Artificial Neural Networks

Dr. Muhammad Aqib

University Institute of Information Technology

PMAS-Arid Agriculture University Rawalpindi

Network Error With Loss

Contents

What is Loss in NN?

- To train a model, we tweak the weights and biases to improve the model’s accuracy and confidence.
- To do this, we calculate how much error the model has.
- The loss function, also referred to as the cost function, is the algorithm that quantifies how wrong a model is. Loss is the measure of this metric.
- Loss is the model’s error, we ideally want it to be 0.

Categorical Cross-Entropy Loss

- If the model has softmax activation function for the output layer, it means it is outputting a probability distribution.
- Categorical cross-entropy is explicitly used to compare a “ground-truth” probability (y or “targets”) and some predicted distribution (y-hat or “predictions”).
- It is also one of the most commonly used loss functions with a softmax activation on the output layer.

Categorical Cross-Entropy Loss cont'd …

- The formula for calculating the categorical cross-entropy of y (actual/desired distribution) and y-hat (predicted distribution) is:

$$L_i = -\sum_j y_{i,j} log(\hat{y}_{i,j})$$

- Where Li denotes sample loss value, i is the i-th sample in the set, j is the label/output index, y denotes the target values, and y-hat denotes the predicted values.

Categorical Cross-Entropy Loss cont'd …

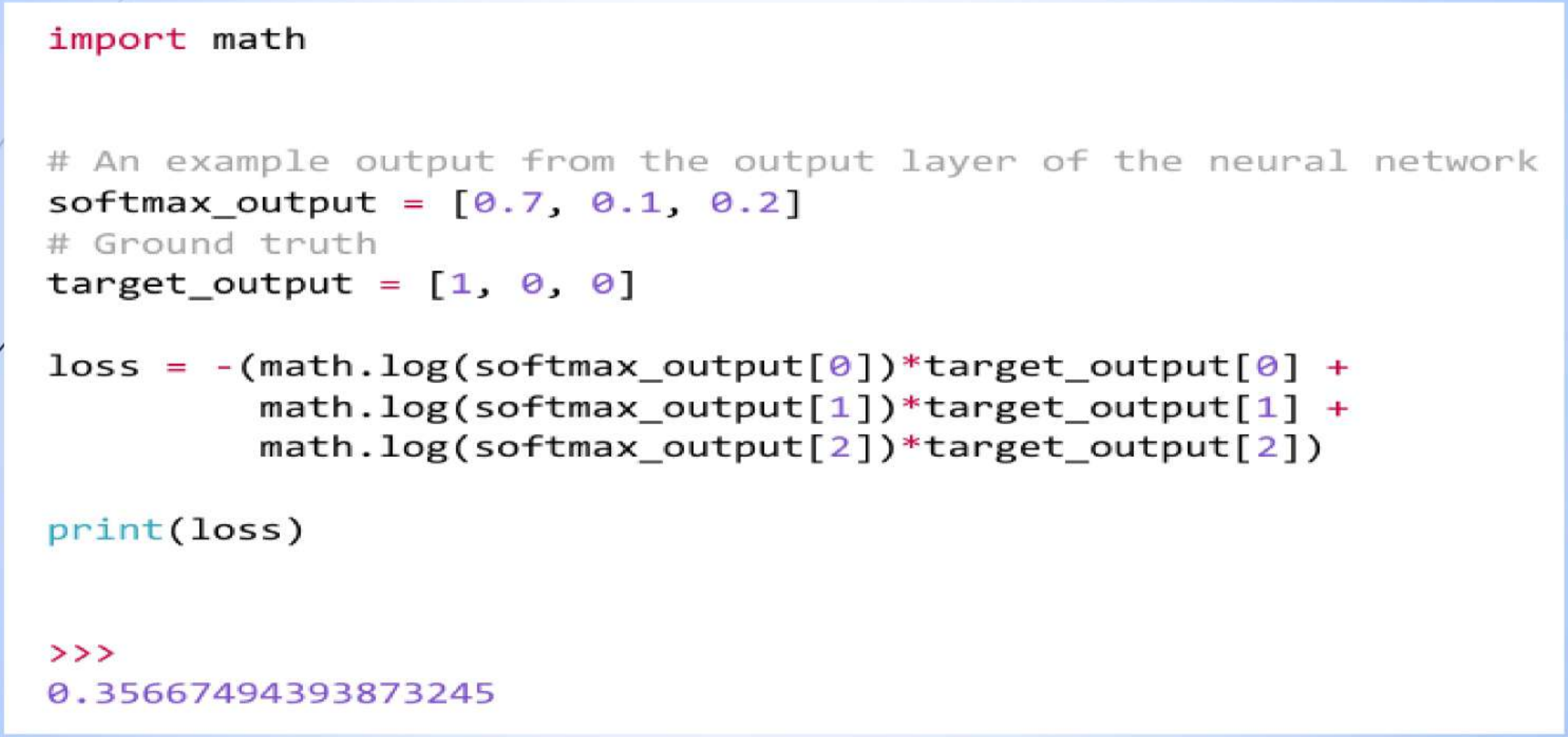
- Let's say a softmax output is: [0.7, 0.1, 0.2].
- We have 3 class confidences in the above output, and let's assume that the desired prediction is the first class (index 0, which is currently 0.7).
- If that's the intended prediction, then the desired probability distribution is [1, 0, 0].
- The desired probabilities will consist of a 1 in the desired class, and a 0 in the remaining undesired classes.
- Arrays or vectors like this are called one-hot, meaning one of the values is "hot" (on), with a value of 1, and the rest are "cold" (off), with values of 0.
- When comparing the model's results to a one-hot vector using cross-entropy, the other parts of the equation zero out, and the target probability's log loss is multiplied by 1, making the cross-entropy calculation relatively simple.
- This is also a special case of the cross-entropy calculation, called categorical cross-entropy.

