PCSE 595
Final Exam Review

Dr. Sam Henry

Exam on Thursday, April 25th @ 8am

Assignment 4

- Send me an email by midnight tonight (Monday April 22nd) if you want me to grade the “Bonus Assignment”

How these slides are organized

- The green slides tell you the topics
- Then, there are white slides that follow that have contents containing the information for those topics

  - You **don't need to memorize any equations**. I will not ask you to write an equation.
  - You should be able to interpret equations if they are provided to you
  - You must know how to compute precision, recall, F1, macro/micro from a confusion matrix. Precision, recall, F1 equations will be provided if needed.
  - Focus on the concepts

Overfitting

1. What is overfitting?
2. Purpose of validation data
3. What is underfitting?
4. Bias, Variance Trade-off
5. Methods to Avoid Overfitting
    a. Regularization
    b. L1 and L2 Regularization
    c. Dropout
    d. Early stopping

1. What is Overfitting

- Overfitting relates to training error decreasing at the expense of model generalization

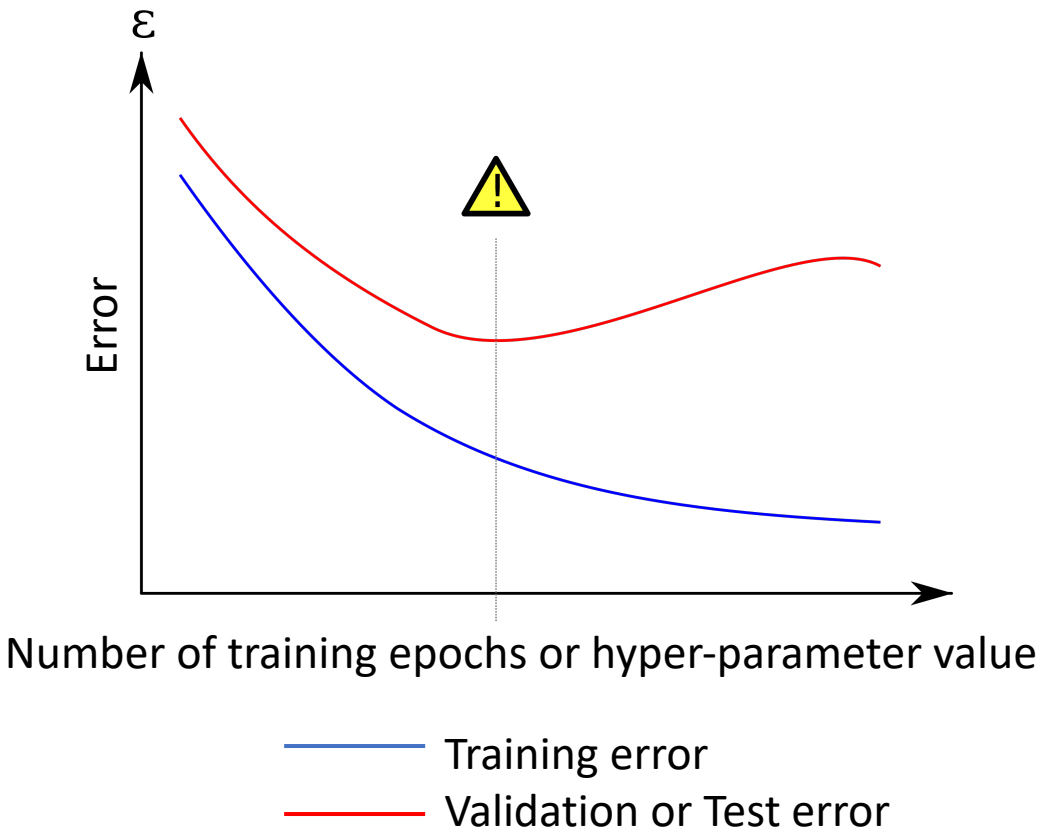
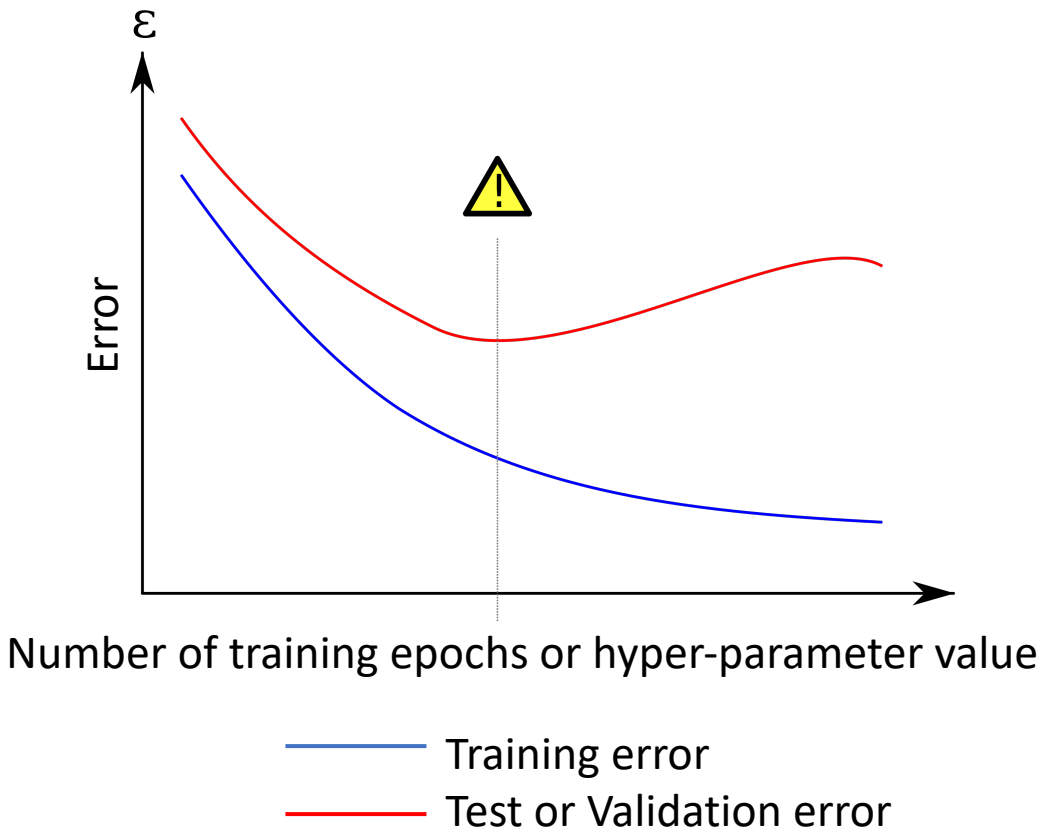


Image from: https://en.wikipedia.org/wiki/Overfitting

2. Purpose of validation data

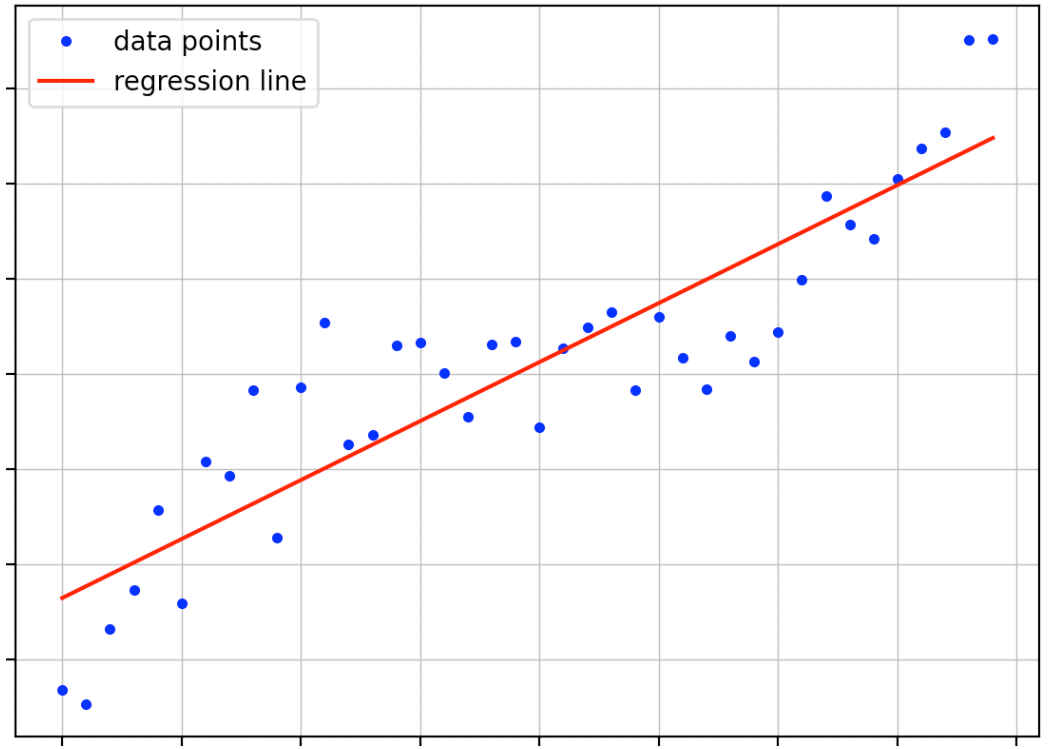


- Maximizing training error does not necessarily maximize generalizability (test/validation error)
- Remember: **generalization is the goal**
  - maximize the model's performance on novel (unseen) data
- Training error reports performance with respect to seen data
- Training error is often optimistic with respect to novel data
- Validation data allows us to estimate the model performance on novel, unseen data
- It is an estimate of the model's generalizability

Image from: https://en.wikipedia.org/wiki/Overfitting

3. What is Underfitting

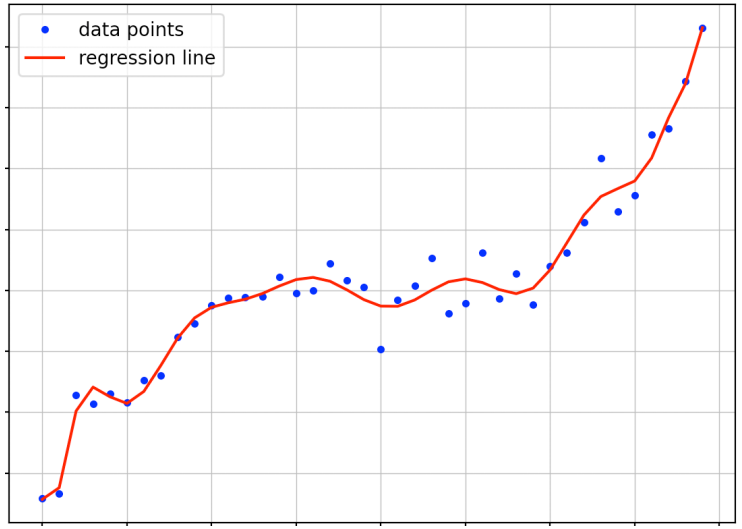
- We can also underfit data
- Fit too simple of a model to our data
- Model has too much bias and is therefore not sensitive enough to the data



Least squares estimate with a 1-degree polynomial

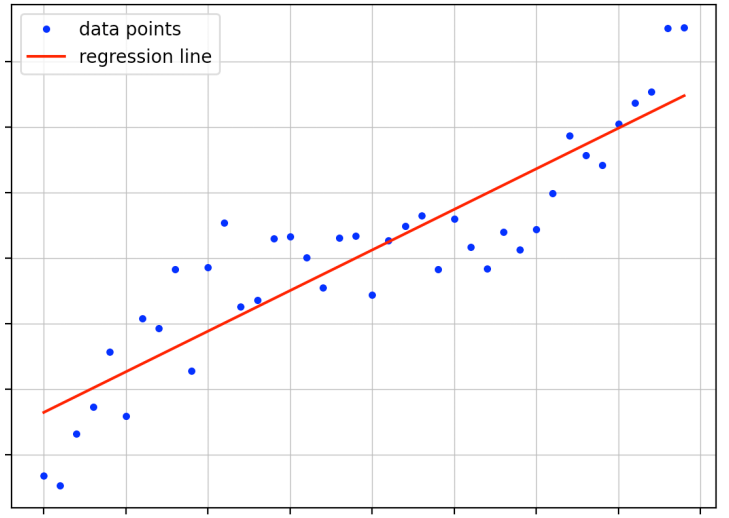
4. Bias-Variance Trade-off

- **Variance** – the amount that your model is sensitive to the data
  - High variance will vary a lot between runs of cross-validation
- **Bias** – the amount of assumptions (bias) built into the model
  - High bias will vary little between runs of cross-validation



