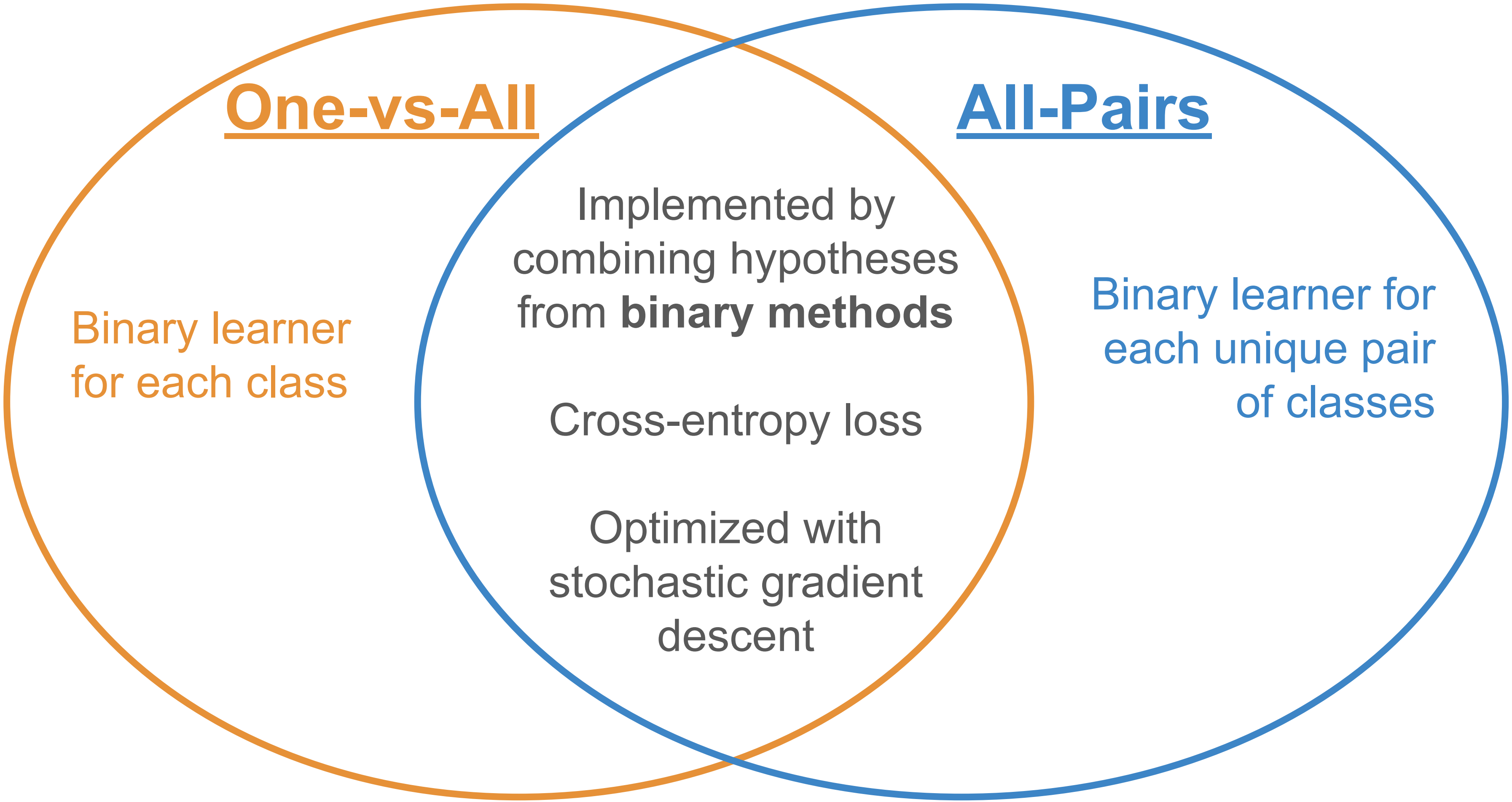
# Multiclass Classification: **One-vs-All** and **All-Pairs**

