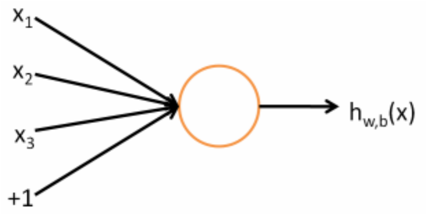
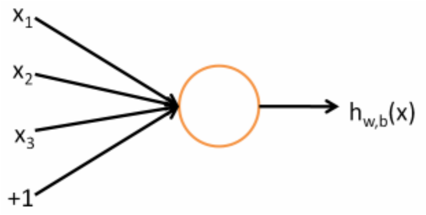
Multilayer Networks

Natural Language Processing: Jordan Boyd-Graber
University of Maryland
SLIDES ADAPTED FROM ANDREW NG

Logistic Regression by Another Name: Map inputs to output



Logistic Regression by Another Name: Map inputs to output

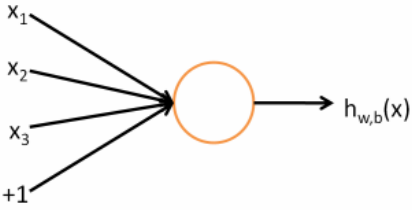


Input

Vector $x_1 \dots x_d$

inputs encoded as real numbers

Logistic Regression by Another Name: Map inputs to output



Input

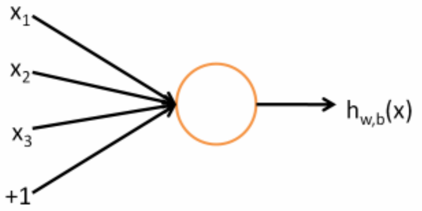
Vector $x_1 \dots x_d$

Output

$$f\left(\sum_i W_i x_i + b\right)$$

multiply inputs by

Logistic Regression by Another Name: Map inputs to output



Input

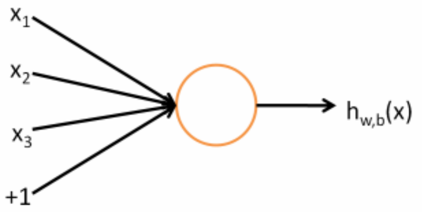
Vector $x_1 \ldots x_d$

Output

$$f\left(\sum_i W_i x_i + b\right)$$

add bias

Logistic Regression by Another Name: Map inputs to output



Input

Vector $x_1 \dots x_d$

Output

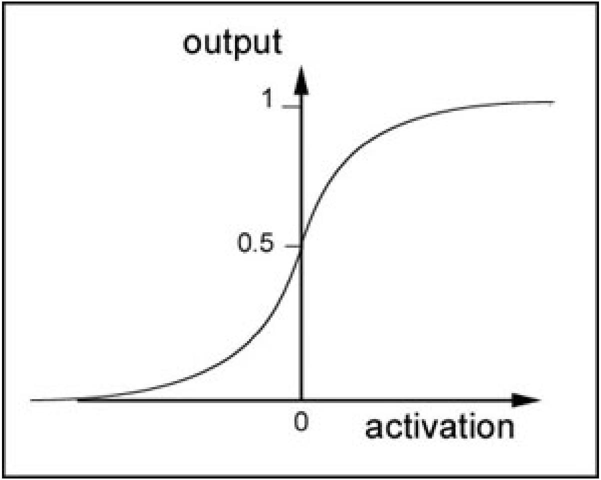
$$f\left(\sum_i W_i x_i + b\right)$$

Activation

$$f(z) \equiv \frac{1}{1 + \exp(-z)}$$

pass through nonlinear sigmoid

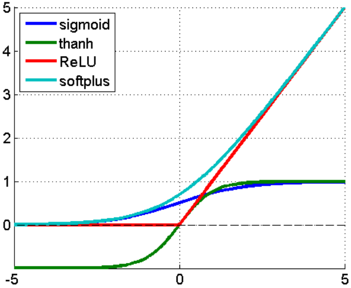
Why is it called activation?



