Machine Learning & Deep Learning

Week-13

Instructor: *Engr. Najam Aziz*

National Center for Big Data & Cloud Computing (NCBC), UET Peshawar

Types of Artificial Neural Network

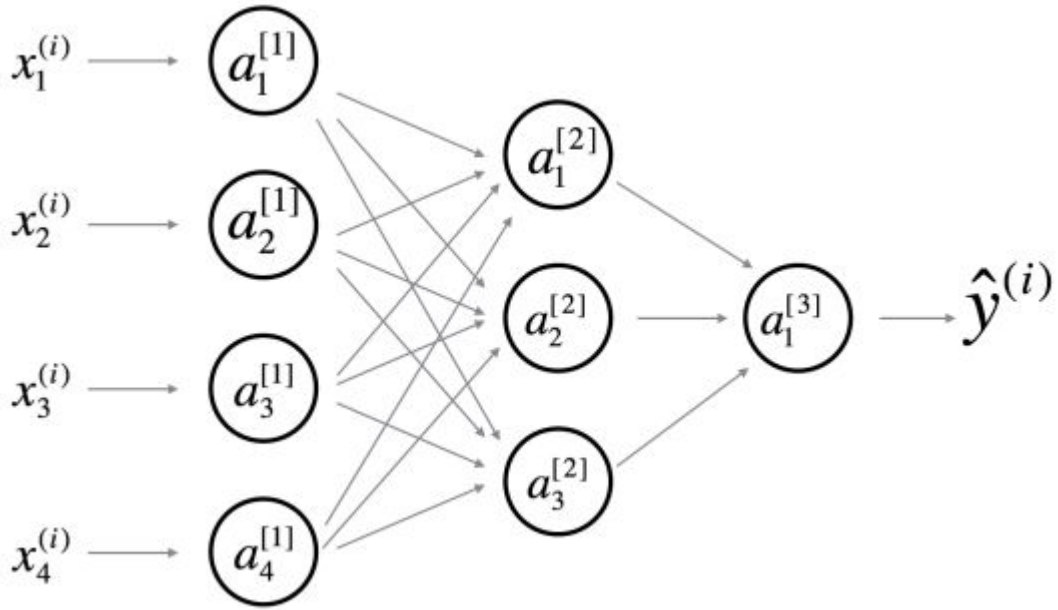
Based on the information flow, there are basically two types of ANN

- Feedforward Neural Network (FFNN)
  - Multilayer Perceptron (MLP)
  - Convolutional Neural Network (CNN)

- Feedback Neural Network (FBNN)
  - Recurrent Neural Network (RNN)

Artificial Neural Network - MLP

- Multiple interconnected Neurons(perceptrons) processing information collectively



Standard Representation of Neural Network

What Neural Network Does? (During training)

Takes input, multiply it with randomly initialized ***weights,*** *sum all*, add ***bias*** and pass through an ***activation function*** (for all neurons) which gives some output. In next layer, multiply the previous layer output with current layer weights, add bias and pass through an activation function which gives output.

Next, to find ***error/loss - Cost,*** target/desired output is subtracted from predicted output. What is our **Aim: To minimize error/loss**

For error minimization or decreasing loss - ***backpropagate*** this error/loss through NN. and adjust(***increase/decrease***) weights.

Again network gives some output, again error is calculated, backpagated and weights are adjusted.
The process goes on till no or minimum error/loss **(**Repeat for loop till convergence. Meaning till the weights ,Wij new approximately or equal to Wij old.)

What Neural Network Does? (During training)

Training Process: Let Assume we have D={ xi's , yi's }
Next need to initialize weights (Wij) randomly
For each xi's in D dataset

1) Pass xi forward through the network. → Forward Prop

2) Compute the loss.[yi -y^i]

3) Compute all the derivative(Gradient) using chain rule and memoization

4) Update the weight (Wij) from end of the network to the start. → BackProp

Repeat for loop till convergence. Meaning till the weights ,Wij new approximately or equal to Wij old.

"Note : The weights are modified/updated using a function called Optimization Function/Optimizers"

The process

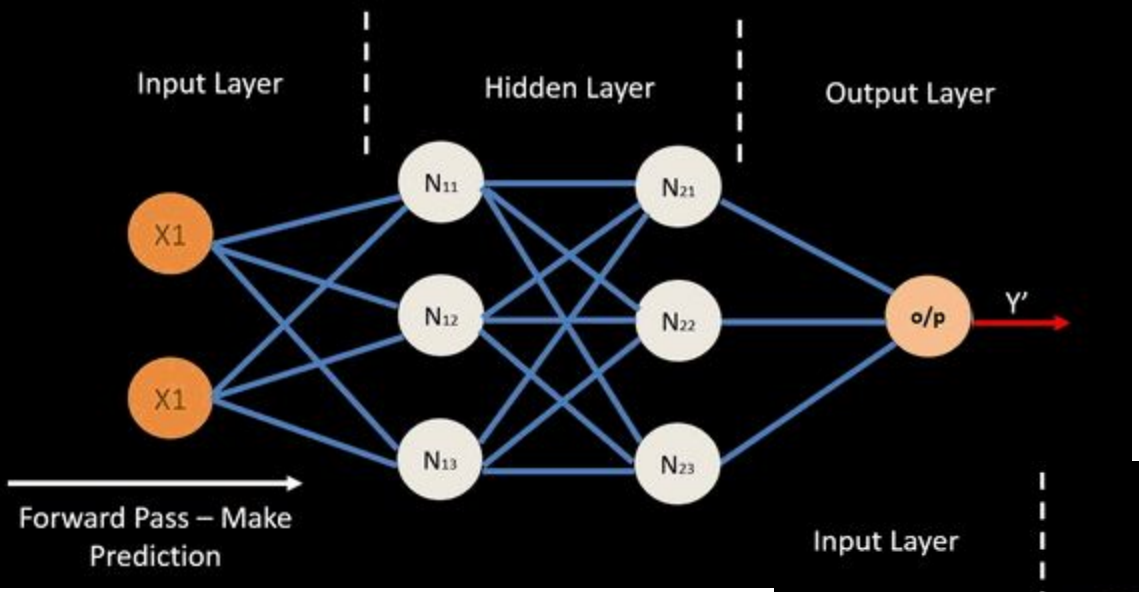
Forward Pass

1. Takes input
2. Multiply input with weights, sum all and add bias
3. Pass through an Activation function and gives output
4. Calculate error: Predicted - target output

   For all layers.

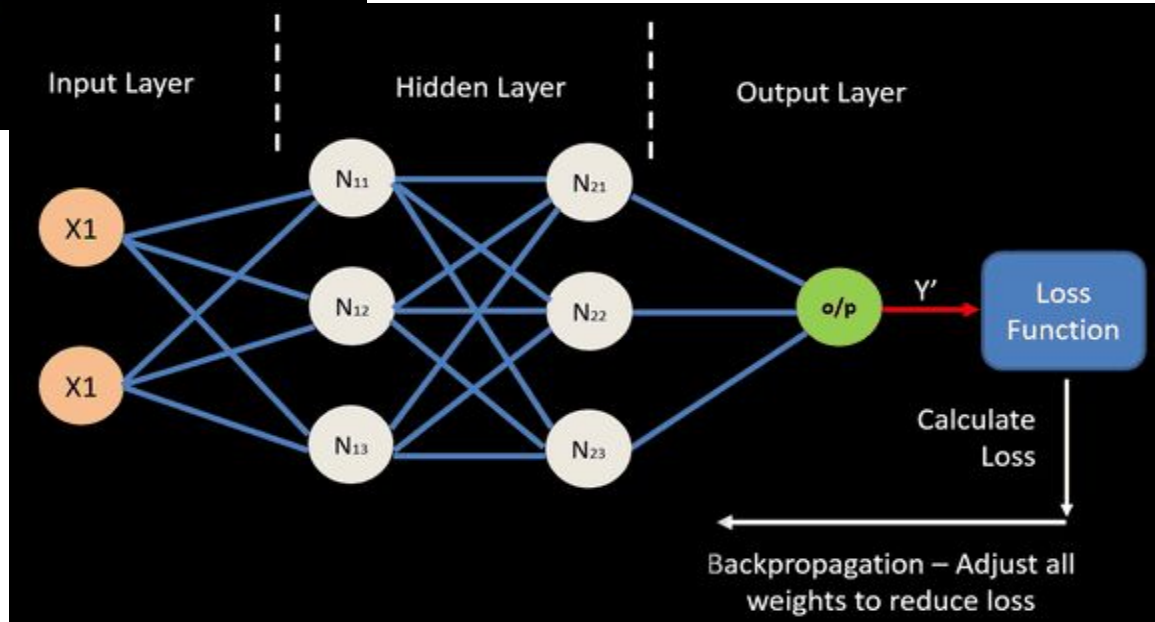
Backward Pass

1. Backpropagate the error
2. Find the Direction and Magnitude of the change (using Gradient Descent) and adjust the weights accordingly.



