Neural Networks and Sentiment Analysis
Summer of Science
Mid-Term Report

Mohit Kumar
Harshit Shrivastava

April 30, 2020

Introduction

Idea behind Machine Learning is to model and predict the outcomes of an environment based upon previous experiences and performance measures, and also without previous experiences(Clustering) where data is segregated based on similarity of parameters and proximity in vector space thus enabling us to find discrepancies if any in the data amongst other requirements. Before delving into modern advancements in the field of Neural Networks and Deep Learning, a thorough pre-requisite knowledge of Machine Learning Algorithms, their uses, drawbacks and various parameters for measuring performance of the model is required. Algorithms can be grouped on the basis of their learning style and similarity in form or function amongst other factors. On the basis of learning style, they can be classified into Supervised, Semi-supervised and Unsupervised learning algorithms which are explain further in this report.

This report shall contain:

1. Types of Learning
   (a) Based on Input data labels
      i. Supervised
      ii. Semi-Supervised
      iii. Unsupervised
   (b) Based on Similarity
      i. Regression Algorithms
      ii. Instance based Algorithms
      iii. Regularization Algorithms
      iv. Decision Tree Algorithms
      v. Bayesian Algorithms
      vi. Clustering Algorithms
2. Regression
3. Classification
4. Performance and Optimizing parameters
   (a) Normalization
   (b) Standardisation
   (c) Cost/Loss/Empirical Risk/Objective function
      i. MSE
      ii. Cross-Entropy
   (d) Gradient Descent
   (e) Regularization
      i. L2
      ii. L1
5. Neural Networks
6. Training Neural Networks
   (a) Hidden Layers and Activations
      i. ReLU
      ii. LReLU
      iii. ELU
      iv. Softmax/Sigmoid
      v. tanh
   (b) Depth vs Breadth of Hidden Layers
   (c) Loss Functions in Neural Networks
      i. Regression Problem
      ii. Binary Classification
      iii. Multi-class Classification

(d) Weight Initialization

(e) Parameter updates

i. Loss

ii. Learning Rate

iii. Gradient Descent Algorithms or Optimizers

A. Adagrad

B. SGD

C. Momentum

D. RMSprop

E. Adam

iv.

(f) Metrics

i. Accuracy

ii. Recall

iii. Specificity

iv. Precision

v. F1 Score

7. CNNs

(a) Introduction

(b) Kernels

(c) Layers

(d) MNIST Dataset Implementation and comparison

8. Recurrent Neural Networks

(a) Introduction

(b) Application Areas

(c) Architectures

i. LSTM

ii. GRU

iii. Bi-Directional RNN

(d) Sentiment Analysis with RNNs and Recurrent-CNN

(e) Results and Discussion

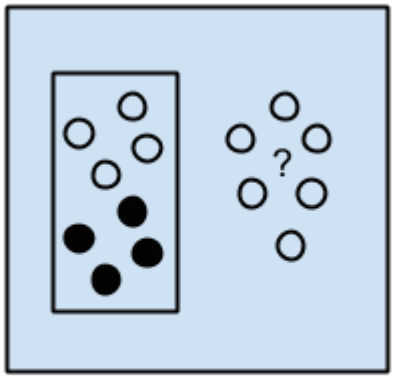
9. References

1 Types of Learning

Type of learning and model approximation depends upon whether the input data is labelled/corresponding outputs to input features is available or not. Also it can be classified on the basis of similarity of working principle.

1. Based on Input data labels:

- Supervised:



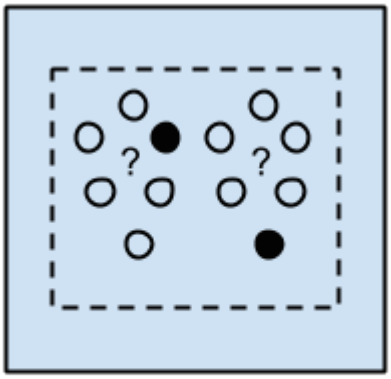
Supervised Learning Algorithms

If the input data(training data) has known labels corresponding to respective input features as a mail is spam/not spam. Here mail's contents are input features and mail labelled as spam/not spam is known output.

Model is prepared through a training process of making assumptions by model with its initial coefficients/weights and the output is checked with known output. The training process continues until model achieves a desired level of accuracy on the training data or pre-defined number of iterations are done.

Examples are Linear Regression, Multiple Linear Regression, Logistic Regression, Neural Networks, etc.

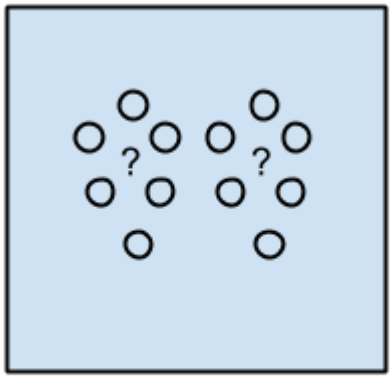
- Semi-Supervised:



Semi-supervised Learning Algorithms

Input data is a mixture of labelled and unlabelled data. There is a desired prediction problem but the model must learn the structures to organize the data as well as make predictions. This type of classification problem is Semi-Supervised classification.

- Unsupervised:



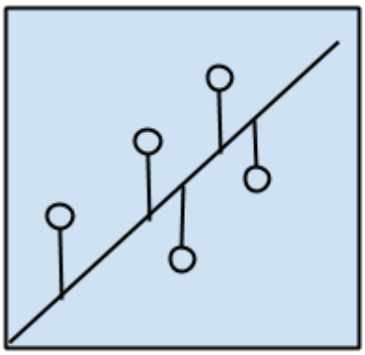
Unsupervised Learning Algorithms

Input data is not labelled, i.e., doesn't have corresponding actual outputs to input features to train the model. Model is prepared by deducing structures present in the input data which can be based on similarity of data in vector space or angles subtended by individual data points in the vector space or minimum distance to a class in vector space which itself has many variations to classify data in various formats.

Examples are Nearest Neighbors, K/C-Nearest Neighbors, K-Means, etc.

2. Based on similarity:

- Regression Algorithems :

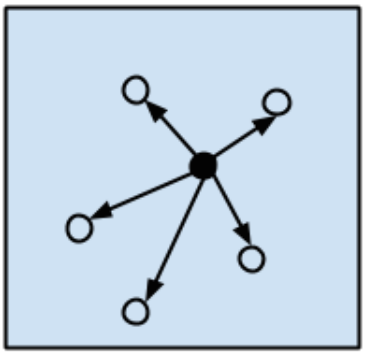


Regression Algorithms

Regression is concerned with modelling the relationship between variables that is refined with iterations using a measure of error in the predictions made by the model and outputs a continuous value. Regression methods are the workhorse of statistics. Some of it's examples are as follows,

  - Linear Regression
  - Logistic Regression
  - Stepwise Regression
  - Multivariate Adaptive Regression Splines(MARS)

- Instance based Algorithms :

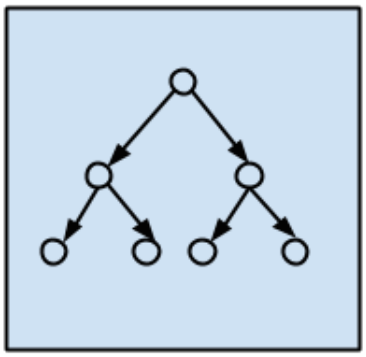


Instance-based Algorithms

Such methods typically build up a database of example data and compare new data to the database using similarity measure in order to find the best match and make a prediction. For this reason instance based methods are also called Memory-Based Learning. Some of it's examples are as follows,

- k-Nearest Neighbors (KNN)
- Support Vector Machines (SVM)
- Minimum Distance to Mean

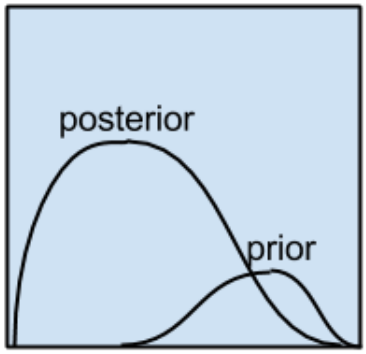
- Regularization Algorithms : It is an extension made to other methods which are typically Regression methods that penalizes the model based on their complexity, favoring simpler models by reducing prediction in weights by a small amount to prevent the model from overfitting and generalize better.
  - Ridge Regression
  - Elastic Net
  - Least Absolute Shrinkage and Selection Operator(LASSO)
- Decision Tree Algorithms:



Decision Tree Algorithms

Decision tree methods construct a model of decisions made based on actual values of attributes in the data.

  - Classification and Regression Tree
  - Decision Stump
  - Conditional Decision Trees
  - Chi-Squared Automatic Interaction Detection (CHAID)
  - Iterative Dichotomiser 3 (ID3)
- Bayesian Algorithms:

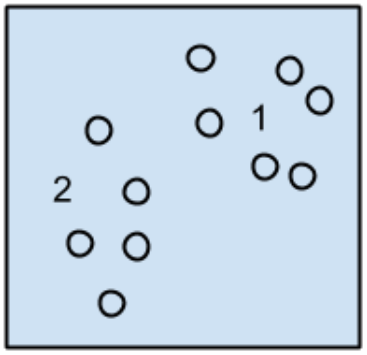


Bayesian Algorithms

As the name suggests, those models that explicitly apply Bayes' Theorem for problems like Classification and Regression.

- Naives Bayes
- Gaussian Naive Bayes
- Multinomial Naive Bayes
- Bayesian Network

- Clustering Algorithms:



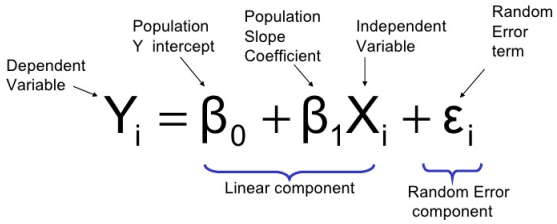
Clustering Algorithms

Clustering methods are concerned with using the inherent structures in the data to best organize the data into groups of maximum commonality.

- k-Means
- k-Medians
- Expectation Maximisation (EM)

2 Regression

- Regression refers to estimation of a continuous numerical value(output) from a set of inputs where outputs from different inputs might be same or not.



3 Classification

- Classification refers to assigning of classes/clusters based on the value obtained and threshold used on output. In Classification, output value is not continuous but rather discrete and input data is used to generate an output result which

based on threshold decides the class of input data.

$$output = 1, if Y_i > 0.5 \tag{1}$$

$$(or) 0, if Y_i < 0.5 \tag{2}$$

4 Performance and Optimizing parameters

- Normalization:
  This is done when the features have different ranges thus affecting their effect on output. So to provide equal values to all features in determining the output in beginning at the time of weight initialization, normalization is done.It is particularly useful in image data where DN values of individual pixels vary from 0 to 255. Thus it brings down the entire range to 0 to 1. To Normalize a feature, all the values of that feature is divided by the range of that feature(maximum value - minimum value) to bring all the values in range [0,1].
  The equation 3 shows the updation of original feature values,

$$\frac{x_{[i]}}{(x_{[max]} - x_{([min])})}, \tag{3}$$

  for i in range 0 to (m-1)
  where m = number of samples.

- Standardisation:
  Standardisation/Z-score normalization is rescaling of data to conform it to Gaussian Distribution standards where mean of the data is subtracted from the feature values followed by division of data with Standard Deviation of the feature data, thus making mean of the data equal to 0 and standard deviation equal to 1. It's done when the features are in different scales so one feature's weights might be updated more quickly than others.

$$\frac{x_i - \bar{x}}{\sigma} \tag{4}$$

  where, $\bar{x}$ is calculated as follows,

$$\frac{\sum_{i=1}^{n} x_i}{n} \tag{5}$$

  σ is calculated as,

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

- Cost function/Loss function/Empirical Risk/ Objective function:
  Cost function determines the deviation of the predicted output from the ground truth with respect to modelling the relationship between the input/features and the output we observe. As we'll see different different types of cost functions are used for different implementations in further sections, it is good to have basic idea about what we're measuring. Some common types of cost functions are listed below:

  1. Mean Squared Error(MSE)
     This is one of the simplest cost functions yet amongst the most effective. Other name for this cost function is Quadratic Cost function or Sum of Squared Errors.

$$MSE = \frac{1}{N}\sum_{i=1}^{n}(y_i - \bar{y_i})^2$$

     Where N is the number of samples/outputs, can be seen either way. y_i is the ground truth value and $\bar{y_i}$ is the predicted value at sample i.
     As we can see the difference between predicted and true value is squared which is there for a reason as it makes the cost function quadratic thus only Global minima present. But it is highly susceptible to noise in the training data.

