Machine Learning Primer

1. Machine Learning Overview

(supervised) machine learning

- ML systems learn how to combine input to produce useful predictions on never-before-seen data.
- supervised learning is used whenever we want to predict a certain outcome from a given input, and we have examples of input/output pairs. We build a machine learning model from these input/output pairs, which comprise our training set. Our goal is to make accurate predictions for new, never-before-seen data. Supervised learning often requires human effort to build the training set, but afterward automates and often speeds up an otherwise laborious or infeasible task
- Discrete data – classification or categorization problem
- Continuous data – regression

Unsupervised Machine Learning

- Training data includes only unlabeled features, no labels
- Unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data
- Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. **The goal of unsupervised learning is to find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format.**
- Task is to cluster data into groups
- Discrete output – clustering
- Continuous output – dimensionality reduction

Labels

- **A label is the thing we're predicting—the y variable in simple linear regression**. The label could be the future price of wheat, the kind of animal shown in a picture, the meaning of an audio clip, or just about anything

Features

- A feature is an input variable—the x variable in simple linear regression. A simple machine learning project might use a single feature, while a more sophisticated machine learning project could use millions of features

Examples

- An example is a particular instance of data, x. (We put x in boldface to indicate that it is a vector.) We break examples into two categories:
- ***labeled examples***
    - Use labeled examples to train the model
    - A labeled example includes both feature(s) and the label
- ***unlabeled examples***
    - An unlabeled example contains features but not the label.
- Once we've trained our model with labeled examples, we use that model to predict the label on unlabeled examples

Models

A model defines the relationship between features and label. For example, a spam detection model might associate certain features strongly with "spam". Let's highlight two phases of a model's life:

- ***Training*** means creating or learning the model. That is, you show the model labeled examples and enable the model to gradually learn the relationships between features and label.
- ***Inference*** means applying the trained model to unlabeled examples. That is, you use the trained model to make useful predictions (y'). For example, during inference, you can predict medianHouseValue for new unlabeled examples.

Regression vs. classification (two types of supervised machine learning problems)

- A regression model predicts continuous values. For example, regression models make predictions that answer questions like the following:
    - What is the value of a house in California?
    - What is the probability that a user will click on this ad?
    - the goal is to predict a continuous number, or a floating-point number in programming terms (or real number in mathematical terms). Predicting a person's annual income from their education, their age, and where they live is an example of a regression task
- A classification model predicts discrete values. For example, classification models make predictions that answer questions like the following:
    - Is a given email message spam or not spam?
    - Is this an image of a dog, a cat, or a hamster?
    - **the goal is to predict a class label, which is a choice from a predefined list of possibilities.**
        - Separated into binary and multiclass classification

Generalization, Overfitting, and Underfitting

- **Generalization:** If a model can make accurate predictions on unseen data, we say it is able to generalize from the training set to the test set. We want to build a model that can generalize as accurately as possible.
- **Overfitting:** Building a model too complex for the amount of information we have is called overfitting. ***Overfitting occurs when you fit a model too closely to the particularities of the training set and obtain a model that works well on the training set but is not able to generalize to new data***
    - a predictor might work very well on the training set, yet produces poor performance on the new testing set or real word
- **Underfitting:** Choosing too simple a model is called underfitting
    - Underfitting refers to a model that can neither model the training data nor generalize to new data.

Note: *The more complex we allow our model to be, the better we will be able to predict on the training data. However, if our model becomes too complex, we start focusing too much on each individual data point in our training set, and the model will not generalize well to new data.*

Note: *Model complexity is intimately tied to the variation of inputs contained in your training dataset: the larger variety of data points your dataset contains, the more complex a model you can use without overfitting*

k-Nearest Neighbors

- The k-NN algorithm is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset. To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset—its "nearest neighbors."
- ***k-Neighbors classification***
    - In its simplest version, the k-NN algorithm only considers exactly one nearest neighbor, which is the closest training data point to the point we want to make a prediction for. The prediction is then simply the known output for this training point
    - ***When considering more than one neighbor***, we use voting to assign a label. This means that for each test point, we count how many neighbors belong to class 0 and how many neighbors belong to class 1. We then assign the class that is more frequent: in other words, the majority class among the k-nearest neighbors
    - Considering more and more neighbors leads to a smoother decision boundary. A smoother boundary corresponds to a simpler model. In other words, using few neighbors corresponds to high model, and using many neighbors corresponds to low model complexity
    - **Impact of choosing lower and higher values of k:**
        - *Using a single neighbor (k = 1) results in a decision boundary that follows the training data closely*
            - *Fewer neighbors leads to high model complexity*

