COMPSCI 389
Introduction to Machine Learning

Classification

Prof. Philip S. Thomas (pthomas@cs.umass.edu)

Note: This presentation covers (and provides additional context/information regarding) `22 Classification.ipynb`

Regression and Classification (Review)

- Within supervised learning, recall that a data set is a set of input-output pairs (X, Y).
- **Regression**: Y is a continuous number.
  - Multivariate Regression: Y is a vector. That is, $Y \in \mathbb{R}^m$ and $m > 1$.
- **Classification**: Y is categorical (mapped to an integer).
  - Binary Classification: $Y \in \{0,1\}$ or $Y \in \{-1,1\}$.
  - Multi-Class Classification: $Y \in \{0,1, \ldots, k\}$.

Regression → Classification

- Two changes for parametric methods:
  1. Change the parametric model so that it outputs a discrete label as a prediction rather than a number
  2. Select a loss function that is appropriate for classification tasks
- Note: Techniques differ for non-parametric methods
  - E.g., we discussed nearest neighbor (and variants) for classification
  - E.g., there are other custom non-parametric methods for classification like *decision trees*, which are beyond the scope of this course.
- Terminology: Each possible value of the label is called a **class**

Parametric models for classification

- Assume m classes (possible values of the label)
- Change parametric model to have m outputs rather than one.
- **Deterministic**:
  - Class with the highest output is the predicted class.
  - Simple and effective
  - Gradient of the loss function is typically zero, making this impractical for training.
- **Stochastic**:
  - The m outputs are converted to a probability distribution over the classes, and the label is sampled from this distribution.
  - The larger the output, the higher the probability of the class being selected

Stochastic Models: Softmax

- The **softmax** function converts the m outputs to a distribution over the m class values.
- Let $\mathrm{out}_1, \ldots, \mathrm{out}_m$ be the model outputs.
- Probabilities cannot be negative, so convert each output to a positive value:

$$\mathrm{out}_1, \ldots, \mathrm{out}_m \rightarrow \mathrm{e}^{\mathrm{out}_1}, \ldots, e^{\mathrm{out}_m}$$

- A probability distribution must sum to one, so divide each by the sum:

$$\frac{e^{\mathrm{out}_1}}{\sum_{k=1}^{m} e^{\mathrm{out}_k}}, \frac{e^{\mathrm{out}_2}}{\sum_{k=1}^{m} e^{\mathrm{out}_k}}, \ldots, \frac{e^{\mathrm{out}_m}}{\sum_{k=1}^{m} e^{\mathrm{out}_k}}.$$

Stochastic Models: Softmax

$$\Pr(\hat{Y}_i = \hat{y}) = \frac{e^{\mathrm{out}_{\hat{y}}}}{\sum_{k=1}^{m} e^{\mathrm{out}_k}}.$$

Binary Classification

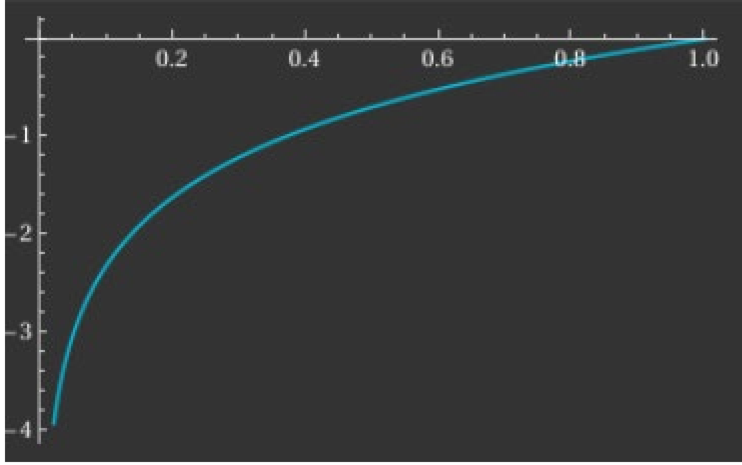
- Special case where $Y_i \in \{0,1\}$ or $Y_i \in \{-1,1\}$
  - Typically 1 is called the "positive class"
- Parametric models need only have one output, not $m = 2$
  - This output encodes the probability of the positive class.
  - The probability of the negative class is $1 - \Pr(\text{positive class})$.
- The output of the model must be scaled to $[0,1]$.
  - This can be done using the logistic function (sigmoid):

$$\Pr(\hat{Y}_i = 1) = \sigma(\text{out}_1),$$

where $\sigma(z) = \frac{1}{1+e^{-z}}$, and

$$\Pr(\hat{Y}_i = 0) = 1 - \Pr(\hat{Y}_i = 1).$$

Loss Functions for Classification



$\ln(p)$

- There are many loss functions for classification.
  - You can make your own that is tailored to your problem!
- Cross-Entropy Loss (log loss) is the most common.

$$\text{Cross-Entropy Loss}(w, D) = -\frac{1}{n}\sum_{i=1}^{n} \ln\Big(\Pr(Y_i = \hat{Y}_i)\Big).$$

- The $\frac{1}{n}$ is sometimes omitted (it makes no difference).

