3

3. **(10 points)** The use of ℓ_2 regularization for training multi-layer neural networks has a special name: *weight decay*. Assume an arbitrary dataset $\{(x_i, y_i)\}_{i=1}^{n}$ and a loss function $\mathcal{L}(w)$ where w represents the trainable weights (and biases).

   a. Write down the ℓ_2 regularized loss, using a weighting parameter λ for the regularizer.

   b. Derive the gradient descent update rules for this loss.

   c. Conclude that in each update, the weights are "shrunk" or "decayed" by a multiplicative factor before applying the descent update.

   d. What does increasing λ achieve algorithmically, and how should the learning rate be chosen to make the updates stable?

Answer : a) Regularized loss function , $L(w) = l(w) + \lambda \|w\|_2^2$
where $l(w)$ can be MSE or cross validation loss (softmax)

b) Gradient descent update rule becomes,

$$\nabla L(w) = \nabla l(w) + 2\lambda w$$

$$\boxed{w_{t+1} \leftarrow w_t - \alpha(\nabla L(w))}$$

, where $\alpha \rightarrow$ learning rate
$\lambda \rightarrow$ weighting parameter for the regularizer before

$$w_{t+1} \leftarrow w_t - \alpha \nabla l(w) - 2\lambda\alpha w_t$$

$$w_{t+1} \leftarrow w_t(1-2\lambda\alpha) - \alpha \nabla l(w)$$

c) The weight at each iteration decays by a factor of $(1-2\lambda\alpha)$ which can be seen from the above equation.

Without the regularizer $\nabla L(w) = \nabla l(w)$ and the G.D update would be $w_{t+1} \leftarrow w_t - \alpha \nabla l(w)$. Because of the regularizer the weights at every iteration are getting decayed by a factor of $(1-2\lambda\alpha)$ in addition to the changes provided by $\alpha \nabla l(w)$. Here we constrain the

Euclidean norm of w, we are encouraging many of the coefficients of w to become small. Because of this the variance of w is reduced. In practice this penalizes large weights or decays large weights and effectively limits the freedom in your model.

d) Increasing λ will move the weights closer and closer to zero. This means if λ value is too high, your model will be simple, but there a high chance of underfitting. This is because there is an excessive constraint on w to move around. One way to offset the effects of high λ value is to use a very small learning rate. Using a smaller learning rate would ensure that the model would make smaller changes in w, thereby covering a good number of potential optimal w values.

2

1. **(10 points)** Consider a one-hidden layer neural network (without biases for simplicity) with sigmoid activations trained with squared-error loss. Draw the computational graph and derive the forward and backward passes used in backpropagation for this network.

$$\hat{y} = W_2\sigma(W_1x), \;\; \mathcal{L} = \|\hat{y} - y\|_2^2$$

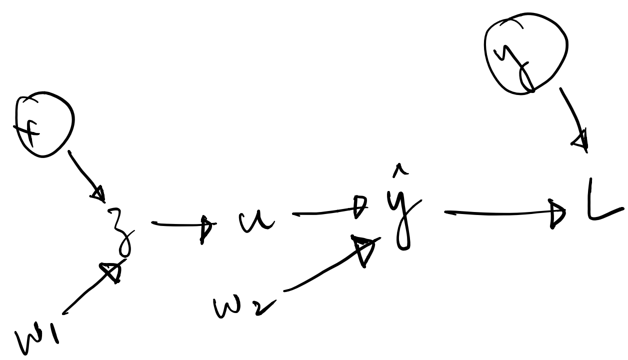
Qualitatively compare the computational complexity of the forward and backward passes. Which pass is more expensive and by roughly how much?

Answer:

$$y = W_2 \cdot \sigma(W_1 x)$$

$$L = \|\hat{y} - y\|_2^2$$

$$L(y) = \Sigma(\hat{y} - y)^2$$



<u>During forward pass</u> → Compute the output at every node as a function of its parents.

The values for L, u, $\hat{y}$ and z will be calculated at this stage using the relationship between the variables

$$\left.\begin{aligned} z &= w_1 x \\ u &= \sigma(z) \\ \hat{y} &= w_2 u \\ L &= (\hat{y} - y)^2 \end{aligned}\right\} \text{Relationship between the variables of the above network}$$

<u>During backward propagation</u> → Compute all gradients.

$$\frac{\partial L}{\partial L} = 1$$

$$\frac{\partial L}{\partial \hat{y}} = 2\hat{y} - 2y = 2(\hat{y} - y)$$

$$\frac{\partial L}{\partial u} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial u} = 2w_2(\hat{y} - y)$$

$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} = 2u(\hat{y} - y)$$

$$\frac{\partial L}{\partial z} = \frac{\partial L}{\partial u} \cdot \frac{\partial u}{\partial z} = 2w_2(\hat{y} - y) \cdot \sigma'(z)$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial z} \cdot \frac{\partial z}{\partial w_1} = 2w_2(\hat{y} - y)\,\sigma'(z) \cdot x$$

2. **(10 points)** Suppose that a convolutional layer of a neural network has an input tensor $X[i, j, k]$ and computes an output as follows:

$$Z[i, j, m] = \sum_{k_1} \sum_{k_2} \sum_{n} W[k_1, k_2, n, m] X[i + k_1, j + k_2, n] + b[m]$$
$$Y[i, j, m] = \text{ReLU}(Z[i, j, m])$$

for some kernel W and bias b. Suppose X and W have shapes $(48, 64, 10)$ and $(3, 3, 10, 20)$ respectively.

a. What are the shapes of Z and Y?

b. What are the number of input and output channels?

c. How many multiply- and add- operations are required to perform a forward pass through this layer? Rough calculations are OK.

d. What are the total number of trainable parameters in this layer?

(2)

a) Shape of $Z \rightarrow (46, 62, 20)$

Shape of $Y \rightarrow (46, 62, 20)$

Here we assume padding $= 0$ and stride $= 1$

Here $F = 3$, which is the Kernel size

$$W_{out} = W_{in} - F + 1$$
$$= 64 - 3 + 1 = 62$$
$$H_{out} = H_{in} - F + 1$$
$$= 48 - 3 + 1 = 46$$

b) Number of input channels $= 10$

Number of output channels $= 20$

c) No of add operations $= (3 \times 3 \times 10).(62 \times 46) \times 20 + 20$

$= 5133620$

No of multiply operations $= (3 \times 3 \times 10)(62 \times 46) \times 20$

$= 5133600$

d) Total number of trainable parameters $= (3 \times 3 \times 10 \times 20) + 20$

$= 1820$