Least squares estimate with a 15-degree polynomial

High Variance, Low Bias

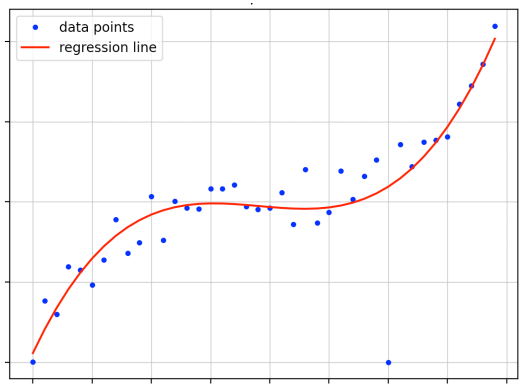


Least squares estimate with a 1-degree polynomial

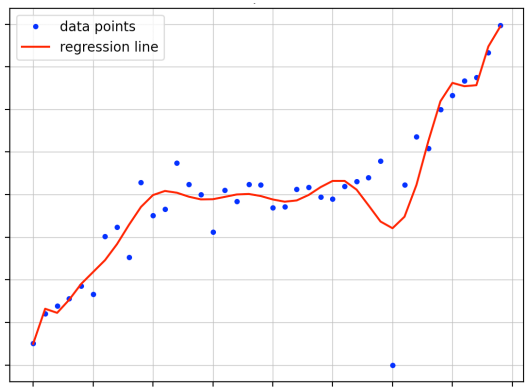
Low Variance, High Bias

5a. Regularization

- Complex models will likely overfit our data
- We want to reduce the complexity of our models
  - **Complexity relates to how large our weights are**
  - Large weights indicate more of a bend
  - Weights of 0 will ‘turn a bend off’



Least squares estimate with a 3-degree polynomial



Least squares estimate with a 15-degree polynomial

Regularization = We redefine the loss function to account for both training error (how well we fit the sampled data) and model complexity

$$Loss = Error + \lambda * Complexity$$

5b. L1 and L2 Regularization

- L2 Regularization uses the L2 norm of the weight vector

$$J(\theta) = Error + \lambda\theta^T\theta$$

Written as a summation:

$$\theta^T\theta = \sum_{i=1}^{d}(\theta_i)^2$$

- L1 Regularization uses the L1 norm of the weight vector

$$J(\theta) = Error + \lambda|\theta|$$

Written as a summation:

$$|\theta| = \sum_{i=1}^{d}|\theta_i|$$

5b. L1 and L2 Regularization

- With L1 regularization makes weights go **to** zero, effectively turning off features.
- With L2 regularization, weights get near zero= but features don’t turn off

Terminology:

- Ridge Regression = Sum of Squared Error with L2 Regularization
- LASSO = L1 Regularization

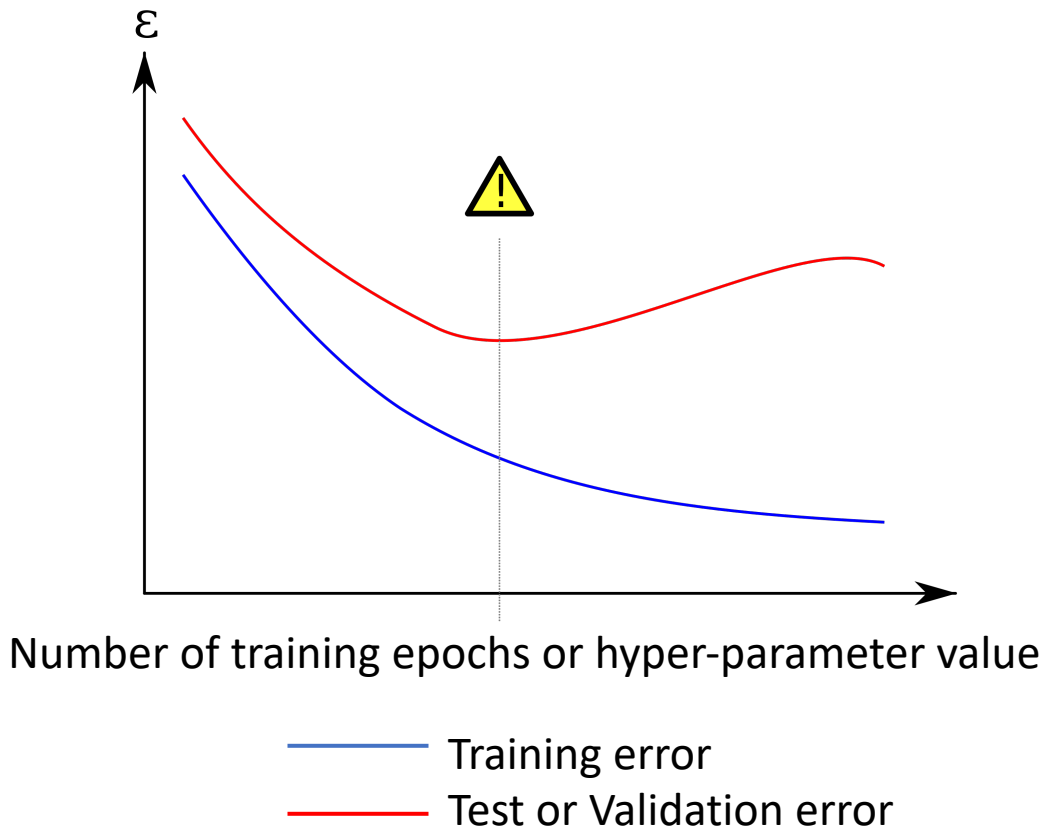
5c. Drop Out

- Typically used in Deep Neural Networks to prevent of overfitting
- Does so by adding “noise” to the network
  - It does this by “dropping out” nodes, which means setting their output to 0

- Claims to be as effective as regularization (weight decay) without added computational complexity
- Prevents overfitting by inducing sparsity in the network
  - Meaning many weights go to zero

5d. Early Stopping

- Stopping training before loss stops decreasing
- Avoids overfitting (see image)
- Caution – may stop too early and therefore underfit the model



Gradient Descent

1. Relationship between loss function and model weights
2. How weight updates work
3. Variations on Gradient Descent
   - Stochastic
   - Mini-batch
   - “vanilla”
4. Conceptually how momentum works

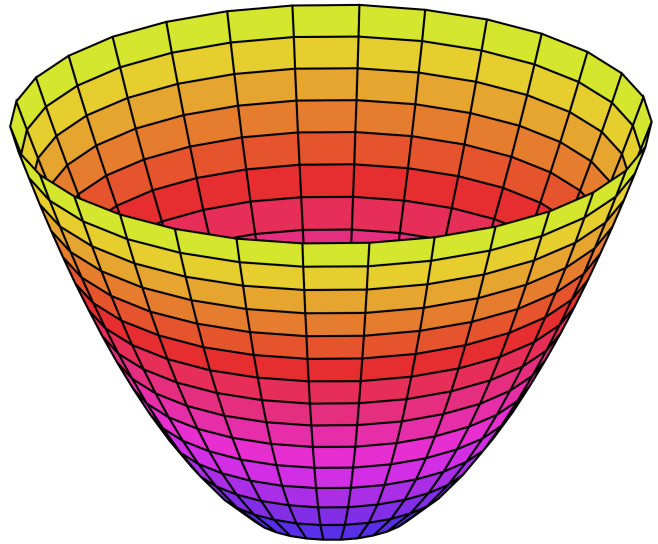
1. Relationship between loss function and model weights

- Model weights determine the predicted values of the model
- The predicted values determine the model loss
- So, model loss is ultimately a function of the model weights
- This is what we descend in gradient descent, and how we find weights by minimizing loss

2. How Weight Updates Work

1. Start somewhere on surface defined by the loss function
   - Initialize θ
2. Find the direction to move
   - Find direction of the gradient
3. Move in that direction
   - Update θ
4. Repeat steps 2 and 3

Image from: https://en.wikipedia.org/wiki/Paraboloid



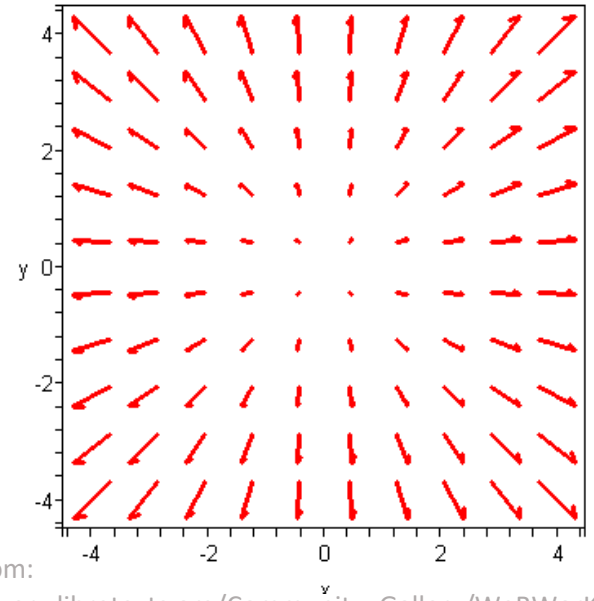


Image from: https://query.libretexts.org/Community_Gallery/WeBWorK_Assessments/Calculus_-_multivariable/Vector_fields/

3. Variations of Gradient Descent

- “Vanilla” Gradient Descent
  - The gradient is estimated using the entire dataset
  - Single update per epoch
- Stochastic Gradient Descent
  - The gradient is estimated using a single point
  - Updates after each sample
- Mini-batch Gradient Descent
  - The gradient is estimated using a subset of the data (a batch)
  - Updates after each batch

Most commonly used. Middle ground between stochastic and “Vanilla” Gradient Descent

4. Momentum

- Mini-batches can create rough and unstable estimates of loss
  - Causing step to go in a bad direction
- Momentum can help reduce this instability
  - Also, momentum can help us overcome local minima (if present)

- Analogous to the momentum of a ball rolling down hill
- Take some of the previous direction and speed (the gradient from previous estimate) and add it to the current step

Model Evaluation

1. Problem Types
2. Loss functions
3. Evaluation metrics
    a. Precision and recall trade-off
    b. Macro and micro averaging
4. Train, test, validation sets

1. Problem Types

Regression - Assign a real-valued number(s) to a sample

Classification – assign a class or label to a sample

Classification Problems:

- Binary – each sample is a member of exactly one class
  - Final output is a 0 or 1 indicating class membership
- Multi-class – each sample is a member of exactly one class
  - Final Output vector contains exactly one “1”
- Multi-label – each sample is a member of zero or more labels
  - Final Output vector contains zero or more “1’s”

2. Loss Functions

- Sum of Squared Error
  - For Regression Problems

$$J(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \left(Y - \hat{Y}\right)^T \left(Y - \hat{Y}\right)$$

- Mean Squared Error
  - For Regression Problems

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \left(Y - \hat{Y}\right)^T \left(Y - \hat{Y}\right)$$

- Binary Cross Entropy
  - For Binary or Multilabel Classification Problems

$$J(\theta) = -\sum_{i=1}^{n} y_i log(\hat{y}_i) + (1 - y_i) lo\, g(1 - \hat{y}_i)$$

- Categorical Cross Entropy
  - For Multiclass Problems

$$J(\theta) = -log\left(\frac{e^{\widehat{y_p}}}{\sum_{i=1}^{c} e^{\widehat{y_i}}}\right)$$

2. Loss Functions

Cross Entropy (CE)

$$-\sum_{i=1}^{c} y_i log(\hat{y}_i)$$

Binary Cross Entropy (BCE)

$$y_i log(\hat{y}) + (1 - y_i) \log(1 - \hat{y})$$

binary cross entropy is the specific case for binary classification

Categorical Cross Entropy (CCE)

$$-log\left(\frac{e^{\widehat{y_p}}}{\sum_{i=1}^{c} e^{\widehat{y_i}}}\right)$$

categorical cross entropy is the specific case for one-hot encoded multiclass labels with a softmax activation function

3. Evaluation Metrics

		Predicted Class: Class 1 (pos)	Predicted Class: Class 2 (neg)	
True Class	Class 1 (pos)	TP	FN	P=TP+FN
True Class	Class 2 (neg)	FP	TN	N=FP+TN

$$Accuracy = \frac{TP + TN}{P + N}$$

Measures the percent of predictions that were correct. Not suitable for class-imbalanced problems, but a great choice if classes are balanced.

Don't need to memorize equations

For class imbalanced problems, use:

$$Precision = \frac{TP}{TP + FP}$$

What percent of samples that I predict are true are actually true?

$$Recall = \frac{TP}{TP + FN}$$

What percentage of true samples am I predicting to be true?

