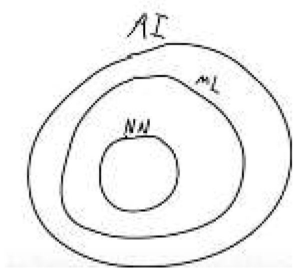
Intro to Neural Nets.ipynb - https://colab.research.google.com/github/google/eng-edu/blob/main/ml/cc/exercises/intro_to_neural_nets.ipynb?utm_source=mlcc&utm_campaign=colab-external&utm_medium=referral&utm_content=intro_to_nn_tf2-colab&hl=en

https://www.freecodecamp.org/news/how-to-build-your-first-neural-network-to-predict-house-prices-with-keras-f8db83049159/

Machine Learning with Python

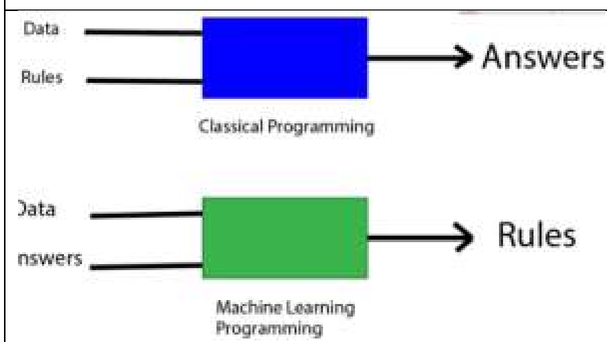
1.TensorFlow

- Machine Learning Fundamentals A - TensorFlow 2.0 Course - https://www.youtube.com/watch?v=KwL1qTR5MT8

<table>
<tr>
<td>
</td>
<td>AI - is the effort to automate intellectual tasks normally performed by humans

Early AI - used a predefined set of rules and coded into the computer.

No deep learning ML crazy algorithms happening.
AI simply simulating some intellectual human behavior.

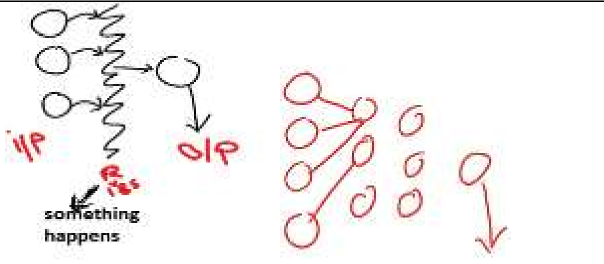
Now AI - has evolved into a much more complex field (ML, NN and other techniques)
</td>
</tr>
<tr>
<td>
</td>
<td>ML - is a part of AL. what is ML?

<li style="list-style:none">
AI used a predefined set of rules, means feed some data, go through the rules, analyze the data with the rules and produce the output
Example of chess, in check, looks at the sets of rules and then it moves to somewhere else.

ML is actually figuring out the rules for us instead of hard coding the rules

Means take the input, output data and figure out the rules for us

ML requires a lot(ton) of input data to really train a good model.
ML models do not have 100% accuracy (trying to simulate like a human, can make mistakes), which means may not necessarily get the correct answer every single time.
Our goal create ML models is to raise accuracy as high as possible
</td>
</tr>
</table>

	**NN (Neural networks or deep learning)** – It's a form of ML that uses a layered representation of data. • Input feed (bubbles) to this set of rules, something happens in here and get some output. • NN have more than 2(multiple) layers. An input layer (first layer of data), have some layers in between output layer, that are all connected together.
	**Example**, Students data set - info about students, midterm 1, 2 & final grade. • Student 1 - midterm 1 grade 70, midterm 2 grade 80 & final term grade 77 • How can I use this information to predict any one of these three columns. ○ So if I were given a student's midterm 1 and final grade, how could I predict their midterm 2 grade.

Features and labels

- **Features (Input information)** – Give midterm 1 and final grade to the model, to get some output is called features.
  - Training a model to look at midterm 1 and final grade, to make a new prediction
- **Labels (Output information** – midterm 2) trying to predict midterm 2.
  - Not have midterm 2 information & Not pass in the model, pass features (midterm 1 & final) to get the output of midterm 2

- **Intro to TensorFlow B - TensorFlow 2.0 Course -** https://www.youtube.com/watch?v=r9hRyGGjOgQ
  - **what is a tensor? It's a vector generalized to higher dimensions.**
  - **What is a vector? Any linear algebra or basic vector calculus. It is kind of a data point**. It doesn't necessarily have a certain coordinate.
  - For example, **2 dimensional data point (x & y value),**
  - A vector can have any amount of dimensions (1, 2, 3, 4 (image data), 5 (video data))
  - https://www.tensorflow.org/guide/tensor -
  - **A tensor** is a generalization of vectors and matrices to potentially higher dimensions. Internally, TensorFlow represented tensors as n dimensional arrays of base datatypes.
  - **Tensors**, are so important to **TensorFlow -** going to be working with manipulating and viewing

    Each tensor represents a partially defined computation that will eventually produce a value.

    TensorFlow is creating them & going to store partially defined computations in the graph.
    Later, build the graph and have the session running, run different parts of the graph (execute different tensors) and get different results from our tensors.

    Each tensor has a data type and a shape.

    Data type is kind of information is stored in the tensor – like numbers, strings, ….

    string = tf.Variable("this is a string", tf.string) – string tensor contains value and datatype

number = tf.Variable(123, tf.int16)
floating = tf.Variable(123.456, tf.float64)

Rank/Degree of tensors - the number of dimensions involved in the tensor.

rank0_tensor = tf.Variable("first ", tf.string) - S**calar, contains a single value, and no "axes".**

rank1_tensor = tf.Variable(["first ", “OK”], tf.string) - V**ector, a list of values & one axis.**

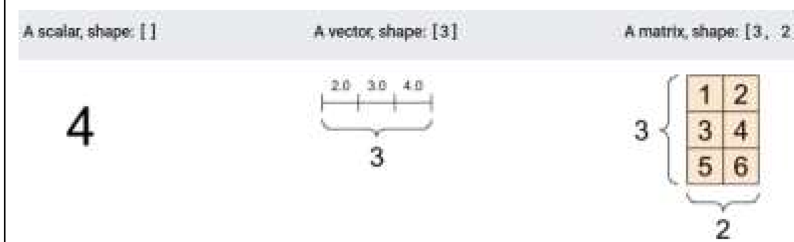
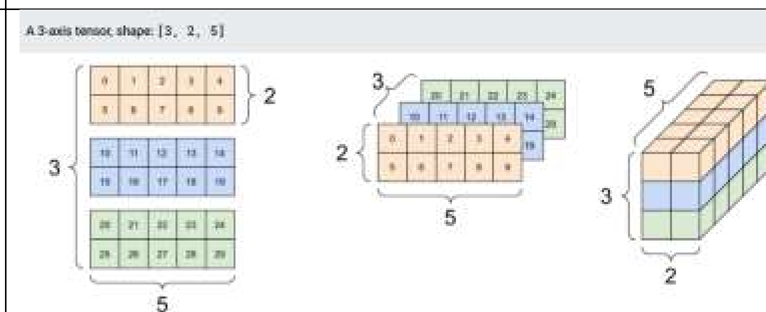
rank2_tensor = tf.Variable([["first",”OK”], ["second", “yes”]], tf.string) - **Matrix, has 2 axes**

tf.rank(rank2_tensor) => <tf.Tensor: shape=(), dtype=string, numpy=2> , **numpy=2 mean rank2**

Shapes of a tensor - how many items we have in each dimension.

Tensors have shapes. Some vocabulary:

Shape: The length (number of elements) of each of the axes of a tensor.
Rank: Number of tensor axes. A scalar has rank 0, a vector has rank 1, a matrix is rank 2.
Axis or Dimension: A particular dimension of a tensor.
Size: The total number of items in the tensor, the product of the shape vector's elements.

