A. Research scenario

Discrepancies in patient selection, lack of representative samples, specific treatment conditions can have a controlling effect on the outcomes of clinical trials. Further, randomized trials can be very expensive to run. Thus, regular assessment of adverse effects and efficacy of drugs even after they have been authorized is extremely crucial.

B. Research Question

The goal of this project is to analyze drug reviews and categorize them as ineffective, marginally effective, moderately effective, considerably effective or highly effective. This information can be used by:

1. Pharmaceutical companies- to take public opinion into consideration, increase revenue by adopting required marketing strategies based on the reviews and ensuring safety of individuals from any harmful side effects of the drugs.
2. Patients- to help them assess efficiency of various drugs and decide which one to go with.
3. Clinicians- to ensure patient safety, assess drug reactions, complications and effectiveness before prescribing them to patients.

C. Data description

Name: Drug review Dataset

Source:UCI Machine Learning Repository

Dimensions: 6 columns/features and 161297 rows/tuples. For the purpose of this project, I will be using 2 attributes: drug review and its respective rating.

Attribute description:

Sr No.	Attribute	Description
1	drugName	Name of the drug
2	condition	Name of the condition
3	review	Patient review
4	rating	10 star patient rating
5	date	Date of review entry
6	usefulcount	Number of users who found the review useful

D. Data preprocessing

1. Null values: These were found in 2 columns. Rows with null values were dropped.
2. Subset data: I will be using the top 5000 reviews for the purpose of this project.
3. Text Cleaning: I converted the reviews to lowercase, removed special characters and numbers. I filtered out encoded characters, mentions, urls, hashtags, emojis and additional white spaces. I removed commonly occurring stopwords from the data. These include articles, conjunction, prepositions and pronouns. I performed stemming and lemmitization to convert words to their root/base form.
4. Bag of Words model: I tokenized the documents/ reviews into words and and computed the frequency of each token using CountVectorizer().

```
column_sum=bow_matrix.sum(axis=1)
column_sum.sort_values()  #occurence of each word in the documents
```

```
lightend         1
lumpectomi       1
lunchbox         1
lunchtim         1
lusciou          1
               ...
year          2044
month         2075
ben           2405
take          2974
day           3037
Length: 8105, dtype: int64
```

For instance, out of 8105 terms or tokens, “day” occurred highest number of times.
By computing a row sum across all reviews,

```
[ ]  row_sum=bow_matrix.sum(axis=0)
     row_sum.sort_values()  # words in each document/review
```

```
Review 1903      1
Review 3272      1
Review 4322      1
Review 982       1
Review 2650      1
                ...
Review 1633    120
Review 2331    129
Review 4810    161
Review 1610    166
Review 55      186
Length: 5000, dtype: int64
```

We can interpret that the 55th review had highest number of terms.

5. Normalizing term frequencies: This was done using TF/IDF I.e. term frequency/inverse document frequency in order to evaluate the relative importance of each term in the reviews. TF measures the frequency of the term and IDF measures how rare the term is in the documents.

```
difuctli      0.100807
flactuat      0.100807
virtu         0.103872
freqent       0.103872
enlong        0.103872
                ...
month       142.208067
ben         153.066827
work        158.946037
take        177.921730
day         179.525255
Length: 8105, dtype: float64
```

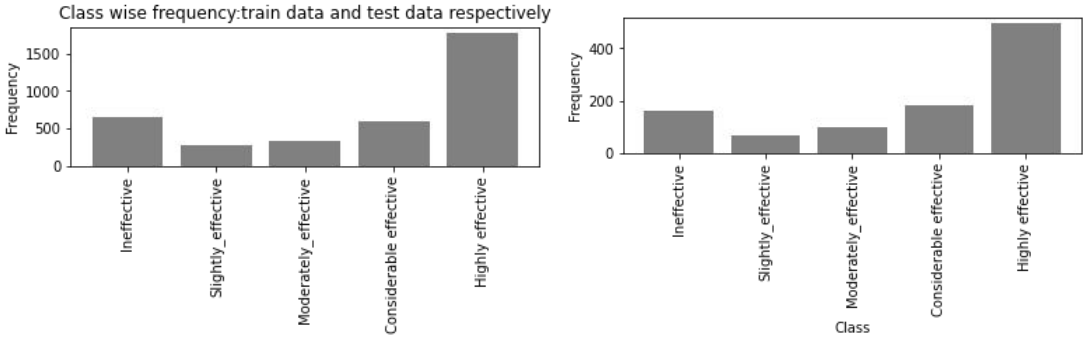
As seen before, “day” had highest importance in the reviews.

