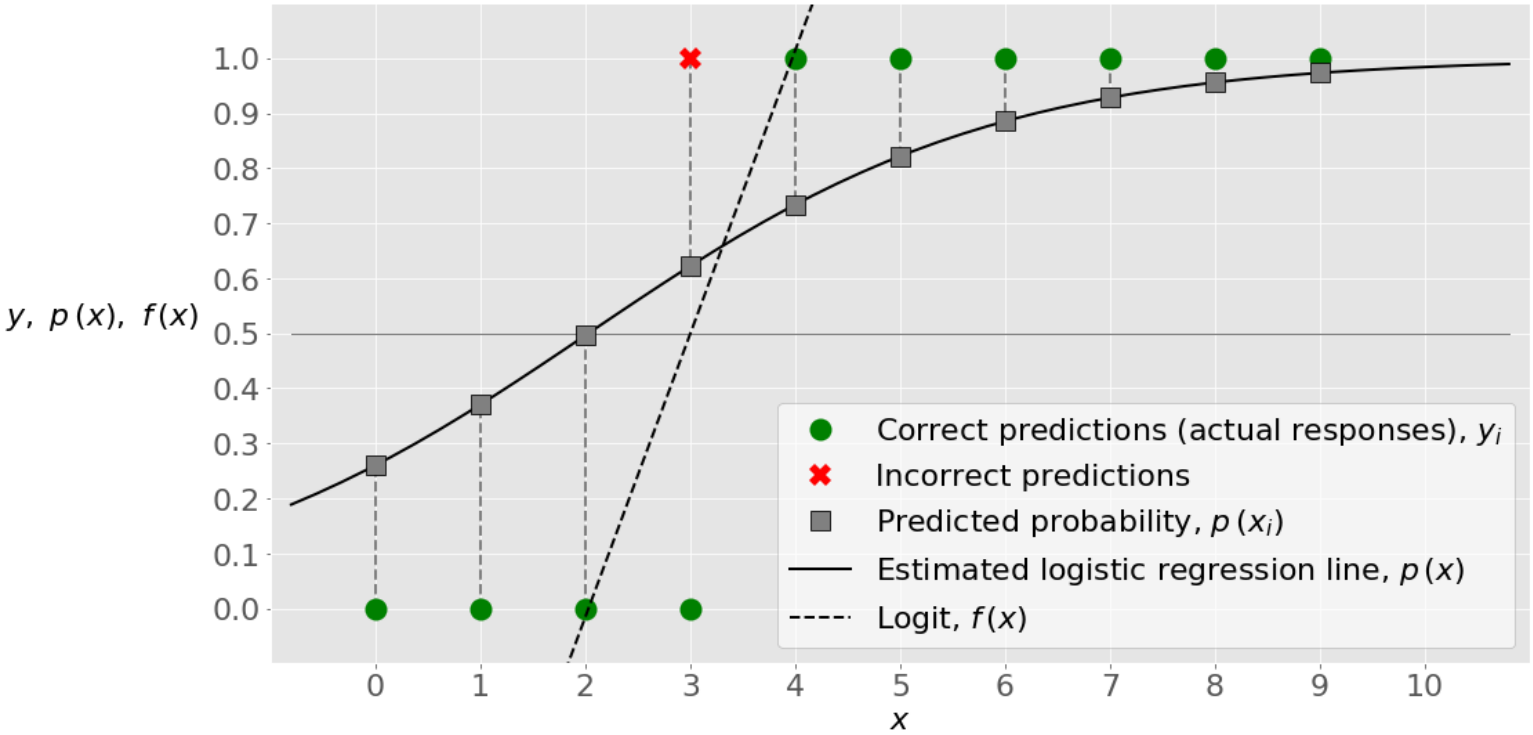
ImageNet Large Scale Visual Recognition Challenge. Russakovsky et al.'2015

Logistic Regression : Probabilistic Classifier

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$p(y \mid x; \theta) = (h_\theta(x))^y \, (1 - h_\theta(x))^{1-y}$$

$y \in \{0,1\}$



https://realpython.com/logistic-regression-python/

Logistic Regression - Learning parameters

$$\ell(\theta) = \log L(\theta)$$

$$= \sum_{i=1}^{m} y^{(i)} \log h(x^{(i)}) + (1 - y^{(i)}) \log(1 - h(x^{(i)}))$$

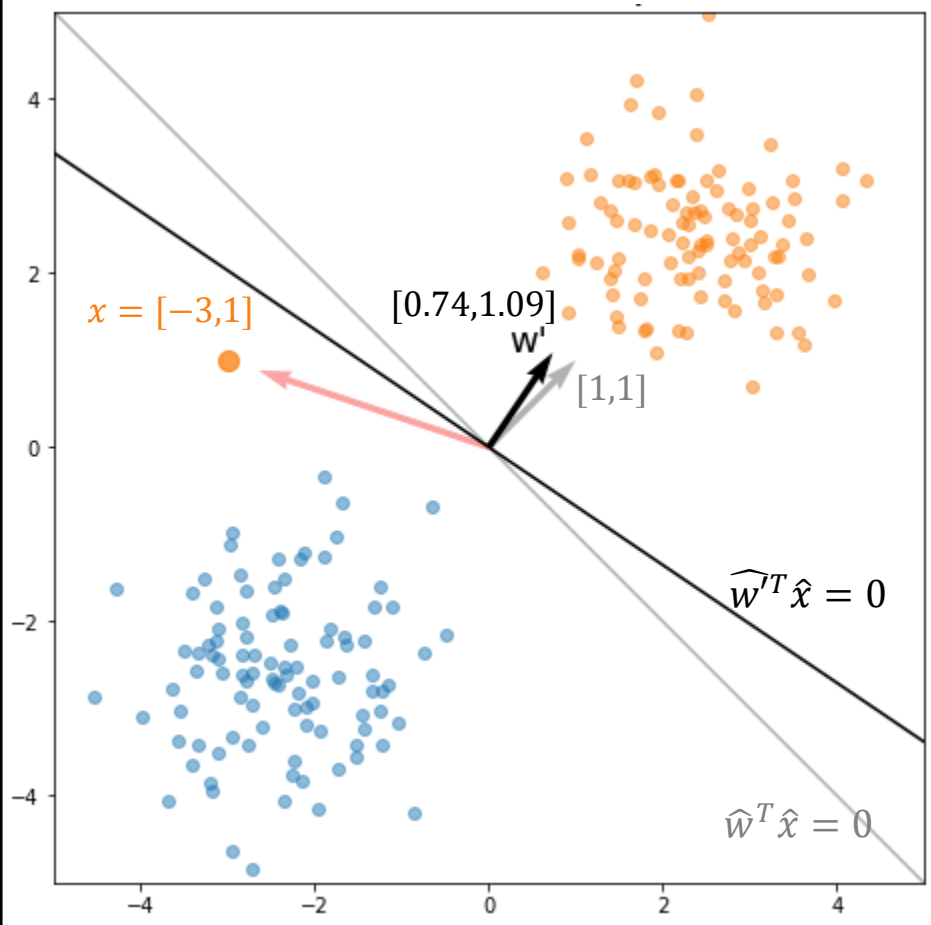
$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

$$\frac{\partial}{\partial \theta_j} \ell(\theta) = (y - h_\theta(x))\, x_j$$

No closed form.
Cannot set to 0 and solve for θ

$$\nabla_w L(\hat{y}^{(i)}, y^{(i)}) = (\hat{y}^{(i)} - y^{(i)})\hat{x}^{(i)}$$

Logistic Regression – Weight update using SGD



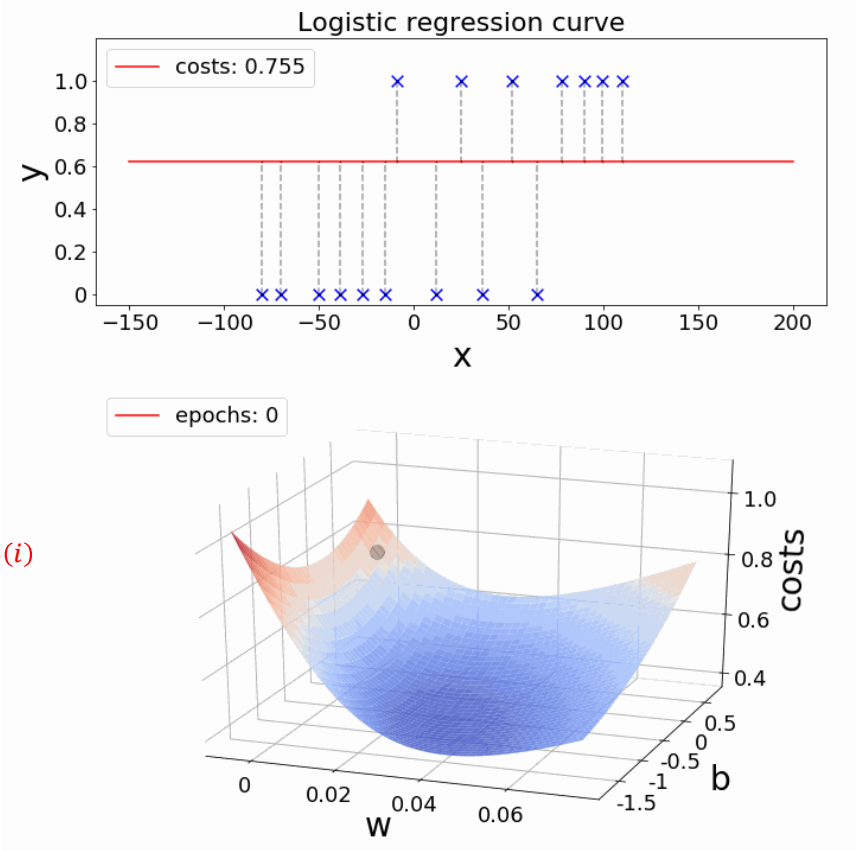
$$\nabla_w L(\hat{y}^{(i)}, y^{(i)}) = (\hat{y}^{(i)} - y^{(i)})\hat{x}^{(i)}$$

$$\hat{x}^{(i)} = \begin{bmatrix} 1 \\ x^{(i)} \end{bmatrix}$$

$$\hat{y}^{(i)} = \sigma(\hat{w}^T \hat{x}^{(i)})$$

$$\widehat{w'} = \hat{w} - \eta \nabla_w L(\hat{y}^{(i)}, y^{(i)})$$

Visualizing Logistic Regression: Optimization



$$\nabla_w L\left(\hat{y}^{(i)}, y^{(i)}\right) = \left(\hat{y}^{(i)} - y^{(i)}\right)\hat{x}^{(i)}$$

$$\widehat{\boldsymbol{w}'} = \widehat{\boldsymbol{w}} - \boldsymbol{\eta}\boldsymbol{\nabla}_{\boldsymbol{w}}\boldsymbol{L}\left(\widehat{\boldsymbol{y}}^{(\boldsymbol{i})}, \boldsymbol{y}^{(\boldsymbol{i})}\right)$$

https://towardsdatascience.com/animations-of-logistic-regression-with-python-31f8c9cb420

Multinomial Logistic Regression

$$\Pr(Y_i = 1) = \frac{e^{\beta_1' \cdot \mathbf{X}_i}}{1 + \sum_{k=1}^{K-1} e^{\beta_k' \cdot \mathbf{X}_i}}$$

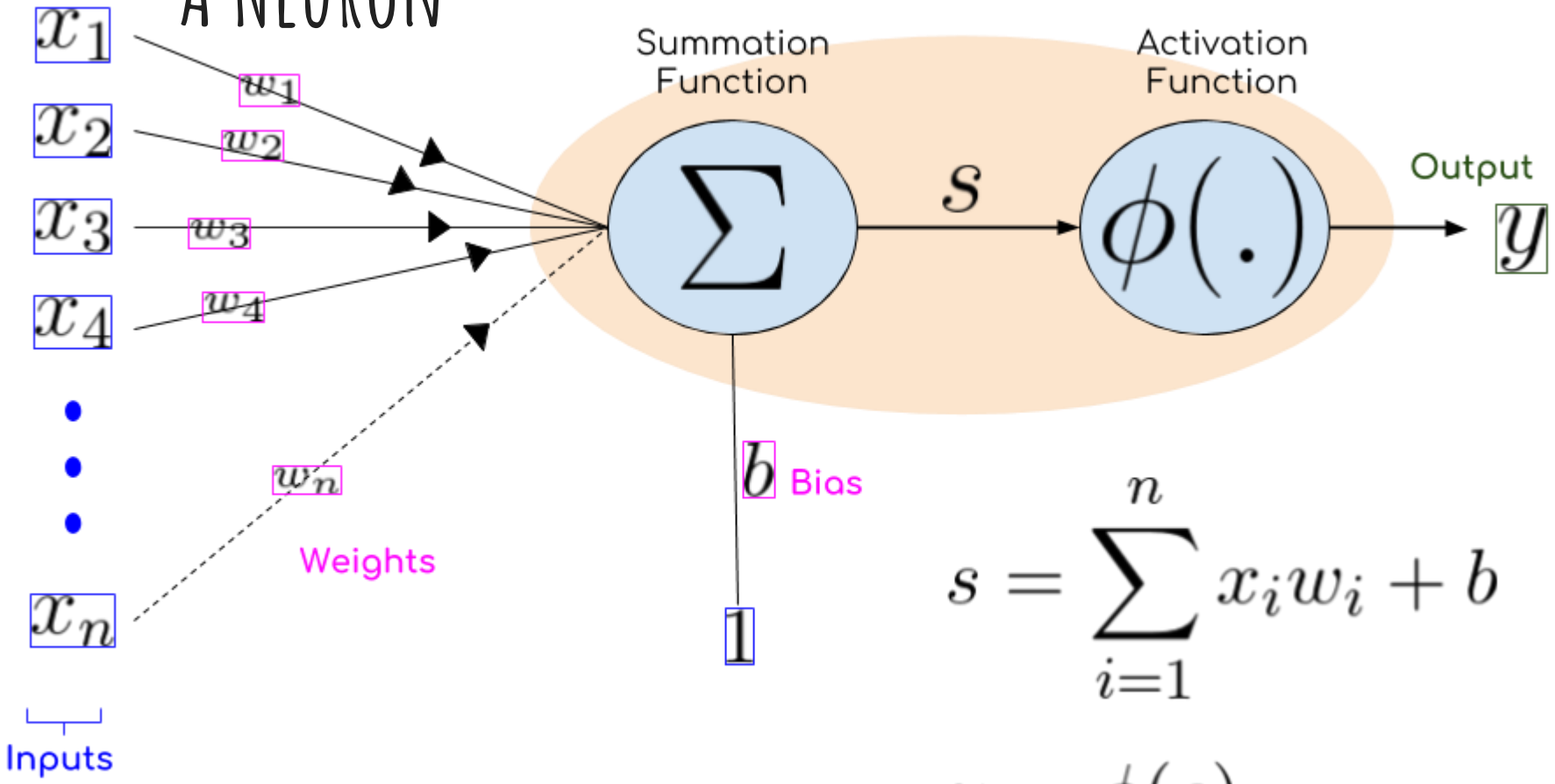
$$\cdots\cdots$$

$$\Pr(Y_i = K-1) = \frac{e^{\beta_{K-1}' \cdot \mathbf{X}_i}}{1 + \sum_{k=1}^{K-1} e^{\beta_k' \cdot \mathbf{X}_i}}$$

$$\Pr(Y_i = K) = \frac{1}{1 + \sum_{k=1}^{K-1} e^{\beta_k' \cdot \mathbf{X}_i}}$$

NEURAL NETWORKS

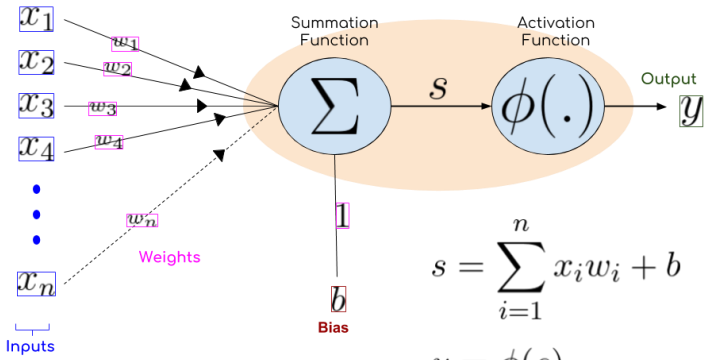
A Neuron



$$s = \sum_{i=1}^{n} x_i w_i + b$$

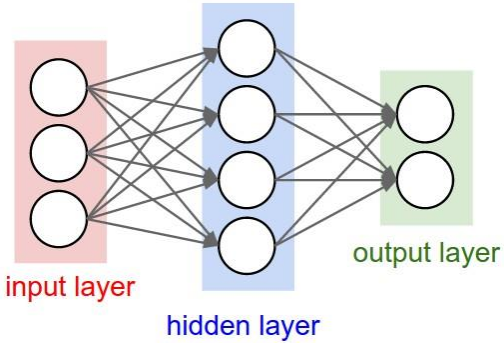
$$y = \phi(s)$$

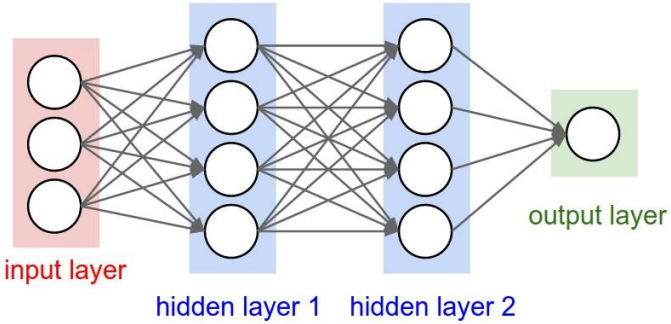
ACTIVATION FUNCTIONS

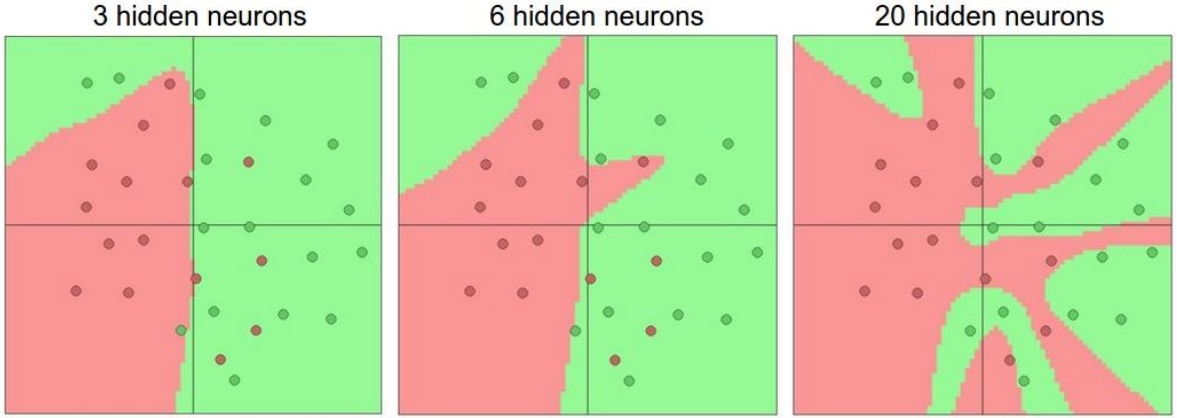


Activation Function	Equation	Example	1D Graph
Linear	$\phi(z) = z$	Adaline, linear regression	
Unit Step (Heaviside Function)	$\phi(z) = \begin{cases} 0 & z < 0 \\ 0.5 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Sign (signum)	$\phi(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ 1 & z > 0 \end{cases}$	Perceptron variant	
Piece-wise Linear	$\phi(z) = \begin{cases} 0 & z \le -½ \\ z + ½ & -½ \le z \le ½ \\ 1 & z \ge ½ \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \frac{1}{1 + e^{-z}}$	Logistic regression,	
Hyperbolic Tangent (tanh)	$\phi(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$		
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$		

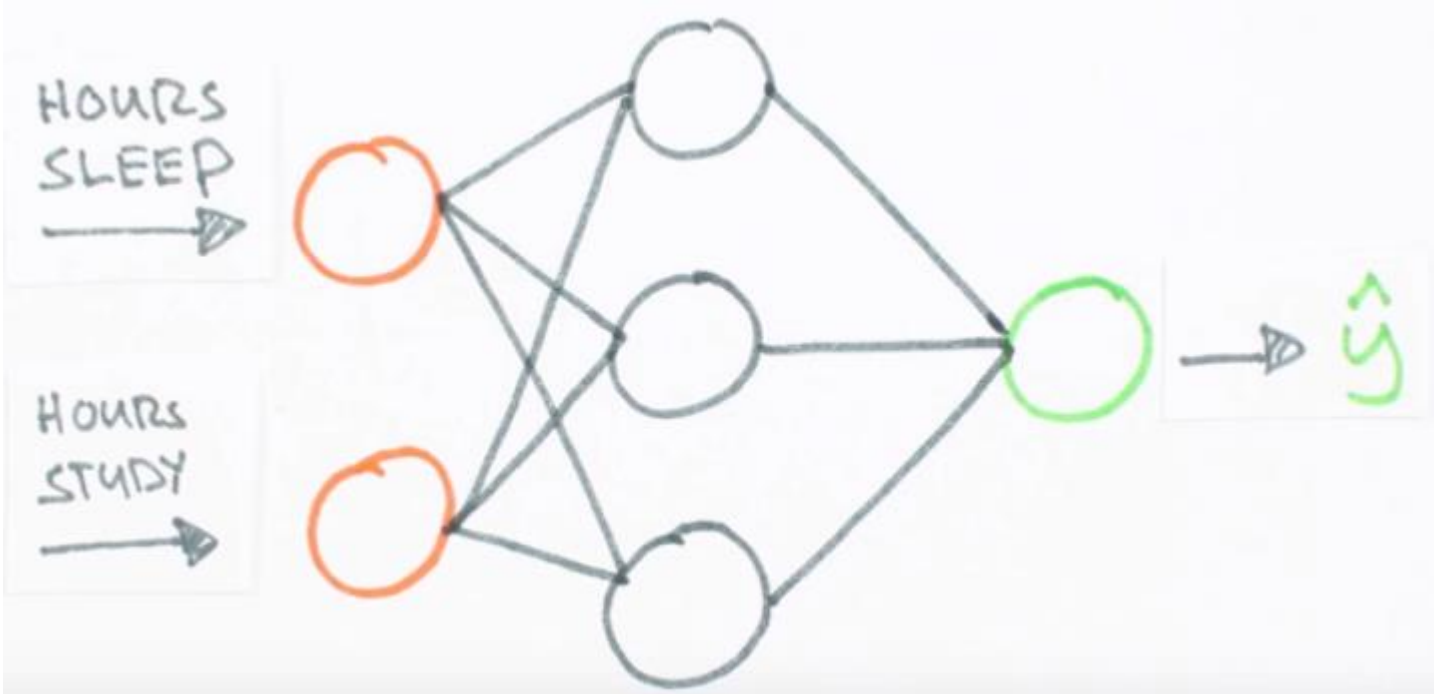
Why Use Only One Neuron ?