Be able to calculate for binary, multi-label, and multi-class problems

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

A measure that balances the trade-off between precision and recall

3a. Precision and Recall Tradeoff

There is a trade-off between precision and recall

There is typically a threshold you can adjust adjust the trade-off

High Recall = Fishing with a big net

High Precision = Spear fishing

- As you increase the size of your net, you catch more fish
- You'll catch a lot of of what you wanted, but you'll get increasingly more bycatch

3b. Macro and Micro Averaging

- Micro Averaged Precision, Recall, F1
  - Classes with large number of samples dominate
  - Equally weights the performance of each sample
  - Tells us the performance over all samples
- Macro Averaged Precision, Recall, F1
  - All classes contribute equally
  - Equally weights the performance of each class
  - Tells us the average performance over all classes

- Given equations for precision and recall, calculate the macro and micro averaged precision and recall given the following table
- Be able to explain their difference

		Predicted Class		
		Class 1	Class 2	Class 3
True Class	Class 1	4	50	1
	Class 2	10	290	65
	Class 3	0	10	2

4. Train, Test, and Validation Sets

- Training set
  - Used to find model weights
- Validation set
  - Used to estimate model generalizability
  - Used during the hyperparameter tuning process
- Test set
  - Withheld from model training and hyperparameter tuning
  - Used to evaluate the performance on the final, tuned model on novel unseen data

Other Topics

- Artificially Balance the Data
  - Oversampling, undersampling
  - SMOTE
- Class Weights

Artificially Balance the Dataset

- Under-sample the majority class
  - Randomly select a fixed number of samples from each class such that the resulting dataset is balanced
  - Problem: you are removing data which may be informative to the system
- Over-sample the minority class
  - Randomly repeat a fixed number of samples from each minority class such that the resulting dataset is balanced
  - Problem: you are repeating data which makes those points artificially more important. What if the points are outliers?

$$Training\_Loss = J(\theta) = -\frac{1}{n}\sum_{i=1}^{n} loss(\hat{y}_i, y_i)$$

Each of these methods make it so that each class contributes equally to loss

Artificially Balance the Dataset

- Generate synthetic data for the minority class
  - This is typically a better option than over-sampling, but could be problematic if your synthetic data is not representative of your real data.
  - A popular data generation method is "Synthetic Minority Oversampling Technique" (SMOTE)
  - The authors recommend a combination of under-sampling the majority class and using SMOTE to over-sample the minority class. Stating that it outperforms doing only under-sampling or doing only over-sampling.

SMOTE: Synthetic Minority Over-sampling Technique by Chawla, et al., 2002 https://arxiv.org/pdf/1106.1813.pdf

“Synthetic Minority Oversampling Technique” (SMOTE)

- SMOTE works by:
  1. For a single point, find the k-nearest neighbors and randomly select one
  2. For each feature:
     1. Find the difference between the sample and the its neighbor
     2. Randomly generate a number between 0 and 1 and multiply it by the difference
     3. Add the scaled difference to the feature value of that original point
- This effectively creates new feature values somewhere on a line between the point and its nearest neighbor.
- “This approach effectively forces the decision region of the minority class to become more general.”

SMOTE: Synthetic Minority Over-sampling Technique by Chawla, et al., 2002 https://arxiv.org/pdf/1106.1813.pdf

Class Weights

- Change how much each sample contributes to loss based on its weight

$$Training\_Loss = J(\theta) = -\frac{1}{n}\sum_{i=1}^{n} c_i * loss(\hat{y}_i, y_i)$$

- Where c_i is the class weight for the class of sample I
- You can choose any value for a class weight, but typically:

$$c_i = 1 - \frac{number\ of\ samples\ of\ class\ i}{total\ number\ of\ samples}$$

Note: Imbalanced Data

- Consider a sample that you are unsure about
  - Shouldn’t you learn to label it the majority class, since you’d be right most of the time?
- When you modify the dataset, you learning something that is not representative of the real world.
- It is best to learn with the original data distribution.
- Use artificial balancing techniques only after trying to learn with the real-world data distribution

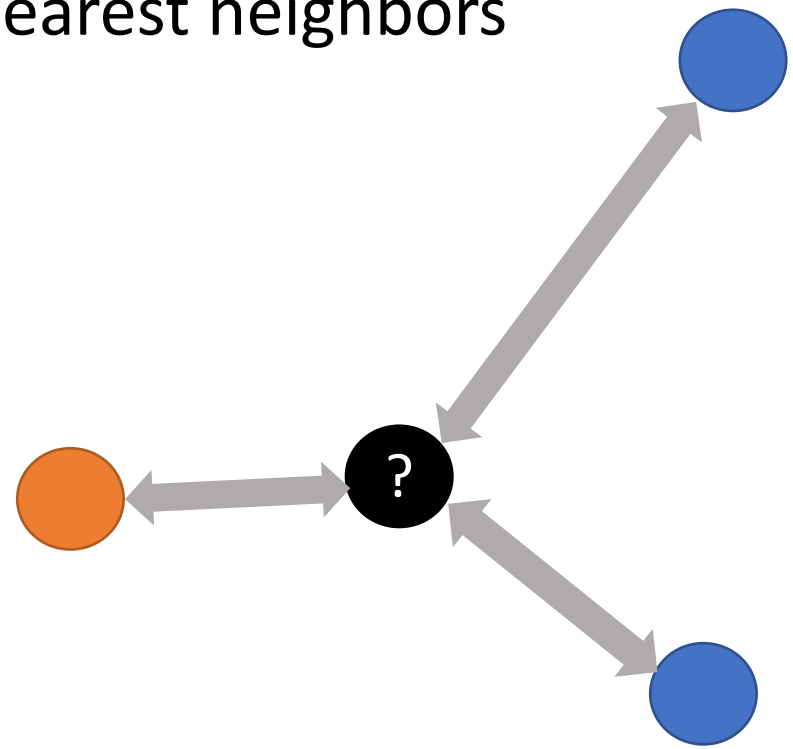
Different Classifiers

Be able to explain how each of these classifiers works

1. K-Nearest Neighbors (KNN)
2. Nearest Centroid Classifier
3. Decision tree
    a. Decision Stump
4. Linear/logistic Regression
5. Artificial neural network (ANN)
    a. Deep learning architectures
6. ~~Support Vector Machine (SVM)~~
7. Ensemble Methods
    a. Stacking
    b. Bagging
    c. Boosting
    d. Random Forests

1. K-Nearest Neighbors

- Given a labeled dataset, determine the label of a new point
- Classify the new point by voting among the k-nearest neighbors



- Generalization of nearest neighbor
  - Nearest neighbor is K-nearest neighbors with k = 1

2. Nearest Centroid Classifier

1. Calculate the centroid (mean vector) of each class
2. Classify based on the closest class centroid

- Benefits and Drawbacks:
  - Faster, especially for large datasets because it only compares to the number of classes, rather than number of datapoints
  - Simpler and doesn't model complex class distributions

…so, we are modeling a set of data with a single point

3. Decision Tree

Algorithm: Divide(node)

1. If (all classes are the same)
2. return
3. feature = find_max_info_gain(node)
4. Datasets = split data by feature
5. For data in datasets:
6. child = add node to tree (data)
7. divide(child)

We should be able to handle continuous data

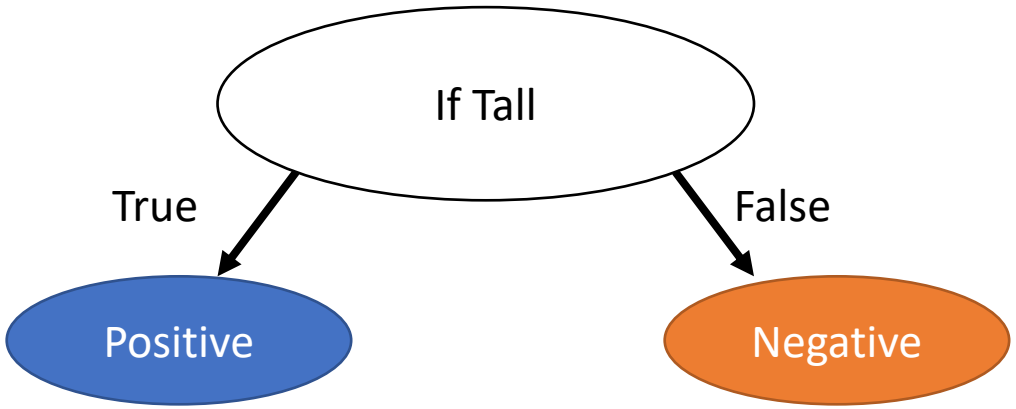
3. Decision Tree: Handling Numeric Data

- Decision Trees perform univariate tests
  - In other words they test one variable at a time.
  - Data splits are therefore parallel to an axis
    - No linear combinations of attributes for splits
- Therefore, generate candidate splits one feature at a time
  - Idea: Find borders between classes and split at border

Attribute Value	40	48	50	54	60	70
Class	F	F	T	T	T	F

3a. Decision Stump

- The Complexity of the classification function depends on the depth of the decision tree
- A depth-1 decision tree is called a **decision stump**



4. Linear and Logistic Regression

- Finds a linear function to fit the data
- Linear function can be found via a closed form or via gradient descent
- Nice theoretical properties
  - Convex optimization function with a single global minimum. No local minima
- Logistic regression is linear regression put through a logistic (sigmoid) function
  - Better for classification because data is Bernoulli distributed
    - Put simply, data is 0 or 1 and sigmoids tend to be 0 or 1
  - Typically uses binary cross entropy as a loss function
- For regression Sum of Squared Error is the typical loss function

5. Artificial Neural Network

See category slide on Artificial Neural Networks

7. Ensemble Methods

- See Category Slide on Ensemble Methods

Neural Networks

1. Explain a neuron
   - Different components
   - As an equation
   - As a figure
2. Explain a simple 2-layer neural network
   - As a figure
3. Explain how back-propagation works
   - What is it doing?
4. Discuss different activation functions
5. Appropriate last layers and loss functions

1. Explain a Neuron

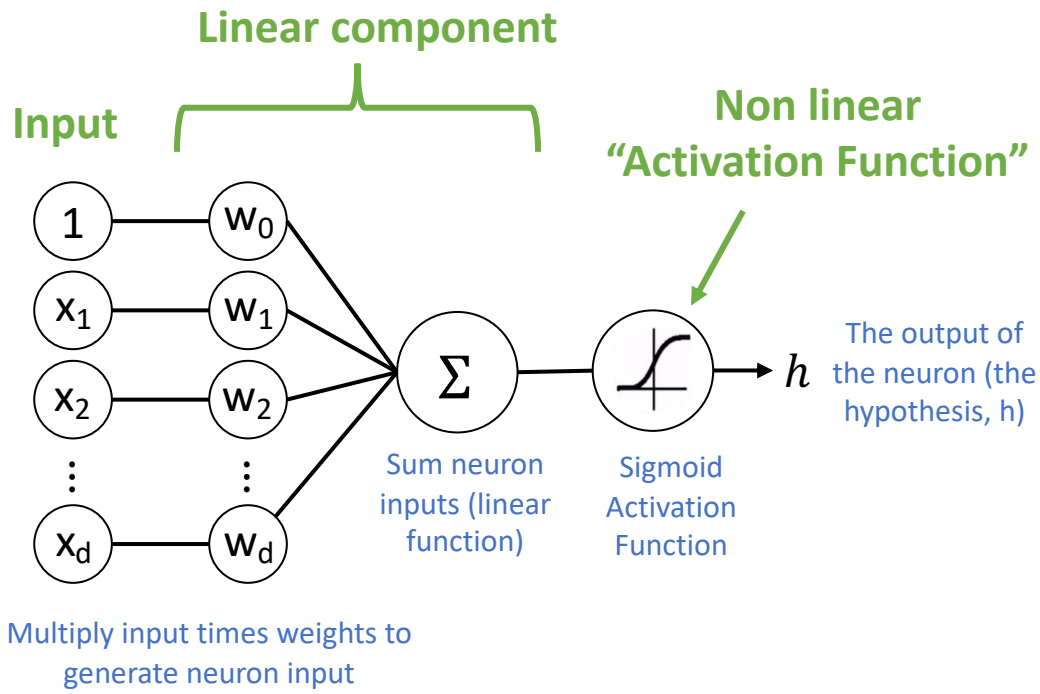
Each neuron computes Logistic Regression

$$\hat{y} = s\left(\sum_{i=1}^{d} \theta_i * x_i\right) = s(\theta X)$$

Where *s* is a sigmoid-type function such as logistic

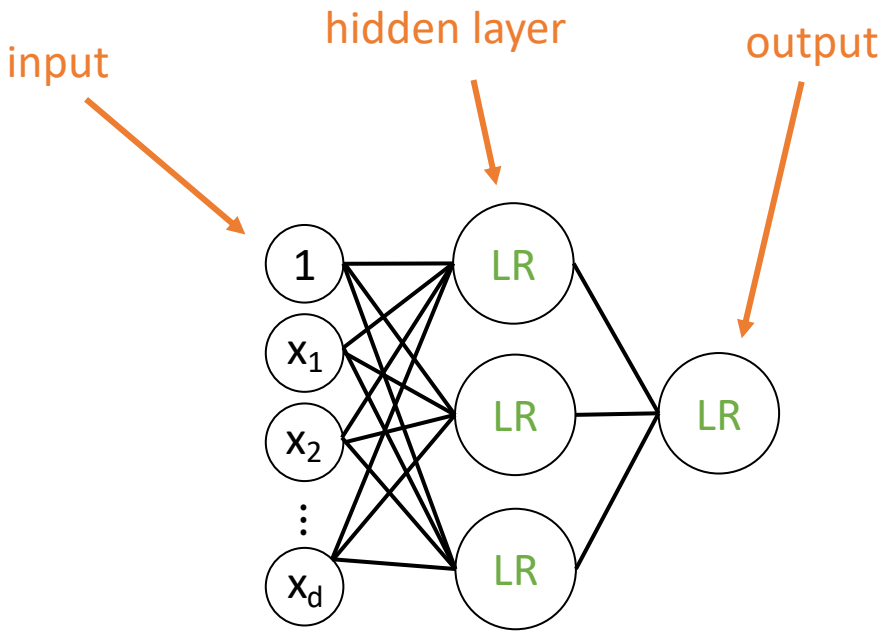
$$h = \frac{1}{1 + e^{-\sum_{i=1}^{d} \theta_i * x_i}}$$

