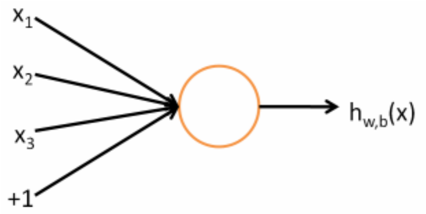
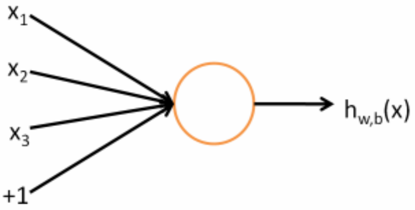
Multilayer Networks

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

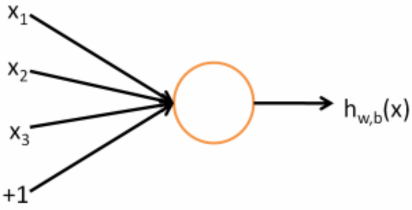


Input

Vector $x_1 \dots x_d$

inputs encoded as real numbers

Logistic Regression by Another Name: Map inputs to output



Input

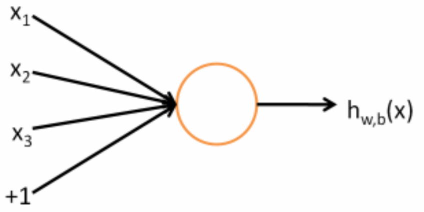
Vector $x_1 \dots x_d$

Output

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

multiply inputs by

Logistic Regression by Another Name: Map inputs to output



Input

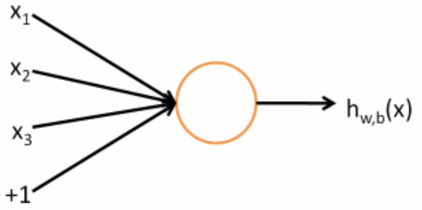
Vector $x_1 \dots x_d$

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

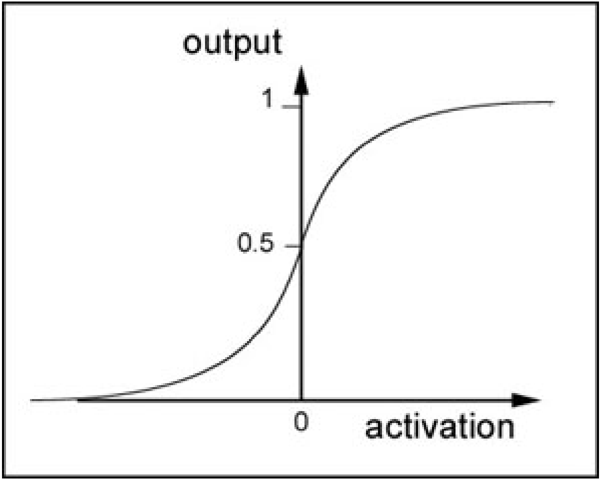
$$f\left(\sum_i W_i x_i + b\right)$$

Activation

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

pass through nonlinear sigmoid

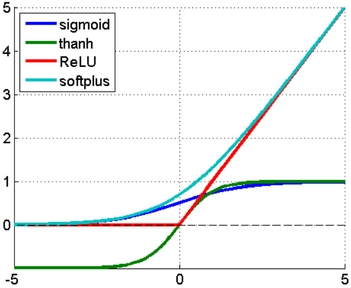
Why is it called activation?



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

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

- tanh

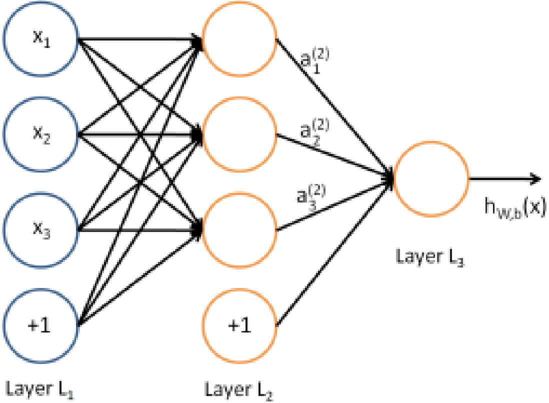
$$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1 \tag{2}$$

- ReLU

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

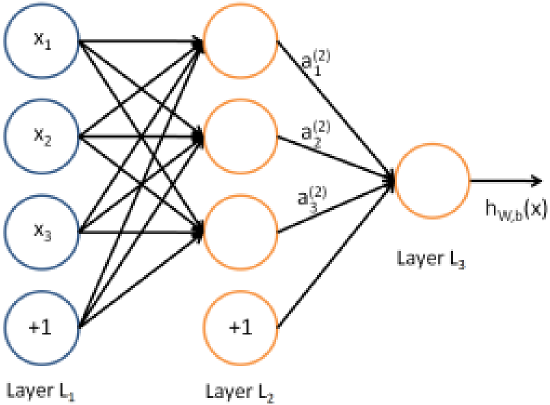
- SoftPlus: $f(x) = \ln(1 + e^{x})$

Learn the features and the function



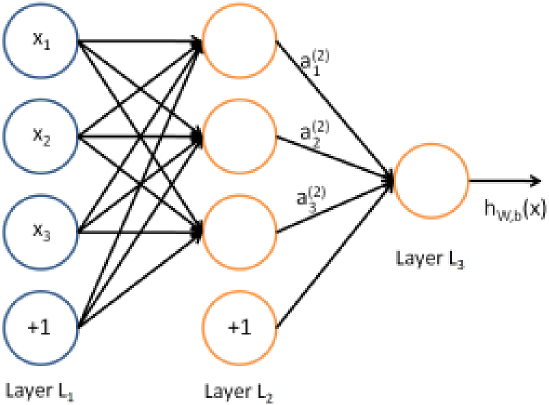
$$a_1^{(2)} = f\left(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)}\right)$$

Learn the features and the function



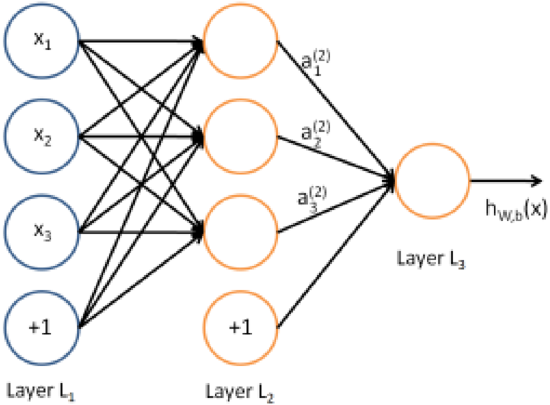
$$a_2^{(2)} = f\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}\right)$$

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

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

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:

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

Objective Function

Putting it all together:

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

- 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:

