INFO 6105
Data Science Engineering Methods and Tools

Lecture 5
Neural Networks

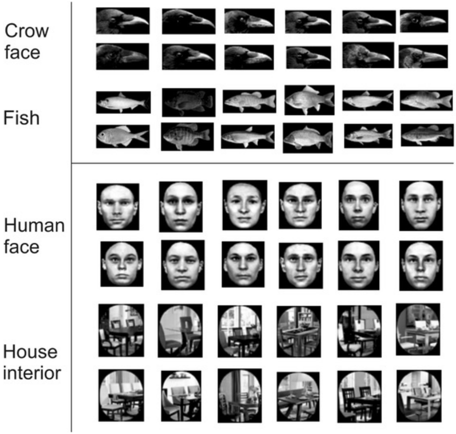
Ebrahim Nasrabadi
nasrabadi@northeastern.edu

College of Engineering
Northeastern University

Spring 2019

Human Face Detection

Face detection: detect whether a digital image is a human face.



[Source: https://doi.org/10.1007/s00359-017-1211-7]

Human Face Detection

Note that

- images are 567×541 pixels
- each image can be represented as a vector of pixel values

Human Face Detection

Note that

- images are 567×541 pixels
- each image can be represented as a vector of pixel values

We can represent the images as

$$(\vec{x}_1, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_n, y_n)$$

where $\vec{x}_i \in \mathbb{R}^{306,747}$ and $y_i \in \{0, 1\}$ for i^{th} images with

$$x_{ij} = \text{the value of color in pixel } j \text{ in image } i$$

$$y_i = \begin{cases} 0 & \text{if it is a non-human face} \\ 1 & \text{otherwise} \end{cases}$$

The face detection is a classification problem.

Logistic Regression

Modeling Choice: We choose

$$P(\text{the image is a human fcae given } \vec{x}) = \frac{1}{1+e^{-\vec{\beta}^T\vec{x}}}$$
$$P(y=1|X) = \frac{1}{1+e^{-\beta^T\vec{x}}}$$

where $f(z) = \frac{1}{1+e^{-z}}$ is the *logistic function* and $\vec{\beta}^T = (\beta_0, \beta_1, \ldots, \beta_m)$ are the *model parameters*.

Logistic Regression

Modeling Choice: We choose

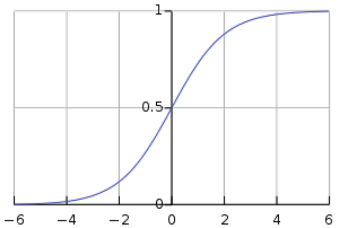
$$P(\text{the image is a human fcae given } \vec{x}) = \frac{1}{1+e^{-\vec{\beta}^T\vec{x}}}$$

$$P(y=1|X) = \frac{1}{1+e^{-\beta^T\vec{x}}}$$

where $f(z) = \frac{1}{1+e^{-z}}$ is the *logistic function* and $\vec{\beta}^T = (\beta_0, \beta_1, \dots, \beta_m)$ are the *model parameters*.

Properties:

- $0 \leq f(z) \leq 1$
- $f(0) = 0.5$
- $f(z) \longrightarrow 1$ as $z \longrightarrow \infty$
- $f(z) \longrightarrow 0$ as $z \longrightarrow -\infty$



How to determine the model parameters?

- Find the model parameters $\vec{\beta}$ such that the predicted probability

$$h_\beta(\vec{x}_i) = P(y_i | \vec{x}_i; \beta)$$

  - is close to one if $y_i = 1$
  - is close to zero if $y_i = 0$

Maximum Likelihood

Note that

$$p(y_i|\vec{x}_i;\beta) = h_\beta(\vec{x}_i)^{y_i} \cdot (1 - h_\beta(\vec{x}_i))^{1-y_i}$$
$$= \begin{cases} h_\beta(\vec{x}_i) & \text{if } y_i = 1 \\ 1 - h_\beta(\vec{x}_i) & \text{if } y_i = 0 \end{cases}$$

