Neural Networks in Structured Prediction

November 17, 2015

HWs and Paper

- Last homework is going to be posted soon
  - Neural net NER tagging model
  - This is a new structured model
- Paper - Thursday after Thanksgiving (Dec 3)
  - Bring draft of paper to class for discussion

Goals for This Week's Lectures

- Overview of neural networks (terminology, basic architectures, learning)

- Neural networks in structured prediction:

  - Option 1: locally nonlinear factors in globally linear models

  - Option 2: operation sequence models

  - Option 3: global, nonlinear structured models [speculative]

Neural Nets: Big Ideas

- Nonlinear function classes
- Learning via "backpropagation" of errors
- Neural networks as feature inducers

$$\hat{\mathbf{y}} = \mathbf{W}\mathbf{x} + \mathbf{b}$$

What features should we use??

- The multitask hypothesis

Recall: Parameters

We want to condition on lots of information,
but recall that $\rho_{X,Y}(x,y) = \rho_{X|Y=y}(x)\rho_Y(y)$

$$O(xy + y) = O(xy)$$

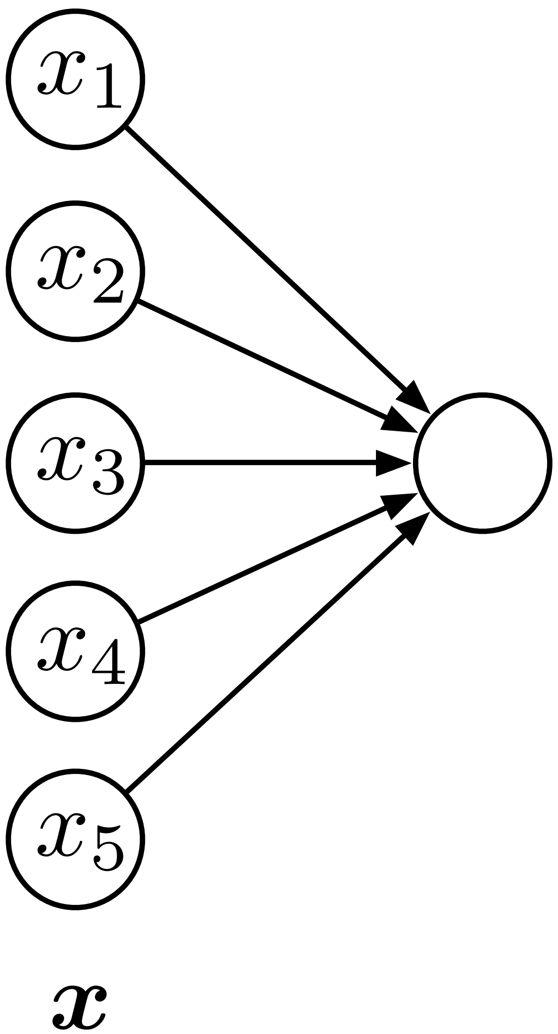
Neural networks let us learn arbitrary joint interactions, but **control the number parameters.**

- Control overfitting/improve generalization
- Reduce memory requirements

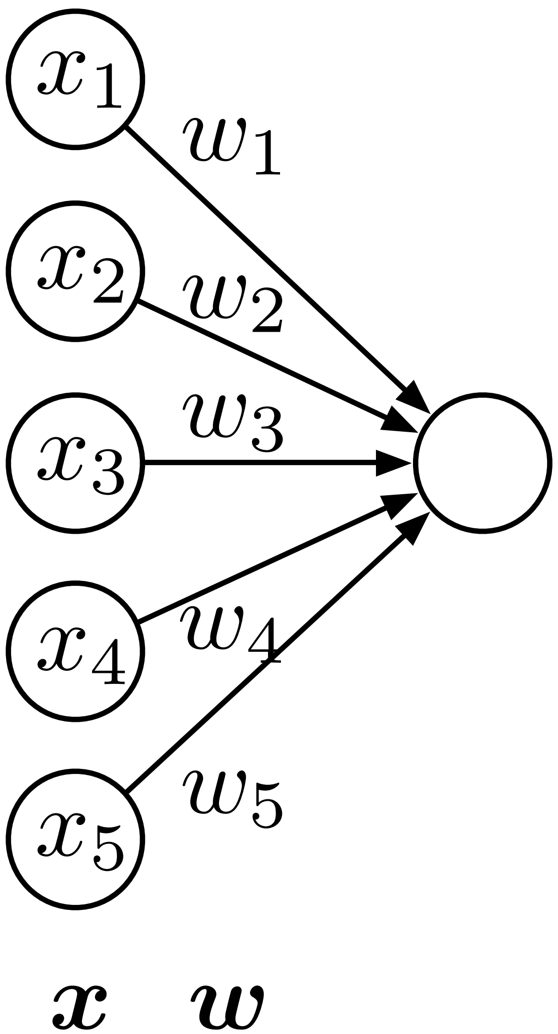
When to use them?

- If you want to condition on a lot of information (entire documents, entire sentences, …)
- But you don't know what features are useful
- Neural networks are great!

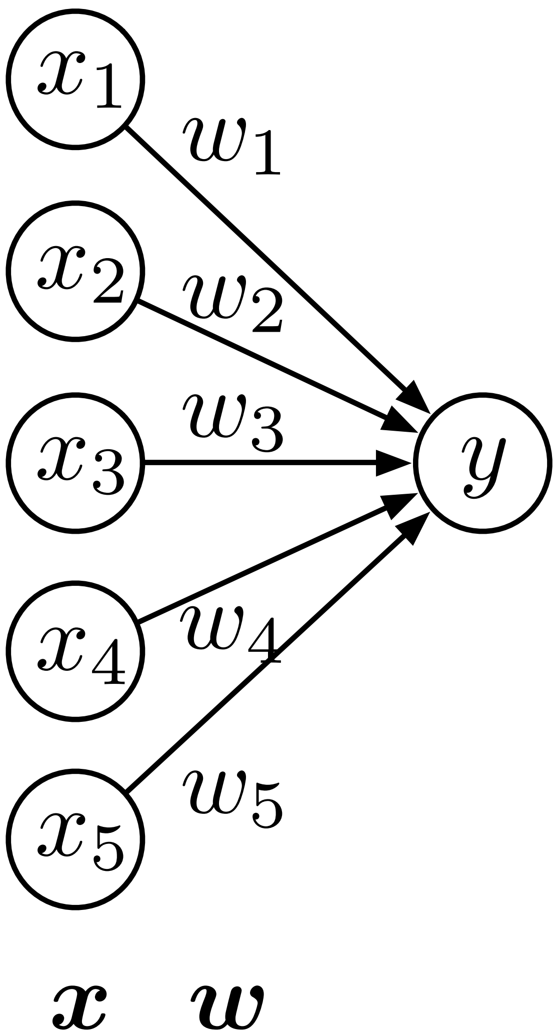
“Neurons”



"Neurons"

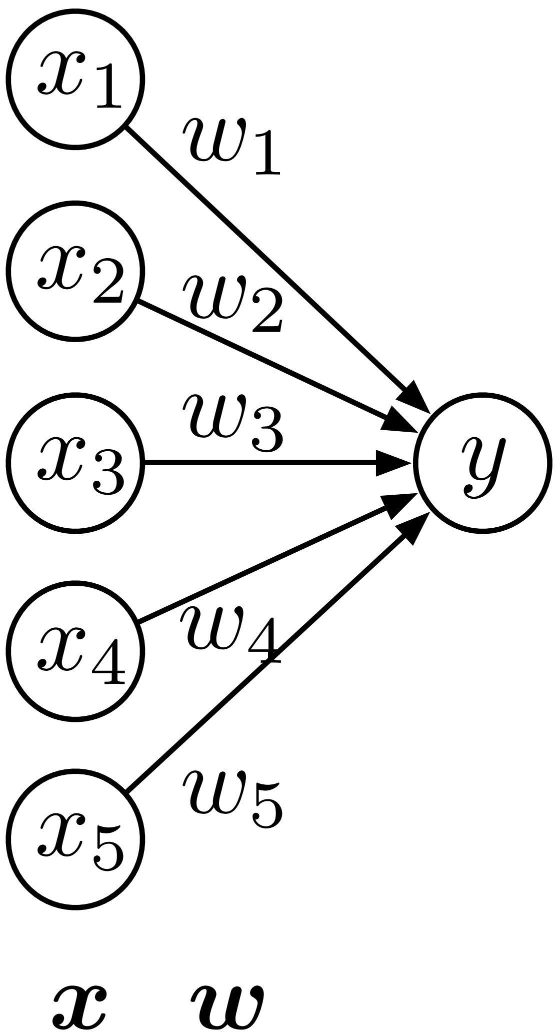


“Neurons”



$$y = \boldsymbol{x} \cdot \boldsymbol{w}$$

“Neurons”

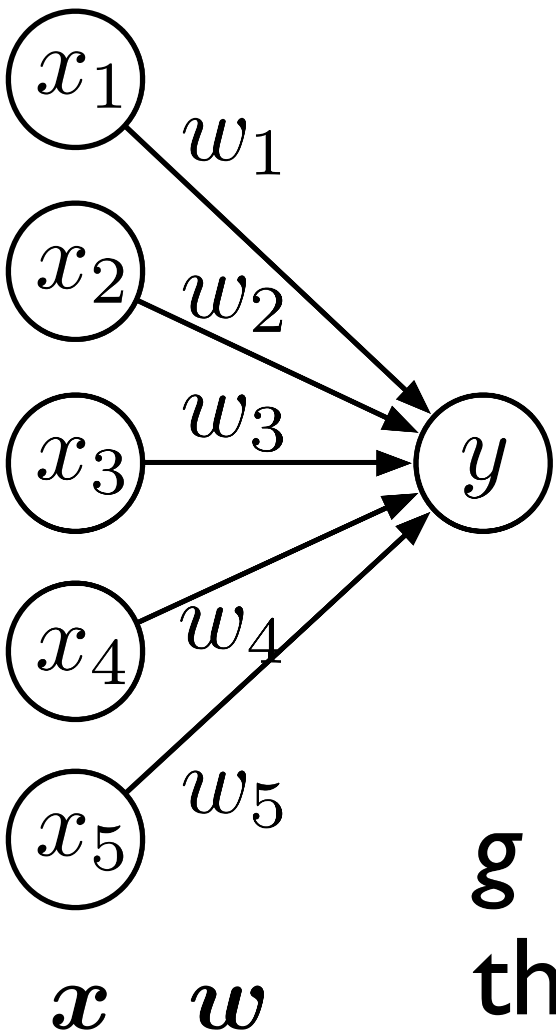


$$\cancel{y = \boldsymbol{x} \cdot \boldsymbol{w}}$$

$$y = \boldsymbol{w} \cdot \boldsymbol{x} + b$$

b is a bias term, can also be encoded by forcing one of the inputs to always be 1.

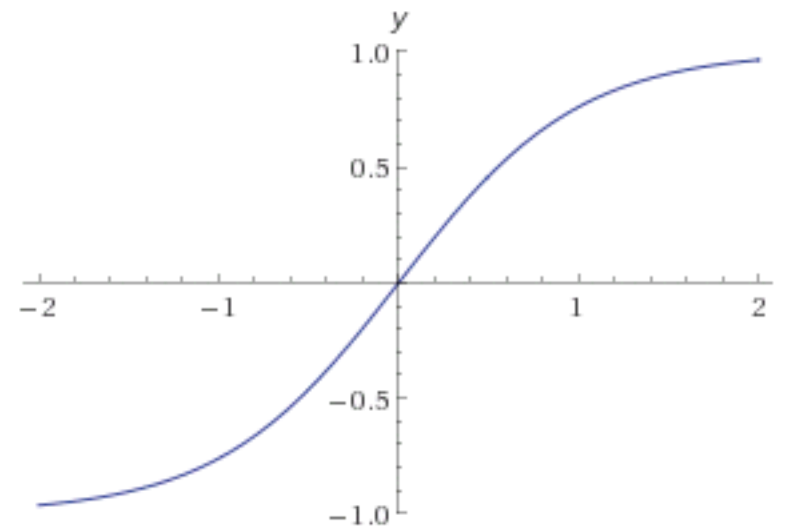
“Neurons”



