CMPUT 651 (Fall 2019)

Deep Neural Networks

Lili Mou
lmou@ualberta.ca
lili-mou.github.io

Last Lecture: Classification

- Logistic regression $$y^{(j)} = \frac{1}{1 + e^{-(\theta_0 + \boldsymbol{\theta}^\top \boldsymbol{x}^{(j)})}}$$

- Loss $$J = \sum_{j=1}^{n} \left[-t^{(j)} \log y^{(i)} - (1 - t^{(j)}) \log(1 - y^{(j)}) \right]$$

- Optimization: gradient descent

$$\theta_i = \theta_i - \alpha \frac{\partial J}{\partial \theta_i} \qquad \frac{\partial J}{\partial \theta_i} = (y - t) x_i$$

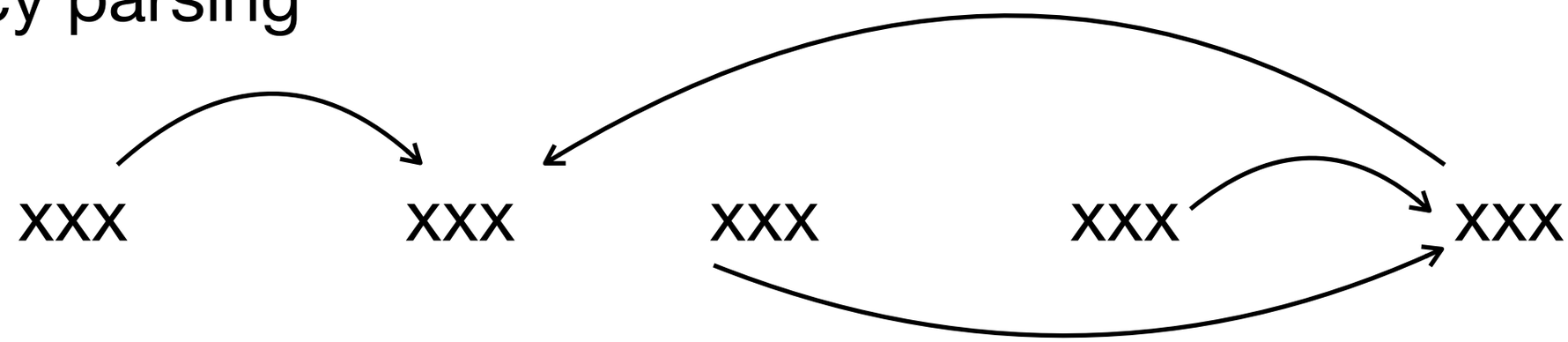
- Softmax $$p(Y = i) \stackrel{\Delta}{=} y_i \propto_i \exp\{\boldsymbol{w}_i^\top \boldsymbol{x}\}$$

$$y_i = \frac{\exp\{w_i^\top \boldsymbol{x}\}}{\sum_i^{'} \exp\{w_{i'}^\top \boldsymbol{x}\}}$$

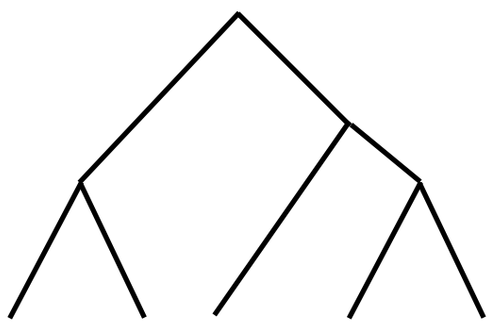
UNIVERSITY OF ALBERTA

Last Lecture: NLP Tasks

- Part-of-speech tagging `DT NN VB`
- Chunking `_ _ _ _ / _ _ / _ _ _`
- Dependency parsing



- Constituency parsing



Drawbacks

- Classification is linear
  - Deep learning
- Not modeling the relationship of labels within one data sample
  - Structured prediction

Decision Boundary

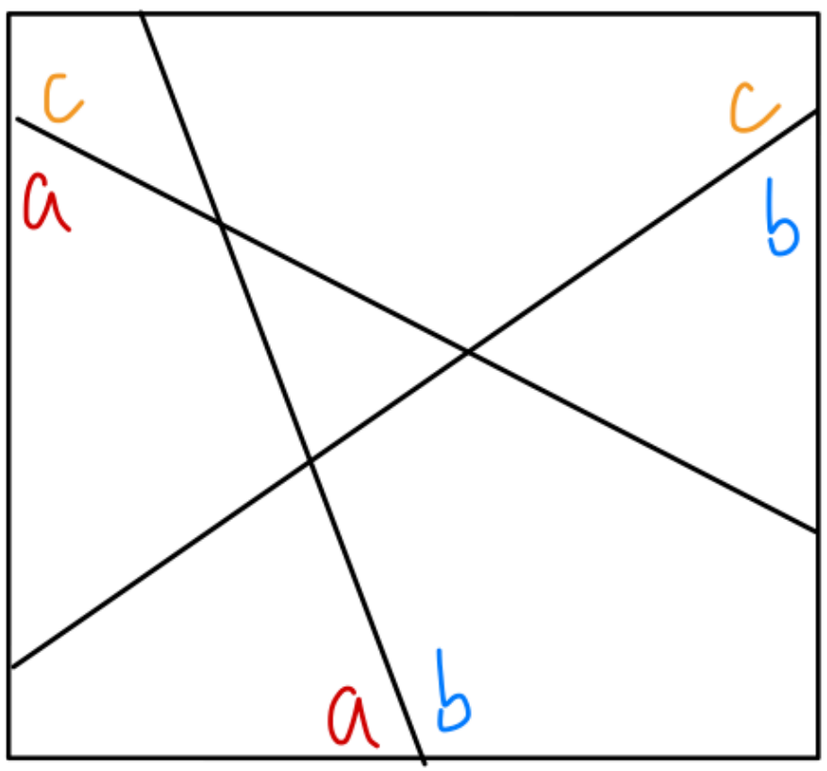
- Input feature $\boldsymbol{x} \in \mathbb{R}^n$
- Output candidate labels: $0,\ 2,\ \ldots,\ m-1$
- Q: **In which area of $\mathbb{R}^n$ does the model predict class i?**
- Model outputs i <=> $i = \text{argmax}_{i'}\, p(Y = i' | \boldsymbol{x})$

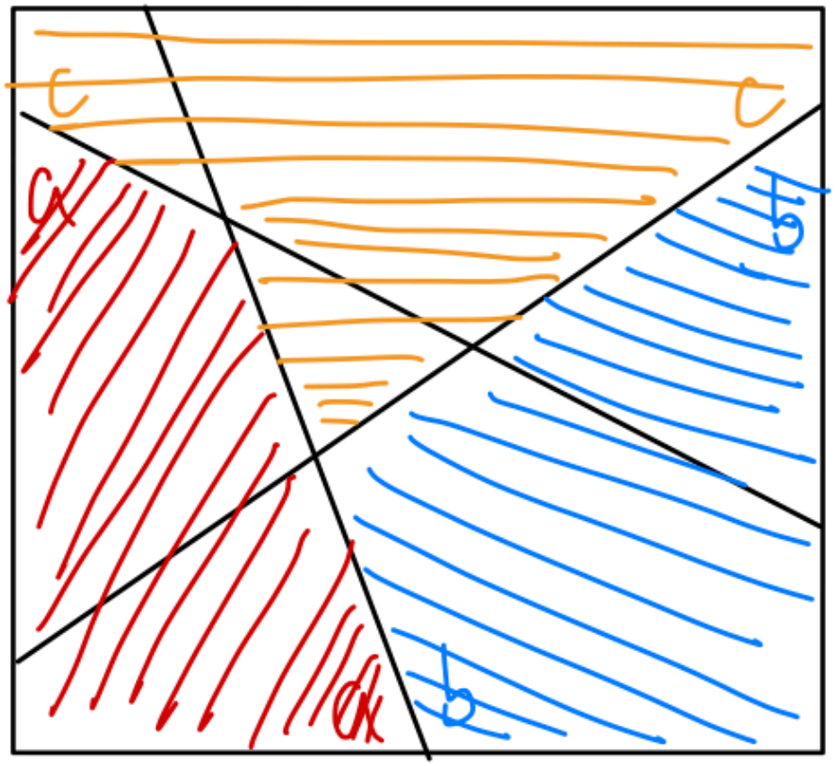
- Considering Class i and Class j $(i \neq j)$:
  - Class i is preferred than j: $\{\boldsymbol{x} \in \mathbb{R}^n : p(Y = i | \boldsymbol{x}) > p(Y = j | \boldsymbol{x})\}$

$$\frac{\exp\{\boldsymbol{w}_i^\top \boldsymbol{x} + b_i\}}{\sum_k \exp\{\boldsymbol{w}_k^\top \boldsymbol{x} + b_k\}} > \frac{\exp\{\boldsymbol{w}_j^\top \boldsymbol{x} + b_j\}}{\sum_k \exp\{\boldsymbol{w}_k^\top \boldsymbol{x} + b_k\}}$$

