Thank you Antong for the introduction, and thank everyone for joining this presentation. I am Juan and I am very interested in this Image Analytics position at Merck. Antong told me to choose a topic that relates to image processing, so I chose this natural language project since both apply neural networks. I hope to show you what I have done in this project, and would love to hear from you if there is any suggestion on how to improve the model. I will talk for about 25 minutes, and leave the last 5 minutes for questions.

Let’s get started.

**1:** the project is on sentiment analysis using Deep neural network and long short term memory, an recurrent neural network.

**2:** content. First I will give a short overview of NLP, and walk through the modeling process. Then I will touch upon transfer learning and word embedding and explain a little bit of LSTM. After that, I will compare the results from 2 models, DNN and RNN. I will wrap up the presentation with conclusion and future work.

**3:** Language is getting a lot of importance due to the recent development of conversational agents such as chatbots and voice recognition devices. It has other use cases, mainly focuses on two tasks understand and generate the language. Examples include information retrieval such as google search, text categorization such as spam filter and sentiment analysis, Conversational agent such as chatbot or amazon alexa. it involves 3 areas: Computer Science, Linguistics, and Artificial Intelligence. Language problem is not an easy task mainly because of two reasons. Sparsity. There is big vocabulary and it usually results in a large sparse matrix. And ambiguity. On the right, I have included a few data samples from our dataset. Each consists of a few words, and corresponds to one of the 5 sentiments, love, sports, sadness, happiness, and food. So we need to build a model to identify the sentiment of a given sentence.

**4**: the modeling process is quite similar to image processing, here 3 steps in preprocessing whereas in image it’s convolution and pooling. Tokenization, we split the sentence by space and punctuations, and take out each word, then covert to lower cases, stem it by removing the affix, and lemmatize to the base form, such as driving to drive, drove to drive. Once we have the words in base from, we need to vectorize them into features so the models can recognize them. It mainly has 2 ways, encoding and embedding. Encoding is quite straightforward, it basically converts each word into a feature, and the values of the matrix are just the frequency of that word in the sample. It usually results in a large sparse matrix due to the large vocabulary. Embedding involves a concept called transfer learning. The goal of embedding is still to create a feature matrix that the model can recognize. Here I have a slide on transfer learning and word embedding.

**Go to slide 5**: There are a lot of public models trained on much larger dataset and we can incorporate that into our models as the first few layers, and we only need to train the last few layers. It is quite useful if our dataset is small. For NLP, there is a trained model called global vector, it is trained on Wikipedia articles. It represents word in a continuous vector space where semantically similar words are mapped to nearby points. Also the vector difference captures the meanings between word pairs. The vector difference between man and woman is similar to that of king and queen, which is quite amazing. The vector I used includes 400,000 words and each is in a 50-dimension space. Here I listed two vectors from the Global Vector.

**Go back to slide 4**: once we have the feature matrix, we can train the data using various models such as Naïve Bayes, logistic regression, neural networks. For Evaluation we can use accuracy, F score, TPR, FPR, ROC, AUC and so on.

**6:** Now I will give a brief description of RNN and LSTM before going to our model fitting. RNN is different from DNN and CNN in that it takes into account the sequence of input. Here is a standard RNN unit. Input is the current word, and the hidden state from the previous step which contains the information from previous step. After a single tanh layer, it returns a modified hidden state and the output. This is one unit. By looping though the input, it is able to remember the past data in the hidden state. Therefore, it can be used in many applications where the sequence of input is important, such as language translation, voice detection and so on. But there are mainly two issues with basic RNN unit. Gradient vanishes when doing network training. During the back-propagation of the error rate, when the gradient is very small, it never reaches the initial layers, and the weights cannot be updated. Also RNN is not very good at capturing long term dependency, meaning the memory cannot remember long sequence. After many RNN steps, the memory about the initial word is overwritten by the later words, so the information cannot be carried on. So these are the two major issues with RNN.

**7:** Fortunately, there is a an update version. LSTM is a new type of RNN architecture and used to learn long-term dependency. The main difference from basic RNN unit is more complex node design. It adds 3 gates to control how much information gets through. They are forget gate, input gate, and output gate, all of which are composed out of a sigmoid neural net layer. Forget gate is applied on the previous cell state to control how much of C t-1 enter the current cell state, input gate controls how much current input x enter the current cell state, and output controls how much to output. Using these 3 gates, the flow of information is controlled. A value of zero from the sigmoid means “let nothing through,” while a value of one means “let everything through!”. These 3 gates allow the network to prioritize between what is ‘important’ and what is ‘not-important’, so important information at the beginning can be carried on to the point where it is needed. It is well-suited to process sequence input where the relevant information has gaps.