        - *As we consider more neighbors (k =2, 3….) leads to a smoother boundary. A smoother boundary corresponds to simpler model*
            - *Using many neighbors leads to low model complexity*
        - *In an extreme case, where we have k = no. of observations each test point would have exactly the same neighbors (all training points) and all predictions would be the same -> that is the class that is most frequent in the training set*

- Training and testing data are from the same distribution
    - However, the learner does not know anything about the underlying distribution
    - Also assume the correct labels are provided while training

Empirical Risk Minimization

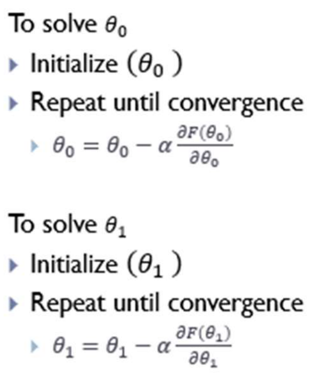
- The goal of a learning algorithm is to find a predictor **h** that minimizes the error with respect to the unknown distribution **D** and labelling function **f**
- Since the learner does not know what D and f are, the true error is not available to the error
    - Therefore, the training error is used
- *ERM Inductive Bias*
    - By restricting the learner to choosing a predictor from H, we bias it toward a particular set of predictors. Such restrictions are called inductive bias.
- *ERM: Understanding the Training Set*
    - The training set **S** is a window through which the learner gets partial information about the distribution **D** over the world and the labelling function **f.**
    - **The larger the sample gets, the more likely it is to reflect D and f**
- ERM: Corollary

Linear Regression

Linear Regression (With single feature)

- Figure out the relationship between feature and label (for example, relationship between price (y) and square feet of house (x)).
- Function y = f(x) that maps from x to y
- Training stage --- figure out what the coefficients are good for building the model
    - Testing stage – make predictions
- **What are the best coefficients for the linear model? –** goal of regression in machine learning
- ***Linear Regression: Cost function***
    - Choose parameters so that f(x) is close to y for all training samples (x,y)
    - To find optimal coefficients (theta_0, thetha_1) – use the **iterative method**
        - Start with thetha_0, thetha_1 as 0
        - Keeping changing values of theta to reduce the function (thetha_0, theta_1)
        - Stop when certain conditions are satisfied

        - ***Pros***
            - Easy to implement
        - ***Cons***
            - Difficult to converge
            - Hard to find solutions when solving highly complicated loss functions
- ***<u>Quadratic Functions</u>***
    - Minimize F(theta_0) using the iterative function
    - Start with some initial value of thetha_0
    - Keep changing thetha_0 to reduce F(theta_0)
    - Stop while certain conditions are satisfied
    - ***<u>Gradient Descent Method</u>*** – following the gradient direction to reach the minimum point of the cost function
        - <u>Method description</u>



To solve θ_0

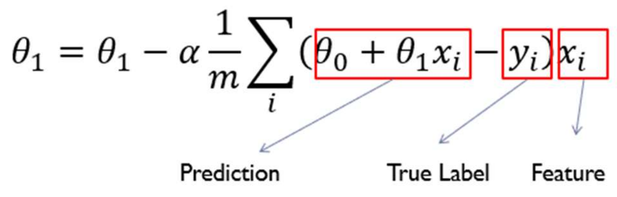
- Initialize (θ_0)
- Repeat until convergence
    - $\theta_0 = \theta_0 - \alpha \frac{\partial F(\theta_0)}{\partial \theta_0}$

To solve θ_1

- Initialize (θ_1)
- Repeat until convergence
    - $\theta_1 = \theta_1 - \alpha \frac{\partial F(\theta_1)}{\partial \theta_1}$

<u>Gradient Descent Method</u>

- Gradient Descent is an optimization algorithm for finding a local minimum of a differentiable function. Gradient descent is simply used in machine learning to find the values of a function's parameters (coefficients) that minimize a cost function as far as possible.
- Initialize (theta_0, theta_1)
- Repeat until convergence: theta_j = theta_j – alpha(Df/dthetha_j) for j = 1, 2….



