27.09.2024

Statistical Methods in AI (CS7.403)

Lecture-14: Neural Networks-2

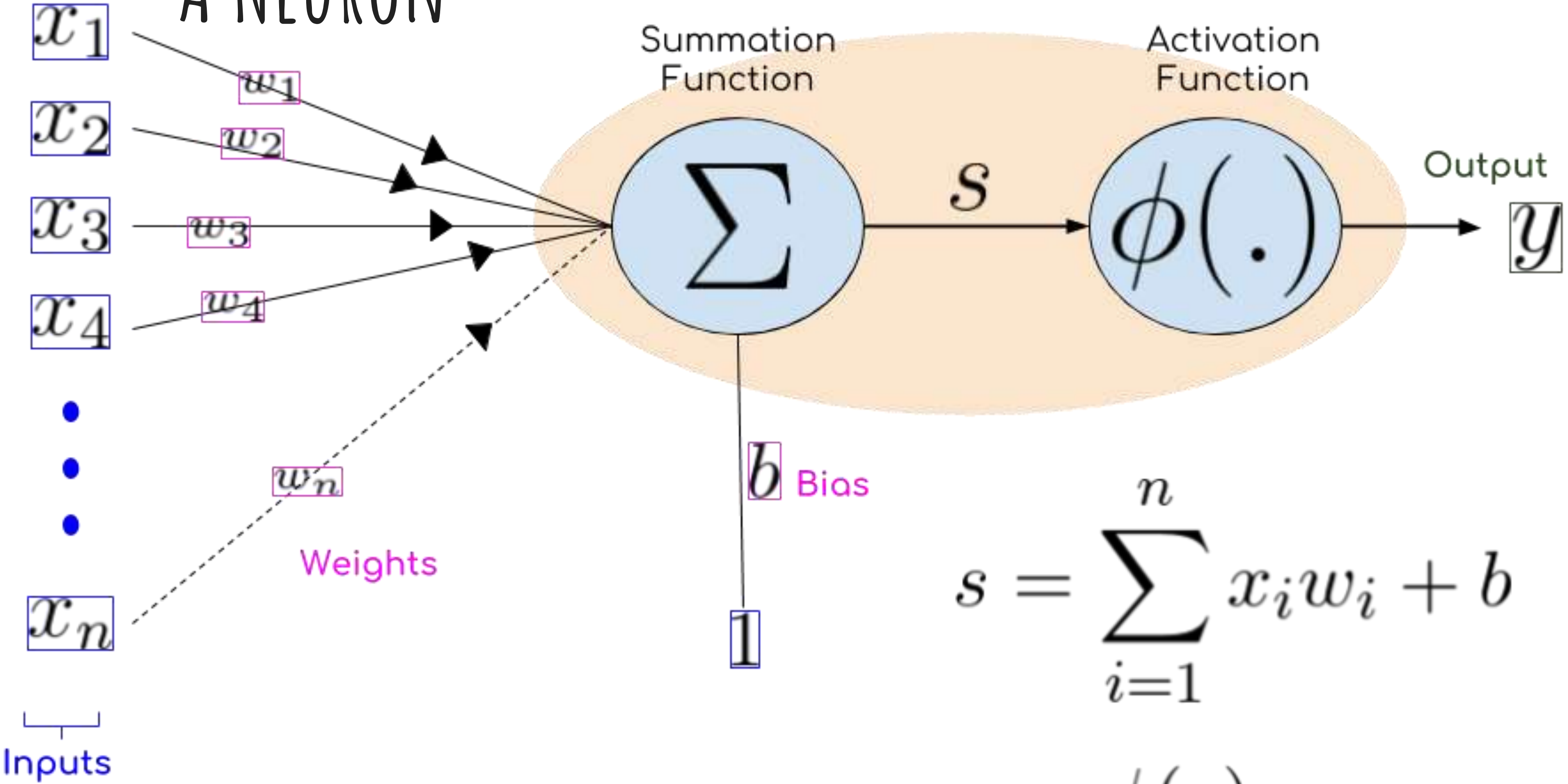
Ravi Kiran (ravi.kiran@iiit.ac.in)

https://ravika.github.io

@vikataravi

Center for Visual Information Technology (CVIT)
IIIT Hyderabad

A Neuron



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

$$y = \phi(s)$$

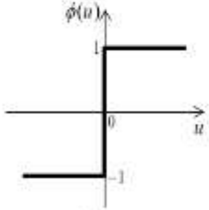
The Perceptron Cost Function

- Prediction is correct if $y^{(i)}\boldsymbol{x}^{(i)}\boldsymbol{\theta} > 0$
- 0/1 loss

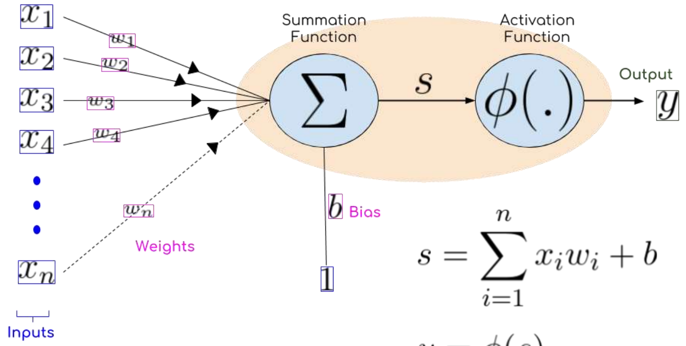
$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(sign(x^{(i)}\boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise

sign function



$$\phi_{sign}(u) = \begin{cases} 1 & \text{if } u \geq 0 \\ -1 & \text{otherwise} \end{cases}$$



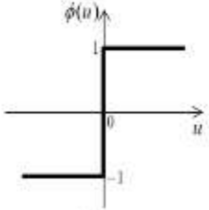
$$s = \sum_{i=1}^{n} x_i w_i + b$$

$$y = \phi(s)$$

Based on slide by Alan Fern

The Perceptron Cost Function

sign function

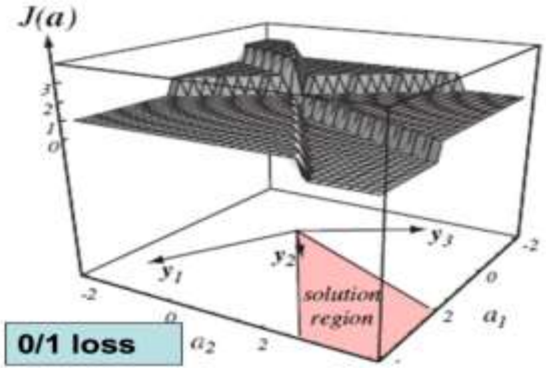


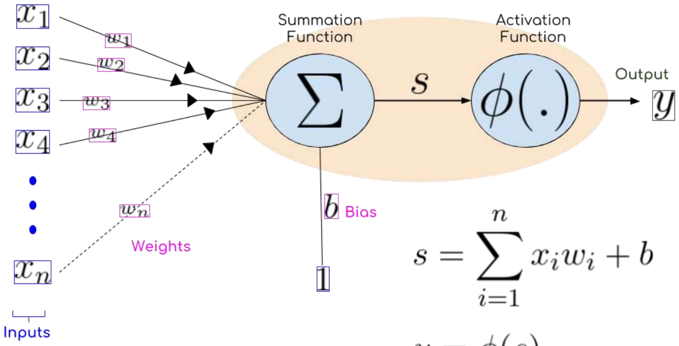
$$\phi_{sign}(u) = \begin{cases} 1 & \text{if } \; u \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- Prediction is correct if $y^{(i)} \boldsymbol{x}^{(i)} \boldsymbol{\theta} > 0$
- 0/1 loss

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n} \sum_{i=1}^{n} \ell(sign(x^{(i)} \boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise





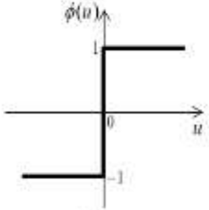
$$s = \sum_{i=1}^{n} x_i w_i + b$$

$$y = \phi(s)$$

Based on slide by Alan Fern

The Perceptron Cost Function

sign function

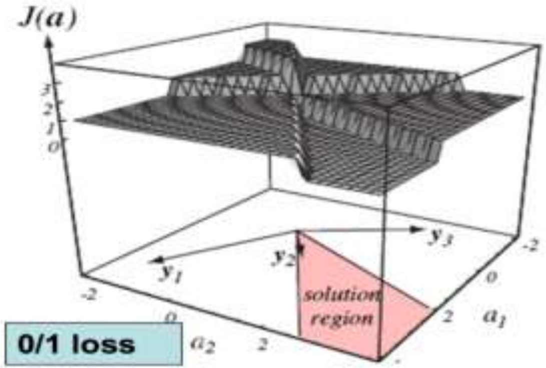


$$\phi_{sign}(u) = \begin{cases} 1 & \text{if} \quad u \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

- Prediction is correct if $y^{(i)}\boldsymbol{x}^{(i)}\boldsymbol{\theta} > 0$
- 0/1 loss

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(sign(x^{(i)}\boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise



Doesn't produce a useful gradient

No gradient ➔ We cannot use gradient-based optimization methods. Does NOT mean ‘unsolvable’

Based on slide by Alan Fern

Improving the perceptron

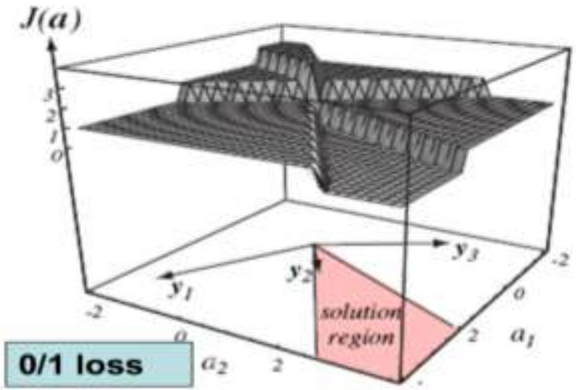
Problem #0: Non-differentiable loss function