Categorical Cross-Entropy Loss cont'd …

- To exemplify — if we take a softmax output of [0.7, 0.1, 0.2] and targets of [1, 0, 0], we can apply the calculations as follows:

$$L_i = -\sum_j y_{i,j} log(\hat{y}_{i,j}) = -(1 \cdot log(0.7) + 0 \cdot log(0.1) + 0 \cdot log(0.2)) =$$
$$= -(-0.35667494393873245 + 0 + 0) = 0.35667494393873245$$

Categorical Cross-Entropy Loss cont'd …



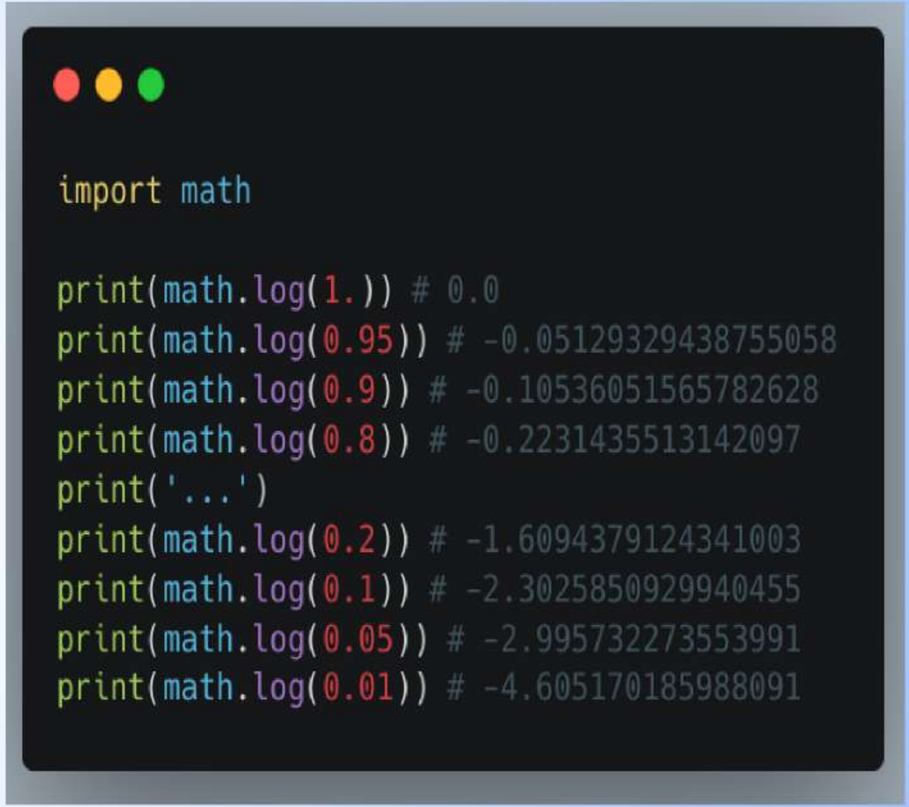
```
import math


# An example output from the output layer of the neural network
softmax_output = [0.7, 0.1, 0.2]
# Ground truth
target_output = [1, 0, 0]

loss = -(math.log(softmax_output[0])*target_output[0] +
         math.log(softmax_output[1])*target_output[1] +
         math.log(softmax_output[2])*target_output[2])

print(loss)


>>>
0.35667494393873245
```

Categorical Cross-Entropy Loss cont'd …

- Let's suppose the example confidence level might look like [0.22, 0.6, 0.18] or [0.32, 0.36, 0.32].
- In both cases, the argmax of these vectors will return the second class as the prediction, but the model's confidence about these predictions is high only for one of them.
- The Categorical Cross-Entropy Loss accounts for that and outputs a larger loss the lower the confidence. Example is given next.

Categorical Cross-Entropy Loss cont'd …

- When the confidence level equals 1, meaning the model is 100% "sure" about its prediction, the loss value for this sample equals 0.
- The loss value raises with the confidence level, approaching 0.



```
import math

print(math.log(1.)) # 0.0
print(math.log(0.95)) # -0.05129329438755058
print(math.log(0.9)) # -0.10536051565782628
print(math.log(0.8)) # -0.2231435513142097
print('...')
print(math.log(0.2)) # -1.6094379124341003
print(math.log(0.1)) # -2.3025850929940455
print(math.log(0.05)) # -2.995732273553991
print(math.log(0.01)) # -4.605170185988091
```

