ECE 685D HW2

Zanwen Fu (zf93)

Submission Instructions

1. Upload your Jupyter Notebook (`.ipynb` file).
2. Export all outputs and necessary derivations as one or multiple PDF files and upload them.

Mathematical Derivations

1. You can submit handwritten derivations, but LaTeX is strongly encouraged.
2. Illegible handwriting will result in a 10% point deduction, and you will need to resubmit the work in LaTeX or Unicode within 2 days.

LLM policy. The use of large language models (LLMs) is permitted for this assignment; if you use them, you **MUST** disclose how.

1 Problem 1: Backprop on a Residual MLP (30 pts)

We define f as a k-layer fully connected neural network (MLP) with *identity skip connections* at every hidden layer. Let $x \in \mathbb{R}^{m\times 1}$ and set $h_0 := x$. For $\ell = 1, \dots, k-1$:

$$a_\ell = W_\ell^\top h_{\ell-1} + b_\ell, \tag{1}$$

$$h_\ell = g(a_\ell) \; + \; h_{\ell-1}, \tag{2}$$

where g is an element-wise 1-Lipschitz activation with derivative $g'(\cdot)$. The output layer remains linear:

$$\hat{y} \;=\; W_k^\top h_{k-1} + b_k. \tag{3}$$

Let $k = 3$ and use the mean-squared error (MSE) loss

$$L(Y, \hat{Y}) \;=\; \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2.$$

Tasks.

(a) Derive $\dfrac{\partial L}{\partial W_3}$ and $\dfrac{\partial L}{\partial b_3}$.

q2_HW2

September 29, 2025

0.1 Problem 2: Customized Multi-View Dataset

[1]:
```
import random
from pathlib import Path

import torch
from torch.utils.data import Dataset, DataLoader
from PIL import Image
import matplotlib.pyplot as plt
import torchvision
from torchvision import transforms
```

[2]:
```
# Reproducibility
SEED = 7
random.seed(SEED)
torch.manual_seed(SEED)
```

[2]:
```
<torch._C.Generator at 0x1165249f0>
```

[3]:
```
def get_augment_pipeline():
    return transforms.Compose([
        transforms.RandomResizedCrop(size=28, scale=(0.80, 1.00)),
        transforms.RandomAffine(degrees=15, translate=(0.10, 0.10), shear=10),
        transforms.ToTensor(),
        transforms.RandomErasing(p=0.20, scale=(0.02, 0.20), ratio=(0.3, 3.3),␣
 ↪inplace=True),
    ])

base_transform = transforms.ToTensor()
augment_transform = get_augment_pipeline()
```