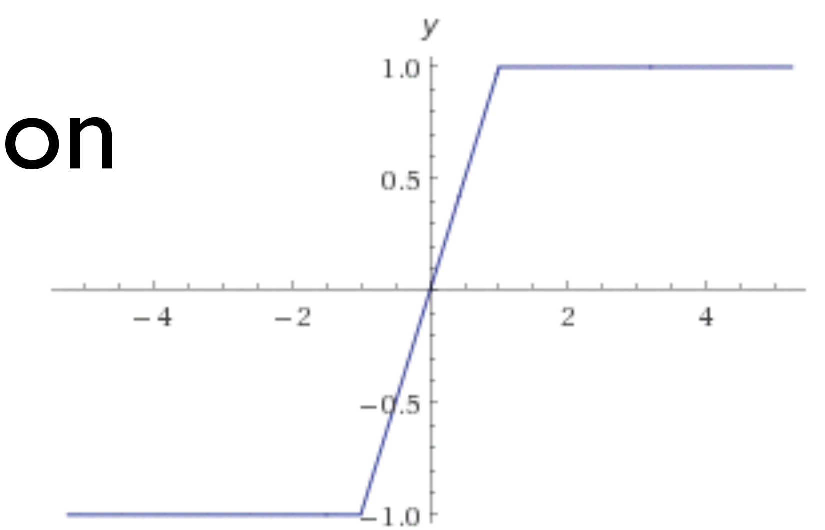
$$\cancel{y = \boldsymbol{x} \cdot \boldsymbol{w}}$$

$$\cancel{y = \boldsymbol{w} \cdot \boldsymbol{x} + b}$$

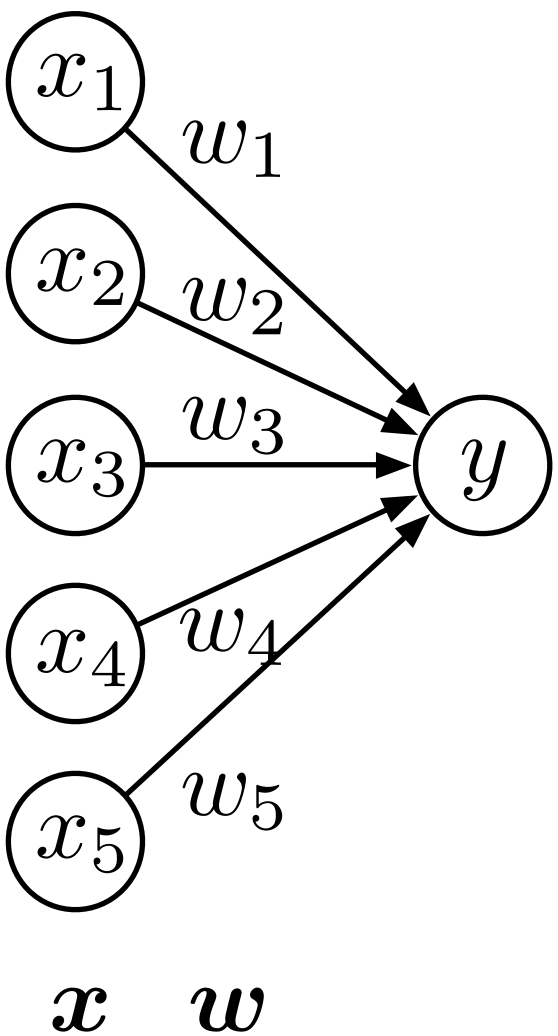
$$y = g(\boldsymbol{w} \cdot \boldsymbol{x} + b)$$



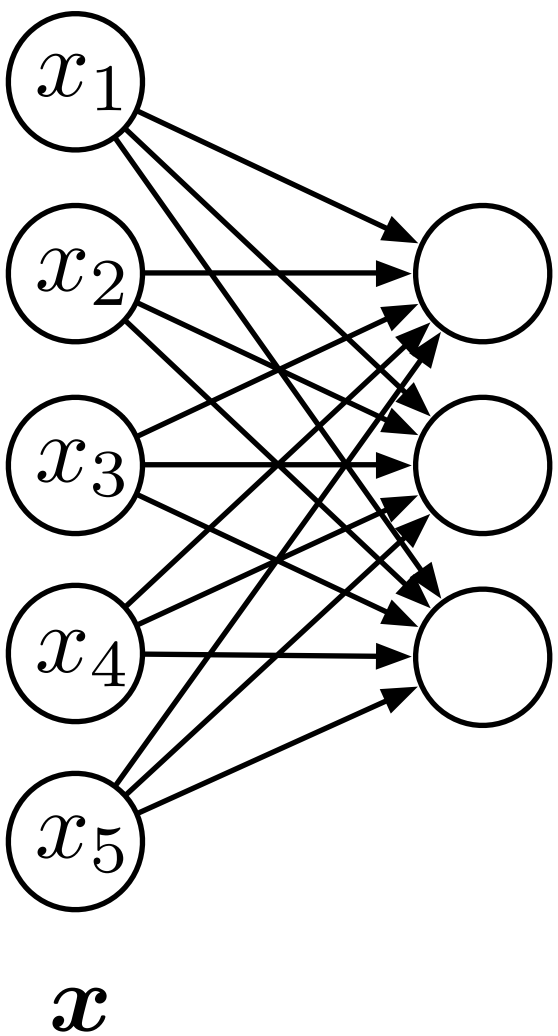
g is a nonlinear function that usually has a sigmoid shape



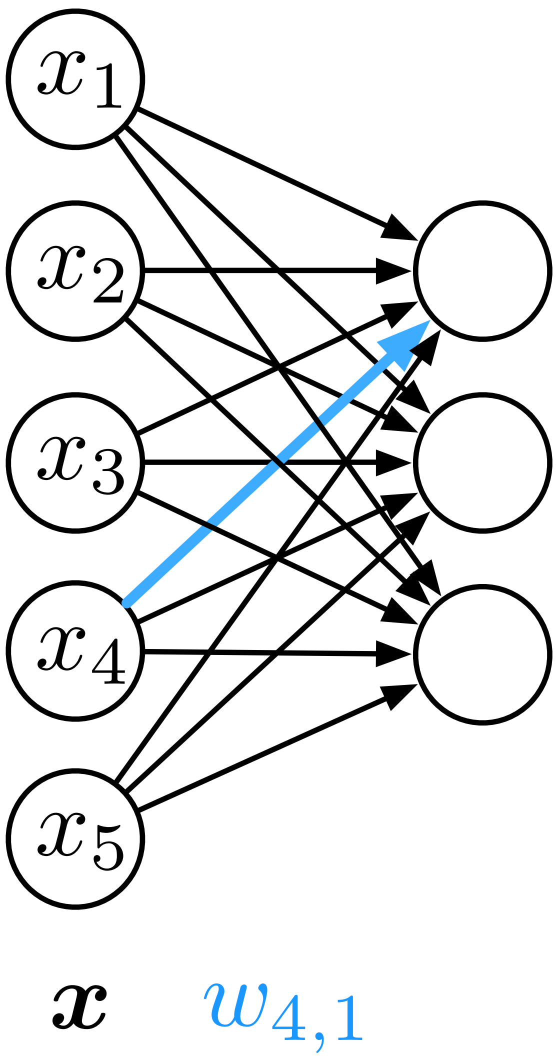
“Neurons”



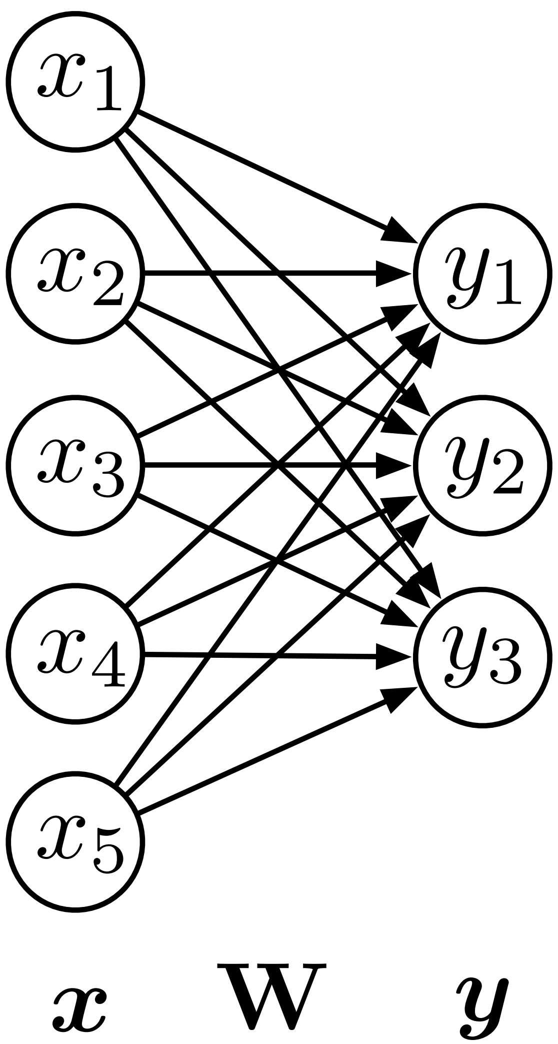
“Neural” Networks



“Neural” Networks

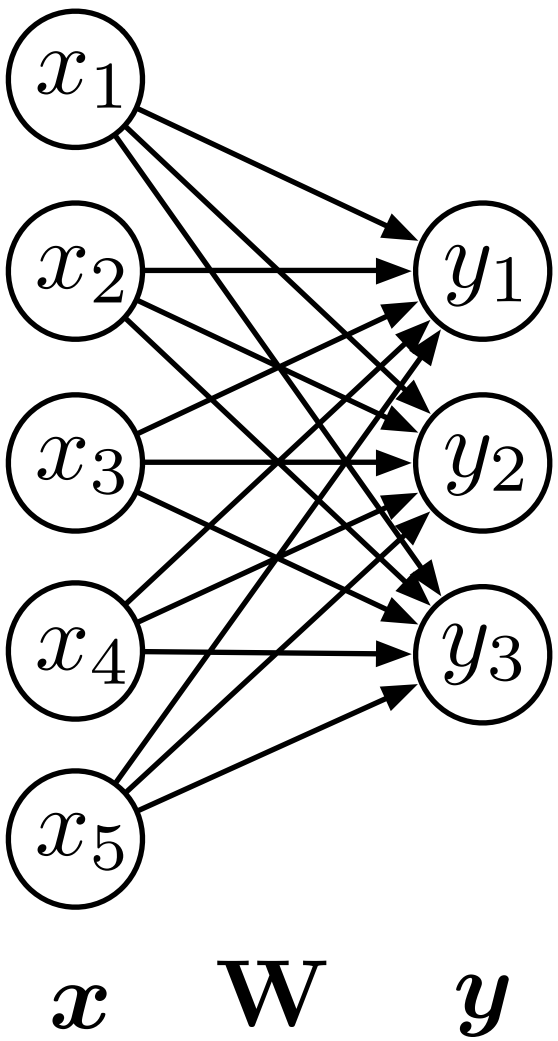


“Neural” Networks



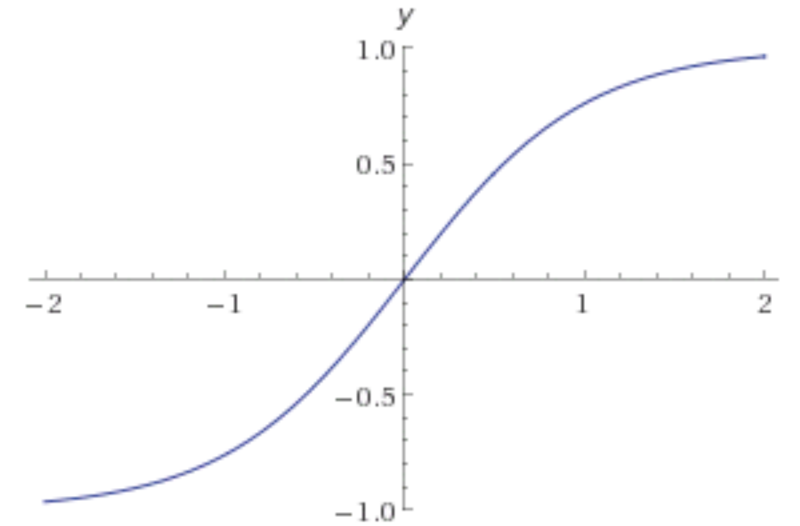
$$\boldsymbol{y} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$$

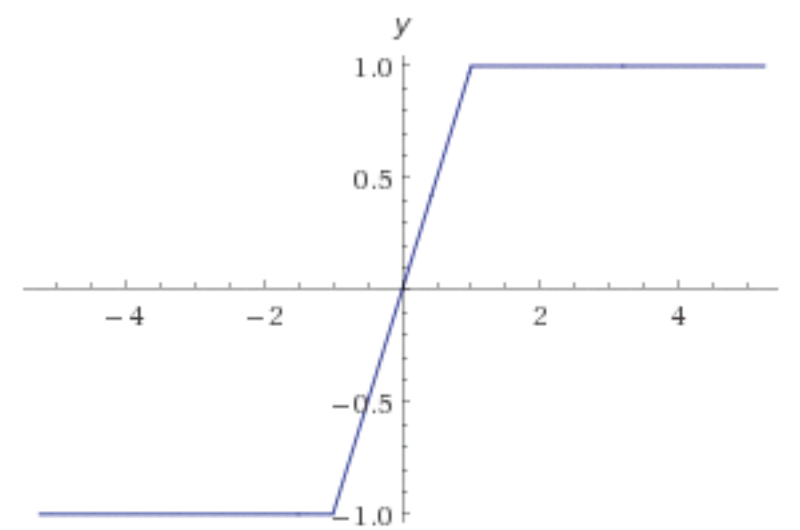
“Neural” Networks



$$\cancel{\boldsymbol{y} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}}$$