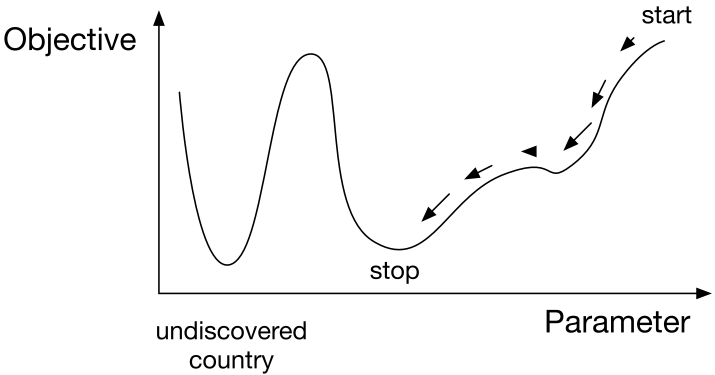
$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}||h_{W,b}(x^{(i)}) - y^{(i)}||^2\right] + \frac{\lambda}{2}\sum_{l}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{l}\right)^2 \quad (6)$$

- 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

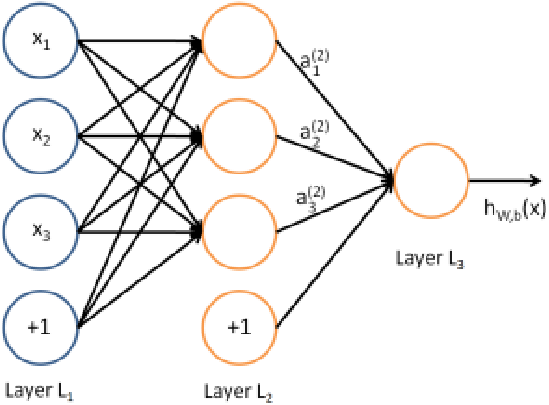
- Compare against baselines: randomized features, nearest-neighbors, linear models
- Optimization is hard (alchemy)
- Models are often not interpretable
- Requires specialized hardware and tons of data to scale

Multilayer Networks

Natural Language Processing: Jordan Boyd-Graber
University of Maryland

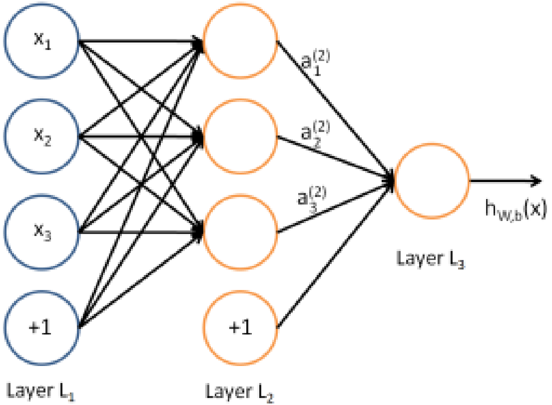
MATHEMATICAL DESCRIPTION

Learn the features and the function



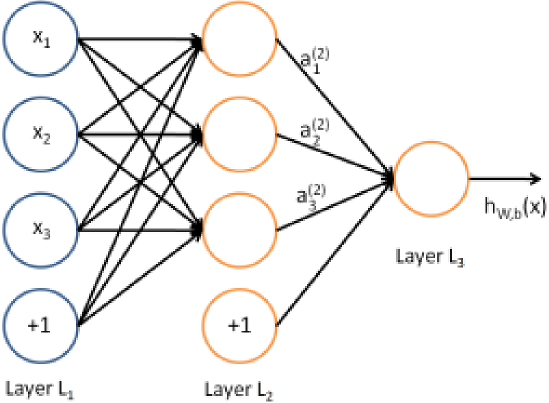
$$a_1^{(2)} = f\left(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)}\right)$$

Learn the features and the function



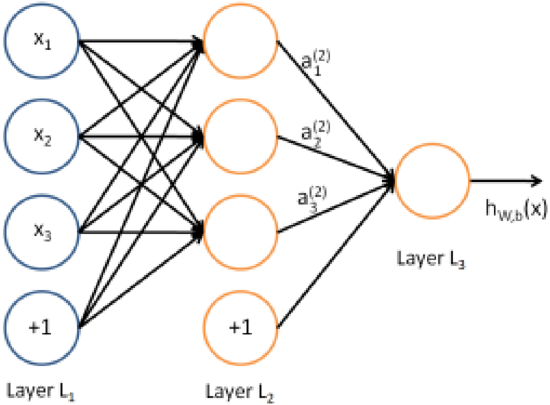
$$a_2^{(2)} = f\left(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)}\right)$$

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 $h_{W,b}(x)$.

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

Objective Function

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

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

- 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 $h_{W,b}(x)$.

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

- 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

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

Objective Function

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

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

- 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

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

Objective Function

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

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

- 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

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

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 $h_{W,b}(x)$.

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

- 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

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

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 $h_{W,b}(x)$.

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

- 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

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

Sum over all destinations

Objective Function

Putting it all together:

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

Objective Function

Putting it all together:

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

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

Objective Function

Putting it all together:

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

- 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:

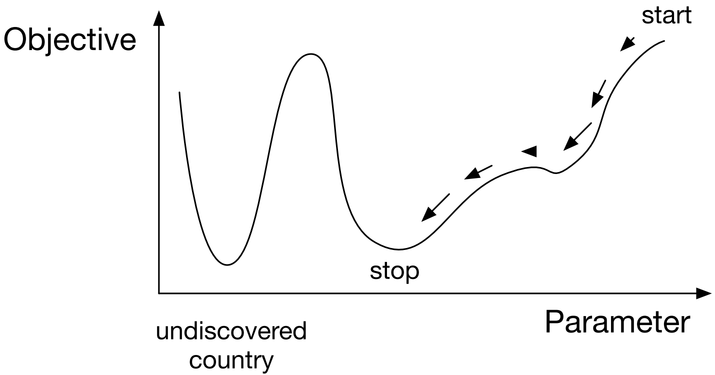
$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}||h_{W,b}(x^{(i)}) - y^{(i)}||^2\right] + \frac{\lambda}{2}\sum_{l}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{l}\right)^2 \tag{3}$$

- 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

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{4}$$

Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{4}$$

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

Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{4}$$

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

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} ||y - h_{w,b}(x)||^2 = -\left(y_i - a_i^{(n_l)}\right) \cdot f'\left(z_i^{(n_l)}\right) \frac{1}{2} \tag{5}$$

Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{4}$$

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

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} ||y - h_{w,b}(x)||^2 = -\left(y_i - a_i^{(n_l)}\right) \cdot f'\left(z_i^{(n_l)}\right) \frac{1}{2} \tag{5}$$

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

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{t+1}} W_{ji}^{(l+1)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \tag{6}$$

Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{4}$$

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

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} ||y - h_{w,b}(x)||^2 = -\left(y_i - a_i^{(n_l)}\right) \cdot f'\left(z_i^{(n_l)}\right) \frac{1}{2} \tag{5}$$

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

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{t+1}} W_{ji}^{(l+1)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \tag{6}$$

Backpropigation

- For convenience, write the input to sigmoid

$$z_i^{(l)} = \sum_{j=1}^{n} W_{ij}^{(l-1)} x_j + b_i^{(l-1)} \tag{4}$$

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

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} ||y - h_{w,b}(x)||^2 = -\left(y_i - a_i^{(n_l)}\right) \cdot f'\left(z_i^{(n_l)}\right) \frac{1}{2} \tag{5}$$

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

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{t+1}} W_{ji}^{(l+1)} \delta_j^{(l+1)} \right) f'(z_i^{(l)}) \tag{6}$$

(chain rule)

Partial Derivatives

- For weights, the partial derivatives are

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b;x,y) = a_j^{(l)} \delta_i^{(l+1)} \tag{7}$$

- For the bias terms, the partial derivatives are

$$\frac{\partial}{\partial b_i^{(l)}} J(W,b;x,y) = \delta_i^{(l+1)} \tag{8}$$

- 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

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} U^{(l)} \right) \right] \tag{9}$$

$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} V^{(l)} \right] \tag{10}$$

4. Repeat until weights stop changing

Frameworks

Natural Language Processing: Jordan Boyd-Graber
University of Maryland
INTRODUCTION

Slides adapted from Chris Dyer, Yoav Goldberg, Graham Neubig

Neural Nets and Language

Language

Discrete, structured (graphs, trees)

Neural-Nets

Continuous: poor native support for structure

Big challenge: writing code that translates between the {discrete-structured, continuous} regimes

Why not do it yourself?

