Basics of Training Neural Nets
Logistic Regression – Simplest Network

\[
h(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}}
\]
Logistic Regression – Simplest Network

\[ h(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}} \]

Can model class probabilities

\[ P(y = 1|w, b, x) = h(w^T x + b) \]

\[ P(y = 0|w, b, x) = 1 - h(w^T x + b) \]
Classification with Logistic Regression

Can model class probabilities

\[ P(y = 1|w, b, x) = h(w^T x + b) \]

\[ P(y = 0|w, b, x) = 1 - h(w^T x + b) \]

e.g.,

\( y=1 \Rightarrow \text{Cat} \)
\( y=0 \Rightarrow \text{Dog} \)
Classification with Logistic Regression

Can model class probabilities

\[ P(y = 1 | w, b, x) = h(w^T x + b) \]

\[ P(y = 0 | w, b, x) = 1 - h(w^T x + b) \]

e.g.,

\[ y=1 \Rightarrow \text{Cat} \]
\[ y=0 \Rightarrow \text{Dog} \]
Classification with Logistic Regression

Can model class probabilities

\[ P(y = 1 | w, b, x) = h(w^T x + b) \]

\[ P(y = 0 | w, b, x) = 1 - h(w^T x + b) \]

e.g.,

\( y=1 \) => Cat
\( y=0 \) => Dog
Classification with Logistic Regression

Can model class probabilities

\[ P(y = 1|w, b, x) = h(w^T x + b) \]

\[ P(y = 0|w, b, x) = 1 - h(w^T x + b) \]

e.g.,

\[ y=1 \implies \text{Cat} \]
\[ y=0 \implies \text{Dog} \]
Classification with Logistic Regression

Can model class probabilities

\[ P(y = 1|w, b, x) = h(w^T x + b) \]

\[ P(y = 0|w, b, x) = 1 - h(w^T x + b) \]

e.g.,

\[ y=1 \Rightarrow \text{Cat} \]
\[ y=0 \Rightarrow \text{Dog} \]

The above two equations can be combined as:

\[ P(y|w, b, x) = (h(w^T x + b))^y (1 - h(w^T x + b))^{1-y} \]
Classification with Logistic Regression

Can model class probabilities

\[ P(y = 1|w, b, x) = h(w^T x + b) \]

\[ P(y = 0|w, b, x) = 1 - h(w^T x + b) \]

The above two equations can be combined as:

\[ P(y|w, b, x) = (h(w^T x + b))^y (1 - h(w^T x + b))^{1-y} \]

The overall probability for M training examples:

\[ P(Y|w, b, X) = \prod_{i=1}^{M} (h(w^T x_i + b))^{y_i} (1 - h(w^T x_i + b))^{1-y_i} \]
Classification with Logistic Regression

The overall probability for $M$ training examples:

$$P(Y|w, b, X) = \prod_{i=1}^{M} (h(w^T x_i + b))^{y_i} (1 - h(w^T x_i + b))^{1-y_i}$$

The goal of Logistic Regression:

$$\max_{w,b} P(Y|w, b, X)$$
The overall probability for $M$ training examples:

$$P(Y|w, b, X) = \prod_{i=1}^{M} (h(w^T x_i + b))^{y_i} (1 - h(w^T x_i + b))^{1-y_i}$$

The goal of Logistic Regression:

$$\max_{w, b} P(Y|w, b, X)$$

If we do it correctly, we know, $h$ would be high for cats and low for dogs
Classification with Logistic Regression

The overall probability for M training examples:

$$P(Y|w, b, X) = \prod_{i=1}^{M} (h(w^T x_i + b))^{y_i} (1 - h(w^T x_i + b))^{1-y_i}$$

The goal of Logistic Regression:

$$\max_{w,b} P(Y|w, b, X)$$

If we do it correctly, we know, $h$ would be high for cats and low for dogs.

For a new example, $x$, we compute $h(w^T x + b)$ to get its probability for being a cat.
Multiple Layers

To model more complex relations – we can go deep

Sometimes called Multilayer Perceptron (MLP)


Intuitive explanation:
- Compute approximation error at the output
- Propagate error back by computing individual contributions of parameters to error


Intuitive explanation:
- Compute approximation error at the output
- Propagate error back by computing individual contributions of parameters to error

Very hand wavy
Confusion guaranteed!
Gradient Descent

\[ \max_w f(w) \quad \text{or} \quad \min_w e(w) \]

Gradient descent \[ \min_w e(w) \]

Algorithm: start with \( w_0, \ t = 0 \)

1. Compute gradient \( g_t = \frac{\partial e}{\partial w} \bigg|_{w=w_t} \)
2. Update \( w_{t+1} = w_t - \eta g_t \)
3. Set \( t \leftarrow t + 1 \)
How to get gradients

1. Analytically Compute Gradients

$$\nabla e(w) = \frac{\partial e}{\partial w}$$

If you can do it, that’s the best option!

*Fast, Accurate, Exact!*

**BUT**

*Tedious to derive*
*Tedious to Implement*
*Error prone*
*Tedious to debug*
How to get gradients

2. Numerically Compute Gradients

\[
\frac{\partial e}{\partial w_i} = \frac{e(w_i + \epsilon | w \setminus w_i) - e(w_i - \epsilon | w \setminus w_i)}{2\epsilon}
\]

Trivially Easy to Implement

BUT

Slow and Approximate

2N function calls if N parameters for every gradient descent step (Not Good)
3. Automatic Differentiation (Exploit Chain Rule)

\[ \frac{\partial}{\partial x} g(f(x)) = \frac{\partial g}{\partial f} \cdot \frac{\partial f}{\partial x} \]
A function can be represented as a directed acyclic Computation Graph in terms of primitive functions. Once a function is represented as a computation graph, implementing chain rule becomes an exercise in memoization...
Memoization of Derivatives

While computing a node function in the graph store the derivative of the node output with respect of its input.
Memoization Example: Composition of Functions

The terms in red represent the memoized derivatives stored during the forward pass. They are the derivatives of the respective node’s output with respect to its input.

The terms in blue flow and get collected during the backward pass. They represent the derivative of the main output function with respect to the output of the respective node.
Memoization Example: Addition of functions

Function Summation

The red and blue terms represent the same quantities as in the previous slide. In this example, when two paths merge during backward pass, the sum of the two quantities flow backwards.
Memoization Example: Quadratic

Chain Rule on our Quadratic Example
Now we know how to compute gradients – so run Gradient Descent

\[
\max_w f(w) \text{ or } \min_w e(w)
\]

Gradient descent \( \min_w e(w) \)

Algorithm: start with \( w_0, t = 0 \)

1. Compute gradient \( g_t = \frac{\partial e}{\partial w} \bigg|_{w=w_t} \)
2. Update \( w_{t+1} = w_t - \eta g_t \)
3. Set \( t \leftarrow t + 1 \)
Problems with Back Propagation

Back propagation doesn’t work well for deep sigmoid networks:

- Diffusion of gradient signal (multiplication of many small numbers)
- Attractivity of many local minima (random initialization is very far from good points)
- Requires a lot of training samples
- Need for significant computational power
Tricks to handle problems

To obtain more flexibility/non-linearity we use additional function prototypes:

- Sigmoid
- Rectified linear unit (ReLU)
- Pooling
- Dropout
- Convolutions
Tricks to handle problems

Rectified Linear Unit (ReLU)
- Drop information if smaller than zero
- Fixes the problem of vanishing gradients to some degree

Dropout
- Drop information at random
- Kind of a regularization
Next Time: Indexing for Fast Retrieval