RNN/CNN-based Natural Language Inference

Lu Yin, Github Link: https://github.com/ly1123/DS-1011-NLP

October 2018

1 Acknowledge

Part of the codes in this project are selected from the course <Natural Language Processing with Representation Learning> DS-GA 1011 AU2018 by Professor Cho from NYU

The sort method in the RNN model for this project is inspired by the code provided by Facebook Research: https://github.com/facebookresearch/DrQA/blob/master/drqa/reader/layers.pyL103-L165

2 Introduction

The SNLI task poses the following problem: Given two pieces of text, a premise and a hypothesis, you (or a model) have to choose one of the following: 1. The premise entails the hypothesis. 2. The premise contradicts the hypothesis. 3. The premise neither entails nor contradicts the hypothesis, and is thus neutral to it. For example:

Premise: A man inspects the uniform of a figure in some East Asian country. Hypothesis: The man is sleeping. This is a contradiction, since the man cannot both be sleeping and inspecting the uniform.

Premise:A soccer game with multiple males playing. Hypothesis: Some men are playing a sport. This is an entailment, because if the former is true, the latter must be true.

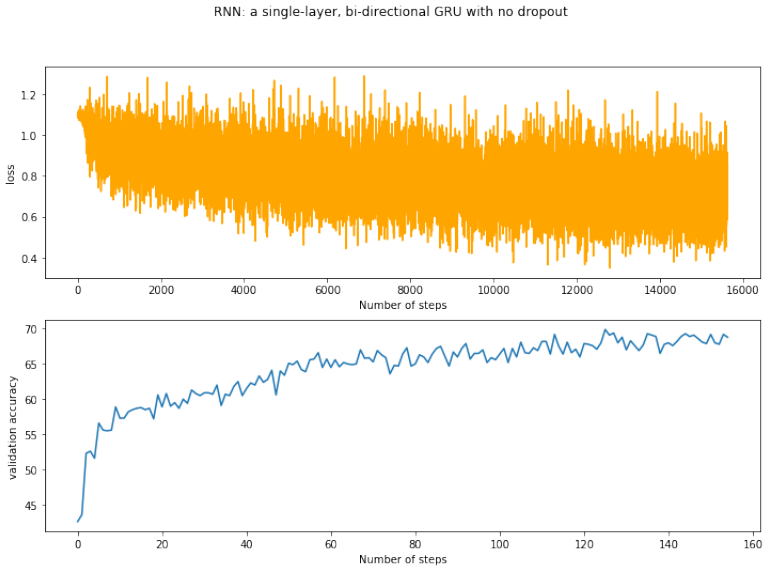
Premise: An older and younger man smiling. Hypothesis: Two men are smiling and laughing at the cats playing on the floor. This is neutral, because the premise does not contain any information about cats that the hypothesis posits.

3 Model

The two models that are used in this report are basic RNN and CNN models.

1. Base RNN model: single-layer, bi-directional GRU with no dropouts

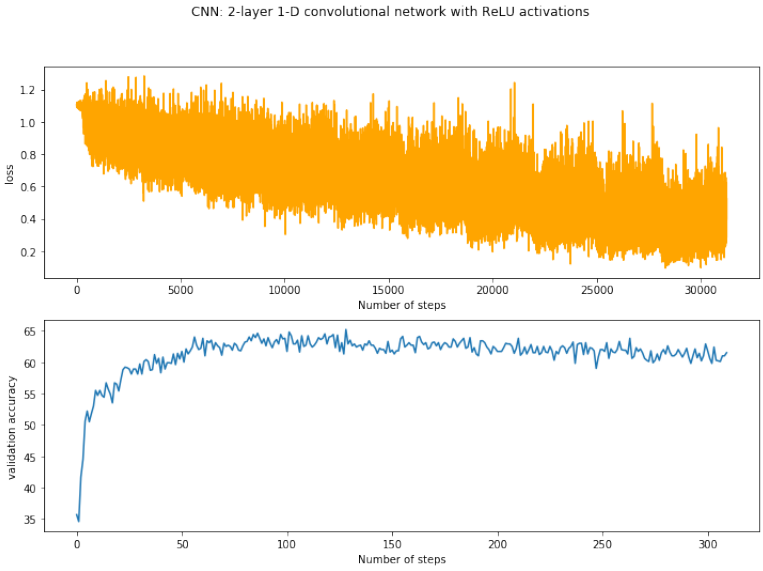
```
RNN(nn.Module):
f __init__(self, emb_size, hidden_size, num_layers, linear_size, dropout_p, num_classes):

  super(RNN, self).__init__()

  self.num_layers, self.hidden_size = num_layers, hidden_size
  self.embedding = nn.Embedding.from_pretrained(torch.from_numpy(pretrained_matrix).float(), freeze=True)
  self.bi_gru = nn.GRU(emb_size, hidden_size, num_layers, batch_first = False, dropout = dropout_p, bidirectional=True)
  self.linear1 = nn.Linear(hidden_size*2, linear_size)
  self.linear2 = nn.Linear(linear_size, num_classes)
```



