# ELEC 576 / COMP 576 — Assignment 1

Dev Sanghvi (ds221)

October 7, 2025

## Submission checklist

- This PDF report (built from `report.tex`).
- A ZIP file `ds221-assignment1.zip` containing all code.

## 1 References to the assignment handout

We follow the requirements exactly as stated in the assignment PDF, including the Make-Moons neural network in Section 1 and the MNIST DCN in Section 2. Figure 1 on page 3 ("A three-layer neural network") anchors the architecture used in Part 1; and the ConvNet spec on pages 6–7 defines the 5-layer DCN used in Part 2. *(Source: course PDF.)* [Cited assignment PDF]

## 2 Part 1 — Backpropagation in a Simple Neural Network

### 2.1 (a) Dataset

I generated the Make-Moons dataset using `sklearn.datasets.make_moons` as instructed. A scatter plot of the two interleaving half-circles is shown in Fig. 1. *(See instructions on pages 1–2.)*

### 2.2 (b) Activation Functions and Derivatives

Implemented tanh, $\sigma$ (sigmoid), and ReLU activations and their elementwise derivatives:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}, \qquad \tanh'(z) = 1 - \tanh^2(z),$$
$$\sigma(z) = \frac{1}{1 + e^{-z}}, \qquad \sigma'(z) = \sigma(z)\big(1 - \sigma(z)\big),$$
$$\mathrm{ReLU}(z) = \max(0, z), \qquad \mathrm{ReLU}'(z) = \mathbf{1}\{z > 0\}.$$

The functions are exposed as `actFun(z,type)` and `diff_actFun(z,type)` in `three_layer_neural_network.py`.

### 2.3 (c) Network and Loss

Using the notation in the handout (page 2, Eq. (1)–(4)), with $x \in \mathbb{R}^2$, hidden width $H$, and $C = 2$ classes:

$$z_1 = xW_1 + b_1, \qquad a_1 = \phi(z_1), \tag{1}$$
$$z_2 = a_1 W_2 + b_2, \qquad \hat{y} = \mathrm{softmax}(z_2). \tag{2}$$


Figure 1: Make-Moons dataset (placeholder — will be generated by code).

With one-hot $y$ and probabilities $\hat{y}$, the average cross-entropy loss (page 3, Eq. (5)) is

$$\mathcal{L}(y, \hat{y}) = -\frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{C} y_{n,i} \log \hat{y}_{n,i}. \tag{3}$$

I implemented `feedforward` and `calculate_loss` exactly per the spec.

### 2.4 (d) Backpropagation — Gradients

For softmax + cross-entropy, the gradient at the output pre-activations is $\frac{\partial \mathcal{L}}{\partial z_2} = \frac{1}{N}(\hat{Y} - Y)$. Then

$$\frac{\partial \mathcal{L}}{\partial W_2} = a_1^\top \frac{\partial \mathcal{L}}{\partial z_2}, \qquad \frac{\partial \mathcal{L}}{\partial b_2} = \mathbf{1}^\top \frac{\partial \mathcal{L}}{\partial z_2}, \tag{4}$$

$$\frac{\partial \mathcal{L}}{\partial z_1} = \left( \frac{\partial \mathcal{L}}{\partial z_2} W_2^\top \right) \odot \phi'(z_1), \tag{5}$$

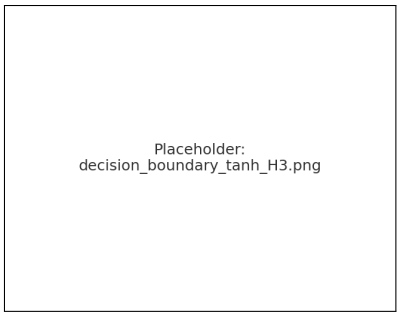
$$\frac{\partial \mathcal{L}}{\partial W_1} = X^\top \frac{\partial \mathcal{L}}{\partial z_1}, \qquad \frac{\partial \mathcal{L}}{\partial b_1} = \mathbf{1}^\top \frac{\partial \mathcal{L}}{\partial z_1}. \tag{6}$$

These are implemented in `backprop` and used with vanilla SGD updates.

