# hw1

*Yutong Dai*

*9/2/2018*

## Theory

### Forward-Propagation

$$X \to Z = WX + b_1 \to H = \sigma(Z) \to U = CH + b_2 \to S = F_{softmax}(U) \to \rho(S, y) = \log S_y,$$

where $S_y = \frac{\exp(U_y)}{\sum_{j=0}^{K-1}\exp(U_j)}$ is the y-th element of the $S$ and $U_y$ is the y-th element of the $U$.

### Backward-Propagation

$$\frac{\partial \rho}{\partial U_t} = \begin{cases} S_t(U), & t \neq y \\ 1 - S_t(U). & t = y \end{cases} \Longrightarrow \frac{\partial \rho}{\partial U} = e_y - S(U),$$

where $e_y$ is the unit vector, which y-th coordinate equals to 1 and 0 elsewhere.

$$\begin{aligned}
\frac{\partial \rho}{\partial b_2} &= \frac{\partial \rho}{\partial U}\frac{\partial U}{\partial b_2} = e_y - S(U) \\
\frac{\partial \rho}{\partial C} &= \frac{\partial \rho}{\partial U}\frac{\partial U}{\partial C} = (e_y - S(U))H^T \\
\frac{\partial \rho}{\partial H} &= \frac{\partial \rho}{\partial U}\frac{\partial U}{\partial H} = C^T\frac{\partial \rho}{\partial U} = C^T(e_y - S(U)) \\
\frac{\partial \rho}{\partial b_1} &= \frac{\partial \rho}{\partial H}\frac{\partial H}{\partial Z}\frac{\partial Z}{\partial b_1} = \frac{\partial \rho}{\partial H} \odot \sigma'(Z) \\
\frac{\partial \rho}{\partial W} &= \big(\frac{\partial \rho}{\partial H} \odot \sigma'(Z)\big)X^T
\end{aligned}$$

### Algorithm

Mini-Batch Stochastic gradient algorithm for updating $\theta = \{W, b_1, C, b_2\}$:

- Step1: Specify batch_size $M = 1$, activation function $\sigma(z) = Relu(z)$, and initialize $W^{(0)}, b_1^{(0)}, C^{(0)}, b_2^{(0)}$;
- Step2: At iteration $t$:
  - a. Select a data sample $\{X^{(t)}, y^{(t)}\}$ uniform at random from the full dataset $\{X^{(n)}, y^{(n)}\}_{n=1}^N$
  - b. Compute forward-propagation:
    * $Z^{(t)} = W^{(t)}X^{(t)} + b_1^{(t)}$
    * $H^{(t)} = \sigma(Z^{(t)}) = \max(Z^{(t)}, 0)$(element-wise operation)
    * $U^{(t)} = C^{(t)}H^{(t)} + b_2^{(t)}$
    * $S^{(t)} = F_{softmax}(U^{(t)})$
  - c. Compute backward-propagation:

    - $\frac{\partial \rho}{\partial b_2} = e_{y^{(t)}} - S^{(t)}$
    - $\frac{\partial \rho}{\partial C} = (e_{y^{(t)}} - S^{(t)}){H^{(t)}}^T$
    - $\frac{\partial \rho}{\partial H} = C^T(e_{y^{(t)}} - S^{(t)})$
    - $\frac{\partial \rho}{\partial b_1} = \frac{\partial \rho}{\partial H} \odot \sigma'(Z^{(t)})$, where $\sigma'(Z^{(t)}) = 1(if Z > 1)0(o.w)$(element-wise operation)
    - $\frac{\partial \rho}{\partial W} = \left(\frac{\partial \rho}{\partial H} \odot \sigma'(Z^{(t)})\right){X^{(t)}}^T$
  - d. Given learning rate $\eta_t$, update parameters as follows:
    - $b_2^{(t+1)}) \leftarrow b_2^{(t)}) + \eta_t \frac{\partial \rho}{\partial b_2}$
    - $C^{(t+1)}) \leftarrow C^{(t)}) + \eta_t \frac{\partial \rho}{\partial C}$
    - $b_1^{(t+1)}) \leftarrow b_1^{(t)}) + \eta_t \frac{\partial \rho}{\partial b_1}$
    - $W^{(t+1)}) \leftarrow W^{(t)}) + \eta_t \frac{\partial \rho}{\partial W}$
- Step3: Repeat Step2 until some convergence criteria is met.

## Numerical Experiment

```
nn = MnistModel(x_train, y_train, x_test,
y_test, hidden_units=100, learning_rate=0.01, num_epochs=20, seed=1234)

# training sample size: [(60000, 784)]
# test sample size:[(10000, 784)]
# hidden units number: [100]
# batch_size:[1]


nn.train()

training sample size: [(60000, 784)]
test sample size:[(10000, 784)]
hidden units number: [100]
batch_size:[1]
epoch:1 | Training Accuracy:[0.9299666666666667]
epoch:2 | Training Accuracy:[0.9691]
epoch:3 | Training Accuracy:[0.9773833333333334]
epoch:4 | Training Accuracy:[0.9828166666666667]
epoch:5 | Training Accuracy:[0.9845]
epoch:6 | Training Accuracy:[0.9875166666666667]
epoch:7 | Training Accuracy:[0.9939333333333333]
epoch:8 | Training Accuracy:[0.99495]
epoch:9 | Training Accuracy:[0.9957]
epoch:10 | Training Accuracy:[0.9963166666666666]
epoch:11 | Training Accuracy:[0.9969166666666667]
epoch:12 | Training Accuracy:[0.9970833333333333]
epoch:13 | Training Accuracy:[0.9968333333333333]
epoch:14 | Training Accuracy:[0.9972833333333333]
epoch:15 | Training Accuracy:[0.9967833333333334]
epoch:16 | Training Accuracy:[0.9971666666666666]
epoch:17 | Training Accuracy:[0.9975333333333334]
```

```
epoch:18 | Training Accuracy:[0.9972]
epoch:19 | Training Accuracy:[0.9976333333333334]
epoch:20 | Training Accuracy:[0.9976]
```

```
print("Test Accuracy: [{}]".format(nn.test()))

# Test Accuracy: [0.982]
```

```
training sample size: [(60000, 784)]
test sample size:[(10000, 784)]
hidden units number: [100]
batch_size:[1]
epoch:1 | Training Accuracy:[0.9299666666666667]
epoch:2 | Training Accuracy:[0.9691]
epoch:3 | Training Accuracy:[0.9773833333333334]
epoch:4 | Training Accuracy:[0.9828166666666667]
epoch:5 | Training Accuracy:[0.9845]
epoch:6 | Training Accuracy:[0.9875166666666667]
epoch:7 | Training Accuracy:[0.9939333333333333]
epoch:8 | Training Accuracy:[0.99495]
epoch:9 | Training Accuracy:[0.9957]
epoch:10 | Training Accuracy:[0.9963166666666666]
epoch:11 | Training Accuracy:[0.9969166666666667]
epoch:12 | Training Accuracy:[0.9970833333333333]
epoch:13 | Training Accuracy:[0.9968333333333333]
epoch:14 | Training Accuracy:[0.9972833333333333]
epoch:15 | Training Accuracy:[0.9967833333333334]
epoch:16 | Training Accuracy:[0.9971666666666666]
epoch:17 | Training Accuracy:[0.9975333333333334]
epoch:18 | Training Accuracy:[0.9972]
epoch:19 | Training Accuracy:[0.9976333333333334]
epoch:20 | Training Accuracy:[0.9976]
```

```
print("Test Accuracy: [{}]".format(nn.test()))
```

```
Test Accuracy: [0.982]
```