2. Base CNN model: 2-layer 1-D convolutional network with ReLU activations

```
class CNN(nn.Module):

    def __init__(self, emb_size, hidden_size, linear_size, kernel_size, num_classes):

        super(CNN, self).__init__()

        self.hidden_size = hidden_size
        self.embedding = nn.Embedding.from_pretrained(torch.from_numpy(pretrained_matrix).float(), freeze=True)

        self.conv1 = nn.Conv1d(emb_size, hidden_size, kernel_size, padding=0)
        self.conv2 = nn.Conv1d(hidden_size, hidden_size, kernel_size, padding=0)
        self.linear1 = nn.Linear(hidden_size, linear_size)
        self.linear2 = nn.Linear(hidden_size, num_classes)
```

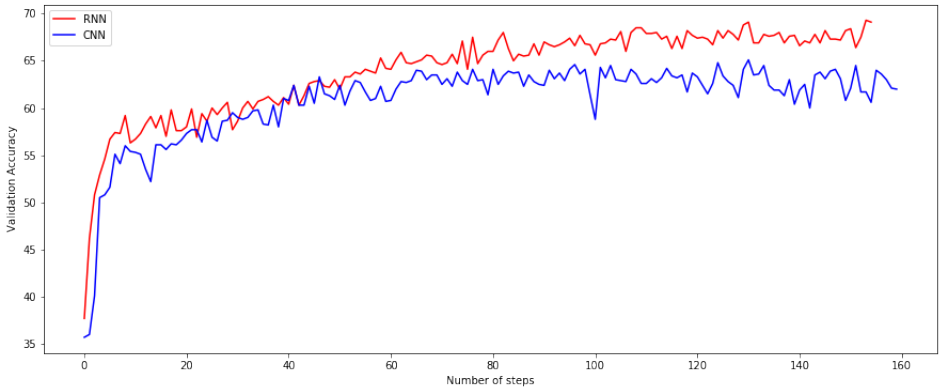
```
def forward(self, x1, x2, length1, length2):

    batch_size, seq_len1 = x1.size()
    _, seq_len2 = x2.size()

    embed_s1 = self.embedding(x1)
    hidden1 = self.conv1(embed_s1.transpose(1,2)).transpose(1, 2)
    hidden1 = F.relu(hidden1)
    hidden1 = self.conv2(hidden1.transpose(1,2)).transpose(1,2)
    hidden1 = F.relu(hidden1)

    embed_s2 = self.embedding(x2)
    hidden2 = self.conv1(embed_s2.transpose(1,2)).transpose(1,2)
    hidden2 = F.relu(hidden2)
    hidden2 = self.conv2(hidden2.transpose(1,2)).transpose(1,2)
    hidden2 = F.relu(hidden2)

    hidden3 = torch.cat((hidden1, hidden2), 1) #dim = 1
    hidden3 = torch.max(hidden3, 1)[0] #dim = 1

    hidden3 = self.linear1(hidden3)
    hidden3 = F.relu(hidden3)
    logits = self.linear2(hidden3)

    return logits
```



As it shows, the number of epochs also influence the modeling result. The validation accuracy slightly goes down from 65 to 60 with more iterations even though the loss is continuously dropped. To prevent this overfitting I choose number of epochs = 4 instead of 10 for this CNN model.

