

Recap...

- Objectives of NLP
- Characteristics of language
 - Word
 - Morphology
 - Inflection
 - Derivations
 - Compounding
 - MWE
 - Phrasal verbs
 - Syntax
 - Language models
 - Parsing
 - Grammar
 - Coreference
 - Semantics
 - Word similarity
 - Distributional Semantics
- Applications
 - POS
 - NER
 - Parsing
- NLP Features
 - Unigram, bigram, n-gram
 - Tf-idf
- Building NLP systems
 - Data annotation
- Machine Learning
 - Regression
 - Naïve Bayes
 - Decision Tree
 - Ensemble models
- Word Representations
 - Word2Vec

NLP Applications

1. Language modeling
2. Text classification
3. Sentiment analysis
4. Part-of-speech tagging
5. Named entity recognition
6. Question answering
7. Text summarization
8. Machine translation
9. Topic modelling
10. Semantic role labeling
10. Dialogue systems
11. Relation extraction
12. Coreference resolution
13. Text normalization
14. Paraphrase detection
15. Emotion detection
16. Irony detection
17. Language identification
18. Spell correction
19. Text generation

Language Model

- The students opened their
- *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____*

Text classification

RR vs GT Live Score, IPL 2023: Wriddhiman Saha, Shubman Gill give Gujarat Titans a good start

Rajasthan Royals skipper Sanju Samson won the toss and decided to bat first against Gujarat Titans in Jaipur. Gujarat Titans will be eager to move on from their underwhelming batting display when they... [READ MORE](#)

COMMENTARY

SCORECARD

INFOGRAFCRICS

ALL

EDITOR'S NOTE

WICKETS

SIXES

FOURS

21:42 (IST)
May 05

EDITOR'S NOTE

IPL 2023 Live Score: Gujarat 25/0 after 4 overs | Target 119

21:37 (IST)
s.com/sports/tennis

EDITOR'S NOTE

IPL 2023 Live Score: Gujarat 16/0 after 3 overs | Target 119

4

Sentiment Analysis

- This movie is not good

Its **battery** is awesome but **camera** is very poor.

Positive about the **battery** but *negative* about the **camera**

- IDENTIFY THE PERSON NAMES
- Harry potter and Hermione Granger invented a new spell

POS Tagging

IDENTIFY PARTS OF SPEECH

John saw the Girl

John saw the saw

“saw” is more likely to be a verb V
rather than a noun N

The second “saw” is a noun N because
a noun N is more likely to follow a
determiner.

- How can I know $P(V|PN)$, $P(\text{saw}|V)$?
- Obtaining from training data

Training Data:

- (x^1, \hat{y}^1) 1 Pierre/**NNP** Vinken/**NNP** ,/, 61/**CD** years/**NNS** old/**JJ** ,/, will/**MD**
 join/**VB** the/**DT** board/**NN** as/**IN** a/**DT** nonexecutive/**JJ** director/**NN**
 Nov./**NNP** 29/**CD** ./
- (x^2, \hat{y}^2) 2 Mr./**NNP** Vinken/**NNP** is/**VBZ** chairman/**NN** of/**IN** Elsevier/**NNP**
 N.V./**NNP** ,/, the/**DT** Dutch/**NNP** publishing/**VBG** group/**NN** ./
- (x^3, \hat{y}^3) 3 Rudolph/**NNP** Agnew/**NNP** ,/, 55/**CD** years/**NNS** old/**JJ** and/**CC**
 chairman/**NN** of/**IN** Consolidated/**NNP** Gold/**NNP** Fields/**NNP** PLC/**NNP**
 ,/, was/**VBD** named/**VRN** a/**DT** nonexecutive/**JJ** director/**NN** of/**IN**
 this/**DT** British/**JJ** industrial/**JJ** conglomerate/**NN** ./
- ⋮

Structured prediction (sequence tagging) – label for a word depends on other labels

- Rather than classifying each word independently

Q&A

Example

Has anyone been charged in relation to Jiah Khan's murder?

Sooraj Pancholi

What has he been charged with?

Abetting murder

Among the 39 witnesses listed by the CBI in the chargesheet against Sooraj Pancholi is his childhood friend , Karan Joshi(22), who said the two of them were together when Sooraj learnt of his girlfriend , Jiah Khan 's death.Also Read : There were pieces of glass in Jiah 's hand , says Pancholis ' domestic help In the CBI chargesheet , which names Sooraj for abetting the actress 's suicide , Karan reveals that it was Sooraj 's father Aditya Pancholi who had broken the news to them . In a December 12 report , mid day had already drawn up a timeline of events leading up to the moment Jiah was found hanging from the ceiling fan in her Juhu home(` CBI chargesheet logs details of Sooraj Jiah 's chaotic relationship '). Karan 's statement is useful in tracing what took place after this , when Sooraj learnt of her death . On the night of Jiah 's suicide June 3 , 2013 Karan received a call from Aditya at about 11.45 pm . Aditya asked for Sooraj , who was at his flat in Juhu with Karan . Aditya called both of them to his bungalow , where he was waiting with a friend , Kavya . Read Story : Suicide or homicide ? Forensic experts differ Karan told the police that when they got there , the whole group headed to the living room , where Aditya told Sooraj that Jiah had been found hanging . " Immediately , Suraj broke down , " said Karan in his statement . Aditya then asked Sooraj when he had last had contact with Jiah . Sooraj narrated the day 's events to his father . Around 1.30 am , another friend of Aditya 's , Munnu , came by the bungalow and informed them that Jiah 's body had been shifted to Cooper Hospital . " Suraj hugged Munnu and cried badly , " said Karan . Also Read : Actor Sooraj Pancholi drove Jiah to suicide : CBI charge sheet Around 2 am , Aditya and Munnu left to meet Jiah 's mother Rabiya at her residence . Afterwards , they returned to the bungalow , where Sooraj , Karan and Kavya had stayed put . After a while , everyone left the house . Aditya and Sooraj took one car , while Karan was asked to follow them in another . They drove to Seven Bungalows in Andheri , where Aditya and Sooraj " discussed something " in the car , after which Sooraj moved to Karan 's car and they all returned home and went to bed . Karan 's timeline ends on the morning of June 4 , 2013 , when Sooraj 's mother Zarina Wahab came to the Bandra bungalow , after which they directly went to Juhu Police station for questioning .

Text Summarization using NLP

Natural Language Processing

Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The result is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them. The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves.

Summary

`summarize(text, 0.6)`

Natural Language Processing

Natural language processing (NLP) is a subfield of linguistics, computer science, and artificial intelligence concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data.

Machine Translation

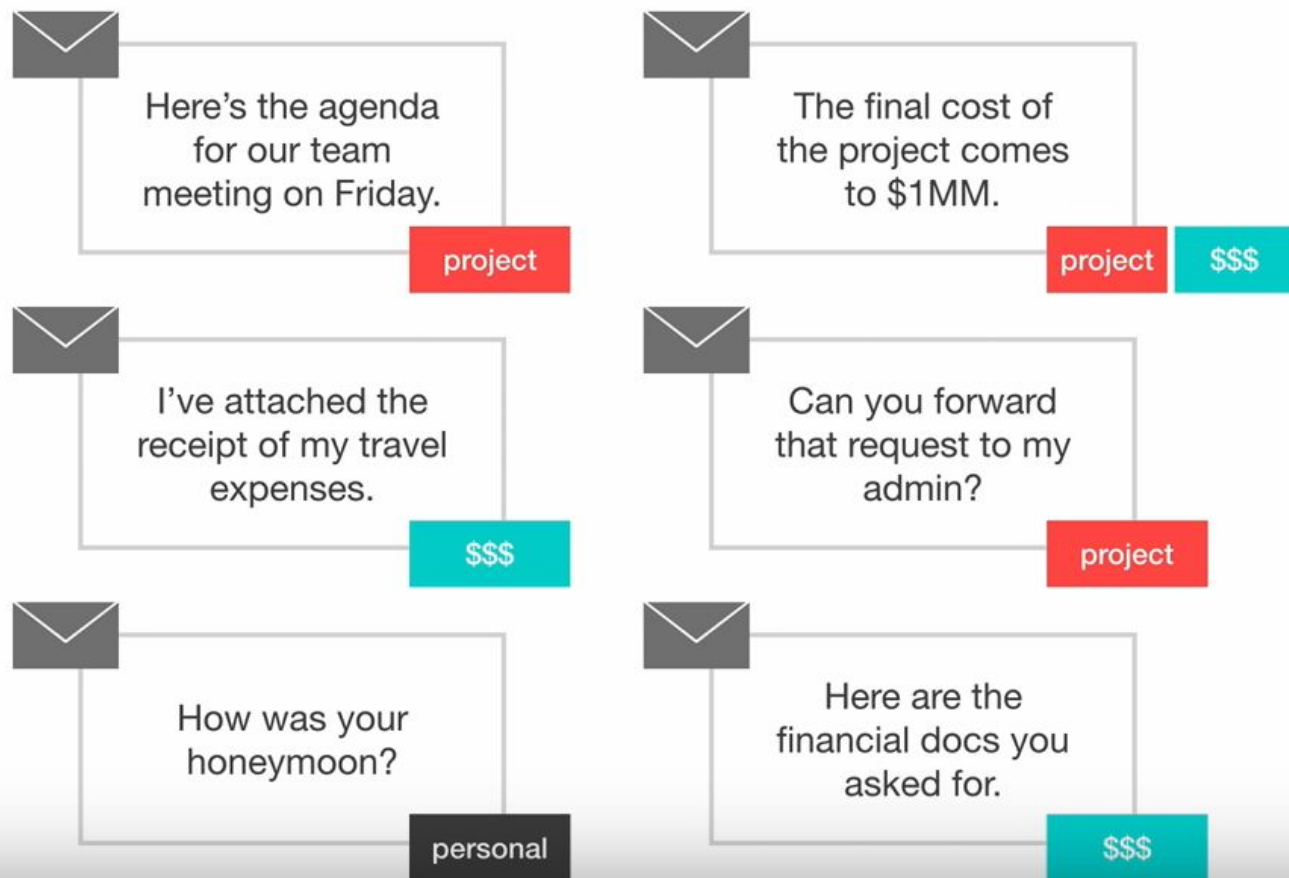
- Machine Translation (MT) is the task of translating a sentence x from one language (the source language) to a sentence y in another language (the target language).

Be the change you want to see in the world

वह परिवर्तन बनो जो संसार में देखना चाहते हो



Topic Modeling



Role Labeling

- Multiple teams of Gandhinagar police's local crime branch (LCB) arrested Shailesh Patel, one of the two prime accused in the Navin Shah kidnapping-murder case, and also seized the SUV in which the murder took place. Shah, 69, was director of printing of Navneet Education and was killed in a moving car on July 25 on SG Road.

Ram Kills Shyam

Ram ne Shyam ko mArA

Relation Extraction task

- Relation Extraction is the task of predicting attributes and relations for entities in a sentence

Barack Obama was born in Honolulu, Hawaii.

predict each relation - r

(Barack Obama, $r1$, Honolulu)

(Barack Obama, $r2$, Hawaii)

(Honolulu, $r3$, Hawaii)


Ideally - what should be predicted

$r1$ = was born in

$r2$ = was born in

$r3$ = is in

*“I voted for Nader because he was most
aligned with my values,” she said.*



- **Sentence 1** : Amrozi accused his brother, whom he called "the witness", of deliberately distorting his evidence.
- **Sentence 2** : Referring to him as only "the witness", Amrozi accused his brother of deliberately distorting his evidence.
- **Class** : 1 (true paraphrase)

Introduction to Neural Networks

Tirthankar Dasgupta

TCS Research

what is deep learning defining deep learning in the context of intelligence intelligence is actually the ability to process information such that it can be used to inform a future decision

now the field of artificial intelligence or AI is a science that actually focuses on building algorithms to do exactly this to build algorithms to process information such that they can inform

