CSCE 421 Final Project

Nikita Udodenko
Computer Science
Texas A&M University
College Station, TX, USA
nudodenko99@tamu.edu

***Abstract*—This document is going to explain what Deep Neural Network is and what it is used for. It will briefly describe two different types of Neural Networks, CNN and RNN. This document will cover where these two models are used, and what data they require. After that it is going to explain what is LSTM and what is the difference between LSTM and RNN models. After all of these explanations I am going to explain how to build an RNN model that will use 30 characters as input sequence and predict the next 10 characters. I will cover all the aspects of building a model, training the model, and preparing data for a model. Besides that, I will provide the accuracy of my model as well as different issues that I ran into.**

I. Introduction

Before we start talking about my implementation of the model, lets talk about what DNN (Deep Neural Networks) is and what is it used for. According to [1] "the DNN are extremely powerful learning models that achieve excellent performance on difficult problems such as speech recognition and visual object recognition, DNNs are powerful because they can perform arbitrary parallel computation for a modest number of steps". Think of a human brain as an example. Human brain contains neurons that are connected to each other. We can perform difficult tasks as recognise different faces, understand speech, understand that different people have different voices and see the difference between the different shapes and figures. Besides these tasks there are a lot more we can think of, for example we can think quickly, come up with logic answers, connect related parts, find patterns and notice different things very fast. Similar idea applies to the Neural Network. It contains neurons that are connected to each other. And when we train the model it learns how to establish those connections. It will learn how to manipulate incoming signals to come up with the best answer. The existing neurons will receive a signal and pass it on to other neurons and once it finds the best solution it will generate an output signal.

II. Types Of Neural Networks

Lets talk about two different types of Neural Networks that I learned in CSCE 421.

A. Convolution Neural Networks

CNN (Convolution Neural Networks) is made primarily for the image recognition. It is going to take the image as an input and depending on the task execute the command. For example we can train the model to recognize faces. We will teach the model by giving it a lot of different images of faces and it will learn how to identify a face by looking at different details. What are some common things that each face has? Eyes, nose, mouth, ears. It will start looking for those details and if the model is not going to find them then it will say that it is not a face otherwise if the model is going to find those details then it will decide that it is a face.

Another great example would be the classification between different images of different objects. Think about a shoe and a t-shirt. They are two different object, but they do not have any special identifiers on them like faces have. The t-shirt can have a painting on it or it can be clear, the shoes can be of different styles or they can have different symbols on it and etc. You will probably think that identifiers are important, but not in this case. A t-shit has one shape and a shoe has completely another shape. They are two completely different shapes and the model is going to recognise that. It does not matter if the t-shirt or shoes have unique identifiers as long as their shapes are different then the model can classify those two objects as different. According to [2] "the Convolutional neural networks are the go-to networks for image recognition tasks because they are well suited for detecting special patterns".

B. Recurrent Neural Networks

RNN (Recurrent Neural Networks) is made primarily for the sequential data. Hence the name recurrent. It is used primarily for the natural language processing and speech recognition. The next output in RNN is determined based on the previous computations. According to [3] one great drawback of the RNN is the vanishing gradient problem. It means that the neural network model is not going to be trained well. This situation can occur to the neural network that has a huge amount of layers, that are necessary to process big and complex data. The solution to this problem is the LSTM (Long Short Term Memory). LSTM is a type of RNN, but they can learn long-term dependencies. According to [4] "LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn". LSTM does the same thing as RNN, but better and more efficient. The main difference between LSTM and RNN is the gating mechanism. Take a closer look at Figure 1. The LSTM gating functions are going to be learned together with the weights and based of the current computations and the previous state it will figure out the amount of information that it thinks is necessary.

