Understanding depth in neural networks for sentiment analysis and image classification

Akilesh Badrinaaraayanan (20137625) **Kundan Kumar (20084523)**

Pravish Sainath (20125633) **Swechha (20123258)** **Vicki Anand (20137598)**

1 Introduction

Deep neural networks have had a remarkable impact at solving some of the very important machine learning problems over the last decade. Some of the notable performance gains have been achieved for the following tasks: image classification, machine translation, speech recognition and synthesis, etc. Despite this tremendous success, properties of deep neural networks are poorly understood. There is a lack of principled methods for choosing width and depth of neural networks. In theory, the deeper the network is, the more capacity it has, assuming that the width of each layer is the same and each layer is followed by a non-linearity. However, is the optimization process (usually one of stochastic gradient descent based methods) able to find the more complicated function (which lies in the set of all possible functions learnable by the deeper network)? He et al. [2016] has shown that this is not always possible, and has proposed a way to counter that by making the deep networks learn the residual functions. There are many hyper-parameters (variables) that impact the training performance (i.e. achievable capacity of the neural network. Achievable capacity may be defined as the set of functions are within the reach of stochastic optimization process in reasonable number of iterations) and test set performance (generalization performance). In this project, we aim to understand the impact of depth of the neural networks on training set performance and test set performance with carefully designed experiments, keeping all other variables to be the same. We perform our experiments on two tasks: image classification and sentiment analysis. The tasks were chosen so that we could evaluate different types of neural networks i.e. convolutional and recurrent. In the following sections we give detailed explanation of the problem, experimental setup and results obtained.

2 Sentiment Analysis

Sentiment analysis is one of the most important applications of Natural Language Processing. Movie reviews are an important method to gauge the performance of a movie. Analyzing the sentiment can help us determine the attitude of a reviewer and overall polarity of the review. In this work we use sentiment analysis to analyze the overall reaction of the reviewers to a movie i.e whether they liked it or hated it.

2.1 Large Movie Review Dataset

We use the movie reviews dataset taken from IMDb that is available here. This dataset consists of 25k reviews for training and 25k for testing. In addition, each of training/testing set consist of 12.5k positive and 12.5k negative reviews. The sentiment of these reviews are binary meaning the IMDb rating of < 5 results in a sentiment score of 0 and $>=7$ results in a sentiment score of 1.

Preprint. Work in progress.

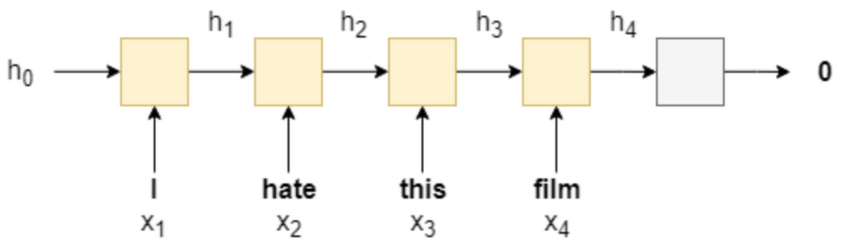


Figure 1: Our basic RNN architecture [1]

2.2 Data preprocessing

The IMDb dataset has a predefined train/test split. We further divide the train set into train/validation set using a 70/30 split of the train set. We convert every word in the training set into a one-hot vector. The number of unique words in our training set is very large (>100K). This means if we directly convert them into one-hot vector , our one-hot vectors will have more than 100K dimensions, leading to very slow training and difficulties in training on a single GPU. Hence, we cut down the vocabulary by using the most-common 25,000 words. If a particular word is present in a sentence, but not present in the vocabulary we replace it with a "<unk"> token.

2.3 Architectures

In this section, we present the different architectures we explored for the sentiment analysis task.

