Deep Learning

Lecture 3

Recap

- Classification
    - Logistic Regression
        - Sigmoid function
        - Cross entropy loss function
- Neural Networks
    - Perceptron
    - Layers
    - Non-Linear activation function

Agenda

- Neural Network Learning
  - Feed forward
  - Back propagation
- Practical Machine Learning
  - Data distribution
  - Overfitting
  - Regularization
  - Data Normalization
  - Hyper parameter Tuning

Computation Graph: Logistic Regression and Gradient

$$z = w^T x + b$$

$$\hat{y} = a = \sigma(z)$$

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

Logistic regression derivatives

$$\begin{tikzcd}
x_1 \arrow[ddr] & & & \\
w_1 \arrow[dr] & & & \\
x_2 \arrow[r] & \boxed{z = w_1 x_1 + w_2 x_2 + b} \arrow[r] & \boxed{a = \sigma(z)} \arrow[r] & \boxed{\mathcal{L}(\mathrm{a}, y)} \\
w_2 \arrow[ur] & & & \\
b \arrow[uur] & & &
\end{tikzcd}$$

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial a} * \frac{\partial a}{\partial z} = a - y$$

$$\frac{\partial a}{\partial z} = a(1 - a) \qquad \frac{\partial \mathcal{L}}{\partial a} = -\frac{y}{a} + \frac{1 - y}{1 - a}$$

Logistic regression on m examples

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

$$a^{i} = \hat{y} = \sigma\left(z^{i}\right) = \sigma(w^{T}x^{i} + b)$$

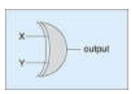
$$\frac{\partial}{\partial w_i}J(\theta, b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\partial}{\partial w_i}\mathcal{L}(a^{i}, y^{i})$$

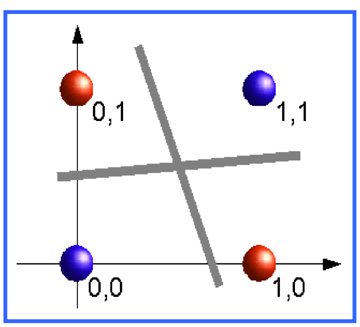
Artificial Neural Networks

Learning highly non-linear functions

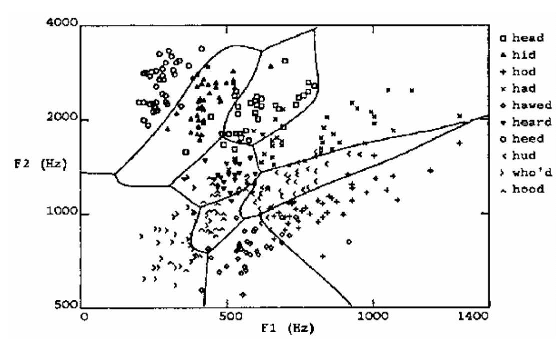
f: X → Y

- f might be non-linear function
- X (vector of) continuous and/or discrete vars
- Y (vector of) continuous and/or discrete vars

The XOR gate 



Speech recognition



A Recipe for Machine Learning

1. Given training data 1:m

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}$$

2. Choose each of these:
   - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1} \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^{N}$$