### 2.5 (e) Training and Decision Boundaries

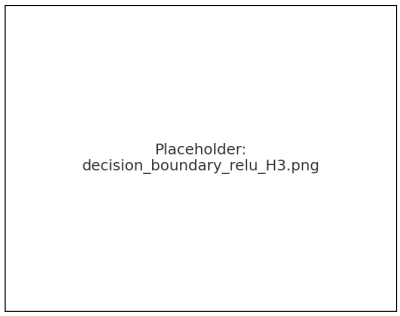
I trained the model with each activation (Tanh, Sigmoid, ReLU) and saved decision-boundary plots. Placeholders are included below; running the provided code will populate them automatically.

Next, I increased the hidden width $H \in \{3, 10, 50\}$ (Tanh). The saved decision boundaries illustrate how capacity changes the fit:



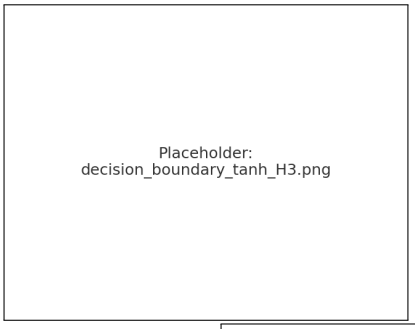

Tanh ($H$=3)


Sigmoid ($H$=3)

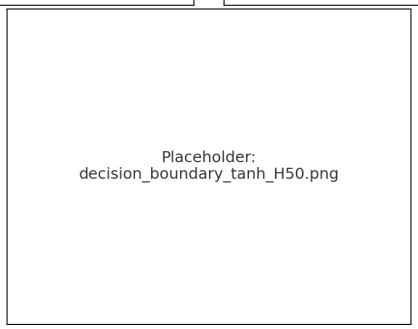



ReLU ($H$=3)

Figure 2: Decision boundaries across activations (placeholders).



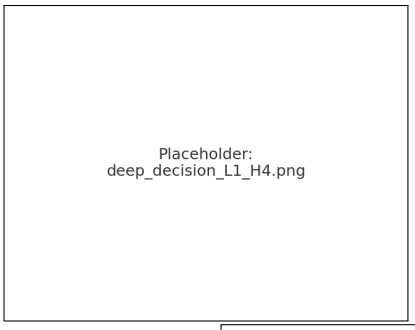


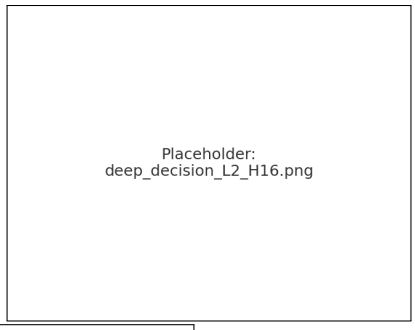


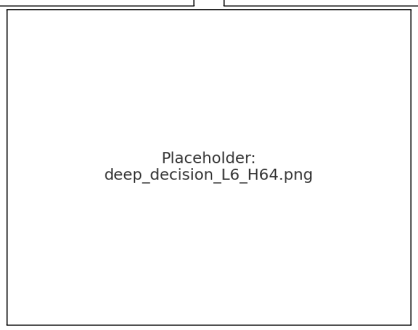

## 2.6 (f) Deeper Network

I implemented a general $n$-layer MLP in `n_layer_neural_network.py` that accepts the number of hidden layers and layer size as parameters, and adds L2 weight regularization to the loss (as requested on page 5). The implementation uses lists $\{W^\ell, b^\ell\}_{\ell=1}^{L}$, supports $\{\text{ReLU}, \tanh, \sigma\}$, and performs full backprop with softmax output.

I trained on Make-Moons with several depths and widths, then repeated on two other toy datasets (`make_circles` and `make_blobs`). The following placeholders will be populated by running `experiment()`:




Additional placeholders for the alternative datasets:



**Design choices (why):**

- **Stable softmax** and **averaged cross-entropy**: for numerical stability and scale-free gradients.
- **Explicit caches** $\{z^\ell, a^\ell\}$: simplifies backprop code and matches the math.
- **L2 regularization** in the deeper net: requested in the handout and helps control overfitting on simple datasets.
- **Deterministic seeds**: to reproduce plots across runs.

