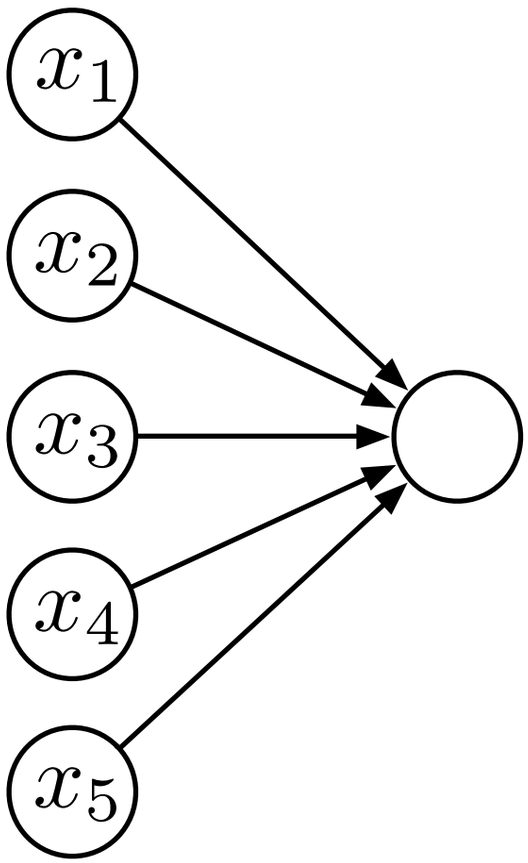
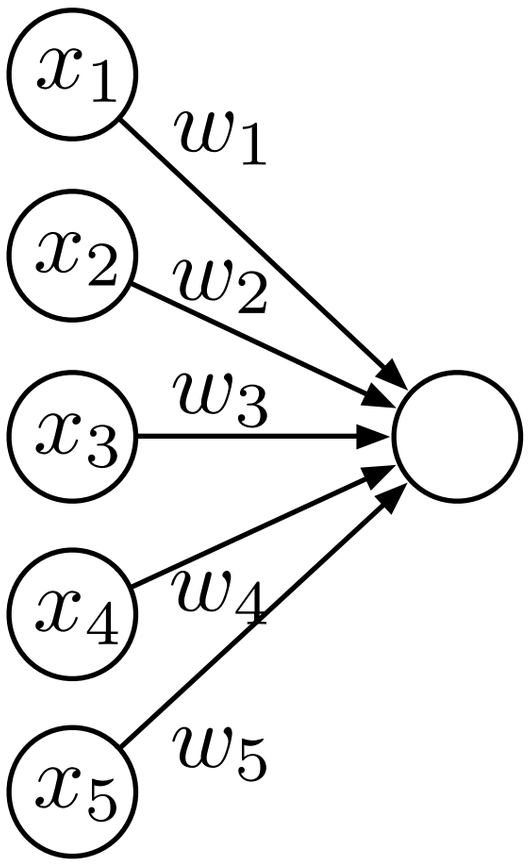
Neural Models in MT

April 24, 2014

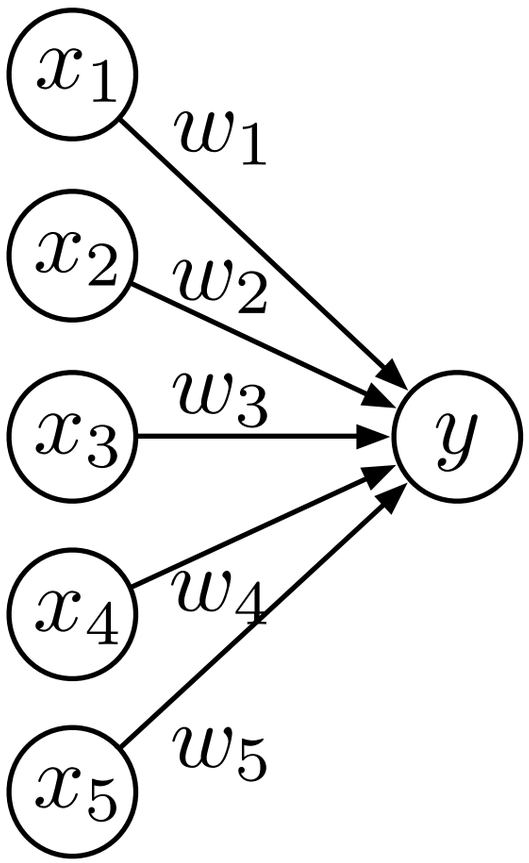
“Neurons”



“Neurons”

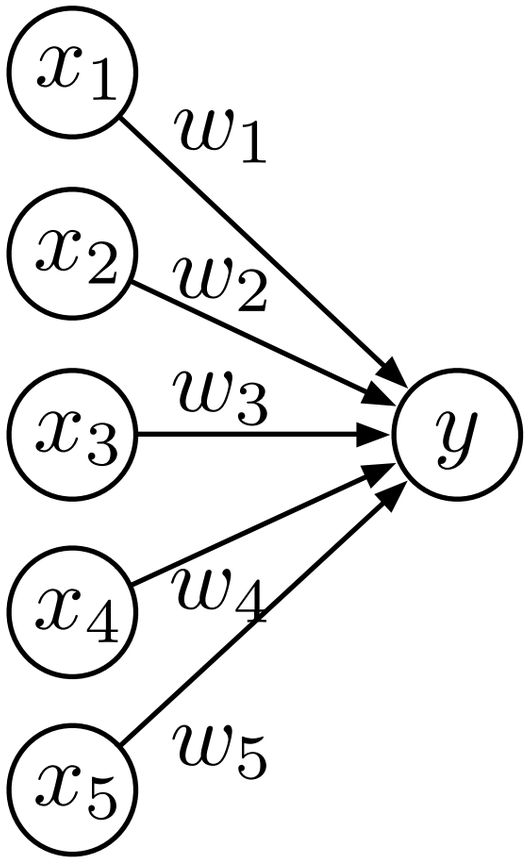


“Neurons”



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

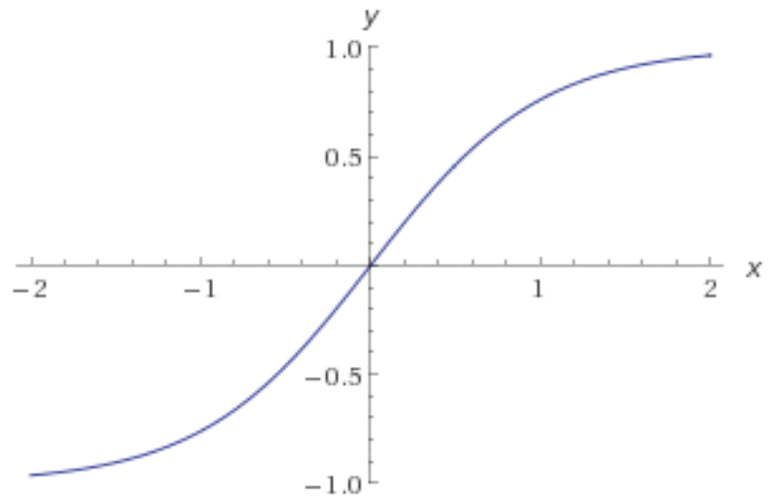
“Neurons”

