HW1P1 Bootcamp

Akshara, Anurag, Ron

01-17-26

Checklist

1. Getting Started
   - [ ] Watch the recitation 0 lectures (like Python, PyTorch, Debugging, HW Workflows), if required
   - [ ] Download code handout and extract the file
   - [ ] Setup new environment and Install required python libraries
   - [ ] Read the whole assignment write-up for an overview
   - [ ] Once read the write-up, you can find all the formulas in the Appendix section for easy access
   - [ ] Watch the 10-min video Backpropagation by 3B1B to understand the `backward` methods [1].
2. Complete the Components of a Multilayer Perceptron Model
   - [ ] Revisit lecture 2 about linear classifier, activation function, and perceptron
   - [ ] Complete the linear layer class
   - [ ] Complete the 4 activation functions
3. Complete 3 Multilayer Perceptron Models using Components Built
   - [ ] Revisit lecture 2 about MLP
   - [ ] Write a MLP model with 0 hidden layers
   - [ ] Write a MLP model with 1 hidden layers
   - [ ] Write a MLP model with 4 layers
4. Implement the Criterion Functions to evaluate a machine-learning model
   - [ ] Revisit lecture 3 about Loss
   - [ ] Implement Mean Squared Error (MSE) Loss for regression models
   - [ ] Implement Cross-Entropy Loss for classification models
5. Implement an Optimizer to train a machine learning model
   - [ ] Revisit lecture 6 about momentum, and lecture 7 about SGD
   - [ ] Implement SGD optimizer
6. Implement a Regularization method: Batch Normalization
   - [ ] Revisit lecture 8 about Batch normalization
   - [ ] Translate the element-wise equations to matrix equations
   - [ ] Write the code based on the matrix equations you wrote
7. Hand-in
   - [ ] Set all flags to `True` in `hw1p1_autograder_flags.py`
   - [ ] Make sure you pass all test cases in the local autograder
   - [ ] Make the `handin.tar` file and submit to autolab

Please track your progress using the Checklist provided in the Writeup!

System Setup

- Step 1: Download the Anaconda installer specific to your OS.
- Step 2: Open Anaconda Prompt (Windows) or Terminal (macOS/Linux).
- Step 3: File Extraction: **`tar -xvf HW1P1_S26_handout.tar`**
- Step 4: Navigate to your HW1P1 directory and create Environment:

```
conda create -n idls26 python=3.13 -y
```

- Step 4: Activate the Environment: conda activate **`idls26`**
- Step 5: Install the Requirements: **`pip install -r requirements.txt`**

Local Development & Submission Workflow

- Local Testing:
  - Set flags to **`True`** in hw1p1_autograder_flags.py to test individual components.
  - Execute autograder:

```
(idls26) python3 autograder/hw1p1_autograder.py
```

- Final Submission: Create submission file from the top-level directory:

```
tar -cvf handin.tar models mytorch
```

The Big Picture - Main Components

- Linear layer: Computes an affine transformation.
- Activation: Introduces non-linearity to allow the model to approximate complex functions.
- Loss: Quantifies the mismatch between prediction and ground truth.
- Optimizer: Adjusts model parameters to minimize the loss function based on gradients.
- Regularizer: Used to prevent Overfitting.

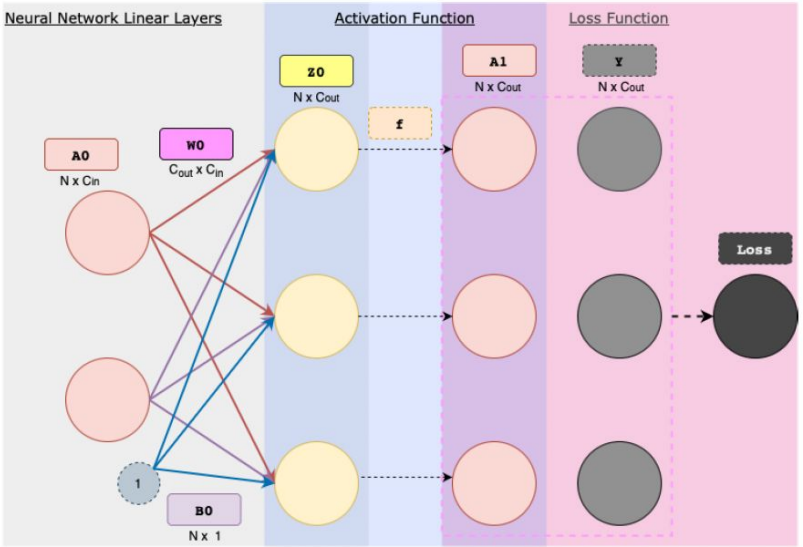


Figure A: An abstraction of an end-to-end topology.

