Mathematical Foundations of Deep Neural Networks, M1407.001200
E. Ryu
Spring 2024

Homework 5
Due 5pm, Monday, April 8, 2024

Problem 1: *Implementing backprop for MLP.* Consider the multi-layer perceptron

$$\begin{aligned} y_L &= A_L y_{L-1} + b_L \\ y_{L-1} &= \sigma(A_{L-1} y_{L-2} + b_{L-1}) \\ &\vdots \\ y_2 &= \sigma(A_2 y_1 + b_2) \\ y_1 &= \sigma(A_1 x + b_1), \end{aligned}$$

where $x \in \mathbb{R}^{n_0}$, $A_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$, $b_\ell \in \mathbb{R}^{n_\ell}$, and $n_L = 1$. Let $\sigma(z) = (1+e^{-z})^{-1}$ be the sigmoid activation function. Let $f_\theta(x) = y_L$ and consider the loss function

$$\ell(\theta) = \frac{1}{2}(f_\theta(X_{\text{data}}) - Y_{\text{data}})^2.$$

Download the starter code `mlp_backprop.py` and implement backprop using the gradient computation of homework 4 problem 6. Your code should roughly be of the form:

```
# forward pass
y_list = [X_data]
y = X_data
for ell in range(L):
  S = sigma if ell<L-1 else lambda x: x
  y = S(A_list[ell]@y+b_list[ell])
  y_list.append(y_next)

# backward pass
dA_list = []
db_list = []
dy = y-Y_data
for ell in reversed(range(L)):
  S = sigma_prime if ell<L-1 else lambda x: torch.ones(x.shape)
  A, b, y= A_list[ell], b_list[ell], y_list[ell]
  db = ...    # dloss/db
  dA = ...    # dloss/dA
  dy = ...    # dloss/dy
  dA_list.insert(0,dA)
  db_list.insert(0,db)
```

The starter code provides code that performs gradient computation using autograd. Compare your results against the autograd results.

Solution. We provide two possible solutions.

Solution #1:

```
db = dy@torch.diag(S(A@y+b).squeeze(1))
dA = db.T@y.T
dy = db@A
```

Solution #2:

```
db = dy*S(A@y+b).T
dA = db.T@y.T
dy = db@A
```

■

Problem 2: *Vanishing gradients.* When training very deep neural networks, one often encounters the problem of vanishing gradients. Consider the MLP of homework 4 problem 6. Assume the activation function σ is the sigmoid activation function. Define $\tilde{y}_i = A_i y_{i-1} + b_i$ for $i = 1, \ldots, L$. So $y_L = \tilde{y}_L$ and $y_i = \sigma(\tilde{y}_i)$ for $i = 1, \ldots, L-1$.

Assume the matrices $A_1, \ldots, A_L$ are all not too large. If A_j is small for some $j \in \{\ell+1, \ldots, L\}$, then

$$\frac{\partial y_L}{\partial b_i}, \quad \frac{\partial y_L}{\partial A_i}$$

for $i = 1, \ldots, \ell$ become small. If $\tilde{y}_j$ has large absolute value for some $j \in \{\ell+1, \ldots, L-1\}$, then

$$\frac{\partial y_L}{\partial b_i}, \quad \frac{\partial y_L}{\partial A_i}$$

for $i = 1, \ldots, \ell$ become small. Explain why this is the case.

Clarification. For the purpose of this problem, let's say that a vector or a matrix is "small" if all of its entries are small. Define "large" analogously. Also, (not too large)×(small)=(small).

Remark. Neural networks built with ReLU tend to suffer less from vanishing gradients compared to networks built with sigmoid or tanh. This is because $\sigma'(z) \to 0$ as $z \to \pm\infty$ for sigmoid and tanh, while $\sigma'(z) \to 0$ only as $z \to -\infty$ for ReLU.

Solution.