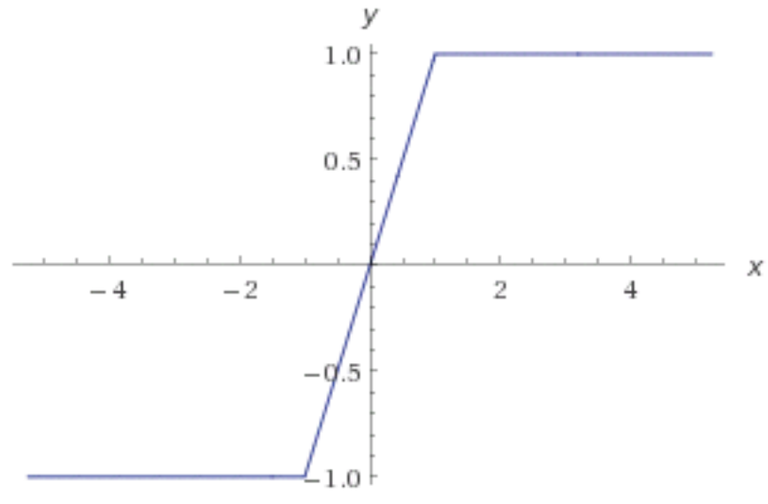


$\boldsymbol{x} \quad \boldsymbol{w}$

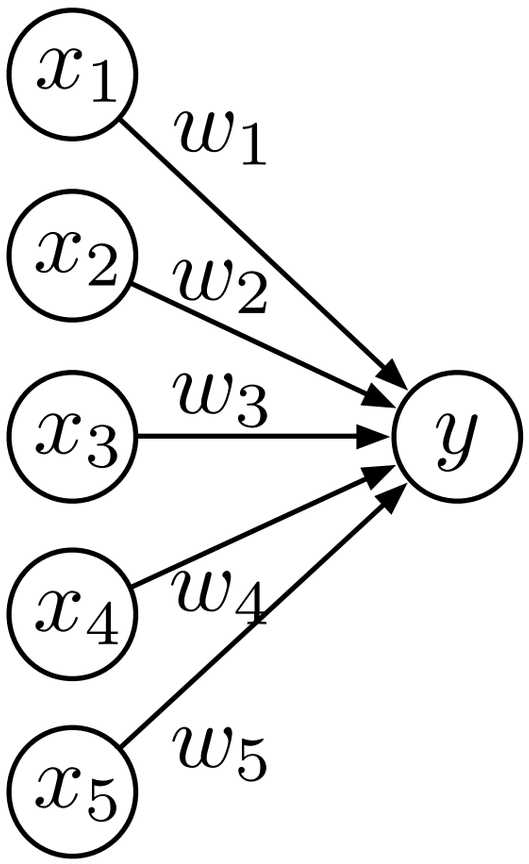
~~$y = \boldsymbol{x} \cdot \boldsymbol{w}$~~

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

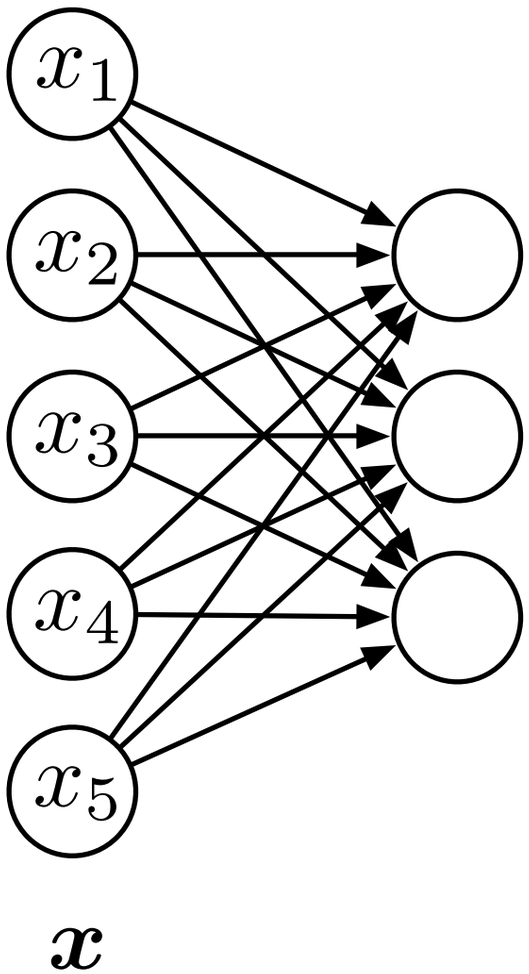




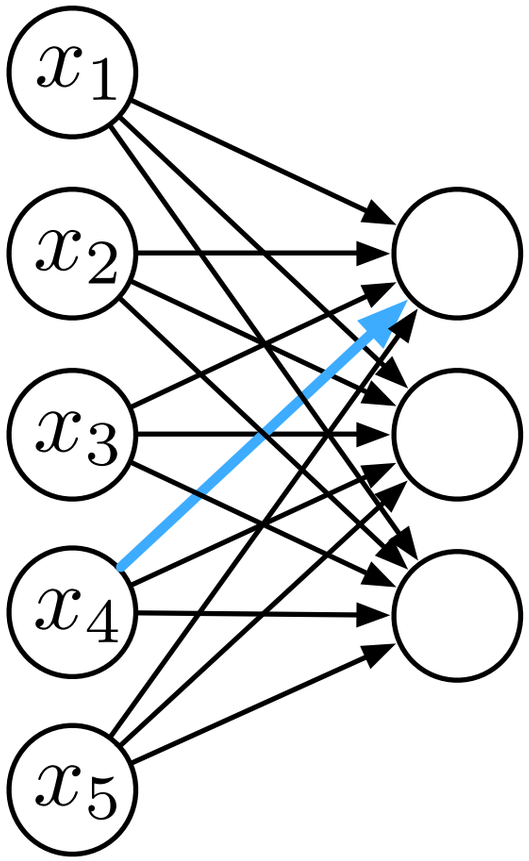
“Neurons”



