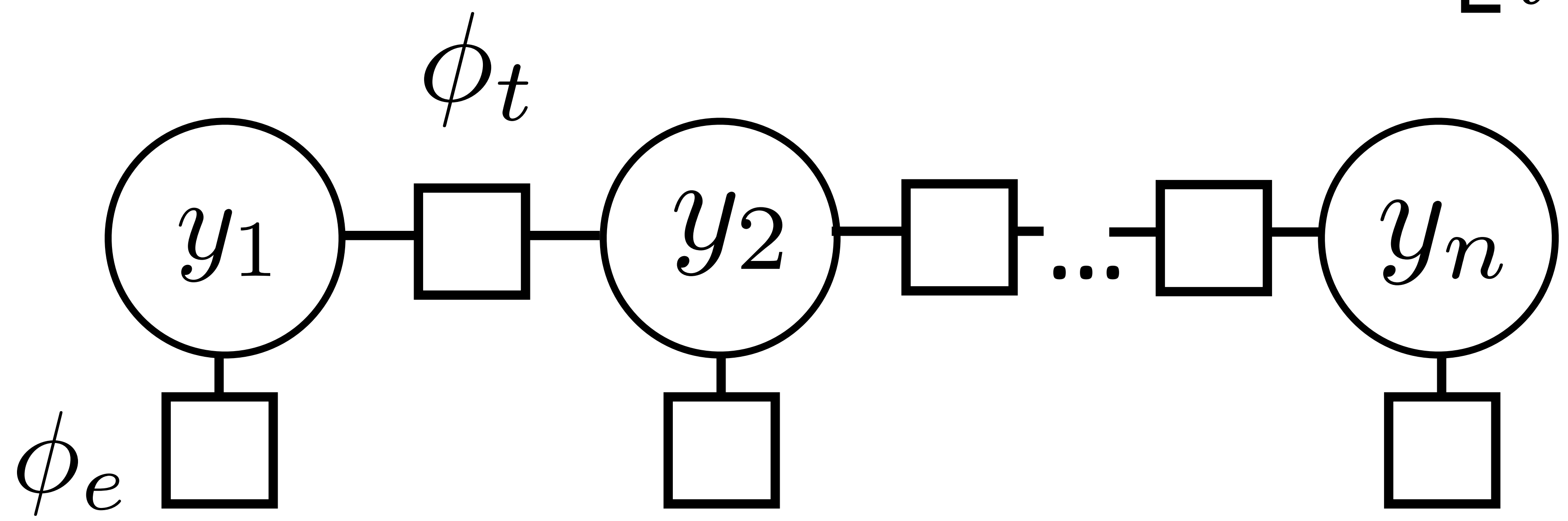
# ANLP

## 11 - Feed Forward NNs (NNs, part I)

David Schlangen
University of Potsdam, MSc Cognitive Systems
Winter 2019 / 2020

# Recall: Sequential CRFs

- Model: $P(\mathbf{y}|\mathbf{x}) \propto \exp w^\top \left[ \sum_{i=2}^{n} f_t(y_{i-1}, y_i) + \sum_{i=1}^{n} f_e(y_i, i, \mathbf{x}) \right]$




- Emission features capture word-level info, transitions enforce tag consistency

- Inference: argmax P(**y**|**x**) from Viterbi

- Learning: run forward-backward to compute posterior probabilities; then

$$\frac{\partial}{\partial w} \mathcal{L}(\mathbf{y}^*, \mathbf{x}) = \sum_{i=1}^{n} f_e(y_i^*, i, \mathbf{x}) - \sum_{i=1}^{n} \sum_{s} P(y_i = s|\mathbf{x}) f_e(s, i, \mathbf{x})$$

# Features for NER

- Word features (can use in HMM)
  - Capitalization
  - Word shape
  - Prefixes/suffixes
  - Lexical indicators
- Context features (can't use in HMM!)
  - Words before/after
  - Tags before/after
- Word clusters
- Gazetteers

*Leicestershire*

*Boston*

*Apple released a new version…*

*According to the New York Times…*

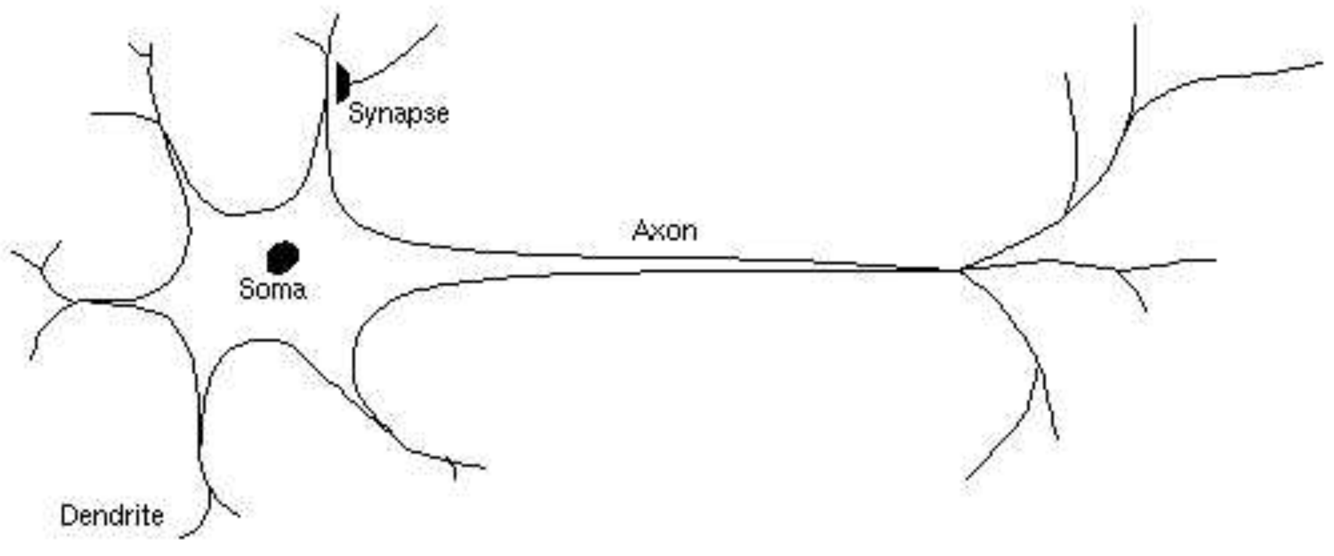
# can we *learn* feature functions?