Binary Cross-Entropy Loss

$$\text{Cross-Entropy Loss}(w, D) = -\frac{1}{n}\sum_{i=1}^{n} \ln\left(\Pr(Y_i = \hat{Y}_i)\right).$$

- What if $Y_i \in [0,1]$, not $Y_i \in \{0,1\}$?

Note: PyTorch clamps the logarithm terms to the range $[-100,100]$.

$$\text{Cross-Entropy Loss}(w, D) = -\frac{1}{n}\sum_{i=1}^{n} Y_i \ln\left(\Pr\left(\hat{Y}_i = 1\right)\right) + (1 - Y_i)\ln\left(\Pr\left(\hat{Y}_i = 0\right)\right)$$

- **Question**: When $Y_i \in \{0,1\}$ is this equivalent to the first expression?
- **Answer**: Yes!

Logistic Regression

- **Logistic regression** uses the **logistic model** or **logit model**
  - Like a "linear" parametric model for classification

$$\sigma\big(w \cdot \phi(X_i)\big)$$

$$\Pr(\hat{Y}_i = 1 | X_i) = \frac{1}{1 + e^{-w \cdot \phi(X_i)}}.$$

- Use cross-entropy loss
  - Equivalent to maximizing the "likelihood" of the data given the model.

$$\text{Cross-Entropy Loss}(w, D) = -\frac{1}{n}\sum_{i=1}^{n} \ln\Big(\Pr(Y_i = \hat{Y}_i)\Big).$$

Stochastic → Deterministic Models

- During training often models are viewed as stochastic (minimizing cross-entropy loss).
- If the model is highly confident of the class for an input, the output for that class will become large
  - No matter how large it is, the resulting probability of the label will not be 1

$$\Pr(\hat{Y}_i = 1 | X_i) = \frac{1}{1 + e^{-w \cdot \phi(X_i)}}.$$

- To enable models to make deterministic predictions, often models are *evaluated* (and then deployed to make predictions for new data) as deterministic models, even if they are trained as stochastic models.

Example: Iris Data (nothing new!)

```python
# Load the Iris dataset
iris = load_iris()
X = iris.data
y = iris.target

# Convert to PyTorch tensors
X_tensor = torch.tensor(X, dtype=torch.float32)
y_tensor = torch.tensor(y, dtype=torch.long)    # NOTE: The labels are now integers

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X_tensor, y_tensor, test_size=0.5, random_state=42)
```

Create ANN model (3 classes, 3 outputs!)

```
# Define the ANN model
class ANN(nn.Module):
    def __init__(self):
        super(ANN, self).__init__()
        self.fc1 = nn.Linear(4, 10)  # 4 input features, 10 hidden nodes
        self.fc2 = nn.Linear(10, 3)  # 3 output classes NOTE: One output per class

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Prepare for Training (select classification loss!)

```
model = ANN()

