WiDS Report


# Chatbot using NLP

Winter Project in Machine Learning with a specialization in Natural Language Processing using python.

Student: Gangisetty Krishna Sai Kusal

Mentor Name: Shravya Suresh

Winter 2023

Analytics Club, IIT Bombay

# Introduction

An abstract is a summary that provides a brief account of the main con-tent of an academic paper. The purpose of the abstract is partly to generate interest, and partly to present the main issue and key results. Most importantly, the abstract should capture what the report is about. The abstract is best written when you have almost completed your project. Only then will you know what you have actually written. A good idea is to work on a draft summary alongside your paper, and revise it as you go along. The summary is a difficult text to write, as it is to cover a lot of con-tent in a small space. But, that is also why it is a useful text to work on – it forces you to formulate what your project is about. The abstract should not be long, only around 250 words

**Keywords:** Keyword 1, keyword 2, ..., keyword 3...Keyword 4, keyword 5, ..., keyword 6

# Contents

# 1. Week 01

This week, we were introduced to Jupyter notebook and some basics of ML, like linear regression, logistic regression, and working with scikit-learn and pandas in python. I have learnt some basics of statistics and probability required for learning ML.Then, I learnt linear and logistic regression. I have tried all given examples with datasets available for me.

**Resource:** Edureka's Data Science full course

## 1.1 Statistics Theory

### Basic Terminologies in Statistics

- **Population:** A collection or set of individuals or objects or events whose properties are to be analyzed.
- **Sample:** A subset of population is called 'Sample'. A well chosen sample will contain most of the information about a particular population parameter
- **Random Sampling:** Each member of the population has equal chance of being selected in the sample.
- **Systematic Sampling:** In Systematic sampling every nth record is chosen from the population to be a part of the sample.
- **Stratified Sampling:** A stratum is a subset of the population that shares at least one common characteristic, for example gender. Random sampling is used to select a sufficient number of subjects from each stratum.

### Different types of Statistics

- **Descriptive Statistics:** Descriptive statistics uses the data to provide descriptions of the population, either through numerical calculations or graphs or tables. Descriptive Statistics is mainly focused upon the main characteristics of data. It provides graphical summary of the data.
- **Inferential Statistics:** Inferential statistics makes inferences and predictions about a population based on a sample of data taken from the population in question. Inferential statistics, generalizes a large dataset and applies probability to draw a conclusion. It allows us to infer data parameters based on a statistical model using a sample data.

### Descriptive Statistics

- **Measures of Central Tendency:**
  - **Mean:** Measure of average of all the values in a sample is called Mean.
  - **Median:** Measure of the central value of the sample set is called Median.
  - **Mode:** The value most recurrent in the sample set is known as Mode.

- **Measures of Spread**
  - **Range:** Range is the given measure of how spread apart the values in a dataset are.Range is Max(X) - Min(X).
  - **Inter Quartile Range:** Quartiles tell us about the spread of a data set by breaking the data set into quarters, just like the median breaks it in half. Inter quartile range is difference between 25% quaritle and 75% quartile.
  - **Variance:** Standard Deviation is the measure of the dispersion of a set of data from its mean. $Var = \frac{\sum(X-\mu)^2}{N}$
  - **Standard Deviation:** Variance describes how much a random variable differs from its expected value. It entails computing squares of deviations. $\sigma = \sqrt{Var}$
- **Confusion Matrix:** A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. Confusion Matrix represents a tabular representation of Actual vs Predicted values.

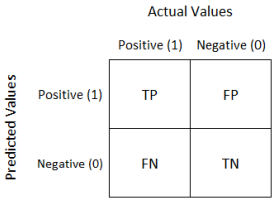


Figure 1: Confusion matrix

**Inferential Statistics**

- **Point Estimation**
- **Confidence Interval**
- **Hypothesis Testing**

## 1.2 Probability Theory

Probability is the ratio of desired outcomes to total outcomes

**Terminologies in Probability**

- **Random Experiment**: An experiment or a process for which the outcome cannot be predicted with certainty
- **Sample Space**: The entire possible set of outcomes of a random experiment is the sample space (S) of that experiment
- **Event**: One or more outcomes of an experiment. It is a subset of sample space(S)

**Probability Distribution**

- **Probability Density Function**: The equation describing a continuous probability distribution is called a Probability Density Function (PDF).
- **Normal Distribution**: The Normal Distribution is a probability distribution that associates the normal random variable X with a cumulative probability. It is also called Gaussian distribution.
- **Central Limit Theorem**: Central limit theorem is a statistical theory that states that when the large sample size has a finite variance, the samples will be normally distributed and the mean of samples will be approximately equal to the mean of the whole population.

**Types of probability**

- **Marginal Probability**: The probability of occurrence of a single event.
- **Joint Probability**: It is a measure of 2 events happening at the same time.
- **Conditional Probability**: Probability of event or outcome based on other events.

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

**Bayes Theorem**

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

## 1.3 Basics of Machine Learning

- **Supervised learning**: This type of learning involves training a model using labeled data, where the desired output is already known. The model is then tested using new input-output pairs to predict the output. Examples include linear regression, logistic regression, and decision trees.
- **Unsupervised learning**: This type of learning involves training a model using unlabeled data, where the desired output is not known. The model is then used to identify patterns or structures in the data. Examples include k-means clustering and Principal Component Analysis (PCA).
- **Reinforcement learning**: This type of learning involves training an agent to make decisions in an environment by receiving rewards or penalties for its actions. The agent learns to maximize its rewards over time. Examples include Q-learning and SARSA.
- **Overfitting**: This occurs when a model is too complex and fits the training data too closely, resulting in poor performance on new, unseen data. It can be mitigated using techniques such as regularization and early stopping.

- **Underfitting**: This occurs when a model is too simple and does not capture the underlying patterns in the data. It can be mitigated by increasing model complexity or using more data for training.
- **Bias-variance trade-off**: This concept states that as model complexity increases, the bias of the model decreases but the variance increases. Finding the right balance between bias and variance is important for good model performance.
- **Feature extraction**: This is the process of identifying and extracting the most relevant features from the data to be used in the learning process. It is an important step in preparing the data for machine learning.
- **Gradient Descent**: Gradient descent is an optimization algorithm used to minimize some functions by iteratively moving in the direction of the steepest/most rapid decrease in the function.