$$\boldsymbol{y} = g(\mathbf{W}\boldsymbol{x} + \boldsymbol{b})$$

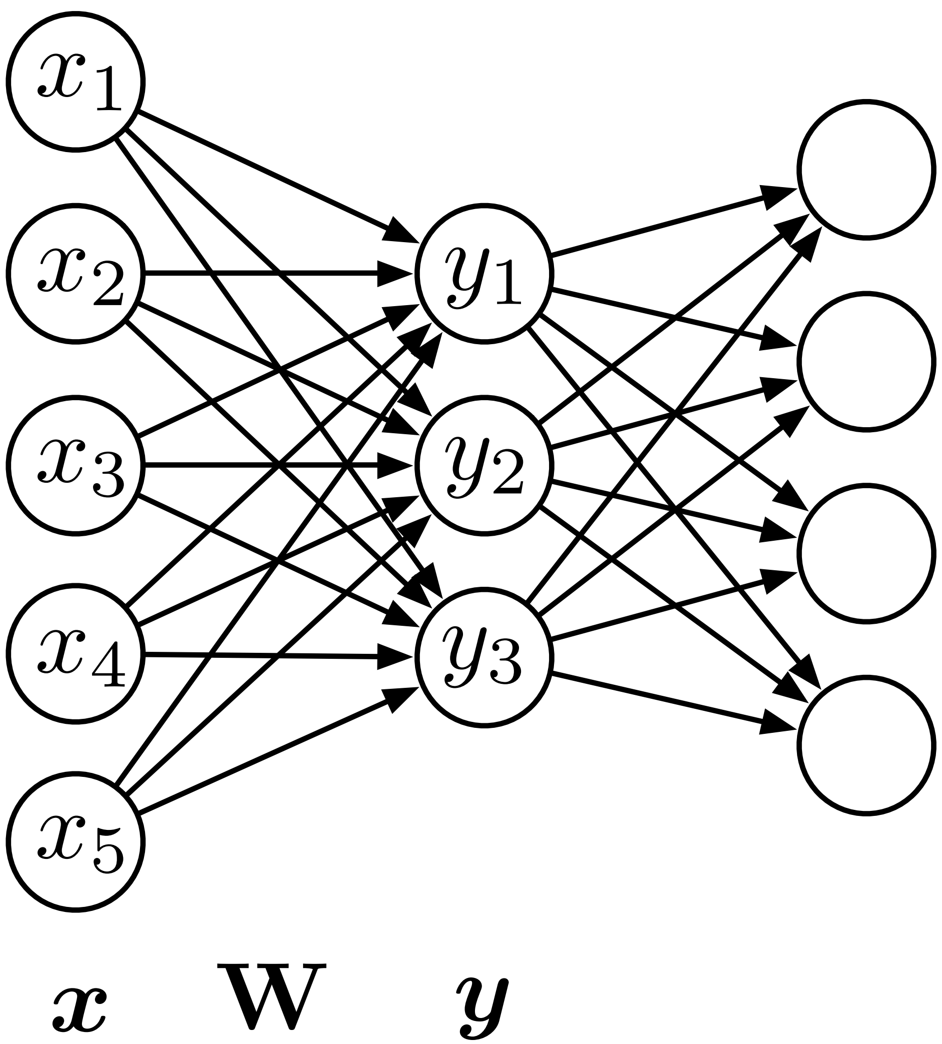




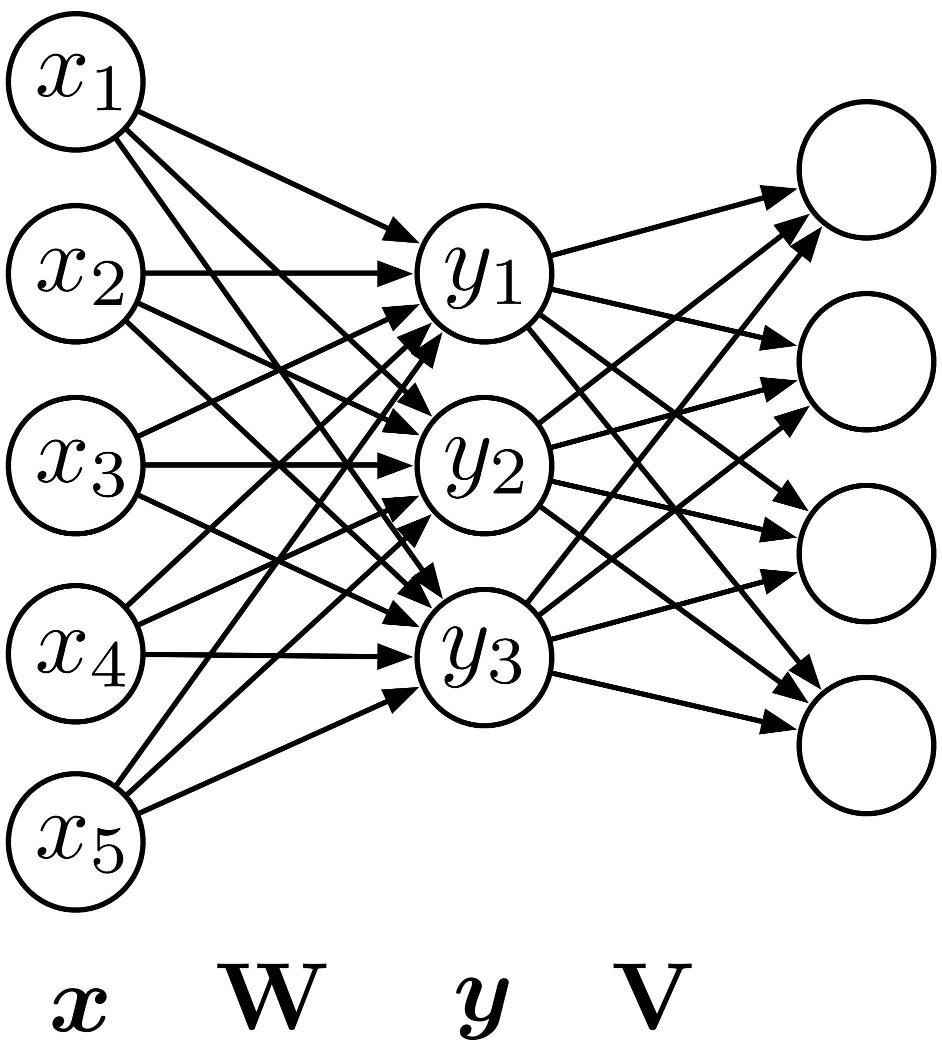
“Soft max”

$$g(\boldsymbol{u})_i = \frac{\exp u_i}{\sum_{i'} \exp u_{i'}}$$

“Deep”

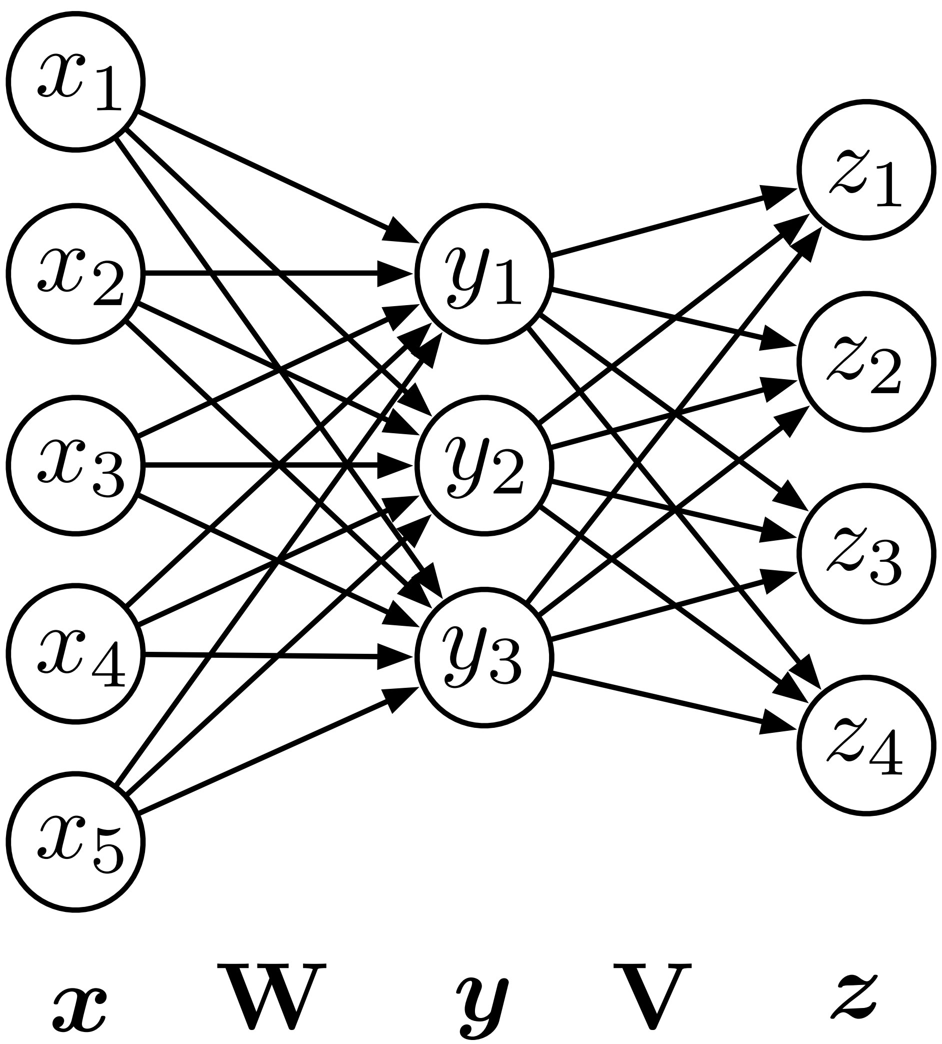


"Deep"



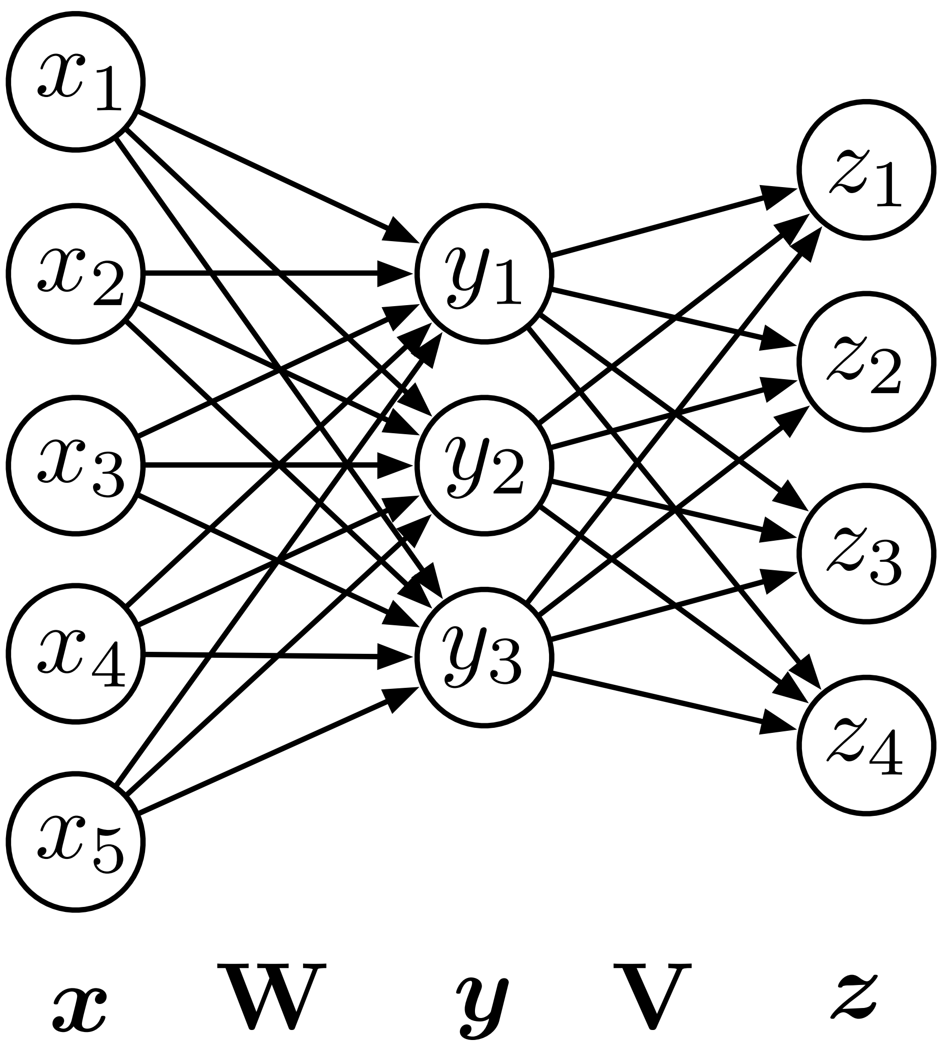
x_1
x_2
x_3
x_4
x_5
y_1
y_2
y_3
$\boldsymbol{x}$ $\mathbf{W}$ $\boldsymbol{y}$ $\mathbf{V}$

"Deep"



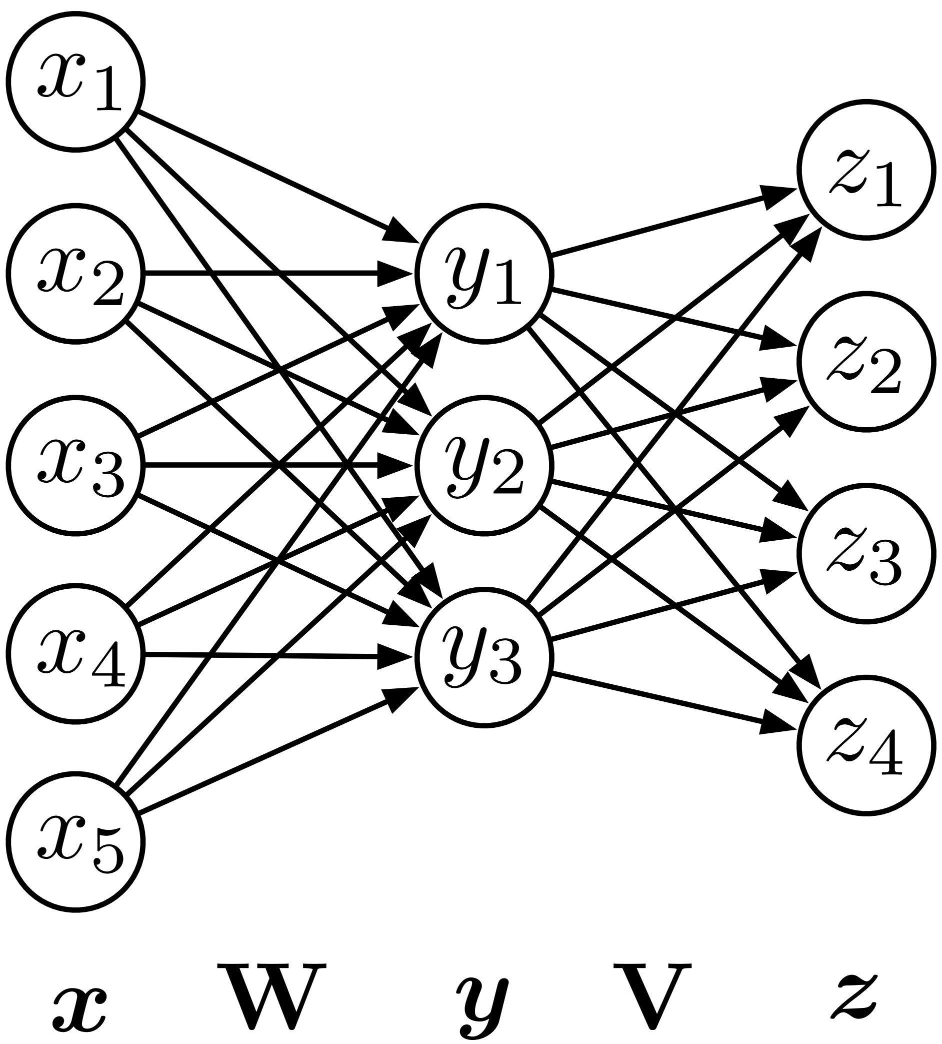
x_1
x_2
x_3
x_4
x_5
y_1
y_2
y_3
z_1
z_2
z_3
z_4
$\boldsymbol{x}$
$\mathbf{W}$
$\boldsymbol{y}$
$\mathbf{V}$
$\boldsymbol{z}$

