Exploring Deep Learning Models for Sequential Data

Summer Internship Project at Sapient Razorfish

Anirban Ray

7th May - 6th July, 2018

Introduction

Neural networks are widely used now a days. Besides Image Processing, Character Recognition, etc., they have different applications on sequential data, like *Forecasting*, *Stock Market Prediction*, *Natural Language Processing*, *Speech Recognition*, *Word Embedding*, *Frame by Frame Video Analysis*, etc. Let us first describe some of the applications in short.

Speech Recognition

Speech recognition develops methods and technologies that enables the recognition of spoken language into text by computers. Now a days, this is becoming very common with the use of Google Home, Amazon Alexa, Microsoft Cortana, etc.

Machine Translation

Machine translation investigates the use of software to translate text from one language to another. We use this often through Google Translate, Bing translate, etc.

Word Embedding

Word embedding is a case of sequence to sequence modelling. Given a text, the words, phrases or documents are encoded to vectors of real numbers. This is widely used in Natural Language Processing to learn the features of the available data set.

Approaches

All of the applications mentioned above has one thing in common. In all the cases, the inputs to the method, may be letters, words, or sentences are not independent. The consecutive units are dependent and in order to predict the output future unit, the past units should be considered. Hence, we need models which take into account this dependence.

Convolutional Neural Networks (CNN)

A CNN is a class of deep, feed-forward artificial neural networks, developed in the 1990's. These use a variation of multi-layer perceptrons designed to require minimal preprocessing. They also use shared-weights architecture and translation invariant characteristics. In short, CNNs perform by generalising the inherent features of the inputs, i.e. it learns those filters itself that in traditional algorithms were hand-engineered. Hence, these models excel in cases of image captioning and image classification. This independence from prior knowledge and human effort in feature design was a major improvement, and hence even now, these are the most commonly used networks.

There are three basic components of a CNN:

- Convolution Layer

- Pooling Layer
- Output Layer

The first layer extracts features from the inputs. After extraction, non-linearity is introduced using some activation function, usually *ReLU*. Then, if the extracted features are too large, it would be required to reduce the number of trainable parameters. Spatial Pooling (also called sub-sampling or down-sampling) reduces the dimensionality of each feature map but retains the most important information in the second layer. These can be of different types: Max, Average, Sum etc. Finally in output layer, fully connected layers are applied to classify the inputs into desired number of classes using *Softmax* activation.

The following picture helps to visualise the CNN architecture.

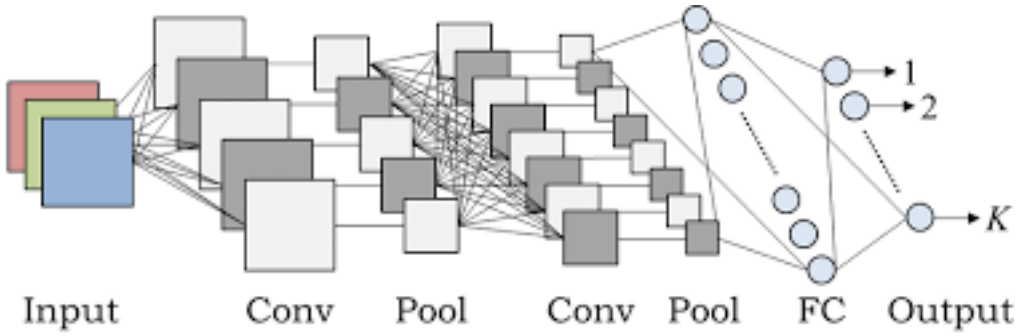


Figure 1: CNN Architecture

Recurrent Neural Networks (RNN)

However, there are two major drawbacks of these traditional networks in the context of sequential data. First of all, CNNs support inputs and outputs only of fixed size, whereas in most of the practical situations, this is not the case with sequential data. Often, they are continuously generated, for example time series data. In such cases, one cannot even apply a CNN model. But most importantly, CNNs consider that the inputs are not affected by each other, i.e. these models are based the assumption of independence of the inputs. This is completely against the concept of sequential data, and to handle these problems, RNNs were developed. For these models, there is no fixed number of hidden layers and inputs are fed to the network one by one and at each time step, the activations are formed using the input corresponding to that time step and the activation formed in the previous time step (thus the dependence between terms is preserved).

Speech Recognition

Traditional Approaches

In 1952 three Bell Labs researchers built a system for single-speaker digit recognition. Their system worked by locating the formants in the power spectrum of each utterance. In 1960, the source-filter model of speech production was developed, which considered speech as combination sound source and a linear acoustic filter, with the assumption of the independence of these two. There was various developments, but up to 2000's, almost all the methods faced the problems of small vocabulary, limited capability (for example, failure to do continuous speech recognition, etc.), too much dependence on the training data (e.g. suitable for only one speaker, etc.), etc. These models used combinations of hidden Markov models and feed forward artificial neural networks. In the new millennium, however, with the exponential improvement in computational power and large data sets, it became possible to implement deep learning methods.

Objective

There are several APIs for speech recognition are available, provided by Google, Microsoft, IBM, Sphinx, Nuance, Wit, etc. In this project, we shall be using the following two speech recognition methods available from the **Speech Recognition** module in *Python*:

- Chrome's Speech API

- Wit.ai

We wish to check the accuracy of these APIs and to find their limitations.

Measures of Accuracy

Our target is to see how all the APIs perform. For that, we shall be using labeled data. We shall try to detect the speech from these samples and comparing with the true text. Then we shall measure the the error in detection and that will help us to conclude with confidence. Before this comparison, we shall discuss how to compare two strings.

Levenshtein Distance

Levenshtein distance is a metric which depicts the minimum number of single character edits to get one string from the other. These edits are deletion, insertion and substitution.

Implementation

We have implemented this in *Python*. We shall slightly modify it to be the proportion of minimum number of single character edits required to get the true text from the recognized text to the total number of characters of the true text. This modification will facilitate the comparison, as then results for different references will be comparable. Also, we shall consider only the letters and space for string comparison.

Word Error Rate (WER)

Word error rate correspond to Levenshtein distance between two strings considering the words as the units of comparison, instead of characters. Since this is defined by the relative Levenshtein distance between two strings, this can be used as a measure for comparing APIs. It is worth to mention that it is much more common to use Word Accuracy, which is nothing but unity minus the WER.

Implementation

As before, we have implemented the word error rate also in *Python*. Now to minimize the errors generated by petty errors, we shall modify it in such a way that this ignores the cases and punctuation marks.

Calculations

Illustration

First, let us provide an example with an audio file of length 5 seconds. We shall detect the speech by both methods and also measure the time taken by our function to judge the efficiency.

Original Audio

```
Google:  "it's time for some invention hear the part of your car don't look back"
Wit:     "hey it's time for some intervention here step out of your car don't look back"

Time taken:  7.14 seconds
```

Observations

1. This example shows that the outputs from different methods vary between themselves, but not too much.
2. We also note that the speech recognition process is not at all efficient, as it takes more than thrice the length of the audio file.

Now, we shall study the effects of adding noise. We shall add five perturbations to this audio file and do the same as before for each of the modified files.

