Backpropagation

Felix Portillo

EEC 289Q

Backpropagation (HW3)

- Forward Pass:
  - Inputs: Image data
  - Outputs: hAct{L}, L = numHidden+1
- Compute cost:
  - out = log(hAct{L})
  - Get index of actual label (i.e. index=4 if digit is 3), maybe using ‘sub2ind’
  - Compute cost:

$$J(\theta) = -\left[\sum_{i=1}^{m}\sum_{k=1}^{K} 1\left\{y^{(i)} = k\right\} \log \frac{\exp(\theta^{(k)\top} h_{W,b}(x^{(i)}))}{\sum_{j=1}^{K} \exp(\theta^{(j)\top} h_{W,b}(x)^{(i)}))}\right]$$

-sum(out(index))

Backpropagation (cont.)

- From MultiLayerNeuralNetworks:

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}.
2. For each output unit i in layer n_l (the output layer), set

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2}\|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

Backpropagation (cont.)

- From MultiLayerNeuralNetworks:

1. Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}.

2. ~~For each output unit i in layer n_l (the output layer), set~~

$$\cancel{\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2}\|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})}$$

$$\delta^{(n_l)} = -\sum_{i=1}^{m}\left[\left(1\{y^{(i)} = k\} - P(y^{(i)} = k | x^{(i)}; \theta)\right)\right]$$

3. For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$

   For each node i in layer l, set

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)}\right) f'(z_i^{(l)})$$

4. Compute the desired partial derivatives, which are given as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)}$$
$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)}.$$

What you need

- Get one-hot encoding of actual label
  - Ex. y_actual = zeros(size(hAct{L})); y_actual(index) = 1;

- For each layer, need to compute ∂, ∇W, ∇b

- Also need two more variables within algorithm, we'll call them dz and g'
- Store gradients for W, (i.e. ∇W) in gradStack{l}.W and b (i.e. ∇b) in gradStack{l}.b

The algorithm

- Compute forward pass and store outputs of each layer in hAct{l}, where l is the layer number.
- Compute ∂ $\delta^{(n_l)} = -(y - a^{(n_l)}) \bullet f'(z^{(n_l)})$
  - For softmax output: ∂ = hAct{L} – y_actual;
- For layer l = L:-1:1

```
if l = L, is last layer
        g' = ones(size(∂))
else
        g' = derivative of activation function (i.e. for sigmoid: g' = hAct{l} * (1 – hAct{l}) )
dz = ∂ * g' %update dz using current ∂ value, which is error from previous layer
```

$$\delta^{(l)} = \left((W^{(l)})^T \delta^{(l+1)}\right) \bullet f'(z^{(l)})$$

```
if l > 1, is not first layer %Store weight gradients
        gradStack{l}.W = dz * hAct{l−1}^T % use the activation of previous layer
else, is first layer
        gradStack{l}.W = dz * data^T % use input image
gradStack{l}.b = sum(dz);  %store bias gradients
∂ = stack{l}. W^T * dz;  %update ∂ for next iteration of loop
```

$$\nabla_{W^{(l)}} J(W,b;x,y) = \delta^{(l+1)} (a^{(l)})^T,$$
$$\nabla_{b^{(l)}} J(W,b;x,y) = \delta^{(l+1)}.$$