2. Cross Entropy

   With it's origin at Information theory where it determined how many bits have been lost, in ML it's defined as the difference between probability distributions of Predicted model and Environment over the output with it's value ranging between 0 to 1. Value increases as the predicted distributions stray away from target Label. Below is the loss function multi-class classification with N number of classes. Separate loss is calculated for each class then summed up together.

$$H(x) = -\sum_{i=1}^{n} p(x).logq(x)$$

   p(x) is 1 for true class and 0 for rest of the classes. Thus mapping Probability distributions of all the classes. Let's take an Example,

$$P(Orange) = [0.6, 0.3, 0.1]$$

$$A(Orange) = [1, 0, 0]$$

$$CrossEntropy = -(1 * Log(0.6) + 0 * Log(0.3) + 0 * Log(0.1)) = 0.51$$

3. Kullback-Leibler(KL) Divergence

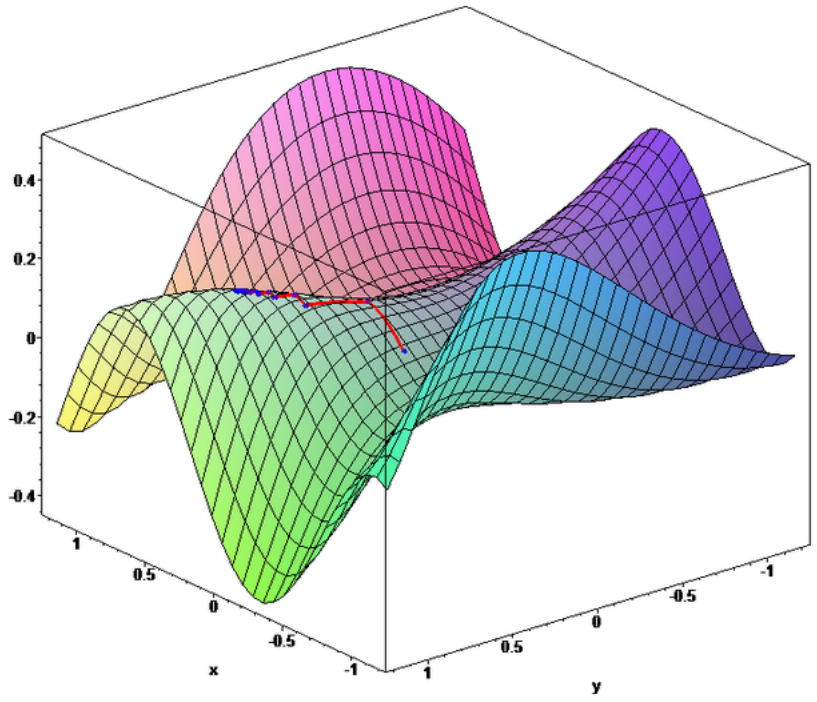
   Let's first look at it's algorithm,

$$KL(P||Q) = \sum_{i=1}^{N} p(x).log\frac{p(x)}{q(x)}$$

   It still measures difference between probability distributions p and q but is not a distance between the two distributions whereas cross entropy is.

- Gradient Descent:

  Gradient Descent is the way of minimizing the loss value we calculated using any of the above steps. It attempts to find a local or global minima of a function where the loss is at it's minimum or near minimum. Below is an image representing loss surfaces gradient descent might have to work on based on loss functions.



  It can further be of two main types,

  1. Stochastic Gradient Descent

     In this methods, each output is compared with true value and update of weights is done at every sample step.

  2. Batch Gradient Descent

     In this methods, a bunch of outputs are generated corresponding to respective input values and a single loss function is calculated for the whole batch and minimizing that value becomes the target of the function.

- Regularization

  Regularization works in tandem with gradient descent and loss function and aims to prevent the over-fit of the data by minimizing parameter values thus keeping them comparatively smaller and make the model generalize better. It's two types are discussed below,

  1. Ridge or L2 Regularization

     Here, predictors are penalized if coefficients value deviate too much from 0 this way we keep predictors while keeping complexity low. λ is the regularization parameter whose value need to be adjusted to control the strength of regularization. It is also known as soft regularization as it tends to lower the parameters but don't necessarily make them zero.

     $$J(w) = \lambda w^2 + \sum_i (w^T x_i - y_i)^2. \tag{6}$$

     Then the stationary condition is

     $$\frac{\partial J}{\partial w} = \lambda w + \sum_i (w^T x_i - y_i) x = 0 \tag{7}$$

     $$(XX^T + \lambda I) w = Xy \tag{8}$$

     $$w = (XX^T + \lambda I)^{-1} Xy \tag{9}$$

     $$= X\alpha. \tag{10}$$

  2. Lasso or L1 Regularization

     In this regularization, penalty is assigned for non-zero or far away from 0 coefficients but it is not applied on sum of squared values of coefficients but rather on absolute sum of coefficients thus making many of the coefficients which take abruptly high value equal to zero thus removing dependency from those predictors thus it is also known as feature selector and regularizer. Thus also called Hard Regularizer.

     $$J(w) = \lambda w + \sum_i (w^T x_i - y_i)^2. \tag{11}$$

5 Neural Networks

Why do we need Neural Networks?

Simple answer would be, Non-Linear models. But that can be solved using logistic regression too then why Neural Networks? Answer is because of complexity of feature designing which is a task in itself. Suppose if there are 100 inputs from environment and we observe that the model is under-fitting then our first thought would be to square the inputs and then feed them as features. Let's look at it how many additional features we get now, with squaring of features, number of features grow by half of the squared number of inputs and that would be equal to 5000 features. and when talking about thousands of features like inputs of CNNs, that would very well lead to millions of features. Here Neural Networks come to our rescue as it automatically design features based on the number of layers and neurons specified by the user.

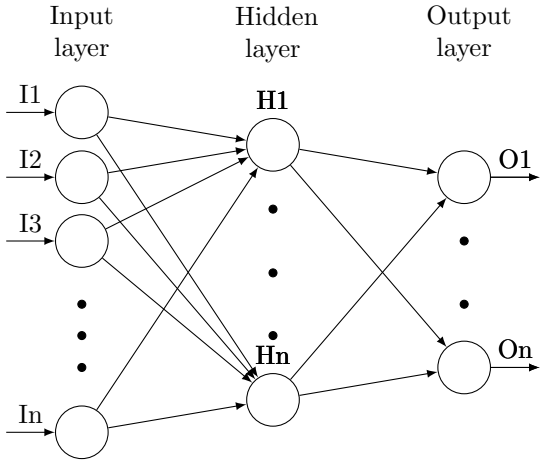


Figure 1: Sample Neural Network Architecture

Each of the Hidden Layers Neurons and output layer neurons have activation functions which is upto us to define linear or non-linear.

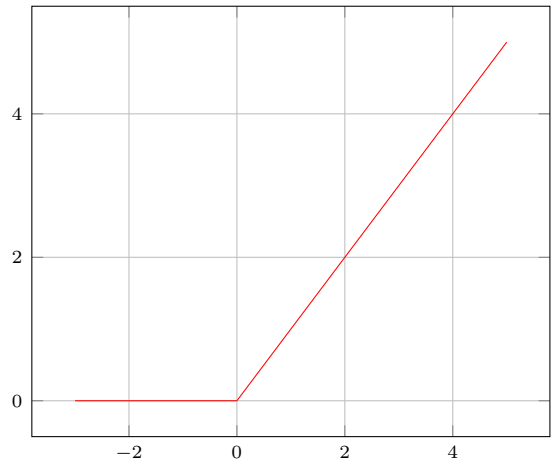
6 Training Neural Networks

1. Hidden Layers

   Determining number of hidden layers to keep in our Neural Network is one of the crucial things to keep in mind as it defines the complexity of the model proportionally. Hidden layers take parameter weighted sum of the features from previous layer and apply an activation function to the weighted sum providing non-linearity at nodes. Some of the common activation functions are,

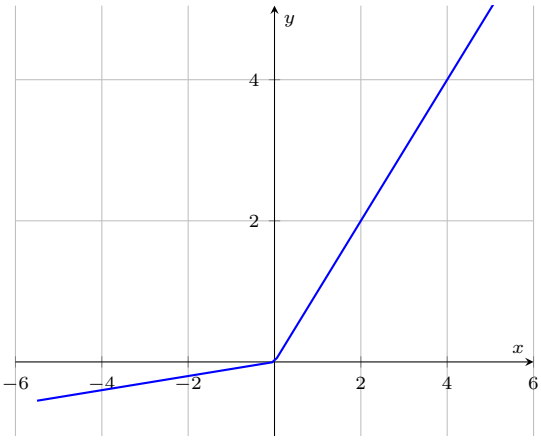
   - ReLU/Rectified Linear Unit

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{12}$$



   - LReLU/Leaky ReLU

$$f(x) = \begin{cases} 0.01x & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{13}$$



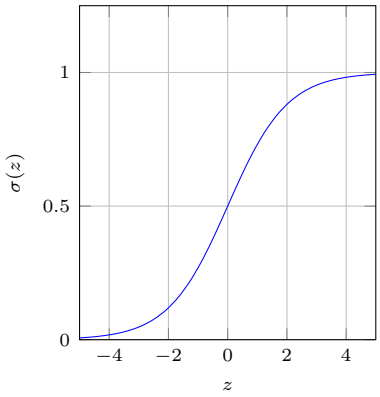
   - ELU/Exponential Linear Unit

$$f(x) = \begin{cases} \alpha.(e^z - 1) & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \tag{14}$$

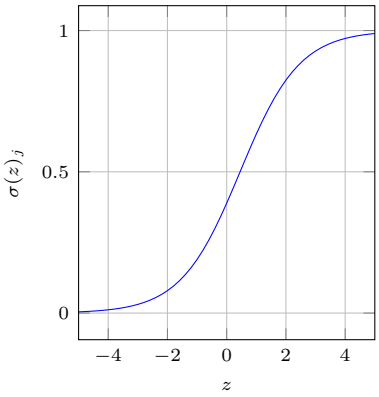
   - Softmax/Sigmoid

     Sigmoid is used for binary classification and Softmax is used for multi-class classification.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_j}} \tag{15}$$



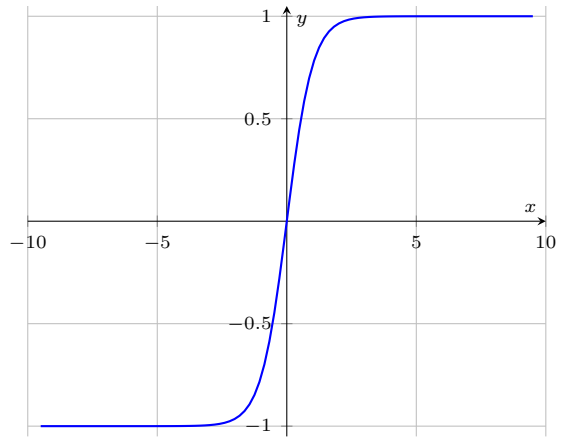
(a) Sigmoid activation function.



(b) Softmax activation function.

Figure 2: Sigmoid and Softmax activation functions

- tanh



2. Depth vs Breadth of Hidden Layers

   Empirical results show that a neural network with more depth than width is more capable of representing complex functions particularly when number of parameters are very large and environment is very complex. Then having an infinite number of neurons which will be calculated for simultaneously becomes impractical. Thus, it's bettwe to go deep than wide.

3. Loss Functions in Neural Networks

   Choice of activation functions directly affect choice of loss functions in a Neural Network, thus, the two design elements are connected.