Adam Ibrahim, Nicolas Kim, Erin Okey

https://github.com/nickkim1/data_2060_final_project.git


One-vs-All
All-Pairs
Binary learner for each class
Implemented by combining hypotheses from **binary methods**
Cross-entropy loss
Optimized with stochastic gradient descent
Binary learner for each unique pair of classes

# Representation

## Data

d numeric features

$$\mathcal{X} = \mathbb{R}^d$$

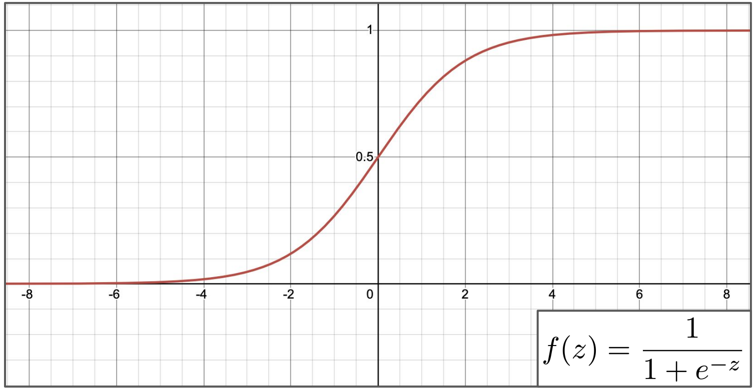
labeled 1 or 0

$$\mathcal{Y} = \{1, 0\}$$

## Hypothesis class

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{-\langle \mathbf{w}, \mathbf{x} \rangle}}$$

probability of class 1



$\boxed{\text{Loss}}$

$$L_s(h_w) = -\frac{1}{m}\sum_{i=1}^{m}\left(\boxed{y_i \log h_w(x_i)} + \boxed{(1-y_i)\log(1-h_w(x_i))}\right)$$

