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

- c. Compute backward-propagation:
  - $\frac{\partial \rho}{\partial b_2} = \frac{1}{M}\sum_{m=1}^{M} e_{y^{(t,m)}} - S^{(t,m)}$
  - $\frac{\partial \rho}{\partial C} = \frac{1}{M}\sum_{m=1}^{M} (e_{y^{(t,m)}} - S^{(t,m)}) {H^{(t,m)}}^T$
  - $\frac{\partial \rho}{\partial H} = \frac{1}{M}\sum_{m=1}^{M} C^T (e_{y^{(t,m)}} - S^{(t,m)})$
  - $\frac{\partial \rho}{\partial b_1} = \frac{1}{M}\sum_{m=1}^{M} \frac{\partial \rho}{\partial H} \odot \sigma'(Z^{(t,m)})$
  - $\frac{\partial \rho}{\partial W} = \frac{1}{M}\sum_{m=1}^{M} \left(\frac{\partial \rho}{\partial H} \odot \sigma'(Z^{(t,m)})\right) {X^{(t,m)}}^T$
- d. Given learning rate η_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.

To avoid unnecessary `for-loop` we can vectoruize the above algorithm.

- Step1: Specify batch_size M, activation function $\sigma(z)$, and initialize $W^{(0)}, b_1^{(0)}, C^{(0)}, b_2^{(0)}$;
- Step2: At iteration t:
  - a. Select M data samples $\{X^{(t,m)}, y^{(t,m)}\}_{m=1}^{M}$ uniform at random from the full dataset $\{X^{(n)}, y^{(n)}\}_{n=1}^{N}$
  - b. Compute forward-propagation:
    - $Z^{(t)} = W^{(t)}X^{(t)} + b_1^{(t)}$, where $X^{(t)} = (X^{(t,1)}, ..., X^{(t,M)})$ and the summation on b_1 will be column-wise.
    - $H^{(t)} = \sigma(Z^{(t)})$, where $H^{(t)} = (H^{(t,1)}, ..., H^{(t,M)})$ and $\sigma(.)$ is element wise operation.
    - $U^{(t)} = C^{(t)}H^{(t)} + b_2^{(t)}$
    - $S^{(t)} = F_{softmax}(U^{(t)})$, where the $F_{softmax}$ is column-wise operation.
  - c. Compute backward-propagation:
    - $\frac{\partial \rho}{\partial b_2} = \text{np.mean}(e_{y^{(t)}} - S^{(t)}, \text{axis=1})$
    - $\frac{\partial \rho}{\partial C} = \frac{1}{M}(e_{y^{(t)}} - S^{(t)}){H^{(t)}}^T$
    - $\frac{\partial \rho}{\partial H} = \text{np.mean}(C^T(e_{y^{(t)}} - S^{(t)}), \text{axis=1})$
    - $\frac{\partial \rho}{\partial b_1} = \text{np.mean}(\frac{\partial \rho}{\partial H} \odot \sigma'(Z^{(t)}), \text{axis=1})$
    - $\frac{\partial \rho}{\partial W} = \frac{1}{M}((\frac{\partial \rho}{\partial H} \odot \sigma'(Z^{(t)})){X^{(t)}}^T)$
  - d. Given learning rate η_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

```
n n = MnistModel(x_train, y_train, x_test,
y_test, hidden_units=100, batch_size=1, learning_rate=0.01, num_epochs=5, seed=1234)

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

```
start = time.time()
nn.train()
end = time.time()

# epoch:1 | Training Accuracy:[0.9296]
# epoch:2 | Training Accuracy:[0.96985]
# epoch:3 | Training Accuracy:[0.9783333333333334]
# epoch:4 | Training Accuracy:[0.9819166666666667]
# epoch:5 | Training Accuracy:[0.98605]
```

```
print("Running Time: [{}] second".format(end - start))

# Running Time: [672.2338092327118] second
```

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

# Test Accuracy: [0.975]
```