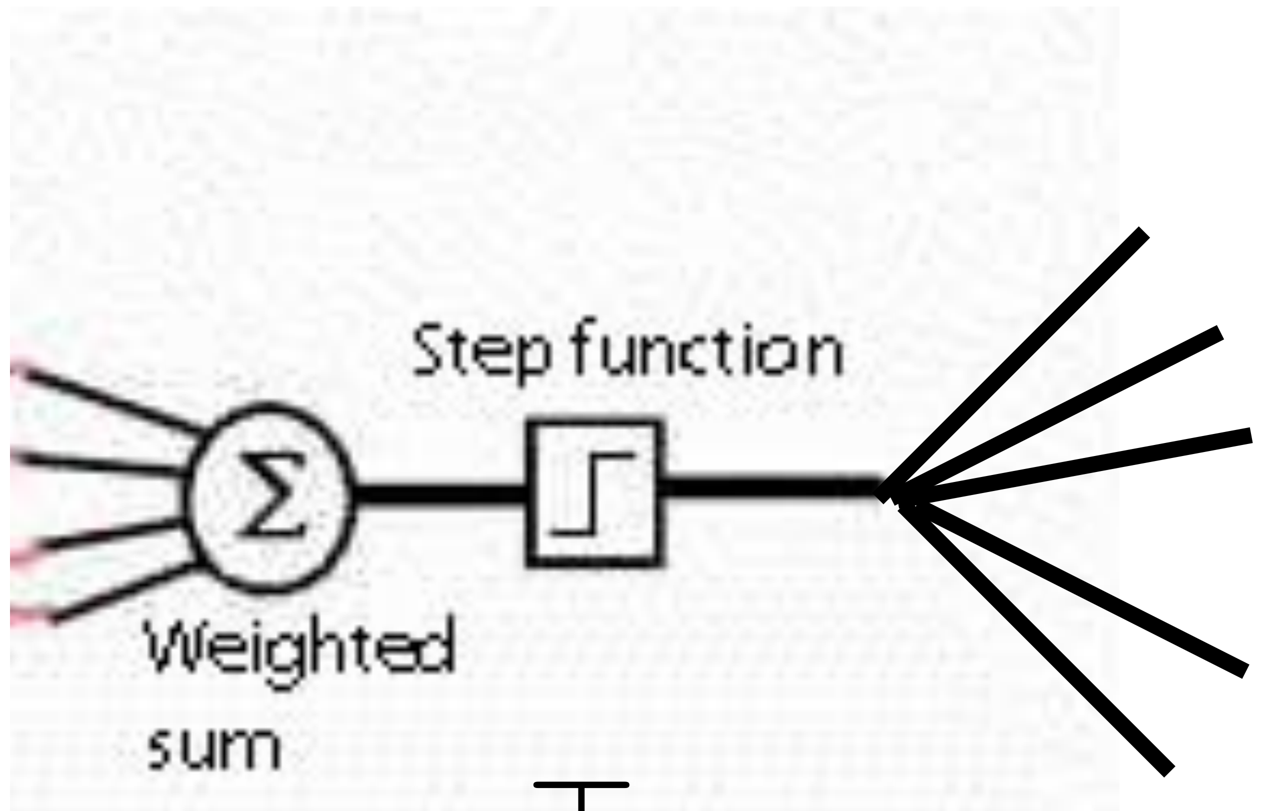
- we have seen that it matters *what* information is presented to learner, and *how* it is represented
- but maybe the learner knows best how it should be represented?
- we have already seen that by designing clever representations, we can make data linearly separable that doesn't look like it is…

# Neural Net History

# the prehistory

- the perceptron, invented by Frank Rosenblatt, 1958
  - explicitly inspired by what was known then about neurons