“Deep”



$$\boldsymbol{z} = g(\mathbf{V}\boldsymbol{y} + \boldsymbol{c})$$

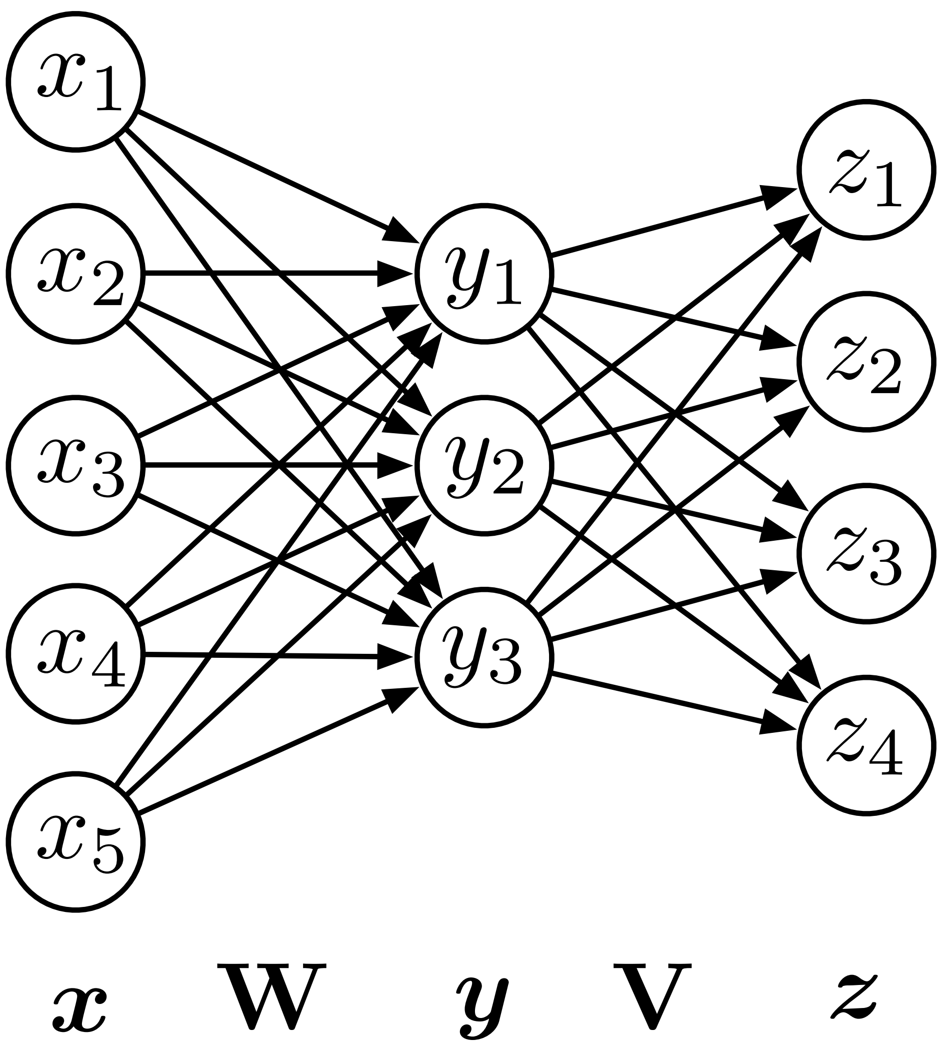
"Deep"



$$\boldsymbol{z} = g(\mathbf{V}\boldsymbol{y} + \boldsymbol{c})$$

$$\boldsymbol{z} = g(\mathbf{V}h(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}) + \boldsymbol{c})$$

“Deep”



$$\boldsymbol{z} = g(\mathbf{V}\boldsymbol{y} + \boldsymbol{c})$$

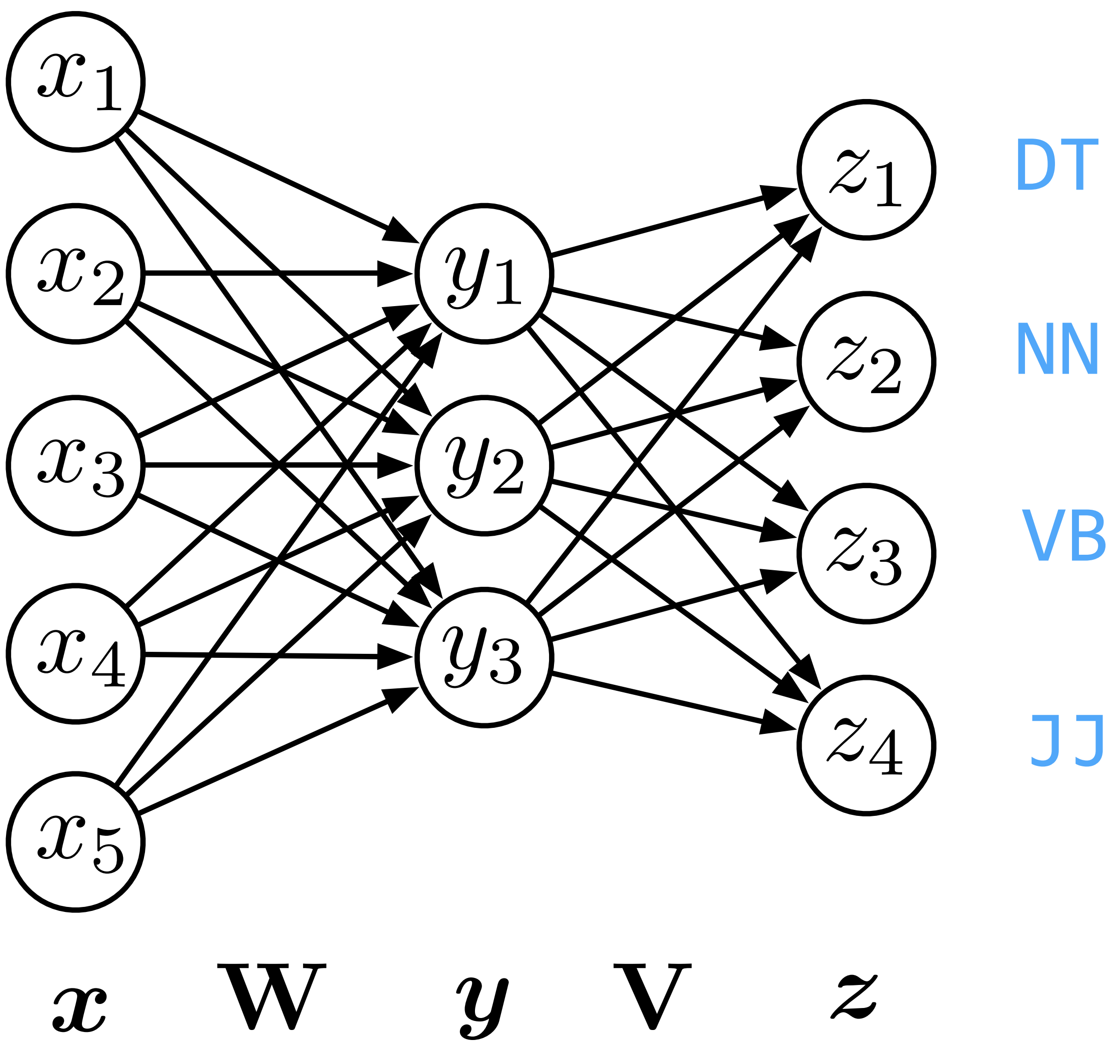
$$\boldsymbol{z} = g(\mathbf{V}h(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}) + \boldsymbol{c})$$

Note:

if $g(\boldsymbol{x}) = h(\boldsymbol{x}) = \boldsymbol{x}$

$$\begin{aligned}\boldsymbol{z} &= \mathbf{V}(\mathbf{W}\boldsymbol{x} + \boldsymbol{b}) + \boldsymbol{c} \\ &= \underbrace{\mathbf{VW}\boldsymbol{x}}_{\mathbf{U}\boldsymbol{x}} + \underbrace{\mathbf{V}\boldsymbol{b} + \boldsymbol{c}}_{\boldsymbol{d}}\end{aligned}$$

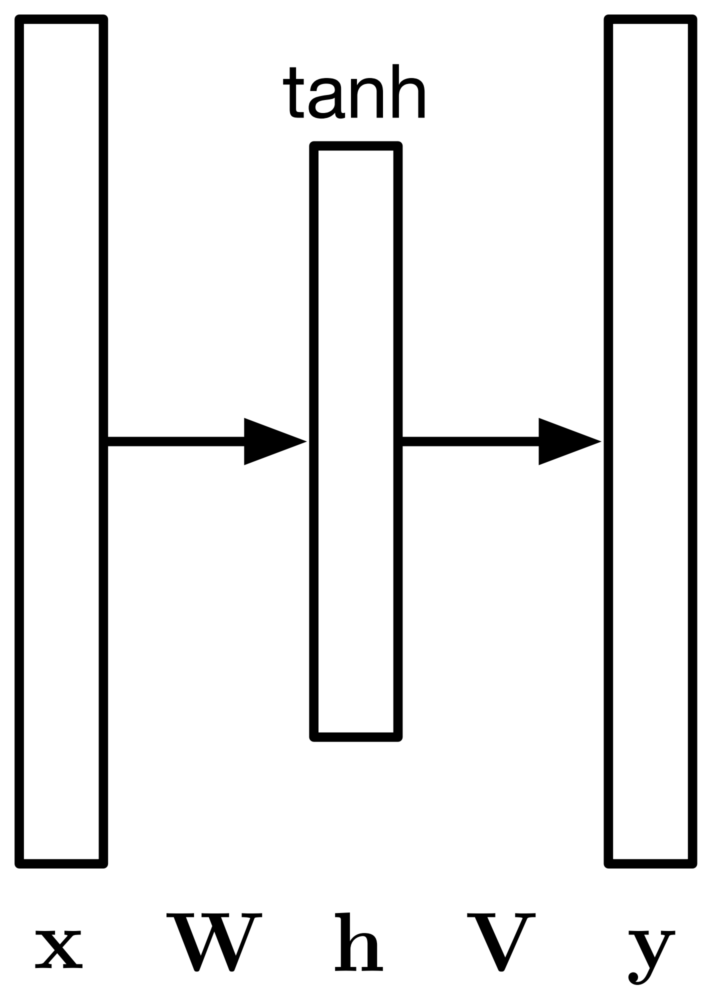
Predicting Discrete Objects



Soft max

$$g(\boldsymbol{u})_i = \frac{\exp u_i}{\sum_{i'} \exp u_{i'}}$$

More Concise



$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b})$$

$$\mathbf{y} = \mathbf{V}\mathbf{h} + \mathbf{a}$$

Feature Induction

What features should we use??

$$\hat{\boldsymbol{y}} = \mathbf{W}\boldsymbol{x} + \boldsymbol{b}$$

$$L = \sum_i ||\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i^*||_2^2$$

In linear regression the goal is to learn **W** such that L is minimized on a training set.

$$\hat{\boldsymbol{y}} = \mathbf{W}\underbrace{g(\mathbf{V}\boldsymbol{r} + \boldsymbol{c})}_{\boldsymbol{x}} + \boldsymbol{b}$$