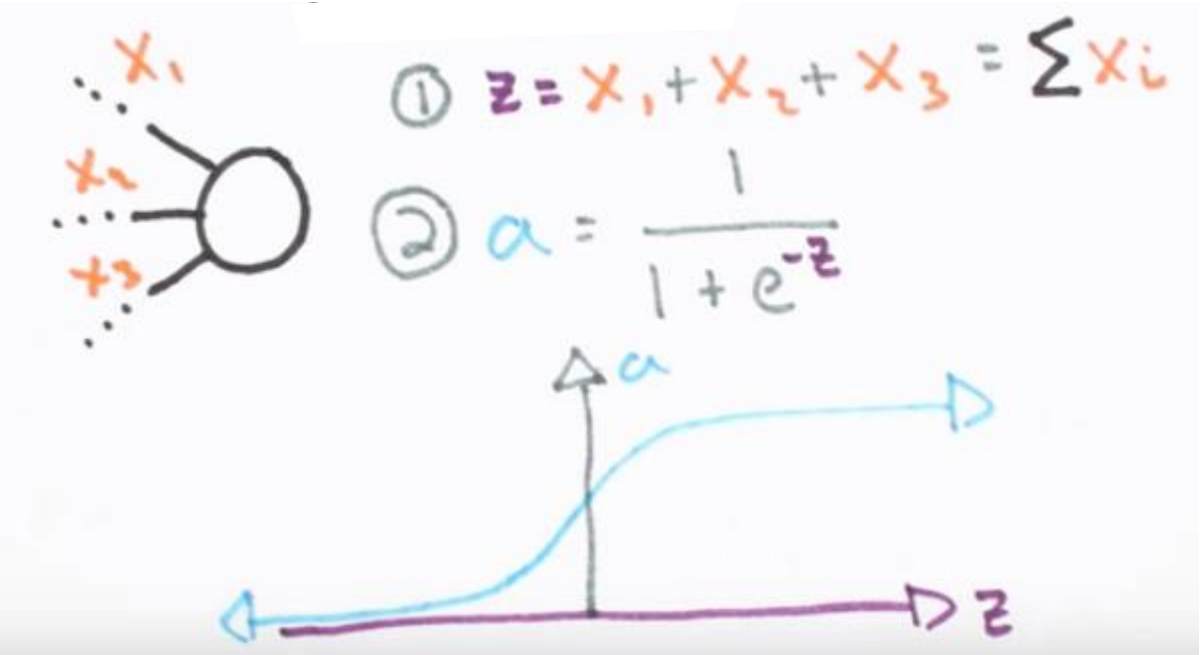


Multi-Neuron Networks :: Architecture



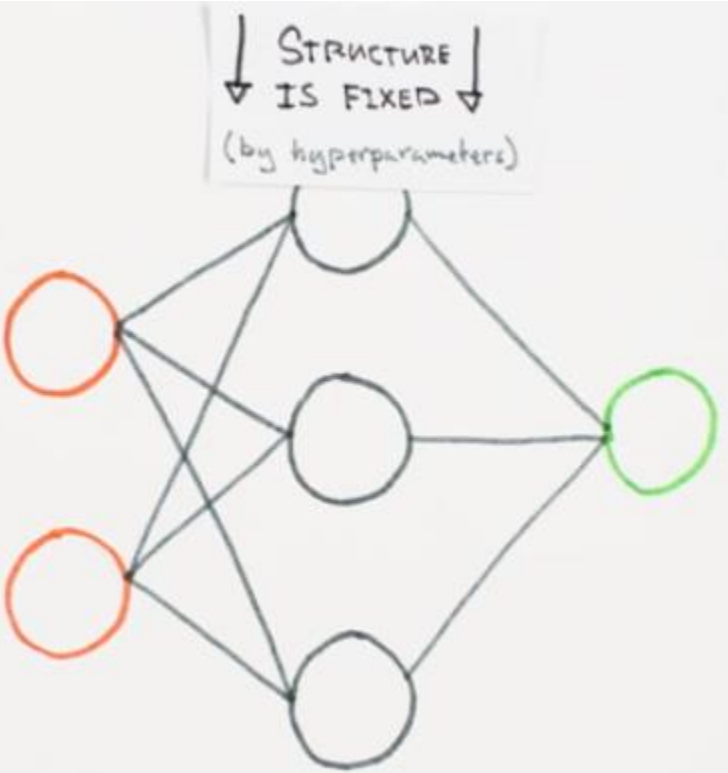
Multi-Neuron Networks :: Architecture

Neuron



Multi-Neuron Networks :: Architecture

```python
class Neural_Network(object):
    def __init__(self):
        #Define Hyperparameters
        self.inputLayerSize = 2
        self.outputLayerSize = 1
        self.hiddenLayerSize = 3

        #Weights (parameters)
        self.W1 = np.random.randn(self.inputLayerSize,self.hiddenLayerSize)
        self.W2 = np.random.randn(self.hiddenLayerSize,self.outputLayerSize)
```



Multi-Neuron Networks :: training

Initialize network with random weights

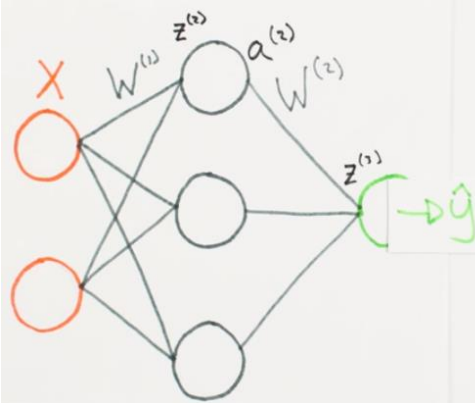
While [not converged]

Do forward prop

Do backprop and determine change in weights

Update ALL weights in ALL layers

Multi-Neuron Networks :: forward propagation



$$z^{(2)} = XW^{(1)} \quad (1)$$

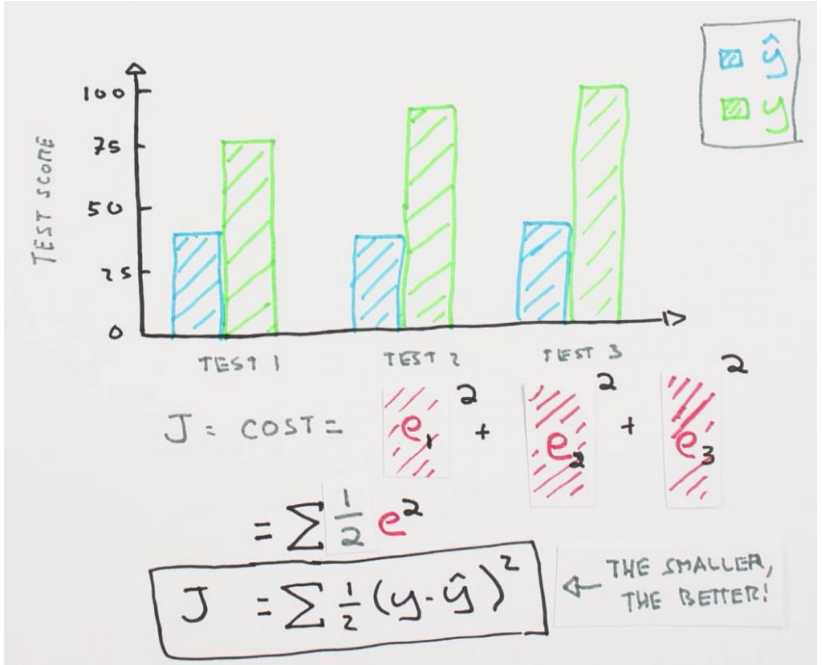
$$a^{(2)} = f(z^{(2)}) \quad (2)$$

$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$

$$\hat{y} = f(z^{(3)}) \quad (4)$$

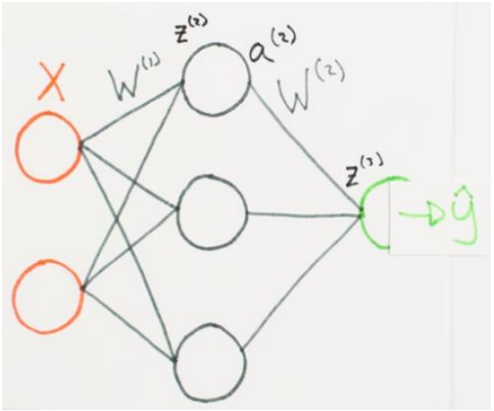
```python
def forward(self, X):
    #Propogate inputs though network
    self.z2 = np.dot(X, self.W1) # z2 = X * W1
    self.a2 = self.sigmoid(self.z2) # a2 = sigmoid(z2)
    self.z3 = np.dot(self.a2, self.W2) # z3 = a2 * W2
    yHat = self.sigmoid(self.z3) # yHat = sigmoid(z3)
    return yHat

def sigmoid(self, z):
    #Apply sigmoid activation function to scalar, vector, or matrix
    return 1/(1+np.exp(-z))
```

Multi-Neuron Networks :: Gradient Descent




```python
def costFunction(self, X, y):
    #Compute cost for given X,y, use weights already stored in class.
    self.yHat = self.forward(X)
    J = 0.5*sum((y-self.yHat)**2)
    return J
```

Multi-Neuron Networks :: Backpropagation



$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$

HOW DOES THIS CHANGE IF I CHANGE THESE?

$$\frac{\partial J}{\partial W}$$

$$W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \end{bmatrix}$$

$$\frac{\partial J}{\partial W^{(1)}} = \begin{bmatrix} \frac{\partial J}{\partial W_{11}^{(1)}} & \frac{\partial J}{\partial W_{12}^{(1)}} & \frac{\partial J}{\partial W_{13}^{(1)}} \\ \frac{\partial J}{\partial W_{21}^{(1)}} & \frac{\partial J}{\partial W_{22}^{(1)}} & \frac{\partial J}{\partial W_{23}^{(1)}} \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} W_{11}^{(2)} \\ W_{21}^{(2)} \\ W_{31}^{(2)} \end{bmatrix}$$

$$\frac{\partial J}{\partial W^{(2)}} = \begin{bmatrix} \partial J/\partial W_{11}^{(2)} \\ \partial J/\partial W_{21}^{(2)} \\ \partial J/\partial W_{31}^{(2)} \end{bmatrix}$$

Multi-Neuron Networks :: Backpropagation

$$J = \sum \frac{1}{2}(y - \hat{y})^2$$

$$z^{(2)} = XW^{(1)} \quad (1)$$
$$a^{(2)} = f(z^{(2)}) \quad (2)$$
$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$
$$\hat{y} = f(z^{(3)}) \quad (4)$$

$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$

How does this change if I change these?

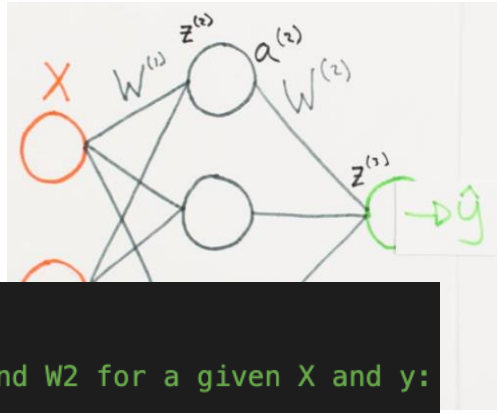
$$\frac{\partial J}{\partial W}$$

$$\frac{\partial J}{\partial W^{(2)}} = (a^{(2)})^T \delta^{(3)}$$

$$\delta^{(3)} = -(y - \hat{y}) f'(z^{(3)})$$

$$\frac{\partial J}{\partial W^{(1)}} = X^T \delta^{(2)}$$

$$\delta^{(2)} = \delta^{(3)} (W^{(2)})^T f'(z^{(2)})$$




```
# backpropagation
def costFunctionPrime(self, X, y):
    #Compute derivative with respect to W1 and W2 for a given X and y:
    self.yHat = self.forward(X)

    delta3 = np.multiply(-(y-self.yHat), self.sigmoidPrime(self.z3))
    dJdW2 = np.dot(self.a2.T, delta3)

    delta2 = np.dot(delta3, self.W2.T)*self.sigmoidPrime(self.z2)
    dJdW1 = np.dot(X.T, delta2)

    return dJdW1, dJdW2
```

Multi-Neuron Networks :: training

Initialize network with random weights

While [not converged]

Do forward prop

Do backprop and determine change in weights

Update All weights in All layers

```
NN = Neural_Network()
cost1 = NN.costFunction(X, y)
print('cost1=',cost1)
dJdW1, dJdW2 = NN.costFunctionPrime(X, y)
print('dJ/dW1=',dJdW1)
print('dJ/dW2=',dJdW2)
eta = 0.01
NN.W1 = NN.W1 - eta * dJdW1
NN.W2 = NN.W2 - eta * dJdW2
```

One Iteration

Multi-Neuron Networks :: Backpropagation

$$J = \sum \frac{1}{2}(y - \hat{y})^2$$

$$\frac{\partial J}{\partial W^{(2)}} = (a^{(2)})^T \delta^{(3)}$$

$$\delta^{(3)} = -(y - \hat{y}) f'(z^{(3)})$$

$$\frac{\partial J}{\partial W^{(1)}} = X^T \delta^{(2)}$$

$$\delta^{(2)} = \delta^{(3)} (W^{(2)})^T f'(z^{(2)})$$

$$z^{(2)} = XW^{(1)} \quad (1)$$

$$a^{(2)} = f(z^{(2)}) \quad (2)$$

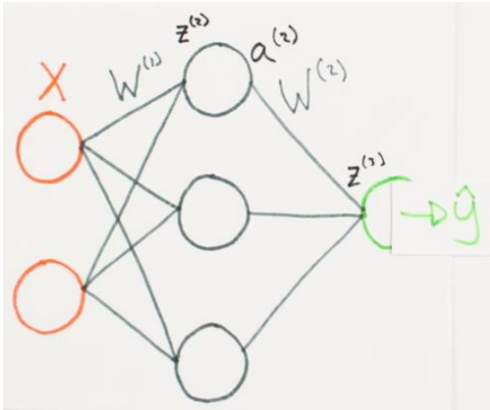
$$z^{(3)} = a^{(2)} W^{(2)} \quad (3)$$

$$\hat{y} = f(z^{(3)}) \quad (4)$$

$$J = \sum \frac{1}{2}(y - f(f(XW^{(1)})W^{(2)}))^2$$

HOW DOES THIS CHANGE IF I CHANGE THESE?

$$\frac{\partial J}{\partial W}$$




```
NN = Neural_Network()
cost1 = NN.costFunction(X, y)
print('cost1=',cost1)
dJdW1, dJdW2 = NN.costFunctionPrime(X, y)
print('dJ/dW1=',dJdW1)
print('dJ/dW2=',dJdW2)
eta = 0.01
NN.W1 = NN.W1 - eta * dJdW1
NN.W2 = NN.W2 - eta * dJdW2
cost2 = NN.costFunction(X, y)
print('cost2=',cost2)
```

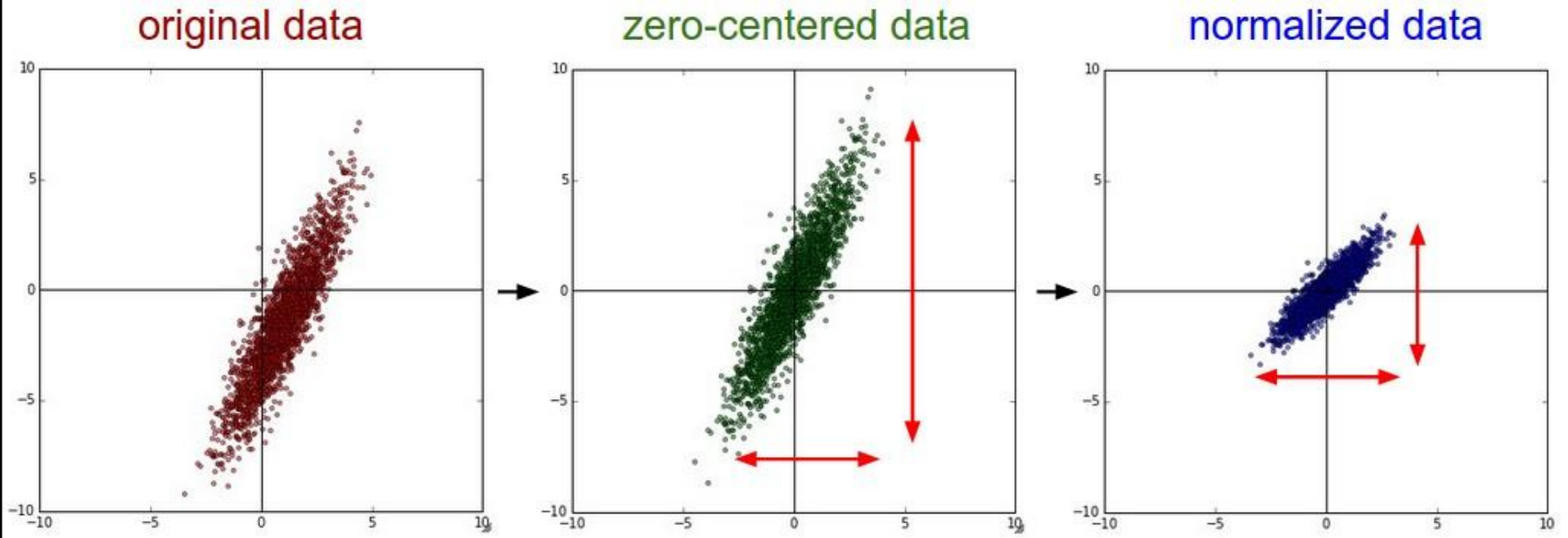
```
cost1= [0.44735371]
dJ/dW1= [[-0.08913117 -0.04750461 -0.00562623]
 [-0.05862425 -0.03130539 -0.00351033]]
dJ/dW2= [[-0.4110688 ]
 [-0.37530217]
 [-0.4590466 ]]
cost2= [0.44202336]
```

$$W^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{12}^{(1)} & W_{13}^{(1)} \\ W_{21}^{(1)} & W_{22}^{(1)} & W_{23}^{(1)} \end{bmatrix} \qquad \frac{\partial J}{\partial W^{(1)}} = \begin{bmatrix} \frac{\partial J}{\partial W_{11}^{(1)}} & \frac{\partial J}{\partial W_{12}^{(1)}} & \frac{\partial J}{\partial W_{13}^{(1)}} \\ \frac{\partial J}{\partial W_{21}^{(1)}} & \frac{\partial J}{\partial W_{22}^{(1)}} & \frac{\partial J}{\partial W_{23}^{(1)}} \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} W_{11}^{(2)} \\ W_{21}^{(2)} \\ W_{31}^{(2)} \end{bmatrix} \qquad \frac{\partial J}{\partial W^{(2)}} = \begin{bmatrix} \partial J/\partial W_{11}^{(2)} \\ \partial J/\partial W_{21}^{(2)} \\ \partial J/\partial W_{31}^{(2)} \end{bmatrix}$$