## 1.4 Linear Regression

- Linear regression is a supervised learning algorithm that is used to predict a continuous target variable based on one or more input variables (features).
- The goal of linear regression is to find the best-fitting line (or hyperplane in the case of multiple features) that minimizes the difference between the predicted values and the true values.
- Linear regression models the relationship between the data points by fitting a linear equation to the observed data, where the coefficients of the equation are learned from the data.
- The line of best fit is represented by the equation Y = mX + b, where Y is the dependent variable, X is the independent variable, m is the slope of the line, and b is the y-intercept.
- Linear regression can be of two types: Simple Linear Regression and Multiple Linear Regression.
- Simple Linear Regression: It is used when we want to predict a dependent variable based on a single independent variable.
- Multiple Linear Regression: It is used when we want to predict a dependent variable based on multiple independent variables.
- Linear regression assumes linearity and homoscedasticity among the variables.
- Linear regression can be used for both simple and complex datasets, and it is a powerful tool for understanding relationships between variables.
- Linear Regression can be implemented using OLS (Ordinary Least Square) method, Gradient Descent, and Normal Equation.
- Linear Regression is sensitive to Outliers, thus it needs to be handled carefully.

### 1.5 Logistic Regression

- Logistic regression is a supervised learning algorithm that is used to predict a binary target variable based on one or more input variables (features).
- Logistic regression models the probability of the default class (e.g., yes or no, true or false) as a function of the input features.
- Logistic regression uses the logistic function (also called the sigmoid function) to model the probability of the default class.
- Logistic regression models the relationship between the data points by fitting a logistic equation to the observed data, where the coefficients of the equation are learned from the data.
- Logistic regression can be of two types: Simple Logistic Regression and Multiple Logistic Regression.
- Simple Logistic Regression: It is used when we want to predict a dependent variable based on a single independent variable.
- Multiple Logistic Regression: It is used when we want to predict a dependent variable based on multiple independent variables.
- Logistic regression is a classification algorithm, and it is used for predicting binary outcomes.
- Logistic regression is sensitive to class imbalance and multicollinearity among the variables.
- Logistic Regression can be implemented using maximum likelihood estimation (MLE) method and gradient descent.
- Logistic Regression can be extended to handle multiple classes using the one-vs-all or softmax approach.
- Logistic Regression is not only used for binary classification but it can be used for multi-class classification as well with appropriate modifications.

# 2. Week 2 and Week 3

This week we tried to learn some theories regarding natural language processing. And, we also got acquainted with TensorFlow. Check tensorflow.ipynb in week 3 for the work I have done.

**Resource:** Speech and Language Processing by Daniel Jurafsky.

### 2.1 Words

- **corpus:** a computer-readable collection of text or speech.
- **utterance:** Speech with disfluencies, especially while talking. Words like uh, um, dis-disco contribute. ”uh”, ”um” and similar words are called **fillers**. Broken

words like **dis-disco** are called **fragments**. These can be removed or can be used to predict upcoming words.

- **lemma:** It is a set of lexical forms having the same stem, the same major part of speech.
- We need two ways of talking about words. **Types** are no. of distinct words in a corpus. If the set of words in the vocabulary is $V$, the no. of types is the vocabulary size $—V—$. **Tokens** are the total number $N$ of running words. Larger the corpora, the more word types we find. This is called **Herdan's law.**

  $$|V| = kN^{\beta}$$

  The value of $\beta$ ranges from 0.67 to 0.75.
- We need to identify other languages with the same algorithm too. And there will be some sentences in which there are more than 2 languages used. Some words' meanings will change with the genre. The text also represents users' demographic characteristics. And language changes with time too. So, to include all these, we must create a datasheet.
- Datasheet might include Motivation, Situation, Language variety, Speaker demographics, collection process, annotation process, and distribution.
- **Text Normalization:** At least three tasks are commonly applied as part of the annotation process: Tokenizing words, Normalizing word formats and Segmenting sentences.
- **Tokenization:**
  - There is one UNIX command for tokenizing words. It is a naive approach.

    ```
    tr -sc 'Á-Za-z' '\n' <file.txt | sort | uniq -c
    ```

    This list out all vocabulary in a sorted way, with no. of times a word is repeated. Now we can sort based on occurrences.

    ```
    tr -sc 'Á-Za-z' '\n' <file.txt | sort | uniq -c | sort -n -r
    ```

  - This naive approach is insufficient for tokenization. There are some texts which shouldn't be broken down based on punctuation marks, such as URLs.
  - A tokenizer can also be used to expand clitic contractions that are marked by apostrophes, for example, converting *what're* to the two tokens *what are*, and *we're* to *we are*.
  - It might even break multi-word expressions, such as New York.
  - There is one commonly used tokenization standard known as **Penn Treebank tokenization**. Generally, we use algorithms based on *regex* for tokenizing.

- If there are words like 'low, new, newer' in our training set and word 'lower' in testing, then it shouldn't be new for our system. So, we must tokenize them into **subwords**.
- **BytePair Encoding(BPE) Algorithm:**
- **Word normalization:** It is the task of putting words/tokens in a standard format, **Case folding** is another kind of normalization.
- **Lemmatization:** Task of determining that two words have the same root, despite their surface differences. It is done using **porter stemmer** algorithm.
- **Sentence segmentation:** It is breaking sentences based on punctuation marks. Here, the problems are words like Mr. and Inc. etc..
- There might be some spelling mistakes, and some intutive words might be missing. For them, **Edit distance** gives us a way to quantify both of these intuitions about string similarity. By minimizing it, we get the word/sentnece intended.
- **Minimum edit distance algorithm:**

```
def LD(s, t):
    if s == "":
        return len(t)
    if t == "":
        return len(s)
    if s[-1] == t[-1]:
        cost = 0
    else:
        cost = 1

    res = min([LD(s[:-1], t)+1,
               LD(s, t[:-1])+1,
               LD(s[:-1], t[:-1]) + cost])

    return res
```

This algorithm is recursive and there is one iterative argument too. This algorithm assumes that addition/deletion accounts for +1 of distance. So, substitution is +2. The above function gives us the distance between 2 words. This can be parsed in all words and find the least one.

## 2.2 N-gram Language models

- It is the theory of estimating the next word from the previous words. If we consider the previous single word, it is bigram and similarly trigram, etc...

