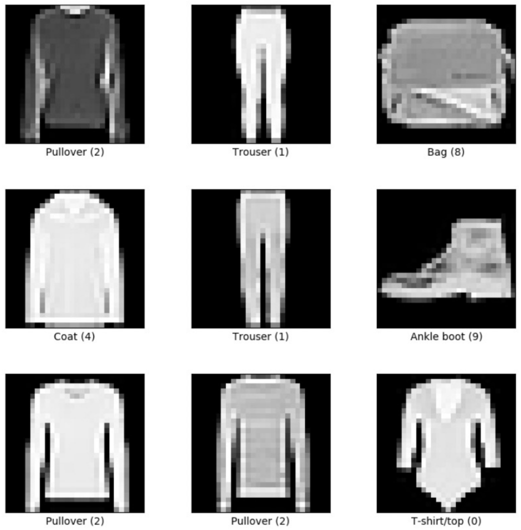
HW1 Instructions

Deep Learning

Tasks

- Design a **MLP** to label images from Fashion-MNIST
- Implement your **activation function** (Part 4)
- Implement your **MLP model** (Part 4)
- Implement **multi-class cross entropy loss** (Part 5)
- Implement mini-batch SGD with **momentum and weight decay** (Part 6)

Dataset: Fashion-MNIST



- Fashion-MNIST is a dataset of Zalando's article images
- Training set: 60,000 examples
- Test set: 10,000 examples
- Each example is a 28x28 grayscale image, associated with a label from 10 classes.

Source: https://www.tensorflow.org/datasets/catalog/fashion_mnist

Grading

1. MLP 15% (Part 4)
2. Activation function 15% (Part 4)
3. Loss function 15% (Part 5)
4. Optimizer 15% (Part 6)

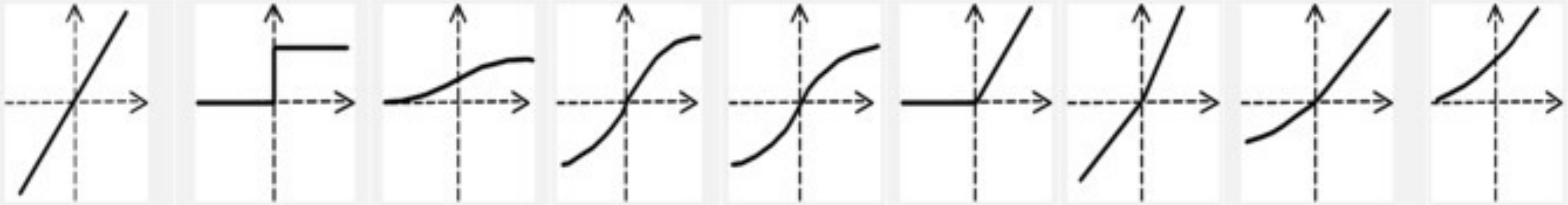
Grading

5. Model size 15%:
    - 10%: If your model (the number of parameters) is smaller than **2MB**, you will get 10%.
    - 5%: The remaining 5% will depend on your ranking within the class.
6. Model accuracy 15%:
    - 10%: If your accuracy is higher than **85%**, you will get 10%.
    - 5%: The remaining 5% will depend on your ranking within the class.
7. Model accuracy on another dataset 10%: it will depend on your ranking within the class.

Rules

- You can use Google Colab for implementation
- Please do NOT call any other existing libraries for implementations.
- You can use tensor functions in torch.
- Only use torch.nn.Linear() for your MLP design.
- Don’t directly call torch.nn functions for other implementations.
- Please do NOT attempt to modify the sections **DO NOT MODIFY**.

Activation Function (Part 4)

Activation Function									
	Identity	Binary Step	Logistic	TanH	ArcTan	ReLU	PreLU	ELU	SoftPlus
PLOT									
EQUATION	$f(x) = x$	$f(x) = \begin{cases} 0 \; for \; x<0 \\ 1 \; for \; x\geq 0 \end{cases}$	$f(x) = \frac{1}{1+e^{-x}}$	$f(x) = tangh(x) = \frac{2}{1+e^{-2x}} - 1$	$f(x) = tan^{-1}(x)$	$f(x) = \begin{cases} 0 \; for \; x<0 \\ x \; for \; x\geq 0 \end{cases}$	$f(x) = \begin{cases} \propto x \; for \; x<0 \\ x \; for \; x\geq 0 \end{cases}$	$f(x) = \begin{cases} \propto (e^{-x}-1) \; for \; x<0 \\ x \; for \; x\geq 0 \end{cases}$	$f(x) = log_e (1+e^x)$
DERIVATE	$f'(x) = 1$	$f'(x) = \begin{cases} 0 \; for \; x\neq 0 \\ ? \; for \; x=0 \end{cases}$	$f'(x) = f(x)(1-f(x))$	$f'(x) = 1-f(x)^2$	$f'(x) = \frac{1}{x^2+1}$	$f'(x) = \begin{cases} 0 \; for \; x<0 \\ 1 \; for \; x\geq 0 \end{cases}$	$f'(x) = \begin{cases} \propto \; for \; x<0 \\ 1 \; for \; x\geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x)+\propto \; for \; x<0 \\ 1 \; for \; x\geq 0 \end{cases}$	$f'(x) = \frac{1}{1+e^{-x}}$

