Handin 2

« Neural Nets for Multiclass Classification »

Rasmus Graversgaard Pedersen, 202006588

Emmanouil Platakis, 202402638

October 24, 2024

Contents

1 Part I: Derivative

The softmax function is:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

The NLL loss is :

$$L(z) = -\ln(\text{softmax}(z)_j)$$

Derivative of the Loss with Respect to z

We now compute the gradient of the loss $L(z)$ with respect to each component z_i. We will use the chain rule for this computation.

1. Derivative of ln

The derivative of $-\ln(x)$ is:

$$\frac{d}{dx}[-\ln(x)] = -\frac{1}{x}$$

Thus, the derivative of the loss with respect to $\text{softmax}(z)_j$ is:

$$\frac{\partial L}{\partial \text{softmax}(z)_j} = -\frac{1}{\text{softmax}(z)_j}$$

2. Derivative of the Softmax Function

The derivative of the softmax is computed like this:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

For the correct class j:

$$\frac{\partial \text{softmax}(z)_j}{\partial z_j} = \text{softmax}(z)_j \cdot (1 - \text{softmax}(z)_j)$$

For the incorrect classes that are not j:

$$\frac{\partial \text{softmax}(z)_i}{\partial z_j} = -\text{softmax}(z)_i \cdot \text{softmax}(z)_j$$

3. Chain Rule

For the correct class j:

$$\frac{\partial L}{\partial z_j} = \frac{\partial L}{\partial \text{softmax}(z)_j} \cdot \frac{\partial \text{softmax}(z)_j}{\partial z_j} = -\frac{1}{\text{softmax}(z)_j} \cdot \text{softmax}(z)_j \cdot (1 - \text{softmax}(z)_j) = \text{softmax}(z)_j - 1$$

For the incorrect classes that are not j:

$$\frac{\partial L}{\partial z_i} = \frac{\partial L}{\partial \text{softmax}(z)_j} \cdot \frac{\partial \text{softmax}(z)_i}{\partial z_j} = -\frac{1}{\text{softmax}(z)_j} \cdot (-\text{softmax}(z)_i \cdot \text{softmax}(z)_j) = \text{softmax}(z)_i$$

Final Derivatives

For the correct class j:

$$\frac{\partial L}{\partial z_j} = \text{softmax}(z)_j - 1$$

For the incorrect classes:

$$\frac{\partial L}{\partial z_i} = \text{softmax}(z)_i$$

We can combine the two cases into a single expression using the Kronecker delta $\delta_{i,j}$, where $\delta_{i,j} = 1$ if $i = j$ and 0 otherwise:

$$\frac{\partial L}{\partial z_i} = -\delta_{i,j} + \text{softmax}(z)_i$$

These derivatives can be used in the backpropagation algorithm to update the neural network weights.

2 Part II: Implementation and test

Forward pass

```
z1 = np.dot(X, W1)+b1          # Input to hidden layer
a_relu = relu(z1)              # RELU Activation
z2 = np.dot(a_relu, W2)+b2     # Hidden to output layer (final output)
s = softmax(z2)                # Compute softmax probabilities

# Compute total cost including cross-entropy loss and weight decay
cost = -np.sum(labels * np.log(s + 1e-12)) / X.shape[0]
cost += c * (np.sum(W1 ** 2) + np.sum(W2 ** 2))
```

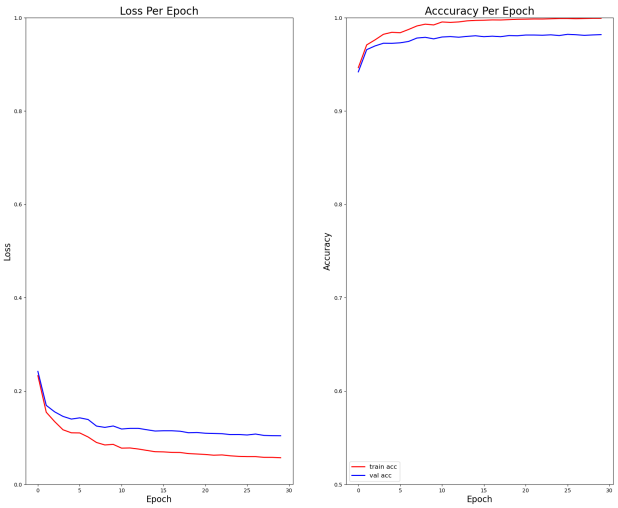
Backward pass

```
d_z2 = (s - labels) / X.shape[0]

d_w2 = np.dot(a_relu.T, d_z2) + 2 * c * W2
d_b2 = np.sum(d_z2, axis=0, keepdims=True)

d_a_relu = np.dot(d_z2, W2.T)
d_z1 = d_a_relu * (z1 > 0)

d_w1 = np.dot(X.T, d_z1) + 2 * c * W1
d_b1 = np.sum(d_z1, axis=0, keepdims=True)
```



The plot is as expected, the can clearly see the model improving rapidly in the beginning before hitting a plateau after 15 epochs.