- Actually, the probability of a total line being correct according to the dataset is

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})....P(w_n|w_{1:n-1})$$

- The intuition of the n-gram model is that instead of computing the probability of a word given its entire history, we can approximate the history by just the last few words.
- When we use a bigram model to predict the conditional probability of the next word, we are thus making the following approximation:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1})$$

  This assumption is called **Markov's Assumption**
- Similarly, N-gram is given by:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-N+1:n-1})$$

- To estimate these bigram or N-gram probabilities, we take **MLE(Maximum Likelihood Estimator)**. For example, the probability of finding a word B beside A is no. of occurrences of word B besides A divided by no. of occurrences of word A. For an N-gram parameter estimation:

$$P(w_n|w_{n-N+1:n-1}) = \frac{C(w_{n-N+1:n-1}w_n)}{C(w_{n-N+1:n-1})}$$

  This ratio is called **relative frequency**.
- We always represent and compute language model probabilities in log format as log probabilities. Multiplying enough n-grams together would result in numerical underflow. By using log probabilities instead of raw probabilities, we get numbers that are not as small.
- Many results show that as N in N-gram increases, we could construct more meaningful sentences. So, to evaulate them, we need to use some form of evaluation.
- The best way to evaluate the performance of a language model is to embed it in an application and measure how much the application improves. Such end-to-end evaluation is called **extrinsic evaluation**. But these operations are expensive, so we go with other models.
- . An **intrinsic evaluation** metric is one that measures the quality of a model-independent of any application. This is an infamous method used by many ML models. It basically divides data into 3 parts: **Training, Development, Testing**. Often, we divide data into 80% training, 10% development, and 10% testing.
- In practice, we don't use probability but rather use **perplexity**. The perplexity (sometimes called PP for short) of a language model on a test set is the inverse probability of the test set, normalized by the number of words.

$$PP(W) = P(w_1w_2w_3....w_N)^{-\frac{1}{N}}$$

  We can replace probability with the above formulas.

- Minimizing perplexity is equivalent to maximizing the test set probability according to the language model. Generally, the perplexity of $UnigramBigram$, etc.. The perplexity of the two language models is only comparable if they use identical vocabularies.
- Our models may still be subject to the problem of **sparsity**. Sparsity is a problem arose when a word is not beside a particular word in the corpus(training set) but appears in testing, then the probability will be 0. There is a problem with unknown words too. They are called **out of vocabulary(OOV)** words. They shall be replaced by word $UNK$.

## 2.3 Naive Bayes and Sentiment Classification

:

- Naive Bayes's primary application is text categorization such as sentiment classifier, and spam detector. Naive Bayes basically understands a document as a bag of words and it doesn't consider the order in which they were.
- Most classification cases in language processing are done via supervised machine learning. Formally, the task of supervised classification is to take an input $x$ and a fixed set of output classes $Y = y_1, y_2, y_3, ..., y_M$ and return a predicted class $y \in Y$. A **probabilistic classifier** additionally will tell us the probability of the observation being in the class.
- Naive Bayes is a probabilistic classifier, meaning that for a document d, out of all classes $c \in C$ the classifier returns a class $\hat{c}$ which has the maximum posterior probability given the document.

$$\hat{c} = argmax_{c \in C} P(c|d)$$

- From Bayes theorem in week 1, we can replace $P(x|y)$ by $P(y|x) \times P(x)$. So, we can represent t a document d as a set of features $f_1, f_2, ..., f_n$:

$$\hat{c} = argmax_{c \in C} P(f_1, f_2, ...f_n|c) \times P(c)$$

  $P(f_1, f_2, ...f_n|c)$ is **likelihood** and $P(c)$ is called **prior** of a probability function.
- **Naive Bayes Assumption:** this is the conditional independence assumption that the probabilities $P(f_i|c)$ are independent given the class c and hence can be 'naively' multiplied as follows,

$$P(f_1, f_2, ...f_n|c) = P(f_1|c) \times P(f_2|c) \times P(f_3|c)... \times P(f_n|c)$$

- Naive Bayes calculations, like calculations for language modeling, are done in log space, to avoid underflow and increase speed.

$$c_{NB} = argmax_{c \in C} logP(c) + \sum_{i \in positions} logP(w_i|c)$$

- Let me show one example for a sentiment classifier:

$$\hat{P}(\text{"}fantastic\text{"}|positive) = \frac{count(\text{"}fantastic\text{"}, positive)}{\sum_{w \in V} count(w, positive)}$$

  We can also use Laplace smoothing in this. The above example shows the probability of the word "fantastic" in positive statements. So, they divided the count of "fantastic" in all positive statements and divided it by no. of words in positive statements.

- We will have a set of words to determine the statement whether it is positive or not. If our test data asks us a word that is not in the set, we will shift it to unknown words, and beforehand, we take the probability of all unknown words i.e, not in the list, and we can substitute it over there.

**Algorithm 1** TRAIN NAIVE BAYES

**Input:** $D$, $C$
**Output:** log $P(c)$ and log $P(w|c)$

**foreach** *class* $c \in C$ **do**
  // Calculate $P(c)$ terms $Ndoc \leftarrow$ number of documents in $D$ $Nc \leftarrow$ number of documents from $D$ in class $c$ $logprior[c] \leftarrow \log \frac{Nc}{Ndoc}$ $V \leftarrow$ vocabulary of $D$ $bigdoc[c] \leftarrow$ append$(d)$ for $d \in D$ with class $c$ **foreach** *word* $w$ *in* $V$ **do**
    // Calculate $P(w|c)$ terms $count(w, c) \leftarrow$ number of occurrences of $w$ in $bigdoc[c]$ $loglikelihood[w, c] \leftarrow \log \frac{count(w,c)+1}{\sum_{w_0 \in V} count(w_0,c)+1}$
  **end**
**end**
**return** *logprior, loglikelihood, V*

**Algorithm 2** TEST NAIVE BAYES

**Input:** *testdoc*, *logprior*, *loglikelihood*, $C$, $V$
**Output:** best $c$

**foreach** *class* $c \in C$ **do**
  $sum[c] \leftarrow logprior[c]$ **foreach** *position* $i$ *in testdoc* **do**
    $word \leftarrow testdoc[i]$ **if** $word \in V$ **then**
      $sum[c] \leftarrow sum[c] + loglikelihood[word, c]$
    **end**
  **end**
**end**
**return** $\arg\max_c sum[c]$

- There are some ideas to improvise the algorithm. We can remove duplicate words for counting in the same sentence. This is called **binary NB**. There is a problem with sentences like "don't dismiss this film, doesn't let us get bored". They are positive but there is a chance they can be treated as negative. So, during text normalization, prepend the prefix NOT to every word after a token of logical negation (n't, not, no, never) until the next punctuation mark. Sometimes our

dataset isn't sufficient for getting sentiment of some statements. There we can use **sentiment lexicons**. They are available on the internet.

- Naive Bayes has a lot of other applications too. For other tasks, like language ID—determining what language a given piece of text is written in—the most effective naive Bayes features are not words at all, but character n-grams, 2-grams ('zw') 3-grams ('nya', ' Vo'), or 4-grams ('ie z', 'thei'), or, even simpler byte n-grams, where instead of using the multibyte Unicode character representations called codepoints, we just pretend everything is a string of raw bytes.
- We also need to evaluate our machine-learning algorithm. To evaluate any system for detecting things, we start by building a confusion matrix. A confusion matrix is a table for visualizing how an algorithm performs with respect to the human gold labels, using two dimensions (system output and gold labels), and each cell labels a set of possible outcomes. Gold labels are a human-labeled dataset. We will calculate **precision** and **accuracy** using a confusion matrix as shown in the figure.

