Sistemas Inteligentes

Backpropagation

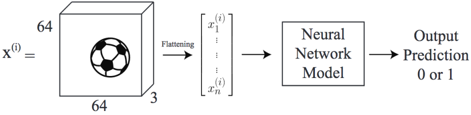
José Eduardo Ochoa Luna
Dr. Ciencias - Universidade de São Paulo

Maestría C.C. Universidad Católica San Pablo
Sistemas Inteligentes

23 de Octubre 2018

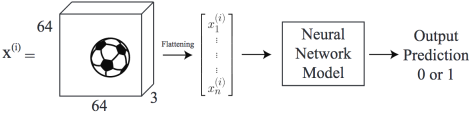
Backpropagation

- Suppose we wish to detect whether there is a soccer ball in a image or not



Backpropagation

- Suppose we wish to detect whether there is a soccer ball in a image or not
- Given an input image $x^{(i)}$, we wish to output a binary prediction (1 there is a ball)



The Problem

- Images can be represented as a matrix. In figure we have a $64 \times 64 \times 3$ containing a soccer ball. It is flattened into a single vector (12,288 elements).

The Problem

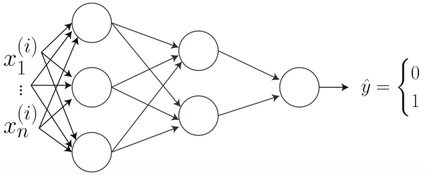
- Images can be represented as a matrix. In figure we have a $64 \times 64 \times 3$ containing a soccer ball. It is flattened into a single vector (12,288 elements).
- NN model: i) the network architecture (layers, neurons, connections) ii) the parameters (weights)

The Problem

- Images can be represented as a matrix. In figure we have a $64 \times 64 \times 3$ containing a soccer ball. It is flattened into a single vector (12,288 elements).
- NN model: i) the network architecture (layers, neurons, connections) ii) the parameters (weights)
- How to learn the parameters?

Parameters Initialization

Consider the following NN. The input is a flattened image vector $x^{(1)}, \ldots, x_n^{(i)}$. In the first hidden layer, all inputs are connected to all neurons in the next layer. This is called a fully connected layer.



How Many Parameters?

Forward propagation

$$
\begin{aligned}
z^{[1]} &= W^{[1]}x^{(i)} + b^{[1]} \\
a^{[1]} &= g(z^{[1]}) \\
z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\
a^{[2]} &= g(z^{[2]}) \\
z^{[3]} &= W^{[3]}a^{[2]} + b^{[3]}
\end{aligned}
$$

How Many Parameters? II

- $z^{[1]}, a^{[1]} \in \mathbb{R}^{3\times 1}$

How Many Parameters? II

- $z^{[1]}, a^{[1]} \in \mathbb{R}^{3\times 1}$
- $z^{[2]}, a^{[2]} \in \mathbb{R}^{2\times 1}$

How Many Parameters? II

- $z^{[1]}, a^{[1]} \in \mathbb{R}^{3\times 1}$
- $z^{[2]}, a^{[2]} \in \mathbb{R}^{2\times 1}$
- $z^{[3]}, a^{[3]} \in \mathbb{R}^{1\times 1}$

How Many Parameters? II

- $z^{[1]}, a^{[1]} \in \mathbb{R}^{3\times 1}$
- $z^{[2]}, a^{[2]} \in \mathbb{R}^{2\times 1}$
- $z^{[3]}, a^{[3]} \in \mathbb{R}^{1\times 1}$
- $W^{[1]}$?

$$z^{[1]} = W^{[1]}x^{(i)} = \mathbb{R}^{3\times 1} \text{ written as: } \mathbb{R}^{3\times 1} = \mathbb{R}^{?\times ?} \times \mathbb{R}^{n\times 1}$$

How Many Parameters? II

- $z^{[1]}, a^{[1]} \in \mathbb{R}^{3\times 1}$
- $z^{[2]}, a^{[2]} \in \mathbb{R}^{2\times 1}$
- $z^{[3]}, a^{[3]} \in \mathbb{R}^{1\times 1}$
- $W^{[1]}$?