6. Discretizing data: The rating column was discretized into 5 unique categories of ineffective, marginally effective, moderately effective, considerably effective or highly effective.
7. One hot encoding: I one hot encoded the dependent/response variable “rating” to get k-1 dummy variables, where k= number of classes=5.
8. Splitting the data: I split the data into train, test and validation sets.

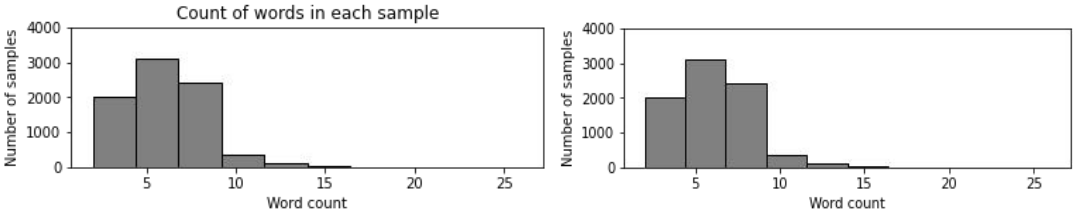
I used a 80-20% split for training and test sets. 10% from the training data was allocated as the validation set.

E. Data Visualization

I performed certain visualization to get more insights into the data.



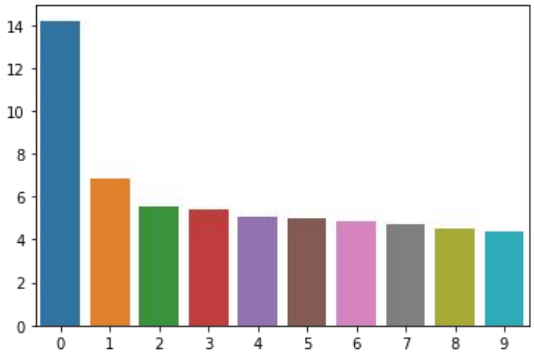
A higher proportion of drug reviews were highly effective and least were slightly effective, in train and test sets.



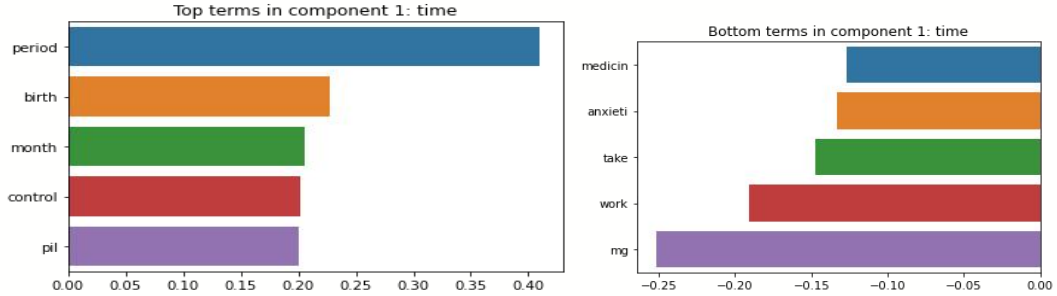
Most words occurred between 4 to 7 times in the reviews, in train and test sets. The train and test data had a similar right skewed distribution for occurrence of terms in the reviews.

I performed latent semantic analysis to find clusters of similar words. This gives 3 matrices (term-topic matrix,topic-topic matrix and topic-document matrix).The figure below shows 10 topics/ components andtop 10 words associated with each topic.Further, I tried to see
how much each component/topic explains the variance in the data

```
Topic 0:  ['day', 'take', 'ben', 'month', 'work', 'efect', 'wek', 'year', 'pain', 'period']
Topic 1:  ['period', 'birth', 'month', 'control', 'pil', 'cramp', 'gain', 'weight', 'blede', 'acn']
Topic 2:  ['pain', 'cramp', 'day', 'insert', 'back', 'hour', 'relief', 'period', 'got', 'blede']
Topic 3:  ['work', 'efect', 'side', 'great', 'wel', 'pain', 'use', 'medicin', 'birth', 'control']
Topic 4:  ['pain', 'help', 'year', 'life', 'gain', 'ben', 'weight', 'depres', 'control', 'anxieti']
Topic 5:  ['efect', 'side', 'pain', 'day', 'headach', 'wek', 'nausea', 'weight', 'lb', 'experienc']
Topic 6:  ['period', 'anxieti', 'mg', 'work', 'panic', 'control', 'depres', 'pil', 'atack', 'birth']
Topic 7:  ['weight', 'lb', 'gain', 'work', 'lost', 'pound', 'eat', 'pain', 'lose', 'wek']
Topic 8:  ['pil', 'help', 'take', 'slep', 'control', 'birth', 'night', 'hour', 'medicin', 'make']
Topic 9:  ['mg', 'take', 'control', 'birth', 'ben', 'pil', 'acn', 'night', 'year', 'month']
```