Minimize # of misclassifications

- Could have used 0/1 loss

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(sign(x^{(i)}\boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise



Doesn't produce a useful gradient

Improving the perceptron

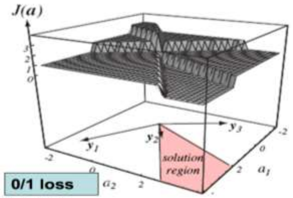
Minimize # of misclassifications

Problem #0: Non-differentiable loss function

- Could have used 0/1 loss

$$J_{0/1}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell(sign(x^{(i)}\boldsymbol{\theta}), y^{(i)})$$

where $\ell()$ is 0 if the prediction is correct, 1 otherwise



Doesn't produce a useful gradient

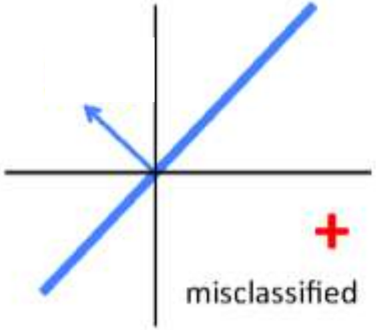
Based on slide by Alan Fern

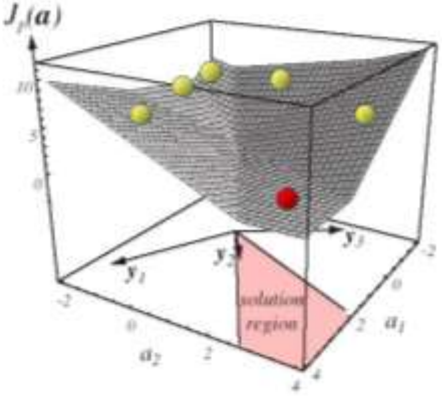
8

Solution: Minimize the 'misclassification distance'

$$J_p(\mathbf{a}) = \sum_{\mathbf{x}\in\mathbb{M}} (-\mathbf{a}^{\mathrm{T}}\mathbf{x})$$

$\mathbb{M}$ ➡ Set of misclassified samples

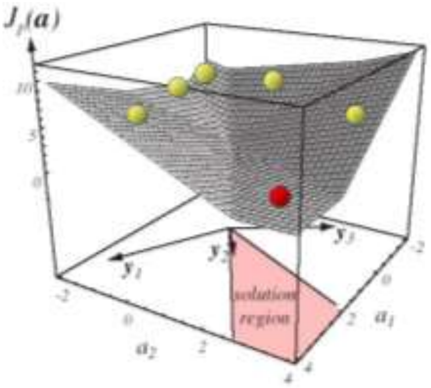




Solution: Minimize the 'misclassification distance'

$$J_p(\mathbf{a}) = \sum_{\mathbf{x}\in\mathbb{M}} (-\mathbf{a}^{\mathbf{T}}\mathbf{x})$$

$\mathbb{M}$ ➡ Set of misclassified samples



$$\nabla J_p(\mathbf{a}) = \sum_{\mathbf{x}\in\mathbb{M}} (-\mathbf{x})$$

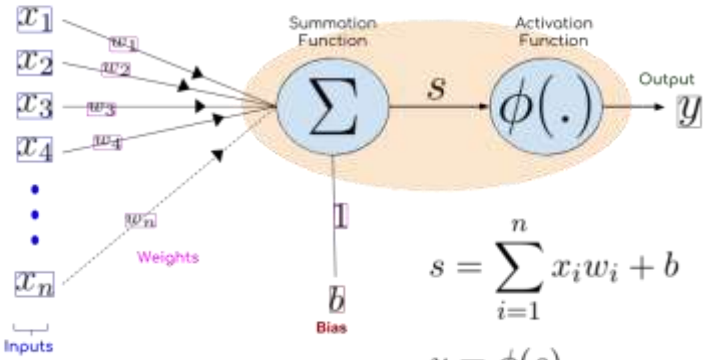
$$a^{(k+1)} = a^k - \eta^k \nabla J_p(a)$$

$$a^{(k+1)} = a^k + \eta^k \sum_{\mathbf{x}\in\mathbb{M}} \mathbf{x}$$

c.f. logistic regression

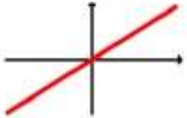
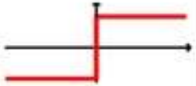
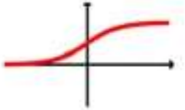
$$\nabla_a J\left(\hat{y}^{(i)}, y^{(i)}\right) = (\hat{y}^{(i)} - y^{(i)})\hat{x}^{(i)}$$

ACTIVATION FUNCTIONS

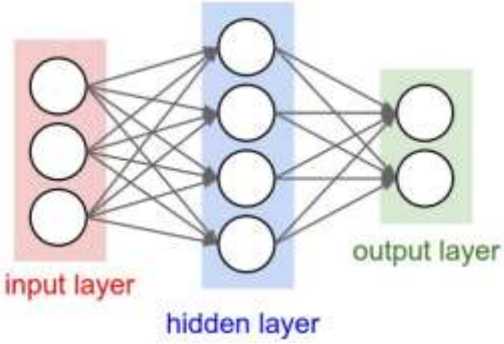


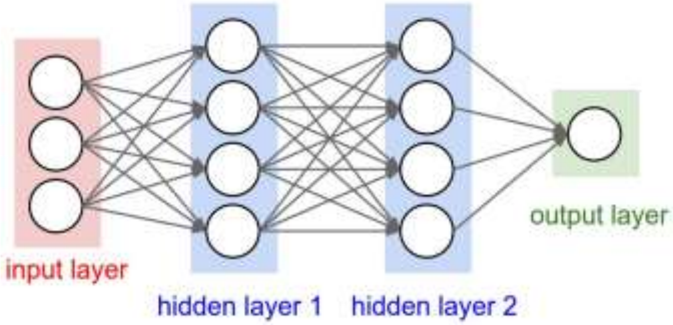
$$s = \sum_{i=1}^{n} x_i w_i + b$$

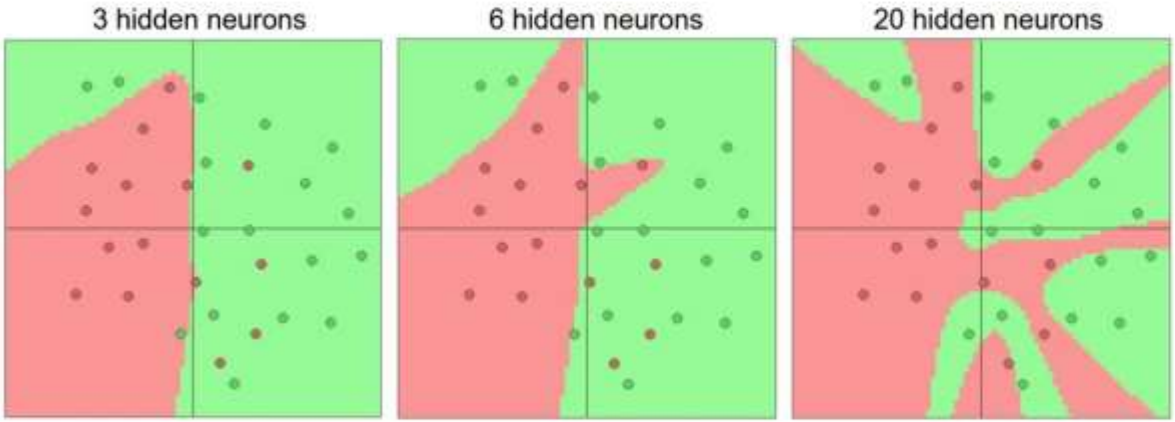
$$y = \phi(s)$$

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	$\phi(z) = \begin{cases} 0 & z \le -\frac{1}{2} \\ z + \frac{1}{2} & -\frac{1}{2} \le z \le \frac{1}{2} \\ 1 & z \ge \frac{1}{2} \end{cases}$	Support vector machine	
Logistic (sigmoid)	$\phi(z) = \dfrac{1}{1 + e^{-z}}$	Logistic regression,	
Hyperbolic Tangent (tanh)	$\phi(z) = \dfrac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$		
ReLU	$\phi(z) = \begin{cases} 0 & z < 0 \\ z & z > 0 \end{cases}$		