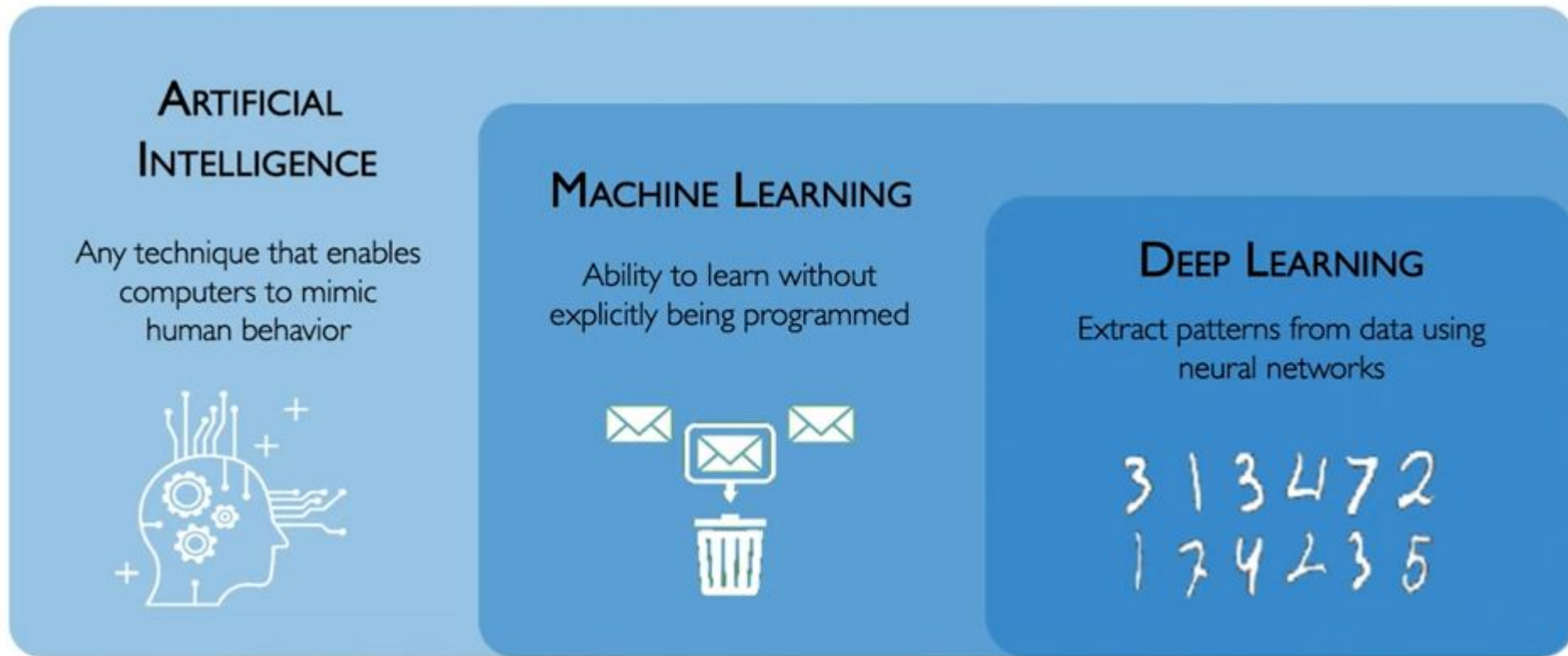
future predictions now machine learning you can think of this as just a subset of AI that actually focuses on teaching an algorithm to learn from experiences without being explicitly

programmed now deep learning takes this idea even further and it's a subset of machine learning that

focuses on using neural networks to automatically extract useful patterns in raw data and then using

these patterns or features to learn to perform that task

What is Deep Learning?



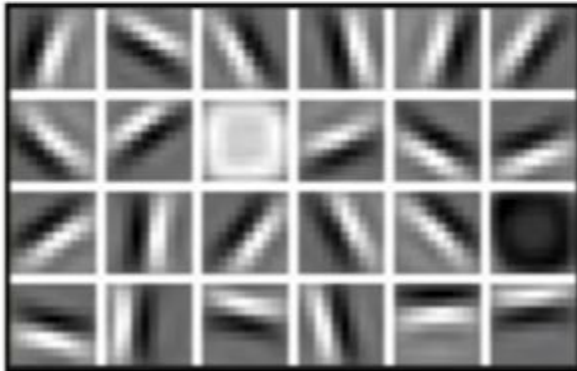
- let's start by asking ourselves a question about why do we all care about deep learning and
- specifically why do we care right now understand that it's important to actually understand first
- why is deep learning or how is deep learning different from traditional machine learning
- now traditionally machine learning algorithms define a set of features in their data usually
- these are features that are handcrafted or hand engineered and as a result they tend to be pretty
- brittle in practice when they're deployed the key idea of deep learning is to learn these
- features directly from data in a hierarchical manner that is can we learn if we want to learn
- how to detect a face for example can we learn to first start by detecting edges in the image
- composing these edges together to detect mid-level features such as a eye or a nose
- or a mouth and then going deeper and composing these features into structural facial features
- so that we can recognize this face this is this hierarchical way of thinking is really core
- to deep learning as core to everything that we're going to learn in this class

Why Deep Learning? Why Now?

Hand engineered features are time consuming, brittle, and not scalable in practice

Can we learn the **underlying features** directly from data?

Low Level Features



Lines & Edges

Mid Level Features



Eyes & Nose & Ears

High Level Features

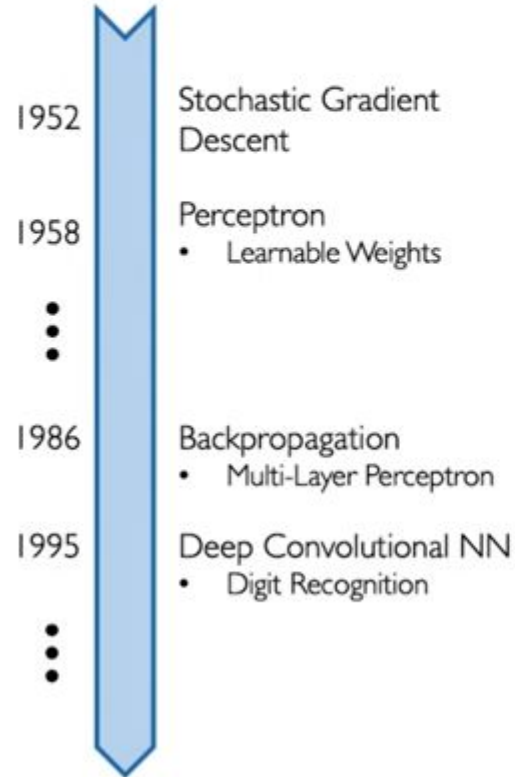


Facial Structure

- actually the fundamental building blocks though of deep learning and neural networks have actually
- existed for decades so one interesting thing to consider is why are we studying this now now is an
- incredibly amazing time to study these algorithms and for one reason is because data has become
- much more pervasive these models are extremely hungry for data and at the moment we're living
- in an era where we have more data than ever before secondly these algorithms are massively
- parallelizable so they can benefit tremendously from modern gpu hardware that simply did not exist
- when these algorithms were developed and finally due to open source toolboxes like tensorflow
- building and deploying these models has become extremely streamlined

Why Now?

Neural Networks date back decades, so why the resurgence?



1. Big Data

- Larger Datasets
- Easier Collection & Storage

IMAGENET



WIKIPEDIA
The Free Encyclopedia



2. Hardware

- Graphics Processing Units (GPUs)
- Massively Parallelizable



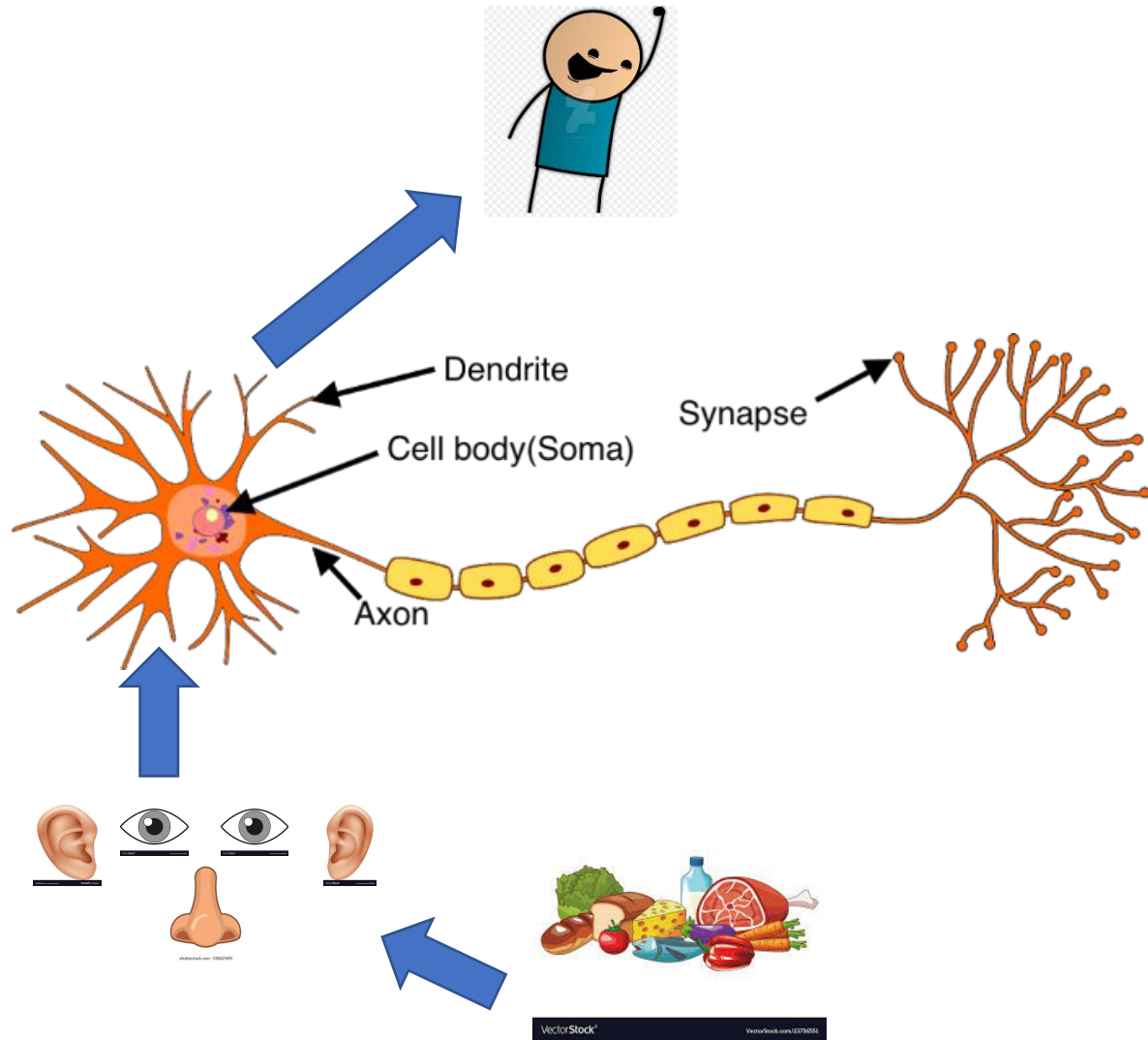
3. Software

- Improved Techniques
- New Models
- Toolboxes



Recap...Neural Networks

Overtly Simplified Definition of Biological Neuron



1. Dendrite: responsible for getting incoming signals from outside

2. Soma

Soma is the cell body responsible for the processing of input signals and deciding whether a neuron should fire an output signal

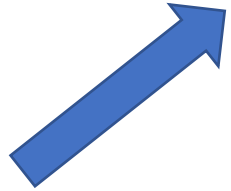
3. Axon

Axon is responsible for getting processed signals from neuron to relevant cells

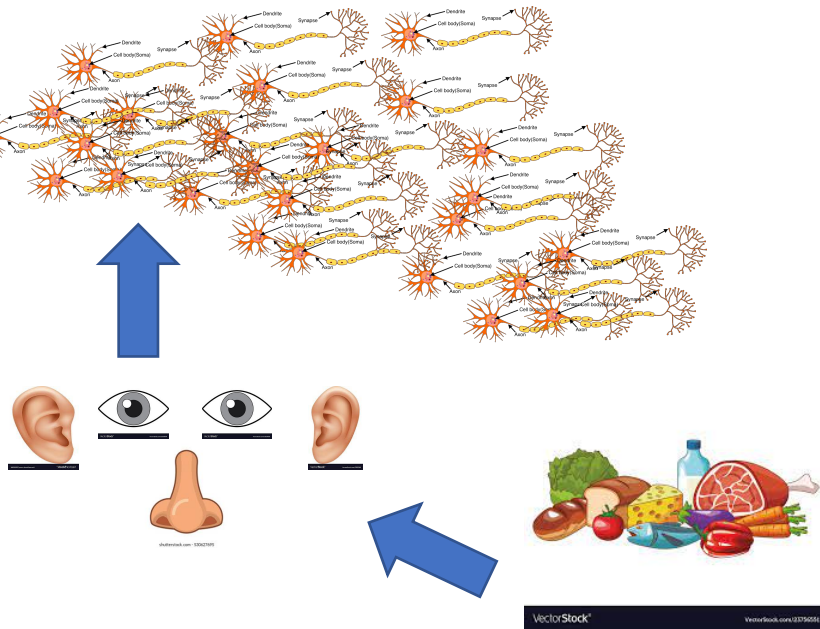
4. Synapse

Synapse is the connection between an axon and other neuron dendrites

Overtly Simplified Definition of Biological Neuron

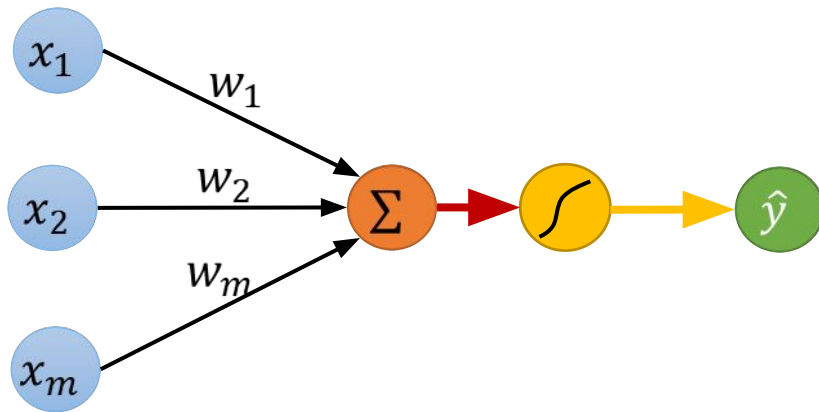


- Not just a single neuron
- Massively parallel interconnected network of neurons
 - Multi layered
- Sense organs relay signals to the lowest layer
- Some of these neurons may 'fire' based on the processed information, which gets transmitted to other neurons
- An average human brain contains around 10^{11} neurons
- Each neuron perform a certain role or respond to certain stimulus