2.3.1 Recurrent neural network

Recurrent Neural Network (RNNs) are commonly used for analysing sequences. An RNN takes in sequence of words, $X = \{x_1, x_2, \dots x_T\}$, one word at a time and produces a hidden state h, for each word. At a particular time step t, we feed in current word x_t as well as previous hidden state h_{t-1}, to produce next hidden state h_t as described in equation 1.

$$h_t = RNN(x_t, h_{t-1}) \tag{1}$$

Our model consists of an embedding layer, RNN and a fully-connected layer. The embedding layer first transforms sparse one-hot vector into dense embedding vector (dimension of this embedding vector is smaller and they are real-values). RNN takes in this dense vector that comes from embedding layer and the previous hidden state h_{t-1} and obtains new hidden state as per equation 1. After the final hidden state h_T is obtained, we feed it through a fully-connected layer (FC) to predict the sentiment of the given review as shown in figure 1.

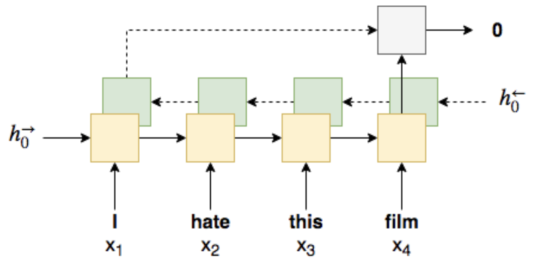
2.3.2 Improved LSTM architecture

In our next model, we use pre-trained word embeddings instead of initializing the word embeddings randomly. We use Glove embeddings Pennington et al. [2014] for each word that is a vector of 100 dimensions. These pre-trained vectors have words with similar semantic meaning close together in vector space and exhibit interesting vector arithmetic properties. We use Long Short Term Memory (LSTM) architecture instead of standard RNNs as RNNs suffer from vanishing gradient problem. LSTMs have an additional cell-state(c) that controls how much information from previous states should be retained/forgotten through gating mechanism. The update equation for LSTM is described in equation 2

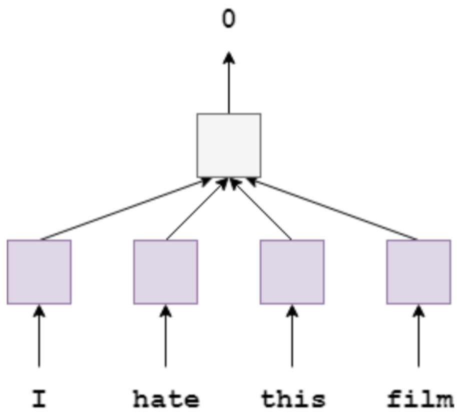
$$(h_t, c_t) = LSTM(x_t, h_{t-1}, c_{t-1}) \tag{2}$$

In a bi-directional LSTM, there are two LSTMs: one processing words in the forward direction (from first word to last word $h_{T\rightarrow}$), one processing words from last to first (a backward LSTM $h_{T\leftarrow}$). These two representations are concatenated before passing through FC to obtain final prediction as shown in figure 2a.

[1]Image taken from here



(a) Bidirectional LSTM architecture



(b) FastText model

Figure 2: Bidirectional LSTM and FastText. [2]

In multi-layer LSTM, we have other layers of LSTMs on top of the base LSTM. The hidden state output of the first LSTM at time t will be the input to LSTM above it. The prediction is made from the final hidden layer of highest layer LSTM.

We use a multi-layer bi-directional LSTM with pre-trained word embeddings in our experiments. In addition to this, we use dropout regularization Srivastava et al. [2014] which is a technique in which we randomly drop out (set to 0), certain fraction of neurons in forward pass.

2.3.3 FastText model

The FastText model Joulin et al. [2016] has only embedding layer and a FC layer (no RNN layer). The embeddings obtained are averaged using `avg_pool2d` and then fed through FC layer to obtain final prediction as shown in figure 2b. One important aspect in FastText model is that `n-grams` of an input sentence are calculated and appended to the end of a sentence. In our model, we use `bi-grams`.

