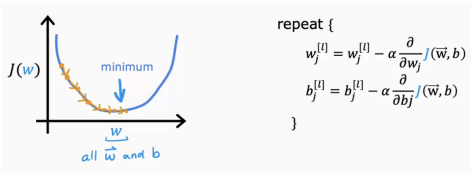
3.Gradient Descent

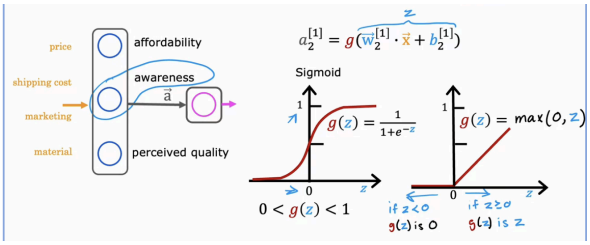
- Minimise cost function
- Compute derivatives using back propagation, using model.fit (X,y,epochs = 100)



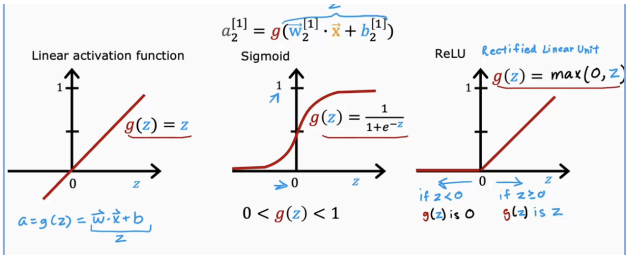
Alternatives to the Sigmoid Acivation

- Recall demand prediction example:

Using sigmoid assumes awareness is binary (aware or not). But it is unlikely to be binary - we can model it as a non-negative number, from 0 up to a very large value. Here, we can use a ReLU (Rectified Linear Unit) function.



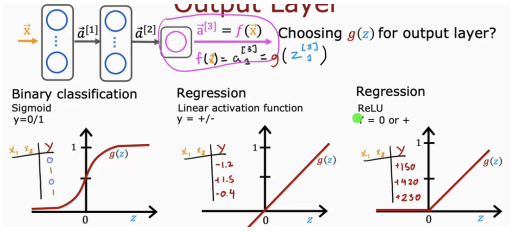
Most commonly used activation functions:



Choosing Activation Functions

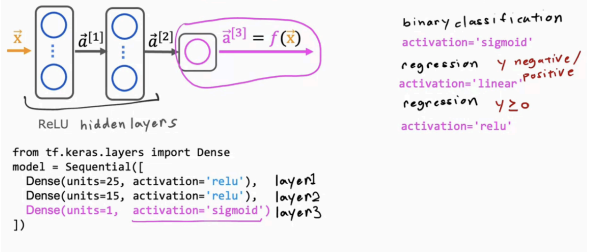
1) Choosing activation function for output layer

Often, there is a natural choice.
Binary classification problem (y=0 or 1) → Sigmoid
Regression problem e.g. stock price prediction —> Linear activation function
If y can only take non-negative values (house price) → ReLU



2) Hidden Layer

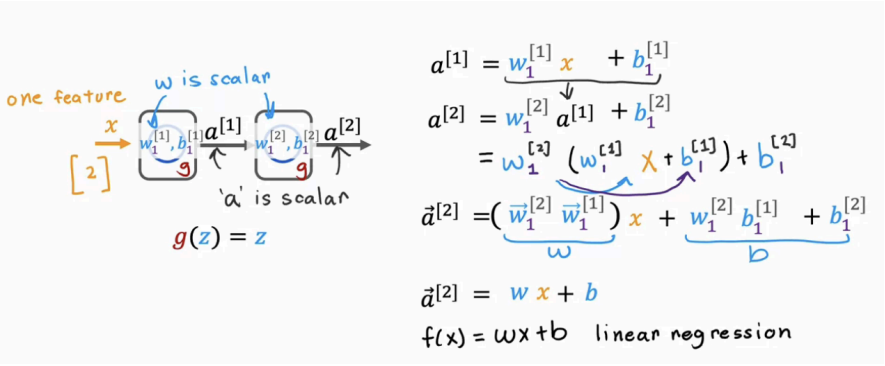
- ReLU generally used, as is easier to compute and goes flat only in one part, compared to two in sigmoid. This results in cost function being flatter, leading to gradient descent being slower



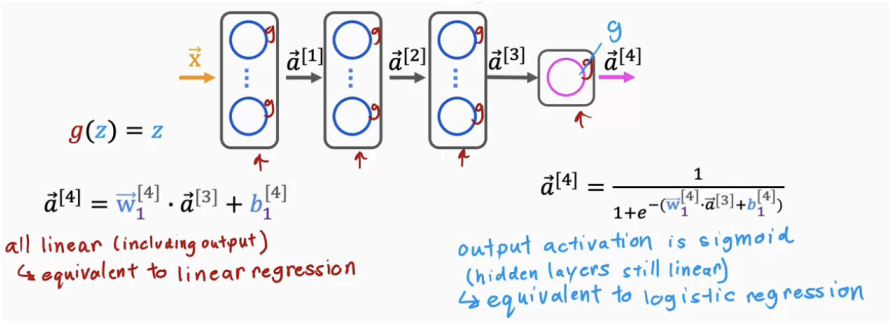
Why do we need activation functions?

If we used a linear activation for all nodes in this neural network, it will become no different from just linear regression. This would defeat the point of using a neural network.

Simpler example: a linear fn of a linear fn is itself a linear fn



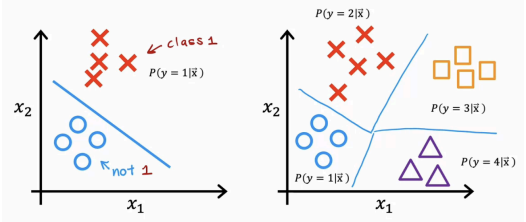
With hidden layers, if all are linear activation fns, you get linear regression. If the output activation is sigmoid, with hidden layers still linear, you just have logistic regression.



Rule of thumb: **Don't use linear activations in hidden layers (use ReLU)**

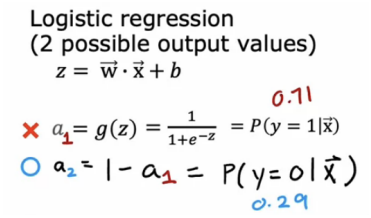
Multiclass Classification

- Classification problem with more than two possible outputs
- With two-class classification, can use logistic regression to estimate decision boundary (softmax); more complex if multiclass



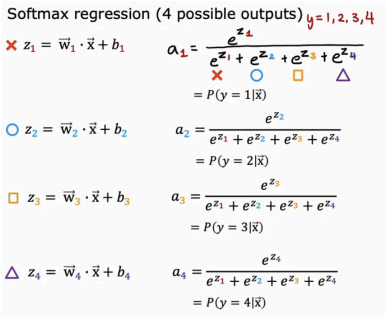
Softmax

Logistic regression applies when y can take on either 0 or 1 as output. (Softmax is a generalisation of logistic regression - if you do softmax on 2 possible outputs, it reduces to logistic regression).



Generalising this to softmax, where y can take on 4 possible outputs:

- a1 is interpreted as the estimate of the chance of y = 1, given the input features x; a2 is the estimated chance of y=2, etc.



Generalising this to N possible outputs where y = 1,2,3,...N:

$$z_j = \vec{w}_j \cdot \vec{x} + b_j \quad j = 1, \ldots, N$$

parameters $w_1, w_2, \ldots, w_N$ $b_1, b_2, \ldots, b_N$

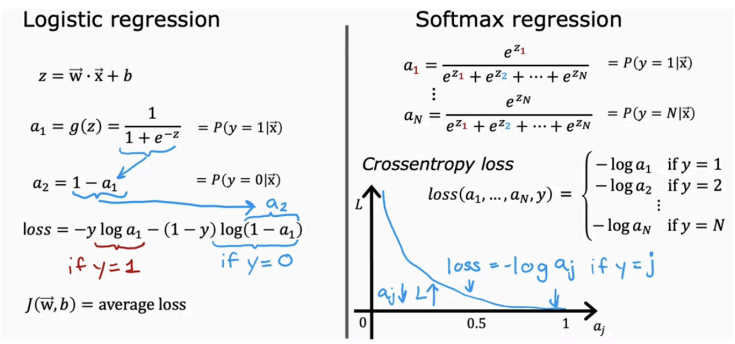
$$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y = j|\vec{x})$$

Where aj is the estimate that y = j. Note that a1 + a2 + … + aN = 1

Cost Function

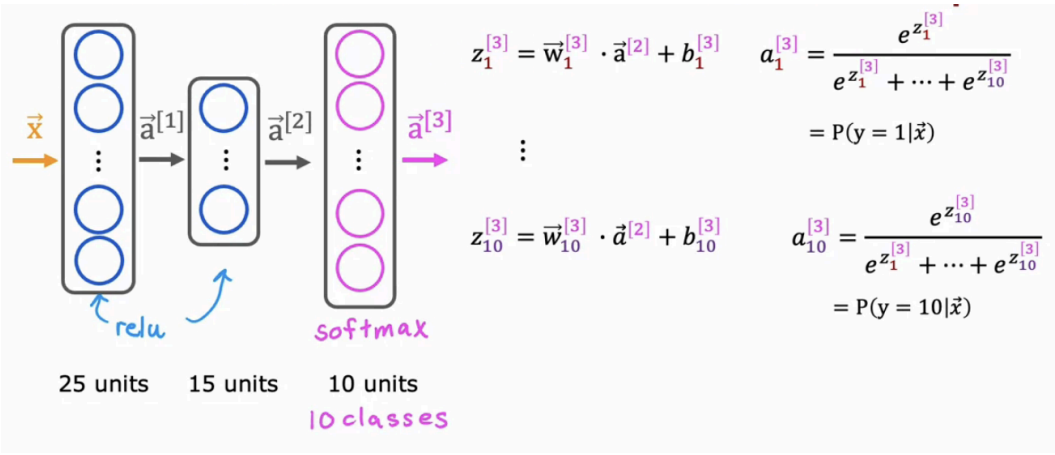
For softmax, the loss for if the algorithm outputs y =1, the loss is the negative log of the probability that y=1 according to algorithm. I.e. if y = j, loss = -logaj.

- If aj very close to 1, loss is very small - smaller aj means bigger loss, incentivising the algorithm to make aj as close to 1 as possible - we want the chance of y being the value said by the algorithm to be large.
- As y can only take on one value in every training example, end up computing -logaj only for one value of aj (whatever the actual value of y = j in that training example e.g. if y=2, computer -loga2).

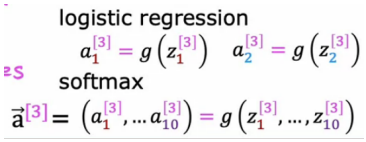


Neural Network with Softmax output (for multiclass classification)

P(y=1|x) refers to chance y is equal to 1.

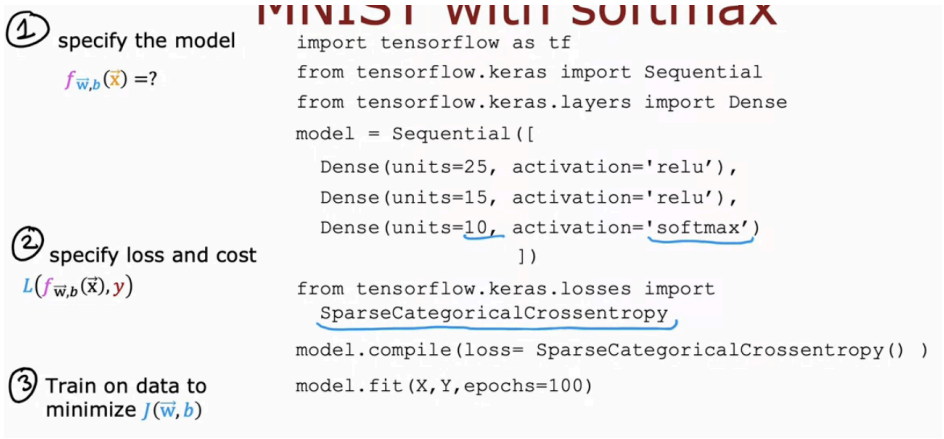


Note that each a in softmax depends on all z, not just a1 depending on z1 as in logistic regression:



Implementation in TensorFlow (3 steps)

- NB for softmax, use a different crossentropy fn (compared to binary crossentropy in logistic regression)

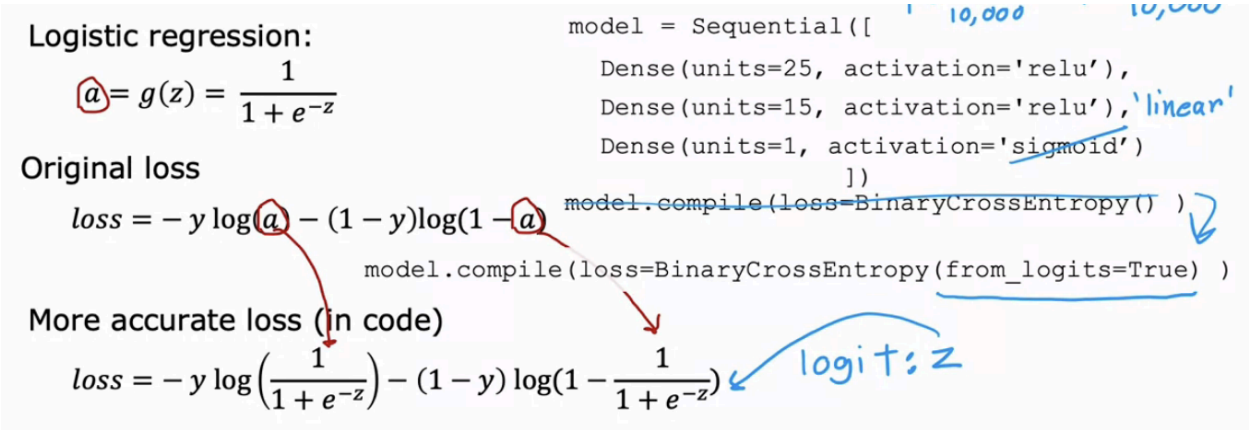


NB there is a better version written later.