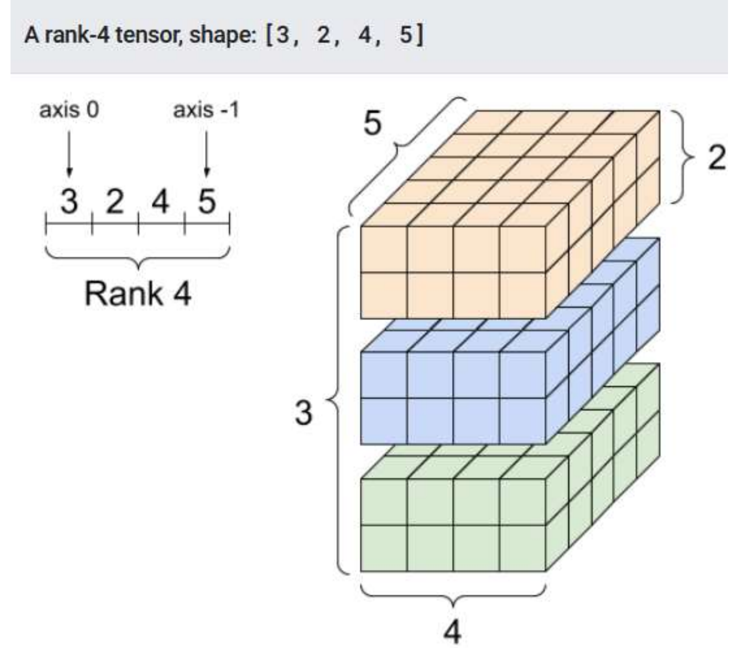
rank_0_tensor = tf.constant(4) => print(rank_0_tensor) => tf.Tensor(4, shape=(), dtype=int32) rank_1_tensor = tf.constant([2.0, 3.0, 4.0]) rank_2 = tf.constant([[1, 2], [3, 4], [5, 6]], dtype=tf.float16)	
rank_3_tensor = tf.constant([[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]], [[10, 11, 12, 13, 14], [15, 16, 17, 18, 19]], [[20, 21, 22, 23, 24], [25, 26, 27, 28, 29]],])	

```
rank_4 = tf.zeros([3, 2, 4, 5])
```

A tensor shape is like a vector.   A 4-axis tensor

```
print("Type of every element:", rank_4.dtype)
print("Number of axes:", rank_4.ndim)
print("Shape of tensor:", rank_4.shape)
print("Elements along axis 0 of tensor:", rank_4.shape[0])
print("Elements along the last axis of tensor:",
rank_4.shape[-1])
print("Total number of elements (3*2*4*5): ",
tf.size(rank_4).numpy())
```

```
Type of every element: <dtype: 'float32'>
Number of axes: 4
Shape of tensor: (3, 2, 4, 5)
Elements along axis 0 of tensor: 3
Elements along the last axis of tensor: 5
Total number of elements (3*2*4*5):  120
```



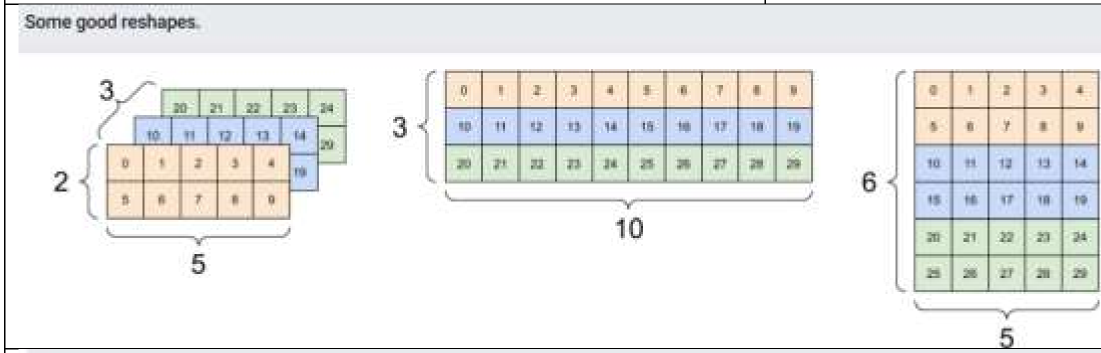
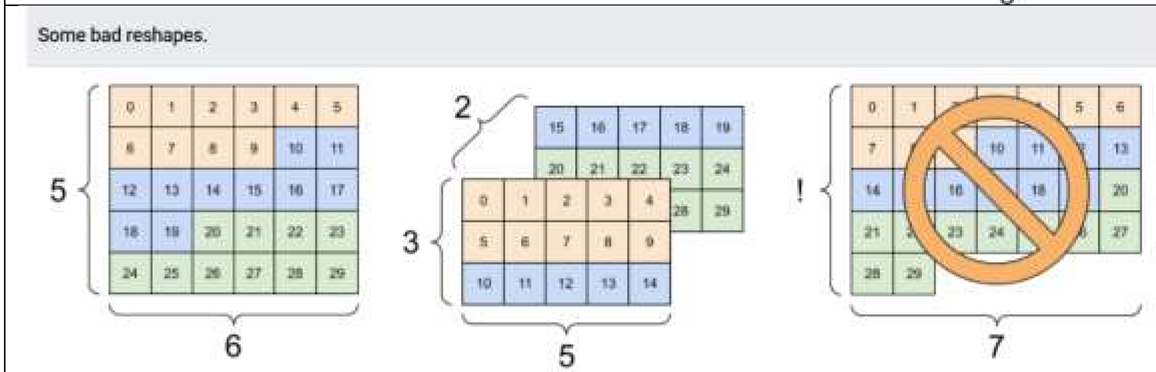
Reshape/Changing/Manipulating shape

```
[ ]  tensor1 = tf.ones([1,2,3])  # tf.ones() creates a shape [1,2,3] tensor full of ones
     tensor2 = tf.reshape(tensor1, [2,3,1])  # reshape existing data to shape [2,3,1]
     tensor3 = tf.reshape(tensor2, [3, -1])  # -1 tells the tensor to calculate the size of the dimension in that place
                                             # this will reshape the tensor to [3,3]

     # The numer of elements in the reshaped tensor MUST match the number in the original
```

```
tf.Tensor(
[[[1. 1. 1.]
  [1. 1. 1.]]], shape=(1, 2, 3), dtype=float32)
tf.Tensor(
[[[1.]
  [1.]
  [1.]]

 [[1.]
  [1.]
  [1.]]], shape=(2, 3, 1), dtype=float32)
tf.Tensor(
[[1. 1.]
 [1. 1.]
 [1. 1.]], shape=(3, 2), dtype=float32)
```

```
%tensorflow_version 2.x
import tensorflow as tf
print(tf.version)

t = tf.zeros([5,5,5,5])

t = tf.reshape(t, [625])
```

rank_3_tensor = tf.constant([[[0, 1, 2, 3, 4], [5, 6, 7, 8, 9]], [[10, 11, 12, 13, 14], [15, 16, 17, 18, 19]], [[20, 21, 22, 23, 24], [25, 26, 27, 28, 29]],]) print(rank_3_tensor) tf.Tensor([[[0 1 2 3 4] [5 6 7 8 9]] [[10 11 12 13 14] [15 16 17 18 19]] [[20 21 22 23 24] [25 26 27 28 29]]], shape=(3, 2, 5), dtype=int32) **# A `-1` passed in the `shape` argument says "Whatever fits".**	tf.Tensor([0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29], shape=(30,), dtype=int32) Typically the only reasonable use of tf.reshape is to combine or split adjacent axes (or add/remove 1s). For this 3x2x5 tensor, reshaping to (3x2)x5 or 3x(2x5) are both reasonable things to do, as the slices do not mix: print(tf.reshape(rank_3_tensor, [3*2, 5]), "\n") print(tf.reshape(rank_3_tensor, [3, -1])) tf.Tensor([[0 1 2 3 4] [5 6 7 8 9] [10 11 12 13 14] [15 16 17 18 19] [20 21 22 23 24] [25 26 27 28 29]], shape=(6, 5), dtype=int32) tf.Tensor([[0 1 2 3 4 5 6 7 8 9] [10 11 12 13 14 15 16 17 18 19]

print(tf.reshape(rank_3_tensor, [-1]))	[20 21 22 23 24 25 26 27 28 29]], shape=(3, 10), dtype=int32)
	
	

Types of tensors – Variable, Constant, Placeholder and SparseTensor, a few other ones as well.

Except variable, all are immutable (value may not change during execution).

Evaluating tensors - create a session.

Sometimes need tensor object throughout our code, to do just use of default template

```
[ ]  with tf.Session() as sess:  # creates a session using the default graph
       tensor.eval()  # tensor will of course be the name of your tensor
```

- **Core Learning Algorithms A - TensorFlow 2.0 Course** - https://www.youtube.com/watch?v=u5lZURgcWnU

TensorFlow core learning algorithms, but not specific to TensorFlow, but they are used within there.

