COMP [56]630– Machine Learning

Lecture 9 – Neural Networks

Perceptron Algorithm

The Perceptron

- Consider modifying the logistic regression method to “force” it to output values that are either 0 or 1 or exactly. To do so, it seems natural to change the definition of g to be the threshold function:

$$g(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{if } z < 0 \end{cases}$$

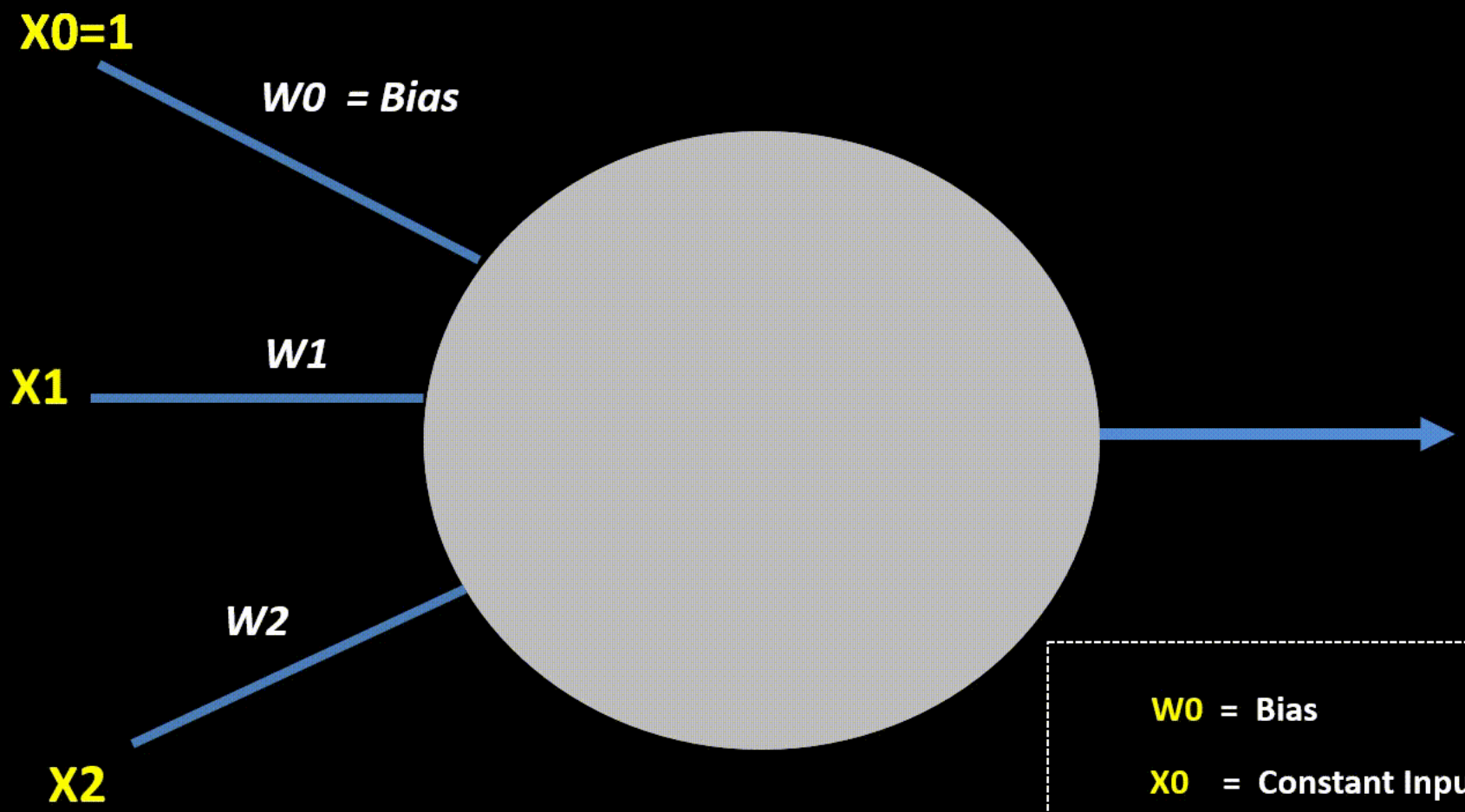
- If we then let h(x) = g(θT x) as before but using this modified definition of g, and if we use the update rule

$$\theta_j := \theta_j + \alpha \left(y^{(i)} - h_\theta(x^{(i)}) \right) x_j^{(i)}$$

Perceptron

- In the 1960s, this “perceptron” was argued to be a rough model for how individual neurons in the brain work.
- Note however that even though the perceptron may be cosmetically similar to the other algorithms we talked about, it is actually a very different type of algorithm than logistic regression.
  - The hypothesis in logistic regression provides a measure of uncertainty in the occurrence of a binary outcome based on a linear model.
  - The output from a step function can of course not be interpreted as any kind of probability.
  - Since a step function is not differentiable, it is not possible to train a perceptron using the same algorithms that are used for logistic regression.