Neural Network Layer

Linear Layer

- The Linear layer connects every input neuron to every output neuron via learnable weights and biases.
- Forward pass equation: $Z = A \cdot W^T + \iota_N \cdot b^T$
- Backward pass equations [Backpropagation]:

$$\frac{\partial L}{\partial A} = \left(\frac{\partial L}{\partial Z}\right) \cdot W$$

$$\frac{\partial L}{\partial W} = \left(\frac{\partial L}{\partial Z}\right)^T \cdot A$$

$$\frac{\partial L}{\partial b} = \left(\frac{\partial L}{\partial Z}\right)^T \cdot \iota_N$$

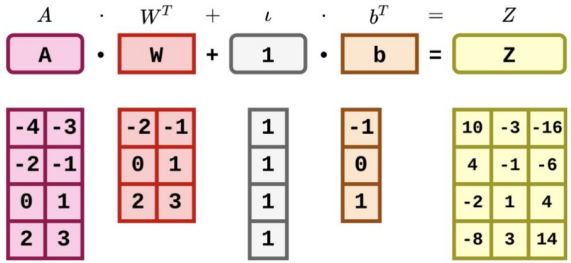
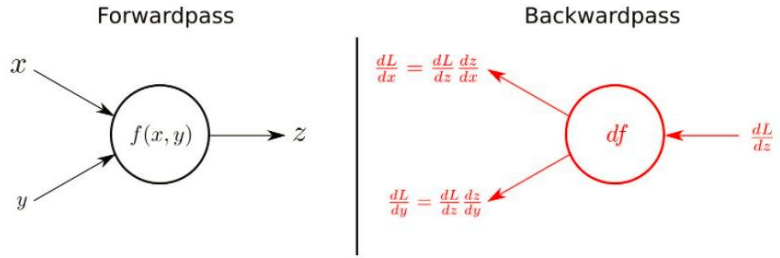


Figure B: Linear Layer Forward Example



1. ACTIVATIONS INTRO

The "Why": Neural networks need nonlinear components to approximate nonlinear functions. Without activations, the network remains linear regardless of depth.

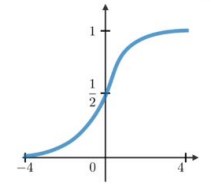
Types:

a. **Scalar Activation:** Applied element-wise like Sigmoid, ReLU and Tanh (each output depends on single input only)
b. **Vector Activation:** Each output element depends on all input elements (e.g. Softmax)

2. ALL ACTIVATIONS YOU IMPLEMENT

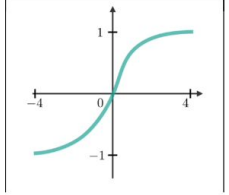
Sigmoid

$$\frac{1}{1+e^{-z}}$$



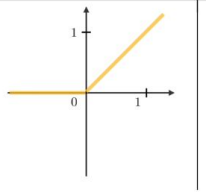
Tanh

$$\frac{e^{z}-e^{-z}}{e^{z}+e^{-z}}$$



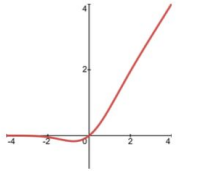
ReLU

$$\max(0, z)$$



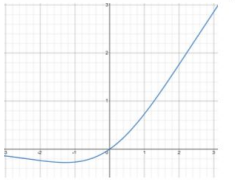
GELU

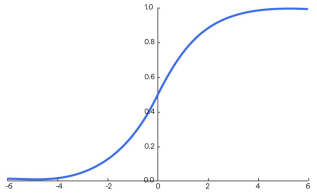
$$\frac{1}{2}Z \odot \left[1 + erf\left(\frac{Z}{\sqrt{2}}\right)\right]$$



