Assignment Two

Qintai Liu (QL819)

October 28, 2018

Abstract

This report is about how I implemented RNN/CNN-based Natural Language Inference model, what kind of hyper-parameters I tried to tune, and analyzing the result.

1 Natural Language Inference

Natural language inference is a task to identify the relationship between a premise and a hypothesis. And there are total three relationships, which are entails, contradicts, and neutral.

2 Implementation

2.1 Data loader and Pre-trained word embedding

To load the SNLI data, I use torchtext.TabularDataset. And I select fast-Text as pre-trained word embedding

2.2 RNN-based model

For each example, it's composed of two sentences and the relationship between them. For each sentence, I use bi-directional GRU to get two vectors representation for the same sentence. To combine these two vectors, I use max pooling to form the final vector representation for a sentence. Finally, I try four ways to interact the two encoded sentences, which are concatenation, element-wise multiplication, element-wise max pooling, and average.

For the number of trained parameters on GRU(input_size = word_embedding_size =300),

- h_0: the initial hidden state for sentence A size: =(hidden_size)
- h_1: the initial hidden state for sentence B size: =(hidden_size)
- weight_ih_l0: the learnable input-hidden weights size =(3*hidden_size, input_size)
- weight_hh_l0: the learnable hidden-hidden weights size =(3*hidden_size, hidden_size)
- bias_ih_l0: the learnable input-hidden size =(1,3*hidden_size)
- bias_hh_l0: the learnable hidden-hidden bias size =(3*hidden_size)
- since we use bi-directional GRU, we also have weight_ih_l0_reverse, weight_hh_l0_reverse, bias_ih_l0_reverse, bias_hh_l0_reverse

For the number of trained parameters on the classifier, it depends on the way about how to interact with the two encoded sentences.

For concatenation,

- linear_1.weight size: =(hidden_size,2*hidden_size)
- linear_1.bias size: =(hidden_size)
- linear_2.weight size =(3, hidden_size)
- linear_2.bias size =(3)

For element-wise multiplication, element-wise max pooling, and average,

- linear_1.weight size: =(3,hidden_size)
- linear_1.bias size =(3)

2.3 CNN-based model

I use a 2-layer 1-D convolutional network and ReLU as activation function. Then use max pooling to form the vector representation for a sentence. And two sentences vectors are concatenated and feed through two-layer neutral network to generate the possibility for each relationship.

For the number of trained parameters on 2-layer CNNs (input_size = word_embedding_size =300, kernel_size=3),

- CNN_1.weight: the learnable weights of the first layer CNN size: =(hidden_size, input_size, kernel_size)
- CNN_1.bias: the learnable bias of the first layer CNN size: =(hidden_size)
- CNN_2.weight: the learnable weights of the second layer CNN size =(hidden_size, hidden_size, kernel_size)
- CNN_2.bias: the learnable bias of the second layer CNN size =(hidden_size)

The number of trained parameters on the classifier, when interacting with the two encoded sentences by concatenation,

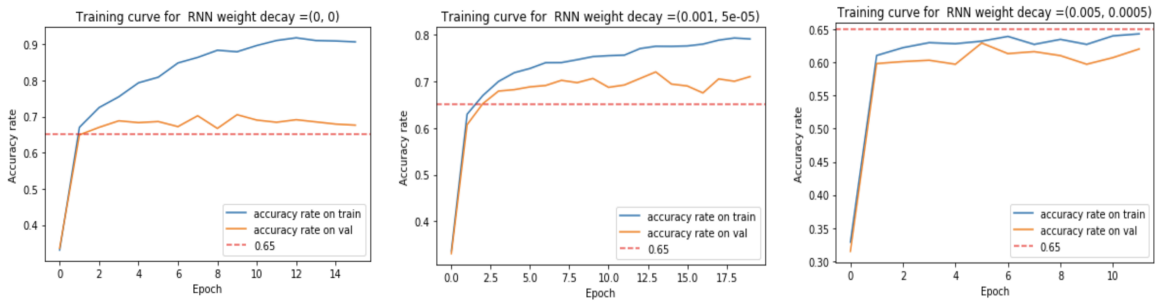
- linear_1.weight size: =(hidden_size,2*hidden_size)
- linear_1.bias size: =(hidden_size)
- linear_2.weight size =(3, hidden_size)
- linear_2.bias size =(3)

3 Analysis

3.1 weight_decays

In the hyper-parameters tuning for weight_decays, two sentences' vector representation are just concatenated together and the hidden_size is 300.