Maximum Likelihood

Note that

$$
\begin{aligned}
p(y_i|\vec{x}_i;\beta) &= h_\beta(\vec{x}_i)^{y_i} \cdot (1 - h_\beta(\vec{x}_i))^{1-y_i} \\
&= \begin{cases} h_\beta(\vec{x}_i) & \text{if } y_i = 1 \\ 1 - h_\beta(\vec{x}_i) & \text{if } y_i = 0 \end{cases}
\end{aligned}
$$

Assuming the data is generated independently, the probability of the observing $y_1, \ldots, y_n$ is given by

$$
\begin{aligned}
L(\beta) = P(Y|\vec{x};\beta) &= \prod_{i=1}^{m} P(y_i|\vec{x}_i;\beta) \\
&= \prod_{i:y_i=1} h_\beta(X_i) \cdot \prod_{i:y_i=0} (1 - h_\beta(\vec{x}_i))
\end{aligned}
$$

Maximum Likelihood

Note that

$$p(y_i|\vec{x}_i;\beta) = h_\beta(\vec{x}_i)^{y_i} \cdot (1 - h_\beta(\vec{x}_i))^{1-y_i}$$
$$= \begin{cases} h_\beta(\vec{x}_i) & \text{if } y_i = 1 \\ 1 - h_\beta(\vec{x}_i) & \text{if } y_i = 0 \end{cases}$$

Assuming the data is generated independently, the probability of the observing $y_1, \ldots, y_n$ is given by

$$L(\beta) = P(Y|\vec{x};\beta) = \prod_{i=1}^{m} P(y_i|\vec{x}_i;\beta)$$
$$= \prod_{i:y_i=1} h_\beta(X_i) \cdot \prod_{i:y_i=0} (1 - h_\beta(\vec{x}_i))$$

Choose β so as to maximize $L(\beta)$ or equivalently minimize $-\log L(\beta)$.

Maximum Likelihood

Choose β so as to maximize $L(\beta)$ or equivalently minimize $-\frac{1}{n}\log L(\beta)$.

$$\mathcal{L}(\vec{\beta}) := -\frac{1}{n}\log L(\beta) = -\frac{1}{n}\sum_i \left[y_i \log(h_\beta(x_i)) + (1-y_i)\log(1-h_\beta(x_i))\right]$$

where $h_\beta(x_i) = \frac{1}{1+e^{-\beta^T \vec{x}_i}}$.
This does not have an explicit solution, we need to minimize numerically!

Batch Gradient Descent

Start with some initial β and repeatedly take a step in the direction of steepest decrease of $-\log L(\beta)$:

$$\beta_j = \beta_j - \eta \frac{\partial}{\partial \beta_j} \mathcal{L}(\vec{\beta})$$

η is called *learning rate*.

Batch Gradient Descent

Start with some initial β and repeatedly take a step in the direction of steepest decrease of $-\log L(\beta)$:

$$\beta_j = \beta_j - \eta \frac{\partial}{\partial \beta_j} \mathcal{L}(\vec{\beta})$$

η is called *learning rate*.

Update Rule: For training examples $(\vec{x}_1, y_1), \ldots, (\vec{x}_n, y_n)$:

$$\beta_j = \beta_j - \frac{\eta}{n} \sum_{i=0}^{n} (h_\beta(\vec{x}_i) - y_i) x_{ij} \qquad\qquad j = 0, 1, \ldots, m$$

Note: The magnitude of the update is proportional to the error term

Stochastic Gradient Descent

In practice, n can be very large and hence the batch gradient descent can be costly.

Stochastic Gradient Descent

In practice, n can be very large and hence the batch gradient descent can be costly.

An alternative approach is to scan the training data one by one and update the parameters with respect to a single training example. For a single training example (x, y), the update rule is

$$\beta_j = \beta_j - \frac{\eta}{n}(h_\beta(\vec{x}) - y_i)x_j \qquad\qquad j = 0, 1, \ldots, m$$

Stochastic Gradient Descent

In practice, n can be very large and hence the batch gradient descent can be costly.