Forward Pass

Backward Pass



Building simple ANN in Python

```
#Import required Libraries
import numpy as np
import pandas as pd
#Mount Drive
from google.colab import drive
drive.mount('/content/drive')
#Read Data
df =pd.read_csv('/content/drive/MyDrive/3S2_1PS_1S2_csv.csv')
#Data Shape
df.head()
```

	B1	B2	B3	B4	Label
0	192	217	273	3317	0
1	211	223	311	3047	0
2	218	244	363	2855	0
3	281	269	334	2881	0
4	268	283	329	2855	0

```
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
#Splitting the Data into Train & Test data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=42)
#Training Data shape
X_train.shape #(114944, 4)
Y_train.shape #(114944,)
```

```
#Import Libraies and functions
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.layers import Dense
from tensorflow.keras.models import Sequential
#Model Building
model = Sequential()
model.add(Dense(7, input_shape=(4,) , activation='relu')) #1st Hidden Layer with input shape
model.add(Dense(7, activation='relu'))#2nd Hidden layer
model.add(Dense(10, activation='softmax')) #Output Layer
#Compile Model
model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
To train a model with fit(), you need to specify a loss function, an optimizer, and
optionally, some metrics to monitor. You pass these to the model as arguments to
the compile() method:
#Fitting/Training the Model
history = model.fit( X_train, Y_train, epochs=100, batch_size=32, validation_data = (X_test,
Y_test))
#Evaluate Model
model.evaluate(X_test, Y_test, verbose = 0)  #Evaluating the model on Test data
```

Parameters, Function and Algorithm used

- Input
- Weights + Bias
- Activation Function
- Error/Loss - Cost Function
- Gradient Descent (Optimization technique/algorithm/rule)
- Backpropagation

What are Input, Weights, and Bias

Input: X_i

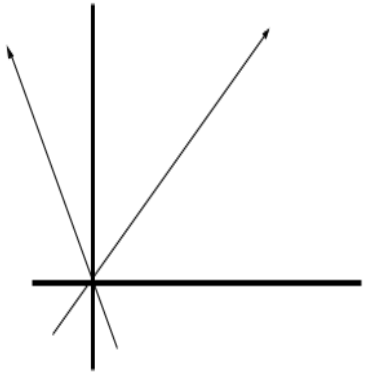
- Features of the data sample we want to predict

Weights: W_{ij}

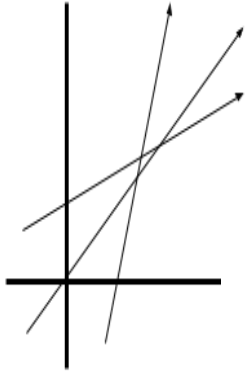
- Represent the importance of the input(features)
- Initialized randomly, and we aim for finding the optimal weights, for which error is minimum.

Bias: b_j

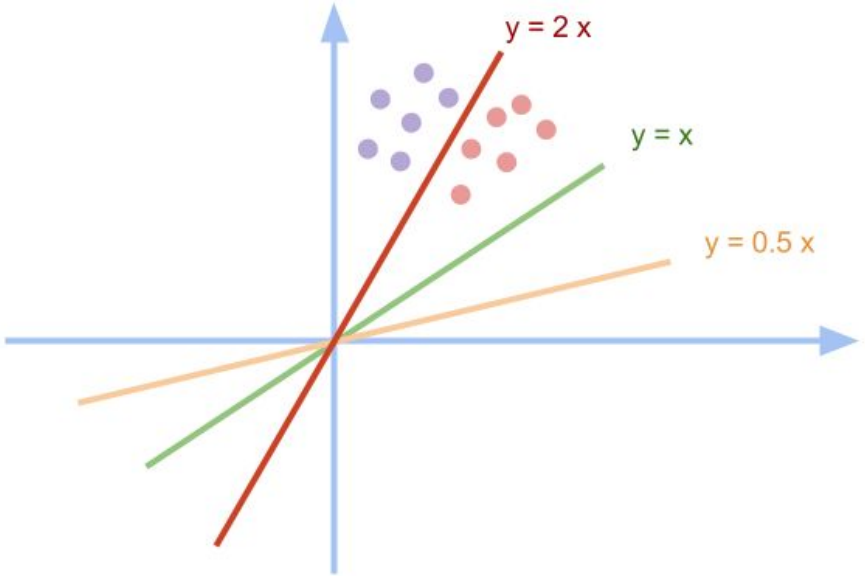
- Represent the Bias(favor) - Favor could be due to network, in data or any other thing.
- Practically, Allows to shift the activation function by adding a constant (i.e. the given bias) to the input.



Y = mx



Y = mx



Activation Function

Activation function

- Having lot of information. It is important to divide it into useful and non-useful. Coz not all the information is equally useful. Some of it is just noise. This is where activation functions come into picture. The activation functions help the network use the important information and suppress the irrelevant data points.
- Let Z be summation of ((Wi*Xi) + b)
  The value of Z can be anything ranging from -inf to +inf. Meaning it has lot of information ,now neuron must know to distinguish between the “useful” and “not -so-useful” information.To build this sense into our network we add ‘activation function (f)’— Which will decide whether the information passed is useful or not based on the result it get fired.

- An activation function transforms the shape/representation of the data going into it.
- Decide whether the Neuron should be activated or not.

➔ The choice of, which function(s) are best for a specific problem using a particular neural architecture is still under a lot of discussions. However, these representations are essential for making high-dimensional data linearly separable, which is one of the many uses of a neural network.

Types of Activations Functions

Linear function: A neural networks with a linear activation function is simply a linear regression model.

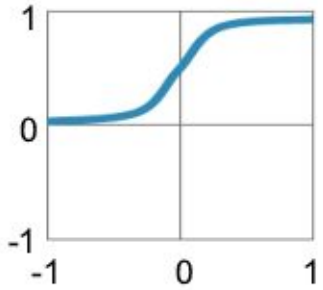
Binary Step function: Binary step function are popular known as “ Threshold function”.

Non-Linear function: Create complex mappings between the network’s inputs and outputs, which are essential for learning and modeling complex data, such as images, video, audio, and data sets which are non-linear or have high dimensionality.)

- **Logistic/Sigmoid, Tanh, Relu, Leaky Relu, Parametric-Relu(Prelu), Swish, Softmax, Softplus**

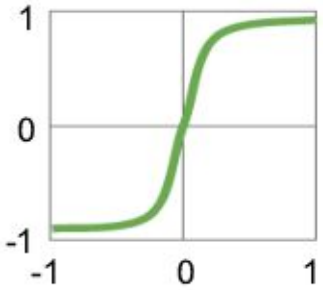
Traditional Non-Linear Activation Functions

Sigmoid



$y=1/(1+e^{-x})$

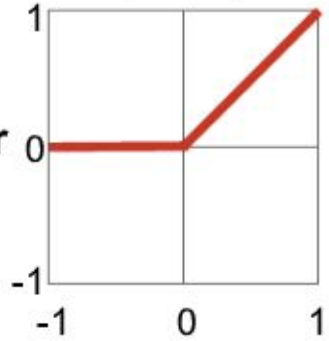
Hyperbolic Tangent



$y=(e^{x}-e^{-x})/(e^{x}+e^{-x})$

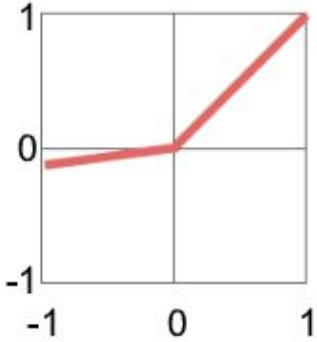
Modern Non-Linear Activation Functions

Rectified Linear Unit (ReLU)



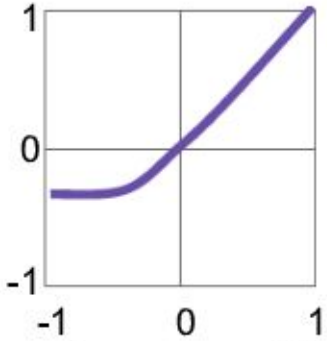
$y=\max(0,x)$

Leaky ReLU



$y=\max(\alpha x,x)$

Exponential LU



$$y=\begin{cases} x, & x\geq 0 \\ \alpha(e^{x}-1), & x<0 \end{cases}$$

α = small const. (e.g. 0.1)

Which one is better to use ? How to choose a right one?