- Hard to compare with exting models
- Obscures difference between model and optimization
- Debugging has to be custom-built
- Hard to tweak model

Outline

- Computation graphs (general)
- Neural Nets in PyTorch
- Full example

Computation Graphs

Expression

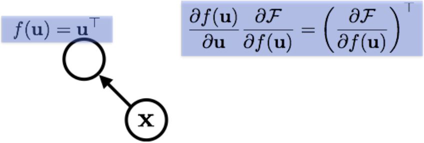
$\vec{x}$

graph:

Computation Graphs

Expression

$\vec{x}^\top$



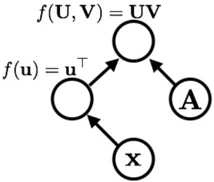
- Edge: function argument / data dependency
- A node with an incoming edge is a function $F \equiv f(u)$ edge's tail node
- A node computes its value and the value of its derivative w.r.t each argument (edge) times a derivative $\frac{\partial f}{\partial u}$

Computation Graphs

Expression

$\vec{x}^\top A$

graph:



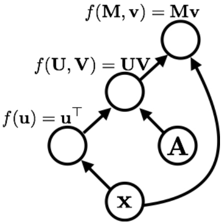
Functions can be nullary, unary, binary, ... n-ary. Often they are unary or binary.

Computation Graphs

Expression

$\vec{x}^{\top} A x$

graph:



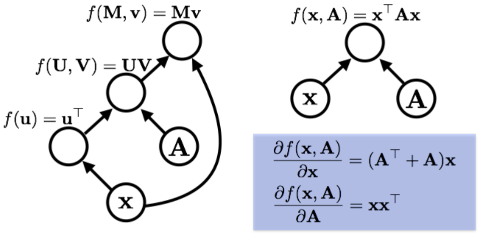
Computation graphs are (usually) directed and acyclic

Computation Graphs

Expression

$\vec{x}^\top A x$

graph:

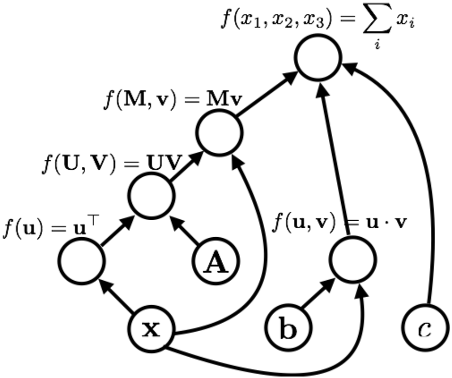


Computation Graphs

Expression

$\vec{x}^\top A x + b \cdot \vec{x} + c$

graph:

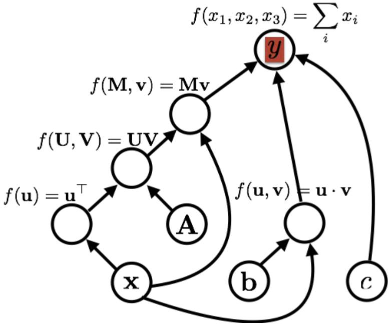


Computation Graphs

Expression

$y = \vec{x}^\top Ax + b \cdot \vec{x} + c$

graph:

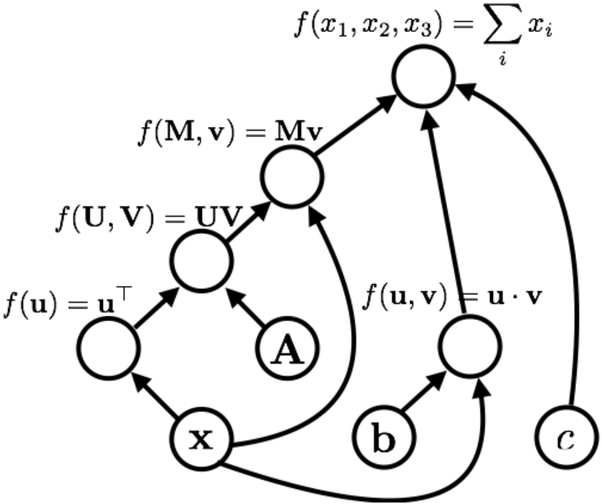


Variable names label nodes

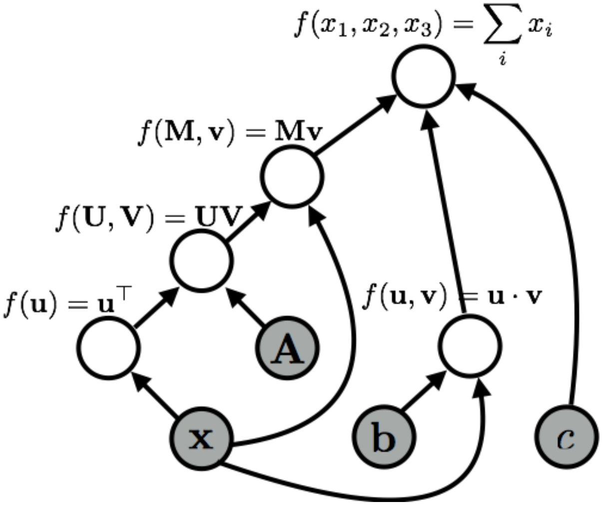
Algorithms

- Graph construction
- Forward propagation
  - Loop over nodes in topological order
  - Compute the value of the node given its inputs
  - Given my inputs, make a prediction (i.e. “error” vs. “target output”)
- Backward propagation
  - Loop over the nodes in reverse topological order, starting with goal node
  - Compute derivatives of final goal node value wrt each edge’s tail node
  - How does the output change with small change to inputs?