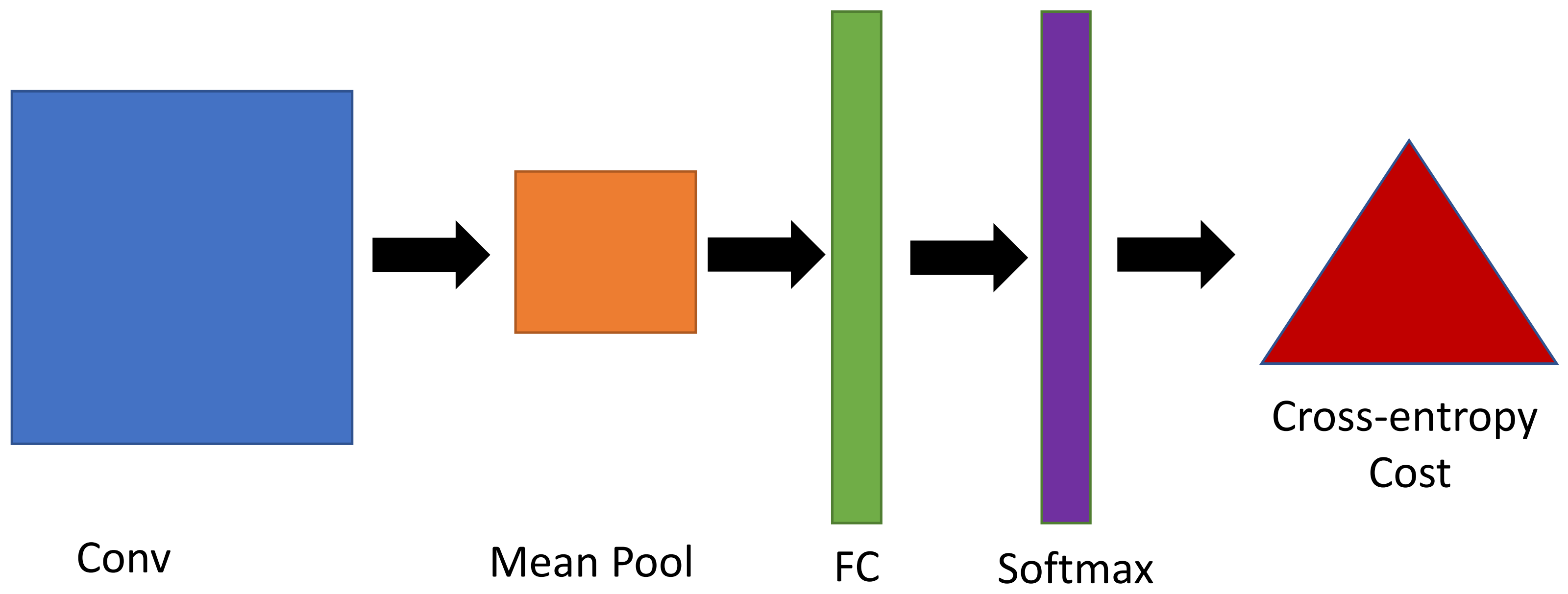
Things to remember

- At end, ‘supervised_dnn_cost.m’ returns ‘grad’ so use:

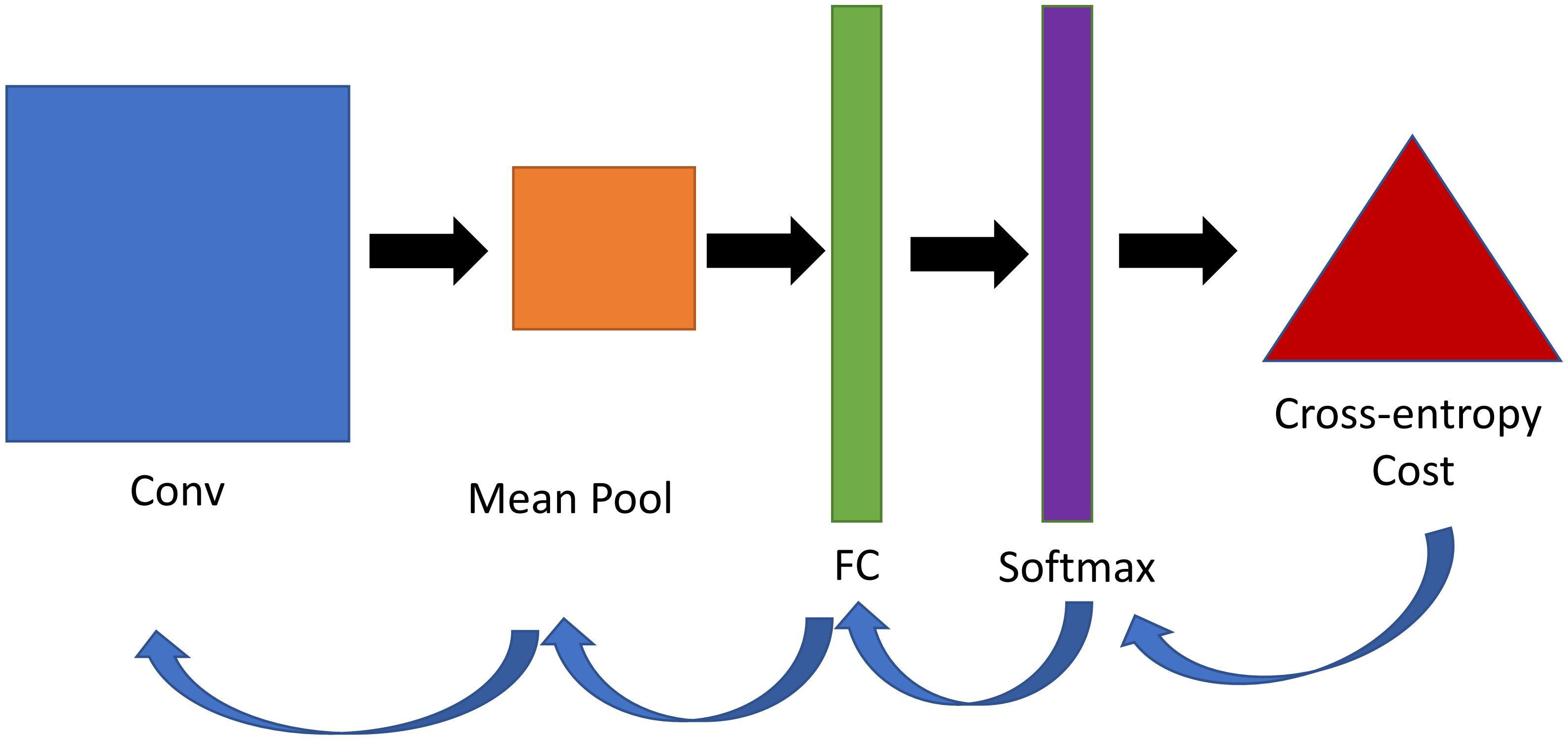
  [grad] = stack2params(gradStack)

- May be easier to do vectorized implementation over batch for speed and to avoid confusion.
  - Double check sizes of each variable to ensure they are compatible!