- No hard and fast rule
- Each have its own pros and Cons
- All the good and bad will be decided based on the trail.

But based on the properties of the problem we might able to make a better choice for easy and quicker convergence of the network.

- Sigmoid functions and their combinations generally work better in the case of classification problems
- Sigmoids and tanh functions are sometimes avoided due to the **vanishing gradient** problem
- ReLU activation function is widely used in modern era.
- In case of dead neurons in our networks due to ReLu then leaky ReLU function is the best choice
- ReLU function should only be used in the hidden layers

“As a rule of thumb, one can begin with using ReLU function and then move over to other activation functions in case ReLU doesn’t provide with optimum results”

Error/Loss and Cost Function

In ML, we need to estimate how good/bad models are performing. Put simply, we need to measure, how wrong the model is in terms of its ability to estimate the relationship between X and y. This is typically expressed as a difference or distance between the predicted value and the actual value. And loss/cost function does this estimation.

Error/Loss and Cost Function

- **Error/Loss:** The Difference b/w Predicted & Actual output
- ‘Loss’ helps us to understand how much the predicted value differ from actual value
- **Loss/Cost Function** Function that calculate the loss or cost is called as “Loss/Cost function”
- A loss function/error function is for a single training example/input. A cost function, on the other hand, is the average loss over the entire training (or mini-batch) dataset.
- The loss/error is computed for a single training example. If we have ‘m’ number of examples then the average of the loss function of the entire training set is called ‘Cost function’.
  Cost function (J) = 1/m (Sum of Loss error for ‘m’ examples)
- Cost function and Loss function are synonymous and used interchangeably but they are “different”.
- “Loss functions are helpful to train a neural network”
- The optimization strategies aim at “minimizing the cost function”.

Error/Loss or Cost Function

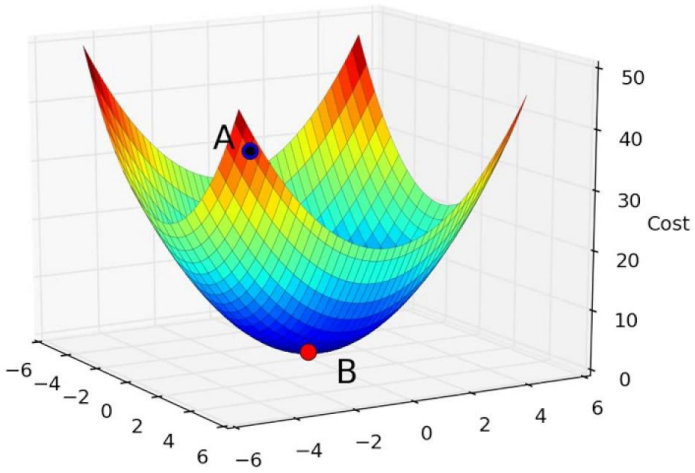
It is a function that measures the performance of a Machine Learning model for given data. Loss/Cost Function quantifies the error between predicted values and expected values and presents it in the form of a single real number.
The purpose of Cost Function is to be either:

Minimized - then returned value is usually called cost, loss or error. The goal is to find the values of model parameters for which Cost Function return as small number as possible.

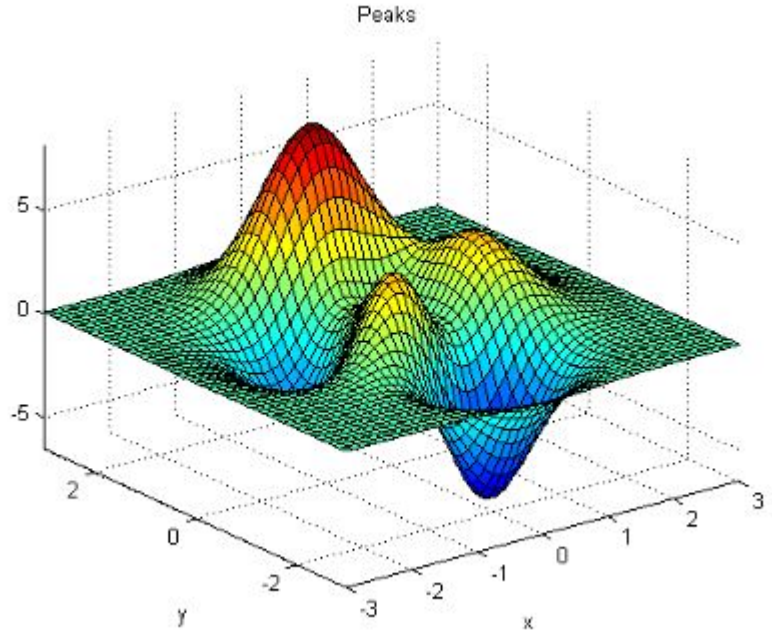
Maximized - then the value it yields is named a reward. The goal is to find values of model parameters for which returned number is as large as possible.

In simple words, the Loss/Cost is used to calculate the gradients. And gradients are used to update the weights of the Neural Net. This is how a Neural Net is trained.

Convex vs Non-Convex Cost function



Convex (single global minima



Non-Convex (Have multiple minima and maxima

Types of Loss/Cost Function

Various techniques/ways/rules for finding the loss i.e the difference between predicted and actual values. Depending on the problem, Cost Function can be formed in many different ways.

1. **Regression Loss Functions**
- Mean Squared Error
- Mean Squared Logarithmic Error Loss
- Mean Absolute Error Loss

2. **Binary Classification Loss Functions**
- Binary Cross-Entropy
- Hinge Loss
- Squared Hinge Loss

3. **Multi-Class Classification Loss Functions**
- Multi-Class Cross-Entropy Loss
- Sparse Multiclass Cross-Entropy Loss
- Kullback Leibler Divergence Loss

Tailoring Cost Function
https://towardsdatascience.com/coding-deep-learning-for-beginners-linear-regression-part-2-cost-function-49545303d29f

Loss/cost as objective function and some use cases

Cost and loss functions are synonymous (some people also call it error function). The more general scenario is to define an objective function first, which you want to optimize. This objective function could be to

- maximize the posterior probabilities (e.g., naive Bayes)

- maximize a fitness function (genetic programming)

- maximize the total reward/value function (reinforcement learning)

- maximize information gain/minimize child node impurities (CART decision tree classification)

- minimize a mean squared error cost (or loss) function (CART, decision tree regression, linear regression, adaptive linear neurons, …

- maximize log-likelihood or minimize cross-entropy loss (or cost) function

- minimize hinge loss (support vector machine)

Final layer activation function and loss function

Problem Type	Output Type	Final Activation Function	Loss Function
Regression	Numerical value	Linear	Mean Squared Error (MSE)
Classification	Binary outcome	Sigmoid	Binary Cross Entropy
Classification	Single label, multiple classes	Softmax	Cross Entropy
Classification	Multiple labels, multiple classes	Sigmoid	Binary Cross Entropy

Problem Type	Last-layer Output Nodes	Hidden-layer activation	Last-layer activation	Loss function
Binary classification	1	RELU (first choice), Tanh (for RNNs)	Sigmoid	Binary Crossentropy
Multi-class, single-label classification	Number of classes		Softmax	Categorical Crossentropy
Multi-class, multi-label classification	Number of classes		Sigmoid (one for each class)	Binary Crossentropy
Regression to arbitrary values	1		None	MSE
Regression to values between 0 and 1	1		Sigmoid	MSE/Binary Crossentropy

Goal & the Problems at face!

