CS 615 – Deep Learning

Learning

Slides adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

Objectives

- Updating weights
- Gradient descent

Updating Weights

- Ok fine, we can do forward-backward propagation.
- But how does this help us learn the weights to minimize our objective function?
- When we hit a layer that has weights (for now, that's just the fully-connected layer) we can use the incoming (backcoming?) gradient to update the weights!
- For example, our fully-connected layers have weights, W, and biases, b
- So, we'll want to compute $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$ and move/update our weights by going *some amount* in the direction of the gradient.

Gradient: Fully Connected Layer

- Imagine that our FC layer has a backpropagated gradient $\frac{\partial J}{\partial h} \in \mathbb{R}^{1 \times K}$ coming into it.
- We then need $\frac{\partial H}{\partial W}$ and $\frac{\partial H}{\partial b}$ in order to compute $\frac{\partial J}{\partial W}$ and $\frac{\partial J}{\partial b}$
- Let's start with $\frac{\partial J}{\partial b}$
- $h \in \mathbb{R}^{1 \times K}$ and $b \in \mathbb{R}^{1 \times K}$, so what is the size of $\frac{\partial h}{\partial b}$?
  - A $K \times K$ Jacobian matrix
- What are the elements of $\frac{\partial h}{\partial b}$?
  - Recall that $h = xW + b$
  - Identity matrix!
- And if we have multiple observations, then $\frac{\partial H}{\partial b}$ is a $N \times (K \times K)$ tensor of identity matrices.

Gradient: Fully Connected Layer (bias)

- So what is $\frac{\partial J}{\partial b}$ then?
- For a single observation it will be:

$$\frac{\partial J}{\partial b} = \frac{\partial J}{\partial h} \cdot \frac{\partial h}{\partial b} = \frac{\partial J}{\partial h}$$

- If there's multiple observations?

$$\frac{\partial J}{\partial H} \otimes \frac{\partial H}{\partial b} \in \mathbb{R}^{N \times (1 \times K)}$$

- So to get $\frac{\partial J}{\partial b}$ we just take the average of the N elements (vectors) of this tensor.

Gradient: Fully Connected Layer (weights)

- How about $\frac{\partial J}{\partial W}$?
- For the single observation case we need $\frac{\partial h}{\partial W}$.
- So what is the derivative of a vector with respect to a matrix?
- Since $h \in \mathbb{R}^{1\times K}$ and $W \in^{D\times K}$ intuitively this should be $\in \mathbb{R}^{K\times(D\times K)}$
  - And for a particular output $h_k \in \mathbb{R}^{D\times K}$
- So what are the elements of $\frac{\partial h_k}{\partial W}$?

Gradient: Fully Connected Layer

$$\boxed{h = xW + b}$$

- Let's start with, what is $\frac{\partial h_k}{\partial W_{ij}}$?
  - If $k == j$

$$\frac{\partial h_k}{\partial W_{ij}} = x_i$$

  - Otherwise

$$\frac{\partial h_k}{\partial W_{ij}} = 0$$

- How can we interpret this?
- Let's look at a somewhat simple example…

Gradient: Fully Connected Layer

$$h = xW + b$$

- Let's look at somewhat simple example…
- Let $W \in \mathbb{R}^{3\times 2}$

$$W = \begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \\ W_{31} & W_{32} \end{bmatrix}, b = [b_1, b_2]$$

- What is $\frac{\partial h_1}{\partial W}$?
  - $h_1 = x_1 W_{11} + x_2 W_{21} + x_3 W_{31} + b_1$
  - $\frac{\partial h_1}{\partial W_{11}} = x_1, \frac{\partial h_1}{\partial W_{12}} = 0, \frac{\partial h_1}{\partial W_{21}} = x_2, \frac{\partial h_1}{\partial W_{22}} = 0, \frac{\partial h_1}{\partial W_{31}} = x_3, \frac{\partial h_1}{\partial W_{32}} = 0$
  - $\frac{\partial h_1}{\partial W} = \begin{bmatrix} x_1 & 0 \\ x_2 & 0 \\ x_3 & 0 \end{bmatrix}$

Gradient: Fully Connected Layer

$$h = xW + b$$

$$\frac{\partial h_1}{\partial W} = \begin{bmatrix} x_1 & 0 \\ x_2 & 0 \\ x_3 & 0 \end{bmatrix}$$

- How about $\frac{\partial h_2}{\partial W}$?
  - $h_2 = x_1 W_{12} + x_2 W_{22} + x_3 W_{32} + b_1$

$$\frac{\partial h_2}{\partial W} = \begin{bmatrix} 0 & x_1 \\ 0 & x_2 \\ 0 & x_3 \end{bmatrix}$$

- So each matrix $\frac{\partial h_k}{\partial W}$ is all zeros, except for column k, which has x^T on it.
- So maybe we just use x^T somehow?