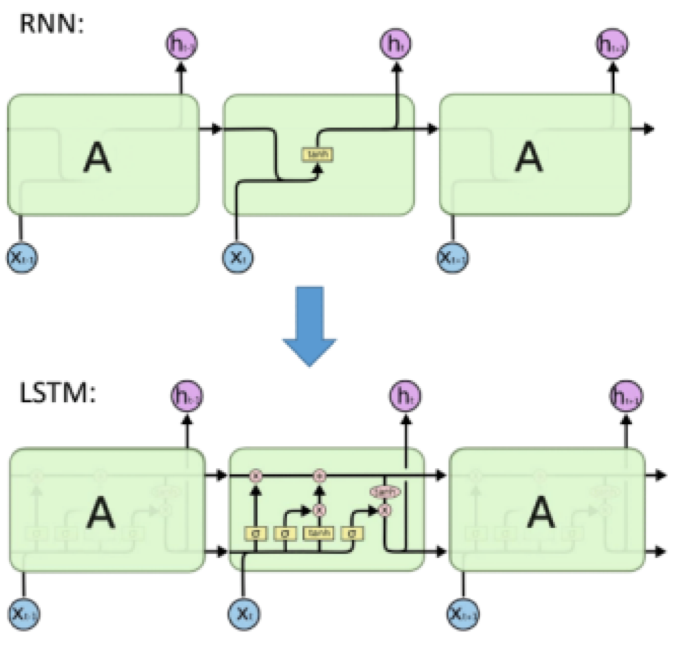

Fig. 1. RNN vs LSTM

III. Project Description

Now, after we talked about what Deep Neural Network is and described different types of neural network, lets talk about the project that I was working on. Given a very big amount of data that consists of different characters (there are special characters, English characters, and characters of a lot of other different languages) I need to build a model that is going to take 30 characters as input data and predict the next 10 characters. For this project I was building a Recurrent Neural Network, because we are given a sequence of characters and we are trying to predict another sequence of characters. Meaning that the data is sequential. Take a look at Figure 2. According to [1] ”model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. The model stops making predictions after outputting the end-of-sentence token. Note that the LSTM reads the input sentence in reverse, because doing so introduces many short term dependencies in the data that make the optimization problem much easier.”. While working on this project I applied similar logic, but instead of taking 3 input characters and predicting 4 output characters, my model is going to take 30 input characters and predict 10 output characters. For this project I used PyTorch, because we had previous assignments in it and I am familiar with it. The project was split into data preparation, building an RNN model, and training the model.

IV. Data Preparation

Given the data, I needed to read it, create a bunch of character arrays, format it, create a unique character array, get indexes of those characters from the unique character array, convert the character arrays with the corresponding indexes from the unique character array, split into input and output sequence, shape the data, encode the data using one hot encoder and split the data into train, test, validation.

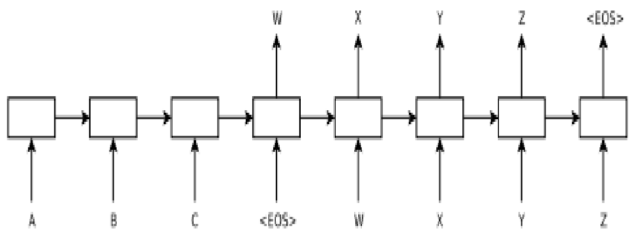

Fig. 2. Sequence to sequence model

A. Given Data

For this project we were given a file called enwik8 which contains large amount of characters. The file contained characters of different languages, different symbols and numbers.

- First step that I did is converted a file to txt format, because I was not able to open it.
- Second step, I removed all the newline characters from the file, because later on this character would cause compiler errors on the step where I need to get the indices of my unique character array.
- Third step, I am iterating through the first 100000 characters and if the character exists in the English alphabet then I am going to store this character in the clean data array. I will describe later in the Issues section why I decided to do that.
- Fourth step I am creating a set that will hold character arrays of size 40. Character array of size 40 represents the input data and the output data together. The first 30 will be considered as input data and the remaining 10 as the output data. I am doing a set, because I want to avoid repetition of the same character arrays.
- Fifth step, I am going to loop 10000 times and create strings from the clean data containing 40 characters. Those strings are going to be structured in a way where the first string will be the first 40 characters of the clean data and the next string will be the same as the first one but moved 1 character forward. For example lets assume the first string is ”abcdefg...xyz”, the second string will be ”bcdefg...xyz” + the next character in the clean data. So each consecutive string is moved by 1 character. I have a total of 9815 different strings.
- Six step, I am going to find all the unique characters that I have in my data.
- Seventh step, I am assigning index values to my unique character array. Now I can covert the characters into numbers and all these numbers will correspond to their character index.
- Eights step, I am going to convert all the strings into number arrays.

