INFO7374 Team 4 Final Report

Summary	In this codelab, you'll learn the knowledge about Natural Language Inference
URL	pwapp-chatbot
Category	Web
Environment	web, kiosk, io2016, pwa-dev-summit, pwa-roadshow, chrome-dev-summit-2016, io2017, typtwd17, gdd17, cds17, io2018, tag-web, jsconfeu, devfest18
Status	Formal
Feedback Link	
Author	Wenjun song Haiming zhang Yanjun lyu

Problem

As a core part of the 'Chatbot', Natural language inference is the task of determining whether a "hypothesis" is true (entailment), false (contradiction), or undetermined (neutral) given a "premise". The aim is to determine whether a premise sentence is entailed, neutral, or contradicts a hypothesis sentence - i.e. "A soccer game with multiple males playing" entails "Some men are playing a sport" while "A black race car starts up in front of a crowd of people" contradicts "A man is driving down a lonely road". Modeling natural language inference is a very challenging task. With the availability of large annotated data, it has recently become feasible to train complex models such as neural-network-based inference models, which have shown to achieve the state-of-the-art performance.

Reference paper

2.1.Learning Natural Language Inference using Bidirectional LSTM model and Inner-Attention, written by Yang Liu, Chengjie Sun, Lei Lin and Xiaolong Wang.
2.2. Learning Natural Language Inference with LSTM, written by Shuohang Wang, Jing Jiang.
2.3. An overview of Natural Language Inference Data Collection, written by Stergios Chatzikyriakidis, Robin Cooper, Simon Dobnik, Staffan Larsson.
2.4. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data, witten by Alexis Conneau, Douwe Kiela, Holger Schwenk, Loic Barrault, Antoine Bordes.

Data description

This repository contains a simple Keras baseline to train a variety of neural networks to tackle. The data is Stanford Natural Language Inference (SNLI) corpus. The SNLI corpus (version 1.0) is a collection of 570k human-written English sentence pairs manually labeled for balanced classification with the labels entailment, contradiction, and neutral, supporting the task of natural language inference (NLI), also known as recognizing textual entailment (RTE).
The corpus is designed aiming at serving both as a benchmark for evaluating representational systems for text, especially including those induced by representation learning methods, as well as a resource for developing NLP models of any kind. The corpus is distributed in JSON lines and will be load into our python application in JSON format.
The example test is as follow: containing ‘text’, corresponding ‘hypothesis’, and 5 ‘judgments’ for each ‘T-H’ pair.

Text	Judgments	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction C C C C C	The man is sleeping
An older and younger man smiling.	neutral N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A black race car starts up in front of a crowd of people.	contradiction C C C C C	A man is driving down a lonely road.
A soccer game with multiple males playing.	entailment E E E E E	Some men are playing a sport.
A smiling costumed woman is holding an umbrella.	neutral N N E C N	A happy woman in a fairy costume holds an umbrella.