$$w^\top x > 0$$

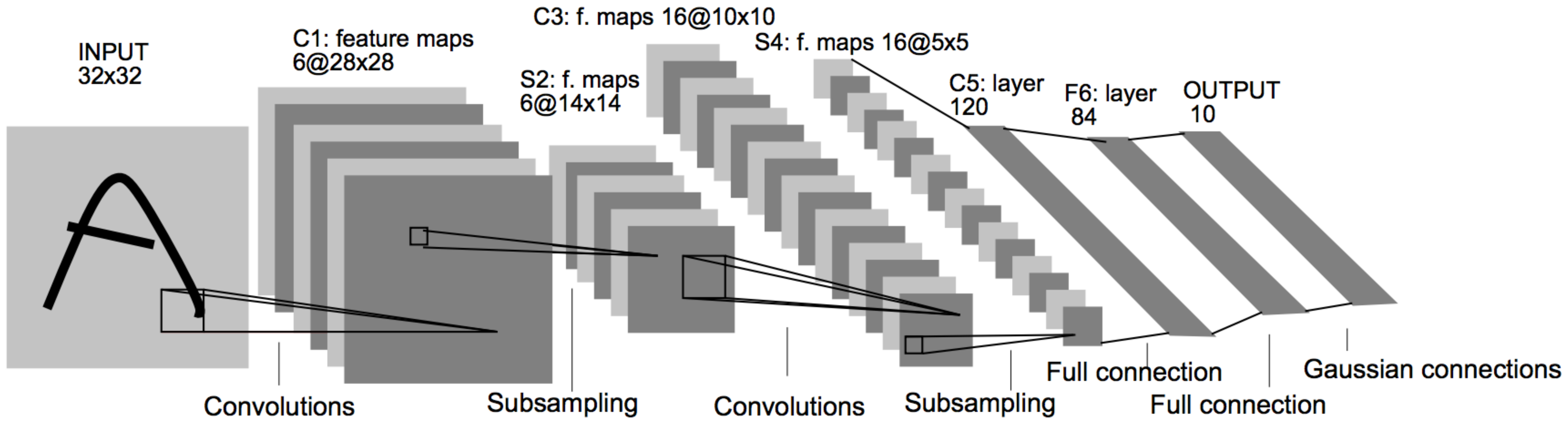
- can’t do XOR, is dead (Minsky & Papert, 1969)

# the history

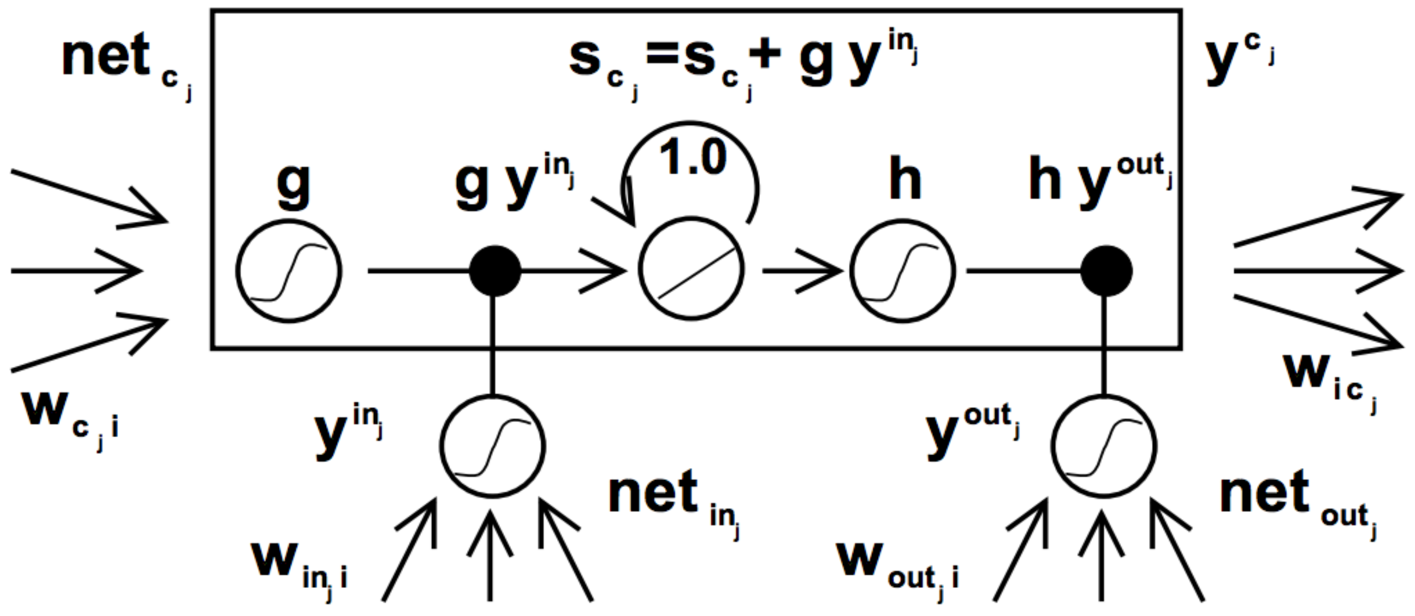
- some work on language processing in the 70s, 80s, 90s
- mostly from a cognitive science perspective (i.e., not large scale processing, but rather small toy examples.. “can it in principle do this?”)

