Ch. 1.

* What is machine learning:-

- Machine learning is a subfield of computer science that is concerned with building algorithms which rely on a collection of examples. These examples can come from anywhere.

* Types of learning:-

1- Supervised learning:-

- The dataset is a collection of labled examples $\{(x_i, y_i)\}_{i=1}^{N}$. Each element x_i among N is called a feature vector. A feature vector is a vector in which each dimension $j=1 \ldots D$ contains a value that describes the example. That value is called a feature and is denoted as $x^{(j)}$

- For all examples in the dataset, the feature at position j in the feature vector always contains the same kind of information.

☆ Continued:-

- The goal of supervised learning is to use the dataset to produce a model that takes feature x as input and outputs information that allows deducing the label for this feature vector.

2- Unsupervised learning:-

- The dataset is a collection of unlabeled examples $\{x_i\}_{i=1}^{N}$. X is a feature vector, and the goal of this algorithm is to create a model that takes a feature vector x as input and either transforms it into another vector or into a value that can be used to solve a practical problem.

Unsupervised Light

- Clustering: The model returns the id of the cluster for each vector in the dataset.

- Dimensionality Reduction: The output of the model is a feature vector that has fewer features than input x

- Outlier detection: The output is a real number that indicates how x is different from a "typical" example in the dataset

3- Semi- Supervised learning

- The dataset contains both labled and unlabeled examples, with unlabled examples being higher. The goal is the same as supervised learning

- It could look counter-intutive that learning could benefit from adding more unlabeled examples. However, when you add more information about the problem, you get better probability distribution ~~table~~.

4. Reinforcement learning:-

- It's a subfield of machine learning where the machine "lives" in an environment and is capable of perceiving the state of that environment as a vector of features. The machine can execute actions in every state. Different actions bring different rewards and could move the machine to another state. The goal is to learn a policy.

- Policy: a function (similar to a model) that takes the feature vector of a state as input and outputs ~~as~~ an optimal action to execute in that state.

solves a problem where decision making is sequential

*How Supervised learning works:-

- It starts with gathering the data as a collection of inputs and outputs. Inputs could be anything. Outputs are usually real numbers or labels.

- You need to convert the inputs into a feature vector. If it's words, one common way is to use a bag of words.

- Then you might need to transform the outputs from labels into number. eg. 0 for answer a and 1 for answer b.

- Support vector machin (SVM):

This algorithm requires that the positive label has the numeric value +1 and negativ label as -1.

~~Start to get the models~~

- SVM sees every feature vector as a point in a high-dimensional space. The algorithm puts all feature vectors on an imaginary hyperplane that seperates examples with positive labels and examples with negative labels.

*Continued SVM....

- The boundary seperating the examples of different classes is called the decision boundary.

- The equation of the hyperplane is given by two parameters, a real-valued vector w of the same dimensionality as our input feature vector x, and a real number b like this:

$$wx - b = 0$$

- where the expression wx means $w^{(1)}x^{(1)} + w^{(2)}x^{(2)} + \ldots + w^{(D)}x^{(D)}$.

- Now, the predicted label for some input feature vector x is given like:

$$y = \text{sign}(wx - b)$$

"Sign" will return +1 if the input is +ve and -1 if -ve.

- The goal of SVM is to leverage the dataset and find the optimal values w^* and b^* for parameters w and b. Once these optimal values are found, the model $f(x)$ is:

$$f(x) = \text{Sign}(w^* x - b^*)$$

★ Continued SVM...

- To predict, take the input and convert it into a feature vector. This will give the prediction +1 and −1.

- How does the machine find w^* and b^*?

- It solves an optimization problem.

- ~~Machines are~~
- Machines are good at optimizing functions under constraints.

- What are the constraints?

- First of all, we want the model to predict the labels correctly. Remember that each example $i = 1, \dots, N$ is given by a pair (x_i, y_i), where x_i is the feature vector and y_i is its label that gives +1 or −1. So the constraints are:

$$wx_i - b \geq +1 \quad \text{if } y_i = +1$$

$$wx_i - b \leq -1 \quad \text{if } y_i = -1$$

* Continued SVM...

- We would also prefer the hyperplane to seperate positive examples from negative examples with the largest margin. A large margin means better generalization. To achieve that, we need to minimize the euclidean norm of w denoted by $\|w\|$.

- So, the optimization problem that we want will look like:

Minimize $\|w\|$ subject $y_i(wx_i - b) \geq 1$ for $i = 1, \ldots, \ldots, N$. The solution of this, given by w^* and b^* is called the statistical model. The process of building a model is called training.

~~This version~~

✱ Why the model works on New data?

- If 2 classes are seperated by a decision boundary, then examples that belongs to each class are located in two different subspaces which the decision boundary creates.

- The more examples you give, then statistically, it's more likely that new negative examples will be located on the plot not too far from other negative examples, same goes for positive examples.

★ Chapter 2: Notations and definitions.

★ Data structures...

- Scalar: a simple numerical value. Variables that take scalar are denoted by an italic letter.
- Vector: an ordered list of scalars. Denoted by bold characters. They can also be arrows pointing to a direction or points in multidimensional space.

Attribute could be ~~anything~~ not just j

- The index j denotes a specific dimension of the vector.

~~★★~~ - Matrix: a rectangular array of numbers arranged in rows and columns.

- Set: an ~~order~~ unordered ~~set~~ collection of elements. eg. {1, 3, 6, 7, 9, 123}
- Capital Sigma (Σ): summation.
- Capital Pi (Π): multiplication.

✱ Operations on Vectors:-

- The sum of two vectors X and Z is defined as: $[x^{(1)}+z^{(1)} + \dots + x^{(n)}+z^{(n)}]$

  ⇑

- Same goes for difference

- A vector multiplied by a scalar is a vector. eg. $Xc = [cx^{(1)} + cx^{(2)} + \dots cx^{(n)}]$

- Dot product of two vectors is a scalar. eg. $wx = [\sum_{i=1}^{n} w^{(i)}x^{(i)}$

  The vectors must be of the same dimensionality.

✱ Functions:-

- a relations that associates each element x of a set X

- The domain of the function, to a single element y of another set Y, the codomain of the function.

- We say that f(x) has a local min. at X=c if $f(x) \geq f(c)$ for ever x in some open interval

✱ Continued functions:-

- Globcel min: The minimal value among all the local minima.

✱ Max and arg Max:-

- Given a set A. The operator $\max_{a \in A} f(a)$ returns the highest ~~not~~ value $f(a)$.

- The operator $\arg\max_{a \in A} f(a)$ returns the element ~~of~~ of the set A that maximizes $f(a)$

✱ Assignment operator:-

- $a \leftarrow f(x)$ means that a gets the new value: the result of $f(x)$.