Data Setup

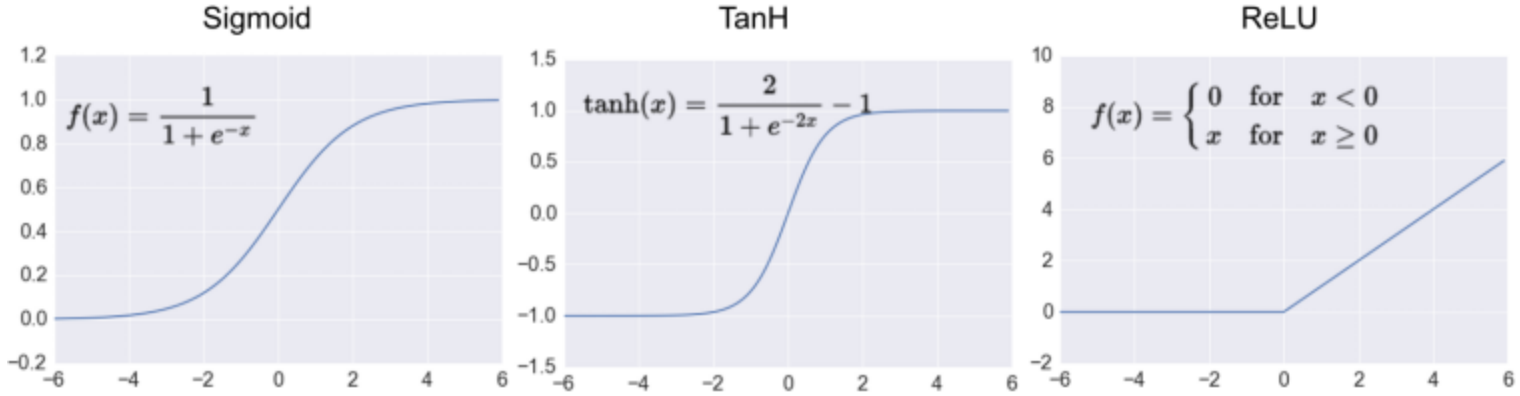
- Preprocessing:



Weight initialization

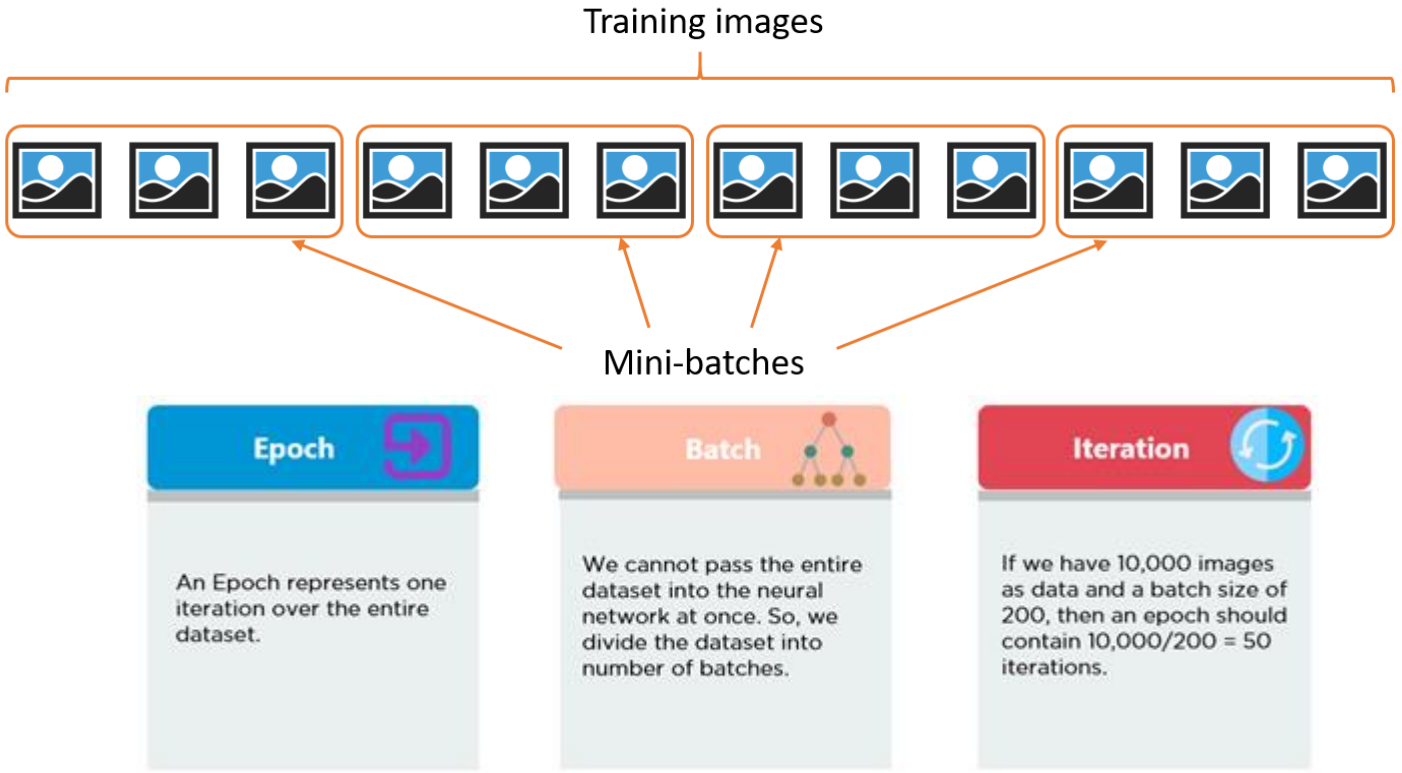
- All Zeros
- Random [0,1]
- Random [-1,1]
- `w = np.random.randn(n) * sqrt(2.0/n), n = # of inputs to neuron`

Activation functions



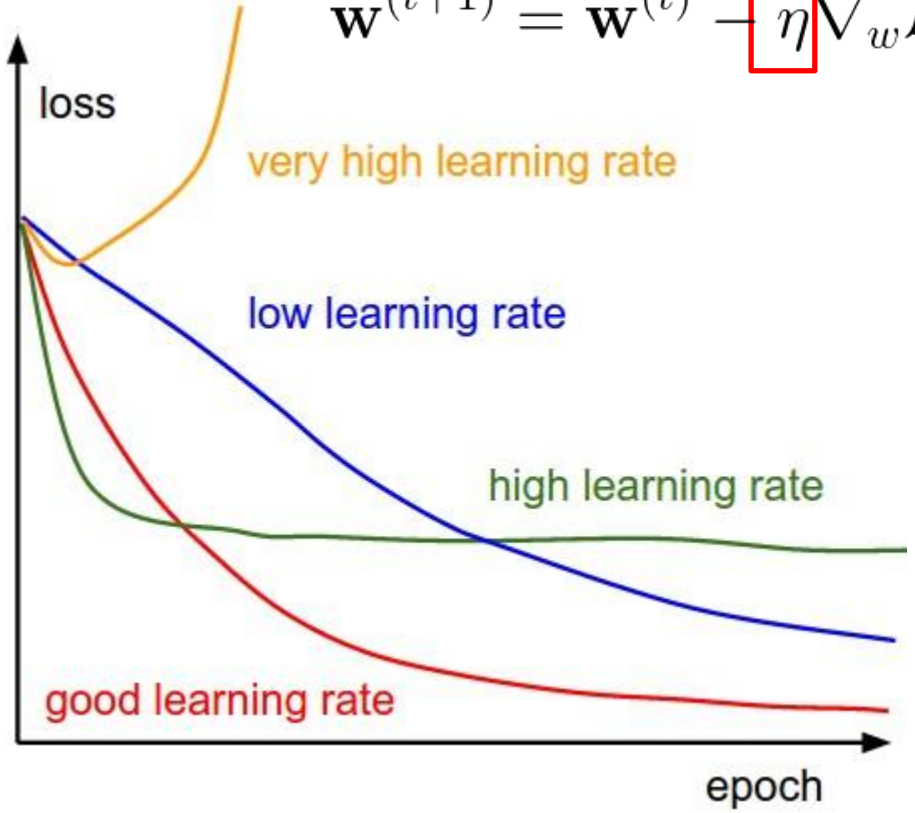
MINIBATCH VS SINGLE

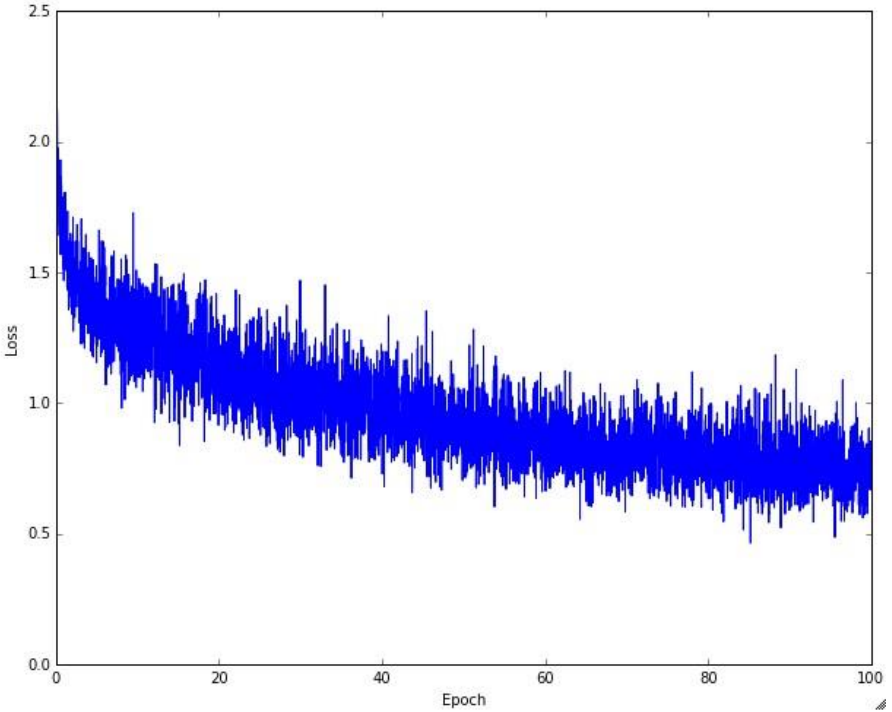
- Average error, gradients



Training – setting learning rate

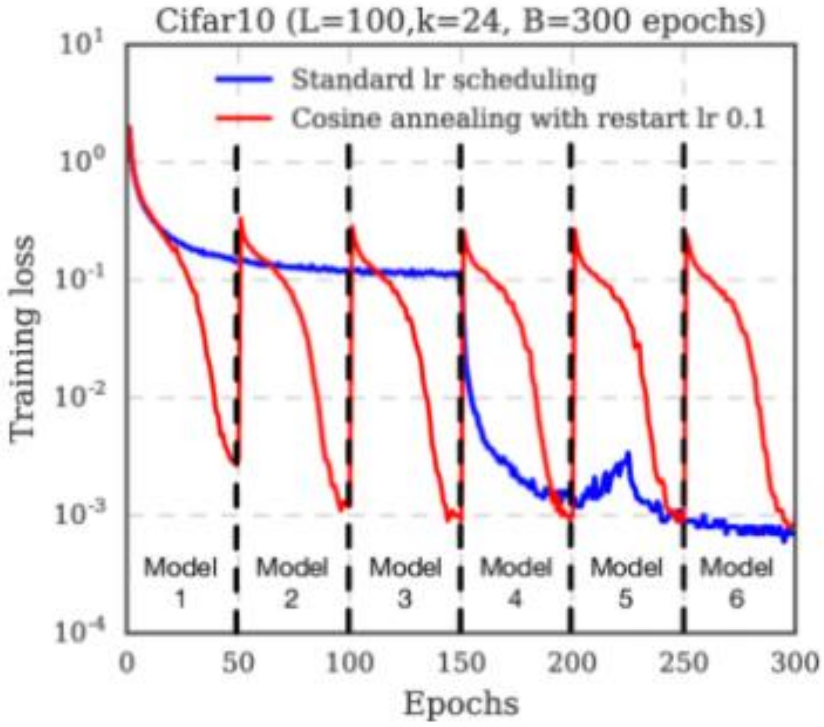
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla_w \mathcal{L}(\mathbf{w})$$





Training – setting learning rate

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta^{(t)} \nabla_{\mathbf{w}} \mathbf{J}(\mathbf{w})$$



lossfunctions.tumblr.com

Search lossfunctions

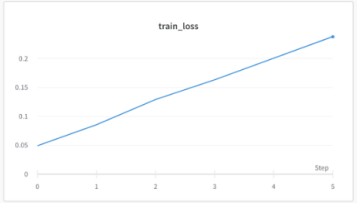
lossfunctions

They are a window to your model's heart.

Contribute loss functions to @karpathy. It doesn't matter if your loss functions are flat, converge, diverge, step or oscillate (or any combination of the above). All loss functions are computed beautiful in their own way.

POSTS ARCHIVE

Charts 1

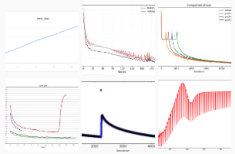


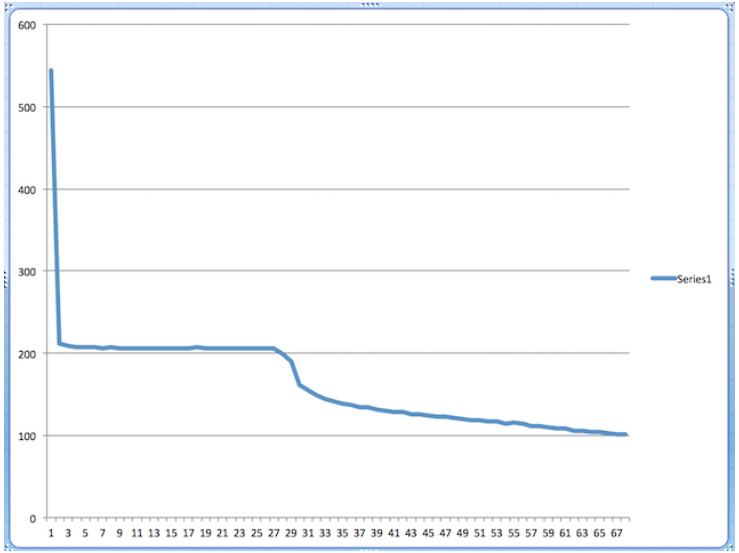
Tired: loss minimalists. Wired: loss maximalists.

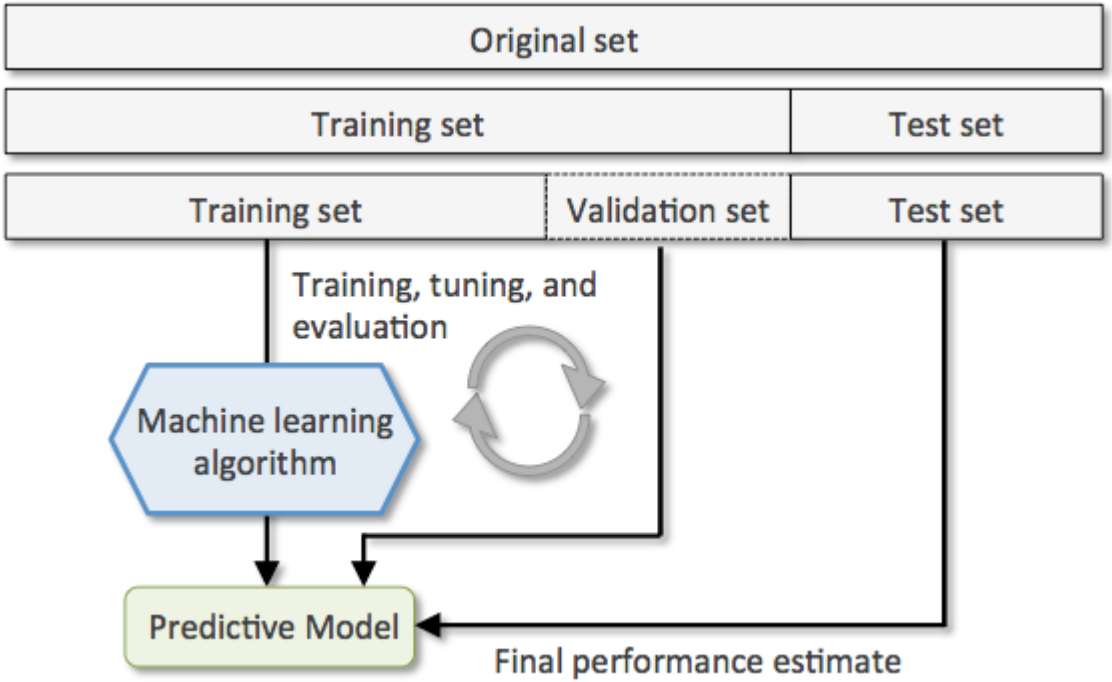
by @sharifshameem :)