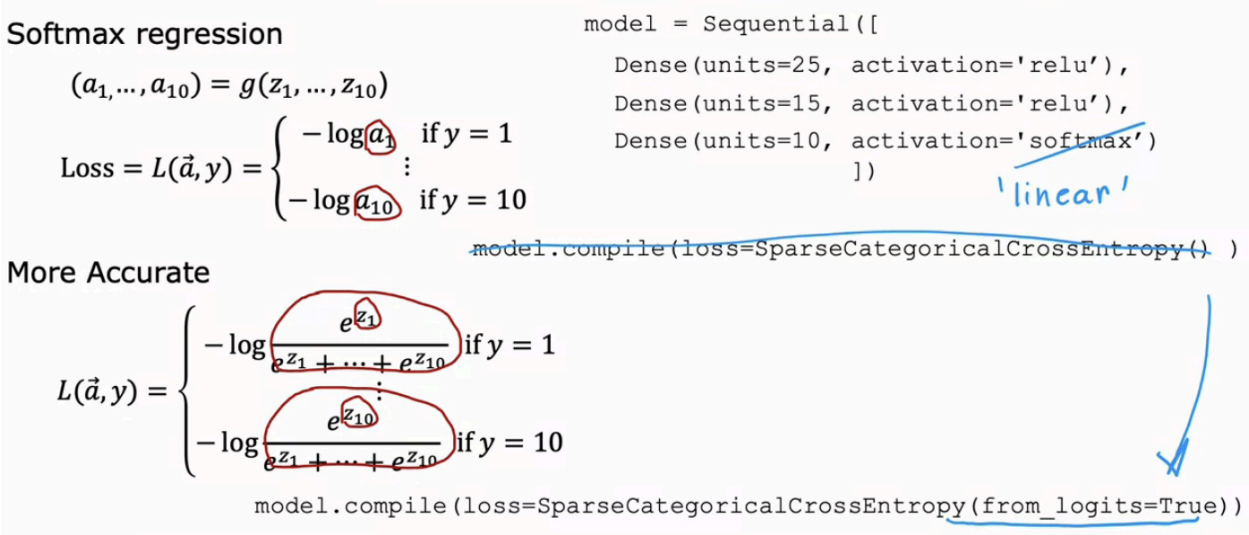
Improved Implementation of SoftMax

In Logistic regression:

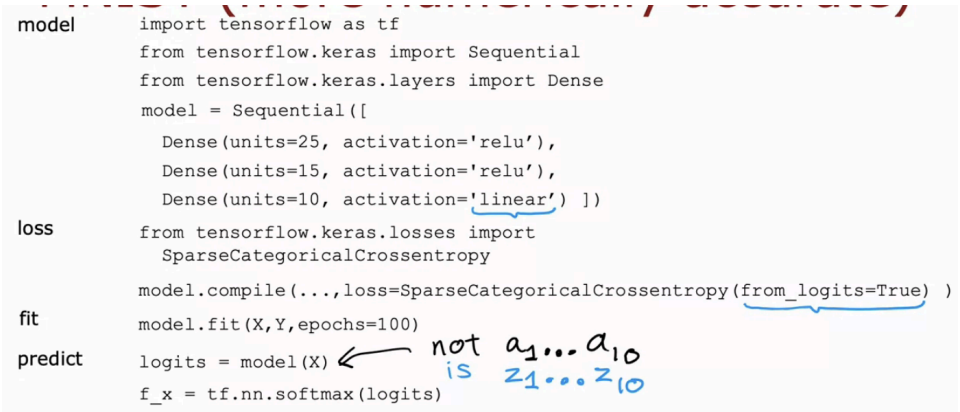
- To compute loss function, first compute a and then compute (binary crossentropy) loss:



This reduces numerical roundoff errors. Applying this to softmax, instead specify formula to give tensorflow ability to rearrange terms and calculate more accurately…



We have now changed the neural network to use a linear activation function rather than softmax, so the neural network's final layer no longer outputs a1…a10 but rather z1…z10.



Similarly, for logistic regression have to change the code…

model

```
model = Sequential([
  Dense(units=25, activation='sigmoid'),
  Dense(units=15, activation='sigmoid'),
  Dense(units=1, activation='linear')
                  ])
```

loss

```
from tensorflow.keras.losses import
  BinaryCrossentropy
model.compile(..., BinaryCrossentropy(from_logits=True))
model.fit(X,Y,epochs=100)
```

fit

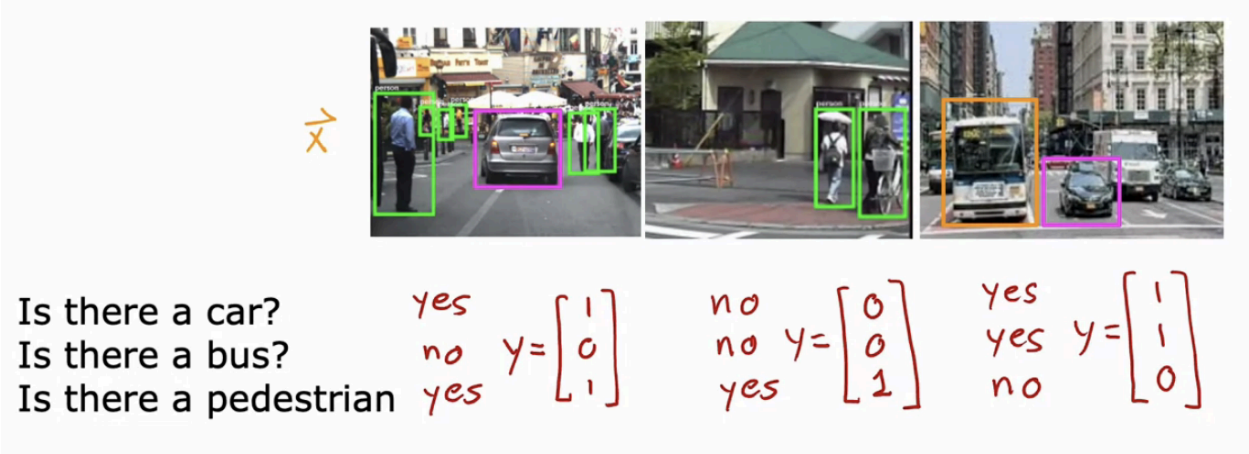
```
logit = model(X)
```

predict

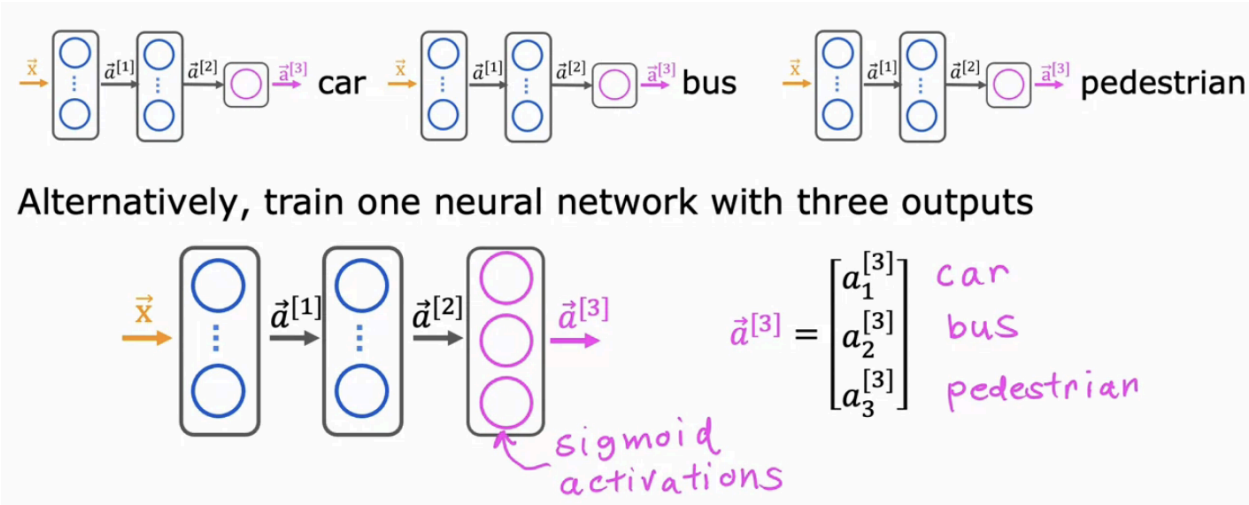
```
f_x = tf.nn.sigmoid(logit)
```

Classification with multiple outputs (multilabel)

- Could be multiple labels associated with each image e.g. is there a car? Is there a bus? Is there a pedestrian? (associated with a single input image x, there are three possible labels)



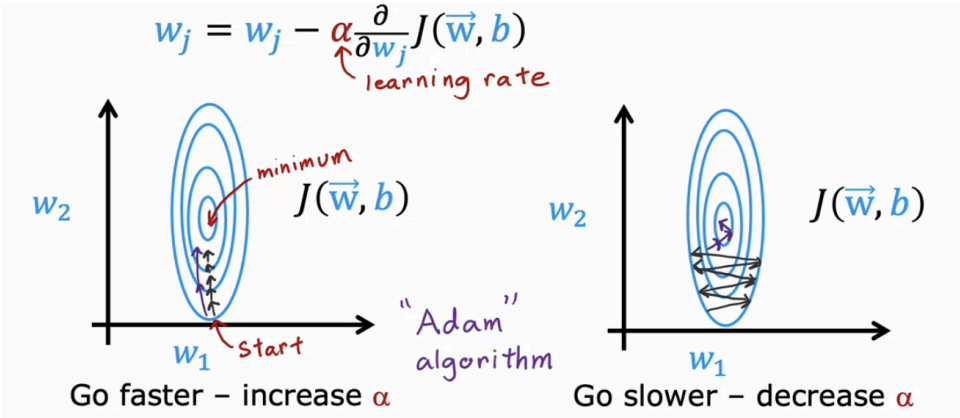
Could do three separate neural networks…



NOT THE SAME AS MULTICLASS CLASSIFICATION.

Advanced Optimisation

- There exist even better training models than gradient descent. “Adam algorithm” automatically adjusts learning rate to increase or decrease alpha, optimising grad. descent

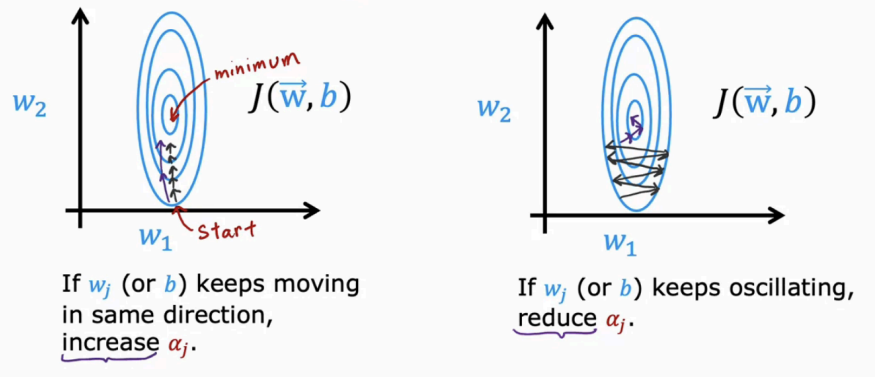


- It does not have just one alpha, uses a different learning rate for every parameter of your model:

Adam: Adaptive Moment estimation not just one α

$$w_1 = w_1 - \alpha_1 \frac{\partial}{\partial w_1} J(\vec{w}, b)$$
$$\vdots$$
$$w_{10} = w_{10} - \alpha_{10} \frac{\partial}{\partial w_{10}} J(\vec{w}, b)$$
$$b = b - \alpha_{11} \frac{\partial}{\partial b} J(\vec{w}, b)$$

- Intuition of adam algorithm:



- Implementing this in code: (VERY STANDARD FOR TRAINING NETWORKS)

model

```
model = Sequential([
        tf.keras.layers.Dense(units=25, activation='sigmoid'),
        tf.keras.layers.Dense(units=15, activation='sigmoid'),
        tf.keras.layers.Dense(units=10, activation='linear')
])
```

compile

$\alpha = 10^{-3} = 0.001$

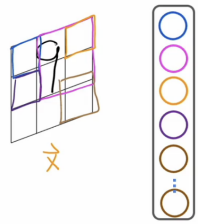
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

```
model.fit(X,Y,epochs=100)
```

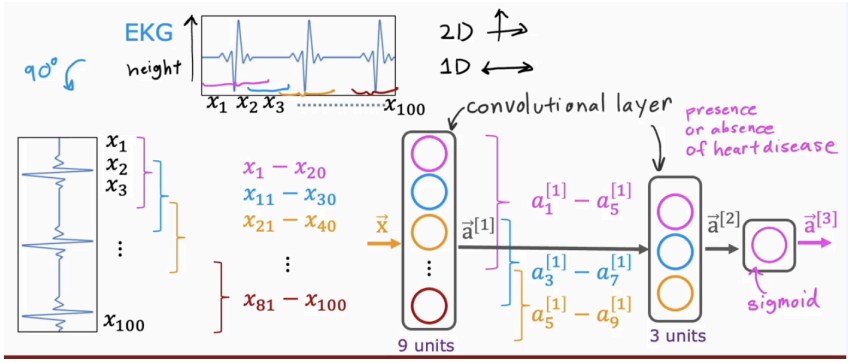
Most practitioners will use Adam rather than gradient descent.