An alternative approach is to scan the training data one by one and update the parameters with respect to a single training example. For a single training example (x, y), the update rule is

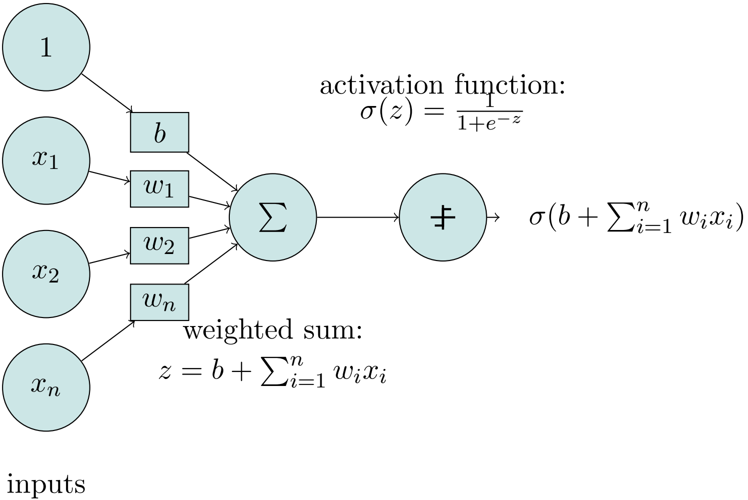
$$\beta_j = \beta_j - \frac{\eta}{n}(h_\beta(\vec{x}) - y_i)x_j \qquad j = 0, 1, \ldots, m$$

For $i = 1, \ldots, m$, update β

$$\beta_j = \beta_j - \frac{\eta}{n}(h_\beta(\vec{x}_i) - y_i)x_{ij} \qquad j = 0, 1, \ldots, m$$

Often, this approach gets β close to the optimal value much faster than batch gradient descent, in particular when the training dataset is large.

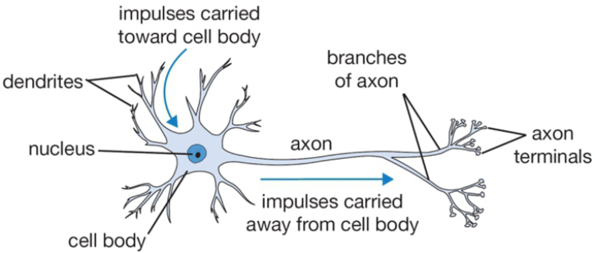
Logistic regression as a Neural Network



The reasoning for this notational difference is conform with standard neural network notation.

Inspiration: The Brain

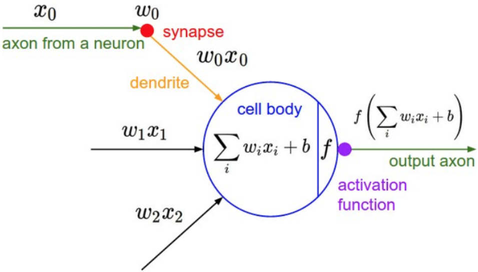
- Neural networks are inspired by biology, e.g., the (human) brain
- Our brain has approximately 86 billion neurons, each of which communicates (is connected) to about $10,000$ other neurons
- The basic computational unit of the brain is called neuron



A cartoon drawing of a biological neuron
[Source: http://cs231n.github.io/neural-networks-1/]

Mathematical Model of a Neuron

- Artificial neurons are called units



A mathematical model of the neuron in a neural network
[Source: http://cs231n.github.io/neural-networks-1/]

Face Detection

- Logistic regression can be viewed as a neural network with a single neuron

Face Detection

- Logistic regression can be viewed as a neural network with a single neuron
- A single neuron is not enough to detect a human face

Face Detection

- Logistic regression can be viewed as a neural network with a single neuron
- A single neuron is not enough to detect a human face
- A more complex neural network is required:

Face Detection

- Logistic regression can be viewed as a neural network with a single neuron
- A single neuron is not enough to detect a human face
- A more complex neural network is required:
  - take the single neuron described above and "stack" them together such that one neuron passes its output as input into the next neuron, resulting in a more complex function.