   - Regression Problem

     Regression problem calls for a neural network which will take features and output a continuous value. Thus, one output layer. Here mean absolute error or mean squared error or mean squared logarithmic error can be used. If we want to output multiple values, then the errors in individual predicted value w.r.t. true value and their weighted sum taking into consideration importance of each output value is calculated. Values of higher importance given more weight.
   - Binary Classification Problem

     Output layer will consist of one node with sigmoid activation function. Loss function used is Binary Cross-Entropy or Hinge Loss can be used.
   - Multi-Class Classification Problem

     Output layer have nodes as many as number of classes in data or we specified, thus softmax activation function with Categorical Cross-Entropy loss or KL Divergence loss can be utilised.

4. Weight Initialization

   If all the weights in a Neural Network are initialized with same value, say 1, then the weighted value of inputs and features fed into the Neural Network will be same for every node, thus, activation functions will output same value at every node and loss calculation will be same for every node in the output layer leading to formation of a Symmetry. To break this symmetry, random weight initialization is employed.

5. Parameter Updates

- Loss is calculated for each sample/batch of samples and then fed into gradient descent algorithm where gradient of the loss is calculated with respect to all the weights of the neural network by employing chain rule.
- Learning Rate

  Learning rate defines the size of steps optimizer takes while moving in the gradient direction. Too small steps will cause optimizer to take long time to converge to a solution and if the learning rate is too high then it risks overshooting the convergence or might even diverge from the minima.
- Gradient Descent Algorithms or Optimizers because they optimize parameter values are of many types. Some popular ones are as follows,
  - Adagrad

    One of the disadvantages of all optimizers is that the learning rate is constant which we want to reduce as we are approaching minima so as to avoid overshooting it.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \epsilon}} \cdot [\frac{\partial L}{\partial w_t}] \tag{16}$$

    where

$$v_t = v_{t-1} + [\frac{\partial L}{\partial w_t}]^2 \tag{17}$$

    Default values from Keras:
    * $\alpha = 0.01$
    * $\epsilon = 10^{-7}$
  - Stochastic gradient descent

    vanilla SGD updates the current weight using the current gradient $\frac{\partial L}{\partial w}$ multiplied by some factor called the learning rate, .

$$w_{t+1} = w_t - \alpha [\frac{\partial L}{\partial w_t}] \tag{18}$$

    where α is the learning rate, L is loss.
  - Momentum

    Instead of depending only on the current gradient to update the weight, momentum of gradient descent can also be used where momentum will be higher is the gradient descent is coming from a steep loss gradient. Now if it hits a local minima, momentum will drive it forward and providing a bigger step of movement then that it would've got using normal learning rate and gradient.

$$w_{t+1} = w_t - \alpha m_t \tag{19}$$

    where

$$m_t = \beta m_{t-1} + 1 - \beta [\frac{\partial L}{\partial w_t}] \tag{20}$$

    Common Default value:
    * $\beta = 0.9$

– RMSprop
Root Mean Square Prop is another adaptive learning rate algorithm that builds on Adagrad. Instead of taking cumulative sum of squared gradients like Adagrad, exponential moving average is taken of gradients.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t + \epsilon}}.[\frac{\partial L}{\partial w_t}] \tag{21}$$

where

$$v_t = \beta v_{t-1} + (1 - \beta)[\frac{\partial L}{\partial w_t}]^2 \tag{22}$$

v is initialised as 0. Default values from Keras:

* $\alpha = 0.001$
* $\beta = 0.9$
* $\epsilon = 10^{-6}$

– Adam
Adam optimizer is simply a combination of RMSprop and momentum, acting upon gradient component by use of m and exponential moving average of gradients just like momentum, and also upon learning rate component by dividing learning rate by square root of v, the exponential moving average of squared gardients like RMSprop.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{v_t} + \epsilon}.\hat{m_t} \tag{23}$$

where

$$\hat{m_t} = \frac{m_t}{1 - \beta_1^t} \tag{24}$$

$$\hat{v_t} = \frac{v_t}{1 - \beta_2^t} \tag{25}$$

are the bias corrections. And,

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)[\frac{\partial L}{\partial w_t}] \tag{26}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)[\frac{\partial L}{\partial w_t}]^2 \tag{27}$$

with m and v initialised to 0. Proposed default values,

* $\alpha = 0.001$
* $\beta_1 = 0.9$
* $\beta_2 = 0.999$
* $\epsilon = 10^{-8}$

- Regularization
As we have defined before, Regularization's aim is to decrease the complexity of the model, increase generalization and prevent overfitting.

6. Metrics

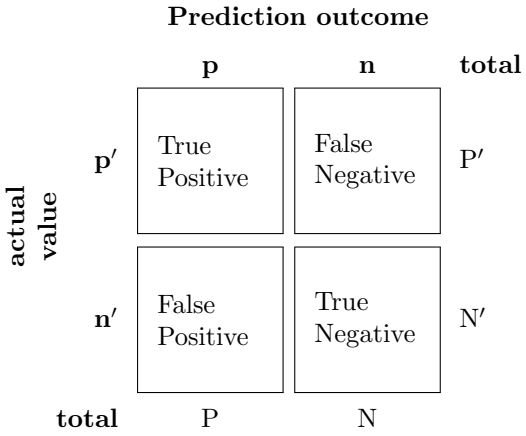
for Classification problems, confusion matrix is formed holding values of True Positives, True negatives, False Positives and False Negatives. Here, we'll define them,

(a) True Positive (TP)
True positives is the number of Positives which were Positive in ground truth as well.

(b) False Positive (FP)
False positive is the number of those samples which were Negative or False in ground truth but classified as Positive or True.

(c) False Negative (FN)
False Negative is the number of those samples which were Positive or True in ground truth but classified as Negative or False.

(d) True Negative (TN) True Negative is the number of those Negatives which were Negative in ground truth as well.



Now, we'll define Performance metrics,

i. Accuracy
Measure of fraction of times model correctly predicted TP and TN out of total number of predictions. But, it can be misleading, as in cases, where class imbalance is very high and one class dominates the data then by predicting each sample as the dominating class, model can get good Accuracy.

$$Precision = \frac{TP + TN}{TP + FP + TN + FN} \tag{28}$$

ii. Recall
Measure of fraction of times the model correctly predicts positive cases(TP) from total number of positives cases(TP+FN)

$$Precision = \frac{TP}{TP + FN} \tag{29}$$

iii. Specificity
Measure of fraction of times the model correctly predicts negative cases(TN) from total number of negatives cases(TN+FP)

$$Precision = \frac{TN}{TN + FP} \tag{30}$$

iv. Precision
Measure of accurate positive(TP) predictions from total positive predictions of the model(TP+FP).

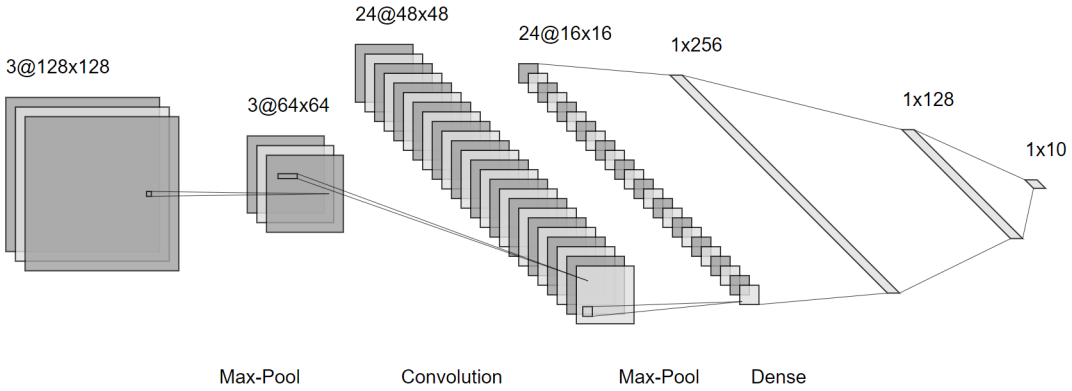
$$Precision = \frac{TP}{TP + FP} \tag{31}$$

v. F1 Score
We need to keep Recall and Precision in balanced and optimum form for best predictions. Thus, F1 score was invented which is a harmonic mean of recall and precision.

$$F1Score = \frac{2 * Prevision * Recall}{Precision + Recall} \tag{32}$$

Now with our Basics Clear, we'll look at complex Neural Network problems with results of their implementations and corresponding results.

7 Convolutional Neural Networks

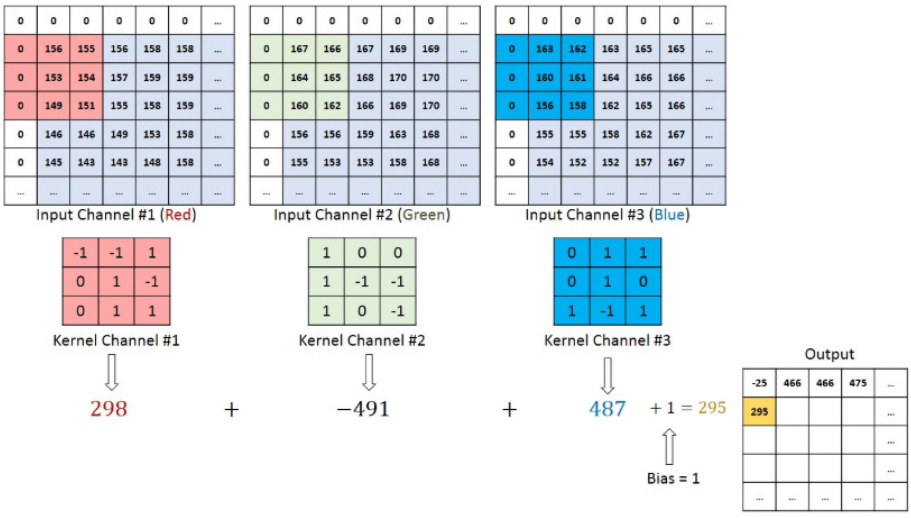


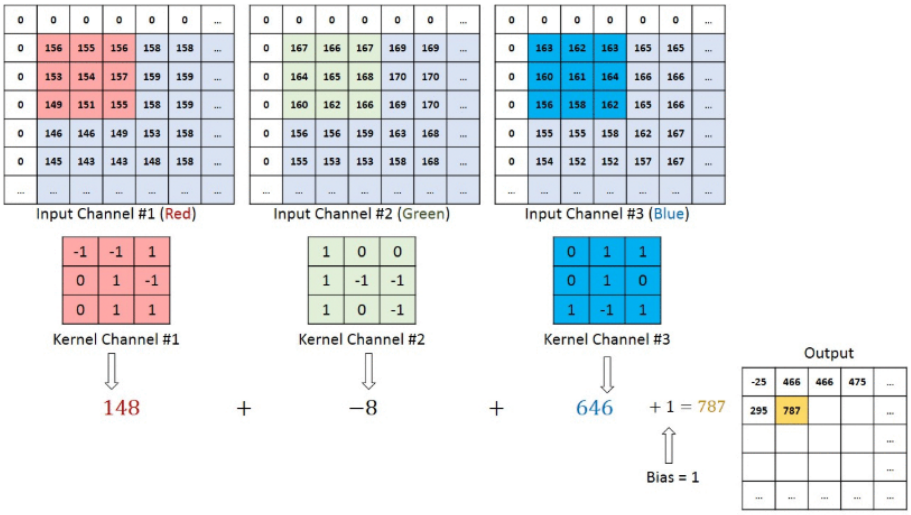
1. Introduction

   Convolutional Neural Network is a Deep Learning algorithm which takes in images data set and assigns learn-able weights and biases to kernel parameters. Kernel parameters are different for different output channels but same for a particular channel/feature map. Thus, CNNs have an advantage of learning small number of weights which are repeated throughout for getting a feature map. Above is an input image which was fed in with dimensions 128*128 and 3 channels(R,G and B). Maxpooling and convolutions are performed, as we can see number of channels are greatly increased towards the end and size of the input image gets smaller and smaller. In the end, flattening is done and final layer has 10 output nodes providing the output. Number of output nodes depend upon number of output classes. If in the end, we are to do flattening and feed it to a dense network then why Convolutional NN, why not simple NN? Answer is that in cases where the environment is complex and there are pixel dependencies throughtout the input image then simple NN would'nt be of much help and would fare poorly relatively. ConvNet is able to successfully capture the Spatial and Temporal dependencies through application of relevant kernels.

