CS4100 Problem Set 3

Silas Nevstad

August 2024

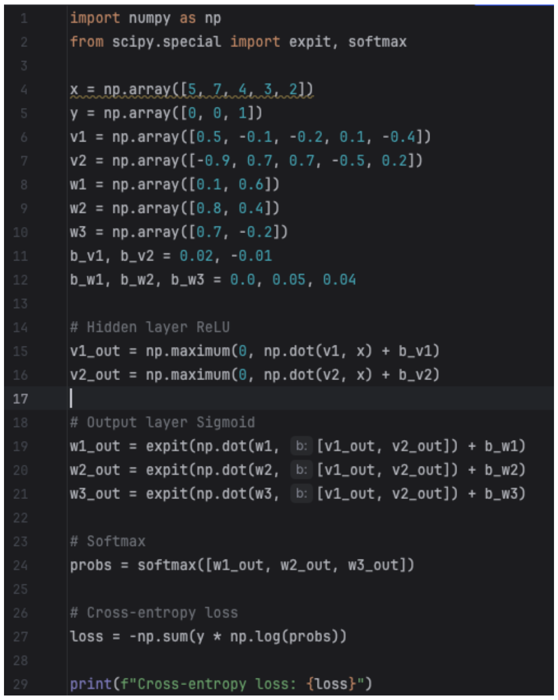
1. $z = w \times x_i + b = (0.5 \times 6) + (0.1 \times 4) + (-0.8 \times 2) + (0.9 \times 3) + (0 \times 1) + 0.05$
   $z = 3 + 0.4 - 1.6 + 2.7 + 0 + 0.05 = 4.55$

   $\sigma(z) = 1 \div (1 + e^{-z})$
   $\sigma(4.55) = 1 \div (1 + e^{-4.55})$
   $\sigma(4.55) \approx 0.9895$

   $f(x_i) = [1 - \sigma(z), \sigma(z)] = [0.0105, 0.9895]$

   $L(x_i, y_i) = -y_i \times log(\hat{y_i})$
   $L(x_i, y_i) = -log(\hat{y_i})$
   $L(x_i, y_i) = -log(0.9895) = 0.0046$

2. Cross Entropy Loss = 1.301



```python
import numpy as np
from scipy.special import expit, softmax

x = np.array([5, 7, 4, 3, 2])
y = np.array([0, 0, 1])
v1 = np.array([0.5, -0.1, -0.2, 0.1, -0.4])
v2 = np.array([-0.9, 0.7, 0.7, -0.5, 0.2])
w1 = np.array([0.1, 0.6])
w2 = np.array([0.8, 0.4])
w3 = np.array([0.7, -0.2])
b_v1, b_v2 = 0.02, -0.01
b_w1, b_w2, b_w3 = 0.0, 0.05, 0.04

# Hidden layer ReLU
v1_out = np.maximum(0, np.dot(v1, x) + b_v1)
v2_out = np.maximum(0, np.dot(v2, x) + b_v2)

# Output layer Sigmoid
w1_out = expit(np.dot(w1,  b: [v1_out, v2_out]) + b_w1)
w2_out = expit(np.dot(w2,  b: [v1_out, v2_out]) + b_w2)
w3_out = expit(np.dot(w3,  b: [v1_out, v2_out]) + b_w3)

# Softmax
probs = softmax([w1_out, w2_out, w3_out])

# Cross-entropy loss
loss = -np.sum(y * np.log(probs))

print(f"Cross-entropy loss: {loss}")
```