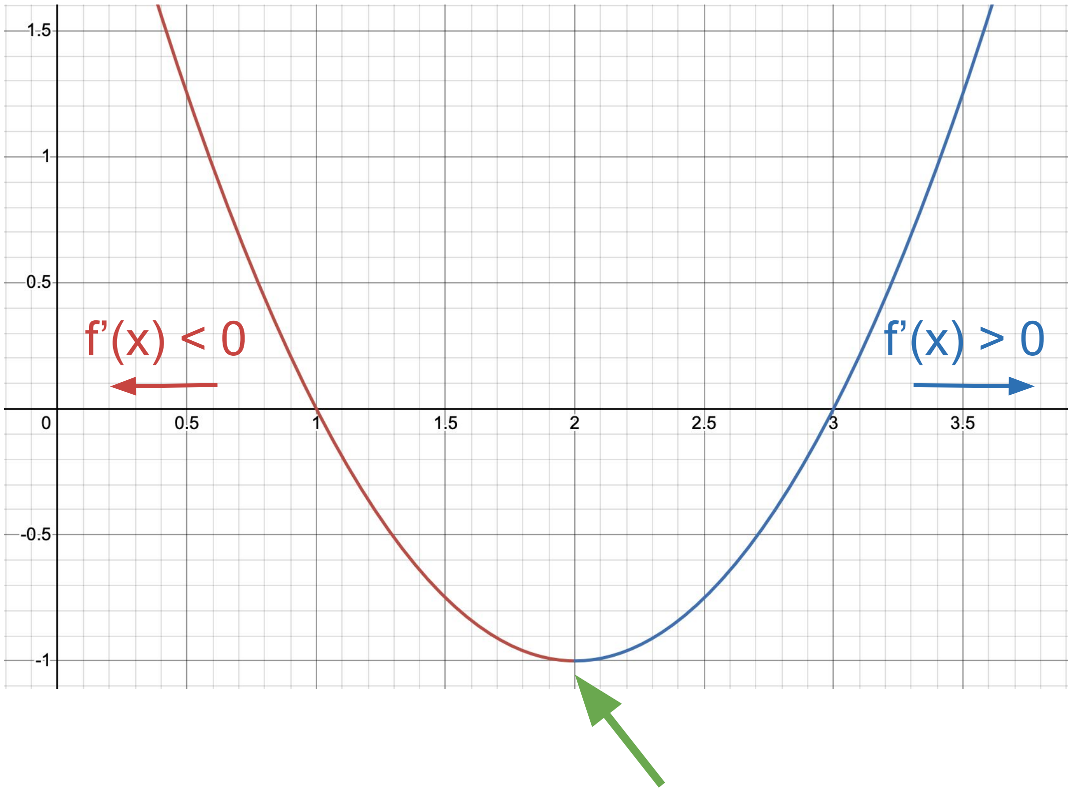
$$\mathcal{Y} = \{\, 1 \,,\, 0 \,\}$$

$$\log(z) \to -\infty \quad \text{as } z \to 0$$

$$\log(z) \to 0 \quad \text{as } z \to 1$$

# Optimizer - Gradient Descent

$$w \leftarrow w - \alpha \frac{\partial L(h_w)}{\partial w}$$



# Optimizer - Stochastic Gradient Descent

Idea: $$w \leftarrow w - \alpha \boxed{\frac{\partial L(h_w)}{\partial w}}$$

Time consuming to compute this using all data points. Instead, only use a random subset

```
Inputs: training examples S , step size α, batch size b < |S|

Initialize w ∈ ℝ^(k×d) randomly
converged ← False
while !converged:
    Shuffle S
    for i = 0 … ⌈ |S| / b ⌉ − 1 :
        S′ ← S[i · b : (i + 1) · b]
        w ← w − α∇L_S′(h_w)
    converged ← check_convergence(S, w)
return
```

pseudocode take from DATA2060 Fall 2025 Lecture 6, Slide 19

# Algorithm (Pseudocode)

**One-versus-All**

**input:**
training set $S = (\mathbf{x}_1,\, y_1), \ldots, (\mathbf{x}_m,\, y_m)$
algorithm for binary classification $A$
**foreach** $i \in \mathcal{Y}$
let $S_i = (\mathbf{x}_1,\ (-1)^{\mathbb{1}_{[y_1 \neq i]}}), \ldots, (\mathbf{x}_m,\ (-1)^{\mathbb{1}_{[y_m \neq i]}})$
let $h_i = A(S_i)$
**output:**
the multiclass hypothesis defined by $h(\mathbf{x}) \in \operatorname{argmax}_{i \in \mathcal{Y}} h_i(\mathbf{x})$

**All-Pairs**

**input:**
training set $S = (\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)$
algorithm for binary classification $A$
**foreach** $i, j \in \mathcal{Y}$ s.t. $i < j$
initialize $S_{i,j}$ to be the empty sequence
**for** $t = 1, \ldots, m$
If $y_t = i$ add $(\mathbf{x}_t, 1)$ to $S_{i,j}$
If $y_t = j$ add $(\mathbf{x}_t, -1)$ to $S_{i,j}$
let $h_{i,j} = A(S_{i,j})$
**output:**
the multiclass hypothesis defined by
$h(\mathbf{x}) \in \operatorname{argmax}_{i \in \mathcal{Y}} \left( \sum_{j \in \mathcal{Y}} \operatorname{sign}(j - i)\, h_{i,j}(\mathbf{x}) \right)$

