Jeramee Oliver

Assignment 6

MSAI 531 A01

Neural Networks Deep Learning

9/22/24

Assignment 6: Learn via Gradient Descent

Objective of the Assignment:

This task is all about teaching a neural network to classify digits (0-9) and testing different loss functions to see which one works best. Basically, the network spits out probabilities for each digit, and the one with the highest probability gets picked as the prediction. Let's dive into the nitty-gritty of how these loss functions behave.

Analysis of Loss Functions:

1. Square of the Difference:

- **Formula:**

$$Loss=(y - \hat{y})^2$$

- **Variables:**
  - y: Correct digit (ground truth).
  - $\hat{y}$: Predicted digit by the network (treated as a single numeric value).
- **Explanation**:

  This loss function squares the difference between the correct digit and the predicted one... like how you'd measure how far off you are from a target. It's a straightforward idea, but when it comes to classifying digits, it's not really ideal. The problem is that it assumes both the true value (y) and the predicted value ($\hat{y}$) are numeric values. But neural networks don't just predict single numbers—they deal with probabilities across different classes. So, this approach doesn't fit well for classification tasks.

- **Suitability:**
  Not suitable for classification tasks because it does not account for the probabilistic nature of the output. It fails to provide meaningful feedback for a multi-class classification problem.

Python Code:

```
# Function to calculate the square of the difference (squared error loss)
def squared_difference_loss(y_true, y_pred):
  return (y_true - y_pred) ** 2
```

Gradient Calculation:

Not suitable but I implemented this for negative log-probability calculations.

```
    # Using TensorFlow's GradientTape
  with tf.GradientTape() as tape:
    # Watch the variables we want to compute gradients with respect to
    tape.watch(predicted_probabilities)

    # Calculate the losses
    squared_loss = squared_difference_loss(correct_digit, predicted_digit)
```

2. Probability of the Correct Digit:

- **Formula:**

$$P(y) = \hat{p}_y$$

- **Variables:**
  - ***P(y)***: Probability of the correct digit as predicted by the network.
  - $\hat{p}_y$**:** Predicted probability assigned to the correct class y.
- **Explanation:**

  This is all about how confident the network is when it guesses the correct digit. It's like asking, "Okay, how sure are you about this?"… a measure of confidence, if you will. However, while it's great to know the network is confident, it doesn't actually measure how wrong it was when it gets messed up. You'd still need something else to figure out how far off you are from the right answer which is our next formula.
- **Suitability:**

  Not suitable as a loss function. Although it expresses confidence, it is devoid of the attributes required to properly direct learning and supply gradient information for optimization.

Python Code:

```
# Function to calculate the probability of the correct digit (not a true loss
function)
def probability_loss(y_true, probabilities):
  return probabilities[y_true]
```

Gradient Calculation:

Not suitable

3. Negative Log-Probability (Cross-Entropy Loss):

- **Formula:**

$$Negative\log-Probability \;=\; -\log\left(\hat{p}_y\right)$$

- **Variables:**
  - $\hat{p}_y$: Predicted probability of the correct digit.
- **Explanation:** This is the big one, often called "cross-entropy loss." It takes the predicted probabilities and compares them to what they should be—a 1 for the correct class, and 0 for everything else. It punishes bad guesses hard and rewards good ones. The more off-target the prediction, the bigger the penalty. On the flip side... the closer it gets to the right answer, the better.
- **Suitability:**

  Suitable this is the bread and butter for classification tasks. Its smooth nature facilitates gradient descent in adjusting the network's weights to improve predictions. It's like a roadmap for how wrong the network is and how to fix it.

Python Code:

```
# Function to calculate the negative log-probability (cross-entropy loss)
def negative_log_probability_loss(y_true, probabilities):
  return -tf.math.log(probabilities[y_true])
```

Gradient Calculation:

```
    # Using TensorFlow's GradientTape
  with tf.GradientTape() as tape:
    # Watch the variables we want to compute gradients with respect to
    tape.watch(predicted_probabilities)

    # Calculate the losses
    neg_log_prob_loss = negative_log_probability_loss(correct_digit,
predicted_probabilities)
```

Example Calculation:

Consider the scenario where the predicted probabilities for digits 0 through 9 are:

$$\hat{p}_y = [0.05, 0.1, 0.2, 0.1, 0.1, 0.3, 0.1, 0.05, 0.05, 0.05]$$

And the correct digit $y = 5$.

1. **Squared Loss**:
   If the predicted digit is $\hat{y} = 4$:

   $$\text{Loss} = (5 - 4)^2 = 1^2 = 1$$
2. **Probability of the Correct Digit**:

   $$P(y) = \hat{p}_5 = 0.3$$
3. **Negative Log-Probability**:

   $$\log(0.3) \approx -1.204$$

   Thus:

   $$Negative\log - Probability = -(-1.204) \approx 1.204$$

Sample Output:

```
Example Data 1:
Predicted Probabilities:
[0.02       0.03       0.1        0.15       0.05       0.05
 0.1        0.40999994 0.05       0.15      ]
Squared Difference Loss: 4
Negative Log-Probability Loss: 0.9162907004356384
Gradient of Negative Log-Probability Loss w.r.t. probabilities:
[ 0.   0.   0.   0.   0.   0.   0.  -2.5  0.   0. ]

Example Data 2:
Predicted Probabilities:
[0.1        0.05       0.2        0.30999994 0.1        0.1
 0.05       0.05       0.02       0.03      ]
Squared Difference Loss: 1
Negative Log-Probability Loss: 1.2039728164672852
Gradient of Negative Log-Probability Loss w.r.t. probabilities:
[ 0.         0.         0.        -3.3333333  0.         0.
  0.         0.         0.         0.       ]

Example Data 3:
Predicted Probabilities:
[0.05       0.1        0.2        0.1        0.1        0.30999994
 0.1        0.05       0.05       0.05      ]
Squared Difference Loss: 1
Negative Log-Probability Loss: 1.2039728164672852
Gradient of Negative Log-Probability Loss w.r.t. probabilities:
[ 0.         0.         0.         0.         0.        -3.3333333
  0.         0.         0.         0.       ]
```