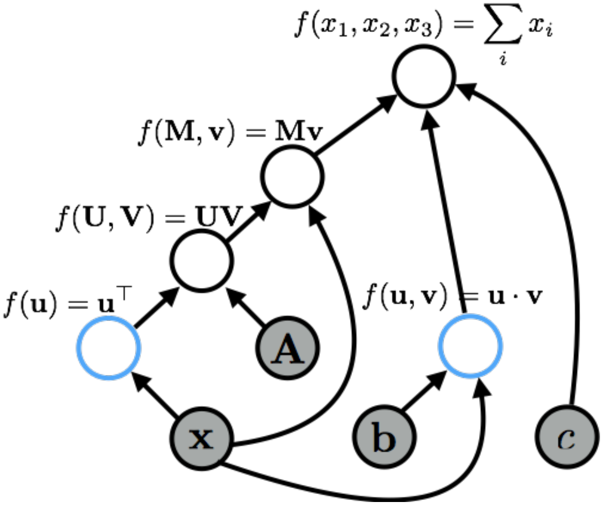
Forward Propagation



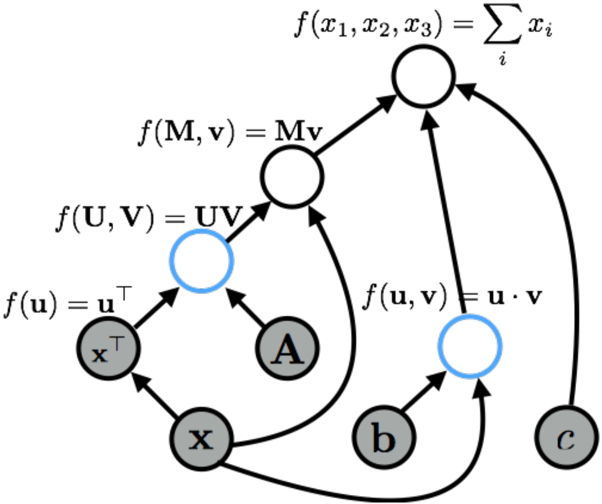
Forward Propagation



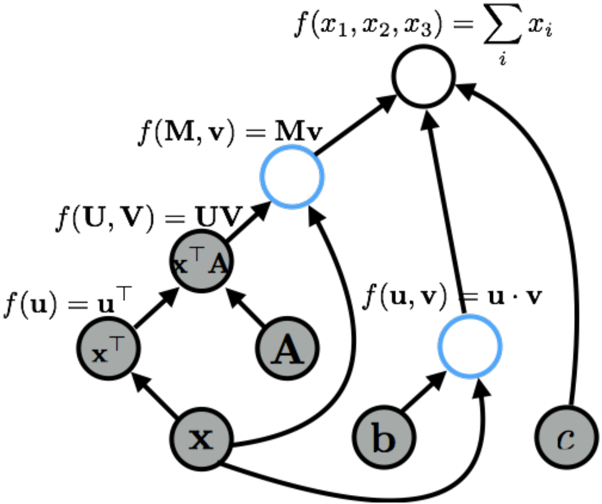
Forward Propagation



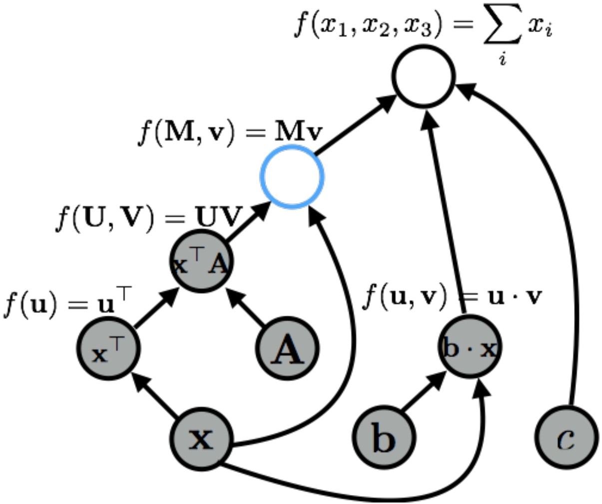
Forward Propagation



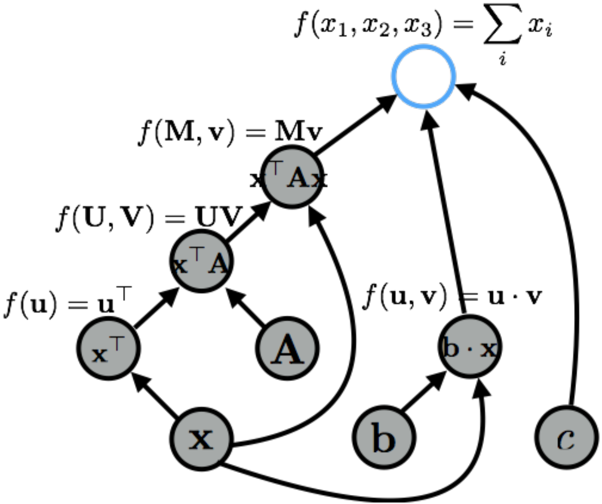
Forward Propagation



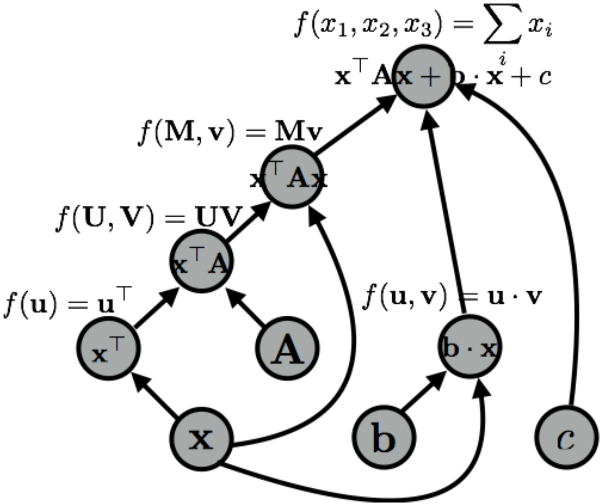
Forward Propagation



Forward Propagation



Forward Propagation



Constructing Graphs

Static declaration

- Define architecture, run data through
- PROS: Optimization, hardware support
- CONS: Structured data ugly, graph language

Theano, Tensorflow

Dynamic declaration

- Graph implicit with data
- PROS: Native language, interleave construction/evaluation
- CONS: Slower, computation can be wasted

Chainer, Dynet, PyTorch

Constructing Graphs

Static declaration

- Define architecture, run data through
- PROS: Optimization, hardware support
- CONS: Structured data ugly, graph language

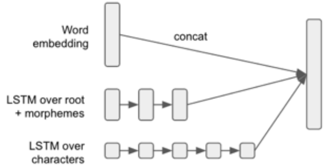
Theano, Tensorflow

Dynamic declaration

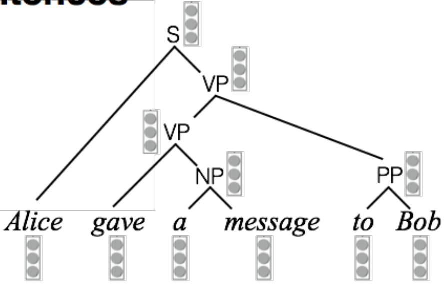
- Graph implicit with data
- PROS: Native language, interleave construction/evaluation
- CONS: Slower, computation can be wasted