This is the function computed by a neuron with a logistic activation function



2. Explain a 2-layer Neural Network

- An artificial neural network is a network of connected artificial neurons
- Each neuron computes logistic regression
- Input and output layer sizes are determined by the number of feature and number of labels
- 2-layer neural networks are universal approximators, meaning they can approximate any function
  - More complex functions require more hidden neurons
  - More hidden neurons = more expressive power, but more potential to overfit
- Adding more hidden layers typically makes learning faster and easier (less local minima)



3. Explain how back-propagation works

- Neural networks learn by iteratively updating the weights in the network
- The update works in the same manner as gradient descent. You find the gradient (partial derivative with respect to each weight) and move in the direction of the negative gradient
- Neural network functions are complex, so calculating the derivative is complex
- **Back-propagation is an algorithm for efficiently calculating the derivative of an arbitrary neural network**
- The error is found in the last layer, and is backpropagated through the network up to the first layer

3. Explain how back-propagation works

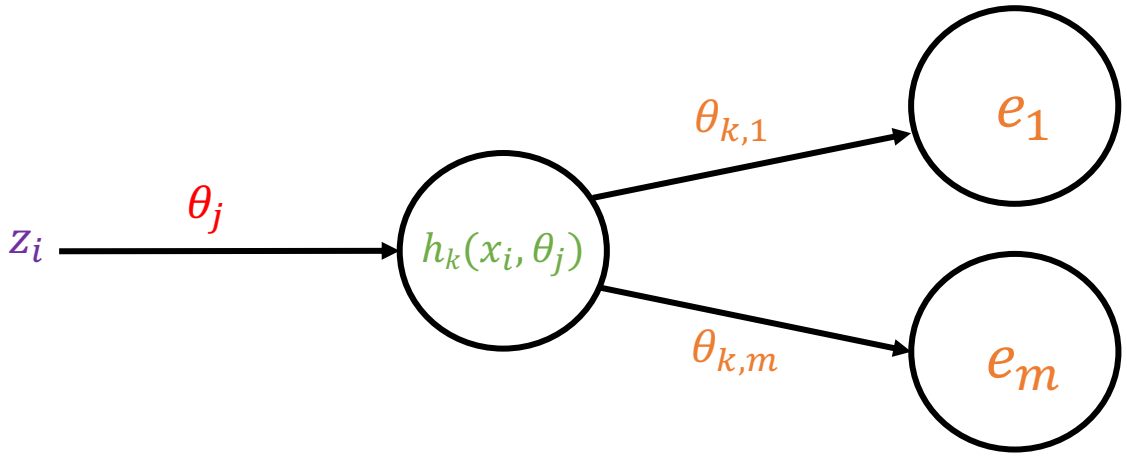
Thanks to special properties of logarithmic functions, **backpropagation is efficient** to calculate

$$gradient_j = z_i * h(z_i, \theta_j)(1 - h(z_i, \theta_j)) * \left(\sum_{m=1}^{m} \theta_{k,m} * e_m \right)$$

z_i is the input into neuron (calculated in forward propagation)

$h(z_i, \theta_j)$ is the output of the neuron (calculated in forward propagation)

The only new calculation is the error. The error is "backpropagated through the network"



You don't need to memorize this equation

3. Explain how back-propagation works

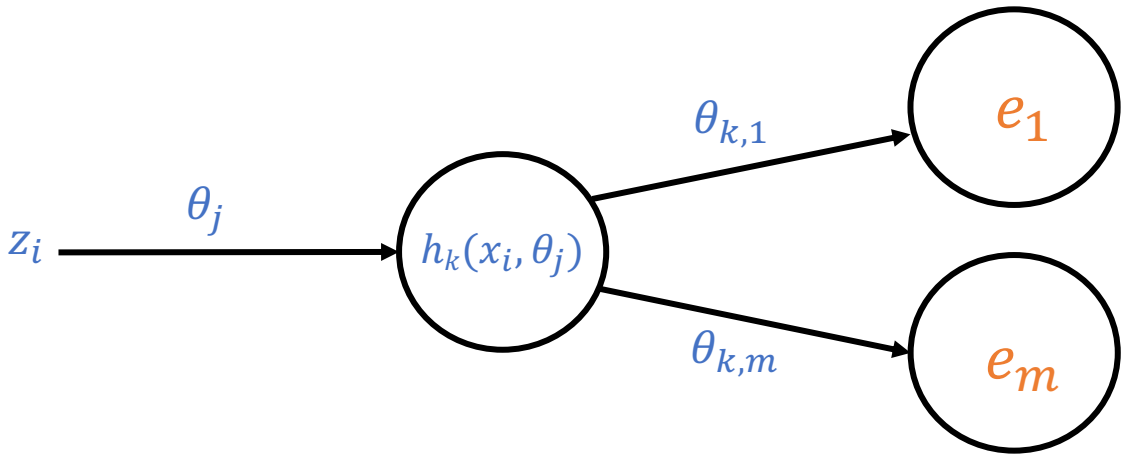
Thanks to special properties of logarithmic functions and the chain rule, backpropagation is can be **calculated locally**. We only need a global estimate of error.

This means backpropagation is **flexible** – it can be calculated no matter the network shape

This also means the backpropagation is **conceptually scalable**, and can be **computationally scalable**

$$gradient_j = z_i * h(z_i, \theta_j)(1 - h(z_i, \theta_j)) * \left(\sum_{m=1}^{m} \theta_{k,m} * e_m \right)$$

You don't need to memorize this equation

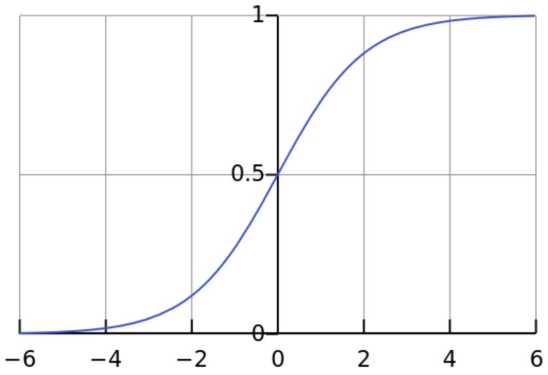


Local input and output values (node and its children)

Global error estimate – must be backpropagated through the network

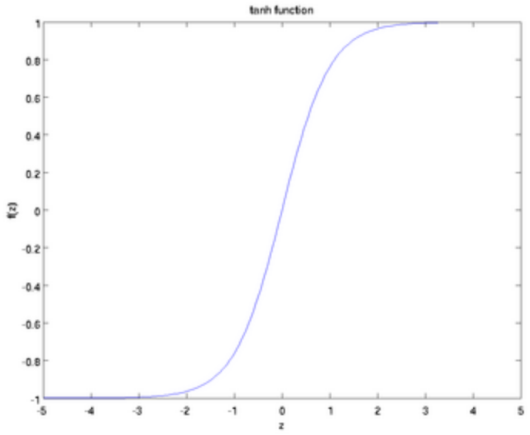
4. Discuss Different Activation Functions

$$A = \frac{1}{1+e^{-x}}$$



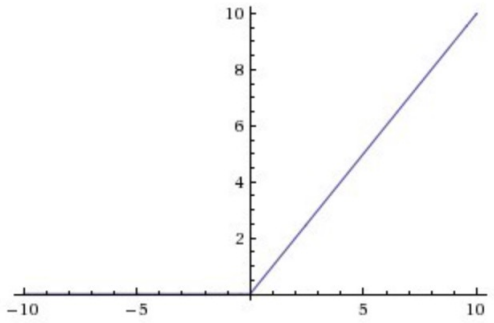
Logistic Function

$$f(x) = tanh(x) = \frac{2}{1+e^{-2x}} - 1$$



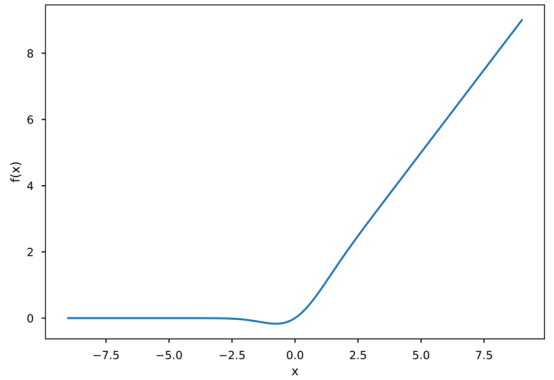
tanh Function

$$A(x) = \max(0,x)$$



ReLu function

$$gelu(x) = \frac{1}{2}x(1 + erf(\frac{x}{\sqrt{2}}))$$



GeLu function

From: https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

4. Different Activation Functions

- **Logistic** – original activation function, scales between 0 and 1
- **Tanh** – developed to reduce the vanishing gradient problem. It's like logistic, but with a steeper slope.
- **ReLu** – developed to reduce the vanishing gradient problem. Induces sparsity in a network, which is good (faster training and more robust models), but may have too much of the network "turn off", causing "the dying ReLu" problem
- **GeLu** – developed to address the dying ReLu prpblem by adding a gaussian bump to the ReLu function – my recommended default

5. Appropriate Last Layers and Loss Functions

- You can mix and match different activation functions, but your choice of the size of and activation function of the output layer in a neural network is problem-dependent
- Binary Classification
  - Output is a probability for a single class
  - Sigmoid activation function on final layer
  - Binary Cross Entropy as loss function
- Multi-class Classification
  - Output is a probability function over all labels
  - Softmax activation function on final layer
  - Categorical Cross Entropy as loss function
- Multi-label Classification
  - Output is independent probabilities for each label
  - Sigmoid activation function on final layer
  - Binary Cross Entropy as loss function

Deep Learning

1. What is Deep Learning?
2. Deep Learning Pros and Cons
3. Exploding and Vanishing Gradient Problem
4. Be able to explain and/or draw pictures of how the following deep learning architectures work:
    a. Deep Feed Forward Neural Network
    b. Encoder-Decoder
        a. Auto-encoder
        b. Transfer Learning with Encoder-Decoders
    c. Convolutional Neural Network
    d. Recurrent Neural Networks (RNN, ~~LSTM, GRU~~)
        i. For a token classification task
        ii. For a text classification task
    e. Attention ~~and Transformers~~

1. What is Deep Learning

- Technically any neural network with 3 or more layers is a deep neural network
- But, the real purpose of deep learning is to automate (or at least enhance) the feature engineering process.

Deep learning accepts raw data as input and automatically learn and classifies features

1. What is Deep Learning

- All these architectures are big neural networks.
  - They consist of neurons
    - Weighted sum through a nonlinear activation function (e.g. sigmoid, softmax, relu)
  - They may consist of other operations
    - Multiply, add, concatenate, max, average
  - All operations must be (and are) differentiable
- They all minimize loss
  - This is done iteratively through back-propagation
  - Back-propagation is a method to calculate the derivative with respect to each weight
  - This allows you to adjust each weight
    - It is just like batch-based gradient descent (with maybe a few little tweaks)

2. Deep Learning Pros and Cons

- Pros
  - Automatically learns features
    - This allows us to accomplish tasks where feature engineering is expensive/difficult
    - Most importantly allows us to accomplish tasks that we perform intuitively (we don’t know what features to use)
      - Tasks such as Machine vision, NLP, speech to text, etc.