# History: NN "dark ages"

- Convnets: applied to MNIST by LeCun in 1998



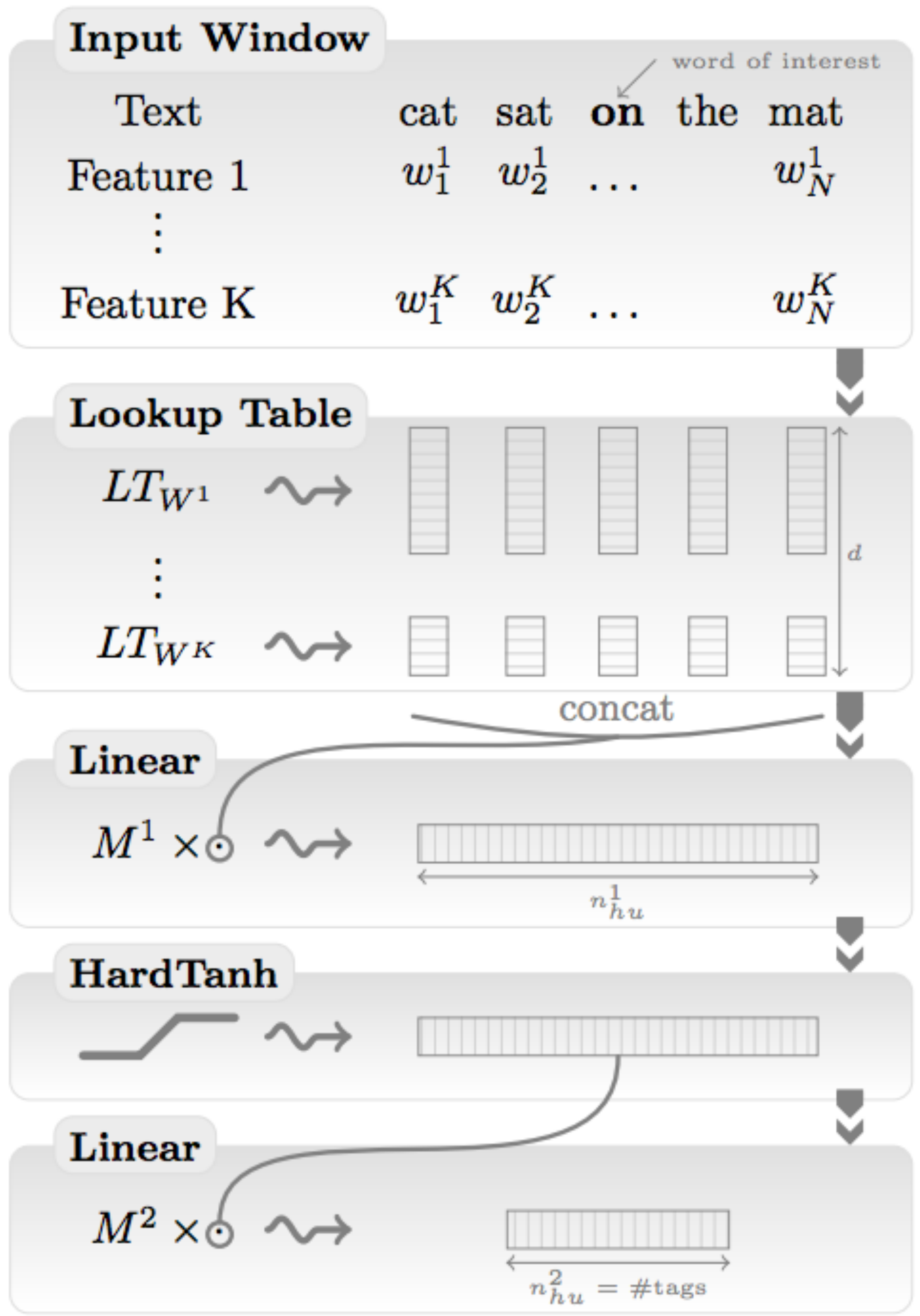

- LSTMs: Hochreiter and Schmidhuber (1997)