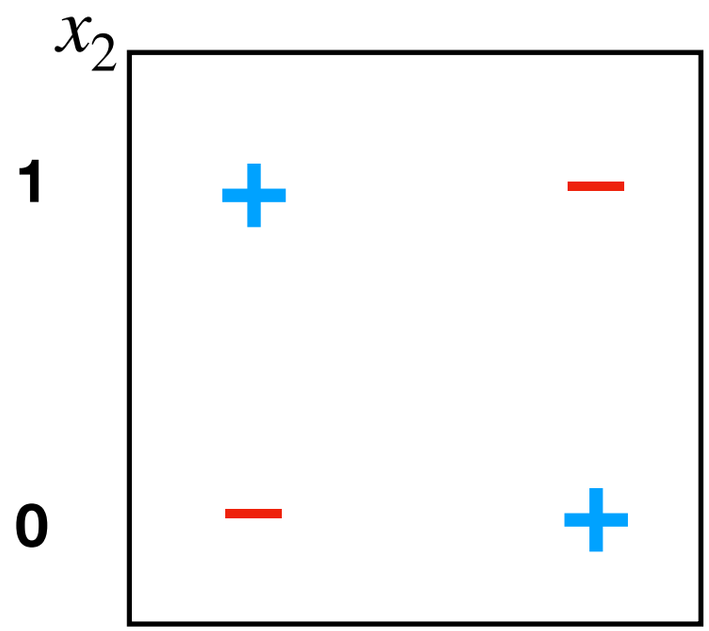
$$\iff \qquad (\boldsymbol{w}_i - \boldsymbol{w}_j)^\top \boldsymbol{x} + (b_i - b_j) > 0$$

Decision Boundary





XOR Problem



- Two classes +/1, -/0
- Logistic regression $y = 1 \iff \frac{1}{1 + e^{-(w_0 + w_1 x_1 + w_2 x_2)}} \geq 0.5$

$$\iff \quad w_0 + w_1 x_1 + w_2 x_2 \geq 0$$

- Hypothesis class $\mathscr{H} = \{\text{all half-planes}\}$

Non-Linear Classification

- Non-linearly mapping to some other space by **human engineering**
  - Input feature: $\boldsymbol{x} \in \mathbb{R}^n$
  - Construct additional features:

    $x_i^2,\ x_i x_j, \sin(x_i)$, etc.
  - Apply linear models in the extended features space $\widetilde{\boldsymbol{x}}$
  - Nonlinear in the original feature space $\boldsymbol{x}$

Non-Linear Classification

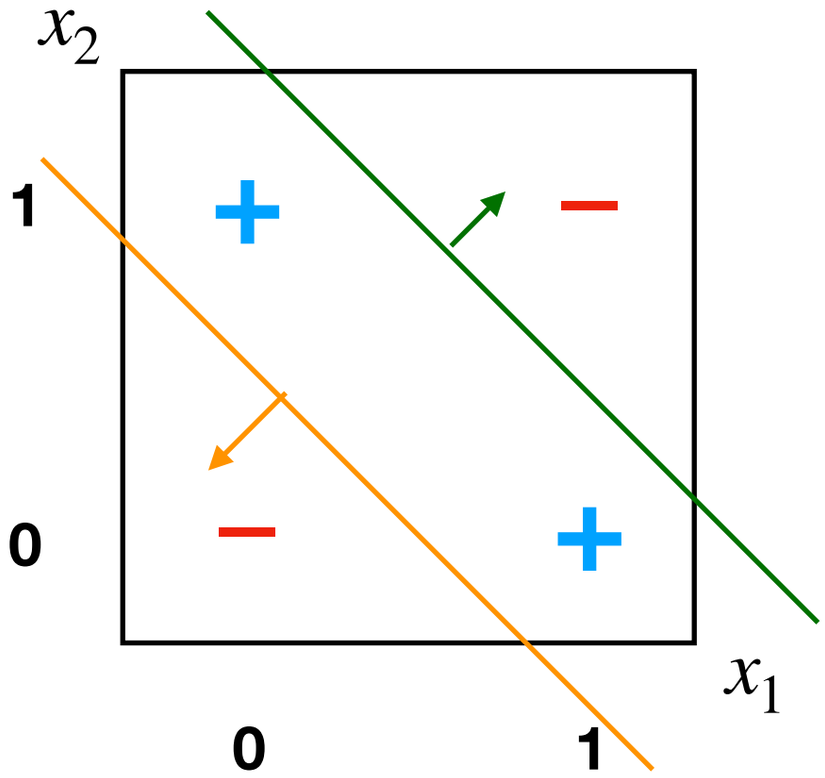
- Non-linearly mapping to some other space by **human engineering**
- Non-linearly mapping to some other space by **kernels**
  - Define the "inner-product" of all pairs of data samples in a way you believe
  - Mercer's Theorem:
    - $K(\cdot, \cdot)$ PSD $\Longleftrightarrow K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle$ for some $\phi(\cdot)$
  - E.g., $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^\top \boldsymbol{x}_j \implies$ Original feature space
  - E.g., $K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp\left\{ \dfrac{-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2}{2\sigma^2} \right\}$

    $\implies$ Mapped to "infinite dimensional space"
- Separability is good. But rely too heavily on the kernel.

Non-Linear Classification

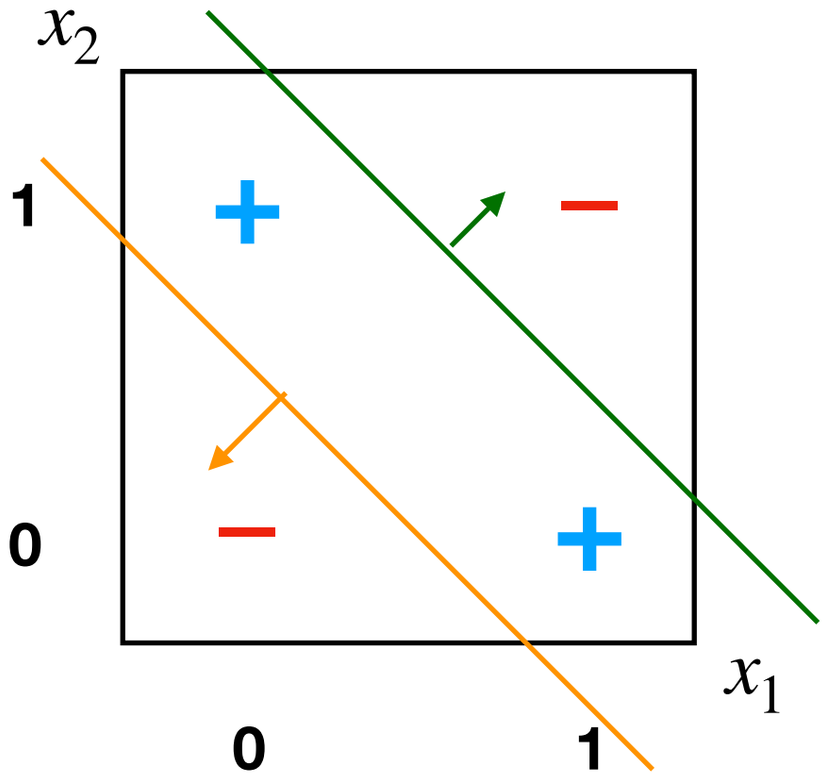
- Non-linearly mapping to some other space by **human engineering**
- Non-linearly mapping to some other space by **kernels**
- Non-linearly mapping to some other space by **learnable composite functions**
  - Neural networks, or deep learning

XOR Problem



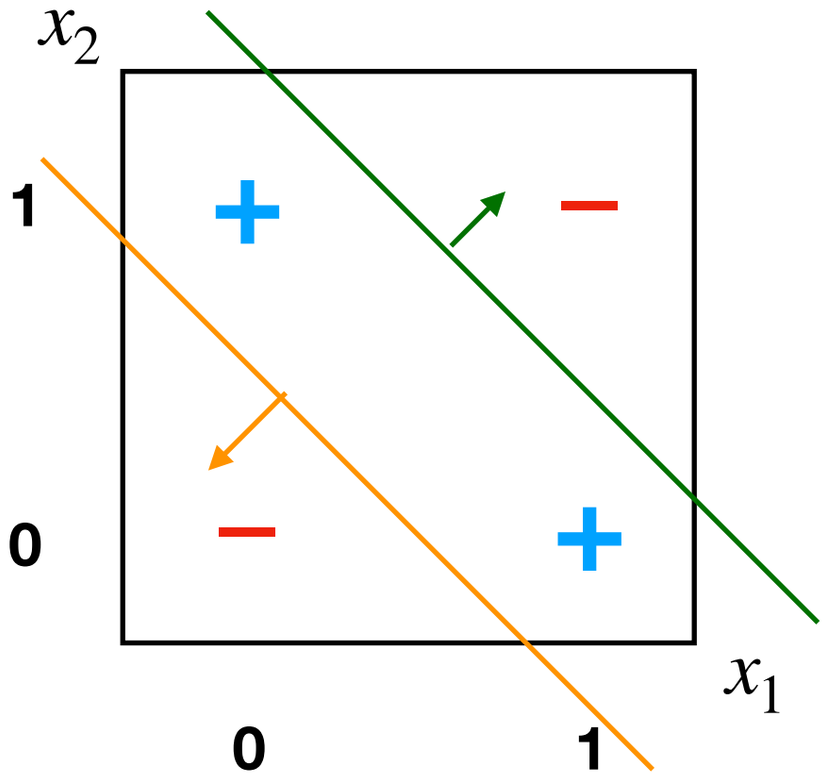
- What if we allow stacking logistic regression classifiers?
- Can we “program” in the “language” of LR stacks?