“Neural” Networks

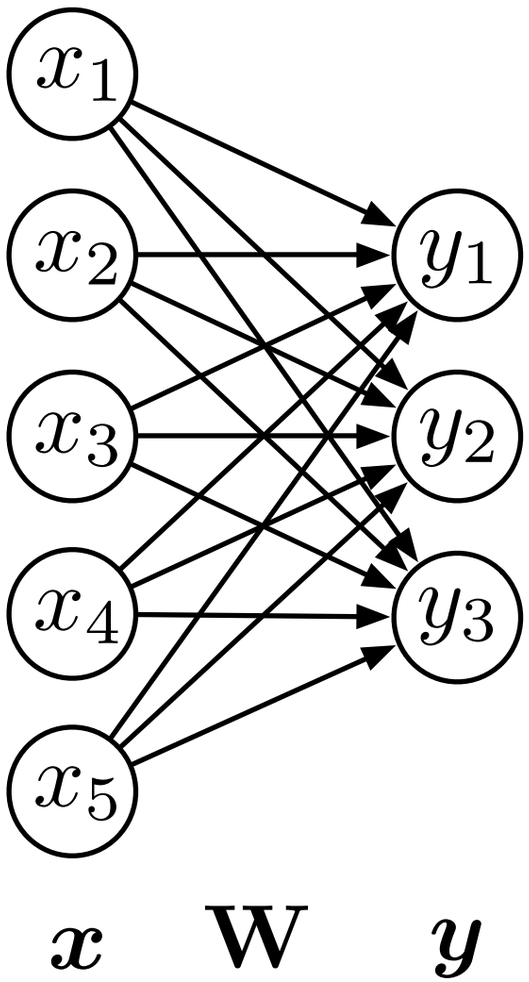


“Neural” Networks



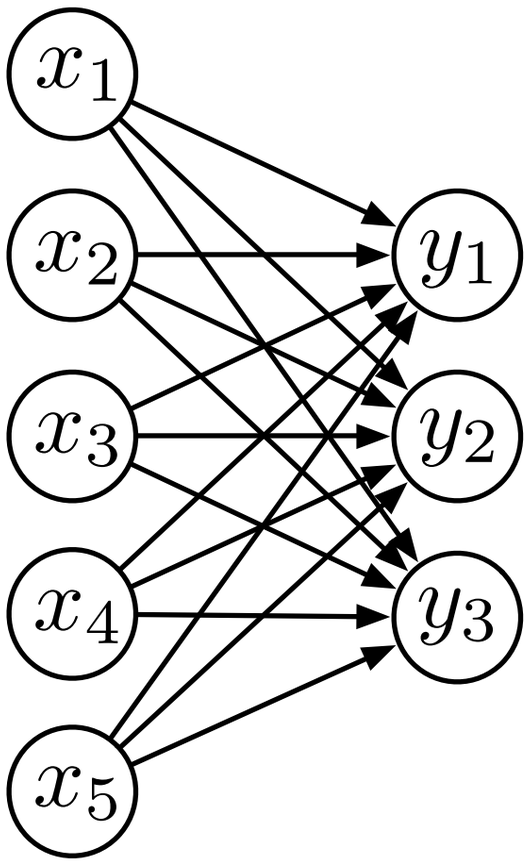
$\boldsymbol{x}$ $w_{4,1}$

“Neural” Networks



$$\boldsymbol{y} = \boldsymbol{x}^\top \mathbf{W}$$

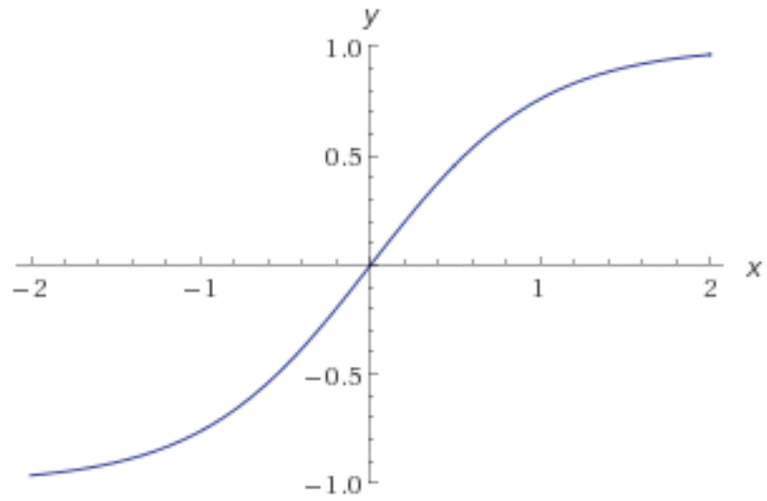
"Neural" Networks

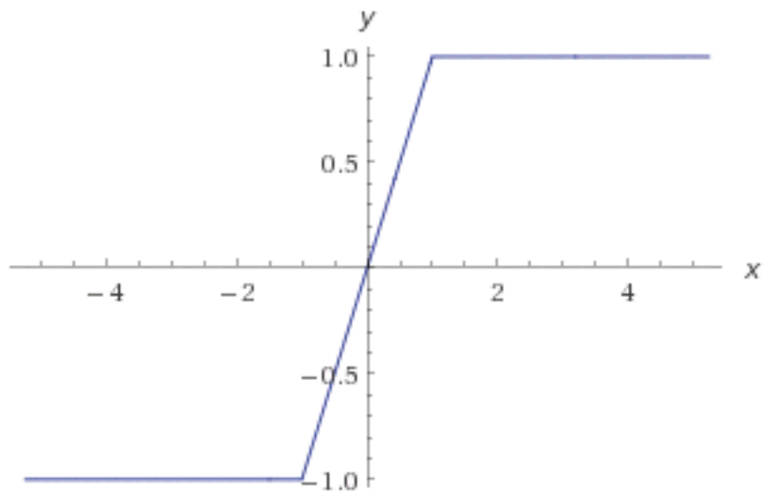


$\boldsymbol{x}$ $\mathbf{W}$ $\boldsymbol{y}$

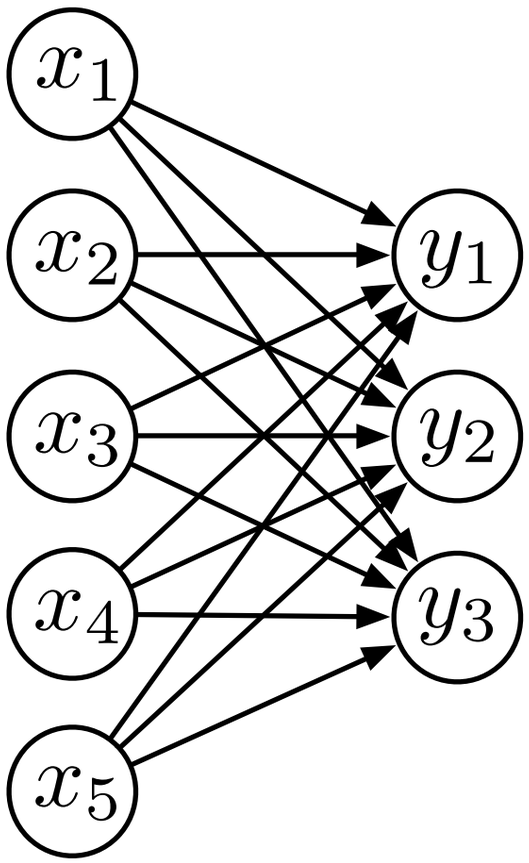
~~$\boldsymbol{y} = \boldsymbol{x}^\top \mathbf{W}$~~