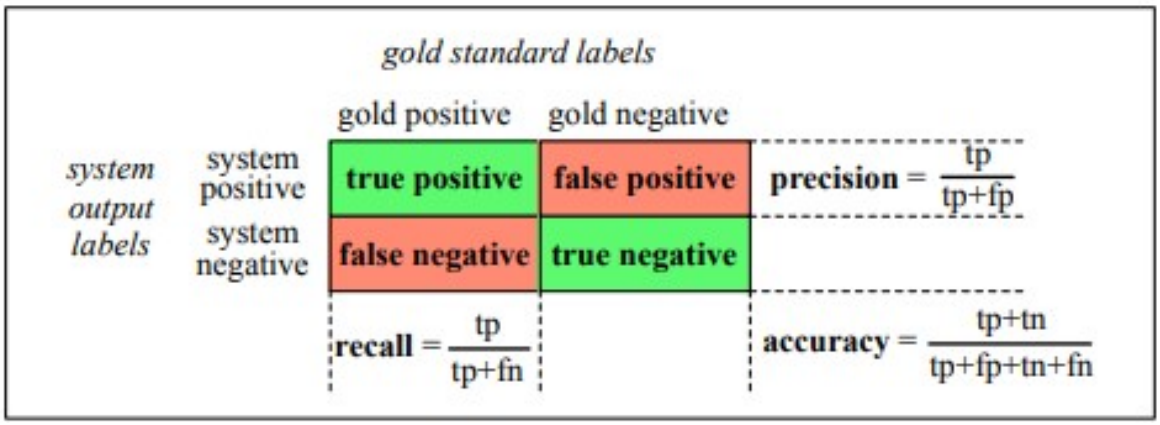


Figure 2: Confusion matrix

- Accuracy is not a good metric when the goal is to discover something that is rare, or at least not completely balanced in frequency, which is a very common situation in the world.
- Recall measures the percentage of items actually present in the input that were correctly identified by the system. The recall is defined as:

$$Recall = \frac{truepositives}{truepositives + falsenegatives}$$

- There are many ways to define a single metric that incorporates aspects of both precision and recall. The simplest of these combinations is the **F-measure**, defined as:

$$F_\beta = \frac{(\beta^2 + 1) \times Precision \times Recall}{(\beta^2 \times Precision) + Recall}$$

. The $\beta$ parameter differentially weights the importance of recall and precision, based perhaps on the needs of an application.

- For classifiers with more than 2 classes, we need to modify precision and recall. We do macro and micro averaging. In macro averaging, we compute the performance for each class and then average over classes. In micro averaging, we collect the decisions for all classes into a single confusion matrix and then compute precision and recall from that table

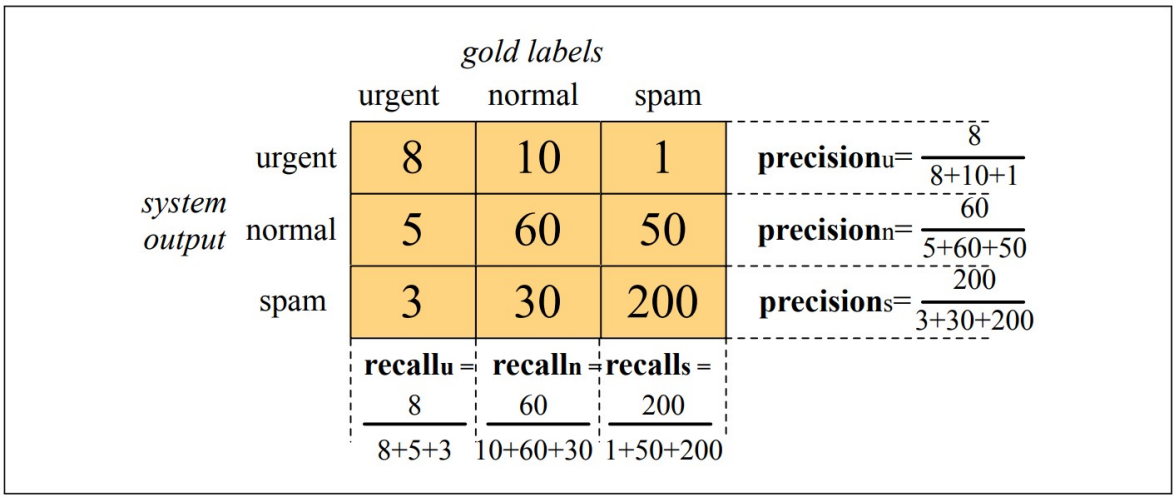


Figure 3: Confusion matrix with more than two classes

## 2.4 Neural Networks and Neural Language Models

- Neural networks are a fundamental computational tool for language processing, and a very old one. The architecture we introduce is called a feedforward network because the computation proceeds iteratively from one layer of units to the next. The use of modern neural nets is often called deep learning because modern networks are often deep (have many layers).
- At its heart, a neural unit is taking a weighted sum of its inputs, with one additional term in the sum called a bias term. Given a set of inputs $x_1...x_n$, a unit has a set of corresponding weights $w_1...w_n$ and a **bias** b, so the weighted sum z can be represented as:

$$z = b + \sum_i w_i x_i$$

This can be written as a dot product of two vectors

$$z = w.x + b$$

- Finally, instead of using z, a linear function of x, as the output, neural units apply a non-linear function f to z. We will refer to the output of this function as the **activation value** for the unit, a.

