Report: Assignment 4
COMP 8740: Neural Networks
ID: U00744746

Hosneara Ahmed

October 26, 2021

1 Introduction

Traditional feed forward deep learning techniques such as Multi-layer perceptron, CNN, etc. these only consider the current feature representation at a time, cannot look back into the historical context. However, RNN has come to get rid of this problem. Simple RNN cannot look back much but later on due to its 'gradient vanishing' or 'gradient exploding' problem. To get rid of it, variations of RNN such as Long Short Term Memory (LSTM) network, Gated Recurrent Unit (GRU) have been developed. In this experiment, we will compare the performance of these three networks on machine translation problem, e.g. English to French translation.

2 Methodology

Problem formulation

Two datasets are given, one contains rows of English sentences. Another contains corresponding translation in French. Each row contains single sentence. So the context that the trained system has to remember to translate does not go beyond a sentence unlike a paragraph. So we have to form a function that will convert an English sentence to French sentence. Suppose X_ith is ith English sentence and Y_ith is corresponding French sentence.

$$X_i = f(X_i) = Y_i$$

2.1 Data Pre-processing

- **Row Data Reading:** Two data files for English and French correspondingly are read as two files and split by new lines.
- **Encoding and Tokenization:** Machine cannot recognize any text, rather it has to be any numeric value. So, each unique English and French word has to be encoded to numeric integer valued tokens.
- **Padding:** All sentences are not of equal length in neither English nor French. But while feeding these sentences to RNN, we have to define the input size which mandates the length or input size equal. For that, we have padded *zero* to the end of sentences which are of small length than the max long sentence in that language.

2.2 Translation Model

After all the data preprocessing steps, we have run three individual models - vanilla RNN, LSTM, and GRU on the English sentences for translation to French and compared the performances. Comparison results are discussed in details in evaluation section.

3 Deep Learning Architecture

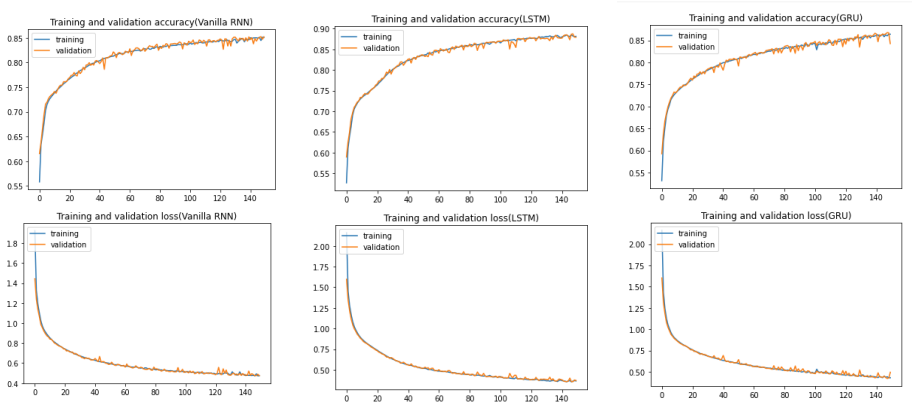
Vanilla RNN, LSTM, and GRU - all three has a common property, that is they can remember some context from the past which will be required to learn better in the next layers. In other way, they decide which information from the past outputs has to remember, which has to forget, and which has to pass to next.

- Vanilla RNN: In vanilla RNN or simple RNN, there are three gates - input gate, forget gate, and output gate. At time step t, the hidden state depends on previous output y_{t-1} at time $t-1$ and current input x_t. As only context it gets, comes from no other source than the previous output, the gradient vanishes or explodes quickly.
- LSTM: Apart from three gates from simple RNN, LSTM has a memory cell, which remembers the context from the previous cell. So the context comes from both the memory cell as well as previous cell. So the gradient does not vanishes unlike simple RNN.
- GRU: GRU has only two gates unlike LSTM, that are - input gate and update gate. It makes the GRU simpler than LSTM apparently without degrading the performance.

4 Experiment and Results

4.1 Experimental Setup

For each of vanilla RNN, LStM, and GRU model, the learning rate is 1e-3. two layers each with 64 neurons are stacked together (e.g. for LSTM, two LSTM layers, for vanilla RNN, two vanilla RNN layers, etc.). Lastly, a TimeDistributed dense layer is added with Softmax activation as this is multi-class classification problem. This TimeDistributed layer adds dense layer to each of the RNN units in each time step. Batch size for training is 128 and models are run 150 epochs with 20%validation split. So, for each training, 80% data are used for training and rest 20% are used for validation.



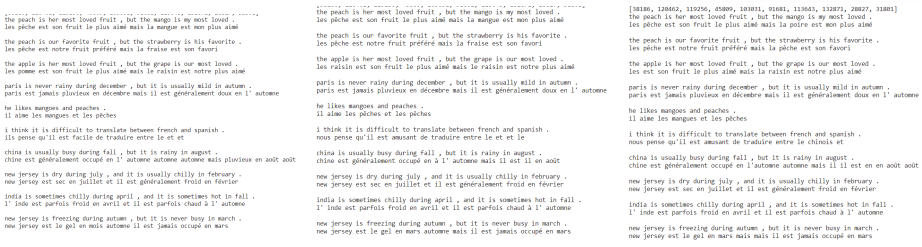
(a) Vanilla RNN (b) LSTM (c) GRU

Figure 1: Comparison of training and validation loss and accuracy in English to French translation

4.2 Discussion and Comparison

From figure 1 we can see, the loss and accuracy curve for both training and validation set are decreasing and increasing gradually which indicates the model is learning well. However, if we look closely, the accuracy in simple RNN and GRU is about 85% while in LSTM it is about 90%. So LSTM and GRU perform better than RNN. IN the same way, if we look at the loss curve, simple RNN has about 50% loss. On the other hand, in LSTM and GRU loss is about 45% which indicates in terms of loss both LSTM and and GRU outperforms simple RNN. But with respect to both loss and accuracy, LSTM clearly outperforms both simple RNN and GRU.