Goal: Find a set of weights and biases that minimizes the cost.

Problem: So, we want to minimize the loss/error by adjusting weights.

But the question is, Whether we should **increase or decrease (Direction)** the weights (as we don't know whether the optimal value(weights value that gives minimum error/loss) is greater or less than the initial value) and by **How much (Magnitude)** (coz we also don't know how much it (the optimal value) deviates from the initial value of weights)

Solution: Optimization Algorithms/Techniques help us in knowing this i.e. whether to increase or decrease and by how much.

"Minimising Cost function *- The goal of any Machine Learning model is to minimize the Cost Function." (maximise in Reinforcement learning)*

Optimization Algorithm (Gradient Descent)

Optimization is a type of searching process and you can think of this search as learning. One popular optimization algorithm is called “gradient descent“, where “gradient” refers to the calculation of an error gradient or slope of error and “descent” refers to the moving down along that slope towards some minimum level of error.

In Gradient Descent (GD), we perform the forward pass using ALL the train data before starting the backpropagation pass to adjust the weights. This is called (one epoch).

Gradient Descent

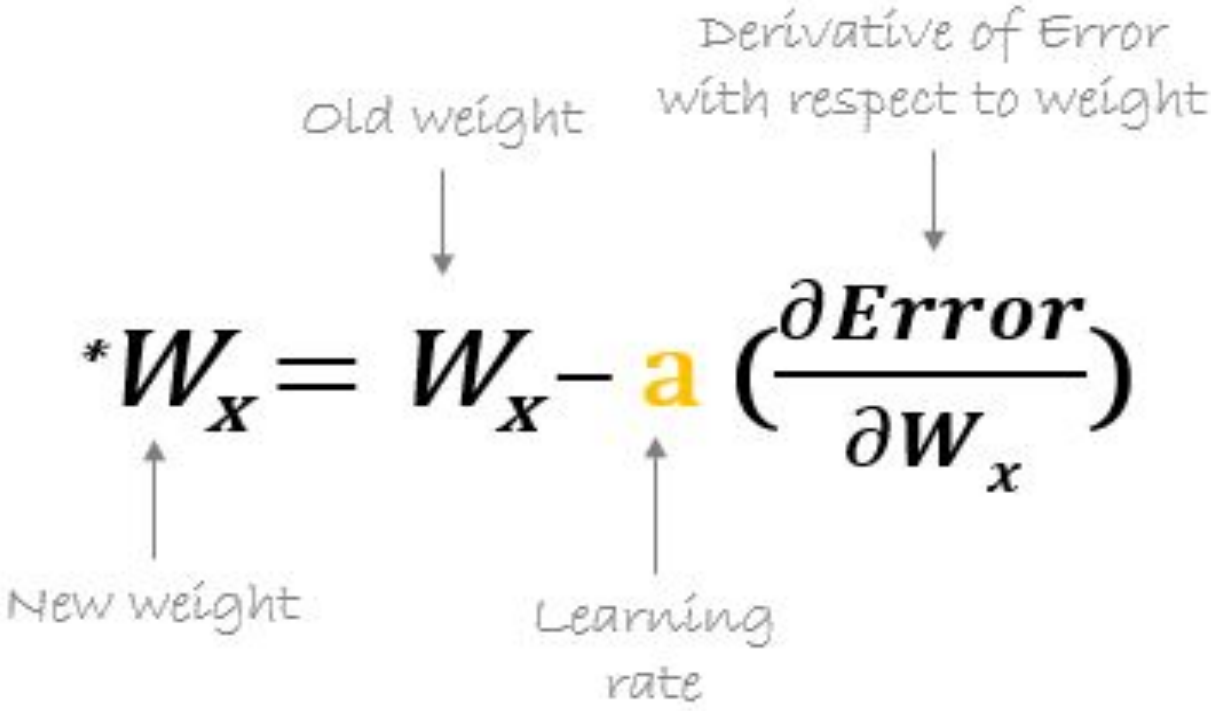
Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent is simply used in machine learning to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.

$$\textbf{Weight}_{(new)} = \textbf{Weight}_{(old)} - \textbf{(Learning rate)(Error Gradient)}$$

- What is Gradient?

"A gradient measures how much the output of a function changes if you change the inputs a little bit." —Lex Fridman (MIT)

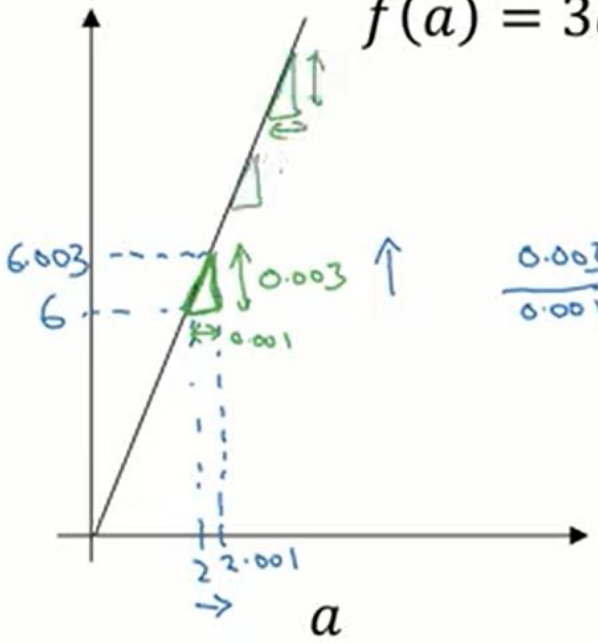
- Magnitude of each component(weights) in the gradient matrix/vector tell us that how sensitive the cost function is to each weight and bias.


Derivative of Error
with respect to weight
Old weight
$^{*}W_x = W_x - a\left(\frac{\partial Error}{\partial W_x}\right)$
New weight
Learning
rate

Error Gradient - The Derivatives

- The derivative is a concept from calculus and refers to the slope of the function at a given point. We need to know the slope so that we know the direction to move the coefficient values in order to get a lower cost on the next iteration.
- The result of the derivative is the slope of your curve at that specific point. A positive number means the curve is increasing, a negative number means the curve is decreasing.

Intuition about derivatives



$\frac{0.003}{0.001}$ $\frac{\text{height}}{\text{width}}$

→ slope (derivative) of $f(a)$ at $a=2$ is 3

→ $a = 2$ $\quad f(a) = 6$

$a = 2.001$ $\quad f(a) = 6.003$

$a = 5$ $\quad f(a) = 15$

$a = 5.001$ $\quad f(a) = 15.003$

slope at $a=5$ is also 3

$$\frac{d\,f(a)}{da} = 3 = \frac{d}{da} f(a)$$

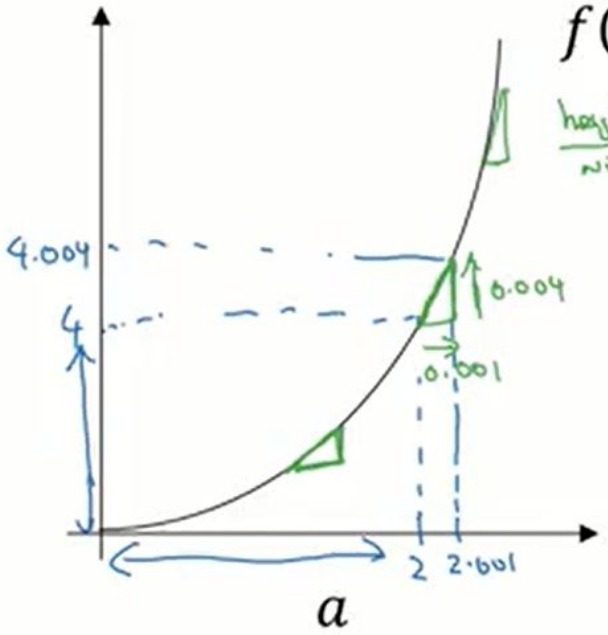
0.001 ←

0.00000001

0.000000000001

Andrew Ng

Intuition about derivatives



$$\frac{d}{da} a^2 = 2a$$

0.001

$(2a) \times 0.001$

0.001 ←

0.00000....01 ←

$a = 2$ $\quad f(a) = 4$

$a = 2.001$ $\quad f(a) \approx 4.004$

$(4.004\boxed{001})$

slope (derivative) of $f(a)$ at $a = 2$ is 4.