✱ Derivative:-

- No explanation needed.

✱ Gradient:-

- A vector of partial derivatives
- gradient of $f(x)$ denoted as ∇f

eg. ~~$f(x) =$~~

~~$f(x,y) = x^2 \cos(y)$~~

$$f(x,y) = x^2 \sin(y)$$

Partial derivative w/ respect to x → $\frac{\partial f}{\partial x} = 2x \sin(y)$

Partial derivative w/ respect to y → $\frac{\partial f}{\partial y} = x^2 \cos(y)$

$$\nabla f = [2x \sin(y), x^2 \cos(y)]$$

Random variables:-

- A variable whose possible values are numerical outcomes of a random phenomenon.

- There are 2 types:-

1- Discrete:

- Takes only a countable number of distinct values.

- Probability distribution of a discrete random variable is described by a list of probabilities associated with each of its possible values.

2- Continuous Random variable (CRV):-

- Takes an infinite ~~#~~ of possible values in some interval.

- Probability distribution of CRV:-

- Defined as a probability density function (Pdf).

★ Continued CRV:-

- The expectation of random variable X is denoted as

$$E[X] = \sum_{i=1}^{k} [x_i \cdot P(X = x_i)]$$

- The expectation of a random variable is also called the mean, average, or expected value. denoted by μ.

- Standard deviation (σ):

$$\sigma = \sqrt{E[(X-\mu^2]}$$

- Variance(σ^2):

$$\sigma^2 = E[(X-\mu^2)]$$

Unbiased estimators:-

- We say that $\hat{\theta}(S_x)$ is an unbiased estimator of some statistic θ calculated using a sample S_x drawn from an unknown probability distribution if $\hat{\theta}(S_x)$ has the following property:

$$E[\hat{\theta}(S_x)] = \theta.$$

where θ is a sample statistic, and not the real θ that can be obtained only knowing f_X.

Baye's Rule:

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$$

Parameters vs. Hyperparameters:-

- Hyperparameter:

- usually a numerical value that influences how the algorithm works.

- They are set by the data analyst, and not by the algorithm.

- Parameters:-

- Variables that define the model learned by the learning algorithm.

- Parameters are directly modified by the learning algorithm based on the training data.

✱ Classification vs. Regression:-

1- Classification:-

- A problem of automatically assigning a label to an unlabeled example.

- The classification problem is solved by a classification learning algorithm that takes a collection of labeled examples as inputs and produces a model that can take unlabeled examples as inputs and directly give them labels.

2- Regression:-

- A problem of predicting a real-valued label given an unlabeled example

- The regression problem is solved by a regression learning algorithm that takes a collection of labeled examples as inputs and produces a model that can take an unlabeled example as input and output a target.

✱ Model based vs. Instance based learning:

1- Model based:-

- Most supervised learning algorithms are model based.

- SVM uses the training data to create a model that has parameters learned from the training data

2- Instance based:-

- Instance-based learning algorithms use the whole dataset as the model.

- eg. K-Nearest ~~Neg~~ Neighbors (KNN).

In classification, to predict a ~~label~~ label the KNN looks at the close neighborhood of the input example in the space of feature vectors and outputs a ~~label~~ label that it saw the most often in this close neighborhood.

✱ Shallow vs. Deep learning :-

1- Shallow :-

- A shallow learning algorithm learns the parameters of the model directly from the features of the training examples

- Most supervised-learning algorithms are shallow.

2- Deep learning :-

- Most model parameters are learned from the outputs of the preceeding layers.

★ Ch.3. Fundamental Algorithms:-

1st algorithm [★ Linear Regression:-

- It's a popular regression learning algorithm that learns a model which is a linear combinations of feature ★ of the input examples.

★ Problem statement :-

- We have a collection of labeled examples $\{(x_i, y_i)\}_{i=1}^{N}$, where N is the size of the collection, x_i is the D-dimensional feature vector of example $i = 1, \ldots, N$, y_i is a real-valued target and every feature $x_i^{(j)}$, $j = 1, \ldots, D$, is also a real number.

- We want to build a model $f_{w,b}(x)$ as a linear combination of feature ★ of example x:

$$f_{w,b}(x) = wx + b$$

where "w" is a D-dimensional vector of parameters and "b" is a real number.

- $f_{w,b}$ means that model f is parametrized by "w" and "b".

*Continued problem statement :-

- We will use the model to predict the unknown y for a given x like this.

$$y \leftarrow f_{w,b}(x)$$

- Also, we want to find the optimal values (w^*, b^*). The optimal values of parameters define the model that makes the most accurate predictions.

- Unlike SVM, the hyperplane in linear regression is chosen to be as close to all training examples as possible.

- We can use the regression line to predict the value of the target y_{new} from a new unlabeled example x_{new}. If our examples are D-dimensional feature vectors for (D > 1), the only difference with one-dimensional case is that the regression model is not a line, but a plane or a hyperplane for (D > 2).

Solution:-

- We need to optimize so that we can find w^* and b^*. ~~The optimization formula:-~~

"we try to minimize it" $\Rightarrow$ $\frac{1}{N}\sum_{i=1}^{N}\left(f_{w,b}(x_i) - y_i\right)^2$. $\Leftarrow$ The cost function

- Objective function: the expression we minimize or maximize.

- The expression $(f_{w,b}(x_i) - y_i)^2$ is called the loss function.

- Empirical risk: average loss

- We should try to minimize the Cost function

- The empirical risk, for a model, is the average of all penalties obtained by applying the model to the training data.

- Why use a linear model?
  It's simple and rarely overfits

- Overfitting: the property of a model such that the model predicts very well labels of the training examples, but makes errors on new examples.

★ Continued solution —

- Why do we care about the derivative of the average loss?

If we can calculate the gradient of the function, we can then set this gradient to zero and find the solution to a system of equations that gives us the optimal values for w^* and b^*.

2nd algorithm ★ Logistic Regression:-

- Logistic Regression is not a regression, but a classification learning algorithm. It's usually used in binary classification.

★ Problem Statement:-

- In logistic regression, we want to model y_i as a linear function of x_i, however, if we just use linear combination features such as $wx_i + b$. If won't work. We need a function whose domain is $(0, 1)$. This is where the sigmoid function comes to play:

$$\text{Sigmoid function} = \frac{1}{1 + e^{x}}$$

✱ Continued problem statement :-

- The logistic regression model :

$$f_{w,b}(x) = \frac{1}{1+e^{-(wx+b)}}$$

- If we optimize the values of w and b, we could interpret the output of f(x) as the probability of y_i being +ve

✱ How do we find w^* and b^* ?

