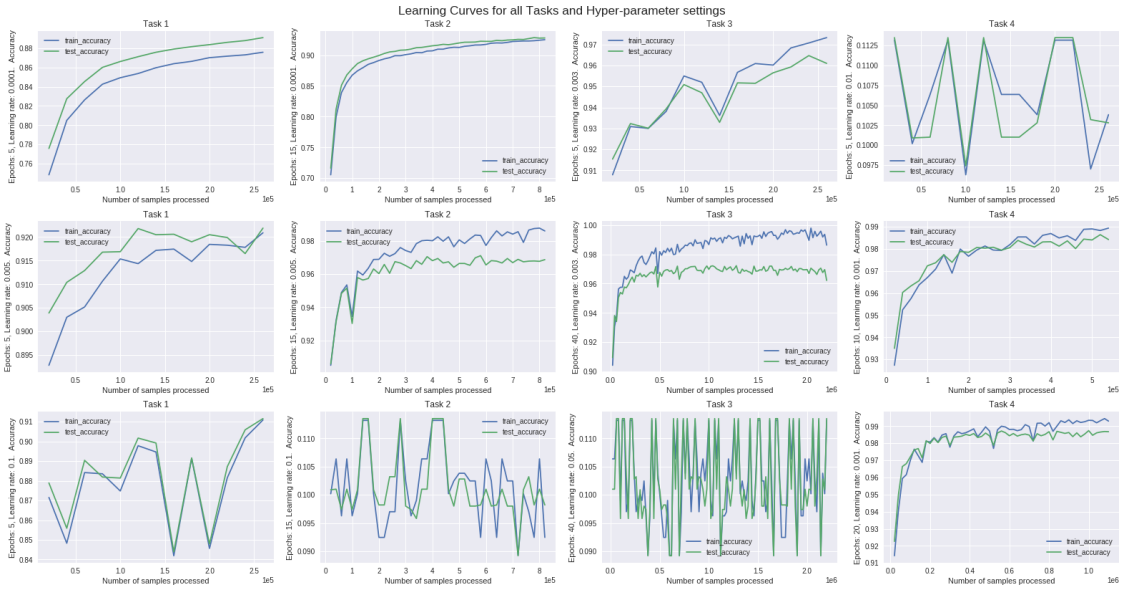
17044633_DL_hw2

February 11, 2018

By: Yuan Zhang
SN: 17044633

1 Results

```
In [34]: plot_learning_curves([experiments_task1, experiments_task2, experiments_task3, experi
```




```
In [35]: plot_summary_table([experiments_task1, experiments_task2, experiments_task3, experimen
```

	Setting 1	Setting 2	Setting 3
Model 1	0.8909	0.9219	0.9116
Model 2	0.9279	0.9687	0.0982
Model 3	0.961	0.9622	0.1135
Model 4	0.1028	0.9842	0.9868

2 Questions

2.0.1 Q1 (32 pts): Compute the following derivatives

Show all intermediate steps in the derivation (in markdown below). Provide the final results in vector/matrix/tensor form whenever appropiate.

1. [5 pts] Give the cross-entropy loss above, compute the derivative of the loss function with respect to the scores z (the input to the softmax layer).

$$\frac{\partial loss}{\partial z} = ?$$

2. [12 pts] Consider the first model (M1: linear + softmax). Compute the derivative of the loss with respect to

- the input x

$$\frac{\partial loss}{\partial x} = ?$$

- the parameters of the linear layer: weights W and bias b

$$\frac{\partial loss}{\partial W} = ?$$

$$\frac{\partial loss}{\partial b} = ?$$

3. [10 pts] Compute the derivative of a convolution layer wrt. to its parameters W and wrt. to its input (4-dim tensor). Assume a filter of size H x W x D, and stride 1.

$$\frac{\partial loss}{\partial W} = ?$$

2.0.2 A1: (Your answer here)

1. [5 pts] Give the cross-entropy loss above, compute the derivative of the loss function with respect to the scores z (the input to the softmax layer).

$$\frac{\partial loss}{\partial z_i} = -y_i + \frac{exp(z_i)}{\sum_{c=1}^{10} \exp(z_i[c])}$$

z_i: the score of i th sample. y_i: the true label of i th sample, a vector having 1 on true class's position.

2. [12 pts] Consider the first model (M1: linear + softmax). Compute the derivative of the loss with respect to

- the input x

$$\frac{\partial loss}{\partial x_i} = \frac{\partial loss}{\partial z_i} * W^T$$

x_i: the input of i th sample.

- the parameters of the linear layer: weights W and bias b

$$\frac{\partial loss}{\partial W} = \sum_{i=1}^{S} x_i^T * \frac{\partial loss}{\partial z_i}$$

$$\frac{\partial loss}{\partial b} = \sum_{i=1}^{S} \frac{\partial loss}{\partial z_i}$$

3. [10 pts] Compute the derivative of a convolution layer wrt. to its parameters W and wrt. to its input (4-dim tensor). Assume a filter of size H x W x D, and stride 1.

$$\frac{\partial loss}{\partial W_{(h,w,d)}} = \sum_{i=1}^{S} \sum_{m=1}^{M} \sum_{n=1}^{N} \frac{\partial loss}{\partial y_i[d,m,n]} * Pad.x_i[m+h-1, n+w-1]$$

$$\frac{\partial loss}{\partial b_d} = \sum_{i=1}^{S} \sum_{m=1}^{M} \sum_{n=1}^{N} \frac{\partial loss}{\partial y_i[d,m,n]}$$

$$\frac{\partial loss}{\partial Pad.x_i[m,n]} = \sum_{d=1}^{D} \sum_{h=max(1,m+1-M)}^{min(H,m)} \sum_{w=max(1,n+1-N)}^{min(W,n)} \frac{\partial loss}{\partial y_i[d,m-h+1,n-w+1]} * W_{(h,w,d)}$$

$W_{(h,w,d)}$: the (h,w,d) th entry of matrix W. $y_i[h,m,n]$: the (d,m,n) th entry of i th output. $Pad.x_i[m+h-1, n+w-1]$: the (m+h-1,n+w-1) th entry of i th padded input with size (M+2)*(N+2).

2.0.3 Q2 (8 pts): How do the results compare to the ones you got when implementing these models in TensorFlow?

1. [4 pts] For each of the models, please comment on any differences or discrepancies in results -- runtime, performance and stability in training and final performance. (This would be the place to justify design decisions in the implementation and their effects).
2. [2 pts] Which of the models show under-fitting?
3. [2 pts] Which of the models show over-fitting?

2.0.4 A2: (Your answer here)

1. Differences

Compared with tensorflow, the performances are very similar. They have similar trend on evevy plot. And the performances are even better than tensorflow(from the table). The trainning process is slower especially for the CNN. This is because when implementing forward_pass and backward_pass of convolutional layer, I use cycle for the kernel size and output size. I think the cycle of output size can be removed and cycle for the kernel size and be done at the same time, which is very fast for GPU.

2. Underfit

Task 2 Setting 3 & Task 3 Setting 3 & Task 4 Setting 1: learning rate is too high
Task 1 Setting 1-3: the model is too simple(linear)
Task 2 Setting 1: not enough training

3. Overfit

Task 2 Setting 2: training for so long & learning rate is high
Task 3 Setting 2: training for so long
Task 4 Setting 3: training for so long & model is complex