$\frac{d}{da} f(a) = 4$ when $a = 2$.

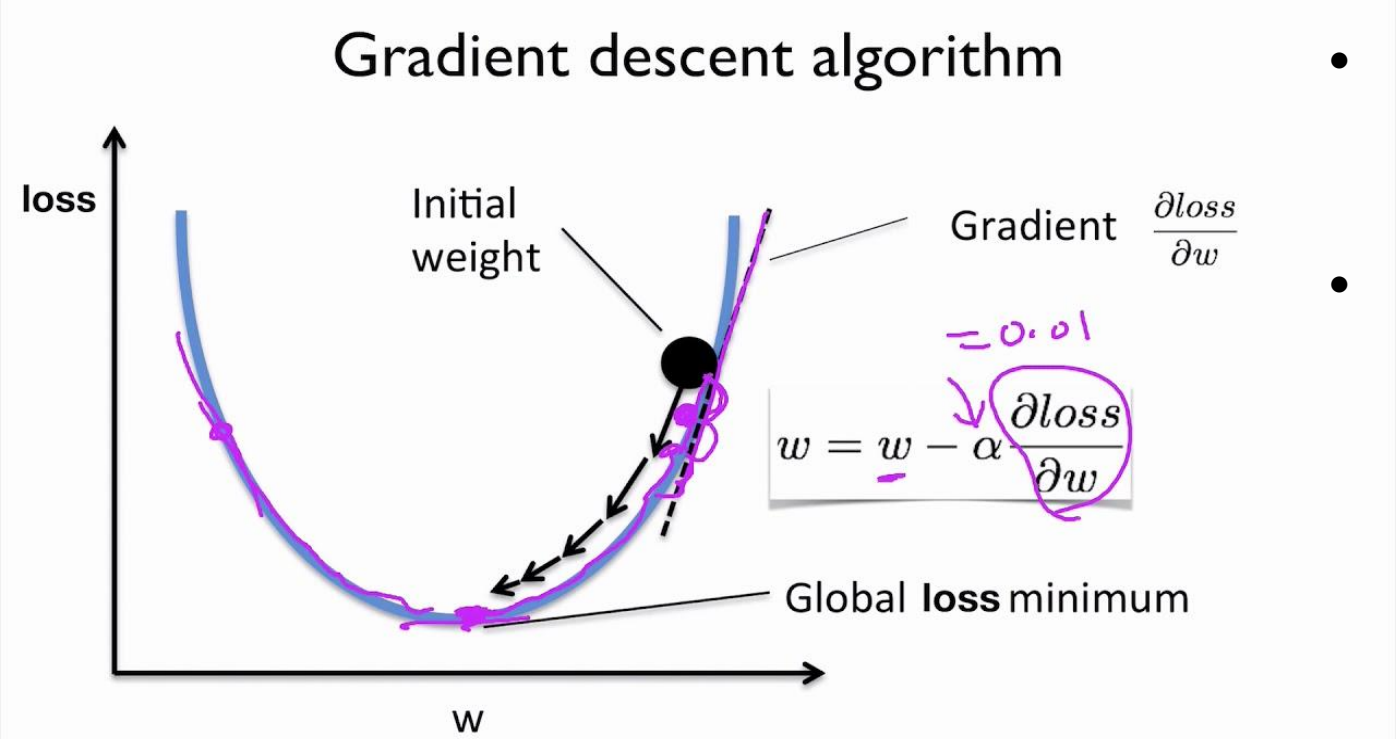
$a = 5$ $\quad f(a) = 25$

$a = 5.001$ $\quad f(a) \approx 25.010$

$\frac{d}{da} f(a) = 10$ when $a = 5$

$\frac{d}{da} f(a) = \frac{d}{da} a^2 = 2a$

Andrew Ng



- **Direction:** knowing the slope, help us know the direction (sign - whether to increase or decrease) to move the coefficient values
- **Magnitude:** knowing the slope (how steep it is), also help us know the magnitude (how much we need to change) the value. (More steep - Greater change, less steep - Lesser change.)- But still the question is how much more or less change/update? The answer is that,
- The size(magnitude) of our update is controlled by the learning rate.

Learning rate - Controlling the size of magnitude change

The size of these steps((magnitude of change in weights values)) is called the learning rate (α) that gives us some additional control over how large of steps we make.

Problem:

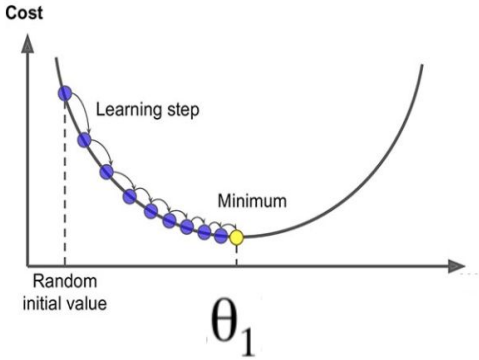
With a large learning rate, we can cover more ground each step, but we risk overshooting the lowest point since the slope of the hill is constantly changing. With a very low learning rate, we can confidently move in the direction of the negative gradient since we are recalculating it so frequently. A low learning rate is more precise, but calculating the gradient is time-consuming, so it will take us a very long time to get to the bottom. The most commonly used rates are: 0.001, 0.003, 0.01, 0.03, 0.1, 0.3.

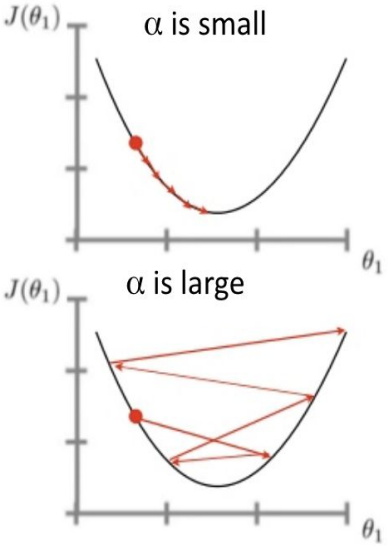
repeat until convergence {

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1)$$

(for $j = 1$ and $j = 0$)

}





Problem with Gradient Descent

Problem: Typical Gradient Descent optimization, uses the whole data set for each iteration (computationally very expensive to perform because we have to use all of the one million samples for completing one iteration, and it has to be done for every iteration until the minimum point is reached)

Variants of GD

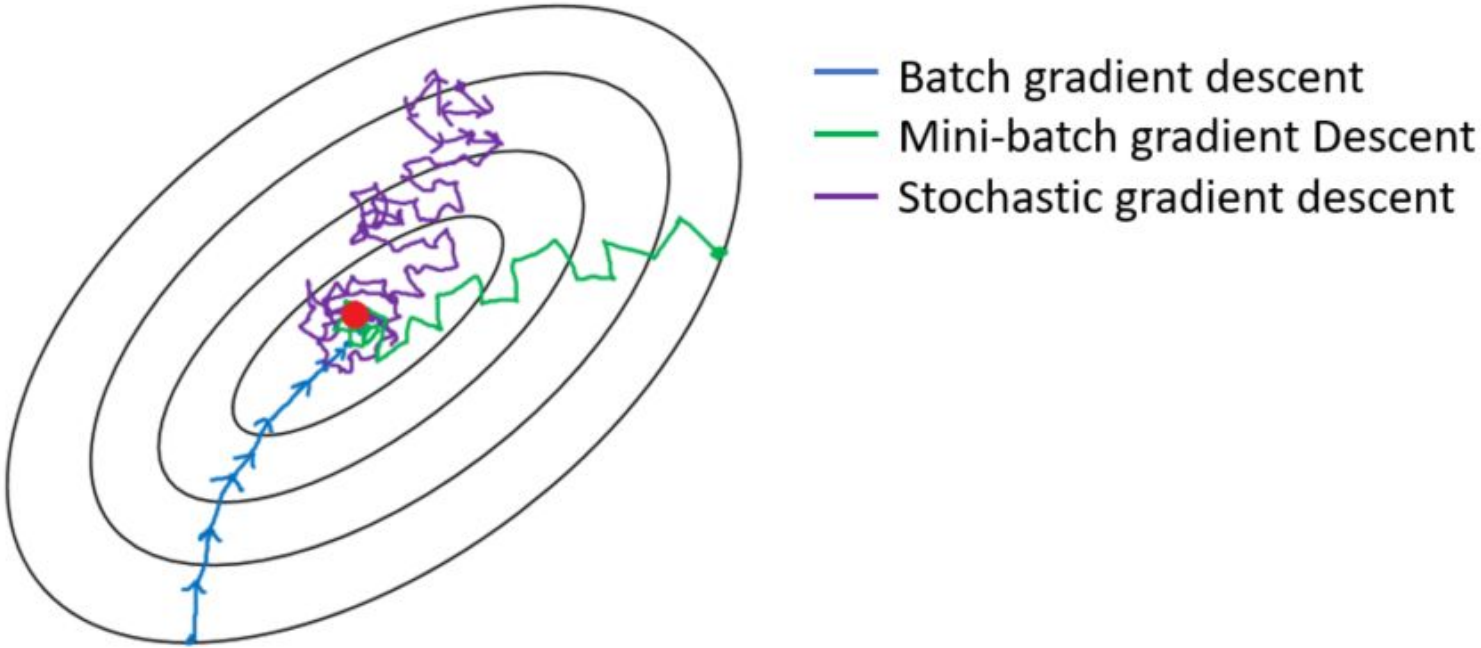
Batch Gradient Descent. Batch Size = Size of Training Set