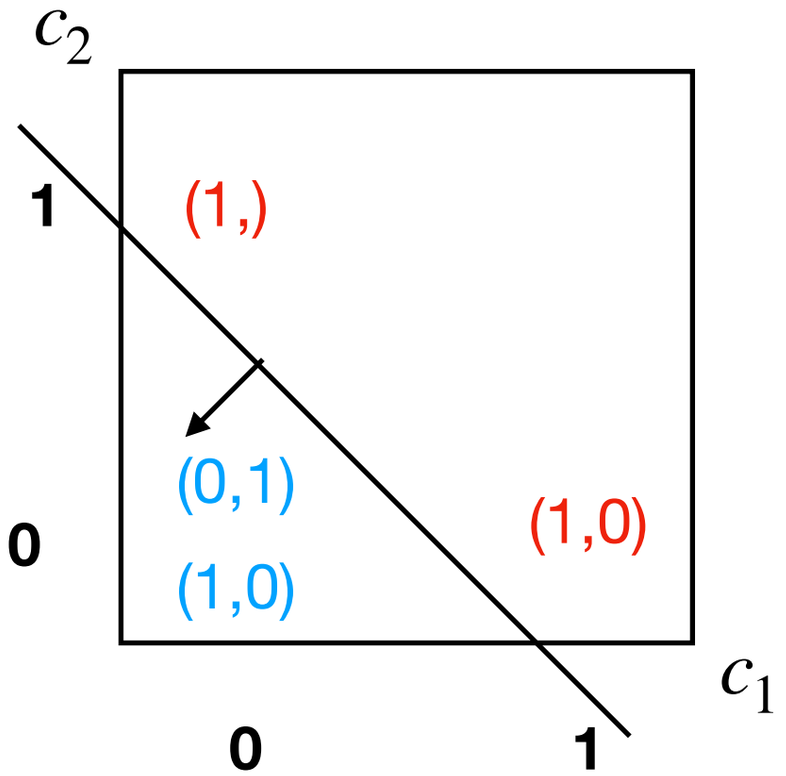
XOR Problem



Point:	(0,0)	(0,1)	(1,0)	(1,1)
Classifier 1:	1	0	0	0
Classifier 2:	0	0	0	1

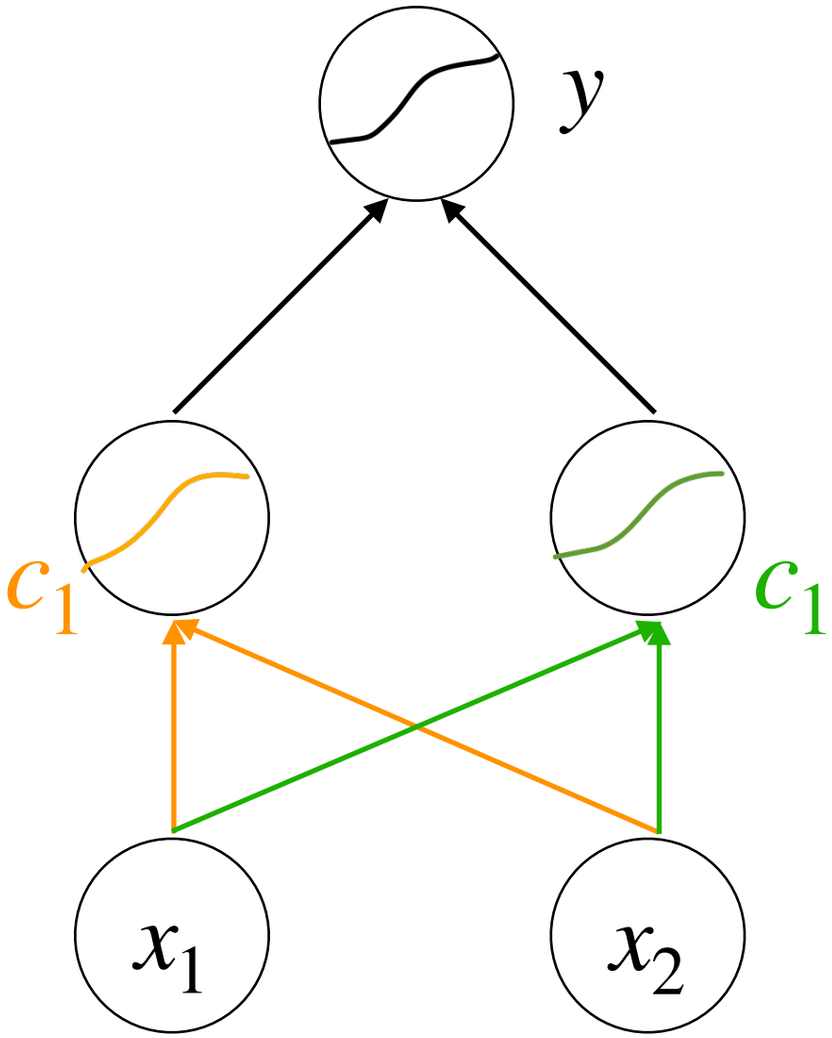
XOR Problem



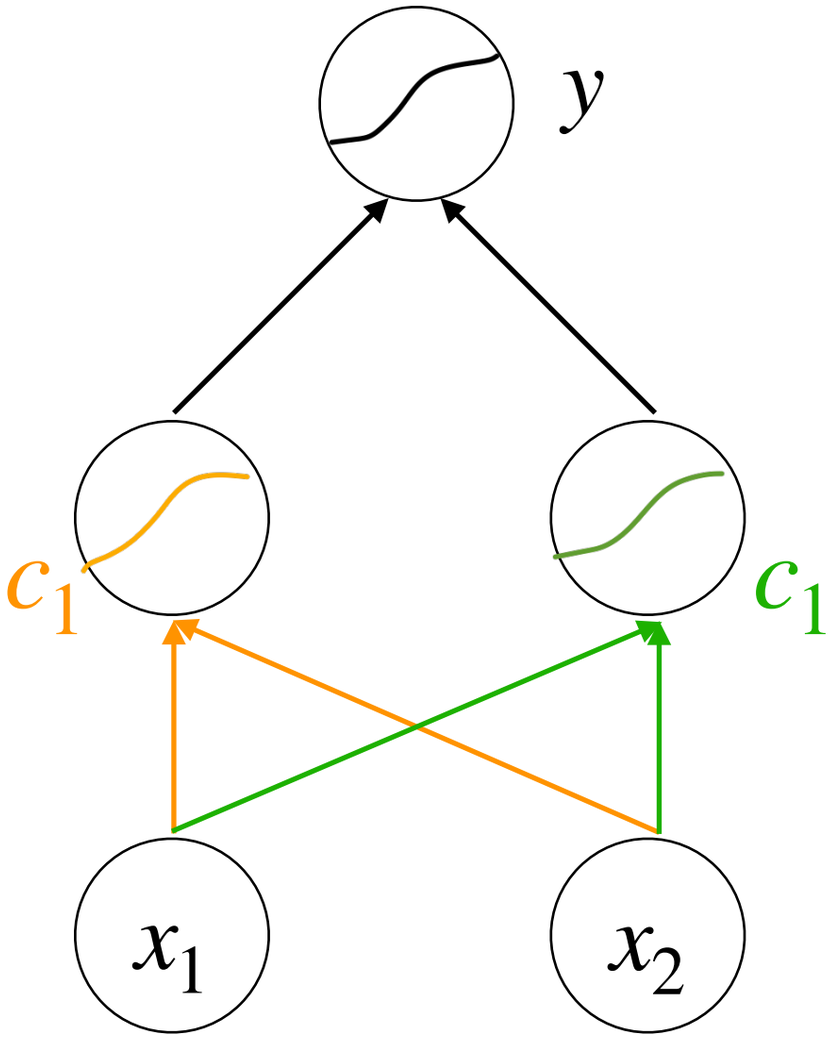


Point:	(0,0)	(0,1)	(1,0)	(1,1)
Classifier 1:	1	0	0	0
Classifier 2:	0	0	0	1

LR Stack We Programmed



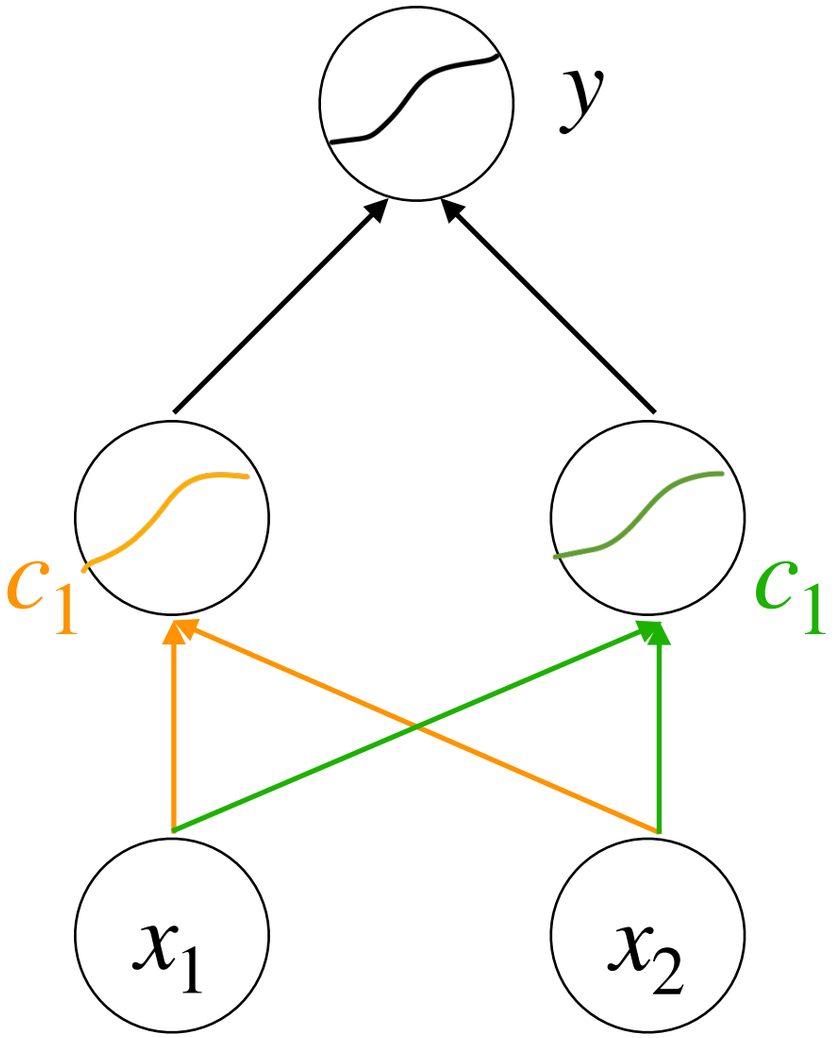
LR Stack We Programmed



Ungraded homework:

Assign the actual weights

LR Stack We Programmed



Ungraded homework:

Assign the actual weights

Student: I refuse to do the homework because

- Programming is tedious
- Only feasible for simple problems

Can We Learn the Weights?

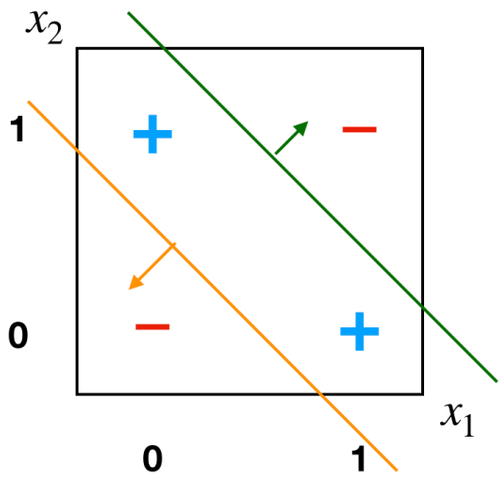
- Yes, we can. Still by gradient descent.

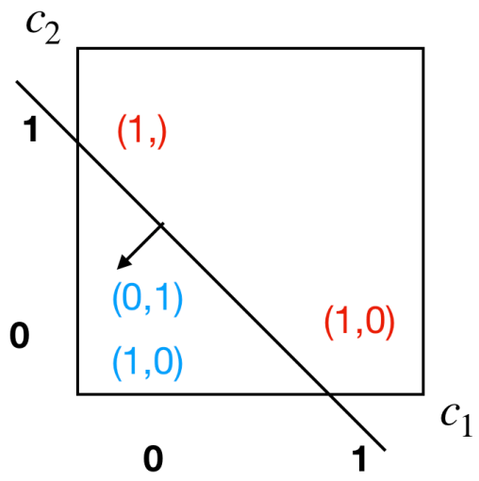