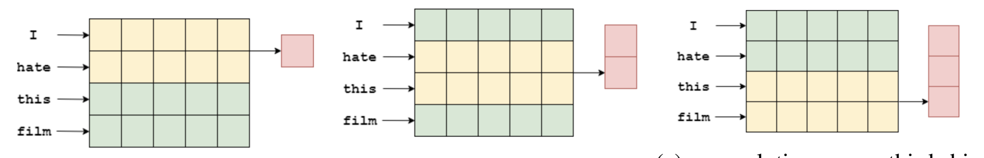
2.3.4 Convolutional Neural Networks

We implement the CNN model as described in Kim [2014]. In FastText model, we appended various `bi-grams` of a sentence to the end of that particular sentence. Here, we use multiple filters of different filter sizes which will looks at different `bi-grams`, `tri-grams` and in general `n-grams` based on chosen filter-size. The primary idea being the appearance of certain `n-grams` in a review, will be a good indicator of final sentiment. If we consider the input sentence to be a 2D grid of dimension `[sentence_length, embedding_dim]`, we use filters of size `[n x embedding_dim]` to cover various `n-grams` sequentially. For instance, when n is 2, figure 3 illustrates how this is done. We then do a max-pooling over these values to obtain "most-important" bi-gram within the review. In our model, we use 100 different filters of filter sizes `[3 x embedding_dim]`, `[4 x embedding_dim]`, `[5 x embedding_dim]`, therefore looking at various `tri-grams`, `4-grams` and `5-grams` to obtain 300 different `n-grams` the model thinks is important. These are then concatenated and passed through a FC to obtain final prediction of the sentiment. We also perform the above experiments by using 150, 200 different filters of mentioned filter-sizes, thereby creating 450, 600 dimensional vector respectively, which are further passed through a FC to obtain prediction.

2.4 Results and discussion

The dimension of the embedding layer is 100 in all the models. The hidden layer dimension of RNN-based models is 256. The loss function used in all experiments is *BCEWithLogitsLoss* in PyTorch as it first feeds the output prediction through a sigmoid layer restricting the values between 0 and 1. In order to ensure a fair comparison of all the models, they are trained with SGD, batch size 64 and for 5 epochs.

[2]Image taken from here



(a) convolution over first bi-gram. (b) convolution over second bi-gram. (c) convolution over third bi-gram.

Figure 3: Illustration of convolution over various bi-grams [3]

Table 1: Results

Architecture	Test Accuracy	Train Accuracy
Basic RNN(1-layer)	45.67	50.07
Basic RNN(2-layer)	48.10	51.35
Bidirectional LSTM(1-layer)	86.04	94.18
Bidirectional LSTM(2-layer)	86.49	94.79
Bidirectional LSTM(3-layer)	83.01	95.82
FastText	85.07	86.13
CNN(100 filters)	88.61	97.04
CNN(150 filters)	**88.79**	97.79
CNN(200 filters)	88.62	98.24

Table 1 summarizes the results for all the models that we tried. With basic RNN model, we are getting an accuracy of 45.67%. On increasing the depth to 2, we observe slight increase in accuracy value of this model (48.10%). With the improved bi-directional LSTM model, we get significant boost in accuracy (86.04%). On increasing the depth of this model to 2, we observe an increase in accuracy (86.49%), which falls down as we increase the depth to 3 (83.01%). This suggests that going beyond 2 layers on this dataset is not beneficial. FastText model gives performance comparable (85.07%) to improved bi-directional LSTM model,with far few parameters and much less training time. CNNs perform the best among all these models. We can see from the table that CNN (with 150 filters) performs the best achieving **88.79%** accuracy on this task. This can intuitively be explained as in our CNN architecture, we look at various `3-grams`, `4-grams`, `5-grams` and the presence of certain `n-grams` in input sentence influence the overall sentiment of the sentence. Increasing the number of filters of each filter size further to 200 led to small drop in performance.

3 Image Classification

We investigate the performance of the convolutional neural networks on image classification task for CIFAR10 datasetKrizhevsky et al. [2014]. Images in the dataset have 32x32 pixels, containing pictures of one of the ten classes. There are in total 50000 training images.

3.1 Data Preprocessing

We use three different image representations: raw pixels, HOG and SIFT.

