DEEPLEARNING.AI-C1

Won

Deep Learning

COURSE 1

NEURAL NETWORKS AND DEEP LEARNING

Logistic Regression with a Neural Network mindset

Mathematical expression of the algorithm:

For one example $x^{(i)}$:

$z^{(i)} = w^T x^{(i)} + b$

$\hat{y}^{(i)} = a^{(i)} = sigmoid(z^{(i)})$

$sigmoid(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}$

$\mathcal{L}(a^{(i)}, y^{(i)}) = - y^{(i)} \log(a^{(i)}) - (1 - y^{(i)}) \log(1 - a^{(i)})$

The cost is then computed by summing over all training examples:

$$J = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}(a^{(i)}, y^{(i)})$$

Key steps:

In this exercise, you will carry out the following steps:

- Initialize the parameters of the model
- Learn the parameters for the model by minimizing the cost
- Use the learned parameters to make predictions (on the test set)
- Analyse the results and conclude

Forward and Backward propagation

Forward Propagation:

- You get X
- You compute $A = \sigma(w^T X + b) = (a^{(1)}, a^{(2)}, ..., a^{(m-1)}, a^{(m)})$
- You calculate the cost function: $J = -\frac{1}{m}\sum_{i=1}^{m} y^{(i)}\log(a^{(i)}) + (1-y^{(i)})\log(1-a^{(i)})$

Here are the two formulas you will be using:

$$\frac{\partial J}{\partial w} = \frac{1}{m}X(A-Y)^T$$

$$\frac{\partial J}{\partial b} = \frac{1}{m}\sum_{i=1}^{m}(a^{(i)} - y^{(i)})$$

What to remember

1. Preprocessing the dataset is important.
2. You implemented each function separately: initialize(), propagate(), optimize(). Then you built a model().
3. Tuning the learning rate (which is an example of a "hyperparameter") can make a big difference to the algorithm. You will see more examples of this later in this course!

NEURAL NETWORK

You've learnt to:

- Build a complete neural network with a hidden layer
- Make a good use of a non-linear unit
- Implemented forward propagation and back propagation, and trained a neural network
- See the impact of varying the hidden layer size, including overfitting.

Mathematically:

For one example $x^{(i)}$:

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1](i)}$$

$$a^{[1](i)} = \tanh(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2](i)}$$

$$\hat{y}^{(i)} = a^{[2](i)} = \sigma(z^{[2](i)})$$

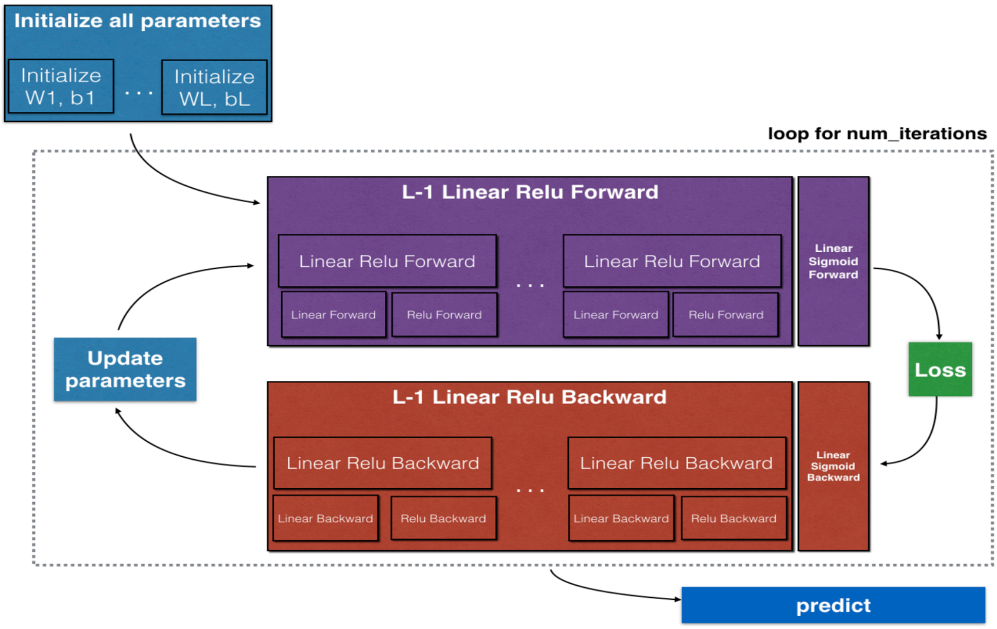
$$y^{(i)}_{prediction} = \begin{cases} 1 & if \quad a^{[2](i)} > 0.5 \\ 0 & otherwise \end{cases}$$