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()  # NOTE: We select a classification loss
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training the model
epochs = 10000
train_losses = []
test_losses = []
```

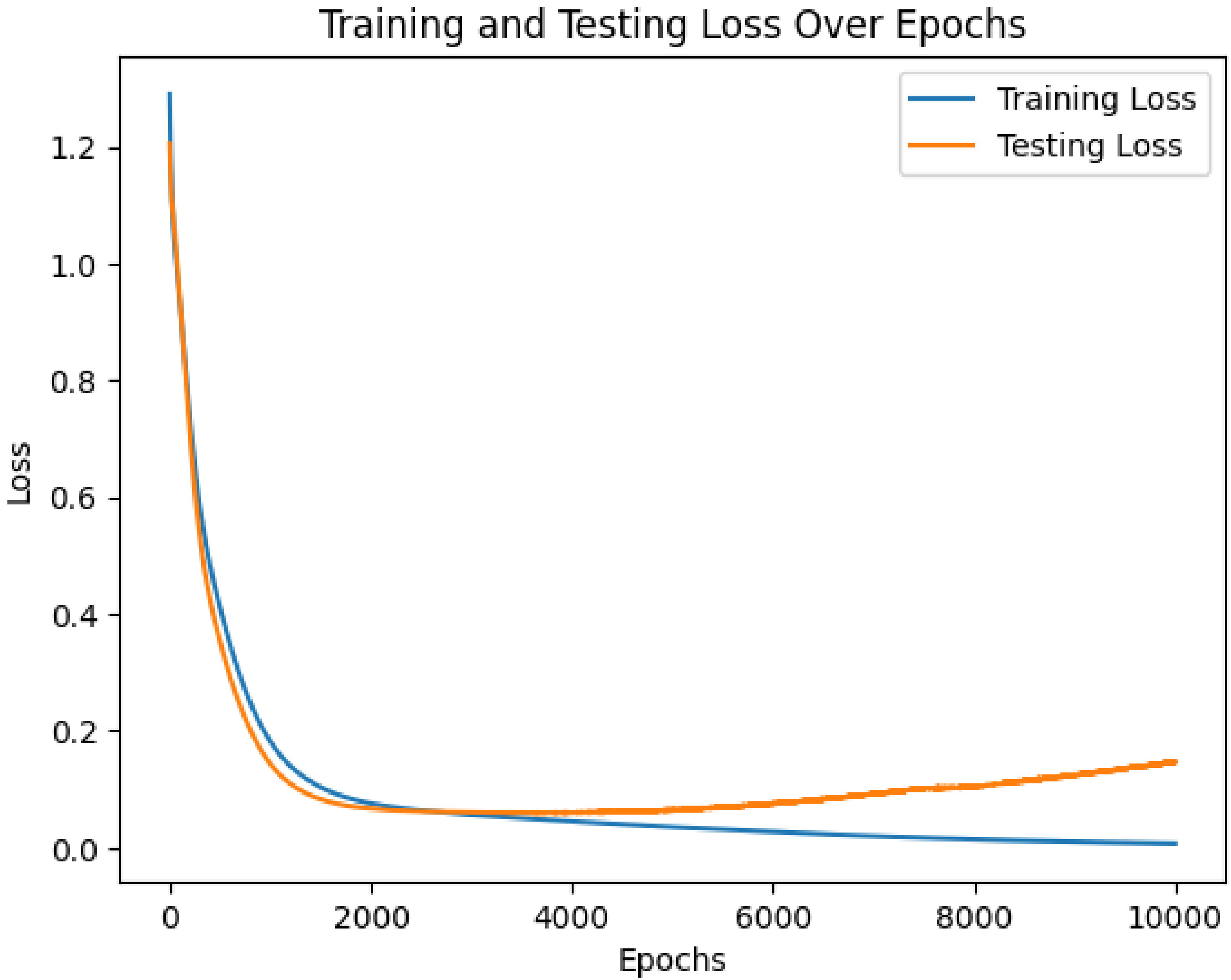
Train (nothing new!)

```python
for epoch in range(epochs):
    optimizer.zero_grad()
    outputs = model(X_train)
    loss = criterion(outputs, y_train)
    loss.backward()
    optimizer.step()
    train_losses.append(loss.item())

    # Evaluation step on testing set
    with torch.no_grad():
        test_outputs = model(X_test)
        test_loss = criterion(test_outputs, y_test)
        test_losses.append(test_loss.item())

    print(f'Epoch {epoch+1}/{epochs}, Training Loss: {loss.item()}, Test Loss: {test_loss.item()}')
```

Plot train/test loss (nothing new!)