3 notes

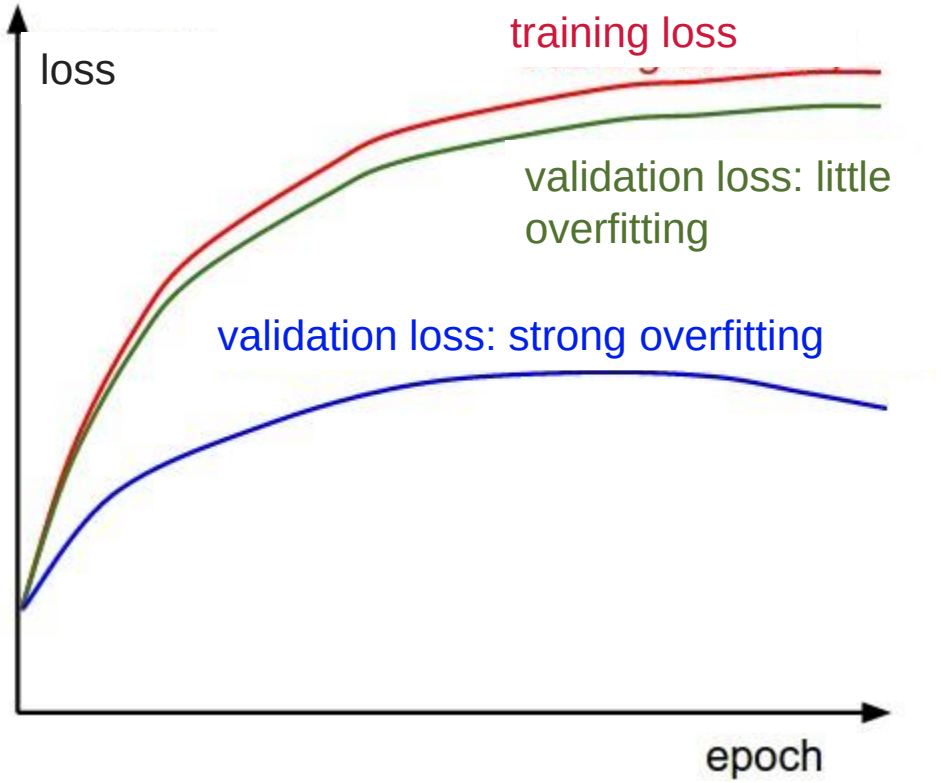
TOP PHOTOS



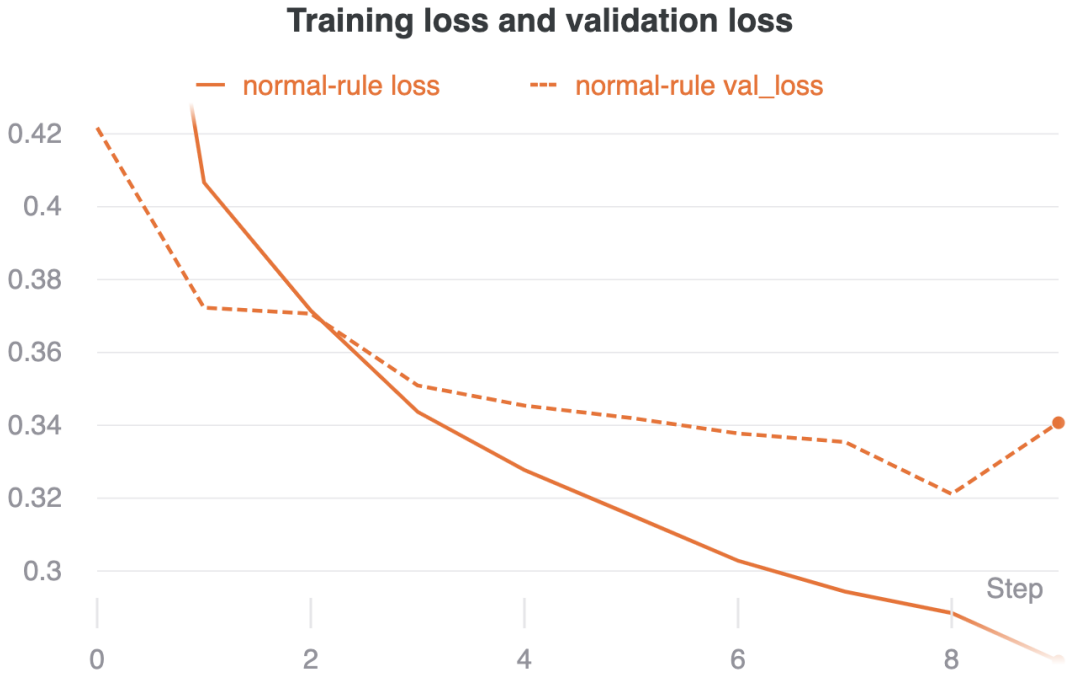


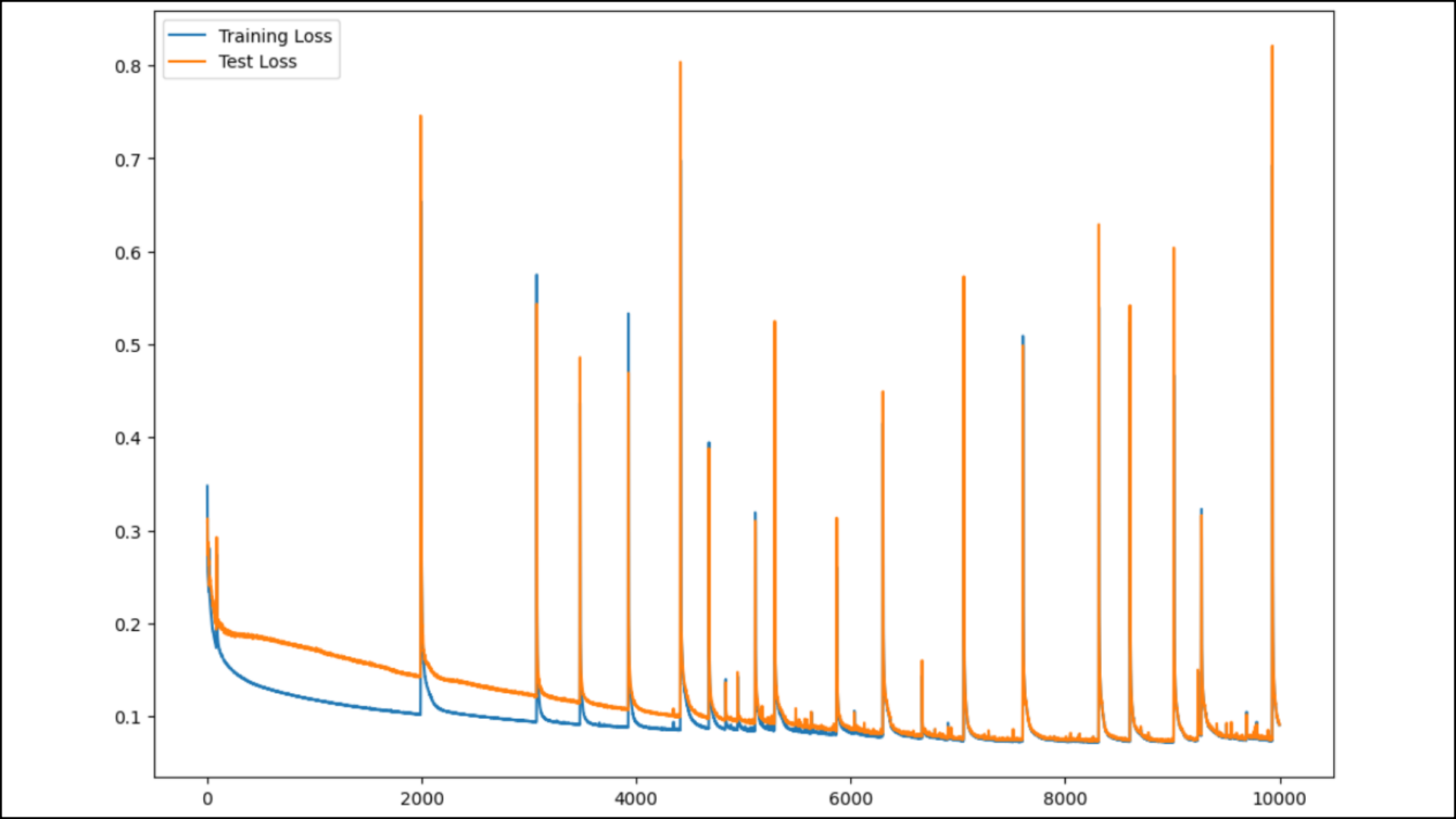

Original set
Training set
Test set
Training set
Validation set
Test set
Training, tuning, and evaluation
Machine learning algorithm
Predictive Model
Final performance estimate

When to stop training

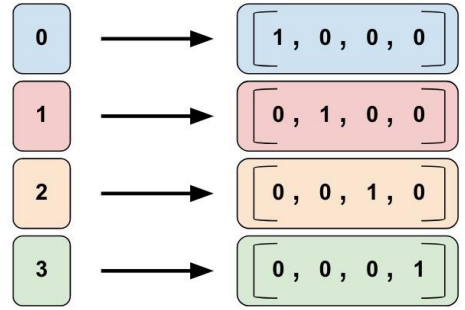


When to stop training



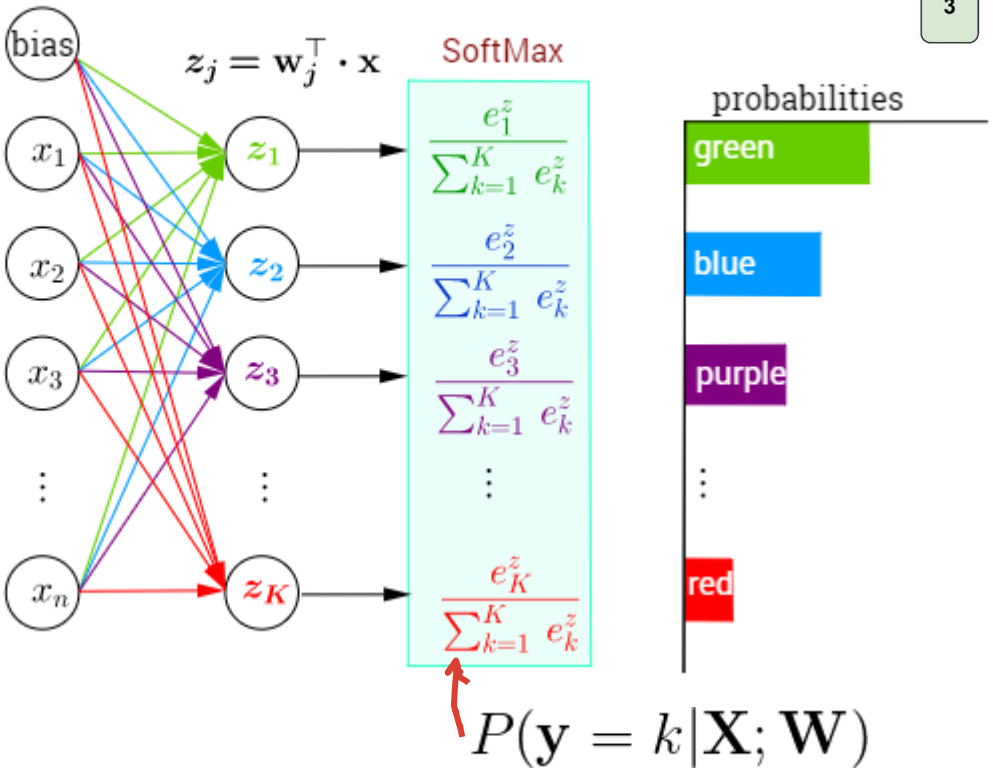

Training Loss
Test Loss
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2000
4000
6000
8000
10000

Classification loss



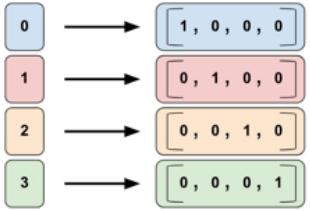
Multi-Class Classification with NN and SoftMax Function

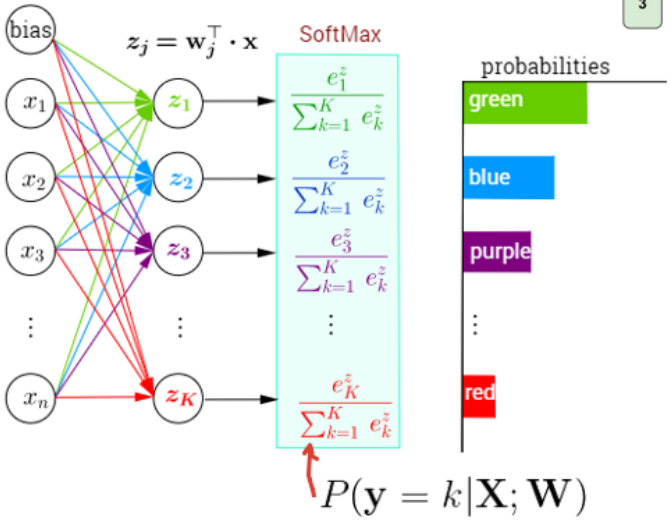
$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

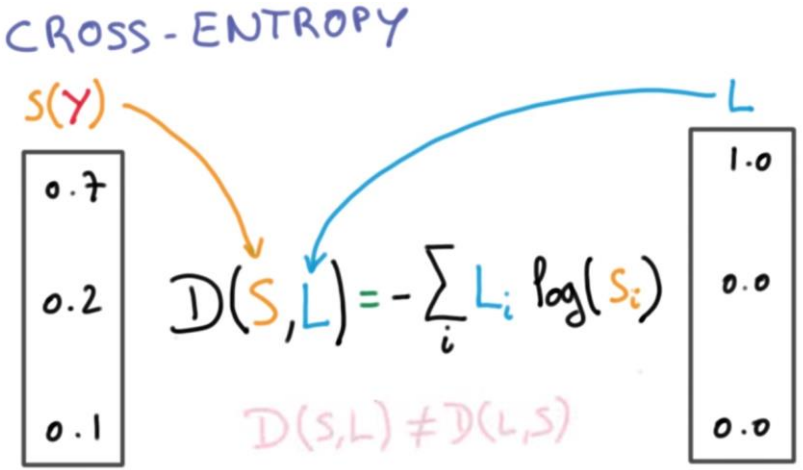


Classification Loss

$$
\begin{aligned}
D_{\mathrm{KL}}(p|q) &= \sum_i p_i \log \frac{p_i}{q_i} \\
&= \sum_i (-p_i \log q_i + p_i \log p_i) \\
&= -\sum_i p_i \log q_i + \sum_i p_i \log p_i \\
&= -\sum_i p_i \log q_i - \sum_i p_i \log \frac{1}{p_i} \\
&= -\sum_i p_i \log q_i - H(p) \\
&= \sum_i p_i \log \frac{1}{q_i} - H(p)
\end{aligned}
$$







Classification loss

A

computed			targets			correct?
0.3	0.3	0.4	0	0	1	yes
0.3	0.4	0.3	0	1	0	yes
0.1	0.2	0.7	1	0	0	no

B

computed			targets			correct?
0.1	0.2	0.7	0	0	1	yes
0.1	0.7	0.2	0	1	0	yes
0.3	0.4	0.3	1	0	0	no

Which classifier is better ?

Classification loss

A

computed			targets			correct?
0.3	0.3	0.4	0	0	1	yes
0.3	0.4	0.3	0	1	0	yes
0.1	0.2	0.7	1	0	0	no

Which classifier is better ?

B

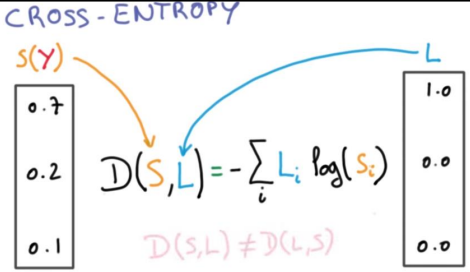
computed			targets			correct?
0.1	0.2	0.7	0	0	1	yes
0.1	0.7	0.2	0	1	0	yes
0.3	0.4	0.3	1	0	0	no

Classification accuracy is a crude way to measure how well NN has been trained!

Classification loss

A

computed			targets			correct?
0.3	0.3	0.4	0	0	1	yes
0.3	0.4	0.3	0	1	0	yes
0.1	0.2	0.7	1	0	0	no



Cross-entropy error ?

B

computed			targets			correct?
0.1	0.2	0.7	0	0	1	yes
0.1	0.7	0.2	0	1	0	yes
0.3	0.4	0.3	1	0	0	no

Classification accuracy is a crude way to measure how well NN has been trained!

Classification loss

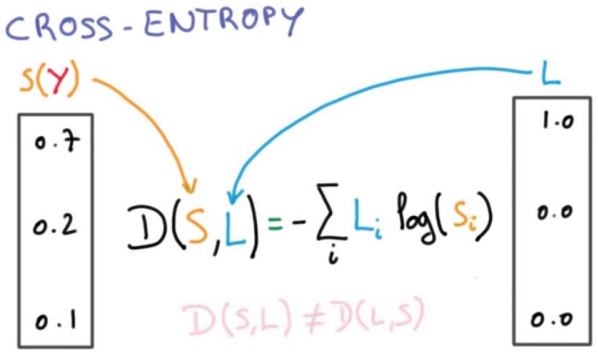
A

MSE ?

computed			targets			correct?
0.3	0.3	0.4	0	0	1	yes
0.3	0.4	0.3	0	1	0	yes
0.1	0.2	0.7	1	0	0	no

B

computed			targets			correct?
0.1	0.2	0.7	0	0	1	yes
0.1	0.7	0.2	0	1	0	yes
0.3	0.4	0.3	1	0	0	no



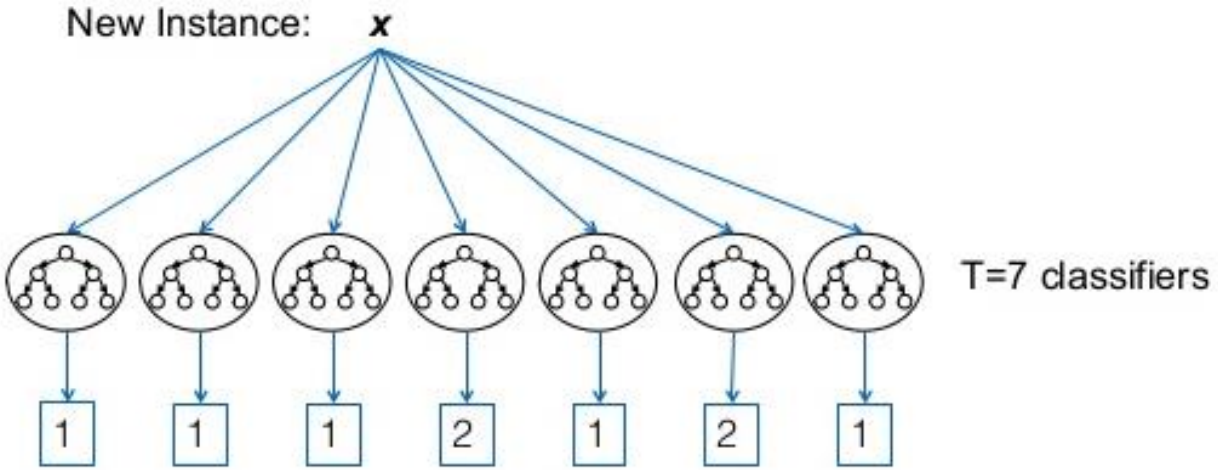
ln() function in cross-entropy takes into account the closeness of a prediction and is a more granular way to compute error.

SPEEDING UP/OPTIMIZING NEURAL NETWORKS

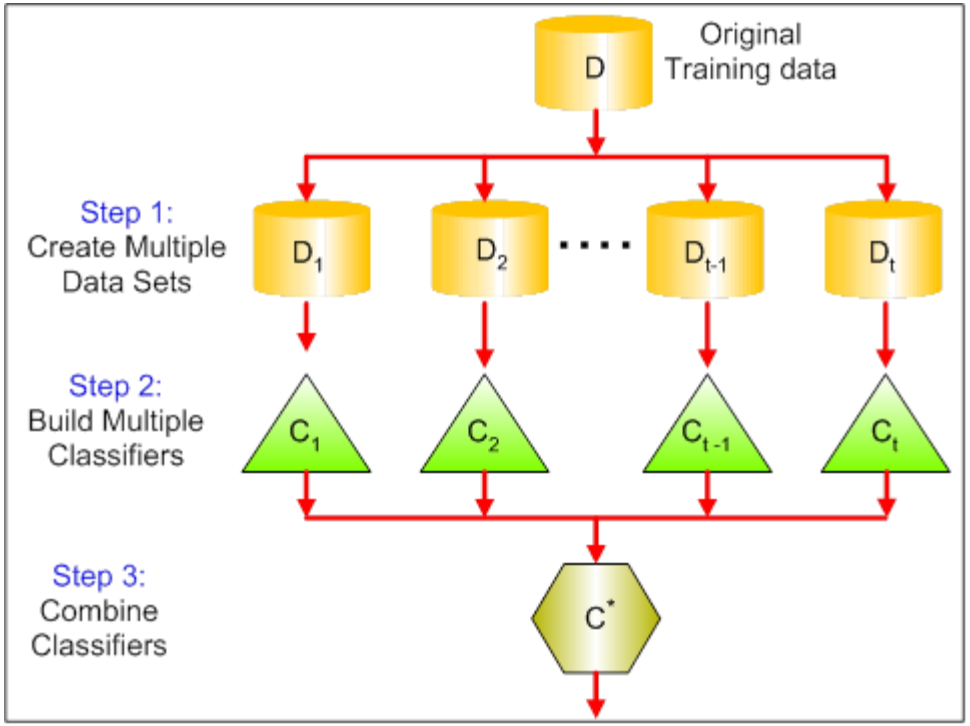
- Batch Normalization
- Increase learning rate.
- Add/Remove Dropout.
- Shuffle training examples more thoroughly
- Reduce the L2 weight regularization.
- Accelerate the learning rate decay.

Ensemble Learning

- An ensemble is a combination of classifiers that output a final classification.