$\boldsymbol{y} = g(\boldsymbol{x}^\top \mathbf{W})$





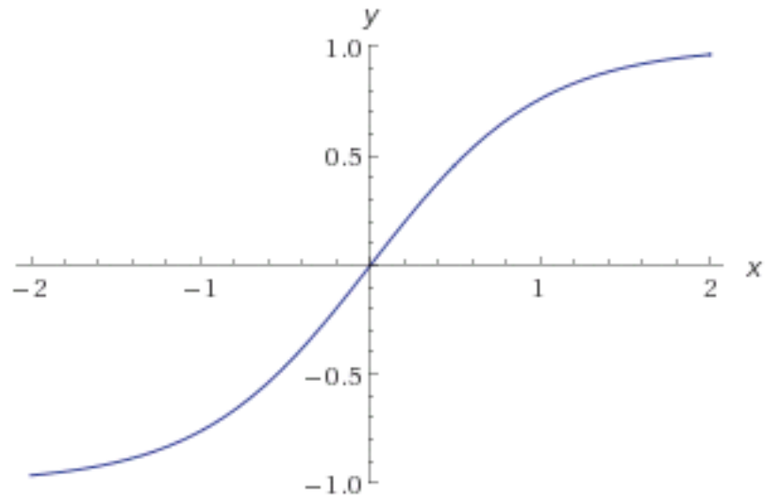
“Neural” Networks

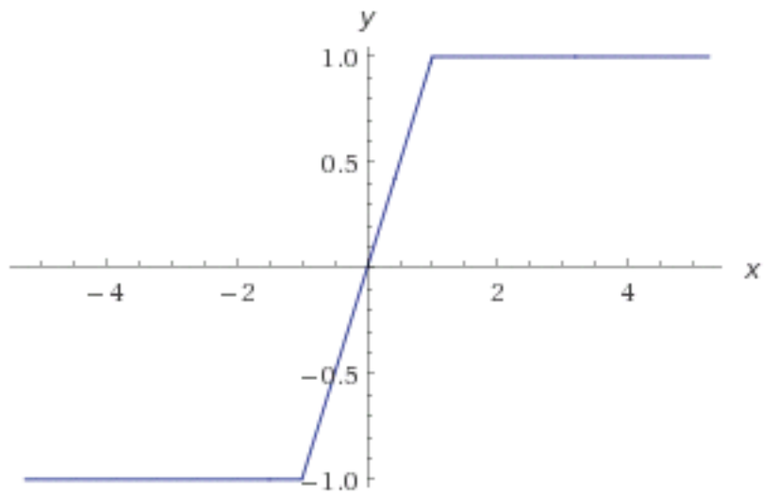


$\boldsymbol{x}$ $\mathbf{W}$ $\boldsymbol{y}$

~~$\boldsymbol{y} = \boldsymbol{x}^\top \mathbf{W}$~~

$\boldsymbol{y} = g(\boldsymbol{x}^\top \mathbf{W})$

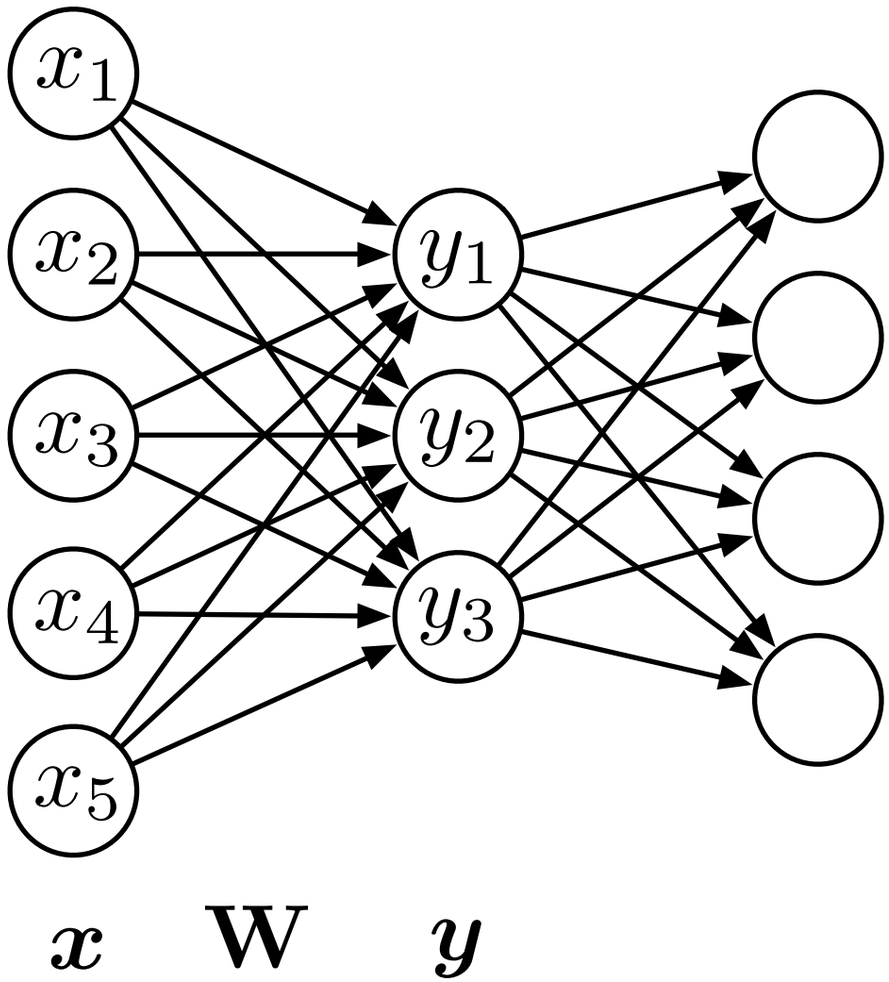




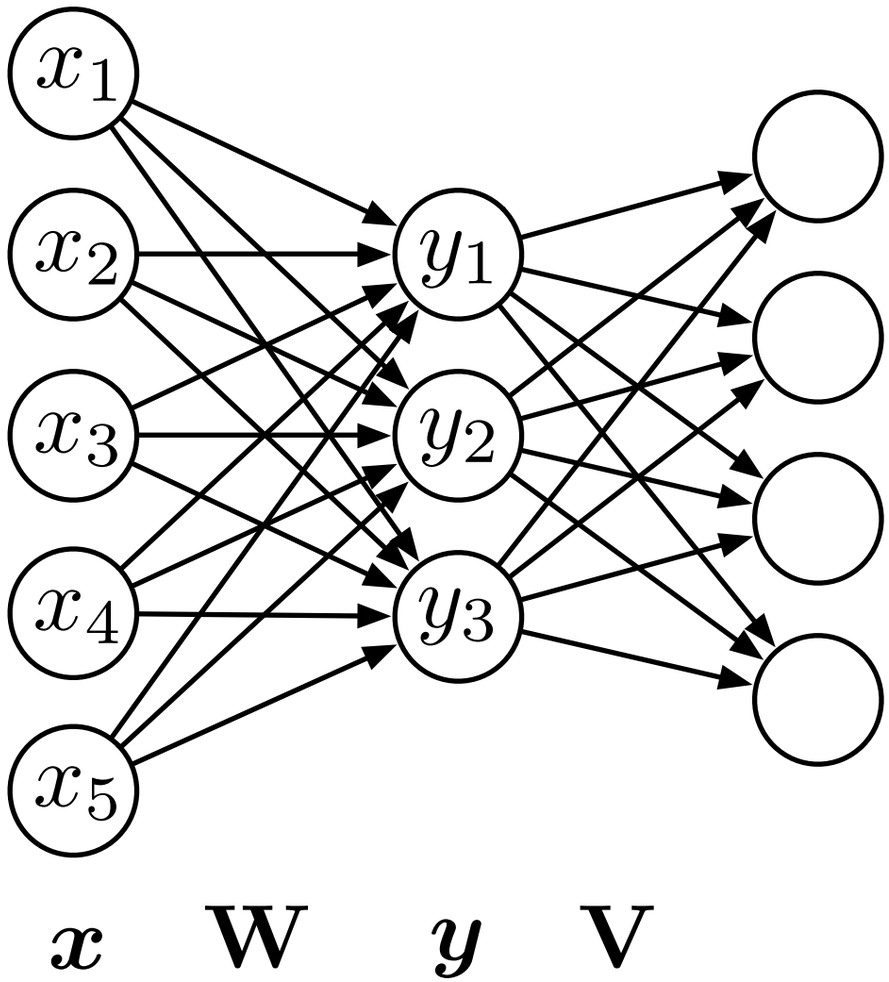
“Soft max”