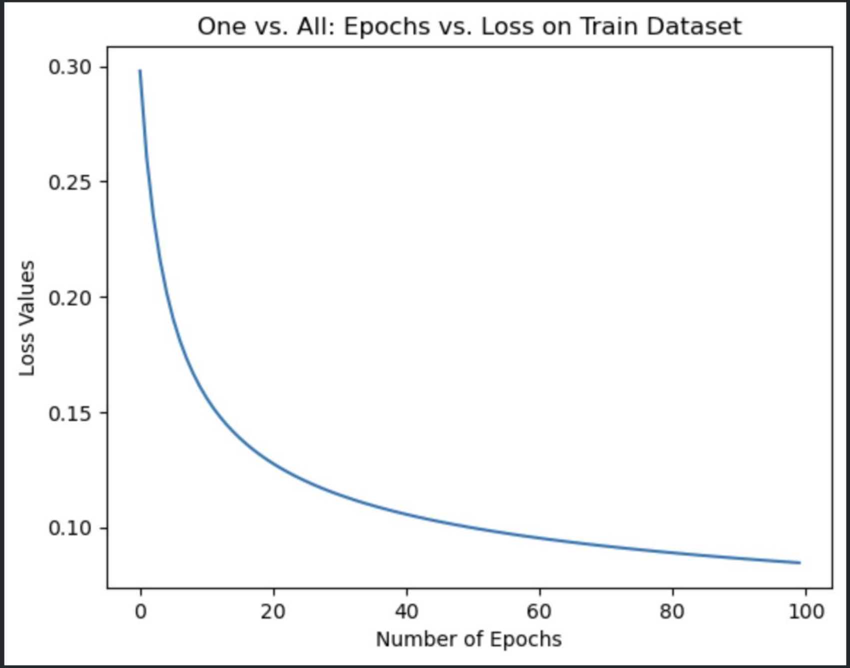
pseudocode take from
DATA2060 Fall 2025
Lecture 5, Slides 13,14

# Hyperparameters (for both one vs. rest and all pairs)

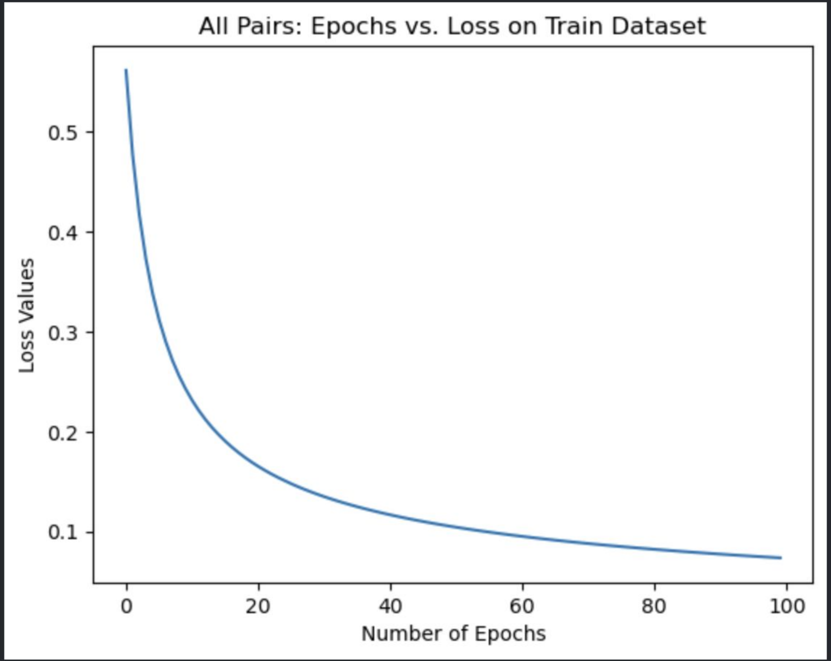
- **Dataset size:** 2000 training examples
  - Digit labels encoded as integers: 0-9 and pixel features: normalized between 0-1 (divide all values by 255)
- **Batch size:** 32
- **Lambda:** 0.001
- **Num epochs:** 100