Categorical Cross-Entropy Loss cont'd …

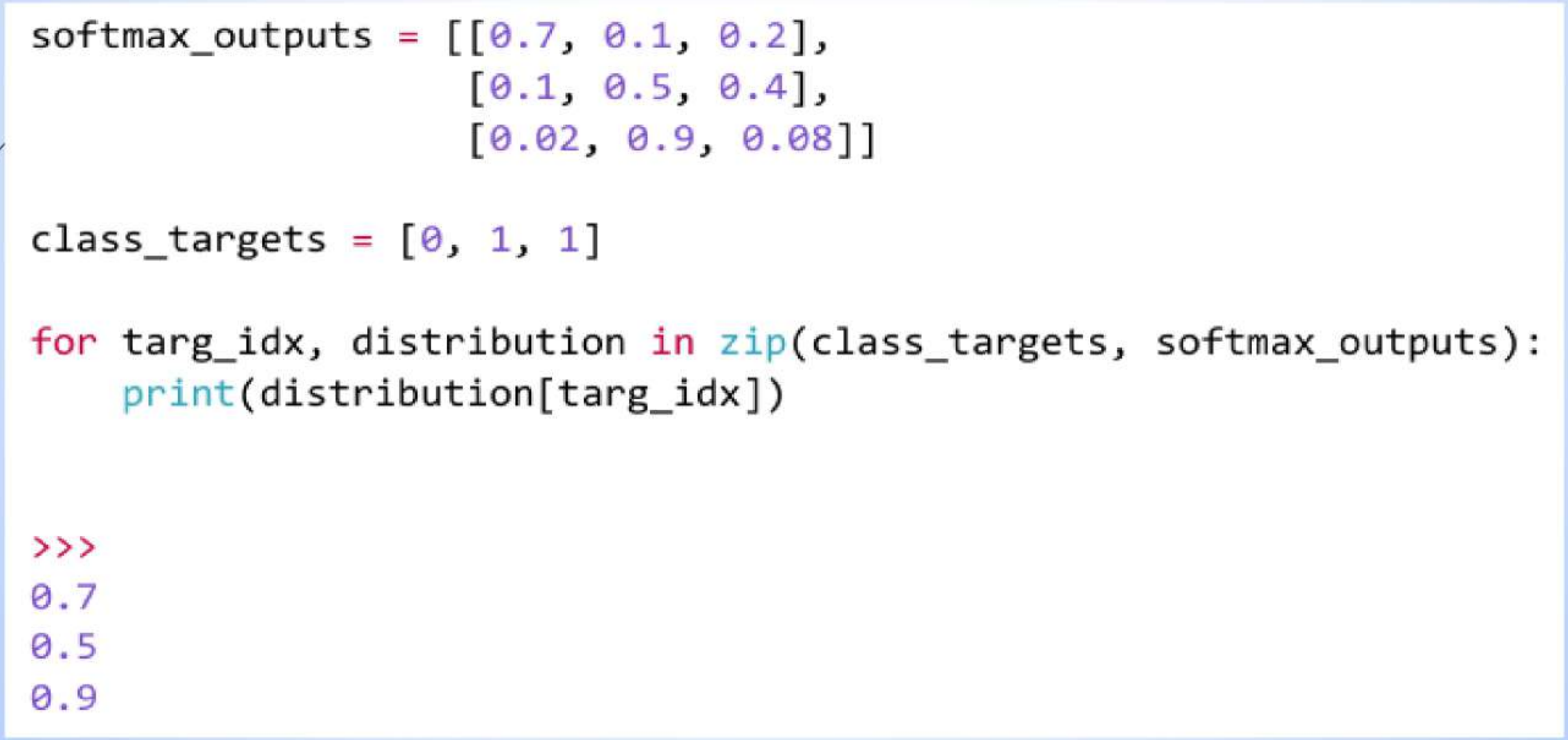
- Let's suppose the example confidence level might look like [0.22, 0.6, 0.18] or [0.32, 0.36, 0.32].
- Consider a scenario with a neural network that performs classification between three classes, and the neural network classifies in batches of three. After running through the softmax activation function with a batch of 3 samples and 3 classes, the network's output layer yields:
- softmax_outputs = np.array([[0.7, 0.1, 0.2], [0.1, 0.5, 0.4], [0.02, 0.9, 0.08]])
- We need to calculate the categorical cross-entropy, which we now know is a negative log calculation.
- To determine which value in the softmax output to calculate the negative log from, we simply need to know our target values.

Categorical Cross-Entropy Loss cont’d …

- In this example, there are 3 classes; let’s say we’re trying to classify something as a “dog,” “cat,” or “human.” A dog is class 0 (at index 0), a cat class 1 (index 1), and a human class 2 (index 2).
- Let’s assume the batch of three sample inputs to this neural network is being mapped to the target values of a dog, cat, and cat. So the targets (as a list of target indices) would be [0, 1, 1].
- In both cases, the argmax of these vectors will return the second class as the prediction, but the model’s confidence about these predictions is high only for one of them.
- The Categorical Cross-Entropy Loss accounts for that and outputs a larger loss the lower the confidence. Example is given next.

Categorical Cross-Entropy Loss cont'd …

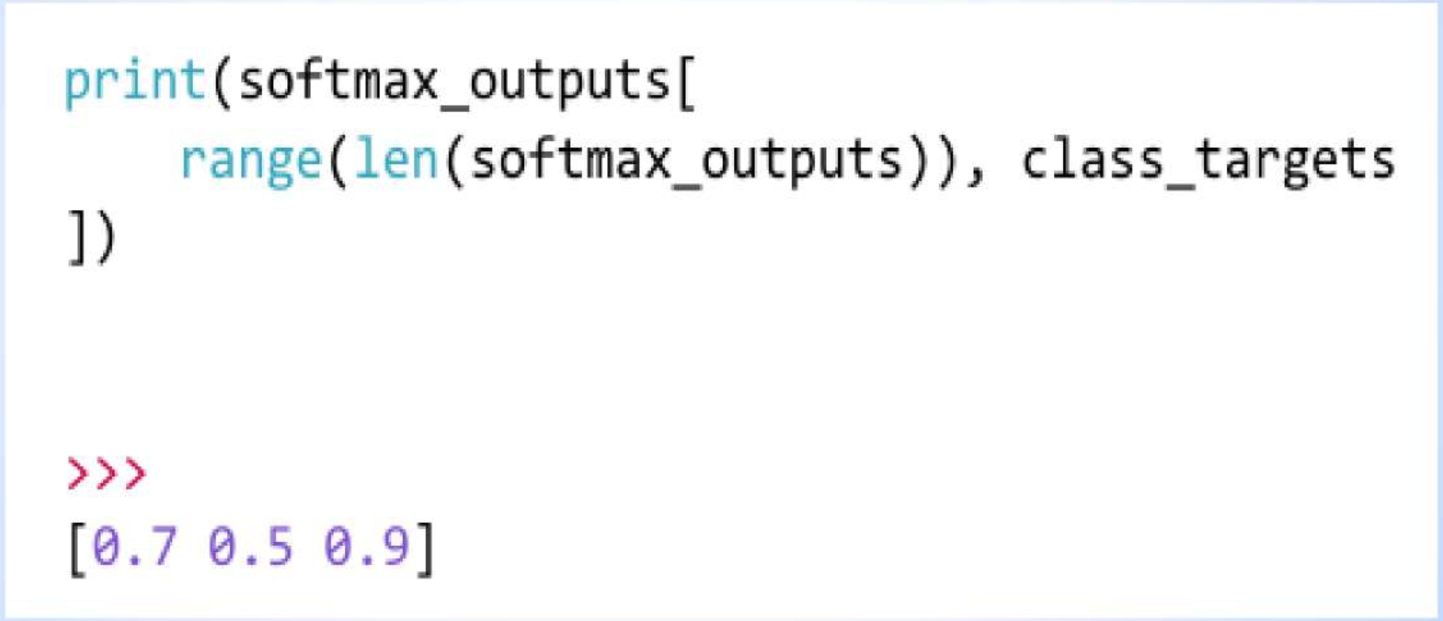
- With a collection of softmax outputs and their intended targets, we can map these indices to retrieve the values from the softmax distributions:



```
softmax_outputs = [[0.7, 0.1, 0.2],
                   [0.1, 0.5, 0.4],
                   [0.02, 0.9, 0.08]]

class_targets = [0, 1, 1]

for targ_idx, distribution in zip(class_targets, softmax_outputs):
    print(distribution[targ_idx])



>>>
0.7
0.5
0.9
```