$$\theta_1 = \theta_1 - \alpha \frac{1}{m} \sum_i (\boxed{\theta_0 + \theta_1 x_i} - \boxed{y_i}) \boxed{x_i}$$

    -
- ***<u>If gradient is 0, prediction is equal to true label</u>***
- Update theta_0 and theta_1 continuously
- Learning rate (alpha) – stepsize
    - Should be set empirically
    - *If alpha too small, convergence will be slow*
    - *If alpha is too large, fail to converge or diverge*
    - *Fixed alpha over time*

- **Batch Gradient Descent**
    - At each step, access all training samples
    - Very time consuming if dataset is very large
- **How to improve convergence**
    - *Access more training samples*
        - Each iteration gets a high quality read
    - *Learning Rate*
        - Make learning rate smaller or bigger
    - *Better initialization*
- **Variants of GD**
    - *Full Bath or Batch Gradient* – time consuming but most precise
        - *all the training data is taken into consideration to take a single step*
    - *Mini batch* – access a portion of the training samples at each iteration
        - *Most popular in modern machine learning*
    - *Online Learning* - Access a single training sample at each iteration

- **Testing**
    - Once a regression model is trained, apply prediction function over each testing sample and compare its prediction (y_i hat) to true label (y)
        - Both labels are real- valued
    - L2 error: (y_i – y_i_hat_)^2
    - L1 error: |y_i – y_i_hat|
    - Coefficient of determination or R^2 is used to measure fit of model – how well observed outcomes are replicated by the model
        - R^2 = 1, indicates that the predicted labels exactly match the true labels
        - R^2 = 0, indicated that the model already predicts y_bar

Note on Customization of Cost Function:

- May want to assign a weight to each sample in the cost function, to indicate how important a sample is (sample quality tends to be different – some our more important than others. Differentiating high quality and low-quality data)
    - The weight can be predefined, or the weight can be taught/learned – curriculum learning
- Instead of using random thetas, pick up two samples and try to solve theta. This theta would be the initial value, in the hope that the theta calculated would be better than a randomization
- Using different alpha (learning rates) for iterations
    - Large alpha for first 500 iterations, then using a smaller alpha for next 1000 iterations
    - Using second order derivative

<u>Dealing with Qualitative Features</u>

- Some features/predictors are not quantitative but are qualitative, taking a discrete set of values
  - Categorical predictors
  - Factor variables
- Use MinMaxScaler to scale large values of features. For example, different features have different min and max values, to scale the data (between) and standardize the data between 0 and 1 we use MinMaxScaler.
- If training score is higher than testing score --- Overfitting
  - Often occurs when we have a small size dataset
  - Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.
- **<u>Regularization – (solution to overfitting)</u>**
  - Try to change/modify the cost function
  - An approach to reduce overfitting regularize the values of the coefficients (theta).
    - Make theta value as small as possible

      $$\min F(\theta) + \lambda\, G(\theta)$$
    - where $F(\theta)$ is the cost function, e.g., the least square for linear regression, and $G(\theta)$ is an extra term over the parameters, λ is a constant
  - ***Ridge Regularization (L2 regularization):***
    - *As we lower alpha/lambda the model starts working better with testing data*
      - *Training error improves with lower values of alpha/lambda*
    - *If alpha/lambda is very big could have a underfitting problem*
    - Ridge Regularization: $G(\theta) = G_{ridge}(\theta) = \theta^T\theta = \sum_i \theta_i^2$
  - ***Lasso Regularization (L1 regularization):***
    - *Sparsity - In a high dimensional space, with huge number of features, a Lasso model would make non-important features (thetas) 0. This makes feature selection far easier*
    - Lasso Regularization : $G(\theta) = G_{lasso}(\theta) = |\theta| = \sum_i |\theta_i|$
  - ***Elastic Net Regularizing (combination of Ridge and Lasso):***
    - Elastic Net Regularizing: $G(\theta) = G_{ridge}(\theta) + G_{lasso}(\theta)$

<u>Best Practices</u>

- Feature Processing
  - Processing is needed to be done before even coding a machine learning system
  - Step 1: Feature Scaling
    - Scale features to be between -1 and 1
    - Apply the same scaling to both testing and training samples
  - Step 2: Mean normalization
    - Replace x_i with $x_i - x_u$ to make features have approximately zero mean

- Learning Rate
    - How do we choose learning rate:
        - Too small – slow convergence
        - Too large – no convergence
    - Try learning rate: 0.001, 0.05, 0.01
- Polynomial Regression
    - If you have small features, redundancies are appreciated as it makes the model more robust. For example:

        Case: Housing price

        $$f_\theta(x) = \theta_0 + \theta_1 \times x_1 + \theta_2 \times x_2 + \theta_3 \times x_3$$

        x_1: area

        x_2: size of living area

        x_3: size of yard

        $x_1 = x_2 + x_3$
        -
    - If you have high dimensional data, reduction of redundancies is required
    - Adding higher-order terms to our cost-function/regression function to increase model complexity
- Normal Equation
    - Works well if obtaining training data is difficult or very noisy
    - We are looking for a theta that best fits our observations (we do not need the best fit in machine learning, just need the best approximation)
    - Use a sample from the total number of observations. For example, using the normal equation on 100 samples on a dataset consisting of 1000 observations.
    - Do not need to choose alpha, no iterations, however, need to compute matrix inverse which might be too large
- Selecting Important Features
    - Some features are more important than others
        - Direct Approach: *Subset Approach*
            - For all possible subsets, compute least square fit and choose to balance training error and model size
        - Alternative Approach
            - *Forward Selection:*
                - Begin with null model - only intercept but not predictors
                - Train multiple models using a single variable and add to the null model the variable that results in the lowest RSS
                - Expand the one-variable model to be two-variable models and keep the one with lowest RSS (two)
                - Keep on adding features using hypothesis tests
                - Repeat this process until you reach desired features
            - *Backward Approach:*
                - Start with all variables in the model (let **d** denote the number of variables)

- Remove one variable from the model to get (d-1) variable model; Test all (d-1) variable models and keep the one with the least training error
- Continue the removal process until a stopping rule is reached

---

Classification and Validation

Binary Classification Problems

- Logistic Regression
  - Let

    $$h_\theta(x) = g(\theta^T x)$$

    so that

    $$0 \le h_{\theta(x)} \le 1$$
  - Choices of functions: Sigmoid (between 0 and 1), Tangent (between -1 and 1)
  - **Prediction function** interpretation: estimated probability that output is 1 on input x
  - Understanding: $h_\theta(x)$

    $$h_\theta(x) = g(\theta^T x)$$

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

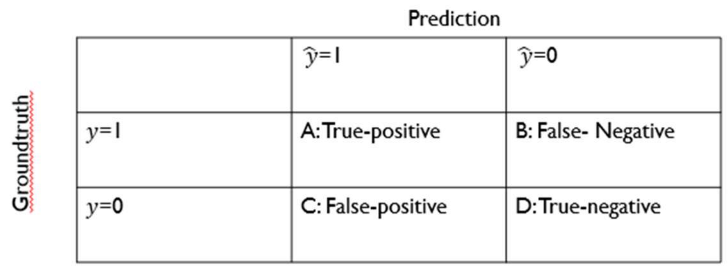
    - If $\theta^T x \ge 0$, then $g(\theta^T x) \ge 0.5$, and it is likely to be y=1
    - If $\theta^T x \le 0$, then $g(\theta^T x) \le 0.5$, and it is likely to be y=0
  - **Cost Function**
    - Cost function:

      $$Cost\ (h_\theta(x_i), y_i) = \begin{Bmatrix} -\log[h_\theta(x_i)] & y_i = 1 \\ -\log[1-h_\theta(x_i)] & y_i = 0 \end{Bmatrix}$$

    - $$F(\theta) = -\boxed{\frac{1}{m}}\left(\sum_i y_i \log h_\theta(x_i) + (1-y_i)\log(1-h_\theta(x_i))\right)$$
  - What is the difference between Least Square Loss and the log loss?
    - Treatment of negative values
      - In Least squares negative values would have a prediction of 1
      - In log loss it would be infinity
    - Cross-entropy is easier to calculate – the order is lower, therefore, easier to optimize
    - ***Optimize Cost Function via Gradient Descent***
- Multi-Class Problems

- One-vs-all (classical way to deal with multi-class problems)
    - For each class **i**, train a logistic regression classifier
    - Train a model for each class – i
    - Apply all models' overs the testing samples, and use the model giving the best predictions
    - Works extremely well in practice

- Evaluation for Classification Problems
  - Missing Rate and False Alarm
  - 1st method: Percentage of correctly classified samples
  - 2nd method: Comprehensive metric – Confusion Matrix
    - For every sample, denote y as the groundtruth label, y_hat as the predicted label
    - We use a confusion matrix to analyze the results of a model