Chainer, Dynet, PyTorch

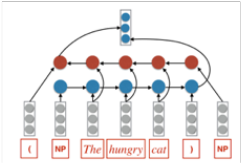
Words



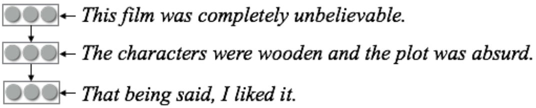
Sentences



Phrases



Documents



Language is Hierarchical

Dynamic Hierarchy in Language

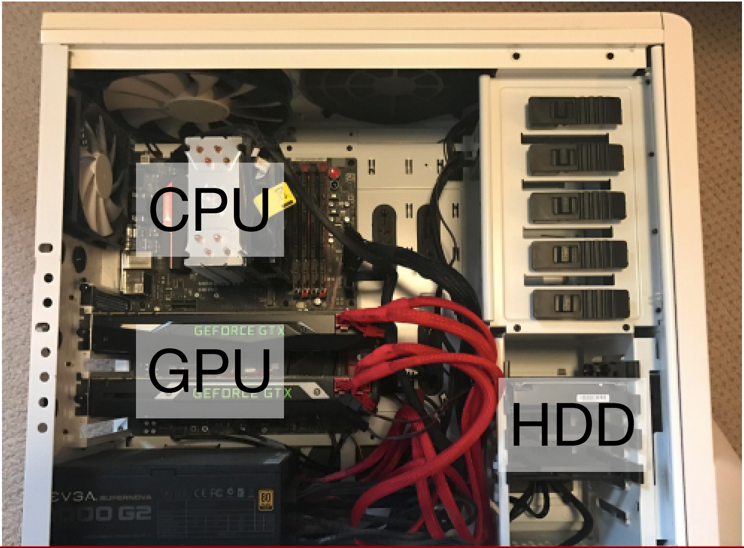
- Language is hierarchical
  - Graph should reflect this reality
  - Traditional flow-control best for processing
- Combinatorial algorithms (e.g., dynamic programming)
- Exploit independencies to compute over a large space of operations tractably

PyTorch

- Torch: Facebook's deep learning framework
- Nice, but written in Lua (C backend)
- Optimized to run computations on GPU
- Mature, industry-supported framework

Why GPU?

Why GPU?



Frameworks

Natural Language Processing: Jordan Boyd-Graber
University of Maryland
BACKPROP IN PYTORCH

Simple Model

```python
import torch
import torch.nn as nn

class LogisticRegression(nn.Module):
    def __init__(self, input_size, num_classes):
        super(LogisticRegression, self).__init__()
        self.linear = nn.Linear(input_size, num_classes)

    def forward(self, x):
        out = self.linear(x)
        return out
```

Simple Model

```
>>> model = LogisticRegression(5, 2)
>>> model.parameters
<bound method Module.parameters of LogisticRegression(
  (linear): Linear(in_features=5, out_features=2, bias=True
)>
>>> model.linear.weight
Parameter containing:
tensor([[ 0.0650,  0.0221,  0.1673, -0.1365, -0.1233],
        [-0.1289,  0.2455,  0.3255,  0.0409, -0.1908]], req
>>> model.linear.bias
Parameter containing:
tensor([-0.2208,  0.2562], requires_grad=True)
```

Where did these numbers come from?

```
class Bilinear(Module):
    r"""Applies a bilinear transformation to the incoming c
    :math:`y = x_1 A x_2 + b`
    """

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
```

Where did these numbers come from?

```python
class Bilinear(Module):
    r"""Applies a bilinear transformation to the incoming d
    :math:`y = x_1 A x_2 + b`
    """

    def reset_parameters(self):
        stdv = 1. / math.sqrt(self.weight.size(1))
        self.weight.data.uniform_(-stdv, stdv)
        if self.bias is not None:
            self.bias.data.uniform_(-stdv, stdv)
```

Beauty and peril of working with something like PyTorch!

Computation Graph and Expressions

- Create basic expressions.
- Combine them using operations.
- Expressions represent symbolic computations.
- Actual computation:

```
.value()
.npvalue()                    #numpy value
.scalar_value()
.cuda()                        # move to GPU
.forward()                    # compute expression
```

Running Computation Forward

```
>>> x = torch.Tensor(1, 5)
>>> x
tensor([[ 0.0000, -0.0000,  0.0000, -0.0000,  0.0000]])
>>> x = x*0 + 1
>>> x
tensor([[1., 1., 1., 1., 1.]])
>>> model.forward(x)
tensor([[-0.2263,  0.5485]], grad_fn=<ThAddmmBackward>)
```

Modules allow computation graph

- Each module must implement forward function
- If forward function just uses built-in modules, autograd works
- If not, you'll need to implement backward function (i.e., backprop)

Modules allow computation graph

- Each module must implement forward function
- If forward function just uses built-in modules, autograd works
- If not, you'll need to implement backward function (i.e., backprop)
  - input: as many Tensors as outputs of module (gradient w.r.t. that output)
  - output: as many Tensors as inputs of module (gradient w.r.t. its corresponding input)
  - If inputs do not need gradient (static) you can return None

Trainers and Backprop

- Initialize a Optimizer with a given model's parameter
- Get output for an example / minibatch
- Compute loss and backpropagate
- Take step of Optimizer
- Repeat ...

Trainers and Backprop

```
optimizer = torch.optim.SGD(model.parameters(),
                            lr=learning_rate)

# Training the Model
for epoch in range(num_epochs):
    for i, (Variable(doc), Variable(label)) in \
            enumerate(train_loader):
        optimizer.zero_grad()
        prediction = model(doc)
        loss = nn.CrossEntropyLoss(prediction, label)
        loss.backward()
        optimizer.step()
```

Options for Optimizers

`Adadelta`
`Adagrad`
`Adam`
`LBFGS`
`SGD`

Closure (LBFGS), learning rate, etc.

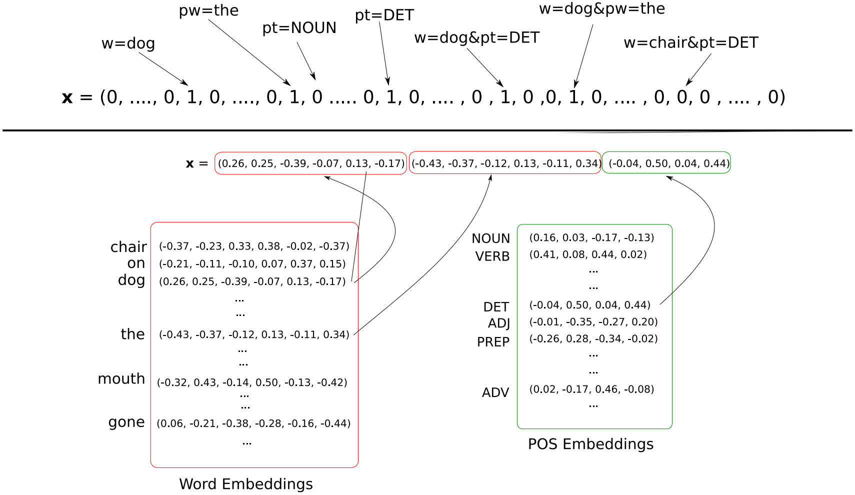
Key Points

- Create computation graph for each example.
- Graph is built by composing expressions.
- Functions that take expressions and return expressions define graph components.

Word Embeddings and Lookup Parameters

- In NLP, it is very common to use feature embeddings
- Each feature is represented as a d-dim vector
- These are then summed or concatenated to form an input vector
- The embeddings can be pre-trained
- But they are usually trained (fine-tunded) with the model

"feature embeddings"



```python
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim

torch.manual_seed(1)
word_to_ix = {"hello": 0, "world": 1}
embeds = nn.Embedding(2, 5)  # 2 words in vocab, 5 dim embe
lookup_tensor = torch.tensor([word_to_ix["hello"]],
                             dtype=torch.long)
hello_embed = embeds(lookup_tensor)
```

Frameworks

Natural Language Processing: Jordan Boyd-Graber

University of Maryland

EXAMPLE IMPLEMENTATION: DAN

Deep Unordered Composition Rivals Syntactic Methods for Text Classification

Mohit Iyyer,[1] **Varun Manjunatha,**[1] **Jordan Boyd-Graber,**[2] **Hal Daumé III**[1]
[1]University of Maryland, Department of Computer Science and UMIACS
[2]University of Colorado, Department of Computer Science
{miyyer,varunm,hal}@umiacs.umd.edu, Jordan.Boyd.Graber@colorado.edu

Implementing a non-trivial example . . .