- In logistic regression, we maximize the likelihood of our training data. The likelihood function defines how likely the observation is according to our model.

- The optimization criterion in logistic regression is called the maximum likelihood.

max likelihood $\Rightarrow L_{w,b} = \prod f_{w,b}(x_i)^{y_i} (1-f_{w,b}(x_i)^{(1-}$

* Continued optimization...

- Because of the exp function used in the model, to avoid numerical overflow, it's more convenient to maximize the log-likelihood instead of the likelihood.

$$\text{Log-likelihood} = \sum_{i=1}^{N} \left[y_i \ln f_{w,b}(x) + (1-y_i) \ln \left(1 - f_{w,b}(x\right)\right.$$

- Because ln is a strictly increasing function, maximizing this function is the same as maximizing it's argument, and the solution to this new optimization problem is the same as the solution to the original problem

3rd algorithm: ★ Decision Tree learning:-

- A dicision tree is an acylic graph that can be used to make decisions.
- The leaf node is the decision.

★ Problem statement:-
- a collection of labeled examples; labels belong to the set {0, 1}.

★ ID3 algorithm:-

- It's a type of decision tree.
- The optimization criterion is the average log-likelihood:

$$\frac{1}{N}\sum_{i=1}^{N}\left[y_i \ln f_{ID3}(x_i) + (1-y_i)\ln\left(1-f_{ID3}(x_i\right)\right.$$

- Contrary to logistic regression which builds a parametric model $f_{w,b}$ by finding an optimal solution. The ID3 optimizes it approximately by constructing a nonparametric model

$$f_{ID3}(x) = P(y=1 \mid x)$$

⇒

Continued ID3 ....

- The ID3 works as follows, let S denote the set of labeled examples. In the beginning, the decision tree only has a start node that contains all the examples: $S = \{(x_i, y_i)\}_{i=1}^{N}$.

- Start with a constant model f_{ID3}^{S} defined as:

$$f_{ID3}^{S} = \frac{1}{|S|} \sum_{(x,y) \in S} y$$

- Then we search through all features $j = 1, \ldots, D$ and all thresholds t, and split the set S into two subsets:

$$S_- = \{(x, y) \mid (x, y) \in S,\ x^j < t\}$$
$$S_+ = \{(x, y) \mid (x, y) \in S,\ x^j \geq t\}$$

- These two new subsets would go to two leaf nodes, and we evaluate, for all possible pairs (j, t), how good the split with pieces S_- and S_+ is.

*Continued ID3 :-

- Finally, we pick the best such values (j, t), split into S− and S+, form two new leaf nodes, and continue recursively on S+ and S−.

- Entropy : measure of uncertainty (randomness) about a random variable.

- Entropy is used to determine how good the split was for the tree.

- Entropy of a set is given by :

$$H(S) = f^S_{ID3} \ln f^S_{ID3} - (1 - f^S_{ID3}) \ln (1 - f^S_{ID3})$$

- The entropy of a split is the weighted sum of 2 entropies :

$$H(S-, S+) = \frac{|S-|}{|S|} H(S-) + \frac{|S+|}{|S|} H(S+)$$

*Continued ID3...:-

- The algorithm stops at a leaf node in any of the below situations:

1- All examples in the leaf node are classified correctly by the one-piece model.

2- We can't find an attribute to split upon

3- The split reduces the entropy less than some ϵ

4- We reach the maximum depth.

4th algorithm: ★ Support Vector Machine:-

- SVM wants to satisfy these constraints:-

$$wx_i - b \geq +1 \quad \text{if } y_i = +1$$
$$wx_i - b \leq -1 \quad \text{if } y_i = -1$$

- We also want to minimize $\|w\|$ so that the hyperplane is equally distant from the closest example of each class.

- Optimization problem for SVM:-

$$\min \frac{1}{2}\|w\|^2, \text{ such that } y_i(wx_i - b) - 1 \geq 0$$

★ Dealing with noise:

- To extend SVM to cases in which the data is not linearly separable, we introduce the hinge loss function:

$$\max(0, 1 - y_i(wx_i - b))$$

- The hinge loss is zero if the constraints are satisfied.

★ Continued SVM...

- We then want to minimize this cost function:

$$C\|w\|^2 + \frac{1}{N}\sum_{i=1}^{K}\max(0, 1 - y_i(wx_i - b))$$

- Where C is a hyperparameter that determines the tradeoff between increasing the size of the decision boundary and ensuring that each x_i lies on the correct side of the decision boundary

- SVM that optimize hinge loss are called soft-margin SVMs, while the original formulation is called hard-margin SVMs.

★ Dealing with inherent non-linearity:-

- Kernel trick: a function to implicitly transform the original space into a higher dimensional space during the cost function optimization.

Continued non-linearity...:-

- Kernel functions: ~~sort~~ efficiently work in higher-dimensional space without doing this transformation explicitly.

- Optimization problem for w and b:-

$$\max_{\alpha_1 \ldots \alpha_N} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} y_i \alpha_i (x_i x_k)$$

$$\text{subject to } \sum_{i=1}^{N} \alpha_i y_i = 0 \text{ and } \alpha_i \geq 0$$

- α_i are called Lagrange multipliers.

- This optimization problem becomes a convex quadratic optimization problem, solvable by quadratic programming algorithms.

- By using the kernel trick, we can get rid of a costly transformation of original feature vector into higher-dimensional vectors and avoid computing their dot-product. We replace that by a simple operation on the original feature vectors that gives the same result.

Continued---

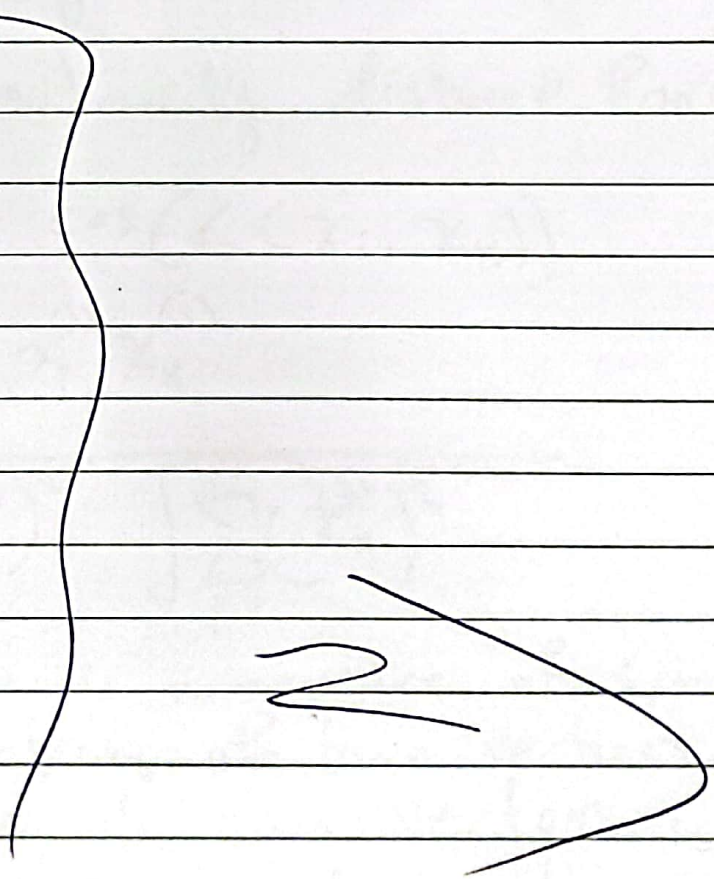
- RBF Kernel:

$$k(x, x') = \exp\left(\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

- Where $\|x - x'\|^2$ is the squared Euclidean distance.

- Euclidean distance equation:

$$d(x_i, x_k) = \sqrt{\sum_{j=1}^{D} \left(x_i^{(j)} - x_k^{(j)}\right)^2}$$



5th algorithm: K-Nearest Neighbors:-

- It's a non-parametric algorithm.
- KNN keeps all training examples in memory
- Once a new, previously unseen example x comes in, the KNN algorithm finds K training examples closest to x and returns the majority label
- The closeness of two examples is given by a distance formula.
- Cosine similarity distance formula:

$$s(x_i, x_k) = \cos(\angle(x_i, x_k))$$

$$= \frac{\sum_{j=1}^{D} x_i^{(j)} x_k^{(j)}}{\sqrt{\sum_{j=1}^{D} \left(x_i^{(j)}\right)^2} \sqrt{\sum_{j=1}^{D} \left(x_k^{(j)}\right)^2}}$$

- This equation is a measure of similarity of the directions of two vectors. If the angle between the two vectors is 0, then the two vectors point to the same direction, and the cosine similarity equals to 1.

* Continued Cosine... :

- If the vectors are orthogonal, the cosine similarity is 0.

- For vectors pointing in opposite directions, the cosine similarity is -1.

- If we want to use cosine similarity as a distance metric, we need to multiply it by -1.

Chapter 4: Anatomy of a Learning Algorithm.

- Building blocks of a learning algorithm:
  1- A loss function
  2- An optimization criterion based on the loss function
  3- An optimization routine leveraging training data to find as solution to the optimization criterion
- Gradient descent or scholastic gradient descent:
  These two are optimization algorithms used in cases where the optimization criterion is differentiable.
  Gradient descent is an iterative optimization algorithm for finding the minimum of a function. To find a local minimum of a function using gradient descent, one starts at some random point and takes steps proportional to the negative of the gradient of the function at the current point.
  Gradient descent can be used to find optimal parameters for linear and logistic regression and SVM.
  For many models, such as logistic regression or SVM, the optimization criterion is convex. Convex functions have only one minimum, which is global.
- Gradient descent:
- For linear regression "the example used in the book":
  The optimization criterion will have two parameters: w and b. The extension to multi-dimensional training data is straightforward: you have variables w^1 w^2 and b for two dimensional data.
  The training examples are in the form (x_i, y_i)
  We look for values of w and b that minimize the mean squared error:

$$l = \frac{1}{N}\sum_{i=1}^{N}(y_i - (wx_i + b))^2$$

  Gradient descent starts with calculating the partial derivative for every parameter:

$$\frac{\partial l}{\partial w} = \frac{1}{N}\sum_{i=1}^{N} -2x_i(y_i - (wx_i + b))$$

$$\frac{\partial l}{\partial b} = \frac{1}{N}\sum_{i=1}^{N} -2(y_i - (wx_i + b))$$

  Gradient descent proceeds in epochs. An epoch consists of using the training set entirely to update each parameter. In the beginning, the first epoch, we initialize the values and find the partial derivatives. At each epoch, we update w and b using the partial derivatives. The learning rate α controls the size of an update.

$$w \leftarrow w - \alpha\frac{\partial l}{\partial w}$$

$$b \leftarrow b - \alpha\frac{\partial l}{\partial b}$$

  We subtract partial derivatives from the values of parameters because derivatives are indicators of the growth of a function. If a derivative is positive at some point, then the function grows at this point. Because we want to minimize the objective function, when the derivative is positive we know that we need to move our parameter in the opposite direction. When the derivative is negative, we need to move our parameter to the right to decrease the value of the function even more

At the next epoch, we recalculate partial derivatives with the updated values of w and b; we continue the proves until convergence. Typically, we need many epochs until we start seeing that the values of w and b don't change much after each epoch; then we stop.
Read the python code in chapter 4
Gradient descent is sensitive to the choice of learning rate. It is also slow for large datasets.

Minibatch stochastic gradient descent (minibatch SGD): is a version of the algorithm that speeds up the computation by approximating the gradient using smaller batches of the training data.
Adagrad: is a version of SGD that scales α for each parameter according to the history of gradients.
Momentum: is a method that helps accelerate SGD by orienting the gradient descent in the relevant direction and reducing oscillations.
In neural network training, variants of SGD such as RMSprop and Adam, are frequently used.

- How machine learning engineers work:
- They use libraries, most of which are open source. A library is a collection of algorithms and supporting tools implemented with stability and efficiency in mind.
- The most frequently used open-source machine learning library is scikit-learn.
- Learning algorithms' particularities:
- Some algorithms, like decision tree learning, can accept categorical features. SVM, logistic, and linear regression, as well as kNN, all expect numerical values for all features
- Some algorithms, like SVM, allow the data analyst to provide weightings for each class. These weightings influence how the decision boundary is drawn. If the weight of some class is high, the learning algorithm tries to not make errors in predicting training examples of this class
- Some classification models, like SVM and kNN, given a feature vector only output the class. Others, like logistic regression or decision trees, can also return the score between 0 and 1 which can be interpreted as either how confident the model is about the prediction or as the probability that the unput example belongs to a certain class
- Some classification algorithms build the model using the whole dataset at once. If you have got additional labelled examples, you have to rebuild the model from scratch. Other algorithms (such as Naïve Bayes, multilayer perceptron,…) can be trained iteratively, one batch at a time. Once new training examples are available, you can update the model using only the new data
- Finally, some algorithms, like decision tree learning, SVM, and kNN can be used for both classification and regression, while others can only solve one type of problem: either classification or regression, but not both.