Perturbed by Dish Noise

```
Google:  "confidential hear frm you call don't look back"
Wit:     "it's time for some intervention here stuff out of your car in the back"

Time taken:  12.82 seconds
```

Perturbed by Bike Noise

```
Google:  "it's time for some invention here step Arabic I don't look back"
Wit:     "okay it's time for some intervention here stuff out of your car don't look back"
```

```
Time taken:  7.06 seconds
```

Perturbed by Pink Noise

```
Google:  "Ethan Pritam invention hear frm you call don't look back"
Wit:     "okay it's time for some intervention here stuff out of your car don't look back"

Time taken:  13.6 seconds
```

Perturbed by Tap Noise

```
Google:  "it's important invention here Departed you call don't mistake"
Wit:     "it's time for some intervention here stuff out of your car to look back"

Time taken:  13.7 seconds
```

Perturbed by White Noise

```
Google:  "it's time to come in attention here the party because don't look back"
Wit:     "it's time for some intervention here stuff out of your car to look back"

Time taken:  8.11 seconds
```

Observations

1. First of all, we note that adding perturbation causes the API's to detect wrong texts.
2. Also, we can note that the differences between the recognitions are now much more than the unperturbed case.

Study of Unperturbed Audio Files

Then, we shall perform the function for `k1 = 30` randomly selected files from `n1 = 104` speech samples by the user *rhys_mcg* of the **Tatoeba** database. Based on the results, we shall find out the average error made by both measures. This will provide an unbiased estimate of the true errors. We shall also perform tests of significance to see whether the errors are significant or not.

```
Average Errors:  [0.3878 0.1916 0.6221 0.4842]

Decisions:   ['Reject', 'Reject', 'Reject', 'Reject']
```

Observations

1. We note that there is notable discrepancy between the two methods in average error rates.
2. For the `k1` sampled files, Google's API has around 39% Levenshtein distance and 62% word error rate.
3. Wit's API beats Google's one comfortably with respect to both measures, with respective rates as 19% and 48%.
4. We should keep mind that the aforementioned results are obtained based on 30 sample files. So this strongly indicates to accuracy difference between the two methods. However, we shall not be attempting to test their difference between these methods now.
5. On the other hand, we wish to perform Student's t-test to test whether the errors are significant or not, assuming normality of the errors. At 5% level of significance, we find that both APIs are significantly erroneous with respect to both measures.

Study of Perturbed Audio Files

Now, we shall consider `k2 = 20` sample samples from `n2 = 1000` audio files and transcriptions available from **Voxforge** corpus. For all these sampled files, we shall perturb by five background noises available from **Speech Command Dataset** provided by *Google*. Then we shall again find out the average errors and test for their significance.

```
Average Errors:
 [0.28063306 0.71946963 0.66862643 0.79289759 0.80104164 0.71013695
 0.12611055 0.69480752 0.62038353 0.72771856 0.72865124 0.70723496
 0.48614193 0.87865842 0.80640339 0.89505037 0.92556319 0.82620421
 0.32446107 0.80368939 0.79274617 0.87794534 0.87112637 0.82986264]
```

Observations

1. From the results for the perturbation case, we can see that not only all the error rates are certainly significant, the raises in the error rates due to perturbation are evident without the need of any statistical test.
2. The results of Student's t-tests also validate conclusion.
3. We can also note that in all the cases, the perturbations caused by "running_tap.wav" and "pink_noise.wav" are very high.
4. The background noise causing least error is "exercise_bike.wav".
5. Although we need much more number of perturbations to state with confidence, but the results lead us to suspect that the volume of the noises affect more seriously than their type. Obviously, this is in parity with common sense.
6. Finally, it can be seen that after perturbation, both Google and Wit perform almost similarly. This may lead us to conclude that Google's API is less affected by perturbation, as Wit's API outperformed significantly without any additional background noises.

Limitations

We can suggest some limitations of speech recognition methods as follows:

1. **Environmental Factor** - Low volume, background noises and other audio disturbances affect the performance of the methods.
2. **Between Speaker Variability** - There are lots of widely different accents, which creates obstacles for generalization by the models.
3. **Within Speaker Variability** - Depending on time and speaking style, the same speaker's pronunciation of the same word may vary, which will be detected by the frequency analysis by the model.
4. **Dictionary** - The quality of methods depend massively on the data set used for training and the built - in dictionary of the model. Absence or low proportion of proper nouns or uncommon words result in the failure of the model to recognize these words. Also, if the vocabulary is increased, it affects the accuracy of the models adversely.
5. **Ambiguity** - People use different idioms, synonyms, etc. to convey the same. Rephrasing sentences affects the performances.
6. **Homophones** - Speech recognition models also have the problem of confusion with words with same or very close pronunciation.
7. **Slow** - Till date, the APIs have the problem of high latency. It takes time for them to process the audio, and hence real time recognition cannot be achieved. Especially at the beginning, lags can be noted quite frequently. Hence, models work quite well for isolated or discontinuous speech, but for long continuous speeches, models perform really badly.

Apart from these, there are also some inherent limitations of the RNNs:

1. It is very difficult to train a RNN.
2. RNNs are quite capable to capture dependence on just a few previous inputs, but they are not suitable for deep models. Though theoretically it is not the case, but practically this happens because of the **Vanishing Gradient Problem**. It is caused by the choice of activation functions. Commonly used ones are hyperbolic tangent and logistic sigmoid functions. Derivatives of both of these functions can be nothing but proper fractions. But the model gets updated by **Back Propagation**, which uses these through **Chain Rule**. Hence for long-range dependencies, the influences, as measured by gradients, become almost negligible.
3. On the other hand, sometimes RNNs face **Exploding Gradient Problem**, where norm of one or two derivatives may be very large and hence the model becomes unable to learn long term temporal relations.

Solution

To get rid of these problems, **Long Short Term Memory** models and many variations of those are proposed, especially **Gated Recurrent Units**. These methods are capable of learning long term contexts by creating memory cells. We will go in details of LSTMs in a later section.

Machine Translation

Traditional Approaches

Machine translation models can use methods based on linguistic rules. These methods were very much used during the Cold War In this method, first the text is resolved into its component parts and those are translated, from which the target language is generated. However, these methods had high error rates. As an alternative, **Statistical Machine Translation** (SMT) models were developed. This was a serious advancement replacing the rule based methods with data driven methods, and these models were the best even at the start of this decade. These models work though highest posterior probability approach. Considering the text translated up to the current step, these models attach probability to all possible ways for the future texts and choose the most probable one. This probability assignments are done based on the corpus on which it is based. It may happen that suitable corpora are not available for obscure language pairs (for example, *Catalan-English*), or even if they are available, it is not often worth to invest resources to train the model. Hence, for those pairs, these models first translate to another more relatable intermediate language (for example, *Spanish*) with which corpus for both the languages are available.

Neural Machine Translation (NMT)

Proposed just in 2014, neural machine translation models use of neural networks to learn a statistical model for machine translation. The key benefit to the approach is that a single system can be trained directly on source and target text, no longer requiring the pipeline of specialized systems used in statistical machine learning. These models use recurrent neural networks organized into an **Encoder - Decoder** architecture that allow for variable length input and output sequences. These models use two recurrent neural networks. The first RNN encode the source text into an internal representation called the context vector, which can be decoded by another RNN. In principle, these context vectors can be decoded to any language using a suitable decoding RNN.