The Perceptron

- The Structural building block



Inputs

Weights

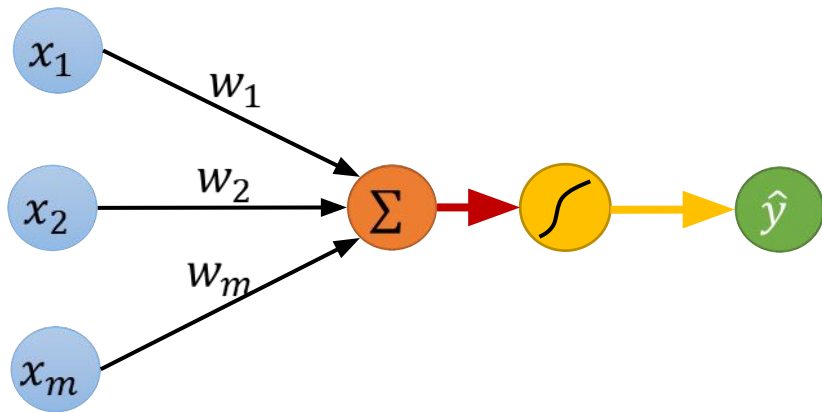
Sum

Non-Linearity

Output

The Perceptron

- The Structural building block



Inputs Weights Sum Non-Linearity Output

Diagram illustrating the mathematical representation of the perceptron output:

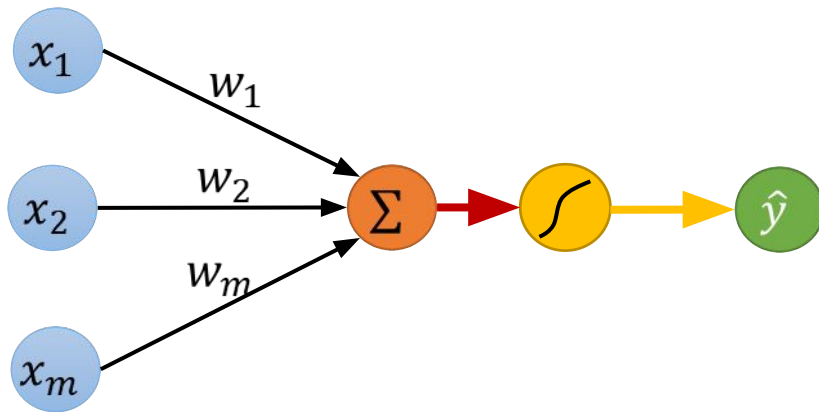
$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

Labels and arrows in the diagram:

- Output:** Points to \hat{y} (purple arrow).
- Linear combination of inputs:** Points to the summation term $\sum_{i=1}^m x_i w_i$ (red arrow).
- Bias:** Points to w_0 (green arrow).
- Non-linear activation function:** Points to g (yellow arrow).

The Perceptron

- The Structural building block



Inputs Weights Sum Non-Linearity Output

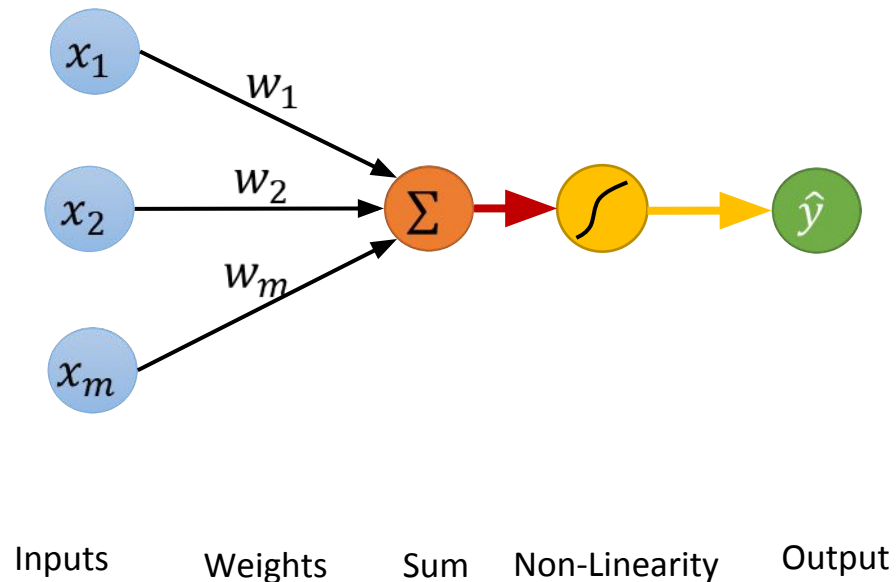
$$\hat{y} = g \left(w_0 + \sum_{i=1}^m x_i w_i \right)$$

$$\hat{y} = g (w_0 + \mathbf{X}^T \mathbf{W})$$

$$\text{where: } \mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \text{ and } \mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

The Perceptron

- The Structural building block

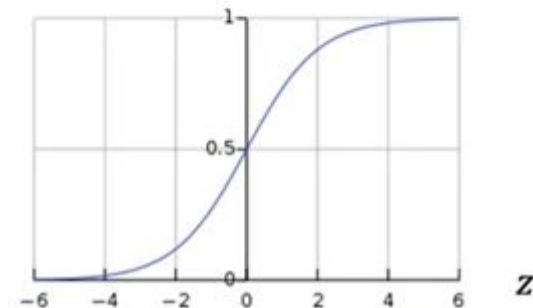


Activation Functions

$$\hat{y} = g(w_0 + \mathbf{X}^T \mathbf{W})$$

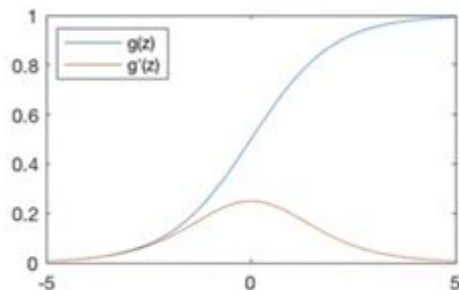
- Example: sigmoid function

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




Common Activation Functions

Sigmoid Function

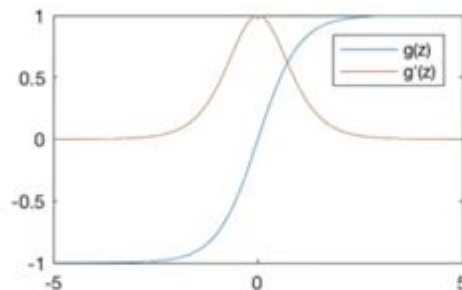


$$g(z) = \frac{1}{1 + e^{-z}}$$

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


 `tf.math.sigmoid(z)`

Hyperbolic Tangent

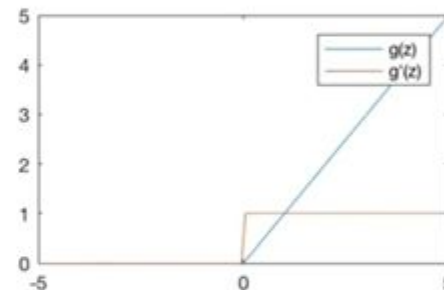


$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

 `tf.math.tanh(z)`

Rectified Linear Unit (ReLU)



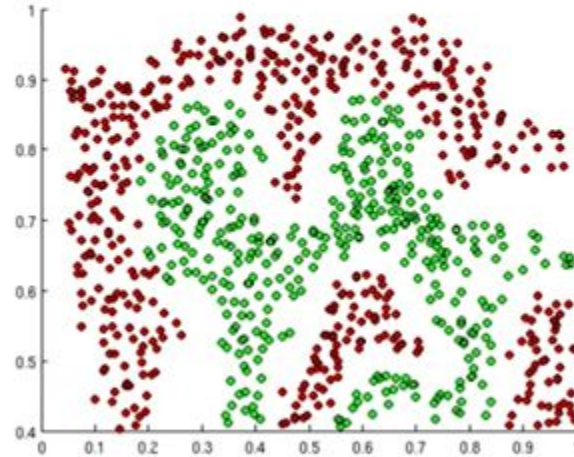
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

 `tf.nn.relu(z)`

Importance of Activation Functions

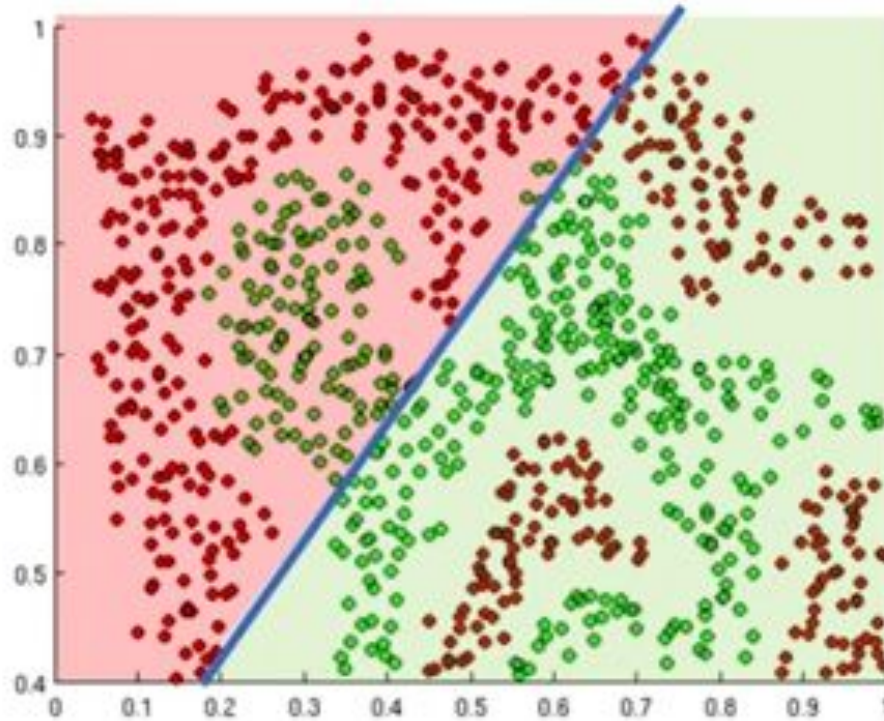
The purpose of activation functions is to **introduce non-linearities** into the network



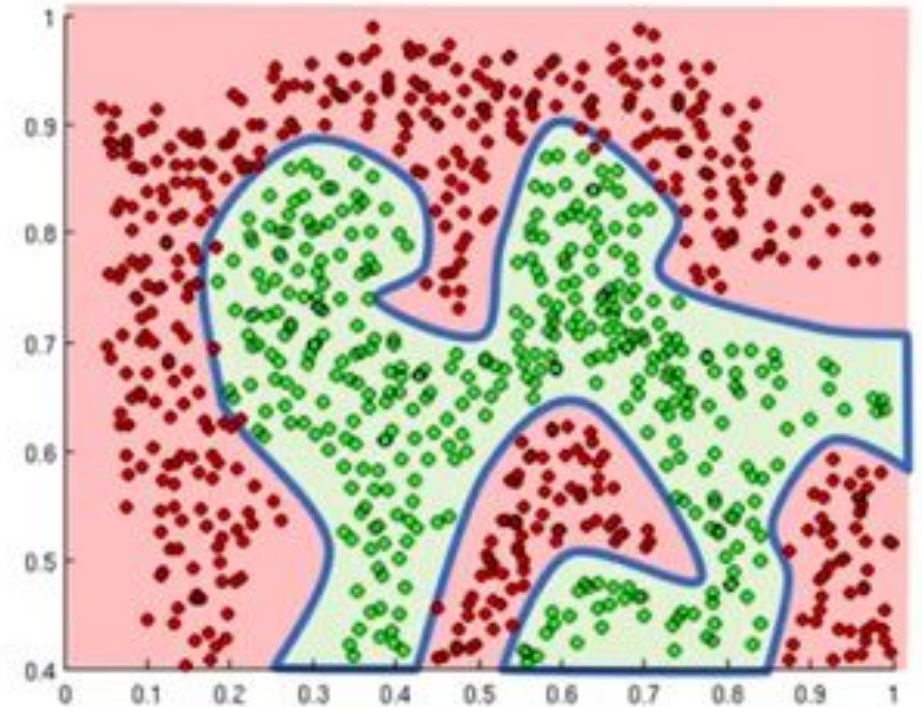
What if we wanted to build a neural network to distinguish green vs red points?