Artificial Neuron



W0 = Bias

X0 = Constant Input 1 for Bias

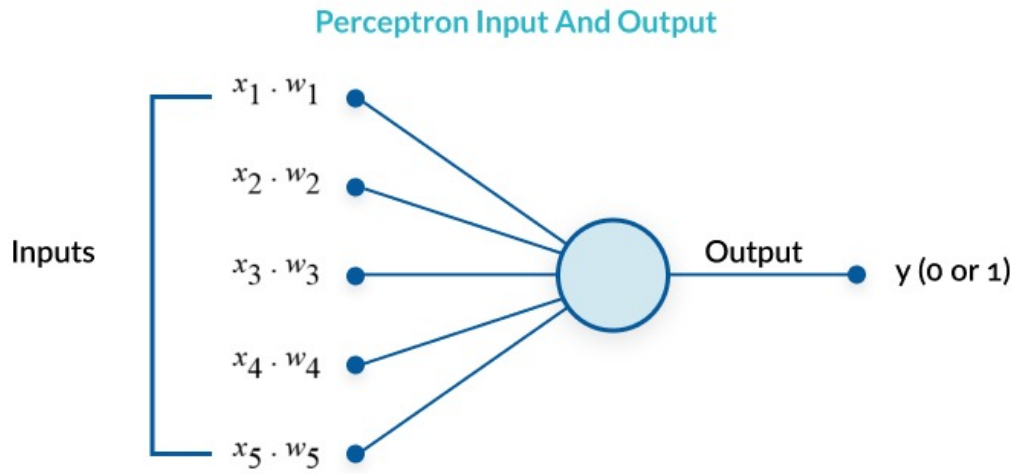
X1,X2 = Attribute Inputs

W1,W2 = Weights of Inputs X1,X2

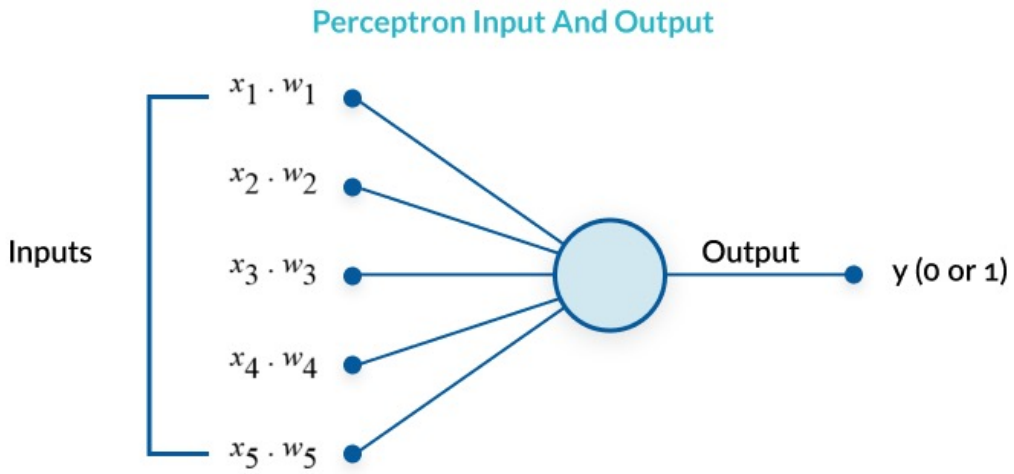
machinelearningknowledge.ai