2. Deep Learning Pros and Cons

- Cons
  - Difficult to tune – tons of hyperparameters – both in terms of network structure, and the parameters of individual network component hyperparameters
  - Extremely data hungry – for good performance we need lots (millions) of annotated samples and annotation is expensive, and often not possible
  - Extremely Computationally expensive
    - All that data and the complexity of the models mean it takes a lot of computers and a lot of electricity to train deep neural networks
    - The need for massive data and computational resources is part of the reason that deep learning success is often dominated by big tech companies (Facebook, Google, etc..). They have access to data and have the resources to exploit it
  - **Not interpretable**

3. Exploding and Vanishing Gradient Problem

Exploding and Vanishing Gradient:

$$g_{u,v} = z_u * h_v * (1 - h_v) * e_v$$

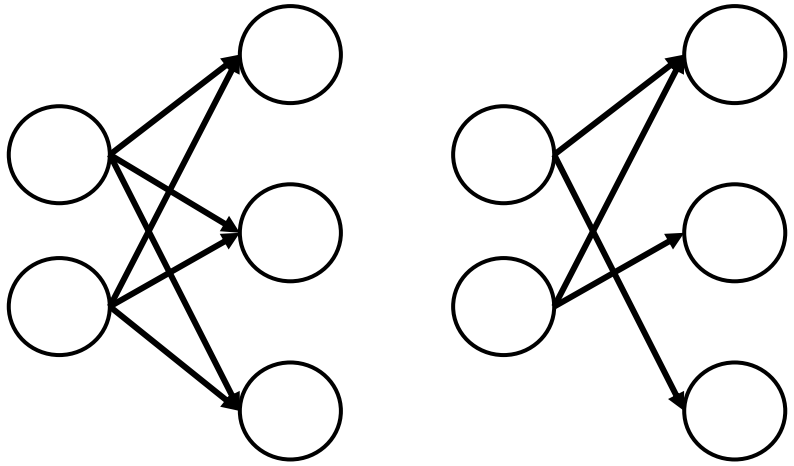
The gradient is a product of values, and the error is itself a product of values.

$$e_v = \left(\sum_{k=1}^{m} \theta_k * e_k \right) = \left(\sum_{h=1}^{c} \theta_h * \left(\sum_{l=1}^{b} \theta_l * \left(\sum_{j=1}^{a} \theta_j * \left(\sum_{i=1}^{n} \hat{y}_i - y_i \right) \right) \right) \right)$$

- As the depth of the network increases, the length of the product to calculate the error increases.
- This means that for deep networks, the product can have extreme behavior. Either going towards zero (**vanishing**), or going towards infinity (**exploding**).
- The vanishing gradient problem has motivated many different deep learning architectures

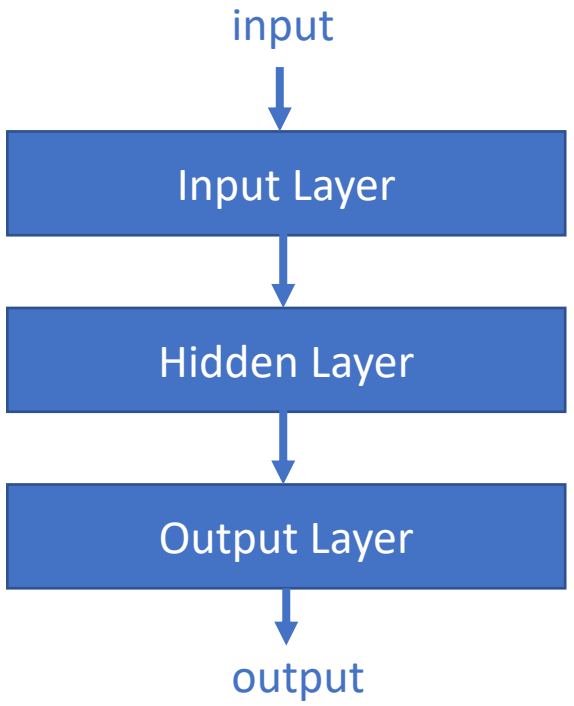
4a. Deep Feed Forward Neural Network

- At their most basic, deep neural networks contain 3 or more layers
- A very simple deep neural network is a **densely connected** feed-forward neural network



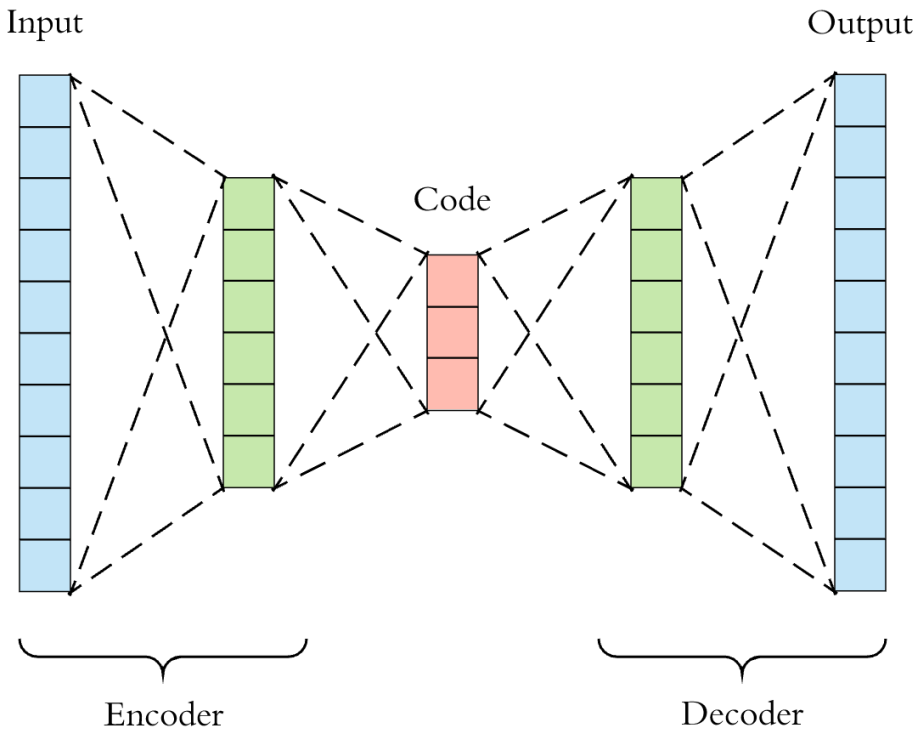
Dense

Not Dense



4b. Encoder-Decoder

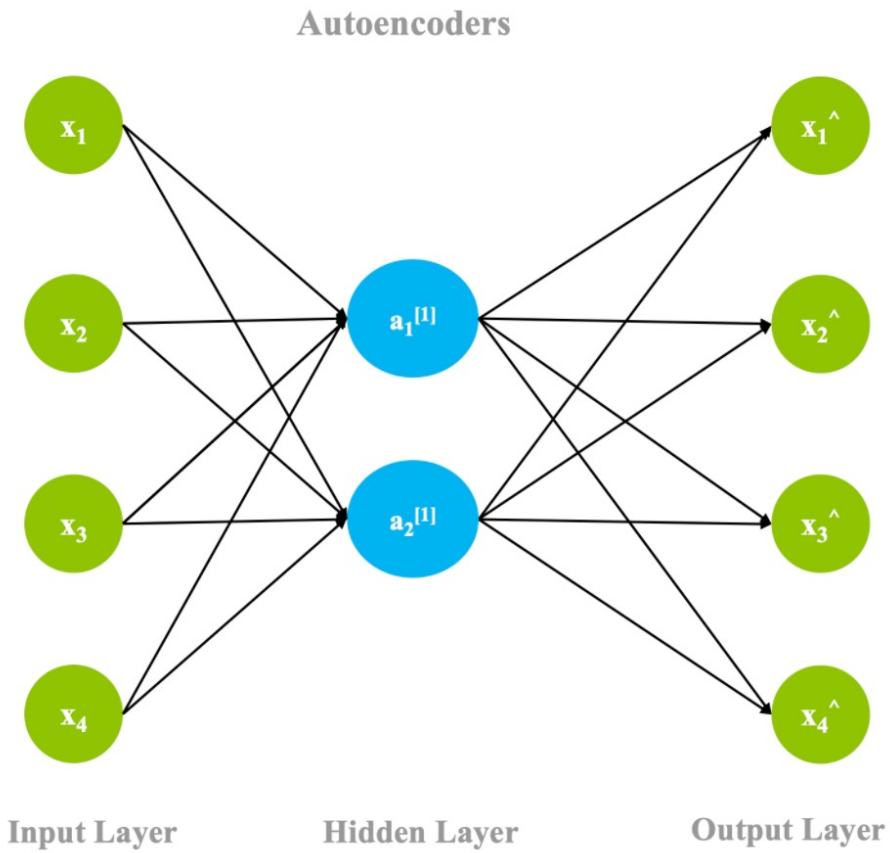
- Encoder-Decoders have two parts
  1. Encoder – represent input in some encoded (typically compressed format)
  2. Decoder – use the encoded representation as input for some task



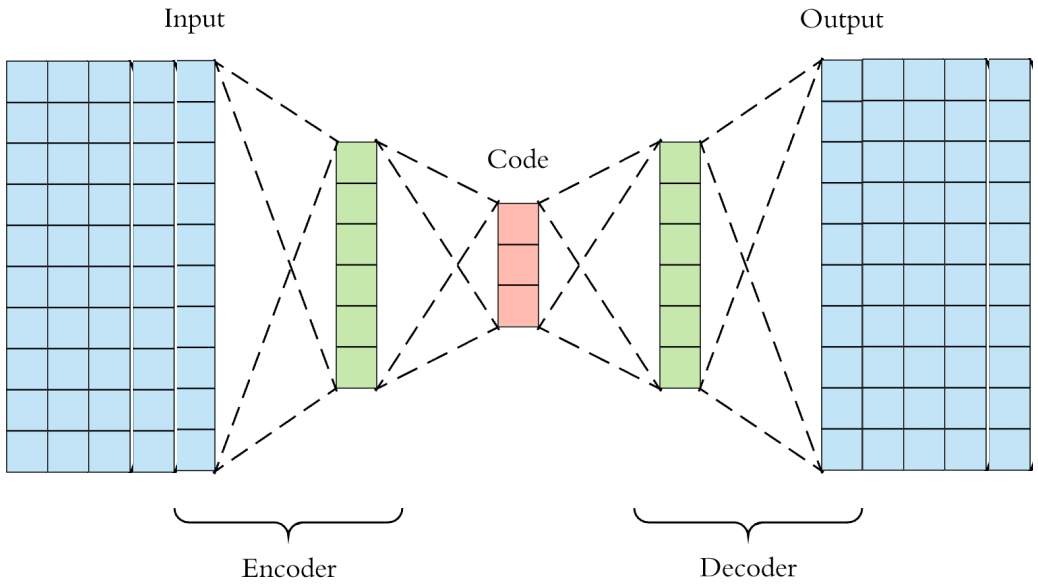
4bi. Auto-Encoder

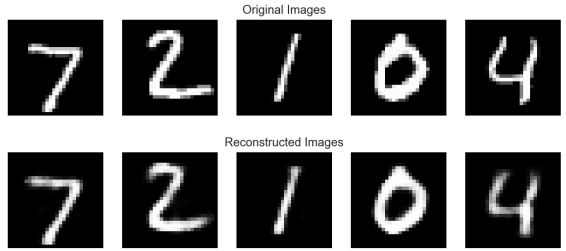
- ANN composed of 3 layers: input, hidden, and output

1. The input layer is encoded into the hidden layer
2. The hidden layer contains the compressed representation of the original input
   - The number of nodes in the hidden layer should be much less than the number of nodes in the input layer
3. The output layer aims to reconstruct the input layer



4bii. Transfer Learning with Encoder-Decoders



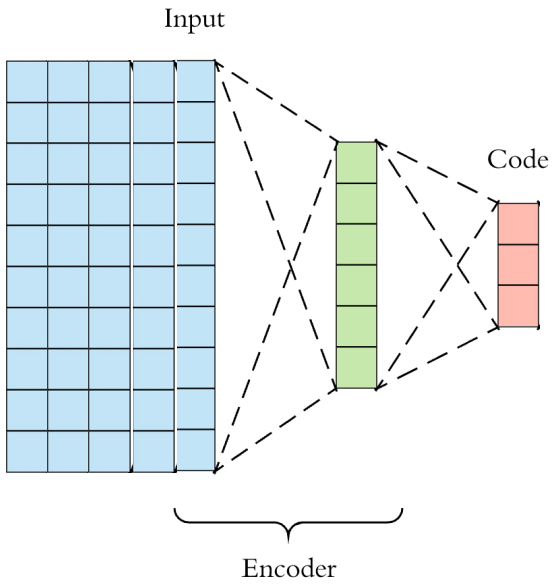


What if we remove the decoder part of the architecture and put a neural network for a different task on?

