# Deep Learning!

Meichen Lu (ml574@cam.ac.uk)

April 19, 2018

## Contents



# 1 Classic neural network

## 1.1 Components

Intuition: 'automatic' feature engineering and discovery! Each unit is like a neuron in the brain.

### 1.1.1 Structure

- Input features: input layer
- Output neuron: output layer
- Everything in the middle: hidden layers (with hidden units)

### 1.1.2 Neurons

Each neuron 'evaluates' the input according to a certain features and is represented as a function. It is called **activation function**. We can break down the function into two distinct computations: (1) $z = w^T x + b$ and (2) $a = g(z)$. Examples of $g(z)$ are

- Sigmoid: $g(z) = \frac{1}{1+e^{-z}}$
- ReLu: $g(z) = \max(z, 0)$
- tanh: $g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

### 1.1.3 Log likelihood

The probability from sigmoid function in the final layer is the log-likelihood function

$$\sum_{i=1}^{m} \left( y^{(i)} \log \hat{y}^{(i)} + (1 - y^{(i)}) \log(1 - \hat{y}^{(i)}) \right) \tag{1}$$

### 1.1.4 Notations

- input: $\boldsymbol{x} = [x_1, x_2, ...]$
  - Sample is noted as $\boldsymbol{x}^{(i)}$
- Linear combination of inputs: $\boldsymbol{z}$
- Neuron output: $\boldsymbol{a}$
- Layer notation: $\boldsymbol{z}^{[1]}$ for first layer
- Final output: $y$
- Predicted output: $\hat{y}$

Putting it all together, the first first hidden unit in the first hidden layer performs the following computations:

$$z_1^{[1]} = \boldsymbol{W}_1^{[1]^T} \boldsymbol{x} + b_1^{[1]} \qquad \text{and} \qquad a_1^{[1]} = g(z_1^{[1]}) \tag{2}$$

## 1.2 Vectorisation

- Fully training set input:

$$X = \begin{pmatrix} | & | & & | \\ \boldsymbol{x}^{(1)} & \boldsymbol{x}^{(2)} & \cdots & \boldsymbol{x}^{(m)} \\ | & | & & | \end{pmatrix}$$

- Fully stacked weight matrix

$$W^{[1]} = \begin{bmatrix} - & \boldsymbol{W}_1^{[1]^T} & - \\ - & \boldsymbol{W}_2^{[1]^T} & - \\ & \vdots & \\ - & \boldsymbol{W}_{n_1}^{[1]^T} & - \end{bmatrix}$$

- Fully projection in the first layer from the inputs $X$ is thus

$$Z^{[1]} = \begin{pmatrix} | & | & & | \\ \boldsymbol{z}^{[1](1)} & \boldsymbol{z}^{[1](2)} & \cdots & \boldsymbol{z}^{[1](m)} \\ | & | & & | \end{pmatrix} = W^{[1]} X + b^[1]$$

### 1.3 Back-propagation

#### 1.3.1 Weights initialization

Usually, we randomly initialize the parameters to small values (e.g., normally distributed around zero; $\mathcal{N}(0, 0.1)$) In practice, it turns out there is something better than random initializa- tion. It is called Xavier/He initialization and initializes the weights:

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

#### 1.3.2 Loss function

**With regularisation**

$$J(W, b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W, b; \boldsymbol{x}^{(i)}, y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{[l]}\right)^2 \tag{3}$$

The weight decay parameter $\lambda$ controls the relative importance of the two terms.

#### 1.3.3 Gradient descend

**Update rules** for gradient descent

$$W_{ij}^{[l]} = W_{ij}^{[l]} - \alpha\frac{\partial}{\partial W_{ij}^{[l]}} J(W, b) \tag{4}$$

$$b_i^{[l]} = b_i^{[l]} - \alpha\frac{\partial}{\partial b_i^{[l]}} J(W, b) \tag{5}$$

Or with regularization

$$\frac{\partial}{\partial W_{ij}^{[l]}} J(W, b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial W_{ij}^{[l]}} J(W, b; \boldsymbol{x}^{(i)}, y^{(i)})\right] + \lambda W_{ij}^{[l]} \tag{6}$$

$$\frac{\partial}{\partial b_i^{[l]}} J(W, b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial b_i^{[l]}} J(W, b; \boldsymbol{x}^{(i)}, y^{(i)}) \tag{7}$$

**Stochastic gradient descend** Calculating the influence on gradient from a single example, using the cost function:

$$J(W, b; \boldsymbol{x}, y) = \frac{1}{2}\left\|h_{W,b}(\boldsymbol{x}) - y\right\|^2 \tag{8}$$

1. Perform a feedforward pass up to the output layer $L_{n_l}$
2. For each output unit $i$ in the output layer $n_l$, compute

$$\delta_i^{[n_l]} = \frac{\partial}{\partial z_i^{[n_l]}}\ \frac{1}{2}\left\|y - h_{W,b}(\boldsymbol{x})\right\|^2 = -(y_i - a_i^{[n_l]}) \cdot f'(z_i^{[n_l]}) \tag{9}$$

3. For $l = n_l - 1, n_l - 2, ..., 2$
   For each node $i$ in layer $l$, set

$$\delta_i^{[l]} = \left( \sum_{j=1}^{s_{l+1}} W_{ji}^{[l]} \delta_j^{[l+1]} \right) f'(z_i^{[l]}) \tag{10}$$

4. Compute the partial derivatives:

$$\frac{\partial}{\partial W_{ij}^{[l]}} J(W, b; \boldsymbol{x}, y) = a_j^{[l]} \delta_i^{[l+1]} \tag{11}$$

$$\frac{\partial}{\partial b_i^{[l]}} J(W, b; \boldsymbol{x}, y) = \delta_i^{[l+1]}. \tag{12}$$

Or in the vectorised form

1. Feedforward
2. $\delta^{[n_l]} = -(y - a^{[n_l]}) \circ f'(z^{[n_l]})$
3. $\delta^{[l]} = \left((W^{[l]})^T \delta^{[l+1]}\right) \circ f'(z^{[l]})$
4. $\nabla_{W^{[l]}} J(W, b; \boldsymbol{x}, y) = \delta^{[l+1]} (a^{[l]})^T$

$\circ$ is the Hadamard product, i.e., element-wise multiplication