ELEG 5491: Homework #2

Due on Tuesday, March 14, 2017, 3:30pm (in class)

Xiaogang Wang

Problem 1

[**20 points**]

A recurrent neural network is shown in Figure 1,

$$\mathbf{h}_t = \tanh(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + \mathbf{b}_h)$$

$$\mathbf{z}_t = \text{softmax}(\mathbf{W}_{hz}\mathbf{h}_t + \mathbf{b}_z)$$

The total loss for a given input/target sequence pair ($\mathbf{x}$, $\mathbf{y}$), measured in cross entropy

$$L(\mathbf{x}, \mathbf{y}) = \sum_t L_t = \sum_t -\log z_{t,y_t}$$

In the lecture, we provide the general idea of how to calculate the gradients $\frac{\partial L}{\partial \mathbf{W}_{hz}}$ and $\frac{\partial L}{\partial \mathbf{W}_{hh}}$. Please provide the details of the algorithms and equations, considering the mapping and cost functions provided above.

Problem 2

[**20 points**]

A RNN model is shown in Figure 2. $\mathbf{x}_1, \dots, \mathbf{x}_t$ are input variables. A hidden variable $\mathbf{h}_t = F_\theta(\mathbf{h}_{t-1}, \mathbf{x}_t)$ contains information about the whole past sequence. It defines a function which maps the whole past sequence $(\mathbf{x}_t, \dots, \mathbf{x}_1)$ to the current state $\mathbf{h}_t = G_t(\mathbf{x}_t, \dots, \mathbf{x}_1)$.

- Give the explicit expression of G_t in terms of F_θ, $\mathbf{x}_1, \dots, \mathbf{x}_t$. [**10 points**]

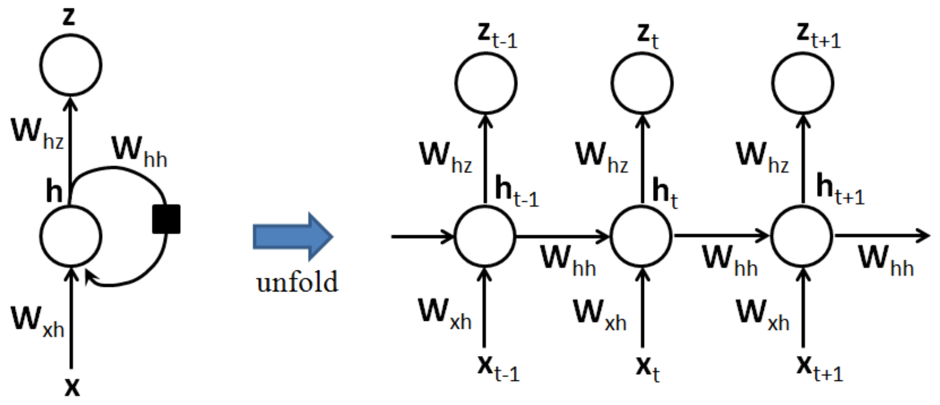

Figure 1:

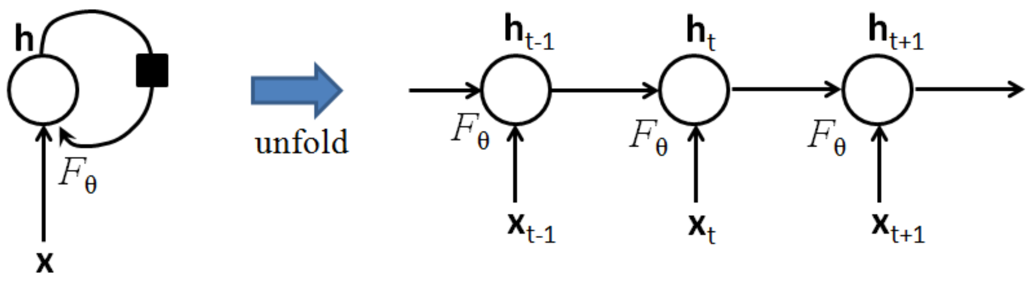

Figure 2:

- There are two major difficulties when directly modeling $\mathbf{h}_t = G_t(\mathbf{x}_t, \ldots, \mathbf{x}_1)$: (1) G_t is different for different t (i.e. sequence length). It limits the generalization power of the model and there may not be enough training samples for each sequence length. (2) The model complexity of G_t may increase exponentially with t. Provide two reasons to explain how RNN solves the two challenges. [**10 points**]

Problem 3

[**10 points**]

Figure 3 is the architecture of CNN customized by Krizhevsky et al. to better leverage model parallelism in multi-GPU training. The architecture consists of two columns each allocated on one GPU. The forward propagation goes through multiple stages. At which stages, do the two GPUs need to be synchronized?

Problem 4

[**20 points**]

Consider a neural network with one input layer (which has d nodes), L hidden layers (each of which has n hidden nodes), and one output layer (which has c nodes). The neurons between two layers are fully connected.

- What is the space complexity of the neural network, i.e. the number of parameters to store? [**5 points**]

Problem 4 continued on next page. . .

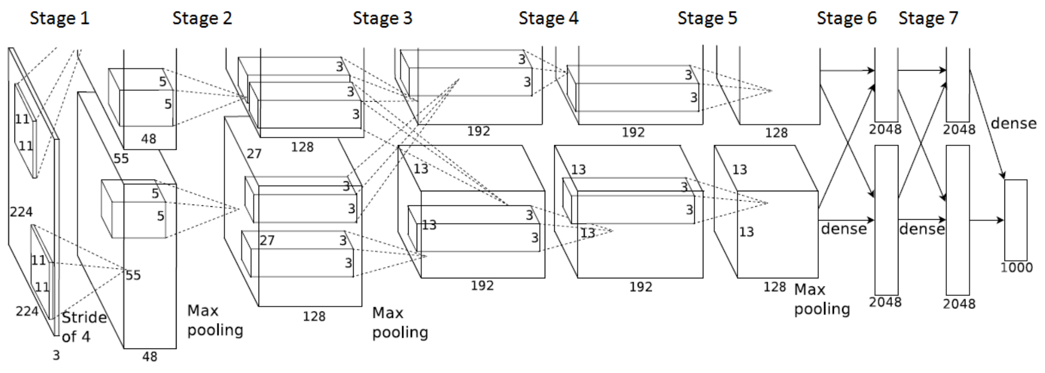

Figure 3:

- Stochastic gradient descent is used for Backpropagation (BP), i.e. only one training sample is used to compute the gradients to update the weights in each training iteration. What is the time complexity for each iteration? Only the number of multiplication operations is considered in this problem. [**5 points**]

- Instead of using BP, the gradients of weights can also be computed in the following way:

$$\frac{\partial u_N}{\partial w_{ji}} = \frac{\partial u_N}{\partial u_i}\frac{\partial u_i}{\partial net_i}\frac{\partial net_i}{\partial w_{ji}},$$

  where u_N is last node to compute the cost function J, w_{ji} is a weight on the connection between node i and node j, net_i is the net activation of node i, and u_i is the output of node i. $\frac{\partial J}{\partial u_i}$ is computed by enumerating all the possible paths connecting nodes i and J, and applying the chain rule to these paths separately.

$$\frac{\partial u_N}{\partial u_i} = \sum_{\text{paths } u_{k_1}\ldots u_{k_n}: k_1=i, k_n=N} \prod_{j=2}^{n} \frac{\partial u_{k_j}}{\partial u_{k_{j-1}}}.$$

  What is the time complexity of this approach of updating the weight at each training iteration? [**10 points**]

Problem 5

[**30 points**]

In this problem we will implement a convolutional neural network to solve the problem of image classification on the CIFAR-10 dataset.

