UDACITY

‹ Return to Classroom

DISCUSS ON STUDENT HUB ›

DNN Speech Recognizer

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Great job finishing the project and congratulations on your last project.
I really want you to take this altitude and be audacity in your future career.

Suggestions to improve.

- Use MFCC to avoid overfitting and better learning in this case as our training data is less and huge number of features with sufficient model capacity will only over-fit the model.
- As we have seen that deeper network performs better, we should apply the idea of recur_layers here and deepen our network for model improvements.
- You can refer to this paper . Its simple to read and helps you grasp a deeper understanding for ASR and provides you opportunity to learn better. https://arxiv.org/pdf/1512.02595v1.pdf
- Experiment with dropout as well if the time permits.

STEP 2: Model 0: RNN

✓

The submission trained the model for at least 20 epochs, and none of the loss values in `model_0.pickle` are undefined. The trained weights for the model specified in `simple_rnn_model` are stored in `model_0.h5`.

Great job but you can see this simple model doesn't fit the data very well and thus the loss is very high.

STEP 2: Model 1: RNN + TimeDistributed Dense

✓

The submission includes a `sample_models.py` file with a completed `rnn_model` module containing the correct architecture.

Great work.

- Good Job on implementing the correct architecture.
- You can also experiment with variants of RNN here like LSTM and also with activation functions like tanh/relu.
- Detailed variable names are provided.
- Constant naming format is followed. `model.summary()` is printed.

✓

The submission trained the model for at least 20 epochs, and none of the loss values in `model_1.pickle` are undefined. The trained weights for the model specified in `rnn_model` are stored in `model_1.h5`.

You can see that this model performed a bit better.

- The model performance improves by ~5x due to BatchNorm and TimeDistributed.
- Time distributed helps to apply dense layer to each timestep rather than just the final state and
- Batch normalization decreases the variance between training samples within a batch
- Try providing inline comments as well. Good work done

STEP 2: Model 2: CNN + RNN + TimeDistributed Dense

✓

The submission includes a `sample_models.py` file with a completed `cnn_rnn_model` module containing the correct architecture.

This is an overfitting model example ;)

- You should note here that on using CNN, your training loss decreases drastically but your validation loss is not.
- A typical example of over-fitting. Had we had ran this for more epochs, the over fit would have increased.
- Dropouts are a way to combat it.
- Try using MFCC as a feature, it would avoid overfitting of model.
- MFCC contains the most important features whereas spectogram contains the full breadth of features .
- MFCC turns out to be better when we have a small data to train as in this case else spectogram is better for large scale fine grained training.

The main idea behind MFCC features is the same as spectrogram features: at each time window, the MFCC feature yields a feature vector that characterizes the sound within the window. Note that the MFCC feature is much lower-dimensional than the spectrogram feature, which could help an acoustic model to avoid overfitting to the training dataset.

✓

The submission trained the model for at least 20 epochs, and none of the loss values in `model_2.pickle` are undefined. The trained weights for the model specified in `cnn_rnn_model` are stored in `model_2.h5`.

Awesome!
The submission trained the model for 20 epochs, and all of the loss values are defined.

STEP 2: Model 3: Deeper RNN + TimeDistributed Dense

✓

The submission includes a `sample_models.py` file with a completed `deep_rnn_model` module containing the correct architecture.

Well done on using `recur_layers` to make your model more robust as now by a single parameter you can make your architecture as deep as you want

✓

The submission trained the model for at least 20 epochs, and none of the loss values in `model_3.pickle` are undefined. The trained weights for the model specified in `deep_rnn_model` are stored in `model_3.h5`.

Good training.
The submission trained the model for 20 epochs, and all of the loss values are defined.

STEP 2: Model 4: Bidirectional RNN + TimeDistributed Dense

✓

The submission includes a `sample_models.py` file with a completed `bidirectional_rnn_model` module containing the correct architecture.

Your model is a bit overfitting.

- Bidirectional model is implemented which helps in capturing the context on both forward and backward direction.
- Correct architecture is implemented. Good Job!!
- For how can we improve performance using Bidirectional you should read this paper https://arxiv.org/abs/1606.06871https://arxiv.org/abs/1606.06871

✓

The submission trained the model for at least 20 epochs, and none of the loss values in `model_4.pickle` are undefined. The trained weights for the model specified in `bidirectional_rnn_model` are stored in `model_4.h5`.

The submission trained the model for 20 epochs.
All of the loss values are defined.

STEP 2: Compare the Models

✓

The submission includes a detailed analysis of why different models might perform better than others.

- Explanation captures the essence of learning which was expected.
- You could have avoided `model_0` from graph so that the remaining graph would not have become so compressed.
- Indeed Deeper network performs better but they also have tendency to over-fit since they have huge number of params, in such scenario we should use dropout.
- All in all great work done explanation justifies the implementation.

I really appreciate your conclusion paragraph and you also point out what could be done for the final model that provides thorough thoughts behind your thoughts. Thank you for sharing that!