Deep Averaging Network

$$
\begin{aligned}
& w_1, \ldots, w_N \\
& \qquad \downarrow \\
z_0 &= \text{CBOW}(w_1, \ldots, w_N) \\
z_1 &= g(W_1 z_0 + b_1) \\
z_2 &= g(W_2 z_1 + b_2) \\
\hat{y} &= \text{softmax}(z_2)
\end{aligned}
$$

- Works about as well as more complicated models
- Strong baseline
- Key idea: Continuous Bag of Words

$$\text{CBOW}(w_1, \ldots, w_N) = \sum_i E[w_i] \tag{1}$$

- Actual non-linearity doesn't matter, we'll use tanh
- Let's implement in PyTorch

Deep Averaging Network

$$w_1, \ldots, w_N$$

$$\downarrow$$

$$z_0 = \text{CBOW}(w_1, \ldots, w_N)$$

$$z_1 = g(z_1)$$

$$z_2 = g(z_2)$$

$$\hat{y} = \text{softmax}(z_3)$$

Initialization

```python
def __init__(self, n_classes, vocab_size, emb_dim=300,
             n_hidden_units=300):
    super(DanModel, self).__init__()
    self.n_classes = n_classes
    self.vocab_size = vocab_size
    self.emb_dim = emb_dim
    self.n_hidden_units = n_hidden_units
    self.embeddings = nn.Embedding(self.vocab_size,
                                   self.emb_dim)
    self.classifier = nn.Sequential(
           nn.Linear(self.n_hidden_units,
                     self.n_hidden_units),
           nn.ReLU(),
           nn.Linear(self.n_hidden_units,
                     self.n_classes))
    self._softmax = nn.Softmax()
```

Deep Averaging Network

$$w_1, \dots, w_N$$
$$\downarrow$$
$$z_0 = \text{CBOW}(w_1, \dots, w_N)$$
$$z_1 = g(z_1)$$
$$z_2 = g(z_2)$$
$$\hat{y} = \text{softmax}(z_3)$$

Forward

```
def forward(self, batch, probs=False):
    text = batch['text']['tokens']
    length = batch['length']
    text_embed = self._word_embeddings(text)
    # Take the mean embedding. Since padding results
    # in zeros its safe to sum and divide by length
    encoded = text_embed.sum(1)
    encoded /= lengths.view(text_embed.size(0), -1)

    # Compute the network score predictions
    logits = self.classifier(encoded)
    if probs:
        return self._softmax(logits)
    else:
        return logits
```

Deep Averaging Network

$$w_1, \ldots, w_N$$

$$\downarrow$$

$$z_0 = \text{CBOW}(w_1, \ldots, w_N)$$

$$z_1 = g(z_1)$$

$$z_2 = g(z_2)$$

$$\hat{y} = \text{softmax}(z_3)$$

Training

```
def _run_epoch(self, batch_iter, train=True):
    self._model.train()
    for batch in batch_iter:
        model.zero_grad()
        out = model(batches)
        batch_loss = criterion(out,
                               batch['label'])
        batch_loss.backward()
        self.optimizer.step()
```

Summary

- Computation Graph
- Expressions ($\approx$ nodes in the graph)
- Parameters, LookupParameters
- Model (a collection of parameters)
- Optimizers
- Create a graph for each example, compute loss, backdrop, update

Multilayer Networks

Natural Language Processing: Jordan Boyd-Graber
University of Maryland
HANDS-ON DEMO

Data and Model

Data

x_1	x_2	y
1.00	1.00	0.00
1.00	0.00	1.00
0.00	0.00	0.00
0.00	1.00	1.00

First Layer

$$w^{(1)} = \begin{bmatrix} 1.00 & 1.00 \\ 1.00 & 1.00 \end{bmatrix} \tag{1}$$

$$b^{(1)} = \begin{bmatrix} -1.00 & 0.00 \end{bmatrix} \tag{2}$$

Second Layer

$$w^{(2)} = \begin{bmatrix} -2.00 & 1.00 \end{bmatrix} \tag{3}$$

$$b^{(2)} = 0.00 \tag{4}$$

Using ReLU as non-linearity

Prediction for $x_0 = (1.00, 1.00)$

Prediction for $x_0 = (1.00, 1.00)$

- Hidden Computation

$$a^{(1)}_{0,0} = f(w^{(1)}_{0,0} \cdot 1.00 + w^{(1)}_{0,1} \cdot 1.00 + b_0) \tag{5}$$

$$= f(1.00 \cdot 1.00 + 1.00 \cdot 1.00 + -1.00) \tag{6}$$

Prediction for $x_0 = (1.00, 1.00)$

- Hidden Computation

$$a_{0,0}^{(1)} = f(w_{0,0}^{(1)} \cdot 1.00 + w_{0,1}^{(1)} \cdot 1.00 + b_0) \quad (5)$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 1.00 + -1.00) \quad (6)$$

$$a_{0,1}^{(1)} = f(w_{1,0}^{(1)} \cdot 1.00 + w_{1,1}^{(1)} \cdot 1.00 + b_1) \quad (7)$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 1.00 + 0.00) \quad (8)$$

Prediction for $x_0 = (1.00, 1.00)$

- Hidden Computation

$$a^{(1)}_{0,0} = f(w^{(1)}_{0,0} \cdot 1.00 + w^{(1)}_{0,1} \cdot 1.00 + b_0) \tag{5}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 1.00 + -1.00) \tag{6}$$

$$a^{(1)}_{0,1} = f(w^{(1)}_{1,0} \cdot 1.00 + w^{(1)}_{1,1} \cdot 1.00 + b_1) \tag{7}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 1.00 + 0.00) \tag{8}$$

- Hidden Layer: [1 2]
- Output Answer

$$a^{(3)}_{0,0} = f(w^{(2)}_{0,0} \cdot 1.00 + w^{(2)}_{0,1} \cdot 2.00 + b_0) \tag{9}$$
$$= f(-2.00 \cdot 1.00 + 1.00 \cdot 2.00 + 0.00) \tag{10}$$

- Prediction: 0.00, Error: 0.00

Prediction for $x_0 = (1.00, 1.00)$

- Hidden Computation

$$a^{(1)}_{0,0} = f(w^{(1)}_{0,0} \cdot 1.00 + w^{(1)}_{0,1} \cdot 1.00 + b_0) \tag{5}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 1.00 + -1.00) \tag{6}$$

$$a^{(1)}_{0,1} = f(w^{(1)}_{1,0} \cdot 1.00 + w^{(1)}_{1,1} \cdot 1.00 + b_1) \tag{7}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 1.00 + 0.00) \tag{8}$$

- Hidden Layer: [1 2]
- Output Answer

$$a^{(3)}_{0,0} = f(w^{(2)}_{0,0} \cdot 1.00 + w^{(2)}_{0,1} \cdot 2.00 + b_0) \tag{9}$$
$$= f(-2.00 \cdot 1.00 + 1.00 \cdot 2.00 + 0.00) \tag{10}$$

