Neural Network Architecture for Natural Language Processing

Colloquium Report
Submitted in Partial Fulfillment of Requirements for the Degree of Master of Technology

Rushikesh Dumbre (2017IS03)

MNNIT Allahabad

Computer Science and Engineering Department
Motilal Nehru National Institute of Technology
Allahabad
Allahabad (India), 211004
2018

Abstract. Neural networks have emerged as powerful machine-learning models better than the previously used linear algorithms, giving highly accurate results in fields such as speech processing. We can also use them for natural language processing. The report covers data preprocessing for NLP, designing of feed-forward, convolutional and recurrent neural networks. Proir to neural networks, Machine learning algorithms were used for Natural Language Processing. These algorithms used linear models such as support vector machines (SVM) or logistic regression. Switching from such linear models to non-linear neural-network models has given better results. This report contains basic background, terminologies, tools and algorithms that will help to understand the designing of the neural network models and its application. The report also contains recent advances like word embeddings that further improve the performance of the neural networks.

1 Introduction

Neural networks are powerful learning models better than machine learning techniques. Neural network architectures can be divided into 2 broad categories, feed-forward and recurrent neural networks. Feed-forward networks contain layers that are completely connected. They may also contain mixture of convolutional and pooling layers. These neural networks can be used for classification problems or prediction problems. This includes 2-label (Yes/No) and multilabel classification problems, as well as linear regression problems.

As the neural networks are non linear highly accurate classification can be achieved. Convolutional neural systems are helpful for grouping assignments in which we hope to discover highlights with respect to class participation, yet these highlights can show up in better places in the information. For instance, in a report subject order issue, a solitary key expression (likewise called as ngram) can help in deciding the theme of the record[6]. A specific arrangement of words can help in recognizing the theme, yet the occurence of the words can be anyplace in the archive. In natural language we often work with structured data of different sizes, such as sentences. We want to obtain regularities (features) between these structures. For this we encode these structures into fixed size arrays (or vectors). Recurrent networks best fit with sequence modelling, while recursive networks are a simpler version of recurrent networks that support tree modelling. Recurrent architectures have produced very strong results for natural language modeling as well as for sequence tagging, auto translation, sentiment analysis. Recursive models were shown to produce strong results for faction and dependency, parse reranking, discussion interpretation, semantic relation classification, political philosophy identification based on concrete syntax trees, opinion classification, and question answering.

2 Feature Modelling

We need to understand how features are represented before moving to the network architectures. Consider that $NN(x)$ is a feed-forward neural network function with x as input. x is a d_{in} dimensional vector. $NN(x)$ produces a d_{out} dimensional output vector. The function classifies input x to one of the d_{out} classes. The function is almost always non-linear. The input x encodes features such as words, bi-words or other linguistic information in case of NLP. Each feature is concatenated into a d-dimensional vector. The feature vectors (weights) are treated as model parameters that are updated to model the problem. The general structure for a feed-forward neural network, NLP classification system is as follows:

1. Extract relevant features $f_1, ..., f_x$ capturing text information for predicting the output class.
2. For each textual feature f_i, obtain the corresponding vector $v(f_i)$.
3. Combine the feature vectors into an input array x
4. Give x as input to the neural network function $NN(x)$.

2.1 Dense Vectors vs. One-Hot Representations

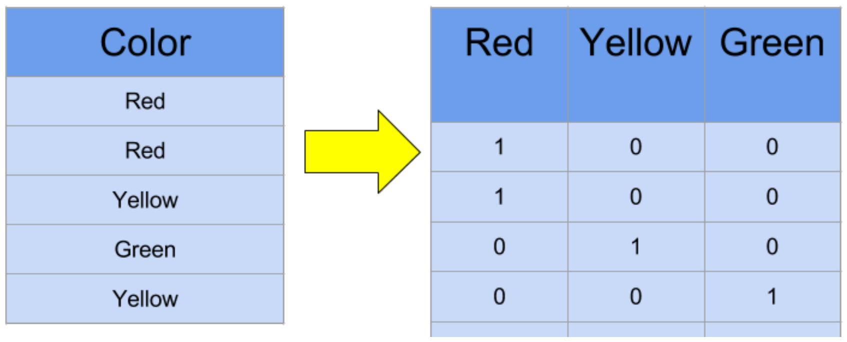
Consider the two kinds of representations:

One Hot:- Each feature is its own dimension.

– Dimensionality of one-hot vector is same as number of distinct features.
– Features are completely independent from one another.

Dense:- Each sample is a d-dimensional vector.

– d is the number of features.
– After training similar samples have similar vectors.
– One benefit of using dense and low-dimensional vectors is computational. Computation is slow for high dimensional vectors.



Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1

Fig. 1. Dense vs One Hot Encoding

Dense representations are able to generalize. For example, assume that the word ‘water’ appears many times in the dataset, but the word ‘steam’ only a few times. If each of these words are kept in separate dimensions then the occurence of ’water’ cannot tell us anything about the occurence of ’stream’. However, the values in the dense vectors representation for ‘water’ may be similar to that of ‘steam’ (or atleast very close). If in case we have few distinct features then one-hot representation is suitable. However, if we believe there are going to be correlations between the different features, it is better to let network carry out the interaction between different features than to do it manually. [2].

Dense vector representation of features is commonly used for neural networks and also not much difference exists in using dense or sparse vectors when it comes to result accuracy. When one-hot vectors are used as sample inputs, rest layer of the network to act as embedding layer which learns the dense vectors.

2.2 Continuous Bag of Words

Feed-forward networks assume that each input sample has same number of dimensions. Thus the extraction of features is easy as the number of features is settled. This number cannot be checked in some cases. However, in some cases the number of features is not known in advance. The number of features is not fixed, but we still need to use vector of predetermined dimensions for neural networks. Continuous Bag Of Words (CBOW) representation can be used in such cases[4].

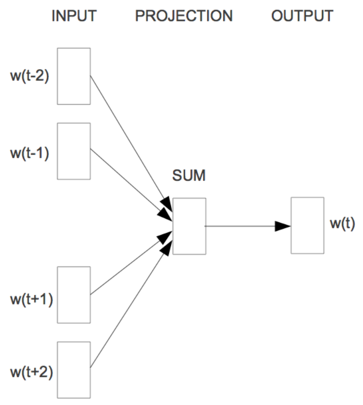


Fig. 2. CBOW predicts current word based on context

$$CBOW(f_1, f_2, ...f_k) = \frac{1}{k}\sum_{i=1}^{k} v(f_i)$$

Here feature f_i may be a word in the document in case of the document classification problem.

2.3 Distance and Position Features

The euclidean interdistance of word vectors can be used as a relation between them. For example, in the problem of finding an event in a text, we take words in a text as arguments against a trigger word, and we have to predict if the argument is similar to the trigger. If the angle between the vectors is less, the argument is similar to trigger. In the traditional NLP model, distances are usually encoded by assigning a bucket to the distances associating each bucket with a one-hot vector. If the input in a neural network has feature vectors that can take more than two values, it is better to allocate a single input value to the distance. However, this approach is not feasiblee. Instead, distance vectors are encoded in such a way that each feature is a d-dimensional vector which are trained as regular parameters in the network.

3 Feed-Forward Neural Networks

Architecture of neural networks is inspired by the learning process in animals. Brain consists of neurons that fire upon execution of a certain event. This firing strengthens the neuron connection and the learning happens. A neuron is a computational unit that has scalar information I/O. Each info has a related weight. The neuron calculates product of each contribution by its weight, at that point computes a summation for all inputs, passes the outcome through a non-linear filter, and passes it to its yield. The neurons are associated with each other, framing a system[2]. Yield of one layer of neurons is bolstered as contribution to the following level. Such systems were appeared to be extremely fit. In the event that the weights are prepared accurately, a neural system with enough neurons and a fitting non-linear enactment capacity can surmise complex scientific applications.

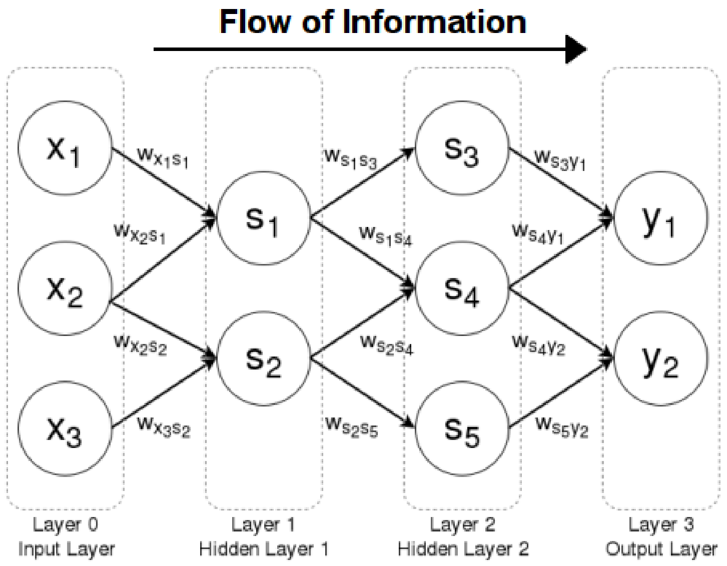


Fig. 3. Feed-Forward Neural Network

Each circle is a neuron, incoming arrows are the neuron's inputs and outgoing arrows are the neuron's outputs. Every arrow conveys a weight. Neurons are organized in layers (Here the stream is from Layer 0 to Layer 3). The principal layer has no approaching arrows since it is the contribution to the system. The last layer has no leaving arrows since it is the yield of the system. The layers amongst information and yield are called as concealed layers. The S_i inside the neurons in the middle layers represents a non-linear function that is applied to the weighted input before output is generated. If each neuron in a layer is connected to all of the neurons in the next layer then the layer is called a fully-connected layer.