3.1.1 RNN-based model



(a) weight decay classifier = 0.0, weight decay GRU=0.0 (b) weight decay classifier = 0.001, weight decay GRU=$5 * 10^{-5}$ (c) weight decay classifier = 0.005, weight decay GRU=$5 * 10^{-4}$

Figure 1: how weight_decay affect RNN-based model's performance

For RNN-based model, I separate the weight_decay between the GRU and the two_layers neutral network. And let weight_decay for the two-layers neutral network is slightly higher than the GRU's. The reason is that the input for the first layer would be a vector with length 600 and the output would be a vector with length 300 which would be a input for the second layer. The output for second layer would be a vector with size 3. So, the matrix size for the first layer would be 600*300, which is relative big. Therefore, I want to put a little bit more weight decay on the two-layer classifier.

In details, I try $(weight\_decay_{classifier}, weight\_decay_{gru}) = [(0, 0), (0.001, 0.00005), (0.005, 0.0005)]$

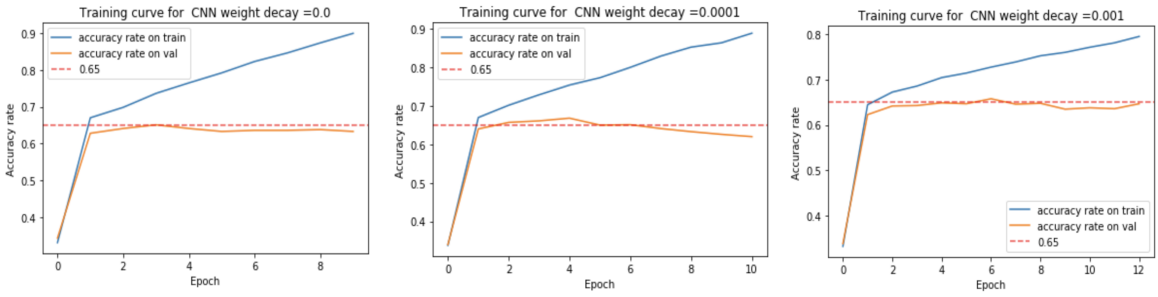
Figure1 shows the training and validation curves with respect to different values of weight_decay for two-layer classifier and GRU respectively. The table2 is the summary about the best validation accuracy for each weight_decay setting. From the figure2, we can see that a very small value of weight_decay does help improve the model's performance, but too

	classifier,GRU	best_epoch	best accuracy(validation data)
0	(0, 0)	9	0.705
1	(0.001, 5e-05)	13	0.720
2	(0.005, 0.0005)	5	0.629

Figure 2: weight_decay for Rnn-model

much weight_decay can actually destroy model's performance. In our case, then the weight_decay for classifier is 0.001 and for GRU is 0.00005, the model achieves best performance, which is 0.72 accuracy rate on validation examples. One thing is worth noticing is that even if the weight_decay only increases a little bit, from 0.001 to 0.005 and 0.00005 to 0.0005 respectively in our case, the performance drops a lot. When there is no weight_decay, the model performance is also relative good.