In the shallow end

- This is still logistic regression
- Engineering features x is difficult (and requires expertise)
- Can we learn how to represent inputs into final decision?

Better name: non-linearity



- Logistic / Sigmoid

$$f(x) = \frac{1}{1 + e^{-x}} \quad (1)$$

- tanh

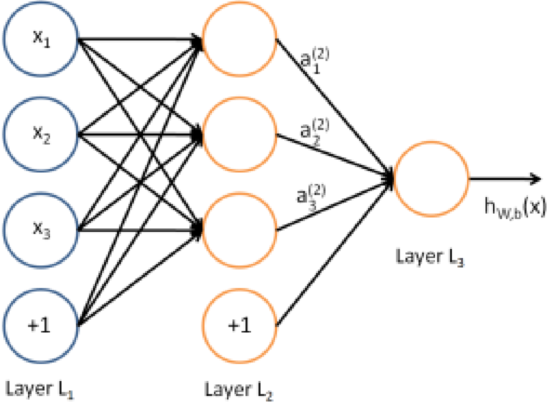
$$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \quad (2)$$

- ReLU

$$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ x & \text{for} \quad x \geq 0 \end{cases} \quad (3)$$

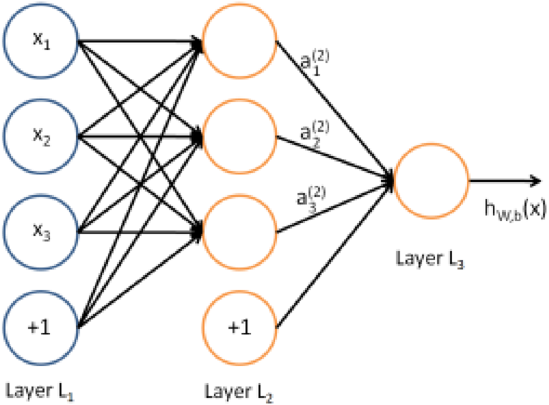
- SoftPlus: $f(x) = \ln(1 + e^x)$

Learn the features and the function



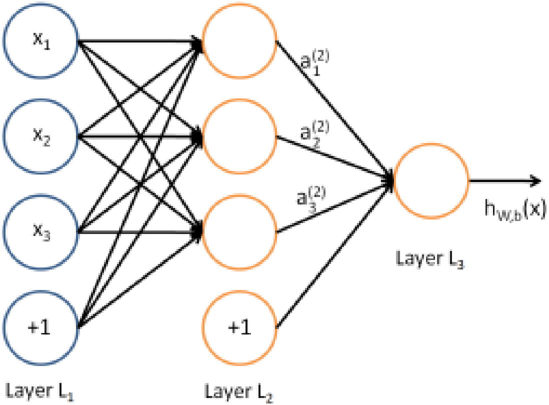
$$a_1^{(2)} = f\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}\right)$$

Learn the features and the function



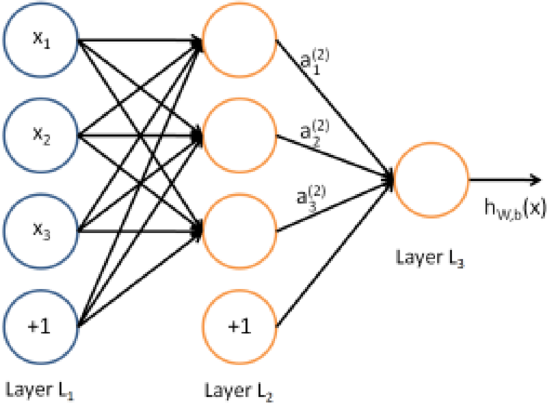
$$a_2^{(2)} = f\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}\right)$$

Learn the features and the function



$$a_3^{(2)} = f\left(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)}\right)$$

Learn the features and the function



$$h_{W,b}(x) = a_1^{(3)} = f\left(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)}\right)$$

Objective Function

- For every example x, y of our supervised training set, we want the label y to match the prediction $h_{W,b}(x)$.