2. Kernels

   Kernels for a particular feature map mover over the entire pixel value grid and perform convolution on the input image and it's channels to form a feature map which can be understood by following images depicting Kernel movement and convolution.





source:https://www.maths.tcd.ie/ dwilkins/LaTeXPrimer/Calculus.html

3. Layers

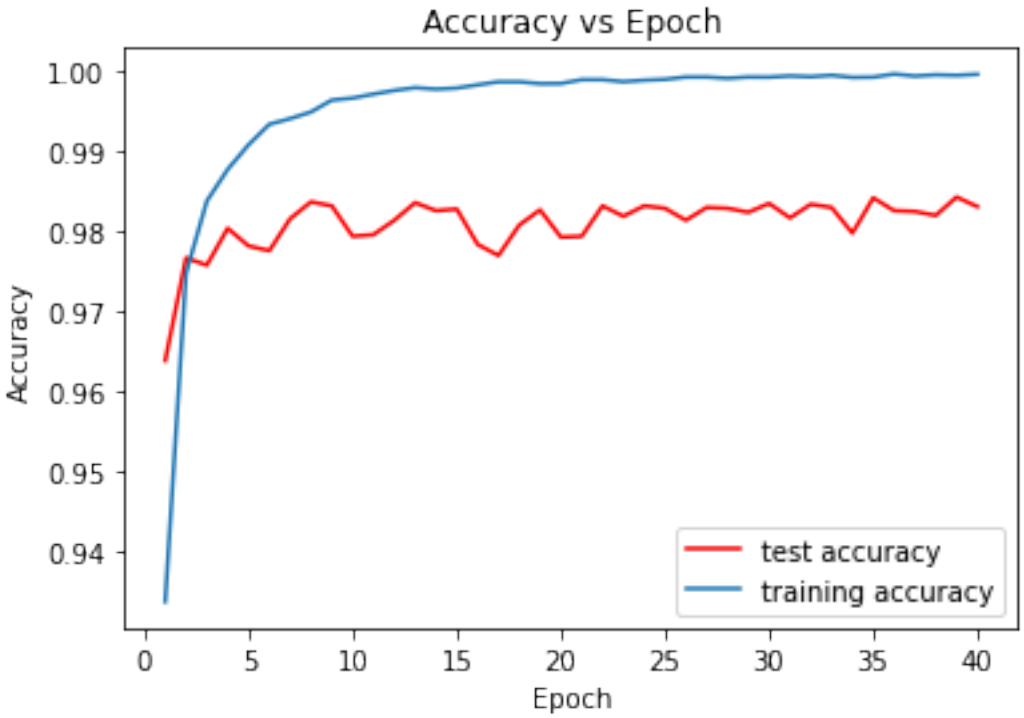
   ConvNets consists of Convolutional, Maxpooling and Flattening layers which in the end feed data to dense connected layers with respective activations and last layer uses activation such as Sigmoid or Softmax based on type of classification problem.

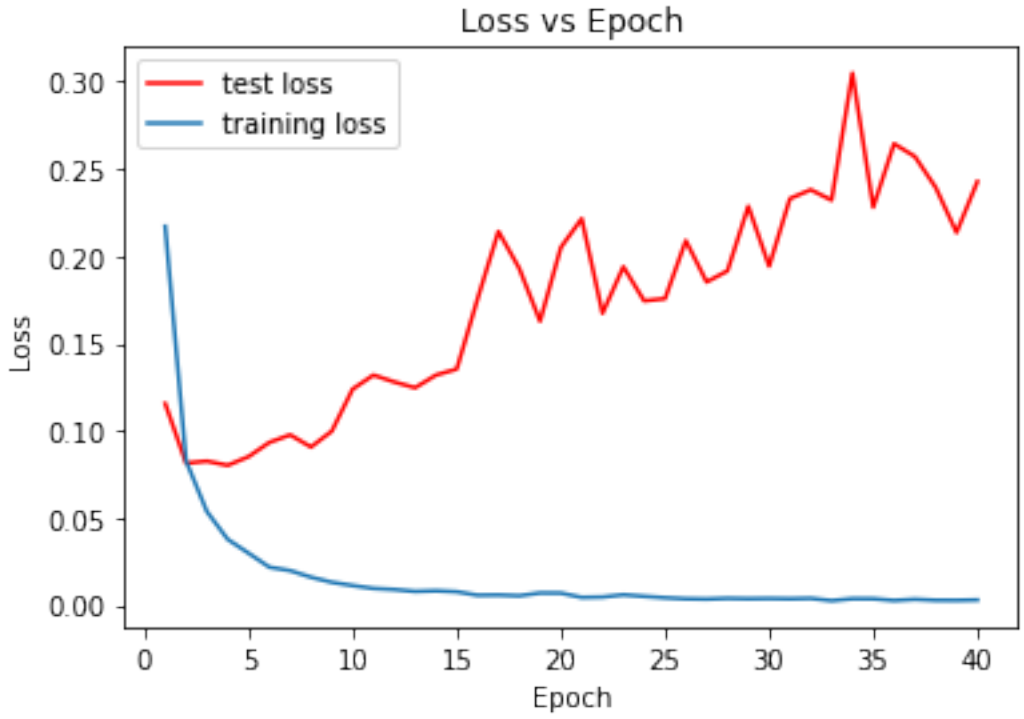
   Maxpooling layer as it's name suggest moves a kernel over the image but it do not convolve, rather picks maximum value present in the grid and outputs it. Strides can be manipulated to define the extent of dimension reduction. For example, a stride of 2 will cause kernel to move 2 pixels ahead instead of normal 1 causing final output to be of half the size of input in terms of dimensions.

4. MNIST Dataset Implementation and comparison

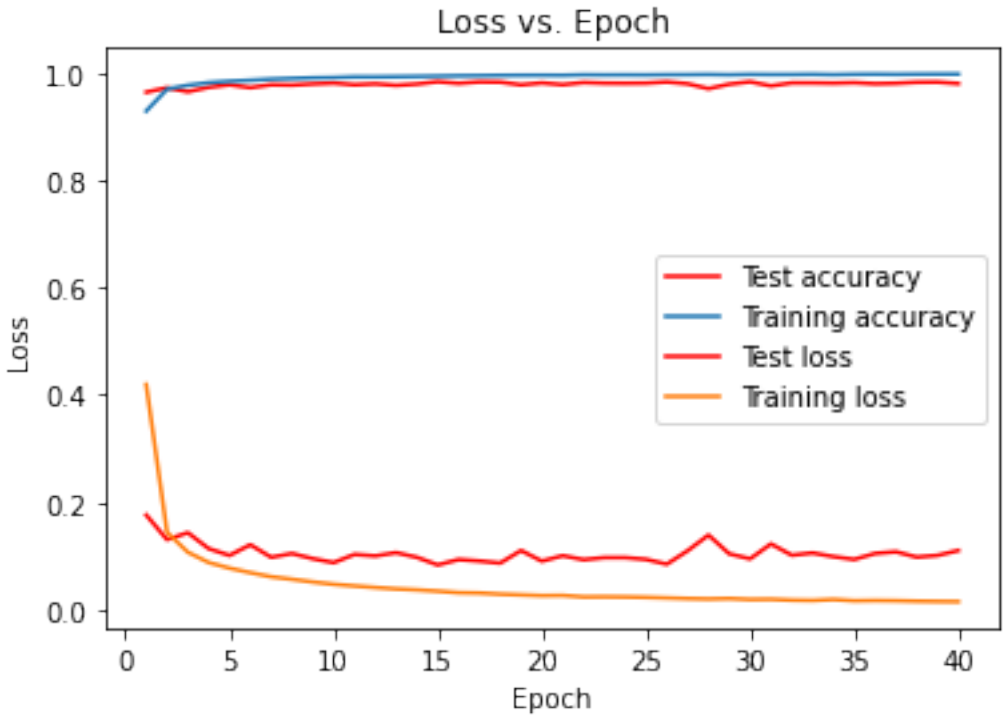
   MNIST dataset is considered as benchmark for testing Convolution models having 70000 grayscale images of (28,28) dimensions which we split into 60000 images for training and 10000 for testing.Three different models were tested. Common Parameters were MNIST data set, 2 512 output dense layers with ReLU activation and output layer with 10 nodes and Softmax activation. RMSprop was used as optimizer, categorical crossentropy loss and accuracy performance metric. Batch size was 128 with 40 epochs.

   (a) ANN without l2 regularization

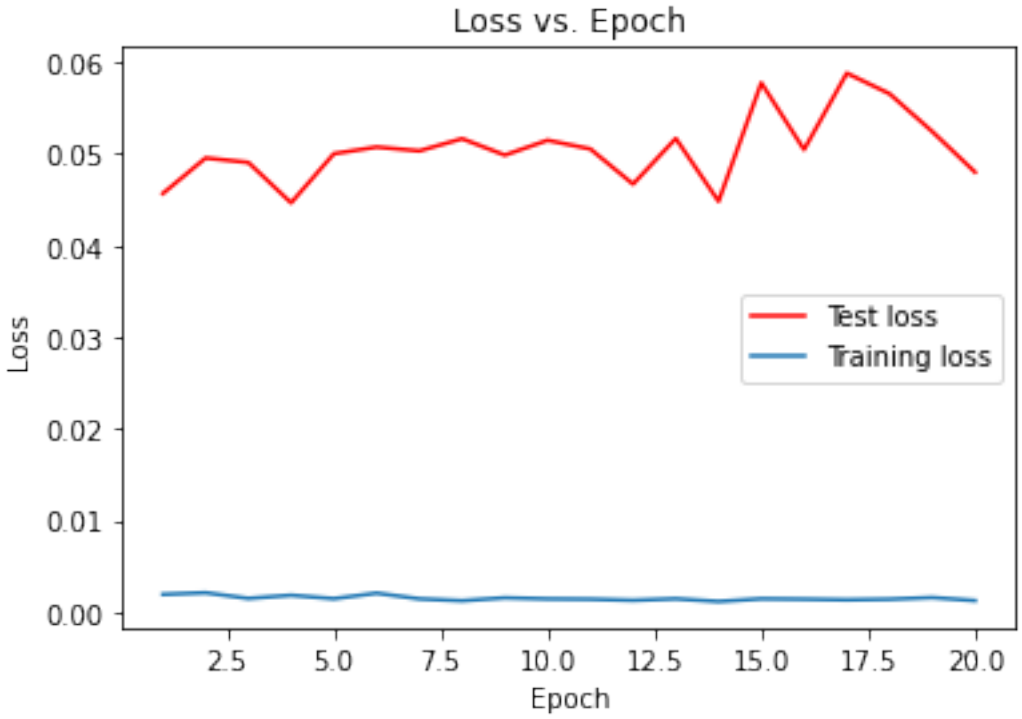


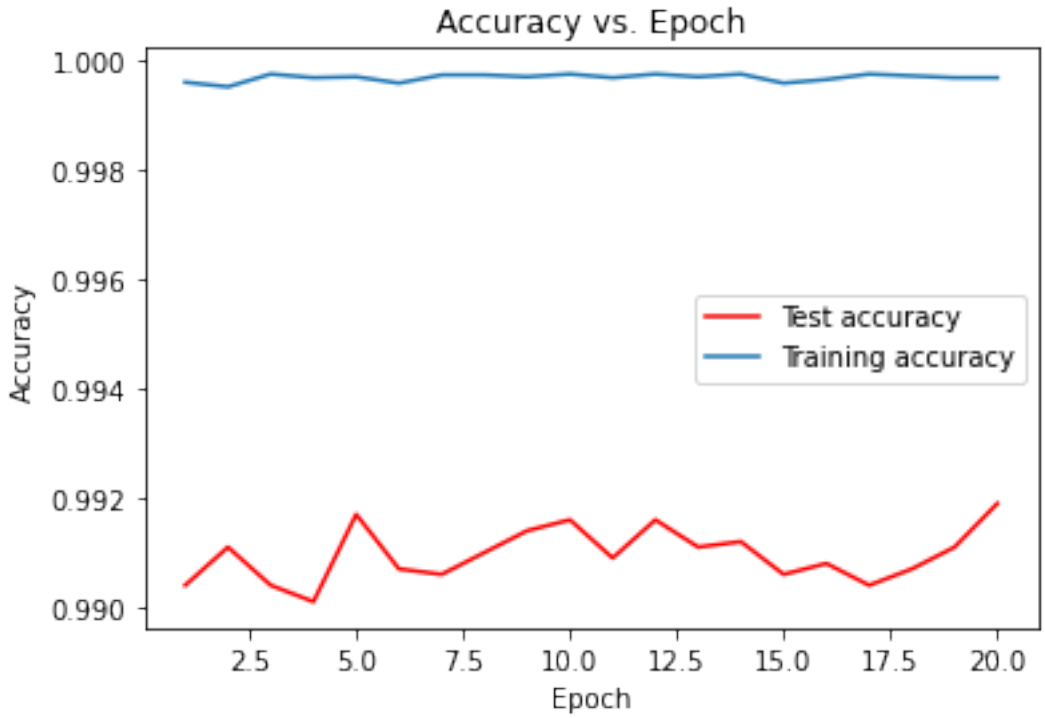


(b) ANN with l2 regularization



(c) CNN model with 2 convolution layers with first one with 24 layered output and second one with 48 channeled output, kernel size was (3,3) for both of them and 2 maxpool layers. Flattening layer and then a dense layer with 128 nodes and last one with 10 nodes for output. Model was run for 20 epochs.