The values of each row of neurons in the network can be thought of as a matrix. The fully connected layer is a linear transformation from $n - 1$ dimensions to n dimensions where n is the current layer. A fully-connected layer implements a matrix multiplication, $h = xW$ where W is the weight matrix and the weight of the connection from the i^{th} neuron in the input row to the j^{th} neuron in the output row is W_{ij} . A non linear function g is then applied to the values of h that is applied to each value before being passed on to the next input. This can be written as:

$$(g(xW_1))W_2$$

where W_1 are the weights of the first layer and W_2 are the weights of the second one.

3.1 Non-linear functions

The non-linear function g can take many forms[2].

1. Sigmoid

   The sigmoid activation function $\sigma(x) = 1/(1 + e^{-x})$, also called the logistic function, transforms each value x into the range $[0, 1]$. The sigmoid was the default non-linearity for neural networks since their beginning, but is currently deprecated for use in hidden layers of neural networks, as the other choices work much better.

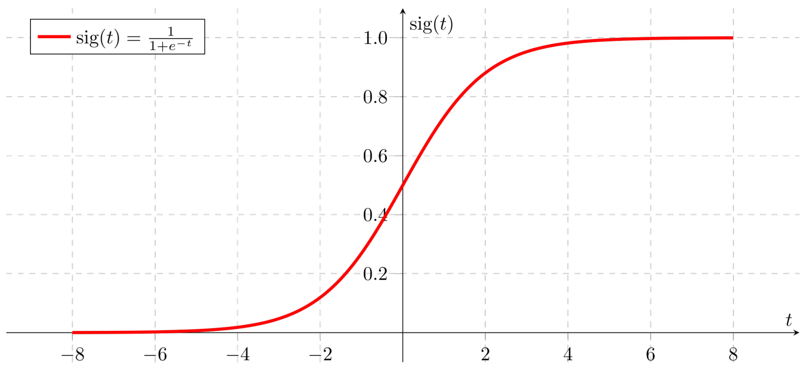


Fig. 4. Sigmoid

2. Hyperbolic Tangent (TANH)
   The hyperbolic tangent $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$ activation function transforms the values x into the range $[-1, 1]$.

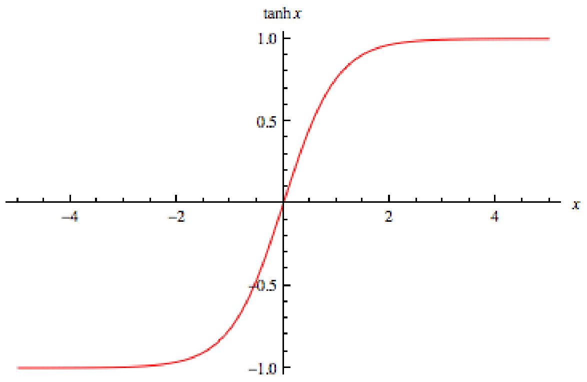


Fig. 5. TANH

3. Rectifier (ReLU)
   The Rectified Linear Unit is a very simple activation function that is also computationally efficient. The ReLU unit sets each negative value at 0. Despite its simplicity, it performs well for many tasks.

$$ReLU(x) = max(0, x)$$

Empirically ReLU units work better than tanh, and tanh works better than sigmoid.

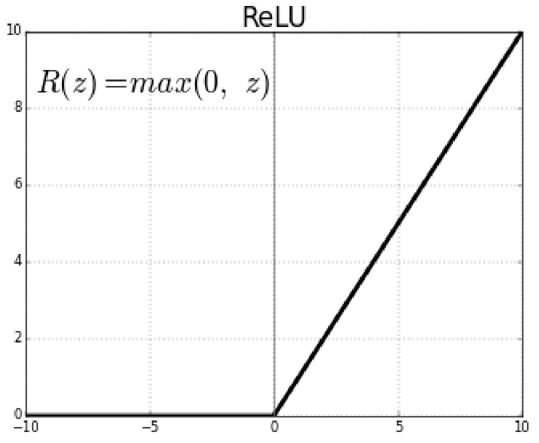


Fig. 6. ReLU

3.2 Output Transformation

In some cases (multilabel classification) output vector is also transformed. One transformation is softmax:

$$softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

The result is a vector whose values range from 0 to 1. This can be interpreted as a probability distribution over k possible outcomes. This transformation is used when we are modeling a discrete probablity over more than two labels.