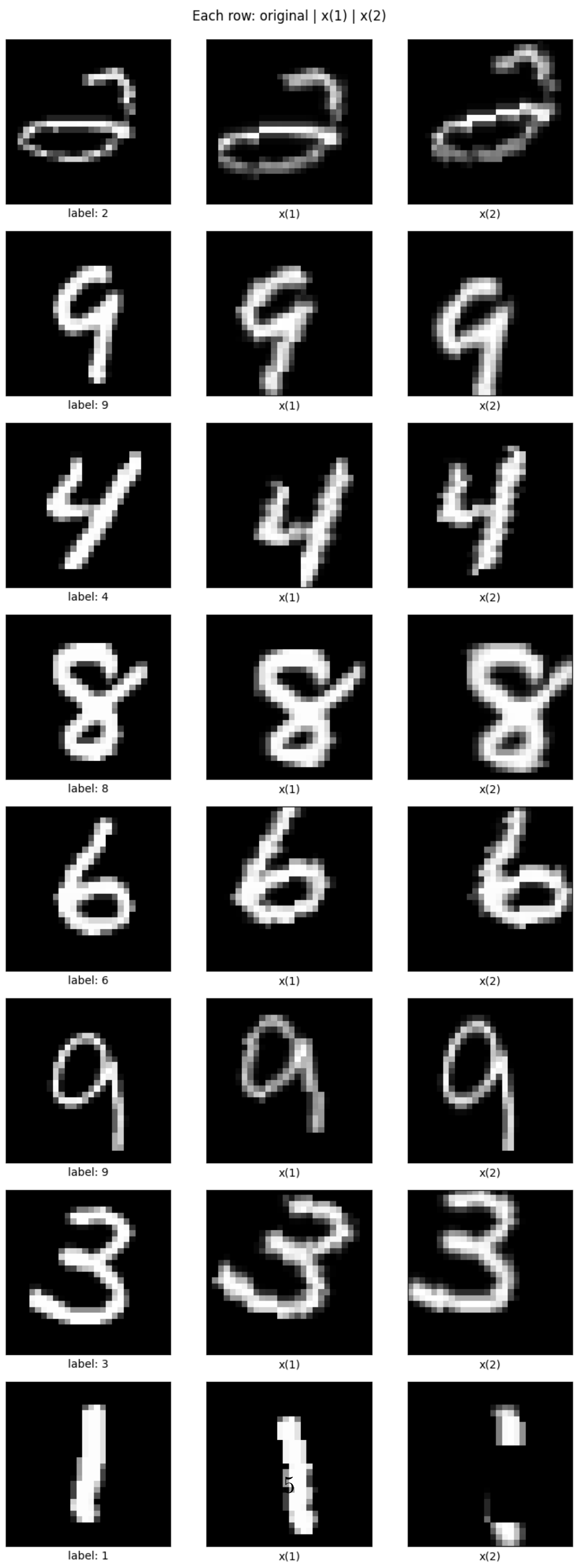
[4]:
```
class MNISTTwoView(Dataset):
    """
    Wrap a base dataset (returns PIL image, label) and produce:
      x0: original tensorized image
      x1, x2: two independently augmented views using the SAME pipeline
      y: label
    """
    def __init__(self, base_dataset, base_transform=None,␣
 ↪augment_transform=None):
        self.base = base_dataset
        self.base_transform = base_transform
        self.augment_transform = augment_transform
        self._to_pil = transforms.ToPILImage()

    def __len__(self):
        return len(self.base)

    def _ensure_pil(self, img):
        # If the base dataset returns a tensor, convert to PIL for torchvision␣
 ↪transforms
        if isinstance(img, Image.Image):
            return img
        return self._to_pil(img)  # tensor (C,H,W) -> PIL

    def __getitem__(self, idx):
        img, y = self.base[idx]
        img = self._ensure_pil(img)

        # Original (no aug besides ToTensor)
        x0 = self.base_transform(img) if self.base_transform else transforms.
 ↪ToTensor()(img)

        # Two independent augmentations using the SAME pipeline
        x1 = self.augment_transform(img) if self.augment_transform else x0
        x2 = self.augment_transform(img) if self.augment_transform else x0

        return x0, x1, x2, y
```

[ ]:
```
data_root = Path("./data")
data_root.mkdir(parents=True, exist_ok=True)

# Real MNIST
train_base = torchvision.datasets.MNIST(root=str(data_root), train=True,␣
 ↪download=True)

train_ds = MNISTTwoView(
    base_dataset=train_base,
    base_transform=base_transform,
    augment_transform=augment_transform
)

train_loader = DataLoader(train_ds, batch_size=8, shuffle=True, num_workers=0)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
```

```
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-
idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-
idx3-ubyte.gz to data/MNIST/raw/train-images-idx3-ubyte.gz

100%|     | 9912422/9912422 [00:03<00:00, 2968105.72it/s]

Extracting data/MNIST/raw/train-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-
idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-
idx1-ubyte.gz to data/MNIST/raw/train-labels-idx1-ubyte.gz

100%|     | 28881/28881 [00:00<00:00, 522154.62it/s]

Extracting data/MNIST/raw/train-labels-idx1-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-
idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-
idx3-ubyte.gz to data/MNIST/raw/t10k-images-idx3-ubyte.gz

100%|     | 1648877/1648877 [00:01<00:00, 1442657.91it/s]

Extracting data/MNIST/raw/t10k-images-idx3-ubyte.gz to data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-
idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-
idx1-ubyte.gz to data/MNIST/raw/t10k-labels-idx1-ubyte.gz

100%|     | 4542/4542 [00:00<00:00, 2943985.28it/s]

Extracting data/MNIST/raw/t10k-labels-idx1-ubyte.gz to data/MNIST/raw
```