Chapter 5: Basic Practice:

- Challenges that must be addressed before making the model:
  1- Feature engineering:
    - Remember that the dataset is a collection of labeled examples.
    - The problem of transforming raw data into a dataset is called feature engineering.
    - For most practical problems, feature engineering is a labor-intensive process that demands from the data analyst a lot of creativity and, preferably, domain knowledge.
    - Everything measurable can be used as a feature. The role of the data analyst is to create informative features: those would allow the learning algorithm to build a model that does a good job of predicting labels of the data used for training
    - Highly informative features are also called features with high predictive power
    - We say that a model has a low bias when it predicts the training data well. That is, the model makes few mistakes when we use it to predict labels of the exampled used to build the model
  2- One-Hot Encoding:
    - Some algorithms only work with numerical feature vectors. When some features in your dataset is categorical, you can transform such a categorical feature into several binary ones.
    - For example: if your features are colors and there are red, green, and yellow. You can transform them into a vector of three numerical values:

      Red = [1, 0, 0]
      Yellow = [0, 1, 0]
      Green = [0, 0, 1]

    - By doing that you increase the dimensionality of your vectors. You should also avoid transforming them into red = 1 and yellow = 2, ..etc because that would imply that there is an order among the values. This could also confuse the learning algorithm because the algorithm will try to find a regularity where there's no one, which may potentially lead to overfitting.
  3- Binning:
    - This situation occurs less frequently in practice, when you have a numerical feature but you want to convert it into a categorical one. Binning (also called bucketing) is the process of converting a continuous feature into multiple binary features called bins or buckets, typically based on value range.
    - In some cases, a carefully designed binning can help the learning algorithm to learn using fewer examples. It happens because we give a hint to the learning algorithm that if the value of a feature falls within a specific range, the exact value of the feature doesn't matter.
  4- Normalization:
    - Normalization is the process of converting an actual range of values which a numerical feature can take, into a standard range of values, typically in the interval [-1, 1] or [0, 1]
    - The normalization formula:

$$\overline{x}^j = \frac{x^j - min^j}{max^j - min^j}$$

- Where min^j and max^j are, respectively, the minimum and the maximum value of the feature j in the dataset.
- Why do we normalize?

  Normalizing the data is not a strict requirement. However, in practice, it can lead to an increased speed of learning.
- Numerical overflow: problems that computers have when working with very small or very big numbers.

5- Standardization:

- Standardization (or z-score normalization) is the procedure during which the feature values are rescaled so that they have the properties of a standard normal distribution with μ = 0 (the mean) and σ = 1 (standard deviation)
- Standard scores (or z-scores) of features are calculated as follows:

$$\hat{x}^j = \frac{x^j - \mu^j}{\sigma^j}$$

- When to use normalization and when to use standardization?

  There is no definitive answer, you should experiment and try. However, if you don't have time to experiment as a rule of thumb:

  1- Unsupervised learning algorithms, in practice, more often benefit from standardization than from normalization
  2- Standardization is also preferred for a feature if the values this feature takes are distributed close to a normal distribution (so-called bell curve)
  3- Again, standardization is preferred for a feature if it can sometimes have extremely high into a very small range
  4- In all other cases, normalization is preferable

- Dealing with missing features:
  - In some cases, the data comes to the analyst in the form of a dataset with features already defined. The values of some features could be missing. And there are some approaches to take in dealing with missing features:

    1- Removing the examples with missing features from the dataset (that can be done if your dataset is big enough so you can sacrifice some training examples)
    2- Using a learning algorithm that can deal with missing feature values (depends on the library and a specific implementation of the algorithm)
    3- Using a data imputation technique
- Data imputation techniques:
  - One data imputation technique consist in replacing the missing value of a feature by an average value of this feature in the dataset:

$$\hat{x}^j \leftarrow \frac{1}{M}\sum_{i=1}^{N} x_i^j$$

  - Where M < N is the number of examples in which the value of the feature j is present, while the summation excludes the examples in which the value of J is absent
  - Another technique is to replace the missing value with a value outside the normal range. If the range is [0, 1] you can se the value to 2. Alternatively, you can replace the missing value by a value in the middle of the range. If the range is [-1, 1] you can

set the value to 0. Here, the idea is that the value in the middle of the range will not significantly affect the prediction

- A more advanced technique is to use the missing values as the target variable for a regression problem. You can use all the remaining features to form a feature vector $\hat{x}_i, set\ \hat{y}_i \leftarrow x^j$, where j is the feature with a missing value. Then you build a regression model to predict $\hat{y}\ from\ \hat{x}$. You will only use those examples from the original dataset, in which value of feature j is present.
- Finally, if you have a significantly large dataset and just a few features with missing values, you can increase the dimensionality of your feature vectors by adding a binary indicator feature for each feature with a missing value. The missing feature value then can be replaced by 0 or any number of your choice
- At prediction time, if your example is not complete, you should use the same data imputation technique to fill the missing feature as the technique you used to complete the training data.

- Learning algorithm selection:

  (Things in bold are important here)

  - Choosing a machine learning algorithm can be a difficult task. Some things to consider when choosing a learning algorithm:
    1- Explainability: **does your model have to be explained to a non-technical audience?** Most very accurate learning algorithms are so-called "black boxes." They learn models that make very few errors, but why a model made a specific prediction could be very hard to understand and even harder to explain.
    2- In-memory vs. out-memory: **Can your dataset be fully loaded into the ram of your computer?** If yes, then you can choose from a wide variety of algorithms. **Otherwise, you want incremental learning algorithms that can improve the model by adding more data gradually**
    3- Number of feature examples: **How many training examples do you have in your dataset?** Some algorithms, including neural networks and gradient boosting, can handle a huge number of examples. Others like SVM, can be very modest in their capacity.
    4- Categorical vs. Numerical features: **Some algorithms cannot handle your dataset directly, and you would need to convert your categorical features into numerical ones**
    5- Nonlinearity of the data: **is your data linearly separable or can it be modeled using a linear model?** If yes, then SVM with the linear kernel, logistic or linear regression can be good choices. Otherwise, deep neural networks or ensemble algorithms might work better
    6- Training speed: **how fast does the model have to be when generating predictions?** Algorithms like SVM, linear and logistic regression are extremely fast at predictions, while others, like kNN, and ensemble algorithms are slower

- Three sets:
    - In practice, we work with three distinct sets of labeled examples:
        1- Training set
        2- Validation set
        3- Test set
    - Once you have got you annotated dataset, the first thing you do I shuffle the examples and split the dataset into three subsets: training, validation, and test. The training set is usually the biggest one; the validation and test sets are roughly the same sizes, much smaller than the size of the training set. The learning algorithm can't use examples from these two subsets to build the model. That is why those two sets are called holdout sets
    - There is no optimal proportion to split the dataset into these three subsets. You usually want the highest percentage to be on the training set and evenly distribute the rest to validation and testing
    - Why do we need two holdout sets and not one?