3.3 Loss Functions

We need to define a loss function $L(\hat{y}, y)$, that states the loss of when the network predicts $\hat{y}$ when the true output is y. The aim is to reduce the loss for all types of samples as much as possible. The loss $L(\hat{y}, y)$ assigns a real number for the network's output $\hat{y}$ where the true value is y. The minimized value for loss function is achieved only for cases where the network predicts correctly. The parameters of the network (the weight matrices W_i , the biases b_i) are then updated in order to minimize the loss. For optimizing the calculations, we use curves whose slope is easily calculable. Below given is the most commonly used loss function, Log Loss function, in neural networks.

$$L_{log}(\hat{\mathbf{y}}, \mathbf{y}) = -(y \log(\hat{y}) - (1-y) \log(1-\hat{y}))$$

3.4 Word Embedding Initialization

At the point when enough administered preparing information is accessible, one can simply treat the component embeddings equivalent to other parameters: introduce the implanting vectors to arbitrary qualities, and let the system preparing strategy tune them into "great" vectors. Some care must be taken in the manner in which the irregular instatement is performed[3].

Practically speaking, one will regularly utilize the irregular instatement way to deal with introduce the embedding vectors of generally happening highlights, for example, grammatical form labels or individual letters, while utilizing some type of managed or unsupervised pre-preparing to instate the conceivably uncommon highlights, for example, highlights for singular words. The pre-prepared vectors can then either be dealt with as settled amid the system preparing procedure, or, all the more regularly, treated like the haphazardly introduced vectors and further tuned to the job that needs to be done.

4 Contexts of Word

The contexts of a word are the surrounding words, whose size depends on the model. It may be a sentence, paragraph or the entire text. We interpret the text and the contexts are derived from the semantic closeness derived from the interpretation. The context can be parts of the word as well.

Neural word embeddings started from dialect displaying, in which a system is prepared to foresee the following word in view of an arrangement of ocurred words. The point in such an issue is to anticipate a word in light of a setting the k past words. While preparing for the dialect displaying assistant expectation issues deliver valuable embeddings, in which one is permitted to take a gander at the past words, this approach is unnecessarily limited by the imperatives of the dialect demonstrating errand. Here we just think about the subsequent embeddings so we can take the setting to be a symmetric window around the concentration word.

5 Neural Network Training

The process of training a model includes minimizing loss over the training set, by updating the weights. Training method is carried out for a considerable number of iterations computing the loss, computing the gradients w.r.t loss function, then using the gradients for updating the parameters backwards. Models differ in how the error estimate is computed, and how parameters are updated.

The basic algorithm, stochastic gradient descent (SGD), is described below . Gradient calculation is central to the approach. Gradients can be efficiently and automatically computed using chain rule of differentiation on a computation graph – a general algorithm for automatically computing the gradient of any network and loss function.

5.1 Stochastic Gradient Training

The most widely used algorithm for training neural networks is the Stochastic Gradient Descent (SGD) algorithm. It receives a loss function f parameterized by W, and the train and test I/O samples. It then attempts to set the parameter W such that the value of f with respect to the training samples is minimized. The algorithm works as follows[2]:

The goal of the algorithm is to update the parameters θ so as to minimize the total loss $\sum_{i=1}^{n} L(f(x_i; W), y_i)$ over the training set. It works by repeatedly sampling each training example and computing the gradient of the error on the example with respect to the parameters W (line 4a) – the input and expected output are fixed, and the loss is treated as a function of the parameters W. The parameters W are then updated in the opposite direction of the gradient

Algorithm 1 Stochastic Gradient Descent

```
1. Function $f(x; W)$ with parameter $W$ is given.
2. Training input samples $x_1, ..., x_n$ and expected labels $y_1, ..., y_n$ are given.
3. Loss function L is also defined.
4. While stopping criteria not met do
   (a) Sample a training example $x_i$ , $y_i$
   (b) Compute the loss $L(f(x_i; W), y_i)$
   (c) $\hat{g} \leftarrow$ gradients of $L(f(x_i; W), y_i)$ w.r.t $W$
   (d) $W \leftarrow W - \eta_t \hat{g}$
5. return $W$
```