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

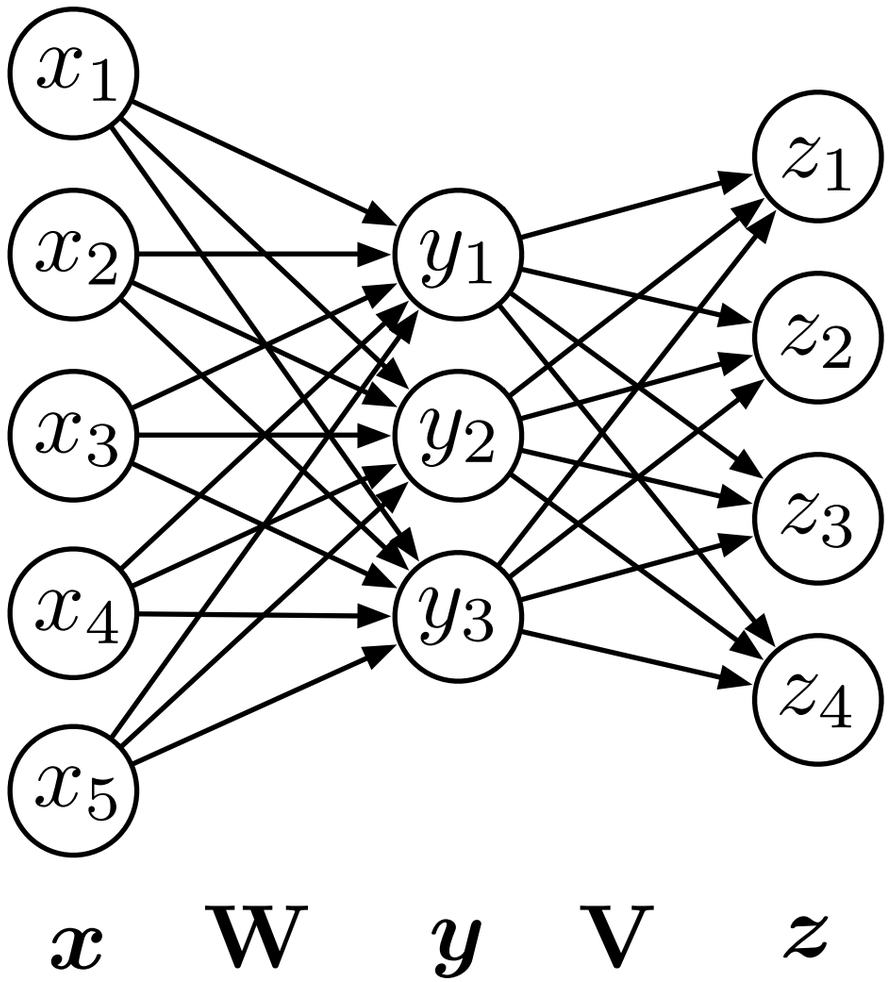
“Deep”



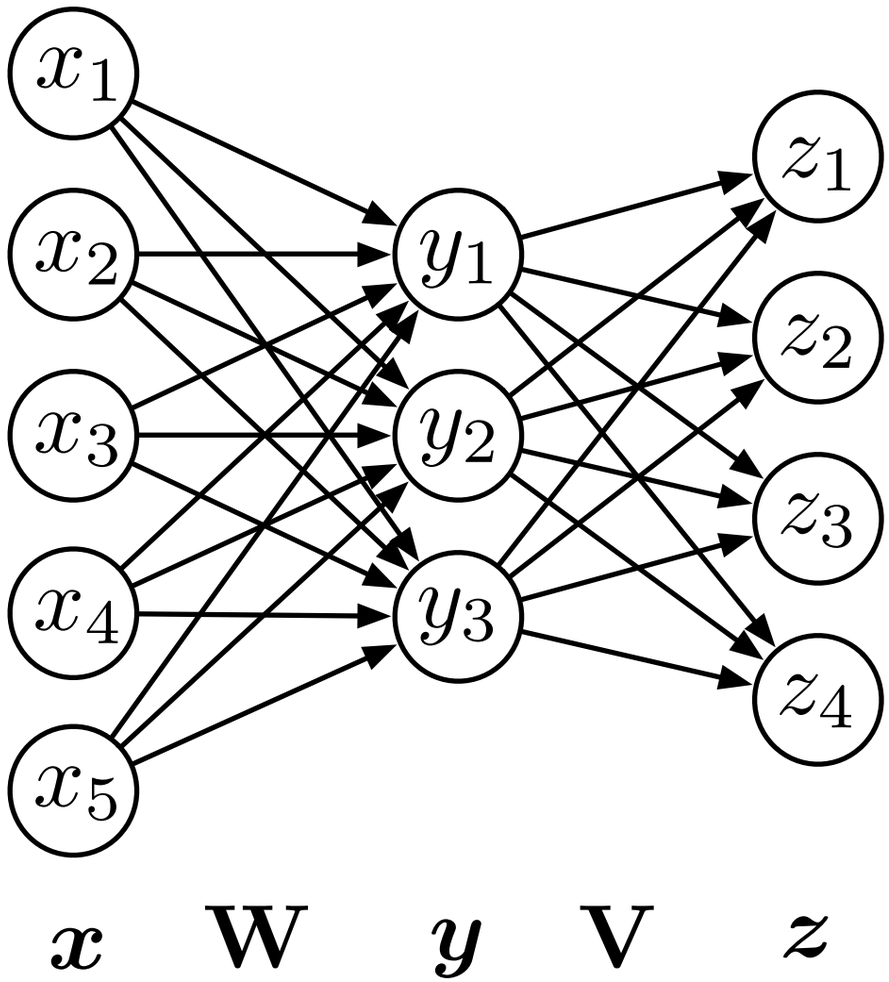
“Deep”


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”