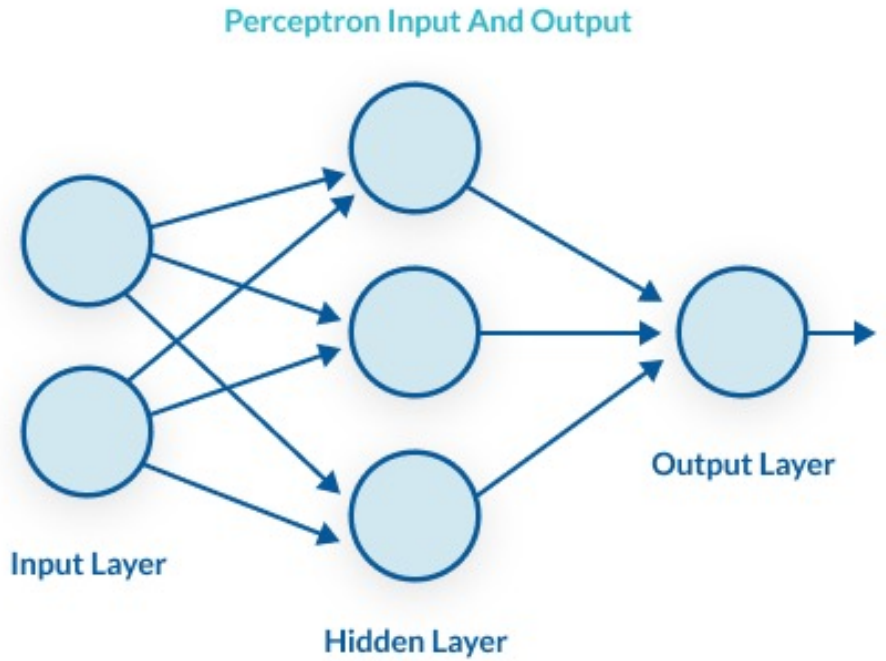
Multi-layer perceptrion

Perceptron vs Multilayer Perceptron (MLP)

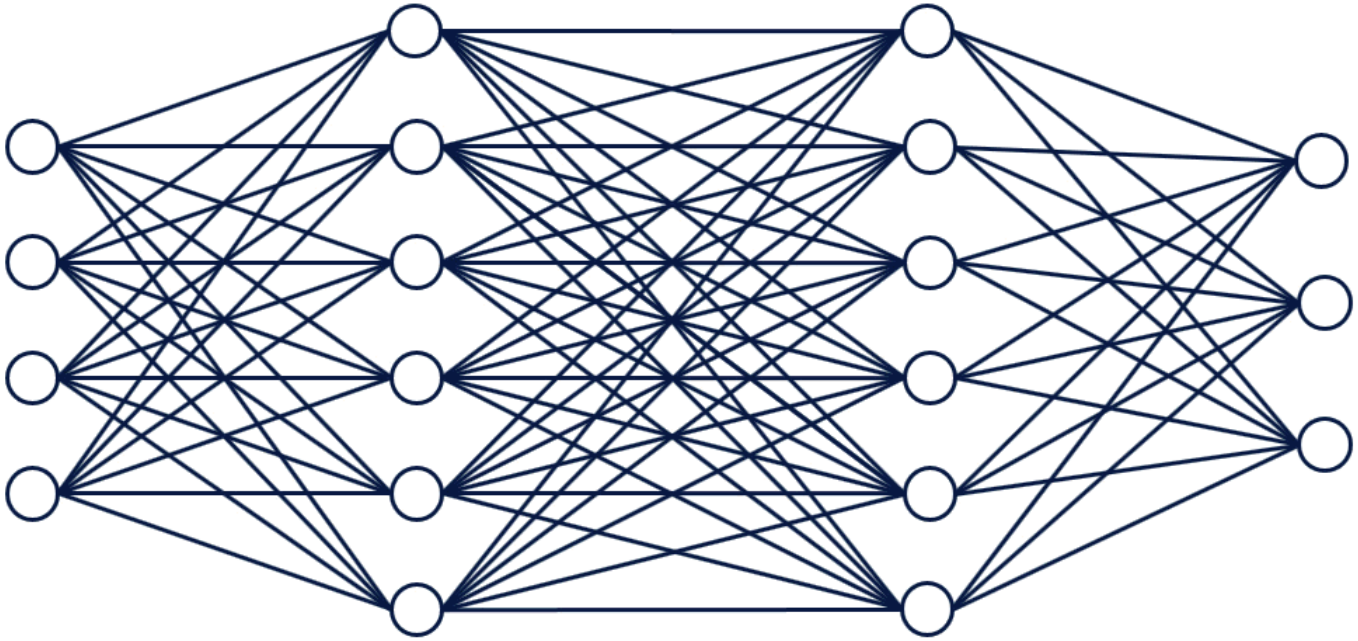


Perceptron vs Multilayer Perceptron (MLP)





Why do we need more layers?