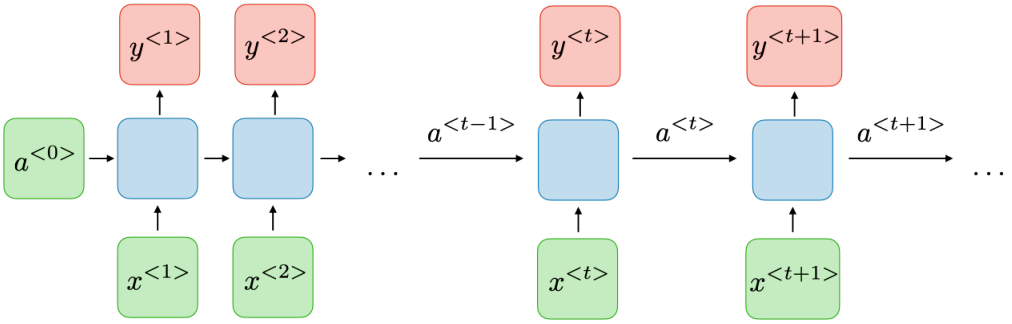




5. Results and Conclusions

   It is evident from the loss and accuracy graphs that NN w/o regularizers performed poorly and loss diverged, thus, increasing with iterations. ANN with regularization showed a more stable behavior. But with CNN model, loss was much lower and accuracy much higher at 0.05 and 99percent respectively.

8 Recurrent Neural Networks



1. Introduction

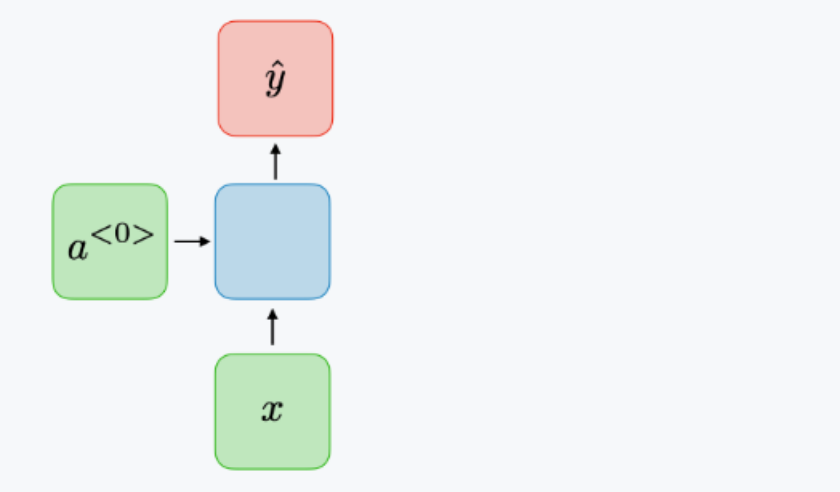
   When there's a sequence in data and past data have a causal relationship with future data then the need for data storage arises, so that previous experiences and sequences can be used to predict future outputs. It's usage scenarios can be from Natural Language Processing to Music Generation. But an architecture need to be defined as it's impossible to store each and every data provided in the past. If the sequence length to remember is shortened then long term dependencies are lost. Thus, requiring an architecture which keeps on important data and discards the rest providing flexibility to add and remove data at each time step. We'll be discussing such algorithms in this section. In the above diagram, a chunk of neural network, model, looks at some input $x^{<t>}$ and outputs a value $y^{<t>}$. A loop allows information to be passed from one step of the network to the next.

2. Application areas

   (a) one to one

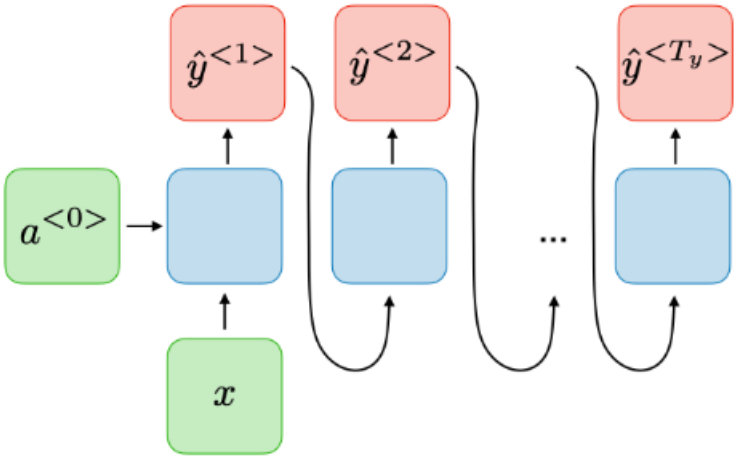
Table 1: Applications

Type of RNN	Example#2
one to one	Traditional NN
one to many	Music Generation
many to one	Sentiment Analysis
many to many	Name entity recognition



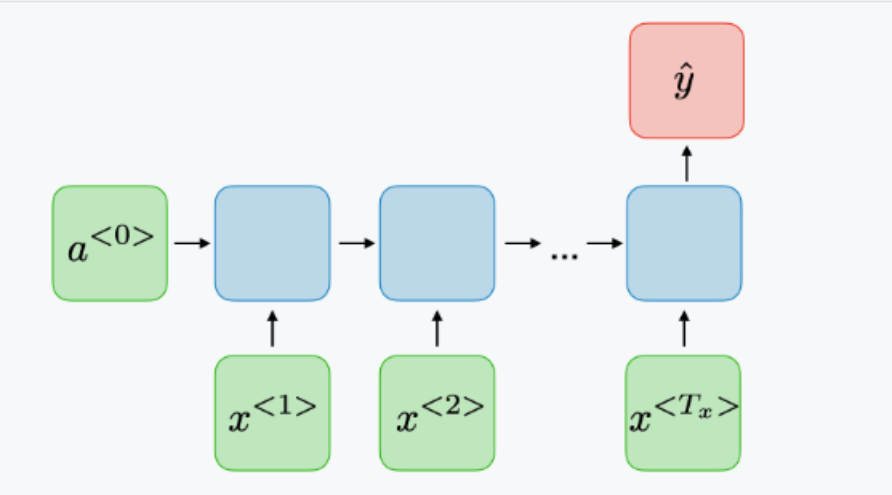
source:[https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks]

(b) one to many



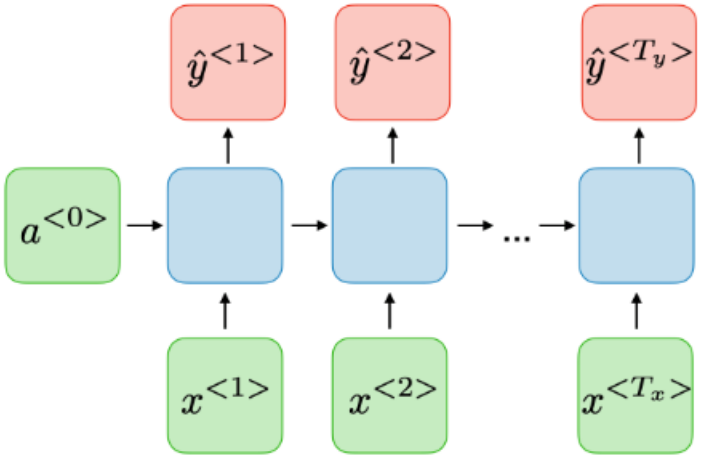
source:https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

(c) many to one



source:https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

(d) many to many

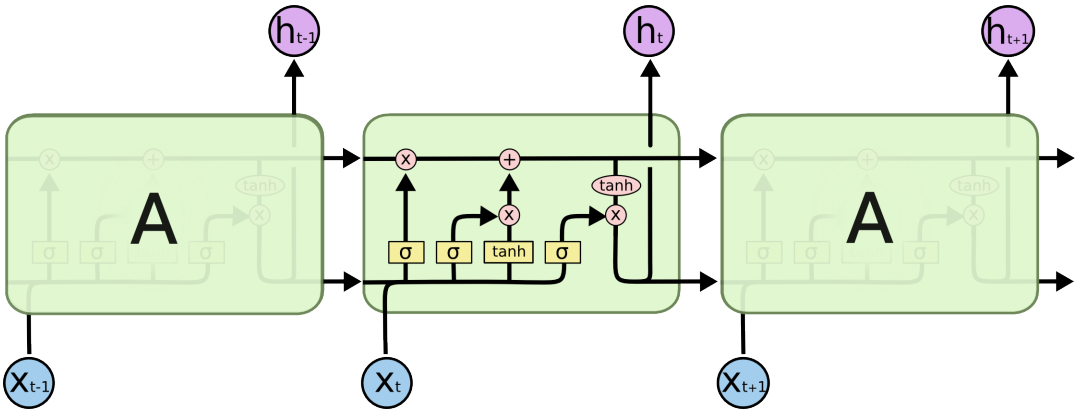


source:https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

3. Architecture

Several architectures have been proposed. Popular ones are listed below,

(a) LSTM



https://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four. The key to LSTMs is the cell state which is kind of like a conveyor belt.

It runs straight down the entire chain, with only some minor linear interactions making it easy for information to flow along it. LSTMs have the ability to remove or add information through regulated structures called Gates.

i. Forget Gate
Information that is no longer considered useful by the model to acquire understanding of the relationship is discarded.

ii. Input Gate
It's responsible for addition of information to the cell state. Sigmoid function determines what information is to be added what is not.

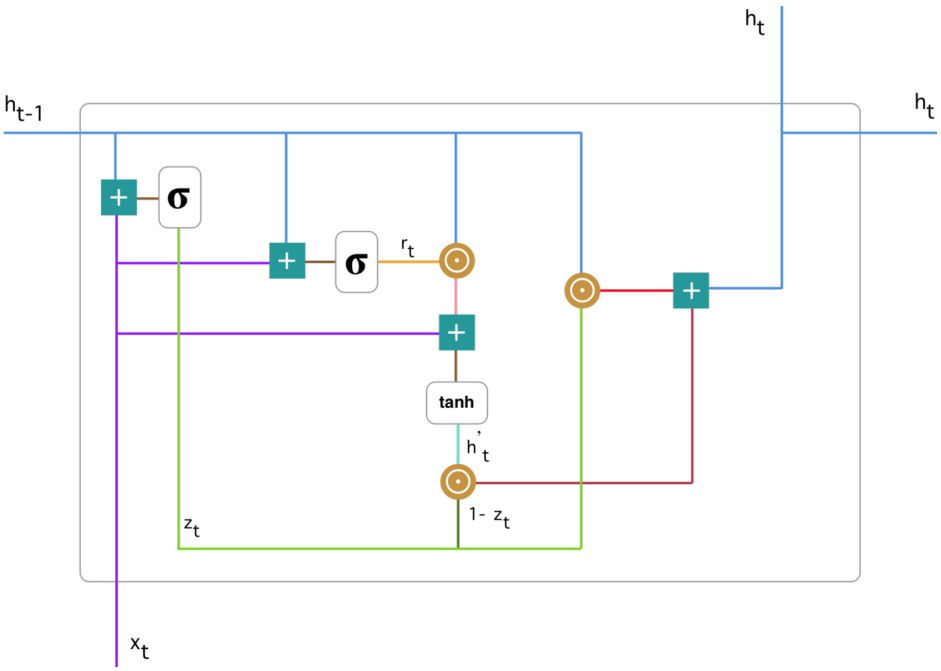
iii. Output Gate function of output gate is as follows:

A. Creating a vector after applying the tanh function to the cell state, thereby scaling the values to the range -1 to +1.