$$J(W,b;x,y) \equiv \frac{1}{2}||h_{W,b}(x) - y||^2 \tag{4}$$

Objective Function

- For every example x, y of our supervised training set, we want the label y to match the prediction $h_{W,b}(x)$.

$$J(W,b;x,y) \equiv \frac{1}{2}||h_{W,b}(x) - y||^2 \tag{4}$$

- We want this value, summed over all of the examples to be as small as possible

Objective Function

- For every example x, y of our supervised training set, we want the label y to match the prediction $h_{W,b}(x)$.

$$J(W, b; x, y) \equiv \frac{1}{2} ||h_{W,b}(x) - y||^2 \tag{4}$$

- We want this value, summed over all of the examples to be as small as possible
- We also want the weights not to be too large

$$\frac{\lambda}{2} \sum_{l}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{l} \right)^2 \tag{5}$$

Objective Function

- For every example x, y of our supervised training set, we want the label y to match the prediction $h_{W,b}(x)$.

$$J(W, b; x, y) \equiv \frac{1}{2} ||h_{W,b}(x) - y||^2 \tag{4}$$

- We want this value, summed over all of the examples to be as small as possible
- We also want the weights not to be too large

$$\frac{\lambda}{2} \sum_{l}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{l} \right)^2 \tag{5}$$

Objective Function

- For every example x, y of our supervised training set, we want the label y to match the prediction $h_{W,b}(x)$.

$$J(W, b; x, y) \equiv \frac{1}{2} ||h_{W,b}(x) - y||^2 \tag{4}$$

- We want this value, summed over all of the examples to be as small as possible
- We also want the weights not to be too large

$$\frac{\lambda}{2} \sum_{l}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{l}\right)^2 \tag{5}$$

Sum over all layers

Objective Function

- For every example x, y of our supervised training set, we want the label y to match the prediction $h_{W,b}(x)$.

$$J(W,b;x,y) \equiv \frac{1}{2}||h_{W,b}(x) - y||^2 \tag{4}$$

- We want this value, summed over all of the examples to be as small as possible
- We also want the weights not to be too large

$$\frac{\lambda}{2} \sum_{l}^{n_l - 1} \color{red}{\sum_{i=1}^{s_l}} \color{black}{\sum_{j=1}^{s_{l+1}} \left(W_{ji}^{l} \right)^2} \tag{5}$$

Sum over all sources

Objective Function

- For every example x, y of our supervised training set, we want the label y to match the prediction $h_{W,b}(x)$.

$$J(W, b; x, y) \equiv \frac{1}{2} ||h_{W,b}(x) - y||^2 \tag{4}$$

- We want this value, summed over all of the examples to be as small as possible
- We also want the weights not to be too large

$$\frac{\lambda}{2} \sum_{l}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{l}\right)^2 \tag{5}$$

Sum over all destinations

Objective Function

Putting it all together:

$$J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} ||h_{W,b}(x^{(i)}) - y^{(i)}||^2 \right] + \frac{\lambda}{2} \sum_{l}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{l} \right)^2 \quad (6)$$

Objective Function

Putting it all together:

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}||h_{W,b}(x^{(i)}) - y^{(i)}||^2\right] + \frac{\lambda}{2}\sum_{l}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{l}\right)^2 \quad (6)$$

- Our goal is to minimize $J(W,b)$ as a function of W and b

Objective Function

Putting it all together:

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}||h_{W,b}(x^{(i)}) - y^{(i)}||^2\right] + \frac{\lambda}{2}\sum_{l}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{l}\right)^2 \tag{6}$$

- Our goal is to minimize $J(W,b)$ as a function of W and b
- Initialize W and b to small random value near zero

Objective Function

Putting it all together:

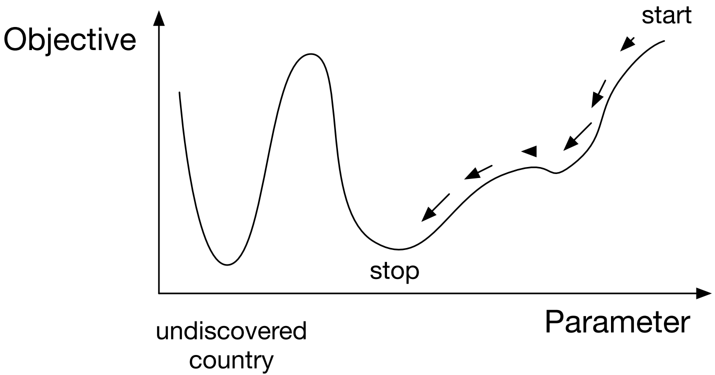
$$J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} || h_{W,b}(x^{(i)}) - y^{(i)} ||^2 \right] + \frac{\lambda}{2} \sum_{l}^{n_l - 1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{l} \right)^2 \tag{6}$$

- Our goal is to minimize $J(W,b)$ as a function of W and b
- Initialize W and b to small random value near zero
- Adjust parameters to optimize J

Gradient Descent

Goal

Optimize J with respect to variables W and b



Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{7}$$

Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{7}$$

- The gradient is a function of a node's error $\delta_i^{(l)}$

Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{7}$$

- The gradient is a function of a node's error $\delta_i^{(l)}$
- For output nodes, the error is obvious:

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} ||y - h_{w,b}(x)||^2 = -\left(y_i - a_i^{(n_l)}\right) \cdot f'\left(z_i^{(n_l)}\right) \frac{2}{2} \tag{8}$$

Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{7}$$

- The gradient is a function of a node's error $\delta_i^{(l)}$
- For output nodes, the error is obvious:

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} ||y - h_{w,b}(x)||^2 = -\left(y_i - a_i^{(n_l)}\right) \cdot f'\left(z_i^{(n_l)}\right) \frac{2}{2} \tag{8}$$

- Other nodes must "backpropagate" downstream error based on connection strength

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{t+1}} W_{ji}^{(l+1)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \tag{9}$$

Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{7}$$

- The gradient is a function of a node's error $\delta_i^{(l)}$
- For output nodes, the error is obvious:

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} ||y - h_{w,b}(x)||^2 = -\left(y_i - a_i^{(n_l)}\right) \cdot f'\left(z_i^{(n_l)}\right) \frac{2}{2} \tag{8}$$

- Other nodes must "backpropagate" downstream error based on connection strength

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{t+1}} W_{ji}^{(l+1)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \tag{9}$$

Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{7}$$

- The gradient is a function of a node's error $\delta_i^{(l)}$
- For output nodes, the error is obvious:

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} ||y - h_{w,b}(x)||^2 = -\left(y_i - a_i^{(n_l)}\right) \cdot f'\left(z_i^{(n_l)}\right) \frac{2}{2} \tag{8}$$

- Other nodes must "backpropagate" downstream error based on connection strength

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{t+1}} W_{ji}^{(l+1)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \tag{9}$$

(chain rule)

Partial Derivatives

- For weights, the partial derivatives are

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)} \tag{10}$$

- For the bias terms, the partial derivatives are

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)} \tag{11}$$

- But this is just for a single example ...

Full Gradient Descent Algorithm

1. Initialize $U^{(l)}$ and $V^{(l)}$ as zero
2. For each example $i = 1 \dots m$
   2.1 Use backpropagation to compute $\nabla_W J$ and $\nabla_b J$
   2.2 Update weight shifts $U^{(l)} = U^{(l)} + \nabla_{W^{(l)}} J(W, b; x, y)$
   2.3 Update bias shifts $V^{(l)} = V^{(l)} + \nabla_{b^{(l)}} J(W, b; x, y)$
3. Update the parameters

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} U^{(l)} \right) \right] \tag{12}$$

$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} V^{(l)} \right] \tag{13}$$

4. Repeat until weights stop changing

But it is not perfect

- Compare against baselines: randomized features, nearest-neighbors, linear models
- Optimization is hard (alchemy)
- Models are often not interpretable
- Requires specialized hardware and tons of data to scale