Importance of Activation Functions

The purpose of activation functions is to *introduce non-linearities* into the network



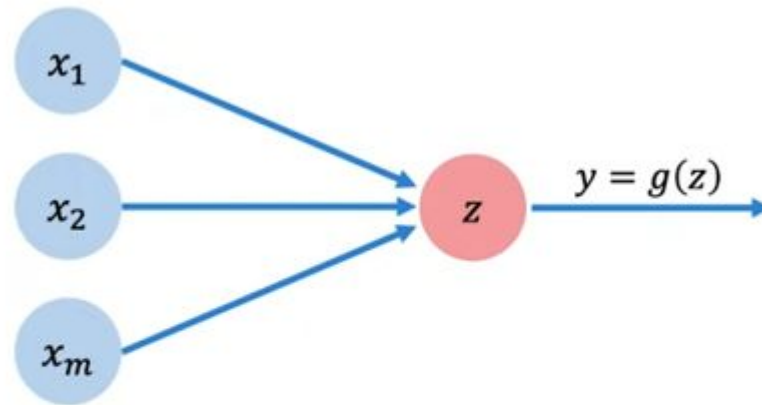
Linear activation functions produce linear decisions no matter the network size



Non-linearities allow us to approximate arbitrarily complex functions

- let's simplify this diagram a little bit let's clean up some of the arrows and remove the bias
- and we can actually see now that every line here has its own associated weight to it and i'll
- remove the bias term like i said for simplicity note that z here is the result of that dot
- product plus bias before we apply the activation function though g the final output though is is
- simply y which is equal to the activation function of z which is our activation value

Perceptron Simplified

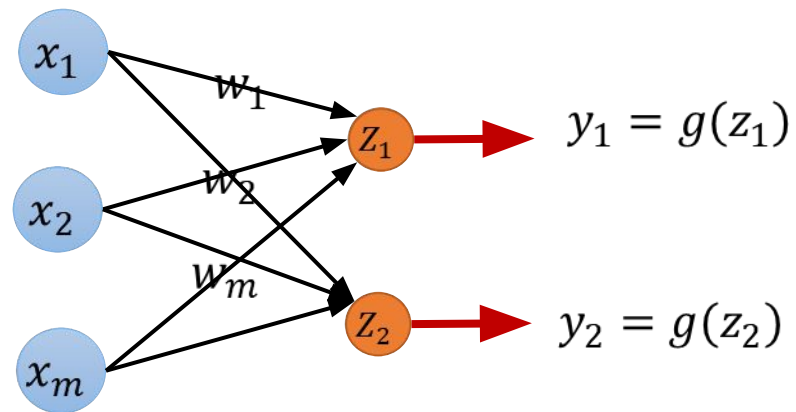


$$z = w_0 + \sum_{j=1}^m x_j w_j$$

- now if we want to define a multi-output neural network we can simply add another perceptron to
- this picture so instead of having one perceptron now we have two perceptrons and two outputs each
- one is a normal perceptron exactly like we saw before taking its inputs from each of the x_i
- ones through x_m 's taking the dot product adding a bias and that's it now we have two outputs each of
- those perceptrons though will have a different set of weights remember that we'll get back to that

Multi Output Perceptron

Because all inputs are densely connected to all outputs, these layers are called **Dense** layers

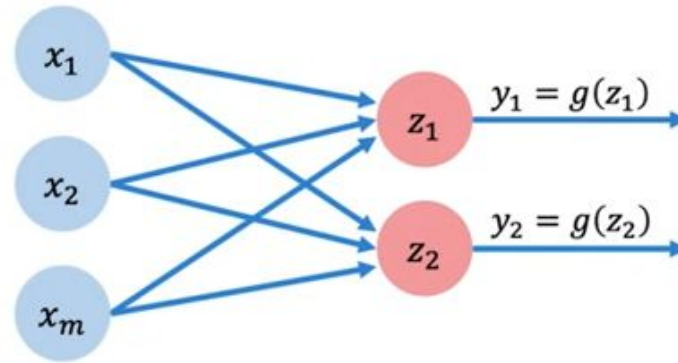


Dense layer from scratch


```
class MyDenseLayer(tf.keras.layers.Layer):  
    def __init__(self, input_dim, output_dim):  
        super(MyDenseLayer, self).__init__()  
  
        # Initialize weights and bias  
        self.W = self.add_weight([input_dim, output_dim])  
        self.b = self.add_weight([1, output_dim])  
  
    def call(self, inputs):  
        # Forward propagate the inputs  
        z = tf.matmul(inputs, self.W) + self.b  
  
        # Feed through a non-linear activation  
        output = tf.math.sigmoid(z)  
  
        return output
```


Multi Output Perceptron

Because all inputs are densely connected to all outputs, these layers are called **Dense** layers

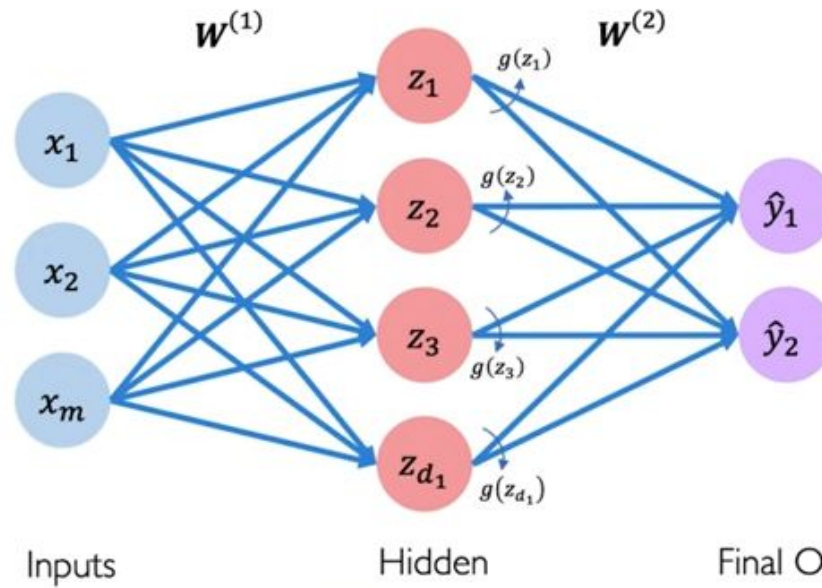


$$z_i = w_{0,i} + \sum_{j=1}^m x_j w_{j,i}$$

```
 import tensorflow as tf  
  
layer = tf.keras.layers.Dense(  
    units=2)
```

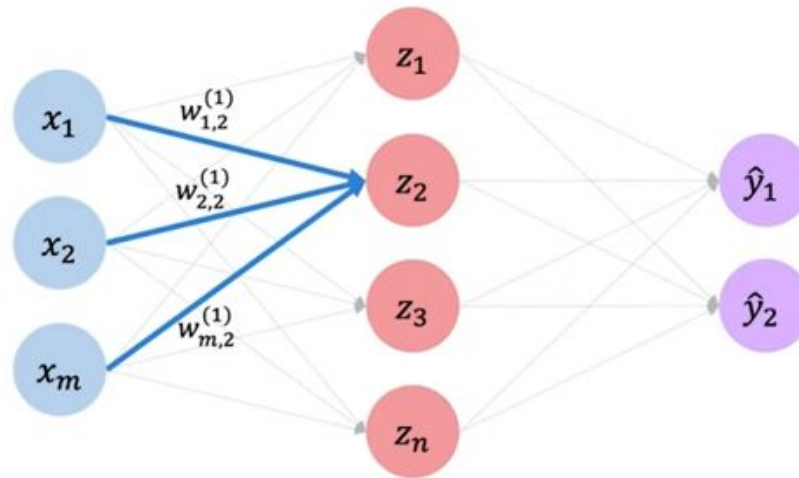

- now let's take a look at what's called a single layered neural network this is one we have a single hidden layer between our inputs and our outputs
- this layer is called the hidden layer because unlike an input layer and an output layer the states of this hidden layer are typically unobserved they're hidden to some extent they're not strictly enforced either
- and since we have this transformation now from the input layer to the hidden layer and from the hidden layer to the output layer each of these layers are going to have their own specified weight matrices we'll call w_1 the weight matrices for the first layer and w_2 the weight matrix for the second layer
- if we take a zoomed in look at one of the neurons in this hidden layer let's take for example z_2 for example this is the exact same perceptron that we saw before we can compute its output again using the exact same story taking all of its inputs x_1 through x_m applying a dot product with the weights adding a bias and that gives us z_2
- if we look at a different neuron let's suppose z_3 we'll get a different value here because the weights leading to z_3 are probably different than those leading to z_2

Single Layer Neural Network



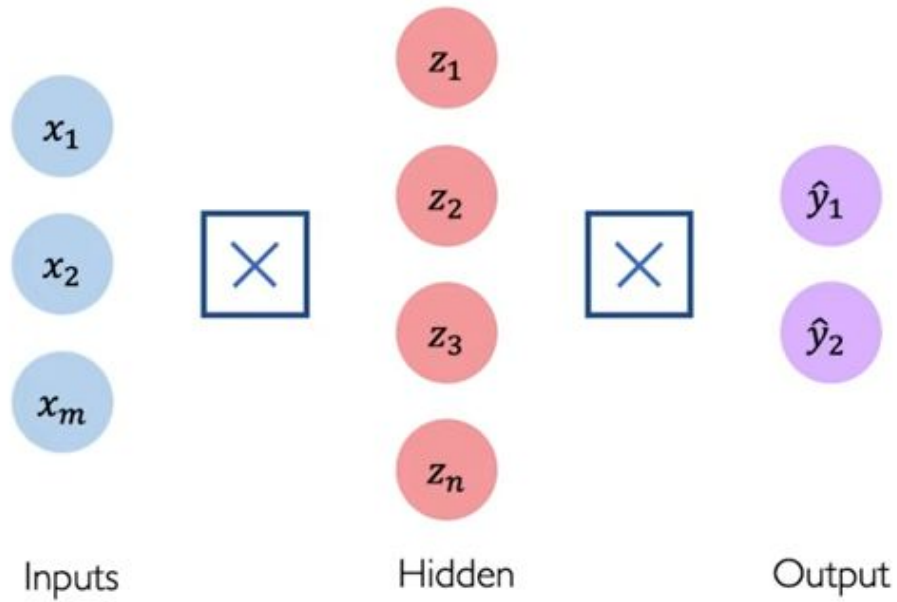
$$z_i = w_{0,i}^{(1)} + \sum_{j=1}^m x_j w_{j,i}^{(1)} \quad \hat{y}_i = g \left(w_{0,i}^{(2)} + \sum_{j=1}^{d_1} g(z_j) w_{j,i}^{(2)} \right)$$

Single Layer Neural Network



$$\begin{aligned} z_2 &= w_{0,2}^{(1)} + \sum_{j=1}^m x_j w_{j,2}^{(1)} \\ &= w_{0,2}^{(1)} + x_1 w_{1,2}^{(1)} + x_2 w_{2,2}^{(1)} + x_m w_{m,2}^{(1)} \end{aligned}$$

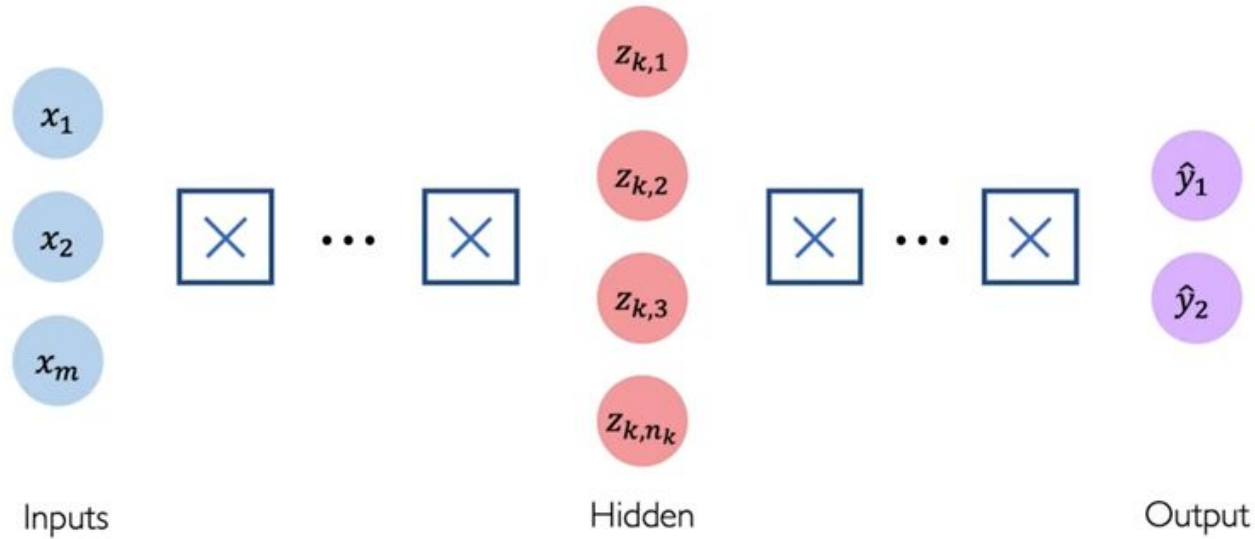
Multi Output Perceptron



```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(n),
    tf.keras.layers.Dense(2)
])
```

