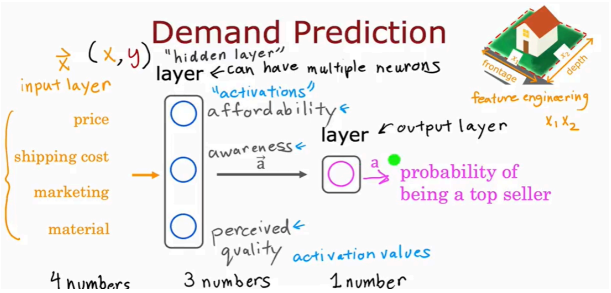
Course 2: Advanced Learning Algorithm

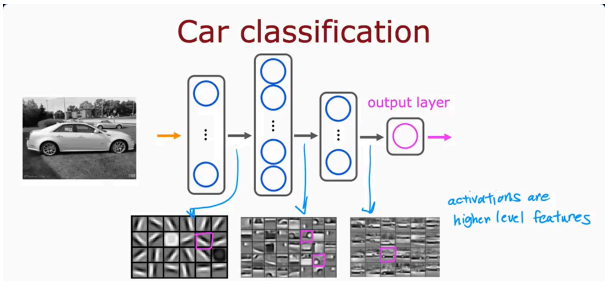
Week 1:

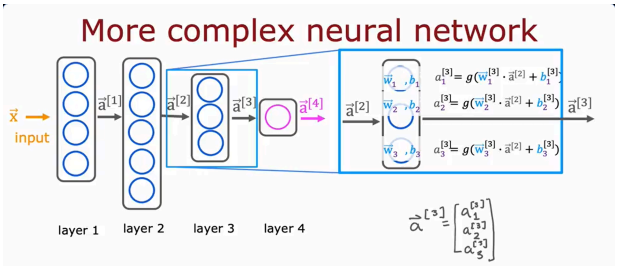
Neural Network Model:

- Deep Learning algorithms helped overcome the performance plateau faced when working with large amounts via traditional AI.
- Activations are the produced outputs from a single neuron.
- Each neuron in a certain layer will have access to every feature from the previous layer



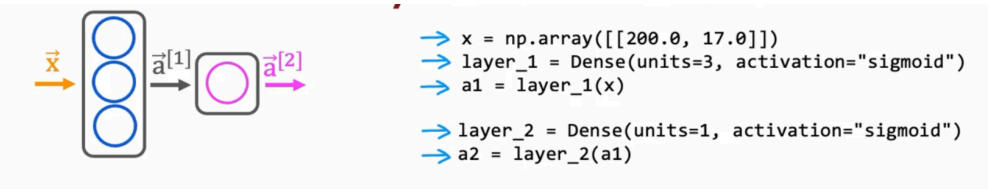
- How many hidden layers and neurons depends on the architecture of the NN
- Images are determined by local patterns (in a hidden layer it may detect a single edge/ line of the overarching image, the next hidden layer may aggregate and look at a bigger subsection of the overarching image) (Ex: **Hidden Layer (HL) 1:** Pixel lines/edges, **HL 2:** Car parts, **HL 3:** Car shapes)



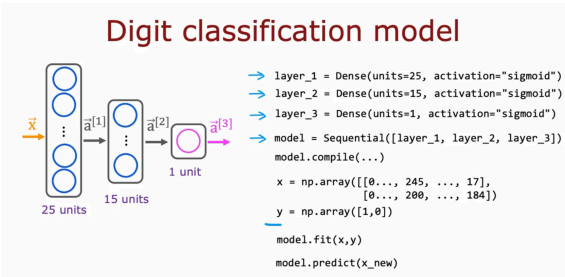


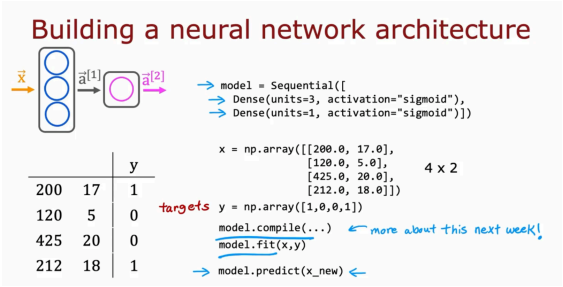
Neural Network Implementations:

- Tensorflow uses matrices to represent data to be more computationally efficient due to the large size of datasets (numpy uses double brackets as single square brackets result in a 1D vector)
- Forward prop using Numpy:



- Building NN using Tensorflow:

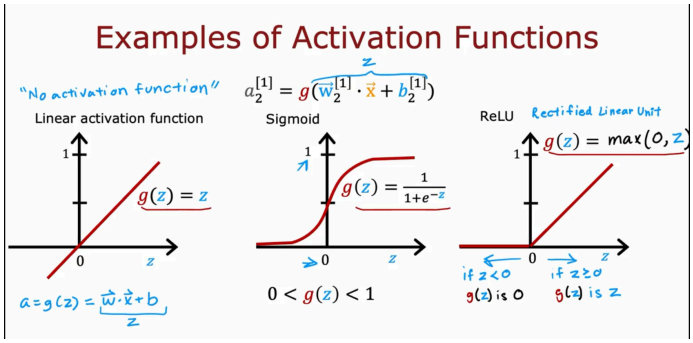




Week 2:

Activation Functions:

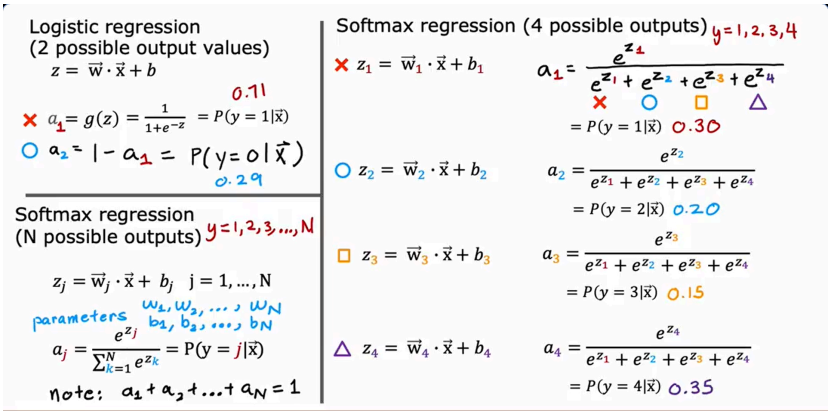
- Types of Activations:



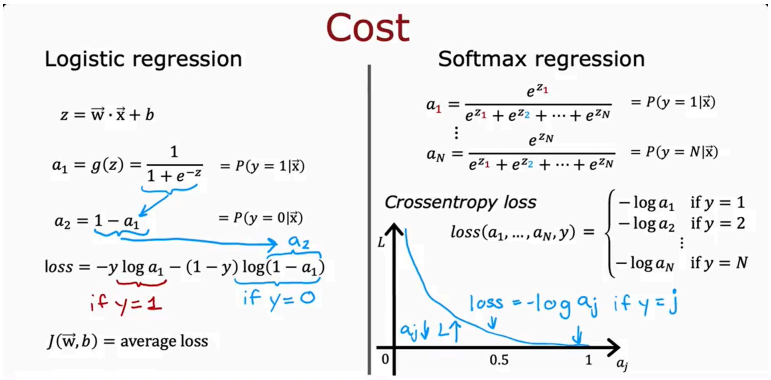
- When to use what activation?
  - **ReLu**: Regression where output always is positive**,** or atleast will never be negative.
  - **Sigmoid**: Binary Classification (0/1)
  - **Linear**: Regression where output can either be positive or negative (stock prices)
- No activation function or linear activation function does not work for model predictions
- Linear activation on all nodes will result in the NN essentially becoming a linear regression model, removing the purpose of creating a NN model as the model does not learn any more complex features. Adding ReLu to the output node transforms to Logistic Regression. RULE OF THUMB: Don't use linear activations for hidden layers