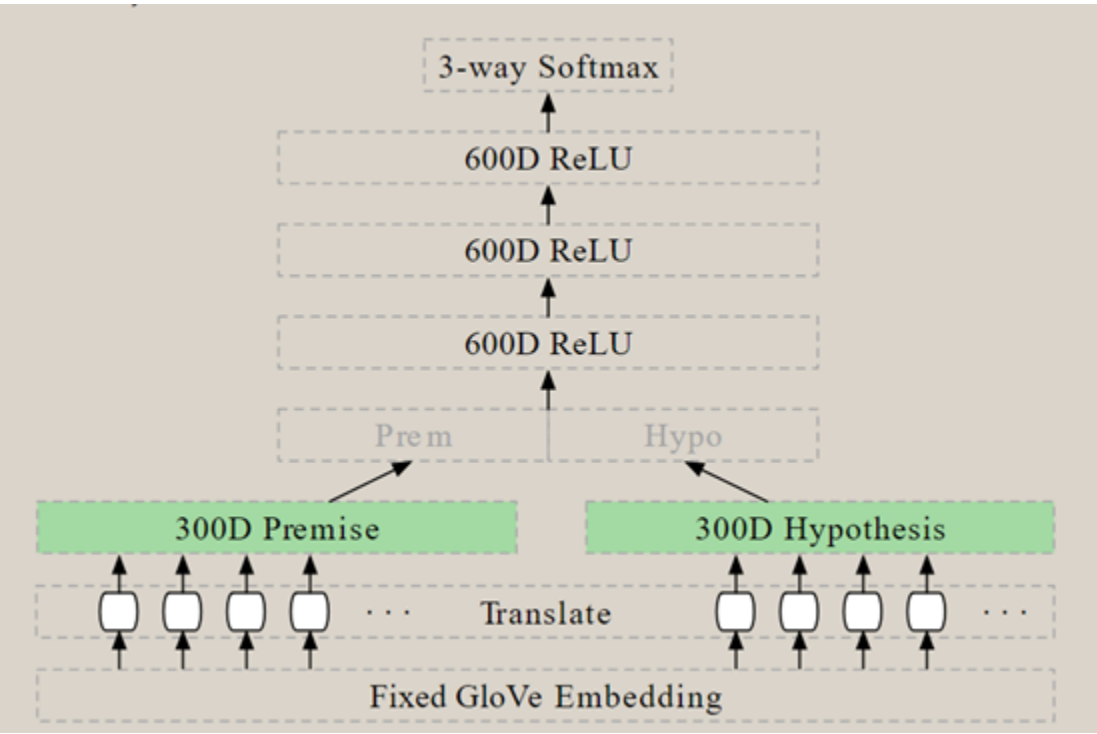
Model description

Basically, we borrowed the model from https://github.com/Smerity/keras_snli.

The model architecture is:

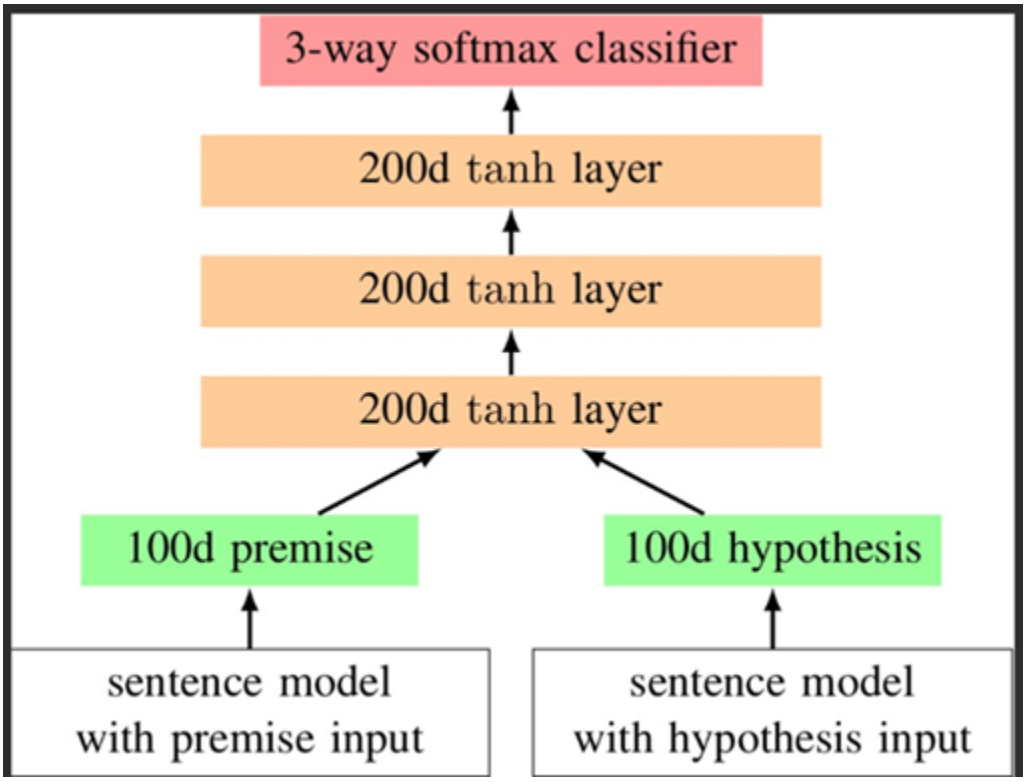
- Extract a 300D word vector from the fixed GloVe vocabulary
- Pass the 300D word vector through a ReLU "translation" layer
- Encode the premise and hypothesis sentences using the same encoder (summation, GRU, LSTM, ...)
- Concatenate the two 300D resulting sentence embeddings
- 3 layers of 600D ReLU layers
- 3 way softmax