Objective

There are several API's for machine translation, provided by Google, Microsoft, Yandex, Baidu etc. here, we shall consider the following two:

- Google Translate
- Yandex

We shall also select the language pair **English - French**. Then applying both the API's to translate numerous texts from one language to the other and checking whether the results match or not, we wish to check the performance of the two API's.

Measures of Accuracy

It is harder to measure the accuracy of a machine translation model. Suppose a text of some language is given to two human beings and they are asked to translate it to another language. Now, it is quite likely that the two translations won't be identical, however they will be very similar and will, of course, convey the same meaning. This is because that translation is not something that can be done in a unique way. Hence we need to bring into account the concept of *Semantic Similarity*.

Semantic Similarity

Semantics is the linguistic and philosophical study of meaning inherent at the levels of words, phrases, sentences and texts. Semantic similarity is a metric defined over a set of documents or terms, where the idea of distance between them is based on the likeness of their meaning or semantic content.

Cosine Similarity

One of the measures of semantic similarity is *Cosine Similarity*. This is actually a measure of similarity between two non-zero vectors that measures the cosine of the angle between them. Since cosine is a decreasing function over the set of acute angles, i.e. over $[0, \frac{\pi}{2})$, higher cosine similarity implies that the two vectors are more similarly oriented. It should be noted that this helps in judging the orientation of the vectors, not measurement. Hence, it is common to apply this over normalized vectors. For two vectors $\vec{a}$ and $\vec{b}$, it is given by,

$$S_c = \frac{\vec{a} \cdot \vec{b}}{|| \vec{a} ||| \vec{b} ||}$$

Now, obviously we cannot apply this measure for translated texts directly. We first need to convert them to vectors. This can be done through word embedding. For that, we will use **tf-idf**. This is a numeric statistic which reflects the importance of a word to a document in a corpus. Without going into details, this is calculated by multiplying two statistics, one being *Term Frequency* (measures simply the frequency of a word), and the other being *Inverse Document Frequency* (measures the specificity of a word in that document). After encoding the translated texts by different API's in vectors, we shall compute the cosine similarities between each possible pairs. Now, if it is found that they cannot be considered to be similar, then it will indicate that the at least one the API's of the pairs is not performing as expected.

Implementation

To implement the above, we shall use the functions available from the **Scikit Learn** module of *Python* for both vectorisation of texts and computation of computation of cosine similarity.

Calculations

To judge the two API's, we shall use the **European Parliament Proceedings Parallel Corpus 1996-2011**. Before translating, we shall perform some data cleaning as follows:

- Tokenising text by white space.
- Normalizing case to lowercase.
- Removing punctuation from each word.
- Removing non-printable characters.
- Converting French characters to Latin characters.
- Removing words that contain non-alphabetic characters

Then, we shall take `k3 = 50` random samples from `n3 = 2007723` sentences available in the corpora. For each of these randomly selected English sentences, we shall translate by both API's and compare their similarity with the provided French translation. Based on the results, we shall note the average cosine similarities, which will once again provide unbiased estimate of the true population ones. Finally, we shall perform test of significance.

```
Average Cosine Similarities:  [0.4338 0.4164]

Decisions:   ['Reject', 'Reject']
```

Observations

1. We note that the two API's perform almost similarly.
2. Google has an average cosine similarity of around 0.43, where Yandex has that of around 0.42.
3. Since these results are based on 50 samples, which is quite a large sample, it may appear that Google's translator perform better, albeit just marginally, but we shall rather hold that conclusion for now.
4. Assuming normality, if we perform Student's t-tests, we note that cosine similarities for both measures vary significantly from 1, the ideal value in case of perfect translation.

Limitations

Some limitations of the machine translation models are as follows:

1. It is important that translators understand the context of the text to accurately portray it. Traditional machine translation makes no consideration of the context, which may lead to the creation of unintelligible phrases on a constant basis. Recurrent neural networks consider the context, but fail to do so for long texts, just like the same problem faced in speech recognition.
2. There are inevitably a multitude of local terms and expressions. For example, a language spoken in a country may differ in many ways from the same language spoken in its central or eastern/western parts, with many expressions, idioms and idiosyncrasies being unique to each region. Similarly, there are lot of frequent profession specific terminologies, which may have very different meaning or no meaning at all in all other contexts. It is not possible for machine translation models to take into account these cases. The models are simply incapable of considering the cultural traditions of the source and target languages to provide accurate translation.

3. The NMT models have a very steep learning curve with respect to the amount of training data. For large corpus, these models perform very well, but take a long time to be trained well not to give results of poor quality.
4. Use of pivot languages lead to higher error rates because of the potential lack of fidelity of the information forwarded in the use of different corpora.

Designing Neural Networks

Speech Classification Models

We now wish to have our own neural network which will be able to perform well for even after perturbation. For that, we will need to train the data set on perturbed versions of the labeled audio files along with the original versions. We plan to do the following:

1. Train a network on unperturbed files only and check its performance
2. Train a network on perturbed files only and check its performance
3. Train a network on both unperturbed and perturbed files only and check its performance

It is easily expected that the third network will perform the best among the trio for new speech files. However, it is not quite intuitive to suggest which one of the first two networks will perform better. In the next step, we will do some adversary testing. Though both of the first two models are expected to perform significantly poor, one may think the second one may be slightly better, considering its exposure to different types of training data along with repetitions.

Implementation

Before delving into implementation, we must note that theoretically CNN should perform better in this situation than RNN. The most effectiveness of CNNs lie in their abilities to learn statistical regularities, which, in this case, can be extracted from many pronunciations of the same words and hence in the corresponding MFCCs.

Mel-frequency Cepstral Coefficients are coefficients that collectively make up a the mel-frequency cepstrum, which is a representation of the short-term power spectrum of a sound, based on a linear cosine transform of a log power spectrum on a nonlinear mel scale of frequency. So, we will now proceed to train a CNN for speech classification.

As said before, we will train networks of the above architecture on unperturbed, perturbed and mixed files. Then we will check their performances in three testing data set: unperturbed, perturbed and mixed.

For all these situations, we will do the following:

1. Read the audio data and convert it to MFCC using *librosa* module of *Python*.
2. Split the data set in `60:20:20` ratio to generate training, validation and testing subsets.
3. Train the networks over the training subset using *keras* module of *Python*.
4. Check performance over the testing subset