Multiclass Classification:

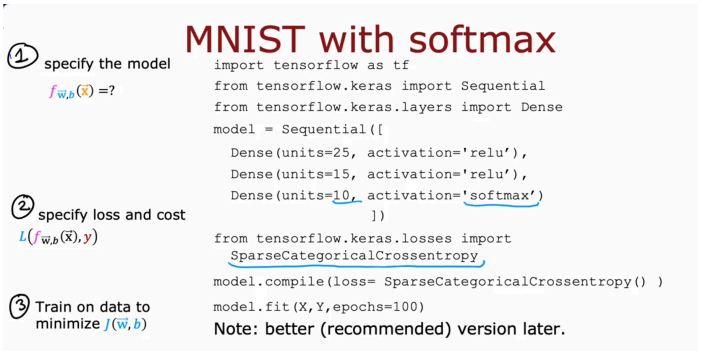
- Refers to when 'Y' has more than 2 outputs. **Requires a Softmax output layer to convert scores into probabilities for class prediction.**
- **Softmax** will have n outputs for n classes, accompanied by w_n ,b_n, a_n and z_n being computed in a vectorized manner. **All vectorized outputs will add up to 1**
- Softmax is considered a generalization of logistic as, if n = 2 for softmax, the computation results in the same as logistic regression



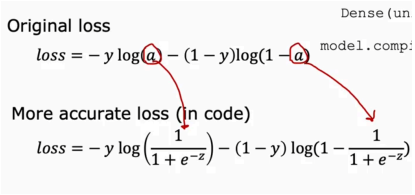
- Loss in softmax is inversely proportional to a_n, the closer $\mathbf{a_n}$ is to 1, the lower the loss

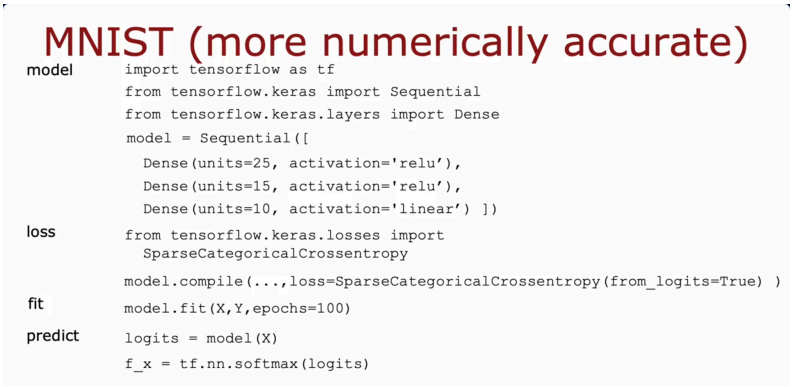


- Uses **Categorical_CrossEntropy** for loss, where labels are chosen as categories and sparse refers to a class receiving one output not multiple.

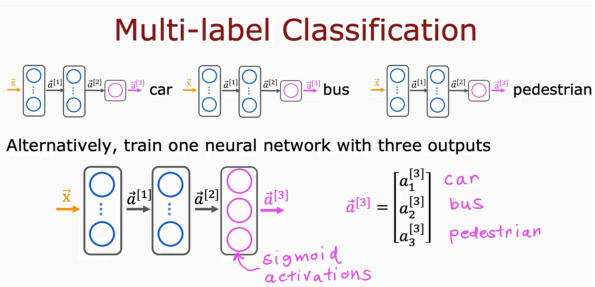


- **model.compile(loss=*BinaryCrossEntropy(*from_logits=True))** is the numerically accurate implementation of loss functions by computing the **"z"** value directly instead of the **"a"** value mitigating numerical roundoff