[6]:
```
# Fetch one minibatch
x0, x1, x2, y = next(iter(train_loader))  # shapes: (8, 1, 28, 28)
B = x0.shape[0]
assert B == 8, f"Expected batch_size=8, got {B}"

fig, axes = plt.subplots(nrows=B, ncols=3, figsize=(8, 2.5*B))
fig.suptitle("Each row: original | x(1) | x(2)", y=0.995)

for i in range(B):
    # Column 0: original + label under it
    axes[i, 0].imshow(x0[i, 0].cpu().numpy(), cmap="gray")
    axes[i, 0].set_xticks([]); axes[i, 0].set_yticks([])
    axes[i, 0].set_xlabel(f"label: {int(y[i])}")

    # Column 1: x(1)
    axes[i, 1].imshow(x1[i, 0].cpu().numpy(), cmap="gray")
    axes[i, 1].set_xticks([]); axes[i, 1].set_yticks([])
    axes[i, 1].set_xlabel("x(1)")

    # Column 2: x(2)
    axes[i, 2].imshow(x2[i, 0].cpu().numpy(), cmap="gray")
    axes[i, 2].set_xticks([]); axes[i, 2].set_yticks([])
    axes[i, 2].set_xlabel("x(2)")

plt.tight_layout()
plt.show()
```


Each row: original | x(1) | x(2)
label: 2
x(1)
x(2)
label: 9
x(1)
x(2)
label: 4
x(1)
x(2)
label: 8
x(1)
x(2)
label: 6
x(1)
x(2)
label: 9
x(1)
x(2)
label: 3
x(1)
x(2)
label: 1
x(1)
x(2)

```
[ ]:
```

Answer (a) Stack the layer–2 activations and outputs over a minibatch as

$$H_2 := [h_2^{(1)}, \dots, h_2^{(N)}] \in \mathbb{R}^{d_h \times N}, \qquad \hat{Y} := [\hat{y}^{(1)}, \dots, \hat{y}^{(N)}] \in \mathbb{R}^{d_y \times N},$$

and write the linear output with bias broadcast as

$$\hat{Y} \;=\; W_3^\top H_2 + b_3\,\mathbf{1}^\top,$$

where $\mathbf{1} \in \mathbb{R}^N$ is the all-ones vector. The MSE is

$$L(Y, \hat{Y}) = \frac{1}{N}\|Y - \hat{Y}\|_F^2.$$

Step 1: Using $L = \frac{1}{N}\operatorname{tr}\big((Y - \hat{Y})^\top (Y - \hat{Y})\big)$,

$$dL = \frac{2}{N}\operatorname{tr}\big((\hat{Y} - Y)^\top d\hat{Y}\big) \;=\; \Big\langle \frac{\partial L}{\partial \hat{Y}},\, d\hat{Y} \Big\rangle_F \;\Rightarrow\; \boxed{\frac{\partial L}{\partial \hat{Y}} = \frac{2}{N}(\hat{Y} - Y)}$$

Step 2: Since $d\hat{Y} = dW_3^\top H_2 + db_3\,\mathbf{1}^\top$,

$$dL \;=\; \Big\langle \frac{\partial L}{\partial \hat{Y}},\, dW_3^\top H_2 \Big\rangle_F + \Big\langle \frac{\partial L}{\partial \hat{Y}},\, db_3\,\mathbf{1}^\top \Big\rangle_F \;=\; \operatorname{tr}\big(H_2 (\tfrac{\partial L}{\partial \hat{Y}})^\top dW_3\big) + \langle (\tfrac{\partial L}{\partial \hat{Y}})\mathbf{1},\, db_3 \rangle.$$

Thus,

$$\boxed{\frac{\partial L}{\partial W_3} = H_2 \Big(\frac{\partial L}{\partial \hat{Y}}\Big)^\top = \frac{2}{N}\, H_2 (\hat{Y} - Y)^\top}$$

and

$$\boxed{\frac{\partial L}{\partial b_3} = \Big(\frac{\partial L}{\partial \hat{Y}}\Big)\mathbf{1} = \frac{2}{N}\sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)}).}$$

Per-sample view: $\dfrac{\partial L}{\partial W_3} = \dfrac{2}{N}\displaystyle\sum_{i=1}^{N} h_2^{(i)} (\hat{y}^{(i)} - y^{(i)})^\top, \quad \dfrac{\partial L}{\partial b_3} = \dfrac{2}{N}\displaystyle\sum_{i=1}^{N} (\hat{y}^{(i)} - y^{(i)}).$

(b) Define $e_\ell := \frac{\partial L}{\partial h_\ell}$ and $\delta_\ell := \frac{\partial L}{\partial a_\ell} = e_\ell \odot g'(a_\ell)$ for $\ell = 1, 2$. Show the *residual recursion*

$$e_{\ell-1} \;=\; e_\ell \;+\; W_\ell\,\delta_\ell \quad \text{for } \ell = 1, 2.$$

(Hint: the skip adds an identity path $h_\ell \to h_{\ell-1}$ with derivative I.)