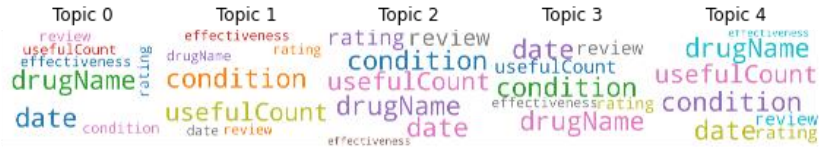


Topic 1 explains highest variance in data. It seems to be associated with time or frequency of drug dosage.



"Period" and "mg" were most and least important terms in topic 1.

I also performed Latent Dirchilet Allocation for topic modelling and visualized its results using a word cloud. LDA gives each term and its associated weightage in the reviews.



The figure above shows a word cloud for each of the 10 topics and top 10 words associated with each topic, computed using LDA.

F. Classification using Neural Networks

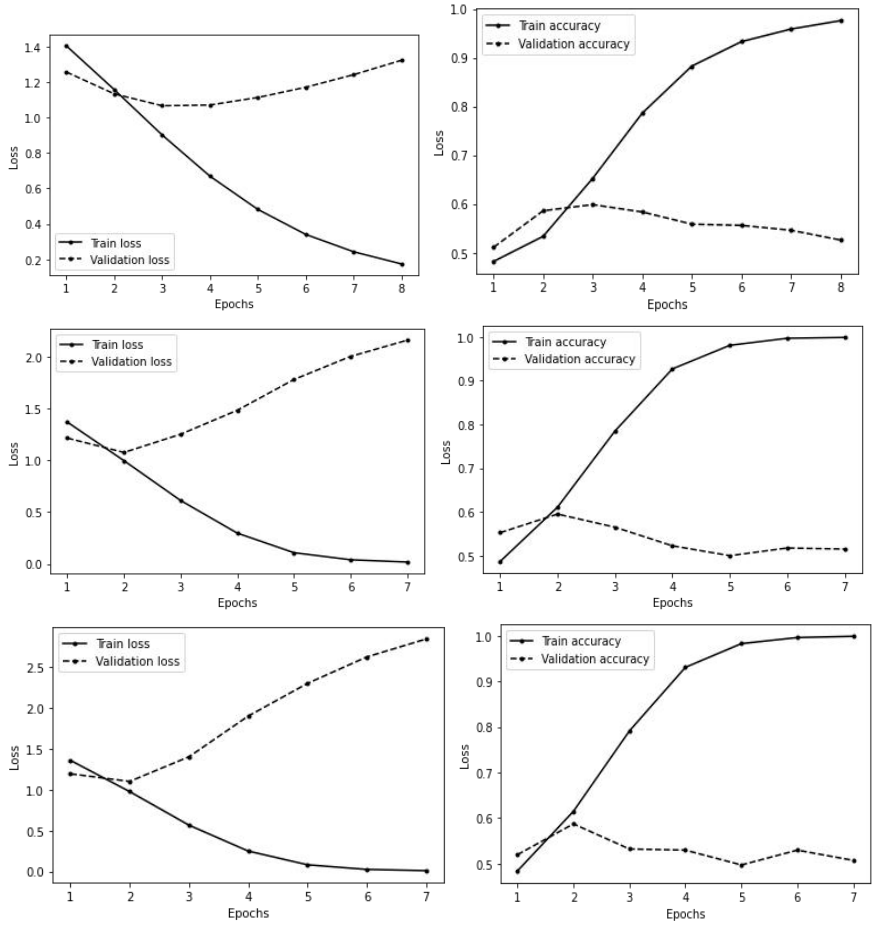
Multi Layer Perceptron

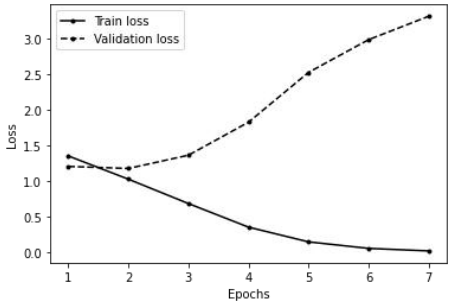
MLP contains one input layer, multiple hidden layers and one output layer. Layers closer to the input are lower layers and layers closer to output are upper layers. Every layer Is fully connected to the next layer. It helps overcome disadvantages of simple perceptron such as the inability to solve non linear problems. Its training involves a forward pass where weighted sum of inputs is computed at each hidden layer (W.X where W is weight vector and X is input vector) and an appropriate activation function is applied. In the backward pass, gradient of the network's error is computed with respect to every connection weight across all layers (from output to input layer). Back propagation computes how much each parameter contributed towards final loss. It uses chain rule to assess how much of these contributions came from underlying layers. It then tweaks weights using gradient descent.

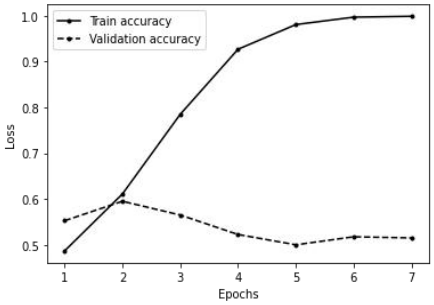
1.MLP with one hidden layer

To start with, I built a simple MLP with one hidden layer consisting of 100 neuron. Relu activation function was used in this layer. Softmax function was used in the output layer (since this is multi class classification) with 5 neurons since there are 5 unique classes. I built, compiled, fit and evaluated the model on test data. I used a model checkpoint callback to save model at end of every epoch and early stopping to halt when performance does not improve for n (=patience) epochs. Adam optimizer was used to weak model parameters. I used categorical cross entropy loss since there are multiple classes and data is already encoded.

2.MLP with 2 hidden layers
3.MLP with 3 hidden layers
4.MLP with 4 hidden layers
