Ryan George
Deep Learning HW5

Problem 1

1. In your own words describe how a variational autoencoder (VAE) works. Include details about the architecture and how it is trained. What are some applications of a VAE?

> A VAE is a network designed to generate new examples of a data collections. For instance, we might create a VAE to simulate CV's of Yale students, or to generate good celebrity names for aspiring chefs.
>
> The architecture of a VAE is similar to other autoencoders in that it contains three main elements: and encoder, a compressed representation of the data, and a decoder. Special to a VAE is the insertion of a representative probability layer just before the compressed representation. VAEs are trained in similar ways to other autoencoders--through backpropogation.

2. In your own words, describe how a restricted Boltzmann machine (RBM) works. Include details about the architecture and how it is trained. What are some applications of an RMB?

> An RBM seeks to approximate the probability distribution over a set of dimensions within a dataset. The representation of a training example is split into visible and hidden layers that can be fully connected. Training is done through gradient descent, but contrastive divergence is used instead of backpropogation. Contrastive divergence decreases energy (increases probability) in observed portions of the distribution and increases energy in unobserved portions of the distribution.

3. In your own words, describe how a generative adversarial network (GAN) works. Include details about the architecture and how it is trained. What are some applications of a GAN?

> A GAN pits two networks against each other. One network is trying to generate examples modeled off a training set, while the other network is trying to determine whether a given input is from the training set or from first network. The generative and discriminatory networks are trained separately with backpropogation on different/opposing loss functions.
>
> A GAN can be used to generate example datasets (e.g. images). It could generate music, stock photos, wallpaper patterns, etc.

4. Compare and contrast a VAE, RMB, and GAN.

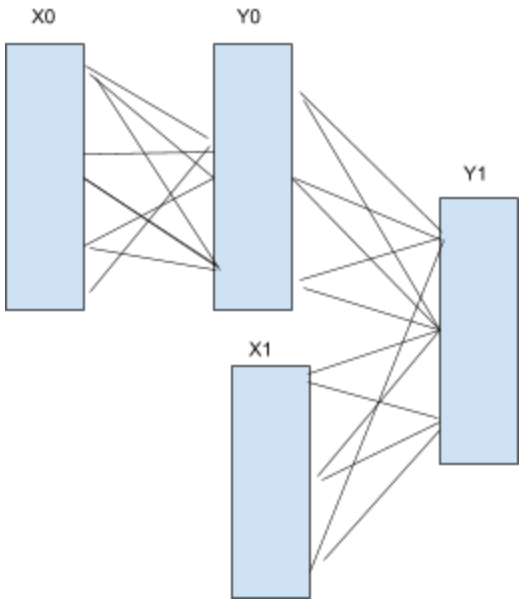
All three of these networks are generative, that is, they can create example data similar to the training data. They ways in which they do this are very different. VAEs and GANs are classic neural networks. A VAE is a feedforward network, trained as such, whereas a GAN is really two networks composed. VAEs are theoretically sound but face practical limitations (such as undesirable convergence to trivial networks). GANs are not theoretically sound, but work well in practice if tricky training obstacles are overcome. An RBM, as an energy-based model, is trained in a completely different manner. It is more concerned with constructing a probability model over both training examples and expected input.

Problem 2

• Look at file tfrnn1.py.

1. Draw a block diagram of the neural network described by this code.

My block representation of the network is as follows:



2. What shape are the X0 batch and X1 batch data? Briefly explain the X0 batch and X1 batch data.

The X0 and X1 batch data are both 3x1 vectors.

3. Explain the functions in lines 21-22 of tfrnn2.py.

The first line creates a set of weights associated with an rnn cell and the second line composes it with the tensorflow placeholders from the two inputs X0 and X1. More specifically, the second specifies both X0 and X1 as inputs to the RNN.

• Look at file tfrnn3.py.

4. Explain lines 35-44.

These lines perform a similar function as described the previous question, but over an arbitrary number of layers. Line 35 takes the tf placeholder, transposes it to put the `n_steps` dimension first, then unstacks it to create a list. Line 41 defines the RNN cell as before, and line 43 connects them as before.

5. Explain the X batch data. Why is it the shape it is, and what do the different dimensions represent?

The `X_batch` variable is shape 4x2x3. The first dimension represents four separate runs/instances of the network. The second dimension represents the two steps/layers in the network. The third dimension represents the neurons in each layer--so the values in the `X_batch` variable are initialization values for these neurons. This is needed in order to perform the recurrent calculations for the network. In other words, these are the values at the edge of the unrolled network.

• Look at file tfrnn4.py.

6. Explain line 25.

Line 25 implements the dynamic_rnn function. This is similar to static_rnn but with dynamic graph unrolling.

7. Compare tfrnn4.py with tfrnn3.py.

The only difference between the code is the use of `static_rnn` versus `dynamic_rnn`. The implementation of `dynamic_rnn` is functionally equivalent to `static_rnn`, but the computation is done differently. Line 25 uses dynamic_rnn so that the graph unrolling of the RNN is done dynamically, that is, at runtime. This will speed up the code in more computationally-intensive situations

• Look at file tfrnn5.py

8. Explain lines 31-32.

This line initializes the rnn but allow for a placeholder to specify the length of the sequence. Based on the values given in line 43, this zeros all output in the second row for the second run.

• Look at file tfrnn6.py

9. What does optimizer =tf.train.AdamOptimizer(learning rate = learning rate) mean?

The Adam optimization method is an adaptive gradient method. The method uses “gradient momentum” so that the update depends on gradients from past steps as well as the current step.

10. How is the accuracy assessed in lines 58-59?

The accuracy is simply the percentage of the time the RNN gets the MNIST handwritten digit correct. (This is the mean of the “correctness” score, which is when the “top 1” digit specified by the network is the correct digit).

Problem 3

Code can be found in `rg_prob3.py`

The difference between sparse and non-sparse softmax is in notation. The sparse version outputs a single label indicating the class, whereas the non-sparse version outputs the entire batch size.

Accuracy is shown here:

Problem 4

Code can be found in `rg_prob4.py`