Gradient: Fully Connected Layer

$$h = xW + b$$

- Let's think about this another way..
- What is $\frac{\partial J}{\partial W_{ij}}$?
- W_{ij} only affects h_j on the output of the fully-connected layer and $h_j = xW_{:j} + b_j$
- So what is $\frac{\partial h_j}{\partial W_{ij}}$?
  - x_i
- So for $\frac{\partial J}{\partial W_{ij}}$ we just need:

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial h_j} x_i$$

Gradient: Fully Connected Layer

$$\boxed{h = xW + b}$$

$$\frac{\partial J}{\partial W_{ij}} = \frac{\partial J}{\partial h_j} x_i$$

- So if $W \in \mathbb{R}^{3\times 2}$, we get:

$$\frac{\partial J}{\partial W} = \begin{bmatrix} \frac{\partial J}{\partial h_1} x_1 & \frac{\partial J}{\partial h_2} x_1 \\ \frac{\partial J}{\partial h_1} x_2 & \frac{\partial J}{\partial h_2} x_2 \\ \frac{\partial J}{\partial h_1} x_3 & \frac{\partial J}{\partial h_2} x_3 \end{bmatrix} = x^T \frac{\partial J}{\partial h}$$

- Or in general:

$$\frac{\partial J}{\partial W} = x^T \delta$$

Gradient: Fully Connected Layer

$$h = xW + b$$

$$\frac{\partial J}{\partial W} = x^T \delta$$

- Dimension check?
  - $\delta \in \mathbb{R}^{1\times K}, x^T \in \mathbb{R}^{D\times 1}$, so $\frac{\partial J}{\partial W} = x^T\delta \in (\mathbb{R}^{D\times 1})(\mathbb{R}^{1\times K}) \in \mathbb{R}^{D\times K}$
  - Same dimensions as W!
- What about multiple observations?
- Actually, pre-multiplying the incoming gradient by X^T will even provide the summation part of the mean of the gradient (so we just need to divide by the number of observations, N)!

$$\frac{\partial J}{\partial W} = \frac{1}{N} X^T \delta$$

Updating Weights

- Now that we have the gradient of our objective function with regards to its weights, we can update them!
- Since all our objective functions were framed as loss function, we actually want to go some amount in the direction **opposite** the gradient:

$$W = W + \eta\left(-\frac{\partial J}{\partial W}\right)$$

- The variable η is called the *learning rate* and it is a hyperparameter that we can choose.
- More on that in a moment, but as a start, we might set it to be a small value like $\eta = 10^{-4}$

Updating Weights

- Let's add a method to our `FullyConnected` layer, called `updateWeights`, that takes an incoming gradient and a learning rate, and updates the layer's weights.

```
def updateWeights(gradIn, eta):
  dJdb = np.sum(gradIn, axis = 0)/gradIn.shape[0]
  dJdW = (self.getPrevIn().T @ gradIn)/gradIn.shape[0]

  self.__weights -= eta*dJdW
  self.__biases -= eta*dJdb
```

Backpropagating

- And now our backpropagation looks like
  - Note, that since calling updateWeights updates the weights and biases of the fully-connected layer, I get the pass-through gradient of the fully-connected layer *before* updating the weights.
    - Otherwise it's like trying to hit a moving target!

```
#backwards!
grad = layers[-1].gradient(Y,h)
for i in range(len(layers)-2,0,-1):
  newgrad = layers[i].backward(grad)

  if(isinstance(layers[i],FullyConnected)):
    layers[i].updateWeights(grad,eta)

  grad = newgrad
```

Gradient Descent

- Nice!
- Do we just update the weights once?
  - No!
- We keep doing forward-backwards propagation until we hit some sort of termination criteria.
  - Ideally we detect convergence of the objective function.
  - Worst case we run out of allotted time.

```
while still learning
    Perform forward propagation
    Perform backwards propagation, updating weights
```

Gradient Descent

- Each time we update the weights is called an ***epoch****.*
- If we're using all the observations for each epoch we call this *batch gradient descent.*
- If we're using just one observation each time, we call this *online gradient descent.*
- Both have their pros and cons.
- Therefore, it is common to do *mini-batch gradient descent.*
- If there's random selection involved, then we call this *stochastic gradient descent.*

References

- Textbook
  - Section 4.3 : Gradient-Based Optimization
  - Section 5.9: Stochastic Gradient Descent
- Web
  - http://ruder.io/optimizing-gradient-descent/
  - https://www.mathworks.com/help/deeplearning/ref/connectlayers.html;jsessionid=5a96c52e8efec11d475011b1222f
  - https://deepnotes.io/softmax-crossentropy
  - https://ml-cheatsheet.readthedocs.io/en/latest/index.html