B. Make a filter such that it can regulate the values that need to be output from the vector created in previous step.

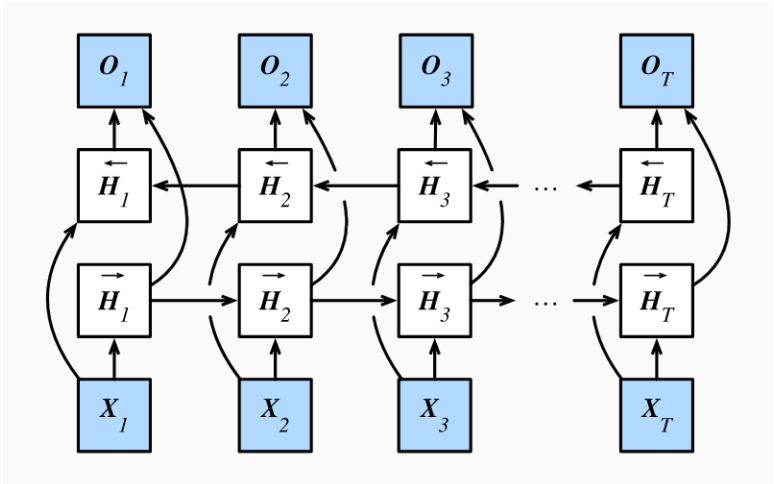
C. Multiplying the value of this regulatory filter to the vector and sending it out as an output along with sending it to the hidden state of the next cell.

(b) GRU



https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be
GRU (Gated Recurrent Unit) aims to solve the vanishing gradient problem which we face in a standard recurrent neural network. GRU can also be considered as a variation on the LSTM because both are designed similarly with the exception that architecture of GRU is simpler. GRU uses update gate and reset gatewhic decide what information should be passed to the output. Information can be stored by them for a long time without washing it through time or diluting information which is relevant to prediction.

(c) Bi-Directional RNN



https://d2l.ai/chapter$_r ecurrent - modern/bi - rnn.html$

Bidirectional RNNs add a hidden layer that passes information in a backward direction to more flexibly process the information. main distinction is that now these forward and backward recursion are devoid of such easily accessible interpretation and treated as generic functions.

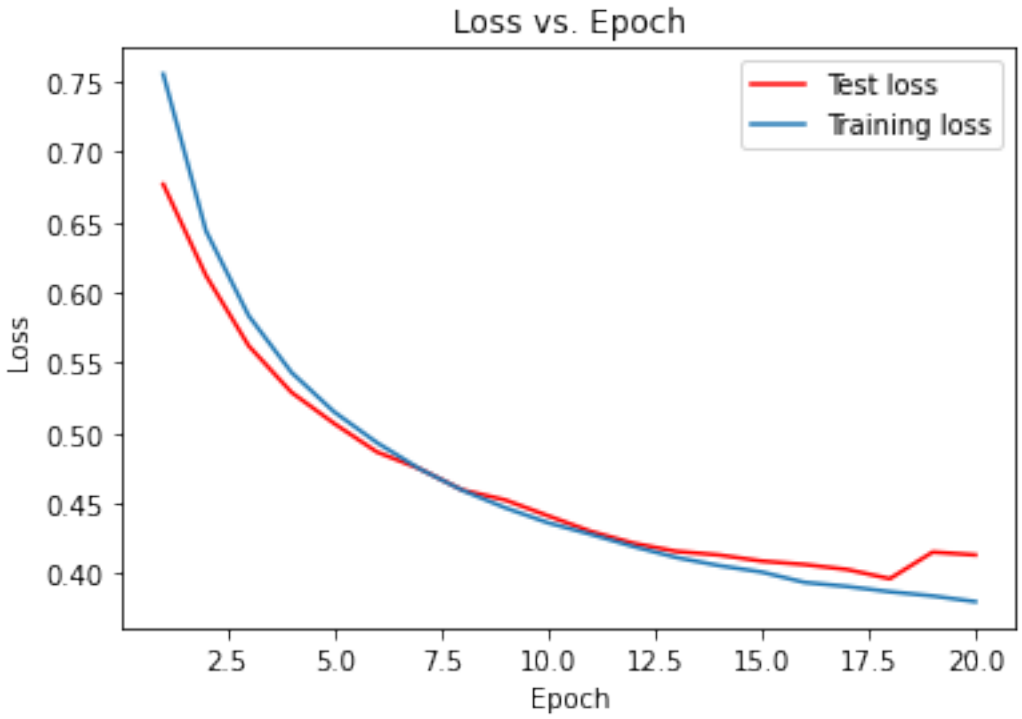
4. Sentiment Analysis with RNNs

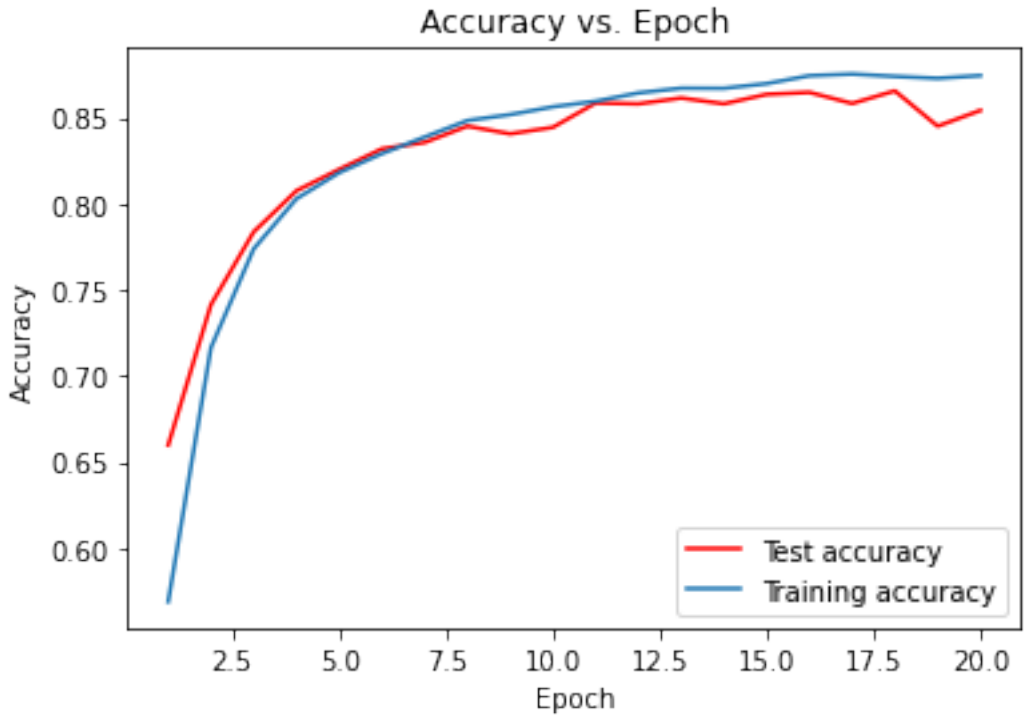
   Various models with variations were tried and results are shared in this section. With each model, it's parameters will be shared. Embedding is used across the models because it decreases the feature space by making relations between similar words in embedding space, l2 regularization with λ=0.01 and Sigmoid is used for final layer activation for obvious reasons. other parameters were,

   optimizer = 'RMSprop', loss='binary$_c rossentropy'$

   (a) Sequential model

   Embedding space of dimension=8, metrics = 'accuracy'

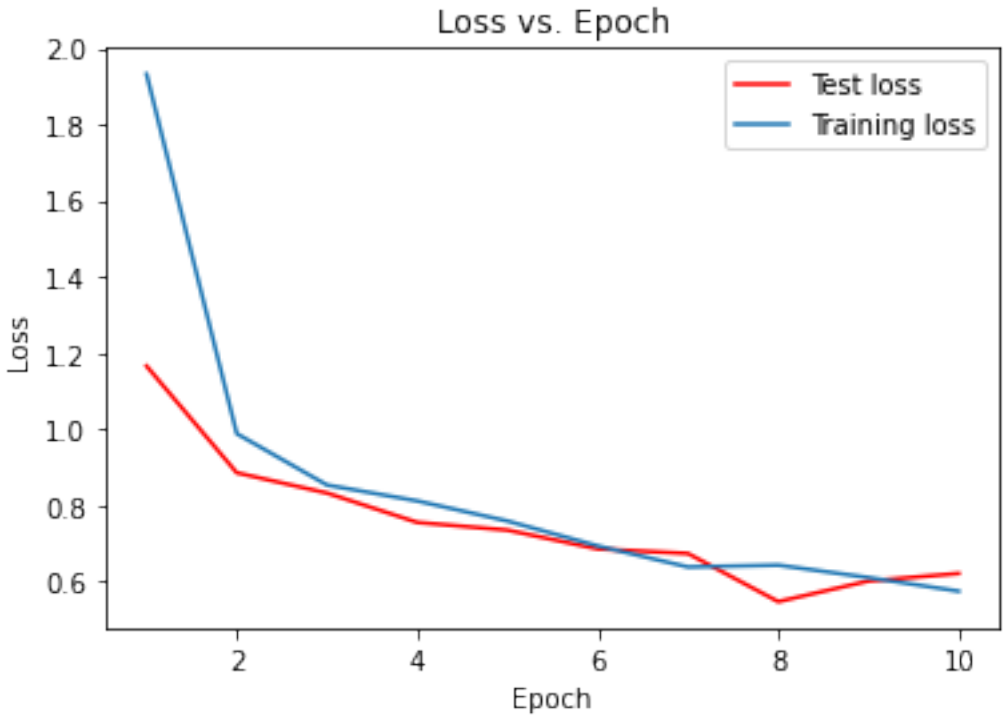


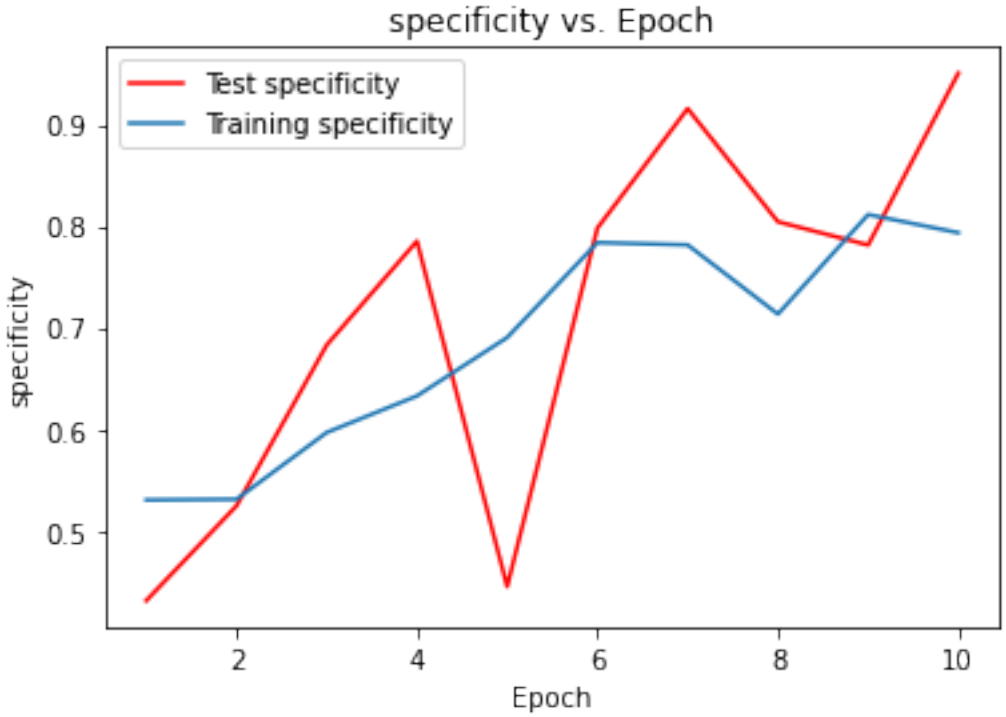



```
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_9 (Embedding)      (None, 250, 8)            80000
_________________________________________________________________
flatten_2 (Flatten)          (None, 2000)              0
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 2001
=================================================================
Total params: 82,001
Trainable params: 82,001
Non-trainable params: 0
_________________________________________________________________
```

```
loss: 0.3798 - accuracy: 0.8744 - val_loss: 0.4130 - val_accuracy: 0.8542
```

