EECS 598 Deep Learning

Assignment 1

Shuyang HUANG 68621288

1.

Pr.1

1. Fully-connected layer.

$Y = W^T \underline{x} + \underline{b}$ which is: $y_i = \sum_j w_{ji} \cdot x_j + b_i$

$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial w_{ji}} = \frac{\partial L}{\partial y_i} \cdot x_j$, thus $\frac{\partial L}{\partial (W^T)} = \frac{\partial L}{\partial Y} \cdot X^T$

Thus $\frac{\partial L}{\partial W} = \left(\frac{\partial L}{\partial (W^T)}\right)^T = \left(\frac{\partial L}{\partial Y} \cdot X^T\right)^T = X \cdot \left(\frac{\partial L}{\partial Y}\right)^T$

$\frac{\partial L}{\partial b_i} = \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial b_i} = \frac{\partial L}{\partial y_i} \Rightarrow \frac{\partial L}{\partial b} = \frac{\partial L}{\partial Y}$

$\frac{\partial L}{\partial x_j} = \sum_i \frac{\partial L}{\partial y_i} \cdot \frac{\partial y_i}{\partial x_j} = \sum_i \frac{\partial L}{\partial y_i} \cdot w_{ji} = \sum_i w_{ij} \frac{\partial L}{\partial y_i}$

Thus $\frac{\partial L}{\partial X} = W \cdot \frac{\partial L}{\partial Y}$

Thus: $\frac{\partial L}{\partial W} = X \cdot \left(\frac{\partial L}{\partial Y}\right)^T$, $\frac{\partial L}{\partial b} = \frac{\partial L}{\partial Y}$, $\frac{\partial L}{\partial X} = W \cdot \frac{\partial L}{\partial Y}$

2. ReLU

Notice that $Y = ReLU(X) = \max\{0, X\}$

Thus $y_{ij} = \begin{cases} x_{ij} & x_{ij} \geq 0 \\ 0 & x_{ij} < 0 \end{cases} \Leftrightarrow y_{ij} = x_{ij} \cdot \mathbb{1}\{x_{ij} \geq 0\}$

$\frac{\partial y_{ij}}{\partial x_{ij}} = \mathbb{1}\{x_{ij} \geq 0\} \Rightarrow \frac{\partial L}{\partial x_{ij}} = \frac{\partial L}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial x_{ij}} = \frac{\partial L}{\partial y_{ij}} \cdot \mathbb{1}\{x_{ij} \geq 0\}$

Thus $\frac{\partial L}{\partial X} = \mathbb{1}\{X \geq 0\} \cdot \frac{\partial L}{\partial Y}$

3. Dropout

$Y = X \odot M$ Notice $M \in \mathbb{B}^{m\times n}$ $\mathbb{B} = \{0,1\}$, binary set.

Thus $y_{ij} = x_{ij} \cdot m_{ij} \Rightarrow \frac{\partial y_{ij}}{\partial x_{ij}} = m_{ij}$.

$$\frac{\partial L}{\partial x_{ij}} = \frac{\partial L}{\partial y_{ij}} \cdot \frac{\partial y_{ij}}{\partial x_{ij}} = \frac{\partial L}{\partial Y_{ij}} m_{ij}.$$

Thus $\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \odot M$.

4. Batch Normalization.

$Y_i = \gamma\left(\frac{x_i - \mu}{\sigma}\right) + \beta$, $\mu = \frac{1}{n}\sum_j x_j$ $\sigma = \sqrt{\frac{1}{n}\sum_j (x_j - \mu)^2 + \epsilon}$