Procedure

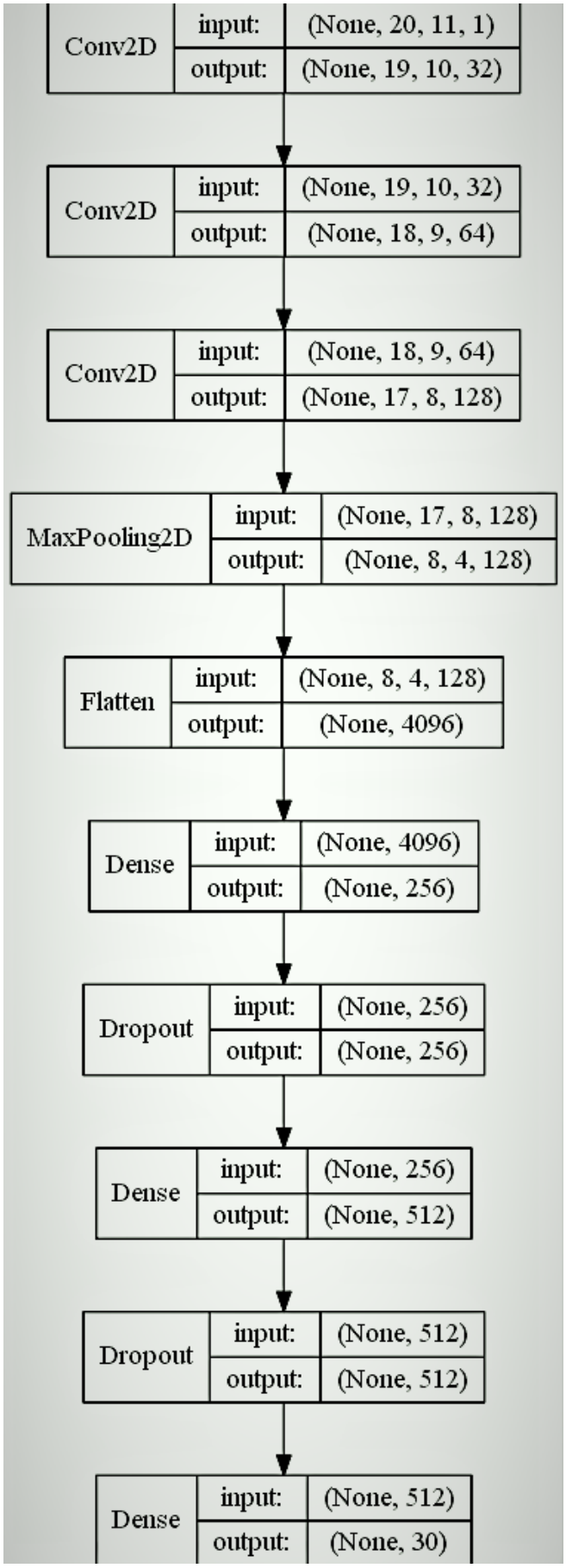
We will proceed to do this in two steps.

First, we will train a normal speech classification model on a larger data set and and note its performance. In the next step, we will choose a smaller data set and on this data set, we will perform the adversarial testing using the architecture and hyper-parameters from the previous model.

Step 1

Here, we will use the **Speech Commands Dataset** provided by *Google*. There are `64721` samples of `30` different words.

After trial and errors, we created the following *CNN* architecture:


Conv2D
input: (None, 20, 11, 1)
output: (None, 19, 10, 32)
Conv2D
input: (None, 19, 10, 32)
output: (None, 18, 9, 64)
Conv2D
input: (None, 18, 9, 64)
output: (None, 17, 8, 128)
MaxPooling2D
input: (None, 17, 8, 128)
output: (None, 8, 4, 128)
Flatten
input: (None, 8, 4, 128)
output: (None, 4096)
Dense
input: (None, 4096)
output: (None, 256)
Dropout
input: (None, 256)
output: (None, 256)
Dense
input: (None, 256)
output: (None, 512)
Dropout
input: (None, 512)
output: (None, 512)
Dense
input: (None, 512)
output: (None, 30)

Training Data \ Testing Data	Unperturbed	Perturbed	Mixed
Unperturbed	97.33%	7.07%	22.11%
Perturbed	6.00%	89.33%	78.39%
Mixed	97.67%	90.67%	88.50%

Table 1: Categorical Accuracies

Figure 2: Classification Architecture

We will choose our hyper-parameters as the following:

- No. of Epochs: `100`
- Batch Size: `100`
- Loss Function: `categorical_crossentropy`
- Optimizer: `Adam`
- Metric of Accuracy: `categorical_accuracy`

However, we have stopped the training as soon as the validation loss stops getting improved beyond three places after decimal for 5 consecutive epochs. Then we get the following results:

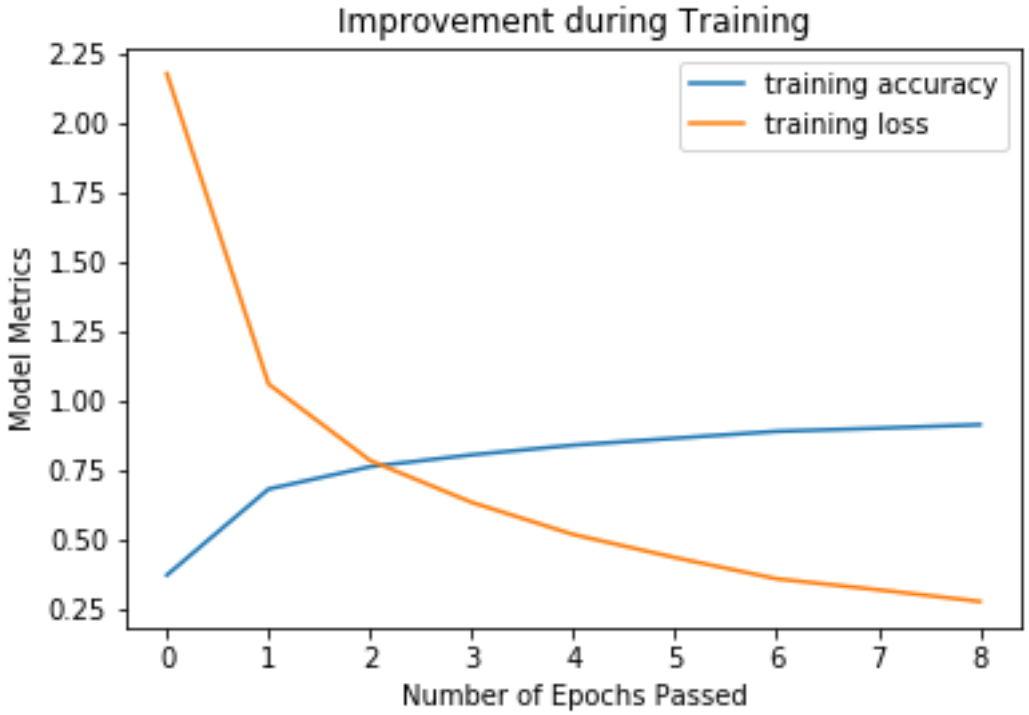


Figure 3: Improvement in Speech Classification Model

`Model categorical_accuracy: 79.96%`

So the model detects the correct class of the testing data set in approximately 80% of the cases. That's quite good and so we will proceed to the next step.

Step 2

Here, we will use the **Free Spoken Digit Dataset**. Here, on each of the `10` digits, we have `50` recordings by `3` speakers each. We will train three models using 60% of the unperturbed, perturbed and mixed data sets respectively, with validation on 20% of the corresponding data sets. Finally, we will test their performances on the 20% of all the data sets. Here, we present the table with the categorical accuracies of the three models in different situations. As expected, performances of the first two models in adversarial situations are extremely poor. We should mention that the significantly better performance of the second model compared to the first one for mixed testing data is not at all surprising, as this is due to the much larger (five times, to be precise) training data set for the second model compared to that of the first data set and the fact that the data set used in third case is the combination of the first

two data sets. On the other hand, the third model's performance is highly satisfactory in all cases, and notably its performance is better (albeit slightly) than the other two models in their favourable situations as well.