The encoder gets really good at compressing the images into a meaningful representation that removes noise and that is good at generalizing to new samples

Input and output are an image
(so I added more vectors)

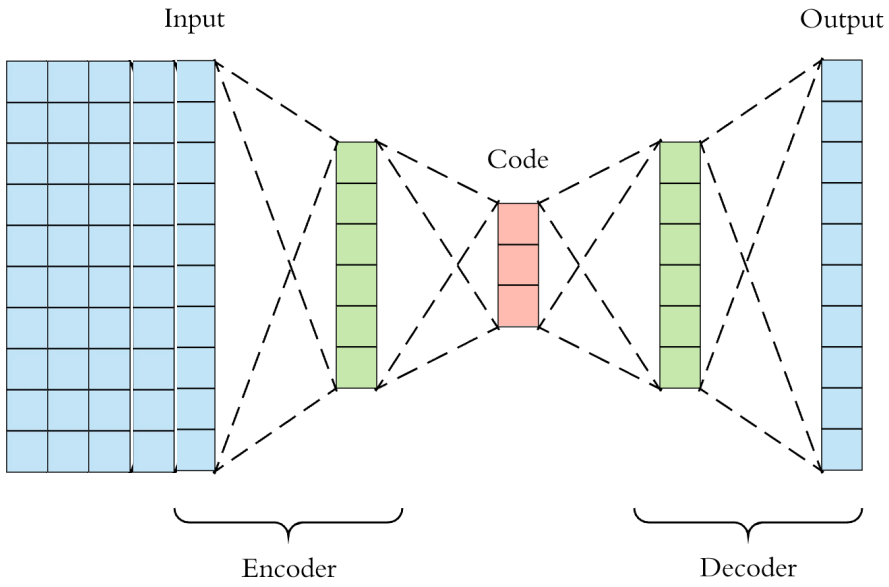
4bii. Transfer Learning with Encoder-Decoders



This is a great compressed representation of the input image

Now we have a network that is really good at encoding things

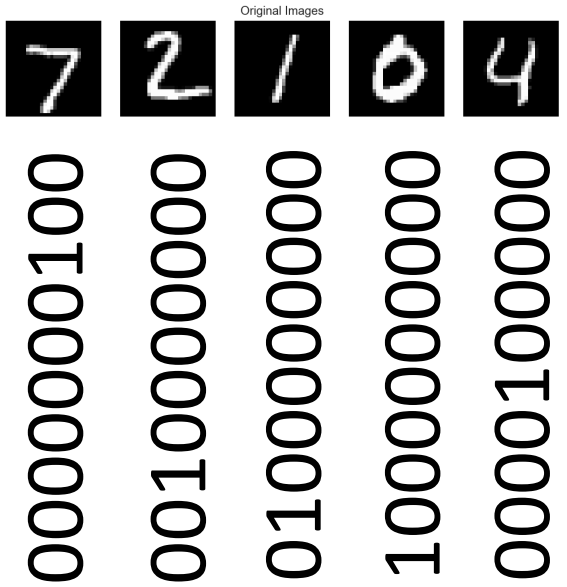
4bii. Transfer Learning with Encoder-Decoders



Original encoder network

New Decoder Network

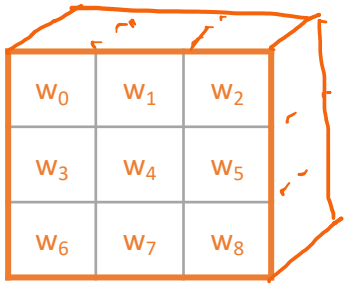
1. Remove the old decoder
2. Replace it with a new network
3. Train the network for the new task



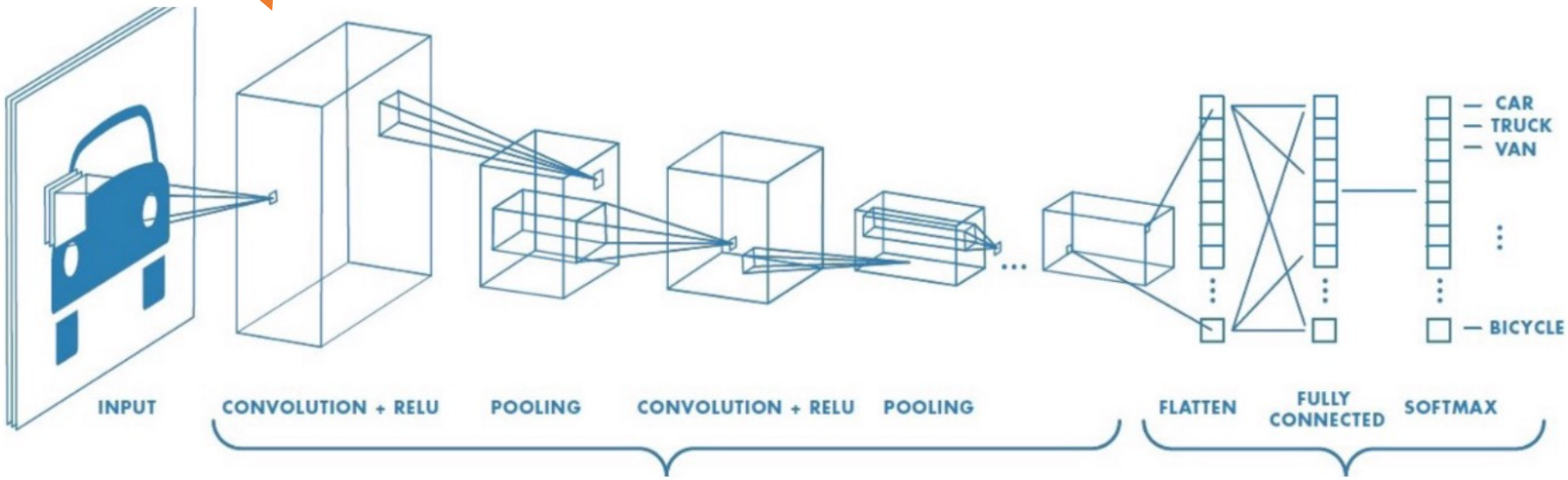
New task is to predict which number is written

4c. Convolutional Neural Network

Convolution layer automatically finds *n m x m x 3* filters which are applied to the whole image



Creates a convolution layer which is the filter applied at each pixel
Each layer of the convolution layer is a different filter, so the convolution layer is dimensionality (*n x length x width*)



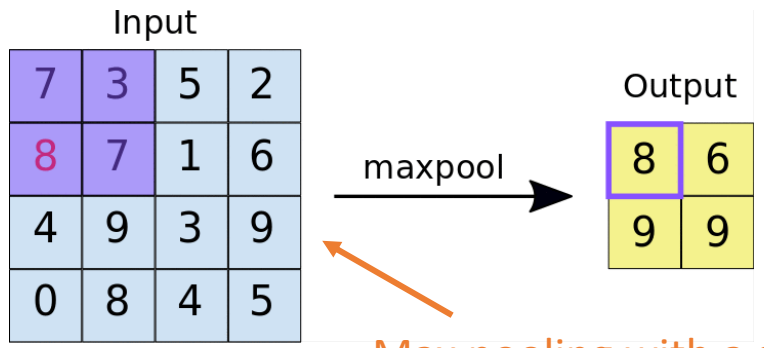
Convolutional Neural Network

Feed Forward Neural Network

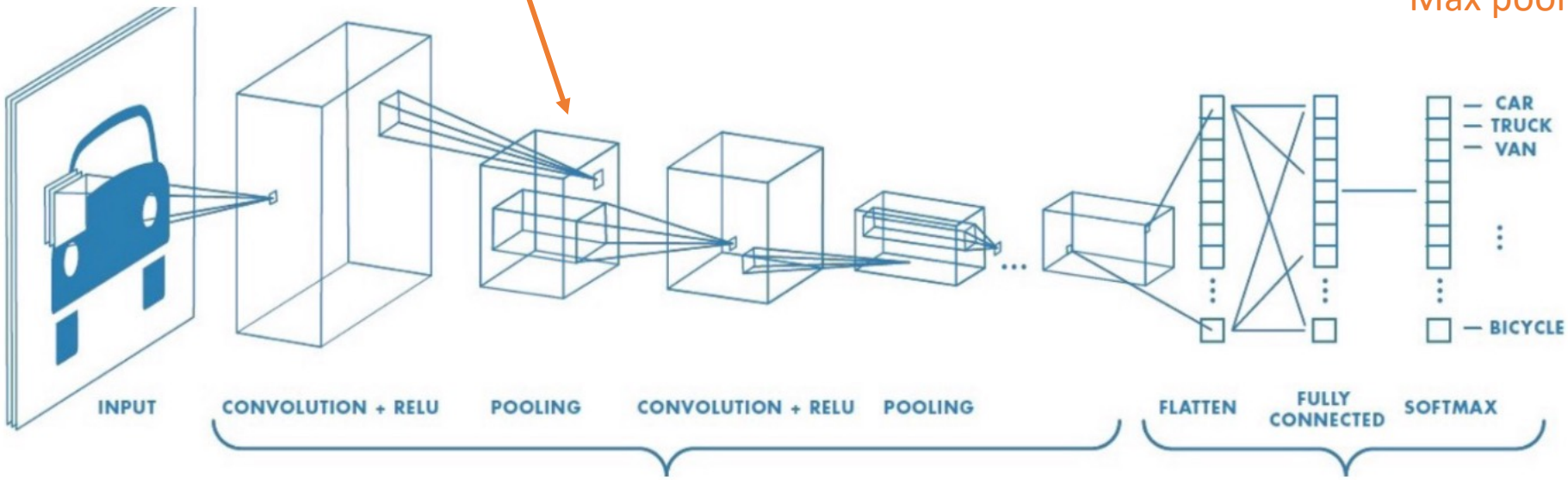
Image from: https://medium.com/swlh/computer-vision-with-convolutional-neural-networks-22f06360cac9

4c. Convolutional Neural Network

We “pool” the convolution layer to decrease its size. Max pooling is common, in which we select a max value within a “stride” sized box, then move the box one stride and continue



Max pooling with a stride 2x2



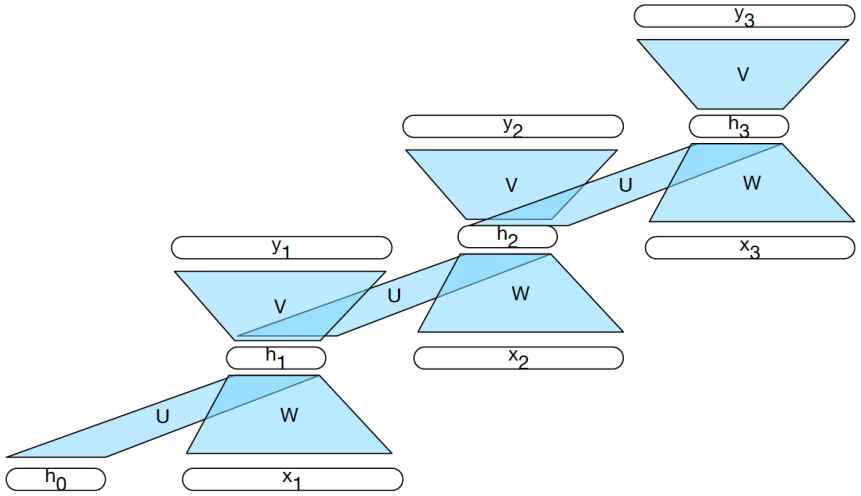
Convolutional Neural Network

Feed Forward Neural Network

Image from: https://medium.com/swlh/computer-vision-with-convolutional-neural-networks-22f06360cac9

4d. Recurrent Neural Network (RNN)

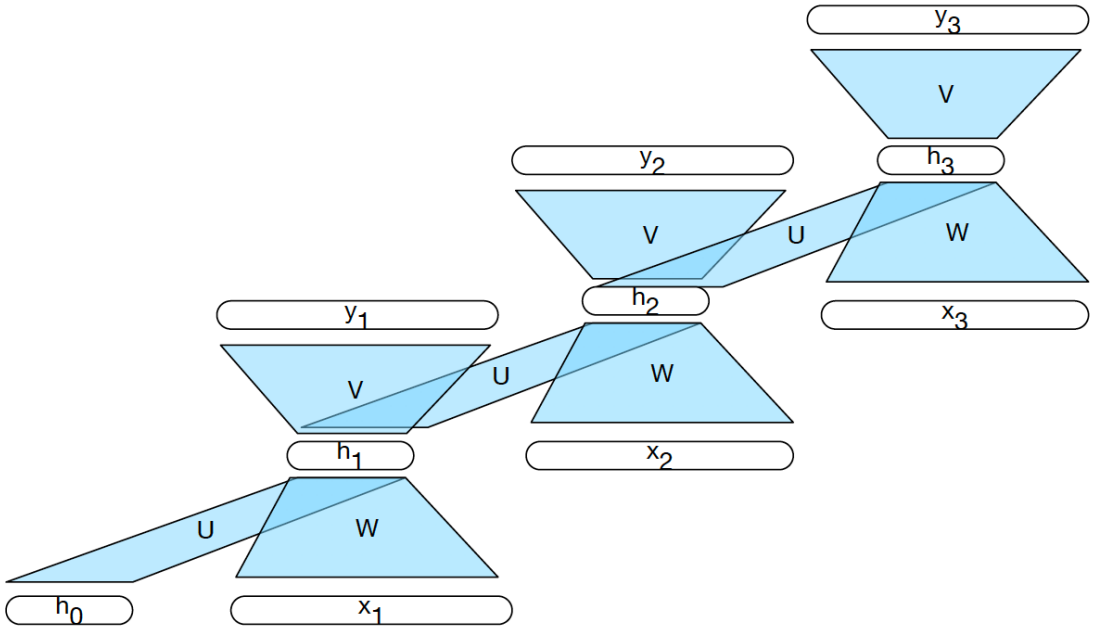
- RNN's key characteristic is that they have some kind of feedback within the network
  - i.e. they are not fully feed-forward
- This feedback gives them a memory of what they have seen before
- RNNs can be unfolded in time and trained with standard back-propagation



Weights U, V and W are shared in common across all timesteps

4di. Recurrent Neural Network for Token Classification

- Input: Pre-trained word embeddings
- Output: probability distribution over the PoS tags generated by a softmax layer serves as output at each time step.
- RNN block represents an unrolled network consisting of an input, hidden, and output layers at each time step, as well as the shared weight matrices.