$$z^{[1]} = W^{[1]} x^{(i)} = \mathbb{R}^{3\times 1} \text{ written as: } \mathbb{R}^{3\times 1} = \mathbb{R}^{?\times ?} \times \mathbb{R}^{n\times 1}$$

- $?\times?$ must be $3 \times n$ and the bias is of size 3

How Many Parameters? III

For each hidden layer:

- $W^{[2]} \in \mathbb{R}^{2\times 3}$, $b^{[2]} \in \mathbb{R}^{2\times 1}$

How Many Parameters? III

For each hidden layer:

- $W^{[2]} \in \mathbb{R}^{2\times 3}$, $b^{[2]} \in \mathbb{R}^{2\times 1}$
- $W^{[3]} \in \mathbb{R}^{1\times 2}$, $b^{[3]} \in \mathbb{R}^{1\times 1}$

How Many Parameters? III

For each hidden layer:

- $W^{[2]} \in \mathbb{R}^{2\times 3}$, $b^{[2]} \in \mathbb{R}^{2\times 1}$
- $W^{[3]} \in \mathbb{R}^{1\times 2}$, $b^{[3]} \in \mathbb{R}^{1\times 1}$
- We have $3n + 3$ in the first layer, $2 \times 3 + 2$ in the second layer and $2 + 1$ in the third layer. Total: $3n + 14$ parameters

Initialization

- zero? $W^{[1]}x^{(i)} + b^{[1]} = 0^{3\times 1}x^{(i)} + 0^{3\times 1}$

Initialization

- zero? $W^{[1]}x^{(i)} + b^{[1]} = 0^{3\times 1}x^{(i)} + 0^{3\times 1}$
- This will cause problems later on when we try to update these parameters (gradients will all be the same)

Initialization

- zero? $W^{[1]}x^{(i)} + b^{[1]} = 0^{3\times1}x^{(i)} + 0^{3\times1}$
- This will cause problems later on when we try to update these parameters (gradients will all be the same)
- same non-zero value?, each activation vector will be the same. Each neuron will receive the exact same gradient update value (symmetry, all neurons will learn the same thing)

Initialization

- zero? $W^{[1]}x^{(i)} + b^{[1]} = 0^{3\times 1}x^{(i)} + 0^{3\times 1}$
- This will cause problems later on when we try to update these parameters (gradients will all be the same)
- same non-zero value?, each activation vector will be the same. Each neuron will receive the exact same gradient update value (symmetry, all neurons will learn the same thing)
- Solution is to randomly initialize the parameters to small values, normally distributed, $N(0, 0.01)$

Xavier /He Initialization

Something better than random initialization

$$w^{[l]} \approx N\left(0, \sqrt{\frac{2}{n^{[l]} + n^{[l-1]}}}\right)$$

- $n^{[l]}$ is the number of neurons in layer l

Xavier /He Initialization

Something better than random initialization

$$w^{[l]} \approx N\left(0, \sqrt{\frac{2}{n^{[l]} + n^{[l-1]}}}\right)$$

- $n^{[l]}$ is the number of neurons in layer l
- For a single layer, consider the variance of the input to the layer as $\sigma^{(in)}$ and the variance of the output (activations) as $\sigma^{(out)}$

Xavier /He Initialization

Something better than random initialization

$$w^{[l]} \approx N\left(0, \sqrt{\frac{2}{n^{[l]} + n^{[l-1]}}}\right)$$

- $n^{[l]}$ is the number of neurons in layer l
- For a single layer, consider the variance of the input to the layer as $\sigma^{(in)}$ and the variance of the output (activations) as $\sigma^{(out)}$
- Xavier/He initialization encourages $\sigma^{(in)}$ to be similar to $\sigma^{(out)}$

Loss Function

After a single forward pass through the NN, the output will be a predicted value $\hat{y}$. We can then compute the loss $\mathcal{L}$, log loss:

$$\mathcal{L}(\hat{y}, y) = -\left[(1-y)\log(1-\hat{y}) + y\log\hat{y}\right]$$

Given this value, we now must update all parameters in layers of the NN. For any given layer index l, we update them

$$\begin{aligned} W^{[l]} &= W^{[l]} - \alpha\frac{\partial \mathcal{L}}{\partial W^{[l]}} \\ b^{[l]} &= b^{[l]} - \alpha\frac{\partial \mathcal{L}}{\partial b^{[l]}} \end{aligned}$$

α is the learning rate. We must compute the gradient with respect to the parameters: $\partial\mathcal{L}/\partial W^{[l]}$ and $\partial\mathcal{L}/\partial b^{[l]}$.