Deep Neural Network



$$z_{k,i} = w_{0,i}^{(k)} + \sum_{j=1}^{n_{k-1}} g(z_{k-1,j}) w_{j,i}^{(k)}$$



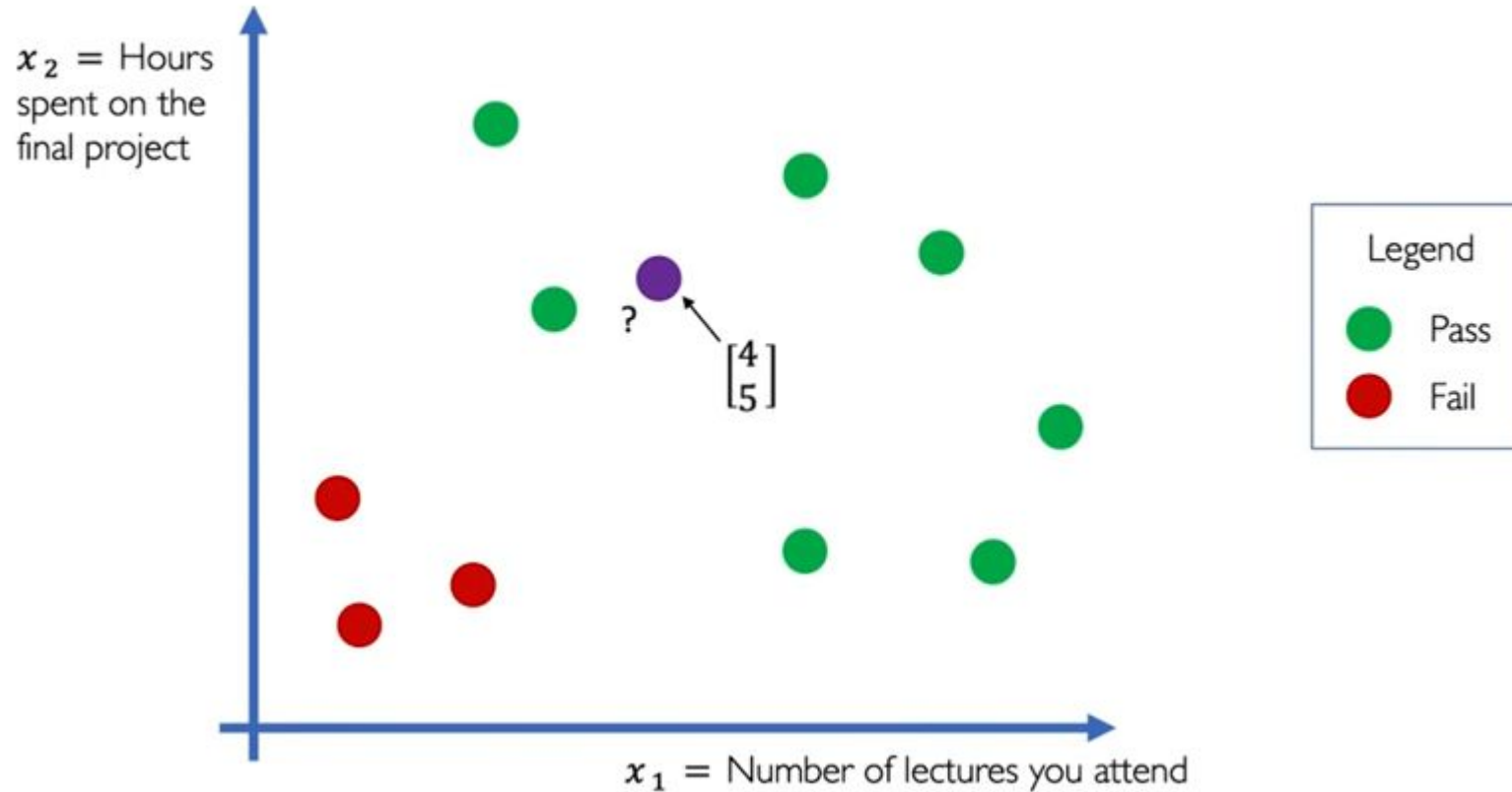
```
import tensorflow as tf

model = tf.keras.Sequential([
    tf.keras.layers.Dense(n1),
    tf.keras.layers.Dense(n2),
    ...,
    tf.keras.layers.Dense(2)
])
```

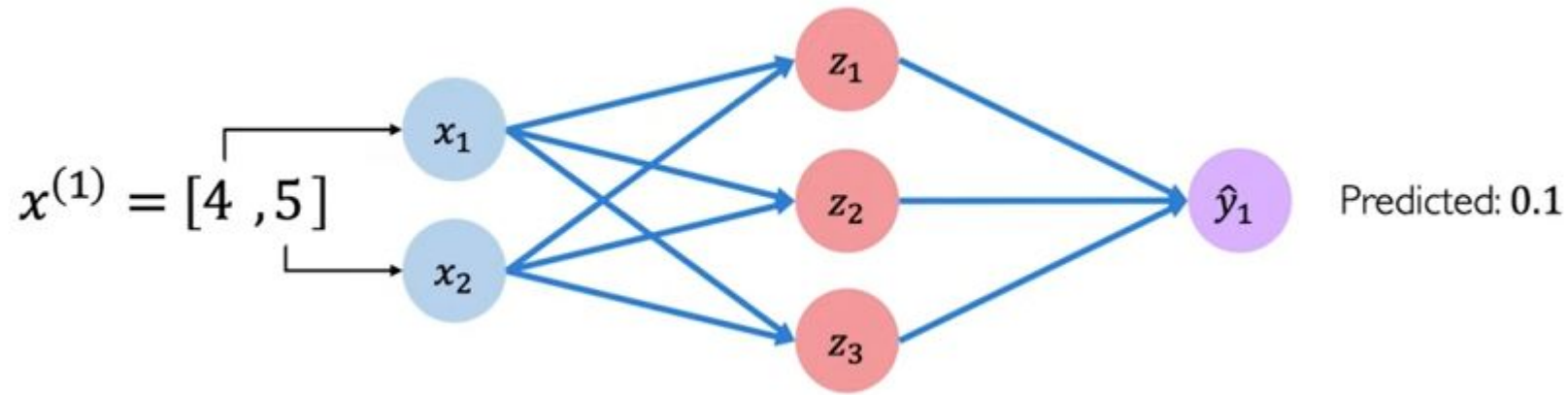
Applying Neural Network

- Example Problem:
 - Will I pass this class?
 - Lets start with a sample two feature model
 - X_1 = No. of lectures you attended
 - X_2 =Hrs. spent in the final project

Example Problem: Will I pass this class?



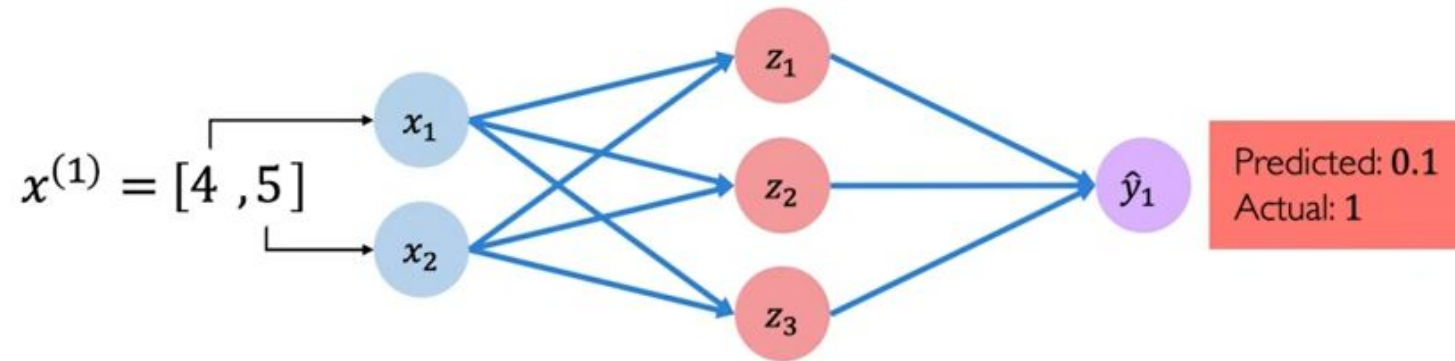
Example Problem: Will I pass this class?



- that's not great but the reason is because that this model uh was never actually trained
- it's basically just a a baby it's never seen any data even though you have seen the data it hasn't
- seen any data and more importantly you haven't told the model how to interpret this data it needs
- to learn about this problem first it knows nothing about this class or final projects or any of that
- so one of the most important things to do this is actually you have to tell the model
- when it's able when it is making bad predictions in order for it to be able to correct itself

Quantifying Loss

The **loss** of our network measures the cost incurred from incorrect predictions

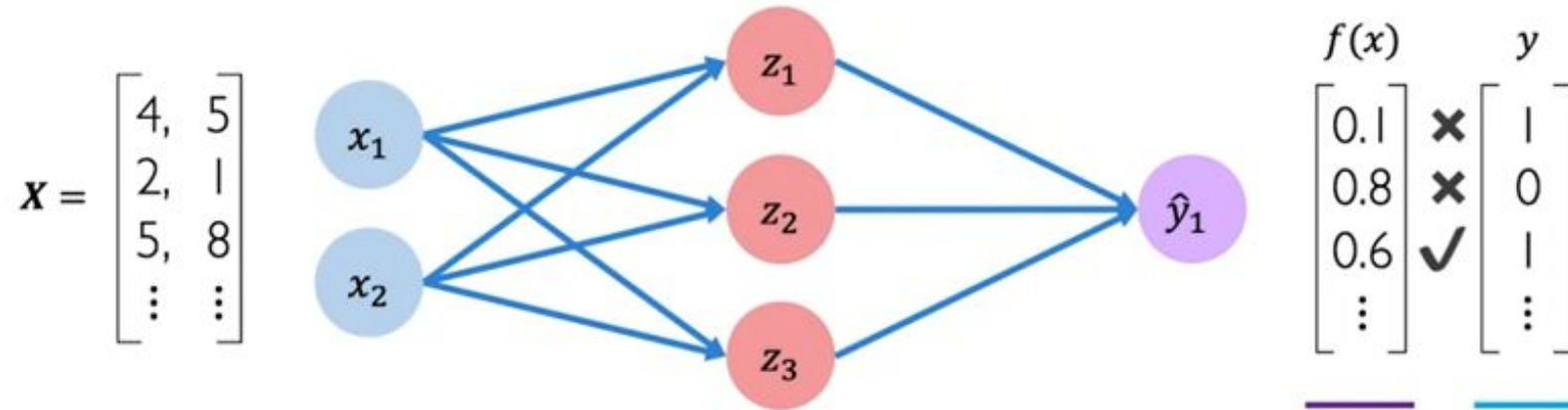


$$\mathcal{L}(\underbrace{f(x^{(i)}; \mathbf{W})}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

- now the loss of a neural network actually defines exactly this it defines how wrong a prediction was
- so it takes as input the predicted outputs and the ground truth outputs now if those
- two things are very far apart from each other then the loss will be very large on the other
- hand the closer these two things are from each other the smaller the loss and the more accurate

Empirical Loss

The **empirical loss** measures the total loss over our entire dataset



Also known as:

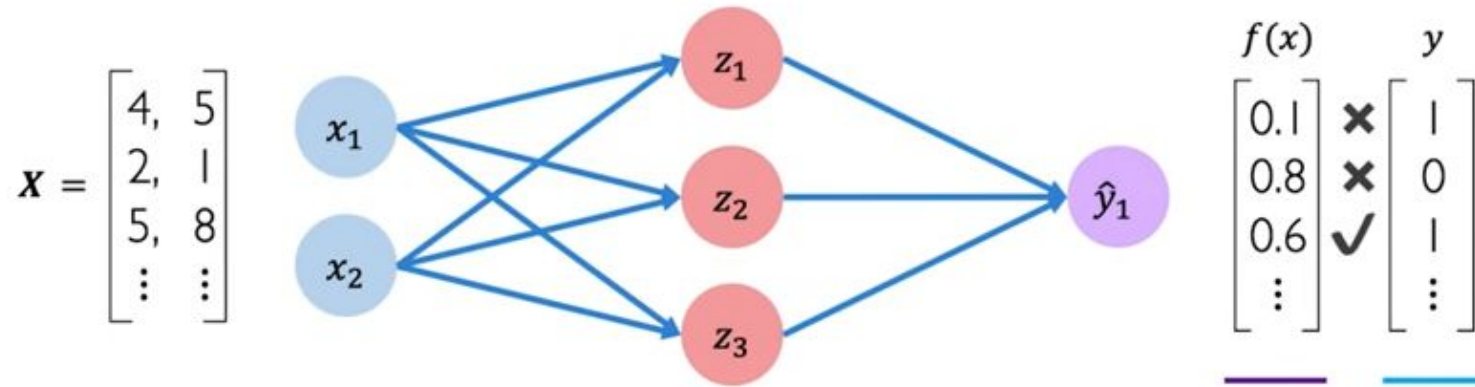
- Objective function
- Cost function
- Empirical Risk

$$J(W) = \frac{1}{n} \sum_{i=1}^n \mathcal{L}(\underbrace{f(x^{(i)}; W)}_{\text{Predicted}}, \underbrace{y^{(i)}}_{\text{Actual}})$$

- now let's assume we have not just the data from one student but as we have in this case
- the data from many students we now care about not just how the model did on predicting just
- one prediction but how it did on average across all of these students this is what
- we call the empirical loss and it's simply just the mean or the average
- of every loss from each individual example or each individual student
- when training a neural network we want to find a network that minimizes
- the empirical loss between our predictions and the true output

Binary Cross Entropy Loss

Cross entropy loss can be used with models that output a probability between 0 and 1



$$J(\mathbf{W}) = -\frac{1}{n} \sum_{i=1}^n \underbrace{y^{(i)}}_{\text{Actual}} \log \left(\underbrace{f(x^{(i)}; \mathbf{W})}_{\text{Predicted}} \right) + (1 - \underbrace{y^{(i)}}_{\text{Actual}}) \log \left(1 - \underbrace{f(x^{(i)}; \mathbf{W})}_{\text{Predicted}} \right)$$