Swish

$$\frac{z}{1+e^{-\beta z}}$$





3. RECOMMENDATIONS

- Use the hints:
  a. **For ReLU** : **Hint**: For coding, search and read the docs on `np.where`.
  b. **For GeLU:** **Hint**: Search the docs of the `math` and `scipy` libraries

BUILDING THE MLPs:

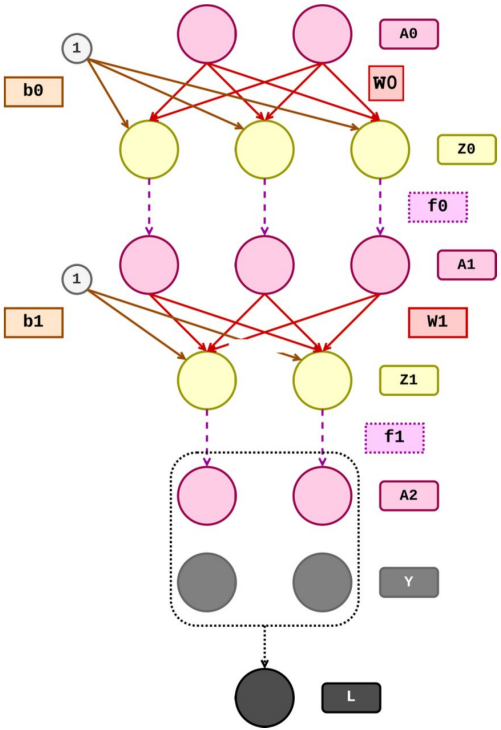


Figure H: MLP 1 Example Topology (Hidden Layers = 1)

COMMON PITFALLS & FIXES:

Mistake 1: Treating Linear and Activation Differently

- Don't write separate logic for Linear vs Activation layers
- Solution: Both are just self.layers[i] - call their forward/backward methods the same way

Mistake 2: Backward in Forward Order

- Don't loop through layers 0 to end during backward pass
- Solution: MUST use **reversed(range(len(self.layers)))** to go from end to 0

Mistake 3: Wrong Dimensions

- Don't guess the input/output sizes for each Linear layer
- Solution: Draw the network on paper! Follow the arrows in Figures G, H, and I from the writeup

What is a Loss Function?

A loss function measures how wrong your model's predictions are. It outputs a single scalar value that quantifies the mismatch between:

- Model Output (A): What your network predicted
- Ground Truth (Y): What the correct answer should be

Lower loss equals better model! We use this value to update network parameters via optimization.

MSE Loss (Mean Squared Error) - For Regression

- Use Case: Predicting continuous values (house prices, temperature, stock prices)