2. Choose each of th

   – Decision function

   $$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   – Loss function

   $$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$\boldsymbol{y}_i)$

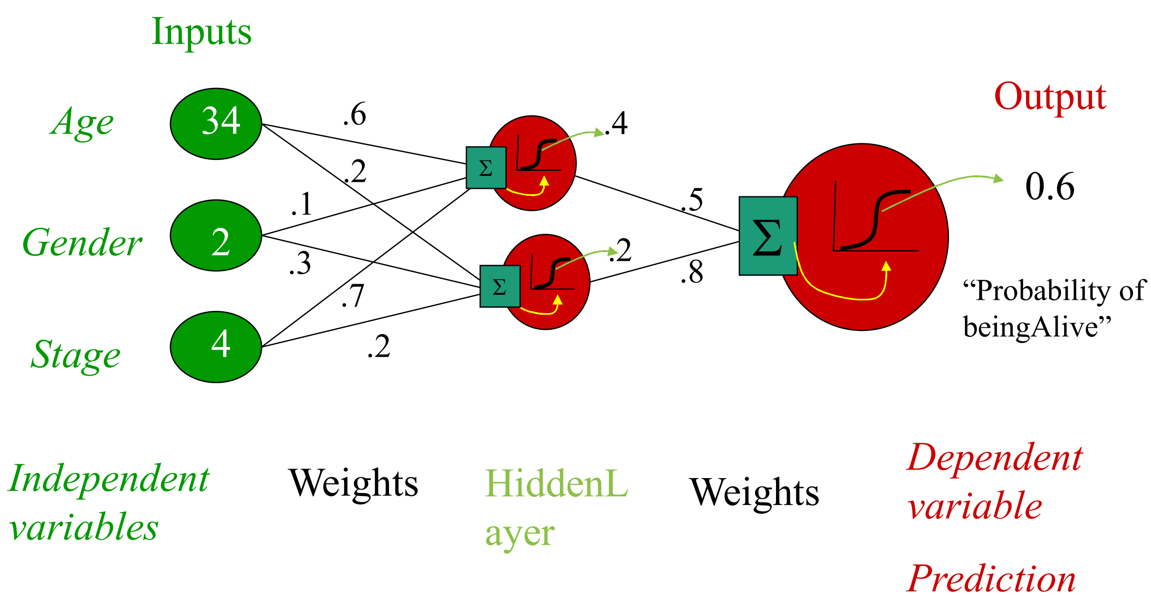
Gradients

Backpropagation can compute this gradient!

the gradient)

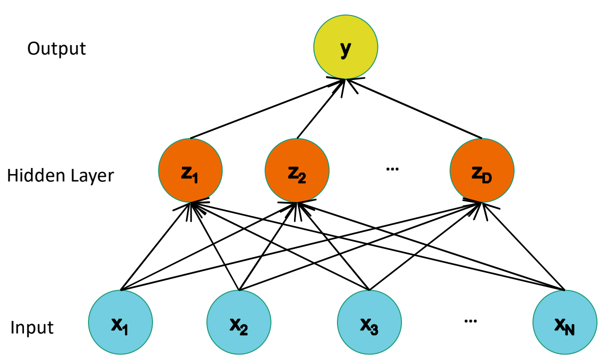
$$\boldsymbol{\theta}^{(t} \quad {}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Neural Network Model



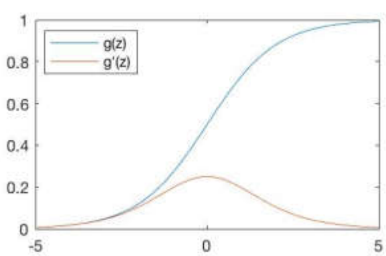
Activation Functions

Neural Network with arbitrary
nonlinear activation functions



Common Activation Functions

Sigmoid Function

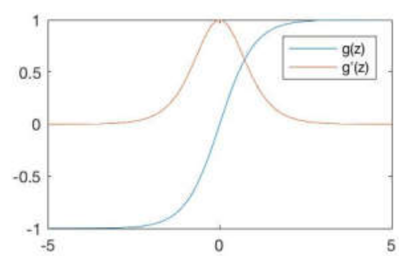


$$g(z) = \frac{1}{1 + e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

`tf.nn.sigmoid(z)`

Hyperbolic Tangent

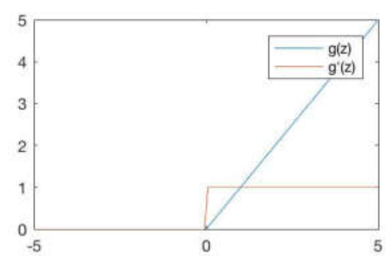


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

`tf.nn.tanh(z)`

Rectified Linear Unit (ReLU)



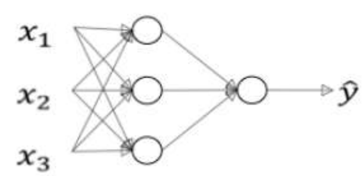
$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