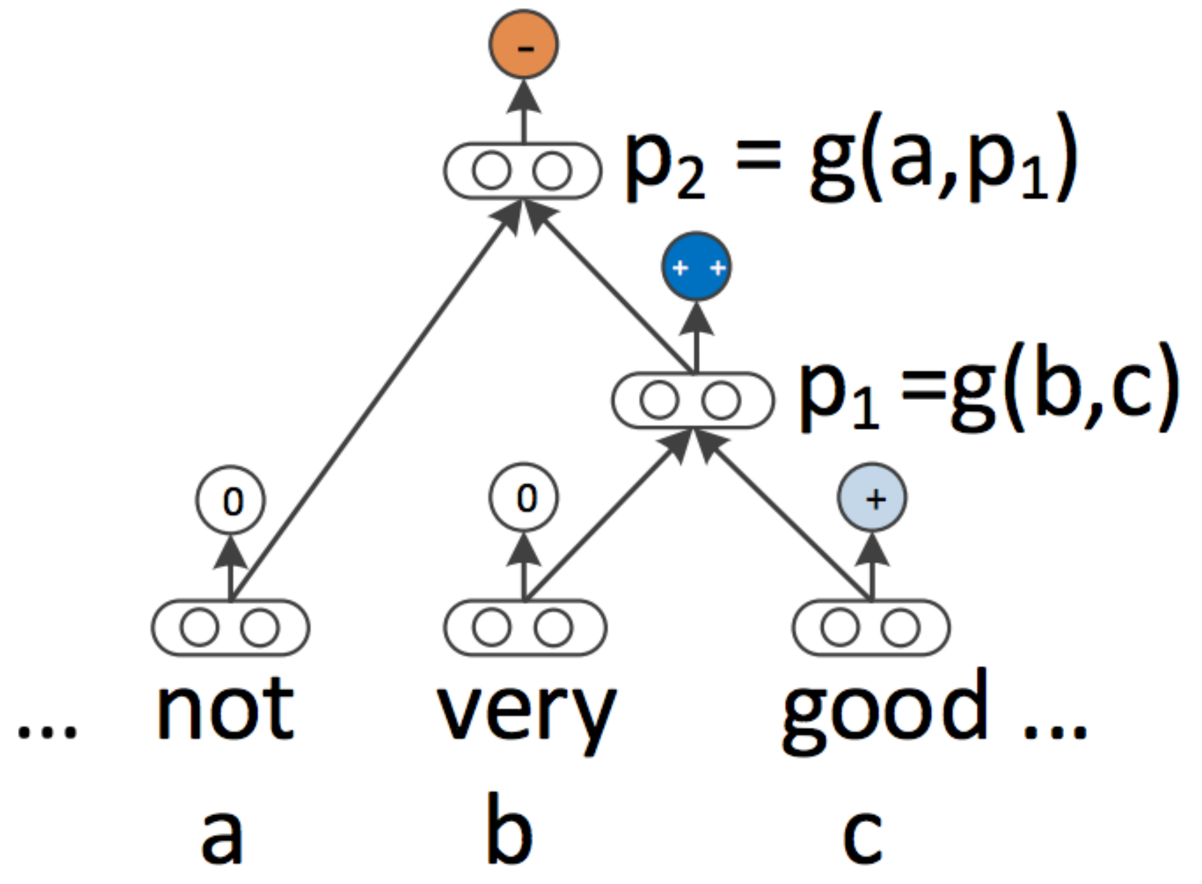

- Henderson (2003): neural shift-reduce parser, not SOTA

# 2008-2013: A glimmer of light...

- Collobert and Weston 2011: "NLP (almost) from scratch"
  - Feedforward neural nets induce features for sequential CRFs ("neural CRF")
  - 2008 version was marred by bad experiments, claimed SOTA but wasn't, 2011 version tied SOTA




- Socher 2011-2014: tree-structured RNNs working okay
- Krizhevskey et al. (2012): AlexNet for vision



# 2014: Stuff starts working

- Kim (2014) + Kalchbrenner et al. (2014): sentence classification / sentiment (convnets)
- Sutskever et al. + Bahdanau et al.: seq2seq for neural MT (LSTMs)
- Chen and Manning transition-based dependency parser (based on feedforward networks)
- 2015: explosion of neural nets for everything under the sun
- What made these work? **Data** (not as important as you might think), **optimization** (initialization, adaptive optimizers), **representation** (good word embeddings)

# Neural Net Basics

# Neural Networks

- Linear classification: $\text{argmax}_y w^\top f(x, y)$

- Want to learn intermediate conjunctive features of the input

*the movie was **not** all that **good***

I[contains *not* & contains *good*]

- How do we learn this if our feature vector is just the unigram indicators?

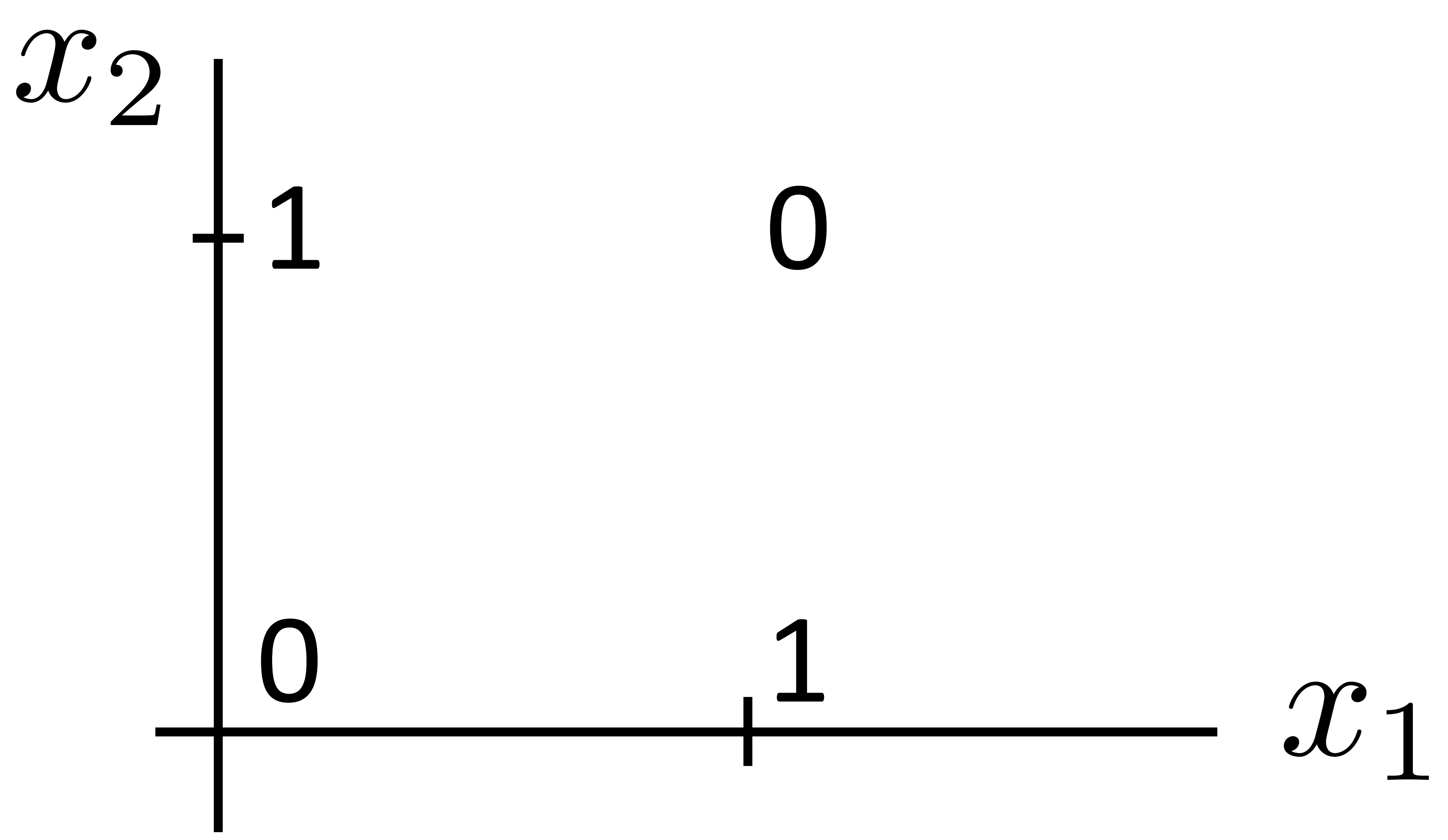
I[contains *not*], I[contains *good*]