Machine Translation Model

This is a case of pure sequence to sequence translation, and hence CNN will be heavily outperformed by RNNs. However, since translation requires that the models should remember the context, we will use LSTM directly. Let us first discuss about this architecture briefly.

Long Short Term Memory Networks (LSTM)

Before delving into the implementation, let us first discuss the way LSTMs perform. LSTMs are developed as a modification of RNNs to regulate the ability to add or remove information to an additional memory cell. These are achieved by using **Forget Gate**, **Input Gate** and **Output Gate**. Gates help to regulate the extent of information using a single sigmoid net layer. The following are just the basic idea of a LSTM network:

1. After time step $t-1$, the network has updated the memory cell to C_{t-1} and it is provided to the next time step.
2. At time step t, the cell takes activation of the previous time step h_{t-1} and input of the current time step X_t as the inputs. Both of these are fed to all of the three gates.
3. First, the forget gate determines how much information should be retained from the stored cell state. So it performs the sigmoid operation based on the two inputs and store the result as f_t.
4. Second, the input gate decides how much information should be updated based on the inputs using another sigmoid layer and store it as i_t.
5. Alongside, the inputs are fed through tanh layer to make updates to the stored cell state. This is stored as $\widetilde{C}_t$.
6. Now, point wise multiplication of C_{t-1} and f_t gives that part of the previous cell the network wishes to keep, and the point wise multiplication of $\widetilde{C}_t$ and i_t results in the amount of information that should be added to the cell state. Addition of these two will form the updated cell state C_t.
7. Then, the output gate finds out how much of the current cell state the model will show as an output. A sigmoid layer filters out the relevant portion based on the current input and store it as o_t.
8. Finally, C_t is filtered by a tanh layer and its element wise multiplication with o_t yields the current activation h_t. This will be used as an input to the $(t+1)^{th}$ time step.

The following picture helps to visualise the above steps.

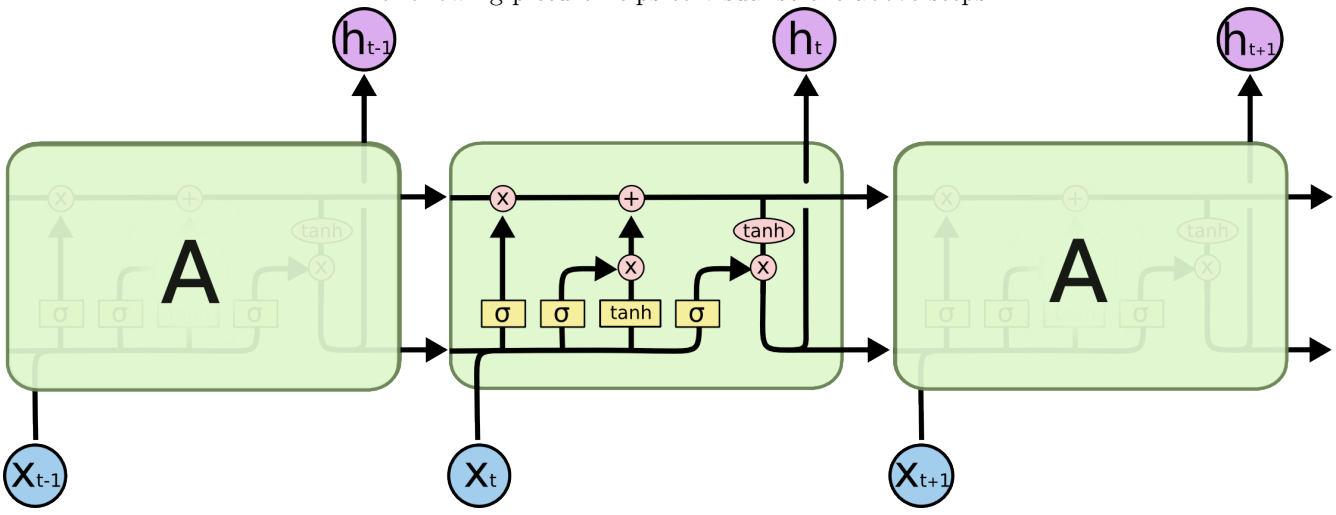


Figure 4: LSTM Architecture

Implementation

Now, we want to train a network which will translate French to English. The data set used is available from *Manythings*, provided by the *Tatoeba* project in forms of tab-delimited bilingual sentence pairs.
Here, we will use the Encoder - Decoder network. The architecture of the model is given below:

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 18, 256)           2143232
_________________________________________________________________
lstm_1 (LSTM)                (None, 256)               525312
_________________________________________________________________
repeat_vector_1 (RepeatVecto (None, 11, 256)           0
_________________________________________________________________
lstm_2 (LSTM)                (None, 11, 256)           525312
_________________________________________________________________
time_distributed_1 (TimeDist (None, 11, 4837)          1243109
=================================================================
Total params: 4,436,965
Trainable params: 4,436,965
Non-trainable params: 0
_________________________________________________________________
```

Though the data set contains 154883 sentence pairs, due to resource constraint, we will consider only the first 35000 pairs, but, of course, at the cost of significant degradation in translation quality.

Here, we train the model up to 100 epochs. However, we stop the training process if validation loss stops getting improved beyond 3 places after decimal or remains unchanged for 10 consecutive epochs. We will be using the **Adam** optimizer, **Categorical Crossentropy** loss function and **Categorical Accuracy** as the metric of accuracy. We shall apply batch gradient descent with batches of size 64.

After the completion of training based on 21000 pairs with validation on 7000 pairs, we test the performance of the model on the remaining 7000 pairs. We get the following results:

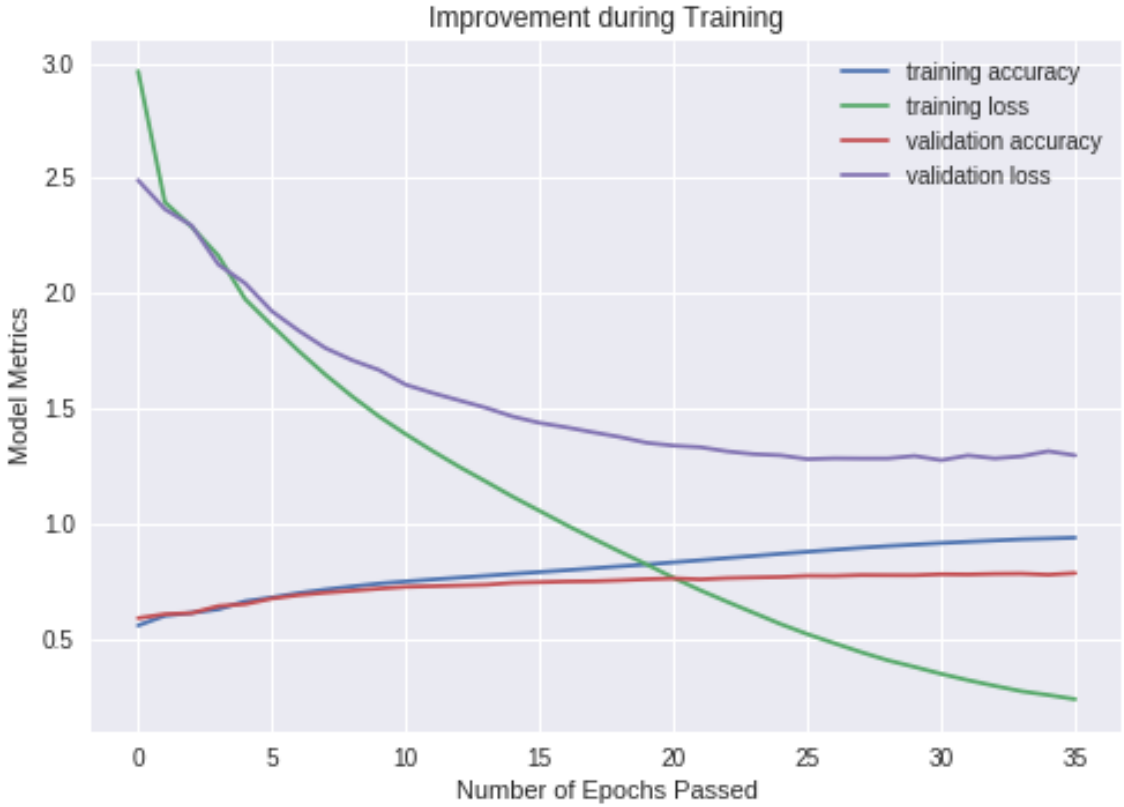


Figure 5: Training Progress