Figure 1: Q2: Cross Entropy Loss Python Code

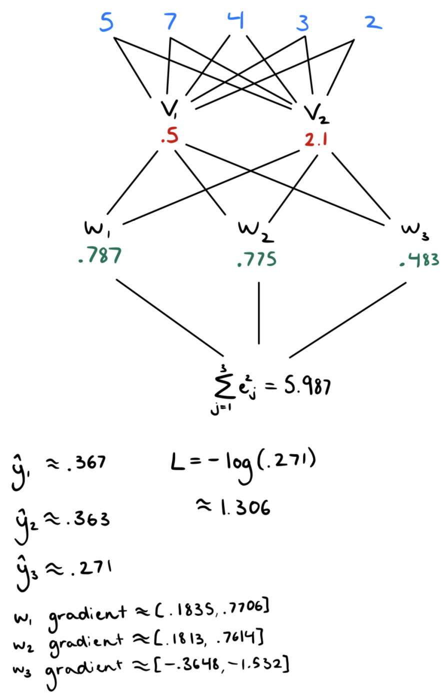


Figure 2: Q3: Computation Graph

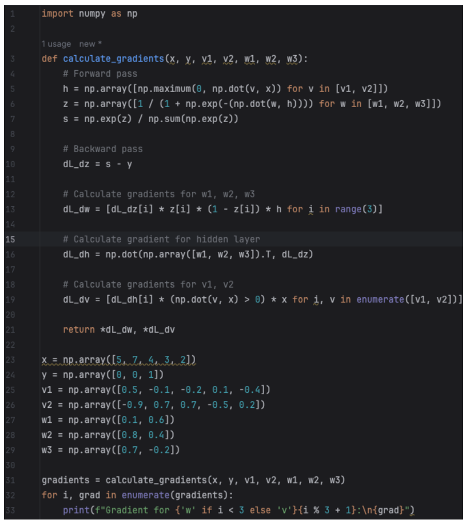
3. $w1 = [0.1835, 0.7706]$

   $w2 = [0.1813, 0.7614]$

   $w3 = [-0.3648, -1.532]$

   $v1 = [-0.9195, -1.2873, -0.7356, -0.5517, -0.3678]$

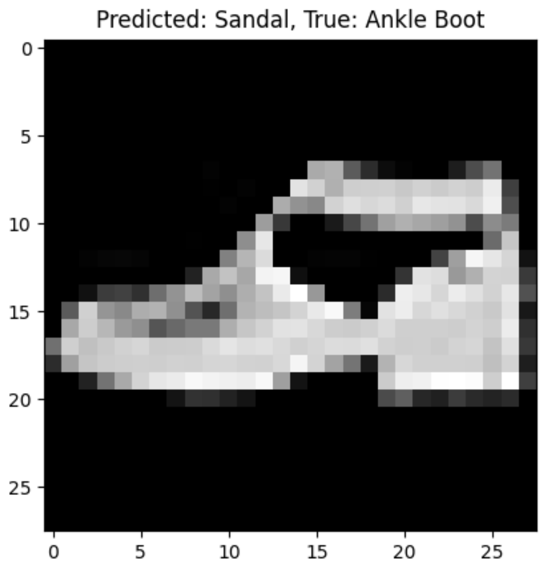
   $v2 = [2.5555, 3.5777, 2.0444, 1.5333, 1.0222]$



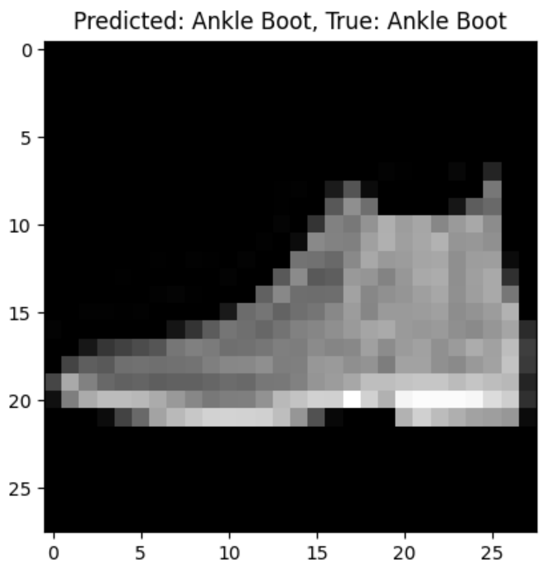
```python
import numpy as np

1 usage  new *
def calculate_gradients(x, y, v1, v2, w1, w2, w3):
    # Forward pass
    h = np.array([np.maximum(0, np.dot(v, x)) for v in [v1, v2]])
    z = np.array([1 / (1 + np.exp(-(np.dot(w, h)))) for w in [w1, w2, w3]])
    s = np.exp(z) / np.sum(np.exp(z))

    # Backward pass
    dL_dz = s - y

    # Calculate gradients for w1, w2, w3
    dL_dw = [dL_dz[i] * z[i] * (1 - z[i]) * h for i in range(3)]

    # Calculate gradient for hidden layer
    dL_dh = np.dot(np.array([w1, w2, w3]).T, dL_dz)

    # Calculate gradients for v1, v2
    dL_dv = [dL_dh[i] * (np.dot(v, x) > 0) * x for i, v in enumerate([v1, v2])]

    return *dL_dw, *dL_dv

x = np.array([5, 7, 4, 3, 2])
y = np.array([0, 0, 1])
v1 = np.array([0.5, -0.1, -0.2, 0.1, -0.4])
v2 = np.array([-0.9, 0.7, 0.7, -0.5, 0.2])
w1 = np.array([0.1, 0.6])
w2 = np.array([0.8, 0.4])
w3 = np.array([0.7, -0.2])

gradients = calculate_gradients(x, y, v1, v2, w1, w2, w3)
for i, grad in enumerate(gradients):
    print(f"Gradient for {'w' if i < 3 else 'v'}{i % 3 + 1}:\n{grad}")
```