Additional Layer Types



- All so far have been **dense** layers - each neuron output is a function of all the activation outputs of the previous layer.
- **Convolutional layer** - Each neuron only looks at part of the previous layer's outputs. Leads to faster computation + needs less training data (less prone to overfitting)

E.g. for looking at a heart beat signal, where the different hidden units in the convolutional layer each look at a different part of the signal. Then in the second layer only looks at a certain number of the activations.



Week 3

1) *Debugging a learning algorithm*

You've implemented regularized linear regression on housing prices

$$J(\vec{w}, b) = \frac{1}{2m}\sum_{i=1}^{m}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$$

But it makes unacceptably large errors in predictions. What do you try next?

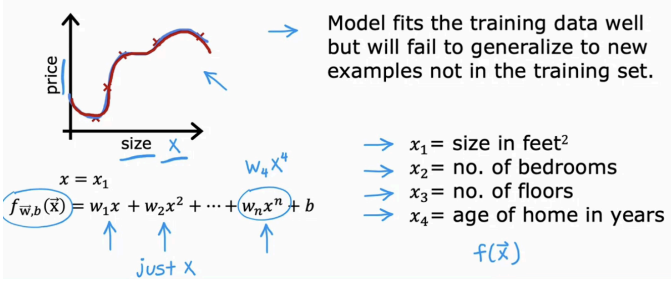
- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, etc)$
- Try decreasing λ
- Try increasing λ

Machine Learning Diagnostics

Evaluating a model

E.g. housing prices

If we have two features, easy to just plot - here, we see that line is v. curvy so will not generalise well. With four features etc., however, cannot plot easily so need another systematic way to evaluate strength of model.



Solution:

1) Use a training and test set

Dataset:

	size	price
70%	2104	400
	1600	330
	2400	369
	1416	232
	3000	540
	1985	300
	1534	315
30%	1427	199
	1380	212
	1494	243

training set → $(x^{(1)}, y^{(1)})$, $(x^{(2)}, y^{(2)})$, ⋮, $(x^{(m_{train})}, y^{(m_{train})})$

m_{train} = no. training examples = 7

test set → $(x_{test}^{(1)}, y_{test}^{(1)})$, ⋮, $(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

m_{test} = no. test examples = 3

2) Train/ test procedure for linear regression (with squared error cost)
   - Training error is how well algorithm is doing on training set

Fit parameters by minimizing cost function $J(\vec{w}, b)$

$$J(\vec{w}, b) = \left[\frac{1}{2m_{train}}\sum_{i=1}^{m_{train}}\left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m_{train}}\sum_{j=1}^{n} w_j^2\right]$$

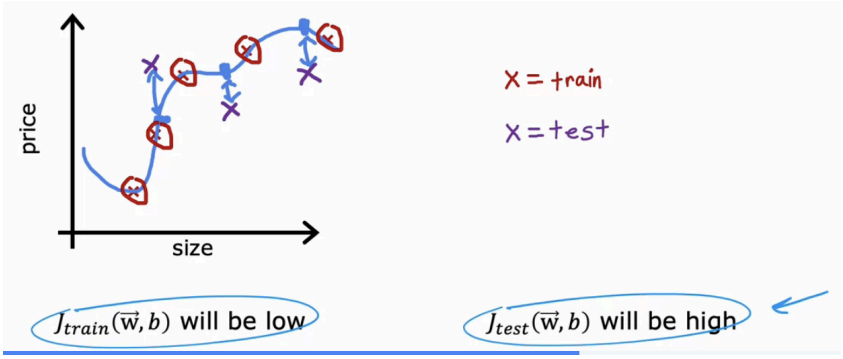
Compute test error:

$$J_{test}(\vec{w}, b) = \frac{1}{2m_{test}}\left[\sum_{i=1}^{m_{test}}\left(f_{\vec{w},b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)}\right)^2\right]$$

Compute training error:

$$J_{train}(\vec{w}, b) = \frac{1}{2m_{train}}\left[\sum_{i=1}^{m_{train}}\left(f_{\vec{w},b}(\vec{x}_{train}^{(i)}) - y_{train}^{(i)}\right)^2\right]$$

- Value of Jtest is a good measure of how well model is generalising. E.g. if it fits training set sell but generalises poorly, Jtrain will be low but Jtest will be high:



How do we apply this to a classification problem?

Fit parameters by minimizing $J(\vec{w}, b)$ to find $\vec{w}, b$

E.g.,

$$J(\vec{w}, b) = -\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} \left[y^{(i)} \log\left(f_{\vec{w},b}(\vec{x}^{(i)})\right) + (1 - y^{(i)}) \log\left(1 - f_{\vec{w},b}(\vec{x}^{(i)})\right) \right] + \frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2$$

Compute test error:

$$J_{test}(\vec{w}, b) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} \left[y_{test}^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right)\right) + \left(1 - y_{test}^{(i)}\right) \log\left(1 - f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right)\right) \right]$$

Compute train error:

$$J_{train}(\vec{w}, b) = -\frac{1}{m_{train}} \sum_{i=1}^{m_{train}} \left[y_{train}^{(i)} \log\left(f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right)\right) + \left(1 - y_{train}^{(i)}\right) \log\left(1 - f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right)\right) \right]$$

But more commonly for classification problems, instead of using logistic loss to measure error, instead measure what is the **fraction of the test set and the fraction of the train set that the algorithm has misclassified.**

$$\hat{y} = \begin{cases} 1 \text{ if } f_{\vec{w},b}(\vec{x}^{(i)}) \geq 0.5 \\ 0 \text{ if } f_{\vec{w},b}(\vec{x}^{(i)}) < 0.5 \end{cases}$$

count $\hat{y} \neq y$

$J_{test}(\vec{w}, b)$ is the fraction of the test set that has been misclassified.

$J_{train}(\vec{w}, b)$ is the fraction of the train set that has been misclassified.

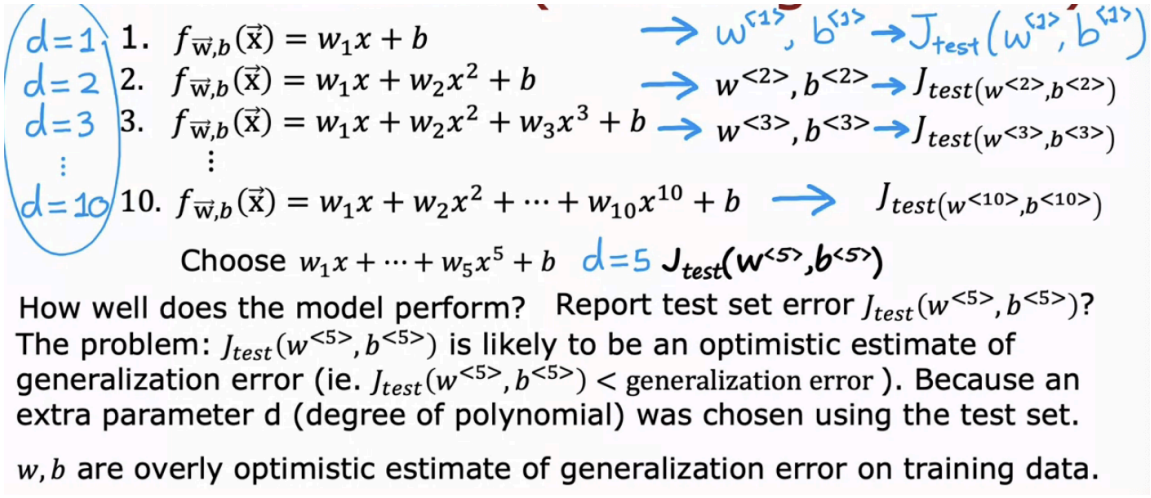
I.e. algo makes a prediction 1 or 0 on each example; count up number of examples where y hat is not equal to the correct output y.

Model selection and training/ cross validation/ test sets

How can we automatically choose a model?

- Once parameters w,b are fit to training set, the training error Jtrain is likely lower than the actual generalisation error. Thus, Jtest is a better estimate of how well the model will generalise to new data compared to Jtrain

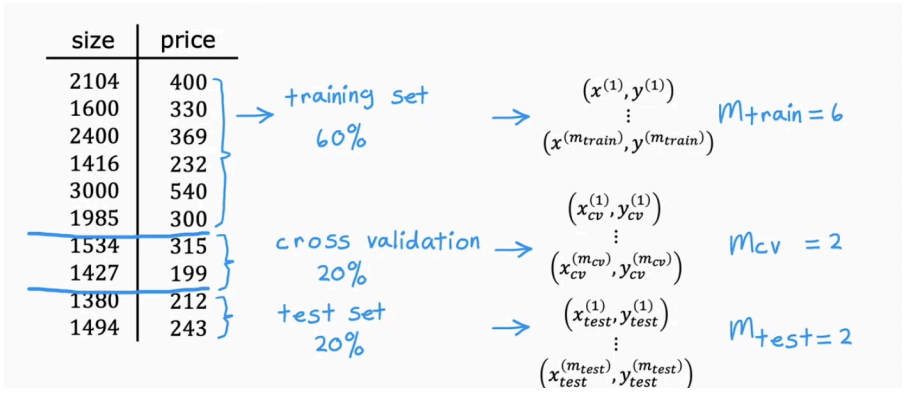
Possible Option: Look at all Jtest and see which gives lowest value



- Have essentially fitted an extra parameter using the test set. Thus, this is a flawed method.

Modification of procedure:

- Split into three: training set, cross validation (/development/dev/validation) set, and test set



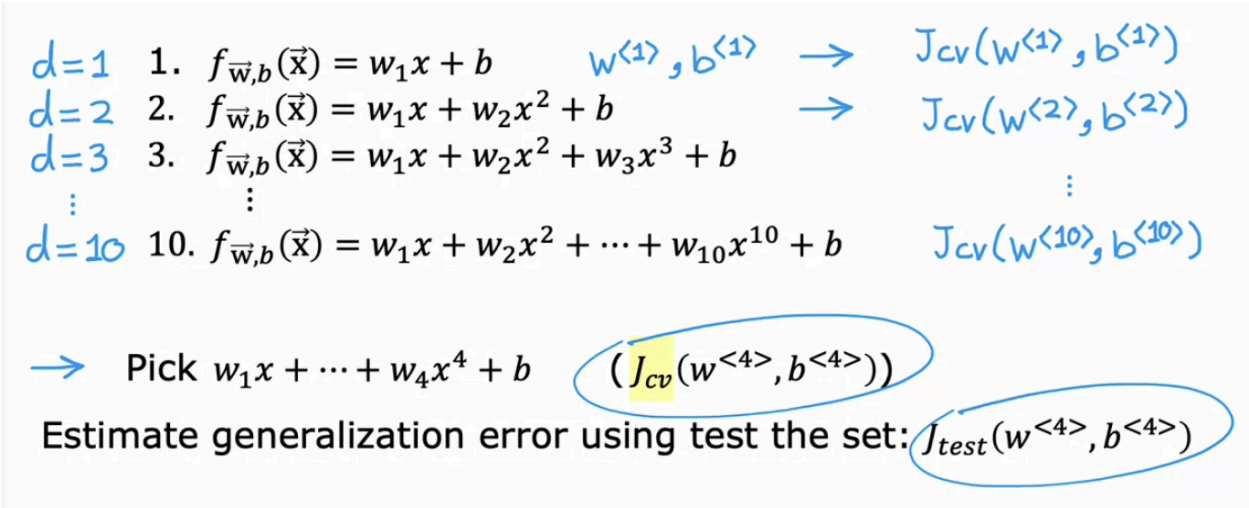
- Then compute the training error, cross validation error, and test error:

Training error: $J_{train}(\vec{w}, b) = \frac{1}{2m_{train}}\left[\sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2\right]$

Cross validation error: $J_{cv}(\vec{w}, b) = \frac{1}{2m_{cv}}\left[\sum_{i=1}^{m_{cv}} \left(f_{\vec{w},b}(\vec{x}_{cv}^{(i)}) - y_{cv}^{(i)}\right)^2\right]$ (validation error, dev error)