Can We Compute the Gradient?

- Yes, y is a differentiable function of weights

$$y = \frac{1}{1 + e^{-(w_0 + w_1 c_1 + w_2 c_2)}}$$

$$= \frac{1}{1 + e^{-\left(w_0 + w_1 \left\{ \frac{1}{1 + e^{-(w_{1,0} + w_{1,1} x_1 + w_{1,2} x_2)}} \right\} + w_2 \left\{ \frac{1}{1 + e^{-(w_{2,0} + w_{2,1} x_1 + w_{2,2} x_2)}} \right\}\right)}}$$



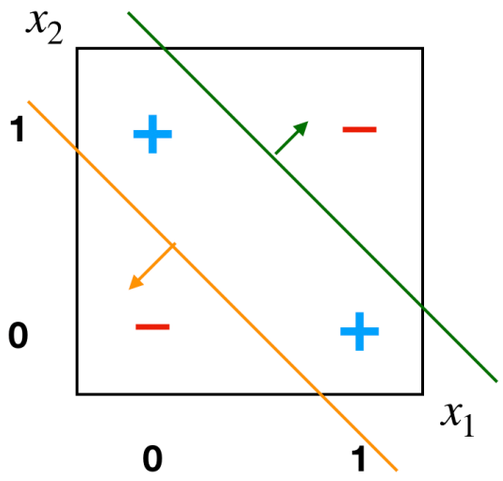


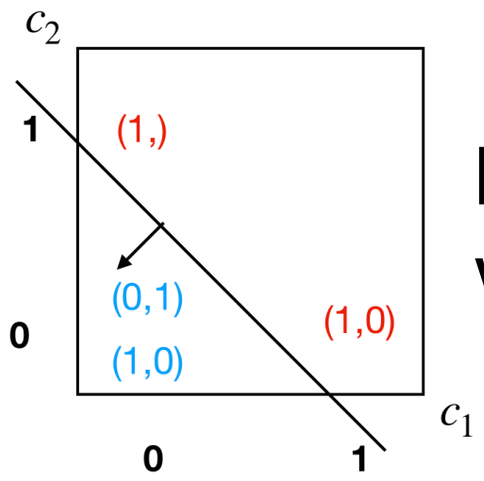
Can We Compute the Gradient?

- Yes, y is a differentiable function of weights

$$y = \frac{1}{1 + e^{-(w_0 + w_1 c_1 + w_2 c_2)}}$$

$$= \frac{1}{1 + e^{-\left(w_0 + w_1 \left\{ \frac{1}{1 + e^{-(w_{1,0} + w_{1,1} x_1 + w_{1,2} x_2)}} \right\} + w_2 \left\{ \frac{1}{1 + e^{-(w_{2,0} + w_{2,1} x_1 + w_{2,2} x_2)}} \right\}\right)}}$$