Feed forward Neural Networks

A neural network can also be represented similar to linear models but basis functions are generalized

$$y(\boldsymbol{x}, \boldsymbol{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\boldsymbol{x})\right)$$

activation function

For regression: identity function

For classification: a non-linear function

Coefficients w_j adjusted during training

There can be several activation functions

Basis functions

$\phi_j(\boldsymbol{x})$ a nonlinear function of a linear combination of D inputs

its parameters are adjusted during training

Activation Functions

- Each activation a_j is transformed using differentiable nonlinear activation functions $z_j = h(a_j)$
- The z_j correspond to outputs of basis functions $\varphi_j(x)$
  - or first layer of network or hidden units
- Nonlinear functions h
- Three examples of activation functions:

1. Logistic sigmoid

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

2. Hyperbolic tangent

$$\tanh(a) = \frac{e^{a} - e^{-a}}{e^{a} + e^{-a}}$$

3. Rectified Linear unit

$$f(x) = \max(0, x)$$

Activation Function

- Determined by the nature of the data and the assumed distribution of the target variables
- For standard regression problems the activation function is the identity function so that $y_k=a_k$
- For multiple binary classification problems, each output unit activation is transformed using a logistic sigmoid function so that $y_k=\sigma(a_k)$
- For multiclass problems, a softmax activation function of the form:

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

Visualization of activation functions

Forward Pass

- Output of a layer l is given by

$$z^{[\ell]} = W^{[\ell]} a^{[\ell-1]} + b^{[\ell]}$$

- Where W^l is the weights of the layer, b^l is the bias and a^l is the activation

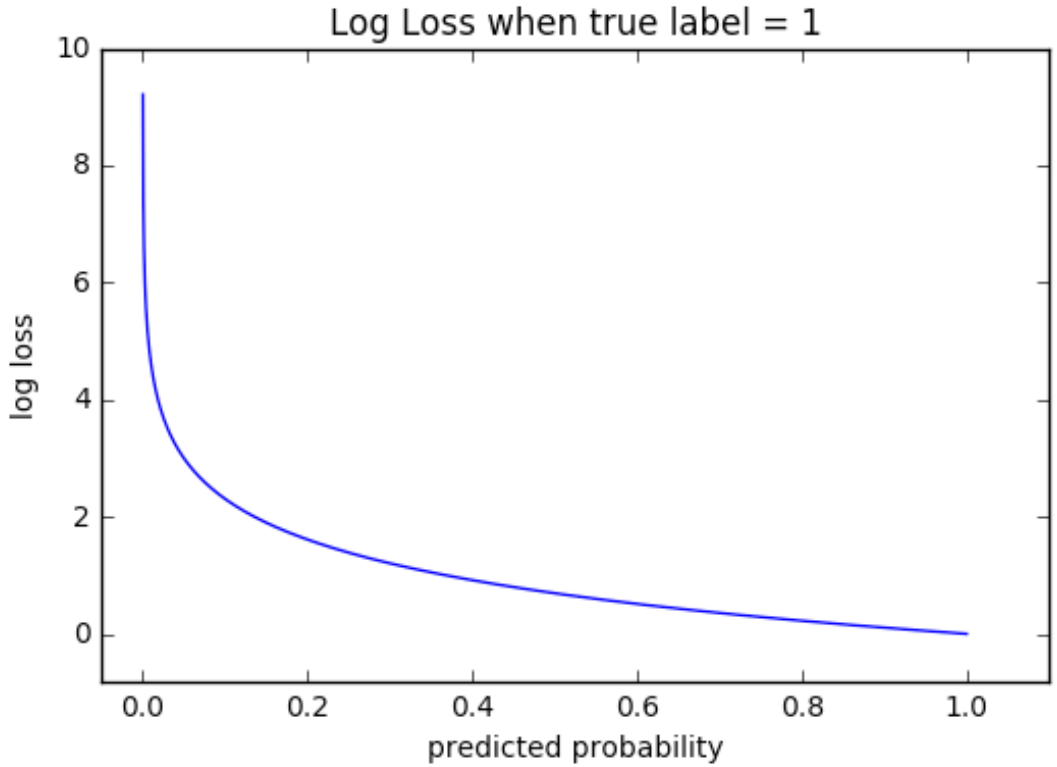
$$a^{[\ell]} = g^{[\ell]}(z^{[\ell]})$$

Loss Function: Log loss

- Measures the performance of a classification model whose output is a probability value between 0 and 1.

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

- Cross-entropy loss increases as the predicted probability diverges from the actual label.



How to train this network?

- Gradient Descent + Backprop!
- What is Backprop?
- Backprop or Backpropagation is a way to train multilayer neural networks using gradients.