```
# Plotting the training and testing losses over epochs
plt.plot(range(epochs), train_losses, label='Training Loss')
plt.plot(range(epochs), test_losses, label='Testing Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Training and Testing Loss Over Epochs')
plt.legend()
plt.show()
```

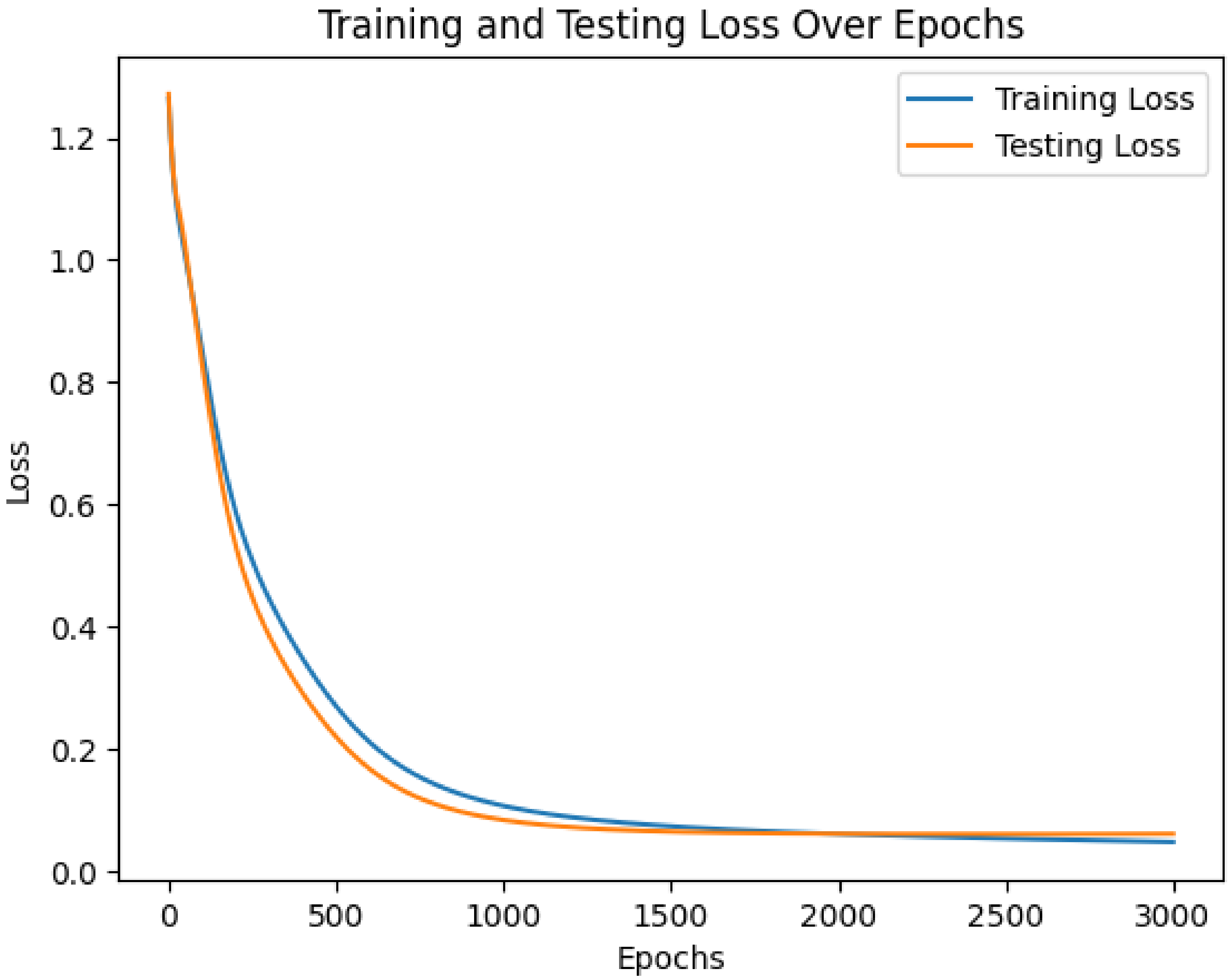
```
Epoch 1/10000, Training Loss: 1.290357232093811, Test Loss: 1.2053983211517334
Epoch 2/10000, Training Loss: 1.2780585289001465, Test Loss: 1.1973154544830322
Epoch 3/10000, Training Loss: 1.2661762237548828, Test Loss: 1.189612269401550З
Epoch 4/10000, Training Loss: 1.2547130584716797, Test Loss: 1.1822878122329712
Epoch 5/10000, Training Loss: 1.243670105934143, Test Loss: 1.1753385066986084
Epoch 6/10000, Training Loss: 1.2330485582351685, Test Loss: 1.1687607765197754
Epoch 7/10000, Training Loss: 1.22284734249115, Test Loss: 1.1625487804412842
Epoch 8/10000, Training Loss: 1.2130643129348755, Test Loss: 1.1566953659057617
Epoch 9/10000, Training Loss: 1.203696250915527З, Test Loss: 1.151192307472229
Epoch 10/10000, Training Loss: 1.1947381496429443, Test Loss: 1.1460297107696533
Epoch 11/10000, Training Loss: 1.1861834526062012, Test Loss: 1.1411958932876587
Epoch 12/10000, Training Loss: 1.1780245304107666, Test Loss: 1.13667869567871 1
Epoch 13/10000, Training Loss: 1.170252323150634 8, Test Loss: 1.13246381282806 4
Epoch 14/10000, Training Loss: 1.1628566980361938, Test Loss: 1.1285368204116821
Epoch 15/10000, Training Loss: 1.155826210975647, Test Loss: 1.1248811483383179
Epoch 16/10000, Training Loss: 1.149147868156433, Test Loss: 1.1214805841445923
Epoch 17/10000, Training Loss: 1.1428083181381226, Test Loss: 1.1183172464370728
Epoch 18/10000, Training Loss: 1.1367932558059692, Test Loss: 1.1153732538223267
Epoch 19/10000, Training Loss: 1.131087303161621, Test Loss: 1.1126306056976318
Epoch 20/10000, Training Loss: 1.1256749629974365, Test Loss: 1.11007106304168 7
Epoch 21/10000, Training Loss: 1.1205400228500366, Test Loss: 1.1076765060424805
Epoch 22/10000, Training Loss: 1.1156669855117798, Test Loss: 1.1054294109344482
Epoch 23/10000, Training Loss: 1.1110390424728394, Test Loss: 1.10331237316131 6
Epoch 24/10000, Training Loss: 1.1066404581069946, Test Loss: 1.1013092994689941
Epoch 25/10000, Training Loss: 1.102455735206604, Test Loss: 1.0994044542312622
...
Epoch 9997/10000, Training Loss: 0.007228388916701078, Test Loss: 0.1477985233068466 2
Epoch 9998/10000, Training Loss: 0.007248805370181799, Test Loss: 0.14795596446990 97
Epoch 9999/10000, Training Loss: 0.007262388709932566, Test Loss: 0.147318750619888 3
Epoch 10000/10000, Training Loss: 0.007249765563756227 5, Test Loss: 0.146829545497894 3
```


Training and Testing Loss Over Epochs
Training Loss
Testing Loss
Loss
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
2000
4000
6000
8000
10000
Epochs

Over-fitting

- We will use early stopping, stopping after 3,000 epochs.
- **Note**: Ideally, we would use a validation set to determine when to stop rather than the actual test error.