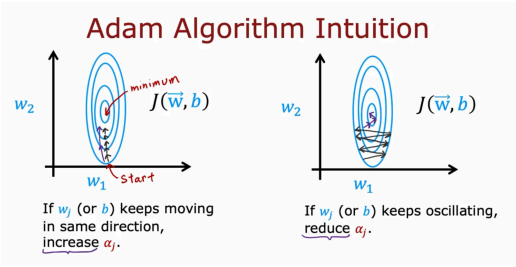


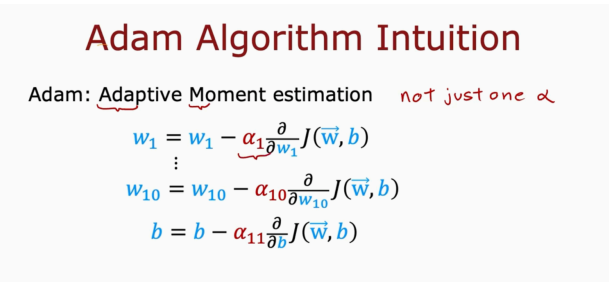
- Multi-class vs Multi-label:
    - Output for multi-class is a single number, even if the output has multiple possible values (handwritten digit).
    - Output for multi-label is a vector of numbers (detecting if a image has people, cars and buses from the same image may yield [1,0, 1] (1: Yes, 0: No)



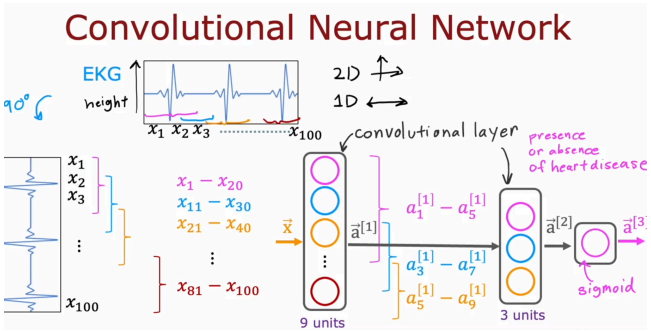
Advanced Neural Network Concepts:

- Adam Optimizer
    - Use Adam optimizer over gradient descent. Adam optimizer increases learning rate if weights move towards minimum cost using the same direction, if it oscillates instead then it decreases learning rate.
    - Every weight/bias has it own unique learning rate, each increasing /decreasing as needed





- Convolution layer:
    - Each **Conv** layer neuron looks at parts of previous layers output instead of all of it leading to faster computation and less overfitting
    - Take an EKG signal of inputs $\mathbf{x_1}$ to $\mathbf{x_{100}}$. One neuron of **CNN layer 1** might look at the $\mathbf{x_1}$ to $\mathbf{x_{20}}$, the next neuron of the same layer might look at $\mathbf{x_{11}}$ to $\mathbf{x_{30}}$ and so on until all **100 instances** are looked at. Like this we are left with multiple neurons looking at partitions of the original input. The next **CNN layer 2** will have neurons that look at a few of neurons from **CNN layer 1** at a time. This new layer will have the an amount of neurons that will combined use all the outputs from the previous layers neurons



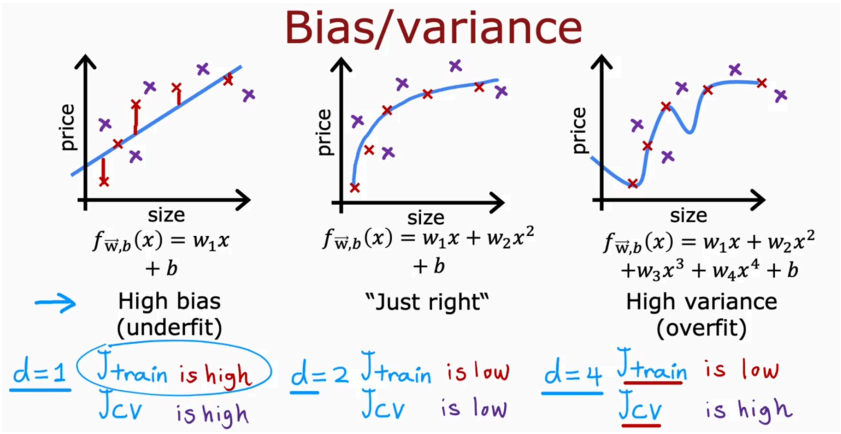
Week 3:

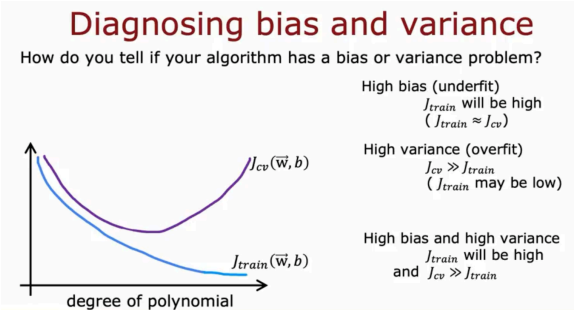
Debugging A Learning Algorithm:

→ Get more training examples	fixes high variance
→ Try smaller sets of features $x, x^2, \cancel{x}, \cancel{x}, \cancel{x}, \ldots$	fixes high variance
→ Try getting additional features	fixes high bias
→ Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, etc)$	fixes high bias
→ Try decreasing λ	fixes high bias
→ Try increasing λ	fixes high variance

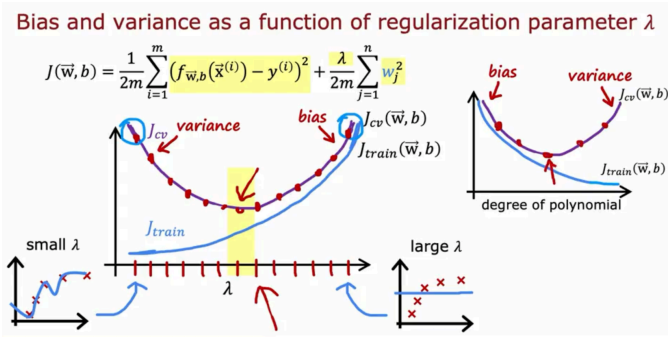
Bias and Variance

- Improving models via diagnosis
    - Split data into train/test and determine the error cost for each. High **test error** and low **train error** means the model has overfit and cannot generalize well. Usually **test error cost** is a better estimate of generalization of new data compared to **train error cost**
    - **Test error** is usually replaced by **CV error**, as **test error** provides an optimistic estimate (test data should be the last dataset to be pushed into the model for final testing)