3.1.2 CNN-based model



(a) weight decay = 0 (b) weight decay = 0.0001 (c) weight decay = 0.001

Figure 3: how weight_decay affect CNN-based model's performance

For CNN-based model, I try $weight\_decay = [0.0, 0.0001, 0.001]$

	CNN weiget	best_epoch	best accuracy(validation data)
0	0.0000	3	0.651
1	0.0001	4	0.668
2	0.0010	6	0.658

Figure 4: weight_decay for CNN-model

Figure3 shows the training and validation curves with respect to different values of weight_decay for CNN-based model. The table4 is the summary about the best validation accuracy for each weight_decay setting. From the figure4, we can see that the overall performance of CNN-based model is worse than RNN-based model. An appropriate weight_decay setting could improve the model's performance a little bit. When the weight_decay=0.0001, the model has highest accuracy on validation examples which is 0.668.

3.2 Hidden_size

In the hyper-parameters tuning for hidden_size, two sentences' vector representation are just concatenated together and weight_decay is set to the best setting based on previous section.

	RNN_hidden_size	best_epoch	best accuracy(validation data)
0	100	20	0.698
1	200	18	0.713
2	300	10	0.724
3	400	13	0.710

(a) hidden_size for RNN-model

	CNN_hidden_size	best_epoch	best accuracy(validation data)
0	100	10	0.656
1	200	5	0.652
2	300	2	0.671
3	400	2	0.661

(b) hidden_size for CNN-model

Figure 5: how hidden_size model's performance

3.2.1 RNN-based model

For RNN-based model, I try $hidden\_size = [100, 200, 300, 400]$

Table5a summarizes the model's performance in terms of validation accuracy for different hidden_size value. From the table, we can see that when the hidden_size is too small, the model is not complex enough to capture

all the relationship among data. When the hidden_size increases, the accuracy rate is also increases until the hidden_size is equal to 400. When hidden_size is 400, this may cause overfitting.

3.2.2 CNN-based model

For CNN-based model, I try $hidden\_size = [100, 200, 300, 400]$

Table5b summarizes the CNN-based model's performance in terms of validation accuracy for different hidden_size value. From the table, we can see that the CNN-based models still have worse performance compared with CNN-based model. The way how hidden_size affects CNN-based models is similar to how RNN-based models are affected. When the hidden_size is to small, the model is not complex enough to capture all the relationship among data. When the hidden_size increases, the accuracy rate is also increases until the hidden_size is equal to 400. When hidden_size is 400, this may cause overfitting.

3.3 Interacting the two encoded sentences

	RNN-based interaction	best_epoch	best accuracy(validation data)
0	concatenation	13	0.720
1	max	7	0.745
2	averge	8	0.655
3	element_wise_multip	7	0.734

Figure 6: Interacting the two encoded sentences

I experiment with 4 different ways of interacting the two encoded sentences in RNN-based model.

- concatenate two sentences vector
- element-wise max(for each index, pick the one with greater value)
- element-wise multiplication
- average two sentences vectors

Figure6 summarizes the result in terms of the performance on validation examples for different interacting ways. We can see that averaging two sentences vectors has the worse performance which makes some sense. Averaging can not distinguish two sentences vectors well and don't know which vector should pay more attention on. Both the element-wise multiplication and element-wise max have the best performance, which also makes some sense especially for element-wise max. Extracting the number with greater value on each dimension make us more likely to capture the important information on each dimension. So the element-wise max give us the best performance with 0.745 accuracy rate on validation examples.

3.4 Best model

After hyper-parameters tuning, the best RNN-based model setting is hidden_size=300, weigh_decay=0 with element-wise max for interaction of two sentences vectors, and the accuracy rate on validation examples is 0.745.

The best CNN-based model setting is hidden_size=300, kernel_size=3, and weight_decay=0.0001 with concatenation of two sentences vectors, and the accuracy rate on validation examples is 0.671.

After extract the incorrect predictions made by the best RNN-based model, below are some explanations about why model makes wrong prediction.

1. sentence1: during this vacation children enjoying their games at the park. sentence2: the kids are home.

   True relationship: contradiction Model prediction: neutral

   Reason: the model may not be able to recognize that children and kids have the similar meaning or the model may not be able to know the relationship between home and park.

2. sentence1: a man is sitting outside wearing blue pants. sentence2: the person is inside.