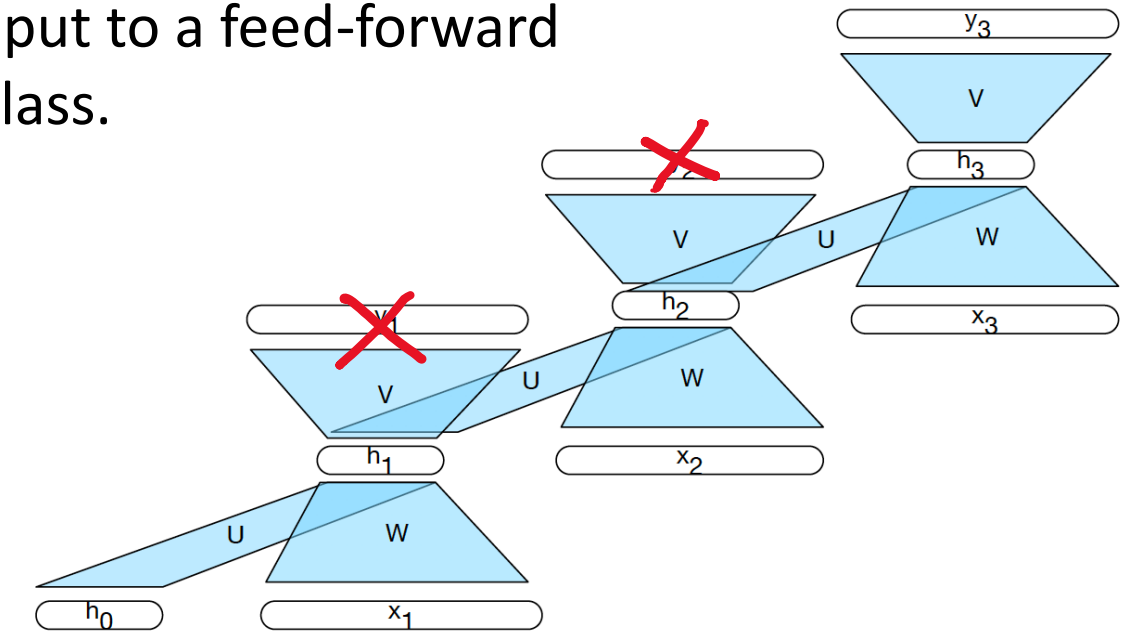


Similar idea for any labeling steps in any sequential data

4dii. Recurrent Neural Network for Text Classification

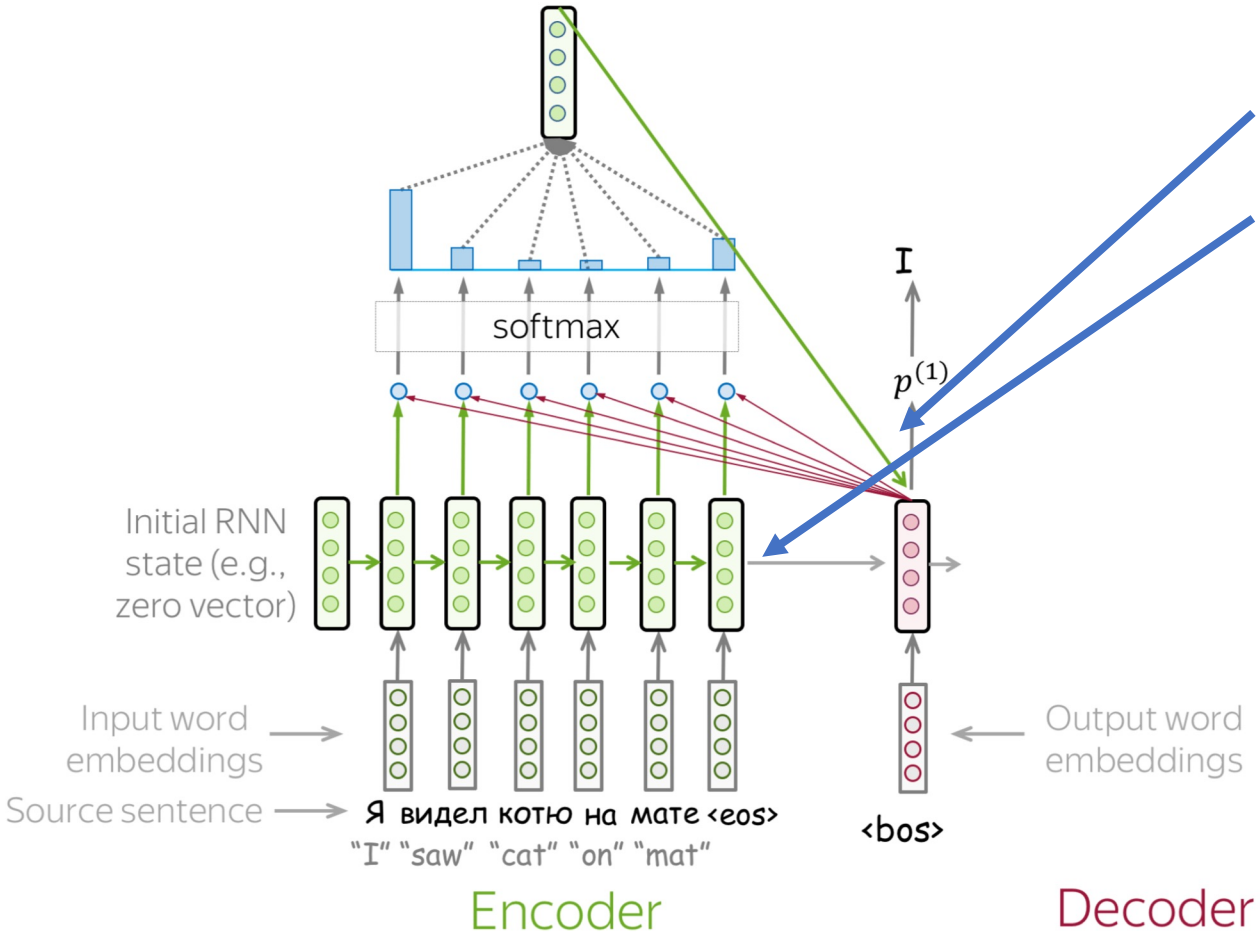
- Hidden layer from the final state of the network is taken to constitute a compressed representation of the entire sequence.
- This representation can serve as the input to a feed-forward network trained to select the correct class.

Use this as the label for the whole sequence



Similar idea for assigning a single label to any type of sequential data

4e. Attention and Transformers



The attention mechanism takes as input:

1) The current decoder state
2) All previous encoder states

The attention mechanism

1. Calculates a score between the decoder state and each encoder state
2. It applies a SoftMax to each score (ensuring all scores sum to 1)
3. Computes the weighted sum of each encoder state and calculated score

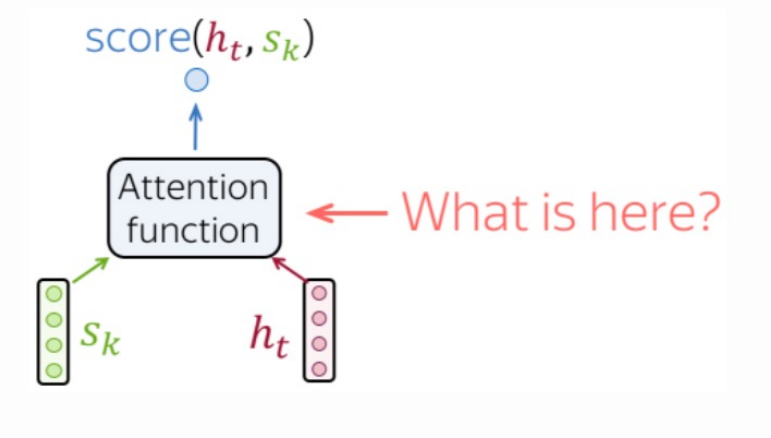
4e. Attention and Transformers

- …so the attention output is just a weighted representation of the encoder states
- The weight indicates the amount of attention applied to that state

How attention scores are calculated can vary

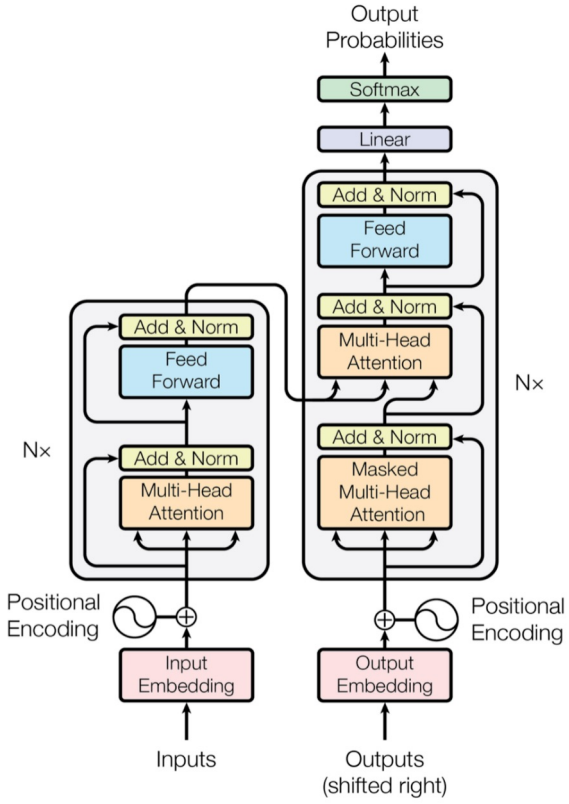
Common methods are:

1. Learned weights
2. Dot-product



4e. Attention and Transformers

- A **transformer** is multiple stacked attention mechanisms and feed forward neural networks



Ensemble Methods

1. Ensemble Methods
2. Stacking
3. Bagging
4. Boosting
5. Random Forest

1. Ensemble Methods

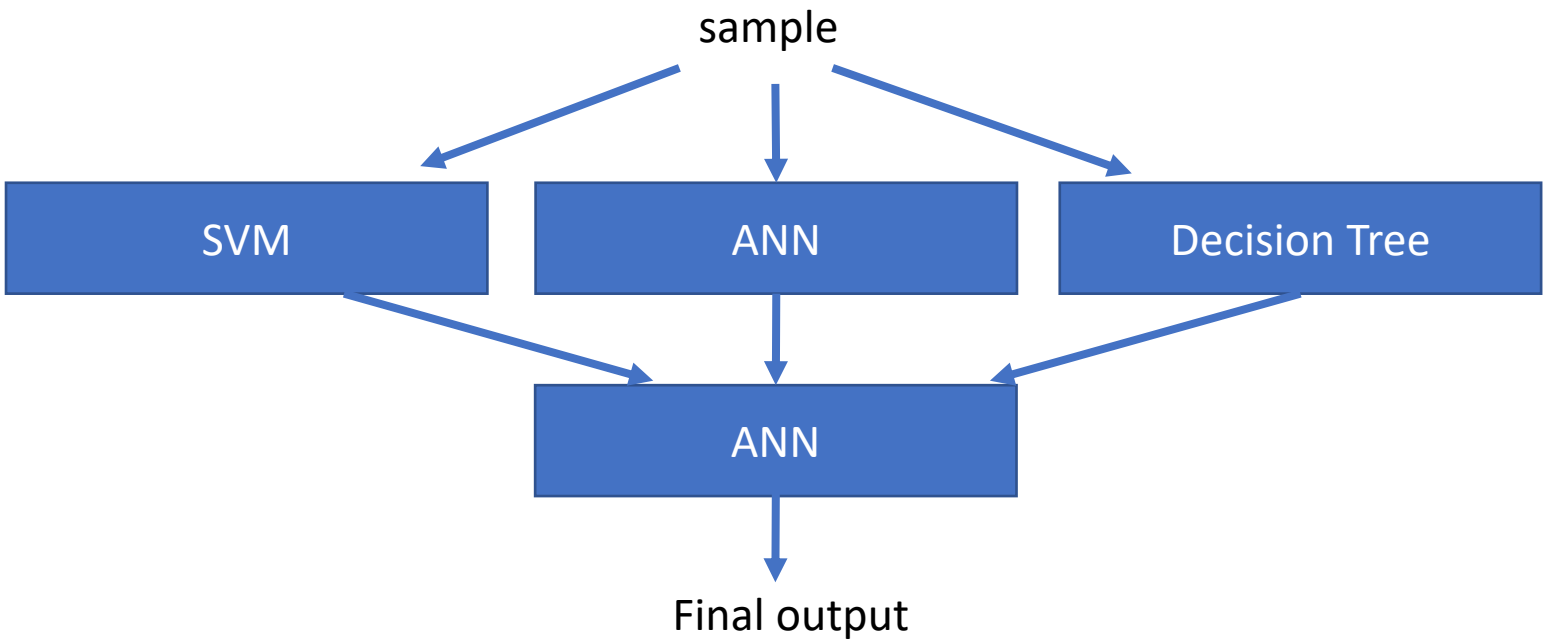
- Based on the idea that many simple learners better than a single complex learner
- Weak Learner – a classification or regression algorithm that performs poorly. Usually (but not strictly) they are computationally simply. Weak learners typically are less prone to overfitting
- Ensemble Method – a method to combine the outputs of different algorithms. Ensemble methods hope to lessen the potential for overfitting