Backpropagation

- Before training the neural network, select an initial value for parameters.
  - Common practice: randomly initialize the parameters to small values (e.g., normally distributed around zero; N(0; 0:1))
- Next step: update the parameters.
- After a single forward pass through the neural network, the output will be a predicted value
- We can then compute the loss L, in our case the log loss

$$\mathcal{L}(\hat{y}, y) = -\Big[(1-y)\log(1-\hat{y}) + y\log\hat{y}\Big]$$

Defining Variables

Let X be the input features . [n x f]
y be the output (target) labels [n x 1]

Define the weights [parameters] of the neural network

First layer → W_1 , bias → b_1
Second layer → W_2 , bias → b_2
Third layer → W_3 , bias → b_3

The forward pass

The output of layer 1 is

$$z_1 = w_1 \cdot x + b_1$$
$$a_1 = g(z_1)$$ where "g" is the activation function

The output of layer 2 is

$$z_2 = w_2 \cdot a_1 + b_2$$
$$a_2 = g(z_2)$$

The output of layer 3 is

$$z_3 = w_3 \cdot a_2 + b_3$$
$$a_3 = g(z_3)$$

The o/p of the neural network is

$$\hat{y} = a_3$$

Defining the loss

Task : Binary classification

$\Rightarrow$ loss is log loss or binary cross entropy

$$L = -\left[(1-y)\cdot\log(1-\hat{y}) + y\cdot\log(\hat{y})\right]$$

Learning with gradient descent

To learn the parameters

(1) Provide random values for weights w_1, w_2, w_3 & biases b_1, b_2, b_3

(2) Update the weights using gradient descent by

$$w_i := w_i - \alpha \cdot \frac{dL}{dw_i}$$

$$b_i := b_i - \alpha \cdot \frac{dL}{db_i}$$

where i refers to the ith layer.

(3) repeat until convergence

Finding dL/dW_3

To find $\frac{\partial L}{\partial w_3}$ to update the third layer.

$$\frac{\partial L}{\partial w_3} = \frac{\partial}{\partial w_3}\left[-\left[(1-y)\cdot\log(1-\hat{y}) + y\cdot\log(\hat{y})\right]\right]$$

$$= -\left[(1-y)\cdot\frac{\partial}{\partial w_3}\left(\log(1-\hat{y})\right) + y\cdot\frac{\partial}{\partial w_3}\left(\log(\hat{y})\right)\right]$$

Finding dL/dW_3

Remember: $\frac{d}{dz}(\log(z)) = \frac{1}{z}$

Finding dL/dW_3

$$\therefore \frac{\partial L}{\partial w_3} = -\left[\frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial}{\partial w_3}(1-\hat{y}) + \frac{y}{\hat{y}} \frac{\partial}{\partial w_3}(\hat{y})\right]$$

$$= -\left[\frac{-(1-y)}{(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3} + \frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3}\right]$$

$$= \frac{(1-y)}{(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3} - \frac{y}{\hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \left[\frac{(1-y)}{(1-\hat{y})} - \frac{y}{\hat{y}}\right] \cdot \frac{\partial \hat{y}}{\partial w_3}$$

Finding dL/dW_3

$$\Rightarrow \frac{\partial L}{\partial w_3} = \frac{\hat{y}(1-\hat{y}) - y(1-\hat{y})}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y\hat{y} - y + y\hat{y}}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

$$= \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot \frac{\partial \hat{y}}{\partial w_3}$$

Finding dL/dW_3

We know that $\hat{y} = a_3 = g(z_3)$

Finding dL/dW_3

if $g(z)$ is a sigmoid function,

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

$$\therefore \frac{\partial \hat{y}}{\partial w_3} = \frac{\partial a_3}{\partial w_3} = \frac{\partial \cdot g(z_3)}{\partial w_3}$$

$$= g(z_3) \cdot (1 - g(z_3)) \cdot \frac{\partial z_3}{\partial w_3}$$

$$= a_3 (1 - a_3) \cdot \frac{\partial}{\partial w_3} [w_3 a_2 + b_3]$$

$$= a_3 (1 - a_3) \cdot a_2$$

$$= \hat{y} (1 - \hat{y}) \cdot a_2$$

Finding dL/dW_3

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.

Finding dL/dW_3

$$\therefore \frac{\partial L}{\partial w_3} = \frac{\hat{y} - y}{\hat{y}(1-\hat{y})} \cdot [\hat{y} \cdot (1-\hat{y}) \cdot a_2]$$

$$\therefore \boxed{\frac{\partial L}{\partial w_3} = (\hat{y} - y) \cdot a_2}$$

$$\boxed{\frac{\partial L}{\partial w_3} = (a_3 - y) \cdot a_2^T}$$

when providing a vectorized solution.