Optimization

- NN parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$

Optimization

- NN parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$
- In order to update them we use stochastic gradient descent (SGD)

Optimization

- NN parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$
- In order to update them we use stochastic gradient descent (SGD)
- We will first compute the gradient with respect to $W^{[3]}$

Optimization

- NN parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$
- In order to update them we use stochastic gradient descent (SGD)
- We will first compute the gradient with respect to $W^{[3]}$
- $W^{[3]}$ is closer to the output $\hat{y}$ in terms of number of computations

Chain Rule

Chain Rule

If $y = f(u)$ and $u = g(x)$, i.e., $y = f(g(x))$, then:

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx} = \frac{df(u)}{du}\frac{dg(x)}{dx}$$

Chain Rule - Example

$$y = f(u) = 5u^4$$
$$\frac{dy}{du} = 20u^3$$

$$u = g(x) = x^3 + 7$$
$$\frac{du}{dx} = 3x^2$$

$$\frac{dy}{dx} = 20(x^3 + 7)^3 \cdot 3x^2$$

Computing $\partial \mathcal{L}/\partial W^{[3]}$

$$
\begin{aligned}
\frac{\partial \mathcal{L}}{\partial W^{[3]}} &= -\frac{\partial}{\partial W^{[3]}}\left((1-y)\log(1-\hat{y}) + y\log\hat{y}\right) \\
&= -(1-y)\frac{\partial}{\partial W^{[3]}}\log\left(1-g(W^{[3]}a^{[2]}+b^{[3]})\right) \\
&\quad -y\frac{\partial}{\partial W^{[3]}}\log\left(g(W^{[3]}a^{[2]}+b^{[3]})\right) \\
&= -(1-y)\frac{1}{1-g(W^{[3]}a^{[2]}+b^{[3]})}(-1)g'(W^{[3]}a^{[2]}+b^{[3]}){a^{[2]}}^T \\
&\quad -y\frac{1}{g(W^{[3]}a^{[2]}+b^{[3]})}g'(W^{[3]}a^{[2]}+b^{[3]}){a^{[2]}}^T
\end{aligned}
$$

Computing $\partial \mathcal{L} / \partial W^{[3]}$ II

$$
\begin{aligned}
&= (1-y)\sigma(W^{[3]}a^{[2]} + b^{[3]}){a^{[2]}}^T - y(1-\sigma(W^{[3]}a^2 + b^{[3]})){a^{[2]}}^T \\
&= (1-y)a^{[3]}{a^{[2]}}^T - y(1-a^{[3]}){a^{[2]}}^T \\
&= (a^{[3]} - y){a^{[2]}}^T
\end{aligned}
$$

- The derivative of the sigmoid: $g' = \sigma' = \sigma(1-\sigma)$
- $a^{[3]} = \sigma(W^{[3]}a^{[2]} + b^{[3]})$

Computing $\partial\mathcal{L}/\partial W^{[2]}$

We can use the chain rule of calculus.

$$\frac{\partial\mathcal{L}}{\partial W^{[2]}} = \frac{\partial\mathcal{L}}{?}\frac{?}{\partial W^{[2]}}$$

We know that $\mathcal{L}$ depends on $\hat{y} = a^{[3]}$, thus

$$\frac{\partial\mathcal{L}}{\partial W^{[2]}} = \frac{\partial\mathcal{L}}{\partial a^{[3]}}\frac{\partial a^{[3]}}{?}\frac{?}{\partial W^{[2]}}$$

We know that $a^{[3]}$ is directly related to $z^{[3]}$

$$\frac{\partial\mathcal{L}}{\partial W^{[2]}} = \frac{\partial\mathcal{L}}{\partial a^{[3]}}\frac{\partial a^{[3]}}{\partial z^{[3]}}\frac{\partial z^{[3]}}{?}\frac{?}{\partial W^{[2]}}$$