Figure 3: Q3: Python Code

4.

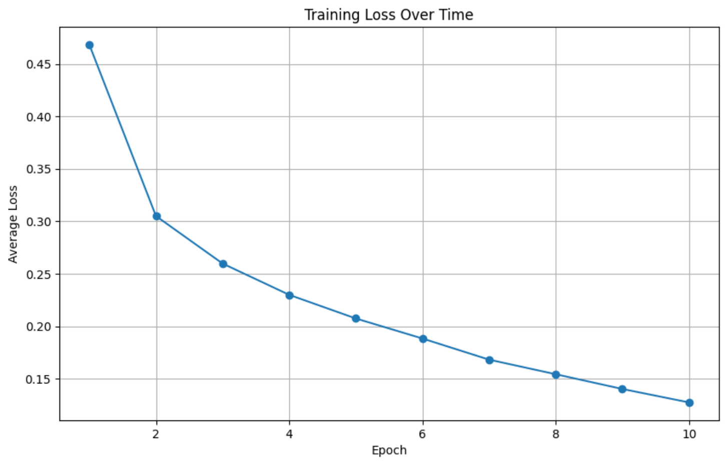


Q4: Incorrectly Classified Image



Q4: Correctly Classified Image

5.



Q5: Training Loss Over Time

6. (a) Accuracy may not still be as good of a metric. Since accuracy measures how often a model is correct overall (across all classes), with imbalanced data, a model could achieve high accuracy with poor performance on the minority class, shirts, by just predicting the majority class, coats, more often. This is misleading the evaluation of the model, specifically when it comes to its ability to predict the

minority class.

(b) To account for this data imbalance, one approach would be to look at each class more specifically (so that each class's contribution to the accuracy is more fair), for example, a weighted or balanced accuracy score. This would work by either getting the average recall from each class or by assigning weights to each class based on their representation in the dataset. This way when there is a majority class and a model is biased to it, it accurately captures the poor performance on the underrepresented class(es).

(c) Lots of other metrics exist to evaluate models and come in useful when trying to gauge the whole-rounded performance of the model (since most evaluations fall short in some way). For example:

- Precision and Recall, focus on single classes at a time.
- F1 Score, harmonic mean of precision and recall.
- Confusion Matrix, visualize correct and incorrect classifications per class
- AUC-ROC, ability to distinguish between classes

7. Convolutional Layer 1: $(1 * 32 * 3 * 3) + 32 = 320$

   Convolutional Layer 2: $(32 * 64 * 3 * 3) + 64 = 18496$

   Fully Connected Layer 1: $((64 * 7 * 7) * 128) + 128 = 401536$

   Fully Connected Layer 2: $(128 * 10) + 10 = 1290$

   $320 + 18496 + 401536 + 1290 = 421642$

   The model has 421,642 tunable parameters in total.