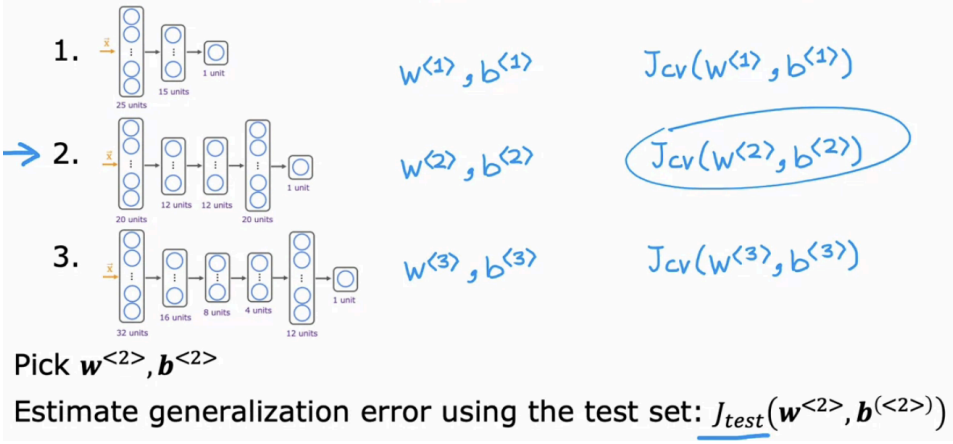
Test error: $J_{test}(\vec{w}, b) = \frac{1}{2m_{test}}\left[\sum_{i=1}^{m_{test}} \left(f_{\vec{w},b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)}\right)^2\right]$

- Then can go about model selection, using the cross validation set to see which model has the lowest cross validation error:



- Jtest here is a fair estimate of the generalisation error because up until this point you have not fit w, b, or d to the test set

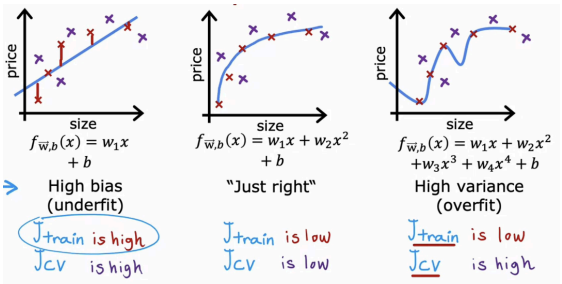
This model selection also applies to neural network architecture selection…



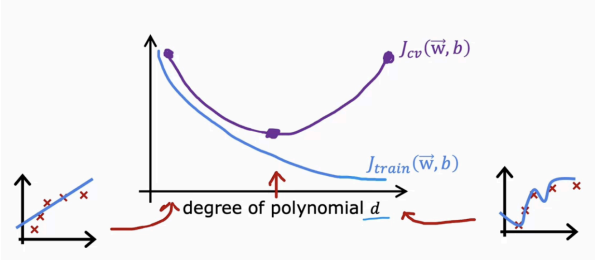
- As this is classification, evaluate using fraction of y hat = y

Diagnosing Bias and Variance

- Good guidance on what to try next when something is working poorly.
- Jtrain & Jcv high → high bias; Jtrain low, Jcv high → high variance



Understand bias and variance:



High bias (underfit)
→ J_{train} will be high
($J_{train} \approx J_{cv}$)

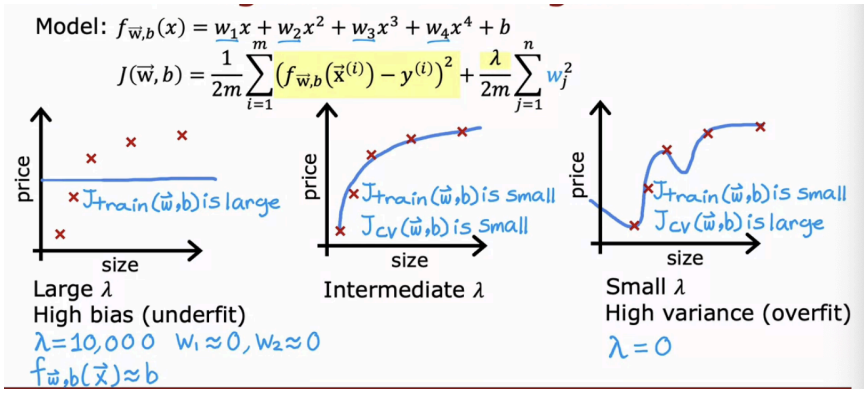
High variance (overfit)
→ $J_{cv} \gg J_{train}$
(J_{train} may be low)

High bias and high variance
J_{train} will be high
and $J_{cv} \gg J_{train}$

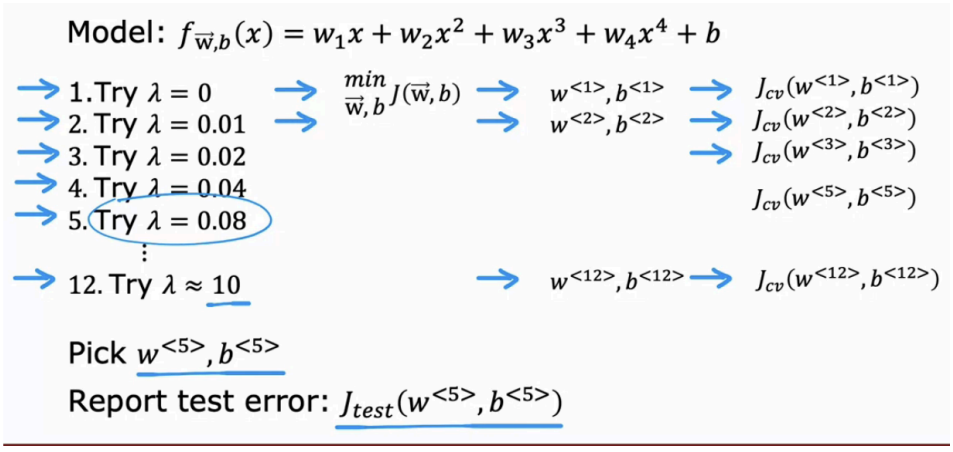
Regularization and bias/variance

- How choice of regularisation parameter lamda affects bias and variance
- Trade off of keeping w small (large lambda) and fitting well (low lambda)

Linear regression with regularisation:

- 

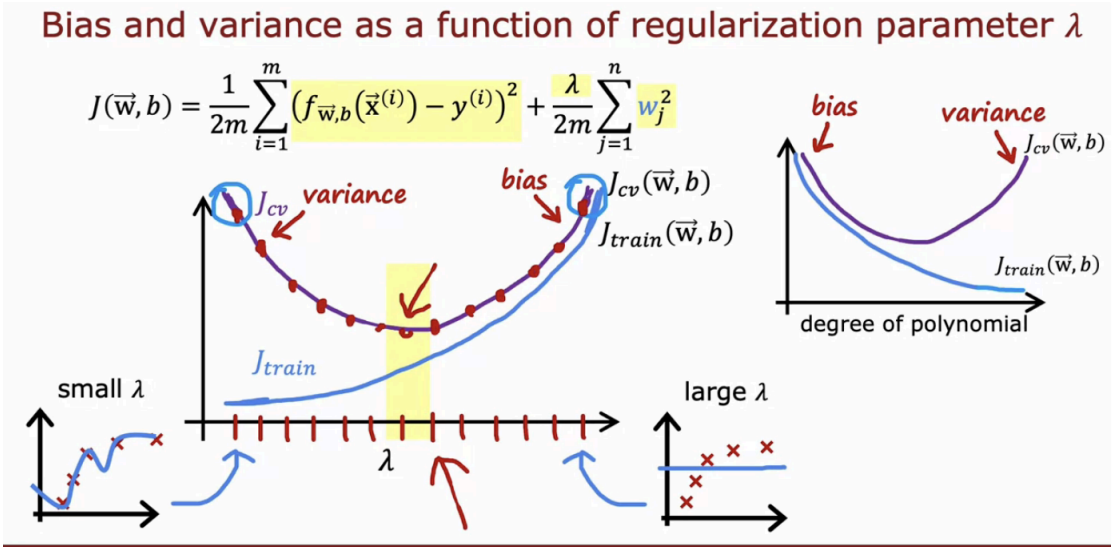
How do we choose a good value of lambda (similar to cross validation)?



Looking at how bias and variance vary as a function of lambda:

– Jtrain: will increase, as in optimisation cost fn, the larger lambda, the more it is trying to keep w small, so the worse it does on the training set, so the train error increases

- Jcv overfits on left, and underfits on right



Establishing a baseline level of performance

E.g. speech recognition (training error = % of incorrect transcriptions)

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

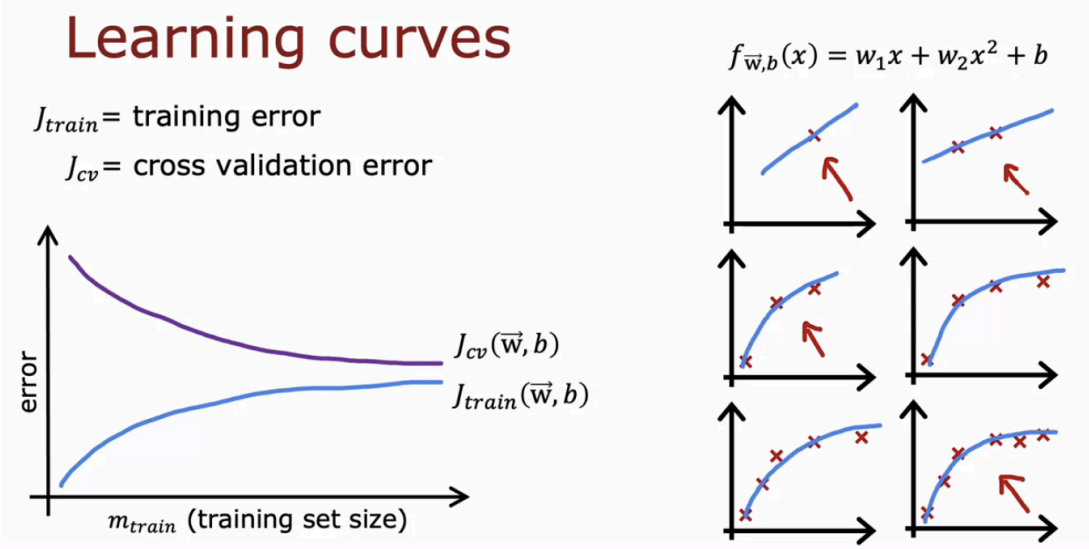
Algo doing quite well on training set, but cross validation higher - so has a variance problem.

Baseline level of performance:

- Human level performance
- Competing algorithms performance
- Guess based on experience

- Gap between baseline and training error large = bias problem
- Gap between training error and cross validation error large = variance problem
- Can have both problems simultaneously

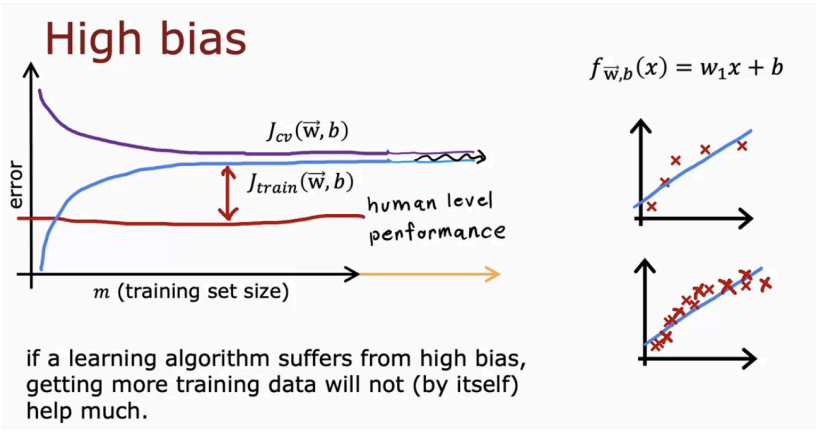
Learning Curves



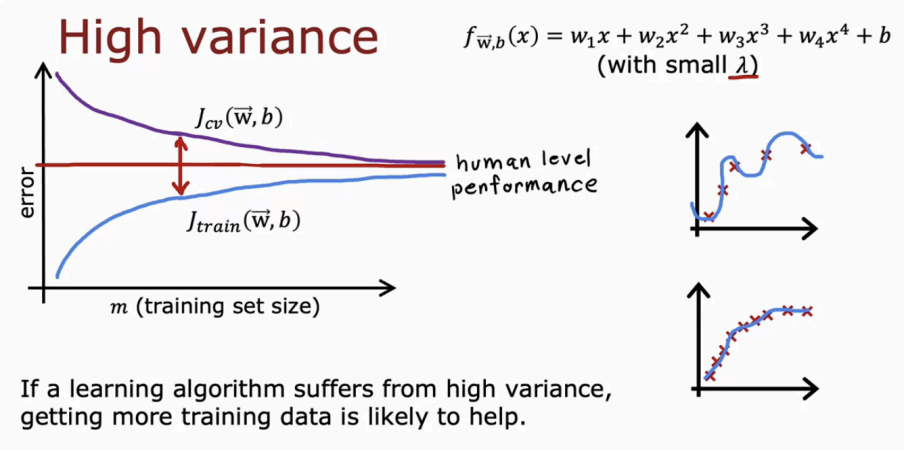
- More training examples → harder to fit all training examples perfectly → Jtrain increases
- Jcv > Jtrain, as expect to do at least slightly better on training set than cross validation set