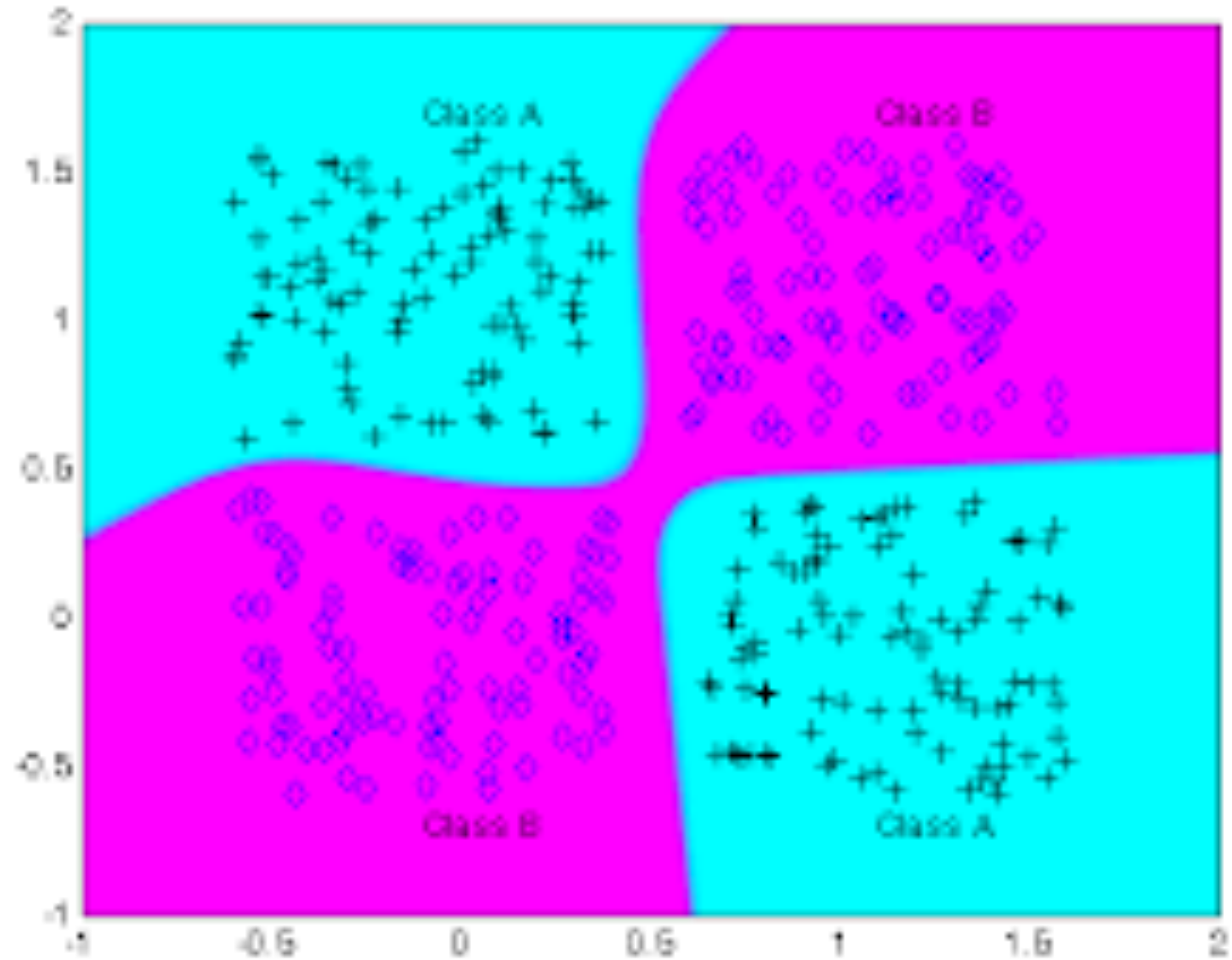
Compute features of naive representation *r*, put result in *x*
Learn to extract features that produce linear separability

Feature Induction

$$\hat{\boldsymbol{y}} = \mathbf{W}\,\underbrace{g(\mathbf{V}\boldsymbol{r} + \boldsymbol{c})}_{\boldsymbol{x}} + \boldsymbol{b}$$

- What can this function represent?
  - If ***x*** is big enough, this can represent any function!
- This is obvious much more powerful than a linear model

Joint Interactions?



Feature Induction

- Neural networks let us use a fixed number of parameters
  - Avoid the curse of dimensionality!
- But they let us learn interactions (conjunctions) - if it is helpful to do so.
  - Let the data decide!

Training

- Neural networks are **supervised**
  - We need pairs of inputs and outputs
  - For LM: input = history, output = word
- Also needed: a differentiable **loss function**

$$\mathcal{L}(x, y) \to \mathbb{R}_+$$

- Losses over training instances sum
- Any of our favorite losses work here…

Loss Functions

- Squared Error (compare two vectors)

$$\mathcal{L} = \frac{1}{2}||\mathbf{y} - \mathbf{y}^*||^2$$

- Cross Entropy / Log Loss

$$\mathcal{L} = -\log p(\mathbf{y}^* \mid \mathbf{x})$$

Loss Functions

- Squared Error (compare two vectors)

$$\mathcal{L} = \frac{1}{2}||\mathbf{y} - \mathbf{y}^*||^2$$

- Cross Entropy / Log Loss

$$\mathcal{L} = -\log p(\mathbf{y}^* \mid \mathbf{x})$$

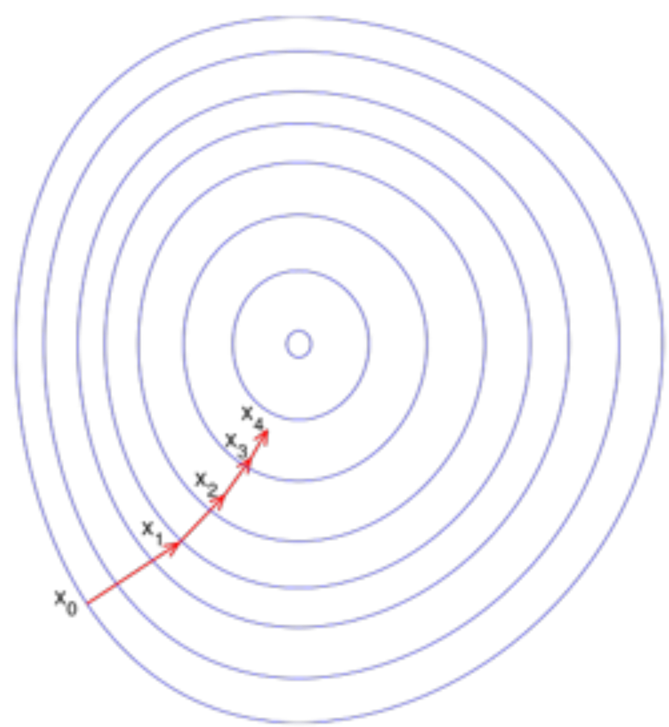
Stochastic Gradient Descent

for *i* = 1, 2, …

Pick random training example *t* and compute:

$$\boldsymbol{g}^{(i)} = \frac{\partial \mathcal{L}(x_t, y_t)}{\partial \boldsymbol{\theta}}\bigg|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(i)}}$$

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} - \eta \boldsymbol{g}^{(i)}$$



Computing Derivatives

- Training neural networks involves computing derivatives
- The standard algorithm for doing this is called backpropagation.
  - It is a variant of *automatic differentiation* (technically “reverse-mode AD”)

Automatic Differentiation?

- Compiler translates a function into a sequence of small operations
- Every small operation (nb. of a differentiable function) is itself differentiable
- The "chain rule" tells us how to compute the derivatives of composite functions using the derivatives of the pieces they are composed of

Let's start with a really simple example.

$$y = \log \sin^2 x$$

What is the derivative at x_0 ?

components	***range***	***differential***	***d-range***
$y = f(u) = \log u$	$\mathbb{R}$	$\frac{dy}{du} = \frac{1}{u}$	$\mathbb{R}$
$u = g(v) = v^2$	$\mathbb{R}$	$\frac{du}{dv} = 2v$	$\mathbb{R}$
$v = h(x) = \sin x$	$\mathbb{R}$	$\frac{dv}{dx} = \cos x$	$\mathbb{R}$

$$\left.\frac{dy}{dx}\right|_{x=x_0} = \left.\frac{dy}{du}\right|_{u=g(h(x_0))} \cdot \left.\frac{du}{dv}\right|_{v=h(x_0)} \cdot \left.\frac{dv}{dx}\right|_{x=x_0}$$

In general, for our applications $\mathbf{x}$ in $f(\mathbf{x})$ will be a *vector*.

$$y = \sum_{i=1}^{n} (\mathbf{W} \exp \mathbf{x})_i \quad \text{where} \quad \mathbf{x} \in \mathbb{R}^d \quad \text{and} \quad \mathbf{W} \in \mathbb{R}^{n \times d}$$

components	***range***	***differential***	***d-range***
$y = f(\mathbf{u}) = \sum_{i=1}^{n} u_i$	$\mathbb{R}$	$\frac{\partial y}{\partial \mathbf{u}} = \mathbf{1}$	$\mathbb{R}^{1 \times n}$
$\mathbf{u} = g(\mathbf{v}) = \mathbf{W}\mathbf{v}$	$\mathbb{R}^n$	$\frac{\partial \mathbf{u}}{\partial \mathbf{v}} = \mathbf{W}$	$\mathbb{R}^{n \times d}$
$\mathbf{v} = h(\mathbf{x}) = \exp \mathbf{x}$	$\mathbb{R}^d$	$\frac{\partial \mathbf{v}}{\partial \mathbf{x}} = \operatorname{diag}(\exp \mathbf{x})$	$\mathbb{R}^{d \times d}$

$$\left.\frac{dy}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0} = \overbrace{\left.\frac{dy}{d\mathbf{u}}\right|_{\underset{\in \mathbb{R}^n}{\mathbf{u}=g(h(\mathbf{x}_0))}}}^{\mathbb{R}^{1 \times n}} \cdot \overbrace{\left.\frac{d\mathbf{u}}{d\mathbf{v}}\right|_{\underset{\in \mathbb{R}^d}{\mathbf{v}=h(\mathbf{x}_0)}}}^{\mathbb{R}^{n \times d}} \cdot \overbrace{\left.\frac{d\mathbf{v}}{d\mathbf{x}}\right|_{\underset{\in \mathbb{R}^d}{\mathbf{x}=\mathbf{x}_0}}}^{\mathbb{R}^{d \times d}}$$

Two Evaluation Strategies

"Forward"

$$\left.\frac{dy}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0} = \overset{\mathbb{R}^{1\times n}}{\left.\frac{dy}{d\mathbf{u}}\right|_{\mathbf{u}=g(h(\mathbf{x}_0))}} \cdot \overset{\mathbb{R}^{n\times d}}{\left.\frac{d\mathbf{u}}{d\mathbf{v}}\right|_{\mathbf{v}=h(\mathbf{x}_0)}} \cdot \overset{\mathbb{R}^{d\times d}}{\left.\frac{d\mathbf{v}}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0}}$$

$\in \mathbb{R}^n$ $\in \mathbb{R}^d$ $\in \mathbb{R}^d$

"Backward" or "Adjoint"

Two Evaluation Strategies

“Forward”

$$\left.\frac{dy}{d\mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_0} = \overset{\mathbb{R}^{1\times n}}{\left.\frac{dy}{d\mathbf{u}}\right|_{\underset{\in \mathbb{R}^n}{\mathbf{u}=g(h(\mathbf{x}_0))}}} \cdot \overset{\mathbb{R}^{n\times d}}{\left.\frac{d\mathbf{u}}{d\mathbf{v}}\right|_{\underset{\in \mathbb{R}^d}{\mathbf{v}=h(\mathbf{x}_0)}}} \cdot \overset{\mathbb{R}^{d\times d}}{\left.\frac{d\mathbf{v}}{d\mathbf{x}}\right|_{\underset{\in \mathbb{R}^d}{\mathbf{x}=\mathbf{x}_0}}}$$

“Backward” or “Adjoint”

Learning with Backprop

Training data

x_1	x_2	y^*
0	0	**0**
1	0	**1**
0	1	**1**
1	1	**0**

Models

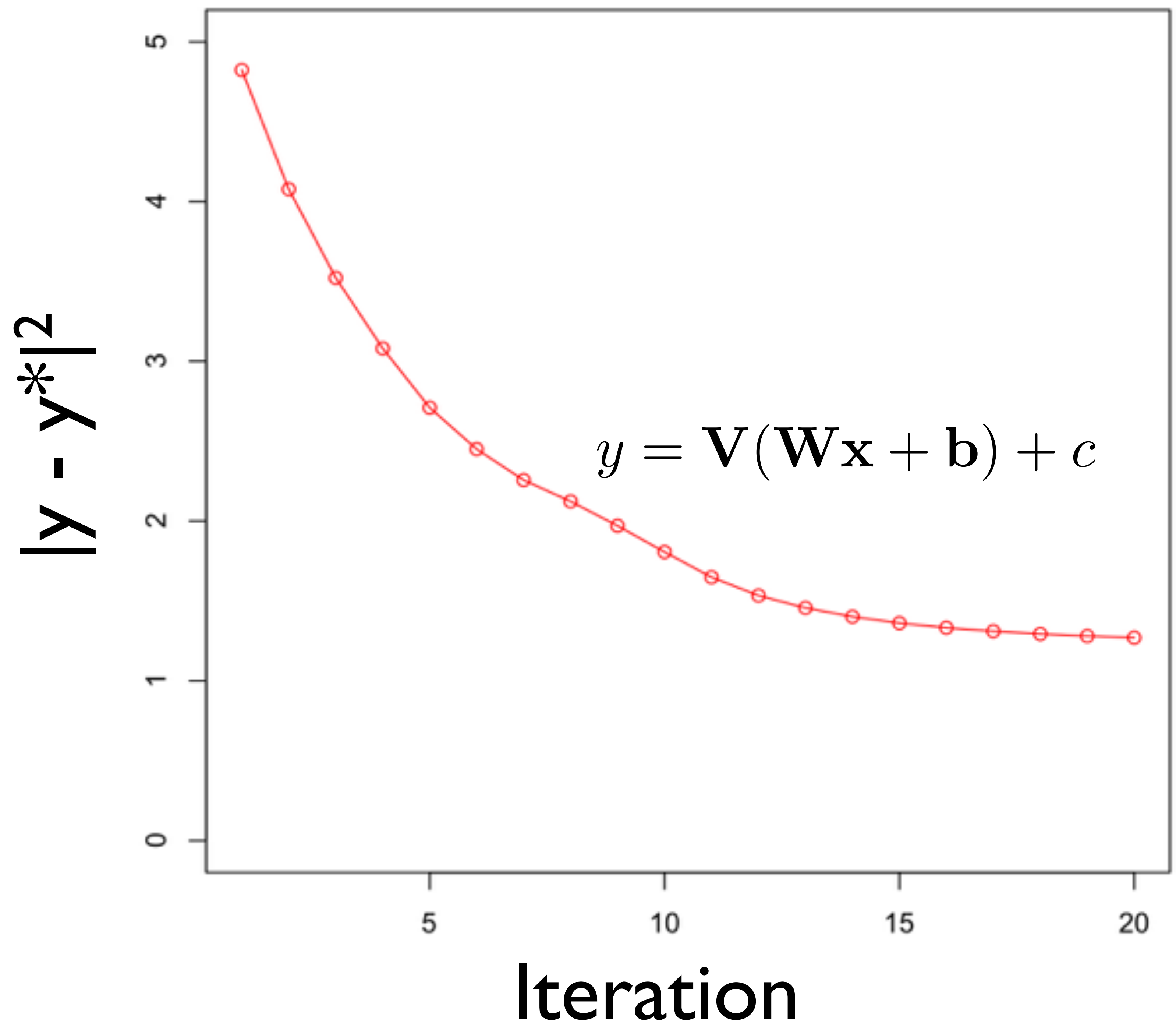
$$y = \mathbf{V}(\mathbf{W}\mathbf{x} + \mathbf{b}) + c$$