Algorithm 2 Minibatch Stochastic Gradient Descent

```
1. Function $f(x; W)$ with parameter $W$ is given.
2. Training input samples $x_1, ..., x_n$ and expected labels $y_1, ..., y_n$ are given.
3. Loss function L is also defined.
4. do for number of iterations or loss not minimized
   (a) Sample a minibatch of m examples $(x_1, y_1), ..., (x_m, y_m)$
   (b) $\hat{g} \leftarrow 0$
   (c) for $i = 1$ to m do
       i. Compute the loss $L(f(x_i; W), y_i)$
       ii. $\hat{g} \leftarrow \hat{g} +$ gradients of $\frac{1}{m} L(f(x_i; W), y_i)$ w.r.t $W$
   (d) $W \leftarrow W - \eta_t \hat{g}$
5. return $W$
```

(back-propagation), scaled by a learning rate η_t (line 4d). The learning rate can either be constant, or decay as a function of the time step (epoch) t. Note that the loss computed in line 4b depends on a solitary preparing case, and is along these lines only a gauge of the vast misfortune that we are expecting to limit. The commotion in the misfortune calculation may bring about off base inclinations. A typical method for diminishing this clamor is to assess the blunder and the angles in view of m tests rather than one example. This offers ascend to the minibatch SGD calculation.

When preparing a neural system, the parameterized work f is the neural system, and the parameters W are the straight change lattices, predisposition terms, installing networks et cetera. The inclination calculation is a key advance in the SGD calculation, and in addition in all other neural system preparing calculations. The inquiry is, at that point, how to process the angles of the system's blunder as for the parameters. Luckily, there is an simple arrangement as the backpropagation calculation. The backpropagation calculation is an extravagant name for deliberately registering the subsidiaries of an unpredictable articulation utilizing the chain- manage, while reserving middle person comes about. All the more for the most part, the backpropagation calculation is an uncommon instance of the invert mode programmed separation calculation. The accompanying area portrays invert mode programmed separation with regards to the calculation chart reflection.

5.2 Computation Graph

While one can register the slopes of the different parameters of a system by hand and actualize them in code, this method is unwieldy and blunder inclined. For generally purpostures, it is desirable over utilize programmed apparatuses for inclination calculation. The calculation chart deliberation enables us to effectively build subjective systems, assess their expectations for given contributions (forward pass), and process inclinations for their parameters concerning self-assertive scalar misfortunes (in reverse pass)[6]. A calculation diagram is a portrayal of a subjective numerical calculation as a diagram. It is a coordinated non-cyclic diagram (DAG) in which hubs compare to scientific tasks or (bound) factors and edges relate to the stream of mediator esteems between the hubs. The chart structure characterizes the request of the calculation as far as the conditions between the diverse segments. The diagram is a DAG and not a tree, as the consequence of one task can be the contribution of a few continuations. Consider for instance a diagram for the calculation of (a $*$ b + 1) $*$ (a $*$ b + 2).

The calculation of a $*$ b is shared. We confine ourselves to the situation where the calculation diagram is associated. Since a neural system is basically a scientific articulation, it can be spoken to as a calculation diagram. For instance, presents the calculation diagram for a MLP with one covered up-layer and a softmax yield change. In our documentation, oval hubs speak to mathematical tasks or works, and shaded square shape hubs speak to parameters (bound factors). System inputs are dealt with as constants, and drawn without an encompassing hub. Information and parameter hubs have no approaching bends, and yield hubs have no cordial curves. The yield of every hub is a grid, the dimensionality of which is shown over the hub. This chart is fragmented: without indicating the sources of info, we can't process a yield demonstrates a total diagram for a MLP that takes three words as data sources, and predicts the circulation over grammatical form labels for the third word. This chart can be utilized for expectation, yet not for preparing, as the yield is a vector (not a scalar) and the chart does not consider the right answer or the misfortune term. At last, the diagram demonstrates the calculation diagram for a particular preparing case, in which the sources of info are the (embeddings of) the words "the", "dark", "canine", and the normal yield is "Thing" (whose list is 5). The pick hub executes an ordering activity, getting a vector and a list (in this case, 5) and restoring the relating section in the vector. Once the diagram is manufactured, it is clear to run either a forward calculation (compute the consequence of the calculation) or a retrogressive calculation (figuring the angles), as we appear beneath. Building the diagrams may look overwhelming, yet is in reality simple utilizing committed programming libraries and APIs.

6 Optimization Issues

Once the inclination calculation is dealt with, the system is prepared utilizing SGD or another inclination based enhancement calculation. The capacity being improved isn't raised, and for quite a while preparing of neural systems was viewed as a "dark workmanship" which must be finished by chose few. Undoubtedly, numerous parameters influence the enhancement procedure, and care needs to be taken to tune these parameters. While this instructional exercise isn't planned as a thorough manual for effectively preparing neural systems, we do list here a couple of the conspicuous issues.