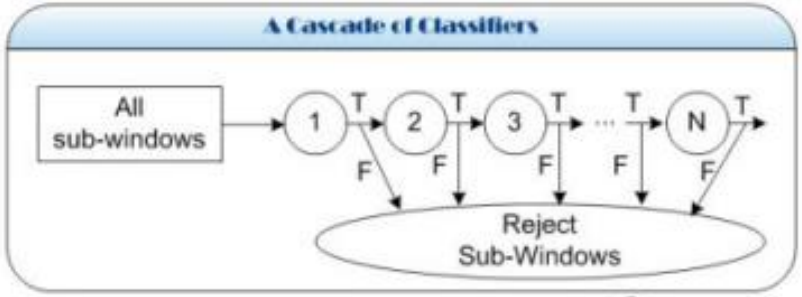


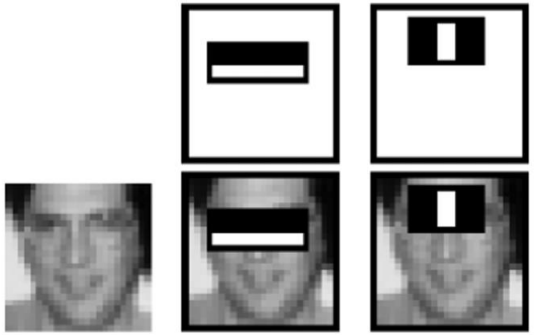
Bootstrap Aggregating (Bagging)

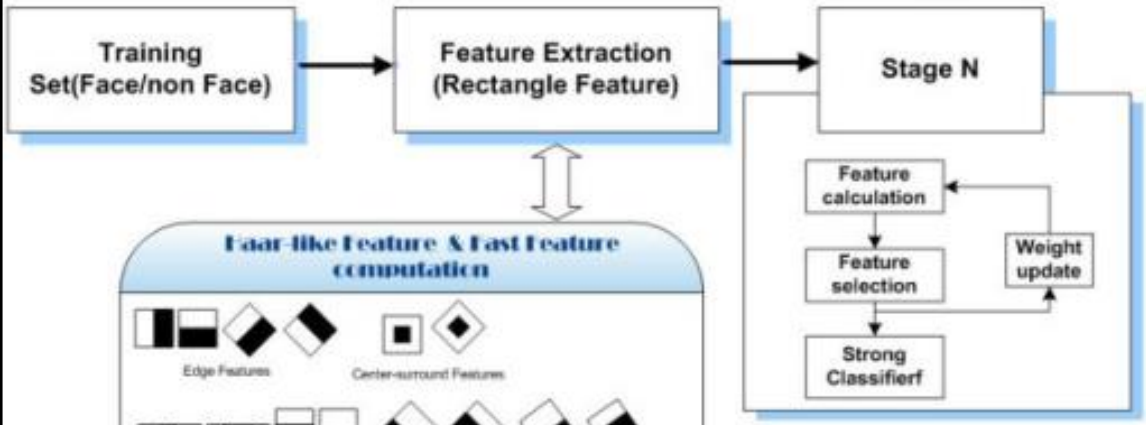


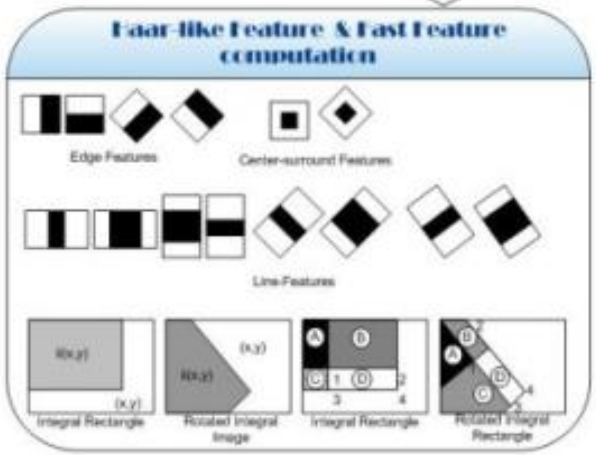
"Bagging Predictors" [Leo Breiman, 1994]
http://statistics.berkeley.edu/sites/default/files/tech-reports/421.pdf

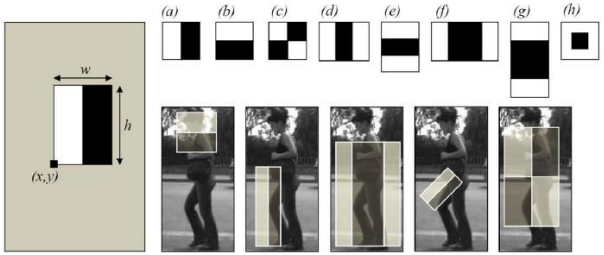
Viola-Jones Face Detection

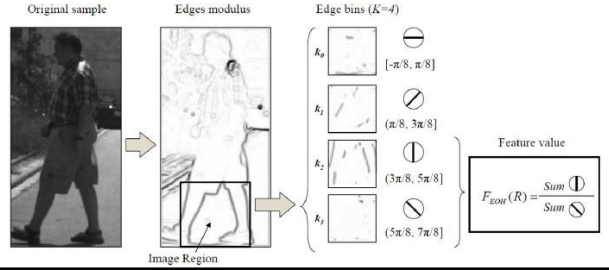






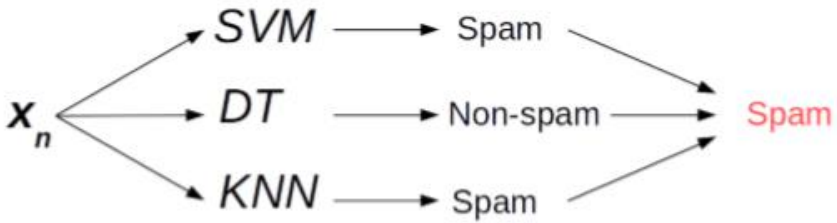




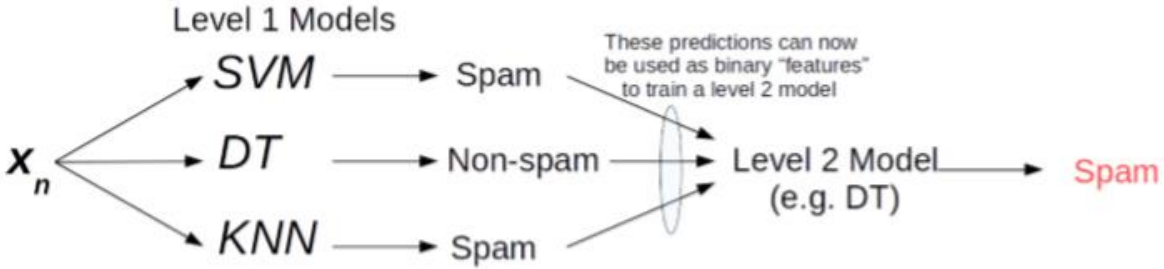


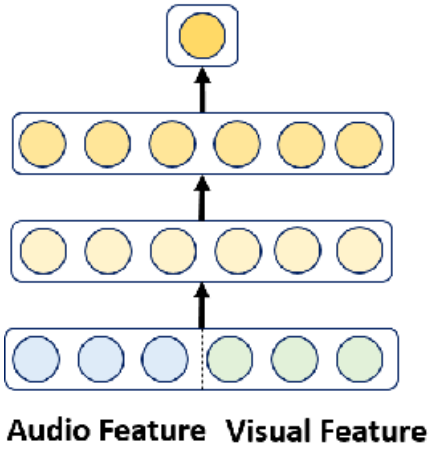
Train different models on same data

- Voting or Averaging of predictions of multiple pre-trained models

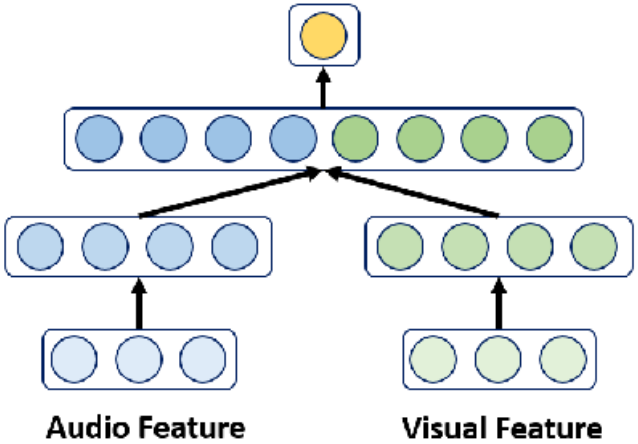


- "Stacking": Use predictions of multiple models as "features" to train a new model and use the new model to make predictions on test data

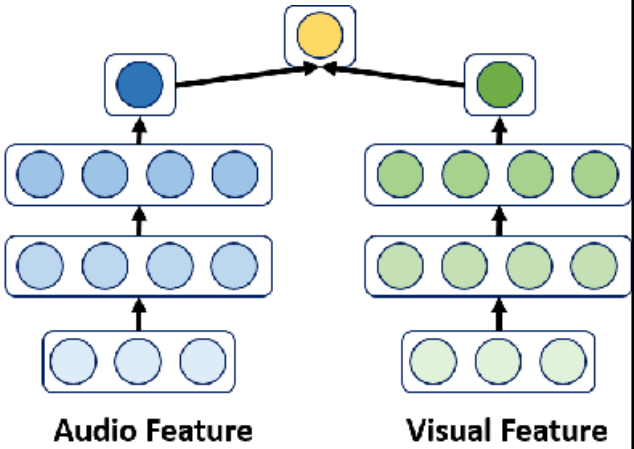




(a) Early Fusion



(b) Model-level Fusion



(c) Late Fusion

Eyes detector

Mouth detector

Ears detector

Skin detector

Meta Classifier

High accuracy!

Low individual accuracy
Computationally efficient

Unsupervised Learning → Density Estimation

Aka "learning without a teacher"

Feature Space $\mathcal{X}$

Word distribution
(Probability of a word)

Task: Given $X \in \mathcal{X}$, learn $f(X)$.

Parametric Density Estimation (GMM)

- Initialize the means μ_k, covariances Σ_k and mixing coefficients π_k
- Iterate until convergence:
  - E-step: Evaluate the responsibilities given current parameters

$$\gamma_k^{(n)} = p(z^{(n)}|\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}^{(n)}|\mu_j, \Sigma_j)}$$

  - M-step: Re-estimate the parameters given current responsibilities

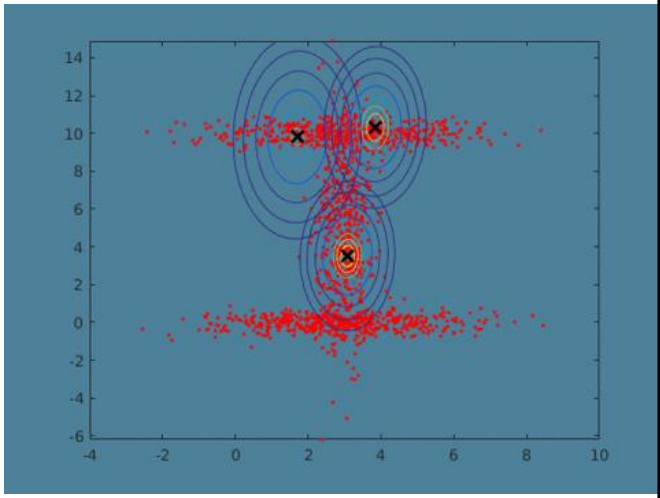
$$\mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} \mathbf{x}^{(n)}$$

$$\Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma_k^{(n)} (\mathbf{x}^{(n)} - \mu_k)(\mathbf{x}^{(n)} - \mu_k)^T$$

$$\pi_k = \frac{N_k}{N} \quad \text{with} \quad N_k = \sum_{n=1}^{N} \gamma_k^{(n)}$$

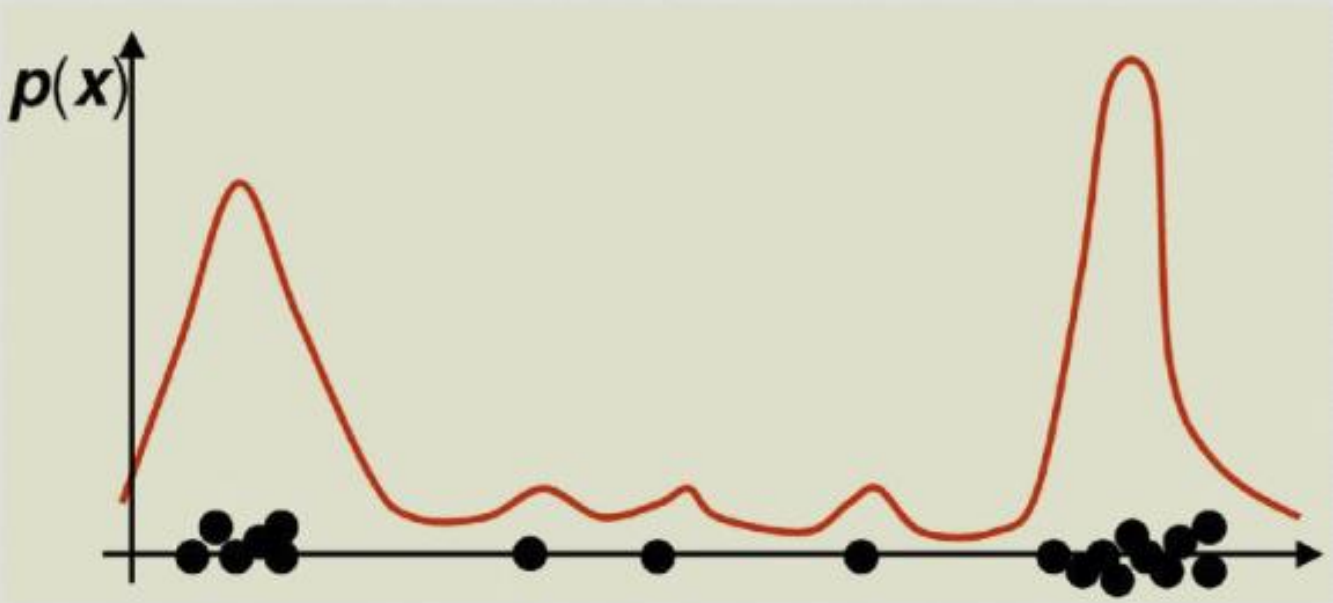
  - Evaluate log likelihood and check for convergence

$$\ln p(\mathbf{X}|\pi, \mu, \Sigma) = \sum_{n=1}^{N} \ln \left(\sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}^{(n)}|\mu_k, \Sigma_k) \right)$$



Basic idea

- In a given point we will estimate p(x) from the density of the data points falling into a small region R around x
  - More samples → larger probability



The histogram

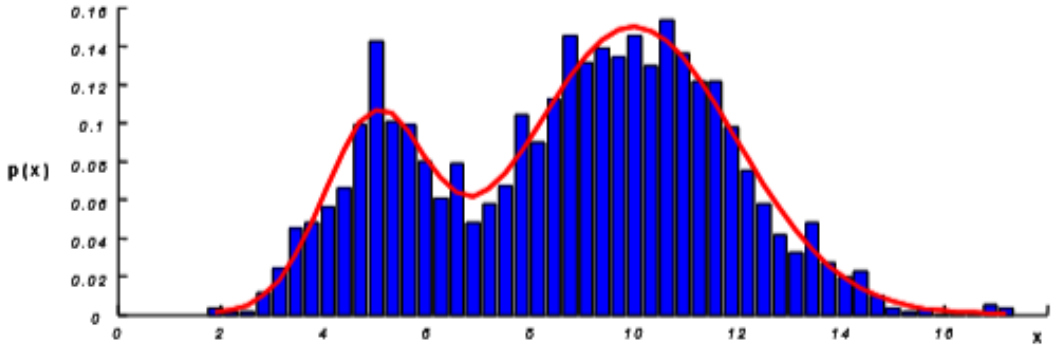
The simplest form of non-parametric DE is the histogram

$$p(x) \cong \frac{k}{NV}$$

- Divide the sample space into a number of bins and approximate the density at the center of each bin by the fraction of points in the training data that fall into the corresponding bin

$$p_H(x) = \frac{1}{N}\frac{[\#\ of\ x^{(k}\ in\ same\ bin\ as\ x]}{[width\ of\ bin]}$$

- The histogram requires two "parameters" to be defined: bin width and starting position of the first bin

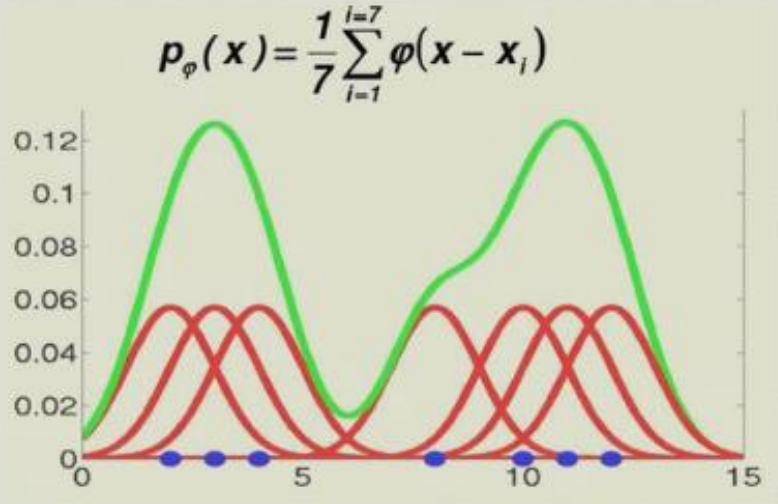


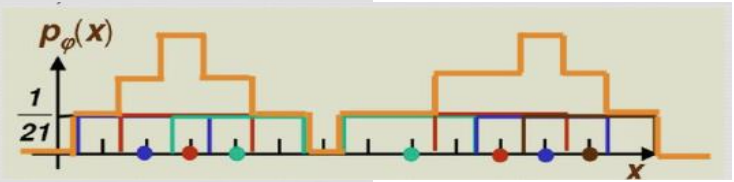
Issues with Histograms

- Not smooth
- Depend on end points of bins
- Depend on width of bins

Gaussian window function - example

- Let's return to our previous example
- D={2,3,4,8,10,11,12}, n = 7, h = 1, d = 1

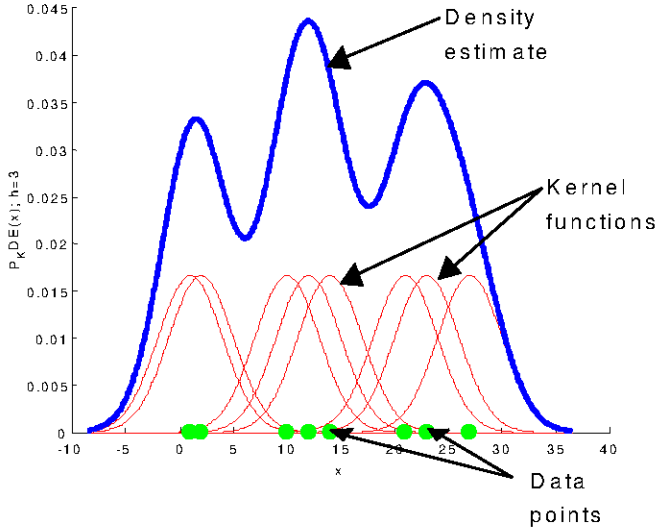




- The estimate for p(x) will be sum of 7 Gaussians, each centerd on one of the sample points, each scaled by 1/7

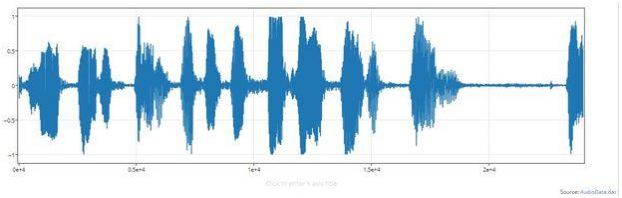
Interpretation

- Just as the Parzen window estimate can be seen as a sum of boxes centered at the data, the smooth kernel estimate is a sum of "bumps"
- The kernel function determines the shape of the bumps
- The parameter h, also called the smoothing parameter or bandwidth, determines their width



Recurrent Neural Networks

Sequential data

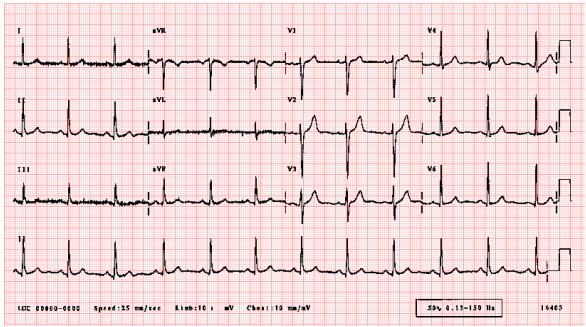


Audio

Video

O Nachiketa, after pondering well the pleasures that are or seem to be delightful, you have renounced them all. You have not taken the road abounding in wealth, where many men sink. *(Kathopanishad, II:3)*