$$y = a = f(z)$$

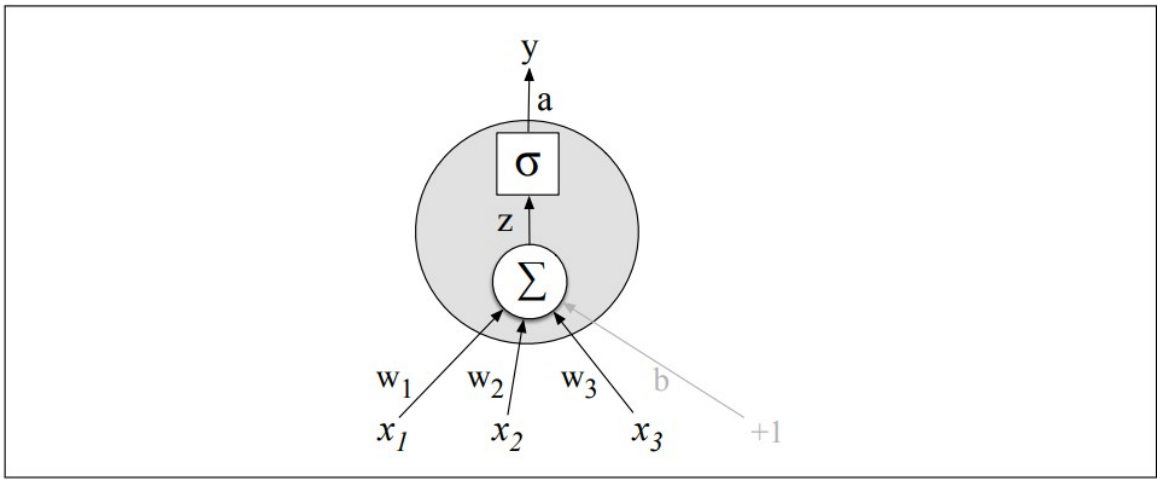


Figure 4: Basic Neural Network

- The popular functions are **sigmoid**, **tanh** and **ReLU**(rectified linear unit).
  - Sigmoid Function:
  $$f(z) = \frac{1}{1 + e^{-z}}$$
  - tanh function:
  $$f(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$
  - ReLU Function:
  $$y = max(x, 0)$$
- In practice, the sigmoid is not commonly used as an activation function. A function that is very similar but almost always better is the tanh function. The simplest activation function, and perhaps the most commonly used, is the rectified linear unit (ReLU). In the sigmoid or tanh functions, very saturated high values of z result in values of y that are saturated, i.e., extremely close to 1, and have derivatives very close to 0. Zero derivatives cause problems in learning.
- Perceptron is a simple neural network, which has a binary output and doesn't have a non-linear activation function. It's very easy to build a perceptron that can compute the logical AND and OR functions of its binary inputs, but it is impossible to build for XOR.
  $$f(x) = \begin{cases} 0, & \text{if } w.x + b \leq 0 \\ 1, & \text{otherwise} \end{cases}$$
  The intuition behind this important result relies on understanding that a perceptron is a linear classifier. Since XOR is not linearly separable(Figure 7), we need more than one layer to solve.

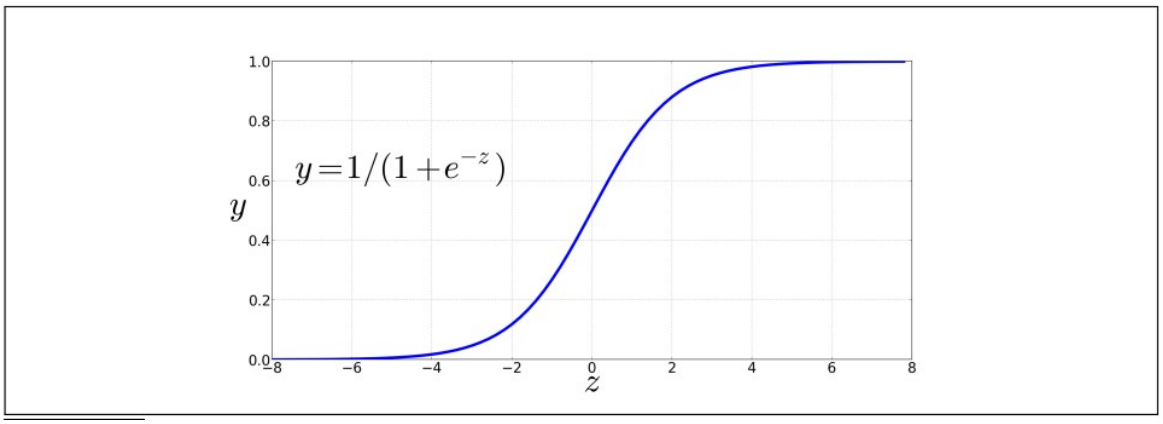


Figure 5: Sigmoid Function

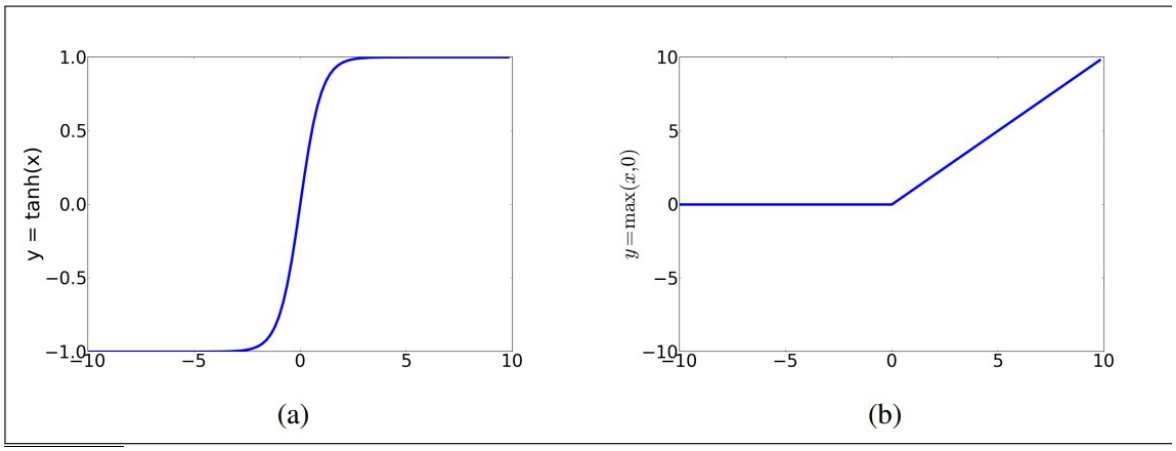


Figure 6: tanh and ReLU Functions

- The solution for this is neural networks. Check Figure 8 for understanding. It has 2 layers, the first layer having 2 ReLU functions $h_1 and h_2$ with weights and biases. mentioned. The second layer has a single ReLU function which depends on the output of two functions in the first layer. Figure 9 shows linear separability after the first layer.
- Note that the solution to the XOR problem requires a network of units with non-linear activation functions. A network made up of simple linear (perceptron) units cannot solve the XOR problem. This is because a network formed by many layers of purely linear units can always be reduced (i.e., shown to be computationally identical to) a single layer of linear units with appropriate weights, and we've already shown (visually, in Fig. 7) that a single unit cannot solve the XOR problem.
- A feedforward network (Figure 10) is a multilayer network in which the units are connected with no cycles; the outputs from units in each layer are passed to units in the next higher layer, and no outputs are passed back to lower layers. They are sometimes called multi-layer perceptrons.

