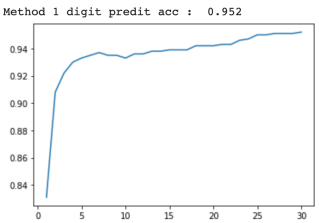
Deep Learning Mini-projects

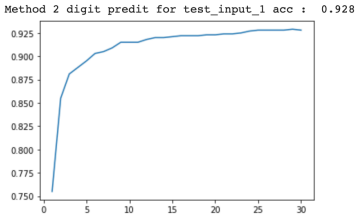
[Lai,Ying-Ru & Chen, Hong-Guai]

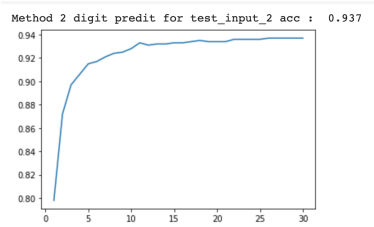
Project 1 – Classification, weight sharing, auxiliary losses

[Method 1] Compare the accuracy between two inputs trained on the same model with weight sharing and of the use of an auxiliary loss, and two other models trained separately without weight sharing and of the use of an auxiliary loss.

- Accuracy of digit prediction
  - Trained on the same model with weight sharing and of the use of an auxiliary loss : **95.2%** [Better]
  - Trained on two separate models: **92.8% & 93.7%**







- Accuracy of target and predicted output (i.e. for each pair, if the first digit is lesser or equal to the second) :
  - Trained on the same model with weight sharing and of the use of an auxiliary loss : **96.5%**[Better]
  - Trained on two separate models: **95.3%**

```
# predicts for each pair if the first digit is lesser or equal to the second
acc1, output_class_1 = compute_nb_errors(model, test_input_1, test_classes_1, mini_batch_size)
acc2, output_class_2 = compute_nb_errors(model, test_input_2, test_classes_2, mini_batch_size)
target_output = predict_target(output_class_1, output_class_2)
#print(target_output.size())
target_acc = compute_num_errors(target_output, test_target)
print('Method 1 acc : ', target_acc)
```

```
Method 1 acc :  0.965
```

```
# predicts for each pair if the first digit is lesser or equal to the second
acc1, output_class_2_1 = compute_nb_errors(model1, test_input_1, test_classes_1, mini_batch_size)
acc2, output_class_2_2 = compute_nb_errors(model2, test_input_2, test_classes_2, mini_batch_size)
target_output2 = predict_target(output_class_2_1, output_class_2_2)
#print(target_output.size())
target_acc = compute_num_errors(target_output2, test_target)
print('Method 2 acc : ', target_acc)
```

```
Method 2 acc :  0.953
```

[Method 2] Compare the accuracy and parameters usage between training on MLP and CNN(which with weight sharing and of the use of an auxiliary loss)

- **Accuracy of digit prediction**
  - MLP: 90.4%

- CNN : 96.5% [Better]

- **Total parameters usage**
  - MLP: 414.418
  - CNN : 341,026 [Better]

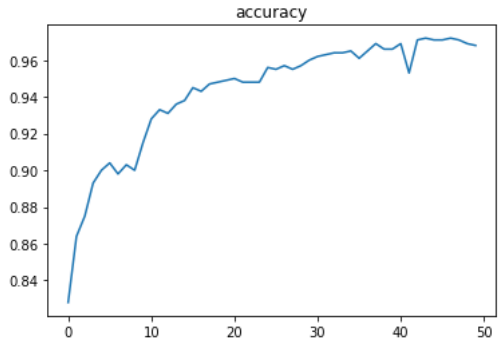
```
MLP digit predit acc :  0.904
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1                  [-1, 500]          98,500
            Linear-2                  [-1, 500]         250,500
            Linear-3                  [-1, 128]          64,128
            Linear-4                   [-1, 10]           1,290
================================================================
Total params: 414,418
Trainable params: 414,418
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.01
Params size (MB): 1.58
Estimated Total Size (MB): 1.59
----------------------------------------------------------------
```