STEP 2: Final Model

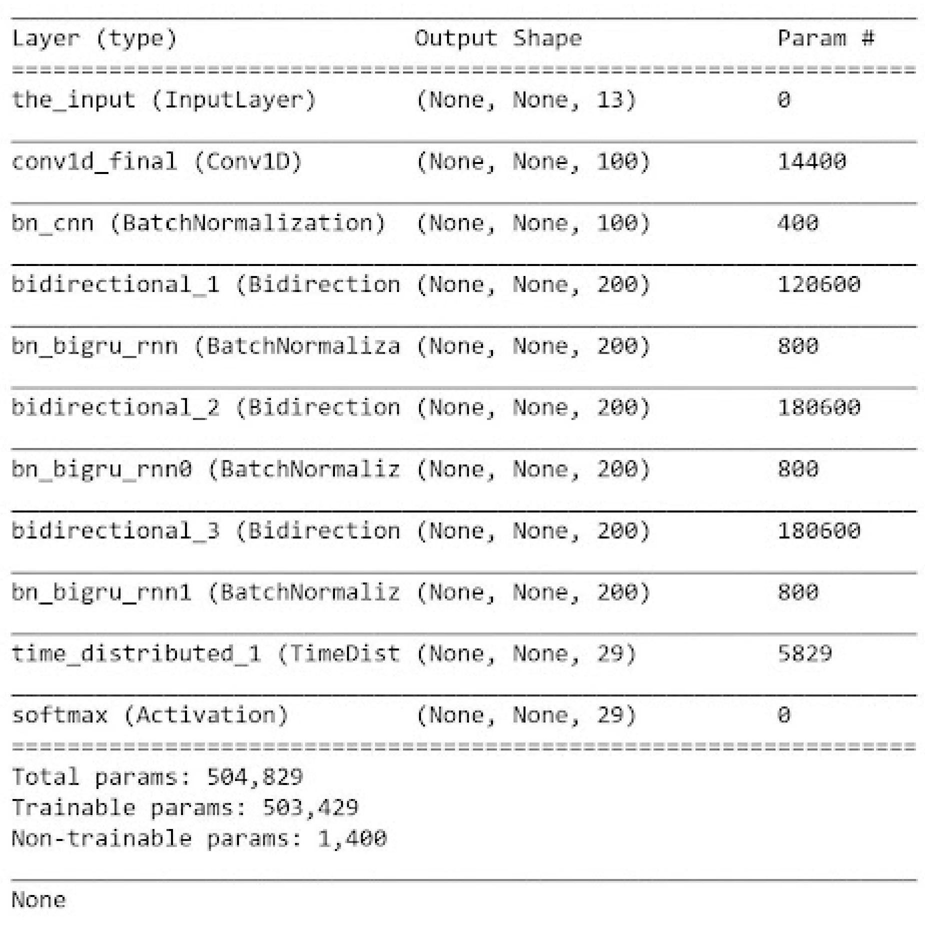
✓

The submission trained the model for at least 20 epochs, and none of the loss values in `model_end.pickle` are undefined. The trained weights for the model specified in `final_model` are stored in `model_end.h5`.

Interesting model. It performs pretty well, and if you used more epochs I bet the loss would decrease even further.

However, you are overfitting by a little bit, and thus would likely see improved results if you were more aggressive with dropout and recurrent dropout.

Here, I have a suggested model structure for you if you want to try more



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
the_input (InputLayer)       (None, None, 13)          0
_________________________________________________________________
conv1d_final (Conv1D)        (None, None, 100)         14400
_________________________________________________________________
bn_cnn (BatchNormalization)  (None, None, 100)         400
_________________________________________________________________
bidirectional_1 (Bidirection (None, None, 200)         120600
_________________________________________________________________
bn_bigru_rnn (BatchNormaliza (None, None, 200)         800
_________________________________________________________________
bidirectional_2 (Bidirection (None, None, 200)         180600
_________________________________________________________________
bn_bigru_rnn0 (BatchNormaliz (None, None, 200)         800
_________________________________________________________________
bidirectional_3 (Bidirection (None, None, 200)         180600
_________________________________________________________________
bn_bigru_rnn1 (BatchNormaliz (None, None, 200)         800
_________________________________________________________________
time_distributed_1 (TimeDist (None, None, 29)          5829
_________________________________________________________________
softmax (Activation)         (None, None, 29)          0
=================================================================
Total params: 504,829
Trainable params: 503,429
Non-trainable params: 1,400
_________________________________________________________________
None
```

✓

The submission includes a `sample_models.py` file with a completed `final_model` module containing a final architecture that is not identical to any of the previous architectures.

Great model on suggesting the model and implementing it based on your comparison from earlier discussion.

✓

The submission includes a detailed description of how the final model architecture was designed.

Nice job. Your reasoning is sound here. Did the model perform as well as you thought it would? Why or why not?

Thanks for the model explanation. You are really creating a deep model, and the results could be better if you try to normalization it better to generalize the training.

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review

START