Computing $\partial \mathcal{L}/\partial W^{[2]}$ II

Furthermore we know that $z^{[3]}$ is directly related to $a^{[2]}$.
A common element is required for backpropagation

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{?} \frac{?}{\partial W^{[2]}}$$

Again, $a^{[2]}$ depends on $z^{[2]}$, which $z^{[2]}$ directly depends on $W^{[2]}$, which allows us to complete the chain:

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \frac{\partial \mathcal{L}}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}} \frac{\partial z^{[3]}}{\partial a^{[2]}} \frac{\partial a^{[2]}}{\partial z^{[2]}} \frac{\partial z^{[2]}}{\partial W^{[2]}}$$

Computing $\partial \mathcal{L}/\partial W^{[2]}$ III

Recall:

$$\frac{\partial \mathcal{L}}{\partial W^{[3]}} = (a^{[3]} - y)a^{[2]}$$

Since we computed $\partial \mathcal{L}/\partial W^{[3]}$ first, we know that $a^{[2]} = \partial z^{[3]}/\partial W^{[3]}$. Similarly we have $(a^{[3]} - y) = \partial \mathcal{L}/\partial z^{[3]}$. These can help us compute $\partial \mathcal{L}/\partial W^{[2]}$. We substitute:

$$\frac{\partial \mathcal{L}}{\partial W^{[2]}} = \underbrace{\frac{\partial \mathcal{L}}{\partial a^{[3]}} \frac{\partial a^{[3]}}{\partial z^{[3]}}}_{(a^{[3]}-y)} \underbrace{\frac{\partial z^{[3]}}{\partial a^{[2]}}}_{W^{[3]}} \underbrace{\frac{\partial a^{[2]}}{\partial z^{[2]}}}_{g'(z^{[2]})} \underbrace{\frac{\partial z^{[2]}}{\partial W^{[2]}}}_{a^{[1]}} = (a^{[3]} - y)W^{[3]}g'(z^{[2]})a^{[1]}$$

Computing $\partial\mathcal{L}/\partial W^{[2]}$ IV

The order of matrix multiplication in previous equation is not clear. We must reorder:

$$\underbrace{\frac{\partial\mathcal{L}}{\partial W^{[2]}}}_{2\times 3} = \underbrace{(a^{[3]} - y)}_{1\times 1}\underbrace{W^{[3]}}_{1\times 2}\underbrace{g'(z^{[2]})}_{2\times 1}\underbrace{a^{[1]}}_{3\times 1}$$

Using matrix algebra, the correct ordering is

$$\underbrace{\frac{\partial\mathcal{L}}{\partial W^{[2]}}}_{2\times 3} = \underbrace{W^{[3]^T}}_{2\times 1}\circ\underbrace{g'(z^{[2]})}_{2\times 1}\underbrace{(a^{[3]} - y)}_{1\times 1}\underbrace{a^{[1]^T}}_{1\times 3}$$

Computing $\partial \mathcal{L} / \partial W^{[1]}$

- Exercise (next week)
- It is important to use intermediate results we have computed for $\partial \mathcal{L} / \partial W^{[2]}$ and $\partial \mathcal{L} / \partial W^{[3]}$

Optimization: Gradient descent

Gradient descent, for any single layer l, the update rule is defined as:

$$W^{[l]} = W^{[l]} - \sigma \frac{\partial J}{\partial W^{[l]}}$$

- J is the cost function $J = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}^{(i)}$ and $\mathcal{L}^{(i)}$ is the loss for a single example

Optimization: Gradient descent

Gradient descent, for any single layer l, the update rule is defined as:

$$W^{[l]} = W^{[l]} - \sigma \frac{\partial J}{\partial W^{[l]}}$$

- J is the cost function $J = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}^{(i)}$ and $\mathcal{L}^{(i)}$ is the loss for a single example
- The difference between the GD update vs the SGD version is that the cost function J GD gives more accurate gradients whereas $\mathcal{L}^{(i)}$ may be noisy