Stochastic Gradient Descent. Batch Size = 1

Mini-Batch Gradient Descent. 1 < Batch Size < Size of Training Set
In the case of mini-batch gradient descent, popular batch sizes include 32, 64, and 128 samples

What if the dataset does not divide evenly by the batch size?

This can and does happen often when training a model. It simply means that the final batch has fewer samples than the other batches.


Batch gradient descent
Mini-batch gradient Descent
Stochastic gradient descent

Backpropagation

Backpropagation

- Backpropagation is a way/an algorithm for computing/calculating the gradient.
- One very important thing to note is that for backpropagation to work, our Activation Functions must be differentiable. Functions which are easy/fast to differentiate speeds up the overall process as it takes less time for computing derivatives.
- First, remember that the derivative of a function gives the direction in which the function increases, and its negative, the direction in which the function decreases.

  Training a model is just minimising the loss function, and to minimise you want to move in the negative direction of the derivative. Back-propagation is called like this because to calculate the derivative you use the chain rule from the last layer (which is the one directly connected to the loss function, as it is the one that provides the prediction) to the first layer, which is the one that takes the input data. You are "moving from back to front".

- Back-propagation is the process of calculating the derivatives and gradient descent is the process of descending through the gradient, i.e. adjusting the parameters of the model to go down through the loss function.

Computing derivatives (gradients) using chain rule

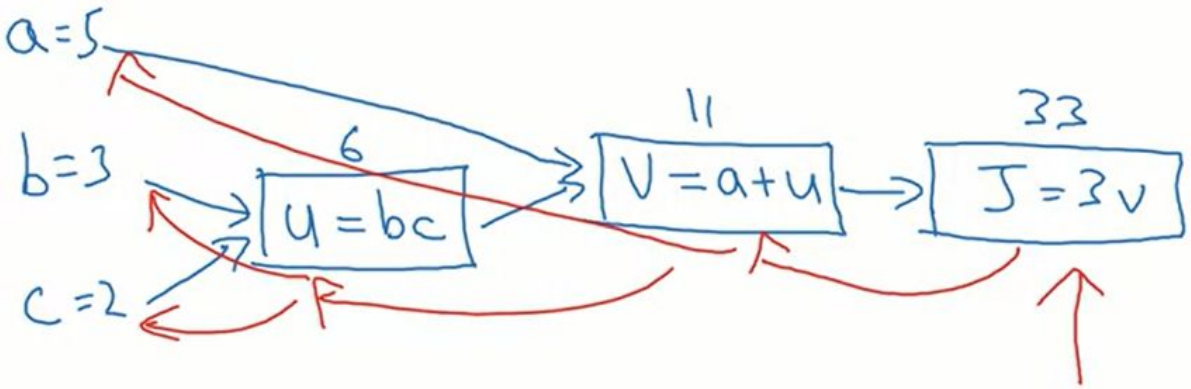
Computation Graph

$J(a,b,c) = 3(a+bc) = 3(5+3\times 2) = 33$

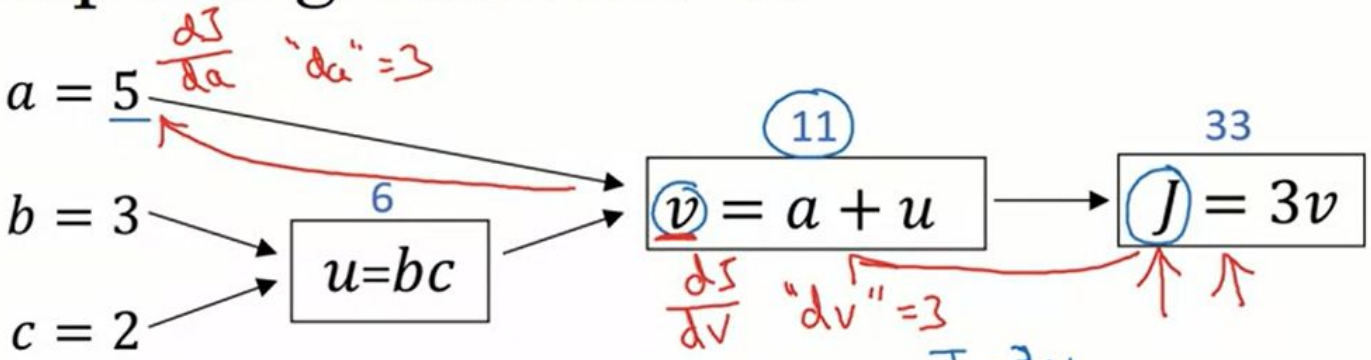
$u = bc$

$v = a+u$

$J = 3v$



Computing derivatives



$\frac{dJ}{dv} = ? = 3$

$\frac{dJ}{da} = 3 = \frac{dJ}{dv} \frac{dv}{da}$

3×1

$\frac{dv}{da} = 1$

$a \to v \to J$

$J = 3v$

$v = 11 \to 11.001$

$J = 33 \to 33.003$

$a = 5 \to 5.001$

$\to v = 11 \to 11.001$

$J = 33 \to 33.003$

$f(a) = 3a$

$\frac{df(a)}{da} = \frac{df}{da} = 3$

$J = 3v$

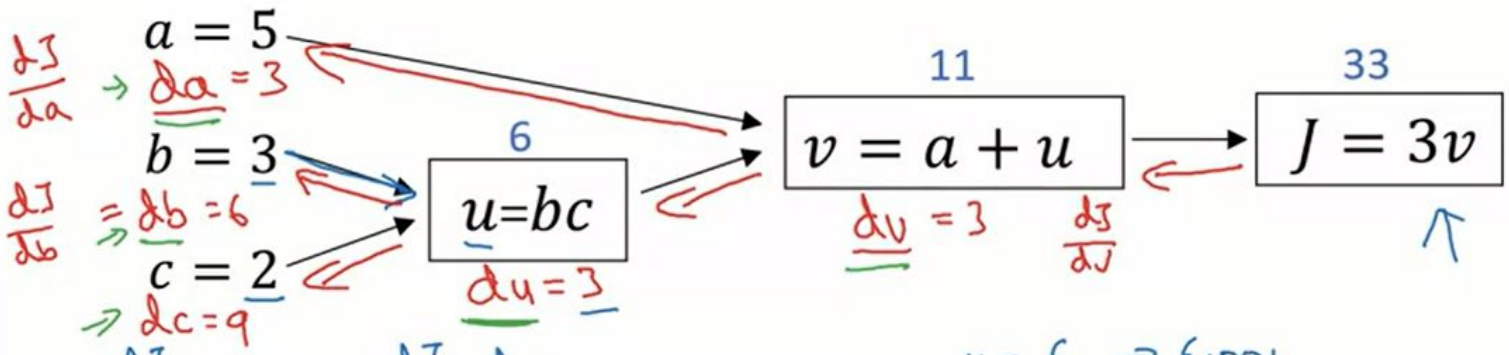
$\frac{dJ}{dv} = 3$

$\frac{d\,\text{FinalOutputVar}}{d\,\text{var}}$ → "dvar"

dJ/dvar

Andrew Ng

Computing derivatives



$$\frac{dJ}{du} = 3 = \frac{dJ}{dv} \cdot \frac{dv}{du}$$

($\frac{dJ}{dv} = 3$, $\frac{dv}{du} = 1$)

$$\frac{dJ}{db} = \frac{dJ}{du} \cdot \frac{du}{db} = 6$$

($\frac{dJ}{du} = 3$, $\frac{du}{db} = 2$)

$$\frac{dJ}{dc} = \frac{dJ}{du} \cdot \frac{du}{dc} = 9$$

($\frac{dJ}{du} = 3$, $\frac{du}{dc} = 3$)