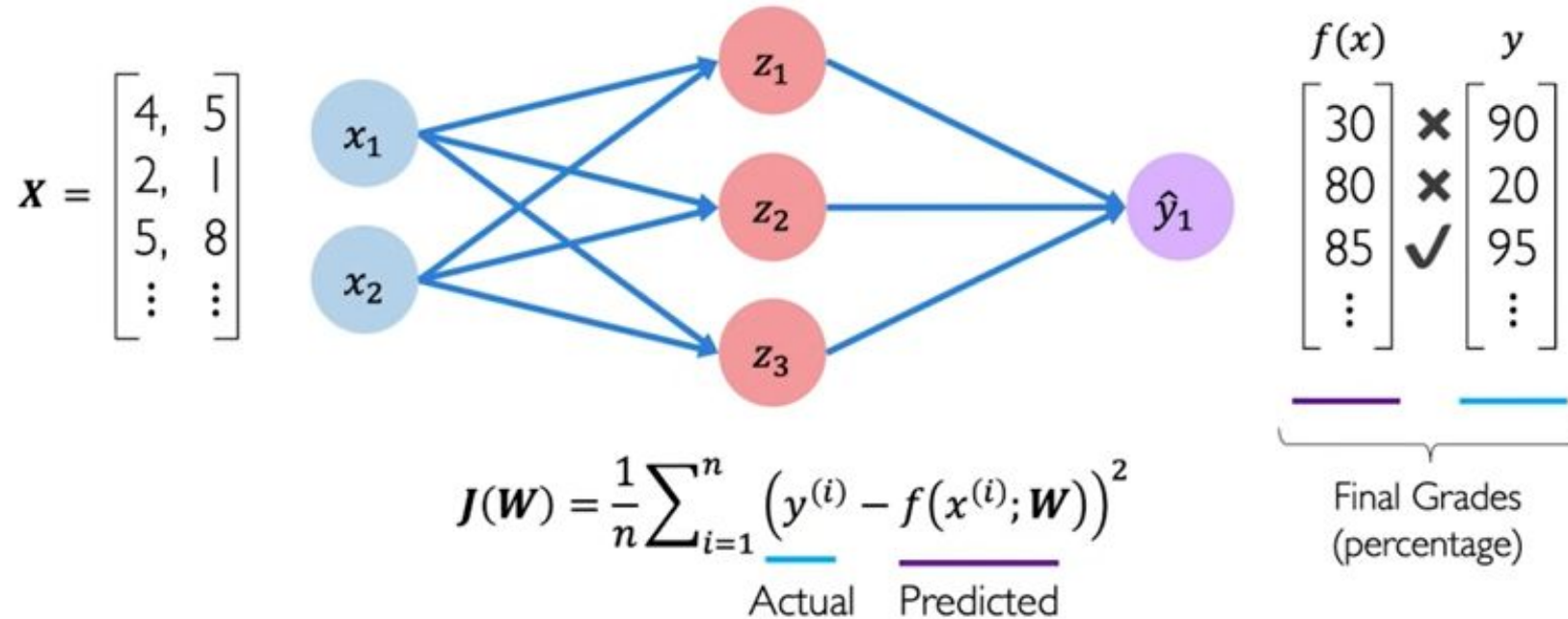


```
loss = tf.reduce_mean( tf.nn.softmax_cross_entropy_with_logits(y, predicted) )
```

- now if we look at the problem of binary classification where the neural network
- like we want to do in this case is supposed to answer either yes or no one or zero we can use
- what is called a soft max cross entropy loss now the soft max cross entropy loss is actually built
- is actually written out here and it's defined by actually what's called the
- cross entropy between two probability distributions it measures how far apart
- the ground truth probability distribution is from the predicted probability distribution

Mean Squared Error Loss

Mean squared error loss can be used with regression models that output continuous real numbers



```
loss = tf.reduce_mean( tf.square(tf.subtract(y, predicted)) )  
loss = tf.keras.losses.MSE( y, predicted )
```


- let's suppose instead of predicting binary outputs will i pass this class or will i not
- pass this class instead you want to predict the final grade as a real number not a probability
- or as a percentage we want the the grade that you will get in this class now in this case because
- the type of the output is different we also need to use a different loss here because our outputs
- are no longer 0 1 but they can be any real number they're just the grade that you're going to get on
- on the final class so for example here since this is a continuous variable the grade we want to use
- what's called the mean squared error this measures just the the squared error the squared difference
- between our ground truth and our predictions again averaged over the entire data set

Training NNs

Loss Optimization

We want to find the network weights that *achieve the lowest loss*

$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} \frac{1}{n} \sum_{i=1}^n \mathcal{L}(f(x^{(i)}; \mathbf{W}), y^{(i)})$$

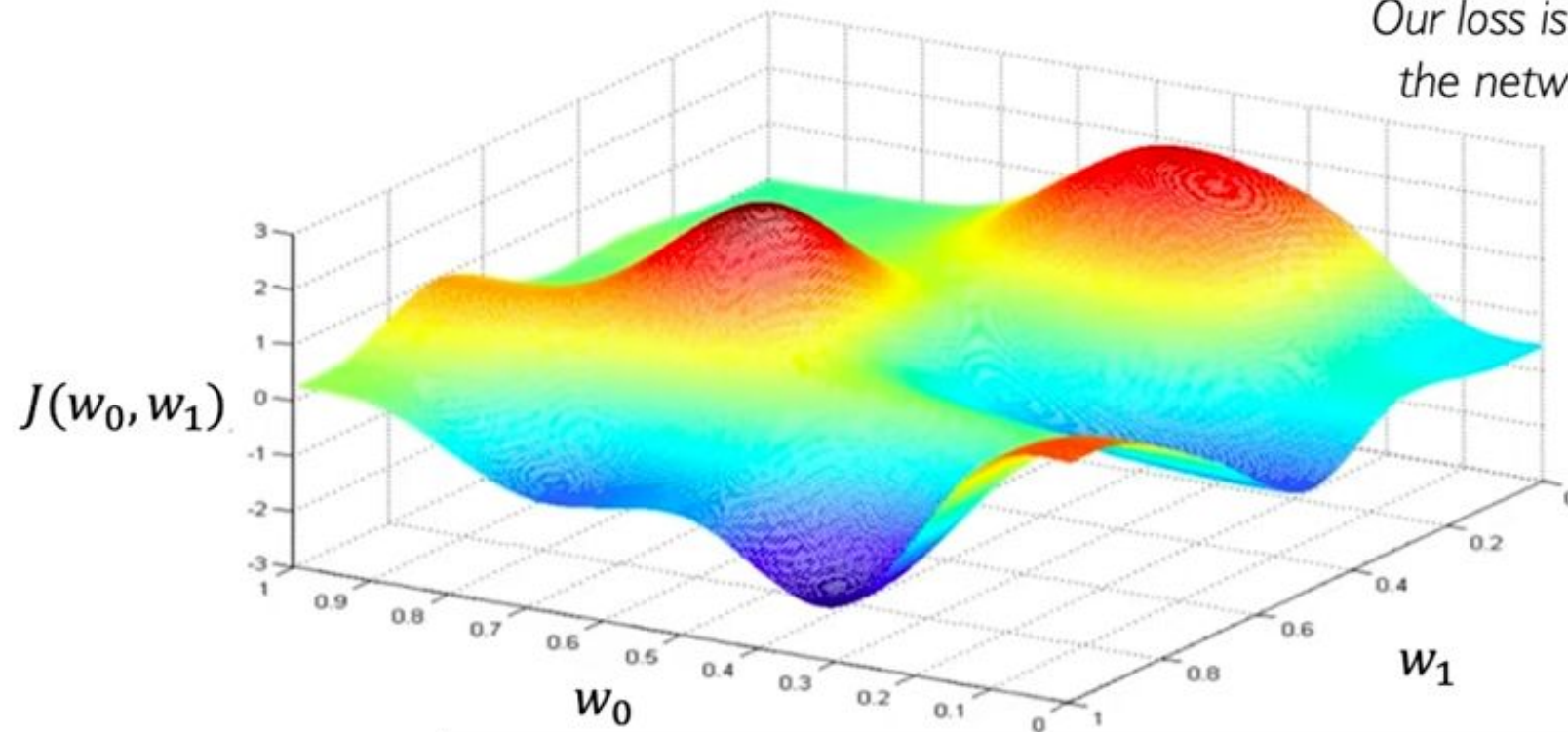
$$\mathbf{W}^* = \operatorname{argmin}_{\mathbf{W}} J(\mathbf{W})$$

- okay great so now we've seen two loss functions one for classification binary outputs as well as
- regression continuous outputs and the problem now i think that we need to start asking ourselves is
- how can we take that loss function we've seen our loss function we've seen our network now we have
- to actually understand how can we put those two things together how can we use our loss function
- to train the weights of our neural network such that it can actually learn that problem
- well what we want to do is actually find the weights of the neural network
- that will minimize the loss of our data set that essentially means
- that we want to find the w s in our neural network that minimize J of w $J(w)$ is our empirical cost
- function that we saw in the previous slides that average loss over each data point in the data set
- now remember that w capital w is simply a collection of all of the weights in our
- neural network not just from one layer but from every single layer so that's
- w_0 from the zeroth layer to the first layer to the second layer all concatenate into one
- in this optimization problem we want to optimize all of the w 's to minimize this empirical loss

Loss Optimization

$$W^* = \operatorname{argmin}_W J(W)$$

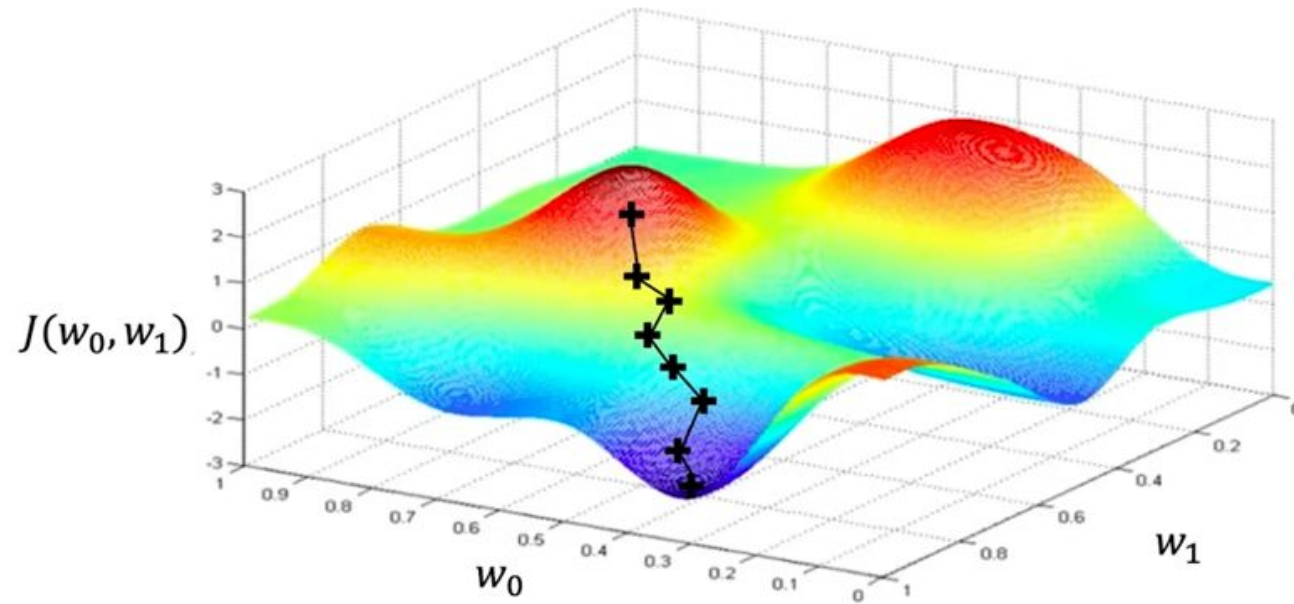
Remember:
*Our loss is a function of
the network weights!*



- now remember our loss function is just a simple function of our weights if we have
- only two weights we can actually plot this entire lost landscape over this grid of weight so on the
- one axis on the bottom you can see weight number one and the other one you can see weight zero
- there's only two weights in this neural network very simple neural network so we can actually plot
- for every w_0 and w_1 what is the loss what is the error that we'd expect to see and obtain from this
- neural network now the whole process of training a neural network optimizing it is to find the lowest
- point in this lost landscape that will tell us our optimal w_0 and w_1 now how can we do that the first
- thing we have to do is pick a point so let's pick any w_0 w_1 starting from this point we can compute
- the gradient of the landscape at that point now the gradient tells us the direction of
- highest or steepest ascent okay so that tells us which way is up
- okay if we compute the gradient of our loss with respect to our weights that's
- the derivative our gradient for the loss with respect to the weights that tells
- us the direction of which way is up on that lost landscape from where we stand right now
- instead of going up though we want to find the lowest loss so let's take the negative of our
- gradient and take a small step in that direction okay and this will move us a little bit closer
- to the lowest point and we just keep repeating this now we compute the gradient at this point

Gradient Descent

Repeat until convergence



- this algorithm
- is also known as gradient descent so we start by initializing all of our weights randomly and we
- start and we loop until convergence we start from one of those weights our initial point
- we compute the gradient that tells us which way is up so we take a step in the opposite direction we
- take a small step here small is computed by multiplying our gradient by this factor η
- and we'll learn more about this factor later this factor is called the learning rate