```
Average Cosine Similarity: 0.46
Model categorical_accuracy: 77.95%
```

We have noted that the outputs have started to become meaningful, and in most of the cases, the translations are close to the expectations, even though structurally not identical. Though the value of average cosine similarity indicates moderate semantic similarity, but the translations are certainly conveying the sense (or at least, part of it) of the source sentences. We can surely hope to improve significantly, had we used the full data set during training.

Attention Mechanism

Theoretically speaking, LSTMs should not have any problem for long sequences. But in practical situations, that is not the case, though it is not that much serious as RNNs. Most of the LSTMs work by encoding the input vector, no matter however large or small it be, in a fixed length context vector, which is somewhat counter-intuitive. Again, LSTMs learn the context mostly from the vicinity, i.e. the order of the inputs plays a role in their effectiveness. But for long inputs, if the model detects that the context is from far too past, then though it tries to trace back and retrieve the relevant memory, but it is resource consuming. There are some situations where there are some workarounds to the problem, like reversing the input, or feeding the input sequence twice. To avoid these, Dzmitry Bahdanau proposed the concept of *Attention*, which is, loosely speaking, letting the model know, or rather learn, where to look for to perform its task with more efficiency.

Idea behind Atention

When analyzing a situation, most of the time people don't analyze the whole situation at once. They focus on small parts separately while ignoring the rest of the problem, and then combine the observations on all the parts. Following this behaviour of human beings, neural networks can be trained in such a way that they will perform by studying a relatively small portion of the data with high priority, and considering the rest of low priority and then switch the portions iteratively and finally combining everything. This can be divided in following steps:

1. Based on the previous step, choose a part of the input and read it.
2. Extract available information from the chosen part and store it.
3. Predict which part of the input should be read next.
4. Combine the previously available and current information.

Applications

Attention mechanisms have a wide range of applicability. Let us explore two of these below:

Machine Translation

Let us review our neural machine translation model. It uses an encoder - decoder model. The performance can be visualised by the following image.

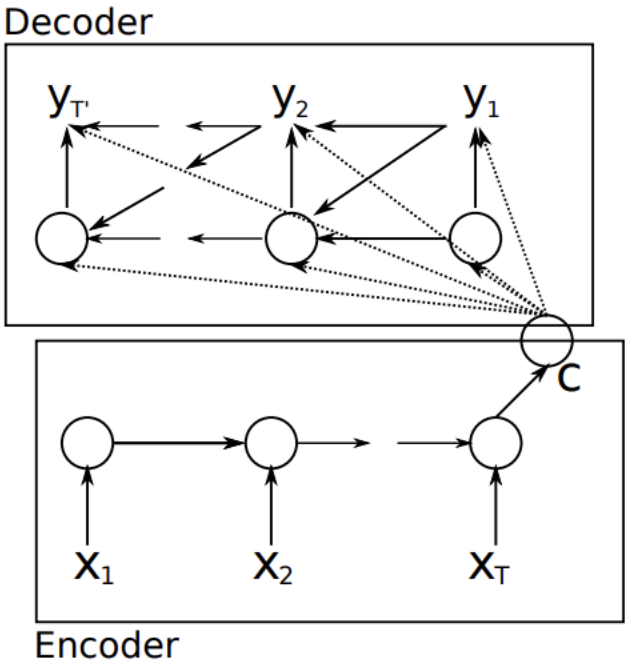


Figure 6: Encoder - Decoder Network