Text

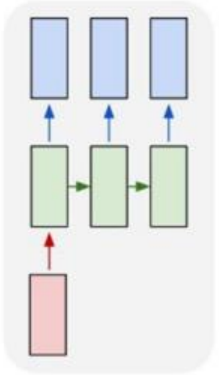


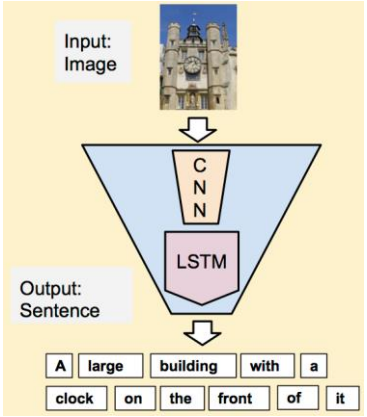
Biological

many more ……

Tasks involving sequences

one to many

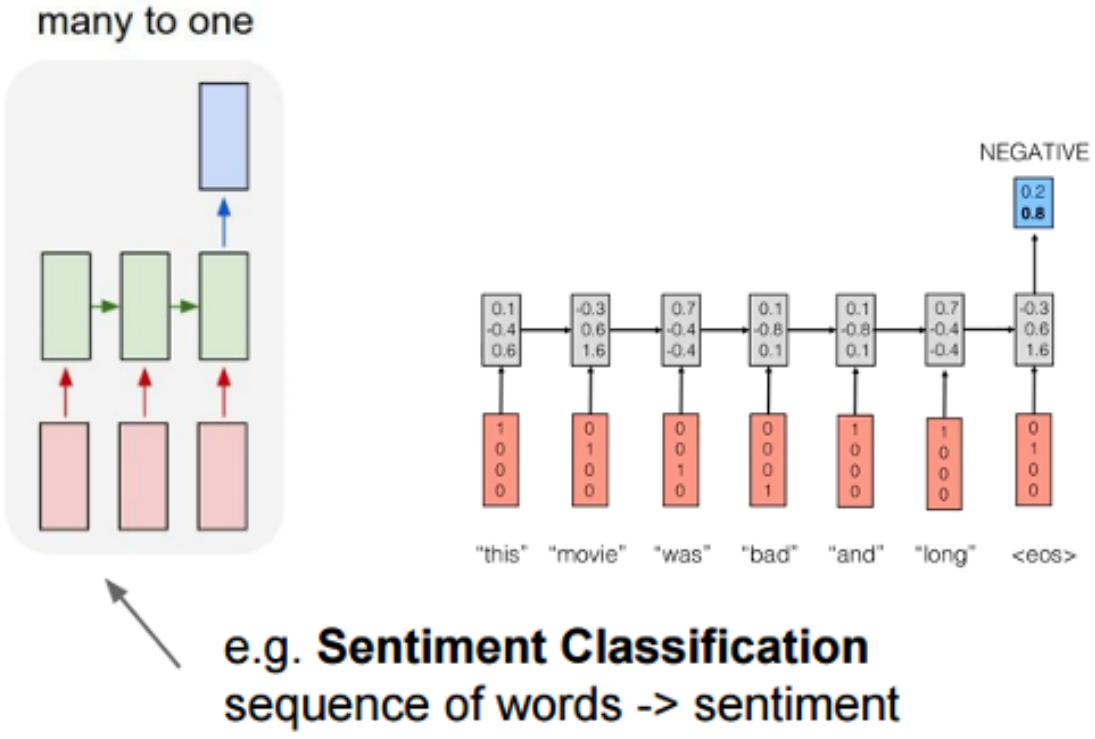




e.g. **Image Captioning**
image -> sequence of words

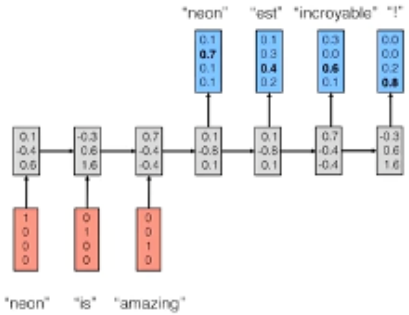
http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

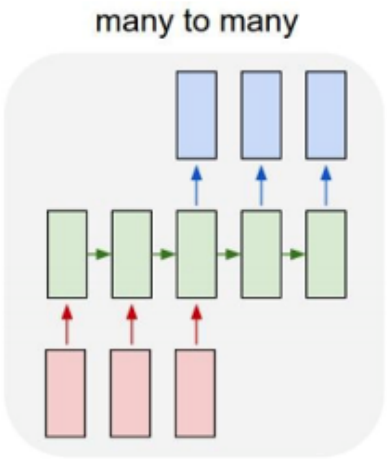
Tasks involving sequences



http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf

Tasks involving sequences

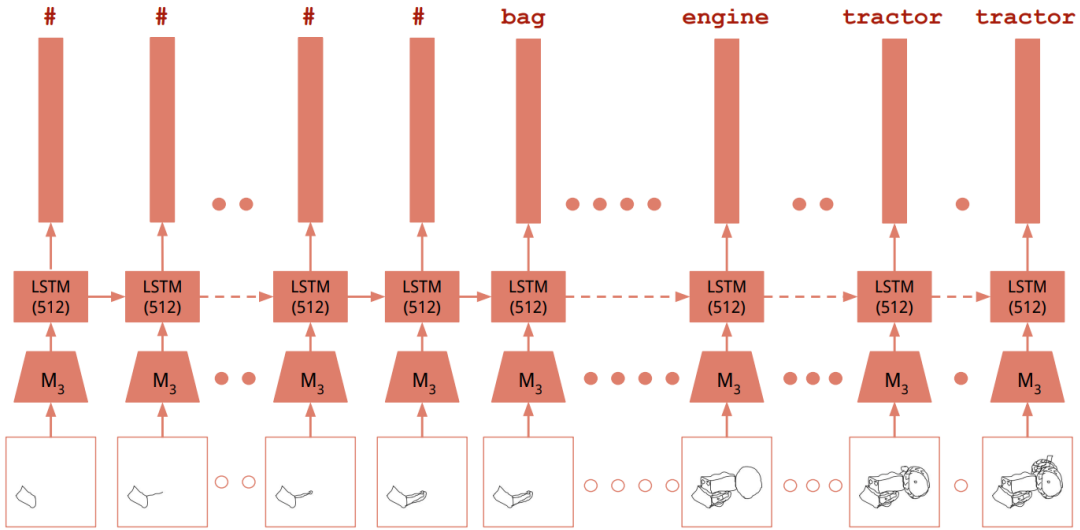


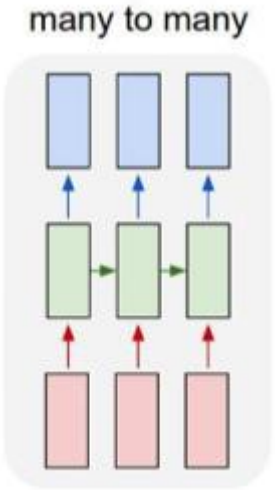


e.g. **Machine Translation**
seq of words -> seq of words

http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf, https://www.youtube.com/watch?v=Ukgii7Yd_cU

Tasks involving sequences



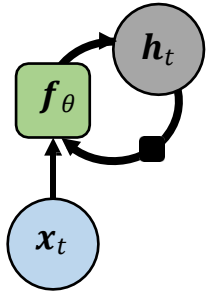


Game of Sketches: Deep Recurrent Models of Pictionary-style Word Guessing
Ravi Kiran Sarvadevabhatla, Shiv Surya, Trisha Mittal, R. Venkatesh Babu
***AAAI** 2018*

Recurrent Neural Network

We can process a sequence of vectors $\boldsymbol{x}_t$ by applying a recurrence formula at every time step:

$$\boldsymbol{h}_t = f_\theta(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$



Recurrent Neural Network

We can process a sequence of vectors $\boldsymbol{x}_t$ by applying a recurrence formula at every time step:

Task-specific abstraction for portion of sequence seen up to this point $(x_1, x_2 \ldots x_{t-1})$

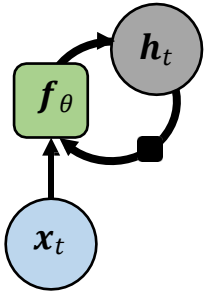
new state

prior state

$$\boldsymbol{h}_t = f_\theta(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$

some function with parameters θ

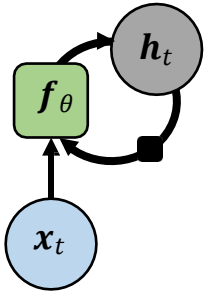
input vector at the current step



Recurrent Neural Network

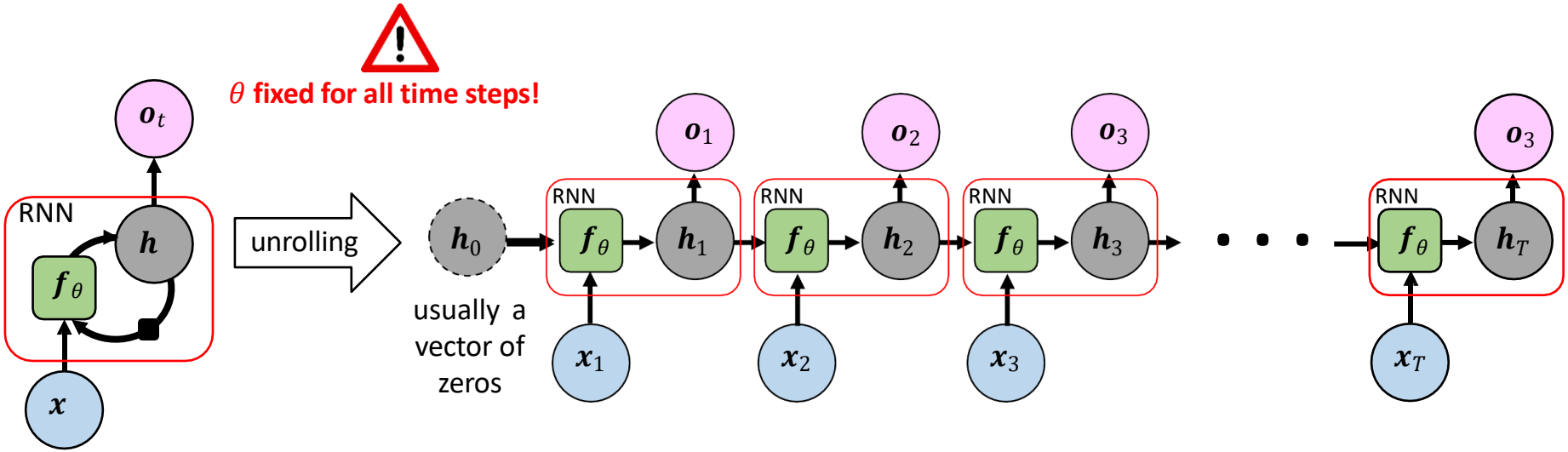
We can process a sequence of vectors $\boldsymbol{x}_t$ by applying a recurrence formula at every time step:

$$\boldsymbol{h}_t = f_\theta(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$



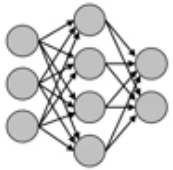
the same function f_θ and the same set of parameters θ are used at every time step.

RNN unrolling = regular NN



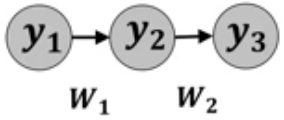
Training RNNs

Recap: Training CNN



$$y_i = g\left(\sum_j W_{ij}x_j + b_i\right)$$

$$\frac{\partial C}{\partial W} = \frac{\partial C}{\partial g} \cdot \frac{\partial g}{\partial a} \cdot \frac{\partial a}{\partial W}$$



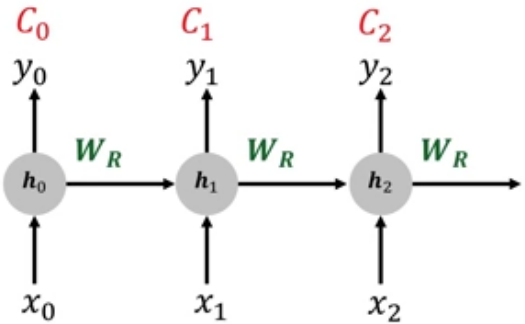
$$\boldsymbol{y}_k = g(W\boldsymbol{y}_{k-1} + \boldsymbol{b})$$

Single, final cost C
Non-shared weights (W_1, W_2)

https://www.youtube.com/watch?v=Ukgii7Yd_cU

Training RNNs - Backpropagation through time

$$\boldsymbol{h}^{(t)} = g_h(W_{\mathrm{I}}\boldsymbol{x}^{(t)} + W_{\mathrm{R}}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$
$$\boldsymbol{y}^{(t)} = g_y(W_y\boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$



$$\frac{\partial C}{\partial W_R} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial W_R}$$

$$\frac{\partial C_t}{\partial W_R} = \sum_{k=1}^{t} \frac{\partial C_t}{\partial y_t}\frac{\partial y_t}{\partial h_t}\frac{\partial h_t}{\partial h_k}\frac{\partial h_k}{\partial W_R}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} W_R^T \mathrm{diag}(g_h'(W_I x^{(i)} + W_R h^{(i-1)} + b_h))$$

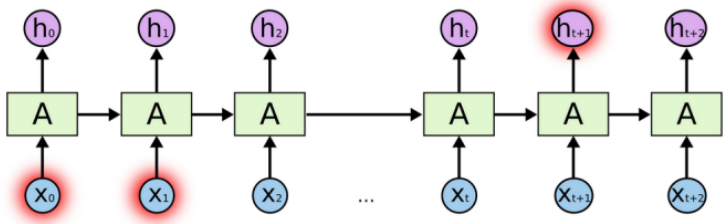
(Potentially) multiple, intermediate costs C_0,C_1,C_2
Shared weights $\mathbf{W_R}$

https://www.youtube.com/watch?v=Ukgii7Yd_cU

Fundamental issues with RNNs

Problem of Long-term dependency

- There are cases where we need more context
- To predict the last word in the sentence "I grew up in France⋯.I speak fluent *French*"
- Using only recent information suggests that the last word is the name of a language. But more distant past indicates that it is French
- It is possible that the gap between the relevant information and where it is needed is very large

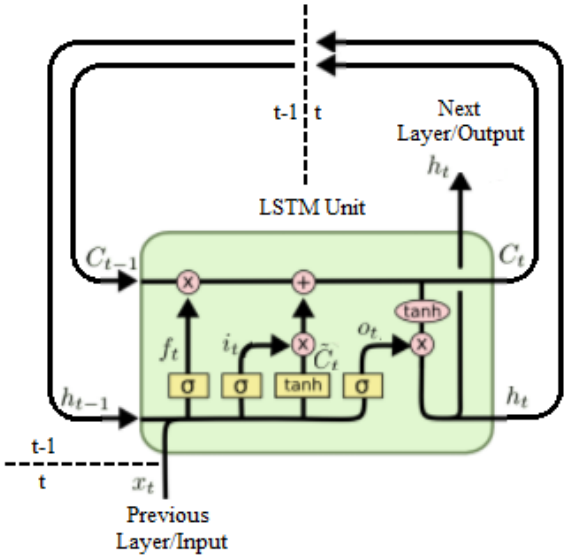


http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

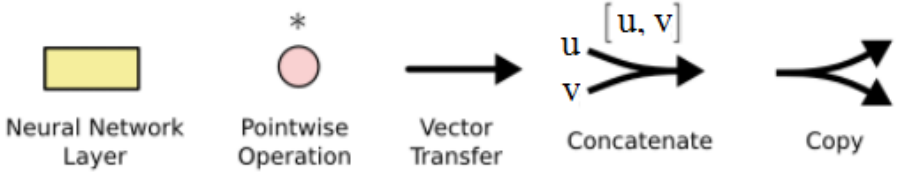

I'LL EMAIL THIS TO MYSELF SO I CAN REMEMBER IT
HEY LOOK, A NEW EMAIL!


Not only is my short-term memory horrible, but so is my short-term memory.
Fb.com/MinionQuote
DespicableMeMinions.org

Understanding LSTM Networks



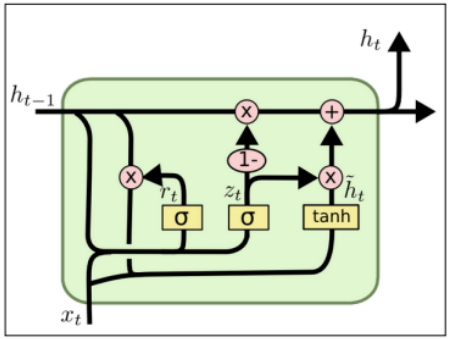
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$
$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$



GRU: a LSTM variant

Gated Recurrent Unit (GRU)

- A dramatic variant of LSTM
  - It combines the forget and input gates into a single update gate
  - It also merges the cell state and hidden state, and makes some other changes
  - The resulting model is simpler than LSTM models
  - Has become increasingly popular



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

http://www.cedar.buffalo.edu/~srihari/CSE676/LSTM.pdf

Applications of RNNs in CV

Image/Video Captioning

Visual Question Answering

….

Image Captioning

man in black shirt is playing guitar.

construction worker in orange safety vest is working on road.

two young girls are playing with lego toy.

(Slides by Marc Bolaños): Karpathy, Andrej, and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." CVPR 2015

Language generation

Training this on a lot of sentences would give us a language model. A way to predict

P(next word | previous words)