Training uses RMSProp and stops after N epochs have passed with no improvement to the validation loss. Notation: following Liu et al. 2016, the GloVe embeddings are not updated during training. Following Munkhdalai & Yu 2016, the out of vocabulary embeddings remain zeroed out.



Model Summary:
Model, 300D sum(word vectors) + 3 x 600D ReLU
Parameters, 1.2m

Our research targets mainly focus on Sentence encoding-based models which is the main domain of NLI.
LSTM encoders:
We can use h and p to encoder the input data, and then put the outcome to three fully connections layers with activation function, next we use a softmax function to output the final. This model is simple for NLI, and there are many papers were based on this model, such as Tree-based CNN encoders, and InferSent, Structured Self Attention, mLSTM & word-by-word attention, Decomposable Attention, enhanced sequential infer-ence model (ESIM), and KIM & DMAN.



We use this model to train the SNLI dataset to analyze the relationship between premise and hypothesis.
Firstly, we should split the dataset into three parts: premise and hypothesis and the labels.

```
def get_data(fn, limit=None):
    raw_data = list(yield_examples(fn=fn, limit=limit))
    left = [s1 for _, s1, s2 in raw_data]
    right = [s2 for _, s1, s2 in raw_data]
    print(max(len(x.split()) for x in left))
    print(max(len(x.split()) for x in right))

    LABELS = {'contradiction': 0, 'neutral': 1, 'entailment': 2}
    Y = np.array([LABELS[l] for l, s1, s2 in raw_data])
    Y = np_utils.to_categorical(Y, len(LABELS))

    return left, right, Y
```

Then we tokenizing the data, and set the paremeters of the encoding model:

```
VOCAB = len(tokenizer.word_counts) + 1
LABELS = {'contradiction': 0, 'neutral': 1, 'entailment': 2}
LABELS_INV = dict((v,k) for k,v in LABELS.items())
RNN = None
LAYERS = 1
USE_GLOVE = True
TRAIN_EMBED = False
EMBED_HIDDEN_SIZE = 300
SENT_HIDDEN_SIZE = 300
BATCH_SIZE = 256
PATIENCE = 4 # 8
MAX_EPOCHS = 10
MAX_LEN = 42
DP = 0.2
L2 = 4e-6
ACTIVATION = 'relu'
OPTIMIZER = 'rmsprop'
print('RNN / Embed / Sent = {}, {}, {}'.format(RNN, EMBED_HIDDEN_SIZE, SENT_HIDDEN_SIZE))
print('GloVe / Trainable Word Embeddings = {}, {}'.format(USE_GLOVE, TRAIN_EMBED))
```

Then we download the pre-trained model-- glove.840B.300d.txt from GLOVE model website to train SNLI dataset like this:

```
GLOVE_STORE = 'precomputed_glove.weights'
if USE_GLOVE:
    if not os.path.exists(GLOVE_STORE + '.npy'):
        print('Computing GloVe')
        embeddings_index = {}
        f = open('glove.840B.300d.txt')
        for line in f:
            values = line.split(' ')
            word = values[0]
            coefs = np.asarray(values[1:], dtype='float32')
            embeddings_index[word] = coefs
        f.close()

        embedding_matrix = np.zeros((VOCAB, EMBED_HIDDEN_SIZE))
        for word, i in tokenizer.word_index.items():
            embedding_vector = embeddings_index.get(word)
            if embedding_vector is not None:
                embedding_matrix[i] = embedding_vector
            else:
                print('Missing from GloVe: {}'.format(word))

        np.save(GLOVE_STORE, embedding_matrix)
```

Total params: 13,900,803
Trainable params: 1,178,703
Non-trainable params: 12,722,100

Model Evaluation

The First limitation of the LSTM model was still using a single vector representation of the premise, namely h a N , to match the entire hypothesis. The second limitation is that the model does not explicitly allow us to place more emphasis on the more important matching results between the premise and the hypothesis and down-weight the less

critical ones. In order to solve two limitations above, the author proposed to use an LSTM to sequentially match the two sentences.
First, the author process the premise and the hypothesis using two LSTMs, but the author did not feed the last cell state of the premise to the LSTM of the hypothesis. Next, we generate the attention vectors ak similarly to Eqn (2). Our h is the hidden state at position k generated from our mLSTM. This LSTM models the matching between the premise and the hypothesis.