For high bias:

- Plateaus because fitting a straight line won't change up with more examples - it's such a simple model

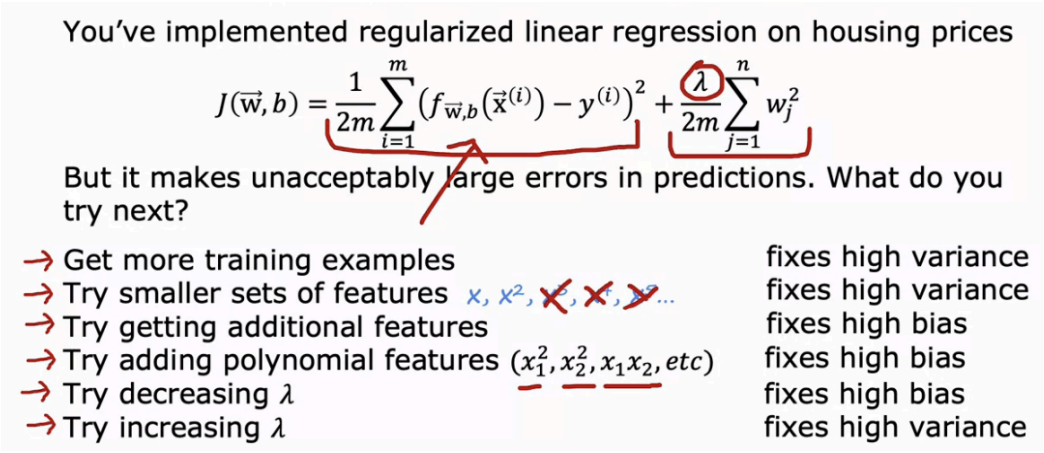


For high variance:



Deciding what to do next

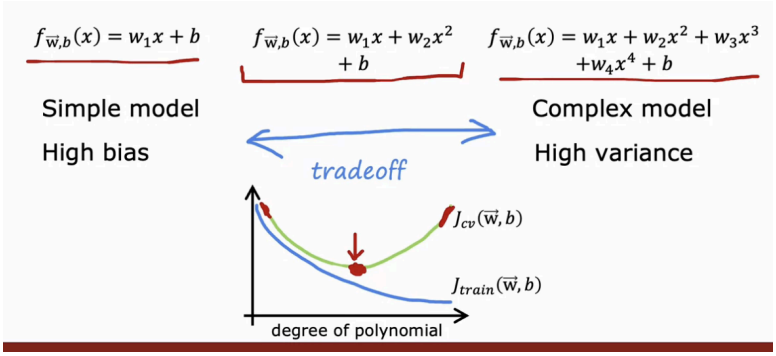
Debugging a learning algorithm



Bias/ variance and neural networks

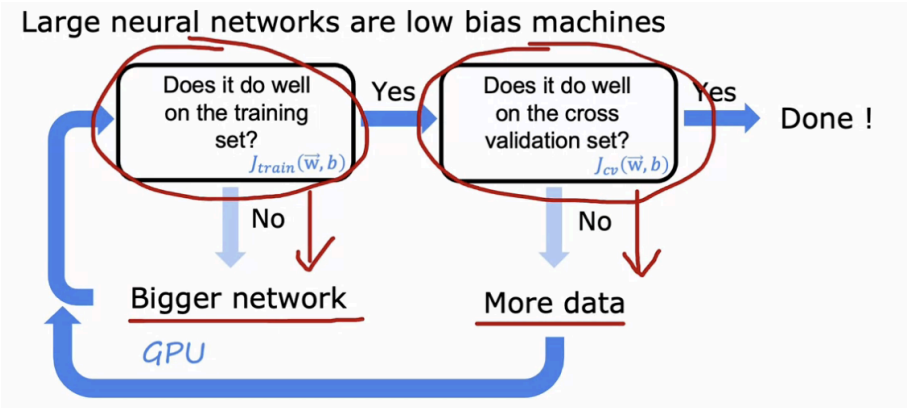
The bias variance tradeoff

- Go in the middle to pick model with lowest cross-validation error



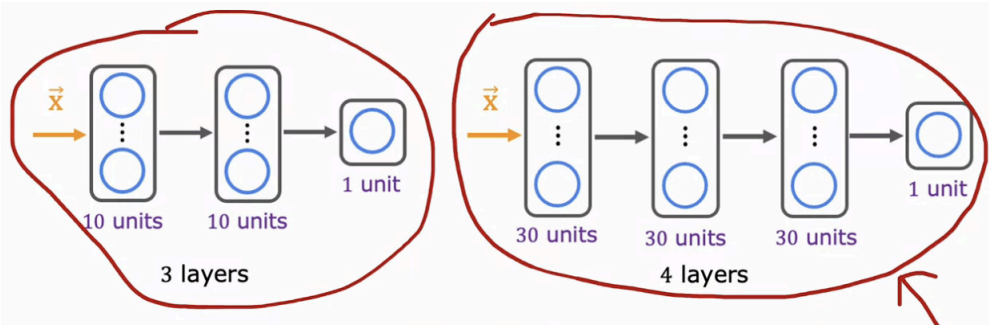
Neural networks offer us a way out of this:

- **Large neural networks are low bias machines (don't need to trade off)**

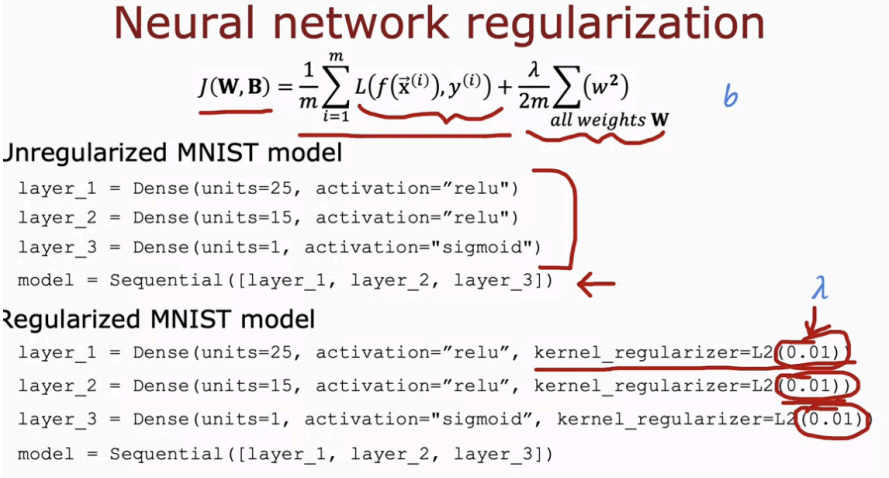


Neural networks and regularization:

- **A large neural network will usually do as well or better than a smaller one so long as regularization is chosen appropriately (does not overfit)**

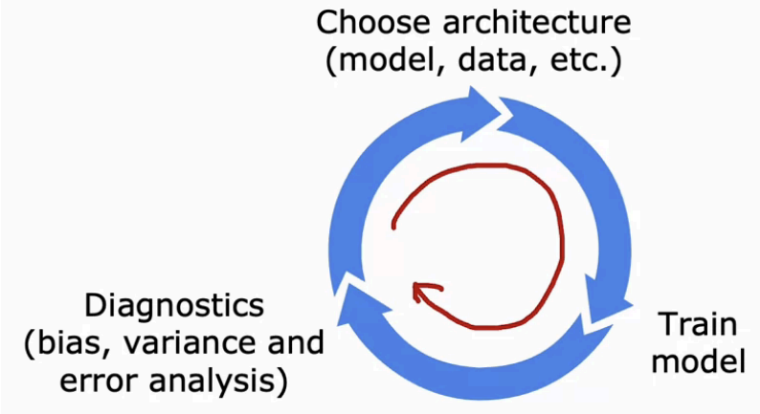


To regularise it practically in TensorFlow:



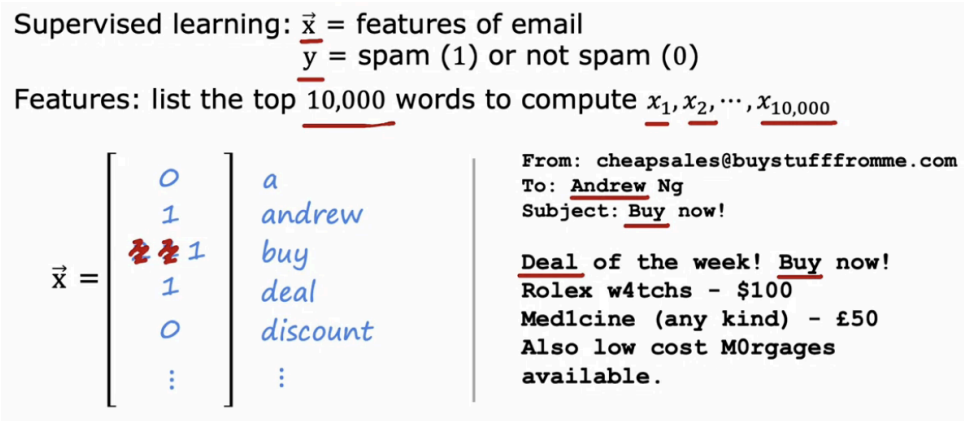
- Often you fight variance problems rather than bias problems with a large neural network

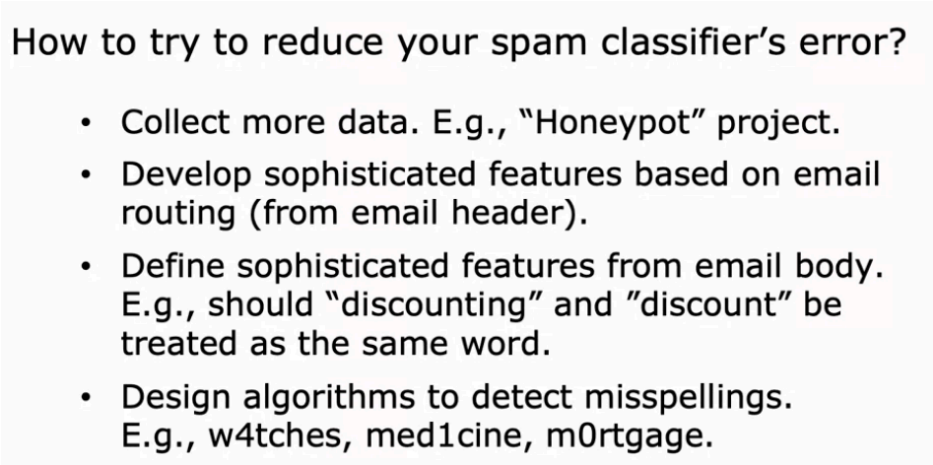
<u>Iterative loop of ML development</u>



Building a spam classifier

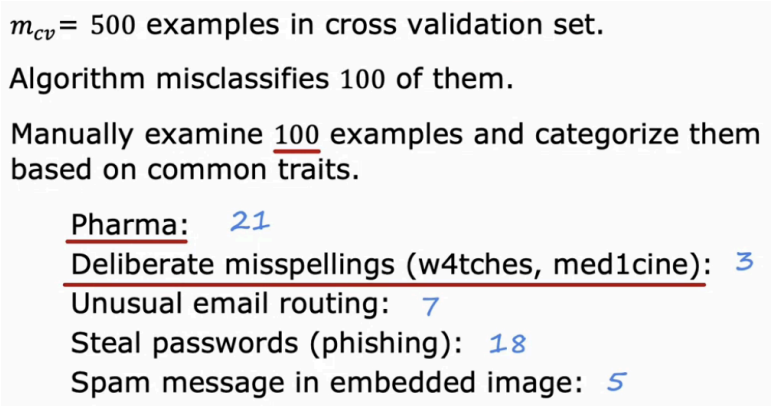
- Text classification
- Set words that appear to 1, ones that do not to 0
- Can then train model to predict y from x





Error Analysis

- Manually examine errors: (if very large misclassified set, randomly sample them)



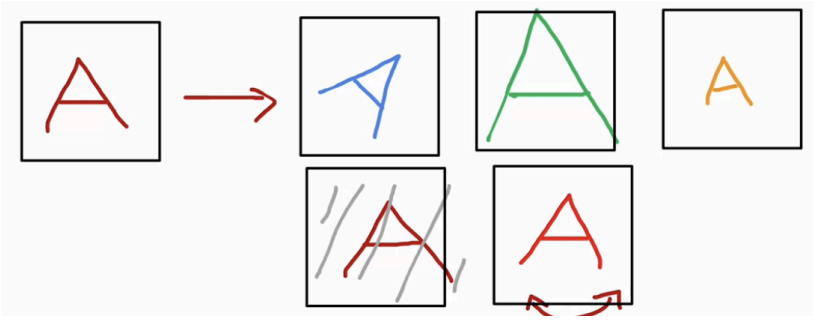
- Identify where the major errors are occurring - here, pharmaceutical spam emails. Specifically target these e.g. look at more data features here of these specific email types

Adding Data

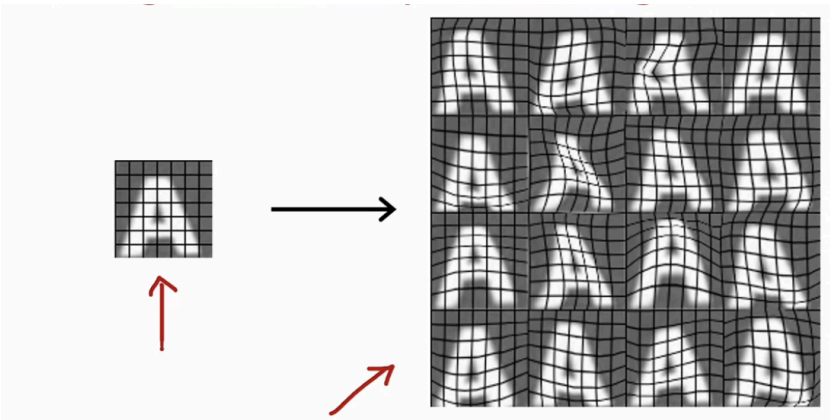
- **Add more data of the types where error analysis has indicated it might help.**

E.g. go to unlabeled data and find more examples of pharma related spam.