I added an additional hidden layer with 100 neurons to the previous model. For the next 2 models, I added one and two more hidden layers respectively (each with 100 neurons) to assess the difference in performance. The learning curves below show training and validation loss, accuracy across all epochs for MLP with 1, 2, 3 and 4 layers respectively.







We can observe that as number of layers is increased, the difference in training and validation metrics also increases. This could indicate overfitting.

Model	Accuracy with early stopping	Loss with early stopping	Accuracy without early stopping	Loss without early stopping
MLP (1 layer)	0.5500	0.1877	0.5160	1.6120
MLP(2 layers)	0.5590	0.1876	0.5030	2.6396
MLP(3 layers)	0.4910	1.2790	0.4890	3.7317
MLP(4 layers)	0.5330	1.2552	0.4820	5.3042

Multi Layer Perceptron with 2 layers gives highest accuracy and lowest loss so far.It can also be observed that accuracy of model on test data is higher when early stopping is used.

5.Experimenting with different activation functions
['tanh','elu','relu','sigmoid','softplus']

- Sigmoid:

y= 1/ (1+ e^-z) where z= weighted sum of inputs
It has a nice derivative and a non zero gradient which is beneficial for Gradient Descent. However it is saturating and not symmetrical. Its range is 0 to 1.

- Tanh:

y= (e^z - e^-z)/(e^z + e^-z)
It has a good derivative and is also symmetrical. Its range is -1 to +1.

- ReLu

y= z if z>=0
0, otherwise
In other words, y=max(0,z). It helps solve the problem of unstable gradients, it is non saturating and fast to compute. However, it has a “dying” problem which occurs when the neurons do not output anything other than 0 (since relu gives zero output for negative values). This can be solved by using leaky relu which introduces a small slope, alpha, to ensure that neurons never die.

- ELU or exponential relu

y= z if z>=0
alpha*(e^z - 1), otherwise
It has a non zero gradient for z<0 and thus avoid the dead neurons and vanishing gradient problem. It is faster and more efficient.

- Softmax

It calculates probability of each class.
y= e^z/ (sum(e^z)) for all classes
Class with the highest probability is considered as the actual class. It is evaluated using cross entropy loss, which considers two values: probability, p, of the class and target probability or ground truth, t. By minimizing CE, we are trying to push predicted class probability towards target probability.