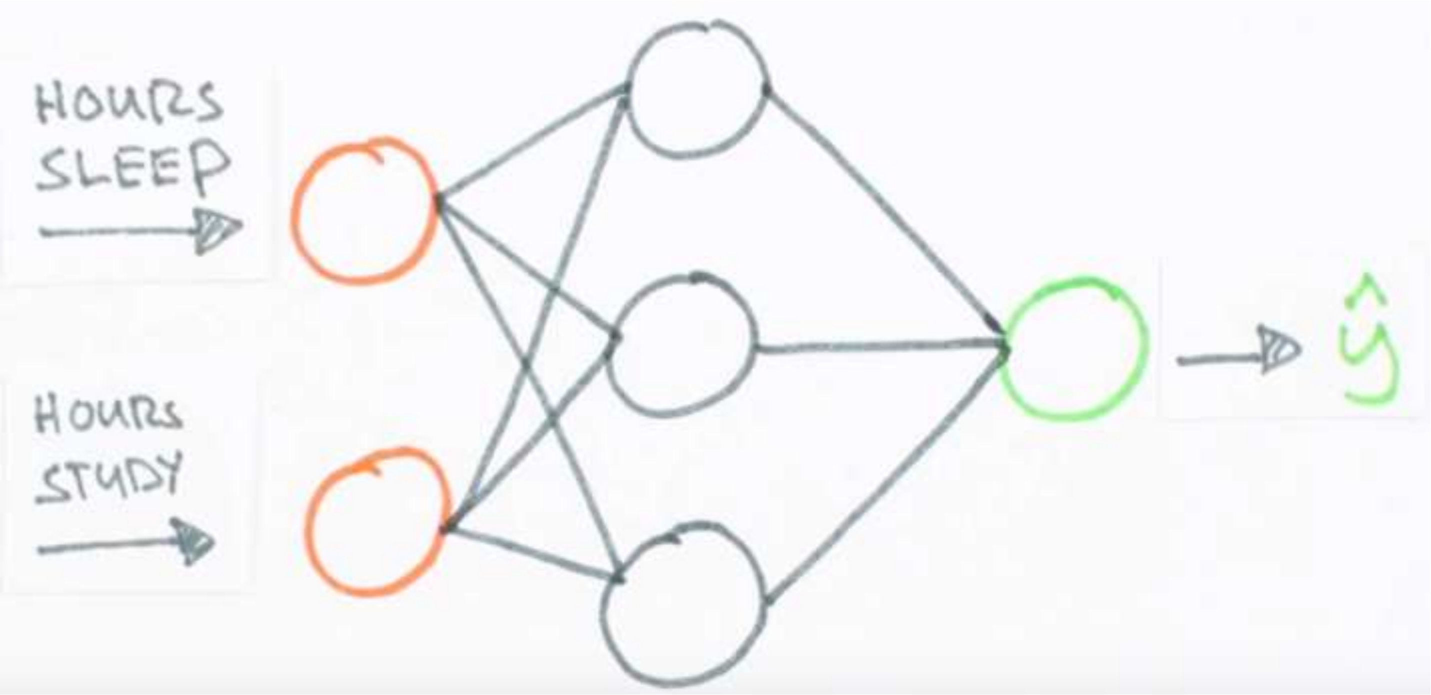
Why Use Only One Neuron ?





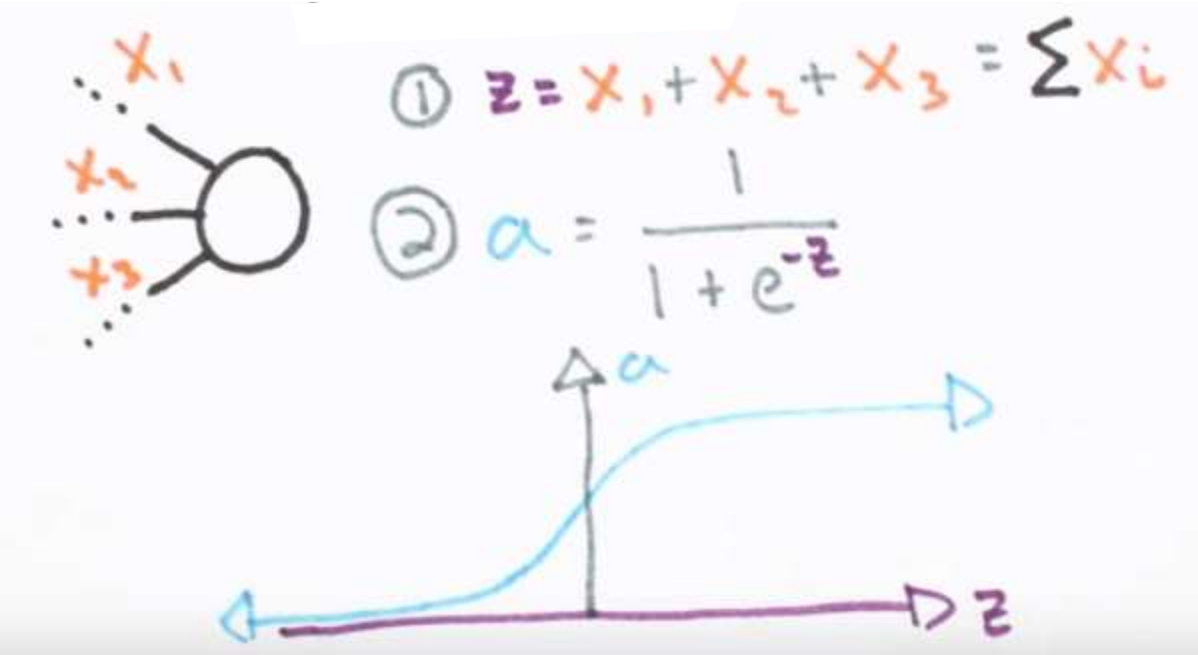


Multi-Neuron Networks :: Architecture



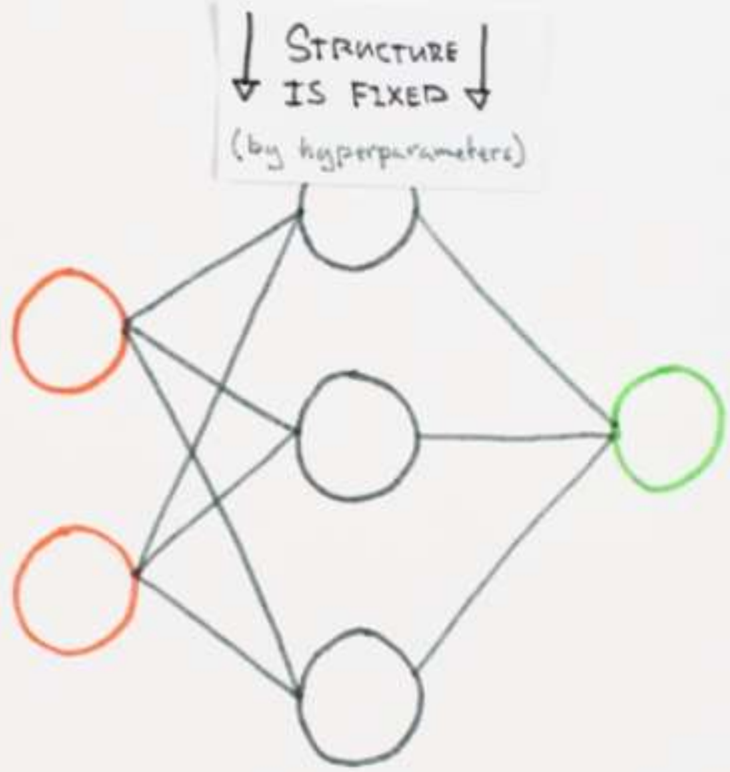
Multi-Neuron Networks :: Architecture

Neuron



Multi-Neuron Networks :: Architecture

```
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)
```



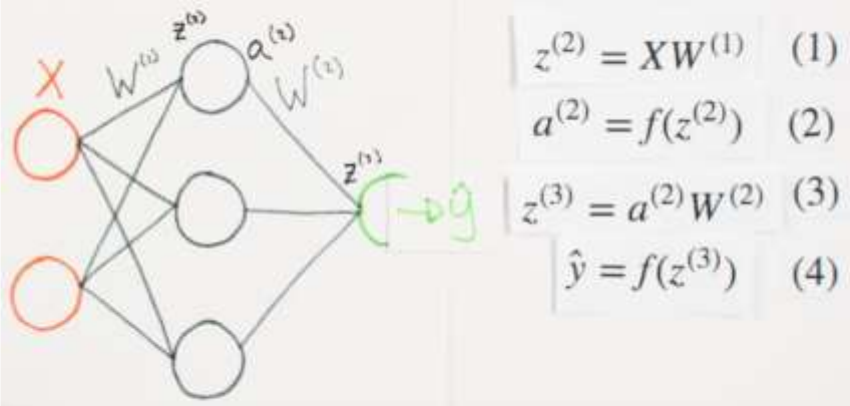
Multi-Neuron Networks :: training

Initialize network with random weights

While [not converged]

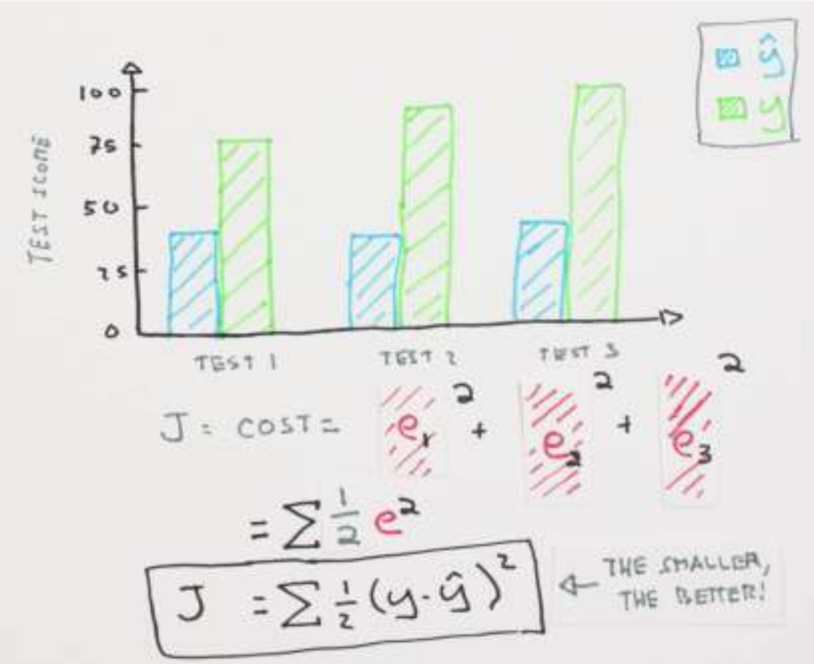
- Do forward prop
- Do backprop and determine change in weights
- Update ALL weights in ALL layers