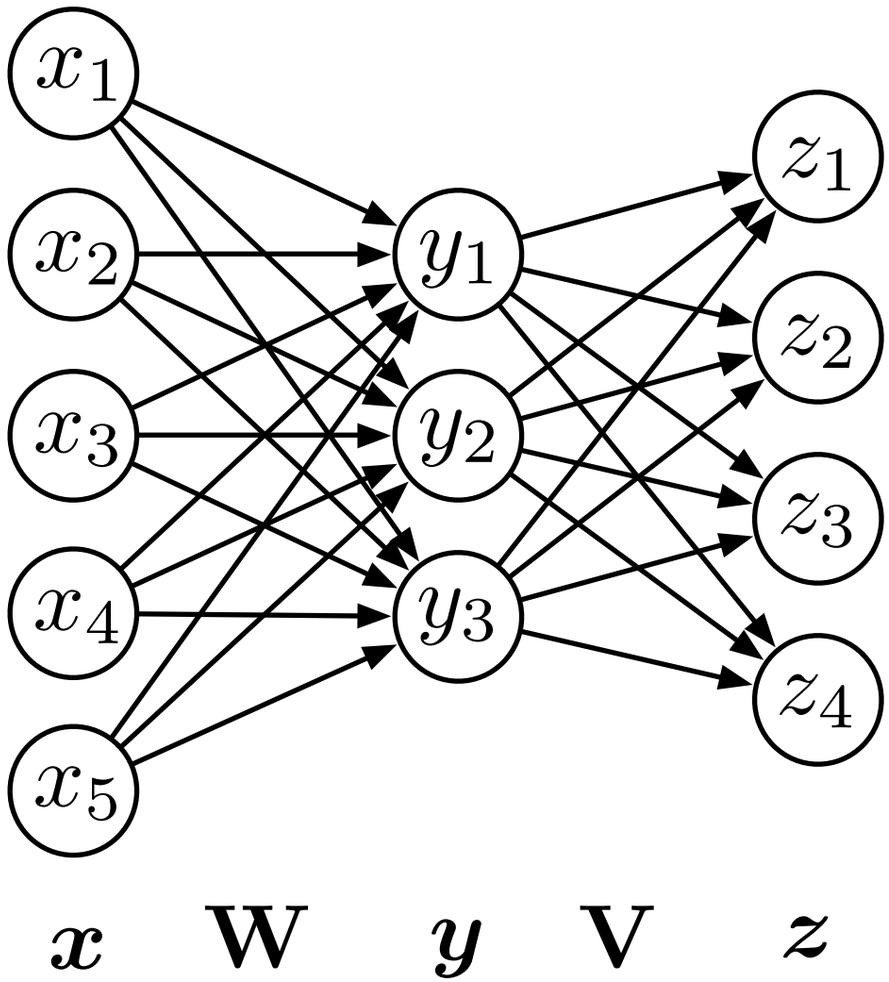

x_1
x_2
x_3
x_4
x_5
y_1
y_2
y_3
z_1
z_2
z_3
z_4
$\boldsymbol{x}$
$\mathbf{W}$
$\boldsymbol{y}$
$\mathbf{V}$
$\boldsymbol{z}$

“Deep”



$$\boldsymbol{z} = g(\boldsymbol{y}^\top \mathbf{V})$$

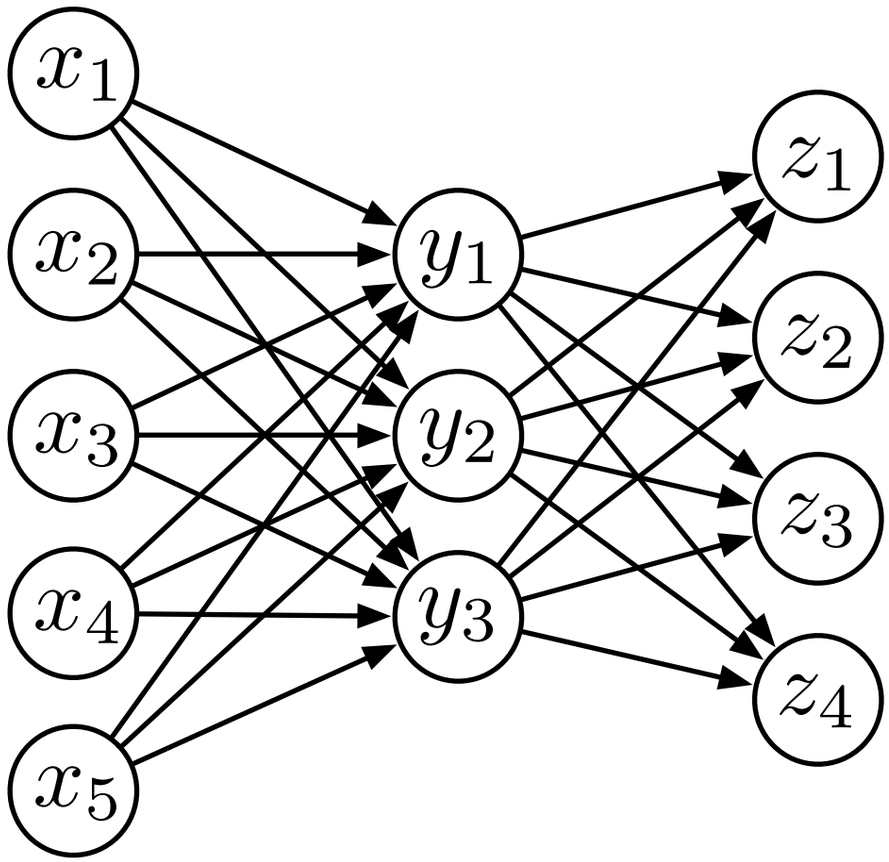
“Deep”



$$\boldsymbol{z} = g(\boldsymbol{y}^\top \mathbf{V})$$

$$\boldsymbol{z} = g(h(\boldsymbol{x}^\top \mathbf{W})^\top \mathbf{V})$$

“Deep”

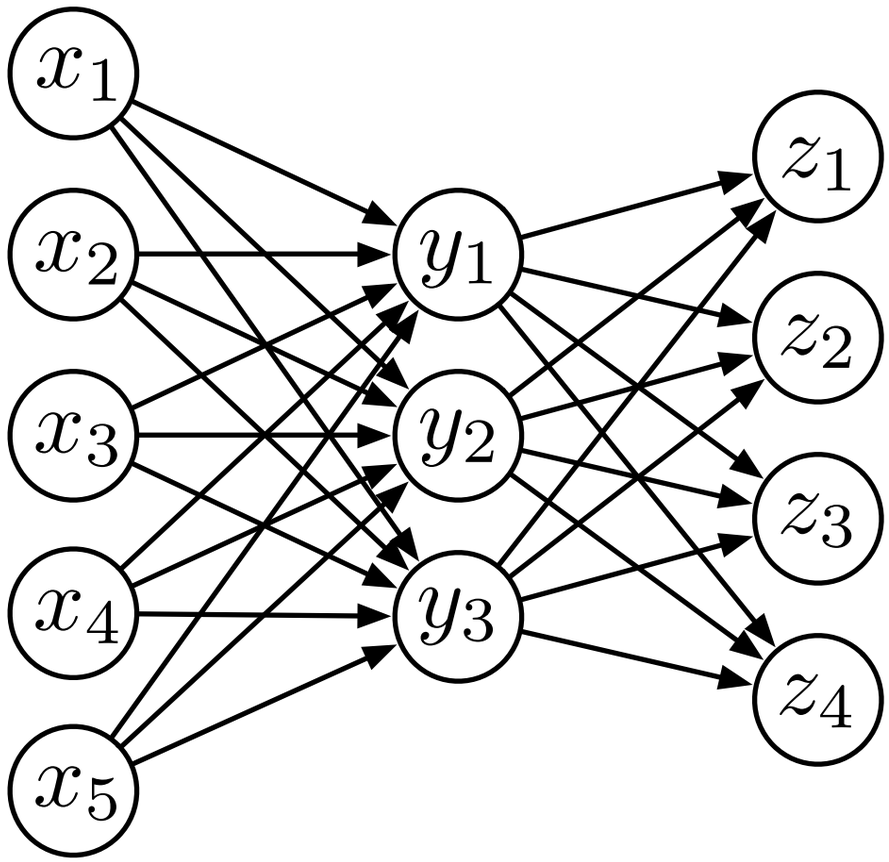


$$\boldsymbol{z} = g(\boldsymbol{y}^\top \mathbf{V})$$

$$\boldsymbol{z} = g(h(\boldsymbol{x}^\top \mathbf{W})^\top \mathbf{V})$$

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

"Deep"