6.1 Initialization

If the loss function is non-convex (multiple local minima/maxima) the value of loss may get stuck at a local minima, and that initializing from different values (different random values for the parameters) may result in different results. In this way, it is informed to run a few restarts with respect to the preparation beginning at various irregular instatements, and picking the best one in view of an advancement set. The measure of fluctuation in the outcomes is distinctive for various system models and datasets, and can't be anticipated ahead of time. The extent of the irregular qualities importantly affects the achievement of preparing. An effective scheme called xavier initialization, suggests initializing a weight matrix $W \in R^{d_{in} x d_{out}}$ as[2]:

$$W \sim U\left[-\frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}, \frac{\sqrt{6}}{\sqrt{d_{in}+d_{out}}}\right]$$

where U [a, b] is a uniform distribution in the range [a, b].

6.2 Vanishing and Exploding Gradients

In profound systems, usually for the mistake slopes to either vanish (turn out to be exceedingly near 0) or detonate (turn out to be exceedingly high) as they proliferate back through the computation diagram. The issue turns out to be more serious in more profound systems, and particularly so in recursive and intermittent systems[5]. Managing the vanishing slopes issue is as yet an open research question. Arrangements incorporate making the systems shallower, step-wise preparing (first prepare the principal layers in view of some assistant yield flag, at that point settle them and prepare the upper layers of the entire system in view of the genuine undertaking signal), performing group standardization (for each minibatch, normalizing the contributions to every one of the system layers to have zero mean and unit difference) or utilizing particular models that are intended to aid angle stream (e.g., the LSTM and GRU designs for intermittent systems). Managing with the detonating slopes has a straightforward yet extremely successful arrangement: cutting the inclinations on the off chance that their standard surpasses a given limit. Give $\hat{g}$ a chance to be the slopes of all parameters in the network, and $\|\hat{g}\|$ be their L_2 norm. It is suggested to set: $\hat{g} \leftarrow \frac{threshold}{\|\hat{g}\|}\hat{g}$ if $\|\hat{g}\| > threshold.$

6.3 Learning Rate

Determination of the learning rate is vital. Too huge learning rates will keep the system from focalizing on a powerful arrangement. Too little learning rates will set aside long opportunity to focalize. As a dependable guideline, one should try different things with a scope of beginning learning rates in go [0, 1], e.g. 0.001, 0.01, 0.1, 1. Screen the system's misfortune after some time, and diminishing the learning rate once the misfortune quits progressing. Learning rate booking diminishes the rate as a capacity of the quantity of watched minibatches. A typical timetable is isolating the underlying learning rate by the cycle number[3]. It prescribes utilizing a learning rate of the shape $\eta_t = \eta_0(1 + \eta_0\lambda t)^{-1}$ where η_0 is the underlying learning rate, η_t is the learning rate to use on the tth preparing illustration, and λ is an extra hyperparameter. He further prescribes deciding a decent estimation of η_0 in view of a little example of the information before running on the whole dataset.

6.4 Minibatches

Parameter refreshes happen either every preparation illustration (minibatches of size 1) or each k preparing illustrations. A few issues advantage from preparing with bigger minibatch sizes. In terms of the calculation diagram deliberation, one can make a calculation chart for each of the k preparing illustrations, and after that interfacing the k misfortune hubs under an averaging hub, whose yield will be the loss of the minibatch. Expansive minibatched preparing can likewise be helpful as far as calculation effectiveness on particular registering structures, for example, GPUs, and supplanting vector-lattice tasks by grid framework activities. This is past the extent of this instructional exercise.

6.5 Regularization

Overfitting can occur easily in a neural network models having many parameters . Overfitting is the condition where the network is not able to generalize the dataset. This can be reduced to some extent by regularization[2]. A common regularization method is L_2 regularization, penalizes the weights by summing $\frac{\lambda}{2}||\theta||^2$ term to the parameters, where θ is the set of model parameters, $||\theta||_2^2$ is the squared L_2 norm (sum of squares of the values), and λ is called the regularization parameter. Sometimes dropout regularization is also used.

The dropout strategy is intended to keep the system from figuring out how to depend on particular weights. It works by nullifying the effect of some of the neurons in the system (or in a particular layer) in each preparation test. The dropout regularization procedure is one of the key variables adding to exceptionally solid consequences of neural-organize techniques on picture characterization errands , uncommonly when joined with ReLU initiation units. The dropout strategy is successful likewise in NLP uses of neural systems.