   True relationship: contradiction Model prediction: entailment

   Reason: the model may misunderstand the relationship between the word 'inside' and the word 'outside'

3. sentence1: performers sing together on stage. sentence2: the performers are singing solo.

   True relationship: contradiction Model prediction: neutral

   Reason: the model may not be able to figure out the different meaning between singing together and singing solo.

Rnn_based model	GENRE	best accuracy(validation data)
0	SNLI	0.745000
1	telephone	0.483582
2	slate	0.435130
3	travel	0.478615
4	government	0.479331
5	fiction	0.491457

(a) Evaluating on MultiNLI for RNN-based model

CNN_based model	GENRE	best accuracy(validation data)
0	SNLI	0.671000
1	telephone	0.456716
2	slate	0.420160
3	travel	0.442974
4	government	0.432087
5	fiction	0.421106

(b) Evaluating on MultiNLI for CNN-based model

Figure 7: Evaluating on MultiNLI

4 Evaluating on MultiNLI

From figure7, we can see that both RNN-based model and CNN-based model have worse performance on MNLI dataset no matter what genre is compare with SNLI dataset. Take RNN-based model as an example, for MNLI dataset, the model has the best performance with 0.491 accuracy rate when the genre is fiction compared with other genres. However, the same model can achieve 0.745 accuracy rate on SNLI dataset. This huge difference gives me a insight that the text classifier I built on SNLI dataset may not generalize well to another domain with different data distribution.

From figure7, we can also see that the slate genre has the lowest accuracy rate and the telephone genre has a relatively high accuracy rate compared with other genres for both RNN-based and CNN-based model, which implies that some genres are more difficult to predict than other genres

5 Fine-tuning on MultiNLI

the genre where fine-tuning	Evaluate on genre telephone	slate	travel	government	fiction
telephone	0.610945	0.521956	0.529532	0.557087	0.559799
slate	0.537313	0.525948	0.551935	0.567913	0.564824
travel	0.541294	0.530938	0.573320	0.559055	0.536683
government	0.532338	0.495010	0.535642	0.582677	0.524623
fiction	0.558209	0.516966	0.536660	0.556102	0.578894

(a) Evaluating on MultiNLI for RNN-based model after fine-tuning

the genre where fine-tuning	Accuracy improvement for each genre telephone	slate	travel	government	fiction
telephone	0.127363	0.086826	0.050916	0.077756	0.068342
slate	0.053731	0.090818	0.073320	0.088583	0.073367
travel	0.057711	0.095808	0.094705	0.079724	0.045226
government	0.048756	0.059880	0.057026	0.103346	0.033166
fiction	0.074627	0.081836	0.058045	0.076772	0.087437

(b) the improvement after fine-tuning

Figure 8: Evaluating on MultiNLI

Table8a represents the evaluation of each fine-tuned model on every genre. The index of Table8a denotes which genre the best SNLI model is trained on. The column denotes which genre the fine-tuned model is evaluated on. For example, the cell at second row(slate) and third column(travel), whose value is 0.551935, represents the validation accuracy rate on travel genre for the fine-tuned model which is fine tuned on slate genre.

Table8b is the combination of Table8a and Table7a. It represents how much the fine-tuned model improves compared with the best Rnn-based model on SNLI dataset. For example, the cell at second row(slate) and third column(travel) in Table8b, whose value is 0.073320, represents the accuracy of the fine-tuned model trained on slate is 0.07 higher than the best SNLI RNN-based model when evaluating on travel genre validation dataset. 0.551935(from Table8a) - 0.478615(from Table7a) = 0.073320(form Table8b)

From Table8b, we can see that the model is fine-tuned on one genre, not only the the accuracy of the fine-tuned model improves in terms of this genre, but also on other genres. For example, the model fine tuned on telephone genre has 0.127 higher accuracy rate, from 0.483 to 0.610, on telephone genre validation dataset. Meanwhile, the accuracy rate improves about 0.05 on other genres.

6 Github link

https://github.com/qltf8/ds-1011-nlp/blob/master/hw2/hw_2_ql819.ipynb