Obviously, $\frac{\partial \mu}{\partial x_i} = \frac{1}{n}$

$$\sigma = \sqrt{[(1-\tfrac{1}{n})x_i - \tfrac{1}{n}\textstyle\sum_{j\neq i} x_j]^2 + \epsilon} \quad \frac{\partial \sigma}{\partial x_i} = \frac{1}{2\sigma} \cdot 2\left[(1-\tfrac{1}{n})x_i - \tfrac{1}{n}\textstyle\sum_{j\neq i} x_j\right]$$

$$= \frac{1}{\sigma n}[x_i - \mu].$$

$$\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial Y_i} \cdot \frac{\partial Y_i}{\partial x_i} + \sum_j \frac{\partial L}{\partial Y_j}\left(\frac{\partial Y_j}{\partial \mu} \cdot \frac{\partial \mu}{\partial x_i} + \frac{\partial Y_j}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial x_i}\right)$$

$$= \frac{\partial L}{\partial Y_i} \cdot \frac{\gamma}{\sigma} + \sum_j \frac{\partial L}{\partial Y_j}\left(\frac{-\gamma}{\sigma} \cdot \frac{1}{n} + \frac{1}{\sigma n}[x_i - \mu] \cdot [-\frac{1}{\sigma^2}\gamma(x_j - \mu)]\right)$$

$$= \frac{\partial L}{\partial Y_i} \cdot \frac{\gamma}{\sigma} - \frac{\gamma}{\sigma n}\sum_j \frac{\partial L}{\partial Y_j} - \frac{1}{n\sigma^3}\gamma[x_i - \mu]\sum_j \frac{\partial L}{\partial Y_j}(x_j - \mu)$$

Thus $\frac{\partial L}{\partial x_i} = \frac{\partial L}{\partial Y_i} \cdot \frac{\gamma}{\sigma} - \frac{\gamma}{\sigma n}\sum_j \frac{\partial L}{\partial Y_j} - \frac{\gamma}{n\sigma^3}[x_i - \mu]\sum_j \frac{\partial L}{\partial Y_j}(x_j - \mu)$

5. Convolution.

$$\left(\frac{\partial L}{\partial X_{n,c}}\right)_{ij} = \left[\sum_{fpq} \frac{\partial L}{\partial Y_{n,f,p,q}} \cdot \frac{\partial Y_{n,f,p,q}}{\partial X_{n,c,i,j}}\right]$$

$$= \left[\sum_{fpq} \frac{\partial L}{\partial Y_{n,f,i-p+1,j-q+1}} \cdot \frac{\partial Y_{n,f,i-p+1,j-q+1}}{\partial X_{n,c,i,j}}\right].$$

$$= \left[\sum_{f,p,q} \frac{\partial L}{\partial Y_{n,f,i-p+1,j-q+1}} \cdot W_{f,c,p,q}\right].$$

$$= \sum_{f} W_{f,c} *_{full} \frac{\partial L}{\partial Y_{n,f}}.$$

$$\left(\frac{\partial L}{\partial W_{f,c}}\right)_{i,j} = \left[\sum_{n} \frac{\partial L}{\partial W_{f,c,i,j}}\right] = \left[\sum_{n,p,q} \frac{\partial L}{\partial Y_{n,f,p,q}} \cdot \frac{\partial Y_{n,f,p,q}}{\partial W_{f,c,i,j}}\right].$$

$$= \left[\sum_{n,p,q} \frac{\partial L}{\partial Y_{n,f,i-p+1,j-q+1}} \cdot \frac{\partial Y_{n,f,i-p+1,j-p+1}}{\partial W_{f,c,i,j}}\right].$$

$$= \left[\sum_{n,p,q} \frac{\partial L}{\partial Y_{n,f,i-p+1,j-q+1}} \cdot X_{n,c,i,j}\right].$$

$$= \sum_{n} X_{n,c} *_{filt} \left(\frac{\partial L}{\partial Y_{n,f}}\right)$$

2.Logistic Classifier

a.

```
model = LogisticClassifier(input_dim=20, hidden_dim = None, reg = 0,
weight_scale=1e-1)
controller = Solver(model, data,
                    update_rule='sgd_momentum',
                    optim_config={
                    'learning_rate': 1e0,
                },
                    lr_decay=0.95,
                    num_epochs=160, batch_size=50,
                    print_every=100)
```

This is the parameters used, and reached an accuarcy of 92% with test data.

b.

```
model = LogisticClassifier(input_dim=20, hidden_dim = 120, reg = 0,
weight_scale=1e-1)
controller = Solver(model, data,
                    update_rule='sgd_momentum',
                    optim_config={
                    'learning_rate': 1e0,
                },
                    lr_decay=0.95,
                    num_epochs=160, batch_size=50,
                    print_every=100)
```

This is the parameters used, and reached an accuracy of 91.4% with test data.