4.3 Results



(a) Vanilla RNN (b) LSTM (c) GRU

Figure 2: English to French translation

In figure 2, we have randomly selected 10 English sentences and predicted with simple RNN, LSTM, and GRU model. The same random sentences are for each model.

5 Conclusion

In the given dataset, the sentences are not much long (max English sentence length is 15, max French sentence length is 21). So, apparently simple RNN, LSTM, GRU - all should perform similar. However, from the evaluation we have observed that overall, LSTM is outperforming simple RNN even in this dataset with small context. So, for long historical context, LSTM should perform better.

6 Appendix

```
def tokenize(x):
    x_tk = Tokenizer(char_level = False)
    x_tk.fit_on_texts(x)
    return x_tk.texts_to_sequences(x), x_tk

def pad(x, length=None):
    if length is None:
        length = max([len(sentence) for sentence in x])
    return pad_sequences(x, maxlen = length, padding = 'post')

def preprocess(x, y):
    preprocess_x, x_tk = tokenize(x)
    preprocess_y, y_tk = tokenize(y)

    preprocess_x = pad(preprocess_x)
    preprocess_y = pad(preprocess_y)

    preprocess_y = preprocess_y.reshape(*preprocess_y.shape, 1)
    return preprocess_x, preprocess_y, x_tk, y_tk

preproc_english_sentences, preproc_french_sentences,
↪  english_tokenizer, french_tokenizer =
↪  preprocess(english_sentences, french_sentences)

def logits_to_text(logits, tokenizer):
    index_to_words = {id: word for word, id in
    ↪  tokenizer.word_index.items()}
    index_to_words[0] = '<PAD>'

    return ' '.join([index_to_words[prediction] for prediction in
    ↪  np.argmax(logits, 1) if prediction != 0])
```

```
#Model build
def stacked_RNN_model(input_shape, output_sequence_length,
↪  english_vocab_size, french_vocab_size):

    learning_rate = 1e-3
    input_seq = Input(input_shape[1:])
    rnn1 = SimpleRNN(64, return_sequences = True)(input_seq)
    rnn2 = SimpleRNN(64, return_sequences=True)(rnn1)
    logits = TimeDistributed(Dense(french_vocab_size+1))(rnn2)
    model = Model(input_seq, Activation('softmax')(logits))
    model.compile(loss = sparse_categorical_crossentropy,
                  optimizer = Adam(learning_rate),
                  metrics = ['accuracy'])
    return model

def stacked_LSTM_model(input_shape, output_sequence_length,
↪  english_vocab_size, french_vocab_size):
    learning_rate = 1e-3
    input_seq = Input(input_shape[1:])
    rnn1 = LSTM(64, return_sequences = True)(input_seq)
    rnn2 = LSTM(64, return_sequences=True)(rnn1)
    logits = TimeDistributed(Dense(french_vocab_size+1))(rnn2)
    model = Model(input_seq, Activation('softmax')(logits))
    model.compile(loss = sparse_categorical_crossentropy,
                  optimizer = Adam(learning_rate),
                  metrics = ['accuracy'])
    return model

def stacked_GRU_model(input_shape, output_sequence_length,
↪  english_vocab_size, french_vocab_size):
    learning_rate = 1e-3
    input_seq = Input(input_shape[1:])
    rnn1 = GRU(64, return_sequences = True)(input_seq)
    rnn2 = GRU(64, return_sequences=True)(rnn1)
    logits = TimeDistributed(Dense(french_vocab_size+1))(rnn2)
    model = Model(input_seq, Activation('softmax')(logits))
    model.compile(loss = sparse_categorical_crossentropy,
                  optimizer = Adam(learning_rate),
                  metrics = ['accuracy'])

    return model

tmp_x = pad(preproc_english_sentences,
↪  max_french_sequence_length)
```

```
tmp_x = tmp_x.reshape((-1, preproc_french_sentences.shape[-2],
↪  1))

# Train RNN
rnn_model = stacked_RNN_model(tmp_x.shape,max_french_se
quence_length,english_vocab_size,french_vocab_size)

rnn_history = rnn_model.fit(tmp_x, preproc_french_sentences,
↪  batch_size=128, epochs=150, validation_split=0.2)

# Train LSTM
lstm_model = stacked_LSTM_model(tmp_x.shape,max_french_s
equence_length,english_vocab_size,french_vocab_size)

lstm_history = lstm_model.fit(tmp_x,
preproc_french_sentences, batch_size=128, epochs=150,
↪  validation_split=0.2)

# Train the GRU
gru_model = stacked_GRU_model(tmp_x.shape,max_french_seq
uence_length,english_vocab_size,french_vocab_size)

gru_history = gru_model.fit(tmp_x,
preproc_french_sentences, batch_size=128, epochs=150,
validation_split=0.2)

# predict 10 random English sentences

model_loaded = tf.keras.models.load_model(base_dir+model)
random.seed(100)
random_sample = random.sample(range(tmp_x.shape[0]), 10)

for i in random_sample:
  prediction = model_loaded.predict(tmp_x[i:i+1])
  print(english_sentences[i])
  print(logits_to_text(prediction[0], french_tokenizer))
  print()
```