Multi-Neuron Networks :: Forward Propagation



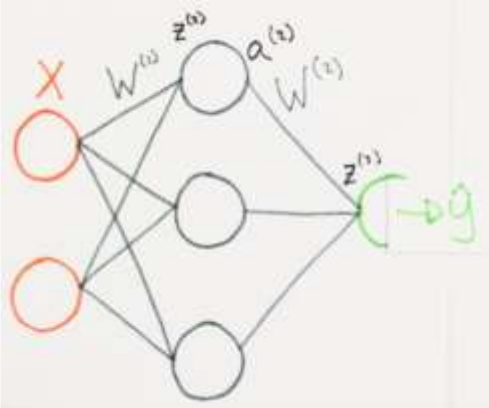

```python
def forward(self, X):
    #Propogate inputs though network
    self.z2 = np.dot(X, self.W1) # z2 = X * W1
    self.a2 = self.sigmoid(self.z2) # a2 = sigmoid(z2)
    self.z3 = np.dot(self.a2, self.W2) # z3 = a2 * W2
    yHat = self.sigmoid(self.z3) # yHat = sigmoid(z3)
    return yHat

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

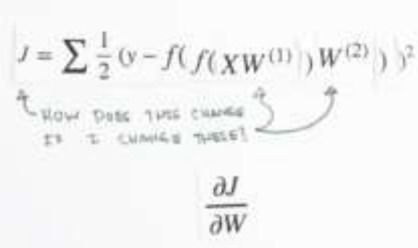
Multi-Neuron Networks :: Gradient Descent

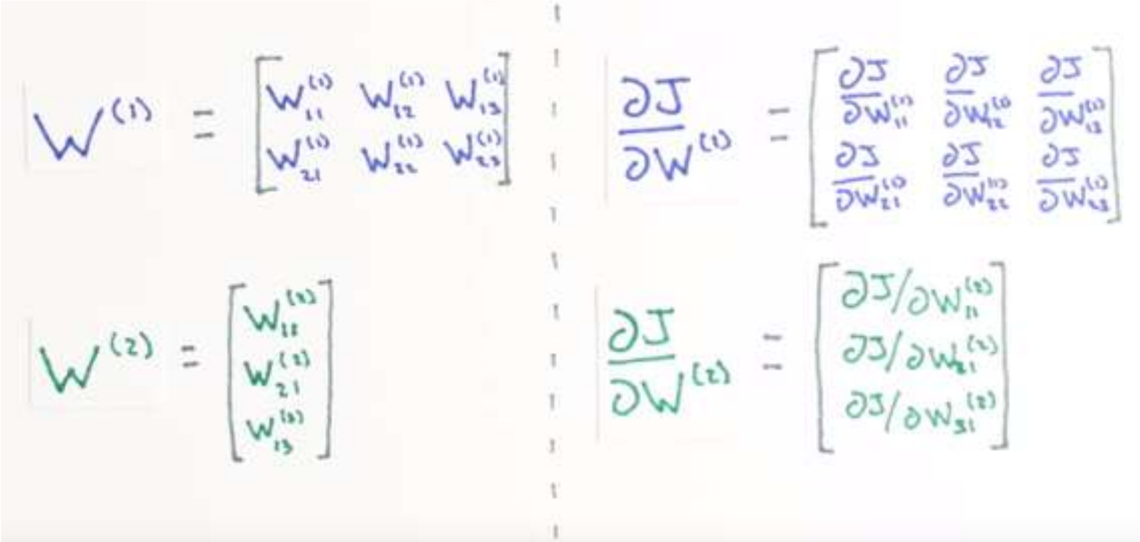



```
def costFunction(self, X, y):
    #Compute cost for given X,y, use weights already stored in class.
    self.yHat = self.forward(X)
    J = 0.5*sum((y-self.yHat)**2)
    return J
```

Multi-Neuron Networks :: Backpropagation







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?

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

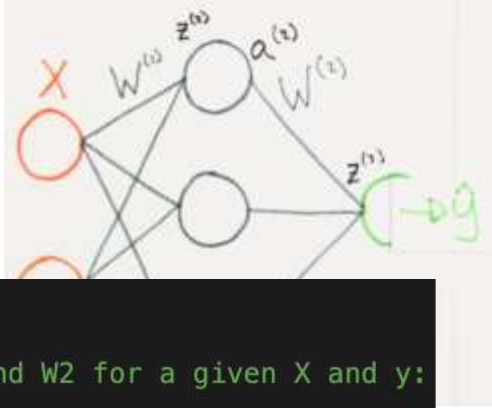
$$z^{(2)} = XW^{(1)} \quad (1)$$

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

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

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

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



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

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

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

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

```python
# 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
```

$$\begin{bmatrix} \frac{\partial J}{\partial W^{(1)}_{12}} \\ \frac{\partial J}{\partial W^{(1)}_{13}} \end{bmatrix}$$

Multi-Neuron Networks :: training

```
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
```

Initialize network with random weights

While [not converged]

- Do forward prop
- Do backprop and determine change in weights
- Update all weights in all layers

One Iteration

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?

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

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

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

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

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

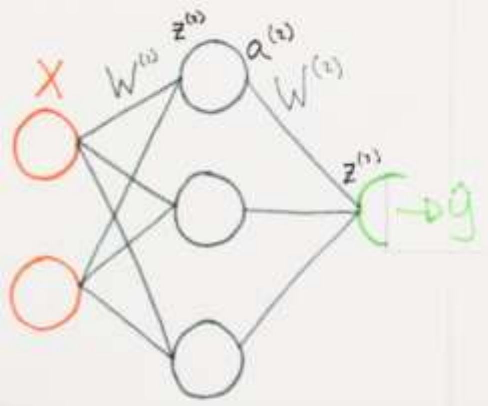
$$\hat{y} = f(z^{(3)}) \quad (4)$$

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

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

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

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