# Neural Networks: XOR

- Let's see how we can use neural nets to learn a simple nonlinear function
- Inputs $x_1, \; x_2$

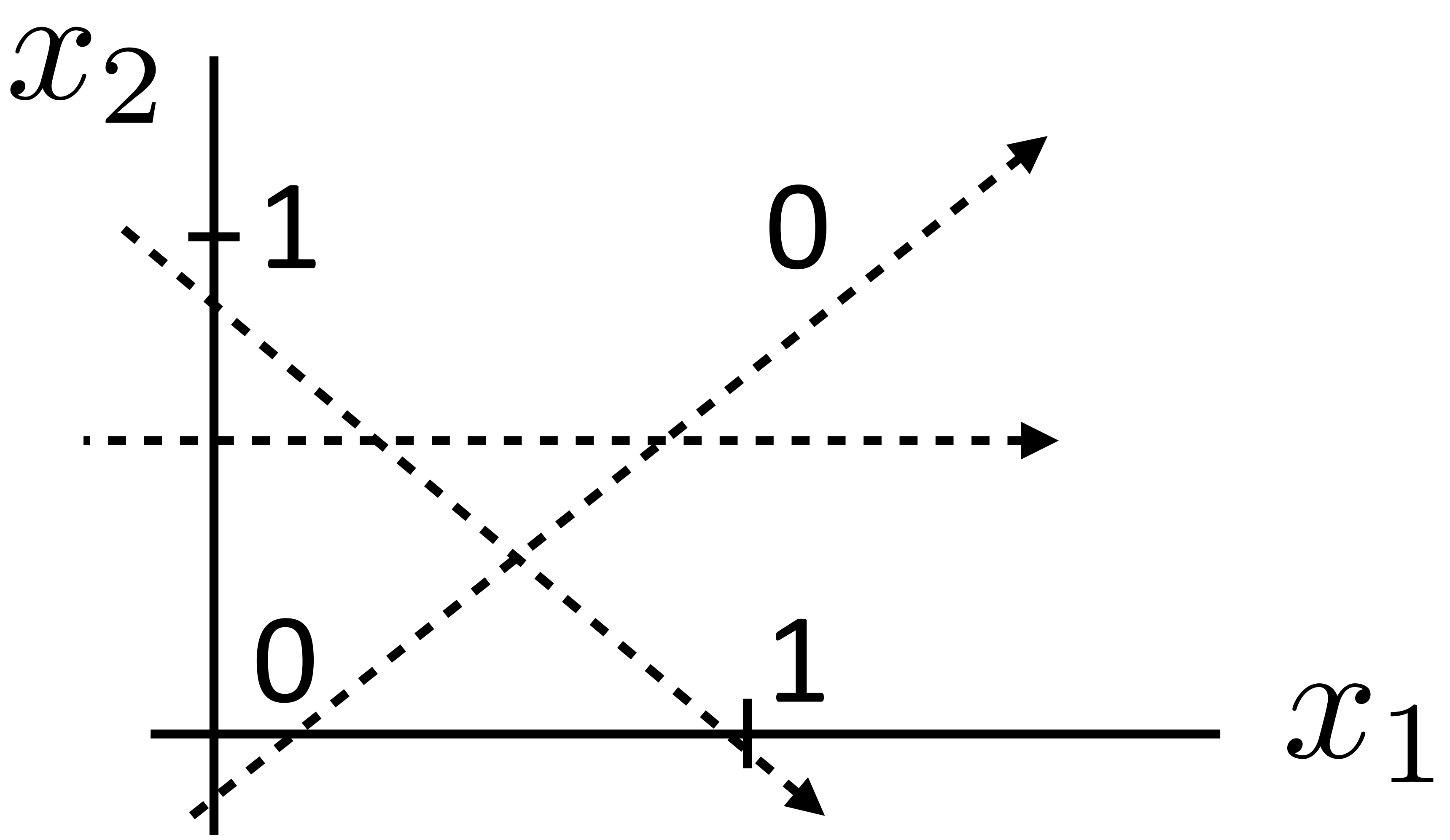
  $(\text{generally } \mathbf{x} = (x_1, \ldots, x_m))$
- Output $y$

  $(\text{generally } \mathbf{y} = (y_1, \ldots, y_n))$




| $x_1$ | $x_2$ | $y = x_1 \text{ XOR } x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