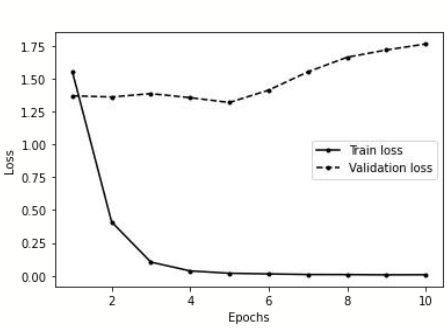
Activation	Loss (on 2 layer MLP)	Accuracy (on 2 layer MLP)
Tanh	1.1932	0.5470
Elu	1.1963	0.5454
Relu	1.2014	0.5530
Sigmoid	1.2098	0.5430
Softplus	1.1712	0.5590

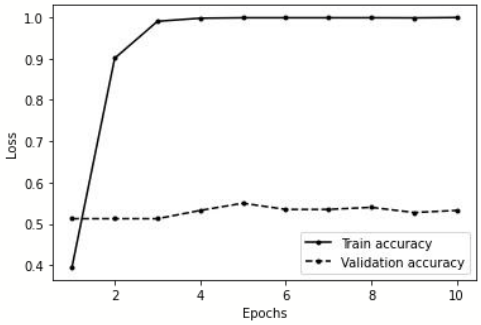
In general, performance was best for Relu activation function.

6.Batch Normalization
It is used to deal with the exploding and vanishing gradients problem. It normalizes inputs at each layer. It standardizes and scales layer inputs. It adds 4 parameters: mean and standard deviation calculated using moving average of mini batches, as well as alpha and beta which are learnt using backpropagation.

Model	Accuracy	Test Loss	Train loss	Validation loss
MLP Without Batch Normalization	0.5430	1.2014	0.0189	2.0996
MLP With Batch Normalization	0.5390	1.9010	0.0065	1.7940

Comparable accuracy and loss were observed with and without batch normalization on test data. Train and validation loss decreased when BN was used.





Further, we can see from the learning curve, that train and validation loss are relatively closer to each other, indicating lesser overfitting.

7.Learning rate scheduling
This is used to tweak learning rates during training.

- Piecewise constant scheduling

A contant learning rate is used for n epochs, then a lower learning rate is used for m epochs, and so on.

- Power scheduling

The learning rate is a function of the iteration. It drops after each step.The initial learning rate, number of steps and power c have to be defined.

- Exponential scheduling

The learning rate is reduced by a factor of 10 after a predefined number of steps.

- Performance scheduling

The error is checked after every N steps and learning rate is dropped by a factor of alpha when the error does not reduce.

- 1cycle scheduling

It first increases the learning rate and then decreases it during second half of training. It is dropped by a higher factor during the last few epochs.

Learning rate scheduler	Accuracy	Loss
Piecewise constant scheduling	0.5050	3.1784
Power scheduling	0.5100	1.9001
Exponential scheduling	0.5100	1..4607
Performance scheduling	0.5240	1.9199
1cycle scheduling	0.4990	2.0211

Best performance was achieved when performance learning rate scheduler was used.

8.Regularization

- L1 regularization

L1 or lasso regularization adds a regularization term, alpha*sum(|W|), to the cost function. It pushes the weight to zero if the feature is not useful. It gives a sparse model since weight of unimportant features is set exactly to zero.

- L2 regularization

L2 or ridge regularization adds a regularization term, alpha*sum((W)^2), to the cost function. It pushes the weight towards zero if the feature is not useful, but never exactly zero.

- Elasticnet regularization

It is a combination of both l1 and l2 regularization. It adds an additional term ‘r’ to balance the effect of features with lesser importance.

- Maxnorm regularization

Instead of adding a regularization term, it constraints weights by rescaling them after each iteration= W.r/ ||W||

- Dropout regularization

A certain number of neurons are dropped at each iteration. At each step, every neuron has a probability ‘p’ of being dropped for that particular step. A hyperparameter dropout rate (between 0.1 to 0.5) has to be set. The goal is to make all neurons useful and not rely on only a few neurons.

Regularization	Accuracy	Loss
L1	0.4840	2.9052
L2	0.5130	2.5276

Elasticnet	0.5240	2.7915
Dropout (rate=0.2, 0.3, 0.5)	0.5060	1.3029
	0.4990	1.3467
	0.5260	1.2314
Alpha dropout	0.4950	1.3243

Best performance was achieved using dropout regularization with rate=0.5.