$\boldsymbol{x}$ $\mathbf{W}$ $\boldsymbol{y}$ $\mathbf{V}$ $\boldsymbol{z}$

$$\boldsymbol{z} = g(\boldsymbol{y}^\top \mathbf{V})$$

$$\boldsymbol{z} = g(h(\boldsymbol{x}^\top \mathbf{W})^\top \mathbf{V})$$

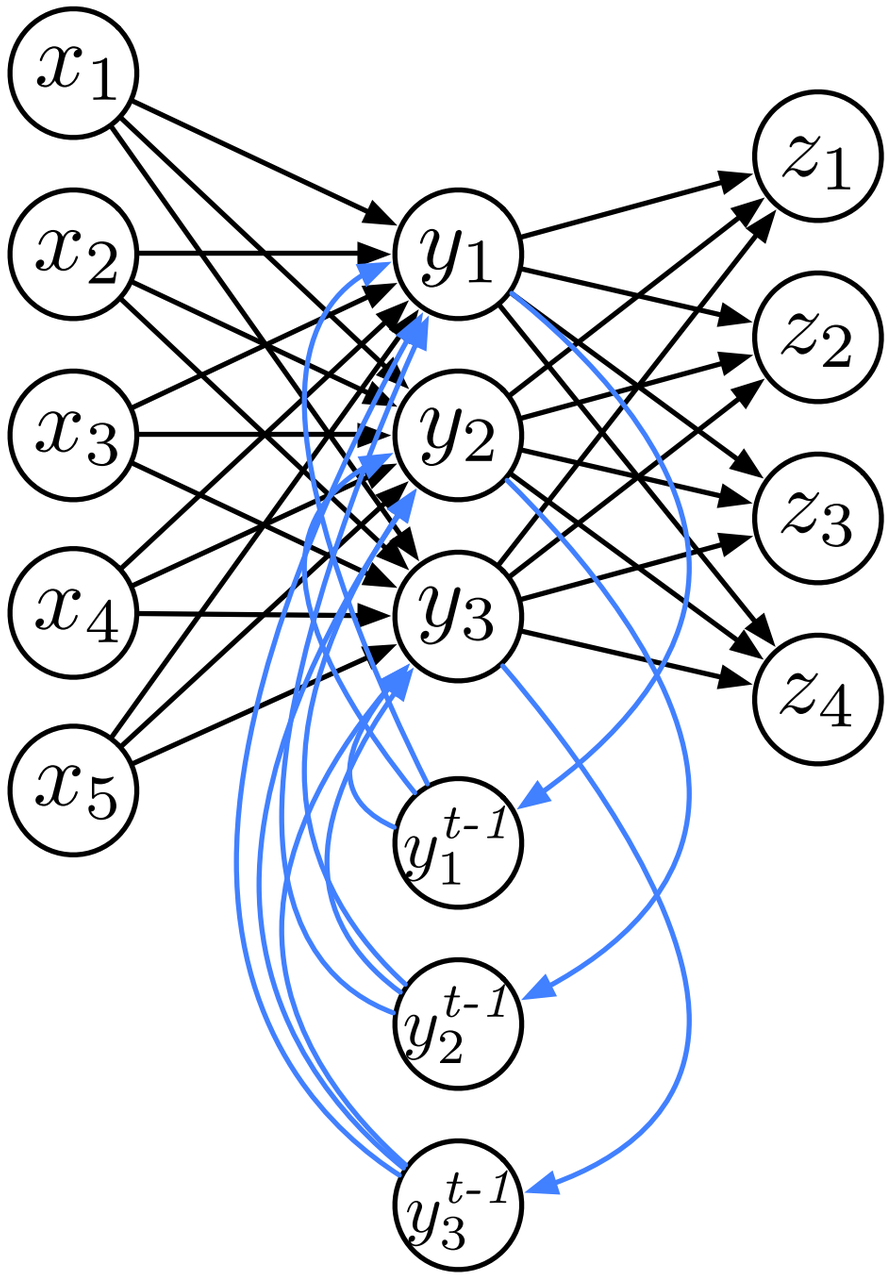
$$\boldsymbol{z} = g(\mathbf{V} h(\mathbf{W} \boldsymbol{x}))$$

Note:

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

$$\boldsymbol{z} = \underbrace{(\mathbf{V}\mathbf{W})}_{\mathbf{U}} \boldsymbol{x}$$

“Recurrent”



Design Decisions

- How to represent inputs and outputs?
- Neural architecture?
  - How many layers? (Requires non-linearities to improve capacity!)
  - How many neurons?
  - Recurrent or not?
- What kind of non-linearities?

Representing Language

- "One-hot" vectors
  - Each position in a vector corresponds to a word type
  - Sequence of words, sequence of vectors
  - Bag of words: multiple vectors
- Distributed representations
  - Vectors encode "features" of input words (character n-grams, morphological features, etc.)

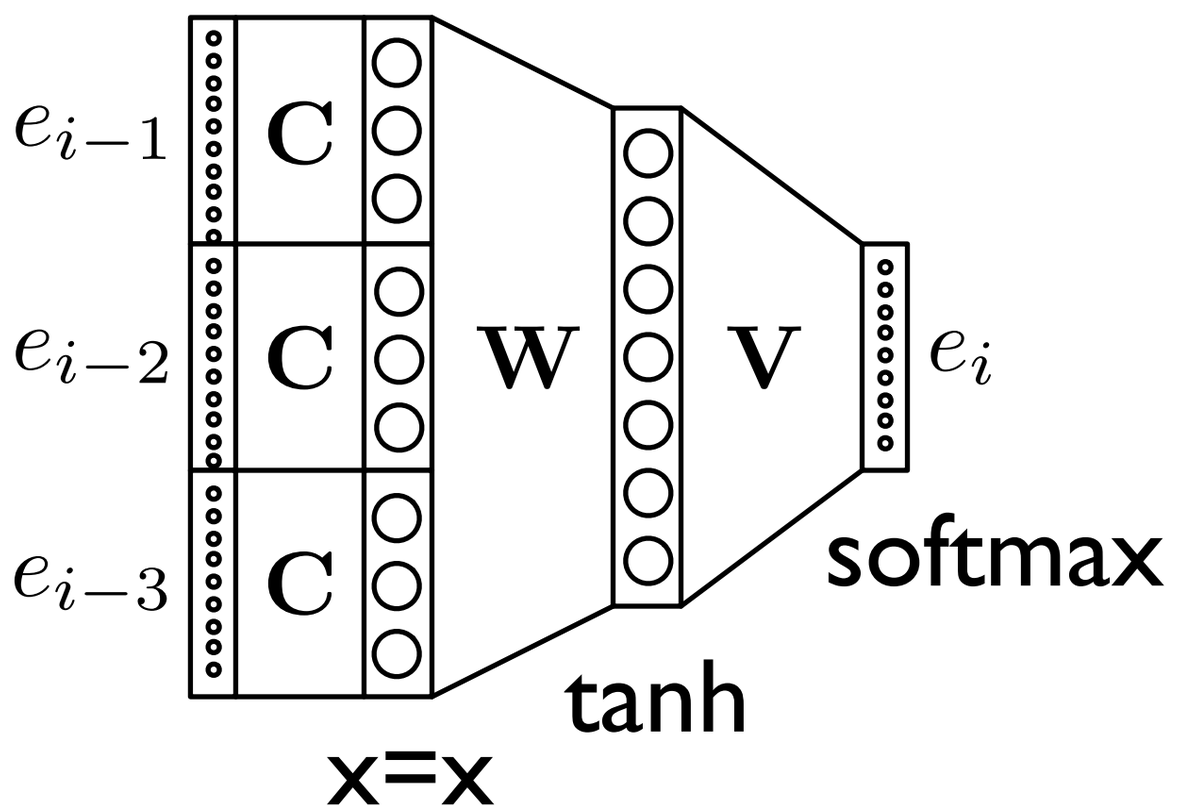
Training Neural Networks

- Neural networks are supervised models - you need a set of inputs paired with outputs
- Algorithm
  - Run for a while
    - Give input to the network, see what it predicts
    - Compute loss(y,y*) and (sub)gradient with respect to parameters. Use the chain rule, aka “back propagation”
    - Update parameters (SGD, AdaGrad, LBFGS, etc.)