$u = 6 \rightarrow 6.001$

$v = 11 \rightarrow 11.001$

$J = 33 \rightarrow 33.003$

$b = 3 \rightarrow 3.001$

$u = b \cdot c = 6 \rightarrow 6.002$

$J = 33.006$

$c = 2$

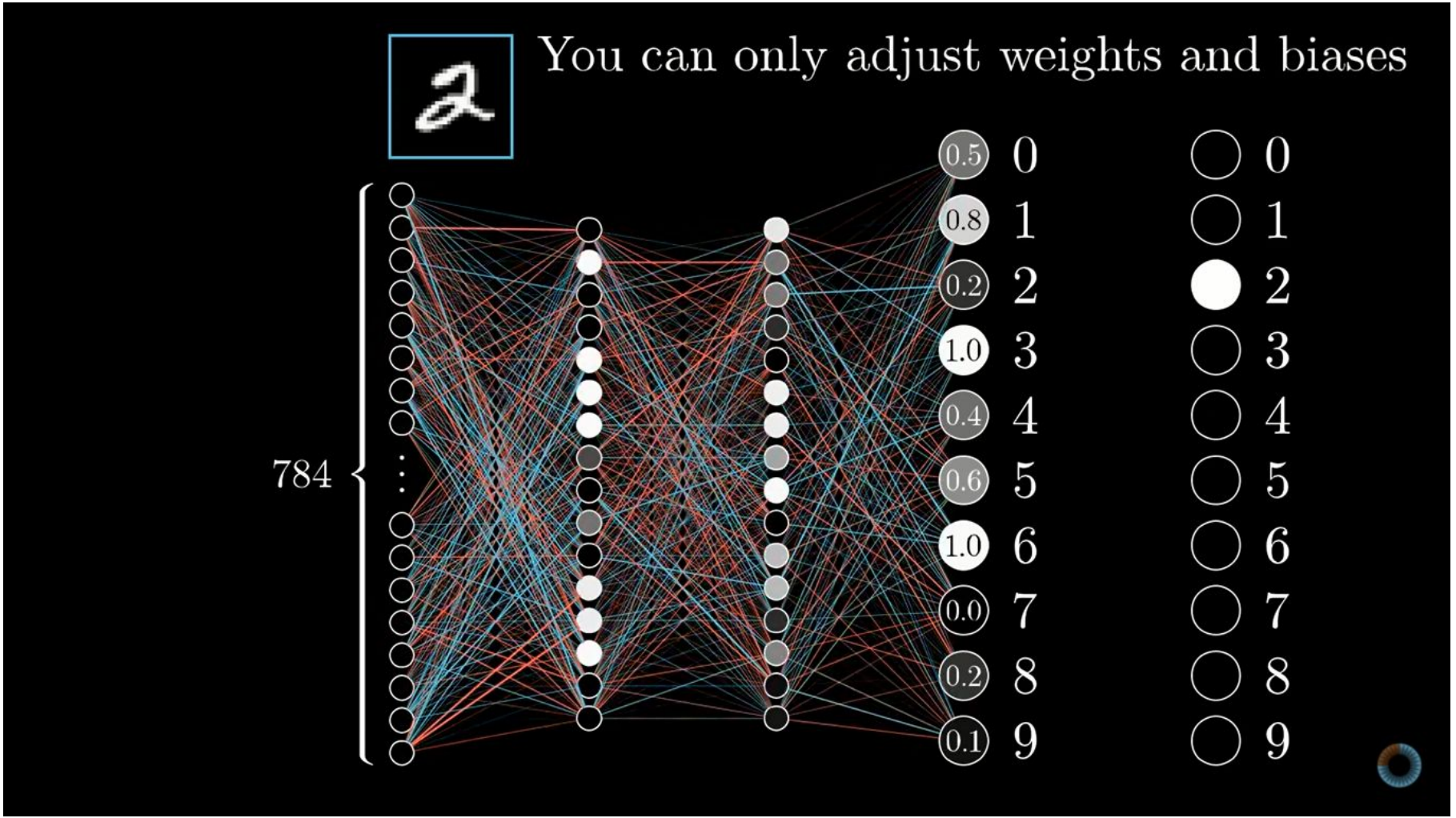
$.006$

$v = 11.002$

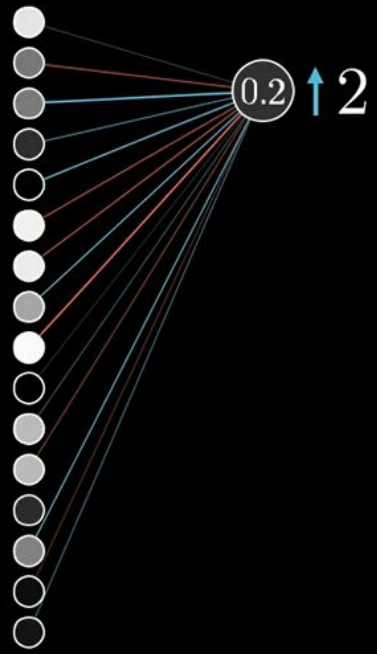
$J = 3v$

Andrew Ng

Example


You can only adjust weights and biases
784
0.5
0.8
0.2
1.0
0.4
0.6
1.0
0.0
0.2
0.1
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9

$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

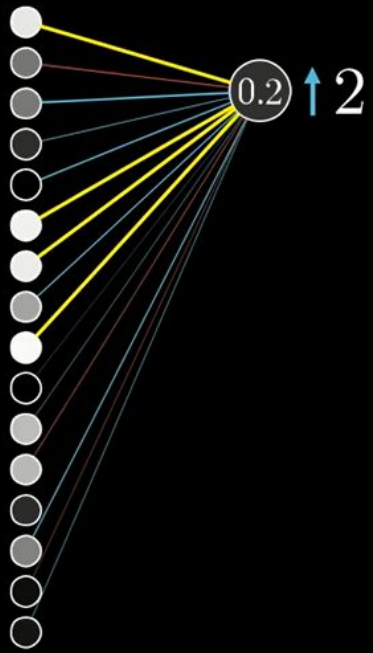


$$0.2 = \sigma(w_0 a_0 + w_1 a_1 + \cdots + w_{n-1} a_{n-1} + b)$$

Increase b

Increase w_i

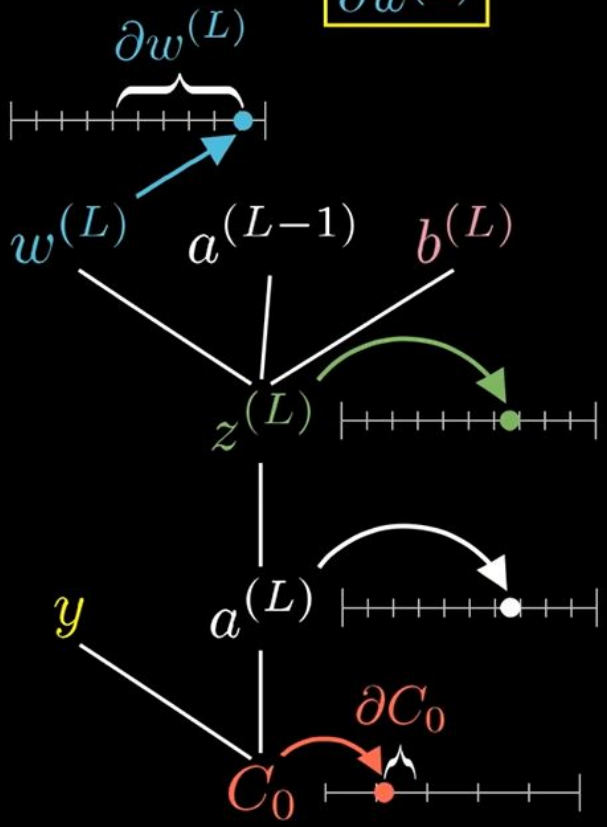
Change a_i



$\frac{\partial C_0}{\partial w^{(L)}}$ What we want $C_0(\dots) = (a^{(L)} - y)^2$