Similarly,

$$\boxed{\frac{\partial L}{\partial b_3} = a_3 - y}$$

Finding dL/dW_2

- We need to use the chain rule from calculus.
- Why?
- There is no direct relationship between the weights W_2 and the loss L.
- You cannot differentiate a variable by another if there is no direct relationship
  - Else it would be 0
  - Not true if the variable/function is composite

Finding dL/dW_2

We know that Loss is dependent on $\hat{y} = a_3$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{?} \cdot \frac{?}{\partial w_2}$$

we know that $a_3 = g(z_3)$

$\Rightarrow a_3$ is dependent on z_3

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{?} \cdot \frac{?}{\partial w_2}$$

Finding dL/dW_2

We know that $z_3 = w_3 a_2 + b_3$

$\Rightarrow z_3$ is dependent on a_2

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{?} \cdot \frac{?}{\partial w_2}$$

Finding dL/dW_2

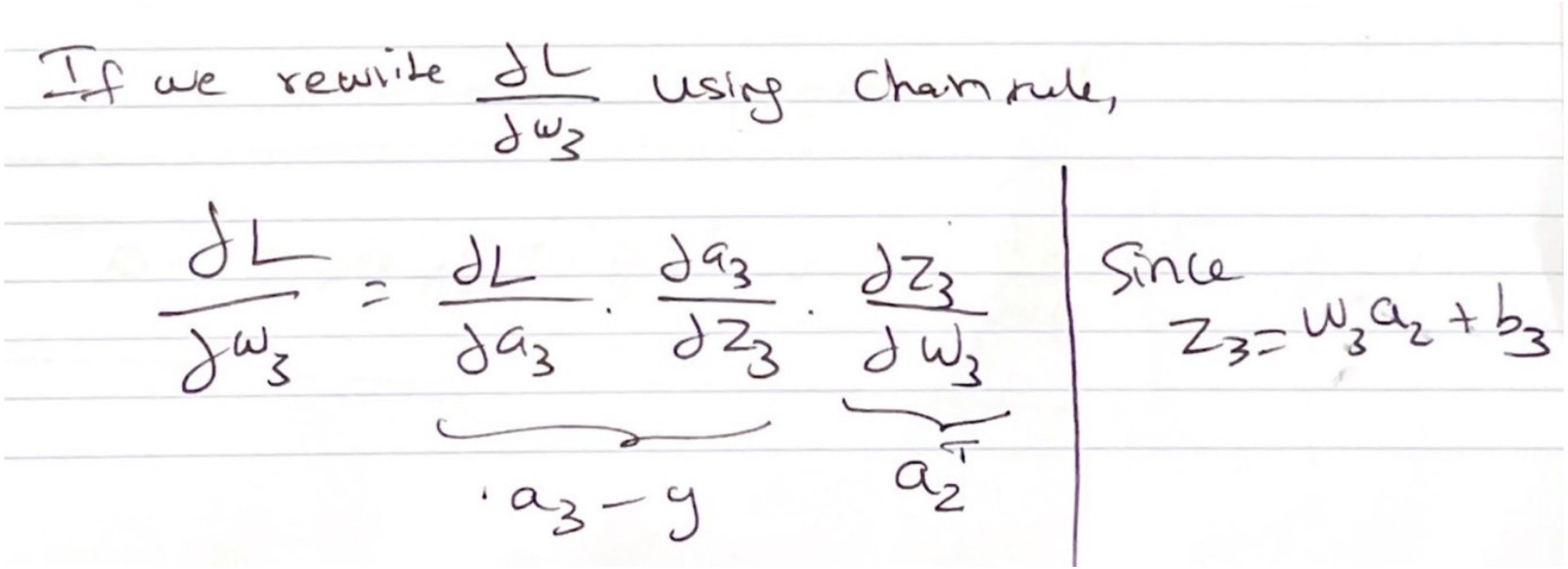
But $a_2 = g(z_2)$

$$\Rightarrow \frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{?} \cdot \frac{?}{\partial w_2}$$

But $z_2 = w_2 \cdot a_1 + b_2$

$$\therefore \boxed{\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial w_2}}$$

Finding dL/dW_2



Finding dL/dW_2

$$\Rightarrow \frac{dL}{dW_2} = (a_3 - y) \cdot \frac{dz_3}{da_2} \cdot \frac{da_2}{dz_2} \cdot \frac{dz_2}{dW_2}$$

If $z_3 = w_3 \cdot a_2 + b_3$

Then $\frac{dz_3}{da_2} = w_3$

if $a_2 = g(z_2)$

then $\frac{da_2}{d(z_2)} = g'(z_2)$

If $z_2 = w_2 \cdot a_1 + b_2$

then $\frac{dz_2}{dw_2} = a_1$

Finding dL/dW_2

$$\Rightarrow \boxed{\frac{\partial L}{\partial w_2} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot a_1}$$

Re-arranging for vectorization,

$$\boxed{\frac{\partial L}{\partial w_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y) \cdot a_1^T}$$

$$\frac{\partial L}{\partial b_2} = w_3^T \cdot g'(z_2) \cdot (a_3 - y)$$

Finding dL/dW_1

Use the chain rule!