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2$$

$$SE(A,Y) = (A-Y) \odot (A-Y)$$

$$SSE(A,Y) = \iota_N^T \cdot SE(A,Y) \cdot \iota_C$$

$$MSELoss(A,Y) = \frac{SSE(A,Y)}{N \cdot C}$$

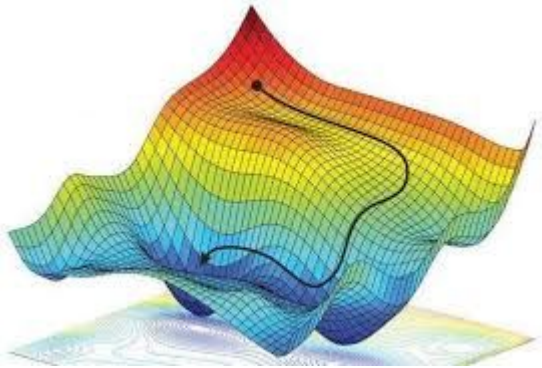
Cross-Entropy Loss - For Classification

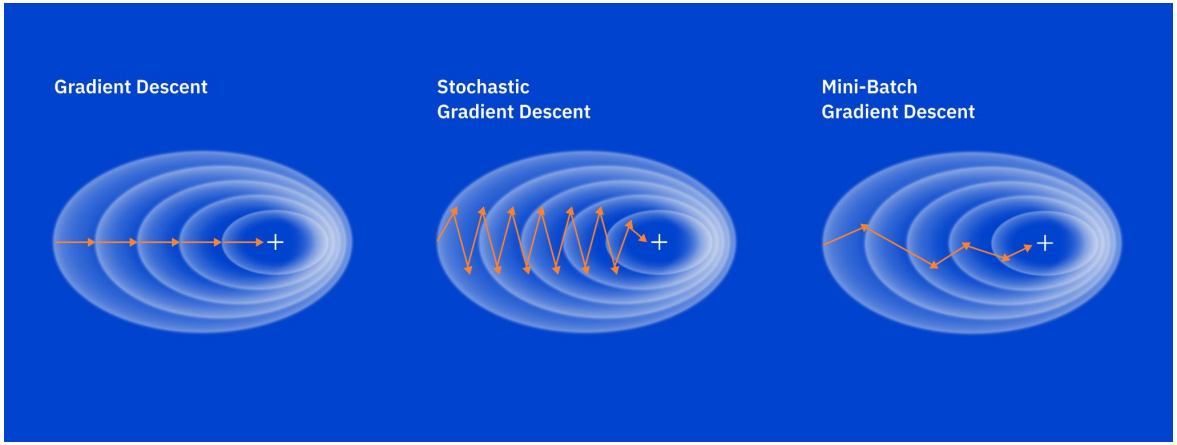
- Use Case: Predicting categories (cat vs dog, digit 0-9, spam vs not spam)
- Negative log probability of correct class

$$L = -\sum_{k=1}^{K} y_k \log(p_k)$$

Optimizers

Stochastic Gradient Descent (SGD)





Without Momentum

$$W := W - \lambda \frac{\partial L}{\partial W}$$

$$b := b - \lambda \frac{\partial L}{\partial b}$$

$$v_W := \mu v_W + \frac{\partial L}{\partial W} \qquad W := W - \lambda v_W$$

$$v_b := \mu v_b + \frac{\partial L}{\partial b} \qquad b := b - \lambda v_b$$

With Momentum

Regularization

Batch Normalization

$$\mu_j = \frac{1}{N}\sum_{i=1}^{N} Z_{ij}$$

$$\sigma_j^2 = \frac{1}{N}\sum_{i=1}^{N}(Z_{ij} - \mu_j)^2$$

$$\hat{Z}_i = \frac{Z_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

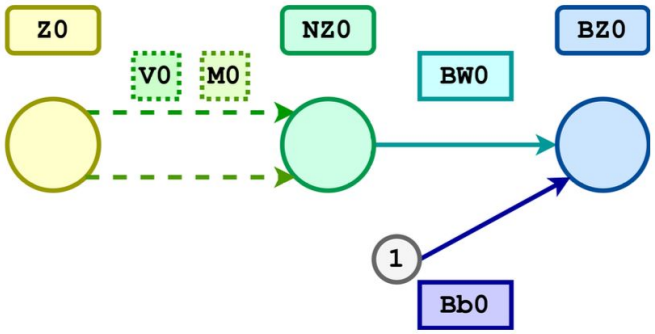
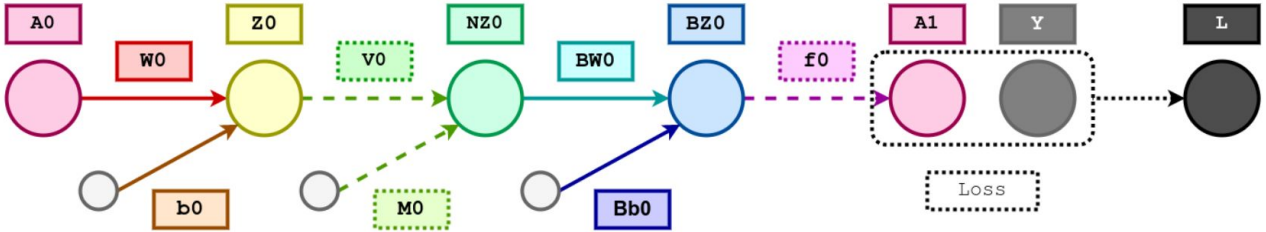


Figure L: Batchnorm Topology

$$E[Z] = \alpha * E[Z] + (1 - \alpha) * \mu$$
$$Var[Z] = \alpha * Var[Z] + (1 - \alpha) * \sigma^2$$

$$\hat{Z}_i = \frac{Z_i - E[Z]}{\sqrt{Var[Z] + \epsilon}}$$

$$\tilde{Z}_i = \gamma \odot \hat{Z}_i + \beta$$



Troubleshooting