6.6 Model Cascading

Model Cascading is a great procedure in which substantial systems are worked by making them out of littler segment systems. For instance, we may have a feed-forward system for foreseeing the grammatical feature of a word in view of its neighboring words as well as the characters that create it. In a pipeline approach, we would utilize this system for foreseeing parts of discourse, and at that point feed the forecasts as info highlights to neural system that does syntactic piecing or on the other hand parsing. Rather, we could think about the concealed layers of this system as an encoding that catches the pertinent data for foreseeing the grammatical feature. In a falling approach, we take the concealed layers of this system and associate them (and not the part of discourse forecast themselves) as the contributions for the syntactic system. We presently have a bigger system that takes as info successions of words and characters, and yields a syntactic structure. The calculation diagram reflection enables us to effortlessly spread the blunder slopes from the syntactic undertaking misfortune the distance back to the characters. To battle the vanishing angle issue of profound systems, and in addition to improve utilization of accessible preparing material, the individual part system's parameters can be bootstrapped via preparing them independently on an applicable errand, before connecting them to the bigger system for additionally tuning. For instance, the grammatical form anticipating system can be prepared to precisely anticipate parts-of-discourse on a generally expansive explained corpus, before connecting its shrouded layer to the syntactic parsing system for which less preparing information is accessible. On the off chance that the preparation information give guide supervision to the two assignments, we can make utilization of it amid preparing by making a system with two yields, one for each errand, processing a different misfortune for each yield, and after that summing the misfortunes into a solitary hub from which we backpropagate the mistake slopes.

7 Convolutional Layers

At times we are keen on making expectations in view of requested arrangements of things (e.g. the succession of words in a sentence, the arrangement of sentences in a report et cetera). Consider for instance foreseeing the feeling (positive, negative or impartial) of a sentence. A portion of the sentence words are extremely instructive of the conclusion, different words are less enlightening, and to a decent estimate, a useful piece of information is useful in any case of its situation in the sentence. We might want to nourish the greater part of the sentence words into a student, and let the preparation procedure make sense of the critical pieces of information. One conceivable arrangement is nourishing a CBOW portrayal into a completely associated system, for example, a MLP[4]. Notwithstanding, a drawback of the CBOW approach is that it overlooks the requesting data totally, relegating the sentences "it was bad, it was entirely awful" and "it was not awful, it was quite great" precisely the same. While the worldwide position of the pointers "not great" and "not awful" does not make a difference for the characterization errand, the neighborhood requesting of the words (that "not" seems acceptable before "terrible") is important. A credulous approach would propose inserting word-sets (bi-grams) rather than words, and building a CBOW over the installed bigrams. While such a design could be powerful, it will bring about enormous inserting frameworks, won't scale for longer n- grams, and will experience the ill effects of information sparsity issues as it doesn't share measurable quality between various n-grams (the inserting of "very great" and "great" are totally autonomous of each other, so if the student saw just a single of them amid preparing, it won't have the capacity to find anything about the other in light of its part words). The convolution-and-pooling (likewise called convolutional neural systems, or CNNs) engineering is an exquisite and powerful answer for this displaying issue. A convolutional neural system is intended to distinguish characteristic neighborhood indicators in an expansive structure, and join them to produce a fixed size vector representation of the structure, capturing these local aspects that are most informative for the prediction task at hand.

7.1 Convolution + Pooling

After applying convolution and pooling over a window, a non-linear function is applied over the output. This transforms a window of k words into a d dimensional vector and the important features are learned by the kernel. The d dimensions of a vector are often called as channels. Each channel is an output of a different kernel. The number of channels in a kernel are fixed based on the number of channels in the input to that kernel. Convolution operation is followed by a max or average pooling in order to reduce the size of input to next layer. This allows faster computations. The gradients calculated using loss function are propagated back during the training. These gradients are used to update the kernels starting from the last layer.