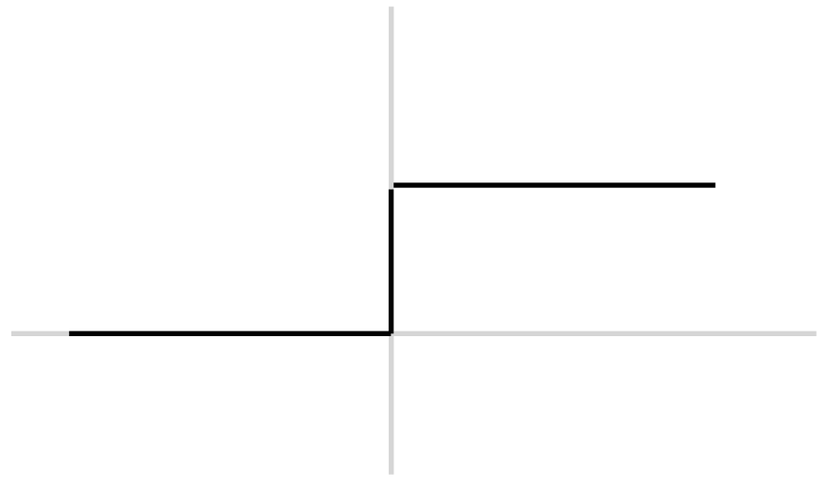




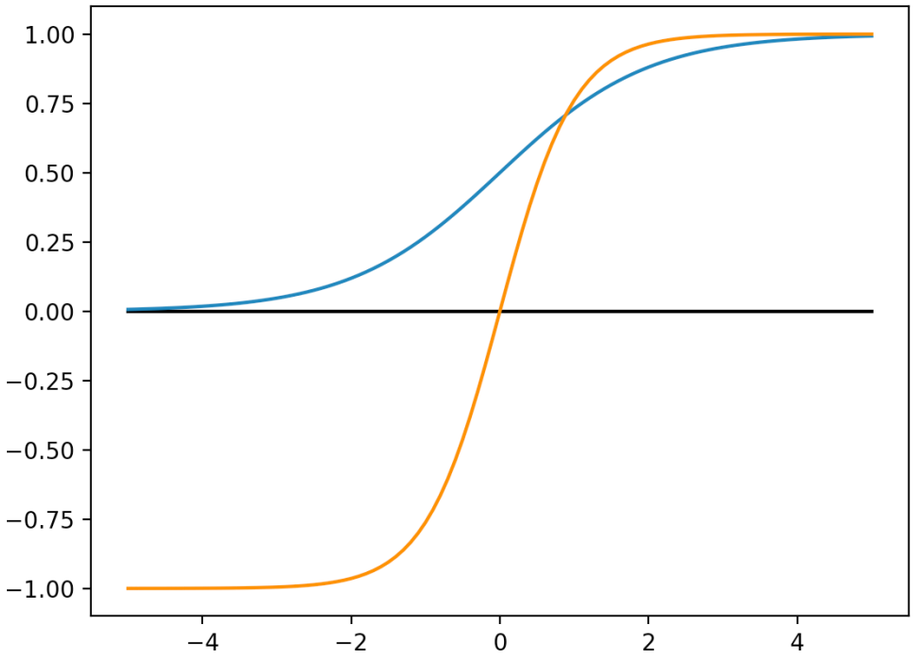
Problem: We need a systematic way of defining a deep architecture

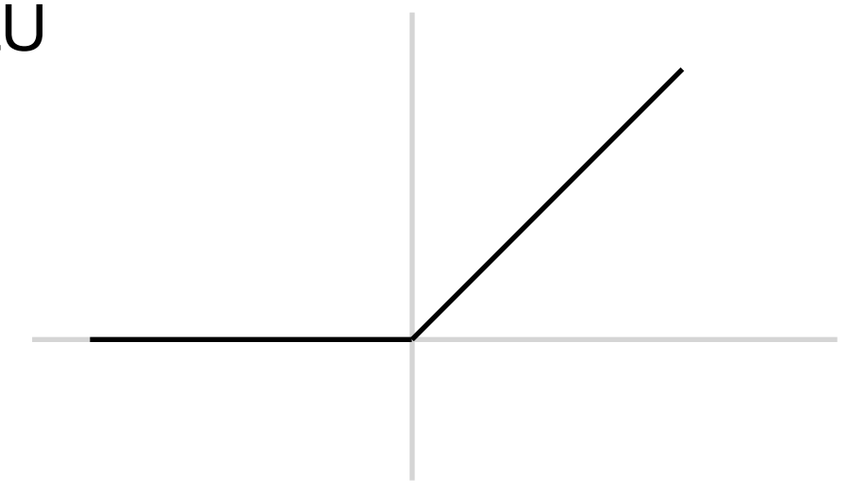
Artificial Neural Network

- Perceptron [Rosenblatt, 1958]
  - $y = f(\boldsymbol{w}^\top \boldsymbol{x} + b)$
  - f : binary thresholding function



- Perceptron-like neuron/unit/node
  - $y = f(\boldsymbol{w}^\top \boldsymbol{x} + b)$
  - f : activation function (usually point-wise)
  - sigmoid
  - tanh
  - ReLU





Artificial Neural Network

- Neurons in our brain

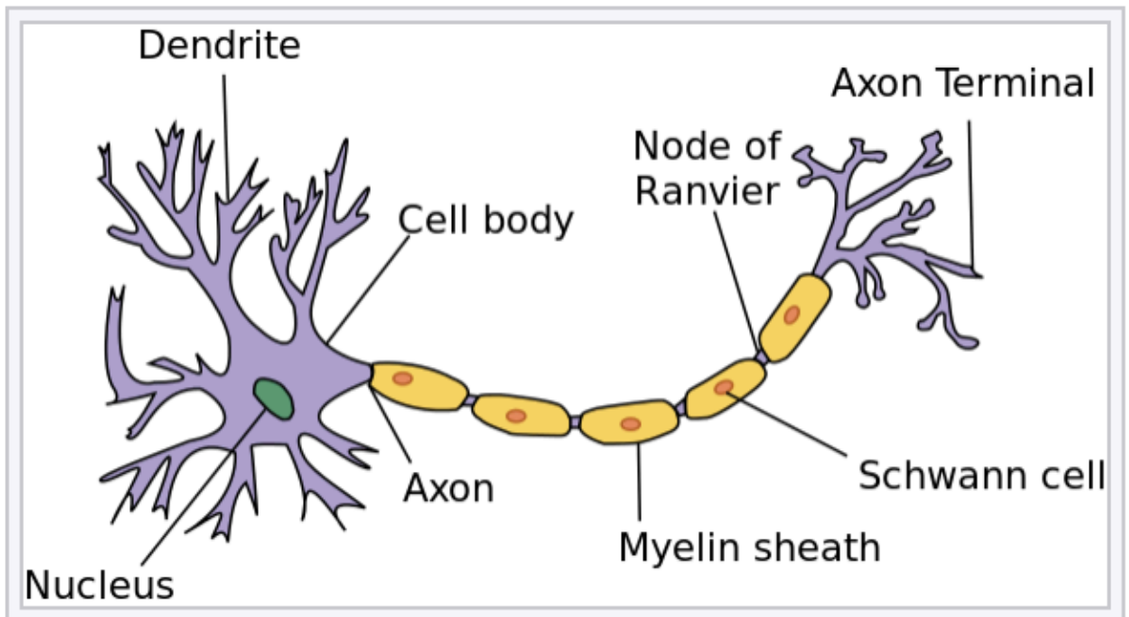
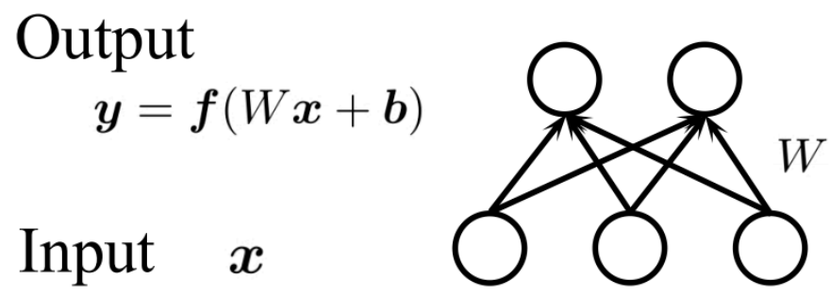


Diagram of neuron

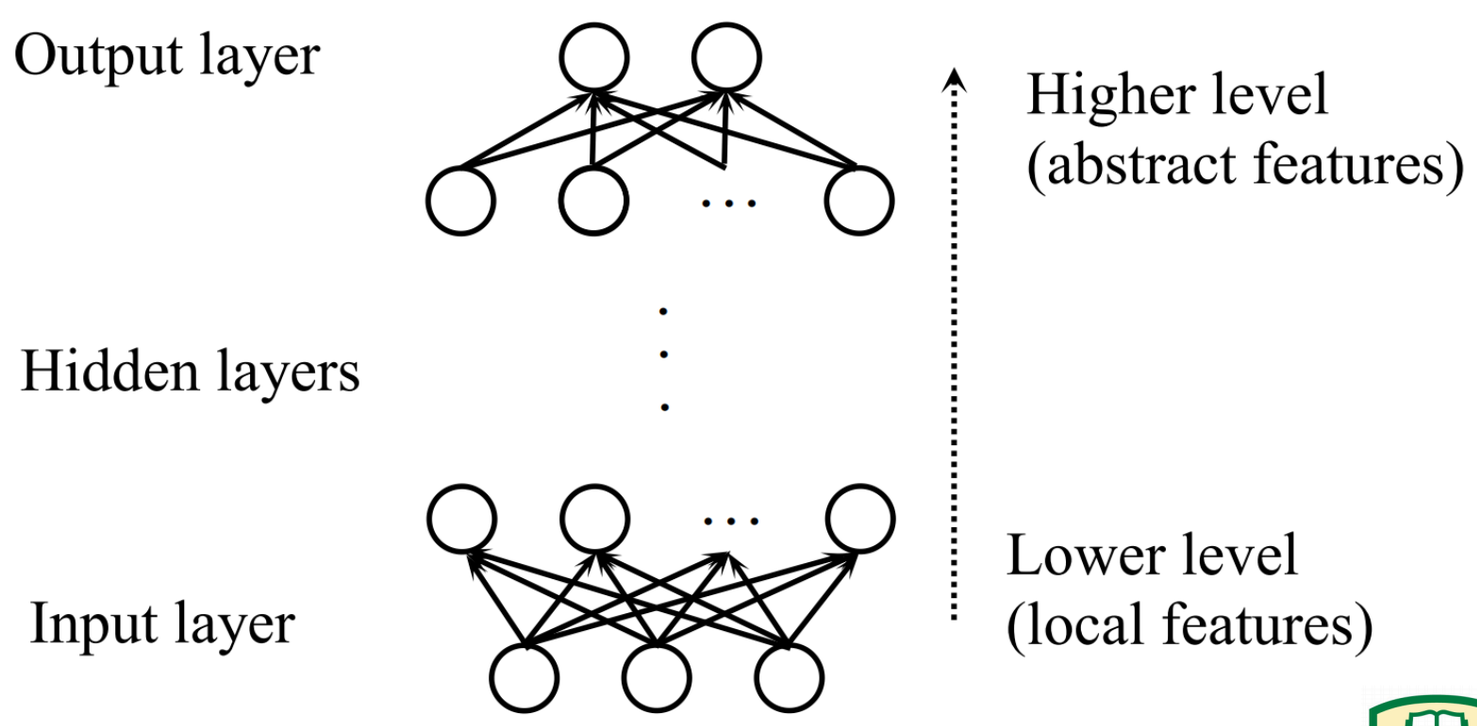
Source: https://en.wikipedia.org/wiki/Neuron