# Neural Networks: XOR



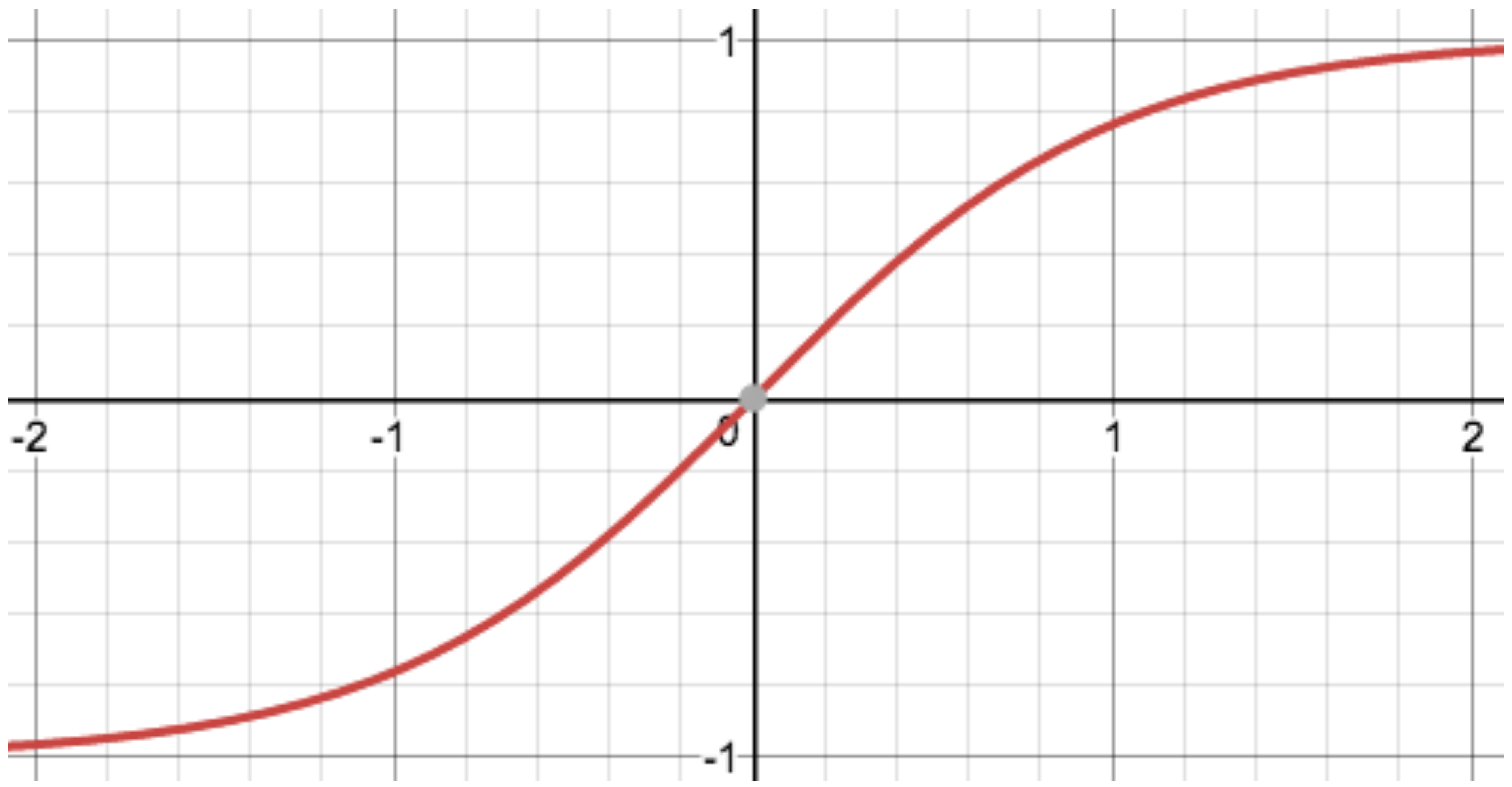

$$y = a_1 x_1 + a_2 x_2$$ X

$$y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)$$ ✓

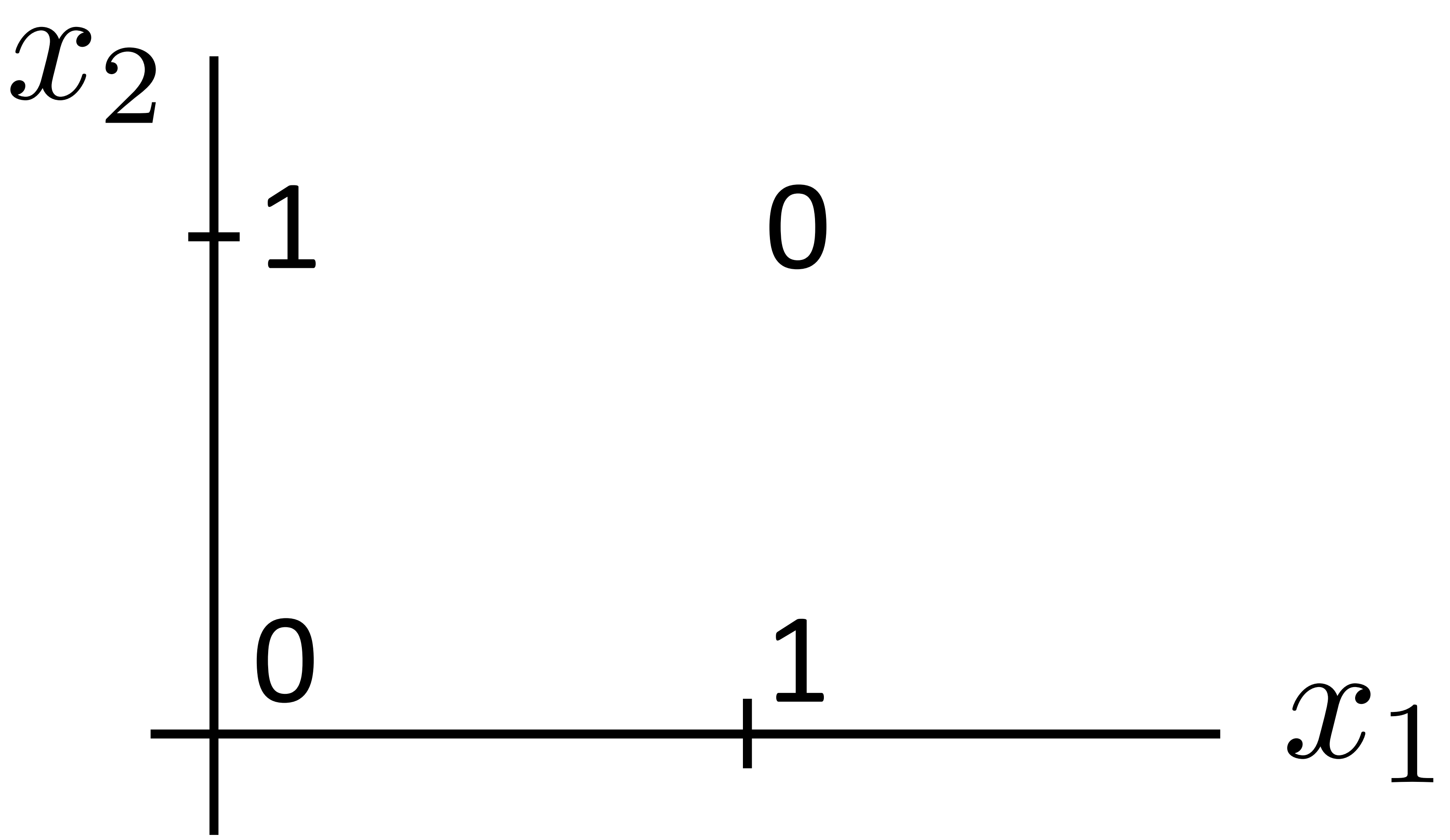
"or"