$$y = \mathbf{V}\tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) + c$$

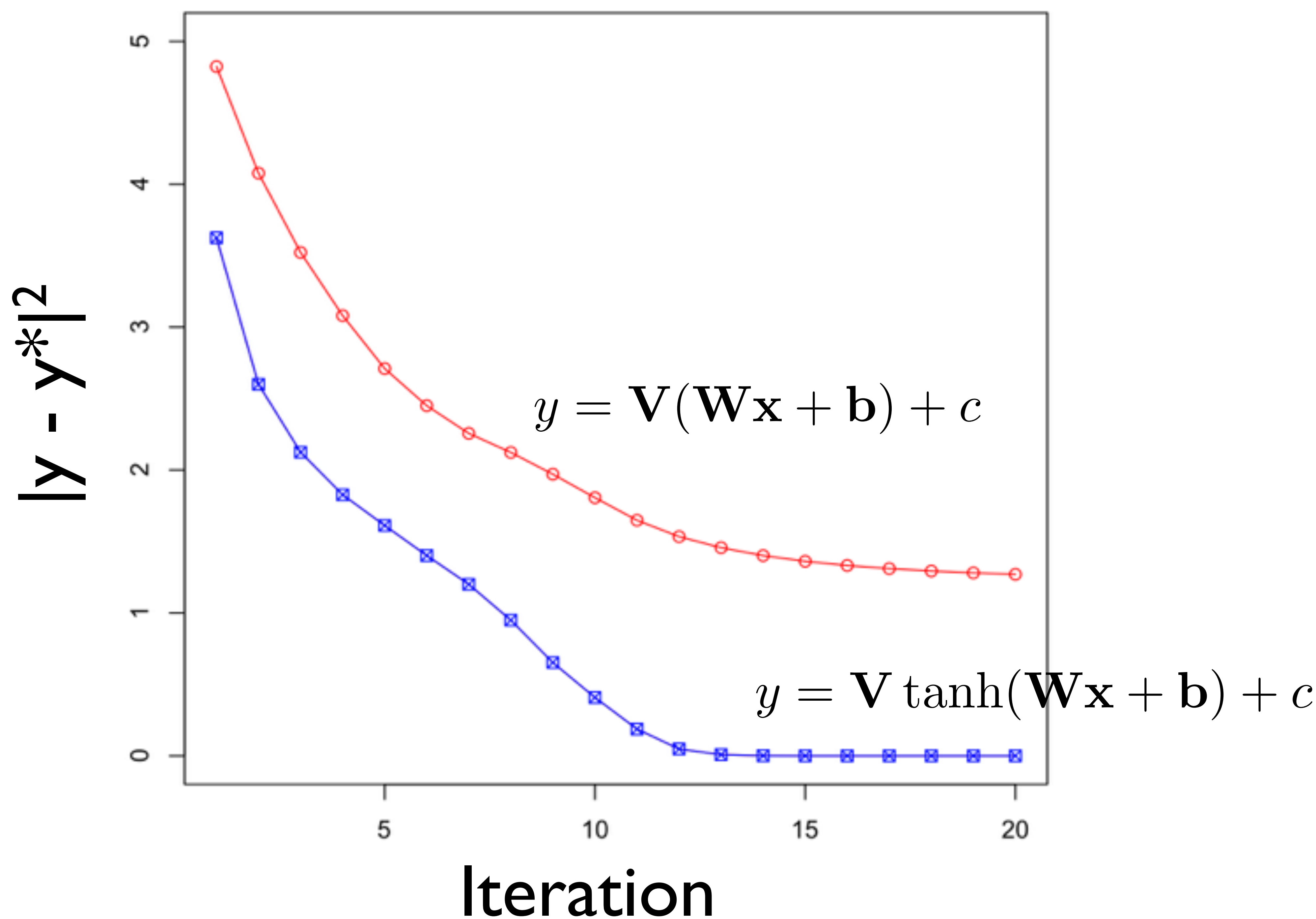
Objective

$$\mathcal{L} = (y(\mathbf{x}) - y^*)^2$$

Learning with Backprop



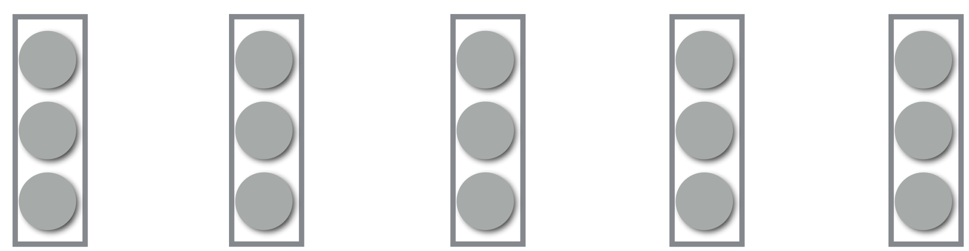
Learning with Backprop



Concept: Word Embedding

- Represent each word in a vocabulary as a vector
- Vectors are learned as parameters
- Words that behave similarly in the model will usually end up with similar embeddings

A simple tagging model



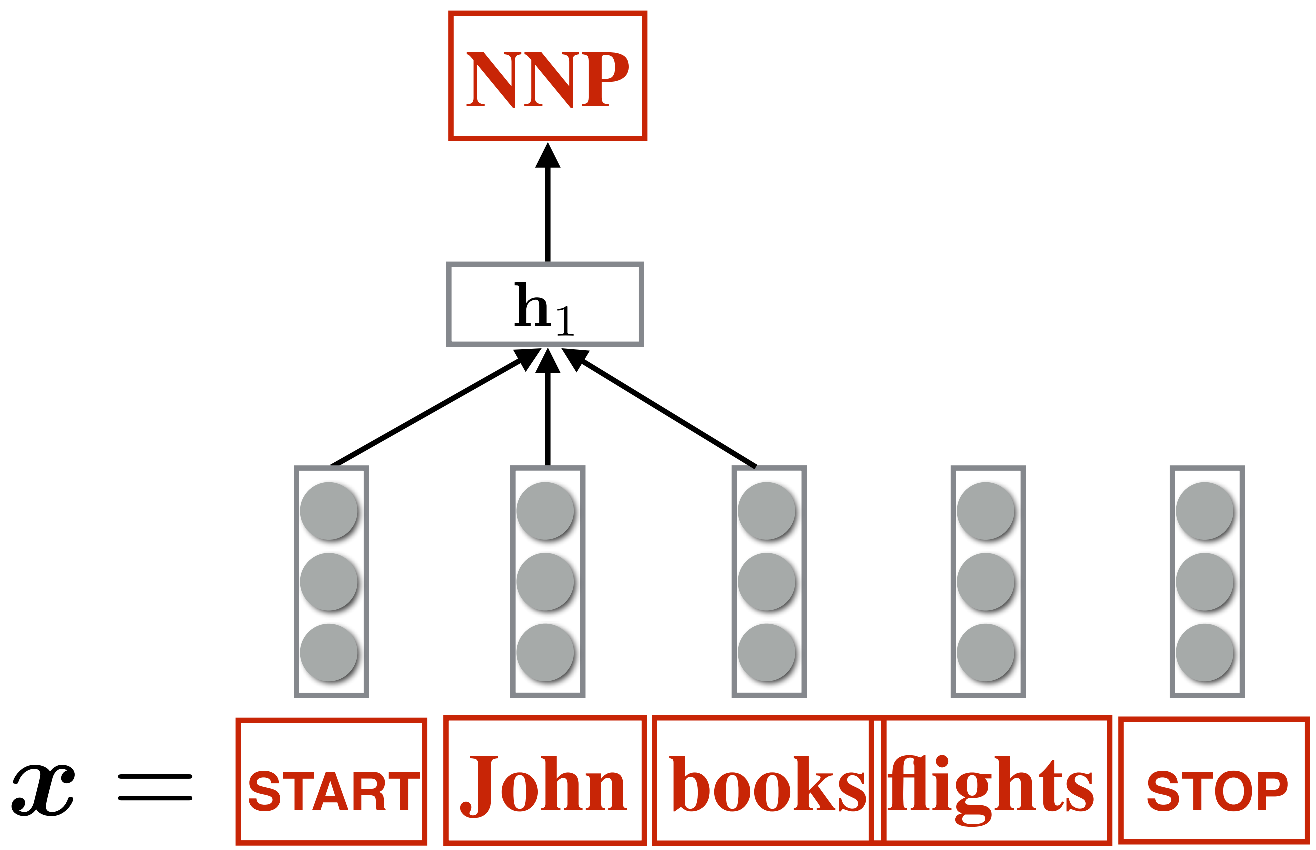
$\boldsymbol{x} =$ START | John | books | flights | STOP

$$\mathbf{h}_t = \tanh\left(\mathbf{C}_{-1}\mathbf{x}_{t-1} + \mathbf{C}_0\mathbf{x}_t + \mathbf{C}_{+1}\mathbf{x}_{t+1} + \mathbf{b}\right)$$

$$p(y_i = y \mid \boldsymbol{x}, i) = \frac{\exp \mathbf{w}_y^\top \mathbf{h}_i + a_y}{\sum_{y' \in Y} \exp \mathbf{w}_y^\top \mathbf{h}_i + a_y}$$

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{i=1}^{|\boldsymbol{y}|} p(y_i \mid \boldsymbol{x}, i)$$

A simple tagging model

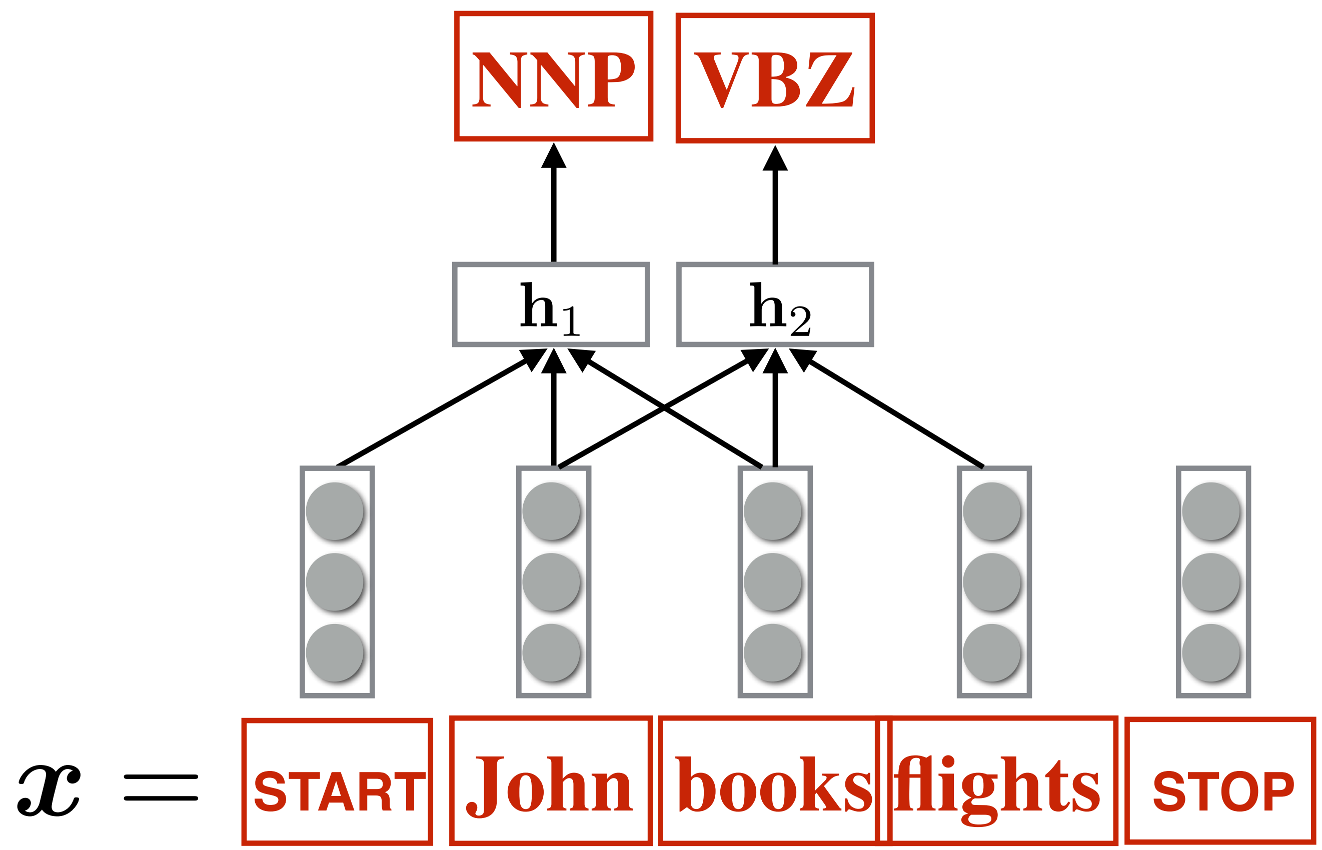


$$\mathbf{h}_t = \tanh\left(\mathbf{C}_{-1}\mathbf{x}_{t-1} + \mathbf{C}_0\mathbf{x}_t + \mathbf{C}_{+1}\mathbf{x}_{t+1} + \mathbf{b}\right)$$

$$p(y_i = y \mid \boldsymbol{x}, i) = \frac{\exp \mathbf{w}_y^\top \mathbf{h}_i + a_y}{\sum_{y' \in Y} \exp \mathbf{w}_y^\top \mathbf{h}_i + a_y}$$