# Accuracies and Loss Curves (One-vs-All - L, All-Pairs - R)

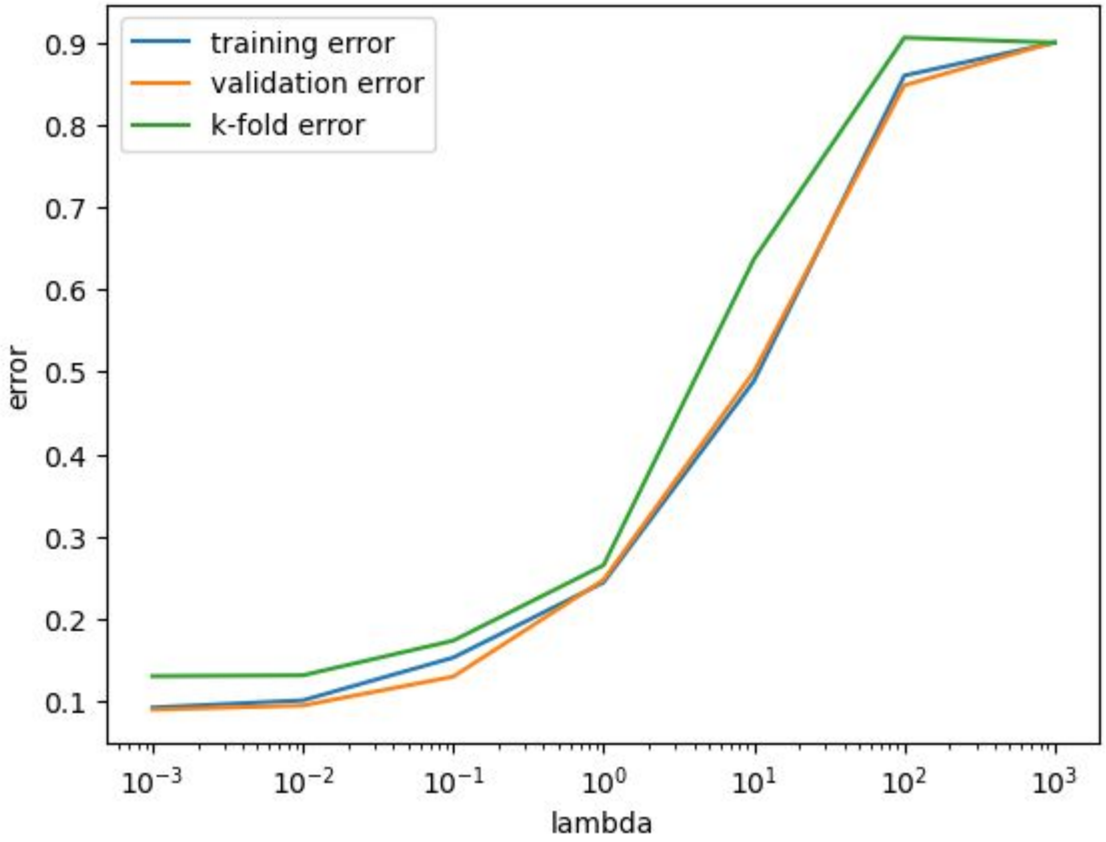
```
One vs. All: Train Accuracy: 0.906875
One vs. All: Validation Accuracy: 0.91
```