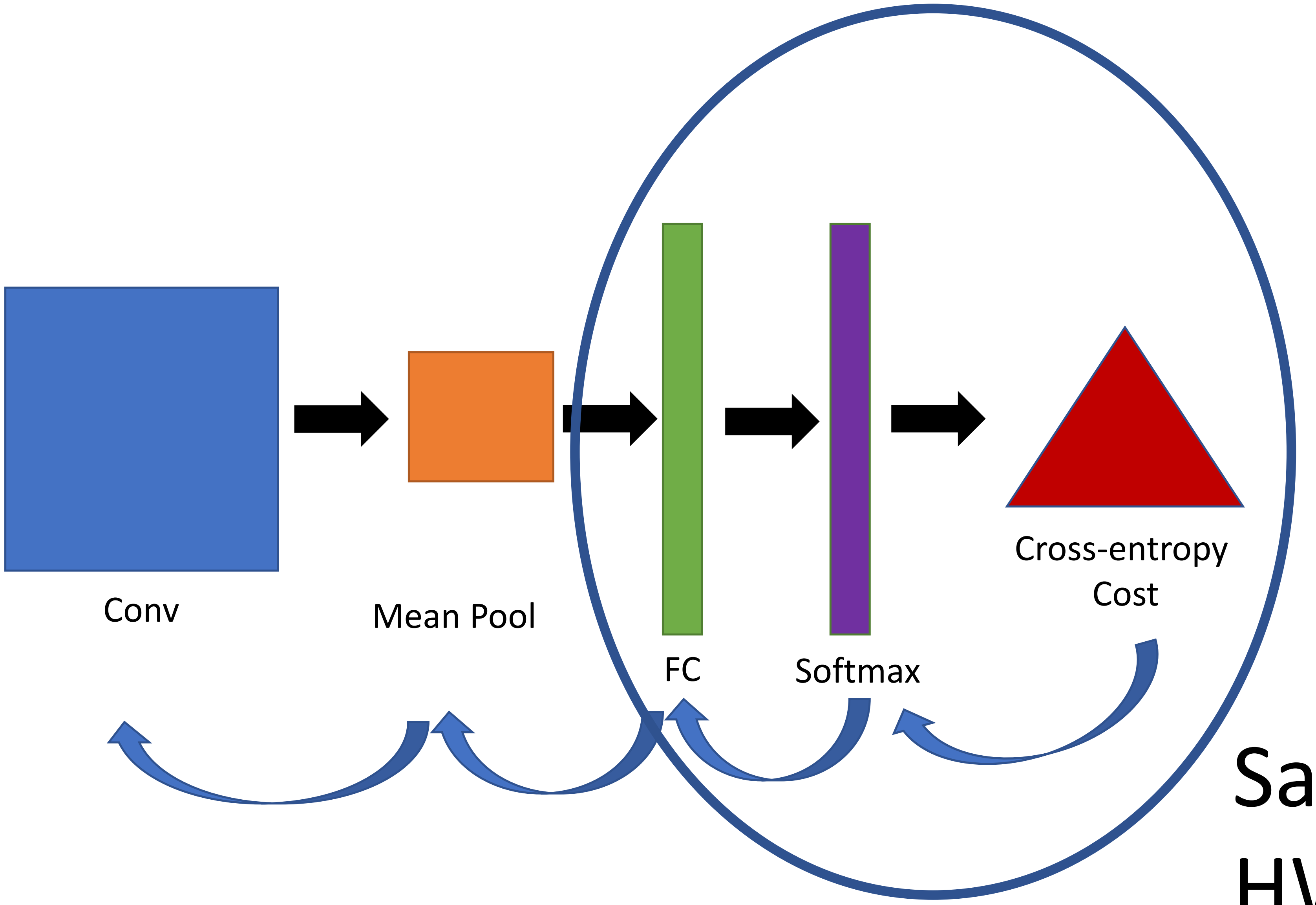
HW4



HW4



HW4



Same as HW3!

Key Points and Tips

- Backprop has same general concepts for both HW3 and HW4
  - Calculate error, ∂, at last layer(L) (softmax error)
  - Go through each layer in network to compute ∂ for layers starting at L-1
    - Find derivative of activation at each layer, if there is activation at that layer
    - Multiply error from layer l+1 with weights of current layer and/or derivative of activation It depends on if it's an FC layer or conv layer, you just need to follow directions according to operation. (Details specified on website for conv)
    - Compute gradients for weights and biases
      - For FC layers, follow outline from HW3, otherwise do conv gradient calculation based on website.
      - Ex. Compute gradients using inputs into layer using either multiplication (FC layer) or convolution with rotated error (conv layer)

Key Points and tips

- Difference b/w backprop for HW3 and HW4
  - HW3: Computes errors (∂) and gradients within same loop
  - HW4: May be best to compute errors (∂) and gradients separately since the layer types are different. Meaning, don't use a loop to go backwards through each layer.
- Remember to reshape matrix after mean pooling into a vector for the FC Layer
  - Similarly, remember to reshape the error vector from the pooling back into a matrix before applying the 'kron' function.
- Pooling has no parameters to update so no need to find gradient for that layer.
- In the end, only returning gradients for $W_{conv}, b_{conv}, W_{FC}, b_{FC}$

Key Points and Tips (SGD)

- For SGD function, remember to check the order the parameters are initiated in ‘cnnInitParams’ to help with the theta update.
  - Hint: Makes it easier for vectorized weight updates