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights



```
import tensorflow as tf

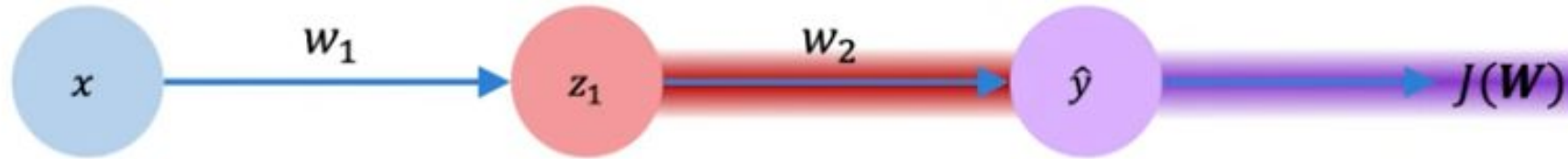
weights = tf.Variable([tf.random.normal()])

while True:    # loop forever
    with tf.GradientTape() as g:
        loss = compute_loss(weights)
        gradient = g.gradient(loss, weights)

    weights = weights - lr * gradient
```

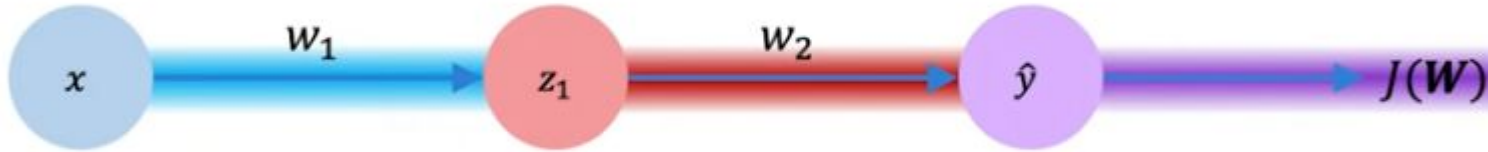

- so how does backpropagation work how do we compute this gradient
- let's start with a very simple neural network this is probably the simplest neural network in existence it only has one input one hidden neuron and one output computing the gradient of
- our loss J of w with respect to one of the weights in this case just w_2 for example tells us how much
- a small change in w_2 is going to affect our loss J so if we move around J infinitesimally small how
- will that affect our loss that's what the gradient is going to tell us of derivative of J of w_2 .
- so if we write out this derivative we can actually apply the chain rule to actually compute it
- so what does that look like specifically we can decompose that derivative into the
- derivative of J $\frac{dJ}{dy}$ multiplied by derivative of our output with respect to w_2

Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_2} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial w_2}}_{\text{red}}$$

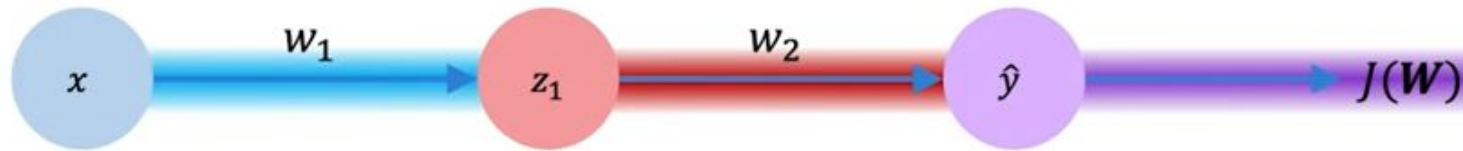
Computing Gradients: Backpropagation



$$\frac{\partial J(W)}{\partial w_1} = \underbrace{\frac{\partial J(W)}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial z_1}}_{\text{red}} * \underbrace{\frac{\partial z_1}{\partial w_1}}_{\text{blue}}$$

This is telling us how a small change in our weight is going to affect our loss so we can see if we increase our weight a small amount it will increase our loss \boxtimes we will want to decrease the weight to decrease our loss

Computing Gradients: Backpropagation

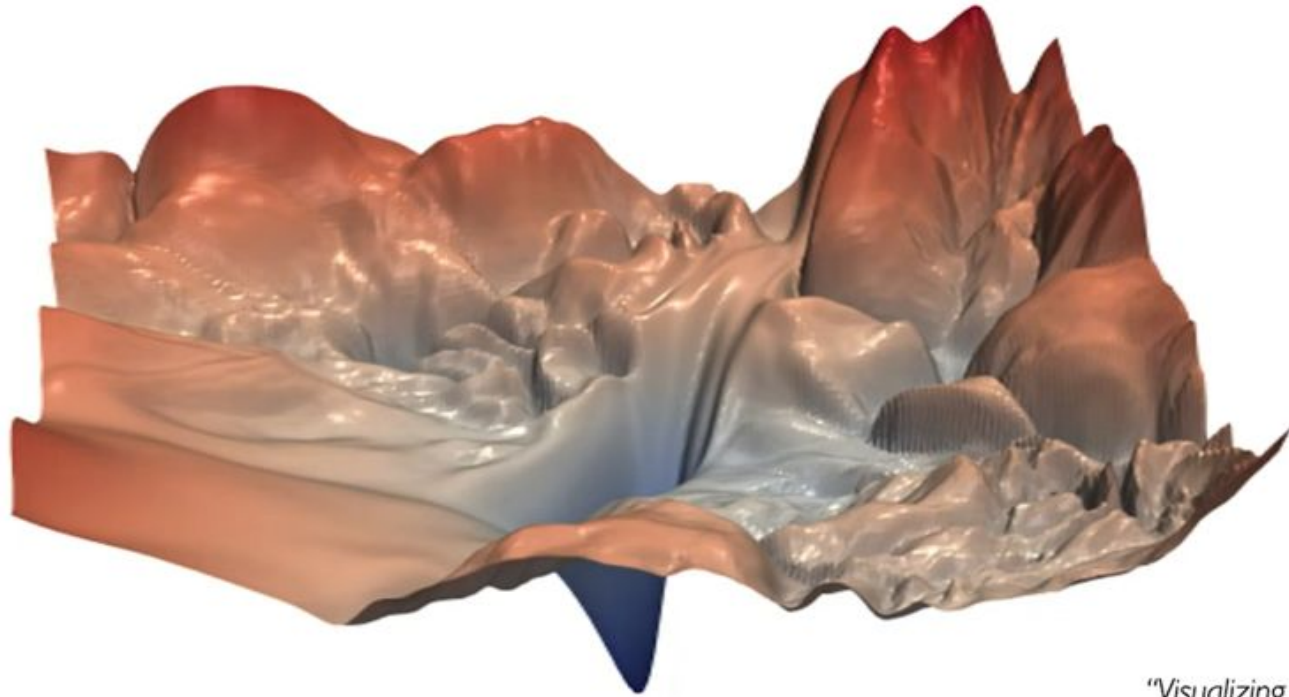


$$\frac{\partial J(\mathbf{W})}{\partial w_1} = \underbrace{\frac{\partial J(\mathbf{W})}{\partial \hat{y}}}_{\text{purple}} * \underbrace{\frac{\partial \hat{y}}{\partial z_1}}_{\text{red}} * \underbrace{\frac{\partial z_1}{\partial w_1}}_{\text{blue}}$$

Repeat this for **every weight in the network** using gradients from later layers

NNs in Practice: Optimization

Training Neural Networks is Difficult



"Visualizing the loss landscape of neural nets". Dec 2017.

Loss Functions Can Be Difficult to Optimize

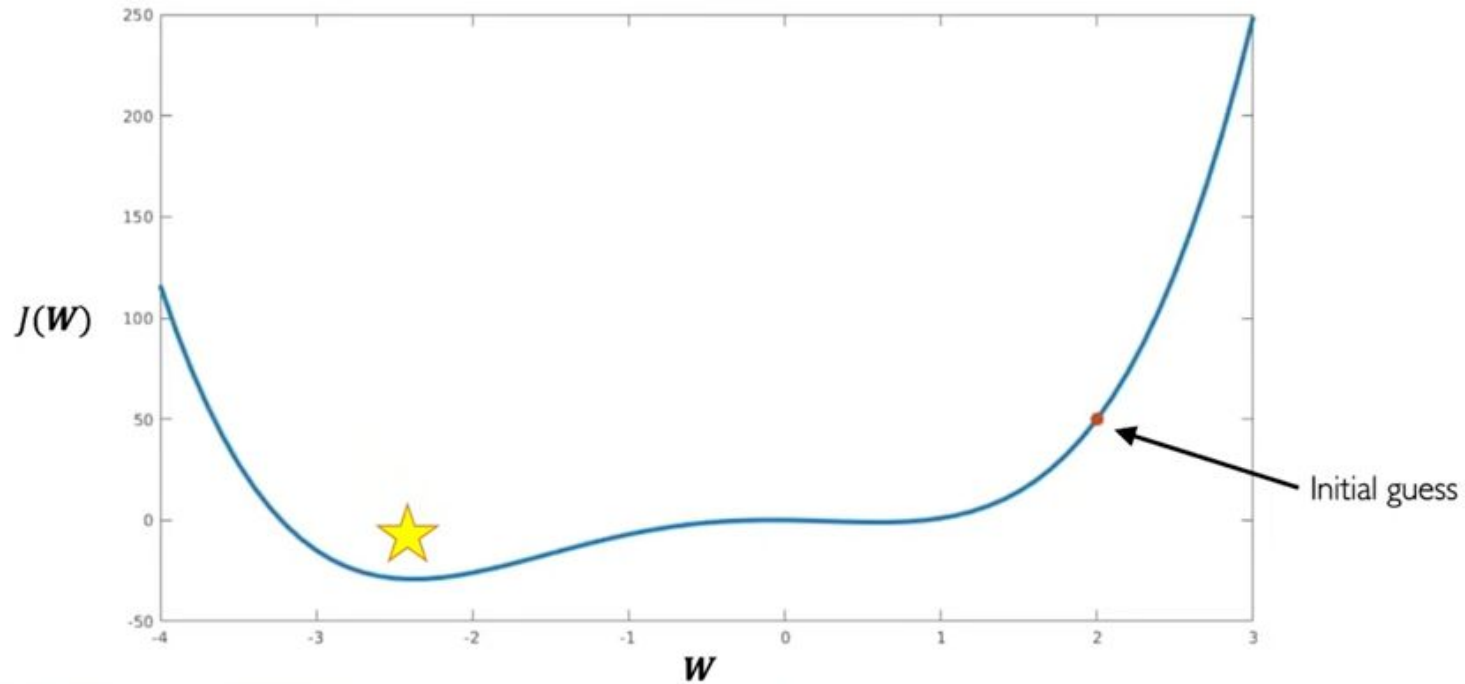
Remember:

Optimization through gradient descent

$$\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$$

Setting the Learning Rate

Small learning rate converges slowly and gets stuck in false local minima



How to deal with this?

Idea 1:

Try lots of different learning rates and see what works “just right”

Idea 2:






Do something smarter!

Design an adaptive learning rate that “adapts” to the landscape

Adaptive Learning Rates

- Learning rates are no longer fixed
- Can be made larger or smaller depending on:
 - how large gradient is
 - how fast learning is happening
 - size of particular weights
 - etc...

Gradient Descent Algorithms

Algorithm	TF Implementation	Reference
• SGD	 <code>tf.keras.optimizers.SGD</code>	Kiefer & Wolfowitz. "Stochastic Estimation of the Maximum of a Regression Function." 1952.
• Adam	 <code>tf.keras.optimizers.Adam</code>	Kingma et al. "Adam: A Method for Stochastic Optimization." 2014.
• Adadelata	 <code>tf.keras.optimizers.Adadelata</code>	Zeiler et al. "ADADELTA: An Adaptive Learning Rate Method." 2012.
• Adagrad	 <code>tf.keras.optimizers.Adagrad</code>	Duchi et al. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." 2011.
• RMSProp	 <code>tf.keras.optimizers.RMSProp</code>	

Additional details: <http://ruder.io/optimizing-gradient-descent/>

Putting it all together

```
import tensorflow as tf

model = tf.keras.Sequential([...])

# pick your favorite optimizer
optimizer = tf.keras.optimizer.SGD()

while True: # loop forever

    # forward pass through the network
    prediction = model(x)

    with tf.GradientTape() as tape:
        # compute the loss
        loss = compute_loss(y, prediction)

    # update the weights using the gradient
    grads = tape.gradient(loss, model.trainable_variables)
    optimizer.apply_gradients(zip(grads, model.trainable_variables))
```



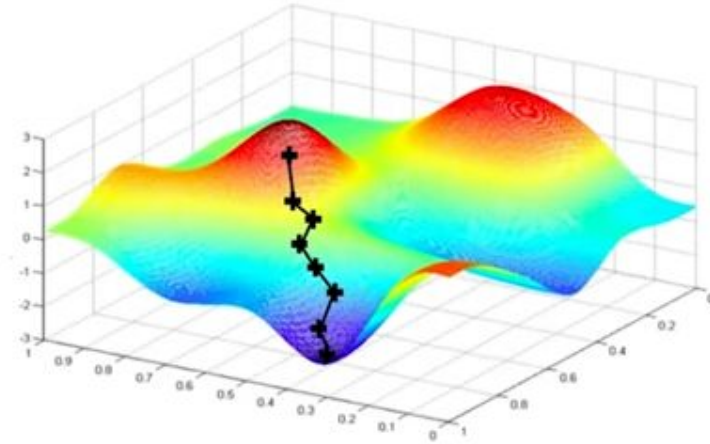
Can replace with any
TensorFlow optimizer!