        We use the validation set to:

        1- Choose the learning algorithm
        2- Find the best values of hyperparameters

        We use the test set to assess the model before delivering it to the client or putting it in production

- Underfitting and overfitting:
    - If the model makes mistakes on the training data, we say that the model has a high bias or that the model underfits
    - Underfitting is the inability of the model to predict well the labels of the data it was trained on.
    - Reasons for underfitting:
        1- Your model is too simple for the data
        2- The features you engineered are not informative enough
    - The solution of underfitting is to try a more model or to engineer features with higher predictive power
    - Overfitting: when the model predicts very well the training data but poorly the data from at least one of the two holdout sets
    - Reasons for overfitting:
        1- Your model is too complex for the data
        2- You have too many features but a small number of training examples
    - Another name for the problem of overfitting, is the problem of high variance
    - Solutions to the problem of overfitting:
        1- Try a simpler model
        2- Reduce the dimensionality of examples in the dataset
        3- Add more training data
        4- Regularize the model

- Regularization:
  - Regularization is an umbrella term that encompasses methods that force the learning algorithm to build a less complex model
  - The two most widely used types of regularization are called L1 and L2 regularization.
  - To create a regularized model, we modify the objective function by adding a penalizing term whose value is higher when the model is more complex
  - An L1-Regularized objective looks like this (Linear regression):

$$min_{w,b}[C|w| + \frac{1}{N}\sum_{i=1}^{N}(f_{w,b}(x_i - y_i)^2$$

    Where |w| equals the summations of all w and C is a hyperparameter that controls the importance of regularization. If we set C to zero, the model becomes a standard non-regularized linear regression model. On the other hand, if we set C to high value, the learning algorithm can underfit. We need to find such a value of the hyperparameter C that doesn't increase the bias too much but reduces the variance to a level reasonable for the problem at hand.
  - An L2-regularized objective looks like this (linear regression):

$$min_{w,b}[C||w||^2 + \frac{1}{N}\sum_{i=1}^{N}(f_{w,b}(x_i - y_i)^2$$

  - L1 produces a sparse model, a model that has most of its parameters equal to zero, provided the hyperparameter C is large enough. So L1 performs feature selection by deciding which features are essential for prediction and which are not.
  - If your only goal is to maximize the performance of the model on the holdout data, then L2 usually gives better results. L2 also has the advantage of being differentiable, so gradient descent can be used for optimizing the objective function
  - L1 and L2 regularization methods were also combined in what is called elastic net regularization
  - The name for L2 is ridge regularization for L2 and lasso for L1.
  - L1 and L2 regularization are also frequently used with neural networks and many other types of models, which directly minimize an objective function (they are also widely used in linear models)
  - Neural networks benefit from two other regularization techniques: dropout and batch normalization.

- Model performance assessment:
    - Model performance assessment can tell you how good the model is.
    - If our model performs well on predicting the labels of the examples from the test set, we say that our model generalizes well or, simply, that it's good.
    - Machine learning specialists use various formal metrics and tools to assess the model performance
    - Model assessment for regression problems:

        The mean model, which always predicts the average of the labels in the training data, generally would be used if there were no informative features. The fit of a regression model should be better than the fit of the mean model. To do that, we compute the mean squared error (MSE) for both training and test data separately. If the MSE of the model is substantially higher than the MSE obtained on the training data, this is a sign of overfitting. Regularization or a better hyperparameter tuning could solve the problem.
    - Model assessment for classification:
    - Confusion matrix
    - Accuracy
    - Cost-sensitive accuracy
    - Precision/recall
    - Area under the ROC curve
- Confusion matrix:
    - Confusion matrix is a table that summarizes how successful the classification model is at predicting examples belonging to various classes.
    - True positives (TP): the actual positive values from the dataset
    - True negatives (TN): the actual negative values from the dataset
    - False positives (FP): the predicted positive values from the model
    - False negatives (FN): the predicted negative values from the model
    - The confusion matrix basically has rows and columns as there are different classes, and it will calculate the amount of TP, TN, FP, and FN from our model.
    - Confusion matrix is used to calculate two other performance metrics: precision and recall.
- Precision/Recall:
    - The two most frequent used metrics to assess the model are precision and recall.
    - Precision is the ratio of correct positive predictions to the overall number of positive predictions
    - Precision formula:

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

    - Recall is the ratio of correct positive predictions to the overall number of positive examples in the dataset
    - Recall formula:

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

    - We must choose between high precision or a high recall. It's usually impossible to have both. We can achieve either of the two by various means:
        1- Assigning a higher weighting to the examples of a specific class
        2- Tuning hyperparameters to maximize precision or recall on the validation set

3- Varying the decision threshold for algorithms that return probabilities of classes

- Accuracy:
  - Accuracy is given by the number of correctly classified examples divided by the total number of classified examples
  - Accuracy formula:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

  - Accuracy is a useful metric when errors in predicting all classes are equally important
- Cost-sensitive accuracy:
  - For dealing with the situation in which different classes have different importance, a useful metric is cost-sensitive accuracy. To compute a cost-sensitive accuracy, you first assign a cost to both types of mistakes: FP and FN. You then compute the counts TP, TN, FP, FN as usual and multiply the counts for FN and FN by the corresponding cost before calculating the accuracy with its formula.
- Area under the ROC curve (AUC):
  - The ROC curve is a commonly used method to assess the performance of classification models. ROC curves use a combination of the true positive rate and the false positive rate to build up a summary picture of the classification performance.
  - True positive rate (TPR) and false positive rate (FPR):

$$TPR = \frac{TP}{TP + FN} \; and \; FPR = \frac{FP}{FP + TN}$$

  - ROC curves can only be used to assess classifiers that return some confidence score or prediction such as logistic regression, neural networks, and decision trees.
  - How to draw a ROC curve?