Categorical Cross-Entropy Loss cont'd …

- We know we're going to have as many indices as distributions in our entire batch, so we can use a range() instead of typing each value ourselves:

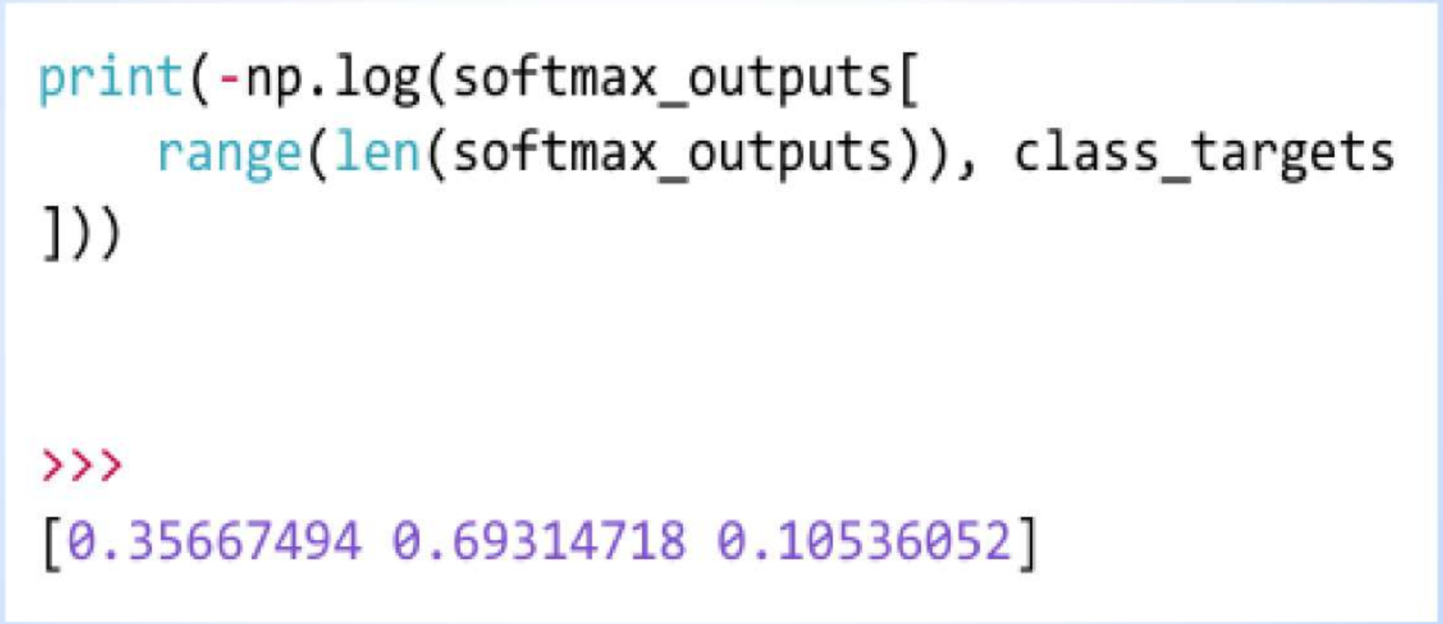


```
print(softmax_outputs[
    range(len(softmax_outputs)), class_targets
])


>>>
[0.7 0.5 0.9]
```

- This returns a list of the confidences at the target indices for each of the samples.