Linear regression

Classification

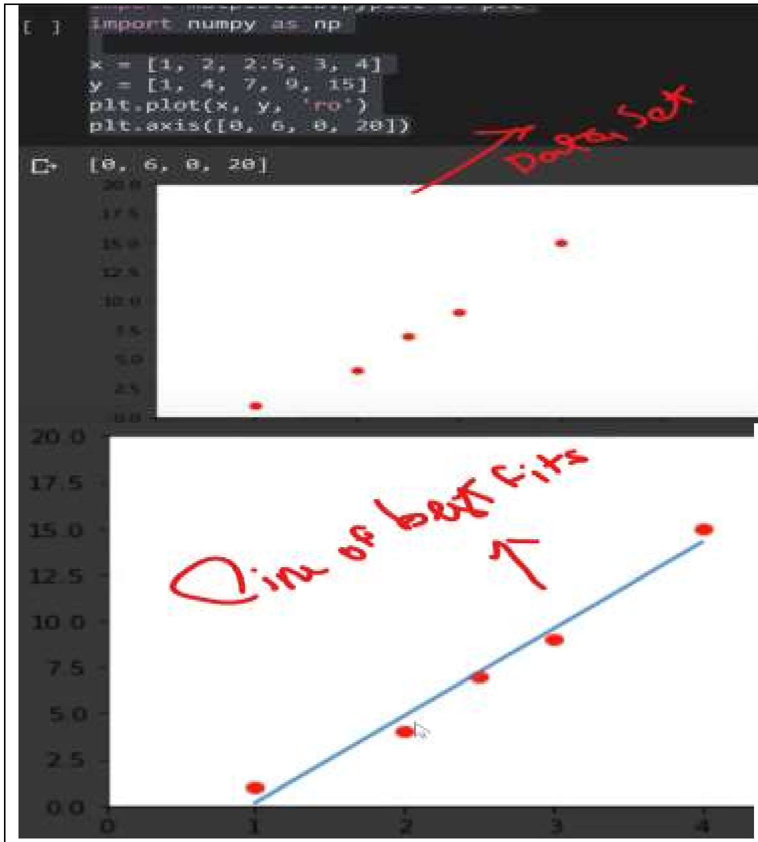
Clustering

Hidden Markov models

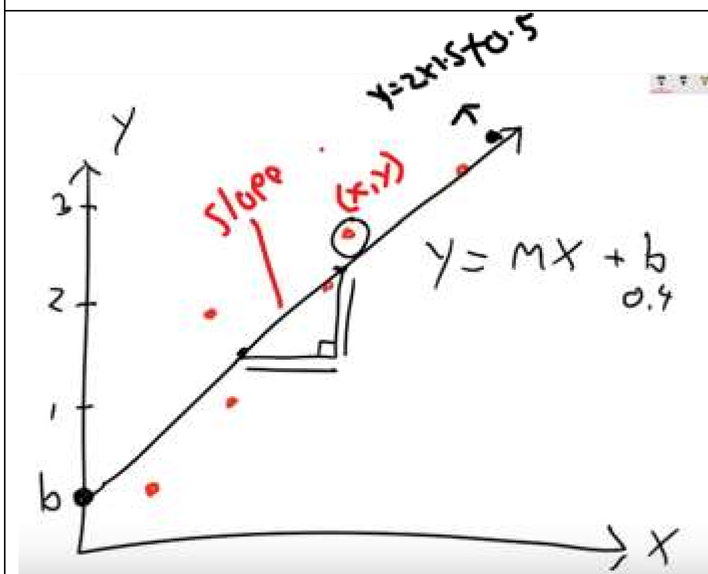
Now there is a ton time like 1000s of different machine learning algorithms.

Linear regression

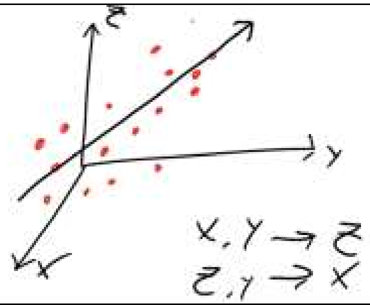
A linear correspondence between data points.



- Good example, plot a little graph & our data set
- use linear regression to come up with a model that can give us some good predictions for our data points.
- In this case, given some x value for a data point & predict the y value.
- See there's kind of some correspondence linearly for **these data points**.
- That means is draw something called a line of best fit through these data points
- Using this (**blue**) line, **can actually predict future values** in our data set.
- This is a very basic example for 2 dimensions with x and y. But oftentimes, have data points contains 8 or 9 kind of input values.
- **Line of best fit** refers to a line through a scatterplot of data points that best expresses the relationship between those points



- Use this line of best fit to predict a new data point.
- All red data points are trained our model with their information that gave to the model so that it could create this line of best fit
- **a line equation => y = mx + b**
- B stands for your y intercept (0.4)
- X and Y stands for the coordinates of this data point.
- M stands for the slope, which is probably the most important part.
- **Calculate the slope of a line** - draw a triangle, pick two data points, calculate 2 distance and divide the distance up by the distance across.
- Looks at all data points, line splits evenly. Means close to every data point as possible.

	• **2 dimension Equation y = 1.5x + 0.5** • X and Y don't have a value, that's because give the value (x or y) to come up with one (x or y) of the other ones (y or x). • If x =2 then y =3.5, data point as a prediction here on this line (black point). If y= 2.7 , then find x.
	• 8 or 9 input variables, predict 1 output variable • 3 dimensions example, pass (x,y) -> predict z or (y,z) -> predict x
Coding https://www.tensorflow.org/install install sklearn (even notebook) => "!pip install -q sklearn" install TensorFlow (notebook only) => "%tensorflow_version 2.x" from _future_import absolute_import, division, print_function, Unicode_literals import numpy as np import pandas as pd import matplotlib.pyplot as plt from IPython.display import clear_output from six.moves import urllib import tensorflow.compat.v2.feature_column as fc import tensorflow as tf	NumPy is a very optimized version of arrays in Python, for lots of multi dimensional array calculations. Pandas is a data analytics tool (loading/view/visualize data sets,…) Matplotlib is a for visual graphs and charts. The ipython display (notebook only) to clear the output. TensorFlow compact v2 feature column is for a feature column when we create a linear regression algorithm or model in TensorFlow **Question** Which type of analysis would be best suited for the following problem?: You have the average temperature in the month of March for the last 100 years. Using this data, you want to predict the average temperature in the month of March 5 years from now.

	Multiple regression Correlation Decision tree [Answer]Linear regression

- **Core Learning Algorithms B - TensorFlow 2.0 Course** https://www.youtube.com/watch?v=u85IOSsJsPI

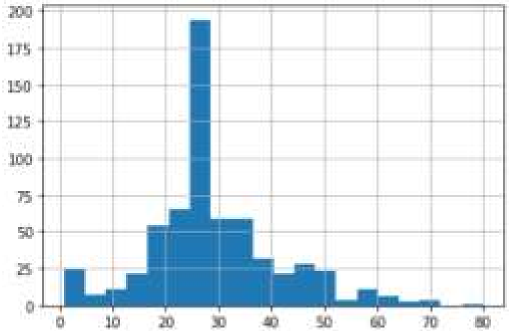
  The Titanic data set - aimed to predict who's going to survive

```
import pandas as pd
dftrain = pd.read_csv("https://storage.googleapis.com/tf-datasets/titanic/train.csv")
```

survived	sex	age	n_siblings_	parch	fare	class	deck	embark_tc	alone
0	male	22	1	0	7.25	Third	unknown	Southampt	n
1	female	38	1	0	71.2833	First	C	Cherbourg	n
1	female	26	0	0	7.925	Third	unknown	Southampt	y
1	female	35	1	0	53.1	First	C	Southampt	n
0	male	28	0	0	8.4583	Third	unknown	Queenstov	y
0	male	2	3	1	21.075	Third	unknown	Southampt	n
1	female	27	0	2	11.1333	Third	unknown	Southampt	n
1	female	14	1	0	30.0708	Second	unknown	Cherbourg	n
1	female	4	1	1	16.7	Third	G	Southampt	n