    You first discretize the range of the confidence score. If it's 0 and 1 you can have it form 0.0 all the way to 1.0

    Then, you use each discrete value as a prediction threshold and predict the labels of examples in your dataset using the model and this threshold
  - The higher the AUC, the better the classifier. A classifier with an AUC higher than 0.5 is better than a random classifier. If AUC is lower than 0.5 then something is wrong with your model. A perfect classifier would have AUC of 1
  - ROC curves are popular because they are relatively simple to understand, and they capture more than one aspect of the classification
- Hyperparameter tuning:
  - Hyperparameters aren't optimized by the learning algorithm itself. The data analyst has to "tune" hyperparameters by experimentally finding the best combination of values, one per hyperparameter
  - One typical way of doing that is to use a grid search.
  - Grid search is the most simple hyperparameter tuning technique. It works by scaling the hyperparameters and find different combinations for it, then trying all combinations and assess their performance using one of the metrics (accuracy and whatnot). Once the best pair of hyperparameters is found, you can try to explore the values close to the best ones in some region around them.
  - There are more efficient techniques, such as random search and Bayesian hyperparameter optimization.
  - Random search differs from grid search in that you no longer provide a discrete set of values to explore for each hyperparameter. Instead, you provide a statistical

distribution for each hyperparameter from which values are randomly samples and set the total number of combinations you want to try.

- Bayesian technique differs from random or grid search in that they use past evaluations results to choose the next values to evaluate. The idea is to limit the number of expensive optimizations of the objective function by choosing the next hyperparameter values based on those that have done well in the past.
- There are also gradient-based techniques, evolutionary optimization techniques, and other algorithmic hyperparameter tuning techniques.
- There are also hyperparameter tuning libraries.

- Cross-validation:
  - When you have few training examples, it could be prohibitive to have both validation and test set. You would prefer to use more data to train the model. In such a case, you only split your data into a training and test set. Then you use cross-validation on the training set to simulate a validation set.
  - How does cross-validation work?

    First, you fix the values of the hyperparameters you want to evaluate. Then you split your training set into several subsets of the same size. Each subset is called a fold. Typically, five-fold cross-validation is used. The five folds are {F1, F2, …, F5}. Each Fk, k = 1 contains 20% of your training data. Then you train five models as follows:

    To train the first model, f1, you use all examples from F2, F3, F4, and F5 as the training set and the examples from F1 as the validation set. Then, you continue building models iteratively like this and compute the value of the metric of interest on each validation set, from F1 to F5. Then you average the five values of the metric to get the final value
  - You can use grid search with cross-validation to find the best values of hyperparameters for your model.

Chapter 6: Neural Networks and Deep Learning:

- Neural networks:

- A neural network, just like regression or an SVM model, is a mathematical function:

$$y = f_{NN}(x)$$

- The function fNN is a nested function.
- Neural networks consist of layers. A 3-layer neural network that returns a scalar would look like this:

$$y = f_{NN}(x) = f_3(f_2(f_1(x)))$$

- In the above equation, f1 and f2 are vector functions of the following form:

$$f_1(z) = g_l(W_{lz} + b_l)$$

- L is called the layer index and can span from 1 to any number of layers. The function gl is called an activation function. It is a fixed, usually nonlinear function chosen by the data analyst before the learning is started. The parameter Wl (a matrix) and bl (a vector) for each layer are learned using the familiar gradient descent by optimizing, depending on the task, a particular cost function with the equation for logistic regression, where you replace gl by the sigmoid function, and you will not see any difference.
- Why is Wl a matrix and not a vector wl?
    - The reason is gl is a vector function. Each row $w_{l,u}z + b_{l,u}$. The output of f1(z) is a vector [gl(al,1), gl(al, 2)….. gl(al, sizeL). Where size L is the number of units in layer l.
- Multilayer perception is often referred to as a vanilla neural network.

- Multilayer perceptron:

- This FN can be a regression or a classification model, depending on the activation function.
- The neural network is represented graphically as a connected combinations of units logically organized into one or more layers. Each unit is represented by either a circle or a rectangle
- The output of each unit is the result of the mathematical operation written inside the rectangle. Circle units don't do anything with the input; they just send their input directly to the output
- The following happens in each rectangle unit:
    - Firstly, all the inputs of the unit are joined together to form an input vector.
    - Then, the unit applies a linear transformation to the input vector, exactly like linear regression model does with its input feature vector.
    - Finally, the unit apples an activation function g to the result of the linear transformation and obtains the output value, a real number
- In a vanilla FFNN, the output value of a unit of some layer becomes an input value of each of the units of the subsequent layer
- In multilayer perceptron, all outputs of one layer are connected to each input of the succeeding layer. This architecture is called fully-connected.

- Feed-Forward Network Architecture:

- Feed forward is basically taking the output of each layer and using as the input of the next layer.
- If we want to solve a regression or a classification problem, the last layer of a neural network usually contains one unit. If the activation function of the last unit is linear, then the neural network is a regression model. If the activation function is a logistic function, the neural network is a binary classification model.
- We can choose any mathematical function assuming it's differentiable.
- The primary purpose of having nonlinear components in the function fNN is to allow the neural network to approximate nonlinear functions. Without nonlinearities, fNN would be

linear, no matter how many layers it has. The reason is that Wlz + bz is a linear function and a linear function of a linear function is also linear

- Popular choices for activation functions are the logistic functions TanH and ReLU

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

$$relu(z) = \begin{cases} 0 \; if \; z < 0 \\ z \; otherwise \end{cases}$$

- Wl in the expression Wlz + bl is a matrix.
- In matrix Wl, each roe u corresponds to a vector of parameters $w_{l,u}$. The dimensionality of the vector equals to the number of units in the layer l-1. The operation Wlz results in a vector al = [$w_{l,1}$ z, $w_{l,2}$ z, …… $w_{l,sizeL}$ z]. Then the sum al + bl gives a sizeL-dimensional vector cl. Finally, the function gl(cl) produces the vector yl = [$y^{(1)}$, …, .. $y^{(sizeL)}$] as output.

- Deep learning:

- Deep learning refers to training neural networks with more than two non-output layers.
- What are the biggest challenged in deep learning and neural networks?
    - Exploding gradient
    - Vanishing gradient
- Exploding gradient is dealt with by applying simple techniques like gradient clipping and L1 and L2 regularization. The problem of vanishing gradient remains intractable for decades.
- What is vanishing gradient and why does it arise?
    - To update the values of the parameters in neural networks the algorithm called backpropagation is typically used. Backpropagation is an efficient algorithm for computing gradients on neural networks using the chain rule. During gradient descent, the neural network's parameters receive an update proportional to the partial derivative of the cost function with respect to the current parameter in each iteration of training. The problem is that in some cases, the gradient will be vanishingly small, effectively preventing some parameters from changing their value. In the worst case, this may completely stop the neural network from further training.