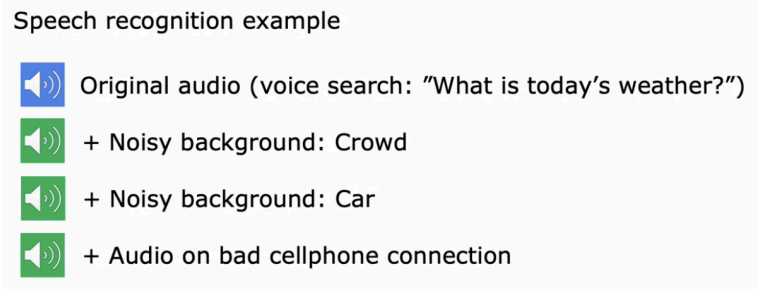
1) **Data augmentation:** modifying an existing training example to create a new training example.



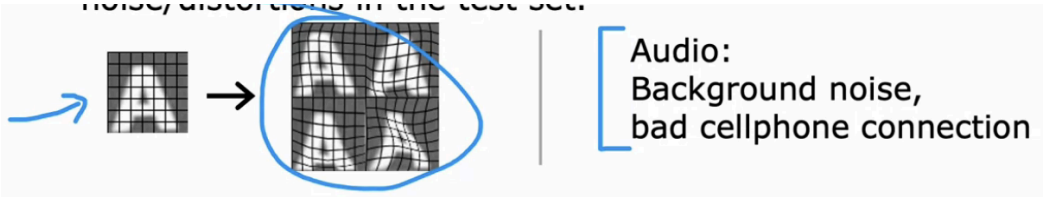
- Can also do data augmentation by introducing distortions



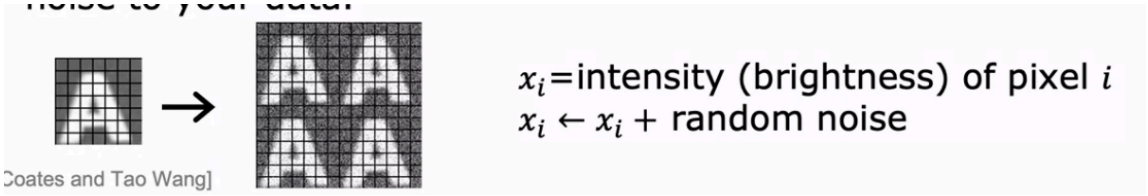
- Can also apply data augmentation to speech data
  - Put noisy backgrounds or crackles etc. (to take one audio clip to create three training examples)



Note that distortions introduced should be representative of the type of noise/ distortions in the test set.

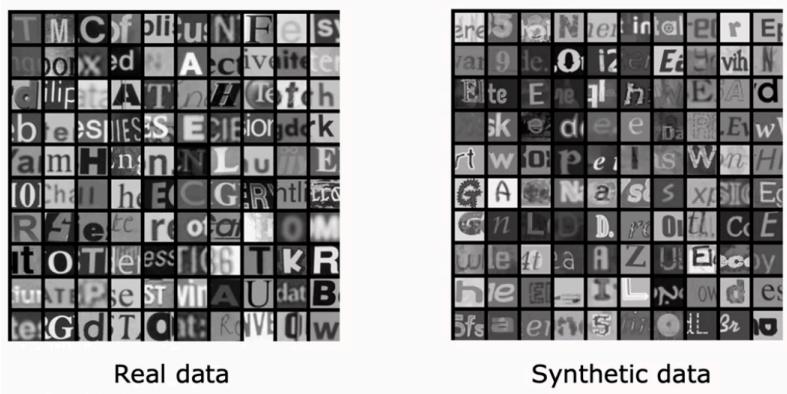


- Usually does not help to add purely random/ meaningless noise to your data

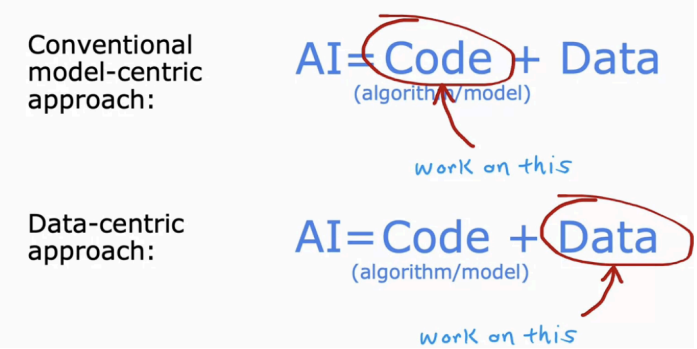


2) **Data synthesis**: using artificial data inputs to create a new training example. Largely used for computer vision.

E.g. photo OCR. Can create artificial data by using different fonts and screenshot it with different backgrounds.



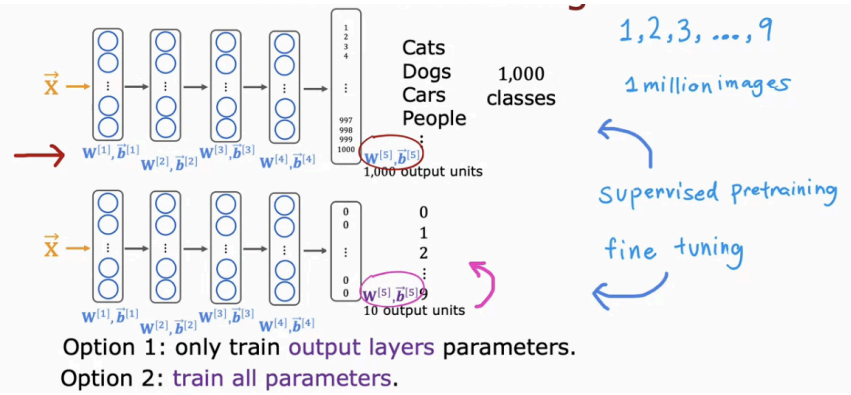
These are all types of data engineering:



When there is not much data, can use:

Transfer learning: Using data from a different task

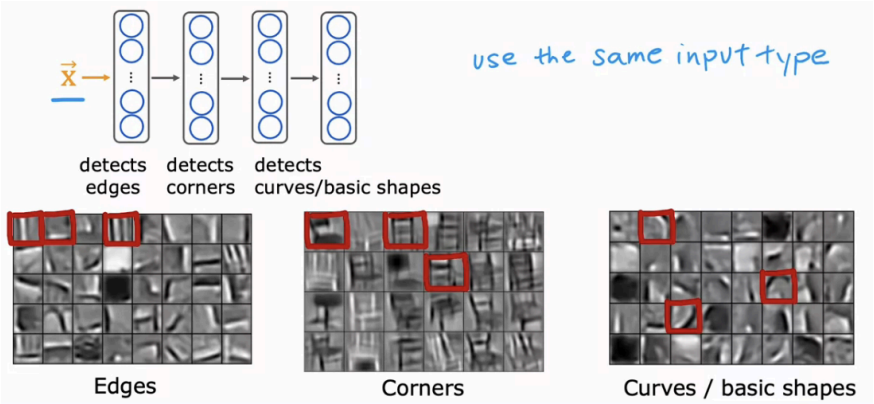
- Want to recognise handwritten digits of 1 to 9; find 1 million images, start to train learning algorithm on 1000 classes
- For transfer learning, make a copy of this neural network with the same w1, b1, w2, b2 etc. but replace the output layer with only 10 output units and a new w5,b5
- So two options for training this neural network's parameters
    - Option 1: only train output layers parameters, and only update w5,b5 to lower cost function (best for a small training set)
    - Option 2: train all parameters in the network; first four layers would be initialised using previous values (best for a larger training set)



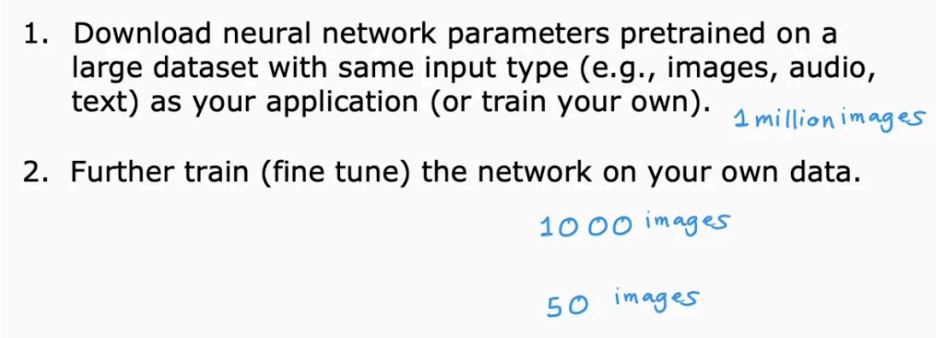
- Can use this to make the most of pre-trained neural networks online

Why does transfer learning work?

- In image recognition, first level might learn to detect edges, the second to detect corders, the third to detect curves/basic shapes etc.
- Restriction: need to use same input type as the input the network was pretrained on

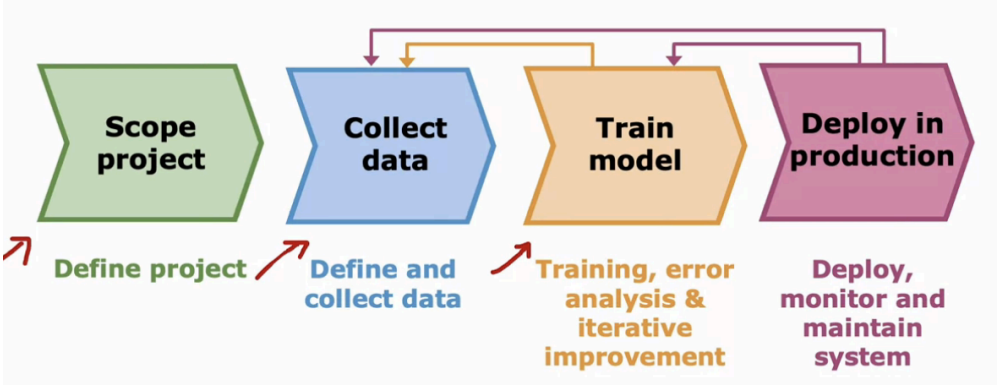


Summary of transfer learning:



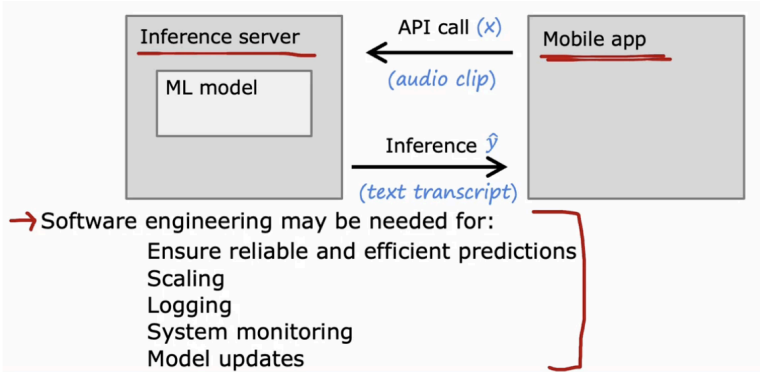
Full Cycle of a Machine Learning Project

Using the example of speech recognition…



What is deployment?

- Implement ML model in an inference server. If team has implemented a mobile ask, when a user talks to app, mobile app can make an API call to parse to inference server, which can then apply the ML model and return the inference prediction



MLOps - how to systematically build, deploy, and maintain ML systems.