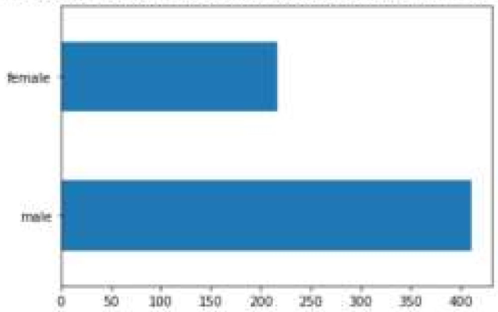
```
dfeval = pd.read_csv("https://storage.googleapis.com/tf-datasets/titanic/eval.csv")
#print(dftrain.head()) #first 5 entries from data set
y_train = dftrain.pop('survived') #removes the column
y_eval = dfeval.pop('survived')
#print(dftrain.head())
#print(y_train)
#print(dftrain.loc[0], y_train.loc[0]) # locating row zero
#print(dftrain["age"])
#dftrain.describe() # describe the overall information
#dftrain.shape
#dftrain.age.hist(bins=20) # histogram of the age
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f196f2b6650>




```
#dftrain.sex.value_counts().plot(kind='barh')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f196f2a2050>




```
#dftrain['class'].value_counts().plot(kind='barh')
pd.concat([dftrain, y_train], axis=1).groupby('sex').survived.mean().plot(kind='barh').set_xlabel('% survive'
)
```

- **Core Learning Algorithms C - TensorFlow 2.0 Course** https://www.youtube.com/watch?v=wz9J1slsi7I&t=6s

  2 different data - training (627,9) & testing (264, 9)
  training data to create the model and testing data to evaluate and make sure that it's working properly.
  doing machine learning models, typically have testing and training data.
  categorical data is not numeric - **transform this data into numbers (integer, 0, 1,2), not in order**
  numeric data – age
  defined categorical & numeric columns – loop through them, create something called **feature columns**.
  feature columns need to feed to our linear estimator or linear model to actually make predictions.

  A CategoricalColumn with in-memory vocabulary -
https://www.tensorflow.org/api_docs/python/tf/feature_column/categorical_column_with_vocabulary_list?version=stable

Code

```
import pandas as pd

import tensorflow.compat.v2.feature_column as fc
```

```python
import tensorflow as tf

dfrain = pd.read_csv("https://storage.googleapis.com/tf-datasets/titanic/train.csv")
dfeval = pd.read_csv("https://storage.googleapis.com/tf-datasets/titanic/eval.csv")
#y_train = dftrain.pop('survived') #removes the column
#y_eval = dfeval.pop('survived')

CATEGORICAL_COLUMNS = ['sex', 'n_siblings_spouses', 'parch', 'class', 'deck', 'embark_town', 'alone']
NUMERIC_COLUMNS = ['age', 'fare']

feature_columns = []
for feature_name in CATEGORICAL_COLUMNS:
  vovabulary =  dftrain[feature_name].unique() # gets a list of all unique from given feature columns
  feature_columns.append(tf.feature_column.categorical_column_with_vocabulary_list(feature_name,  vovabu
lary))

for feature_name in NUMERIC_COLUMNS:
  feature_columns.append(tf.feature_column.numeric_column(feature_name,  dtype=tf.float32))

#print(feature_columns) #[VocabularyListCategoricalColumn(key='sex', vocabulary_list=('male', 'female'),
 dtype=tf.string, default_value=-1, num_oov_buckets=0),
#dftrain['sex'].unique() #array(['male', 'female'], dtype=object)
```

- **Core Learning Algorithms D - TensorFlow 2.0 Course** https://www.youtube.com/watch?v=_cEwvqVoBhI&t=1s
  - **Training Process -** How do feed training data to the model?
    - In our case, only have 627 rows, it can fit that in PC RAM.
    - But if training a crazy ML model like 25 terabytes of data to pass it, can't load that into RAM
    - **Batches** - how this **process works**? Load, a small batch size of 32 entries at once to the model, that can increase our speed dramatically.
    - Don't load it entirely all at once, just load a specific set of kind of elements called **epochs**.
      - **Epochs** are essentially how many times the model is going to see the same data.
    - **Overfitting** – see(pass) the data too much to our model
  - **Input function** - is the way that we define how our data is going to be broke into epochs & into batches to feed model.
    - https://www.tensorflow.org/tutorials/estimator/linear
    - `make input function(data_df, label_df, num_epochs=10, shuffle=True, batch_size=32):` parameters
      - `data_df` - panda's data frame
      - `label_df` - labeled data frame (labels y train or eval)
      - `num_epochs` - number of epochs default 10

shuffle - shuffle data and mix it up before pass it to the model

`batch_size` - how many elements are we going to give to that to the model?

Code

```
from IPython.display import clear_output

def make_input_fn(data_df, label_df, num_epochs=10, shuffle=True, batch_size=32):
  def input_function(): # inner function, this will be returned
    #create a tf.data.Dataset object with the data and its label
    ds = tf.data.Dataset.from_tensor_slices((dict(data_df), label_df))
    if shuffle:
      ds = ds.shuffle(1000) # randomize order of data
    # split dataset into batches of 32 and repeat process for number of epochs.
    ds = ds.batch(batch_size).repeat(num_epochs)
    return ds #retun a bathc of the dataset
  return input_function # return a function object for use

train_input_fn = make_input_fn(dftrain, y_train) #call the function,returned a dataset object can feed t
o the model
eval_input_fn = make_input_fn(dfeval, y_eval, num_epochs=1, shuffle=False)

# Estimators are not recommended for new code.  use keros API
# estimators are just basic implementations of algorithms in TensorFlow
linear_est = tf.estimator.LinearClassifier(feature_columns=feature_columns)

linear_est.train(train_input_fn) # train train_input_fn => train_input_fn()
result = linear_est.evaluate(eval_input_fn) # get model metrics/stats by testing on testing data

clear_output() #clear console output
print(result)
print(result['accuracy']) # the result variable is simply a dict of stats about our model
# 0.75757575 - This accuracy isn't very good, talk about how to improve this.
```

#re run

#Now notice, accuracy changed to 76 , the reason the data is getting shuffled and put in in a different order.

 # It make different predictions and be trained differently.

#Change epochs, to 11, or 15, accuracy will change.

#goal is to get the most accurate model

#TensorFlow models are built to make predictions on a lot of things at once, they're not great at making predictions on#like one piece of data (like one passenger to make a prediction for), they're much better at working in like large
#batches of data.
#Make a prediction for every single point that's in that evaluation data set.

#A dictionary that represents the predictions. For every single, what is it prediction.
#We passed 267 eval input data, it returned a list of all of these different dictionaries that represent each prediction.

```
pred_dicts = list(linear_est.predict(eval_input_fn))
print(pred_dicts)

#[{'logits': array([-
2.0278394], dtype=float32), 'logistic': array([0.11631075], dtype=float32), 'probabilities': array([0.8836892 , 0.11631081], dtype=float32),
#'class_ids': array([0]), 'classes': array([b'0'], dtype=object), 'all_class_ids': array([0, 1], dtype=int32), 'all_classes': array([b'0', b'1'], dtype=object)}, ...

print(pred_dicts[0])
print(pred_dicts[0]['probabilities']) # [0.8836892 , 0.11631081]  the percentage of survival 88% , won't survive is 11%.
print(dfeval.loc[0]) # verify the passenger survey
print(y_eval.loc[0])
print(pred_dicts[0]['probabilities'][1]) # survival %
print(pred_dicts[0]['probabilities'][0]) # non survival %
```

Questions: What are epochs?

[Answer]The number of times the model will see the same data.
A type of graph.
The number of elements you feed to the model at once.