(i) We know that

$$\frac{\partial y_L}{\partial b_i} = \frac{\partial y_L}{\partial y_{L-1}} \cdots \frac{\partial y_{i+1}}{\partial y_i} \frac{\partial y_i}{\partial b_i}, \qquad \frac{\partial y_L}{\partial y_{L-1}} = A_L,$$
$$\frac{\partial y_i}{\partial b_i} = \operatorname{diag}\left(\sigma'(A_i y_{i-1} + b_i)\right), \qquad \text{for } i = 1, \ldots, L-1$$
$$\frac{\partial y_i}{\partial y_{i-1}} = \operatorname{diag}\left(\sigma'(A_i y_{i-1} + b_i)\right) A_i, \qquad \text{for } i = 2, \ldots, L-1,$$

Notice that $\sigma'(x) \leq 1/4$ for any x. So all the matrices terms above are not too large with only one of the matrices are small, so their product is small.
Next,

$$\frac{\partial y_L}{\partial y_i} = \frac{\partial y_L}{\partial y_{L-1}} \cdots \frac{\partial y_{i+1}}{\partial y_i},$$
$$\frac{\partial y_L}{\partial A_i} = \operatorname{diag}\left(\sigma'(A_i y_{i-1} + b_i)\right) \left(\frac{\partial y_L}{\partial y_i}\right)^\intercal y_{i-1}^\intercal, \qquad \text{for } i = 1 \ldots, L-1.$$

So by the same reasoning, $\frac{\partial y_L}{\partial A_i}$ is small.

(ii) We can rewrite the above formula as follows.

$$\frac{\partial y_i}{\partial b_i} = \operatorname{diag}\left(\sigma'(\tilde{y}_i)\right), \qquad \text{for } i = \ell+1, \ldots, L-2,$$
$$\frac{\partial y_i}{\partial y_{i-1}} = \operatorname{diag}\left(\sigma'(\tilde{y}_i)\right) A_i, \qquad \text{for } i = \ell+1, \ldots, L-2,$$

Notice that $\sigma'(x) = 0$ for any x of large absolute value.
In this case, the above gradients vanish to 0, so the resulting gradient terms

$$\frac{\partial y_L}{\partial b_i}, \qquad \frac{\partial y_L}{\partial A_i}$$

for $i = 1, \ldots, \ell$ become 0. ■

Problem 3: *Two forms of momentum SGD.* There are two forms for the (non-Nesterov) momentum SGD. Form I is

$$\theta^{k+1} = \theta^k - \alpha g^k + \beta(\theta^k - \theta^{k-1})$$

for $k = 0, 1, \ldots$, where $\theta^{-1} = \theta^0$. This form is more commonly invoked in mathematical discussions as it makes the "momentum term" $\beta(\theta^k - \theta^{k-1})$ clearly visible. Form II is

$$\begin{aligned} v^{k+1} &= g^k + \beta v^k \\ \theta^{k+1} &= \theta^k - \alpha v^{k+1} \end{aligned}$$

for $k = 0, 1, \ldots$, where $v^0 = 0$. This is the form implemented in PyTorch with the option `Nesterov=false`. Show that the two forms are equivalent in the sense that given a starting point $\theta^0 \in \mathbb{R}^n$ and a sequence of stochastic gradients $g^0, g^1, \ldots \in \mathbb{R}^n$, Forms I and II produce the same $\theta^1, \theta^2, \ldots$ sequence.

Hint. Use induction.

Solution. To avoid confusion, let us label the iterates of Form I as θ^k and Form II as ϕ^k. We use induction to show $\theta^k = \phi^k$ for $k = 0, 1, \ldots$, when $\theta^0 = \phi^0$.

When $k = 1$,

$$\theta^1 = \theta^0 - \alpha g^0 + \beta(\theta^0 - \theta^{-1}) = \theta^0 - \alpha g^0$$

since $\theta^{-1} = \theta^0$, and

$$\phi^1 = \phi^0 - \alpha v^1 = \phi^0 - \alpha g^0$$

so $\theta^1 = \phi^1$.

Next, suppose $\theta^n = \phi^n$ and $\theta^{n-1} = \phi^{n-1}$. Since we can write Form II as