(b) Simple RNN
Embedding space of dimension=32, metrics = sensitivity, specificity

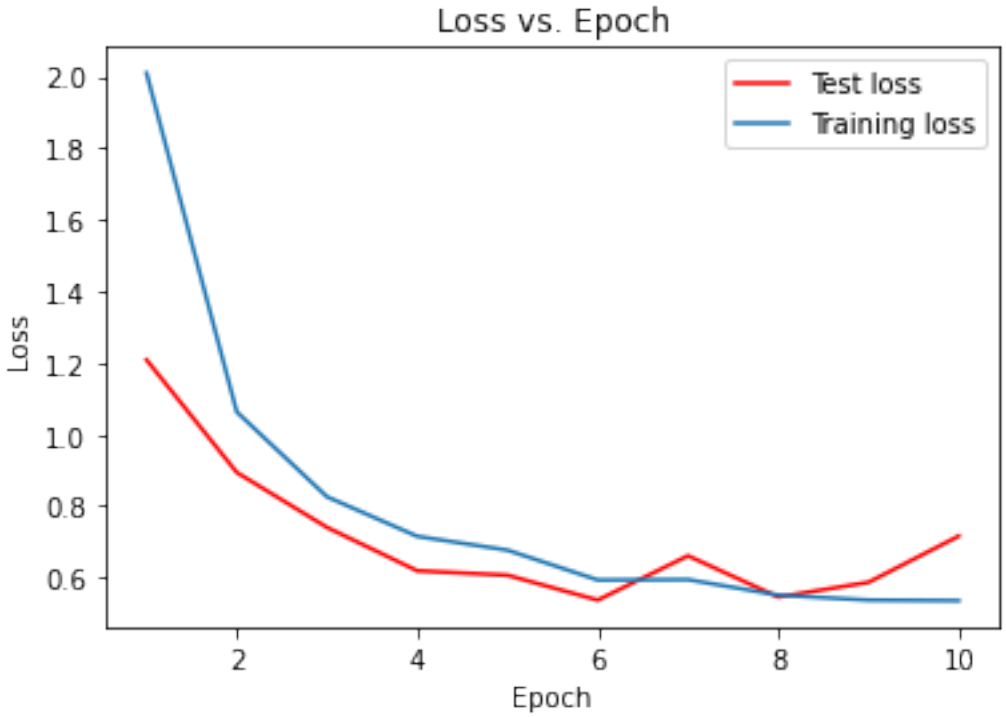


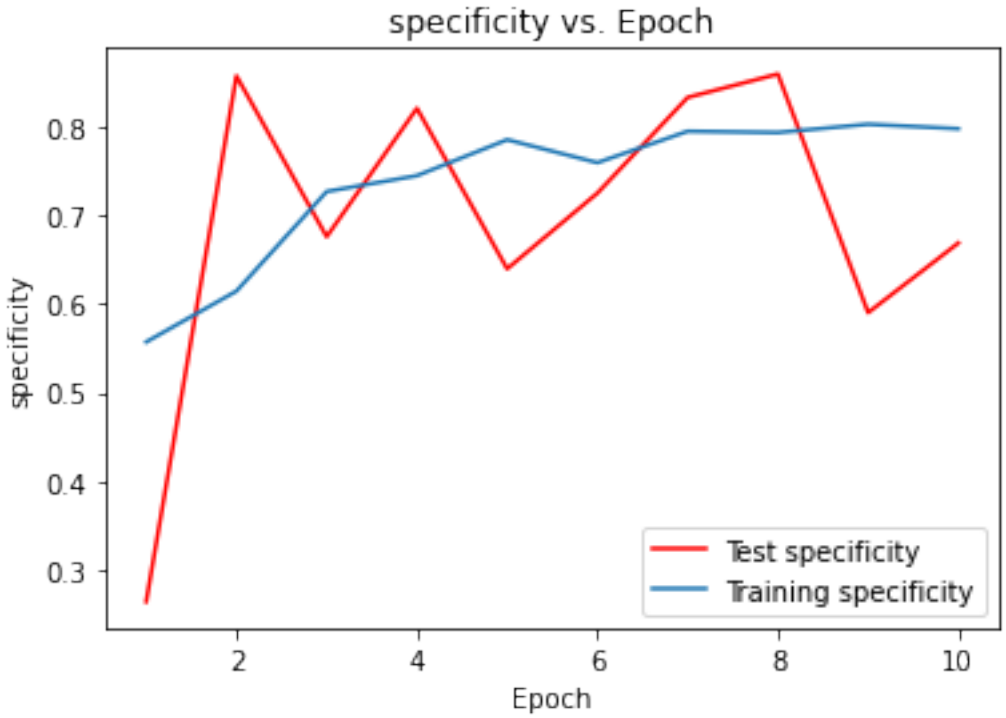



```
Model: "sequential_11"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_11 (Embedding)     (None, None, 32)          320000
_________________________________________________________________
simple_rnn_6 (SimpleRNN)     (None, 32)                2080
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 33
=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
_________________________________________________________________
```

```
loss: 0.5744 - sensitivity: 0.7832 - specificity: 0.7935 - val_loss: 0.6208 - val_sensitivity: 0.5328 - val_specificity: 0.9506
```

(c) GRU without Recurrent Dropout
Embedding space of dimension=32, metrics = sensitivity, specificity



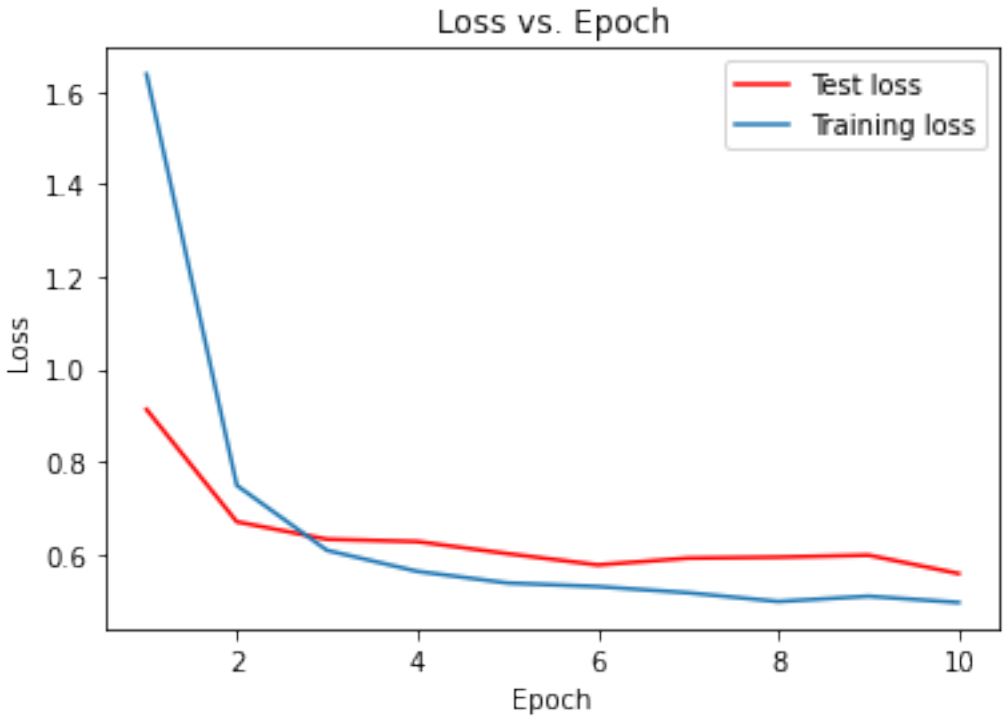


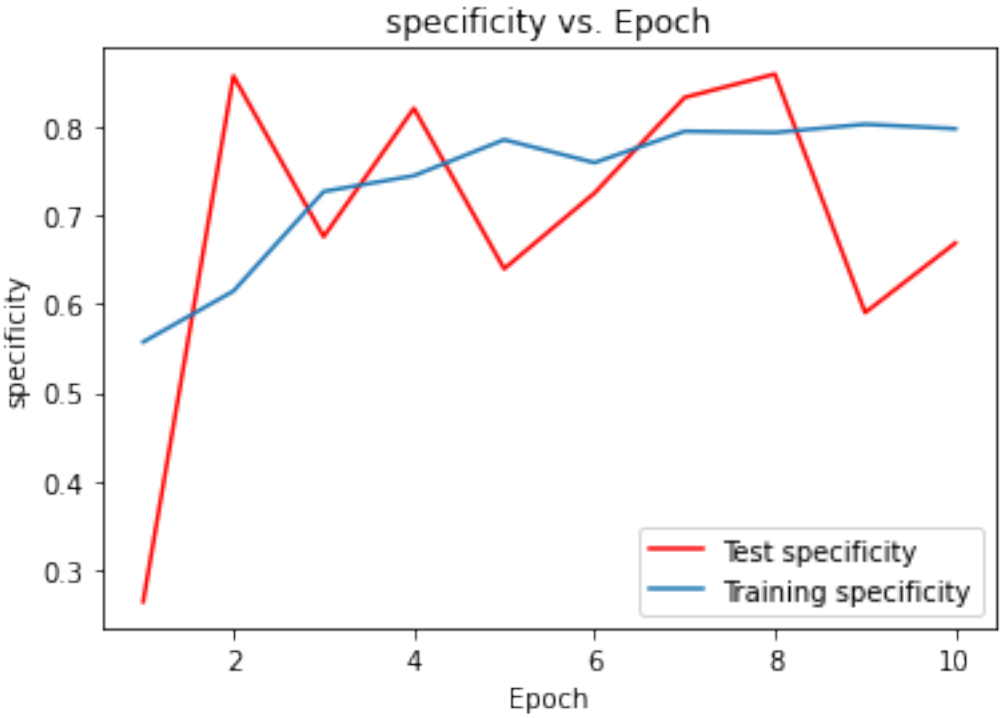
```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_12 (Embedding)     (None, None, 32)          320000
_________________________________________________________________
gru_4 (GRU)                  (None, 32)                6240
_________________________________________________________________
dense_12 (Dense)             (None, 1)                 33
=================================================================
Total params: 326,273
Trainable params: 326,273
Non-trainable params: 0
_________________________________________________________________
```

```
loss: 0.5357 - sensitivity: 0.8350 - specificity: 0.7992 - val_loss: 0.7162 - val_sensitivity: 0.9925 - val_specificity: 0.3687
```

(d) GRU with Recurrent Dropout

Embedding space of dimension=32, metrics = sensitivity, specificity, Dropout =0.5 and Recurrent Dropout=0.1

