CS5242: NEURAL NETWORKS AND DEEP LEARNING

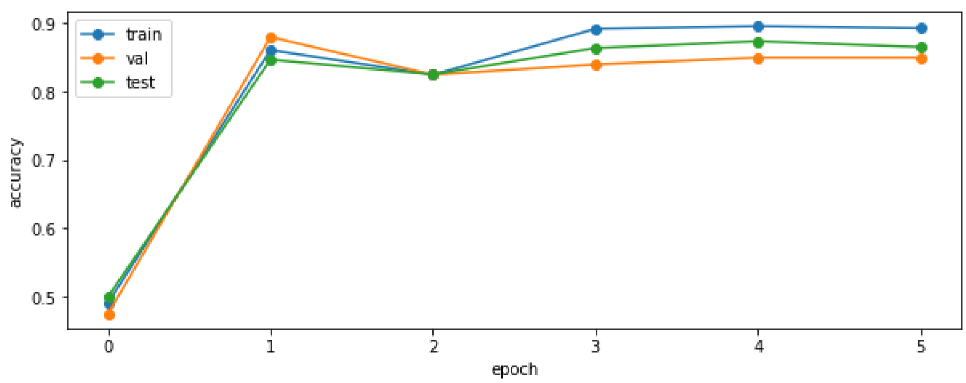
Assignment 2 - Building CNNs

Design choices

The following parameters are used to train the three layer net.

- Number of neurons in hidden layer $\rightarrow$ 1000
- Regularization parameter $\rightarrow$ 0.001
- Dropout rate $\rightarrow$ 0.5
- Optimization Scheme $\rightarrow$ *Adam with* $\alpha = 0.001$ and batch size $= 50$

Adam optimization scheme is used since it allows for faster convergence and is computationally efficient compared to vanilla sgd. It was noted that the accuracy drops when the size of the neurons is small so a hidden layer of size 1000 was chosen. However, this also introduces overfitting to the model. As a remedy, dropout was added to the model. When the dropout rate is low, the model overfits but when the dropout rate is high, the accuracy becomes lower. An intermediate value of 0.5 is used for dropout, which means each neuron has a 50% chance of being dropped. The following figure shows the accuracy during training, validation and testing for every epoch.



The next section explains the logic behind each function.

Pseudo codes of the functions

Algorithm 1 Forward Convolution

procedure CONV_FORWARD
    apply same padding by padding x with pad/2 for both height and width
    calculate height $H'(1 + \frac{l_h+p-k_h}{s})$ *and width* $W'(1 + \frac{l_w+p-k_w}{s})$*of the output*
    for *row in range*(H') **do**
        for *col in range*(W') **do**
            $x_{reshaped} \leftarrow x_{padded}[:, :, i : i + k_h, j : j + k_w].reshape(C * HH * WW)$
            $j \leftarrow j + s.$
        end for
        $i \leftarrow i + s.$
    end for
    *Reshape w to (F, C * HH * WW)*
    $out \leftarrow w_{reshaped} \cdot x'_{reshaped} + b.$
    return *out*
end procedure

Algorithm 2 Backward Convolution

procedure CONV_BACKWARD
    $db \leftarrow \Sigma dout$ ▷ get gradients by summing across all columns except F
    $dw \leftarrow dout' \cdot x_{reshaped}$ ▷ get weight gradients (reshaping is not shown here)
    $dx_{reshaped} \leftarrow w_{reshaped} \cdot dout'$ ▷ get x gradients (reshaping is not shown here)
    for *row in range*(H') **do** ▷ get padded dx
        for *col in range*(W') **do**
            $dx_{padded}[:, :, i : i + k_h, j : j + k_w] += dx_{reshaped}[:, index]$
            $index \leftarrow index + 1.$
            $j \leftarrow j + s.$
        end for
        $i \leftarrow i + s.$
    end for
    $dx \leftarrow dx_{padded}[:, :, p : -p, p : -p]$ ▷ remove paddings
    return dx, dw, db
end procedure

Algorithm 3 Max Pool Forward

procedure MAX_POOL_FORWARD
  for *each input m* **do**
    for *each channel c* **do**
      for *row in range*(h_{pool}, h_x, s) **do**
        for *col in range*(w_{pool}, w_x, s) **do**
          $field \leftarrow x[m, c, row - h_{pool} : row, col - w_{pool} : col]$
          $i_{max}, j_{max} \leftarrow max\_indices(field)$
          $out[m, c, i, j] \leftarrow x[m, c, i_{max} + row - h_{pool}, j_{max} + col - w_{pool}]$
          $j \leftarrow j + 1.$
        end for
        $i \leftarrow i + 1.$
      end for
    end for
  end for
  return x
end procedure

Algorithm 4 Max Pool Backward

procedure MAX_POOL_BACKWARD
  for *each input m* **do**
    for *each channel c* **do**
      for *row in range*(h_{pool}, h_x, s) **do**
        for *col in range*(w_{pool}, w_x, s) **do**
          $i_{max}, j_{max} \leftarrow max\_fields[index]$ ▷ cached from max_pool_fwd
          $field \leftarrow 0$
          $field[i_{max}, j_{max}] \leftarrow 1$ ▷ *only* i_{max}, j_{max} *will have a gradient*
          $field \leftarrow field * dout[m, c, i, j]$
          $dx[m, c, row - h_{pool} : row, col - w_{pool} : col] += field$
          $index \leftarrow index + 1.$
          $j \leftarrow j + 1.$
        end for
        $i \leftarrow i + 1.$
      end for
    end for
  end for
  return dx
end procedure

Algorithm 5 Dropout forward

```
procedure DROPOUT_FORWARD
    if training then
        mask ← random_int(0, 1) > p                    ▷ p = dropout rate
        out ← x ∗ mask/(1 − p)
    else if testing then
        out ← x
    end if
    return out
end procedure
```

Algorithm 6 Dropout backward

```
procedure DROPOUT_BACKWARD
    if training then
        dx ← dout ∗ mask/(1 − p)
    else if testing then
        dx ← dout
    end if
    return dx
end procedure
```

Algorithm 7 Forward Pass for 3 layer convolutional net

```
procedure NET_FORWARD
    a1, cache1 ← conv_relu_pool_forward(X, W1, b1, conv_param, pool_param)
    a2, cache2 ← affine_forward(a1, W2, b2)
    a2, cache3 ← relu_forward(a2)
    scores, cache4 ← affine_forward(a2, W3, b3)
end procedure
```

Algorithm 8 Backward Pass for 3 layer convolutional net

```
procedure NET_BACKWARD
    data_loss, dscores ← softmax_loss(scores, y)
    da2, dW3, db3 ← affine_backward(dscores, cache4)
    da2 ← relu_backward(da2, cache3)
    da1, dW2, db2 ← affine_backward(da2, cache2)
    dX, dW1, db1 ← conv_relu_pool_backward(da1, cache1)
end procedure
```