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{i=1}^{|\boldsymbol{y}|} p(y_i \mid \boldsymbol{x}, i)$$

A simple tagging model

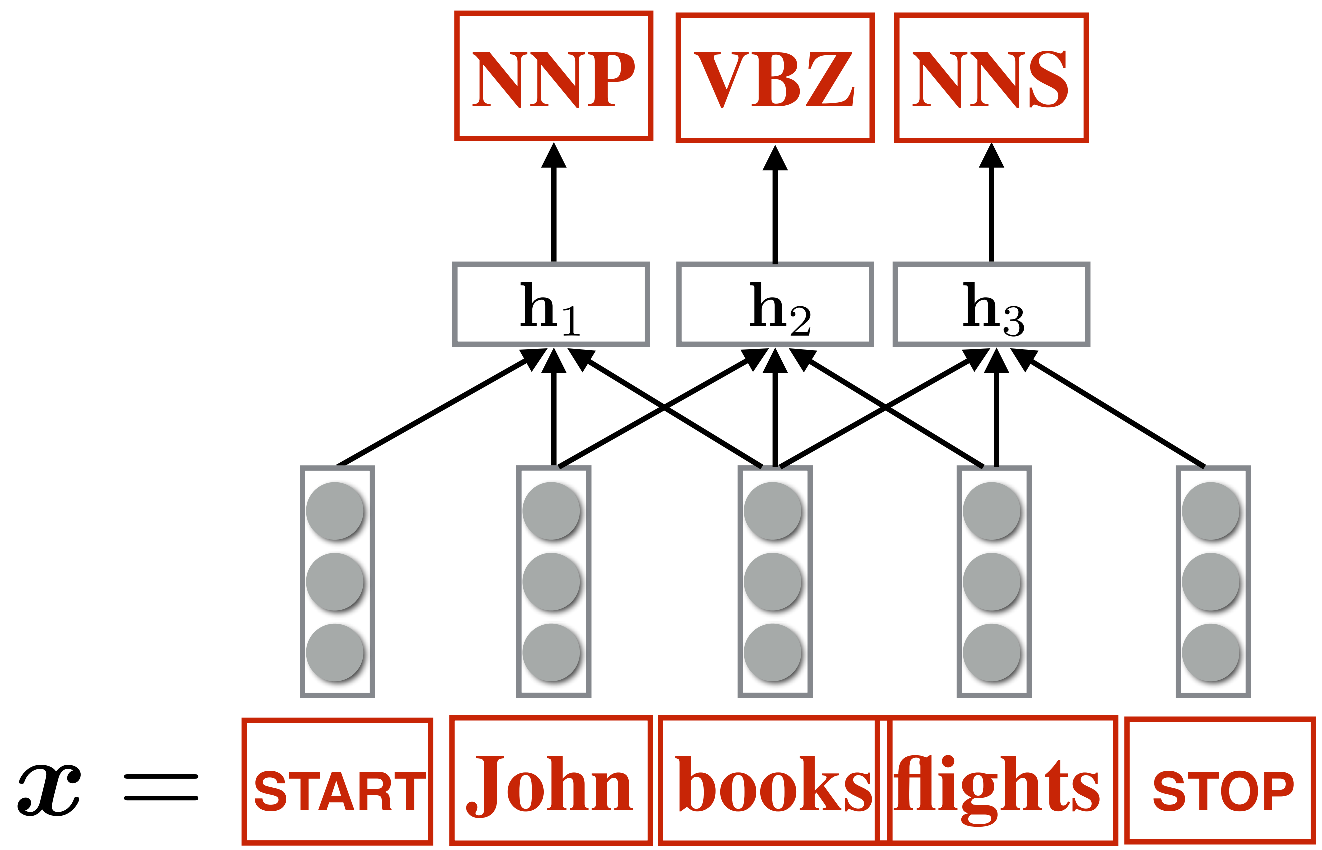


$$\mathbf{h}_t = \tanh\left(\mathbf{C}_{-1}\mathbf{x}_{t-1} + \mathbf{C}_0\mathbf{x}_t + \mathbf{C}_{+1}\mathbf{x}_{t+1} + \mathbf{b}\right)$$

$$p(y_i = y \mid \boldsymbol{x}, i) = \frac{\exp \mathbf{w}_y^\top \mathbf{h}_i + a_y}{\sum_{y' \in Y} \exp \mathbf{w}_y^\top \mathbf{h}_i + a_y}$$

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{i=1}^{|\boldsymbol{y}|} p(y_i \mid \boldsymbol{x}, i)$$

A simple tagging model



$$\mathbf{h}_t = \tanh\left(\mathbf{C}_{-1}\mathbf{x}_{t-1} + \mathbf{C}_0\mathbf{x}_t + \mathbf{C}_{+1}\mathbf{x}_{t+1} + \mathbf{b}\right)$$

$$p(y_i = y \mid \boldsymbol{x}, i) = \frac{\exp \mathbf{w}_y^\top \mathbf{h}_i + a_y}{\sum_{y' \in Y} \exp \mathbf{w}_y^\top \mathbf{h}_i + a_y}$$

$$p(\boldsymbol{y} \mid \boldsymbol{x}) = \prod_{i=1}^{|\boldsymbol{y}|} p(y_i \mid \boldsymbol{x}, i)$$

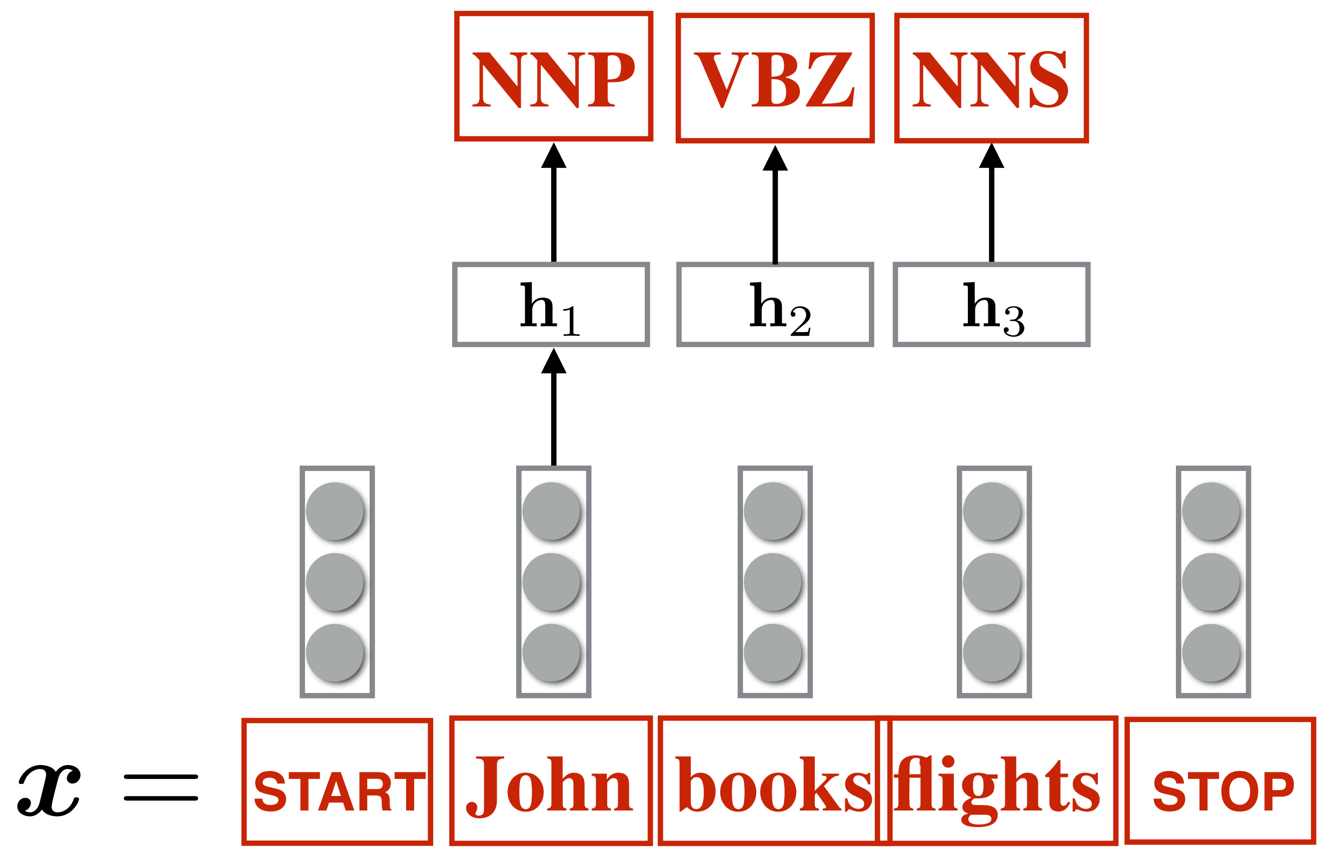
Simple Tagging Model

- Parameters: word embeddings, $\mathbf{C}_{-1}$, $\mathbf{C}_0$, $\mathbf{C}_2$, $\mathbf{W}$, $\mathbf{b}$
- Upsides of this model:
  - Decisions are independent- likelihood/decoding is cheap (no Viterbi!)
- Downsides of this model
  - Say that (A B) and (B A) are both good taggings according to the observations, but (A A) and (B B) are bad taggings. Independence hurts us!
  - Limited context window

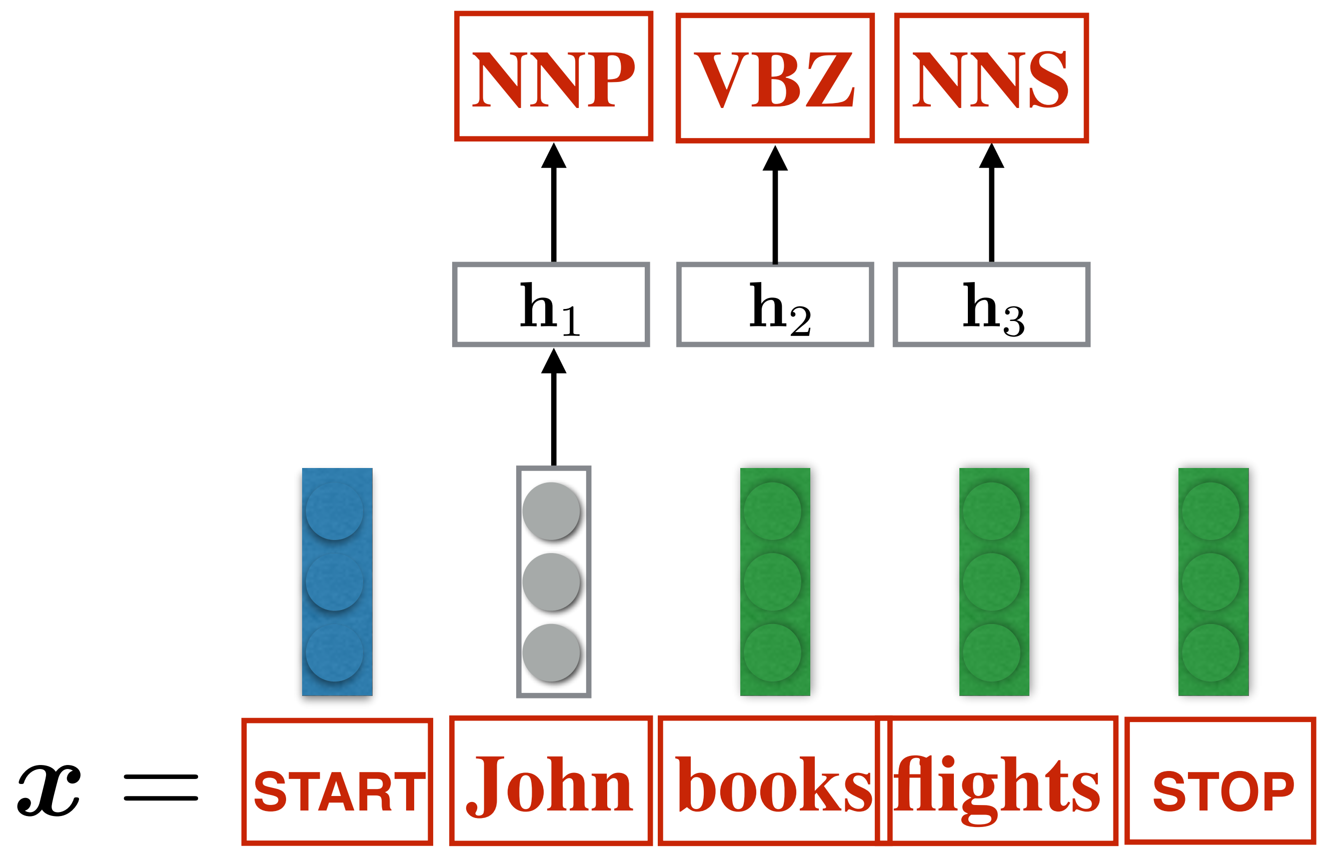
Simple Tagging Model

- In short: this is not a structured model, it is a bunch of independent classification decisions
- But, neural networks are really powerful learners … maybe we don't really need as much structure?
- How can we address the finite horizon problem?