Now, here the understanding is that all the information in the input is stored in the last hidden state, and based on this particular hidden state, the model predicts the output one by one. So h_3 must encode everything we need to know about the input sentence. Instead of that, in attention mechanism, the model chooses a part of the input sentence to translate. Here, it considers a weighted combination of all the input states directly from the encoder, where the weights are proportional to the importance of the input state to predict the output state. This can be visualised from the following diagram.

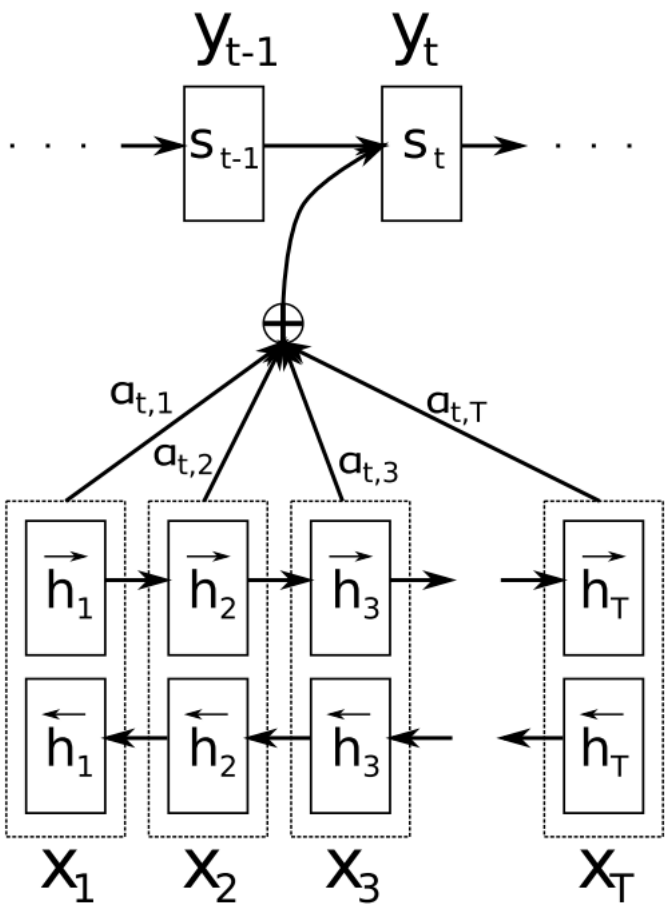


Figure 7: Encoder - Decoder network with Attention Mechanism

The following picture depicts an example of the attention weights of the input sequence to determine the output sequence.

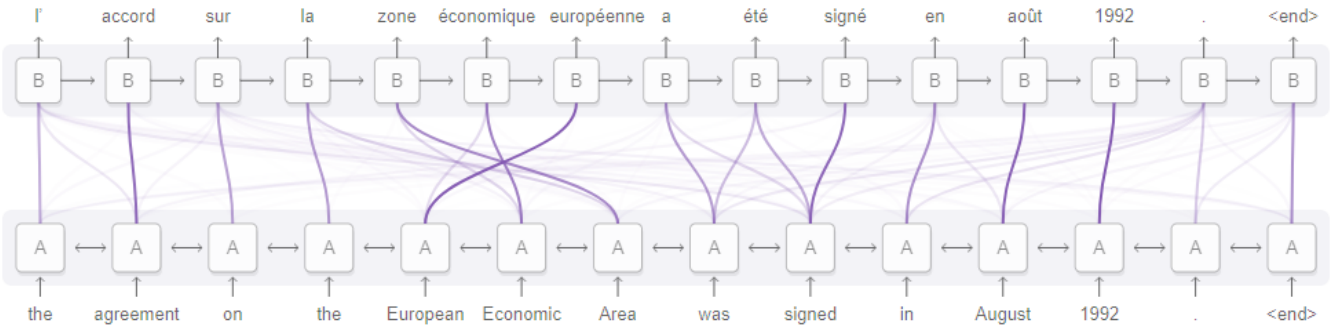


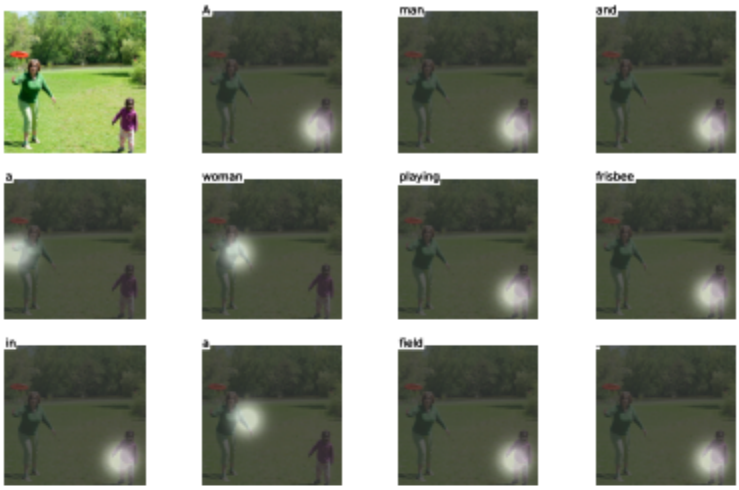
Figure 8: Attention in Neural Machine Translation

Image Captioning

In the paper *Show, Attend and Tell*, the authors have implemented both types of attention mechanisms to the problem of generating image descriptions. They used a CNN to "encode" the image, and a RNN with attention to generate a description. By visualizing the attention weights, we interpret what the model is looking at while generating a word.

In this context, attention can be applied in two ways, *Hard* and *Soft*. Hard attention uses the *argmax* function to select that encoded state with the most attention weight. Arg-max being non-differentiable, this is difficult to implement. On the contrary, soft attention uses *softmax* over all the states weighted based on relevance. Since this

is differentiable, back-propagation can be used with it.
Here's an example of its application in image captioning.



(a) A man and a woman playing frisbee in a field.

(b) A woman is throwing a frisbee in a park.

Figure 9: Generating Image Description: (a) Hard (b) Soft

Studying the effect of Attention Mechanism

Now, we want to create our own network to check the effectiveness of this mechanism. In context of sequential data, attention is mostly used in the context of neural machine translation. Hence, we wish to create one such model implemented with attention. This time we will use **GRU** to translate German to English.

Gated Recurrent Unit (GRU)

GRUs are actually a variation of LSTMs. It combines the forget and input gates into a single**Update Gate**. It also merges the cell state and hidden state. Hence, it can control the flow of information just like LSTM, but without having to use a memory unit. Being much simpler than standard LSTM models, GRU has been growing increasingly popular now a days. The following image provides nice visualiastion of a GRU unit.

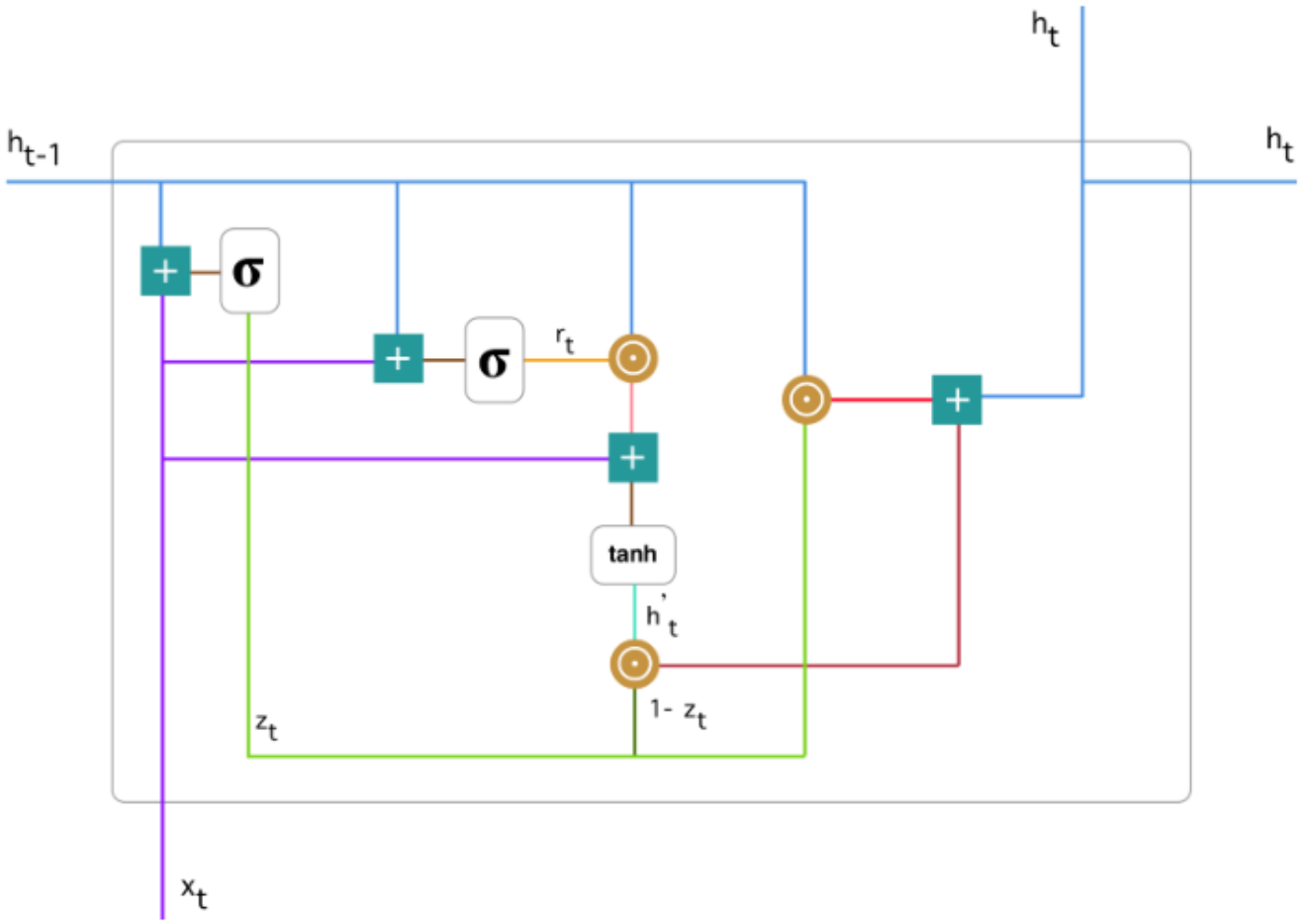


Figure 10: GRU Architecture

Implementation

We will use the same data set as before. We will use the implementation provided by **Google**. We will train both the models for at most 20 epochs, provided the validation losses continue to improve at the order of at least three places after decimal. We shall apply batch gradient descent with batches of size 64. Here, we have used all the 35000 pairs during training, and have also used **Teacher Forcing**.

To illustrate, let's present an example now.