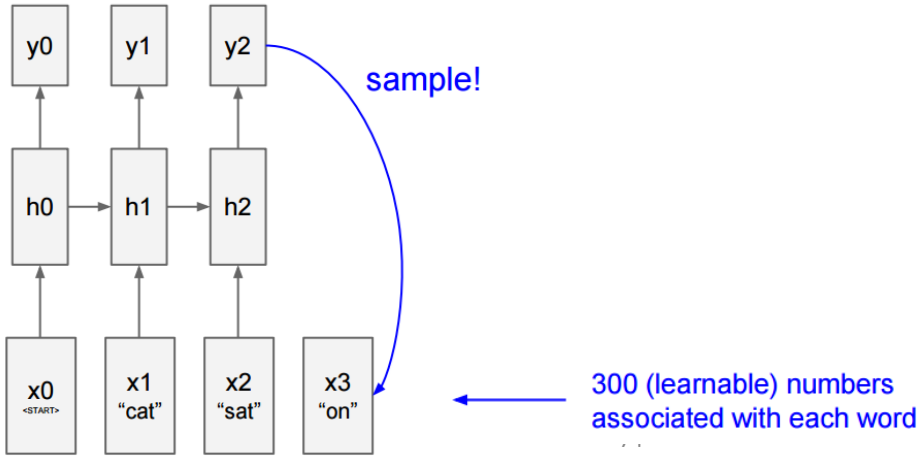


Image Captioning

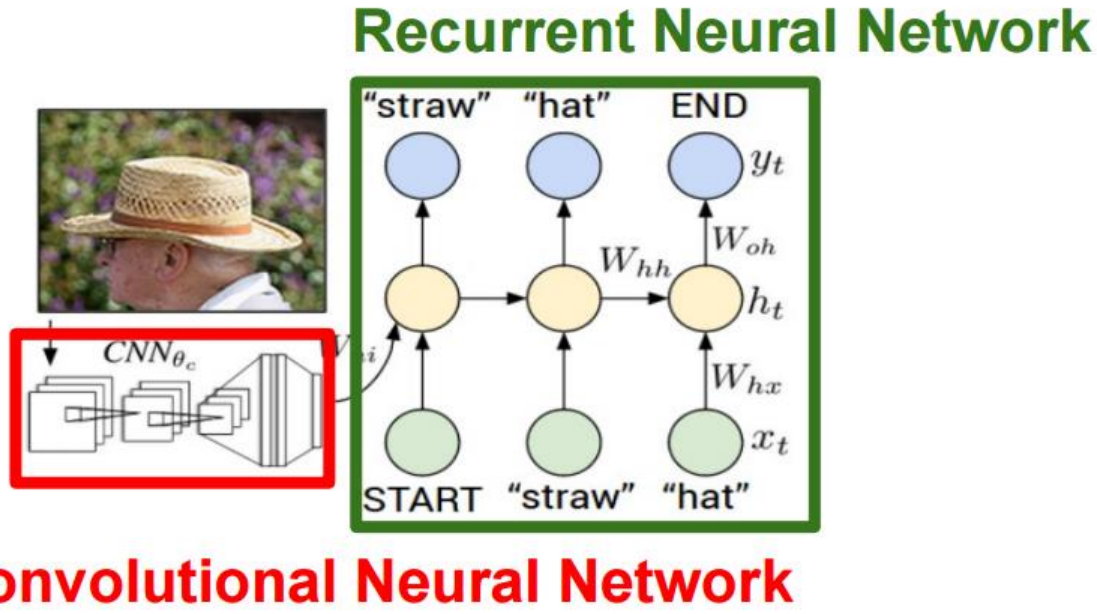


Image Captioning

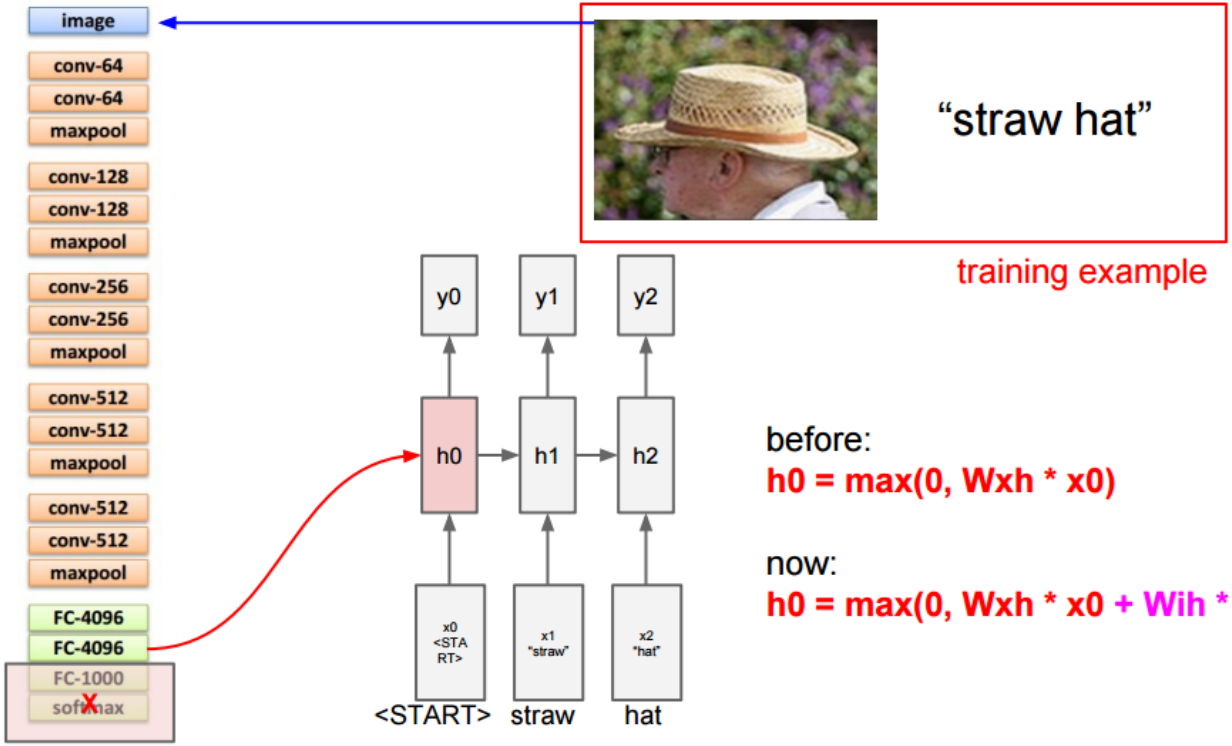


Image Captioning

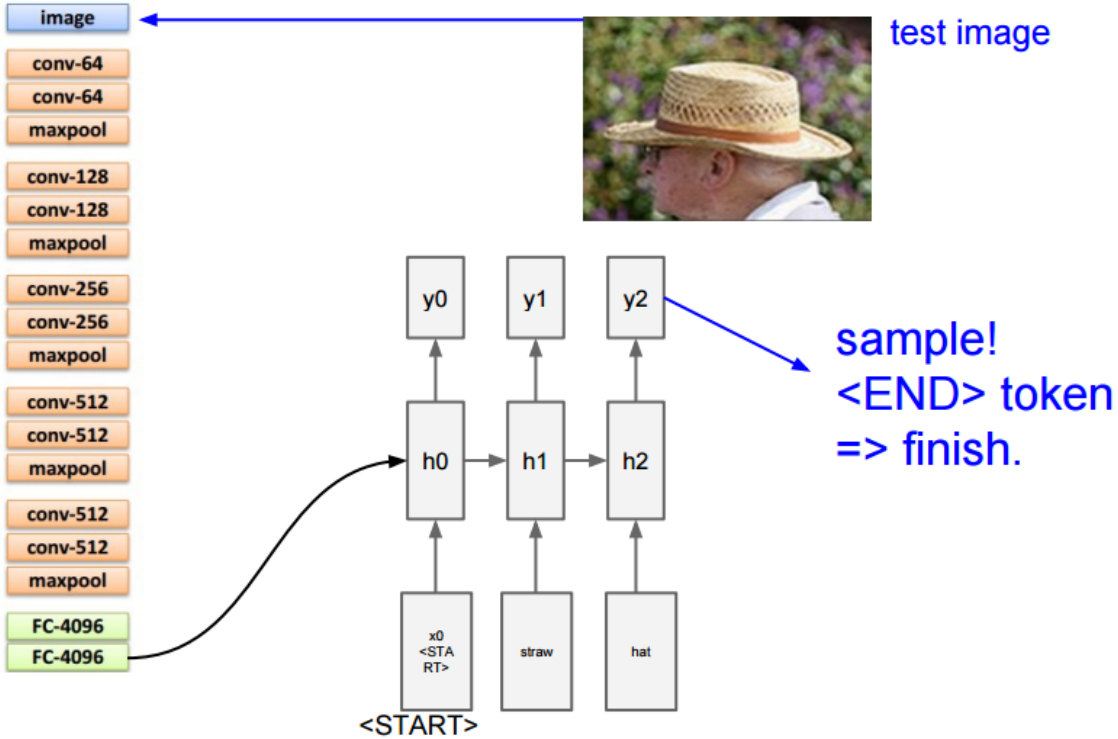
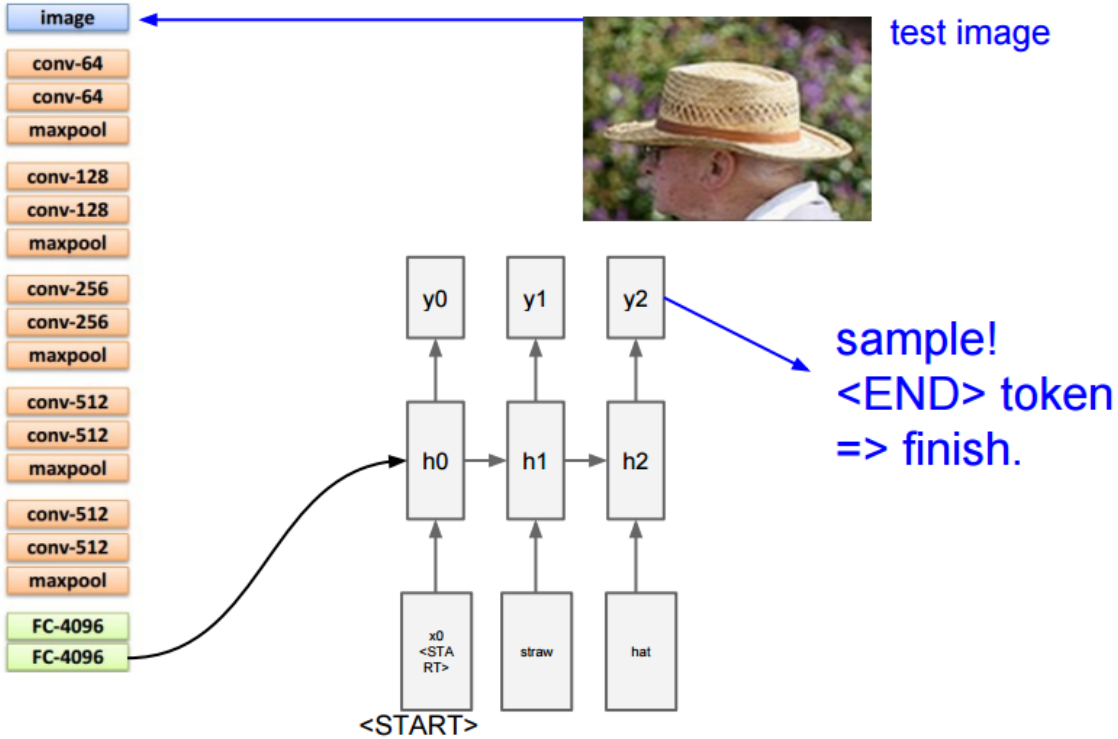
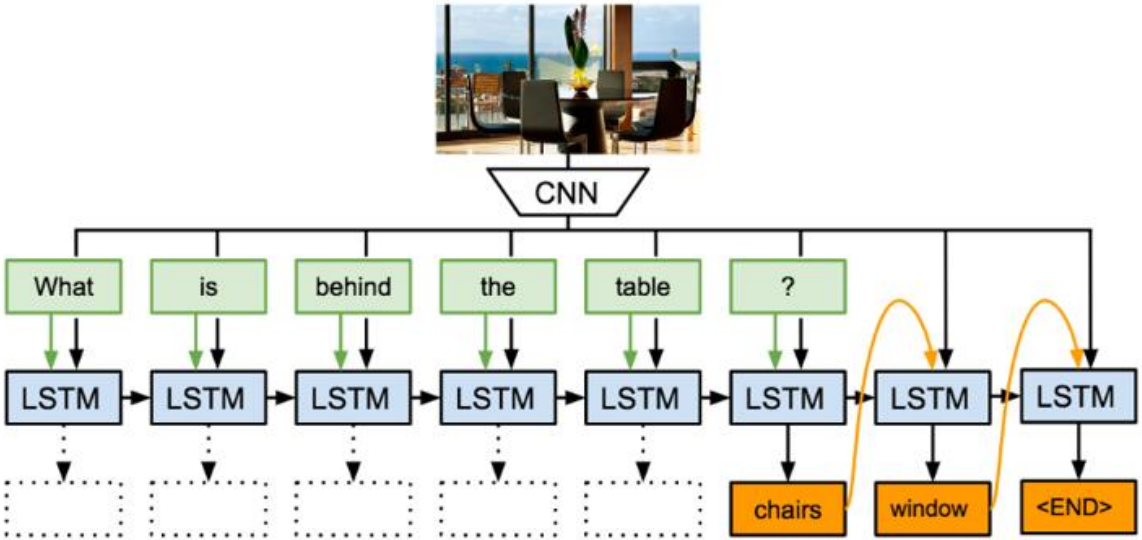


Image Captioning



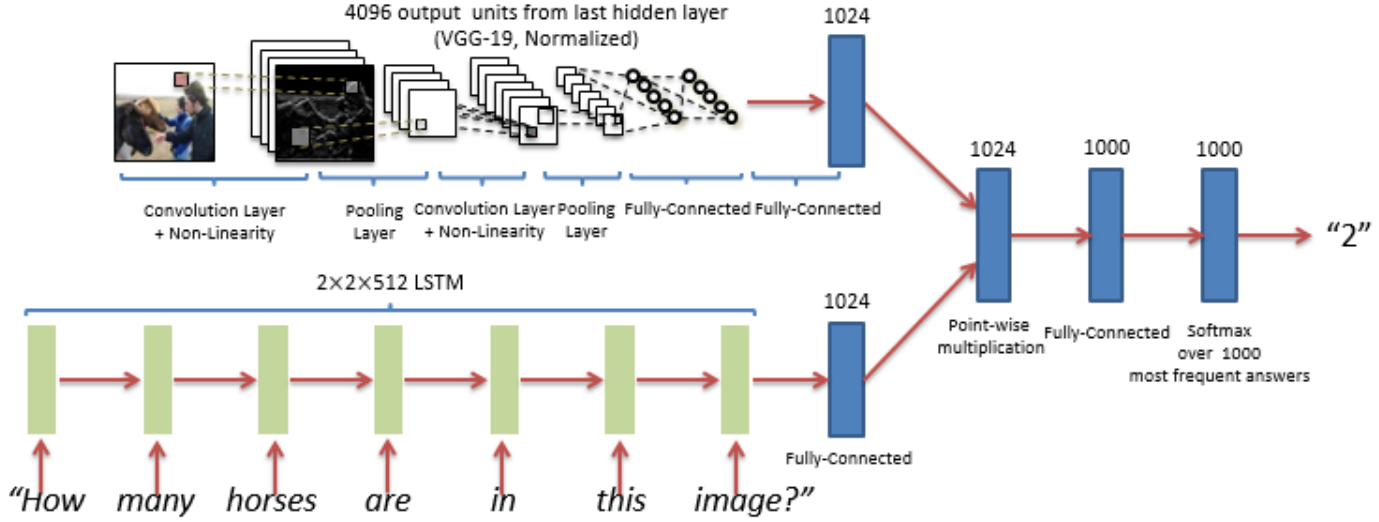
Visual Question Answering

[Malinowski 2015]



Visual Question Answering

[Lu 2015]

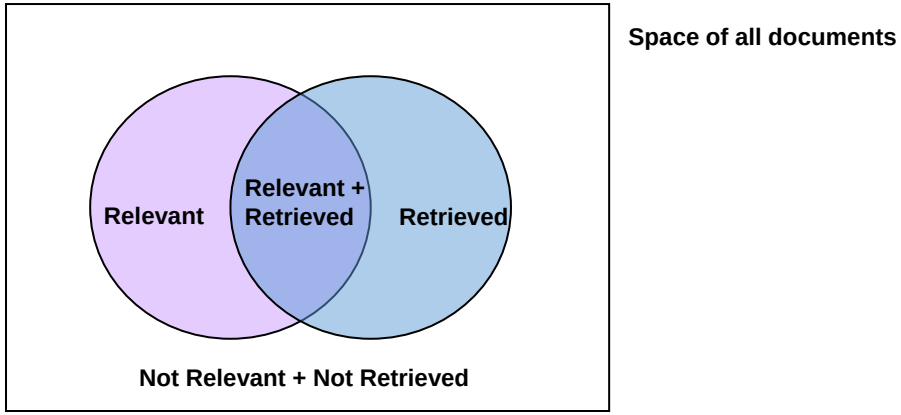


Precision and Recall in Retrieval

- Precision
  - The ability to retrieve top-ranked items that are mostly relevant.
  - Precision P = tp/(tp + fp)
- Recall
  - The ability of the search to find ***all*** of the relevant items in the database.
  - Recall R = tp/(tp + fn)

	Relevant	Nonrelevant
Retrieved	tp	fp
Not Retrieved	fn	tn

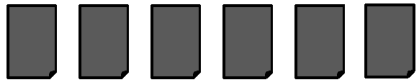
Precision and Recall : Text Retrieval



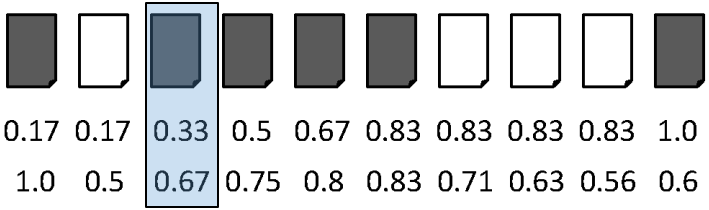
$$recall = \frac{Number\ of\ \ relevant\ documents\ retrieved}{Total\ number\ of\ \ relevant\ documents}$$

$$precision = \frac{Number\ \ \ of\ \ relevant\ documents\ \ retrieved}{Total\ number\ of\ \ documents\ \ \ retrieved}$$

Precision/Recall : Example



= the relevant documents



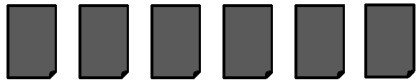
Recall = 2/6 = 0.33

Precision = 2/3 = 0.67

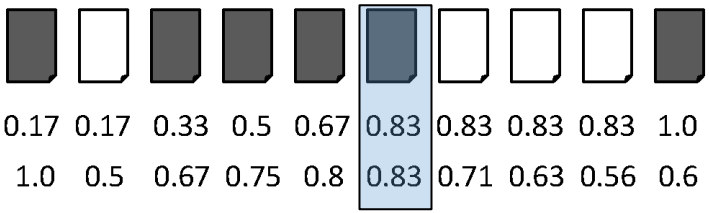
$$\textbf{Precision} = \frac{\textbf{Relevant Retrieved}}{\textbf{Retrieved}}$$

$$\textbf{Recall} = \frac{\textbf{Relevant Retrieved}}{\textbf{Relevant}}$$

Precision/Recall : Example



= the relevant documents



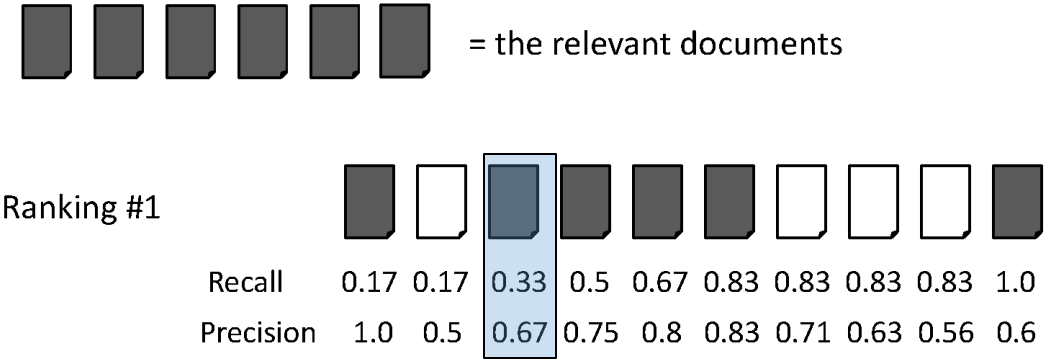
Recall = 5/6 = 0.83

Precision = 5/6 = 0.83

$$\textbf{Precision} = \frac{\textbf{Relevant Retrieved}}{\textbf{Retrieved}}$$

$$\textbf{Recall} = \frac{\textbf{Relevant Retrieved}}{\textbf{Relevant}}$$

F Measure (F1/Harmonic Mean) : example



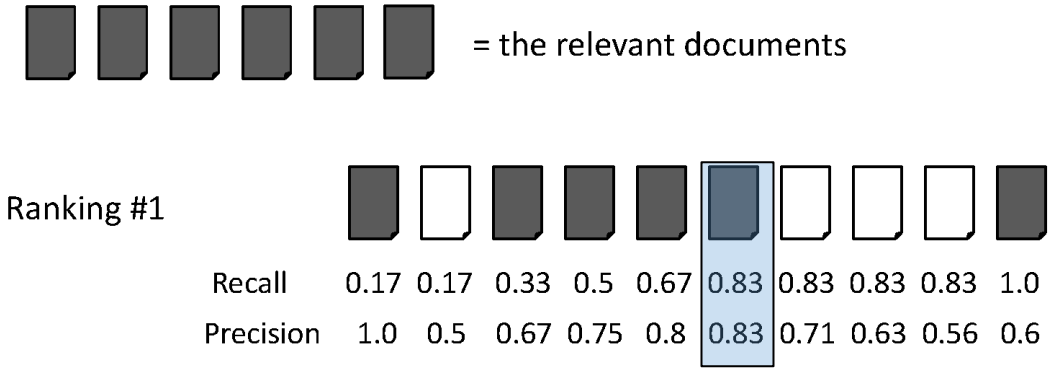
Recall = 2/6 = 0.33

Precision = 2/3 = 0.67

F = 2*Recall*Precision/(Recall + Precision)

= 2*0.33*0.67/(0.33 + 0.67) = 0.44

F Measure (F1/Harmonic Mean) : example



Recall = 5/6 = 0.83

Precision = 5/6 = 0.83

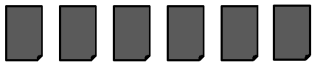
F = 2*Recall*Precision/(Recall + Precision)

= 2*0.83*0.83/(0.83 + 0.83) = 0.83

Mean Average Precision (MAP)

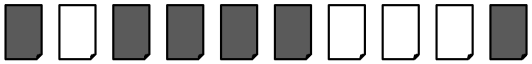
- **Average Precision**: Average of the precision values at the points at which each relevant document is retrieved.
  - Ex1: (1 + 1 + 0.75 + 0.667 + 0.38 + 0)/6 = 0.633
  - Ex2: (1 + 0.667 + 0.6 + 0.5 + 0.556 + 0.429)/6 = 0.625
  - Averaging the precision values from the rank positions where a relevant document was retrieved
  - Set precision values to be zero for the not retrieved documents