B. Data Manipulation

After I finished the conversion of character array into number array, I need to prepare the data for the model.

- First step, I am going to split my data which contains 9815 different arrays of numbers of size 40, into X and Y. As I said before the first 30 numbers will be input sequence and the rest 10 the output sequence.
- Second step, I am going to expand the dimension of X to make it 3D, fit and transform it using One Hot Encoder and make it an array called X1. One Hot Encoder is going to prepare our data by making it categorical. It is going to give us better predictions.
- Third step, I am going to expand dimensions to make X1 array 3D, then I will reshape it, then I will fit and transform my X1 array using One Hot Encoder and reshape the X1 array into its final form. It's final form has to be 3D.
- Fourth step, I am going to reshape and transform my Y array using One Hot Encoder and store the result in y1 array.
- Fifth step, I am going to split the resulted X1 and y1 arrays into X_train, X_val, X_test, y_train, y_val, y_test.

All of these steps are necessary to prepare the data. We need to make sure that our input sequence will be 3D and output sequence 2D. We need to use the One Hot Encoder to make better predictions.

V. RNN model

After I prepared the data I started working on building the RNN model. I used PyTorch to build it, because it is fast, efficient and contains necessary functions. I created an RNNmodel class that will take as input parameters input size, hidden size, output size, number of layers. Lets go over these parameters. According to [5] these parameters represent:

- Input size - the number of expected features in the input X1
- Hidden size - the number of features in the hidden state h
- Output size - the tensor shape (the output and the input has to be the same size)
- Number of layers - number of recurrent layers

Then I use the PyTorch LSTM function and store it in the lstm variable. After that I store the fully connected layer using the PyTorch Linear function in the fc variable. In order to not overfit, I am going to do the dropout(0.3).
Now, lets talk about the forward function. This is where I am doing my step forward. All what I am doing here is calling the variables that I defined earlier for the input data X.

VI. Training

After I described how I build the RNN model, lets talk about how I trained it. Before training the model I need to set the values for the epoch, learning rate, and the batch_size, decide what loss function I am going to use, decide what optimizer I am going to use and create a model. I decide to set the epoch to 100, batch_size to 64, learning rate to 0.0025, my loss function is going to be CrossEntropyLoss() and the optimizer that I used is Adam. I chose Adam because according to [6] it is the best optimizer that exists. I did change the epoch size a couple of times staring from 10 to 100, but I did not notice a lot of difference.

A. Training

- First step, I am preparing the input sequence and the output sequence for the model. I am using the torch.Tensor()
- Second step, I need to set the gradients to zero, that is why I am calling the function zero_grad()
- Third step, I am placing the input sequence into the model and getting the output
- Fourth step, I am placing the output and the target sequence into the loss function to get the loss
- Fifth step, I am going to do the back propagation on the loss
- Six step, I am updating the parameters and then storing the loss in the train_loss array

B. Validation

- First step, I am preparing the input sequence and the output sequence for the model. I am using the torch.Tensor()
- Second step, I am placing the input sequence into the model and getting the output
- Third step, I am placing the output and the target sequence into the loss function to get the loss
- Fourth step, I am appending the loss to the val_loss array
- If the validation loss is lower then the min loss then I am going to replace the min loss with the current validation loss value and save the current state as the best validation