Answer (b) We want to show the residual recursion for $e_\ell := \frac{\partial L}{\partial h_\ell}$ and $\delta_\ell := \frac{\partial L}{\partial a_\ell} = e_\ell \odot g'(a_\ell)$. From the chain rule in Jacobian–transpose form, the gradient w.r.t. $h_{\ell-1}$ is

$$\frac{\partial L}{\partial h_{\ell-1}} = \Big(\frac{\partial h_\ell}{\partial h_{\ell-1}}\Big)^\top e_\ell + \Big(\frac{\partial a_\ell}{\partial h_{\ell-1}}\Big)^\top \delta_\ell.$$

Since $h_\ell = g(a_\ell) + h_{\ell-1}$, the derivative of h_ℓ w.r.t. $h_{\ell-1}$ has a direct identity contribution:

$$\frac{\partial h_\ell}{\partial h_{\ell-1}} = I \;\Rightarrow\; \Big(\frac{\partial h_\ell}{\partial h_{\ell-1}}\Big)^\top e_\ell = e_\ell.$$

Also, $a_\ell = W_\ell^\top h_{\ell-1} + b_\ell$, so

$$\frac{\partial a_\ell}{\partial h_{\ell-1}} = W_\ell \;\Rightarrow\; \Big(\frac{\partial a_\ell}{\partial h_{\ell-1}}\Big)^\top \delta_\ell = W_\ell\,\delta_\ell.$$

Combining both contributions yields the *residual recursion*:

$$\boxed{e_{\ell-1} \;=\; e_\ell \;+\; W_\ell\,\delta_\ell, \qquad \ell = 1,2.}$$

(c) Using (b), give expressions for $\frac{\partial L}{\partial W_2}$, $\frac{\partial L}{\partial b_2}$, $\frac{\partial L}{\partial W_1}$, and $\frac{\partial L}{\partial b_1}$. (The final result should not involve e_ℓ and δ_ℓ)

Answer (c) *O*ver a minibatch of size N, stack columns $H_\ell = [h_\ell^{(1)}, \dots, h_\ell^{(N)}]$, $A_\ell = [a_\ell^{(1)}, \dots, a_\ell^{(N)}]$, $\hat{Y} = [\hat{y}^{(1)}, \dots, \hat{y}^{(N)}]$, $Y = [y^{(1)}, \dots, y^{(N)}]$, and let $\mathbf{1} \in \mathbb{R}^N$ be all ones. From (a),

$$G \;:=\; \frac{\partial L}{\partial \hat{Y}} \;=\; \frac{2}{N}(\hat{Y} - Y).$$

Using $\hat{Y} = W_3^\top H_2 + b_3\mathbf{1}^\top$, the Jacobian $\partial\hat{Y}/\partial H_2 = W_3^\top$, so by the Jacobian–transpose rule

$$e_2 \;=\; \frac{\partial L}{\partial H_2} = \Big(\frac{\partial \hat{Y}}{\partial H_2}\Big)^\top \frac{\partial L}{\partial \hat{Y}} = W_3\,G.$$

Then by definition of δ_2,

$$\delta_2 \;=\; e_2 \odot g'(A_2) \;=\; (W_3 G) \odot g'(A_2).$$

By (b), $e_1 = e_2 + W_2\delta_2$, hence

$$\delta_1 \;=\; e_1 \odot g'(A_1) \;=\; (e_2 + W_2\delta_2) \odot g'(A_1) \;=\; \Big(W_3G + W_2\big((W_3G) \odot g'(A_2)\big)\Big) \odot g'(A_1).$$