3. SVM Classifier

a.

```
model = SVM(input_dim=20, hidden_dim = None, reg = 0, weight_scale=1e-2)
controller = Solver(model, data,
                    update_rule='sgd_momentum',
                    optim_config={
                    'learning_rate': 1e0,
                },
                    lr_decay=0.98,
                    num_epochs=200, batch_size=50,
                    print_every=100)
```

This is the parameters used, and reached an accuracy of 92.8% with test data.

b.

```
model = SVM(input_dim=20, hidden_dim = 120, reg = 0, weight_scale=1e-2)
controller = Solver(model, data,
                    update_rule='sgd_momentum',
                    optim_config={
                    'learning_rate': 1e0,
                },
                    lr_decay=0.95,
                    num_epochs=200, batch_size=50,
                    print_every=100)
```

This is the parameters used, and reached an accuracy of 93.2% with test data.

4. Softmax Regression

a.

```
model = SoftmaxClassifier(input_dim=28*28, hidden_dim = None, reg = 0,
num_classes=10, weight_scale=1e-3)
controller = Solver(model, data,
                    update_rule='sgd_momentum',
                    optim_config={
                    'learning_rate': 1e-5,
                },
                    lr_decay=0.95,
                    num_epochs=10, batch_size=50,
                    print_every=100)
```

This is the parameters used, and reached an accuracy of 90.2% with test data.

b.

```
model = SoftmaxClassifier(input_dim=28*28, hidden_dim = 600, reg = 0,
num_classes=10, weight_scale=1e-3)
controller = Solver(model, data,
                    update_rule='sgd_momentum',
                    optim_config={
                    'learning_rate': 1e-3,
                },
                    lr_decay=0.8,
                    num_epochs=4, batch_size=50,
                    print_every=100)
```

This is the parameters used, and reached an accuracy of 96.71% with test data.

5. Convolutional Neural Network

a.

```
model = ConvNet(input_dim=(1, 28, 28), hidden_dim = 600, reg = 0,
num_classes=10, weight_scale=1e-3, drop_out = False)
controller = Solver(model, data,
                    update_rule='sgd_momentum',
                    optim_config={
                    'learning_rate': 1e-3,
                },
                    lr_decay=0.9,
                    num_epochs=15, batch_size=50,
                    print_every=1)
```

This is the parameters used, and reached an accuracy of 98.7% with test data.

b.

```
model = ConvNet(input_dim=(1, 28, 28), hidden_dim = 600, reg = 0,
num_classes=10, weight_scale=1e-3, drop_out = True)
controller = Solver(model, data,
                    update_rule='sgd_momentum',
                    optim_config={
                    'learning_rate': 1e-3,
                },
                    lr_decay=0.9,
                    num_epochs=15, batch_size=50,
                    print_every=1)
```

This is the parameters used, and reached an accuracy of 98.9% with test data.

6. VGG11

With nothing changed, the result shows:

```
Accuracy of the network on the 10000 test images: 70 %
```

7. Short answer question

a.

Sigmoid function, $f(x) = \frac{1}{1+\exp -x}$, maps the the real number range into the range $[0, 1]$. With this realized, values will more likely to be hushed onto values approaching 0 and 1, and thus saturate at the these two points. With inproper initial values or inproper model, sigmoid have the chance to perform pretty bad.

b.

Suppose that, at a probaility p, we randomly drop a neuron, which means, at probability p, the contribution to a output will not counted into our current estimation $\hat{y}$. Thus, to ensure the difference, or the grandient not mismatch too much, we should consider the $(1-p)y$ as the standard values, to reduce the mismatch.

c.

Reduce the decay, thus to ensure the learning step reduce quickly, to avoid the over-fitting.