```
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]
```

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

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

MULTI-NEURON NETWORK – NUMERICAL GRADIENT CHECKING

MULTI-NEURON NETWORK – NUMERICAL GRADIENT CHECKING

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - F(x)}{\Delta x}$$

$$f(x) = x^2$$

MULTI-NEURON NETWORK – NUMERICAL GRADIENT CHECKING

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - F(x)}{\Delta x}$$

$$f(x) = x^2$$

$$f'(x) = \lim_{\Delta x \to 0} \frac{(x+\Delta x)^2 - x^2}{\Delta x}$$

$$= \lim_{\Delta x \to 0} \frac{x^2 + 2x\Delta x + \Delta x^2 - x^2}{\Delta x}$$

$$= \lim_{\Delta x \to 0} \frac{\Delta x^2 + 2x\Delta x}{\Delta x}$$

$$= \lim_{\Delta x \to 0} \Delta x + 2x = \boxed{2x}$$

MULTI-NEURON NETWORK – NUMERICAL GRADIENT CHECKING

$$f'(x) = \lim_{\Delta x \to 0} \frac{f(x+\Delta x) - F(x)}{\Delta x}$$

$$f(x) = x^2$$

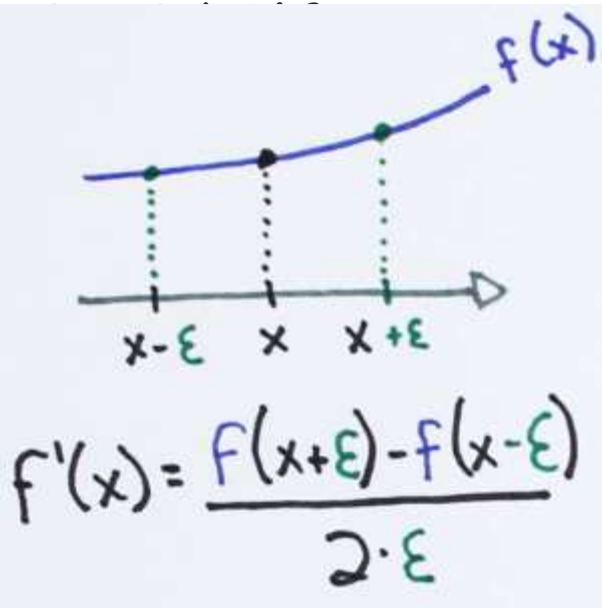
$$f'(x) = \lim_{\Delta x \to 0} \frac{(x+\Delta x)^2 - x^2}{\Delta x}$$

$$= \lim_{\Delta x \to 0} \frac{x^2 + 2x\Delta x + \Delta x^2 - x^2}{\Delta x}$$

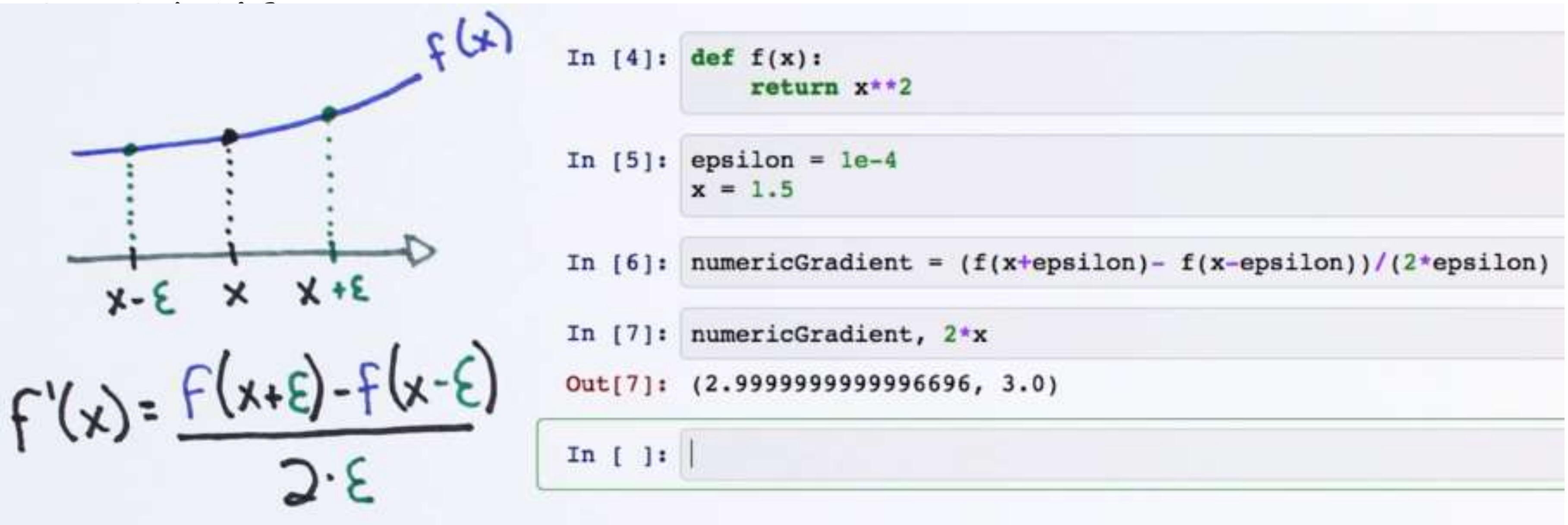
$$= \lim_{\Delta x \to 0} \frac{\Delta x^2 + 2x\Delta x}{\Delta x}$$

$$= \lim_{\Delta x \to 0} \Delta x + 2x = \boxed{2x}$$

MULTI-NEURON NETWORK – NUMERICAL GRADIENT



MULTI-NEURON NETWORK – NUMERICAL GRADIENT



MULTI-NEURON NETWORK – NUMERICAL GRADIENT CHECKING

Parameter vector θ

→ $\theta \in \mathbb{R}^n$ (E.g. θ is "unrolled" version of $\underline{\Theta^{(1)}}, \underline{\Theta^{(2)}}, \underline{\Theta^{(3)}}$)

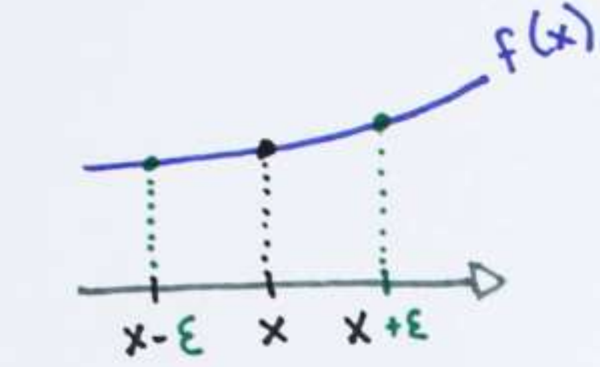
→ $\theta = [\theta_1, \theta_2, \theta_3, \ldots, \theta_n]$

→ $$\frac{\partial}{\partial \underline{\theta_1}} J(\theta) \approx \frac{J(\boxed{\theta_1+\epsilon}, \theta_2, \theta_3, \ldots, \theta_n) - J(\boxed{\theta_1-\epsilon}, \theta_2, \theta_3, \ldots, \theta_n)}{2\epsilon}$$

$$\frac{\partial}{\partial \theta_2} J(\theta) \approx \frac{J(\theta_1, \theta_2+\epsilon, \theta_3, \ldots, \theta_n) - J(\theta_1, \theta_2-\epsilon, \theta_3, \ldots, \theta_n)}{2\epsilon}$$

$$\vdots$$

MULTI-NEURON NETWORK – NUMERICAL GRADIENT CHECKING



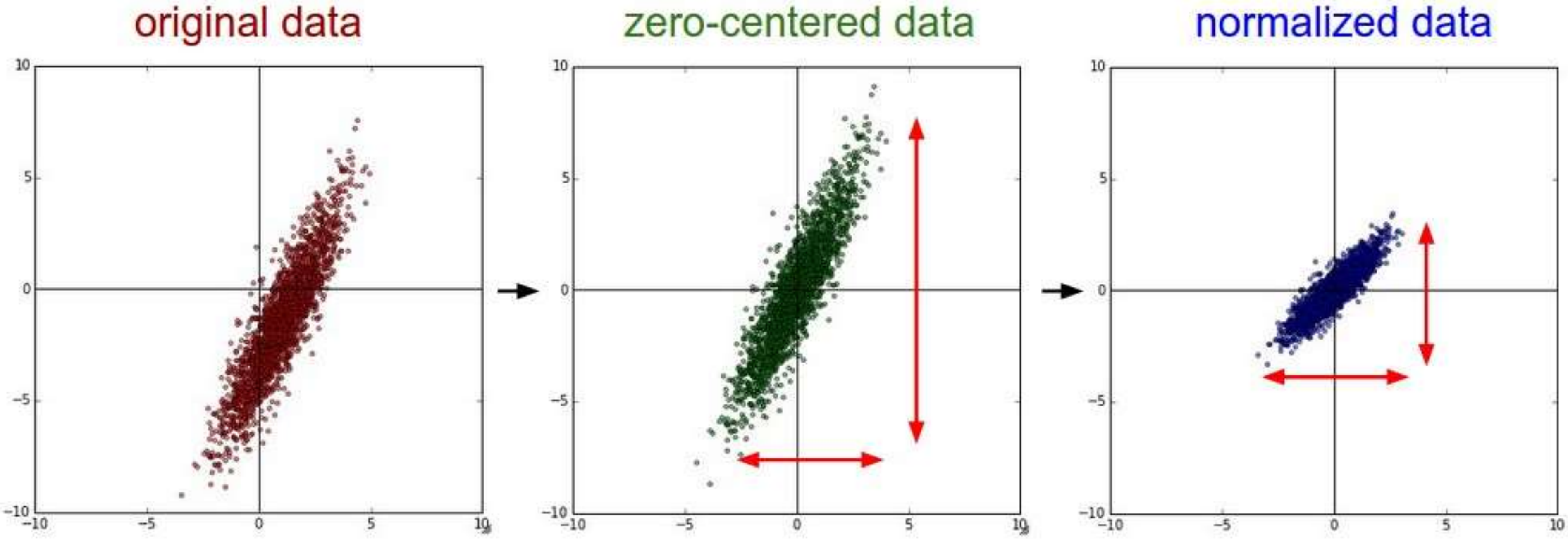
$$f'(x) = \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\cdot\varepsilon}$$

```
In [3]: numgrad = computeNumericalGradient(NN, X, y)

In [4]: grad = NN.computeGradients(X, y)

In [5]: numgrad

Out[5]: array([ -7.10752568e-03,  -6.30194392e-03,  -4.96392693e-03,
                -6.55946987e-04,   7.57595597e-05,  -1.11297012e-03,
                -8.81243102e-03,  -4.45550176e-03,  -1.93471143e-02])

In [6]: grad

Out[6]: array([ -7.10752569e-03,  -6.30194393e-03,  -4.96392693e-03,
                -6.55946985e-04,   7.57595604e-05,  -1.11297012e-03,
                -8.81243100e-03,  -4.45550176e-03,  -1.93471142e-02])
```

```
In [7]: norm(grad-numgrad)/norm(grad+numgrad)