9.Weight initialization

- Glorot: Connection weights of each layer are initialized as 1/ fan(avg) for normal distribution and sqrt(3/fan(avg)) for uniform distribution. Here fan(avg)= (fan(in)+fan(out))/2 , where fan(in) and fan(out) are the number of input and output neurons respectively.
- Lecun: : Connection weights of each layer are initialized as 1/ fan(in) for normal distribution and sqrt(3/fan(in)) for uniform distribution.
- He : Connection weights of each layer are initialized as 2/ fan(in) for normal distribution and sqrt(3*2/fan(in)) for uniform distribution.

Initialization	Accuracy	Loss
He normal	0.5390	1.2143
He uniform	0.5360	1.2220
Lecun normal	0.5370	1.2589
Lecun uniform	0.5450	1.1655
Glorot normal	0.5430	1.2166
Glorot uniform	0.5270	1.4457

Highest accuracy and lowest loss was achieved by lecun_uniform initialization.

10.Optimizers

['SGD','rmsprop','adagrad','adam','nadam','adamax']

- SGD: Unlike Batch Gradient Descent, Stochastic Gradient Descent considers one training instance at a time and computes gradient for that instance only. It is faster and better for larger datasets since only one instance will be in the memory at a time. It is more irregular and is good for getting out of the local minima.
- Adagrad: It considers square of gradients and normalizes or scales the gradient vector. It ensures momentum in the correct direction. It helps point updates towards global minimum. However, it might reduce the learning rate too fast when there is a steep slope and too slow next to optima.
- Rmsprop: It overcomes adagrad's disadvantage and moves fast even when close to minima. It allows for faster convergence. It works in a similar way as adagard

but also includes decay rate, B. It keeps track of moving average of recent gradients.

- Adam: It is a combination of momentum and rmsprop. Thus it keeps track of moving average of past gradients as well as moving average of past squared gradients.
- Adamax: It is similar to adam but is more stable. It scales down updates by ‘s’, which is the maximum of time decayed gradients, unlike adam which normalises updates by sqrt(s).
- Nadam: It is a combination of adam and nesterov. It converges slightly faster than adam.

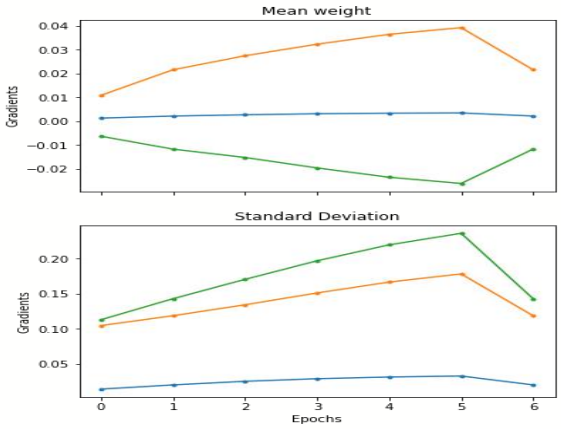
Optimizer	Accuracy	Loss
SGD	0.5510	1.2309
Adagrad	0.4760	1.3801
Rmsprop	0.5410	1.1962
Adam	1.3261	0.5330
Adamax	1.2338	0.5430
Nadam	1.2038	0.5454

Highest accuracy and lowest loss was achieved by Adam Optimizer.

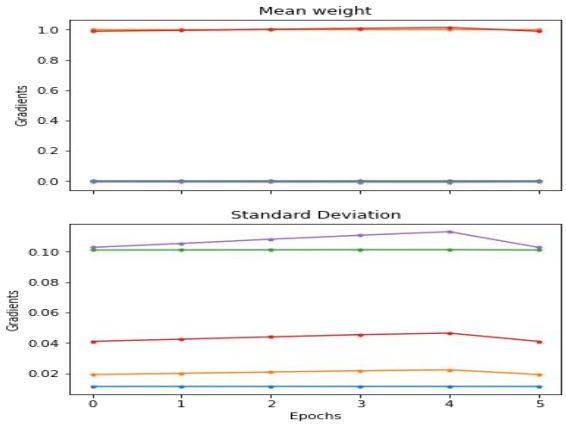
11.Gradient Clipping

It is a way of dealing with the exploding gradients issue. Here, if the gradients exceed a particular threshold, they are clipped off. I visualized the mean and standard deviation of gradients across all epochs:

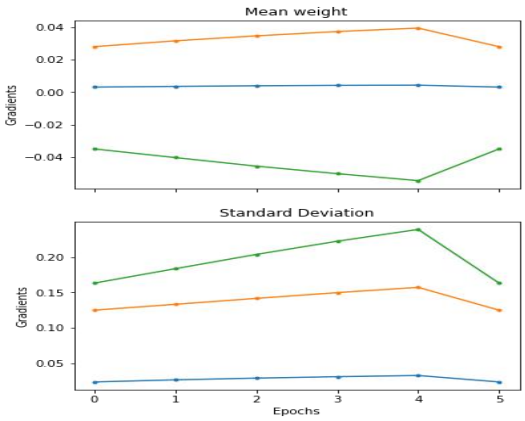
Without Gradient Clipping:



With Batch Normalization:



With Gradient Clipping:



I observed that gradients were most stable for batch normalization, while gradient clipping showed slight improvement as compared to the original model.

12.Number of neurons in each hidden layer

This is an important hyperparameter we have to set. Having too few neurons in the hidden layers will make it difficult to solve complex problems. On the other hand, having too many neurons could lead to overfitting.I experimented with different combinations of neurons: [30, 50,100,200] in both the hidden layers.

Number of neurons in layer 1	Number of neurons in layer 2	Accuracy	Loss
30	30	0.5510	1.2936
30	50	0.5454	1.2817
30	100	0.5350	1.2423
30	200	0.5410	1.2918
50	30	0.5370	1.2100
50	50	0.5340	1.3707
50	100	0.5210	1.4135
50	200	0.5110	1.3647
100	30	0.5500	1.2006
100	50	0.5440	1.5837
100	100	0.5420	1.2179

100	200	0.5360	1.6310
200	30	0.5320	1.2735
200	50	0.5390	1.2993
200	100	0.4950	1.3585
200	200	0.4960	1.3277

Highest accuracy and lowest loss was achieved for 100 neurons in layer 1 and 30 neurons in layer 2.

Recurrent Neural Networks

RNNs have connections pointing in the backward direction. At each time step t, a neuron receives input x(t) as well as output from the previous step y(t-1). Since output is a function of all inputs and outputs from previous steps, RNNs have a form a memory. They preserve state across time steps and are good at learning patterns. They can work on sequences of arbitrary length. Variations include sequence to sequence, sequence to vector (where only last/final output is considered), vector to sequence and encoder-decoder (sequence to vector and vector to sequence) networks. For training RNNs, the network is unrolled through time. Cost function is computed in the forward pass. Gradients are calculated and weights are updated using backpropagation through the unrolled network (BPTT). Same weight and bias is used at each time step and thus, gradients are averaged over all time steps. Deep RNNs contain multiple hidden layers.

1. Layer Normalization

Batch Normalization is not efficient for RNNs. A better solution is layer normalization. This is done across feature dimension, not batch dimension. It normalizes previous layer output for each sample and not across the batch. It thus learns scale and shifts parameters for each input independently.

2. LSTM (Long Short Term Memory)

It is a variation of RNN for dealing with its short term memory problem. RNNs often forget the first word in the sequence or in general, the initial input. Due to all the transformations applied on the data, some information gets lost at every time step. To deal with this, LSTMs have 2 states: short term state (output at current time step). and long term state. The network can learn what to store in the long term state and what not to. There are three gates: forget, input and output. Certain memories are dropped when the long term state passes through the forget gate. The input gate selects certain memories which are then added to the long term state. The final output is then filtered by the output gate to give short term state.

3. GRU (Gated Recurrent Unit)

It is a simpler version of LSTM. It has no output gate and the full state vector will act as output at each time step. The same output controls the forget and input gates. There is a new ‘r’ gate which controls which part of the previous output is given to the main layer.

4. Bidirectional RNN

Usually, a RNN only looks at past and present inputs and not the future. Bidirectional RNNs can also look ahead at next inputs before generating the current output. This is especially useful in NLP tasks where one recurrent layer will read the inputs from left to right and the second recurrent layer will read the inputs from right to left. Eventually their outputs will be concatenated at each time step.

5. Dropout in RNN

Dropout is applied between the network layers to drop certain inputs for every instance.

Recurrent dropout drops inputs between the recurrent connections. In other words, dropout is used to mask connections in the vertical direction or between RNN units, while recurrent dropout is applied in the horizontal direction between recurrent layers across different time steps.

6. 1d Convolution network

A 1d convolution layer can be used to learn sequences by sliding kernels across a sequence. This will give a 1d feature map for each kernel. Since I used 32 kernels, the output will be 32 1-dimensional sequences. I used a stride of 1 and “same” padding, thus output sequence will be of the same length as input sequence. I built a 2 layer 1D Conv Net, flattened the output using a global max pooling layer and fed this to a dense network.