Categorical Cross-Entropy Loss cont’d …

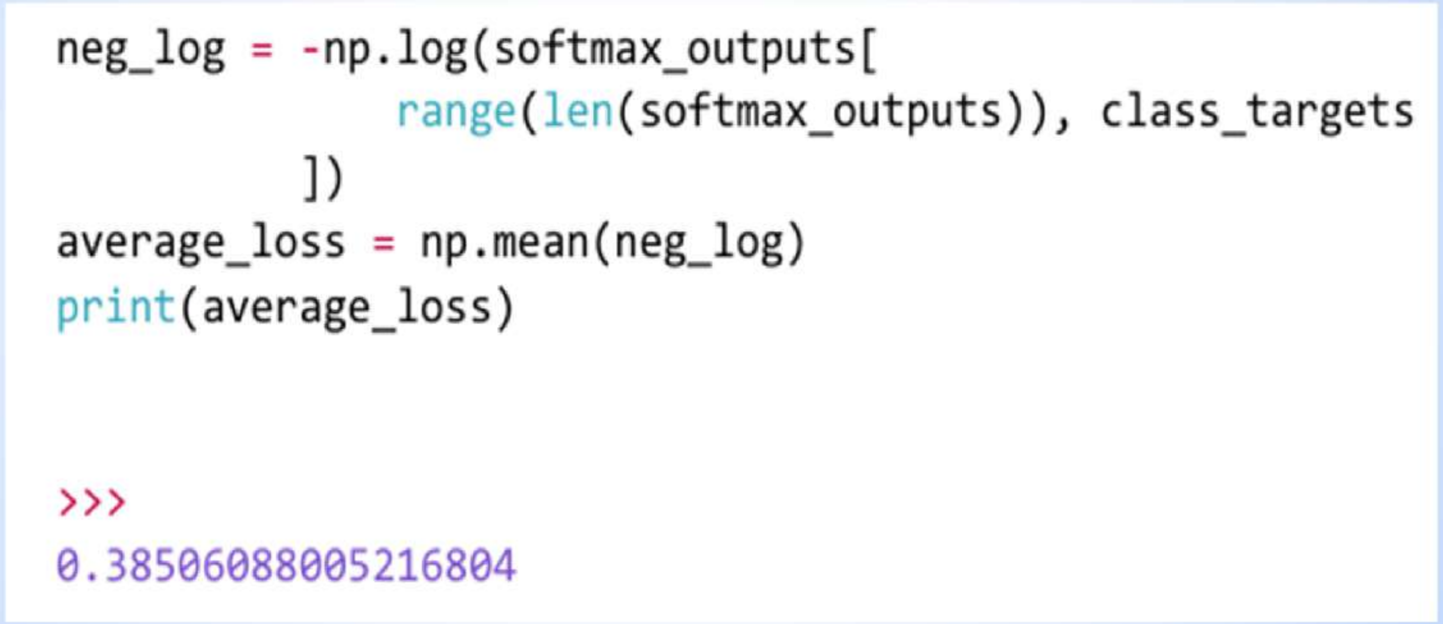
- Now we apply the negative log to this list:



```
print(-np.log(softmax_outputs[
    range(len(softmax_outputs)), class_targets
]))



>>>
[0.35667494 0.69314718 0.10536052]
```

Categorical Cross-Entropy Loss cont'd …

- Finally, we want an average loss per batch to have an idea about how our model is doing during training. We add NumPy's average to the code:



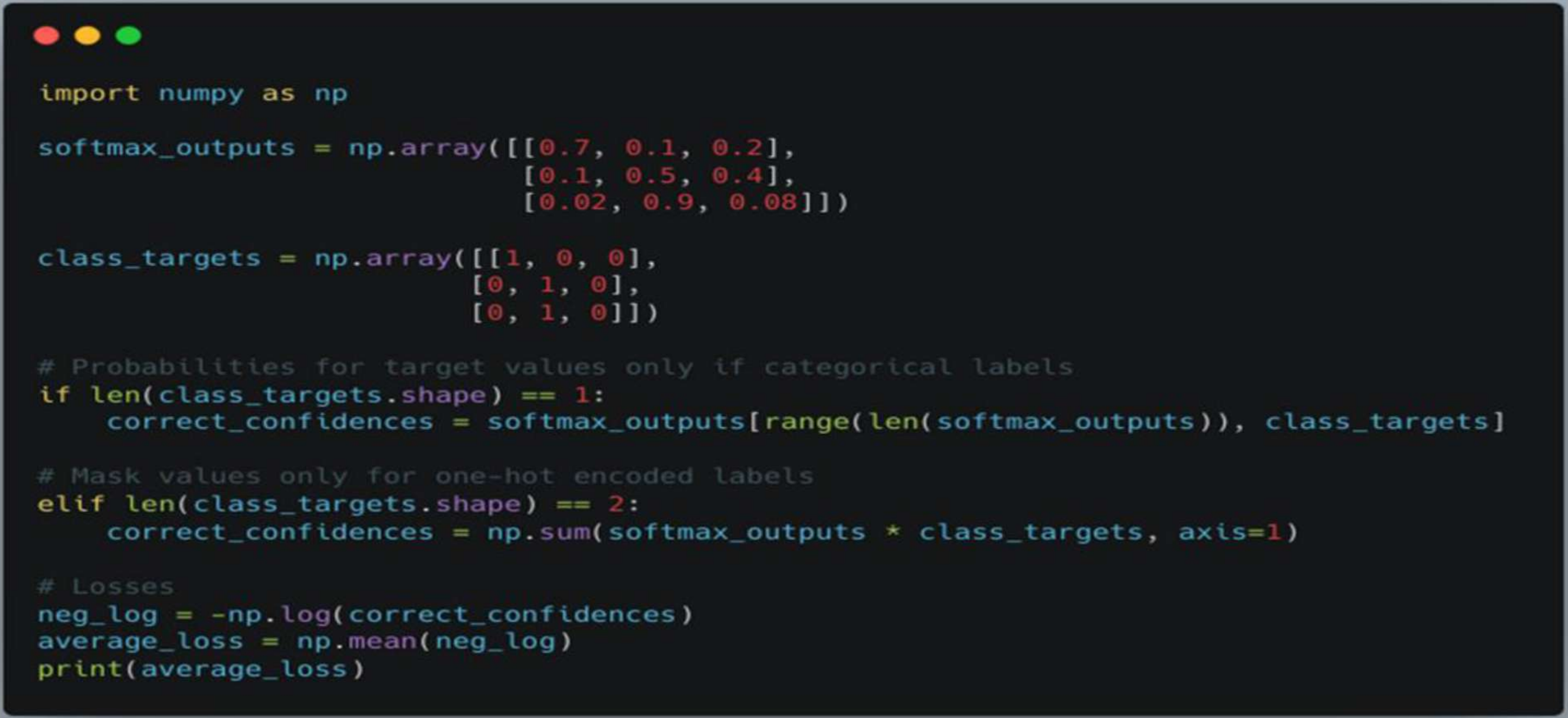
```
neg_log = -np.log(softmax_outputs[
                range(len(softmax_outputs)), class_targets
            ])
average_loss = np.mean(neg_log)
print(average_loss)
```

```
>>>
0.38506088005216804
```