Out[7]: 1.9824969610227768e-09
```

Data Setup

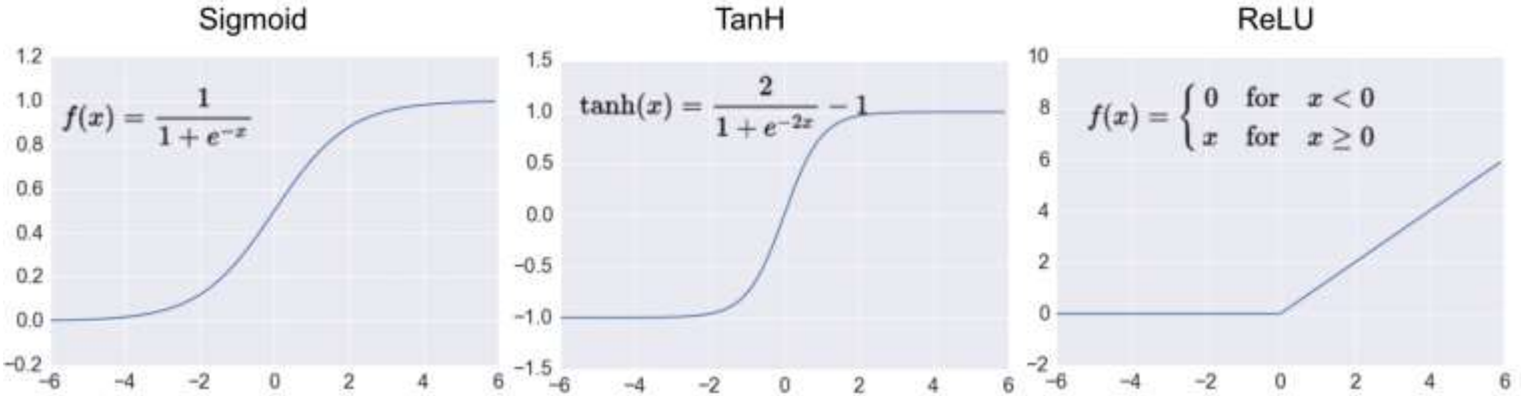
- Preprocessing:



Weight initialization

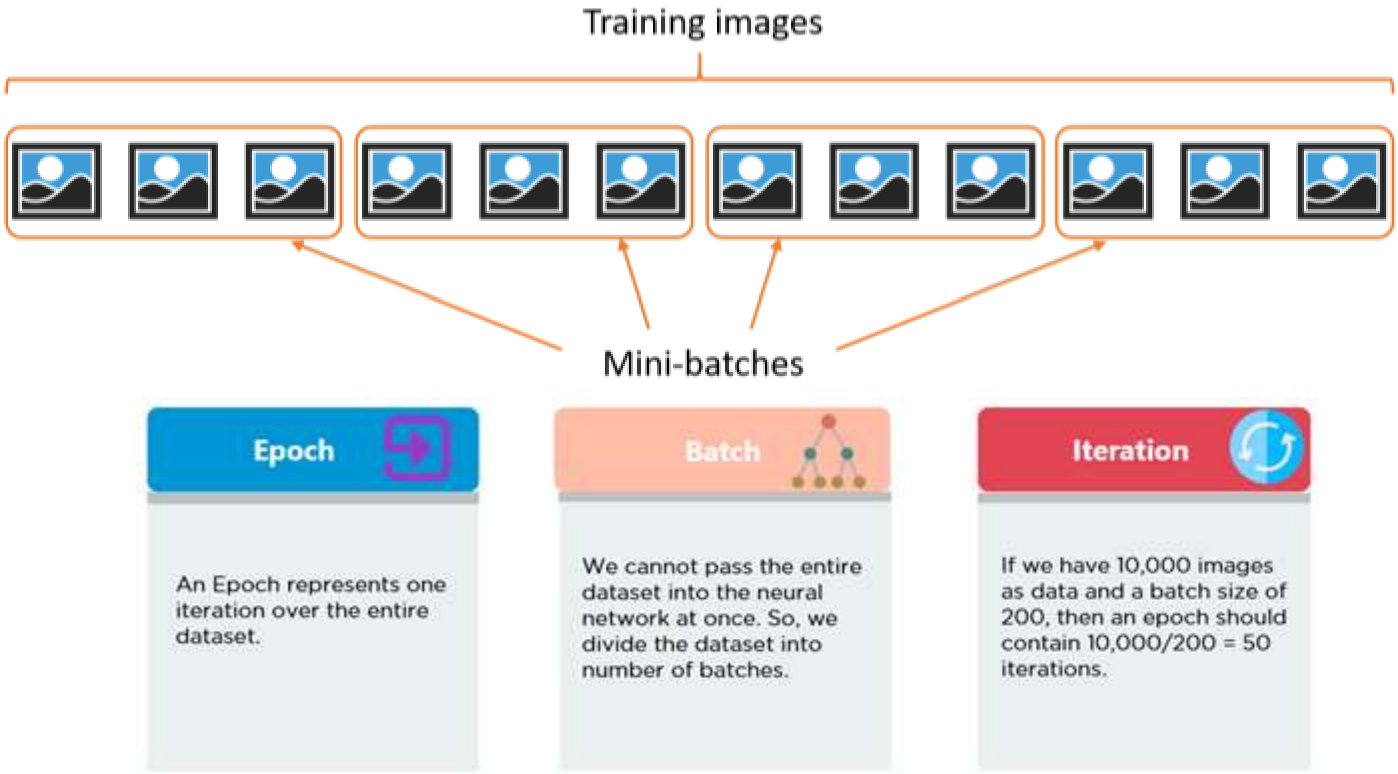
- All Zeros
- Random [0,1]
- Random [-1,1]
- `w = np.random.randn(n) * sqrt(2.0/n), n = # of inputs to neuron`

Activation functions



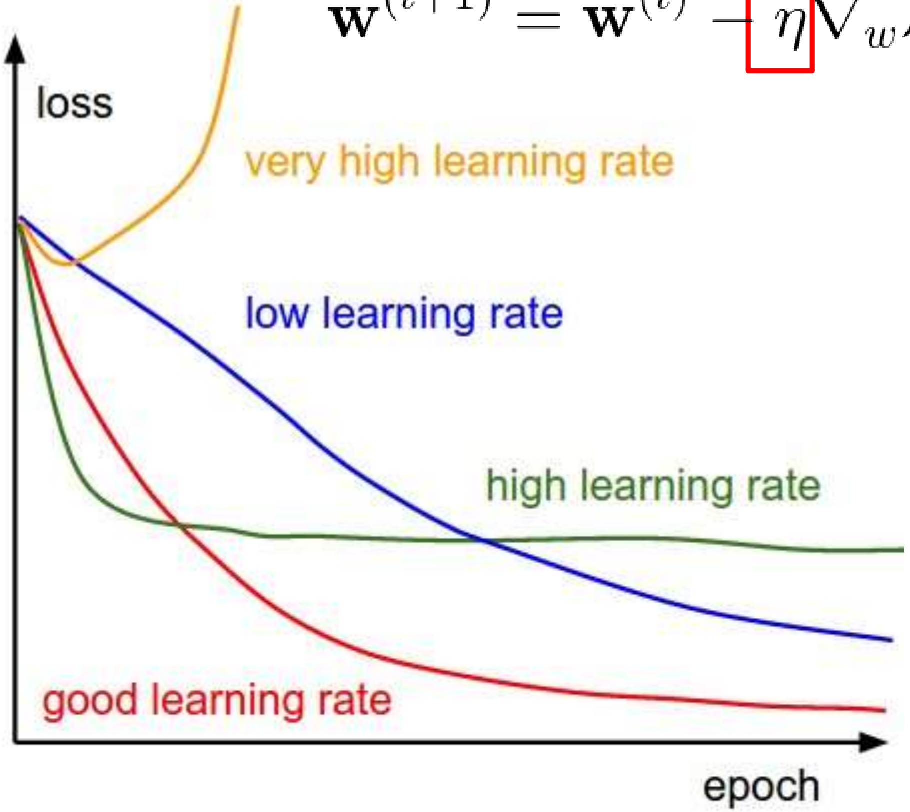
MINIBATCH VS SINGLE

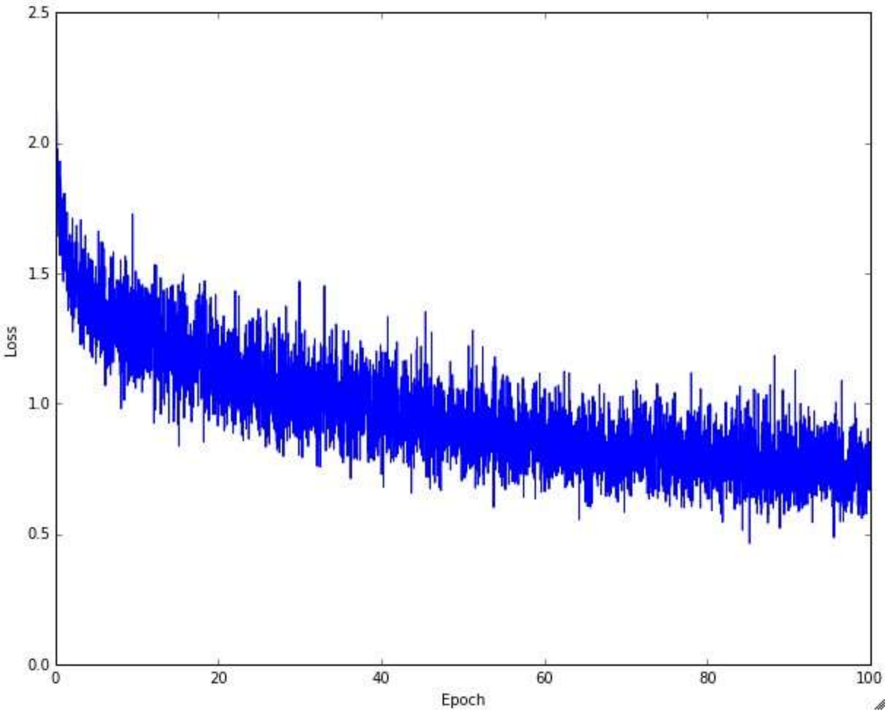
- Average error, gradients



Training – setting learning rate

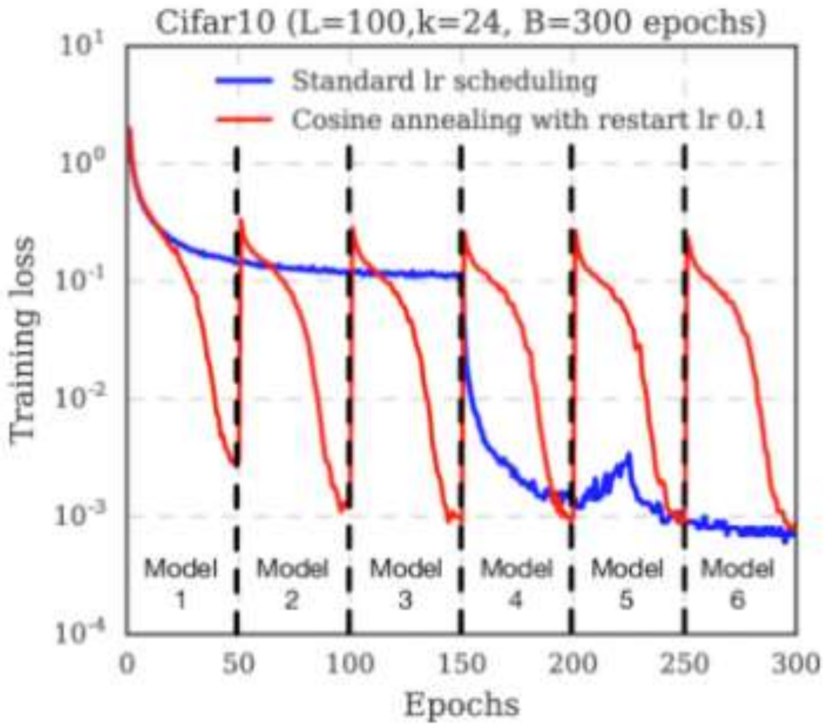
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \boxed{\eta} \nabla_w \mathcal{L}(\mathbf{w})$$

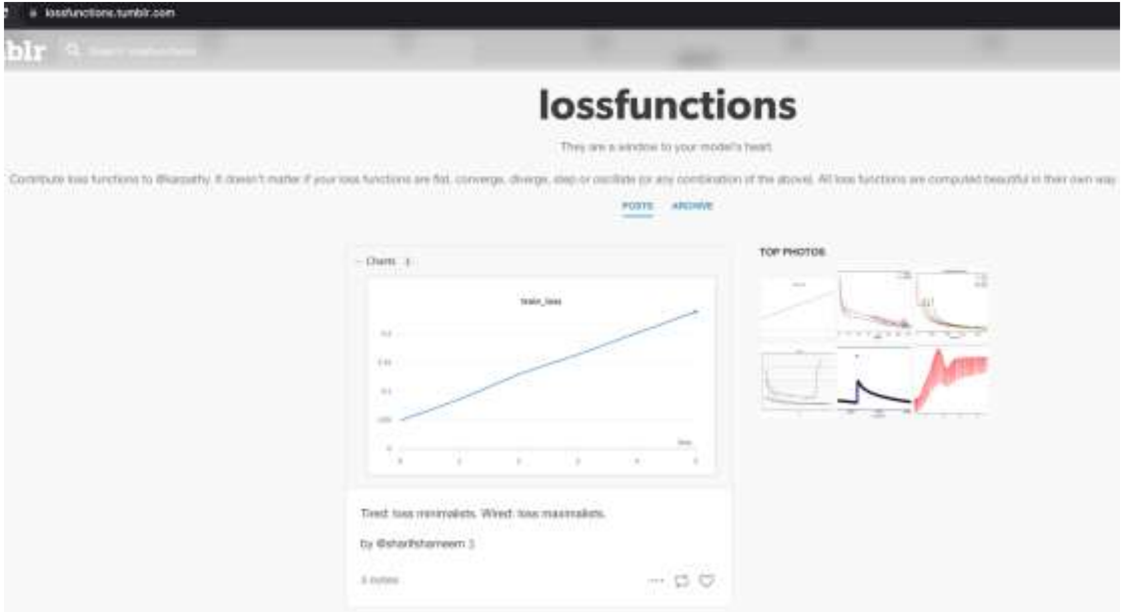


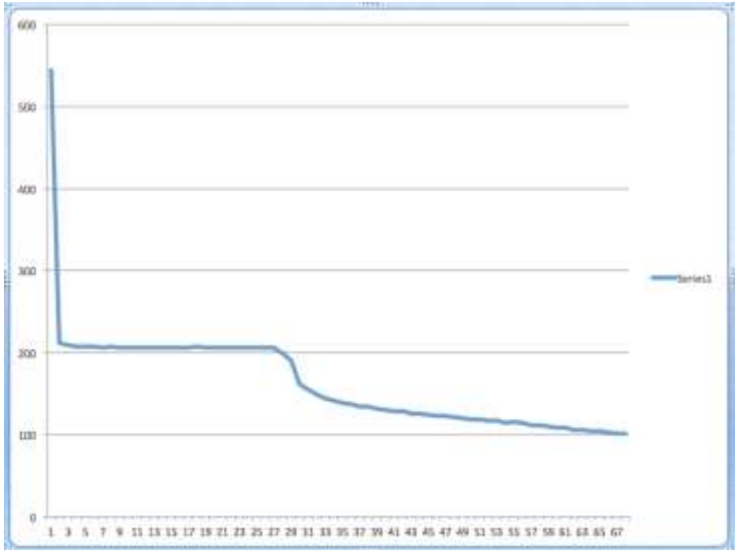


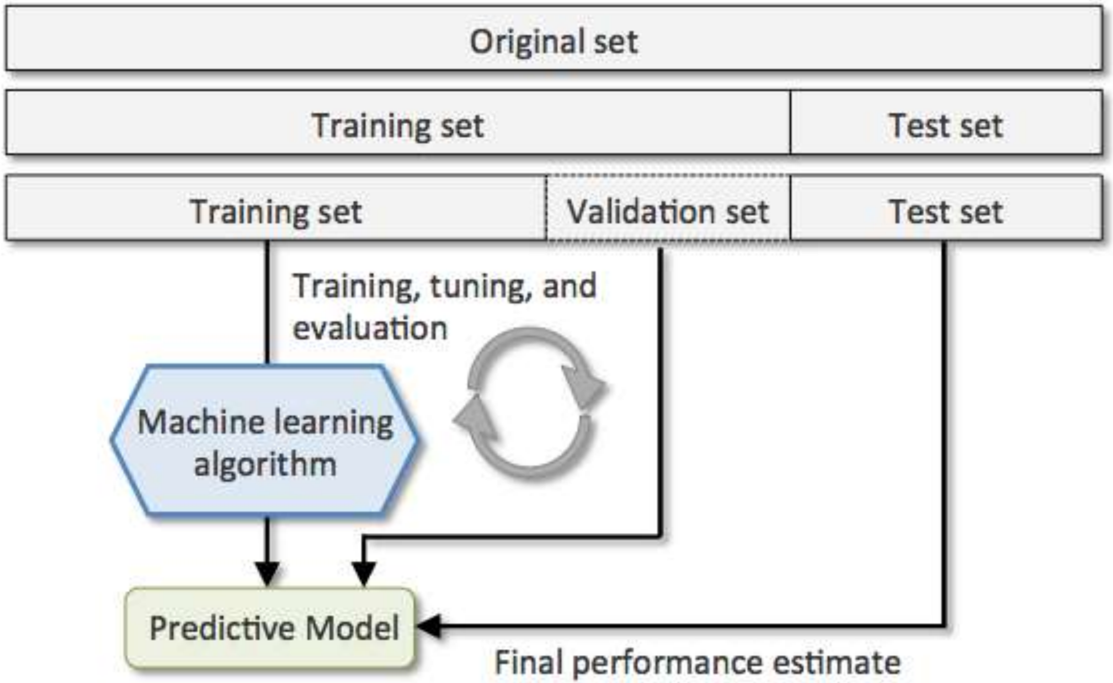
Training – setting learning rate

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

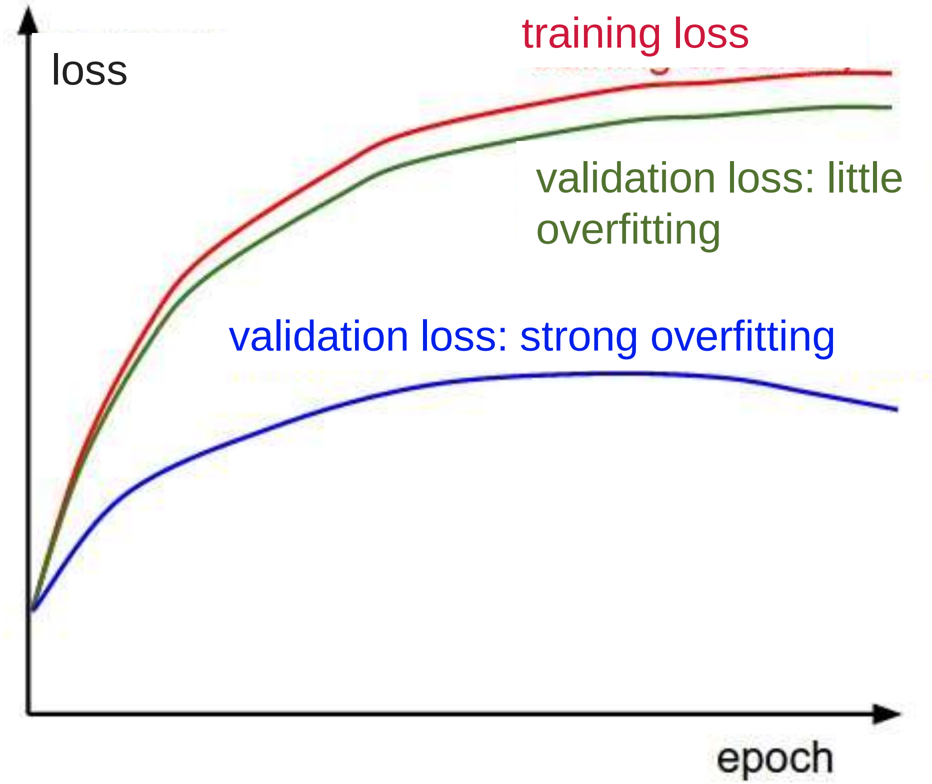



lossfunctions.tumblr.com
lossfunctions
They are a window to your model's heart
POSTS
ARCHIVE
TOP PHOTOS