```
Epoch 1/3000, Training Loss: 1.2640759944915771, Test Loss: 1.27169930934906
Epoch 2/3000, Training Loss: 1.2523120641708374, Test Loss: 1.2588136196136475
Epoch 3/3000, Training Loss: 1.2410619258880615, Test Loss: 1.24654197692871I
Epoch 4/3000, Training Loss: 1.2302714586257935, Test Loss: 1.23495590686798I
Epoch 5/3000, Training Loss: 1.2199515104293823, Test Loss: 1.22402954101562S
Epoch 6/3000, Training Loss: 1.21060477256775, Test Loss: 1.213716387748718B
Epoch 7/3000, Training Loss: 1.2006133794784546, Test Loss: 1.2039904594421387
Epoch 8/3000, Training Loss: 1.1915860176086426, Test Loss: 1.1948440074920654
Epoch 9/3000, Training Loss: 1.1829493045806885, Test Loss: 1.1862380504608154
Epoch 10/3000, Training Loss: 1.1747429370880127, Test Loss: 1.17814540863037I
Epoch 11/3000, Training Loss: 1.1669117212295532, Test Loss: 1.1705280542373657
Epoch 12/3000, Training Loss: 1.1594367027282715, Test Loss: 1.16335022449493A
Epoch 13/3000, Training Loss: 1.1523357629776, Test Loss: 1.156582474708557I
Epoch 14/3000, Training Loss: 1.145584225654602, Test Loss: 1.1501802206039429
Epoch 15/3000, Training Loss: 1.1391065120697021, Test Loss: 1.1441161632537842
Epoch 16/3000, Training Loss: 1.1328951120376587, Test Loss: 1.1383739709854126
Epoch 17/3000, Training Loss: 1.1269614696502686, Test Loss: 1.1329452991485596
Epoch 18/3000, Training Loss: 1.121305227279663, Test Loss: 1.1278512477874756
Epoch 19/3000, Training Loss: 1.1158807277679443, Test Loss: 1.1230573654174805
Epoch 20/3000, Training Loss: 1.1106503009796143, Test Loss: 1.1185215711593628
Epoch 21/3000, Training Loss: 1.1056286096572876, Test Loss: 1.11419892311096Z
Epoch 22/3000, Training Loss: 1.100834846496582, Test Loss: 1.1010927757263184
Epoch 23/3000, Training Loss: 1.0962785482406616, Test Loss: 1.1065083742141724
Epoch 24/3000, Training Loss: 1.0920391082763672, Test Loss: 1.1030521392822266
Epoch 25/3000, Training Loss: 1.0880424976348877, Test Loss: 1.09978640079498E
...
Epoch 2997/3000, Training Loss: 0.04753931611776352, Test Loss: 0.06095428392291069
Epoch 2998/3000, Training Loss: 0.0475285910069942S, Test Loss: 0.0609563253819942S
Epoch 2999/3000, Training Loss: 0.04751782864332199, Test Loss: 0.06095853075385094
Epoch 3000/3000, Training Loss: 0.04750705510377884, Test Loss: 0.06096077710390091
```



Training and Testing Loss Over Epochs
Training Loss
Testing Loss
Loss
Epochs
1.2
1.0
0.8
0.6
0.4
0.2
0.0
0
500
1000
1500
2000
2500
3000

Is the model good?

- We have achieved a cross-entropy loss of roughly 0.06.
  - Is that good?
- Other evaluation metrics are often used to determine the quality of a mode.

Evaluation Metric: Accuracy

The accuracy is the proportion of correct predictions to the total number of predictions:

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions}}.$$

- While relatively simple, accuracy can be misleading if the class distribution is imbalanced.