1. Ensemble Methods

- Stacking – an ensemble method that uses the outputs of classifiers as input into another classifier
- Bagging – an ensemble method that uses bootstrap sampling to generate many independent models on a dataset
- Boosting – an ensemble method in which classifiers are applied sequentially, and the prediction performance of one classifier affects how the next classifier is built

2. Ensemble Methods: Stacking

- Several model outputs are input into another model
- Ensemble methods can be used with any learner
  - It doesn't have to be a weak learner



3. Ensemble Methods: Bagging

Bagging = Bootstrap Aggregating

- Given a training set, *D* of size *n*
- Bagging generates *m* new training, D_i of size *n'* sets using bootstrap sampling
  - That is by sampling from *D* uniformly and with replacement
- By sampling with replacement, some observations may be repeated in each D_i
- Sampling with replacement ensures each D_i is independent from its peers as it does not depend on previous chosen samples

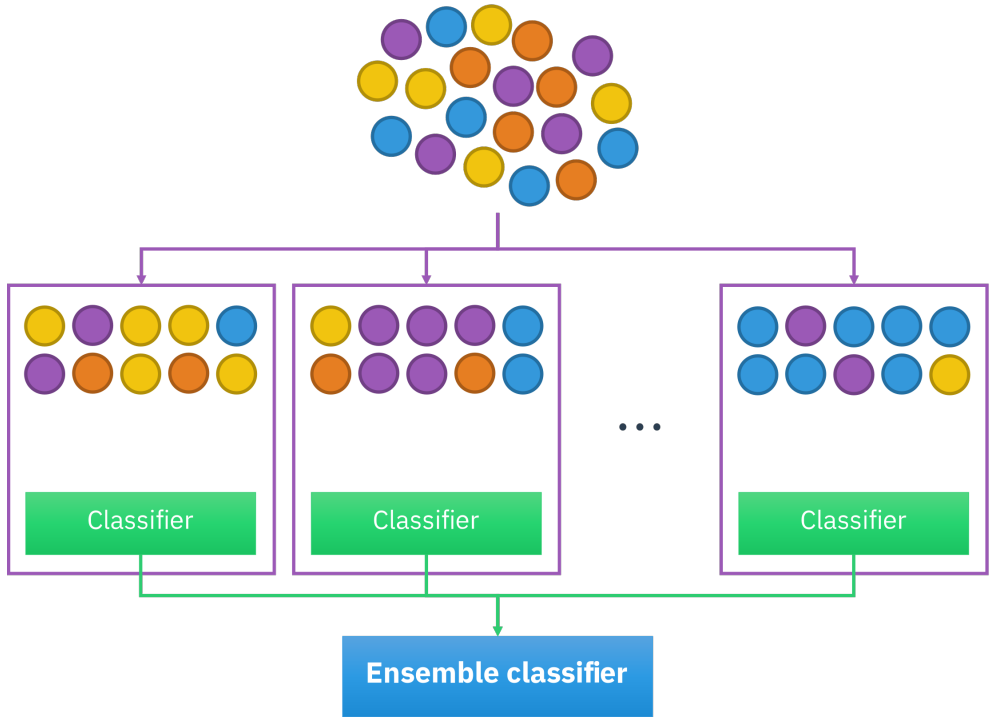


Image From: https://en.wikipedia.org/wiki/Bootstrap_aggregating

3. Ensemble Methods: Bagging

Bagging = Bootstrap Aggregating

- A single classifier is on each D_i
- This results in *m* models
- Given a new sample, a single output is aggregated by:
  1. averaging for regression
  2. voting for classification

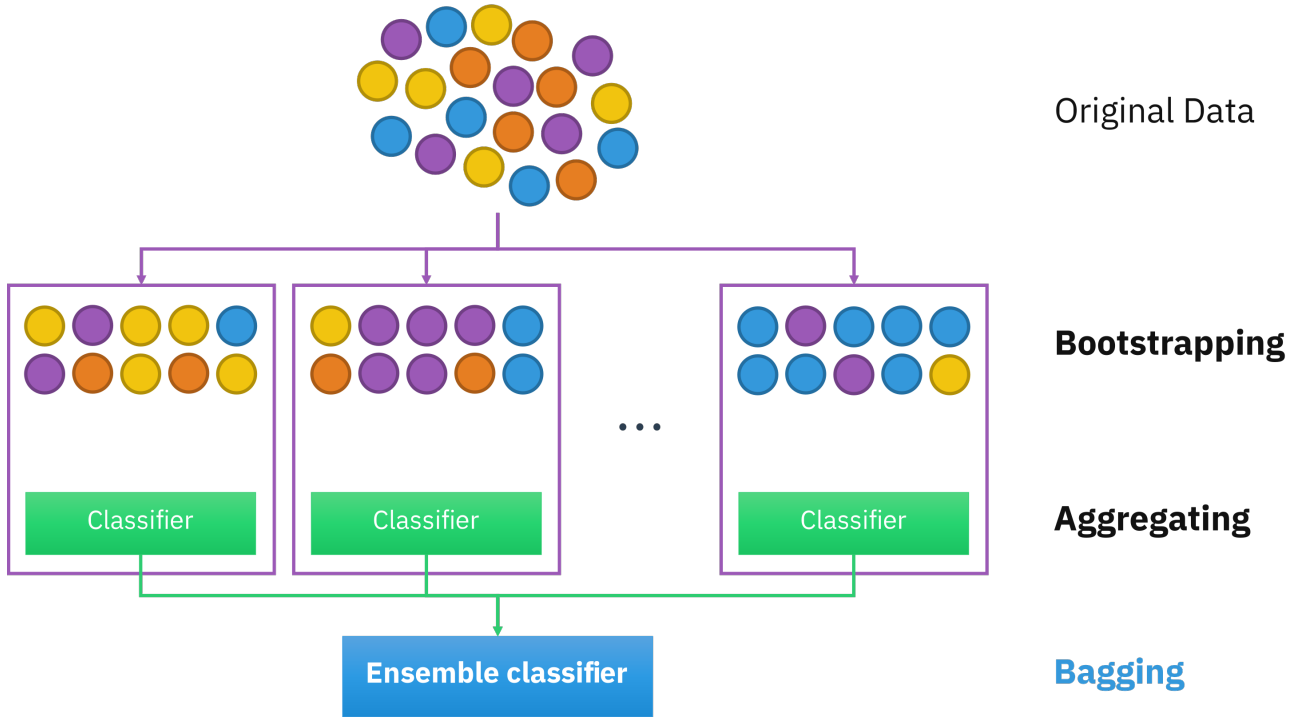


Image From: https://en.wikipedia.org/wiki/Bootstrap_aggregating

4. Ensemble Methods: Boosting

- Boosting consists of creating a series of weak learners.
- The performance of the previous learner influences the creation of the next learner
- The predictions of each learner are combined using an ensemble method (typically weighted voting)

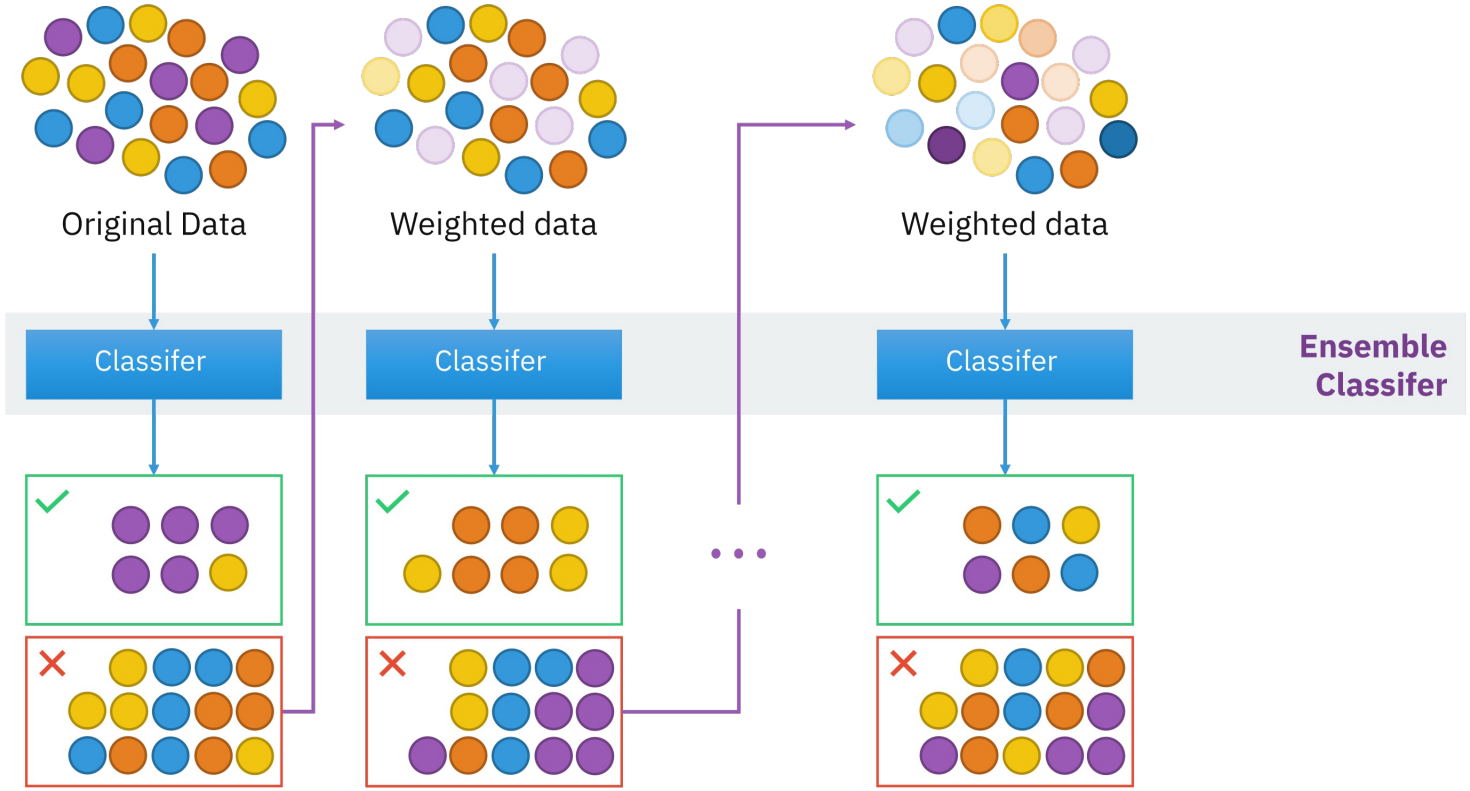


Image From: https://en.wikipedia.org/wiki/Boosting_(machine_learning)

4. Ensemble Methods: Boosting

- Boosting turns a series of weak learners into a strong learner
- There are a variety of boosting methods, at their core most consist of iteratively:
  1. Iteratively learning weak classifiers with respect to a distribution
  2. Weighting the vote of each classifier relative to the classifier's performance
  3. Adding their predictions to a final strong classifier.
- Between iterations of weak learner creation the importance of each sample in the training data is reweighted based on if it was correctly or incorrectly classified

5. Random Forests

- Like bagging, but we use a subset of features for each bootstrapped sample
- For each split, select *m* random features.
  - $\sqrt{number\_of\_features}$ is a good default choice, but in general it is a hyperparameter
  - Find optimum split among those *m* features.
  - Repeat

- This means that the possible features at each split is randomized

Note: if *m* = the number of features of the original model, then this is bagging

Terminology

- Supervised Learning
- Unsupervised Learning
- Self-supervised Learning
- Transfer Learning
- Deep Learning
- Residual Connection
- LASSO Regularization = L1 Regularizer
- Ridge Regression = Linear Regression with L2 Regularizer