Average Precision: Example



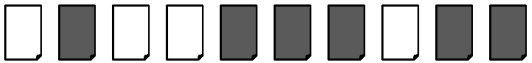
= the relevant documents

Ranking #1



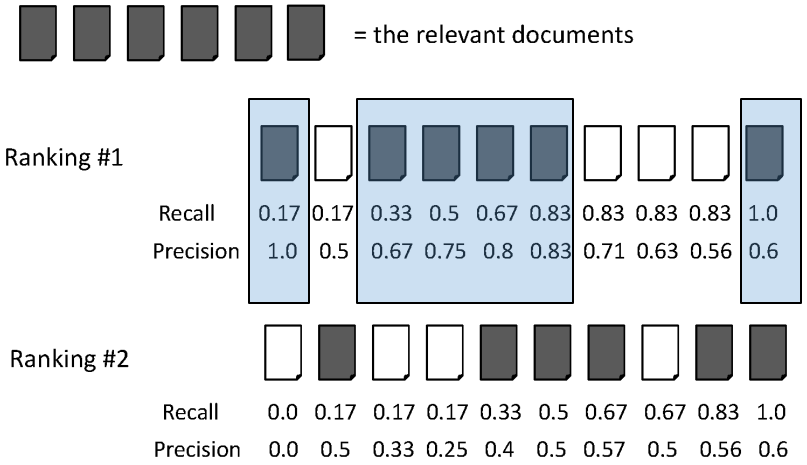
Recall	0.17	0.17	0.33	0.5	0.67	0.83	0.83	0.83	0.83	1.0
Precision	1.0	0.5	0.67	0.75	0.8	0.83	0.71	0.63	0.56	0.6

Ranking #2



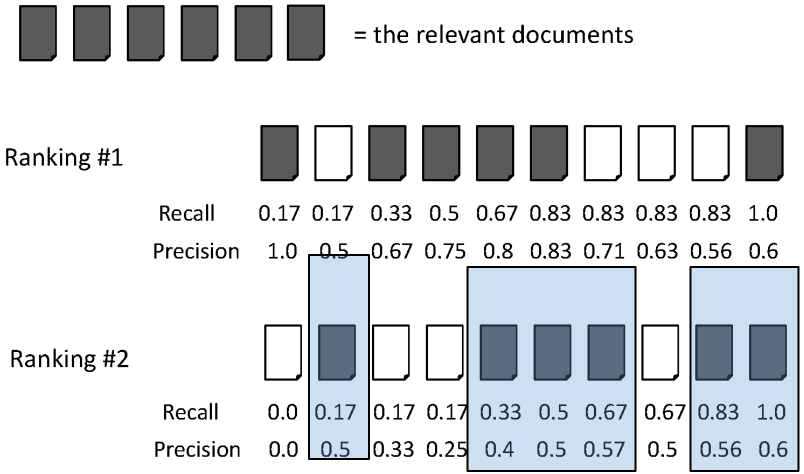
Recall	0.0	0.17	0.17	0.17	0.33	0.5	0.67	0.67	0.83	1.0
Precision	0.0	0.5	0.33	0.25	0.4	0.5	0.57	0.5	0.56	0.6

Average Precision: Example



$$\text{Ranking } \#1\text{: } (1.0 + 0.67 + 0.75 + 0.8 + 0.83 + 0.6)/6 = 0.78$$

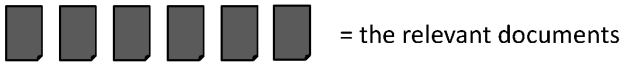
Average Precision: Example



$$\text{Ranking } \#1\text{: } (1.0 + 0.67 + 0.75 + 0.8 + 0.83 + 0.6)/6 = 0.78$$

$$\text{Ranking } \#2\text{: } (0.5 + 0.4 + 0.5 + 0.57 + 0.56 + 0.6)/6 = 0.52$$

Average Precision: Example



Ranking #1

Recall	0.17	0.17	0.33	0.5	0.67	0.83	0.83
Precision	1.0	0.5	0.67	0.75	0.8	0.83	0.71

Ranking #2

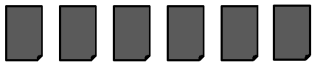
Recall	0.0	0.17	0.17	0.17	0.33	0.5	0.67
Precision	0.0	0.5	0.33	0.25	0.4	0.5	0.57

Miss one relevant document

$$\text{Rank 1} = (1 + 0.67 + 0.75 + 0.8 + 0.83 + 0)/6 = 0.675$$

$$\text{Rank 2} = (0.5 + 0.4 + 0.5 + 0.57 + 0 + 0)/6 = 0.328$$

Average Precision: Example



= the relevant documents

Ranking #1

Recall	0.17	0.17	0.33	0.5	0.67	0.83	0.83
Precision	1.0	0.5	0.67	0.75	0.8	0.83	0.71

Ranking #2

Recall	0.0	0.17	0.17	0.17	0.33	0.5	0.67
Precision	0.0	0.5	0.33	0.25	0.4	0.5	0.57

Miss two relevant documents

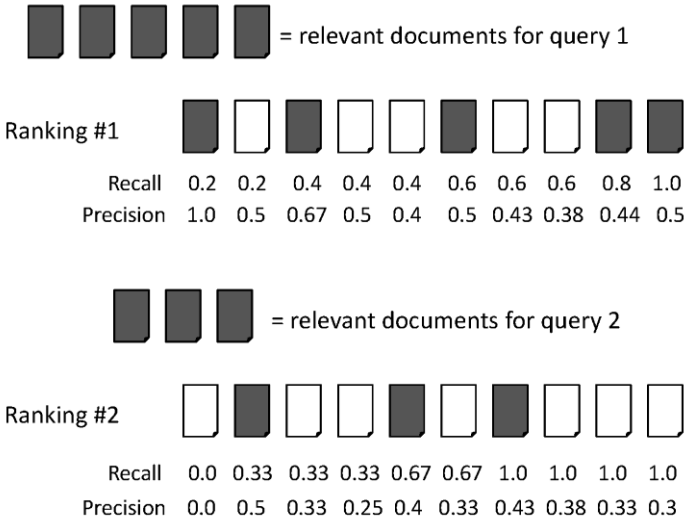
$$\text{Rank 1} = (1 + 0.67 + 0.75 + 0.8 + 0.83 + 0)/6 = 0.675$$

$$\text{Rank 2} = (0.5 + 0.4 + 0.5 + 0.57 + 0 + 0)/6 = 0.328$$

Mean Average Precision (MAP)

- Summarize rankings from multiple queries by averaging average precision
- Most commonly used measure in research papers
- Assumes user is interested in finding **many** relevant documents for each query
- Requires **many** relevance judgments in text collection

Mean Average Precision (MAP)

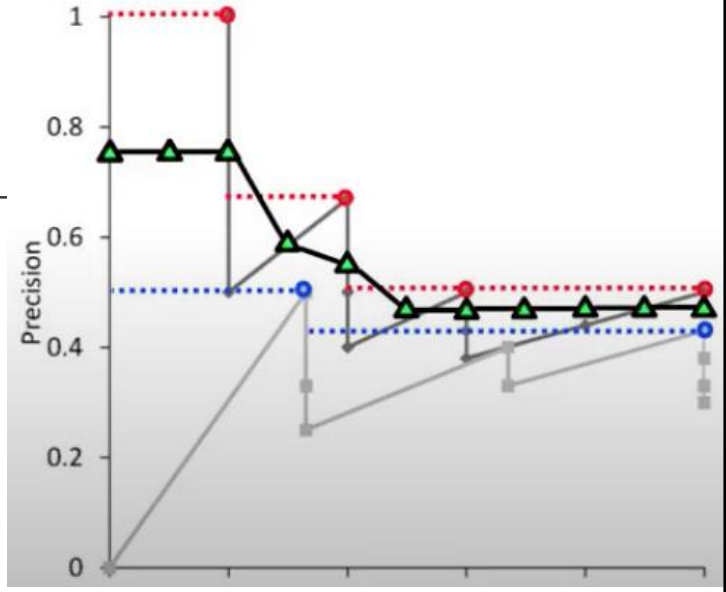


$$average\ precision\ query\ 1 = (1.0 + 0.67 + 0.5 + 0.44 + 0.5)/5 = 0.62$$
$$average\ precision\ query\ 2 = (0.5 + 0.4 + 0.43)/3 = 0.44$$

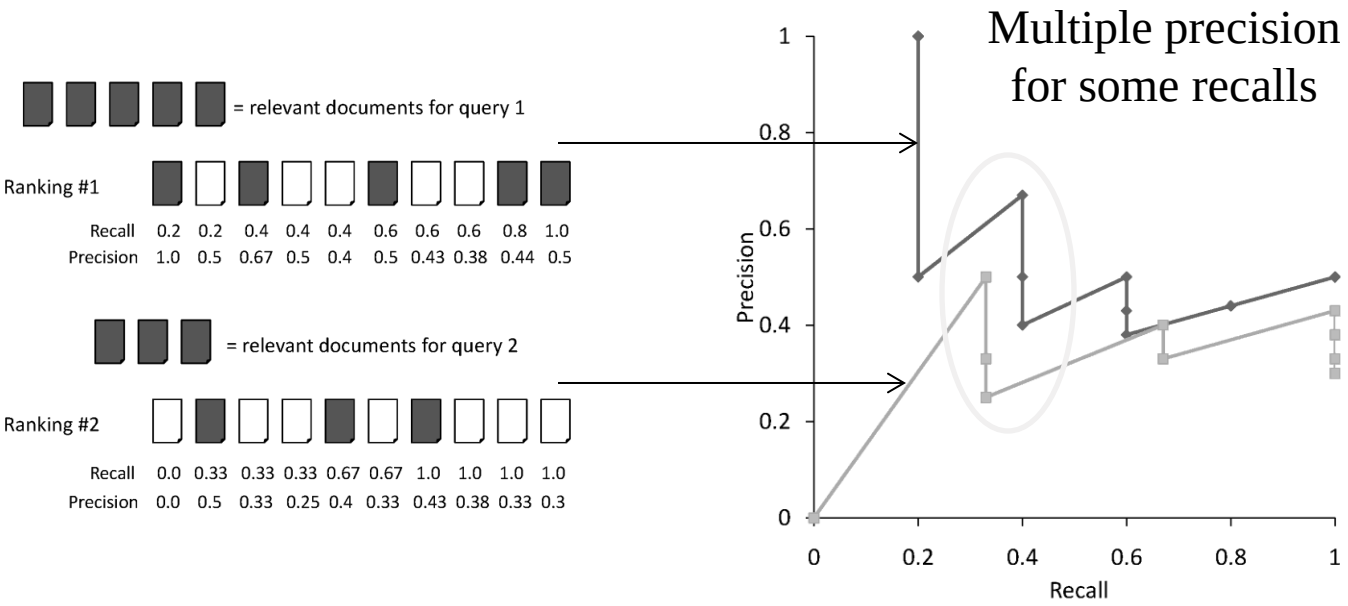
$$mean\ average\ precision = (0.62 + 0.44)/2 = 0.53$$

Recall-Precision Graph

- Slope downward from left to right
  - As more relevant items are retrieved (recall increases), the more nonrelevant items are retrieved (precision decreases).
- Commonly used method for comparing systems.
- Curves closest to the upper right-hand corner of the graph (where recall and precision are maximized) indicate the best performance

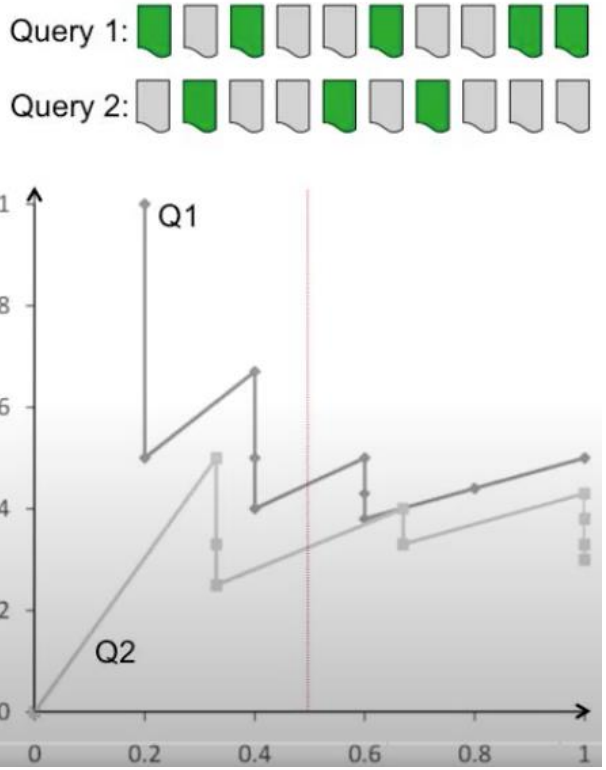


Recall-Precision Graph

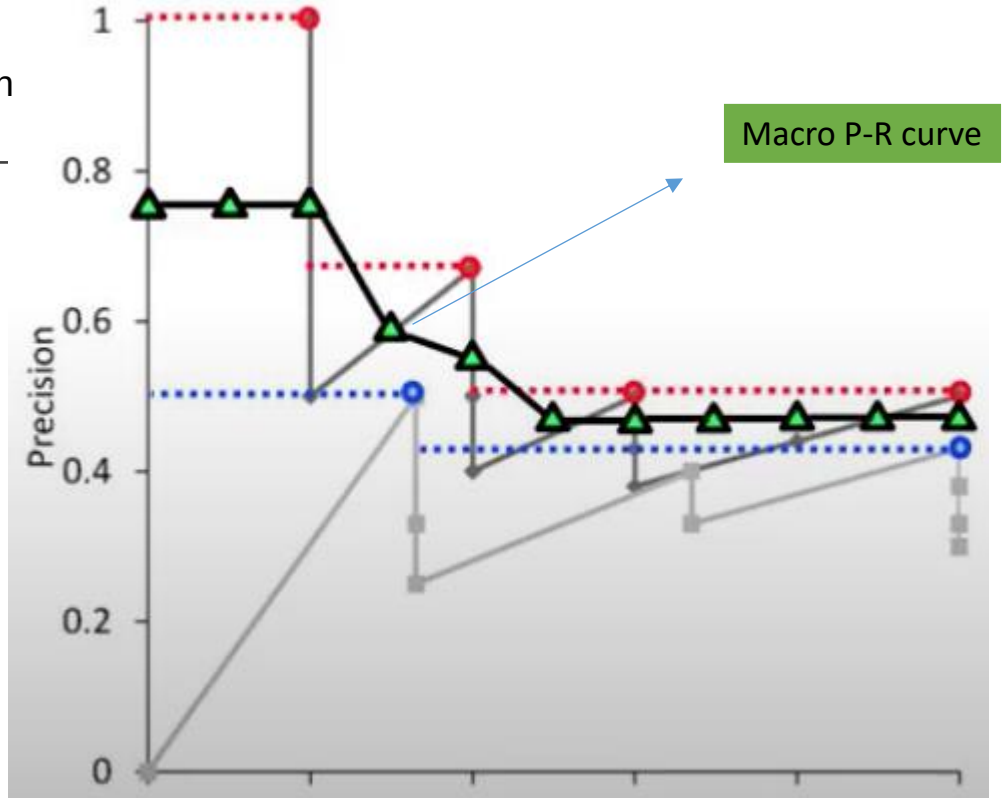


Recall-precision plot (raw)

- Plot precision vs. recall
  - one curve per query
  - detailed picture, but...
    - erratic behaviour
    - want to “average” curves
- Standard averaging
  - at fixed recall levels
    - 0.1, 0.2, 0.3 … 1.0
  - what is precision at 0.5?
  - need to interpolate, how?



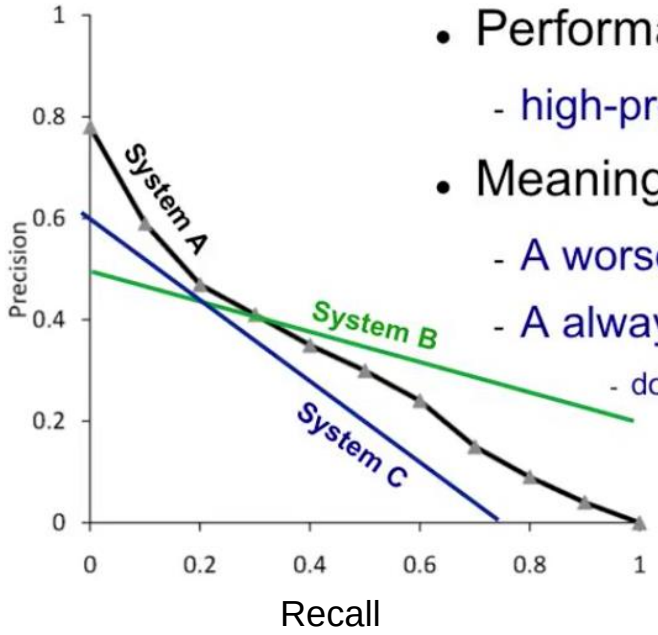
Average Precision



Average Precision (AP) = mean_q AP_q (Area under P-R curve of query q)

Mean Average Precision (mAP) = mean_c AP[c]

Comparing retrieval systems using the “average” P-R curves



- Averaged over 50 queries
- Performance for all user types
  - high-precision and high-recall
- Meaningful system comparisons
  - A worse than B if recall important
  - A always better than C
    - dominates at all recall levels

Recap: Confusion matrix

- The confusion matrix (easily generalize to multi-class)

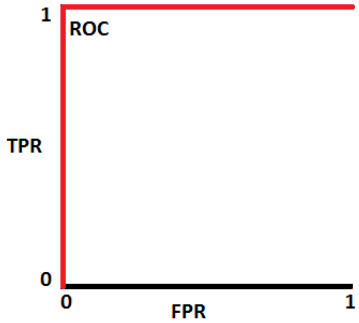
	Actual: Positive	Actual: Negative
Predicted: Positive	tp	fp
Predicted: Negative	fn	tn

- Machine Learning methods usually minimize FP+FN
- TPR (True Positive Rate): TP / (TP + FN) = Recall
- FPR (False Positive Rate): FP / (TN + FP)

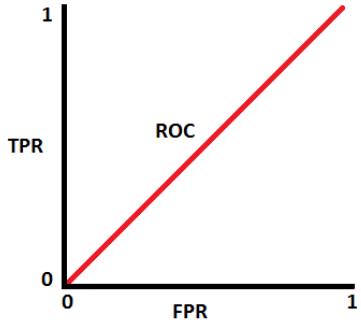
ROC Curves

- A **receiver operating characteristic curve**, i.e. **ROC curve**, is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold is varied.

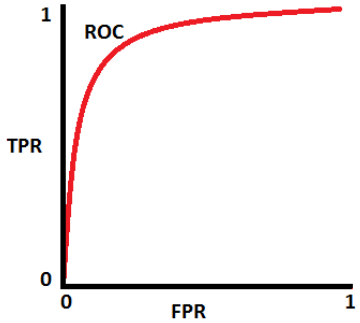
ROC Curves



This is an ideal situation. Model has an ideal measure of separability. It is perfectly able to distinguish between positive class and negative class.

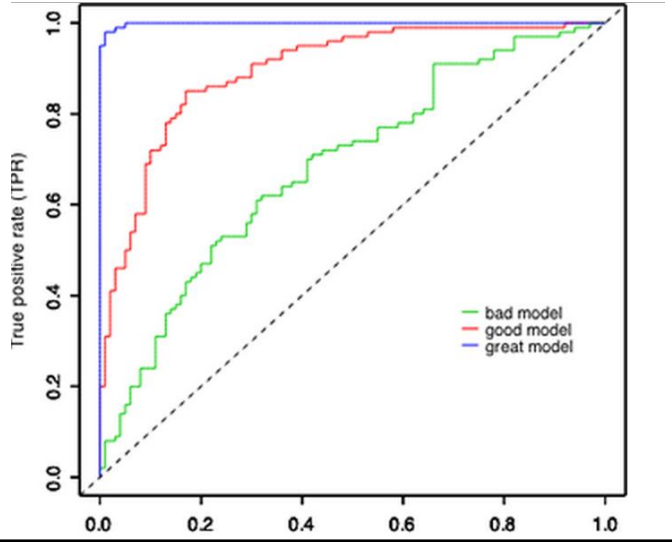


This is the worst situation. When AUC is approximately 0.5, model has no discrimination capacity to distinguish between positive class and negative class. Random predictions.



Multiple ROC Curves

- Comparison of multiple classifiers is usually straight-forward especially when no curves cross each other. Curves close to the perfect ROC curve have a better performance level than the ones closes to the baseline.



PR Curves Vs ROC Curves

- ROC curve : Recall vs False Positive Rate
- PR curve : Precision vs Recall
- If your question is, "How well can this classifier be expected to perform *in general*, go with a ROC curve
- If true negative is not much valuable to the problem, or negative examples are abundant. Then, PR-curve is typically more appropriate.
  - For example, if the class is highly imbalanced and positive samples are very rare, then use PR-curve.
  - How meaningful is a positive result from my classifier