```
Empirical probabilities of labels in the test set:
Label 0: 0.39
Label 1: 0.31
Label 2: 0.31
```

- In this case, 96% accuracy is decent!

```python
# Switch model to evaluation mode
model.eval()

# Calculate the number of correct predictions
with torch.no_grad():
    outputs = model(X_test)
    _, predicted = torch.max(outputs.data, 1)
    total = y_test.size(0)
    correct = (predicted == y_test).sum().item()

# Calculate accuracy
accuracy = 100 * correct / total
print(f'Accuracy on the test set: {accuracy:.2f}%')
```

✓ 0.0s

```
Accuracy on the test set: 96.00%
```

Evaluation Metric: Confusion Matrix

- Accuracy doesn't provide information about what kinds of errors are common
  - Which classes are often confused?
  - Important if some mistakes are worse than others.
- The **confusion matrix** provides this information. It is a matrix with one row per class and one column per class
  - The $(i, j)^{\text{th}}$ entry holds the probability that a row with actual class i is classified as class j.
  - In some cases the matrix reports the number of errors of each type, rather than the estimated probability.

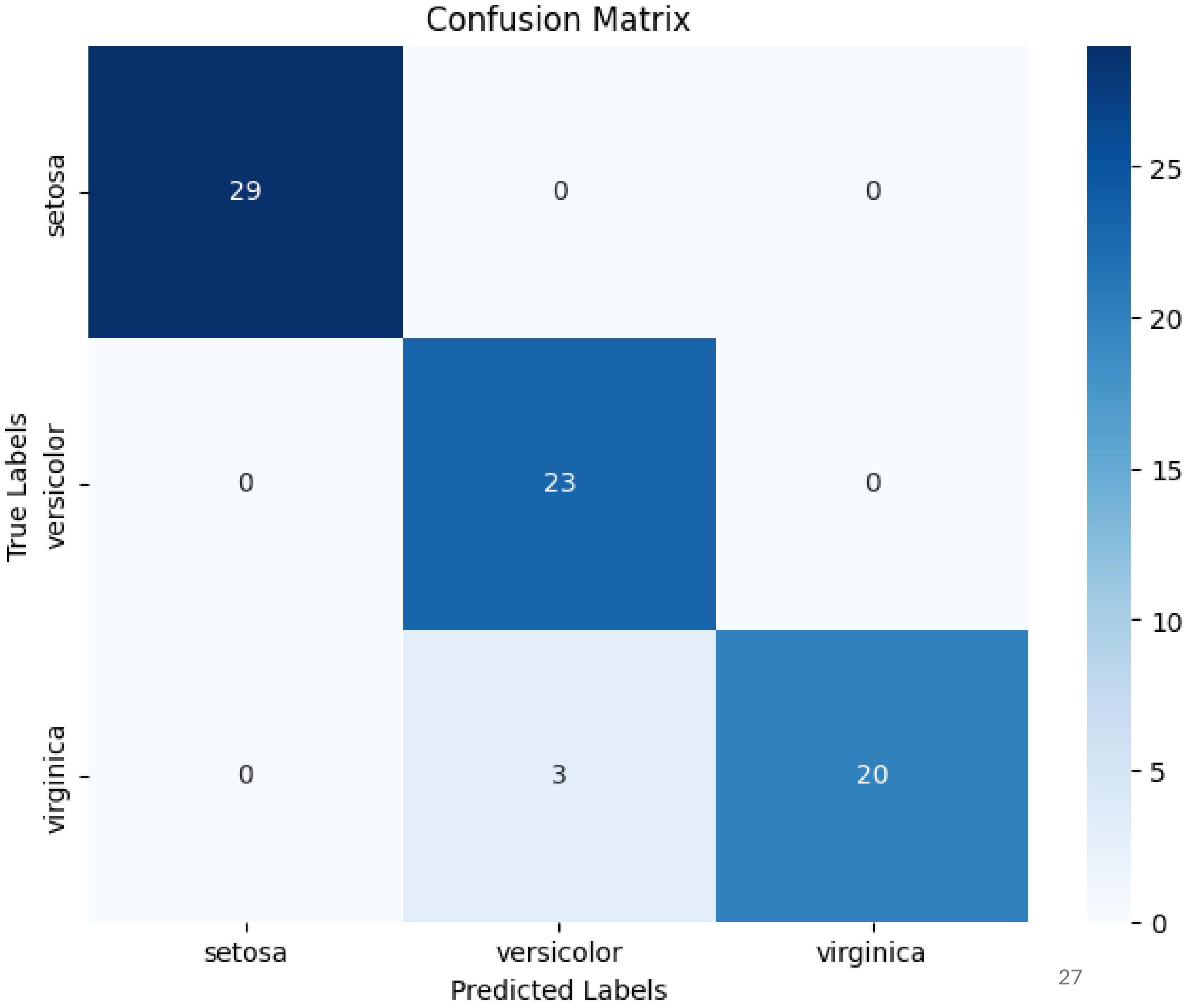
Confusion Matrix