```
French Sentence: Je cherche de l'eau.
English Translation: I am looking for water.
Model Prediction: i ' m looking for water .
```

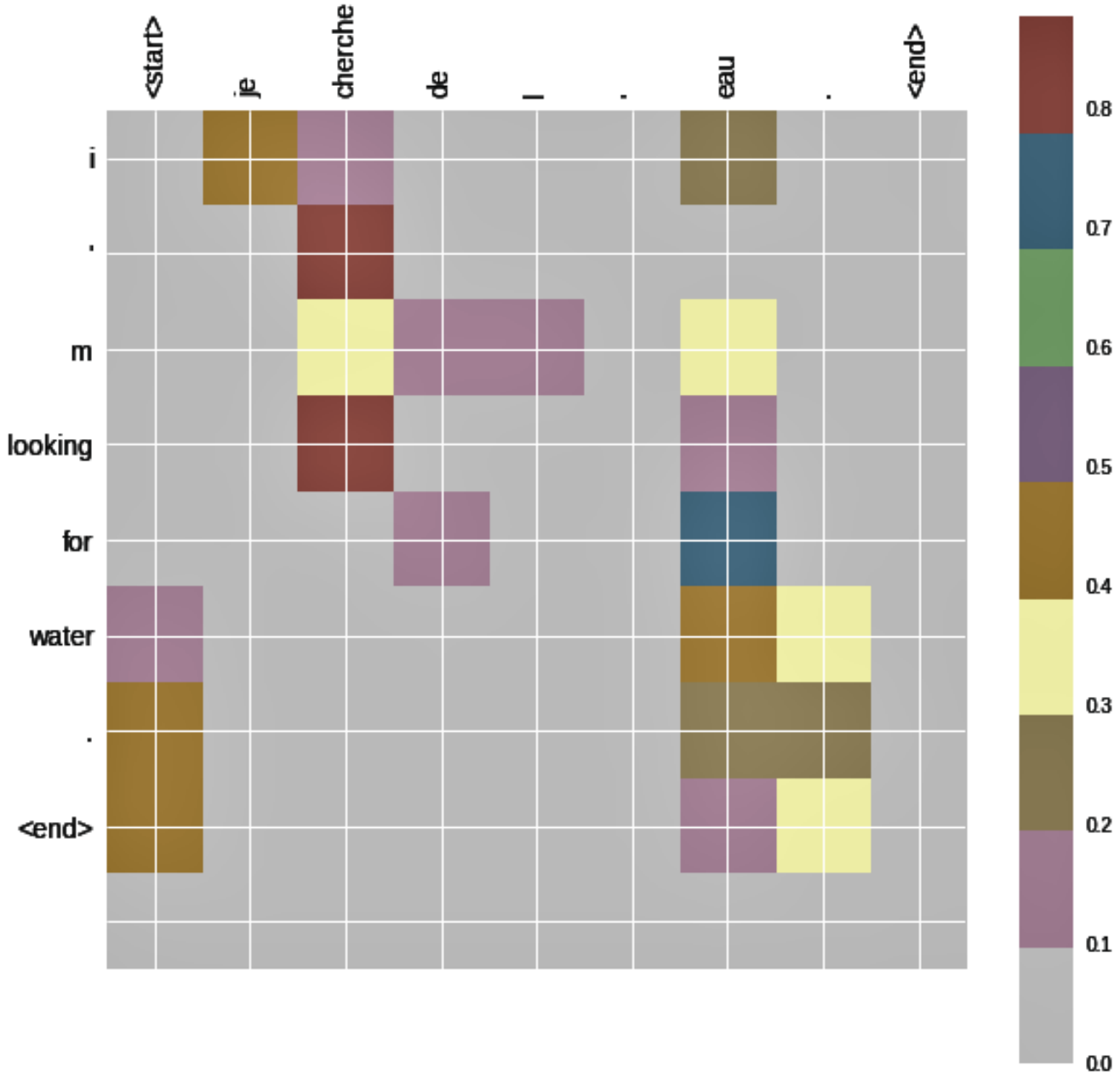


Figure 11: Attention Plot

Now we compare the performances using a few arbitrary sentences, and the results show that attention makes the model more accurate.

Limitations

However, we all know that there is nothing perfect, and attention is no exception. The problem is the very question it was aimed to solve. What we, the human beings, do is essentially focusing on something and neglecting others. This helps to save the computational resources. But attention models fails to do so and instead they focus on everything and then tries to decide on which it should focus. So instead of choosing what to attend, the model instead chooses what part to retrieve from the internal memory. For large inputs, this becomes prohibitively expensive. Hence, attention models are highly time consuming to train. There are some recent proposals of some alternative approaches, such as *Reinforcement Learning Neural Turing Machines*. However, we will not go into these in the current project.

Input	LSTM without Attention	Expected	GRU with Attention
va !	go .	go .	go .
remets - le en place !	put it back .	put it back .	hand it back .
j ' ai fini par l ' emporter .	i finally winning .	i finally won .	i finally won .
j ' ai vu le match .	i saw the yesterday .	i saw the game .	i saw the game .
il me faut me battre .	i have to fight .	i have to fight .	i need to fight .
telephonez au 1 1 0 immediatement .	tom used to to 0 . .	dial 1 1 0 at once .	dial 1 1 0 at once .
ils vont faire des emplettes .	they go shopping .	they go shopping .	they go shopping .
je vous ai laisse une note .	i saw you a note .	i left you a note .	i let you a note .
qui a fait cette tarte ?	who made this pie ?	who made this pie ?	who made this pie ?
je fais de mon mieux .	i ' m best best . .	i ' m trying my best .	i ' m trying my best .
tu es toujours vivante .	you ' re still alive .	you ' re still alive .	you ' re still alive .
ca peut etre difficile .	it may be difficult .	it can be difficult .	that can be difficult .
vous etes fort effronte .	you ' re very forward .	you ' re very forward .	you ' re very forward .
je vais le garder avec moi .	i ' ll to to him .	i ' ll keep it with me .	i ' ll guard with me .

Table 2: Comparison of Machine Translation Models

References

- colah's blog
- Free Spoken Digit Dataset
- Machine Learning Mastery
- Manythings
- Medium
- Towards Data Science
- Tatoeba
- Stack Overflow