- **Core Learning Algorithms E - TensorFlow 2.0 Course** https://www.youtube.com/watch?v=qFF7ZQNvK9E

  classification - is differentiating between data points and separating them into classes.
  So rather than predicting a numeric value (did linear regression earlier), actually want to predict classes.
  for example, iris flower data set, use different properties of flowers to predict what species of flower it is.
  Iris flowers data - separates flowers into three different species (setosa, versicolor, virginica)

The information about each flower is (sepal/petal length & width, petal length & width) that information, Kara's - sub module of TensorFlow.
https://www.tensorflow.org/tutorials/estimator/premade

Code

```
import pandas as pd

import tensorflow.compat.v2.feature_column as fc
import tensorflow as tf

CSV_COLUMN_NAMES = ['SepalLength', 'SepalWidth', 'PetalLength', 'PetalWidth', 'Species']
SPECIES = ['Setosa', 'Versicolor', 'Virginica']

train_path = tf.keras.utils.get_file(
    "iris_training.csv", "https://storage.googleapis.com/download.tensorflow.org/data/iris_training.csv"
)
test_path = tf.keras.utils.get_file(
    "iris_test.csv", "https://storage.googleapis.com/download.tensorflow.org/data/iris_test.csv")

train = pd.read_csv(train_path, names=CSV_COLUMN_NAMES, header=0)
test = pd.read_csv(test_path, names=CSV_COLUMN_NAMES, header=0)
#here using Keras (a tensorlfow module ) to grab datasets and read them into a pandas dataframe

train.head()

train_y = train.pop('Species')
test_y = test.pop('Species')

# The label column has now been removed from the features.
train.head()

train.shape # (120, 4)

#Input function

def input_fn(features, labels, training=True, batch_size=256):
    """An input function for training or evaluating"""
    # Convert the inputs to a Dataset.
    dataset = tf.data.Dataset.from_tensor_slices((dict(features), labels))

    # Shuffle and repeat if you are in training mode.
```

```
    if training:
        dataset = dataset.shuffle(1000).repeat()

    return dataset.batch(batch_size)

#Feature Columns
# Feature columns describe how to use the input.
my_feature_columns = []
for key in train.keys():
    my_feature_columns.append(tf.feature_column.numeric_column(key=key))

print(my_feature_columns)
```

Question : What is classification?

[Answer]The process of separating data points into different classes.

Predicting a numeric value or forecast based on independent and dependent variables.

None of the above.

- **Core Learning Algorithms F - TensorFlow 2.0 Course** https://www.youtube.com/watch?v=5wHw8BTd2ZQ&t=9s

Building the classification model – 100th of different classification pre-made models in TensorFlow.

Now can pick from, options are

DNNClassifier (deep neural network)

LinearClassifier – it works very similarly to linear regression, except it does classification.

Pick either one, but the DNN is the best choice, because may not be able to find a liner correspondence in our data.

It's not that difficult to change models, because most of the work comes from loading and pre processing our data.

Build a DNN with two hidden later with 30 nodes and 10 hidden nodes each.

hidden units is essentially us a building the architecture of the neural network.

An input layer, some middle layers (called hidden layers in a neural network), output layer

Decided 30 nodes in the 1st hidden layer 10 in the 2nd & the no of classes is 3 (3 classes for the flowers)

Code

```
# Build a DNN with 2 hidden layers with 30 and 10 hidden nodes each.
classifier = tf.estimator.DNNClassifier(
    feature_columns=my_feature_columns,
    # Two hidden layers of 30 and 10 nodes respectively.
    hidden_units=[30, 10],
    # The model must choose between 3 classes.
    n_classes=3)
```

Train the model

a lambda is an anonymous function that can be defined in one line

x = lambda:print("hi")

x1() # hi will print

steps = 5000 - similar to an epoch (go the dataset 10 times), but it go through the dataset until hit 5000 numbers

Run the code

it tells us the current step, the loss (the lower is the better), global steps per second.

Now at the end, final step, loss of 39, is pretty high, which means this is pretty bad.

Code

```
# Train the Model.
classifier.train(input_fn=lambda: input_fn(train, train_y, training=True), steps=5000)
```

Evaluation on the model.

Run, Much faster, we get a test accuracy of 80%.

Code

```
eval_result = classifier.evaluate(input_fn=lambda: input_fn(test, test_y, training=False))

print('\nTest set accuracy: {accuracy:0.3f}\n'.format(**eval_result))
```

Predictions on any given flower.

Type some numbers (petal length and width), then it will the predicted class of that flower is.

Not passing any y value, because we're making a prediction, so the model will the answer

type like 2.4, 2.6, 6.5, 6.3

notice we get three probabilities, one for each of the different classes.

class ID – it predicts is actually the flower, two means index array of 2

Code - "dnnclassifier.py"

```
WARNING:tensorflow:Using temporary folder as model dire

Test set accuracy: 0.867

Please type numeric values as prompted:
SepalLength: 2.3
SepalWidth: 2.6
PetalLength: 6.3
PetalWidth: 6.5
<generator object Estimator.predict at 0x7fb0f98d01d0>
Prediction is "Virginica" (92.2%)
```

Question: What kind of estimator/model does TensorFlow recommend using for classification?

LinearClassifier, [Answer]DNNClassifier, BoostedTreesClassifier

- **Core Learning Algorithms G - TensorFlow 2.0** Course https://www.youtube.com/watch?v=8sqIaHc9Cz4&t=1s

Clustering is the first unsupervised learning algorithm.

clustering only works for a very specific set of problems.

When use clustering, have a bunch of i/p information/features, don't have any labels or open information.

what clustering does is finds clusters of like data points and tells the location of those clusters.

Give a bunch of training data, pick how many clusters you want find.

Classifying handwritten digits using k means clustering. 10 different clusters for the digits 0 – 9.

The algorithm finds those clusters in the data set

Clustering

Now that we've covered regression and classification it's time to talk about clustering data!

Clustering is a Machine Learning technique that involves the grouping of data points. In theory, data points that are in the same group should have similar properties and/or features, while data points in different groups should have highly dissimilar properties and/or features. (https://towardsdatascience.com/the-5-clustering-algorithms-data-scientists-need-to-know-a36d136ef68)

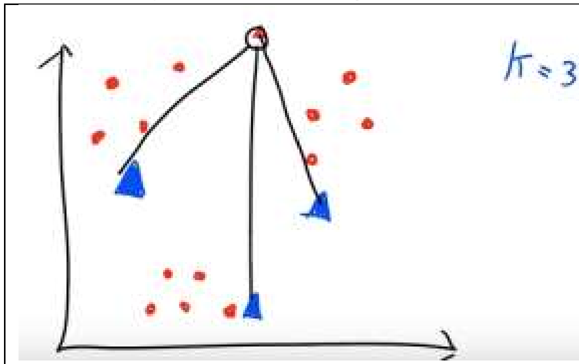
Unfortunalty there are issues with the current version of TensorFlow and the implementation for KMeans. This means we cannot use KMeans without writing the algorithm from scratch. We aren't quite at that level yet so we'll just explain the basics of clustering for now.

Basic Algorithm for K-Means.

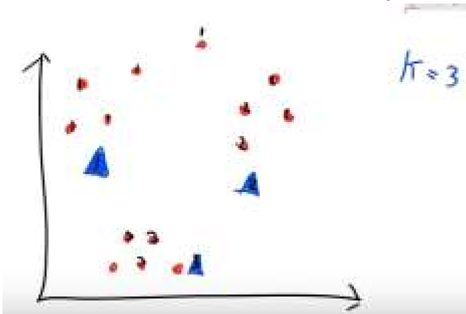
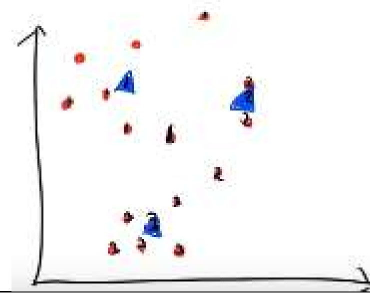
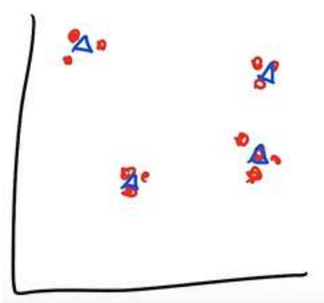
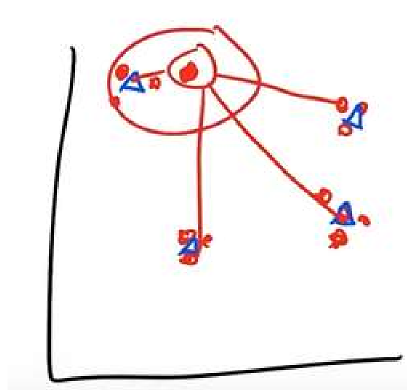
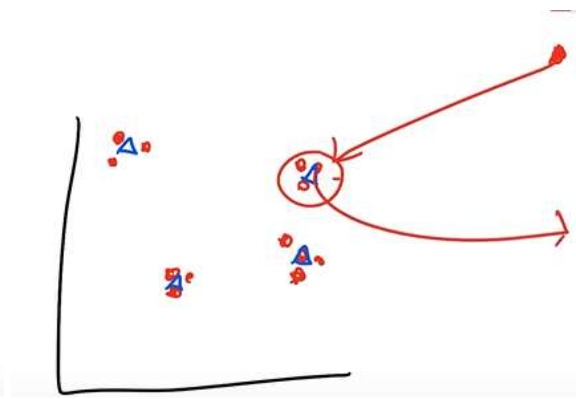
- Step 1: Randomly pick K points to place K centroids
- Step 2: Assign all of the data points to the centroids by distance. The closest centroid to a point is the one it is assigned to.
- Step 3: Average all of the points belonging to each centroid to find the middle of those clusters (center of mass). Place the corresponding centroids into that position.
- Step 4: Reassign every point once again to the closest centroid.
- Step 5: Repeat steps 3-4 until no point changes which centroid it belongs to.