Categorical Cross-Entropy Loss cont’d …

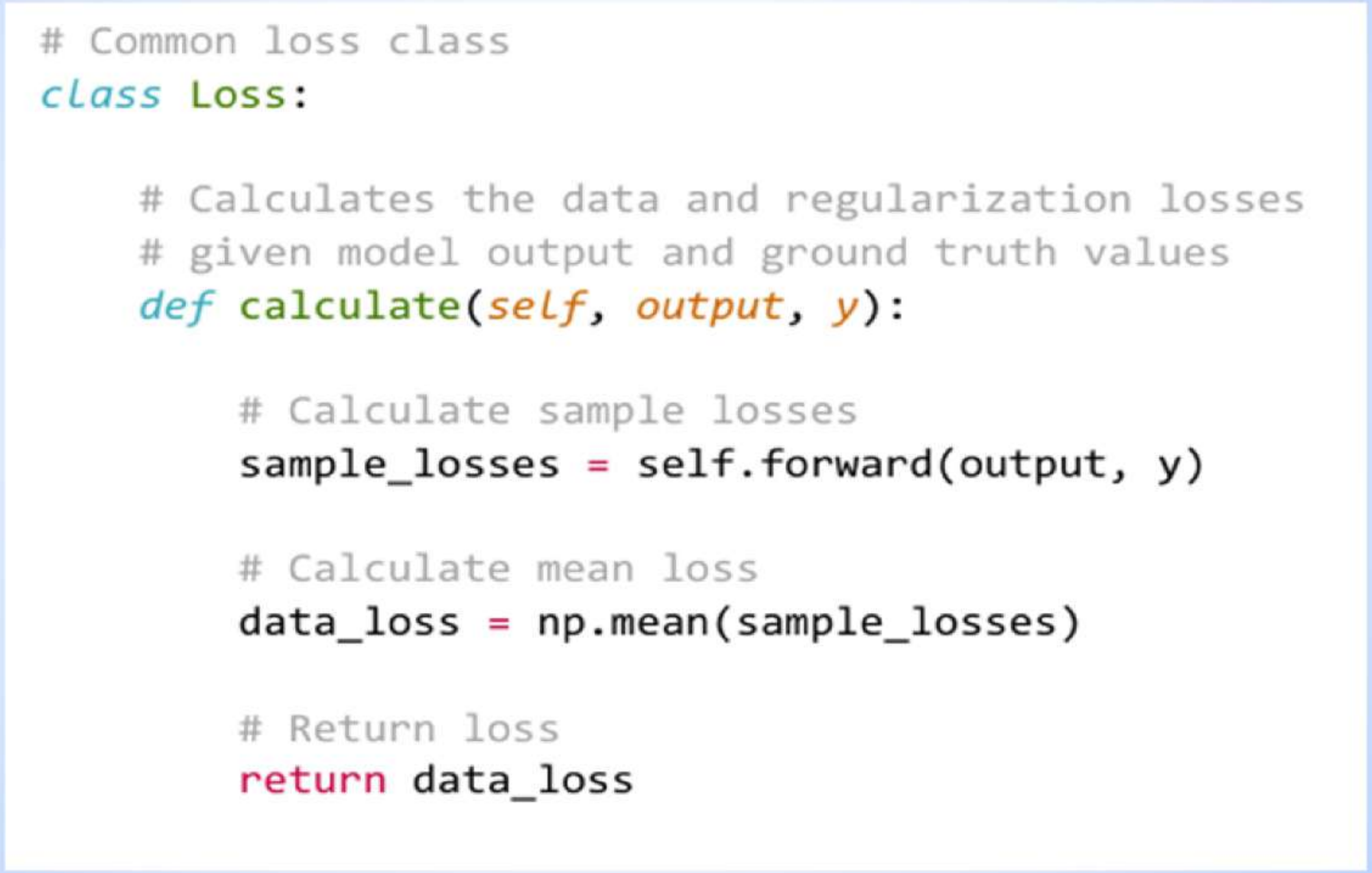
- We have already learned that targets can be one-hot encoded, where all values, except for one, are zeros, and the correct label’s position is filled with 1.
- We have to multiply confidences by the targets, zeroing out all values except the ones at correct labels, performing a sum along the row axis.
- Code for this, is given next!