Face Detection

- Logistic regression can be viewed as a neural network with a single neuron
- A single neuron is not enough to detect a human face
- A more complex neural network is required:
  - take the single neuron described above and "stack" them together such that one neuron passes its output as input into the next neuron, resulting in a more complex function.
  - take a composition of many different functions

Face Detection

Let us now deepen the face detection example.

- Given the image features, we might decide that the problem of face detection depends on whether the image has

Face Detection

Let us now deepen the face detection example.

- Given the image features, we might decide that the problem of face detection depends on whether the image has
  - left eye

Face Detection

Let us now deepen the face detection example.

- Given the image features, we might decide that the problem of face detection depends on whether the image has
  - left eye
  - right eye

Let us now deepen the face detection example.

- Given the image features, we might decide that the problem of face detection depends on whether the image has
  - left eye
  - right eye
  - nose

Face Detection

Let us now deepen the face detection example.

- Given the image features, we might decide that the problem of face detection depends on whether the image has
  - left eye
  - right eye
  - nose
  - mouth

Face Detection

Let us now deepen the face detection example.

- Given the image features, we might decide that the problem of face detection depends on whether the image has
  - left eye
  - right eye
  - nose
  - mouth
- These features can be formulated as functions of the image features.

Face Detection

Let us now deepen the face detection example.

- Given the image features, we might decide that the problem of face detection depends on whether the image has
  - left eye
  - right eye
  - nose
  - mouth
- These features can be formulated as functions of the image features.
- Given these four derived features, we may conclude that the face detection depends on these four features.

- We have described this neural network as if we already have the insight to determine these four factors (left eye, right eye, nose, mouth) ultimately affect the decision.

Neural Networks

- We have described this neural network as if we already have the insight to determine these four factors (left eye, right eye, nose, mouth) ultimately affect the decision.
- Part of the magic of a neural network is that all we need are the input features x and the output y while the neural network will figure out everything in the middle by itself.

Neural Network Formulation

Suppose we have

- four input features x_1, x_2, x_3, x_4 which are collectively called the input layer,
- five hidden units which are collectively called the hidden layer
- one output unit called the output layer

Neural Network Formulation

Suppose we have

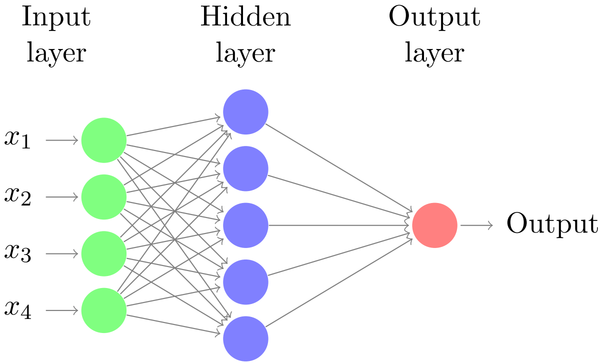
- four input features x_1, x_2, x_3, x_4 which are collectively called the input layer,
- five hidden units which are collectively called the hidden layer
- one output unit called the output layer

Notes:

- The term hidden layer is called "hidden" because we do not have the ground truth/training value for the hidden units.
- This is in contrast to the input and output layers, both of which we know the ground truth values from $(\vec{x}_i, y_i)$.

Neural Network Architecture

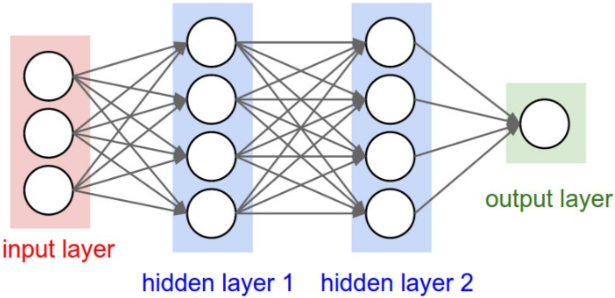
A neural network with one layer of five hidden units:



- Each unit computes its value based on linear combination of values of units that point into it, and an activation function
- a 2-layer neural network (One layer of hidden units, One output layer)