Please refer to the video for an explanation of KMeans clustering.

Explain basic algorithm for K-Means



- 2 dimensions, make some data points (red), by putting them in their own unique little groups
- Now the algorithm starts for K means clustering. understand how this works, by randomly picking k centroids (filled in triangle) k = 3
- Now what happens next is each group/data point, is assigned to a cluster by distance.
- for every single data point find the distance using Euclidean or Manhattan distance,
- looking at this data point (circle) find the distance to all of these different centroids.

Assigned number (red above 1, 2, 3) to the closet to one data points 	Added some more data points, now move these centroids into the middle of all of their data points called **center of mass.** Same thing other 2, remove and rearrange 
keep doing until eventually reach a point where none of these points are changing the centroid, finally the draw is like 	this now our cluster, if new points added, find the closet and assign to the closet cluster  

Question : Which of the following steps is not part of the K-Means algorithm?

Randomly pick K points to place K centeroids.

Assign each K point to the closest K centeroid.

Move each K centeroid into the middle of all of their data points.

[Answer] Shuffle the K points so they're redistributed randomly.

Reassign each K point to the closest K centeroid.

- **Core Learning Algorithms H - TensorFlow 2.0 Course** https://www.youtube.com/watch?v=IZg24y4wEPY&t=4s

  Hidden Markov models - deal with probability distributions.

  - **Example weather model** - predict the weather on any day, given the probability of different events occurring.
  - Using the average temperature on the days, create a hidden Markov model, will make a prediction for the weather in future days
  - in this example, use some predefined probability distributions.

Hidden Markov Models

"The Hidden Markov Model is a finite set of states, each of which is associated with a (generally multidimensional) probability distribution []. Transitions among the states are governed by a set of probabilities called transition probabilities." (http://jedlik.phy.bme.hu/~gerjanos/HMM/node4.html)

A hidden markov model works with probabilities to predict future events or states. In this section we will learn how to create a hidden markov model that can predict the weather.

https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/HiddenMarkovModel

in a hidden Markov model, have a bunch of states.

Weather model, the states is hot & cold day (called **hidden**, because never access/look at these states)

In the model, called **observations** – each state have an observation

Example, If it is hot, 80% happy. If it is cold, 20% happy.

So at that state, we can observe the probability of something happening during that state is x or y

Data

Previous cases, use like 1000s of entries data points for our models to train for this.

Don't need any, only need is just constant values for probability(transition & observation distributions.

Data

Let's start by discussing the type of data we use when we work with a hidden markov model.

In the previous sections we worked with large datasets of 100's of different entries. For a markov model we are only interested in probability distributions that have to do with states.

We can find these probabilities from large datasets or may already have these values. We'll run through an example in a second that should clear some things up, but let's discuss the components of a markov model.

States: In each markov model we have a finite set of states. These states could be something like "warm" and "cold" or "high" and "low" or even "red", "green" and "blue". These states are "hidden" within the model, which means we do not direcly observe them.

Observations: Each state has a particular outcome or observation associated with it based on a probability distribution. An example of this is the following: *On a hot day Tim has a 80% chance of being happy and a 20% chance of being sad.*

Transitions: Each state will have a probability defining the likelyhood of transitioning to a different state. An example is the following: *a cold day has a 30% chance of being followed by a hot day and a 70% chance of being follwed by another cold day.*

To create a hidden markov model we need.

- States
- Observation Distribution
- Transition Distribution

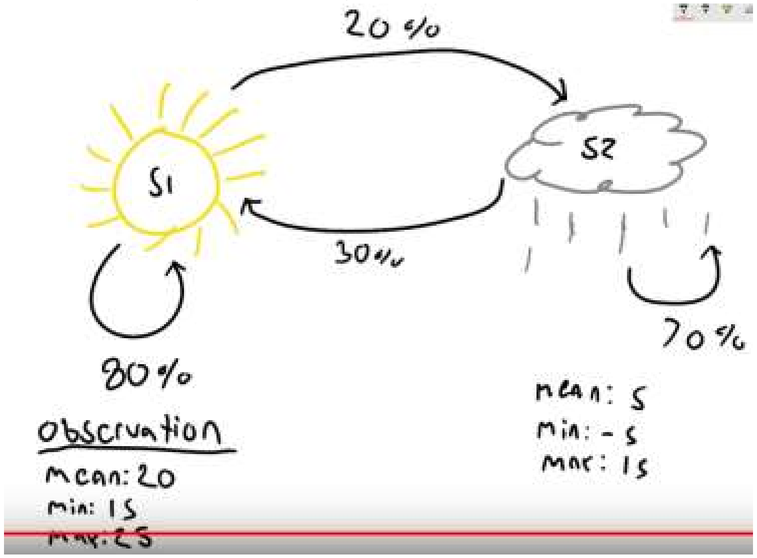
For our purpose we will assume we already have this information available as we attempt to predict the weather on a given day.

Draw Example - 2 state (s1, s2)

probability of transitioning to the other state.

in a hot day, 20% chance of transitioning to a cold day, 80% chance of transitioning to another hot day
in a cold day, a 30% chance of transitioning to a hot day, 70% chance of transitioning to another cold day.

observation probability or distribution



Question : What makes a Hidden Markov model different than linear regression or classification?

[Answer]It uses probability distributions to predict future events or states.

It analyzes the relationship between independent and dependent variables to make predictions.

It separates data points into separate categories.

- **Core Learning Algorithms I - TensorFlow 2.0 Course** - https://www.youtube.com/watch?v=fYAYvLUawnc

TFP is TensorFlow probability.

Initial distribution is 80%, 20%. Transition distribution, the probability is 70%, and 30%, and 20% 8%.

Transition probability - 30% chance, 20% chance for a hot day.

Observation distribution - average temperature is going to be 0 on a hot day. 15 On a cold day
the standard deviation on the cold is 5 (from 5 to -5), and on a hot day to 10.

Steps - is how many days we want to predict for. So the number of steps is how many times we're going
mean, a partially defined tensor.

Weather Model

Taken direclty from the TensorFlow documentation (https://www.tensorflow.org/probability/api_docs/python/tfp/distributions/HiddenMarkovModel).

We will model a simple weather system and try to predict the temperature on each day given the following information.

1. Cold days are encoded by a 0 and hot days are encoded by a 1.
2. The first day in our sequence has an 80% chance of being cold.
3. A cold day has a 30% chance of being followed by a hot day.
4. A hot day has a 20% chance of being followed by a cold day.
5. On each day the temperature is normally distributed with mean and standard deviation 0 and 5 on a cold day and mean and standard deviation 15 and 10 on a hot day.

If you're unfamiliar with **standard deviation** it can be put simply as the range of expected values.

In this example, on a hot day the average temperature is 15 and ranges from 5 to 25.

To model this in TensorFlow we will do the following.

Code - "tfprobability.py"

Question: What TensorFlow module should you import to implement .HiddenMarkovModel()?

