Backpropagation for CNNs

Michael Wollowski

1

Review: Forward pass

- Your forward pass might look like this:

```
public double[] forward(double[][] input){
  conv1Forward();
  pool2Forward();
  conv3Forward();
  pool4Forward();
  ff5Forward();
  ff6Forward();
  outputForward();
}
```

2

Backward Pass

- Your backward pass should then look like this:

```
public void backward(int label) {
  outputBackward(label);
  ff6Backward();
  ff5Backward();
  pool4Backward();
  conv3Backward();
  pool2Backward();
  conv1Backward();
}
```

3

Training

- Your training function, for one example might look like this:

```
public void train(double[][] input, int label) {
  forward(input);
  backward(label);
  updateWeights();
}
```

4

Gradients

- Output gradients:

```
// Gradient of cross-entropy loss with softmax
// For softmax + cross-entropy, gradient is simply: predicted - actual
outputGradients[i] = outputLayer[i] - (i == label ? 1.0 : 0.0);
outputBiasGradients[i] = outputGradients[i];
```

- FF6 gradients:

```
// Backpropagate from output layer to FC2
ff6Gradients[i] += outputGradients[n] * outputWeights[n][i];

// Apply ReLU derivative (gradient is 0 if output was 0)
if (ff6Output[i] <= 0) ff6Gradients[i] = 0.0;
ff6BiasGradients[i] = ff6Gradients[i];
```

- FF5 gradients: analogous to FF6 gradients.

5

Gradients

- Pool4 gradients:

```
// Backpropagate from FF5 to Pool4
pool4Gradients[m][i][j] += ff5Gradients[n] * ff5Weights[n][m][i][j];
```

- Conv3 gradients:

```
// Backpropagate from Pool4 to Conv3 (through max pooling)
// Only the max position gets the gradient
conv3Gradients[f][maxI][maxJ] += pool4Gradients[f][i][j];
…
// Apply ReLU derivative and accumulate bias gradients
conv3BiasGradients[f] += conv3Gradients[f][i][j];
```

6

Gradients

- Pool2 gradients:
  - analogous to Pool4 gradients.
  - Be mindful of the connections
  - Recall that the weights are the filters
- Convolution1 gradients:
  - analogous to Conv3 gradients.
  - Be mindful of the connections
  - Recall that the weights are the filters

7

Updating Weights

- Output weights and biases:

```
outputWeights[n][i] -= learningRate * outputGradients[n] * ff2Output[i];
outputBiases[n] -= learningRate * outputBiasGradients[n];
```

- FF6 weights and biases:

```
ff6Weights[n][i] -= learningRate * ff6Gradients[n] * ff1Output[i];
ff6Biases[n] -= learningRate * ff6BiasGradients[n];
```

- FF5 weights and biases: analogous to FF6

8

Updating Weights

- Conv3 weights and biases:

```
conv3Filters[f][filterIdx][fi][fj] -=
  learningRate * conv3Gradients[f][i][j] * pool2Output[inputMap][poolI][poolJ];
conv3Biases[f] -= learningRate * conv3BiasGradients[f];
```

- Conv1 weights and biases: analogous to Conv2

9