3. Compare base CNN and RNN

   The following graph compares current base RNN and CNN model. RNN perform slightly better than CNN in terms of validation accuracy with the first 160 steps.



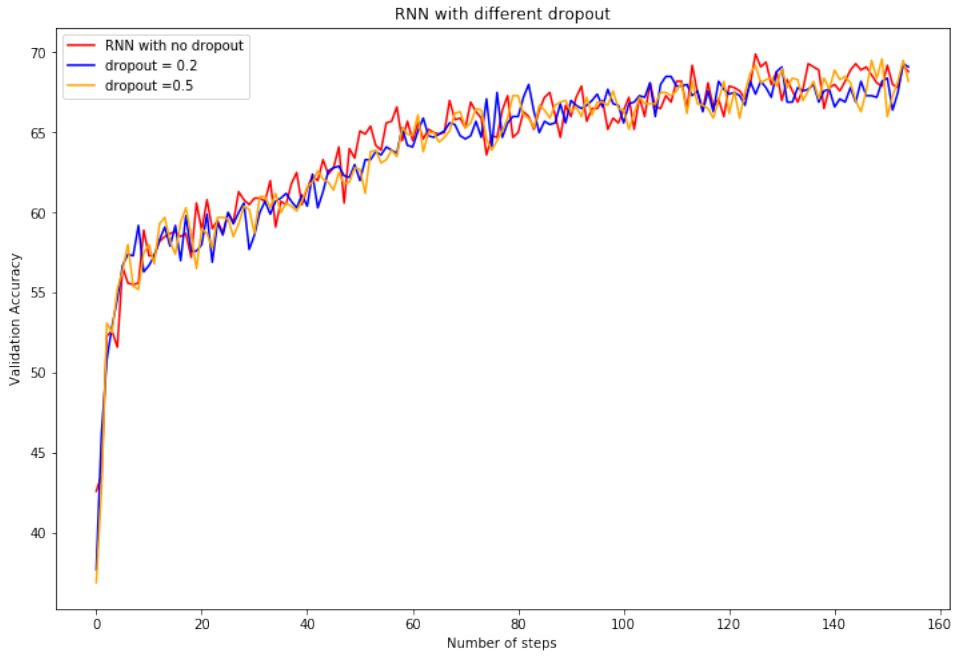
4 Task

4.1 Training on SNLI

1. Exploration of RNN dropout rate p

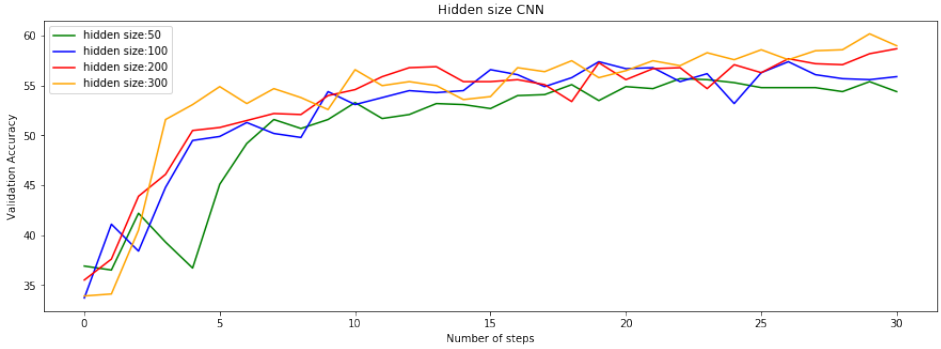
To avoid over-fitting, introduce dropouts on a model is a common improvement. Nodes are ignored with the probability of 1-p in the model so that the interdependence between neurons could be reduced as a regularization.