Prediction for $x_0 = (1.00, 1.00)$

- Hidden Computation

$$a^{(1)}_{0,0} = f(w^{(1)}_{0,0} \cdot 1.00 + w^{(1)}_{0,1} \cdot 1.00 + b_0) \tag{5}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 1.00 + -1.00) \tag{6}$$

$$a^{(1)}_{0,1} = f(w^{(1)}_{1,0} \cdot 1.00 + w^{(1)}_{1,1} \cdot 1.00 + b_1) \tag{7}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 1.00 + 0.00) \tag{8}$$

- Hidden Layer: [1 2]
- Output Answer

$$a^{(3)}_{0,0} = f(w^{(2)}_{0,0} \cdot 1.00 + w^{(2)}_{0,1} \cdot 2.00 + b_0) \tag{9}$$
$$= f(-2.00 \cdot 1.00 + 1.00 \cdot 2.00 + 0.00) \tag{10}$$

Prediction for $x_1 = (1.00, 0.00)$

Prediction for $x_1 = (1.00, 0.00)$

- Hidden Computation

$$a^{(1)}_{1,0} = f(w^{(1)}_{0,0} \cdot 1.00 + w^{(1)}_{0,1} \cdot 0.00 + b_0) \tag{11}$$

$$= f(1.00 \cdot 1.00 + 1.00 \cdot 0.00 + -1.00) \tag{12}$$

Prediction for $x_1 = (1.00, 0.00)$

- Hidden Computation

$$a^{(1)}_{1,0} = f(w^{(1)}_{0,0} \cdot 1.00 + w^{(1)}_{0,1} \cdot 0.00 + b_0) \quad (11)$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 0.00 + -1.00) \quad (12)$$

$$a^{(1)}_{1,1} = f(w^{(1)}_{1,0} \cdot 1.00 + w^{(1)}_{1,1} \cdot 0.00 + b_1) \quad (13)$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 0.00 + 0.00) \quad (14)$$

Prediction for $x_1 = (1.00, 0.00)$

- Hidden Computation

$$a^{(1)}_{1,0} = f(w^{(1)}_{0,0} \cdot 1.00 + w^{(1)}_{0,1} \cdot 0.00 + b_0) \tag{11}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 0.00 + -1.00) \tag{12}$$

$$a^{(1)}_{1,1} = f(w^{(1)}_{1,0} \cdot 1.00 + w^{(1)}_{1,1} \cdot 0.00 + b_1) \tag{13}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 0.00 + 0.00) \tag{14}$$

- Hidden Layer: [0 1]
- Output Answer

$$a^{(3)}_{1,0} = f(w^{(2)}_{0,0} \cdot 0.00 + w^{(2)}_{0,1} \cdot 1.00 + b_0) \tag{15}$$
$$= f(-2.00 \cdot 0.00 + 1.00 \cdot 1.00 + 0.00) \tag{16}$$

- Prediction: 1.00, Error: 0.00

Prediction for $x_1 = (1.00, 0.00)$

- Hidden Computation

$$a^{(1)}_{1,0} = f(w^{(1)}_{0,0} \cdot 1.00 + w^{(1)}_{0,1} \cdot 0.00 + b_0) \tag{11}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 0.00 + -1.00) \tag{12}$$

$$a^{(1)}_{1,1} = f(w^{(1)}_{1,0} \cdot 1.00 + w^{(1)}_{1,1} \cdot 0.00 + b_1) \tag{13}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 0.00 + 0.00) \tag{14}$$

- Hidden Layer: [0 1]
- Output Answer

$$a^{(3)}_{1,0} = f(w^{(2)}_{0,0} \cdot 0.00 + w^{(2)}_{0,1} \cdot 1.00 + b_0) \tag{15}$$
$$= f(-2.00 \cdot 0.00 + 1.00 \cdot 1.00 + 0.00) \tag{16}$$

Prediction for $x_1 = (1.00, 0.00)$

- Hidden Computation

$$a^{(1)}_{1,0} = f(w^{(1)}_{0,0} \cdot 1.00 + w^{(1)}_{0,1} \cdot 0.00 + b_0) \tag{11}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 0.00 + -1.00) \tag{12}$$

$$a^{(1)}_{1,1} = f(w^{(1)}_{1,0} \cdot 1.00 + w^{(1)}_{1,1} \cdot 0.00 + b_1) \tag{13}$$
$$= f(1.00 \cdot 1.00 + 1.00 \cdot 0.00 + 0.00) \tag{14}$$

- Hidden Layer: [0 1]
- Output Answer

$$a^{(3)}_{1,0} = f(w^{(2)}_{0,0} \cdot 0.00 + w^{(2)}_{0,1} \cdot 1.00 + b_0) \tag{15}$$
$$= f(-2.00 \cdot 0.00 + 1.00 \cdot 1.00 + 0.00) \tag{16}$$

Prediction for $x_2 = (0.00, 0.00)$

Prediction for $x_2 = (0.00, 0.00)$

- Hidden Computation

$$
\begin{aligned}
a_{2,0}^{(1)} &= f(w_{0,0}^{(1)} \cdot 0.00 + w_{0,1}^{(1)} \cdot 0.00 + b_0) && (17)\\
&= f(1.00 \cdot 0.00 + 1.00 \cdot 0.00 + -1.00) && (18)
\end{aligned}
$$

Prediction for $x_2 = (0.00, 0.00)$

- Hidden Computation

$$a_{2,0}^{(1)} = f(w_{0,0}^{(1)} \cdot 0.00 + w_{0,1}^{(1)} \cdot 0.00 + b_0) \tag{17}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 0.00 + -1.00) \tag{18}$$

$$a_{2,1}^{(1)} = f(w_{1,0}^{(1)} \cdot 0.00 + w_{1,1}^{(1)} \cdot 0.00 + b_1) \tag{19}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 0.00 + 0.00) \tag{20}$$

Prediction for $x_2 = (0.00, 0.00)$

- Hidden Computation

$$a^{(1)}_{2,0} = f(w^{(1)}_{0,0} \cdot 0.00 + w^{(1)}_{0,1} \cdot 0.00 + b_0) \tag{17}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 0.00 + -1.00) \tag{18}$$

$$a^{(1)}_{2,1} = f(w^{(1)}_{1,0} \cdot 0.00 + w^{(1)}_{1,1} \cdot 0.00 + b_1) \tag{19}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 0.00 + 0.00) \tag{20}$$

- Hidden Layer: [0. 0.]
- Output Answer

$$a^{(3)}_{2,0} = f(w^{(2)}_{0,0} \cdot 0.00 + w^{(2)}_{0,1} \cdot 0.00 + b_0) \tag{21}$$
$$= f(-2.00 \cdot 0.00 + 1.00 \cdot 0.00 + 0.00) \tag{22}$$

- Prediction: 0.00, Error: 0.00

Prediction for $x_2 = (0.00, 0.00)$

- Hidden Computation

$$a_{2,0}^{(1)} = f(w_{0,0}^{(1)} \cdot 0.00 + w_{0,1}^{(1)} \cdot 0.00 + b_0) \tag{17}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 0.00 + -1.00) \tag{18}$$

$$a_{2,1}^{(1)} = f(w_{1,0}^{(1)} \cdot 0.00 + w_{1,1}^{(1)} \cdot 0.00 + b_1) \tag{19}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 0.00 + 0.00) \tag{20}$$

