Deep Learning

1. Logistic regression with a NN mindset
2. Neural Networks
3. Backpropagation
4. Improving your NN

\[
\begin{bmatrix}
X_1 \\
\vdots \\
x_d
\end{bmatrix}
\rightarrow
\begin{bmatrix}
0 \\
0 \\
0 \\
0
\end{bmatrix}
\rightarrow \hat{y}
\]

\[
z^{[1]} = W^{[1]} x + b^{[1]}
\]
\[
a^{[1]} = \sigma(z^{[1]})
\]
\[
z^{[2]} = W^{[2]} a^{[1]} + b^{[2]}
\]
\[
a^{[2]} = \sigma(z^{[2]})
\]
\[
z^{[3]} = W^{[3]} a^{[2]} + b^{[3]}
\]
\[
a^{[3]} = \sigma(z^{[3]})
\]
\[
\hat{y} = a^{[3]}
\]

Optimizing \( w^{[1]}, w^{[2]}, w^{[3]}, b^{[1]}, b^{[2]}, b^{[3]} \)

Loss (Cost) \( f^* J(\hat{y}, y) = \frac{1}{n} \sum_{i=1}^{n} L^{(i)} \)

\[
L^{(i)} = - [ y^{(i)} \log \hat{y}^{(i)} + (1-y^{(i)}) \log (1-\hat{y}^{(i)}) ]
\]

\[
\frac{\partial J}{\partial w^{[2]}} \quad \frac{\partial L^{(i)}}{\partial w^{[2]}}
\]

Backward Propagation

\[
\forall l = 1 \ldots 3
\]
\[
w^{(l)} = w^{(l)} - \alpha \frac{\partial J}{\partial w^{(l)}}
\]
\[
b^{(l)} = b^{(l)} - \alpha \frac{\partial J}{\partial b^{(l)}}
\]
\[
\frac{\partial J}{\partial w^{(2)}} = \frac{1}{n} \sum_{i=1}^{n} \frac{\partial L^{(i)}}{\partial w^{(2)}}
\]

Try: compute \( \frac{\partial J}{\partial w^{(1)}} \), \( \frac{\partial J}{\partial b^{(1)}} \)

**Improving your NN**

**Activation functions**

- **Sigmoid** \( \sigma(z) = \frac{1}{1+e^{-z}} \)
  
  \( \sigma'(z) = \sigma(z)(1-\sigma(z)) \)

- **ReLU** \( z \geq 0 \) \( \sigma(z) = z \), \( z < 0 \) \( \sigma(z) = 0 \)
  
  \( \text{ReLU}'(z) = 1 \) if \( z > 0 \)
  
  replaces \( a^{(j)}(1-a^{(j)}) \) with \( a^{(j)} \)
\[
\tanh (z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}
\]
\[
\tanh'(z) = 1 - \tanh(z)^2
\]

8. Initialization Methods

Normalization input

\[
\mu = \frac{1}{n} \sum_{i=1}^{n} x^{(i)}
\]
\[
\Sigma = \frac{1}{n} \sum_{i=1}^{n} x^{(i)} x^{(i)\top}
\]
\[
x = \Sigma^{-1} x
\]

Vanishing / Exploding gradients

Assume \( b = 0 \)

Activation \( f \) \( : \mathbb{R} \rightarrow \mathbb{R} \)

\[
y = w[k] a[l-1] = w[k] w[l-1] a[l-2]
\]
\[
= w[k] w[l-1] \ldots w[1] x
\]
\[ w^{(t)} = \begin{bmatrix} 1.5 & 0 \\ 0 & 1.5 \end{bmatrix} \sim \begin{bmatrix} 1.5^L & 0 \\ 0 & 1.5^L \end{bmatrix} \]
\[ \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \sim \begin{bmatrix} 0.5^L & 0 \\ 0 & 0.5^L \end{bmatrix} \]

- Avoid by initializing \( w \) close to 1

Example \( w \), one neuron

\[ a = o(z) \]
\[ z = w_1 x_1 + \ldots + w_d x_d \]

large \( d \) \( \rightarrow \) small \( w \)

\[ w_i \sim \frac{1}{\sqrt{d}} \]

\[ w_L = \text{np.random.randn}(\text{shape}) \times \text{npsqrt}(\frac{1}{n^{(L-1)}}) \]

for \( \text{ReLU} \) : 2 instead of 1

**Xavier Initialization**

\[ w^{(L)} \sim \sqrt{\frac{1}{n^{(L-1)}}} \] for \( \tanh \)

**He Initialization**

\[ w^{(L)} \sim \sqrt{\frac{2}{n^{(L)} + n^{(L-1)}}} \]

\( n^{(L)} \) forward prop.

\( n^{(L-1)} \) backward prop.

**Optimization**

**Gradient Descent**

Mini batch gradient Descent
GD + Momentum

\[
\begin{align*}
W &= W - \alpha \frac{\partial L}{\partial W} \\
\n\n\end{align*}
\]

\[
\begin{align*}
\dot{W} &= \beta \cdot \dot{W} + (1 - \beta) \frac{\partial L}{\partial W} \\
\end{align*}
\]

\[\text{\(U\): momentum — weighted average of past updates}\]