Prediction

Groundtruth

	$\hat{y}=1$	$\hat{y}=0$
$y=1$	A: True-positive	B: False- Negative
$y=0$	C: False-positive	D: True-negative

    - Accuracy: (A+D)/(A+B+C+D)
    - Precision rate for one class (1 or 0): A/(A+C) OR B/(B+D)
    - Recall rate for one class (1or 0): A/(A+B) OR C/(C+D)

<u>Discriminant Analysis</u>

- Popular way to employ Bayesian Theorem to solve classification problems, especially multi-class classification problems.
- For any given feature, use the normal distribution
- To classify sample x to be one of K classes (K>2), we aim to find a class k that maximized the following conditional probability:
    - $p_k(x) = \Pr(y = k|\, x)$
- With Bayes Theorem:
    - $$\Pr(y = k|\, x) = \frac{\Pr(x|y = k)\Pr(y = k)}{\Pr(x)}$$
- Pr(X) will be constant
- Why discriminant analysis
  - When classes are well-separate, logistic regression model are not stable, but LDA/QDA is stable

- When number of samples is relatively small and the features are approximately normal in each class, LDA is much more stable for logistic regress
- Can be applied over multi-class problems

<u>Model Selection</u>

- **<u>Questions to be answered:</u>**
    - How to set the learning rate for gradient descent algorithms
    - How to select the best algorithm from multiple algorithms which are applicable foe the same problem
    - How do we set the best algorithm's parameters?
- **<u>Validation</u>**
    - To select models or model parameters, we further divide the training sample into validation and training samples
    - Validation set is a part of the training set
    - ***Regression: Validation (train model over training sample, then calculate the metric below for the validation set and training set)***
        - L1 error
        - L2 error
        - R^2
        - Compare multiple validation set scores for different models
    - Theorem for Model Validation
        - Try to increase the size of the validation set
    - ***Classification***
        - With prediction functions, one need to make a strategy to make discrete predictions over testing samples
          For binary classification, e.g., the label of a sample x_i
          $$\hat{y}_i = \begin{cases} 1, & if\ f_\theta(x_i) > 0.5 \\ 0, & otherwise \end{cases}$$
        -
        - The predicted labels should be compared to the corresponding true labels for calculating measures of success
        - Compare models with fixed thresholds and without thresholds
        - Fixed Threshold – Confusion Matrix
        - Without Threshold – ROC Curve
            - Evaluate how the systems perform under all possible thresholds (with different decision risks)
            - How the error rate changes with a change in threshold
            - High threshold – high precision – high recall
            - Low threshold – low precision – low recall
            - For each point represents a threshold. For each threshold False Positive Rate and True Positive Rate is calculated

- **ROC is a probability curve and AUC represents the degree or measure of separability**
  - Higher the AUC, the better the model is at predicting 0 classes as 0 and 1 classes as 1
  - ***<u>AUC represents the probability that a true positive and true negative data points will be classified correctly.</u>***
  - when AUC is 0.5, it means the model has no class separation capacity whatsoever.
  - When AUC is approximately 0, the model is actually reciprocating the classes. It means the model is predicting a negative class as a positive class and vice versa.

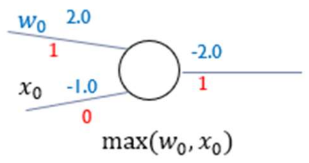
    TPR (True Positive Rate) / Recall /Sensitivity
  - $$\text{TPR /Recall / Sensitivity} = \frac{TP}{TP + FN}$$

    FPR
  - $$\text{FPR} = 1 - \text{Specificity} = \frac{FP}{TN + FP}$$

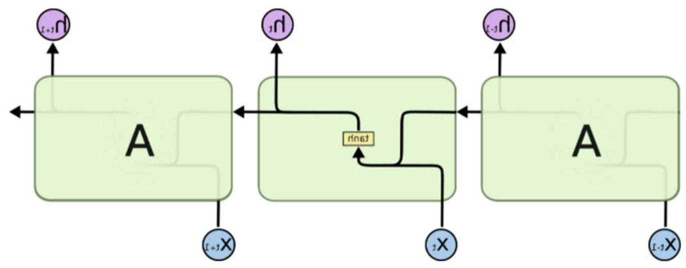
- ***K-Fold Cross Validation***
  - Using every training sample for both training and validation purposes.
  - Partition the training set into K subsets
  - For each subset, train the model using other subsets and validate the model using the subset
  - The average of these K-fold validation errors is used as the measure of success
  - K-fold is not feasible at times as it is time and resource intensive