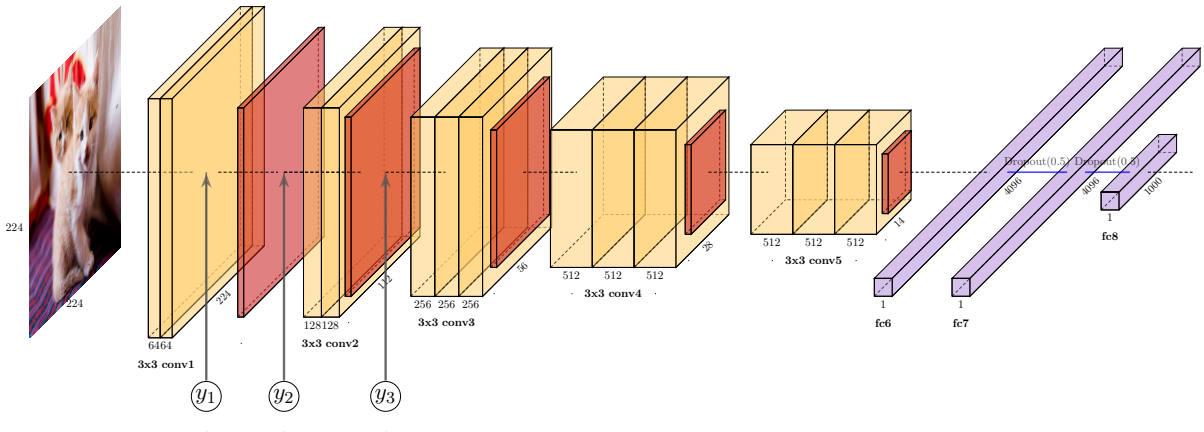
$$\begin{aligned} v^{n+1} &= g^n + \beta v^n = g^n + \frac{\beta}{\alpha}(\phi^{n-1} - \phi^n) \\ \phi^{n+1} &= \phi^n - \alpha v^{n+1} = \phi^n - \alpha g^n + \beta(\phi^n - \phi^{n-1}), \end{aligned}$$

we have $\theta^{n+1} = \phi^{n+1}$. By mathematical induction, we conclude $\theta^k = \phi^k$ for $k = 0, 1, \ldots$. ■

Problem 4: The *receptive field* of a neuron is the set of input pixels the neuron value depends on. Consider the VGG16 network, and consider the intermediate values y_1, y_2, and y_3, which are respectively the outputs of the second convolutional layer, the first maxpool layer, and the second maxpool layer. Assume the input X has a batch size 1 and otherwise has dimensions $3 \times 224 \times 224$. For $p = 1, 2, 3$, describe the receptive field of $y_p[k, i, j]$, i.e., which values of $X[c, m, n]$ does $y_p[k, i, j]$ depend on? Here, m, n and i, j denote the spatial dimensions and c and k denote the channels.

```
class VGG16(nn.Module) :
  def __init__(self) :
    super(VGG16, self).__init__()
    self.conv_layer1 = nn.Sequential(
        nn.Conv2d(3,64,kernel_size=3,padding=1),       #64x224x224
        nn.ReLU(),
        nn.Conv2d(64,64 kernel_size=3,padding=1),      #64x224x224
        nn.ReLU() )
    self.pool1 = nn.MaxPool2d(kernel_size=2,stride=2) #64x112x112
    self.conv_layer2 = nn.Sequential(
        nn.Conv2d(64,128,kernel_size=3,padding=1),    #128x112x112
        nn.ReLU(),
        nn.Conv2d(128,128,kernel_size=3,padding=1),   #128x112x112
        nn.ReLU()  )
    self.pool2 = nn.MaxPool2d(kernel_size=2,stride=2)  #128x56x56
    ...

  def forward(self, x) :
    y1 = self.conv_layer1(x)
    y2 = self.pool1(y1)
    y3 = self.pool2(self.conv_layer2(y2))
    ...
```



Solution. Denote $\mathcal{R}(y_p[k, i_p, j_p])$ as the receptive field of $y_p[k, i_p, j_p]$, here we put subscript p for i, j's to avoid confusion. Note that here, the receptive fields differ by only the spatial dimensions i_p, j_p's. We calculate the receptive fields inductively for the outputs of each layer.

Let y_0 be the the output obtained by passing the input X through the first convolutional layer, which uses 3×3 filters with padding of 1. Then, the receptive field of y_0 is

$$\begin{aligned}\mathcal{R}(y_0[k, i_0, j_0]) = \big\{X[c, m, n] \mid &\max\{0, i_0 - 1\} \le m \le \min\{223, i_0 + 1\},\\ &\max\{0, j_0 - 1\} \le n \le \min\{223, j_0 + 1\},\ c = 1, 2, 3\big\}.\end{aligned}$$