Optimization: Gradient descent

Gradient descent, for any single layer l, the update rule is defined as:

$$W^{[l]} = W^{[l]} - \sigma \frac{\partial J}{\partial W^{[l]}}$$

- J is the cost function $J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}^{(i)}$ and $\mathcal{L}^{(i)}$ is the loss for a single example
- The difference between the GD update vs the SGD version is that the cost function J GD gives more accurate gradients whereas $\mathcal{L}^{(i)}$ may be noisy
- In GD it can be difficult to compute all activations for all examples in a single forward or backward propagation phase

Optimization: Gradient descent

Gradient descent, for any single layer l, the update rule is defined as:

$$W^{[l]} = W^{[l]} - \sigma \frac{\partial J}{\partial W^{[l]}}$$

- J is the cost function $J = \frac{1}{m}\sum_{i=1}^{m} \mathcal{L}^{(i)}$ and $\mathcal{L}^{(i)}$ is the loss for a single example
- The difference between the GD update vs the SGD version is that the cost function J GD gives more accurate gradients whereas $\mathcal{L}^{(i)}$ may be noisy
- In GD it can be difficult to compute all activations for all examples in a single forward or backward propagation phase
- In the mini-batch gradient descent, $J_{mb} = \frac{1}{B}\sum_{i=1}^{B} \mathcal{L}^{(i)}$, where B is the number of examples in the mini-batch

Parameters Analysis

We have initialized the parameters and have optimized the parameters. We evaluate the trained model and observe that it achieves 96% accuracy on the training set but only 64% on the testing set. Solutions?

- Collecting more data

Parameters Analysis

We have initialized the parameters and have optimized the parameters. We evaluate the trained model and observe that it achieves 96% accuracy on the training set but only 64% on the testing set. Solutions?

- Collecting more data
- Employing regularization

Parameters Analysis

We have initialized the parameters and have optimized the parameters. We evaluate the trained model and observe that it achieves 96% accuracy on the training set but only 64% on the testing set. Solutions?

- Collecting more data
- Employing regularization
- Making the model shallower

L2 Regularization

W denote all the parameters. The $L2$ regularization add another term to the cost function:

$$\begin{aligned} J_{L2} &= J + \frac{\lambda}{2}\|W\|^2 \\ &= J + \frac{\lambda}{2}\sum_{ij}|W_{ij}|^2 \\ &= J + \frac{\lambda}{2}W^T W \end{aligned}$$

J is the standard cost function from before, λ is an arbitrary value with a larger value indicating more regularization and W contains all the weight matrices. The update rule with L2 regularization becomes

$$\begin{aligned} W &= W - \alpha\frac{\partial J}{\partial W} - \alpha\frac{\lambda}{2}\frac{\partial W^T W}{\partial W} \\ &= (1-\alpha\lambda)W - \alpha\frac{\partial J}{\partial W} \end{aligned}$$

L2 Regularization

- When we were updating parameters using GD, we did not $(1-\alpha\lambda)W$ term

L2 Regularization

- When we were updating parameters using GD, we did not $(1-\alpha\lambda)W$ term
- With L2 regularization, every update include some penalization, depending on W

L2 Regularization

- When we were updating parameters using GD, we did not $(1 - \alpha\lambda)W$ term
- With L2 regularization, every update include some penalization, depending on W
- This penalization increases the cost J, which encourages individual parameters to be small in magnitude

L2 Regularization

- When we were updating parameters using GD, we did not $(1 - \alpha\lambda)W$ term
- With L2 regularization, every update include some penalization, depending on W
- This penalization increases the cost J, which encourages individual parameters to be small in magnitude
- This is a way to reduce overfitting

General Backpropagation

Forward Propagation (Andrew Ng)

Given input x, we define $a^{[0]} = x$. Then for layer $l = 1, 2, \ldots N$, where N is the number of layers of NN, we have:

- $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$

Forward Propagation (Andrew Ng)

Given input x, we define $a^{[0]} = x$. Then for layer $l = 1, 2, \ldots N$, where N is the number of layers of NN, we have:

- $z^{[l]} = W^{[l]} a^{[l-1]} + b^{[l]}$
- $a^{[l]} = g^{[l]}(z^{[l]})$

Nonlinearities

- We assume nonlinearities $g^{[l]}$ are the same for all layers besides layer N. Why?