VII. Outcome

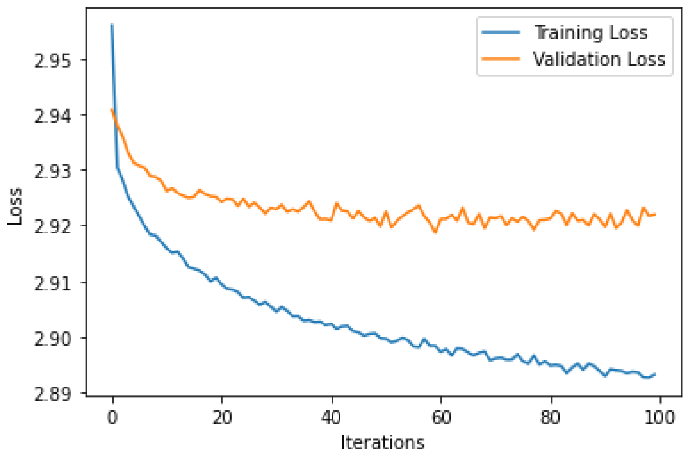

Fig. 3. Training and Validation loss over 100 iteration

After I shaped the data, made an RNN model, and trained the model I received 10.7% accuracy. Lets review what parameters I had. I had 9815 input strings and 9815 output strings. All of them are unique. I had a total of 26 unique characters, those include only English alphabet. Epoch = 100, batch size = 64, loss function is CrossEntropyLoss(), and the optimizer is Adam. Figure 3 provides a graph of my training loss and validation loss.

VIII. Issues

At first I was working with a very small data set. I had only 10 unique strings (input and output). When I build and trained my model, I received 0 accuracy. It is very poor result. I spent a lot of time trying to figure out what is the problem. Then I decided to change the amount of data to something larger. I ended up having 98750 different strings (input and output). With such a big amount of strings I had about 6000 unique characters. I ran my code again, and then I was getting 0.000000539 as my accuracy score. The model trained very slow, about 1 hour and 30 minutes, so I decided to lower the amount of strings. I chose 10000. Even when I lowered the data, I did not get a better result, so I started looking for other issues. After that I though that the problem may be because of the amount of characters that I have. I started scrolling through the data and I noticed that the majority of characters are English and 0-9 and only sometimes other characters appear. Since other characters appear rarely I decided to remove everything and keep only 0-9 and English alphabet. I received an accuracy of 9% and the graph for the validation and training loss from the Figure 4. I decided to go even further and remove all the numeric characters and keep only the English alphabet. And that is where I received the best accuracy score of 10.7%.

Another problem that I ran into is shaping of the X and Y. I spend a lot of time figuring it out, but eventually I came to a solution that I have right now.

Last value that I was changing is the learning rate. At first I had it set up as 0.01 and the best accuracy that I received was 9%, after I changed it to 0.0025, the accuracy bumped up to 10.7%.

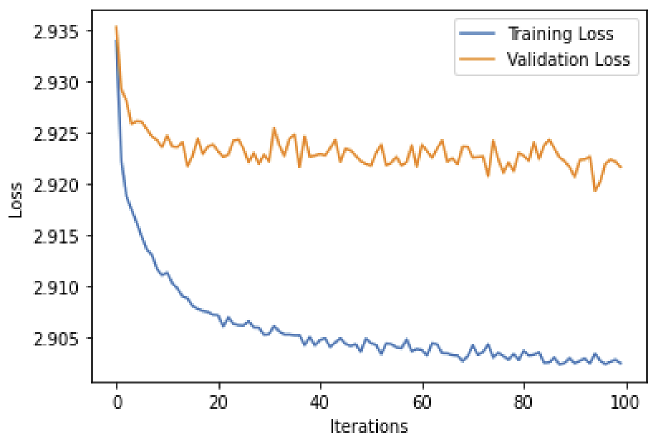

Fig. 4. Training and Validation loss over 100 iteration

References

[1] I. Sutskever, O. Vinyals, Q. V.Le, “Sequence to Sequence Learning with Neural Networks”, 2014.
[2] “Neural Network Programming - Deep Learning with PyTorch”, Available at: <https://deeplizard.com/learn/video/k6ZF1TSniYk>.
[3] N. Laskowski, “recurrent neural networks” Available at: <https://searchenterpriseai.techtarget.com/definition/recurrent-neural-networks>.
[4] “Understanding LSTM Networks”, Available at: <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>, August 27, 2015.
[5] “RNN” Availiable at: <https://pytorch.org/docs/stable/generated/torch.nn.RNN.html>.
[6] B. Basnet, “Data Science and Deep Learning”, Available at:<https://deepdatascience.wordpress.com/2016/11/18/which-lstm-optimizer-to-use/>, November 18, 2016.