Revised Tagging Model



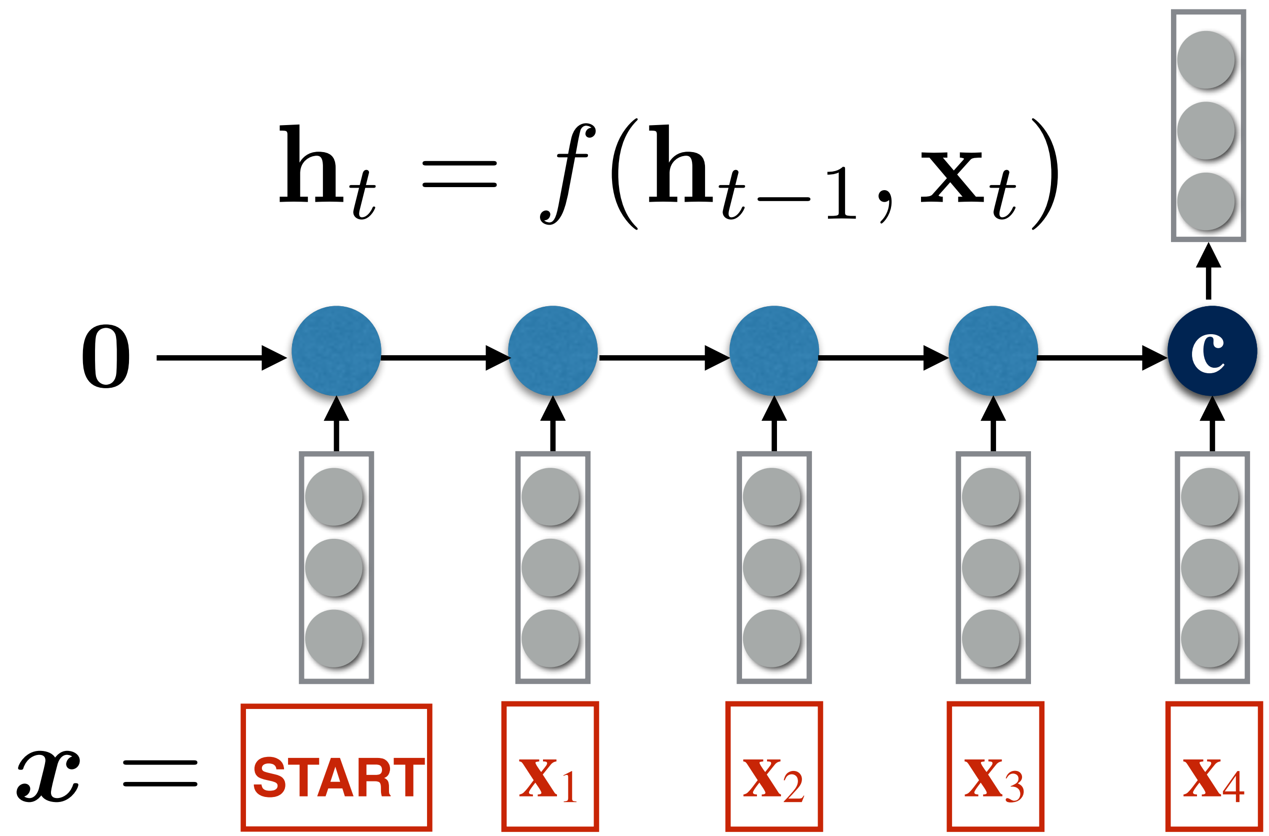
Revised Tagging Model



Recurrent Neural Networks (RNNs)

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)$$

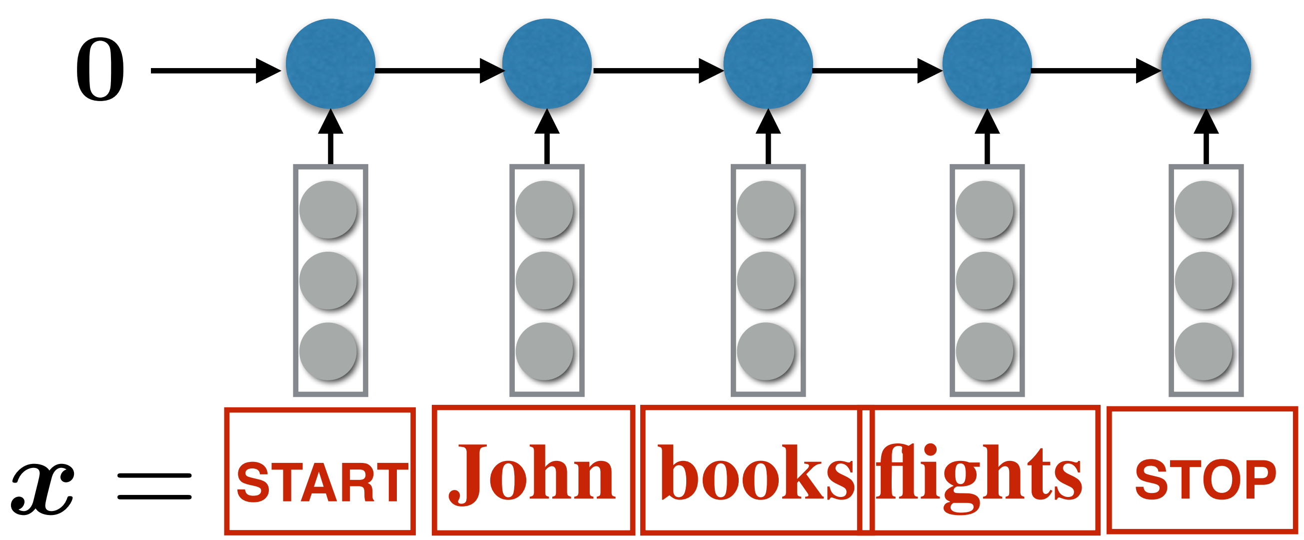
$$\mathbf{c} = \mathrm{RNN}(\boldsymbol{x})$$



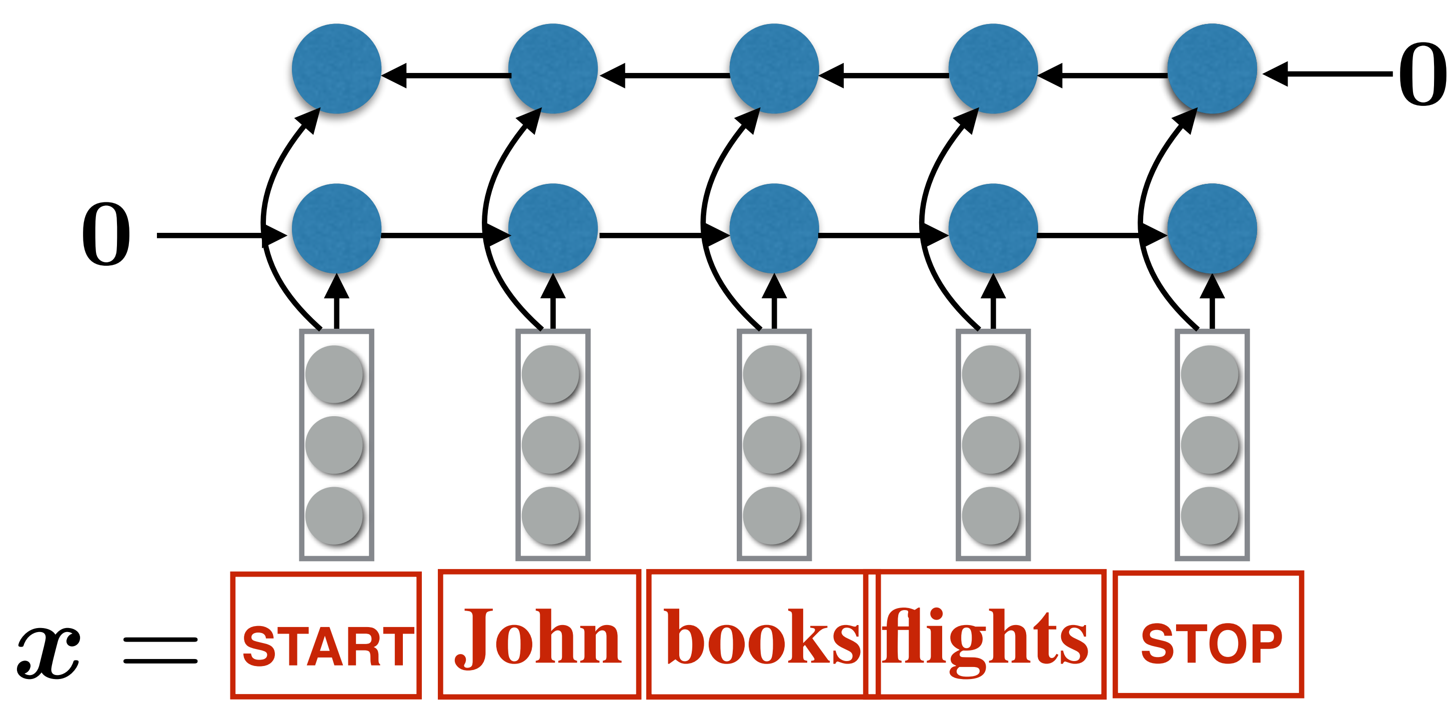
What is a vector representation of a sequence $\boldsymbol{x}$?

Note: numerous definitions exist for f.

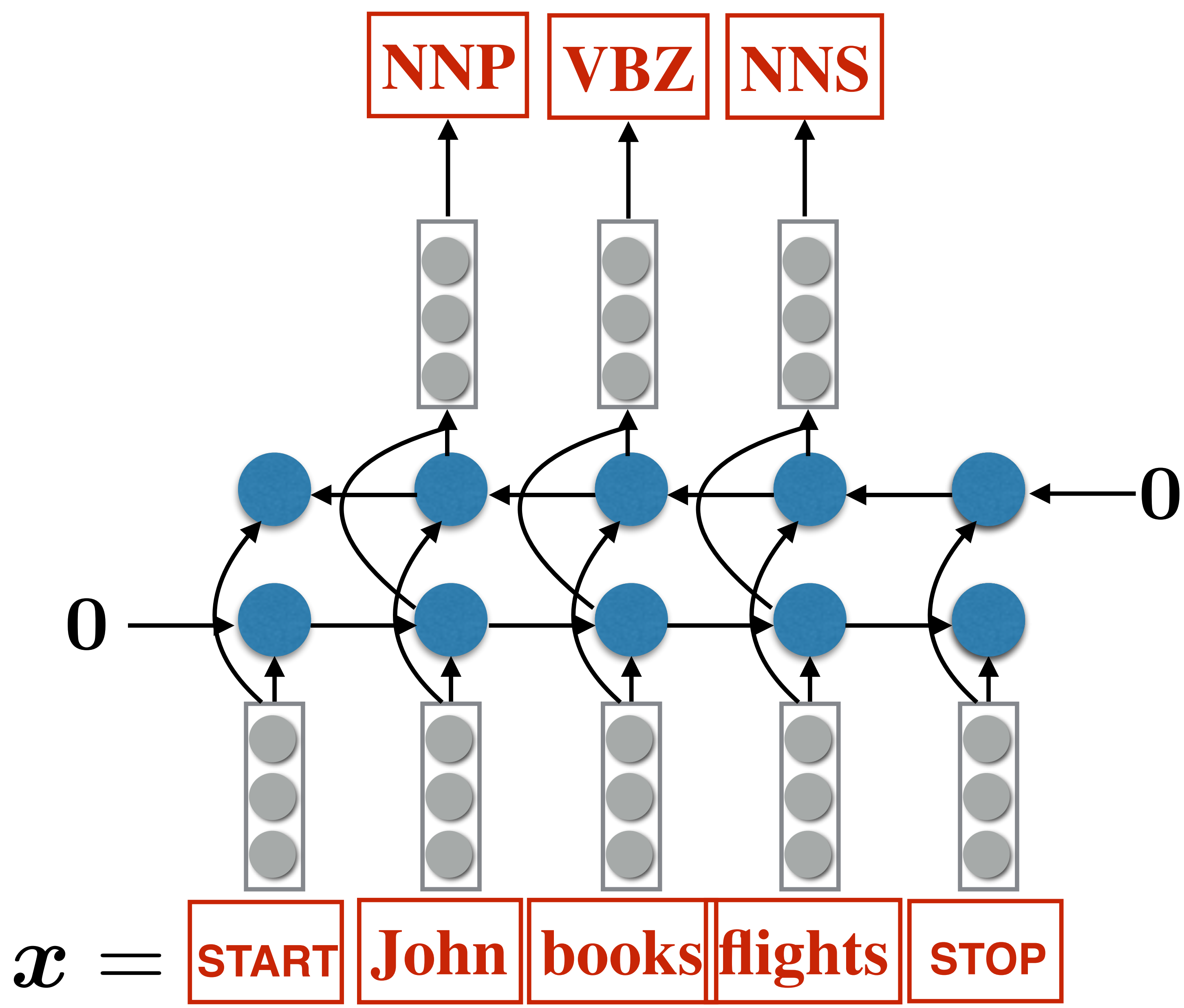
Bidirectional RNN Tagging Model



Bidirectional RNN Tagging Model



Bidirectional RNN Tagging Model



BiRNN Tagging Model

- In short: this is **still not** a structured model, it is a bunch of **independent** classification decisions
- But, neural networks are really powerful learners … maybe we don't really need as much structure?
- This POS tagger is currently state-of-the-art!
  - Maybe POS tagging doesn't need that much structure…

Next time…

- Recurrent models for adding statistical dependencies among output variables