Artificial Neural Network

- Layer-wise fully connected neural networks
  - A **layer** of neutrons



  - Multilayer perceptrons



A Note on Model Capacity

- Layer-wise fully connected neural networks
  - "*Arbitrary decision regions can be arbitrarily well approximated by continuous feedforward neural networks with only a single internal, hidden layer and any continuous sigmoidal nonlinearity.*"

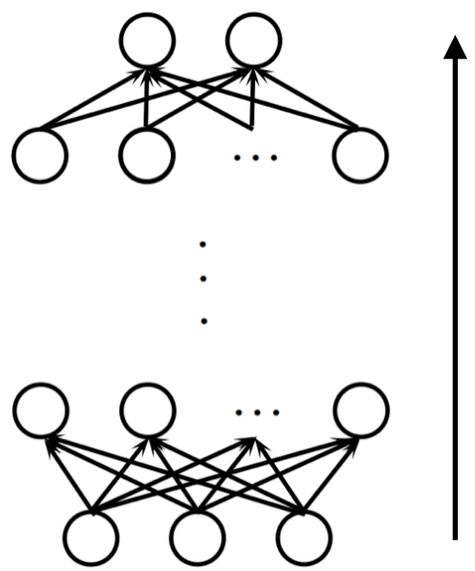
Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. *Mathematics of control, Signals and Systems*, *2*(4), pp.303-314.

Forward Propagation

- Given input $\boldsymbol{x}^{(0)}$, compute output (say, the Lth layer)
- Recursion
  - **Initialization:** Input is known
  - **Recursive step:** For each layer $\boldsymbol{x}^{(l-1)}$, we can compute $\boldsymbol{x}^{(l)}$

$$\boldsymbol{x}^{(l)} = \boldsymbol{f}(W^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b})$$

  - **Termination:** $l = L$



Backward Propagation

- BP, Backprop, Back-propagation
- Compute derivatives (from output to input)
- Recursion (on what?)

$$\frac{\partial J}{\partial \boldsymbol{x}^{(l)}}$$

Backward Propagation

- BP, Backprop, Back-propagation
- Compute derivatives (from output to input)
- Recursion on $\frac{\partial J}{\partial \boldsymbol{x}^{(l)}}$

  - **Initialization**: $\frac{\partial J}{\partial \boldsymbol{x}^{(L)}}$ is given by the loss

Backward Propagation

- BP, Backprop, Back-propagation
- Compute derivatives (from output to input)
- Recursion
  - **Initialization**: $\frac{\partial J}{\partial \boldsymbol{x}^{(L)}}$ is given by the loss
  - **Recursive step:**

    Suppose $\frac{\partial J}{\partial \boldsymbol{x}^{(l)}}$ is known, we compute $\frac{\partial J}{\partial \boldsymbol{x}^{(l-1)}}$

Chain Rule

- **Recursive step:**

  Suppose $\frac{\partial J}{\partial \boldsymbol{x}^{(l)}}$ is known, we compute $\frac{\partial J}{\partial \boldsymbol{x}^{(l-1)}}$

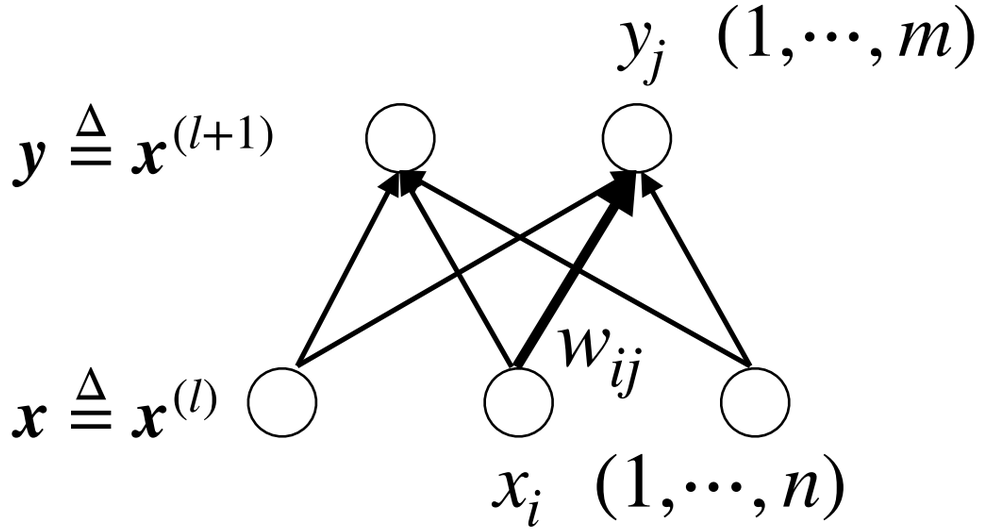
  - Forward:

$$z_j = w_{j1}x_1 + w_{j2}x_2 + \cdots + w_{jn}x_n + b_j$$

$$y_j = f(z_j)$$

  - Backward:

$$\frac{\partial J}{\partial x_i} = \frac{\partial J}{\partial z_j}\frac{\partial z_j}{\partial x_i}$$



Chain Rule

- **Recursive step:**

  Suppose $\frac{\partial J}{\partial \boldsymbol{x}^{(l)}}$ is known, we compute $\frac{\partial J}{\partial \boldsymbol{x}^{(l-1)}}$

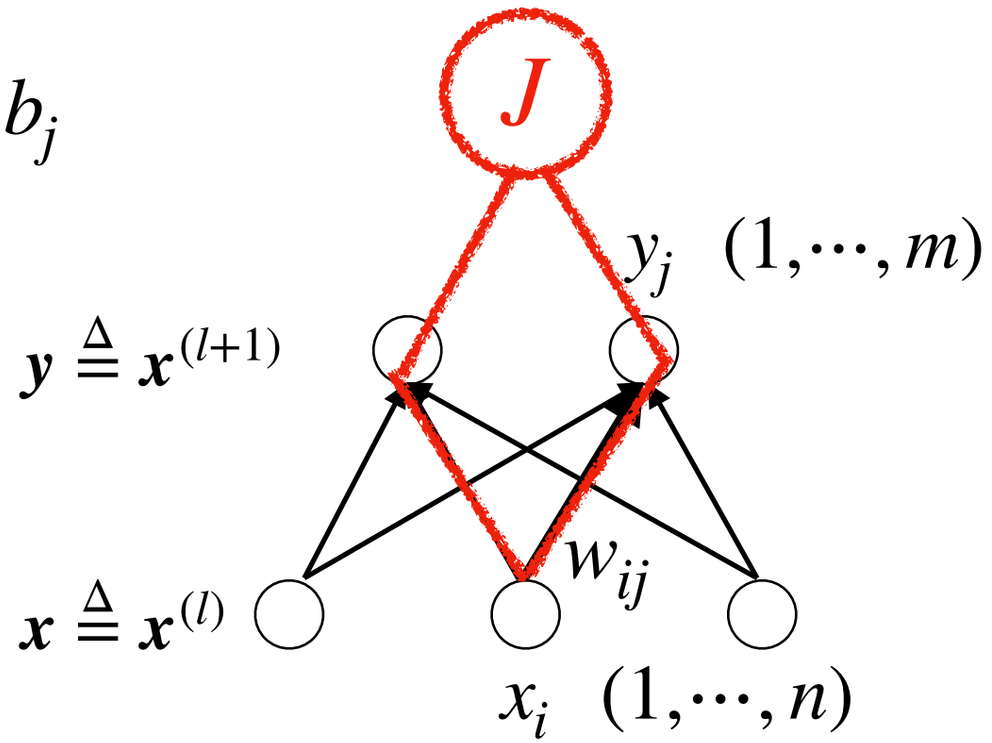
  - Forward:

$$z_j = w_{j1}x_1 + w_{j2}x_2 + \cdots + w_{jn}x_n + b_j$$

$$y_j = f(z_j)$$

  - Backward:

$$\frac{\partial J}{\partial x_i} = \frac{\partial J}{\partial z_j}\frac{\partial z_j}{\partial x_i}$$