Nonlinearities

- We assume nonlinearities $g^{[l]}$ are the same for all layers besides layer N. Why?
- In the output layer we may be doing regression (hence $g(x) = x$)

Nonlinearities

- We assume nonlinearities $g^{[l]}$ are the same for all layers besides layer N. Why?
- In the output layer we may be doing regression (hence $g(x) = x$)
- or binary classification ($g(x) = sigmoid(x)$)

Nonlinearities

- We assume nonlinearities $g^{[l]}$ are the same for all layers besides layer N. Why?
- In the output layer we may be doing regression (hence $g(x) = x$)
- or binary classification ($g(x) = sigmoid(x)$)
- or Multiclass classification ($g(x) = softmax(x)$)

Loss function

Given the output of the NN $a^{[N]}$, also denoted as $\hat{y}$, we measure the loss $J(W, b) = \mathcal{L}(a^{[N]}, y) = \mathcal{L}(\hat{y}, y)$:

- For real valued regression $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$

Loss function

Given the output of the NN $a^{[N]}$, also denoted as $\hat{y}$, we measure the loss $J(W, b) = \mathcal{L}(a^{[N]}, y) = \mathcal{L}(\hat{y}, y)$:

- For real valued regression $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$
- For binary classification using log. regression $\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$

Loss function

Given the output of the NN $a^{[N]}$, also denoted as $\hat{y}$, we measure the loss $J(W, b) = \mathcal{L}(a^{[N]}, y) = \mathcal{L}(\hat{y}, y)$:

- For real valued regression $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$
- For binary classification using log. regression $\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$
- For softmax regression over k classes, we use cross entropy $\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{k} \mathbf{1}\{y = j\} \log \hat{y}_j$. Where $\hat{y}$ is a k-dimensional vector.

Loss function

Given the output of the NN $a^{[N]}$, also denoted as $\hat{y}$, we measure the loss $J(W, b) = \mathcal{L}(a^{[N]}, y) = \mathcal{L}(\hat{y}, y)$:

- For real valued regression $\mathcal{L}(\hat{y}, y) = \frac{1}{2}(\hat{y} - y)^2$
- For binary classification using log. regression $\mathcal{L}(\hat{y}, y) = -(y \log \hat{y} + (1 - y) \log(1 - \hat{y}))$
- For softmax regression over k classes, we use cross entropy $\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{k} \mathbf{1}\{y = j\} \log \hat{y}_j$. Where $\hat{y}$ is a k-dimensional vector.
- If we use y to instead denote the k dimensional vector of zeros with a single 1 at lth position, cross entropy can also be expressed as $\mathcal{L}(\hat{y}, y) = -\sum_{j=1}^{k} y_j \log \hat{y}$

Backpropagation

Let us define

$$\delta^{[l]} = \nabla_{z^{[l]}} \mathcal{L}(\hat{y}, y)$$

We can define a three step recipe for computing the gradients with respect to every $W^{[l]}, b^{[l]}$

Step 1

For output layer N, we have

$$\delta^{[N]} = \nabla_{z^{[N]}} \mathcal{L}(\hat{y}, y)$$

Sometimes we may compute this term directly (e.g $g^{[N]}$ is the softmax funct.), whereas other times ($g^{[N]}$ is sigmoid) we can apply the chain rule:

$$\nabla_{z^{[N]}} \mathcal{L}(\hat{y}, y) = \nabla_{\hat{y}} \mathcal{L}(\hat{y}, y) o (g^{N})'(z^{[N]})$$

where, $(g^{N})'(z^{[N]})$ denotes the element wise derivative w.r.t. z^{N}

Step 2

For $l = N-1, N-2, \ldots, 1$, we have

$$\delta^{[l]} = (W^{[l+1]T}\delta^{[l+1]}) o g'(z^{[l]})$$

where o, denotes the elementwise product.

Step 3

Finally, we can compute the gradients for layer l as

$$\nabla_{W^{[l]}} J(W, b) = \delta^{[l]} a^{[l-1]T}$$

$$\nabla_{b^{[l]}} J(W, b) = \delta^{[l]}$$