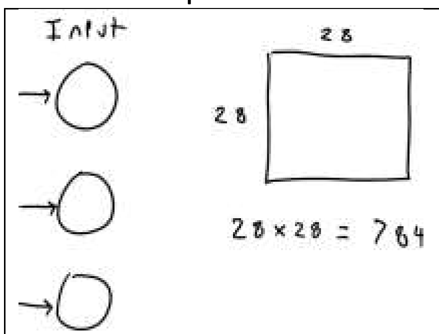
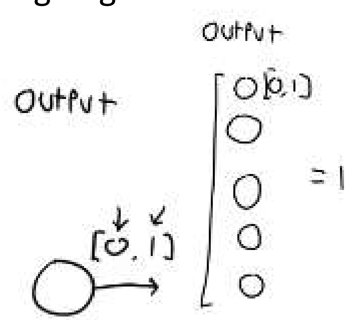
tensorflow.keras

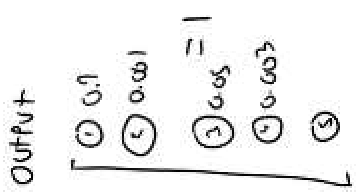
tensorflow_gpu

[Answer]tensorflow_probability

- **Neural Networks with TensorFlow A - TensorFlow 2.0 Course** - https://www.youtube.com/watch?v=uisdfrNrZW4

The whole point of a neural network is to provide, classification or predictions for us.

 If you're predicting for the image, need every single one of those pixels, which is 28 X 28 = 784 input neurons.	Need one input neuron for each piece of that information. Output layer is going to have as many neurons  classification for images - 1 output neuron value between 0 and 1

	5 output neurons have a value between 0 and 1. Then the sum of every single one these values would be equal to one (like a probability distribution). Like be 0.9 (90%), 0.001, 0.05, 0.003

from input to output **- a hidden layer**, can have many, that are connecting to other hidden layers

Every single layer is connected to another layer with something called **weights**.

Densely connected neural network or layer - means that is connected to every node from the previous layer.

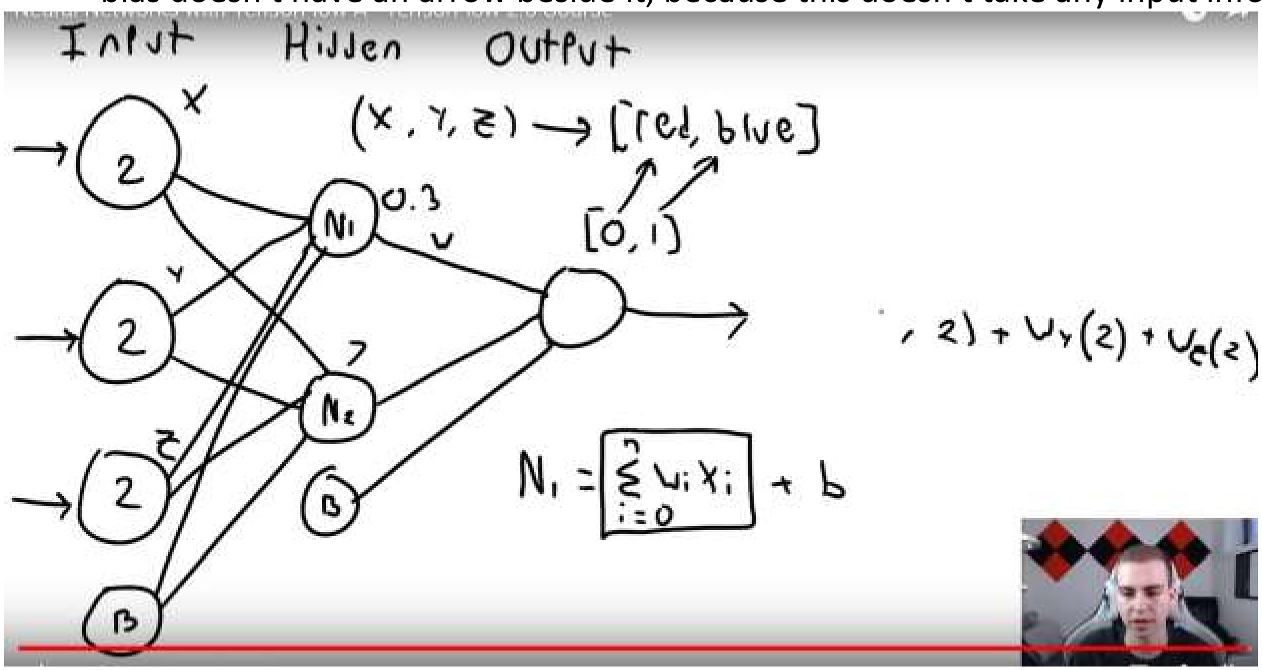
these weights is going to change and optimize to determine the mapping from our input to our output.

Have numbers for every single one of these lines - **trainable parameters**

Biases - only one bias, and a bias exists in the previous layer to the layer that it affects.

in this case, a bias that connects to each neuron in the next layer from this

bias doesn't have an arrow beside it, because this doesn't take any input information.



Question: A densely connected neural network is one in which...:

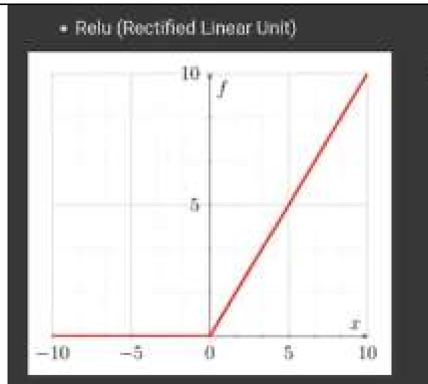
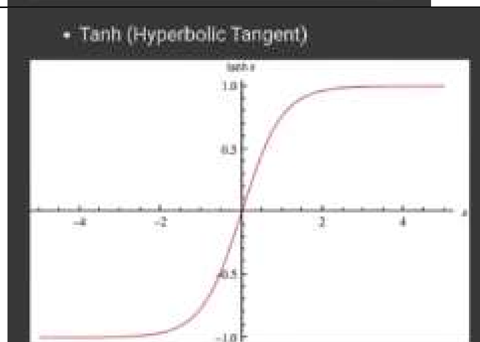
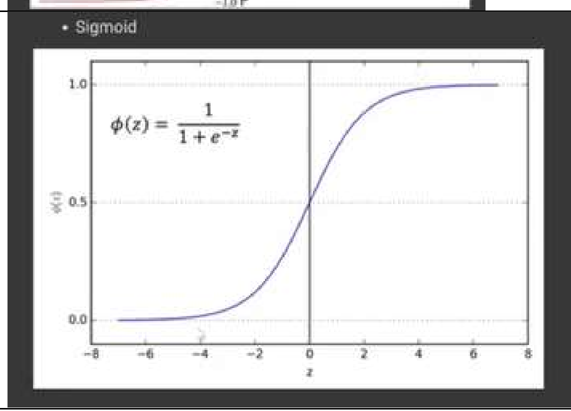
all the neurons in the current layer are connected to one neuron in the previous layer.

all the neurons in each layer are connected randomly.

[Answer]all the neurons in the current layer are connected to every neuron in the previous layer.

- **Neural Networks with TensorFlow B - TensorFlow 2.0 Course** https://www.youtube.com/watch?v=S45tqW6BqRs&t=2s

Activation function - some examples

	**Rectified linear unit** - This activation function does is take any values that are < 0 and make them zero. If any negative, it just makes their y zero. If any positive, it's just equal to whatever their positive value is.
	**hyperbolic tangent** - squishes values between -1 and +1
	**Sigmoid (squishy fire function)** - squish values between 0 and 1 Take any extremely negative numbers and put them closer to zero Take any extremely positive numbers and put them close to one

how use them

Each neurons have an activation function that is applied to the output of that neuron.

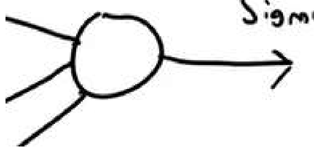
Take the weighted sum + the bias, then apply an activation function to it before send that value to the next neuron.

Activation function of this equation

Output

$$N_1 = F\left(\sum_{i=0}^{n} w_i x_i + b\right)$$

Sigmoid



$$\text{Sigmoid}(N_1 V_0 + N_2 V_1 + b) = [0, 1]$$

why we've used the activation function on like an intermediate layer like this.

the whole point of an activation function is to introduce complexity into our neural network.

A few examples **of loss function** - **mean squared error, mean absolute error and hinge loss**

mean absolute error

$$mae = \frac{\sum_{i=1}^{n} abs\left(y_i - \lambda(x_i)\right)}{n}$$

gradient descent - calculate this loss, then use an gradient descent algorithm, which tells us what direction need to move our function to determine our to get to this global minimum.

Gradient, which is literally just a steepness or a direction.

brought backpropagation Algorithm - go backwards through the network and update the weights and biases

Gradient Descent

Gradient descent and backpropagation are closely related. Gradient descent is the algorithm used to find the optimal paramaters (weights and biases) for our network, while backpropagation is the process of calculating the gradient that is used in the gradient descent step.