Chain Rule

- Suppose $\frac{\partial J}{\partial \boldsymbol{x}^{(l+1)}}$ is known, we compute $\frac{\partial J}{\partial \boldsymbol{x}^{(l)}}$
  - Forward:

$$z_j = w_{j1}x_1 + w_{j2}x_2 + \cdots + w_{jn}x_n + b_j$$

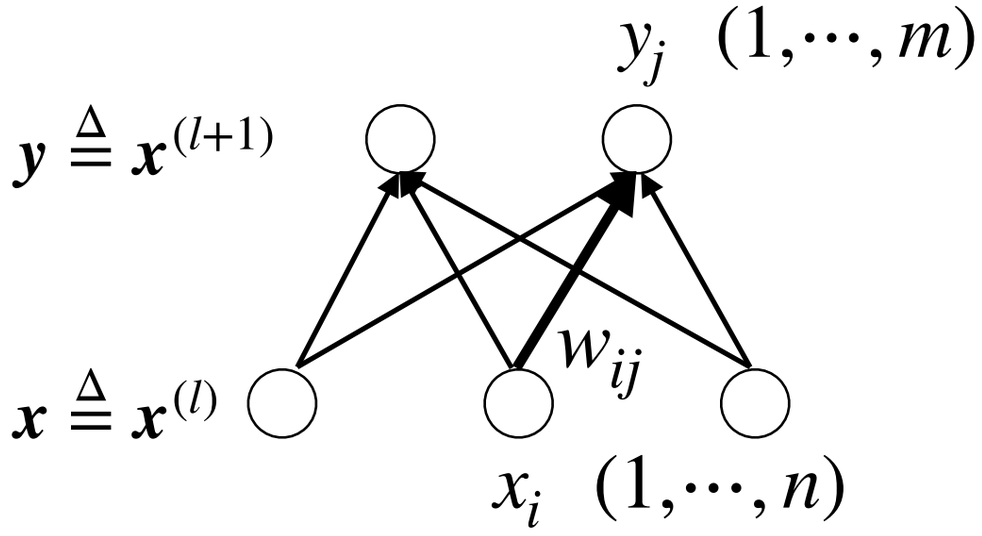
$$y_j = f(z_j)$$

  - Backward:

$$\frac{\partial J}{\partial x_i} = \sum_j \frac{\partial J}{\partial z_j}\frac{\partial z_j}{\partial x_i}$$

$$= \sum_j \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial x_i}$$

if $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ is pointwise



Chain Rule

Softmax derivative

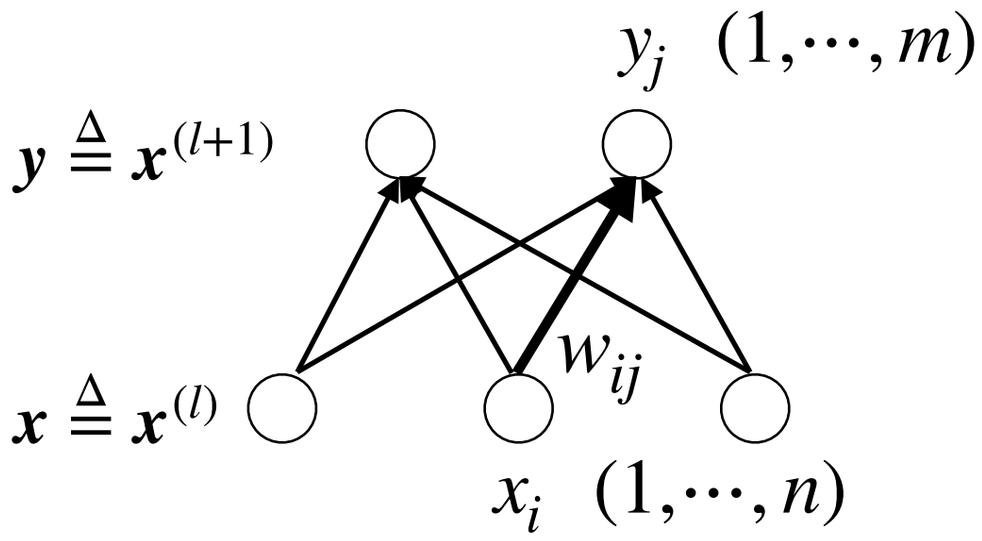
$$\frac{\partial J}{\partial z_i} = y_i - t_i$$

t_i: one-hot representation of groundtruth

$$\frac{\partial J}{\partial x_i} = \sum_j \frac{\partial J}{\partial z_j}\frac{\partial z_j}{\partial x_i}$$

$$= \sum_j \frac{\partial J}{\partial y_j}\frac{\partial y_j}{\partial z_j}\frac{\partial z_j}{\partial x_i}$$

if $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x})$ is pointwise



UNIVERSITY OF ALBERTA

Chain Rule

- Suppose $\frac{\partial J}{\partial \boldsymbol{x}^{(l+1)}}$ is known, we compute $\frac{\partial J}{\partial \boldsymbol{x}^{(l)}}$
  - Forward:

$$z_j = w_{j1}x_1 + w_{j2}x_2 + \cdots + w_{jn}x_n + b_j$$

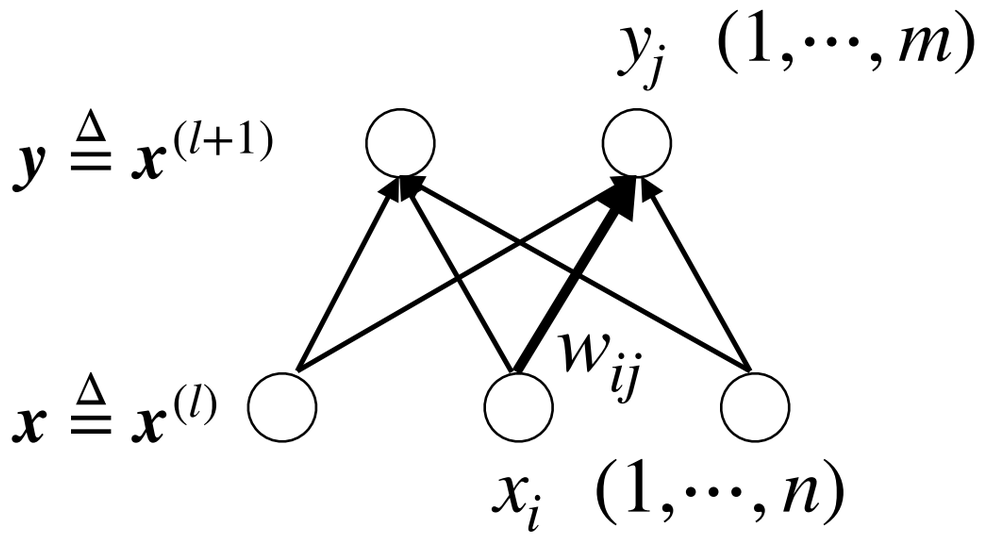
$$y_j = f(z_j)$$

  - Backward:

$\frac{\partial J}{\partial y_j}$ good (previous slide)

$$\frac{\partial J}{\partial w_{ji}} = \frac{\partial J}{\partial z_j} x_i$$

$$\frac{\partial J}{\partial b_j} = \frac{\partial J}{\partial z_j}$$



Back Propagation

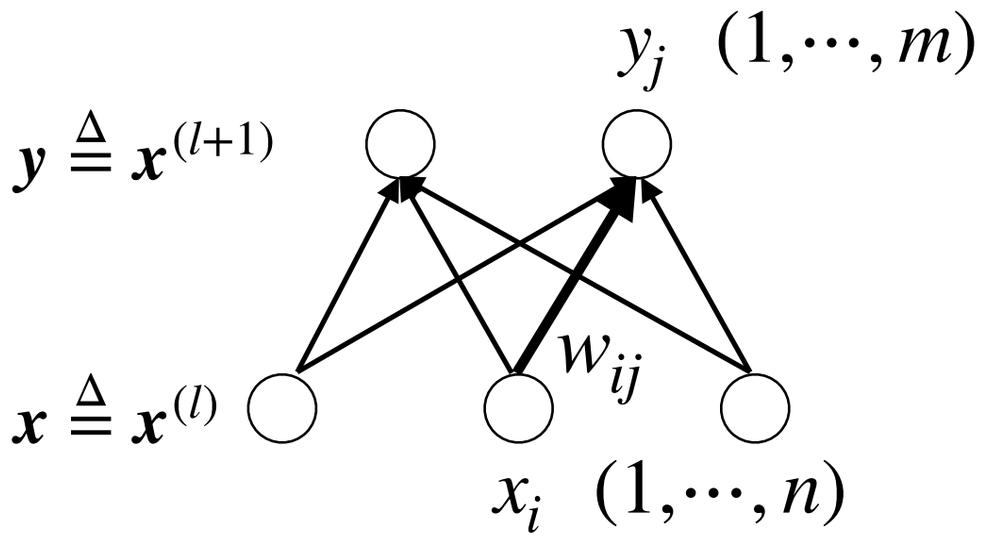
- **Initiliazation:** $\frac{\partial J}{\partial \boldsymbol{x}^{(L)}}$ is given by the loss