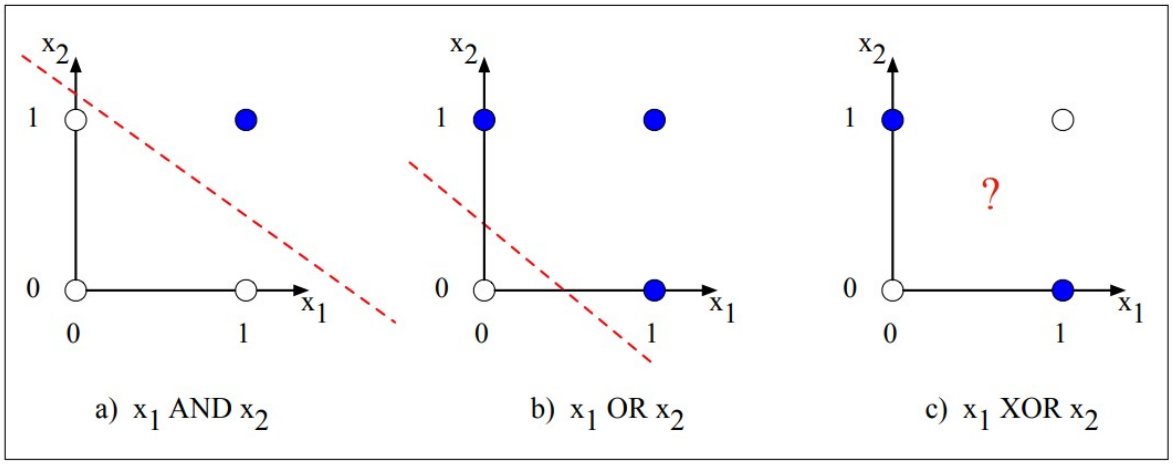


Figure 7: XOR is linearly inseparable

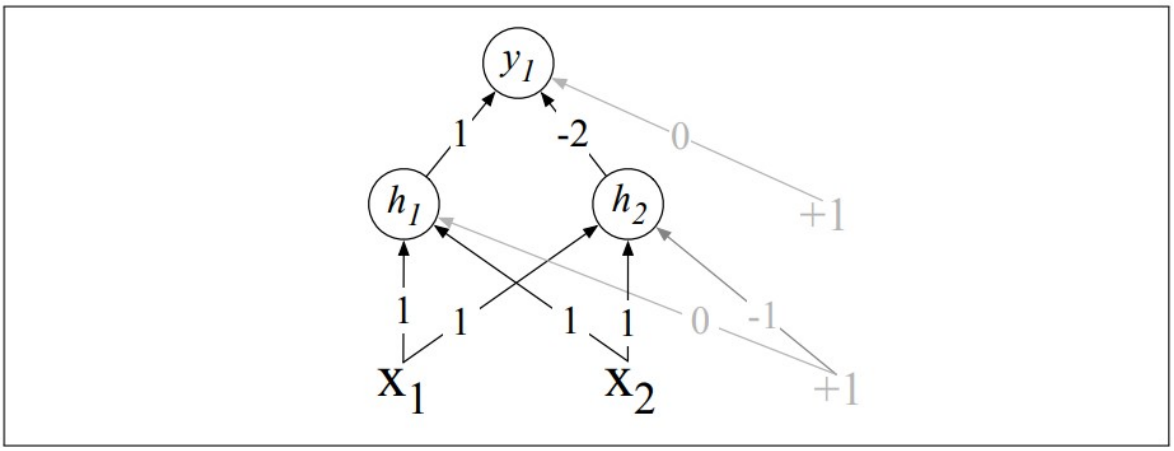


Figure 8: XOR is linearly inseparable

- Simple feedforward networks have three kinds of nodes: input units, hidden units, and output units. In the standard architecture, each layer is fully-connected, meaning that each unit in each layer takes as input the outputs from all the units in the previous layer, and there is a link between every pair of units from two adjacent layers. We represent the parameters for the entire hidden layer by combining the weight vector $w_i$ and bias bi for each unit $i$ into a single weight matrix W and a single bias vector b for the whole layer. Each element $W_{ji}$ of the weight matrix W represents the weight of the connection from the $i^{th}$ input unit xi to the $i^{th}$ hidden unit $h_j$. The computation only has three steps: multiplying the weight matrix by the input vector x, adding the bias vector b, and applying the activation function g (such as the sigmoid, tanh, or ReLU activation function defined above).

  $$h = \sigma(Wx + b)$$

  We're thus allowing $\sigma()$,or any activation function g( $\cdot$ ), to apply to a vector element-wise, so $g[z_1, z_2, z_3] = [g(z_1), g(z_2), g(z_3)]$.

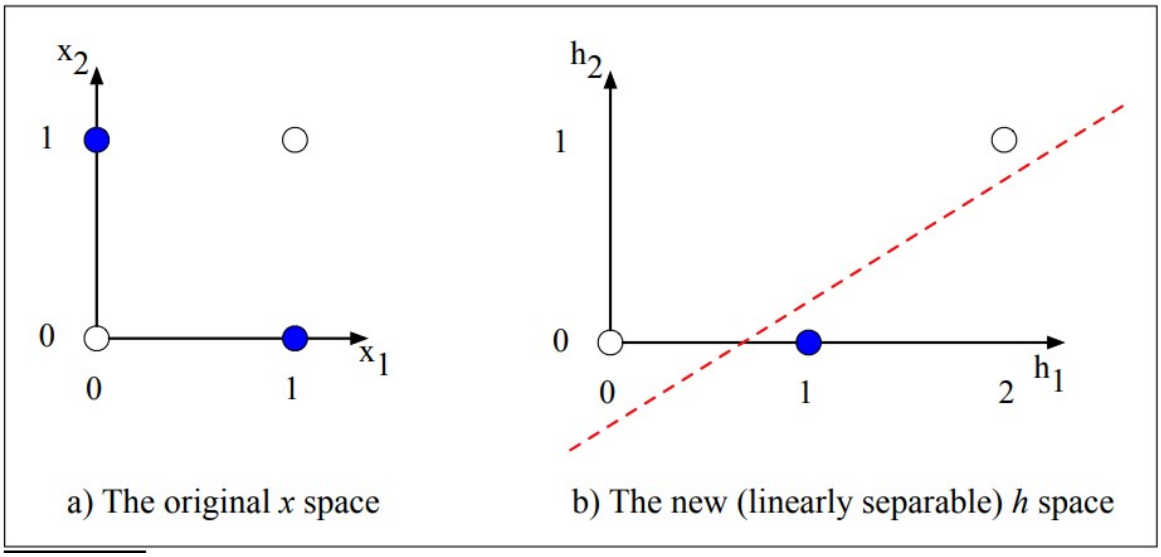


Figure 9: XOR is linearly inseparable

- Now, if we get $n_2$ values in output, we shouldn't report it. We need to normalize it and a convenient function is **softmax** function.

$$softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{d} e^{z_j}}$$

- For neural networks having more than 2 layers, we will follow some rules for representing matrices. We'll use superscripts in square brackets to mean layer numbers, starting at 0 for the input layer. Thus, a 2 layer network is represented as:

$$z^{[1]} = W^{[1]}a^{[0]} + b^{[1]}$$
$$a^{[1]} = g^{[1]}(z^{[1]})$$
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$
$$a^{[2]} = g^{[2]}(z^{[2]})$$
$$\hat{y} = a^{[2]}$$