Multi-Layer Neural Network

A 3-layer neural net with 3 input units, 4 hidden units in the first and second hidden layer and 1 output unit



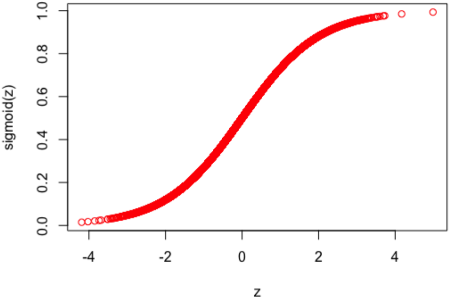
source: http://cs231n.github.io/neural-networks-1/

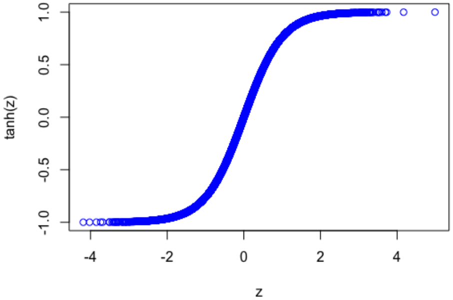
- A N-layer neural network has
  - N -1 layers of hidden units and
  - One output layer

Activation Functions

Most commonly used activation functions:

- Sigmoid: $\sigma(z) = \frac{1}{1+\exp(-z)}$
- Tanh: $\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(-z)+\exp(-z)}$
- ReLU (Rectified Linear Unit): $\mathrm{ReLU}(z) = \max(0, z)$



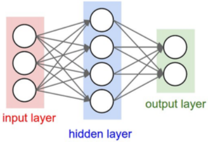


Neural Network Training

We only need to know two algorithms

- Forward pass: calculate the model output
- Backward pass: update the model parameters

Forward pass



Output of the network can be written as:

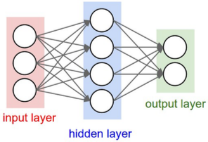
- Hidden layer:

$$h_j(x) = f(b_j^{[1]} + \sum_{i=1}^{D} x_i w_{ji}^{[1]})$$

- Output layer:

$$o_k(x) = g(b_k^{[2]} + \sum_{j=1}^{J} h_j(x) w_{kj}^{[2]})$$

Forward pass



Output of the network can be written as:

- Hidden layer:

$$h_j(x) = f(b_j^{[1]} + \sum_{i=1}^{D} x_i w_{ji}^{[1]})$$

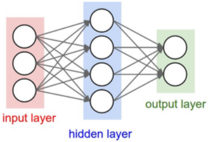
- Output layer:

$$o_k(x) = g(b_k^{[2]} + \sum_{j=1}^{J} h_j(x) w_{kj}^{[2]})$$

where

- j indexing hidden units, k indexing the output units, D number of inputs
- Activation functions f , g are sigmoid/logistic, tanh, or rectified linear (ReLU)

Forward pass

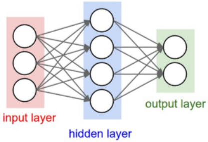


- Computation for the first hidden unit in the first hidden layer:

$$z_1^{[1]} = W_1^{[1]} x + b_1^{[1]} \text{ and } a_1^{[1]} = f(z_1^{[1]})$$

  - $W \in \mathbb{R}^{3\times 4}$ is a matrix of parameters and W_1 refers to the first row of this matrix.
  - The parameters associated with the first hidden unit is the vector $W^{[1]} \in \mathbb{R}^3$ and the scalar $b^{[1]} \in \mathbb{R}$.

