50.021 -AI

Alex

Week 04: Tensorflow

[The following notes are compiled from various sources such as textbooks, lecture materials, Web resources and are shared for academic purposes only, intended for use by students registered for a specific course. In the interest of brevity, every source is not cited. The compiler of these notes gratefully acknowledges all such sources.]

In this class

- the same neural net we had, now in tensorflow (before we move to convolutions)
- adding summaries that monitor train and validation error even those are not available in tensors

in class coding:

Consider `nnet2c_studentversion.py`. It has the typical structure:

- generate placeholder
- generate a feed dict from placeholders and a function that fetches the next batch from the dataset
- define a network
- define a tensor for comparing neural network predictions versus labels
- define a loss function for training
- define an optimizer
- generate an operator that minimizs the loss
- We defined a `do_eval` function, that takes in a session, the eval tensor, and placeholders to compute evaluation performance. This is one way to obtain results, encapsulated in python functions.

For a real deep network tensorflow suggests tfrecord for training. see `https://kwotsin.github.io/tech/2017/02/11/transfer-learning.html`

There are two tasks in it:

1. Your first task: complete the neural network definition. Prefilled is the first layer in the method `somenetclass.predict`. The definition of weight and biases in it are given in: `somenetclass.paramsdict`. A look at the weights and biases tells that the next layer has 250 neurons. add two more fully connected layers with 100 and 10 neurons.

   The structure should be: Fullyconnected(10,250)-ReLU-Fullyconnected(250,100)-ReLU-Fullyconnected(100,10) . It should return the 10 dimensional output of the last linear layer.

   Hint: to define the dimension of Variables, consider the dimensions of input and output layers please.

2. Your second task: `run_training()` calls `do_eval` for training, eval and validation data. it can receive the outputs in python scalars (check `do_eval`). Implement a summary inside the for loop that fetches the python scalars and visualized them in tensor board.

   One way is (it creates three separate diagrams)

   (a) outside the for loop: create for every performance measure (train accuracy, val accuracy, test accuracy) one variable tensor, it must be initialized.

   (b) outside the for loop: use tf.summary_scalar on it, this returns a summary tensor

   (c) inside the for loop: use sess.run to assign the variable the value of the performance measure

   (d) inside the for loop: then run the summary tensor by sess.run, this gives a string

   (e) inside the for loop: use add_summary to add this string to a summary writer

   (f) repeat this for all variables ... they can be put into one summary writer

   if you want all three measures in one graph in tensorboard, then there is a trick to achieve that:

   (a) outside the for loop create only one variable tensor, and only one summary tensor, but three different summary writers who write into three subdirectories.

   (b) inside the for loop: do for every performance measure:

      i. use sess.run to assign the variable the value of the performance measure

      ii. then run the summary tensor by sess.run, this gives a string.

iii. write the string into one of the writers.

Here you write all three results into the same variable and run the only summary tensor This works?! Can use this also for monitoring train and test (crossentropy-) losses in one graph.

3. why one usually splits the non-train data into a val and test set ? does it makes sense for this code?

4. bonus: can you get better by stacking more smaller linear layers with one more layer?