- **Recursive step:**

  $\frac{\partial J}{y_j}, \frac{\partial J}{\partial w_{ji}}, \frac{\partial J}{\partial b_j}$ all good

- **Termination:**

  $\frac{\partial J}{\partial w_{ji}}, \frac{\partial J}{\partial b_j}$ done for all layers



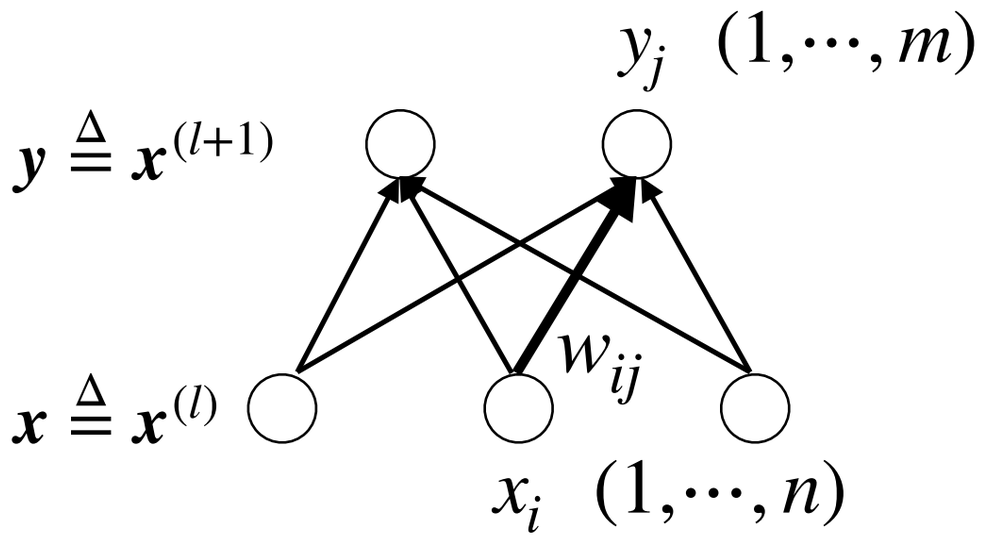
Vectorized Implementation

- Forward propagation:

$$z_j = w_{j1}x_1 + w_{j2}x_2 + \cdots + w_{jn}x_n + b_j$$
$$y_j = f(z_j)$$

$$\boldsymbol{z} = \boldsymbol{x}W + \boldsymbol{b}$$
$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{z})$$

$$\boldsymbol{x} \in \mathbb{R}^{N_{data} \times N_{in}}$$
$$\boldsymbol{x} : \langle data \times in \rangle$$
$$\boldsymbol{z}, \boldsymbol{y} : \langle data \times out \rangle$$
$$W : in \times out$$
$$\boldsymbol{b} : out$$



UNIVERSITY OF ALBERTA

Vectorized Implementation

- Backward propagation:

$$\frac{\partial J}{\partial z} = \frac{\partial J}{\partial y} .* \left(\frac{\partial y}{\partial z}\right)_{data \times out}$$

Pointwise, not Jacobian

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial z} W^\top$$

$$\frac{\partial J}{\partial W} = \boldsymbol{x}^\top \cdot \frac{\partial J}{\partial z}$$

$$\frac{\partial J}{\partial \boldsymbol{b}} = \left(\frac{\partial J}{\partial z}\right)^\top \cdot \mathbf{1}_{data \times 1}$$

(assuming sum of per-sample loss)

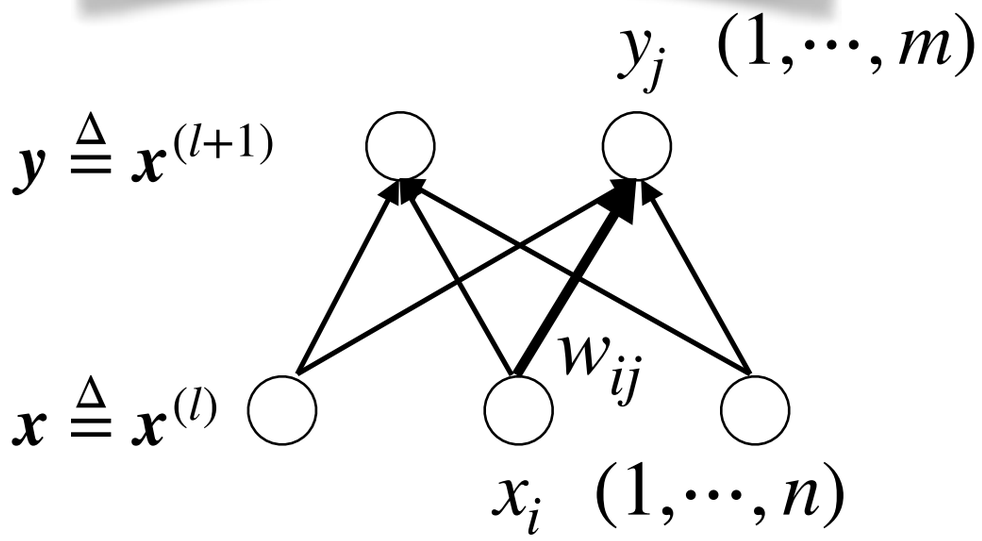
$$\boldsymbol{x} : \langle data \times in \rangle$$
$$z, \boldsymbol{y} : \langle data \times out \rangle$$
$$W : in \times out$$
$$\boldsymbol{b} : out$$

Cheatsheet

$$\boldsymbol{z} = \boldsymbol{x} W + \boldsymbol{b}$$
$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{z})$$

$$\frac{\partial J}{\partial x_i} = \sum_j \frac{\partial J}{\partial z_j} \frac{\partial z_j}{\partial x_i}$$

$$= \sum_j \frac{\partial J}{\partial y_j} \frac{\partial y_j}{\partial z_j} \frac{\partial z_j}{\partial x_i}$$



UNIVERSITY OF ALBERTA

A Few More Thoughts

- Non-layerwise connection
  - Topological sort

- Multiple losses
  - BP is a linear system

- Tied weights
  - Total derivative

Autodiff in General

- Input: <Layers, Edges, Losses>
- Algorithm:
  - Topological sort of all layers
  - Apply losses at respective layers
  - For layer L from last to first:

    For each lower layer ℓ of L

    ℓ.gradient += BP from L

Numerical Gradient Checking

- How can I know if my bp is correct?

- Definition of partial derivative

$$\frac{\partial}{\partial x_i} f(x_1, \cdots, x_i, \cdots x_n) \stackrel{def}{=} \lim_{\delta \to 0} \frac{f(x_1, \cdots, x_i + \delta, \cdots, x_n) - f(x_1, \cdots, x_i, \cdots x_n)}{\delta}$$

- Numerical gradient checking

$$\frac{\partial}{\partial x_i} f(x_1, \cdots, x_i, \cdots x_n) \approx \frac{f(x_1, \cdots, x_i + \delta, \cdots, x_n) - f(x_1, \cdots, x_i - \delta, \cdots, x_n)}{2\delta}$$

Practical Guide of Training DNN

- Weight initialization
- Batch normalization [Ioffe & Szegedy, 2015]
- Layer normalization [Ba et al., 2016]
- Dropout [Srivastava et al., 2014]
- Optimization algorithm (e.g., Adam)

  [Kingma & Ba, 2014]
- Bias-variance tradeoff for DL

  [Zhang et al., 2016]

References

Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167. 2015.

Ba JL, Kiros JR, Hinton GE. Layer normalization. arXiv preprint arXiv:1607.06450. 2016.

Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research. 2014 Jan 1;15(1):1929-58.

Kingma DP, Ba J. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980. 2014.

Zhang, Chiyuan, et al. "Understanding deep learning requires rethinking generalization." *arXiv preprint arXiv:1611.03530.* 2016.

Coding Assignment #2

- **Due: Monday, Sep 30 (Acceptable until Oct 7)**
- Adapt the logistic regression in Assignment#1 to a two-layer neural network

CMPUT 651 (Fall 2019)

Thank you!

Q&A

UNIVERSITY OF ALBERTA