Artificial Neural Networks and Deep Learning 2020

Homework 3 - Visual Question Answering

Team: StavoCorLibbanese

Amorosini Francesco - Awad Yasmin - Fioravanti Tommaso

Model:

In this challenge we used two different kinds of models merged together: a CNN as a feature extractor for the images, and a RNN for sequence learning.

- CNN: our feature extractor has been built using a standard architecture with subsequent Convolutional (*size = 3x3*, *strides = 1x1*) and MaxPooling layers (*size = 2x2*). The number of filters starts from 10 and doubles as the depth of the network increases. The model has a *depth = 5*. The top of the CNN is composed of three fully connected layers (*units = 512*) interleaved with Dropout layers (*0.3*).
- RNN: we embedded our questions using an Embedding layer, and then we fed its output to two stacked LSTM modules. As our input questions can be quite a long sequence (at most 21 words), the LSTM modules are explicitly designed to avoid the long-term dependency problem. Both of our LSTM modules have been wrapped with a Bidirectional layer, and then we applied a slight Dropout (*0.1*) in order to prevent potential overfitting.

Then we concatenated together our CNN and RNN (i.e each image feature vector was concatenated with its embedded question), and we fed the output to two final dense layers (with *128* and *58 units* respectively) interleaved with another Dropout (*0.1)* layer.

The above model has been trained for 30 epochs with an early stopping policy (*patience = 8*), and it produced the best results among all of our experiments (`val_accuracy: 0.5510`,`test_accuracy: 0.5299`)

Layer (type)	Output Shape	Param #	Connected to
input_7 (InputLayer)	[(None, 21)]	0	
embedding_1 (Embedding)	(None, 21, 512)	10752	input_7[0][0]
bidirectional_2 (Bidirectional)	(None, 21, 400)	1140800	embedding_1[0][0]
input_6 (InputLayer)	[(None, 400, 700, 3)	0	
bidirectional_3 (Bidirectional)	(None, 400)	961600	bidirectional_2[0][0]
sequential_4 (Sequential)	(None, 200)	21163188	input_6[0][0]
concatenate_1 (Concatenate)	(None, 600)	0	bidirectional_3[0][0] sequential_4[0][0]
dense_13 (Dense)	(None, 128)	76928	concatenate_1[0][0]
dropout_9 (Dropout)	(None, 128)	0	dense_13[0][0]
dense_14 (Dense)	(None, 58)	7482	dropout_9[0][0]

Total params: 23,360,750
Trainable params: 23,360,750

QA encoding:

The questions have been preprocessed with the help of a Tokenizer, which mapped each word into the index of the corresponding dictionary entry. Then, as each question was likely to have a different number of words, we added some padding in order to have them all with the same length. The final embedding of the questions is performed using the Embedding layer of the RNN.
For what concerns the answers we decided to associate each answer to a class label, so we treated the original problem as a classification problem with 58 classes. This allowed us to use the CategoricalCrossEntropy as a loss function for the whole model.

Other experiments:

- ### VGG:

  As CNN, we have also tried the fine-tuning technique (vqa_model_1 in the notebook) using VGG as a pretrained model, with 15 layers freezed, a Dropout and a Dense layer with 1024 units. This for the CNN feature extractor part of the network. For what concerns the RNN's part of the network we've used the same structure already presented in the best model. This model has been trained for 20 epochs with an early stopping policy (*patience* = *8*), but it didn't improve the handmade CNN performances (`val_accuracy: 0.4332,test_accuracy: 0.41902`).
  In our opinion VGG didn't perform very well because the original images on which it has been trained were very different with respect to those we had in this challenge. Performance may have improved if we completely re-trained VGG, but at that point we decided that it would have been better to write our own CNN from ground up.

- ### InceptionV3:

  Among all the models that we tried, we also got relevant results using the InceptionV3 network. We also tried to add a GAP layer instead of the usual fully connected top (vqa_model_2 in the notebook); this for the image feature extractor's part. For what concerns the RNN's part of the network we've used the same structure already presented in the best model.
  Having less parameters, this model was faster to train, so we tried to increase the complexity of the RNN by stacking multiple bidirectional LSTM modules, but in the end the best performances were achieved with just two of them. This model has been trained for 50 epochs with an early stopping policy (*patience* = *8*), but it didn't improve the handmade CNN performances (`val_accuracy: 0.5123, test_accuracy: 0.50878`).