Fairness, bias, and ethics

Bias:

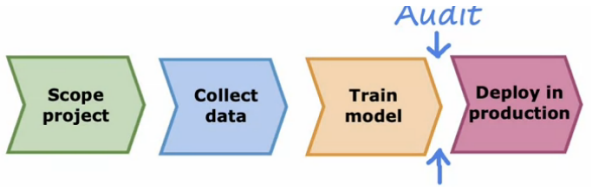
- Hiring tool that discriminates against women
- Facial recognition system matching dark skinned individuals to criminal mugshots
- Biased bank loan approvals
- Toxic effect of reinforcing negative stereotypes

Adverse use cases of ML algorithms:

- Buzzfeed released deepfake of Obama
- Social media spreading toxic/incendiary speech through optimizing for engagement
- Generating fake content for commercial and political purposes
- Using ML to build harmful products, commit fraud etc.
    - Spam vs anti-spam: fraud vs anti-fraud

Guidelines:

- Get a diverse team to brainstorm things that might go wrong, with emphasis on possible harm to vulnerable groups.
- Carry out literature search on standards/ guidelines for your industry
- Audit systems against possible harm prior to deployment



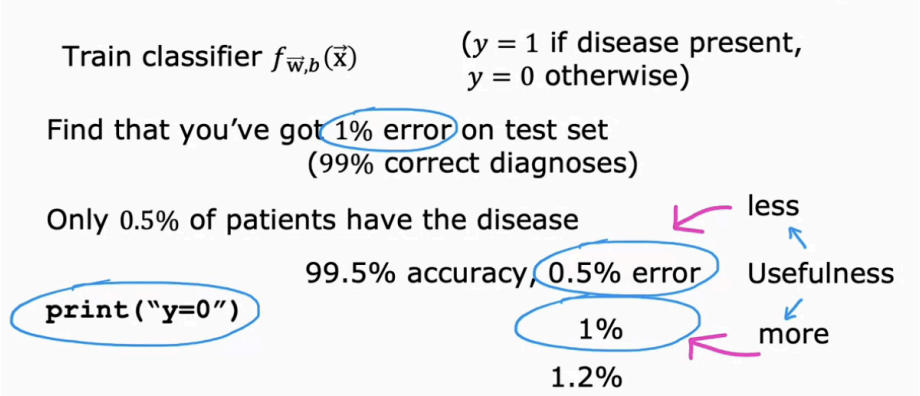
- Develop mitigation plan (if applicable) and, after deployment, monitor for possible harm.

Error metrics for skewed datasets

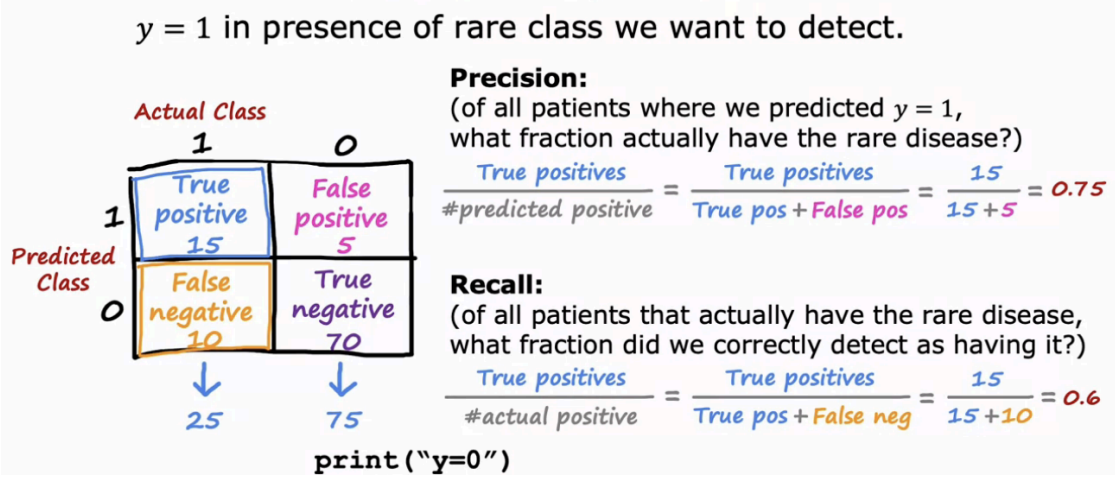
For situations where ratio of positive to negative examples is v. skewed, usual error metrics like accuracy do not work that well.

E.g. Rate disease classification

If you just printed y=0 all the time, would have 99.5% accuracy in this case. Accuracy is relevant:

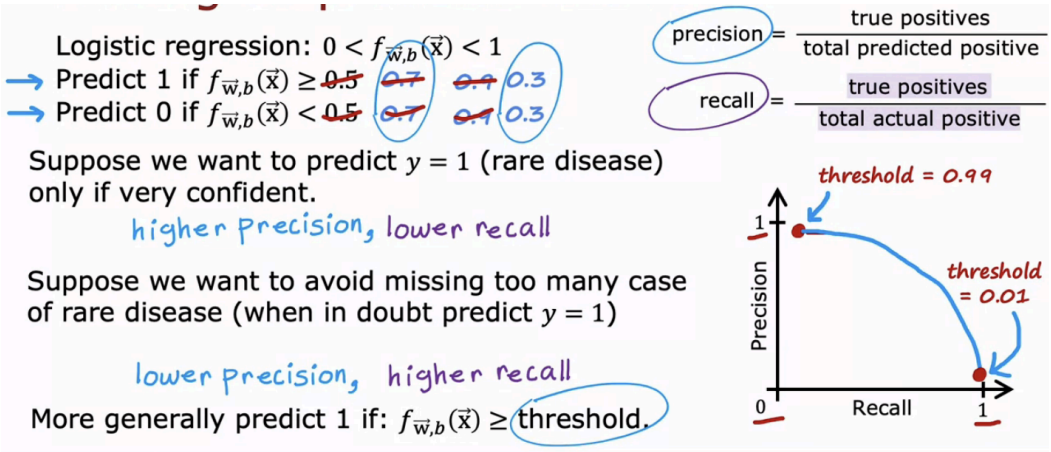


Precision/ recall



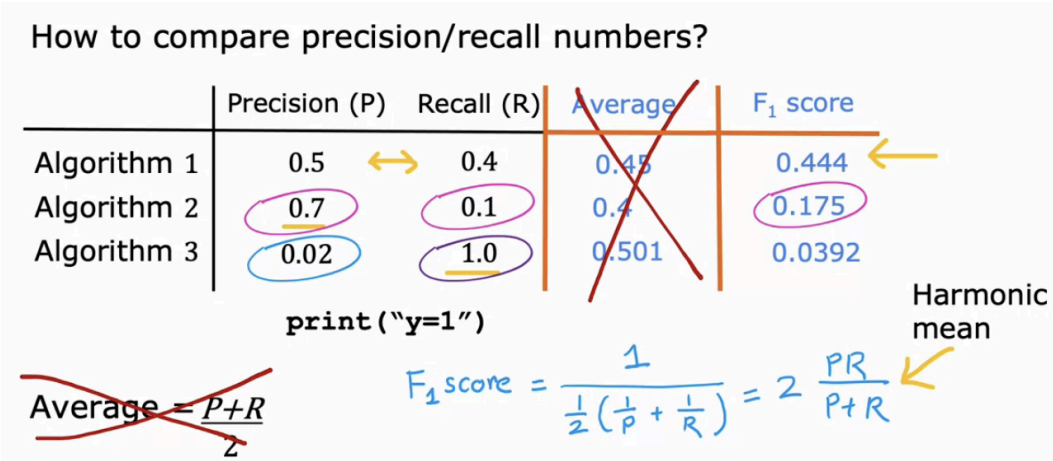
Trading off precision and recall

- Ideally have high precision and high recall, but often there is a tradeoff

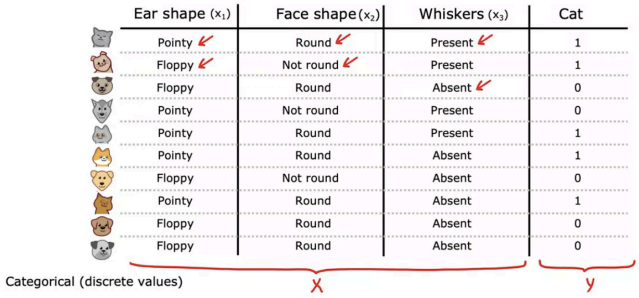


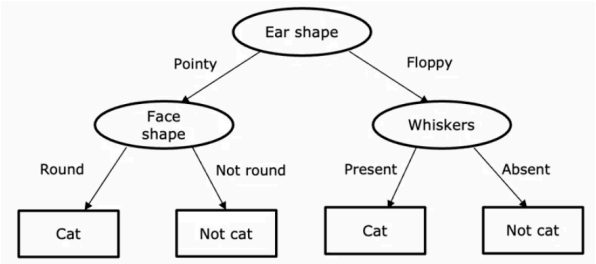
Often need to manually pick the threshold. If you want to automatically trade off precision and recall, can use an "F1 score":

- Taking average of P and R is not a good method. F1 emphasises very low scores:

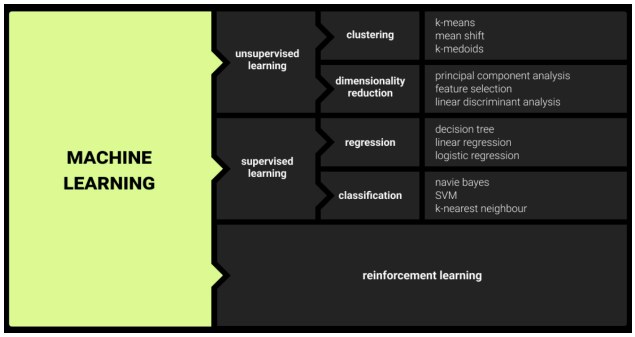


Week 4: Decision Trees





Root node at top; middle nodes are decision nodes; notes at bottom are called leaf nodes.



Algorithm	Description	Use Cases	Reasons to Use
Linear Regression	Models relationships between continuous variables.	House price prediction, sales forecasting, risk assessment	Simple, interpretable, and easy to implement
Logistic Regression	Predicts probabilities for binary outcomes.	Spam detection, fraud detection, customer churn prediction	Effective for binary classification, probabilistic output
Decision Tree	Splits data into branches for decision making.	Loan eligibility, credit scoring, medical diagnosis	Easy to visualize, handles both numerical and categorical data
SVM (Support Vector Machine)	Finds the optimal hyperplane for classification.	Image classification, cancer detection, spam detection	High-dimensional space suitability, robust for classification and regression
Naive Bayes	Applies Bayes' theorem for probabilistic classification.	Text classification, sentiment analysis, spam detection	Fast, handles high-dimensional data well
kNN (k-Nearest Neighbors)	Classifies based on closest training examples.	Product recommendations, anomaly detection, pattern recognition	Simple, intuitive, effective for classification and regression
K-Means	Clusters data into k groups based on similarity.	Customer segmentation, market analysis, image compression	Efficient for large datasets, easy to implement
Random Forest	Constructs multiple decision trees for robust predictions.	Stock market prediction, weather forecasting, medical diagnosis	Reduces overfitting, handles large datasets well
Dimensionality Reduction Algorithms	Reduces feature space while retaining variance.	Data visualization, feature extraction, accelerating model training	Simplifies models, removes noise, improves performance
Gradient Boosting Algorithms	Combines weak learners to form a strong predictor.	Predictive analytics, recommendation systems, fraud detection	High accuracy, leverages multiple models for improved performance

Decision Tree: Learning Process

- Use an algorithm to decide what root nodes etc. to use
- Looking to maximise purity (percentage correctness)

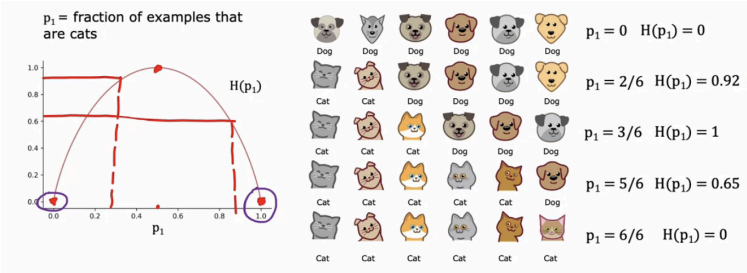
Decision 1: How to choose what feature to split on at each node?

Decision 2: When do you stop splitting?

- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth (root node is at Depth 0)
- When improvements in purity score are below a threshold
- When number of examples in a node is below a certain threshold

Decision Trees: Measuring purity

E.g. p1 = fraction of examples that are cats; y-axis is entropy



Entropy function (log is to base 2):

$$p_0 = 1 - p_1$$

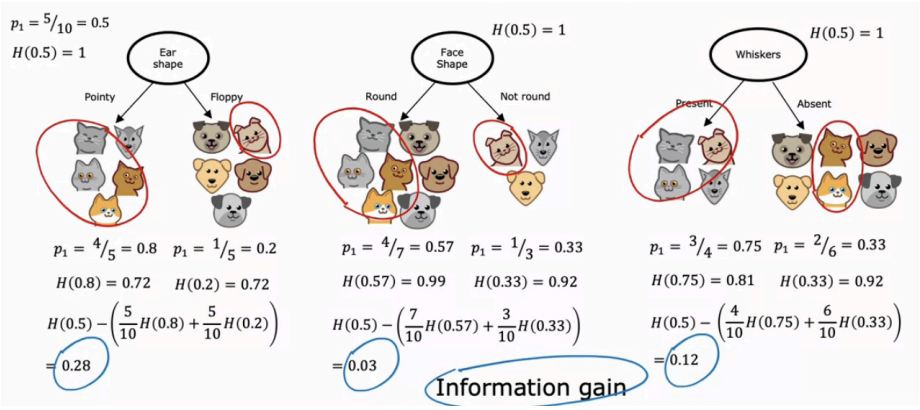
$$H(p_1) = -p_1 log_2(p_1) - p_0 log_2(p_0)$$
$$= -p_1 log_2(p_1) - (1 - p_1) log_2(1 - p_1)$$

Note: "0 log(0)" = 0

- Even if 0log(0) is technically negative infinity

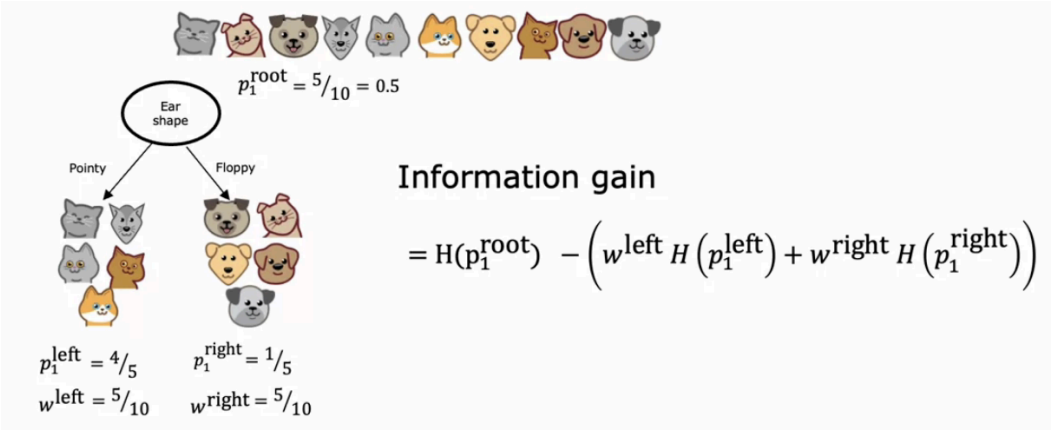
Choosing a split: Information gain

- What choice of feature reduces entropy, *H*, the most (reduction of entropy = "information gain")
- Take a weighted average, based on how many examples + entropy (a large dataset with high entropy is worse than a small one with high entropy)
- Overall, compute weighted reduction in entropy compared to not splitting at all (p1 = 5/10 = 0.5, so H at root node is 1) [as one of the ways to decide whether to split is size of reduction in entropy - not much point in splitting if decrease is v small]



Information Gain

P1 - number of positive examples on left or right, w = fraction of examples that went to right or left branch



Putting it together

Overall process:

- Start with all examples at root node
- Calculate info gain for all poss features, and pick one with highest info gain
- Split dataset according to selected feature, and create left and right branches of the treee
- 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
    - Info gain from additional splits is less than threshold
    - When number of examples in a node is below a threshold

Recursive splitting: to build a decision tree at the root, you build smaller decision trees at the left and right branches.

Using one-hot encoding of categorical features: ***For features that can take on more values than yes or no***

- Split into different features, keeping each to taking on two values. ***If a categorical feature can take on k values, create k binary features (0 or 1 valued).***

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

This also works for training neural networks (not just for decision trees - neural networks expect numbers as inputs):

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

Continuous Valued Features

- Using decision trees to work with not only discrete values, but continuous too (e.g. weight)

Ear shape	Face shape	Whiskers	Weight (lbs.)	Cat
Pointy	Round	Present	7.2	1
Floppy	Not round	Present	8.8	1
Floppy	Round	Absent	15	0
Pointy	Not round	Present	9.2	0
Pointy	Round	Present	8.4	1
Pointy	Round	Absent	7.6	1
Floppy	Not round	Absent	11	0
Pointy	Round	Absent	10.2	1
Floppy	Round	Absent	18	0
Floppy	Round	Absent	20	0

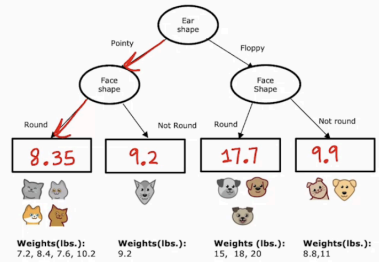
- For a continuous value, pick a threshold which gives the best information gain (calculate information gain for lots of possible thresholds)
- Convention: sort all of examples according to weight/ value of features, and take all the values that are the midpoints between the sorted list of training examples as the values for consideration as a threshold
- If info gain is better from splitting on a given value of this threshold than any other feature, then you decide to split this node at this feature. I.e. there are two thresholds: algorithm has to choose which feature to use first with info gain (considering the different thresholds for continuous ones)

Decision trees in Regression

- So far, only in classification algorithms
- Can also use in regression: using ear shape etc. to predict the weight

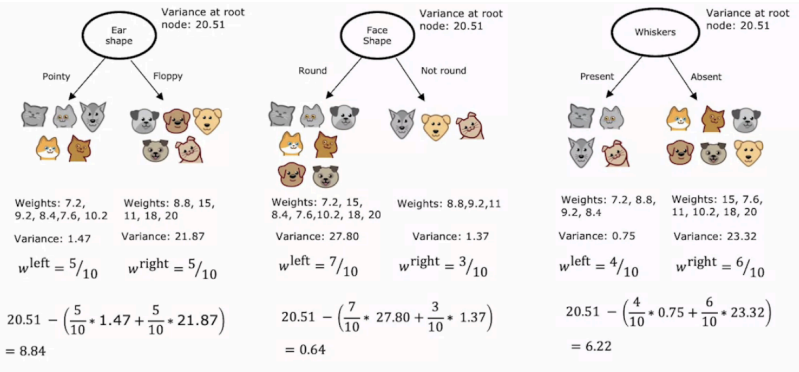
Ear shape	Face shape	Whiskers	Weight (lbs.)
Pointy	Round	Present	7.2
Floppy	Not round	Present	8.8
Floppy	Round	Absent	15
Pointy	Not round	Present	9.2
Pointy	Round	Present	8.4
Pointy	Round	Absent	7.6
Floppy	Not round	Absent	11
Pointy	Round	Absent	10.2
Floppy	Round	Absent	18
Floppy	Round	Absent	20

X y



So, how to choose a split?:

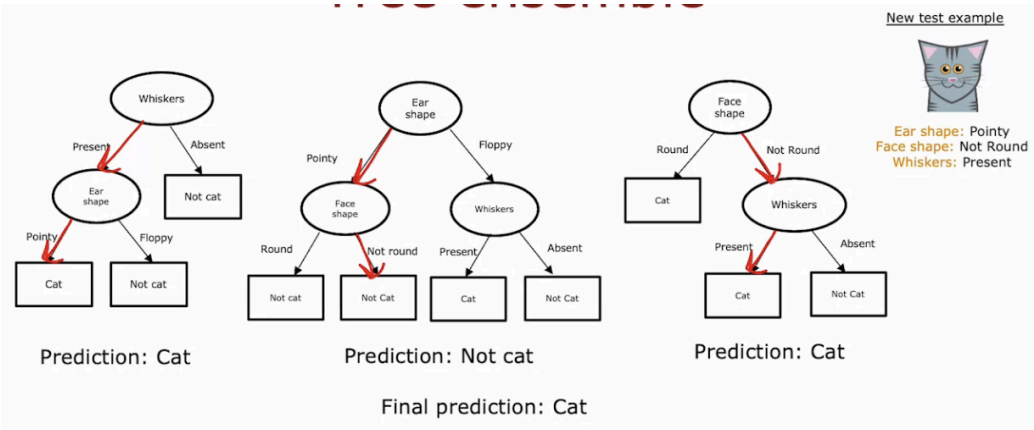
- Instead of reducing entropy, **try to reduce the weighted average variance** (how widely a set of numbers varies)



Then build a new decision tree based on these five examples.

Using Multiple Decision Trees: A tree ensemble

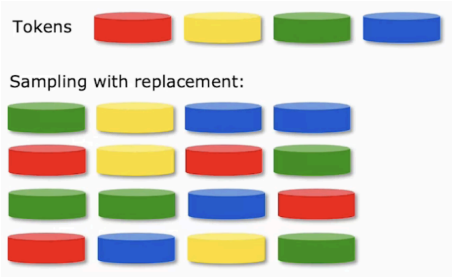
- **Trees are highly sensitive to small changes of the data** (can cause a different split at the root and thus a totally different tree)
- Use many trees, and run all on a new example. Get them to then vote



Having lots of trees makes the algo less sensitive to changes in data. But how to come up with plausible but different decision trees - use **sampling with replacement**.

Sampling with replacement

[Putting the token back - if not, would get the same four tokens every time.]



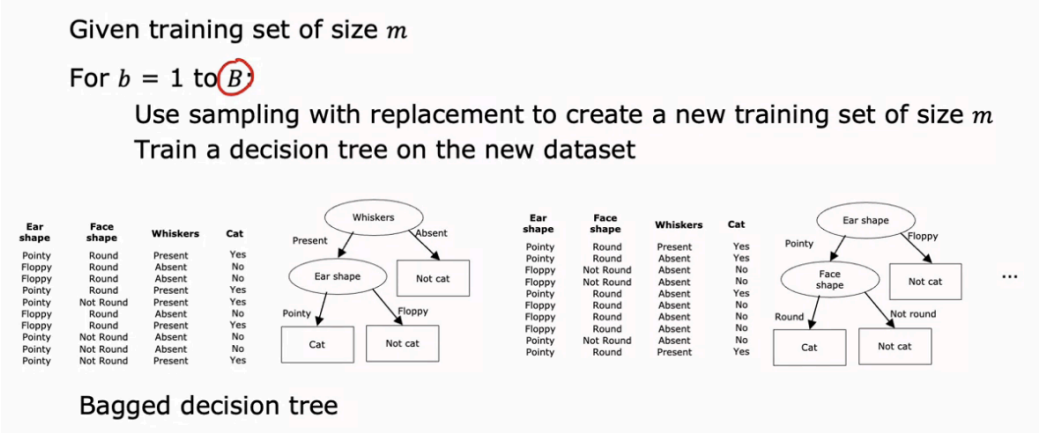
Applying this to decision trees:

- Construct random training sets that are all slightly different from original training set. Put the training examples in a theoretical bag and pick one out, put it back in, and pick again (will get some repeats).

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

Random Forest Algorithm

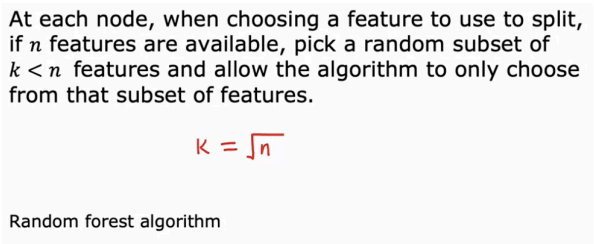
1. Generating a tree sample



- B is normally around 100 max - higher than that slows down computation and also does not improve much
- Each tree then votes

2. Modification: Randomizing the feature choice

- To avoid root note split being often the same



- This is more robust than a single decision tree because sampling with replacement procedure causes the algorithm to explore a lot of small changes in the data already and is averaging over all these changes, and so small changes to training set are less likely to affect overall output of random first

XGBoost

1. Generating a tree sample
    - For every time through this loop, other than first time, when sampling - instead of picking from all m examples from equal probability, make it more likely to pick examples that the previously trained trees do poorly on [FOCUSING ON WHAT WE ARE NOT DOING WELL ON]

Given training set of size m

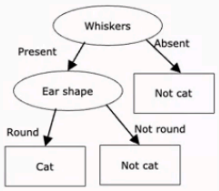
For $b = 1$ to B:

Use sampling with replacement to create a new training set of size m

But instead of picking from all examples with equal (1/m) probability, make it more likely to pick misclassified examples from previously trained trees

Train a decision tree on the new dataset

Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Round	Present	Yes
Pointy	Not Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Present	Yes
Pointy	Not Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Not Round	Present	Yes



Ear shape	Face shape	Whiskers	Prediction
Pointy	Round	Present	Cat ✓
Floppy	Not Round	Present	Not cat ✗
Floppy	Round	Absent	Not cat ✓
Pointy	Not Round	Present	Not cat ✓
Pointy	Round	Present	Cat ✓
Pointy	Round	Absent	Not cat ✗
Floppy	Not Round	Absent	Not cat ✓
Pointy	Round	Absent	Not cat ✗
Floppy	Round	Absent	Not cat ✓
Floppy	Round	Absent	Not cat ✓

$1, 2, \ldots, b-1$ ← b

XGBoost is:

- Open source implementation of boosted trees
- Fast efficient implementation
- Good choice of default splitting criteria and criteria for when to stop splitting
- Built in regularization to prevent overfitting
- Highly competitive algorithm for machine learning competitions (eg: Kaggle competitions)

- Instead of sampling with replacement, assigns different weights to different training examples so does not need to generate a load of randomly chosen training sets, making it a bit more efficient than using a sampling with replacement procedure

Using XGBoost:

Classification

```
from xgboost import XGBClassifier

model = XGBClassifier()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

Regression

```
from xgboost import XGBRegressor

model = XGBRegressor()

model.fit(X_train, y_train)
y_pred = model.predict(X_test)
```

When to use decision trees vs neural networks?

Decision Trees and Tree ensembles

- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast
- Small decision trees may be human interpretable

Neural Networks

- Works well on all types of data, including tabular (structured) and unstructured data
- May be slower than a decision tree
- Works with transfer learning
- When building a system of multiple models working together, it might be easier to string together multiple neural networks

- Can train neural networks all together using gradient descent, but can only train one decision tree at a time