FeedForward Neural Network

- A neural network is a machine learning model applicable to regression, classification, unsupervised learning etc.
- A neural network is a **series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates**
- Making predictions by a function of a function
- Neural Network structure:
    - Input layer: only one
    - Hidden layer: can be 0 or multiple (tens, hundreds, thousands etc.)
    - Output: multiple
- ***How do neural networks work?***
    - **XOR**
        - The XOR, or "exclusive or", problem is a classic problem in ANN research. It is the problem of using a neural network to predict the outputs of XOR logic gates given two binary inputs. An XOR function should return a true value if the two inputs are not equal and a false value if they are equal
    - **Forward Propagation**
        - To generate some output, the input data should be fed in the forward direction only. The data should not flow in reverse direction during output generation otherwise it would form a cycle and the output could never be generated. Such network configurations are known as feed-forward network. The feed-forward network helps in forward propagation.
        - The hidden layer consists of two functions:
            - Pre-activation function: The weighted sum of the inputs is calculated in this function.
            - Activation function: Here, based on the weighted sum, an activation function is applied to make the network non-linear and make it learn as the computation progresses. The activation function uses bias to make it non-linear.
    - **Backward Propagation**
        - Back-propagation is the essence of neural net training. It is the practice of fine-tuning the weights of a neural net based on the error rate (i.e. loss) obtained in the previous epoch (i.e. iteration). Proper tuning of the weights ensures lower error rates, making the model reliable by increasing its generalization.
        - Add gate: gradient distributor
        - Multiplication gate: gradient switcher
        - Max gate: gradient router



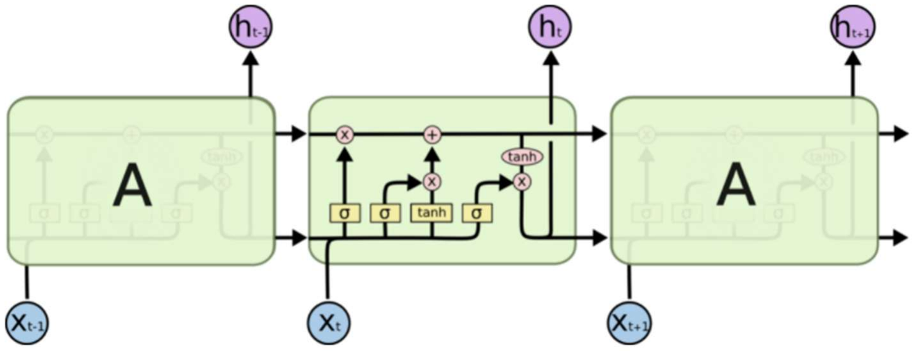
<u>Recurrent Neural Networks</u>

- In feedforward neural networks the output has no relation with time
- RNNSs are designed to model the relationship between sequence of data points and a sequence of output labels (i.e sequence-to-sequence model)
- A class of neural networks that allow previous outputs to be used as inputs while having hidden states
- The output or hidden states of the network at time t are used as inputs to the network at time t+1.
- The network weights are shared over time
- RNNs to a variety of problems: speech recognition, language modeling, translation, image captioning
- **<u>RNN Variants</u>**
    - Multiple input, single output
    - Single Input, single output (Feed Forward)
    - Single Input, Multiple Output
    - Multiple input, multiple output
- RNN Learning
    - Choose cost function
    - Gradient Descent
        - Back propagation through time (BPTT)
- RNN Limitations
    - Training efficiency: the product of multiple matrices would lead to two issues while using gradient descent method:
        - Gradient Shrinkage
        - Gradient Explosion
    - RNN can only model time-wise relationship of fixed-term length
    - Not adaptive
- 

<u>Long Short-Term Memory (LSTM)</u>

- Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies
- LSTM uses constant error flow for RNNs to ensure that gradients do not decay or explode
- Memory cell: used to adapt over time
- **<u>LSTM for Forecasting</u>**

- Model comprises of multiple LSTM cells over time steps and is able to model the time-wise dependencies of the time series
- Might need a large amount of training data samples



- The repeating module in an LSTM contains four interacting layers.
- The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
    - Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.