7. Hybrid model

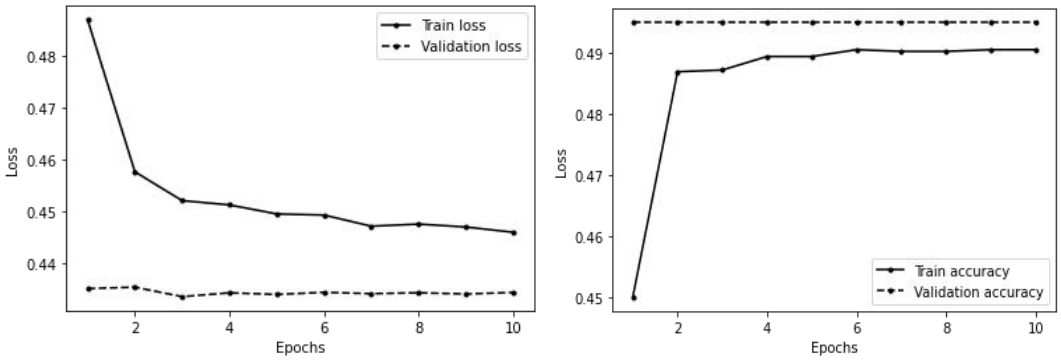
I built a hybrid model using recurrent layers as well as 1d convolution layers. This network comprised of 2 Conv1D layers (each followed by a max pooling layer), 2 bidirectional LSTM layers and a fully connected network.

Number of layers	Accuracy	Loss
Simple RNN (1 layer)	0.4950	0.4341
Deep RNN (2 layers	0.4950	1.3696
Deep RNN with Layer Normalization	0.4935	0.4355
LSTM (1 layer)	0.4954	0.4353
LSTM (2 layers)	0.4950	0.4383
LSTM (with Dropout) Dropout rate=0.5	0.4960	0.4344
GRU (1 layer)	0.4859	0.4362
GRU (2 layers)	0.4950	0.4352

GRU (with dropout) Dropout rate=0.5	0.4950	0.4358
Stacked Bidirectional LSTM	0.4950	1.3661
Stacked Bidirectional LSTM with recurrent dropout	0.4930	0.4357
Stacked Bidirectional LSTM + dense network with 5 layers	0.4950	1.3578
1d Convolutional Neural Network	0.4987	1.3576
Hybrid model (Conv 1d+ recurrent layers)	0.4950	1.3594

Lowest loss so far was achieved by using a 2 layer LSTM network with dropout (rate= 0.5). Learning curves of final model:

Best Model	Loss on test data
2 layer LSTM network with dropout (rate= 0.5)	0.4344



Observation: Here, the validation loss is lower than training loss and validation accuracy is higher than training accuracy. This could be due to the use of dropout. While training, 50% of the neurons are dropped at each iteration which forces each neuron in the network to learn all tasks and not rely on only a few neurons but during testing, all neurons are used thus resulting in a more robust model with higher accuracy. During training, we made it artificially harder for the network to give the correct output. However, during validation all of the neurons are available, thus the network will have its full computational power and perform better.

G. Conclusion

I researched the performance of Multi Layer Perceptron and Recurrent Neural Networks and their different variations in order to categorize drug reviews in terms of drug efficiency . Even after trying a myriad of variations, the model's performance still has a long way to go. However when a machine learning model, Random Forest, is used on this data, we get an accuracy of 32%, which is lower than that achieved by

our final neural network model (49.5%). In fact, it is even lower than the accuracy achieved by a simple MLP network. Thus we can conclude that neural networks showed a significant improvement over statistical machine learning models in this case.
Further, this work can be scaled to inform individuals regarding any possible side effects of drugs and allow pharmaceutical companies to cater to public opinion and eventually improve drug performance.

H. References

[1] https://archive.ics.uci.edu/ml/datasets/Drug+Review+Dataset+%28Drugs.com%29
[2] “Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems 2nd Edition”, by Aurélien Géron
[3] "Ghahramani, Z., & Gal, Y. (2016). A Theoretically Grounded Application of Dropout in Recurrent Neural Networks"
[4]” Nasir, J., Khan, O., Varlamis, I. (2021). “Fake news detection: A hybrid CNN-RNN based deep learning approach”