Activation Function (Part 4)

```
class myActivation(nn.Module):

    def __init__(self):

        super().__init__()

    def forward(self, x):

        # example: identity
        out = x

        return out
```

MLP (Part 4)

```
class myMLP(nn.Module):

    def __init__(self, input_dim, num_classes):

        super(myMLP, self).__init__()

        # example: 2 hidden layers, 1 output layer MLP
        self.mlp = nn.Sequential(
            nn.Linear(input_dim, 512),
            myActivation(),
            nn.Linear(512, 512),
            myActivation(),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):

        out = self.mlp(x)

        return out

model = myMLP(input_dim, num_classes).to(device)
```

Don’t use CNN or other pretrained models

Cross entropy loss (Part 5)

$$\hat{\phi} = \underset{\phi}{\mathrm{argmin}}\left[L[\phi]\right] = \underset{\phi}{\mathrm{argmin}}\left[-\sum_{i=1}^{I}\log\left[Pr(\mathbf{y}_i|\mathbf{f}[\mathbf{x}_i,\phi])\right]\right]. \qquad (5.7)$$

$$L[\phi] = -\sum_{i=1}^{I}\log\left[\mathrm{softmax}_{y_i}\left[\mathbf{f}\left[\mathbf{x}_i,\phi\right]\right]\right]$$

$$= -\sum_{i=1}^{I} \mathrm{f}_{y_i}\left[\mathbf{x}_i,\phi\right] - \log\left[\sum_{k=1}^{K}\exp\left[\ \mathrm{f}_k\left[\mathbf{x}_i,\phi\right]\right]\right]$$

$$\boxed{\mathrm{softmax}_k[\mathbf{z}] = \frac{\exp[z_k]}{\sum_{k'=1}^{K}\exp[z_{k'}]}}$$

Multiclass cross-entropy loss

```
class myLoss(nn.Module):

    def __init__(self):

        super(myLoss, self).__init__()

        self.softmax = nn.Softmax(dim=1)

    def forward(self, outputs, targets):

        # Transform targets to one-hot vector
        # ex.
        # 0 => [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
        # 3 => [0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
        targets_onehot = torch.zeros_like(outputs)
        targets_onehot.zero_()
        targets_onehot.scatter_(1, targets.unsqueeze(-1), 1)

        # example: mean square error loss
        loss = (targets_onehot.float() - outputs) ** 2

        return torch.mean(loss)

criterion = myLoss()
```

Optimizer (Part 6)

$$v_t \leftarrow \beta v_{t-1} - \gamma \eta w_{t-1} - \eta g_{t,t-1}$$
$$w_t \leftarrow w_{t-1} + v_t$$

```
for group in self.param_groups:

    for p in group['params']:

        if p.grad is None:
            continue

        # p.data: weight
        # p.grad.data: gradient

        # example: mini-batch SGD
        # new_weight = old_weight + ( -lr * gradient)
        p.data.add_(p.grad.data, alpha=-group['lr'])

        # hint: momentum need to store additional state, for example 'm_state'.
        # following code show how to add new state into the optimizer and how to get stored state

        # # add new state
        # param_state = self.state[p]
        # if 'm_state' not in param_state:
        #         param_state['m_state'] = xxx

        # # get stored state
        # xxx = param_state['m_state']

return loss
```

g: gradient

w: weights

β: momentum coefficient

γ: weight decay coefficient

η: learning rate

Submission

- Upload your zip file to NTU Cool
- StudentID_HW1.zip
  - DL_HW1_StudentID.ipynb
  - StudentID_submission.pt
  - StudentID_submission.csv
- **Deadline: 3/25 23:59**

TA Hour

- Time: By appointment
- Location: 教研館 317
- If you have any question, please send email to TA first.
- TA’s email: r12725019@ntu.edu.tw