The network should consist of two convolutional layers and two fully connected layers. The first convolution has a kernel size of 5, stride of 1 and outputs 6 channels (without padding); the second convolution shares the same kernel setting and outputs 16 channels. The two fully connected layers have the number of output neurons as 120, 84, respectively. The last layer before the loss is an additional FC layer with ten outputs (the number of object classes on CIFAR-10). Max pooling with stride of 2 is appended after each convolution. And we use ReLU as the activation function throughout the network. Note that you may optionally add the dropout layer after each fully connected layer, depending on the training behavior of your network. The loss is a softmax loss. Other training specifications (recommended and yet not guaranteed):

Problem 5 continued on next page. . .

we use SGD as the optimization method; the base learning rate is 0.001. Momentum and weight decay are set to be 0.9 and 0.0005. The learning rate policy is `step`, which is the way of reducing learning rates. You can refer to AlexNet paper for details. The initial weights of filters are Gaussian distributed with zero mean and 0.01 std. You do not need to worry about these settings. They are already set by default in the code we provide.

We use PyTorch and a tutorial session is scheduled on Feb. 24. The starter code can be downloaded at `https://github.com/eleg5491/hw2_cnn_on_cifar`, which includes the general framework, dataset loading and evaluation process. You can ask issues, pull requests, open discussions via the GitHub link above. Instructions on how to submit your code will be announced later in the `README.md` file of this repository.

Note: We have intentionally designed the network to be *small* so that you can even run the network fast in CPU mode. For adventurers, you can augment the width of the network above to get a sense of the super-efficiency of GPU acceleration (also report it in your submission).

1. **[10 points]** Set up the network architecture specified above. Draw the training curve and test accuracy for at least 5 epoches. Visualize the filters learned in the first convolutional layer. Report and analyze the performance improvement of using softmax loss and regularization.

2. **[10 points]** Weight initialization. A good initialization of the network should ensure the variance of signals among layers constant. For a convolution or fully connected layer, we have

$$\mathbf{y}_l = \mathbf{W}_l \mathbf{x}_l + \mathbf{b}_l, \tag{1}$$

where $\mathbf{y}_l, \mathbf{x}_l$ are the output and input maps of layer l, $\mathbf{W}_l, \mathbf{b}_l$ denote the weights and bias of the filter. We also have $\mathbf{x}_l = f(\mathbf{y}_{l-1})$, where f is the activation function (we use ReLU throughout the discussion). Let y_l, x_l, w_l represent the random variables of each element in $\mathbf{y}_l, \mathbf{x}_l, \mathbf{W}_l$, respectively. Assume that w_l has zero mean and elements in $\mathbf{x}_l, \mathbf{W}_l$ are mutually independent and these two are also independent. Then we have:

$$Var[y_l] = n_l Var[w_l x_l], \tag{2}$$

where n_l is the number of neurons in layer l. In case you are not familiar with the conclusion above, there is a blog. *Verify the following:*

$$Var[y_l] = n_l Var[w_l] E[x_l^2], \tag{3}$$

$$E[x_l^2] = \frac{1}{2} Var[y_{l-1}]. \tag{4}$$

Putting Eqn. 4 back into Eqn. 3 and considering all layers from 1 to L, we have derived the key to the initialization design:

$$Var[y_L] = Var[y_1] \left(\prod_{l=2}^{L} \frac{1}{2} n_l Var[w_l] \right). \tag{5}$$

A good initialization method should avoid reducing or magnifying the magnitudes of input signals exponentially. A sufficient condition from Eqn. 5 is thereby:

$$\frac{1}{2} n_l Var[w_l] = 1, \forall l. \tag{6}$$

From above, we derive a proper way of initializing the weights in the network. Similar conclusion holds for a backpropagation case (see paper for details).

In this question, you need to do two things. (a) Prove Eqn. 4 and Eqn. 3. (b) Now based on the network designed previously, *implement such an idea* to initialize weights in your network and verify if such a design works. Report the test accuracy compared with the result in 1.

Problem 5 continued on next page...

3. **[10 points]** Batch normalization. We have discussed the technique of batch normalization in the lecture. Add BN layer (using existing library) after each convolution and FC layer (except the very last FC for classification) and verify the effectiveness of BN via test accuracy.

4. **[Optional]** Try other ideas. For example, investigate the effect of data augmentation (rotation, scaling, etc.); different optimization policies (Adam, RMSProp, etc.). Note that for each new component, you have to write your own code/layer instead of borrowing from the existing library. And some brief analysis should also be appended.