Understanding Batch Processing in Neural Networks

October 5, 2024

1 Running Instances in Parallel and Parameter Updates

In deep learning, we often process multiple instances (or data samples) at once, known as **batch processing**. Here's a brief explanation of the key ideas:

- **Instances in Parallel**: Instead of processing one data sample at a time, a batch of samples is processed together. This is often done to leverage the computational power of modern hardware (like GPUs) and improve training efficiency.
- **Average Gradients**: After processing the entire batch, the gradients of the loss function with respect to the model parameters (weights and biases) are computed for each instance in the batch. These gradients are then averaged over the batch. This averaging helps smooth out the updates, leading to more stable and generalized learning.
- **Parameter Update**: Once the average gradients are computed, the model parameters are updated based on these average gradients. This is typically done using a gradient descent optimization algorithm:

$$W = W - \eta \cdot \text{average gradient} \tag{1}$$

  Where η is the learning rate.

2 MLP with Batch Index Notation

Now, let's dive into the provided notation for a multilayer perceptron (MLP) using a batch index:

2.1 Variables

- b: Batch index (indicating which sample in the batch is being processed).
- i: Input feature index (refers to the specific feature of the input).
- j: Hidden layer index (refers to the specific node in the hidden layer).
- $\hat{y}$: Possible label (refers to the predicted class or output).

2.2 Parameters

- $W_0[j, i]$: Weights connecting input features to the hidden layer (layer 0).
- $C_0[j]$: Bias for the hidden layer node j.
- $W_1[\hat{y}, j]$: Weights connecting hidden layer nodes to the output layer (layer 1).
- $C_1[\hat{y}]$: Bias for the output node corresponding to label $\hat{y}$.

2.3 Equations

1. **Hidden Layer Activation**:

$$h[b, j] = \sigma \left(\sum_i (W_0[j, i] \cdot x[b, i]) - C_0[j] \right) \tag{2}$$

What it Means: For each hidden layer node j in the batch b, the activation $h[b, j]$ is calculated. This involves summing the products of the input features $x[b, i]$ for the current batch and the corresponding weights $W_0[j, i]$, and then subtracting the bias $C_0[j]$. The activation function σ (which could be ReLU, sigmoid, etc.) is then applied to the result.

2. **Output Layer Activation**:

$$s[b, \hat{y}] = \sigma \left(\sum_j (W_1[\hat{y}, j] \cdot h[b, j]) - C_1[\hat{y}] \right) \tag{3}$$

What it Means: For each predicted label $\hat{y}$ in the batch b, the output $s[b, \hat{y}]$ is calculated. This involves summing the products of the hidden layer activations $h[b, j]$ and the corresponding weights $W_1[\hat{y}, j]$, and then subtracting the bias $C_1[\hat{y}]$. Again, the activation function σ is applied.

3. **Softmax Calculation**:

$$P_\Phi[b, \hat{y}] = \text{softmax}(s[b, \hat{y}]) \tag{4}$$

What it Means: The softmax function is applied to the output scores $s[b, \hat{y}]$ to obtain the probabilities $P_\Phi[b, \hat{y}]$ for each possible label $\hat{y}$ in the batch b. Softmax converts the raw output scores into a probability distribution across all possible classes.

3 Summary

In summary:

- The described process involves using multiple instances (samples) of input data, allowing efficient computation of forward passes through the network.
- By averaging gradients over a batch, the model can make more stable parameter updates.
- The notation describes how inputs are transformed into hidden layer activations and then into output scores, followed by converting those scores into probabilities using the softmax function.