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial a_3} \cdot \frac{\partial a_3}{\partial z_3} \cdot \frac{\partial z_3}{\partial a_2} \cdot \frac{\partial a_2}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1} \cdot \frac{\partial z_1}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1) \cdot x$$

Finally

$$\frac{\partial L}{\partial b_1} = (a_3 - y) \cdot w_3 \cdot g'(z_2) \cdot w_2 \cdot g'(z_1)$$

NumPy Demo

Regularization

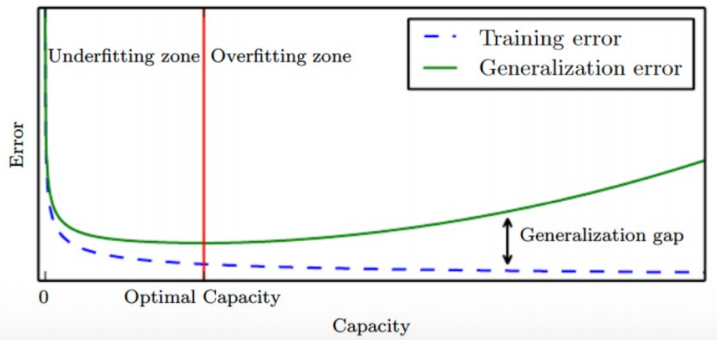
What is regularization?

Generalization error

- Performance on inputs not previously seen
  - Also called as *Test error*

Regularization is:

- "any modification to a learning algorithm to reduce its *generalization error* but not its *training error*"
- Reduce generalization error even at the expense of increasing training error
  - E.g., Limiting model capacity is a regularization method



Goals of Regularization

Some goals of regularization

1. Encode prior knowledge
2. Express preference for simpler model
3. Need to make underdetermined problem determined

Why regularization?

Generalization

- Prevent over-fitting

Occam's razor

Bayesian point of view

- Regularization corresponds to prior distributions on model parameters

Limiting Number of Neurons

No. of input/output units determined by dimensions

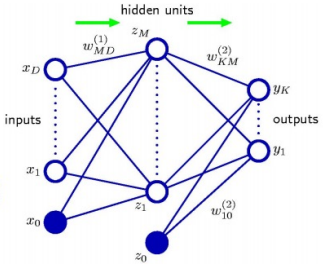
Number of hidden units M is a free parameter

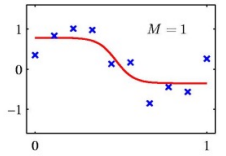
- Adjusted to get best predictive performance

Possible approach is to get maximum likelihood estimate of M for balance between under-fitting and over-fitting

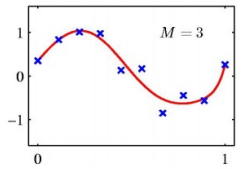
Limiting Number of Neurons

Sinusoidal Regression Prob

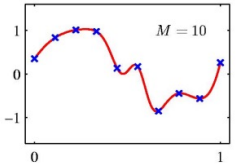




Two layer network trained on 10 data points



$M = 1, 3$ and 10 hidden units



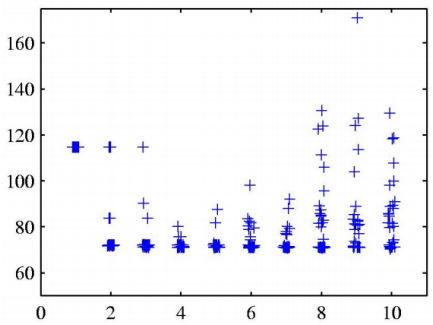
Minimizing sum-of-squared error function
Using conjugate gradient descent

Generalization error is not a simple function of M due to presence of local minima in error function

Using Validation Set to fix no. of hidden units

Plot a graph choosing random starts and different numbers of hidden units M and choose the specific solution having smallest generalization error

Sum of squares *Test* error



No. of hidden units, M

- 30 random starts for each M
  30 points in each column of graph
- Overall best *validation* set performance happened at $M=8$