- So, a code can be written as

**Algorithm 3** Final output for n-layered NN

**for** $i = 1$ *to* $n$ **do**
  $z^{[I]} \leftarrow W^{[I]} \times a^{[i-1]} + b^{[I]}$
  $a^{[i]} \leftarrow g^{[I]}(z^{[I]})$
**end**
$\hat{y} \leftarrow a^{[n]}$

- We can simplify the above argument by replacing the bias matrix with a dummy variable in inputs and altering the W matrix. Instead of our vector x having n

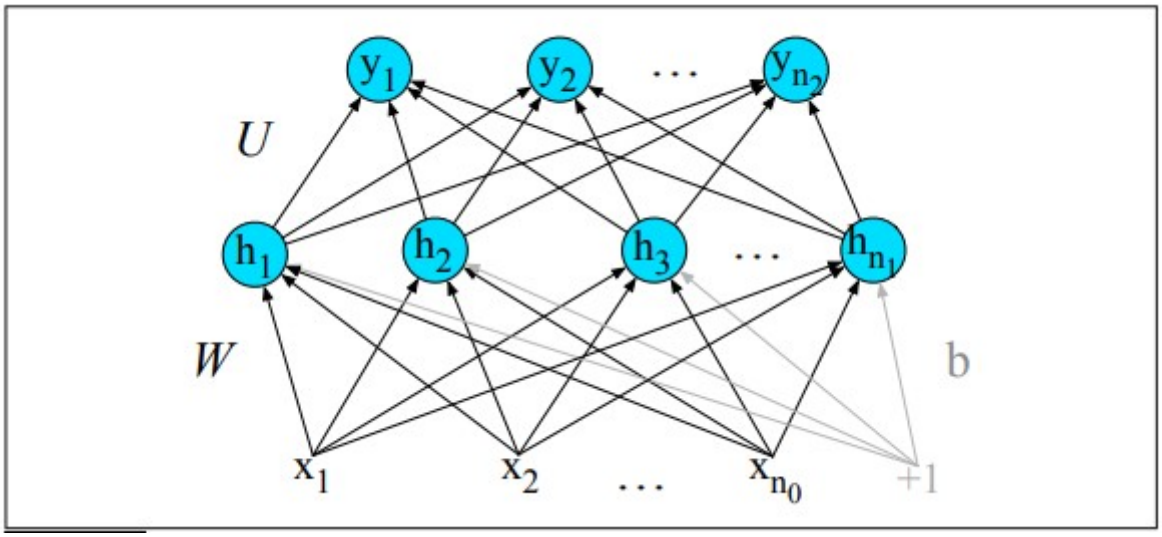


Figure 10: Simple 2 layer feedforward networ

values: $x = x_1, ..., x_n$, it will have n + 1 values, with a new 0th dummy value $x_0 = 1$:$x = x_0, ..., x_{n_0}$.

- A feedforward neural net is an instance of supervised machine learning in which we know the correct output y for each observation x. The goal of the training procedure is to learn parameters $W^{[I]}$ and $b^{[i]}$ for each layer $i$ that make $\hat{y}$ for each training observation as close as possible to the true y. First, we'll need a **loss function** that models the distance between the system output and the gold output, and it's common to use the loss function used for logistic regression, the **cross-entropy loss**. Second, to find the parameters that minimize this loss function. We will use **gradient descent** which is used in the regression. But, the final output contains all layer contributions, so how to finalize the error? The answer is the algorithm called error backpropagation or reverse differentiation.

- The cross-entropy loss that is used in neural networks is the same one we saw for logistic regression.

$$L_{CE}(\hat{y}, y) = -\sum_{i=1}^{C} y_i log\hat{y}_i$$

  We can simplify this equation further. Assume this is a **hard classification** task, meaning that only one class is the correct one and that there is one output unit in y for each class. If the true class is $i$, then y is a vector where $y_i = 1$ and $y_j = 0 \forall j \neq i$. A vector like this, with one value=1 and the rest 0, is called a **one-hot vector**.

- Computing the gradient requires the partial derivative of the loss function with respect to each parameter. Again, to get this, we need an algorithm called **error backpropagation**.

- A computational graph is a graph structure used to represent mathematical expressions. Nodes in the graph represent the variables in the expressions, and

edges represent the operations between them. This allows for the efficient evaluation of gradients of the expressions using techniques such as backpropagation. For, $y = c(a + b)$, the computational graph is shown in figure 11.

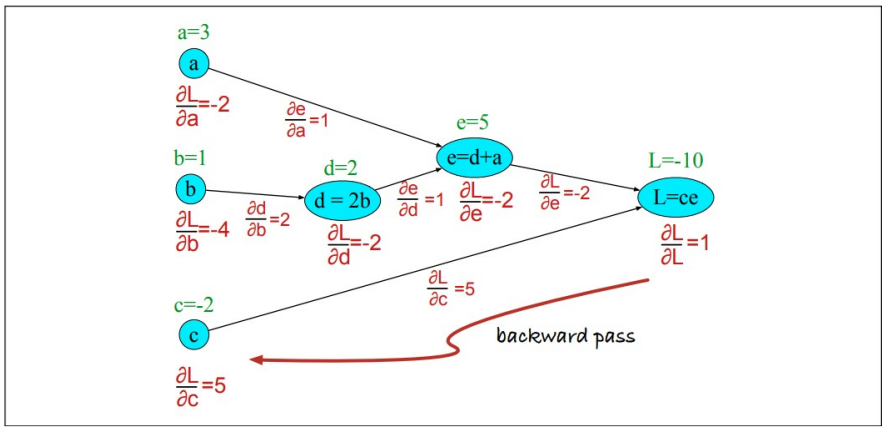


Figure 11: An example of a computational graph, where nodes represent variables and edges represent operations. The arrows represent the flow of gradients during the backpropagation algorithm.

- Backpropagation, also known as reverse-mode differentiation, is an algorithm used to efficiently calculate the gradient of a computational graph. It works by traversing the graph in a reverse direction, starting from the output node, and computing the gradient at each node by applying the chain rule of calculus. The gradients are then used to update the parameters of the model, such as the weights in a neural network. For the above example, we need to find derivatives of y w.r.t a,b,c.

$$\frac{\partial y}{\partial a} = \frac{\partial y}{\partial e}\frac{\partial e}{\partial a}$$

$$\frac{\partial y}{\partial b} = \frac{\partial y}{\partial e}\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}$$

$$\frac{\partial y}{\partial c} = e$$

- Various forms of regularization are used to prevent overfitting. One of the most important is **dropout**: randomly dropping some units and their connections from the network during training. Tuning of **hyperparameters** is also important. The parameters of a neural network are the weights W and biases b; those are learned by gradient descent. The hyperparameters are things that are chosen by the algorithm designer; optimal values are tuned on a dev set rather than by gradient descent learning on the training set. Hyperparameters include the learning rate $\eta$, the mini-batch size, the model architecture (the number of layers, the number of hidden nodes per layer, the choice of activation functions), how to regularize, and so on. Gradient descent itself also has many architectural variants such as **Adam**.
- Let's make a neural network that predicts the next word from previous words, similar to what we did in N-gram language models. Neural net-based language

models turn out to have many advantages over the n-gram language models. For a training set of a given size, a neural language model has much higher predictive accuracy than an n-gram language model. But. neural networks take much more time than n-grams due to complexity. We will describe a simple feedforward neural LM.