3.1.1 HOG

HOG stands for the histogram of oriented gradients. It is one of the commonly used image feature descriptors for various computer vision tasks. Every local patch in an image is represented by a histogram of counts of gradient in various directions. This descriptor has an advantage that it is invariant to geometric transformations and illumination. They are sensitive to scaling and rotation.

[3]Image taken from here

3.1.2 SIFT

SIFT is short form of Scale-invariant Feature Transforms and represents local image features are at specific points (or interests points). SIFT descriptors are again histogram of orientation (computed based on gradients) adjusted with scale, and computed via scale space pyramid. They are often invariant to rotation, scaling, illumination and noise. It was introduced in Lowe [2004].

3.2 Model

In this section, we describe various components used in our model for image classification. We don't do any data augmentation so as to keep training time in check.

3.2.1 Convolutional Block

Convolutional layers are the basic building blocks in most of the successful approaches to solve computer vision problems. In principle, each layer consist of many parameterized kernels (whose weights are learned). The output of the layer is the convolution operation of the kernel on the the input. In general, kernels are very small compared to the size of the image, hence, there is local connectivity in the output hidden units. To learn more expressive functions, non-linearity is applied after the convolution operation. Any hidden unit depends on some local area of the input image. If the hidden unit is at larger depth then it sees larger input area. It has been shown that convolutional neural networks learn hierarchical features i.e. initial layers learning color gradients and very simple patters, while the deeper layer learns more complicated shapes.

3.2.2 Residual Block

Residual blocks have become quite popular recently following the results in He et al. [2016]. In principle, they learn the difference from the identity function (linear transform when the output size is different) i.e. $h(x) = x + non\_linearity(A * x)$ if output size is same as x, otherwise $h(x) = B * x + non\_linearity(A * x)$. This has some nice properties, e.g. the shortest path between the cost computation units and various hidden units is quite small (or doesn't have too many non-linearity). This leads to lower vanishing gradient problems which had prohibited training of very deep networks before.

3.3 Results

In this section, we will present result for the experiments we tried for image classification task.

3.3.1 Training with raw pixels

Using raw pixels enables learning features relevant for the task and making very less assumption about the images. We train convolutional models with raw pixel inputs. The convolutional network has five blocks, each reducing the spatial dimension of its input by a factor of 2, and increasing the number of channels by 2. In each block, we keep **N** resolution and number of channel preserving convolutions with **kernel size 3, padding 1 and stride 1**. Final layer in this block is a strided convolution with **kernel size 3, padding 1, stride 2 and output channels being double of the input channels**. After every convolution layer, there is a rectified linear unit as non-linearity, final convolution is also followed by dropout with probability 0.25. For resnet experiments, our residual block consists of a basic convolutional block, say B, as described above, and an additional strided convolution layer, say S, with exact hyper-parameters as that of the final layer of the basic block. Output of this residual block, is S(x) + B(x). Since each block reduces spatial dimension by 2x, we use 5 such blocks in total, given that our input is 32x32. We vary the depth of the network by varying N (number of resolution preserving convolution layers) in each basic block. We train each model till 200 epochs.

3.3.2 Training with HOG

HoG features were obtained by using a 4 x 4 cell size over the 32 x 32 images with histograms of 9 different orientations. A block normalization was applied over each 4 x 4 block to obtain a final 576 x 1 feature descriptor vector. These feature descriptors were used for training with a multiclass

Block type	N	Train Accuracy (in %)	Test Accuracy (in %)
Basic	1	63.85	63
Basic	2	9.77	10
Basic	3	9.60	10
Basic	4	9.51	10
Residual	1	99.60	74
Residual	2	99.89	**75**
Residual	3	99.60	74
Residual	4	100.	74

Table 2: Results for image classification with raw pixels. **Basic** blocks and **residual** blocks are defined in subsection 3.3.1.

logistic regression (softmax regression) and with MLPs with different number of hidden layers. The MLPs were trained for 100 epochs. The results obtained are summarized in the table 3.