- Convolutional Neural Network:

- A convolutional neural network (CNN) is a special kind of FFNN that significantly reduces the number of parameters in a deep neural network with many units without losing too much in the quality of the model
- CNNs have found applications in image and text processing where they beat may previously established benchmarks
- In images, the pixels that are close to one another usually represent the same type of information. The exception from the rule are the edges
- If we can train the neural network to recognize regions of the same information as well as the edges, this knowledge would allow the neural network to predict the object represented in the image
- Most of the information in the image is local, we can split the image into square patches using a moving window approach. We can then train multiple smaller regression models at once, each small regression model receiving a square patch as input. The goal of each small regression model is to learn to detect a specific kind of pattern in the input patch
- In CNNs, a small regression model has only one layer. To detect some pattern, a small regression model has to learn the parameters of a matrix F (for "filter") of size pxp, where p is the size of the patch.

- The convolution operator is only defined for matrices that have the same number of rows and columns
- There's also a bias parameter b associated with each filter F which is added to the result of a convolution before applying the activation function
- One layer of CNN consists of multiple convolution filters (each with its own bias parameter) just like one layer in a vanilla FFNN consist of multiple units. Each filter of the first layer slides (convolves) across the input image, left to right, top to bottom, and convolution is computed at each iteration
- The filter matrix and bias values are trainable parameters that are optimized using gradient descent with backpropagation
- A nonlinearity is applied to the sum of the convolution and the bias term. Typically, the ReLU activation function is used in all hidden layers. The activation function of the output layer depends on the task
- Since we can have sizeL filters in each layer l, the output of the convolution layer l would consist of sizeL matrices, one for each filter
- If the CNN has one convolution layer following another convolution layer, then the subsequent layer l+1 treats the output of the preceding layer l as a collection of sizeL image matrices. Such a collection is called a volume. The size of that collection is called the volume's depth. Each filter of layer l+1 convolves the whole volume. The convolution of a patch of a volume is simple the sum of convolutions of the corresponding patches of individual matrices the volume consists of
- In computer vision, CNNs often get volumes as input, since an image is usually represented by three channels: R, G, and B, each channel being a monochrome picture.
- Two important properties of convolution are stride and padding.
- Stride is the step size of the moving window. The output matrix is smaller if the stride is bigger
- Padding allows getting a larger output matrix; it's width of the square of additional cells with which you surround the image (or volume) before you convolve it with the filter
- Pooling is a technique used very often in CNNs.
- Pooling works in a way very similar to convolution, as a filter applied using a moving window approach. However, instead of applying a trainable filter to an unput matrix or a volume, pooling layer applies a fixed operator, usually either max or average. Similarly to convolution, pooling has hyperparameters: the size of the filter and stride
- Usually, a pooling layer follows a convolution layer, and it gets the output of convolution as input. When pooling is applied to a volume, each matrix in the volume is processed independently of others. Therefore, the output of the pooling layer applied to a volume is a volume of the same depth as the input
- Pooling only has hyperparameters and doesn't have parameters to learn
- Typically, pooling contributes to the increased accuracy of the model. It also improves the speed of training by reducing the number of parameters of the neural network.

- Recurrent Neural Network:

- Recurrent neural networks (RNNs) are used to label, classify, or generate sequences. A sequence is a matrix, each row of which is a feature vector, and the order of rows matters.
- To classify a sequence is to predict a class for the entire sequence. To generate a sequence is to output another sequence (of a possibly different length) somehow relevant to the input sequence.
- RNNs are often used in text processing because sentences and texts are naturally sequences of either words/punctuation marks or sequences of characters. For the same reason, recurrent neural networks are also used in speech processing.
- A recurrent neural network is not feed-forward: it contains loops. The idea is that each unit u or recurrent layer l has a real-valued state $h_{l,u}$. The state can be seen as the memory of the unit. In RNN, each unit u in each layer l receives two inputs: a vector of states from the previous layer l-1 and the vector of states from this same layer l from the previous time step.
- The first layer in an RNN receives a feature vector as input. The second layer receives the output of the first layer as input.
- Each training example is a matrix in which each row is a feature vector. For simplicity, let's illustrate this matrix as a sequence of vectors X, where length is the length of the input sequence. If our input example X is a text sentence, then feature vector x^t for each t = 1,.., length represents a word in the sentence at position t.
- In an RNN, the feature vectors from an input example are "red" by the neural network sequentially in the order of the timesteps. The index t denotes a timestep. To update the state, at each timestep t in each unit u of each layer l, we first calculate a linear combination of the input feature vector with the state vector of this same layer from the previous timestep. The linear combination of two vectors is calculated using two parameter vector $w_{l,u}$, $u_{l,u}$ and a particular parameter $b_{l,u}$. The value of the state is then obtained by applying an activation function g1 to the result of the linear combination. A typical choice for g1 is tanh. The output is typically a vector calculated for the whole layer l at once. To obtain the output, we use activation function g2 that takes a vector as input and returns a different vector of the same dimensionality. The function g2 is applied to a linear combination of the state vector values calculated using a parameter matrix Vl and a parameter vector cl. In classification, a typical choice for g2 is the softmax function:

$$\sigma(z) = [\sigma^1, \dots, \sigma^D], where\ \sigma^j = \frac{\exp(z^j)}{\sum_{k=1}^{D} \exp(z^k)}$$

- The softmax function is a generalization of the sigmoid function to multidimensional outputs
- The dimensionality of Vl is chosen by the data analyst such that the multiplication of matrix Vl by the vector h_l^t results in a vector of the same dimensionality as that for the vector cl
- The values of $w_{l,u}$, $u_{l,u}$, $V_{l,u}$, and c_l are computed from the training data using gradient descent with backpropagation. To train RNN models, a special version of backpropagation is used called backpropagation through time
- Both tanh and softmax suffer from the vanishing gradient problem
- Another problem RNNs have is that handling of long-term dependencies. As the length of the input sequence grows, the feature vectors from the beginning of the sequence tend to be "forgotten," because the state of each unit, which serves as network's memory, becomes significantly affected by the feature vectors read more recently
- The most effective neural network models used in practice are gated RNNs. These include the long short-term memory (LSTM) networks and networks based on the gated recurrent unit (GRU).

- The beauty of using gated units in RNNs is that such networks can store information in their units for future use, much like bits in a computer's memory. The difference with the real memory is that reading, writing, and erasure of information stored in each unit is controlled by activation functions that take values in the range (0,1).
- Units make decisions about what information to store, and when to allow reads, writes, and erasures. Those decisions are learned from data and implemented through the concept of gates. There are several architectures of gated units. A simple but effective one is called the minimal gated unit and is composed of a memory cell, and a forget gate.
- A gated unit takes in input and stores it for some time. This is equivalent to applying the identity function ($f(x) = x$) to the input.
- Other important extensions to RNNs include bi-directional RNNs, RNNs with attention and sequence-to-sequence RNN models. The latter, in particular, are frequently used to build neural machine translation models and other models for text-to-text transformations. A generalized of an RNN is a recursive neural network.