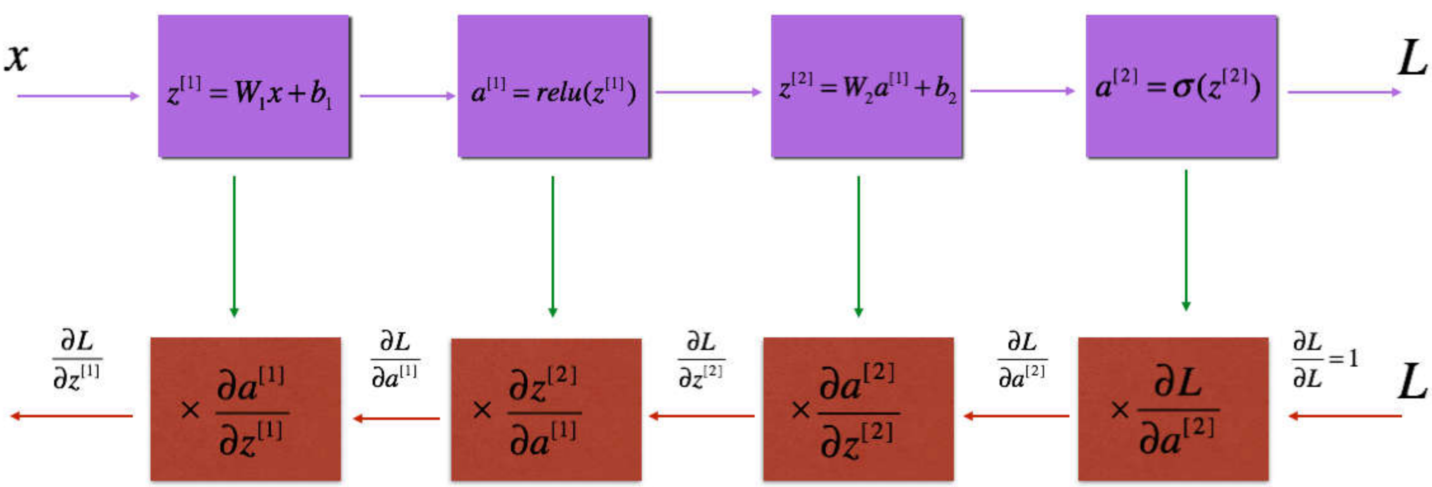
`tf.nn.relu(z)`

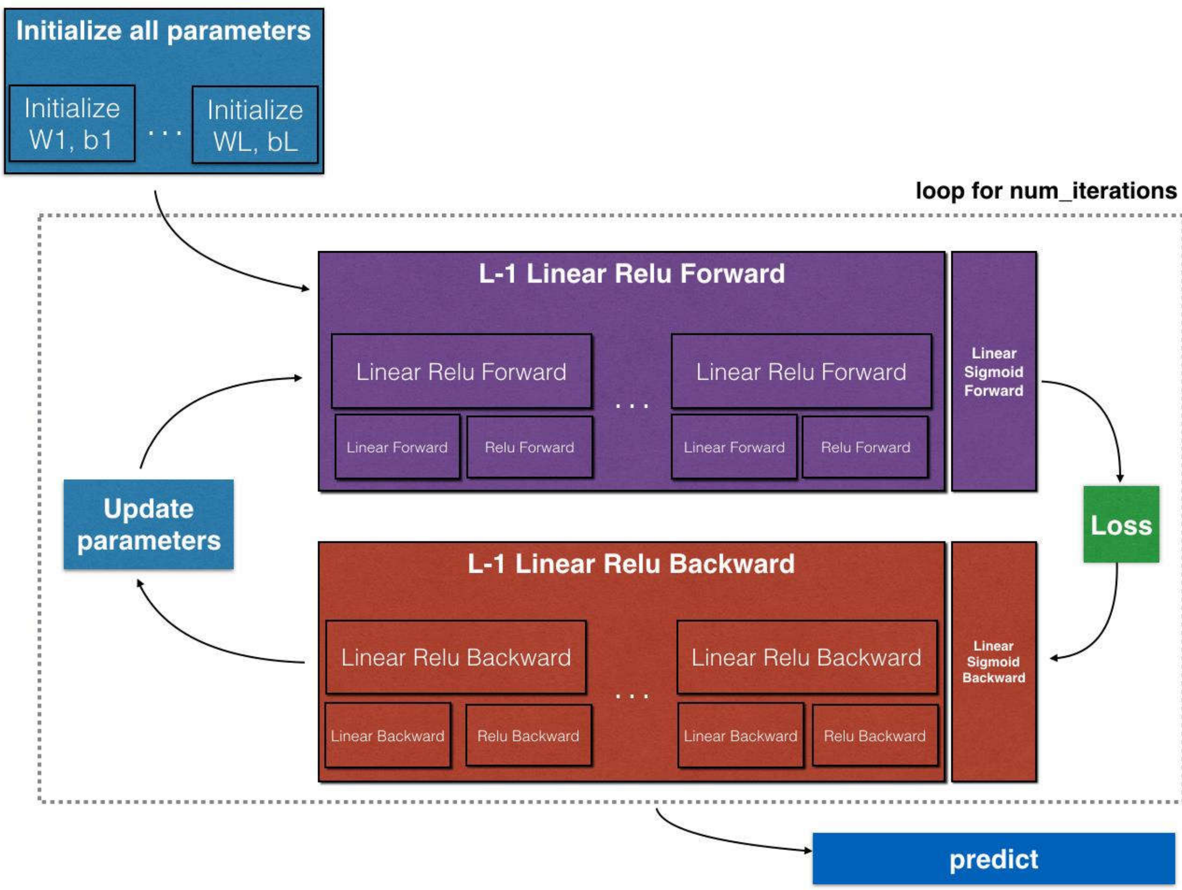
Parameters W[l] and b[l]