Table 3: Results for classification using HoG features

Architecture	Train Accuracy (in %)	Test Accuracy (in %)
Multi-class Logistic Regression	55.16	51.01
MLP-1 hidden layer (32 neurons)	59.64	54.81
MLP-2 hidden layers (32, 24 neurons)	62.16	58.23
MLP-3 hidden layers (32, 24, 16 neurons)	69.34	**64.48**

3.3.3 Training with SIFT

The SIFT feature descriptors were obtained using threshold image gradients sampled over 16x16 array patches of the image to create array of orientation histograms giving 8 orientations of 4 x 4 histogram arrays as 128 bit vectors.

A multiclass logistic regression (softmax regression) and a MLPs with different hidden layers were used for training with these SIFT features. The MLPs were trained for 100 epochs. The results obtained are summarized in the table 4.

Table 4: Results for classification using SIFT features

Architecture	Train Accuracy (in %)	Test Accuracy (in %)
Multi-class Logistic Regression	64.66	60.85
MLP-1 hidden layer (32 neurons)	68.57	63.32
MLP-2 hidden layers (32, 24 neurons)	72.13	68.44
MLP-3 hidden layers (32, 24, 16 neurons)	77.52	**72.21**

4 Conclusion

In this project, we experimented with different neural network architectures and input representations for image classification (on CIFAR10 dataset) and sentiment analysis (on Large Scale Movie Dataset). In the regime of appropriate hyper-parameters for sentiment analysis task, we find that using deeper and/or network usually leads to better training error, however, for convolutional neural network with higher depth and without residual training error, the model doesn't train for higher depths. We also observe that with using MLPs with feature extraction methods like HoG and SIFT, both the train and test accuracy increase with depth. We find that using neural networks gives immense capacity even with very raw input, if appropriate architecture is used (e.g. bidirectional RNN for sentiment classification and residual networks for image classification).

Acknowledgments

We thank pytorch and opencv opensource community for developing awesome software which we used in this project. Kundan and Swechha worked on image classification with convolutional neural networks with raw pixels. First version of the code for this was written by Swechha, which was further modified and made modular by Kundan. Experiments designs and core model architecture was the outcome of discussion between Kundan and Swechha. Swechha and Kundan ran these experiments and compiled the results. Kundan also wrote report for task independent sections e.g. introduction and conclusion and most parts of image classification section. Pravish worked on baseline experiments for image classification, dealing with HOG and SIFT with logistic regression and MLP. Pravish wrote HOG and SIFT section of image classification experiments.

Akilesh and Vicki did all experiments pertaining to sentiment analysis task including data preprocessing, design of different architecures, experimental evaluation of importance of depth/width. They also wrote the report corresponding to these sections.

References

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*, 2016.

Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 2004.

Jeffrey Pennington, Richard Socher, and Christopher Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.

Appendix

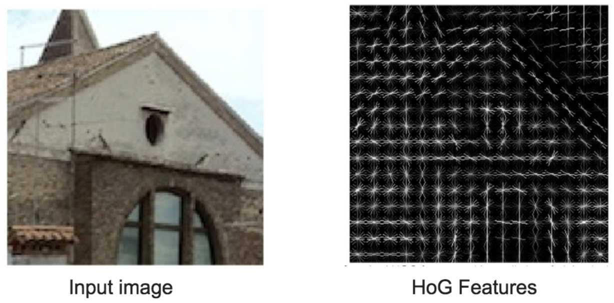


Figure 4: Example HoG Features [4]

[4]Image taken from here

[5]Lowe [2004]

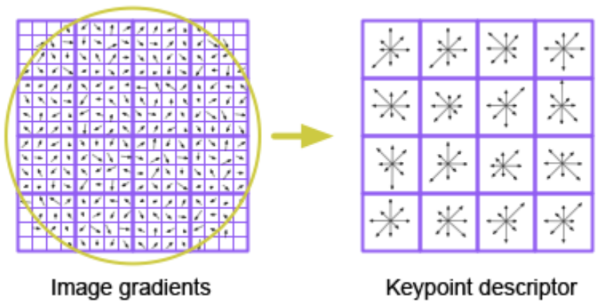


Figure 5: SIFT Feature Extraction[5]