For layer 2: $A_2 = W_2^\top H_1 + b_2\mathbf{1}^\top$ so $dA_2 = dW_2^\top H_1 + db_2\,\mathbf{1}^\top$ and

$$dL = \big\langle \tfrac{\partial L}{\partial A_2},\, dA_2\big\rangle_F = \big\langle \delta_2,\, dW_2^\top H_1\big\rangle_F + \big\langle \delta_2,\, db_2\,\mathbf{1}^\top\big\rangle_F = \operatorname{tr}\big(H_1\delta_2^\top dW_2\big) + \big\langle \delta_2\mathbf{1},\, db_2\big\rangle,$$

hence

$$\frac{\partial L}{\partial W_2} = H_1\delta_2^\top, \qquad \frac{\partial L}{\partial b_2} = \delta_2\,\mathbf{1}.$$

For layer 1: $A_1 = W_1^\top H_0 + b_1\mathbf{1}^\top$, so similarly

$$\frac{\partial L}{\partial W_1} = H_0\delta_1^\top, \qquad \frac{\partial L}{\partial b_1} = \delta_1\,\mathbf{1}.$$

Using the expressions for δ_2 and δ_1 above, we obtain

$$\boxed{\frac{\partial L}{\partial W_2} = H_1\big[(W_3G) \odot g'(A_2)\big]^\top}, \qquad \boxed{\frac{\partial L}{\partial b_2} = \big[(W_3G) \odot g'(A_2)\big]\mathbf{1}},$$

$$\boxed{\frac{\partial L}{\partial W_1} = H_0\Big[\big(W_3G + W_2\big((W_3G) \odot g'(A_2)\big)\big) \odot g'(A_1)\Big]^\top},$$

$$\boxed{\frac{\partial L}{\partial b_1} = \Big[\big(W_3G + W_2\big((W_3G) \odot g'(A_2)\big)\big) \odot g'(A_1)\Big]\mathbf{1}},$$

where $G = \frac{2}{N}(\hat{Y} - Y)$ and $A_\ell = W_\ell^\top H_{\ell-1} + b_\ell\mathbf{1}^\top$, $H_\ell = g(A_\ell) + H_{\ell-1}$.

2 Problem 2: Customized *Multi-View* Dataset (30 pts)

In this problem, you will build a custom dataset `MNISTTwoView` by inheriting `torch.utils.data.Dataset`. For each sample, return *two independently augmented views* of the same image:

$$(x^{(1)},\, x^{(2)},\, y).$$

Both $x^{(1)}$ and $x^{(2)}$ are produced by applying the *same transform pipeline* with independent randomness (e.g., small random crop/resize, random affine, random erasing). Use any standard library (e.g., `torchvision` or `albumentations`). You can either download the MNIST dataset from Kaggle `https://www.kaggle.com/datasets/oddrationale/mnist-in-csv` or directly through torchvision. (Remark: such two view setting is commonly used in self-supervised learning)

Requirements.

(a) Your dataset must be compatible with `torch.utils.data.DataLoader`.

(b) Visualize one minibatch of size 8 as a grid with three columns per example: *original*, $x^{(1)}$, $x^{(2)}$. Show labels under each original image.

3 Problem 3: Logistic Regression using Gradient and Newton's Methods (30 pts)

In this problem, we are going to fit a Logistic Regression model on the Breast Cancer dataset: `https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html` using gradient descent and Newton's methods. Since it's a binary classification problem, we will use the `BCELoss` from PyTorch. Specifically,

$$L(D;w) = \sum_{n=1}^{N} \Big[-y_n \log\!\big(\sigma(w^\top x_n)\big) - (1-y_n)\log\!\big(1-\sigma(w^\top x_n)\big) \Big], \tag{1}$$

$$\sigma(a) = \frac{1}{1+\exp(-a)}, \qquad D = \{x_n, y_n\}. \tag{2}$$

For this problem, you will compute the gradient $\frac{\partial L(D;w)}{\partial w}$ and the Hessian $\frac{\partial^2 L(D;w)}{\partial w_i \partial w_j}$ for the model weights. You can use the autograd tool in this problem.

Then, using the gradients and Hessian computed above, write a Python program to fit the Logistic Regression model on the Breast Cancer dataset using:

1. Gradient descent
2. Newton's method with exact expression for the Hessian
3. Newton's method with diagonal approximation for the Hessian

Plot and compare the loss, accuracy, and runtime graphs on the test set (learning rate = 0.1, test size = 30%) for all three methods. Briefly comment on the performance of the three methods.