Categorical Cross-Entropy Loss cont'd …



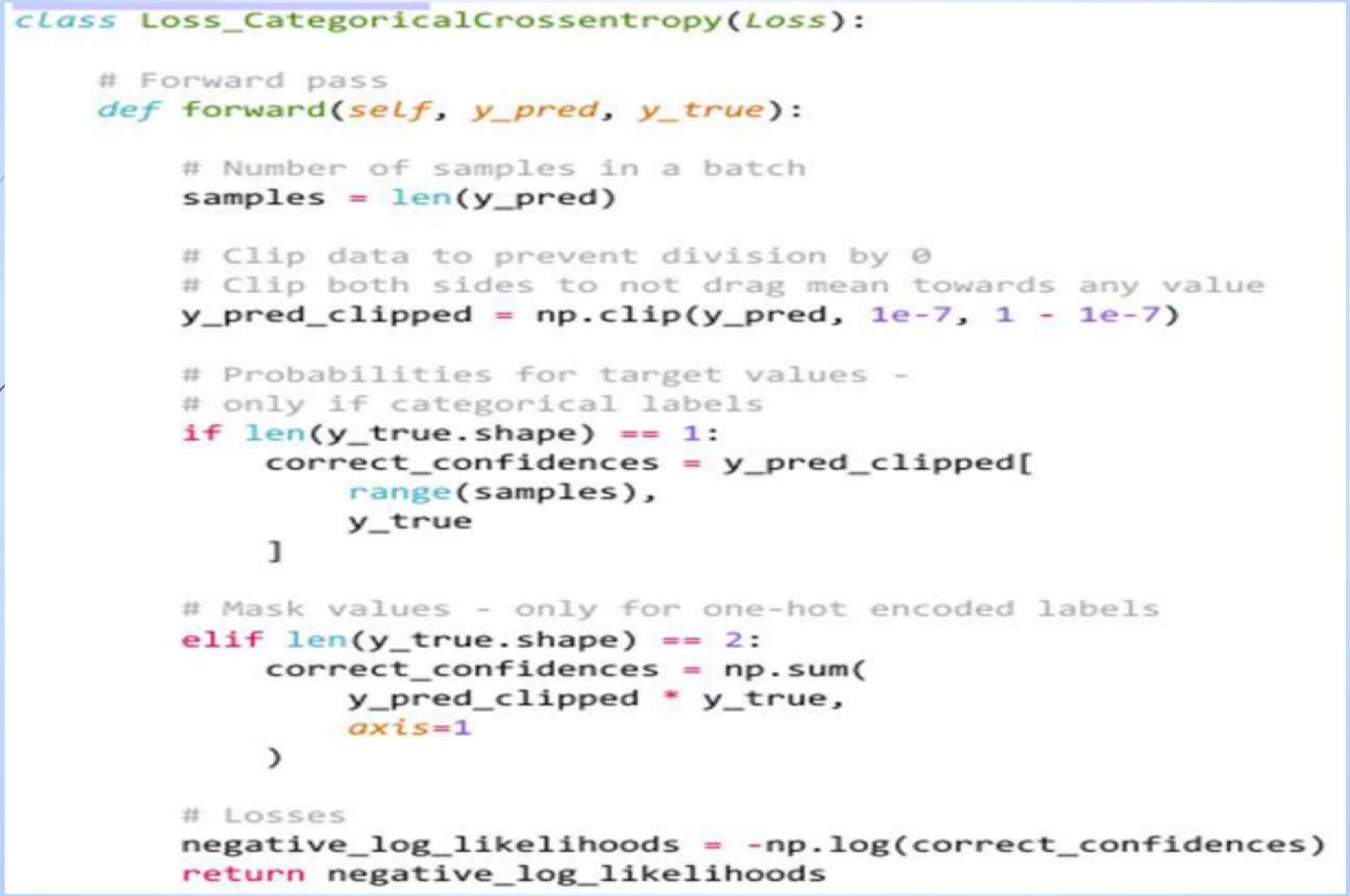
```
import numpy as np

softmax_outputs = np.array([[0.7, 0.1, 0.2],
                            [0.1, 0.5, 0.4],
                            [0.02, 0.9, 0.08]])

class_targets = np.array([[1, 0, 0],
                          [0, 1, 0],
                          [0, 1, 0]])

# Probabilities for target values only if categorical labels
if len(class_targets.shape) == 1:
    correct_confidences = softmax_outputs[range(len(softmax_outputs)), class_targets]

# Mask values only for one-hot encoded labels
elif len(class_targets.shape) == 2:
    correct_confidences = np.sum(softmax_outputs * class_targets, axis=1)

# Losses
neg_log = -np.log(correct_confidences)
average_loss = np.mean(neg_log)
print(average_loss)
```

Categorical Cross-Entropy Loss Classes



```python
# Common loss class
class Loss:

    # Calculates the data and regularization losses
    # given model output and ground truth values
    def calculate(self, output, y):

        # Calculate sample losses
        sample_losses = self.forward(output, y)

        # Calculate mean loss
        data_loss = np.mean(sample_losses)

        # Return loss
        return data_loss
```

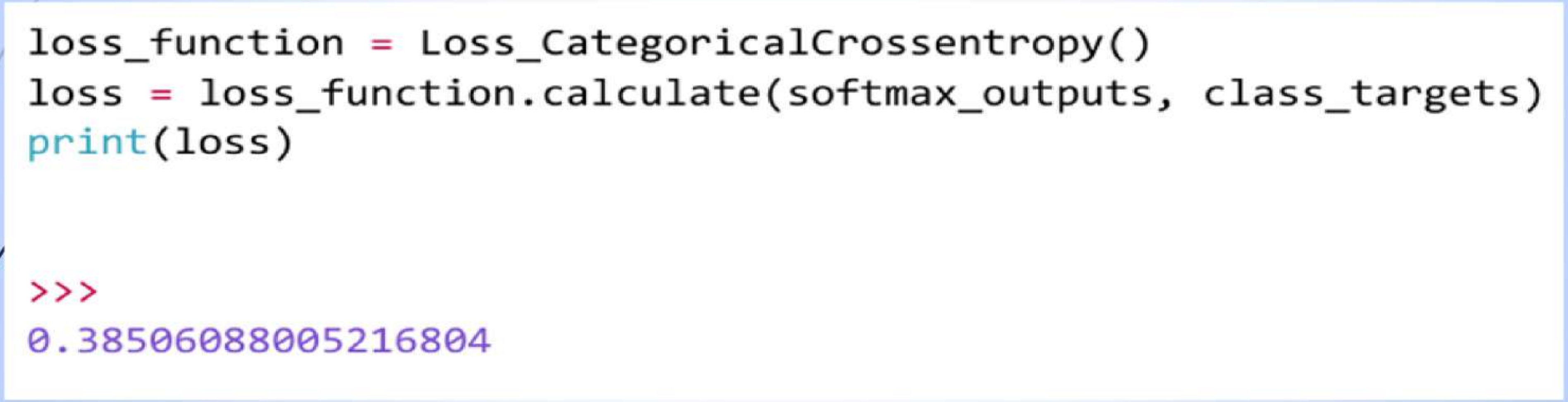
Categorical Cross-Entropy Loss Classes



```
class Loss_CategoricalCrossentropy(Loss):

    # Forward pass
    def forward(self, y_pred, y_true):

        # Number of samples in a batch
        samples = len(y_pred)

        # Clip data to prevent division by 0
        # Clip both sides to not drag mean towards any value
        y_pred_clipped = np.clip(y_pred, 1e-7, 1 - 1e-7)

        # Probabilities for target values -
        # only if categorical labels
        if len(y_true.shape) == 1:
            correct_confidences = y_pred_clipped[
                range(samples),
                y_true
            ]

        # Mask values - only for one-hot encoded labels
        elif len(y_true.shape) == 2:
            correct_confidences = np.sum(
                y_pred_clipped * y_true,
                axis=1
            )

        # Losses
        negative_log_likelihoods = -np.log(correct_confidences)
        return negative_log_likelihoods
```