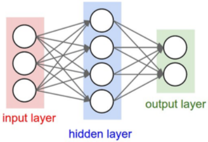
Forward pass



- Computation for the second, third, forth hidden units in the first hidden layer:

$$z_2^{[1]} = W_2^{[1]} x + b_2^{[1]} \text{ and } a_2^{[1]} = f(z_2^{[1]})$$
$$z_3^{[1]} = W_3^{[1]} x + b_3^{[1]} \text{ and } a_3^{[1]} = f(z_3^{[1]})$$
$$z_4^{[1]} = W_4^{[1]} x + b_4^{[1]} \text{ and } a_4^{[1]} = f(z_4^{[1]})$$

Forward pass



- Computation for the second, third, forth hidden units in the first hidden layer:

$$z_2^{[1]} = W_2^{[1]}x + b_2^{[1]} \text{ and } a_2^{[1]} = f(z_2^{[1]})$$
$$z_3^{[1]} = W_3^{[1]}x + b_3^{[1]} \text{ and } a_3^{[1]} = f(z_3^{[1]})$$
$$z_4^{[1]} = W_4^{[1]}x + b_4^{[1]} \text{ and } a_4^{[1]} = f(z_4^{[1]})$$

- Computation for the output layer:

$$z_1^{[2]} = W_1^{[2]}a^{[1]} + b_1^{[2]} \text{ and } a_1^{[2]} = g(z_1^{[2]})$$
$$z_2^{[2]} = W_2^{[2]}a^{[1]} + b_2^{[2]} \text{ and } a_2^{[2]} = g(z_2^{[2]})$$

where $a^{[1]}$ is defined as the concatenation of all first layer

Forward pass

Forward pass can be implemented efficiently using matrix operations in Python:

$$\underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ \vdots \\ z_4^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{4\times 1}} = \underbrace{\begin{bmatrix} - & W_1^{[1]^T} & - \\ - & W_2^{[1]^T} & - \\ & \vdots & \\ - & W_4^{[1]^T} & - \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{4\times 3}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}}_{x \in \mathbb{R}^{3\times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_4^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{4\times 1}}$$

[source: cs229.stanford.edu/notes/cs229-notes-deep_learning.pdf]

Training Neural Networks

- Find weights W to minimize a loss function

Training Neural Networks

- Find weights W to minimize a loss function
- Loss functions:

where o_i is the output given $\vec{x}_i$.

Training Neural Networks

- Find weights W to minimize a loss function
- Loss functions:
  - Squared loss for regression: $\sum_{i=1}^{n}(y_i - o_i)^2$

  where o_i is the output given $\vec{x}_i$.

Training Neural Networks

- Find weights W to minimize a loss function
- Loss functions:
  - Squared loss for regression: $\sum_{i=1}^{n}(y_i - o_i)^2$
  - Cross-entropy for classification: $\sum_{i=1}^{n} -y_i \log o_i - (1 - y_i) \log (1 - o_i)$

  where o_i is the output given $\vec{x}_i$.

Training Neural Networks

- Find weights W to minimize a loss function
- Loss functions:
  - Squared loss for regression: $\sum_{i=1}^{n}(y_i - o_i)^2$
  - Cross-entropy for classification: $\sum_{i=1}^{n} -y_i \log o_i - (1 - y_i) \log(1 - o_i)$

  where o_i is the output given $\vec{x}_i$.
- Gradient descent:

$$W^{t+1} = W^t - \eta \frac{\partial J}{\partial W^t}$$

  where η is the learning rate and J is the loss function

Useful Derivatives

name	function	derivative
Sigmoid	$\sigma(z) = \frac{1}{1+\exp(-z)}$	$\sigma(z) \cdot (1 - \sigma(z))$
Tanh	$\tanh(z) = \frac{\exp(z)-\exp(-z)}{\exp(z)+\exp(-z)}$	$1/\cosh^2(z)$
ReLU	$\mathrm{ReLU}(z) = \max(0, z)$	$\begin{cases} 1, & \text{if } z > 0 \\ 0, & \text{if } z \leq 0 \end{cases}$

Training Neural Networks

- Back-propagation: an efficient method for computing gradients needed to perform gradient-based optimization of the weights in a multi-layer network

Loop until convergence:

- For each example $(\vec{x}_i, y_i)$
  1. Forward pass: Given input $\vec{x}_i$, propagate activity forward $(x_i \rightarrow \mathrm{a}_i^{[1]} \rightarrow o_i)$
  2. Backward pass: Propagate gradients backward
  3. Update each weight via gradient descent