UDACITY

‹ Back to Artificial Intelligence Nanodegree and Specializations

DNN Speech Recognizer

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Hi there!

Congratulations on successfully completing this project! ✨ 🎉

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`.

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.

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`.

Adding batch normalization and a time distributed layer improves the loss by ~6x! Try different units here, `SimpleRNN`, `LSTM`, and `GRU` to see how their performance differs.

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.

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`.

These models are very powerful but have a tendency to severely overfit the data. **You can add dropout to combat this.**

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.

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`.

Adding additional layers allows your network to capture more complex sequence representations, but also makes it more prone to overfitting. **You can add dropout to combat this.**

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.

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`.

These models tend to converge quickly. They take advantage of future information through the forward and backward processing of data.

STEP 2: Compare the Models

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

This is a pretty good analysis of each model. An improvement would be to explain why different models perform better than others. For example, what is it about the nature of a CNN on this problem that leads to extreme overfitting?

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.

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.

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?

DOWNLOAD PROJECT

RETURN TO PATH

Rate this review