```
# Get predictions
with torch.no_grad():
    outputs = model(X_test)
    _, predicted = torch.max(outputs, 1)

# Compute the confusion matrix
cm = confusion_matrix(y_test.numpy(), predicted.numpy())

# Plotting the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='g', cmap='Blues', xticklabels=iris.target_names, yticklabels=iris.target_names
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```

Deterministic model for evaluation (1 = dimension)

`confusion_matrix` comes from scikit-learn

Our model tends to misclassify "virginica" iris plants as "versicolor" iris plants.



Evaluation Metric: Precision, Recall, and F1 Score

- For *binary classification* tasks, statistics like **precision, recall,** and the **F1 score** are often used to evaluate models.
  - Note: These are often used even when the loss function used in training measures something else, like cross-entropy loss.
- These metrics are expressed in terms of the following statistics:

1. **True Positive (TP)**: The number of points (rows) with label 1 and where the model predicted 1.
2. **False Positive (FP)**: The number of points (rows) with label 0, but where the model predicted 1.
3. **False Negative (FN)**: The number of points (rows) with label 1, but where the model predicted 0.
4. **True Negative (TN)**: The number of points (rows) with label 0 and where the model predicted 0.

Deterministic Classifiers

Precision measures the ratio of the correctly predicted positive labels to the total predicted positives. That is:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

Deterministic Classifiers

Precision measures the ratio of the correctly predicted positive labels to the total predicted positives. That is:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}.$$

Recall measures the ratio of the correctly predicted positive labels to the total number of positives. That is:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

Stochastic Classifiers

$$\text{Precision} = \Pr(Y_i = 1 | \hat{Y}_i = 1),$$

$$\text{Recall} = \Pr(\hat{Y}_i = 1 | Y_i = 1).$$

F1 Score

- The F_1 score (often written "F1 score") combines precision and recall:

$$\mathrm{F}_1\ \mathrm{Score} = 2\,\frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

- This is the *harmonic mean* of the precision and recall
  - Places more weight on low values relative to the arithmetic mean
- F1 score ranges from 0 to 1, where 1 denotes perfect precision and recall, and 0 means that either precision or recall is zero.

Evaluation Metric: ROC

- The **receiver operating characteristic** (ROC) curve is a common metric for *binary* classification problems.
- Assumes that the single model output is compared to a threshold.
  - If the output is above the threshold, the prediction is 1 (positive)
  - If the output is below the threshold, the prediction is 0 (negative)
- Tuning the threshold can adjust the tradeoff between different types of errors
  - Too many false positives → increase the threshold
  - Too many false negatives → decrease the threshold
- Note: The model is still trained using a common loss function like cross-entropy loss!