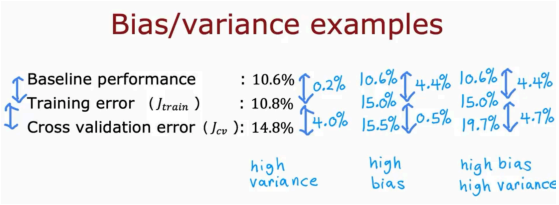


- Altering the regularization parameter will help manipulate bias/variance

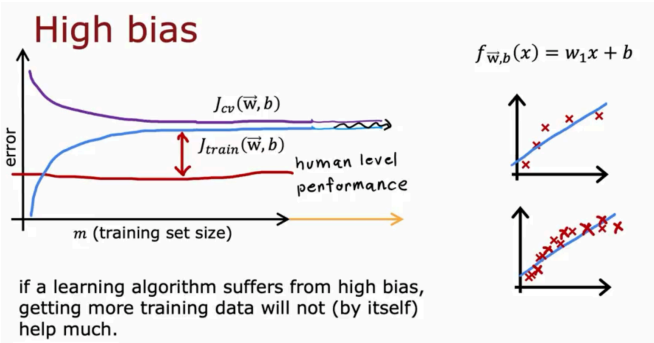


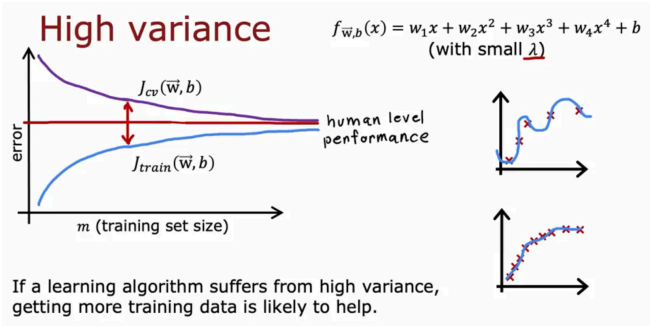
- Baseline of a model:
    - Use human level performance as a benchmark to test the accuracies of your models (How well can a human recognizes human voices vs a machine learning model recognizing human voice)
    - Performance of competing algorithms

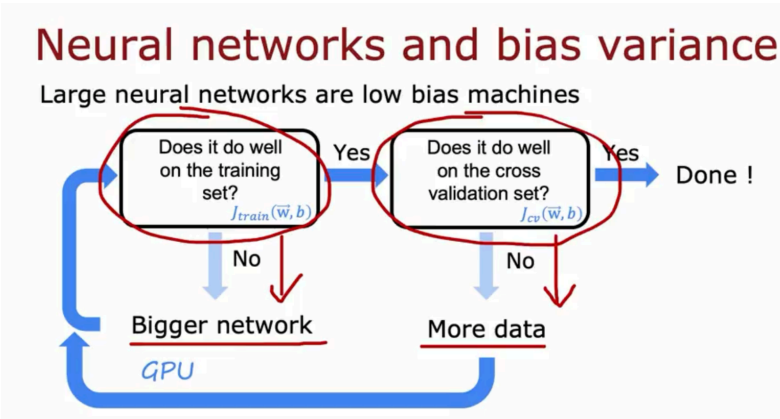
Human level performance : 10.6%
Training error J_{train} : 10.8%
Cross validation error J_{cv} : 14.8%



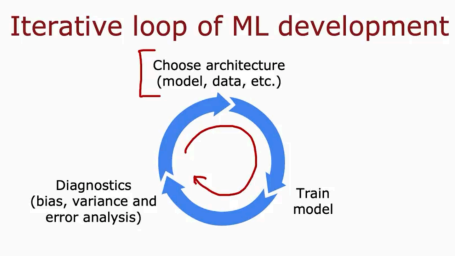
- The larger the training set is, the more difficult it is to find a perfect line to fit all training examples. As such training error tends to be higher the larger the training set is.
- **CV error** tends to be always higher than **training error**
- Plot and compare learning curves of **CV** and **training errors** using only a portion of the full dataset (100 and 200 out of 1000) to discover if the model suffers from overfitting or underfitting early on in the development process
- A large neural network will usually do as well or better than a smaller one so long as regularization for it is chosen appropriately.

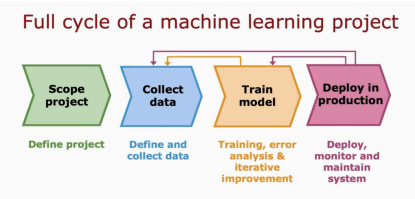


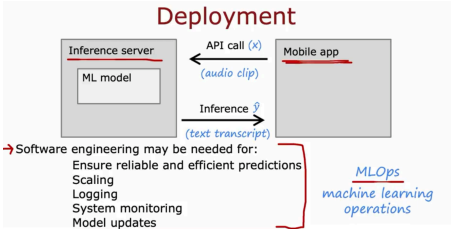




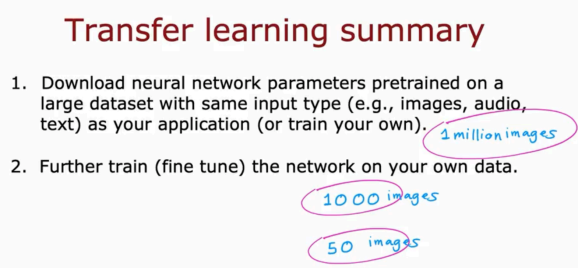
ML Development Process:







- **Error analysis**: After bias/variance, manually examine all the data examples from the CV set that yielded the wrong output and categorize them based on common traits (misclassification of spam). Helps to prioritize what category of errors to prioritize fixing first. For larger datasets, choose a random subset and categorize from there. Applicable when errors can be easily identified by humans, not so much for errors that are difficult for humans to identify (what ad would you click on?)
- Adding data:
    - Instead of gathering data of all types, focus on a subset of data from error prone categories identified from error analysis. **Ex:** Finding unlabeled data of pharmaceutical spam specifically
    - **Data augmentation** of existing data helps create additional data instances that are distorted and have been introduced to noise. Augmentation and noise should be relevant to the data seen on the test set as adding unnecessary noise is meaningless.
    - **Data synthesis**: Generate comparable data artificially. Ex: For optical character recognition, we may use the various fonts that exist in our computers which turn out to be great synthetic data compared to real life characters seen on billboards, flyers, etc.
- Transfer learning is split into 2 steps:
    - **Supervised pre-training** is when a model is trained by another on a huge dataset. The dataset doesn't need to be related to our work, only must be of the same type. This way the weight parameters already have a good starting point.
    - **Fine-tuning** is implementing the dataset we wish to work with into the pre-trained model. Either we can train only the output layer parameter or we can train all parameters of the pre-trained model using the new dataset. This way we can use a much smaller dataset to yield great results with.
- **Pre-trained models** work because when weights are trained on the same type of data, let's say images, they pick up on patterns that are common throughout all kinds of images such as edges, corners and curves. That's why the, despite not being pre-trained on related data, the initial hidden layers have learnt to detect relevant patterns that exists commonly among all data of the same type



Skewed Datasets:

- Use **Precision, Recall** and **F1-Scores** to evaluate datasets (checks for skewed data).
- Use the context and purpose of the data to determine which is more important (precision or recall)

Week 4:

Decision Trees:

- Decision trees split data into respective branches based off of a features **entropy** (purity) and **information gain**

Decision Tree Learning

- Start with all examples at the root node
- Calculate information gain for all possible features, and pick the one with the highest information gain
- Split dataset according to selected feature, and create left and right branches of the tree
- Keep repeating splitting process until stopping criteria is met:
  - When a node is 100% one class
  - When splitting a node will result in the tree exceeding a maximum depth
  - Information gain from additional splits is less than threshold
  - When number of examples in a node is below a threshold

$$H(p_1) = -p_1 log_2(p_1) - p_0 log_2(p_0)$$
$$= -p_1 \, log_2(p_1) - (1 - p_1) log_2(1 - p_1)$$

Information gain

$$= H(p_1^{root}) - \left(w^{left} H\left(p_1^{left}\right) + w^{right} H\left(p_1^{right}\right) \right)$$

- Use **One-Hot Encoding** to address features taking more than two values.