Summary of gradient descent

$dz^{[2]} = a^{[2]} - y$	$dZ^{[2]} = A^{[2]} - Y$
$dW^{[2]} = dz^{[2]} a^{[1]^T}$	$dW^{[2]} = \frac{1}{m} dZ^{[2]} A^{[1]^T}$
$db^{[2]} = dz^{[2]}$	$db^{[2]} = \frac{1}{m} np.sum(dZ^{[2]}, axis = 1, keepdims = True)$
$dz^{[1]} = W^{[2]T} dz^{[2]} * g^{[1]\prime}(z^{[1]})$	$dZ^{[1]} = W^{[2]T} dZ^{[2]} * g^{[1]\prime}(Z^{[1]})$
$dW^{[1]} = dz^{[1]} x^T$	$dW^{[1]} = \frac{1}{m} dZ^{[1]} X^T$
$db^{[1]} = dz^{[1]}$	$db^{[1]} = \frac{1}{m} np.sum(dZ^{[1]}, axis = 1, keepdims = True)$

Reminder: The general methodology to build a Neural Network is to:

1. Define the neural network structure (# of input units, # of hidden units, etc).
2. Initialize the model's parameters
3. **Loop**:
    - Implement forward propagation
    - Compute loss
    - Implement backward propagation to get the gradients
    - Update parameters (gradient descent)
    - Use trained parameters to predict labels



$$W = \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix} \quad X = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \quad b = \begin{bmatrix} s \\ t \\ u \end{bmatrix}$$

Then $WX+b$ will be:

$$WX + b = \begin{bmatrix} (ja + kd + lg) + s & (jb + ke + lh) + s & (jc + kf + li) + s \\ (ma + nd + og) + t & (mb + ne + oh) + t & (mc + nf + oi) + t \\ (pa + qd + rg) + u & (pb + qe + rh) + u & (pc + qf + ri) + u \end{bmatrix}$$

Loop

1_ Forward propagation

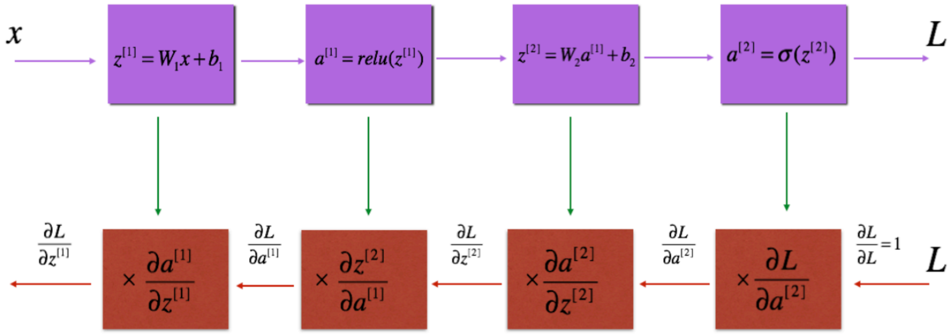
$$Z^{[l]} = W^{[l]} A^{[l-1]} + b^{[l]}$$

- **Sigmoid**: $\sigma(Z) = \sigma(WA + b) = \frac{1}{1 + e^{-(WA+b)}}$.
- **ReLU**: The mathematical formula for ReLu is $A = RELU(Z) = max(0, Z)$

2_ Cost function

$$-\frac{1}{m}\sum \text{limits}_{i=1}^{m} (y^{(i)} \log(a^{[L](i)}) + (1 - y^{(i)}) \log(1 - a^{[L](i)}))$$

3_ Backward propagation



The three outputs $(dW^{[l]}, db^{[l]}, dA^{[l]})$ are computed using the input $dZ^{[l]}$ .Here are the formulas you need:

$$
\begin{aligned}
dW^{[l]} &= \frac{\partial \mathcal{L}}{\partial W^{[l]}} = \frac{1}{m} dZ^{[l]} A^{[l-1]T} \\
db^{[l]} &= \frac{\partial \mathcal{L}}{\partial b^{[l]}} = \frac{1}{m} \sum_{i=1}^{m} dZ^{[l](i)} \\
dA^{[l-1]} &= \frac{\partial \mathcal{L}}{\partial A^{[l-1]}} = W^{[l]T} dZ^{[l]}
\end{aligned}
$$

If $g(.)$ is the activation function, `sigmoid_backward` and `relu_backward` compute

$$dZ^{[l]} = dA^{[l]} * g'(Z^{[l]})$$

.

4_ Update Parameters

using gradient descent:

$$
\begin{aligned}
W^{[l]} &= W^{[l]} - \alpha\, dW^{[l]} \\
b^{[l]} &= b^{[l]} - \alpha\, db^{[l]}
\end{aligned}
$$

where α is the learning rate.