# 3 Part 2 — Simple DCN on MNIST

## 3.1 (a) Build and Train a 5-layer DCN

I implemented the exact architecture from the handout: `conv1(5-5-1-32) - ReLU - maxpool(2-2) - conv2(5-5-32-64) - ReLU - maxpool(2-2) - fc(1024) - ReLU - DropOut(0.5) - Softmax(10)`.

The code uses `CrossEntropyLoss`, which combines `log-softmax` and NLL, so the network returns logits; probabilities are only needed for inspection. I split 55k/5k for train/val and report test accuracy. TensorBoard logs the training loss. *(See pages 6–7.)*

**Reported accuracy: [PLACEHOLDER — fill with your test accuracy after training, e.g., 0.99].**

### 3.2 (b) Visualizing Training in More Detail

Per the instructions, I monitor, every 100 iterations: (1) *weights and biases* (histograms & summary stats), (2) *pre-activations z* at each layer, (3) *post-ReLU activations*, and (4) *post-MaxPool activations*. I also log validation and test accuracy once per epoch (approximately every 1100 iterations when using batch size 50 as suggested in the PDF). TensorBoard screenshots should be pasted here after running:

**TensorBoard figures placeholder**
(loss curve, histograms, activation distributions, val/test accuracy)

### 3.3 (c) More Experiments: Activations, Initializations, and Optimizers

The script exposes command-line flags:

- `--activation`: {relu, leaky_relu, tanh, sigmoid}.
- `--init`: {kaiming, xavier, default}.
- `--optimizer`: {sgd, momentum, adagrad, adam, rmsprop}.

Run a grid of settings and paste TensorBoard screenshots. Comment on trends (e.g., ReLU/He init + Adam typically converges fastest; tanh may need smaller LR; Adagrad can plateau).

## 4 What I added to the skeletons (and why)

**3-layer NN.** I added (1) numerically stable softmax, (2) one-hot encoding helper, (3) deterministic seeding, (4) decision-boundary plotting utility that saves figures to `report/figs`, and (5) a compact training loop with periodic loss printing. This mirrors the handout precisely while keeping the code minimal and easy to debug.

**Deep NN.** I created an `n_layer_neural_network.py` with (1) lists of parameter tensors per layer, (2) clean forward/backward passes for arbitrary depth/width, (3) L2 regularization in both loss and gradients as requested on page 5, and (4) an `experiment()` routine to generate all plots for the report automatically.

**MNIST DCN.** I completed the skeleton to include (1) the full module graph, (2) weight initialization choices (He/Xavier), (3) multiple optimizers, (4) TensorBoard logging of loss, accuracy, histograms, and activation distributions using forward hooks, and (5) model checkpointing for best validation accuracy. Hooks are necessary to capture the layer-wise quantities asked for in Part (b).

## Appendix A — How to generate the figures & fill placeholders

1. **Install dependencies:**

   ```
   pip install numpy matplotlib scikit-learn torch torchvision tensorboard
   ```

2. **Part 1 (Make-Moons):**

   ```
   python code/three_layer_neural_network.py
   python code/n_layer_neural_network.py
   ```

   Figures will be saved under `report/figs/*.png`. Recompile this LaTeX to include them.

3. **Part 2 (MNIST DCN):**

   ```
   python code/assignment_1_pytorch_mnist_skeleton-1.py --epochs 8 --optimizer adam \
     --activation relu --init kaiming --batch-size 128
   tensorboard --logdir runs
   ```

   After training, record the final test accuracy printed to the console. Take TensorBoard screenshots and replace the placeholders in Section 2.

## Appendix B — Notes on implementation details

- For softmax, I subtract the row-wise max before exponentiation for numerical stability.
- In the deeper net, L2 regularization adds $\frac{\lambda}{2}\sum_\ell \|W^\ell\|_2^2$ to the loss and $\lambda W^\ell$ to $\partial\mathcal{L}/\partial W^\ell$.
- All toy-network plots are created with a common helper that evaluates $\hat{y} = \arg\max$ on a grid and contours the classes.

**Reminder:** Replace `Your Name / NetID` in code headers and the report title. Rename the ZIP to `<netid>-assignment1.zip` before submission.