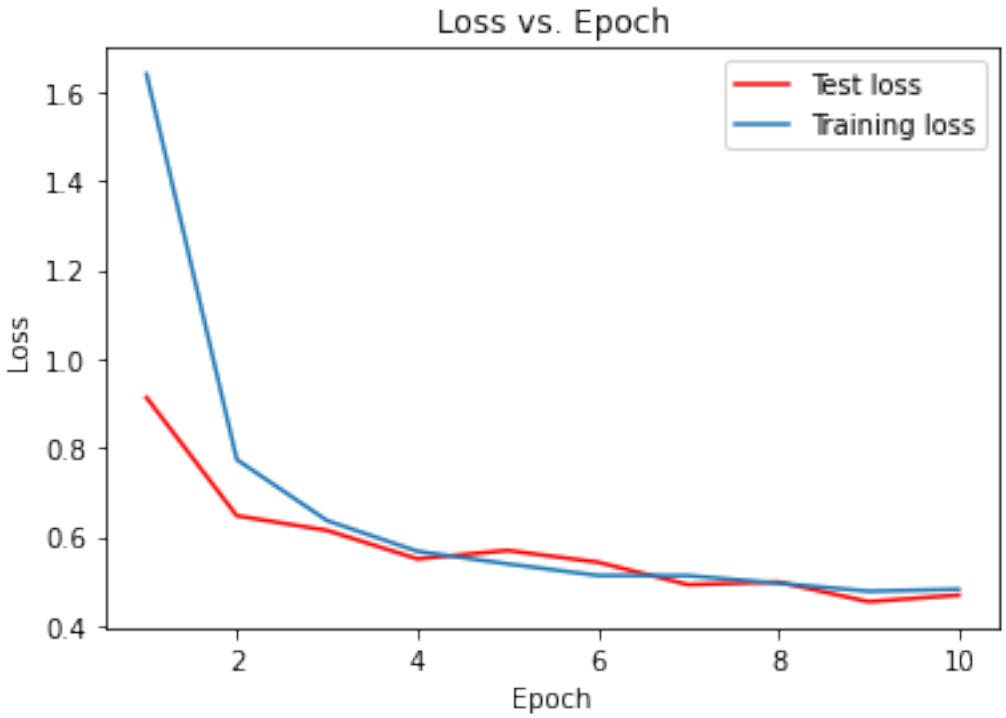


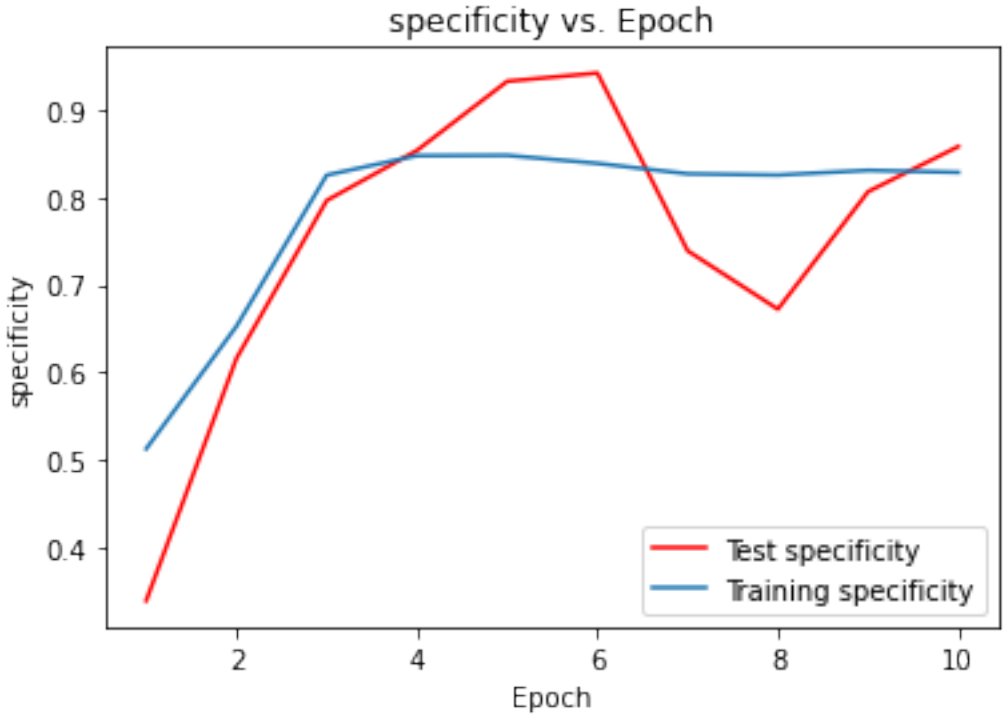



```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_13 (Embedding)     (None, None, 32)          320000
_________________________________________________________________
gru_5 (GRU)                  (None, 32)                6240
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 33
=================================================================
Total params: 326,273
Trainable params: 326,273
Non-trainable params: 0
_________________________________________________________________
```

```
loss: 0.4953 - sensitivity: 0.8242 - specificity: 0.7983 - val_loss: 0.5578 - val_sensitivity: 0.8654 - val_specificity: 0.6691
```

(e) Bidirectional RNN
Embedding space of dimension=32, metrics = sensitivity, specificity, Dropout =0.3 and Recurrent Dropout=0.1



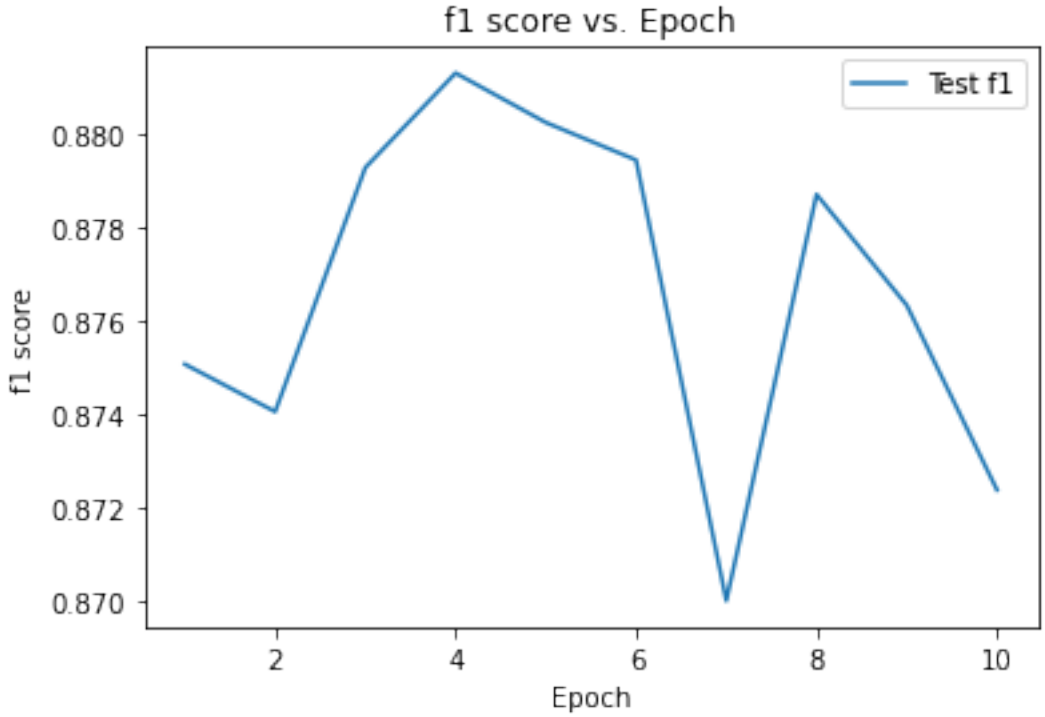



```
Model: "sequential_15"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_15 (Embedding)     (None, None, 32)          320000
_________________________________________________________________
bidirectional_2 (Bidirection (None, 64)                12480
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 65
=================================================================
Total params: 332,545
Trainable params: 332,545
Non-trainable params: 0
_________________________________________________________________
```

```
loss: 0.4818 - sensitivity: 0.8153 - specificity: 0.8283 - val_loss: 0.4686 - val_sensitivity: 0.8040 - val_specificity: 0.8578
```

(f) Bidirectional with f1 Score

Embedding space of dimension=32, metrics = F1 Score, specificity, Dropout =0.3 and Recurrent Dropout=0.02



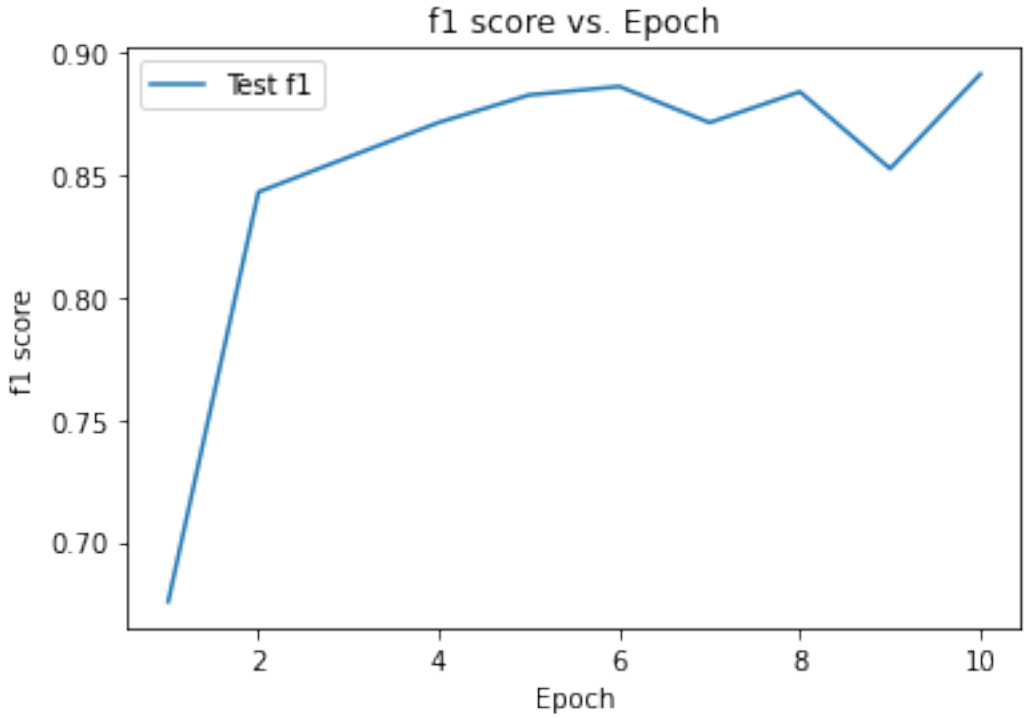
Layer (type)	Output Shape	Param #
embedding_10 (Embedding)	(None, None, 32)	320000
bidirectional_3 (Bidirection	(None, 64)	12672
dense_13 (Dense)	(None, 1)	65

Total params: 332,737
Trainable params: 332,737
Non-trainable params: 0

```
Epoch 10/10
98/98 [==============================] - 1s 14ms/step - loss: 0.3400 - get_f1: 0.9199 - val_loss: 0.4220 - val_get_f1: 0.8724
```

(g) Recurrent-CNN

Embedding space of dimension=32, metrics = F1 Score, specificity, Dropout =0.3 and Recurrent Dropout=0.02




```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_12 (Embedding)     (None, 350, 32)           320000
_________________________________________________________________
conv1d_5 (Conv1D)            (None, 350, 32)           3104
_________________________________________________________________
max_pooling1d_8 (MaxPooling1 (None, 175, 32)           0
_________________________________________________________________
bidirectional_5 (Bidirection (None, 64)                12672
_________________________________________________________________
dense_14 (Dense)             (None, 32)                2080
_________________________________________________________________
dropout_5 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_15 (Dense)             (None, 1)                 33
=================================================================
Total params: 337,889
Trainable params: 337,889
Non-trainable params: 0
_________________________________________________________________
None
```

```
Epoch 10/10
98/98 [==============================] - 120s 1s/step - loss: 0.3414 - get_f1: 0.9144 - val_loss: 0.3953 - val_get_f1: 0.8914
```

5. Results and Analysis

   Out of all the models, Convolutional Bi-Directional model produced the best combination of Specificity and Sensitivity which is represented by f1 score, thus it proves to be the best of all the models. Specificity and Sensitivity, in turn f1 score, were chosen because of the class imbalance between Positive and Negative reviews, thus, accuracy wouldn't have given a clear picture.

9 References

1. https://stanford.edu/ shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks
2. https://d2l.ai/chapter$_r ecurrent-modern/bi-rnn.htmlhttps$: $//tex.stackexchange.com/questions/364520/a-package-to-draw-neural-network-kernels-logic$

3. https://alexlenail.me/NN-SVG/LeNet.html
4. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53: :text=A

5. https://www.maths.tcd.ie/ dwilkins/LaTeXPrimer/Calculus.html

6.