$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$

$a^{(L)} = \sigma(z^{(L)})$



Desired output

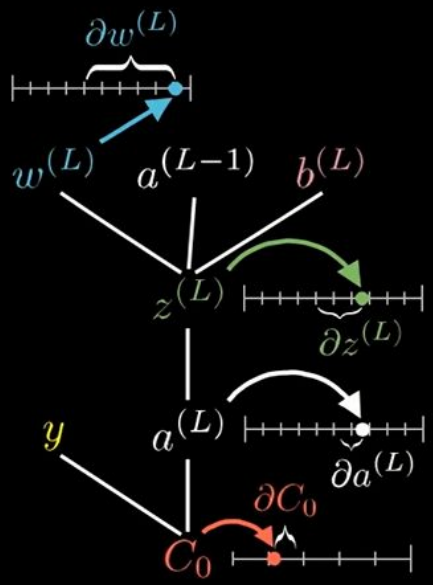
$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}}$$

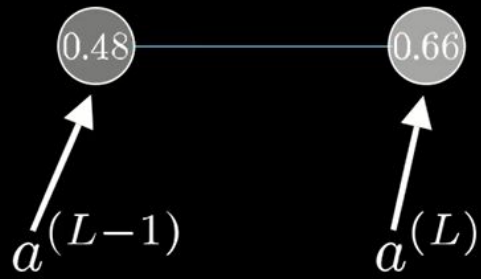
Chain rule

$$C_0(\dots) = (a^{(L)} - y)^2$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$





Desired output

$$\frac{\partial C_0}{\partial w^{(L)}} = \frac{\partial z^{(L)}}{\partial w^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial C0}{\partial a^{(L)}} = a^{(L-1)} \sigma'(z^{(L)}) 2(a^{(L)} - y)$$

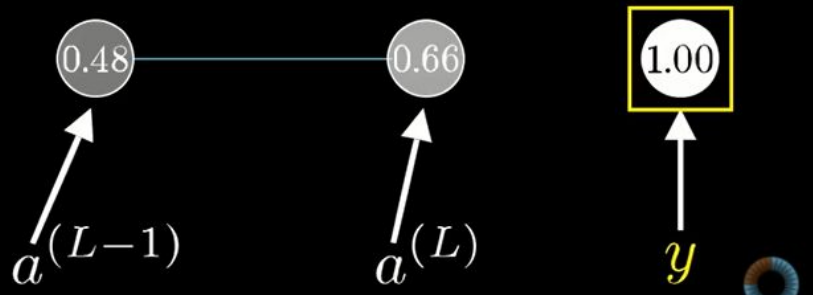
$$C_0 = (a^{(L)} - y)^2$$

Average of all training examples

$$\frac{\partial C}{\partial w^{(L)}} = \overbrace{\frac{1}{n} \sum_{k=0}^{n-1} \frac{\partial C_k}{\partial w^{(L)}}}$$

$$z^{(L)} = w^{(L)} a^{(L-1)} + b^{(L)}$$

$$a^{(L)} = \sigma(z^{(L)})$$



References

1. Cost, Activation, Loss Function|| Neural Network|| Deep Learning. What are these? https://medium.com/@zeeshanmulla/cost-activation-loss-function-neural-network-deep-learning-what-are-these-91167825a4de
2. Activation functions and its types https://medium.com/@vinodhb95/activation-functions-and-its-types-8750f1287464
3. Deep Learning: Which Loss and Activation Functions should I use? https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8
4. Fundamentals of Deep Learning – Activation Functions and When to Use Them? https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/
5. Gradient Descent: An Introduction to 1 of Machine Learning's Most Popular Algorithms https://builtin.com/data-science/gradient-descent
6. https://towardsdatascience.com/coding-deep-learning-for-beginners-linear-regression-part-2-cost-function-49545303d29f
7.

More terminology and Concepts

NN terminology & Hyperparameters

In the neural network terminology:

One Forward-Backward Pass = One time weights Update

one **epoch** = number of complete passes through the training dataset.(the number of times that the learning algorithm will work through the entire training dataset.)

The number of epochs is traditionally large, often hundreds or thousands, allowing the learning algorithm to run until the error from the model has been sufficiently minimized. You may see examples of the number of epochs in the literature and in tutorials set to 10, 100, 500, 1000, and larger.

batch size =number of samples processed before the model is updated.(the number of training examples in one forward/backward pass). The higher the batch size, the more memory space you'll need.

number of iterations = number of passes, each pass using [batch size] number of examples. To be clear, one pass = one forward pass + one backward pass (we do not count the forward pass and backward pass as two different passes).

Example: if you have 1000 training examples, and your batch size is 500, then it will take 2 iterations to complete 1 epoch.

Online vs Batch Learning

Online learning: weights are updated after predicting single sample - (More iterations) - (iteration means forward and backward pass till weight updates) or number of times weights are updated)

i.e ANN takes single sample as input, make prediction, compute error/loss, and loss is used to updates weights. Then takes 2nd sample, make prediction, compute error, and update weights. And the process goes on till updating weights for all the sample in the dataset (one epoch).

Batch Learning: weights are updated after predicting batch of samples (less iterations)

i.e. ANN takes single sample as input, make prediction, compute error/loss. Then, takes another sample as input, make prediction, compute error/loss. Then take 3rd, 4th and so on. And calculates the error for their prediction, till the size of the batch.

Then calculates the average of all these errors/losses for all the samples in a single batch. Then, this average error/loss (cost) of the single batch is used to updates the weights.

This same process is repeated for batch 2nd, 3rd and so on, till all the samples in the dataset.(one epoch)

Worked Example

Finally, let’s make this concrete with a small example.

Assume you have a dataset with 200 samples (rows of data) and you choose a batch size of 5 and 1,000 epochs.

This means that the dataset will be divided into 40 batches, each with five samples. The model weights will be updated after each batch of five samples.

This also means that one epoch will involve 40 batches or 40 updates to the model.

With 1,000 epochs, the model will be exposed to or pass through the whole dataset 1,000 times. That is a total of 40,000 batches during the entire training process.

Xtra

Loss Functions

Loss is nothing but a prediction error of Neural Net. And the method to calculate the loss is called Loss Function.

Loss functions are mainly classified into two different categories that are Classification loss and Regression Loss.

Cross-entropy and mean squared error are the two main types of loss functions to use when training neural network models

Classification Loss(loss functions used in classifications problems): Cross entropy (Categorical cross entropy - for multiclass classifications), Hinge loss, square loss

Regression Loss(loss functions used in regression problems): Mean square error/L2 loss(“MSE is the most commonly used regression loss function”), Mean absolute error/L1 loss, mean bias error

In simple words, the Loss is used to calculate the gradients. And gradients are used to update the weights of the Neural Net. This is how a Neural Net is trained.

how do neural networks get optimal weight and bias values?

The answer is through an error gradient. What we want to know when fixing the current weight and bias (which is initially generated randomly) is whether the current weight and bias values are too large or too small (do we need to decrease or increase our current value?) with respect to their optimal value? And how much it deviates (how much we need to decrease or increase our current value?) from their optimal values.

Optimizer

Optimizers are algorithms or methods used to change the attributes of the neural network such as weights and learning rate to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function.

Having these so many **Activations**, **Error/Loss functions** and **Optimization techniques**. The question is:

Which one to use when!?

It depends on;

What are you trying to solve?