**8:** So now I have introduced the relevant background, I want to show the model results. I used two models, DNN and LSTM. I first tried a 2-layer DNN with 50 nodes. For each sample, each word is projected into a 50-dimension vector using the word embedding, and take average, resulting in a single 50-dimension vector for each sample. so the input dimension is 50, then 50 nodes in the hidden layer, and output 5 categories. Categorical cross-entropy is used for loss function, adam as optimizer, and accuracy as evaluation metrics. The first plot is loss vs epoch, both the training and validation loss decrease nicely and began to flat out at around 300 epochs. The below is accuracy plot. The validation accuracy stops at around 82%. On the right I listed a few new predictions using trained DNN model, and it didn’t predict the last sample correctly, ‘I do not like it’ was categorized as a happy face whereas it should be a sad face. It doesn’t find the effect of not in the sentence. The accuracy is not bad at 82%, but still there is room for improvement.

**9:** so I tried the RNN model, a 2 layer LSTM where each word is an input in the model. Both layers have 128 nodes, and I used dropout to reduce overfitting. The convergence is bumpier than the DNN model, as there are more parameters to train. It tries to find the directions, but eventually it seems to converge. Compare the performance with DNN, there are not much difference, only a small increase in accuracy, but more importantly, it was able to predict the last sample, meaning the sequence of the input is captured by the model to make the right prediction.

**10:** Ok, this is how I conducted a sentiment analysis using DNN and RNN models, and compared their performance. We can see that word embedding is a powerful tool for NLP to capture the semantic meaning in words. RNN model is quite versatile for sequence input, and the advance version LSTM is good at modeling the long term dependency. Another alternative to LSTM is the gated recurrent unit, wither fewer gates, so we can try it and see if it could reduce the overfitting and the training time. RNN models can easily be used for image processing, combined with CNN, the combo has great potential to improve the accuracy of the classification problems.

Ok, this concludes my presentation, now let me know if there is any question from the audience.

*Then you loop through your inputs, pass the word and hidden state into the RNN. The RNN returns the output and a modified hidden state.*

*All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.*

*It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large.*

*Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.*

*The gradient is the value used to adjust the networks internal weights, allowing the network to learn. The bigger the gradient, the bigger the adjustments and vice versa. Here is where the problem lies. When doing back propagation, each node in a layer calculates it’s gradient with respect to the effects of the gradients, in the layer before it. So if the adjustments to the layers before it is small, then adjustments to the current layer will be even smaller.*

*That causes gradients to exponentially shrink as it back propagates down. The earlier layers fail to do any learning as the internal weights are barely being adjusted due to extremely small gradients. And that’s the vanishing gradient problem.*

**Input gate** — discover which value from input should be used to modify the memory. **Sigmoid** function decides which values to let through **0,1.**and **tanh**function gives weightage to the values which are passed deciding their level of importance ranging from**-1** to **1.**

**2. Forget gate**— discover what details to be discarded from the block. It is decided by the **sigmoid function.**it looks at the previous state(**ht-1**) and the content input(**Xt**) and outputs a number between **0(***omit this*)and **1(***keep this***)**for each number in the cell state **Ct−1**.

**3. Output gate** — the input and the memory of the block is used to decide the output. **Sigmoid** function decides which values to let through **0,1.**and **tanh**function gives weightage to the values which are passed deciding their level of importance ranging from**-1** to **1**and multiplied with output of **Sigmoid.**

*The key to LSTMs is the cell state, the horizontal line running through the top of the diagram.*

*The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.*

*Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation.*

*he sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through. A value of zero means “let nothing through,” while a value of one means “let everything through!”*

*An LSTM has three of these gates, to protect and control the cell state.*

*Tanh is used since its output can be both positive and negative hence can be used for both scaling up and down. The output from this unit is then combined with the activation input to update the value of the memory cell.*

The model updates with every iteration, adjusting weights and biases in order to minimize the loss function and improve the accuracy percentage using gradient descent and the backpropagation algorithm.

It’s important to note that every time you train and test the data, you’re going to obtain different loss function costs and train accuracy results because the algorithm chooses different weights and biases per iteration.

(Adam is an adaptive learning rate method, which means, it computes individual learning rates for different parameters. It uses the squared gradients to scale the learning rate like RMSprop and it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum.)

Short Biography:

Juan Chen recently took a break from work after 3 years at American Express, where she worked on regulatory modeling to estimate the credit loss, and also on marketing analytics for customer acquisition. Before that, she was a statistician at Johnson and Johnson for 3 years. Now she is looking to go back to the workplace.