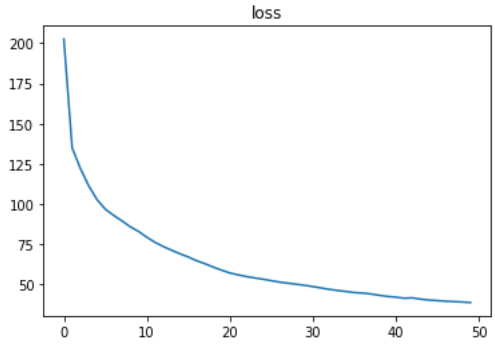
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 32, 12, 12]             320
            Conv2d-2          [-1, 64, 10, 10]          18,496
            Linear-3                 [-1, 200]         320,200
            Linear-4                  [-1, 10]           2,010
================================================================
Total params: 341,026
Trainable params: 341,026
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.09
Params size (MB): 1.30
Estimated Total Size (MB): 1.39
----------------------------------------------------------------
```

Project 2 – Mini deep-learning framework

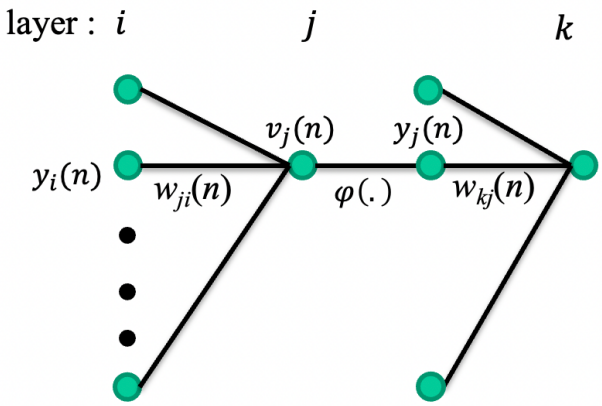
The model we build include several features:

- Flexible neurons and layers.
- The activation function for hidden layers is Relu and Sigmoid for output layer.
- The accuracy up to 90% after three epochs and 97% after fifty epochs.





Formula derivation as below :



$$v_j(n) = \sum_{i=0}^{P} w_{ji}(n) y_i(n)$$

$$y_j(n) = \varphi_j(v_j(n)) \quad , \varphi(.) \text{ is activation function}$$

If we want to modify $\Delta\omega_{ji}$, we need to find $\partial E(n)/\partial w_{ji}(n)$ cause they are proportional to each other. By chain rule, we know gradient could be represent as:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial v_j(n)} \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

$$e_j(n) = d_j(n) - y_j(n)$$

d_j *is expected output*

According to $v_j(n)$ formula,

$$\frac{\partial v_j(n)}{\partial w_{ji}(n)} = \frac{\partial}{\partial w_{ji}(n)} \left[\sum_{i=0}^{p} w_{ji}(n) y_i(n) \right] = y_i(n)$$

And we define

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)}$$

So $\Delta\omega_{ji}$ could be rewritten as

$$\Delta w_{ji}(n) = \eta\delta_j(n) \cdot y_i(n)$$

η is learning rate

When weights need to be modified, we could follow this formula

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n) = w_{ji}(n) + \eta\delta_j(n)y_i(n)$$

1. When *j* neuron is output neuron :

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)}$$

$$= -\frac{\partial}{\partial y_j(n)}\left[\frac{1}{2}\sum_{j\in c}\left(d_j(n) - y_j(n)\right)^2\right] \cdot \frac{\partial\left(\varphi\left(v_j(n)\right)\right)}{\partial v_j(n)}$$

$$= \left(d_j(n) - y_j(n)\right)\varphi'\left(v_j(n)\right)$$

2. When *j* neuron is hidden neuron :

Due to lack of expectations, we need former neurons to get the output in a recursive manner.

$$\delta_j(n) = -\frac{\partial E(n)}{\partial y_j(n)}\frac{\partial y_j(n)}{\partial v_j(n)} = -\frac{\partial E(n)}{\partial y_j(n)}\varphi'\left(v_j(n)\right)$$

By chain rule, derive as follows:

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k \frac{\partial E(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} = \sum_k \frac{\partial E(n)}{\partial v_k(n)} w_{kj}(n)$$

If k is output layer, we know

$$\frac{\partial E(n)}{\partial v_k(n)} = -\delta_k(n) = -\left(d_k(n) - y_k(n)\right)\varphi'\left(v_k(n)\right)$$

So,

$$\delta_j(n) = \varphi'_j\left(v_j(n)\right)\sum_k \delta_k(n) w_{kj}(n)$$

$$= \varphi'_j\left(v_j(n)\right)\sum_k \left(d_k(n) - y_k(n)\right)\varphi'\left(v_k(n)\right) w_{kj}(n)$$

In our model, we use Relu as activation function for hidden layer and sigmoid for output layer.

1. For output layer (sigmoid) :

$$y_j(n) = \varphi\left(v_j(n)\right) = \frac{1}{1+\exp\left(-v_j(n)\right)}, -\infty < v_j(n) < \infty$$

$$\frac{\partial y_j(n)}{\partial v_j(n)} = \varphi'\left(v_j(n)\right) = \frac{\exp\left(-v_j(n)\right)}{\left[1+\exp\left(-v_j(n)\right)\right]^2}$$

$$\varphi'\left(v_j(n)\right) = y_j(n)[1 - y_j(n)]$$

Finally,

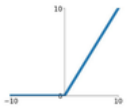
$$\delta_j(n) = \varphi'\left(v_j(n)\right)\sum_k \delta_k(n) w_{kj}(n) \quad = y_j(n)\left(1 - y_j(n)\right)\sum_k \delta_k(n) w_{kj}(n)$$

2. For hidden layer (Relu) :

$$y_j(n) = \varphi\left(v_j(n)\right) = \max\left\{0, v_j(n)\right\} = \begin{cases} v_j(n) & if\ v_j(n) \geq 0 \\ 0 & f\ v_j(n) < 0 \end{cases}$$

ReLU

$\max(0, x)$



$$\delta_j(n) = \varphi'\left(v_j(n)\right)\sum_k \delta_k(n) w_{kj}(n) = \begin{cases} \sum_k \delta_k(n) w_{kj}(n) & if\ v_j(n) \geq 0 \\ 0 & if\ v_j(n) < 0 \end{cases}$$