Basically this is a good model for SNLI dataset, the test accuracy is 0.81 which is enough for inferencing sentences for the realistic problems. However, it is certainly not the best model, we should make more research to develop and evolve the model to get a better output.
Then we put 11 pairs of sentences to do the “unit test and show the results” like this:

```
i=1
while i<=11:
    print("      ")
    print("Round "+str(i)+" : please input")
    text = input("Text: ")
    hypothese=input("Hypothesis: ")
    inferenc=predict_inference(model,text,hypothese)
    i+=1
```

```
Round 1 : please input
Text: A woman with a green headscarf , blue shirt and a very big grin .
Hypothesis: The woman has been shot .
Precicted Judgments:  entailment

Round 2 : please input
Text: An old man with a package poses in front of an advertisement .
Hypothesis: A man poses in front of an ad .
Precicted Judgments:  entailment

Round 3 : please input
Text: An old man with a package poses in front of an advertisement .
Hypothesis: A man poses in front of an ad for beer .
Precicted Judgments:  neutral

Round 4 : please input
Text: An old man with a package poses in front of an advertisement .
Hypothesis: A man walks by an ad .
Precicted Judgments:  contradiction

Round 5 : please input
Text: A statue at a museum that no seems to be looking at .
Hypothesis: The statue is offensive and people are mad that it is on display .
Precicted Judgments:  neutral

Round 6 : please input
Text: A statue at a museum that no seems to be looking at .
Hypothesis: There is a statue that not many people seem to be interested in .
Precicted Judgments:  entailment
```

Then we made three confusion matrixes to help us to observe this model for the training data, validation data and test data.

	pred_contradiction	pred_neutral	pred_entailment
true_contradiction	147628	20826	14733
true_neutral	9921	141736	31107
true_entailment	3161	12554	167701

The accuracy of training data is 0.83

	pred_contradiction	pred_neutral	pred_entailment
true_contradiction	2582	387	309
true_neutral	235	2495	505
true_entailment	81	224	3024

The validation accuracy is 0.82

	pred_contradiction	pred_neutral	pred_entailment
true_contradiction	2524	404	309
true_neutral	231	2414	574
true_entailment	89	257	3022

The test accuracy is 0.81
From the outcome above, we can see the SNLI dataset is a mature one to train a NLI model, and the model we used performanced well for this data. The accuracy for training, validation, and test is around 0.81-0.83 which can inference a relative better output for the real problems. Maybe we can find a better model to have a better output if we make a deeper researchwith more data or with other datasets.

Model distribution

After trained, our task has become how to warp the model into an executable application. in order to accept and predict the income data, we separate the task into two domain:
The development of 1, standards of input data and 2, model warping.

For 1, standards of input data:

- **Data requirements** - the data are required to be loaded by JSON files, the standard format should be like:

- **Standard format** – the format should be fixed just like it in the SNLI corpus:

Colums: gold_label sentence1_binary_parse sentence2_binary_parse sentence1_parse sentence2_parse sentence1 sentence2 captionID
However, the value of Col_‘gold_label’ is set to None since the prediction have not yet begain.

- Return – The returned format is much the same as input, with the Col_‘gold_label’ be the predicted results. The output file will be JSON.

For 2, model warping:

- **Model files** – the model will be keep under a folder, which consist of a model.py file witch is the main model and .h5 files witch is the weights value of the model. The model.py can be running in the python3 Env.

- **Utilization** – The data ‘input.json’ should first be put into the folder. And then run command ‘python3 model.py input.json’. Waiting the application to finish and give the ‘output.json’