One hot encoding and neural networks

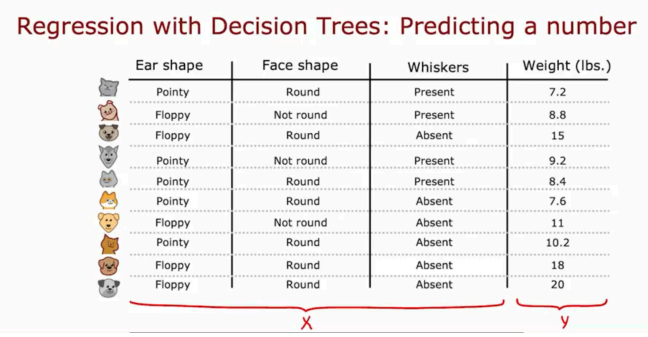
	Pointy ears	Floppy ears	Round ears	Face shape	Whiskers	Cat
	1	0	0	~~Round~~ 1	~~Present~~ 1	1
	0	0	1	~~Not round~~ 0	~~Present~~ 1	1
	0	0	1	~~Round~~ 1	~~Absent~~ 0	0
	1	0	0	~~Not round~~ 0	~~Present~~ 1	0
	0	0	1	~~Round~~ 1	~~Present~~ 1	1
	1	0	0	~~Round~~ 1	~~Absent~~ 0	1
	0	1	0	~~Not round~~ 0	~~Absent~~ 0	1
	0	0	1	~~Round~~ 1	~~Absent~~ 0	1
	0	1	0	~~Round~~ 1	~~Absent~~ 0	1
	0	1	0	~~Round~~ 1	~~Absent~~ 0	1

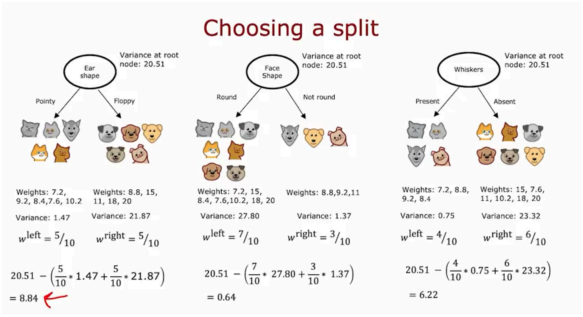
Regression Trees:

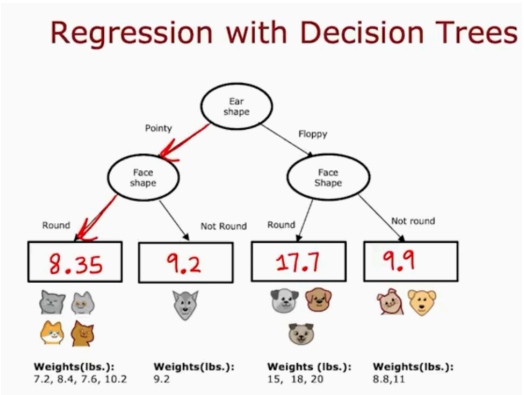
- If a decision tree is required to predict numbers instead of classifying objects, we use regression tree

Ensembling:

- Ensemble decision trees construct multiple decision trees, each yielding different or the same results. The majority result is chosen as the final result of the whole ensemble. This is because decision trees are highly sensitive to data changes meaning we get vastly different trees based on what data we work with first.
- Ensembles are best applied by **creating new test sets using sampling with replacement** on the original dataset, this allows **duplicates of one or multiple data instances to exist in the new training sets**. (It is good for the training sets to have the same number of data instances as the original dataset)
- The number of trees in an ensemble should usually be large but not too large as we get diminishing returns **(going to 1000 is simply inefficient)**
- In an ensemble, a large number of new trees tend to have the **same root feature** making the trees quite similar to each other. **Random forest** fixes this issue by having the trees choose the best feature from a subset of features instead of all the features available introducing randomness to the feature selection. This randomness yields better results due exploring small changes in the data and averaging those changes. as such further little changes to data won't have a huge effect on the output as a whole due to those changes being already explored







- **XGBoost** differs by sampling (with replacement) a higher percentage of misclassified examples from previous trees (similar to focusing on what's lacking and fixing that)
- Choosing Depth:
    - We can leave a large maximum depth but that runs the risk of overfitting
    - Use cross-validation to pick maximum depth, or better yet, open-source libraries
    - Stop splitting if information gain from a split is less than a threshold
    - Stop splitting when a number of examples in a node is below a defined threshold
    -