Recall y_1 is the output obtained by passing y_0 through the second convolutional layer that uses 3×3 filters with padding of 1. Therefore the receptive field of y_1 is

$$\begin{aligned}\mathcal{R}(y_1[k, i_1, j_1]) &= \bigcup_{\substack{\max\{0,i_1-1\}\le i_0\le\min\{223,i_1+1\},\\ \max\{0,j_1-1\}\le j_0\le\min\{223,j_1+1\}}} \mathcal{R}(y_0[k, i_0, j_0])\\ &= \big\{X[c, m, n] \mid \max\{0, i_1 - 2\} \le m \le \min\{223, i_1 + 2\},\\ &\qquad\qquad \max\{0, j_1 - 2\} \le n \le \min\{223, j_1 + 2\},\ c = 1, 2, 3\big\}.\end{aligned}$$

As y_2 is the output obtained by passing y_1 through a maxpool layer with 2×2 filters and stride 2, we have

$$y_2[k, i_2, j_2] = \max\big\{y_1[k, 2i_2, 2j_2], y_1[k, 2i_2 + 1, 2j_2], y_1[k, 2i_2, 2j_2 + 1], y_1[k, 2i_2 + 1, 2j_2 + 1]\big\}.$$

Therefore, the receptive field of y_2 is

$$\begin{aligned}\mathcal{R}(y_2[k, i_2, j_2]) &= \bigcup_{(i_1,j_1)\in\{2i_2,2i_2+1\}\times\{2j_2,2j_2+1\}} \mathcal{R}(y_1[k, i_1, j_1])\\ &= \big\{X[c, m, n] \mid \max\{0, 2i_2 - 2\} \le m \le \min\{223, 2i_2 + 3\},\\ &\qquad\qquad \max\{0, 2j_2 - 2\} \le n \le \min\{223, 2j_2 + 3\},\ c = 1, 2, 3\big\}.\end{aligned}$$

As y_3 is the output obtained by passing y_2 through two convlutional layers that use 3×3 filters with padding of 1, and a maxpool layer with 2×2 filters and stride 2, repeating the similar arguments, we can see the range of m for i changes as following:

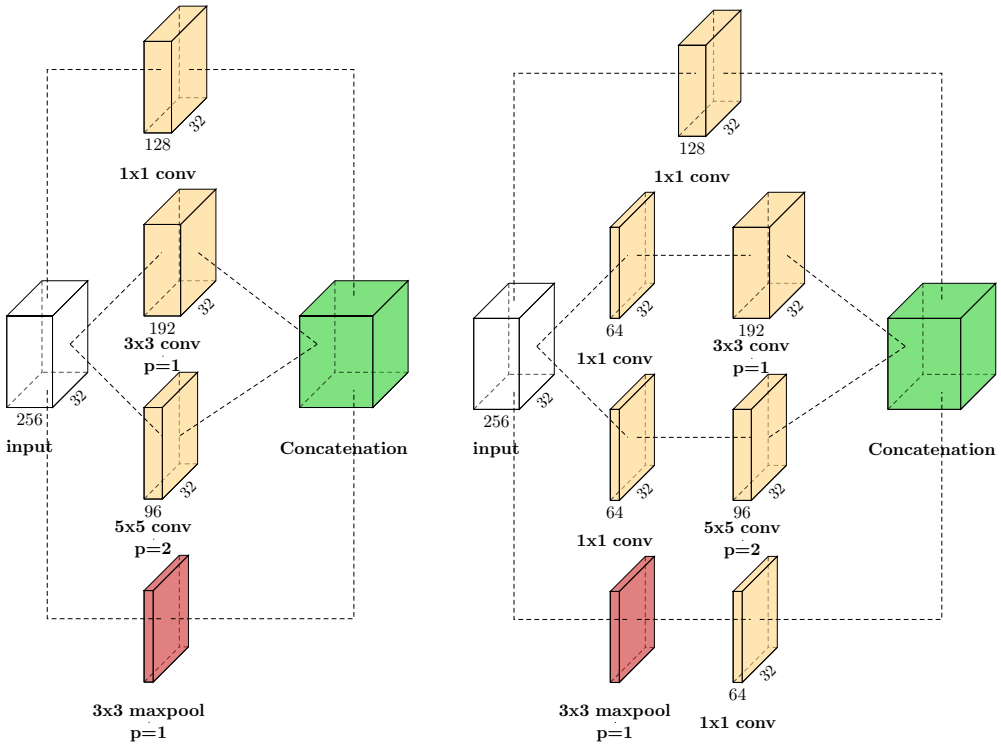
$$\begin{aligned}&\max\{0, 2i - 2\} \le m \le \min\{223, 2i + 3\}\\ \longrightarrow\ &\max\{0, 2i - 4\} \le m \le \min\{223, 2i + 5\}\\ \longrightarrow\ &\max\{0, 2i - 6\} \le m \le \min\{223, 2i + 7\}\\ \longrightarrow\ &\max\{0, 4i - 6\} \le m \le \min\{223, 4i + 9\}.\end{aligned}$$