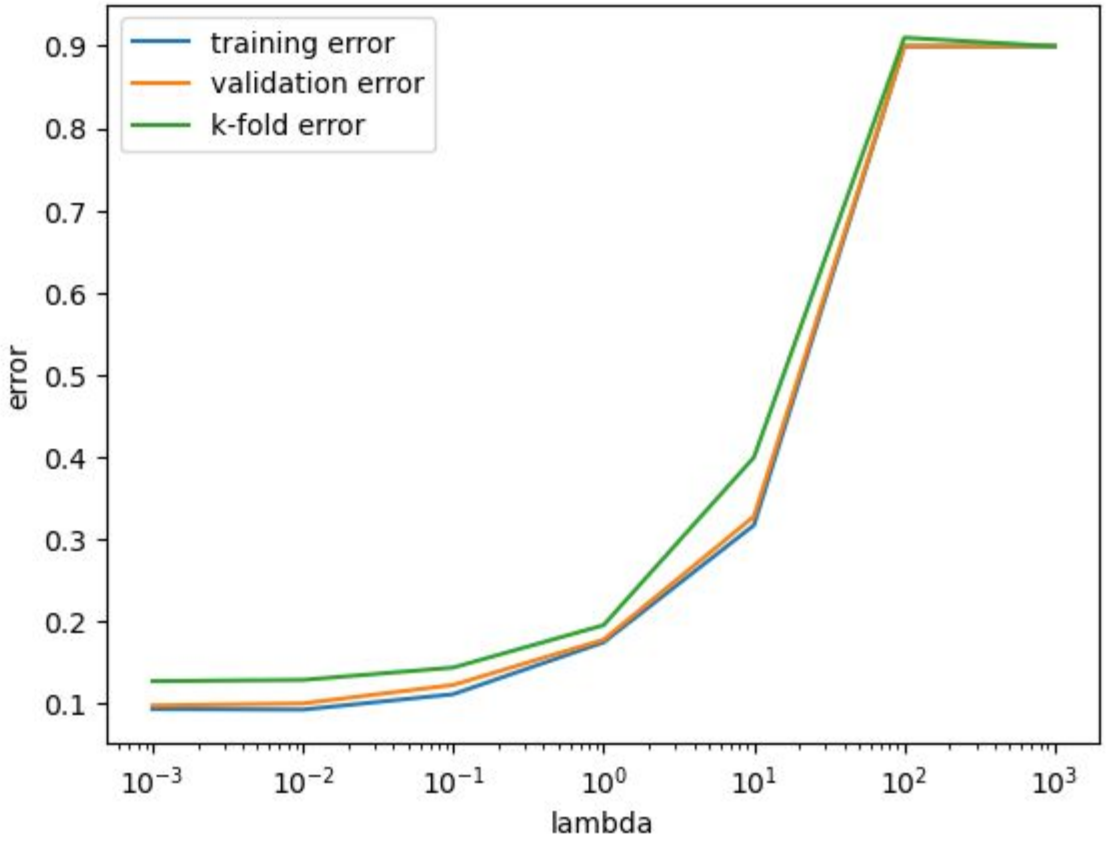

```
All Pairs: Train Accuracy: 0.90875
All Pairs: Validation Accuracy: 0.905
```



# Error vs Regularization Strength (One-vs-All)



# Error vs Regularization Strength (All-Pairs)



# Sklearn Comparison: One-vs-All and All-Pairs Results

```
=== One-vs-Rest (one vs. all) comparison ===
Our OVR logistic regression accuracy:      0.9075
Sklearn OVR logistic regression accuracy: 0.875
Predictions identical on this dataset? False
```

```
=== One-vs-One (all-pairs) comparison ===
Our OVO logistic regression accuracy:      0.9025
Sklearn OVO logistic regression accuracy: 0.9075
Predictions identical on this dataset? False
```

- **Explanation of Differences:** Our implementation uses **manual gradient descent** whereas sklearn implementation uses a **different solver (LBFGS), which converges differently and can reach slightly different optima on the same dataset.**
- There are also various other parameters (tolerance, max_iters) that sklearn sets that we were unable to account for with our basic implementation.
- Either way we were able to demonstrate **highly comparable results to a slight degree of divergence.**

# Summary - what was interesting/challenging?

1. **Manually tuning** hyperparameters was difficult: (e.g., finding the right combination of things like batch size and lambda)
2. **Preprocessing** the data into a suitable representation was also fairly involved (e.g., scaling them, encoding labels)
3. **Vectorizing** the pseudocode in Numpy and making sure shapes were compatible

# References


Khodabakhsh, Hojjat. *MNIST Dataset*. Kaggle. Accessed December 10, 2025. https://www.kaggle.com/datasets/hojjatk/mnist-dataset.

Pedregosa, F. et al., 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct), pp.2825–2830.

Shalev-Shwartz, Shai, and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge university press, 2014.

Zsom, Andras. “Lecture 5: Logistic Regression.”, Lecture, Brown University, September 18, 2025.

Zsom, Andras. “Lecture 6: SGD, Data Prep, and other Practicalities.”, Lecture, Brown University, September 23, 2025.