Bengio et al. (2003)

$$p(\mathbf{e}) = \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-n+1}, \ldots, e_{i-1})$$

$$p(e_i \mid e_{i-n+1}, \ldots, e_{i-1}) =$$



Bengio et al. (2003)

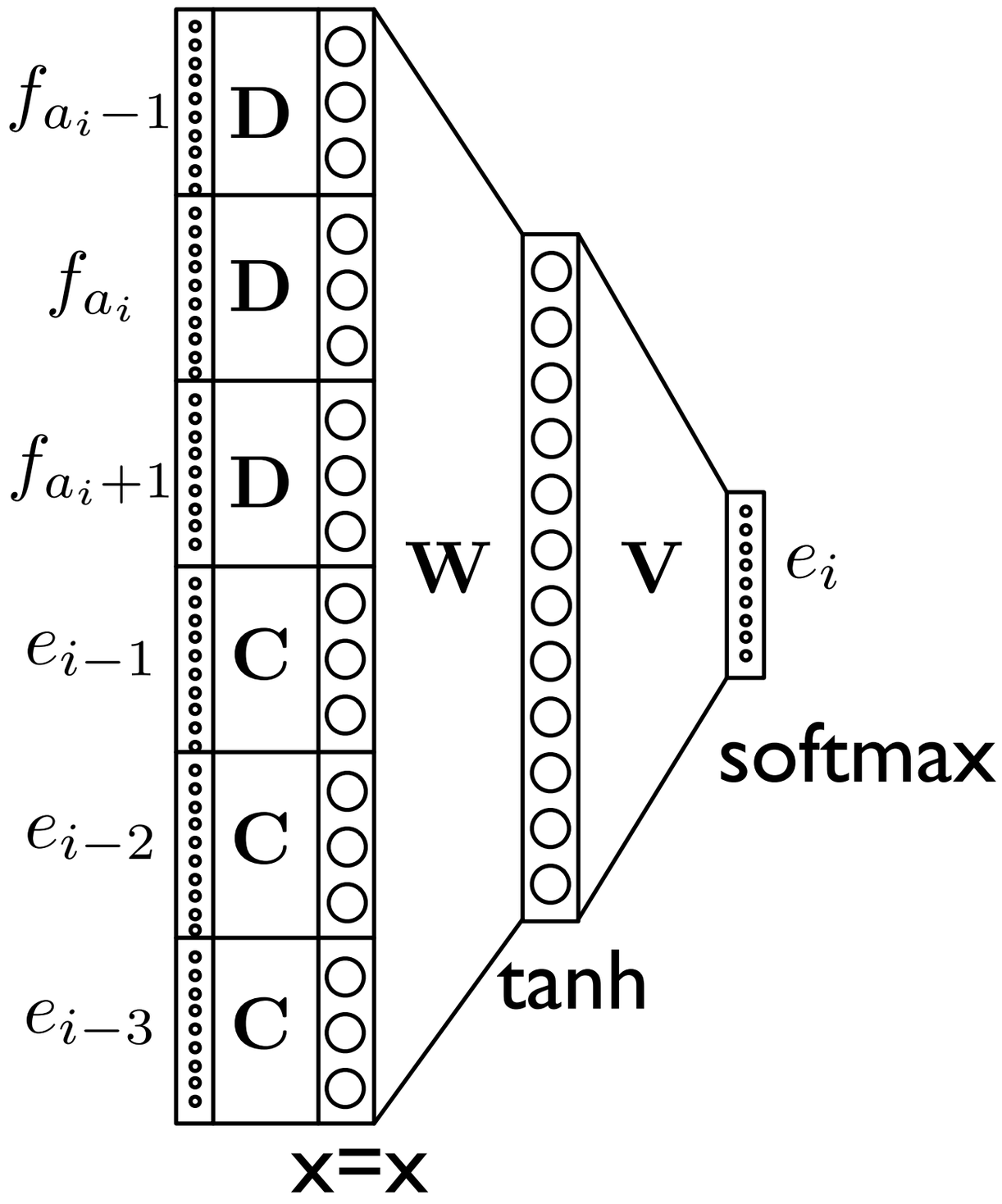
	n	c	h	m	direct	mix	train.	valid.	test.
MLP1	5		50	60	yes	no	182	284	268
MLP2	5		50	60	yes	yes		275	257
MLP3	5		0	60	yes	no	201	327	310
MLP4	5		0	60	yes	yes		286	272
MLP5	5		50	30	yes	no	209	296	279
MLP6	5		50	30	yes	yes		273	259
MLP7	3		50	30	yes	no	210	309	293
MLP8	3		50	30	yes	yes		284	270
MLP9	5		100	30	no	no	175	280	276
MLP10	5		100	30	no	yes		265	**252**
Del. Int.	3						31	352	336
Kneser-Ney back-off	3							334	323
Kneser-Ney back-off	4							332	321
Kneser-Ney back-off	5							332	321

Devlin et al. (2014)

- Turn Bengio et al. (2003) into a translation model
- Conditional model; generate the next English word conditioned on
  - The previous *n* English words you generated
  - The aligned source word, and its *m* neighbors

$$p(\mathbf{e} \mid \mathbf{f}, \mathbf{a}) = \prod_{i=1}^{|\mathbf{e}|} p(e_i \mid e_{i-2}, e_{i-1}, f_{a_i-1}, f_{a_i}, f_{a_i+1})$$

$$p(e_i \mid e_{i-2}, e_{i-1}, f_{a_i-1}, f_{a_i}, f_{a_i+1}) =$$



Devlin et al. (2014)

BOLT Test		
	Ar-En	
	BLEU	**% Gain**
"Simple Hier." Baseline	33.8	-
S2T/L2R NNJM (Dec)	38.4	100%
Source Window=7	38.3	98%
Source Window=5	38.2	96%
Source Window=3	37.8	87%
Source Window=0	35.3	33%
Layers=384x768x768	38.5	102%
Layers=192x512	38.1	93%
Layers=128x128	37.1	72%
Vocab=64,000	38.5	102%
Vocab=16,000	38.1	93%
Vocab=8,000	37.3	83%
Activation=Rectified Lin.	38.5	102%
Activation=Linear	37.3	76%

Kalchbrenner & Blunsom (2013)

- Conditional model
- Represent **f** as a vector (or matrix)
- Use recurrent model + vector representation of **f** to generate translation

Summary

- Two problems in standard statistical models
  - We don't condition on enough stuff
    - We don't know what features to use when we condition on lots of structure
- Punchline: **Neural networks let us condition on a lot of stuff without an exponential growth in parameters**