The range of n for j changes with the same process, therefore the receptive field of y_3 becomes

$$\begin{aligned}\mathcal{R}(y_3[k, i_3, j_3]) = \big\{X[c, m, n] \mid &\max\{0, 4i_3 - 6\} \le m \le \min\{223, 4i_3 + 9\},\\ &\max\{0, 4j_3 - 6\} \le n \le \min\{223, 4j_3 + 9\},\ c = 1, 2, 3\big\}.\end{aligned}$$

■

Problem 5: Consider the naïve inception module and the inception module with 1×1 "bottleneck" convolutions. Assume the input has 32×32 spatial dimensions and 256 channels. The numbers of output channels of each convolution operation are specified in the figure. Assume all convolutions use biases. A nonlinear activation function is applied after every convolution.



Between the two modules, compare:

(i) the number of trainable parameters and

(ii) the number of additions, multiplications, and activation function evaluations (separately for the three types of operations) required to forward-evaluate the module.

Remark. A more complete investigation in the spirit of part (ii) would count the arithmetic operations of a gradient computation via a backward pass. For the sake of simplicity, we only consider the forward pass.

Clarification. For the purpose of this problem, include the additional operations incurred due to zero-padding. You do not need to compare operations incurred by the maxpool; not only does maxpool not utilize any additions, multiplications, or activation evaluations, but the amount of operations of the maxpools of two modules are identical, so there is no need to compare.

Solution. We first organize the calculations for one convolutional layer. Consider convolutional layer with the input $C_{\text{in}} \times h \times h$, the output $C_{\text{out}} \times h \times h$ and $C_{\text{out}} \times C_{\text{in}} \times k_h \times k_h$ kernel. Denoting the input tensor $X \in \mathbb{R}^{B \times C_{\text{in}} \times h \times h}$, output tensor $Y \in \mathbb{R}^{B \times C_{\text{out}} \times h \times h}$, filter $w \in \mathbb{R}^{C_{\text{out}} \times C_{\text{in}} \times k_h \times k_h}$, bias $b \in \mathbb{R}^{C_{\text{out}}}$ and activation function $\sigma \colon \mathbb{R} \to \mathbb{R}$, we have the following

$$Y_{\cdot,\ell,i,j} = \sigma\left(\sum_{\gamma=1}^{C_{\text{in}}} \sum_{\alpha=1}^{k_h} \sum_{\beta=1}^{k_h} w_{\ell,\gamma,\alpha,\beta} X_{\cdot,\gamma,i+\alpha-1,j+\beta-1} + b_\ell\right).$$

Since $\ell \in \{1, \ldots, C_{\text{out}}\}$ and $i, j \in \{1, \ldots, h\}$, we have

- # of parameters : $(C_{\text{in}} \times k_h^2 + 1) \times C_{\text{out}}$.
- # of multiplications : $k_h^2 \times h^2 \times C_{\text{in}} \times C_{\text{out}}$.
- # of additions : $k_h^2 \times h^2 \times C_{\text{in}} \times C_{\text{out}}$.
- # of activation function evaluations : $h^2 \times C_{\text{out}}$.

To add a note on the number of additions, for each ℓ, i, j there are $(k_h^2 \times C_{\text{in}} - 1)$ additions for triple summation and 1 addition for bias. Note, the number of multiplications and additions coincide. Applying the above observation, the calculation details are organized below.

Number of paramters.

	1×1 conv	3×3 conv with or without 1×1 conv	5×5 conv with or without 1×1 conv	3×3 maxpool with or without 1×1 conv
First module	$(256 \times 1^2 + 1) \times 128$	$(256 \times 3^2 + 1) \times 192$	$(256 \times 5^2 + 1) \times 96$	0
Second module	$(256 \times 1^2 + 1) \times 128$	$(256 \times 1^2 + 1) \times 64$ $+ (64 \times 3^2 + 1) \times 192$	$(256 \times 1^2 + 1) \times 64$ $+ (64 \times 5^2 + 1) \times 96$	$(256 \times 1^2 + 1) \times 64$

Number of multiplications/additions.