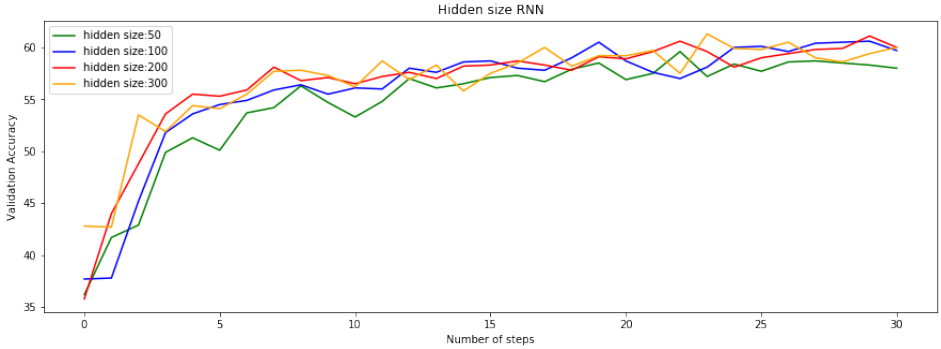
As the result shows by comparing RNN model with(0.2 and 0.5) and without dropout. Dropout = 0.5 is usually used to improve performance for complex RNN model with multiple layers. However, in this example, there is no much difference between dropouts with single layer RNN. It turns out that for single layer RNN, the performance is not necessarily improved by introducing dropout.



2. Exploration of hidden size for RNN and CNN

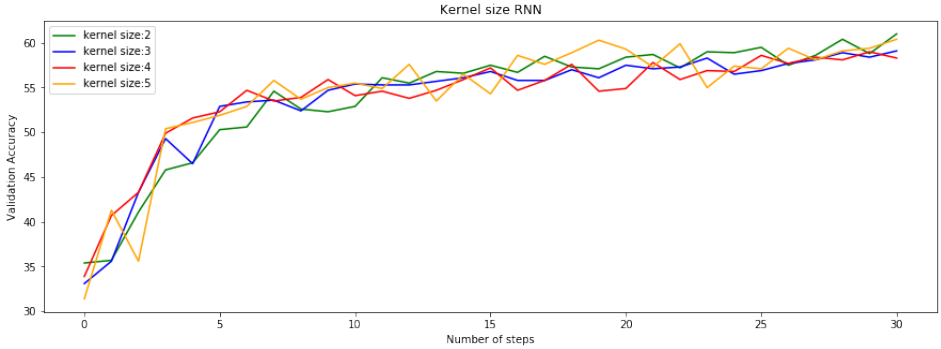
   The hidden size is changed from [50, 100, 200, 300]. It turns out that hidden size = 300 perform slightly better for both CNN and RNN model in this case. A higher hidden size will make the model more complex and may improve the performance of the model in this example. Therefore, hidden size = 300 is selected for both CNN and RNN.





3. Exploration of kernel size for CNN

   kernel size of CNN here represents the shape (2*2, 3*3, 4*4, 5*5) of filter mask when performing convolutional filtering. The kernel size is usually chosen for different type of features in the data. In addition, a larger The model itself may reach the limit on capturing features because of the limit training data and the performance is not changed a lot on different kernel size. Therefore, the kernel size = 3 is chosen for CNN model.



Therefore, the final selection based on parameter exploration for RNN and CNN are:

RNN(emb size=300, hidden size= 300, num layers = 1, linear size =300, dropout p = 0.2, num classes=3) validation accuracy : 69.1

CNN(emb size=300, hidden size=300, linear size =300, kernel size=3, num classes=3) and smaller number of epochs = 4 validation accuracy: 65.1

4. Right and wrong output from current model

```
idx2sentence(right_l)[0]
```

```
['Three women on a stage , one wearing red shoes , black pants , and a gray shirt is sitting on a prop , another is s
itting on the floor',
 'There are two women standing on the stage',
 'contradiction',
 'contradiction']
```

```
idx2sentence(right_l)[1]
```

```
['Four people sit on a subway two read books , one looks at a cellphone and is wearing knee high boots .',
 'Multiple people are on a subway together , with each of them doing their own thing .',
 'entailment',
 'entailment']
```

```
idx2sentence(right_l)[2]
```

```
['bicycles stationed while a group of people socialize .',
 'People get together near a stand of bicycles .',
 'entailment',
 'entailment']
```

For right examples: The first example clearly catches the number of women on the stage and reach the conclusion of contradiction.