- Hidden Layer: [0. 0.]
- Output Answer

$$a_{2,0}^{(3)} = f(w_{0,0}^{(2)} \cdot 0.00 + w_{0,1}^{(2)} \cdot 0.00 + b_0) \tag{21}$$
$$= f(-2.00 \cdot 0.00 + 1.00 \cdot 0.00 + 0.00) \tag{22}$$

Prediction for $x_2 = (0.00, 0.00)$

- Hidden Computation

$$a_{2,0}^{(1)} = f(w_{0,0}^{(1)} \cdot 0.00 + w_{0,1}^{(1)} \cdot 0.00 + b_0) \quad (17)$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 0.00 + -1.00) \quad (18)$$

$$a_{2,1}^{(1)} = f(w_{1,0}^{(1)} \cdot 0.00 + w_{1,1}^{(1)} \cdot 0.00 + b_1) \quad (19)$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 0.00 + 0.00) \quad (20)$$

- Hidden Layer: [0. 0.]
- Output Answer

$$a_{2,0}^{(3)} = f(w_{0,0}^{(2)} \cdot 0.00 + w_{0,1}^{(2)} \cdot 0.00 + b_0) \quad (21)$$
$$= f(-2.00 \cdot 0.00 + 1.00 \cdot 0.00 + 0.00) \quad (22)$$

Prediction for $x_3 = (0.00, 1.00)$

Prediction for $x_3 = (0.00, 1.00)$

- Hidden Computation

$$a_{3,0}^{(1)} = f(w_{0,0}^{(1)} \cdot 0.00 + w_{0,1}^{(1)} \cdot 1.00 + b_0) \tag{23}$$

$$= f(1.00 \cdot 0.00 + 1.00 \cdot 1.00 + -1.00) \tag{24}$$

Prediction for $x_3 = (0.00, 1.00)$

- Hidden Computation

$$a_{3,0}^{(1)} = f(w_{0,0}^{(1)} \cdot 0.00 + w_{0,1}^{(1)} \cdot 1.00 + b_0) \tag{23}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 1.00 + -1.00) \tag{24}$$

$$a_{3,1}^{(1)} = f(w_{1,0}^{(1)} \cdot 0.00 + w_{1,1}^{(1)} \cdot 1.00 + b_1) \tag{25}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 1.00 + 0.00) \tag{26}$$

Prediction for $x_3 = (0.00, 1.00)$

- Hidden Computation

$$a^{(1)}_{3,0} = f(w^{(1)}_{0,0} \cdot 0.00 + w^{(1)}_{0,1} \cdot 1.00 + b_0) \tag{23}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 1.00 + -1.00) \tag{24}$$

$$a^{(1)}_{3,1} = f(w^{(1)}_{1,0} \cdot 0.00 + w^{(1)}_{1,1} \cdot 1.00 + b_1) \tag{25}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 1.00 + 0.00) \tag{26}$$

- Hidden Layer: [0 1]
- Output Answer

$$a^{(3)}_{3,0} = f(w^{(2)}_{0,0} \cdot 0.00 + w^{(2)}_{0,1} \cdot 1.00 + b_0) \tag{27}$$
$$= f(-2.00 \cdot 0.00 + 1.00 \cdot 1.00 + 0.00) \tag{28}$$

- Prediction: 1.00, Error: 0.00

Prediction for $x_3 = (0.00, 1.00)$

- Hidden Computation

$$a_{3,0}^{(1)} = f(w_{0,0}^{(1)} \cdot 0.00 + w_{0,1}^{(1)} \cdot 1.00 + b_0) \tag{23}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 1.00 + -1.00) \tag{24}$$

$$a_{3,1}^{(1)} = f(w_{1,0}^{(1)} \cdot 0.00 + w_{1,1}^{(1)} \cdot 1.00 + b_1) \tag{25}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 1.00 + 0.00) \tag{26}$$

- Hidden Layer: [0 1]
- Output Answer

$$a_{3,0}^{(3)} = f(w_{0,0}^{(2)} \cdot 0.00 + w_{0,1}^{(2)} \cdot 1.00 + b_0) \tag{27}$$
$$= f(-2.00 \cdot 0.00 + 1.00 \cdot 1.00 + 0.00) \tag{28}$$

Prediction for $x_3 = (0.00, 1.00)$

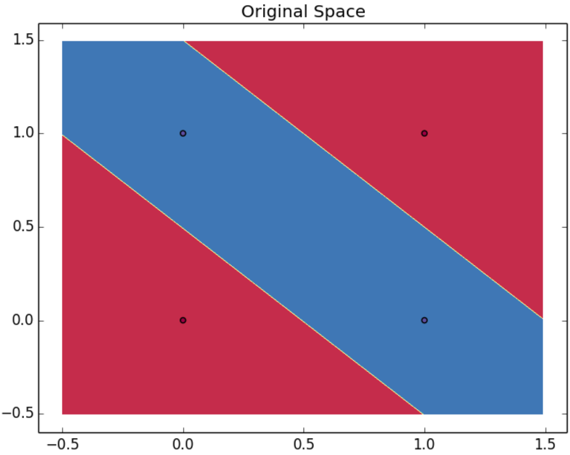
- Hidden Computation

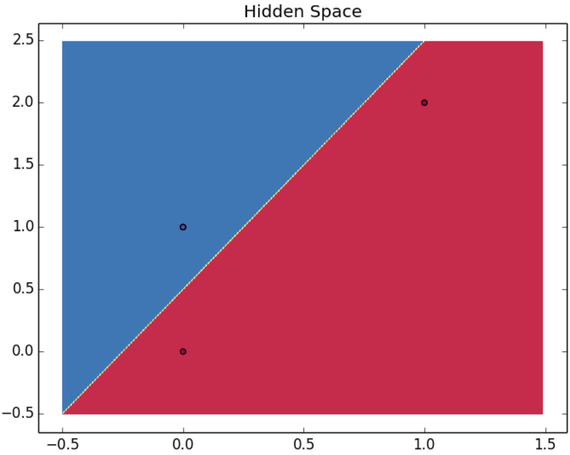
$$a^{(1)}_{3,0} = f(w^{(1)}_{0,0} \cdot 0.00 + w^{(1)}_{0,1} \cdot 1.00 + b_0) \tag{23}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 1.00 + -1.00) \tag{24}$$

$$a^{(1)}_{3,1} = f(w^{(1)}_{1,0} \cdot 0.00 + w^{(1)}_{1,1} \cdot 1.00 + b_1) \tag{25}$$
$$= f(1.00 \cdot 0.00 + 1.00 \cdot 1.00 + 0.00) \tag{26}$$

- Hidden Layer: [0 1]
- Output Answer

$$a^{(3)}_{3,0} = f(w^{(2)}_{0,0} \cdot 0.00 + w^{(2)}_{0,1} \cdot 1.00 + b_0) \tag{27}$$
$$= f(-2.00 \cdot 0.00 + 1.00 \cdot 1.00 + 0.00) \tag{28}$$


Original Space
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5


Hidden Space
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−0.5
0.0
0.5
1.0
1.5

Next Time

- Representing Words
- Updating representations
- Comparing with contextual information