	1×1 conv	3×3 conv with or without 1×1 conv	5×5 conv with or without 1×1 conv	3×3 maxpool with or without 1×1 conv
First module	$1^2 \times 32^2 \times 256 \times 128$	$3^2 \times 32^2 \times 256 \times 192$	$5^2 \times 32^2 \times 256 \times 96$	0
Second module	$1^2 \times 32^2 \times 256 \times 128$	$1^2 \times 32^2 \times 256 \times 64$ $+ 3^2 \times 32^2 \times 64 \times 192$	$1^2 \times 32^2 \times 256 \times 64$ $+ 5^2 \times 32^2 \times 64 \times 96$	$1^2 \times 32^2 \times 256 \times 64$

Number of activation function evaluations.

	1×1 conv	3×3 conv with or without 1×1 conv	5×5 conv with or without 1×1 conv	3×3 maxpool with or without 1×1 conv
First module	$32^2 \times 128$	$32^2 \times 192$	$32^2 \times 96$	0
Second module	$32^2 \times 128$	$32^2 \times 64 + 32^2 \times 192$	$32^2 \times 64 + 32^2 \times 96$	$32^2 \times 64$

Summing up each row, in total we have

	Parameters	Multiplications	Additions	Activation function evaluations
First module	$1,089,952$	$1,115,684,864$	$1,115,684,864$	$425,984$
Second module	$346,720$	$354,418,688$	$354,418,688$	$622,592$

■

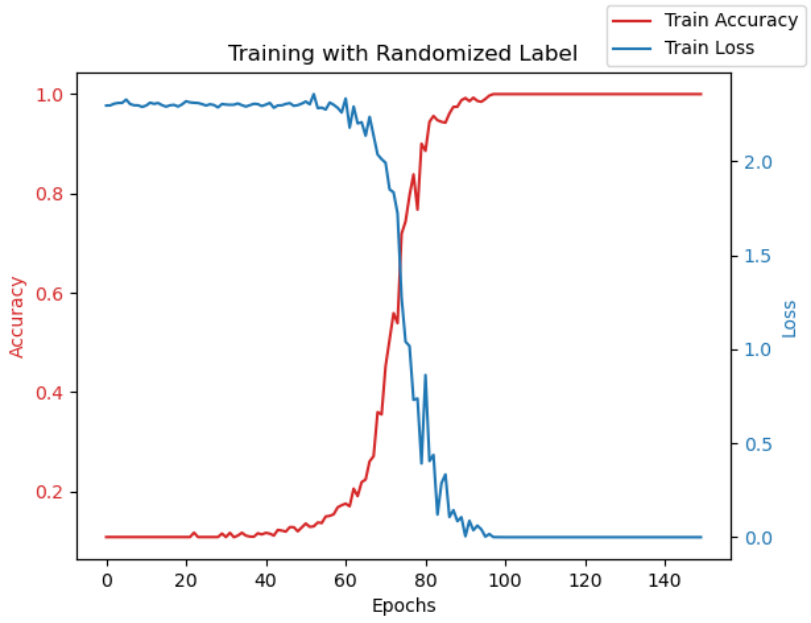
Problem 6: *Large neural networks memorize and interpolate training data.* In 2017, Zhang et al. [1] showed that modern deep neural networks can exactly memorize and interpolate training labels, even when the labels are completely randomized. The experiment considered a 10-class classification problem with (unmodified) data $X_1, \ldots, X_N$ and completely randomized labels $Y_i \sim \text{Uniform}\{1, \ldots, 10\}$ for $i = 1, \ldots, N$. Carry out this experiment with the MNIST data on a variation of the AlexNet architecture as provided in the starter code `label_memorization.py`. Use SGD with learning rate 0.1, batchsize 64, and 150 epochs. To reduce the computation time of this experiment, use only 10% of the MNIST training data, i.e., select a subset of $6,000$ images among the $60,000$ training images.

Clarification. Once Y_i is set to a random class, it should be fixed throughout the training epochs, i.e., do not randomize Y_i again every epoch.

Remark. Of course, the "trained" neural net achieves the generalization performance of 10%.

Remark. This paper by Zhang et al. [1] was highly influential as it shattered all classical statistical expectations. It meant neural networks generalize well despite having the capacity to completely overfit. The insight of this work lead to the formulation of the double descent phenomenon.

Hint. Expect the training to behave as follows:



Solution. See `label_memorization_sol.py`. ■

References

[1] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, Understanding deep learning requires rethinking generalization, *ICLR*, 2017.