Also from the third example we can see that the model can understand stationed bicycles and stand of bicycles. The people from both sentence is learned by the model.

```
idx2sentence(wrong_l)[0]
```

```
['Two people are in a green forest .',
 'The forest is not dead .',
 'entailment',
 'contradiction']
```

```
idx2sentence(wrong_l)[1]
```

```
['Two women , one walking her dog the other pushing a .',
 'There is a snowstorm .',
 'contradiction',
 'netrual']
```

```
idx2sentence(wrong_l)[2]
```

```
['A large group of people stand outside on a road while people on a higher level look on at them .',
 'One group of people are watching what another group does .',
 'entailment',
 'netrual']
```

For wrong examples:

From first example, People in a green forest do indicate that the forest is not dead. However, it is not an obvious conclusion. It seems hard for model to catch relationship between "green" and "not dead" as there may not be many training data on these two terms.

The second example is a good example to indicate the difference between human understanding and model understanding. From the point of the model, it can be True that walk dog outside has nothing to do with the snowstorm. It is hard for the model to build up the link between "snowstorm" and "walk a dog".

The sentence from the third example can be vague from the point of the model. It may not be able to match "people from high level" and "One group of people" together. Therefore, the wrong prediction is neutral.

4.2 Evaluating on MultiNLI

1. MNLI data and validation with current model

	sentence1	sentence2	label	genre
0	and now that was in fifty one that 's forty ye...	It was already a problem forty years ago but n...	neutral	telephone
1	Jon could smell baked bread on the air and his...	Jon smelt food in the air and was hungry .	neutral	fiction
2	it will be like Italian basketball with the uh...	This type of Italian basketball is nothing lik...	contradiction	telephone
3	well i think that 's about uh that 's about co...	Sorry but we are not done just yet .	contradiction	telephone
4	Good job tenure , that is -- because in yet an...	Dr. Quinn , Medicine Woman , was worked on by ...	entailment	slate

Data processing first and then send the input to data loader function that I had before. Here I use both trained RNN and CNN model on validation MNLI data.

```
dic_gen_rnn = {}
for genre in l_g:
    dataset = data_processing(MNLI_val.loc[MNLI_val['genre']  == genre])
    val_dataloader = torch.utils.data.DataLoader(dataset= VocabDataset(dataset),
                                          batch_size=BATCH_SIZE,
                                          collate_fn=vocab_collate_func,
                                          shuffle=False)
    val_acc_score = test_model(val_dataloader, model_rnn)
    dic_gen_rnn[genre] = val_acc_score

dic_gen_cnn = {}
for genre in l_g:
    dataset = data_processing(MNLI_val.loc[MNLI_val['genre']  == genre])
    val_dataloader = torch.utils.data.DataLoader(dataset= VocabDataset(dataset),
                                          batch_size=BATCH_SIZE,
                                          collate_fn=vocab_collate_func,
                                          shuffle=False)
    val_acc_score = test_model(val_dataloader, model_cnn_final) # cnn model
    dic_gen_cnn[genre] = val_acc_score
```

```
dic_gen_rnn
```

```
{'telephone': 43.18407960199005,
 'fiction': 45.62814070351759,
 'slate': 41.71656686626746,
 'government': 43.996062992125985,
 'travel': 44.70468431771894}
```

```
dic_gen_cnn
```

```
{'telephone': 41.791044776119406,
 'fiction': 42.71356783919598,
 'slate': 39.421157684630735,
 'government': 41.43700787401575,
 'travel': 42.56619144602851}
```

2. MNLI result table

   The Multi-Genre Natural Language Inference (MultiNLI) task is a variant of SNLI which covers spoken and written text from different sources, or genres.

   Compared with average 65 accuracy with the SNLI training data. The accuracy for MNLI data is around 42. From the table, the trained RNN model behaves slightly better than the CNN model. Different text sources and genres may make it hard for SNLI models to predict the right conclusion as more patters are included in MNLI data.

   Across the genre, the model has a relatively good prediction on Fiction and Travel genre, on the other hand the model has a bad prediction on Slate magazine. The reason would be the SNLI resource for our training data. The SNLI traning dataset may has more Fiction related text compared to text from Slate.

```
MNLI accuracy table:
```

	RNN	CNN
Telephone	43.184080	41.791045
Fiction	45.628141	42.713568
Slate	41.716567	39.421158
Government	43.996063	41.437008
Travel	44.704684	42.566191