Gradient descent requires some pretty advanced calculus and linear algebra to understand so we'll stay away from that for now. Let's just read the formal definition for now.

"Gradient descent is an optimization algorithm used to minimize some function by iteratively moving in the direction of steepest descent a defined by the negative of the gradient. In machine learning, we use gradient descent to update the parameters of our model." (https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html)

Loss/Cost Function

As we now know our neural network feeds information through the layers until it eventually reaches an output layer. This layer contains the results that we look at to determine the prediciton from our network. In the training phase it is likely that our network will make many mistakes and poor predicitions. In fact at the start of training our network doesn't know anything (it has random weights and biases)!

We need some way of evaluating if the network is doing well and how well it is doing. For our training data we have the features (input) and the labels (expected output), because of this we can compare the output from our network to the expected output. Based on the difference between these values we can determine if our network has done a good job or poor job. If the network has done a good job we'll make minor changes to the weights and biases. If it has done a poor job our changes may be more drastic.

So this is where the cost/loss function comes in. This function is responsible for determining how well the network did. We pass it the output and the expected output and it returns to us some value representing the cost/loss of the network. This effectively makes the networks job to optimize this cost function, trying to make it as low as possible.

Some common loss/cost functions include.

- Mean Squared Error
- Mean Absolute Error
- Hinge Loss

Question : Which activation function squishes values between -1 and 1?

ReLU (Rectified Linear Unit)
[Answer]Tanh (Hyperbolic Tangent)
Sigmoid

- **Neural Networks with TensorFlow C - TensorFlow 2.0 Course** - https://www.youtube.com/watch?v=hdOtRPQe1o4&t=1s

Optimizer - the algorithm that does the gradient descent and back propagation for us**. [Advanced ML Technique]**

Optimizer

You may sometimes see the term optimizer or optimization function. This is simply the function that implements the backpropagation algorithm described above. Here's a list of a few common ones.

- Gradient Descent
- Stochastic Gradient Descent
- Mini-Batch Gradient Descent
- Momentum
- Nesterov Accelerated Gradient

*This article explains them quite well is where I've pulled this list from. *

(https://medium.com/@sdoshi579/optimizers-for-training-neural-network-59450d71caf6)

first official neural network.

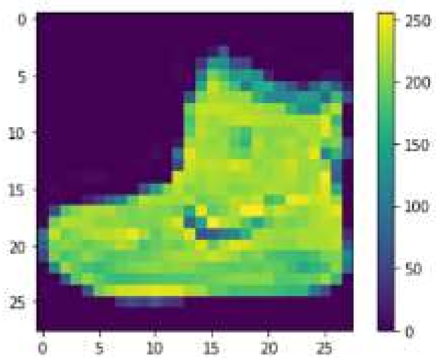
Imports

Dataset – MNIST fashion dataset

contains 60,000 images for training and 10,000 images for validating/testing 70,000 images

it is essentially pixel data of clothing articles

load in this data set from Keras (like beginner)



Layer 1: This is our input layer and it will conist of 784 neurons. We use the flatten layer with an input shape of (28,28) to denote that our input should come in in that shape. The flatten means that our layer will reshape the shape (28,28) array into a vector of 784 neurons so that each pixel will be associated with one neuron.

Layer 2: This is our first and only hidden layer. The *dense* denotes that this layer will be fully connected and each neuron from the previous connects to each neuron of this layer. It has 128 neurons and uses the rectify linear unit activation function.

Layer 3: This is our output later and is also a dense layer. It has 10 neurons that we will look at to determine our models output. Each neur represnts the probabillity of a given image being one of the 10 different classes. The activation function *softmax* is used on this layer to calculate a probability distribution for each class. This means the value of any neuron in this layer will be between 0 and 1, where 1 repres

Question: What is an optimizer function?

A function that increases the accuracy of a model's predictions.

[Answer]A function that implements the gradient descent and backpropagation algorithms for you.

A function that reduces the time a model needs to train.

- **Neural Networks with TensorFlow D - TensorFlow 2.0 Course** - https://www.youtube.com/watch?v=K8bz1bmOCTw&t=1s

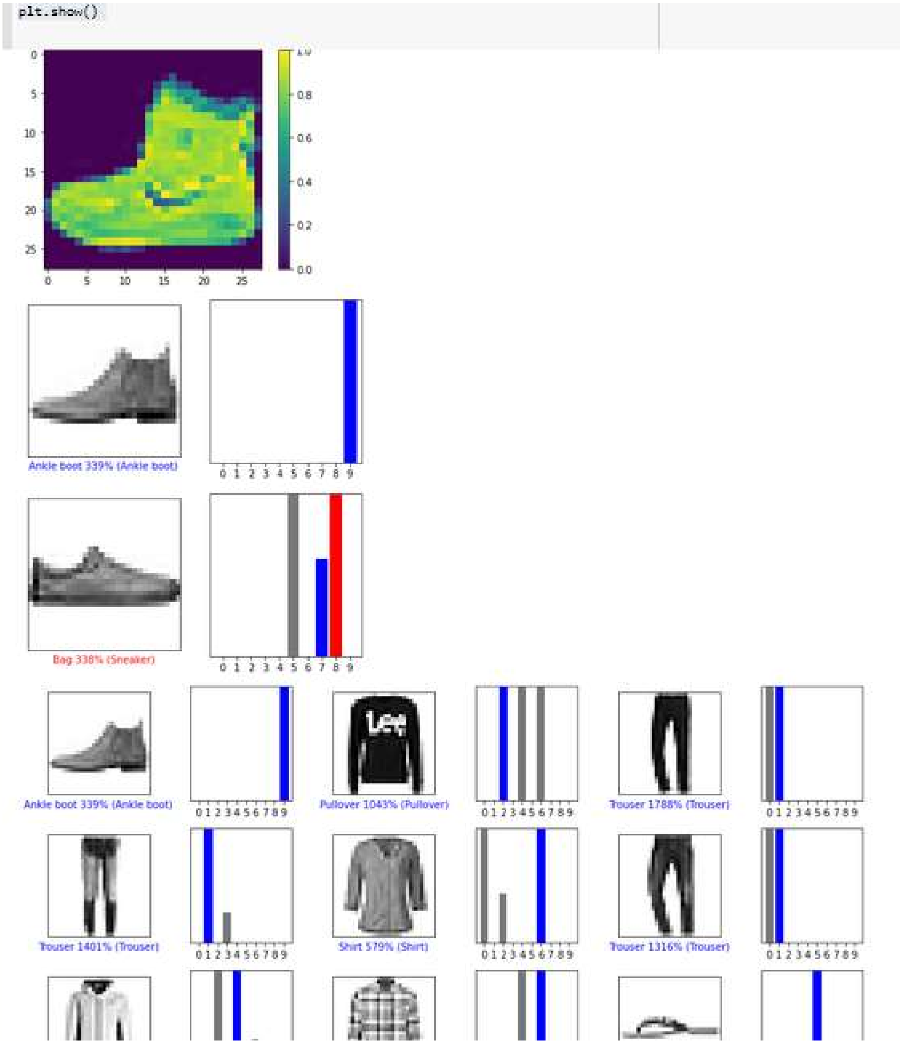
  sequential - the most basic form of neural network, which is just information going from the left side to the right side, passing through the layers sequentially

  Layer 1 – input layer - take in a shape of 28 by 28, flatten all of the pixels into 784 pixels.

Layer 2 – 1st hidden layer, dense layer - the neurons in the previous layer are connected to every neuron in this layer.
128 neurons here, how do we pick that number? We don't know, just came up with it.
activation function will define as rectified linear unit. (pick any one)

Layer 3 – output layer, dense layer of 10 output neurons with the activation of softmax (probability distribution).
why we would have picked 10? There are 10 class_names
softmax will make all of the values of our neurons add up to one, and that there are between zero and one.

Code – "classification.py"



Question: Fill in the blanks below to build a sequential model of dense layers:

model = __A__.__B__([
  __A__.layers.Flatten(input_shape=(28, 28)),
  __A__.layers.__C__(128, activation='relu'),
  __A__.layers.__C__(10, activation='softmax')

])

[Answer]A: keras B: Sequential C: Dense
A: tf B: Sequential C: Categorical
A: keras B: sequential C: dense