Minibatches

Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Compute gradient, $\frac{\partial J(W)}{\partial W}$
4. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(W)}{\partial W}$
5. Return weights

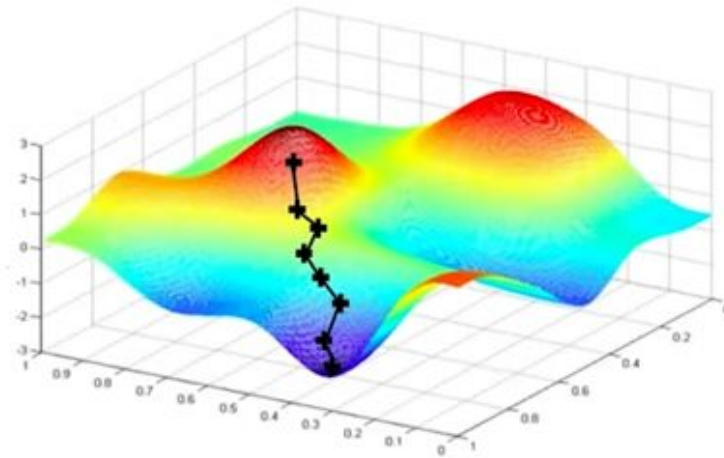


Can be very
computationally
intensive to compute!

Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
 3. Pick single data point i
 4. Compute gradient, $\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}}$
 5. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
6. Return weights

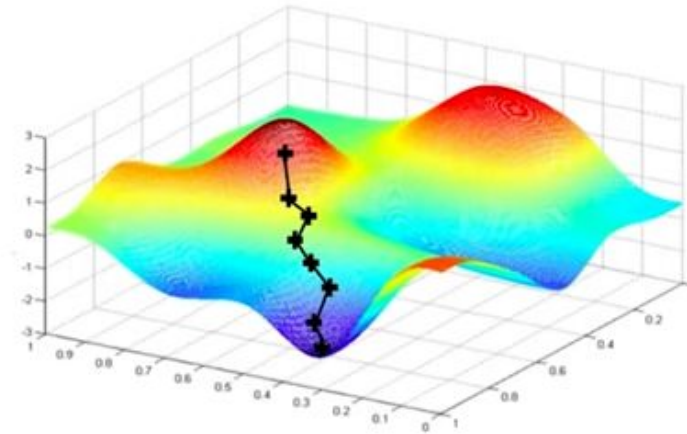


Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick single data point i
4. Compute gradient, $\frac{\partial J_i(\mathbf{W})}{\partial \mathbf{W}}$
5. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
6. Return weights

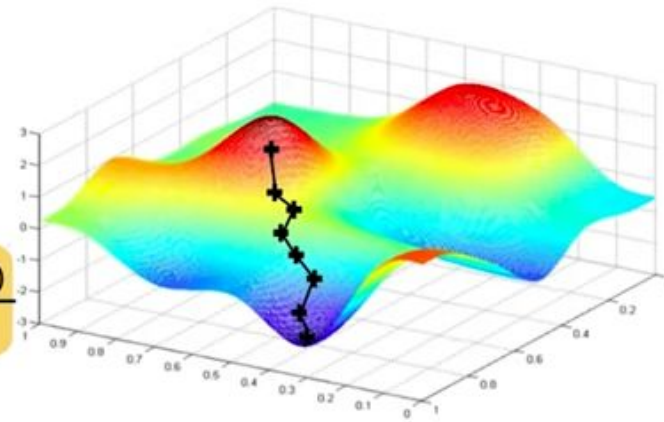
Easy to compute but
very noisy (stochastic)!



Stochastic Gradient Descent

Algorithm

1. Initialize weights randomly $\sim \mathcal{N}(0, \sigma^2)$
2. Loop until convergence:
3. Pick batch of B data points
4. Compute gradient, $\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = \frac{1}{B} \sum_{k=1}^B \frac{\partial J_k(\mathbf{W})}{\partial \mathbf{W}}$
5. Update weights, $\mathbf{W} \leftarrow \mathbf{W} - \eta \frac{\partial J(\mathbf{W})}{\partial \mathbf{W}}$
6. Return weights



Fast to compute and a much better estimate of the true gradient!

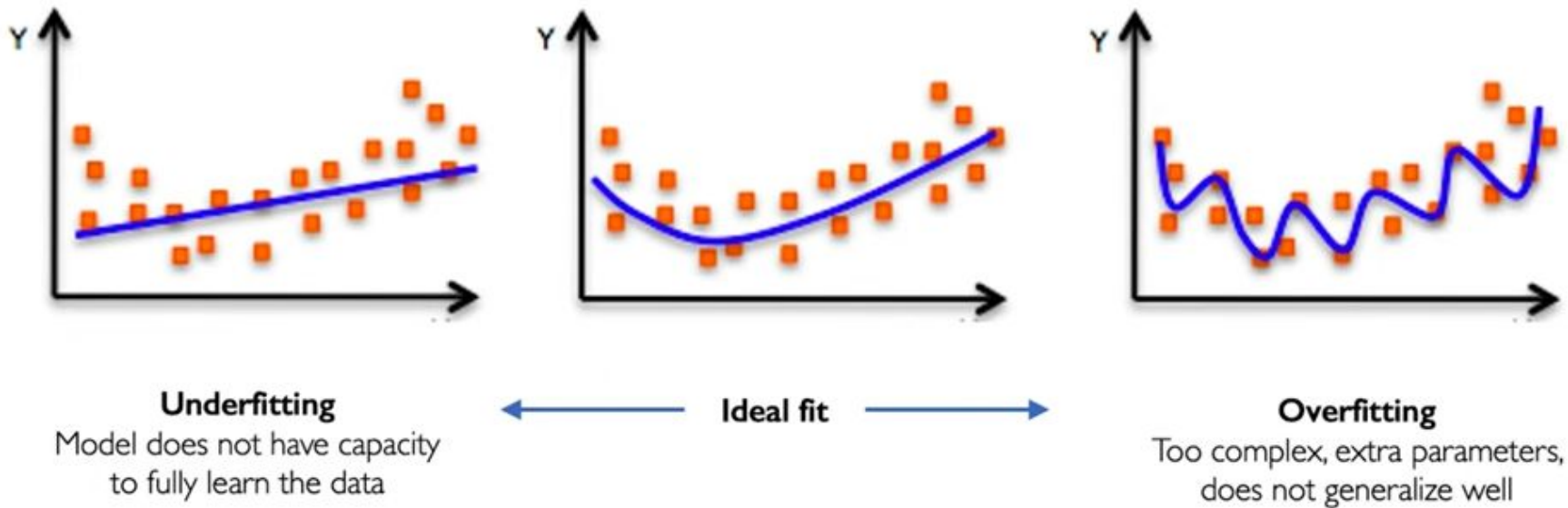
Mini-batches lead to fast training!

Can parallelize computation + achieve significant speed increases on GPU's

- in machine learning we want to learn a model
- that accurately describes our test data not the training data even though we're optimizing this
- model based on the training data what we really want is for it to perform well on the test data

Overfitting and Underfitting

The Problem of Overfitting



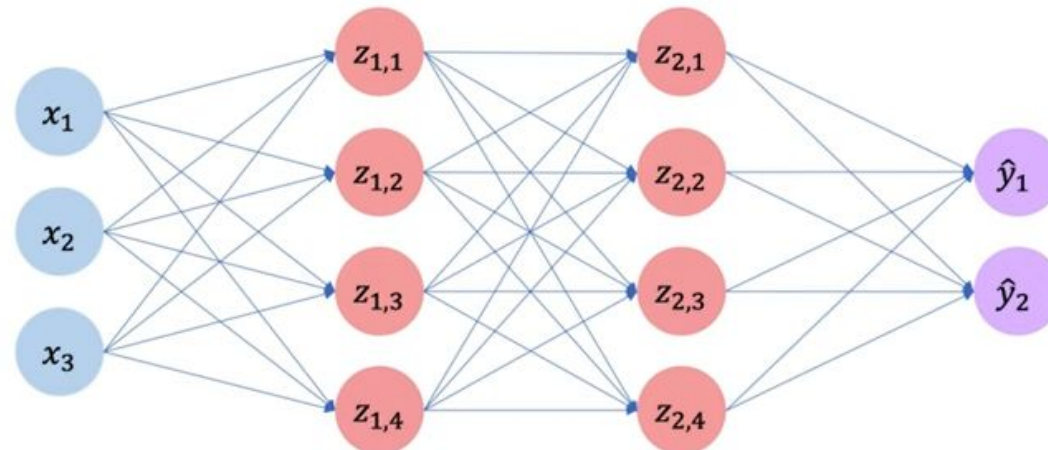
- now to address this problem let's talk about
- regularization how can we make sure that our models do not end up over fit because neural
- networks do have a ton of parameters how can we enforce some form of regularization
- to them now what is regularization regularization is a technique that constrains our optimization
- problems such that we can discourage these complex models from actually being learned and overfit
- right so again why do we need it we need it so that our model can generalize to this unseen
- data set and in neural networks we have many techniques for actually imposing regularization
- onto the model

Regularization

- Improve generalization of our model on unseen data

Regularization I: Dropout


- During training, randomly set some activations to 0

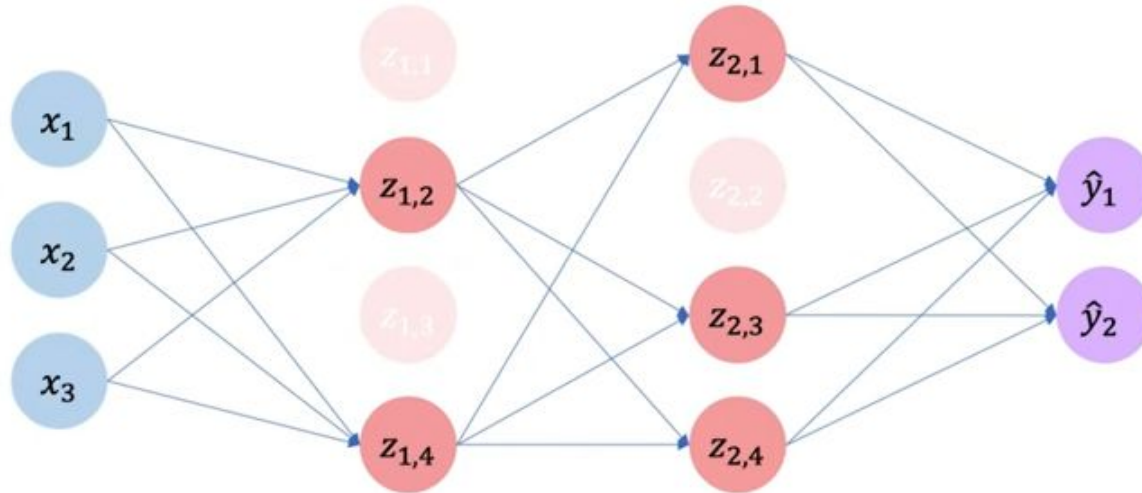


- most popular forms of regularization in deep learning and it's very simple let's
- revisit this picture of a neural network this is a two layered neural network two hidden layers
- and in dropout during training all we simply do is randomly set some of the activations here
- to zero with some probability so what we can do is let's say we pick our probability to be 50 or 0.5
- we can drop randomly for each of the activations 50 of those neurons this is extremely powerful as
- it lowers the capacity of our neural network so that they have to learn to perform better on test
- sets because sometimes on training sets it just simply cannot rely on some of those parameters
- so it has to be able to be resilient to that kind of dropout it also means that
- they're easier to train because at least on every forward pass iterations we're training only 50
- of the weights and only 50 of the gradients so that also cuts our uh gradient computation time
- down in by a factor of two so because now we only have to compute half the number of
- neuron gradients

Regularization I: Dropout

- During training, randomly set some activations to 0
 - Typically 'drop' 50% of activations in layer
 - Forces network to not rely on any 1 node

 `tf.keras.layers.Dropout(p=0.5)`

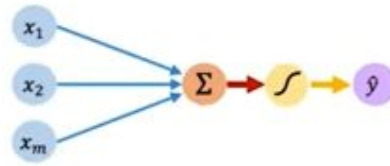


- now on every iteration we dropped out on the previous iteration fifty percent of
- neurons but on the next iteration we're going to drop out a different set of fifty 50 of the
- neurons a different set of neurons and this gives the network it basically forces the network to
- learn how to take different pathways to get to its answer and it can't rely on any one pathway too
- strongly and overfit to that pathway this is a way to really force it to generalize to this new data

Core Foundation Review

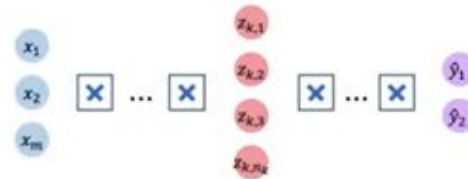
The Perceptron

- Structural building blocks
- Nonlinear activation functions



Neural Networks

- Stacking Perceptrons to form neural networks
- Optimization through backpropagation



Training in Practice

- Adaptive learning
- Batching
- Regularization