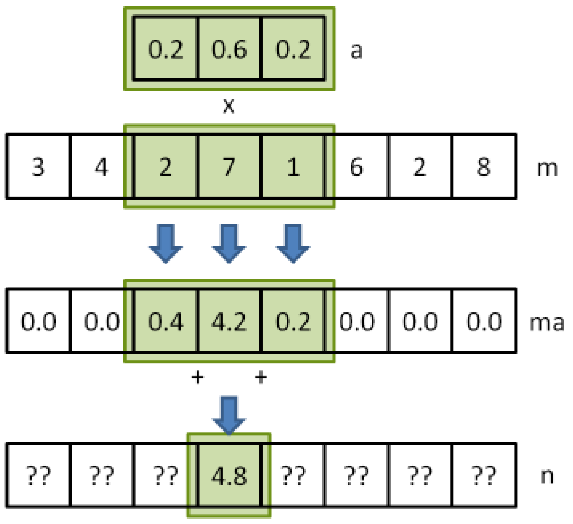


Fig. 7. 1 Dimensional Convolution

Consider a sequence of words $x = x_1, ..., x_n$, each with their corresponding d_{emb} dimensional word embedding $v(x_i)$. A 1d convolution layer of width k works by moving a sliding window of size k over the sentence over each window in the sequence $[v(x_i); v(x_i + 1); ...; v(x_i + k - 1)]$. The filter function is usually a linear transformation followed by a non-linear activation function like ReLU or Sigmoid. Let the concatenated vector of the ith window be $w_i = [v(x_i); v(x_i + 1); ...; v(x_i + k - 1)]$, $w_i \in R^{k.d_{emb}}$. If we pad the sentence with $k-1$ words to each side, we may get either $m = n-k+1$ or $m = n + k + 1$ windows . The result of the convolution layer is m vectors. $p_1, ..., p_m, p_i \in R^{d_{conv}}$ where: $p_i = g(w_i W + b)$, g is a non-linear activation function, $W \in R^{k.d_{emb}.d_{conv}}$ and $b \in R^{d_{conv}}$ are parameters of the network. Each p_i is a d_{conv} dimensional vector, encoding the information in w_i . Ideally, each dimension captures a different feature. The m vectors are then combined using a max pooling layer, resulting in a single d_{conv} dimensional vector c.

$$c_j = \max_{1<i\leq m} p_i[j]$$

$p_i[j]$ denotes the j^{th} component of p_i . The effect of the max-pooling operation is to reduce the input size to the next layer and highlight the dominating value in a window. Ideally, each kernel will learn a particular sort of feature prediction. The resulting vector c is a representation of the sentence in which each dimension reflects the most dominant value with respect to some feature. c is then fed into a downstream network layers, perhaps in parallel to other vectors, culminating in an output layer which is used for prediction.

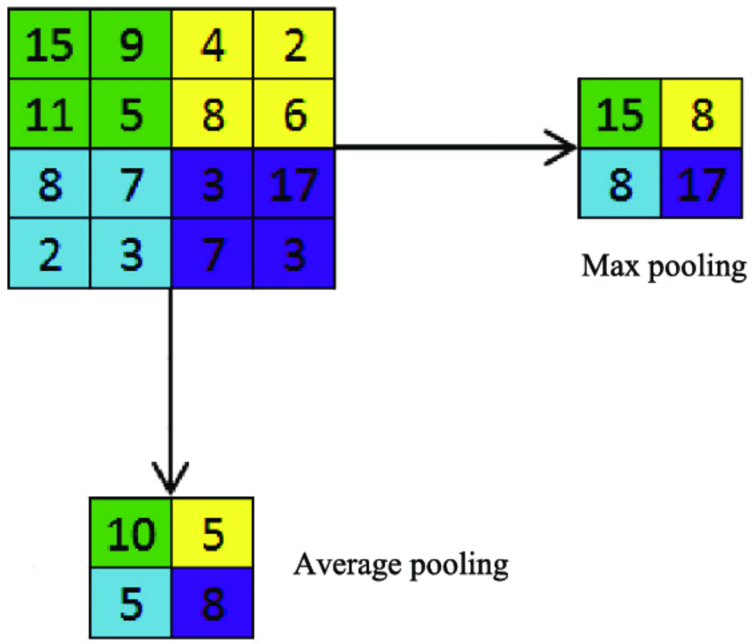


Fig. 8. Average and Max Pooling

Sometimes Average pooling is also used. Best pooling technique is generally determined by testing all the techniques.

8 Conclusion

Neural Network Algorithms are powerful tools for natural language processing applications. In this research, all the techniques that are predominatly used for Natural Language Processing applications are studied. With the help of these techniques we can solve problems like Fake News Detection, Abusive Posts/Tweets Detection, Rumor Propagation on Social Media can be solved.

References

1. Niall J Conroy, Victoria L Rubin, and Yimin Chen. Automatic deception detection: Methods for finding fake news. In *Proceedings of the 78th ASIS, Annual Meeting: Information Science with Impact: Research in and for the Community*, page 82. American Society for Information Science, 2015.
2. Yoav Goldberg. A primer on neural network models for natural language processing. *Journal of Artificial Intelligence Research*, 57:345–420, 2016.
3. Sejeong Kwon, Meeyoung Cha, Kyomin Jung, Wei Chen, et al. Prominent features of rumor propagation in online social media. In *International Conference on Data Mining*. IEEE, 2013.
4. 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.
5. Kai Shu, Amy Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *ACM SIGKDD Explorations Newsletter*, 19(1):22–36, 2017.
6. Eugenio Tacchini, Gabriele Ballarin, Marco L Della Vedova, Stefano Moret, and Luca de Alfaro. Some like it hoax: Automated fake news detection in social networks. *arXiv preprint arXiv:1704.07506*, 2017.