Evaluation Metric: ROC

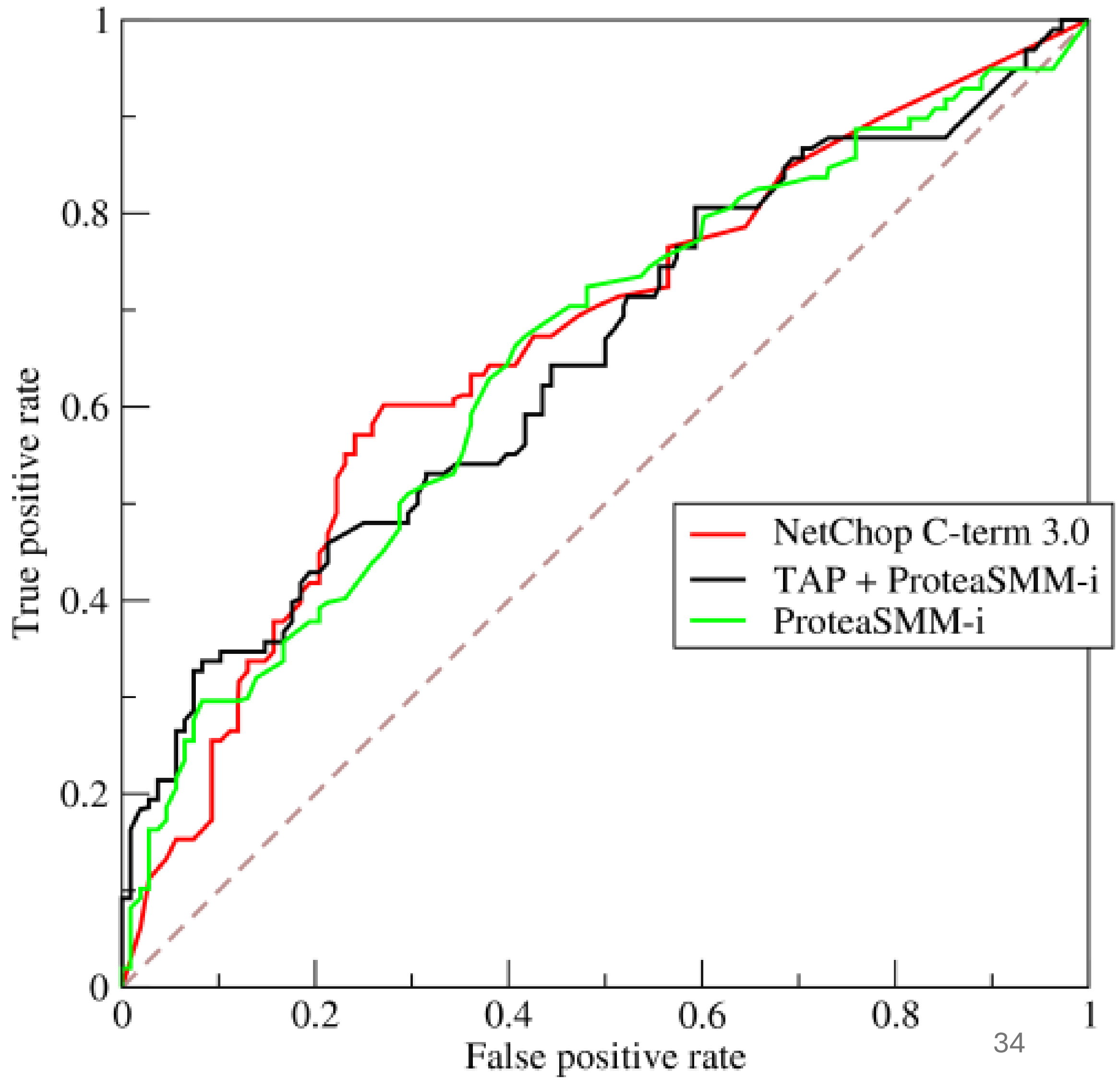
- The ROC curve is a plot of the false positive rate (FPR) and true positive rate (TPR) that a model achieves when the threshold is varied.

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP}+\mathrm{TN}}$$

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}}$$

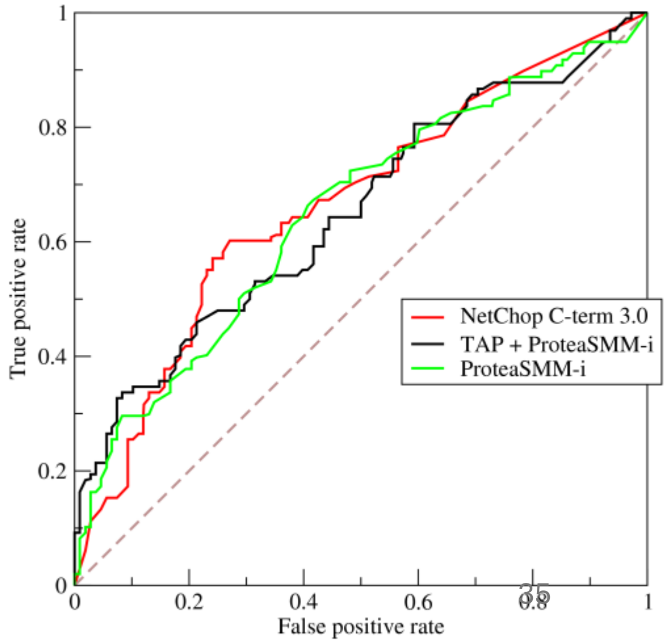
Example ROC Curve

- Curves closer to the top left corner correspond to better models.
- A classifier that ignores the inputs and outputs a uniform random number in $[0,1]$ results in a diagonal line from $(0,0)$ to $(1,1)$



Evaluation Metric: Area Under the ROC Curve (AUC)

- The AUC summarizes the ROC curve with a single number: The area under the ROC curve.
- The best possible value is 1.
- A pessimal model (one that always gets the prediction wrong) would have an AUC of zero.
- The random classifier achieves an AUC of 0.5



End