There are other ways to control the complexity of a neural network in order to avoid over-fitting

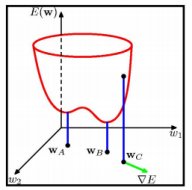
Alternative approach is to choose a relatively large value of M and then control complexity by adding a regularization term

Parameter Norm Penalty

Generalization error not a simple function of M

- Due to presence of local minima
- Need to control capacity to avoid over-fitting
  - Alternatively choose large M and control complexity by addition of regularization term

Simplest regularizer is *weight decay*



$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \frac{\lambda}{2}\boldsymbol{w}^{\mathrm{T}}\boldsymbol{w}$$

- Effective model complexity determined by choice of λ
- Regularizer is equivalent to a Gaussian prior over $\boldsymbol{w}$

In a 2-layer neural network

$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

Weight Decay

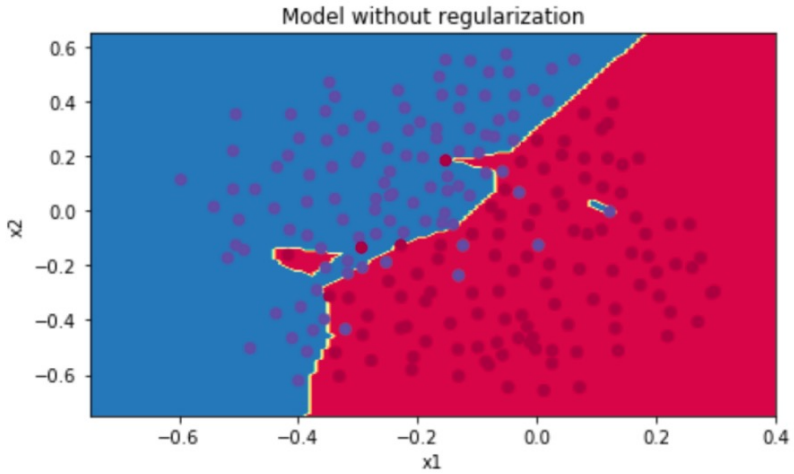
The name weight decay is due to the following

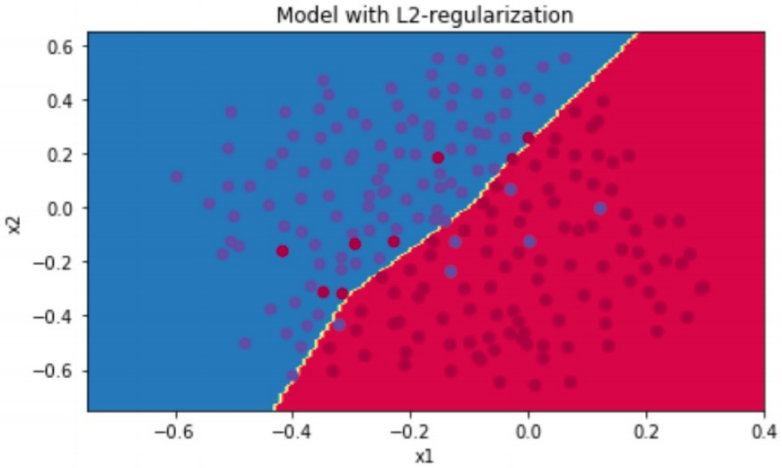
$$w_{t+1} = w_t - \alpha \nabla_w J - \lambda w_t$$

To prevent overfitting, every time we update a weight $\boldsymbol{w}$ with the gradient ∇J in respect to $\boldsymbol{w}$, we also subtract from it $\boldsymbol{\lambda w}$.

This gives the weights a tendency to decay towards zero, hence the name.

Effect of Regularization





$$J_{regularized} = \underbrace{-\frac{1}{m}\sum_{i=1}^{m}\left(y^{(i)}\log\left(a^{[L](i)}\right) + (1-y^{(i)})\log\left(1-a^{[L](i)}\right)\right)}_{\text{cross-entropy cost}} + \underbrace{\frac{1}{m}\frac{\lambda}{2}\sum_{l}\sum_{k}\sum_{j}W_{k,j}^{[l]2}}_{\text{L2 regularization cost}}$$

https://medium.com/@shay.palachy/understanding-the-scaling-of-l%C2%B2-regularization-in-the-context-of-neural-networks-e3d25f8b50db?fbclid=IwAR17rKfnM2JnkWBOZFvQg6ftl0k52diyEra_66UCwI18Gtf7nFd6l3l4VMY