- $z^{[1]} = W^{[1]}X + \ b^{[1]}$
- $a^{[1]} = \ g^{[1]}\left(z^{[1]}\right)$
- $z^{[1]} = W^{[2]}a^{[1]} + \ b^{[2]}$
- $\hat{y} = a^{[2]} = \ g^{[2]}\left(z^{[2]}\right)$



1 hidden layer


x
$z^{[1]} = W_1 x + b_1$
$a^{[1]} = relu(z^{[1]})$
$z^{[2]} = W_2 a^{[1]} + b_2$
$a^{[2]} = \sigma(z^{[2]})$
L
$\frac{\partial L}{\partial z^{[1]}}$
$\times \frac{\partial a^{[1]}}{\partial z^{[1]}}$
$\frac{\partial L}{\partial a^{[1]}}$
$\times \frac{\partial z^{[2]}}{\partial a^{[1]}}$
$\frac{\partial L}{\partial z^{[2]}}$
$\times \frac{\partial a^{[2]}}{\partial z^{[2]}}$
$\frac{\partial L}{\partial a^{[2]}}$
$\times \frac{\partial L}{\partial a^{[2]}}$
$\frac{\partial L}{\partial L} = 1$
L


Initialize all parameters
Initialize W1, b1
...
Initialize WL, bL
loop for num_iterations
L-1 Linear Relu Forward
Linear Relu Forward
Linear Forward
Relu Forward
...
Linear Relu Forward
Linear Forward
Relu Forward
Linear Sigmoid Forward
Loss
Update parameters
L-1 Linear Relu Backward
Linear Relu Backward
Linear Backward
Relu Backward
...
Linear Relu Backward
Linear Backward
Relu Backward
Linear Sigmoid Backward
predict

Forward and backpropagation

$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$\vdots$$
$$A^{[L]} = g^{[L]}\left(Z^{[L]}\right) = \hat{Y}$$

$$dZ^{[L]} = A^{[L]} - Y$$
$$dW^{[L]} = \frac{1}{m} dZ^{[L]} {A^{[L]}}^T$$
$$db^{[L]} = \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[L]}, axis = 1, keepdims = True)$$
$$dZ^{[L-1]} = d{W^{[L]}}^T dZ^{[L]} g'^{[L]}(Z^{[L-1]})$$
$$\vdots$$
$$dZ^{[1]} = d{W^{[L]}}^T dZ^{[2]} g'^{[1]}(Z^{[1]})$$
$$dW^{[1]} = \frac{1}{m} dZ^{[1]} {A^{[1]}}^T$$
$$db^{[1]} = \frac{1}{m} np.\mathrm{sum}(\mathrm{d}Z^{[1]}, axis = 1, keepdims = True)$$

Summary of Neural Network

1. **Neural Networks**…
   - provide a way of learning features
   - are highly nonlinear prediction functions
   - (can be) a highly parallel network of logistic regression classifiers
   - discover useful hidden representations of the input
2. **Backpropagation**…
   - provides an efficient way to compute gradients
   - is a special case of reverse-mode automatic differentiation