- The feedforward neural LM (figure 12) approximates the probability of a word given the entire prior context $P(w_t|w_1 : w_{t-1})$ by approximating based on the N previous words.

$$P(w_i|w_1, ....., w_{t-1}) \approx P(w_t|w_{t-N+1}, ..., w_{t-1})$$

- Representing the prior context as embeddings, rather than by exact words as used in n-gram language models, allows neural language models to generalize to unseen data much better than n-gram language models. For now, we'll assume we already have an embedding dictionary E that gives us, for each word in our vocabulary V, the embedding for that word. Relying on another algorithm to have already learned an embedding representation for input words is called **pretraining**.

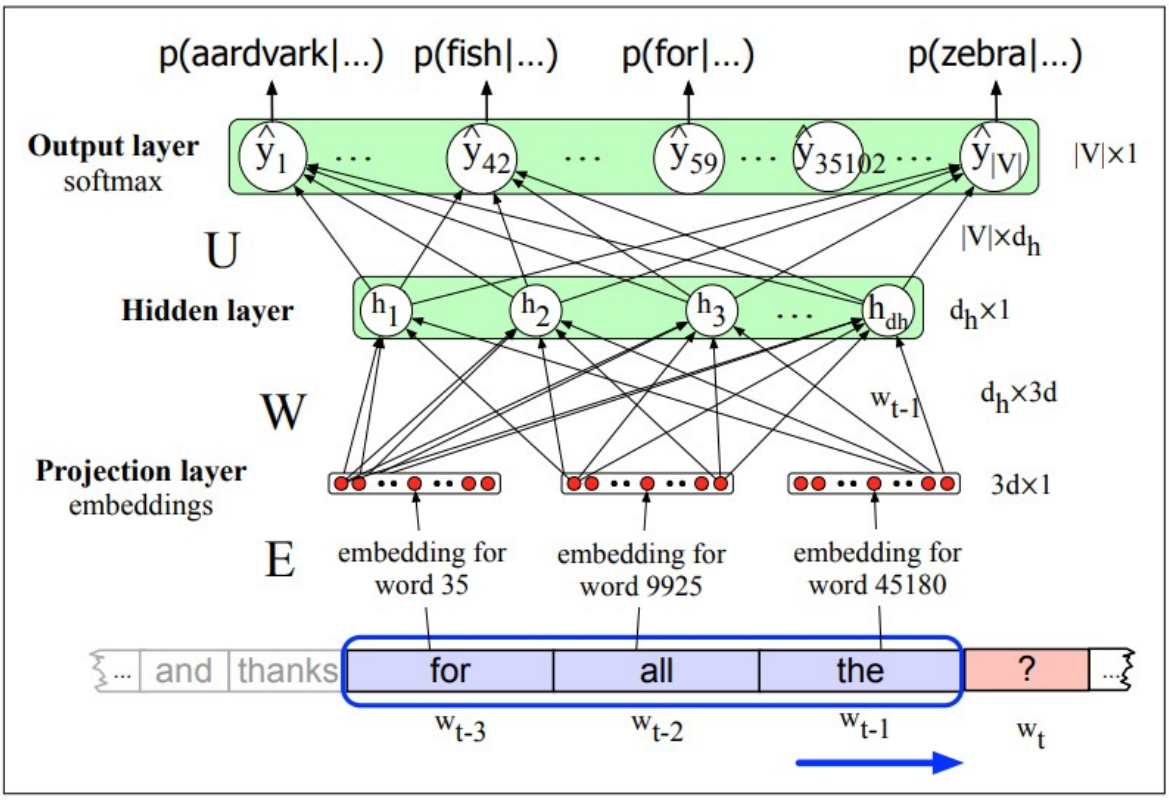


Figure 12: Simplified view of our algorithm

- These are the steps of the given algorithm:
  - **Select three embeddings from E:** Given the three previous words, we look up their indices, create 3 one-hot vectors, and then multiply each by the embedding matrix E. Consider$w_{t-3}$. The one-hot vector for 'the' (index

35) is multiplied by the embedding matrix E, to give the first part of the first hidden **projection layer**, called the projection layer. Since each row of the input matrix E is just an embedding for a word, and the input is a one-hot column vector $x_i$ for word $V_i$, the projection layer for input w will be $Ex_i = e_i$, the embedding for word i. We now concatenate the three embeddings for the context words.

- **Multiply by W:** We now multiply by W (and add b) and pass through the rectified linear (or other) activation function to get the hidden layer h.
- **Multiply by U:** h is now multiplied by U
- **Apply softmax:** After the softmax, each node i in the output layer estimates the probability $P(w_t = i|w_{t-1}, w_{t-2}, w_{t-3})$
- Then we apply the loss function to do gradient descent, to arrive at the correct values. To train the model, i.e. to set all the parameters $\theta = E, W, U, b$, we do gradient descent, using error backpropagation on the computation graph to compute the gradient. We will use the cross entropy function as our loss function. Then, similar to logical regression, we will use learning rate and derivatives to arrive at the final solution.

$$\theta^{s+1} = \theta^s - \eta \frac{\partial - logp(w_i|w_{i-1}, ..., w_{i-n+1})}{\partial \theta}$$

# 3. Week 4

I completed the code which I have uploaded in my GitHub repo. Here, you can visit and see all work I have done. It also has other files where I learned ML using Jupyter for the first time. It also has most of the datasets I used.


# Summary of my WiDS

I have had the opportunity to participate in a highly informative and stimulating project that has broadened my understanding and knowledge of machine learning and natural language processing. My mentor, Shravya, has provided invaluable guidance and support throughout the project, and I have greatly appreciated the opportunity to learn from her expertise. Despite the fact that I have not yet covered all the material, I feel confident that with further study and research, I will be able to deepen my understanding of these complex and rapidly evolving fields.

I would like to extend my sincere gratitude to the Analytics Club for organizing such a valuable event, and for providing me with the opportunity to learn from experienced professionals and peers. I am deeply grateful for the experience, and I look forward to applying what I have learned in future projects and endeavors.