Tired: loss minimalists. Wired: loss maximalists.

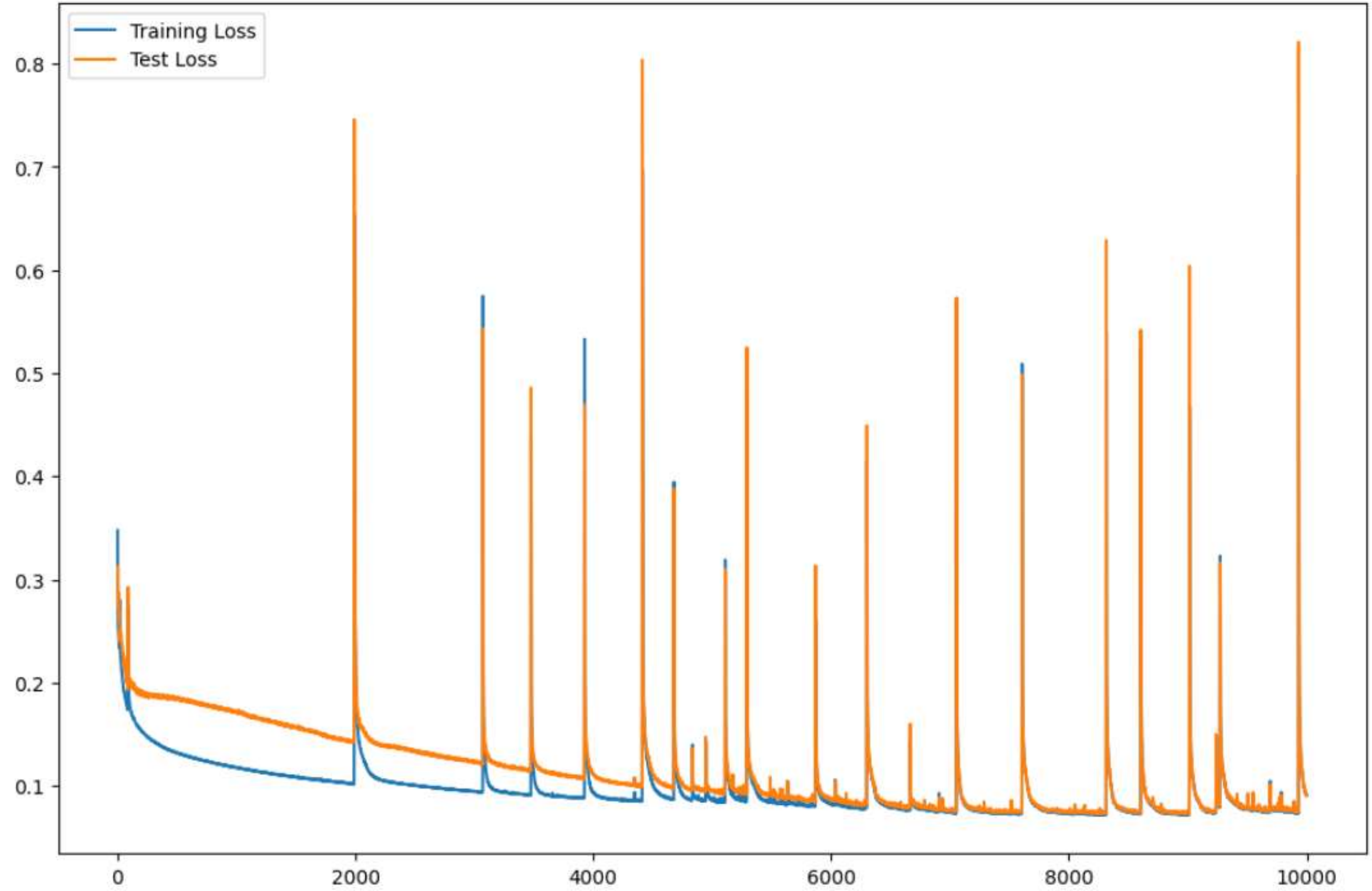

600
500
400
300
200
100
0
Series1


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

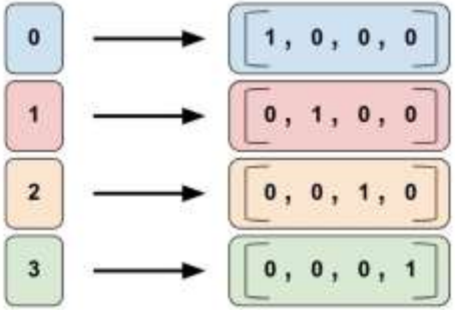
When to stop training



When to stop training

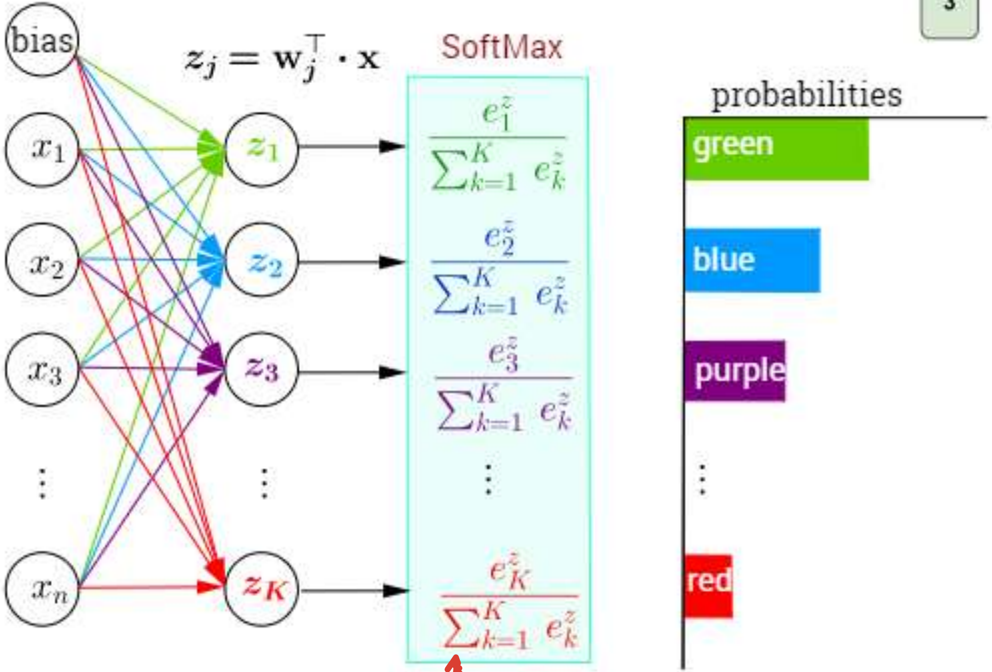

Training Loss
Test Loss
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
2000
4000
6000
8000
10000

Classification loss



Multi-Class Classification with NN and SoftMax Function

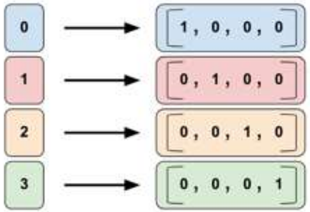
$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} \mathbf{w}_1^\top \\ \mathbf{w}_2^\top \\ \mathbf{w}_3^\top \\ \vdots \\ \mathbf{w}_K^\top \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$

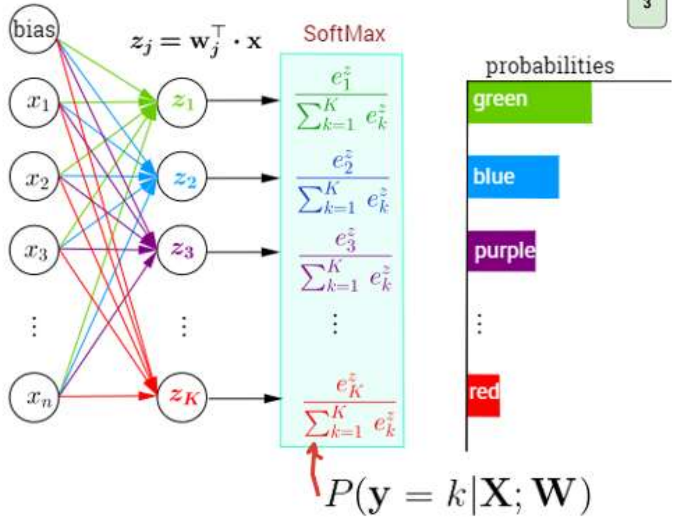


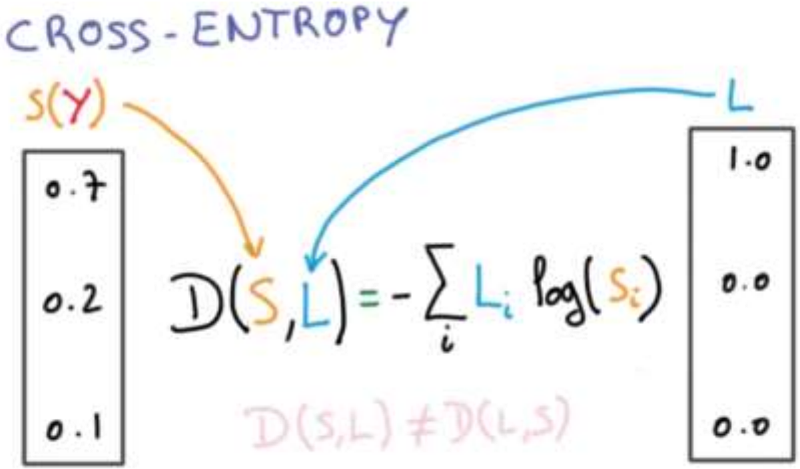
$$P(\mathbf{y} = k | \mathbf{X}; \mathbf{W})$$

Classification Loss

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







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
```