Categorical Cross-Entropy Loss Classes

- For example, using the manually-created output and targets, code can be called as:



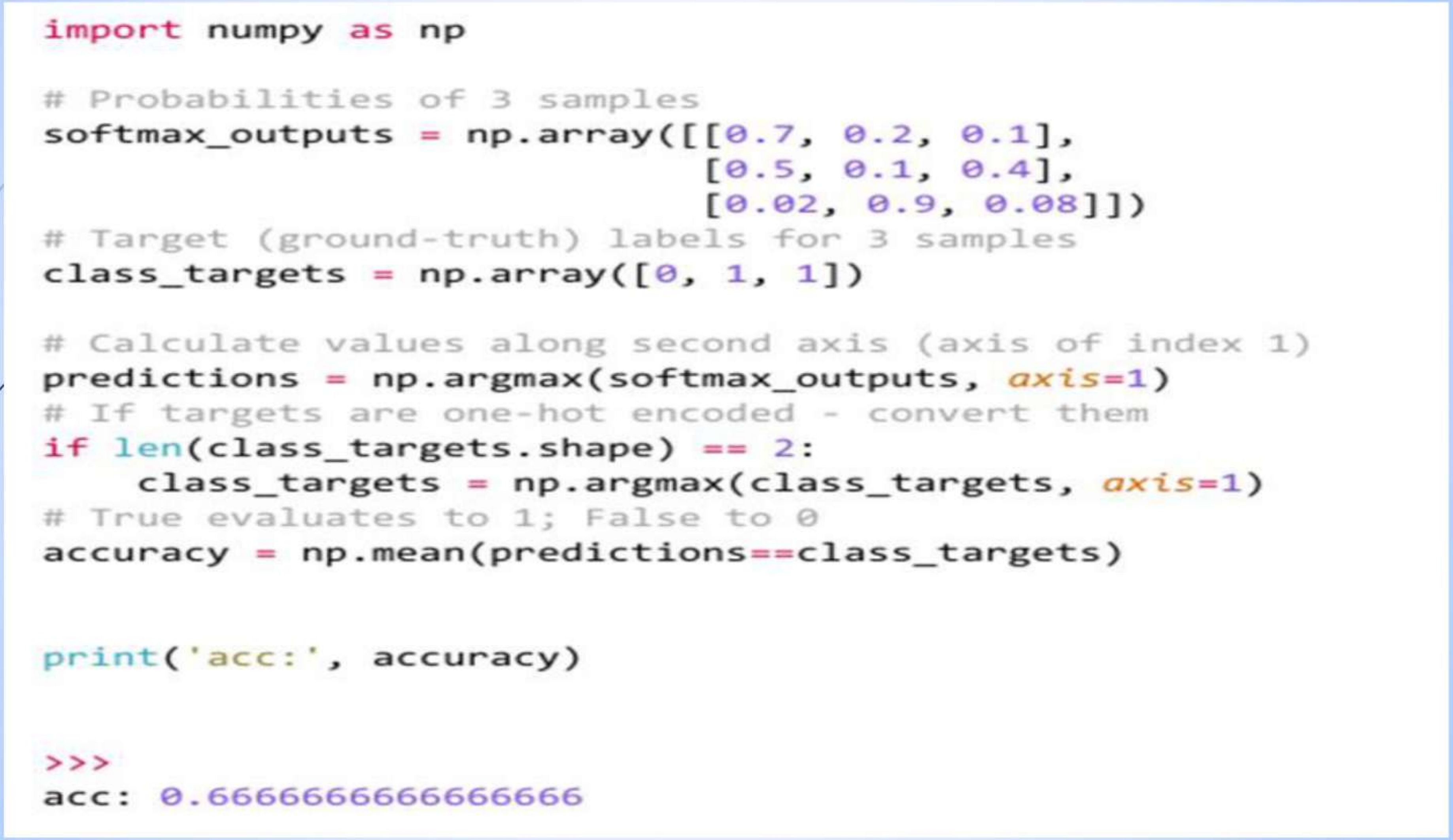
```
loss_function = Loss_CategoricalCrossentropy()
loss = loss_function.calculate(softmax_outputs, class_targets)
print(loss)
```

```
>>>
0.38506088005216804
```

Accuracy Calculation

- The metric commonly used in practice along with loss is the accuracy, which describes how often the largest confidence is the correct class in terms of a fraction.
- We will use the argmax values from the softmax outputs and then compare these to the targets.
- Code is given next!

Accuracy Calculation



```
import numpy as np

# Probabilities of 3 samples
softmax_outputs = np.array([[0.7, 0.2, 0.1],
                            [0.5, 0.1, 0.4],
                            [0.02, 0.9, 0.08]])
# Target (ground-truth) labels for 3 samples
class_targets = np.array([0, 1, 1])

# Calculate values along second axis (axis of index 1)
predictions = np.argmax(softmax_outputs, axis=1)
# If targets are one-hot encoded - convert them
if len(class_targets.shape) == 2:
    class_targets = np.argmax(class_targets, axis=1)
# True evaluates to 1; False to 0
accuracy = np.mean(predictions==class_targets)


print('acc:', accuracy)


>>>
acc: 0.6666666666666666
```

Conclusion

- Calculating Loss and Accuracy can help you give insights into the performance of model.
- If the accuracy is low and loss is pretty much higher, then you are definitely going to optimize it.

Thank You