Average Classification Error ?

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
```

Average Classification 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
```

Average Classification Error ?

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

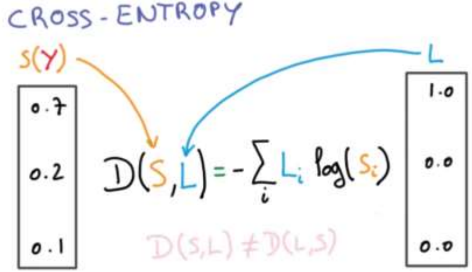
```
computed          | targets              | correct?
---------------------------------------------------
0.1   0.2   0.7   | 0   0   1            | yes
0.1   0.7   0.2   | 0   1   0            | yes
0.3   0.4   0.3   | 1   0   0            | no
```

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.

Resources

- Videos

- Example from lecture: https://www.youtube.com/watch?v=bxe2T-V8XRs&list=PLiaHhY2iBX9hdHaRr6b7XevZtgZRa1PoU

- 3Blue1Brown: https://www.youtube.com/watch?v=aircAruvnKk&list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

- StatQuest: https://www.youtube.com/watch?v=zxagGtF9MeU&list=PLblh5JKOoLUIxGDQs4LFFD--41Vzf-ME1

- NN zero to hero: https://www.youtube.com/watch?v=VMj-3S1tku0&list=PLAqhIrjkxbuWI23v9cThsA9GvCAUhRvKZ

- BP in terms of computational graph (cs221n): https://www.youtube.com/watch?v=i94OvYb6noo

- https://theaisummer.com/weights-and-biases-tutorial/