(looks like action potential in neuron)

| $x_1$ | $x_2$ | $x_1$ XOR $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



# Neural Networks: XOR



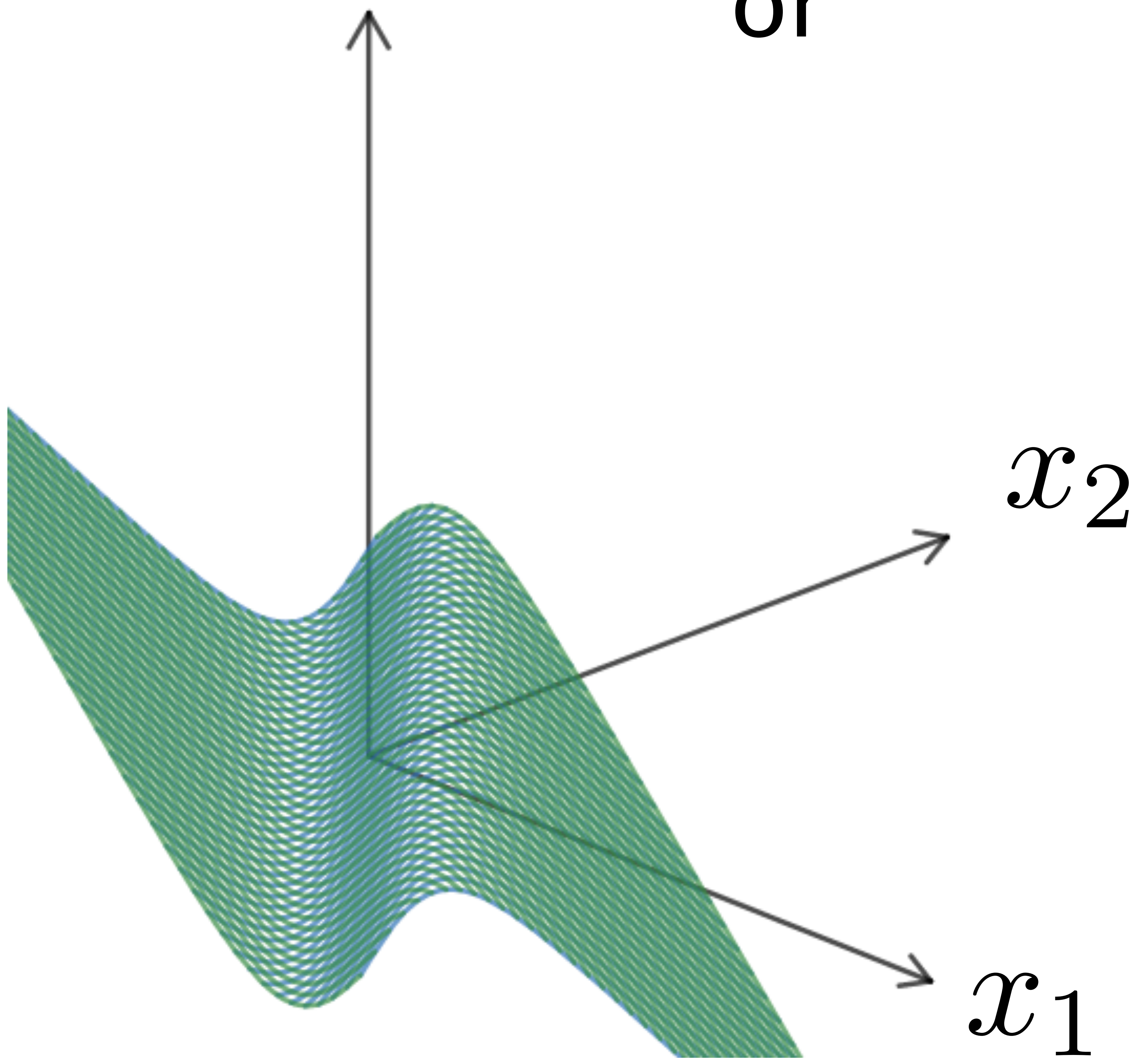

| $x_1$ | $x_2$ | $x_1$ XOR $x_2$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$$y = a_1 x_1 + a_2 x_2$$ X

$$y = a_1 x_1 + a_2 x_2 + a_3 \tanh(x_1 + x_2)$$ ✓

$$y = -x_1 - x_2 + 2\tanh(x_1 + x_2)$$

“or”



# Neural Networks

Linear model: $y = \mathbf{w} \cdot \mathbf{x} + b$

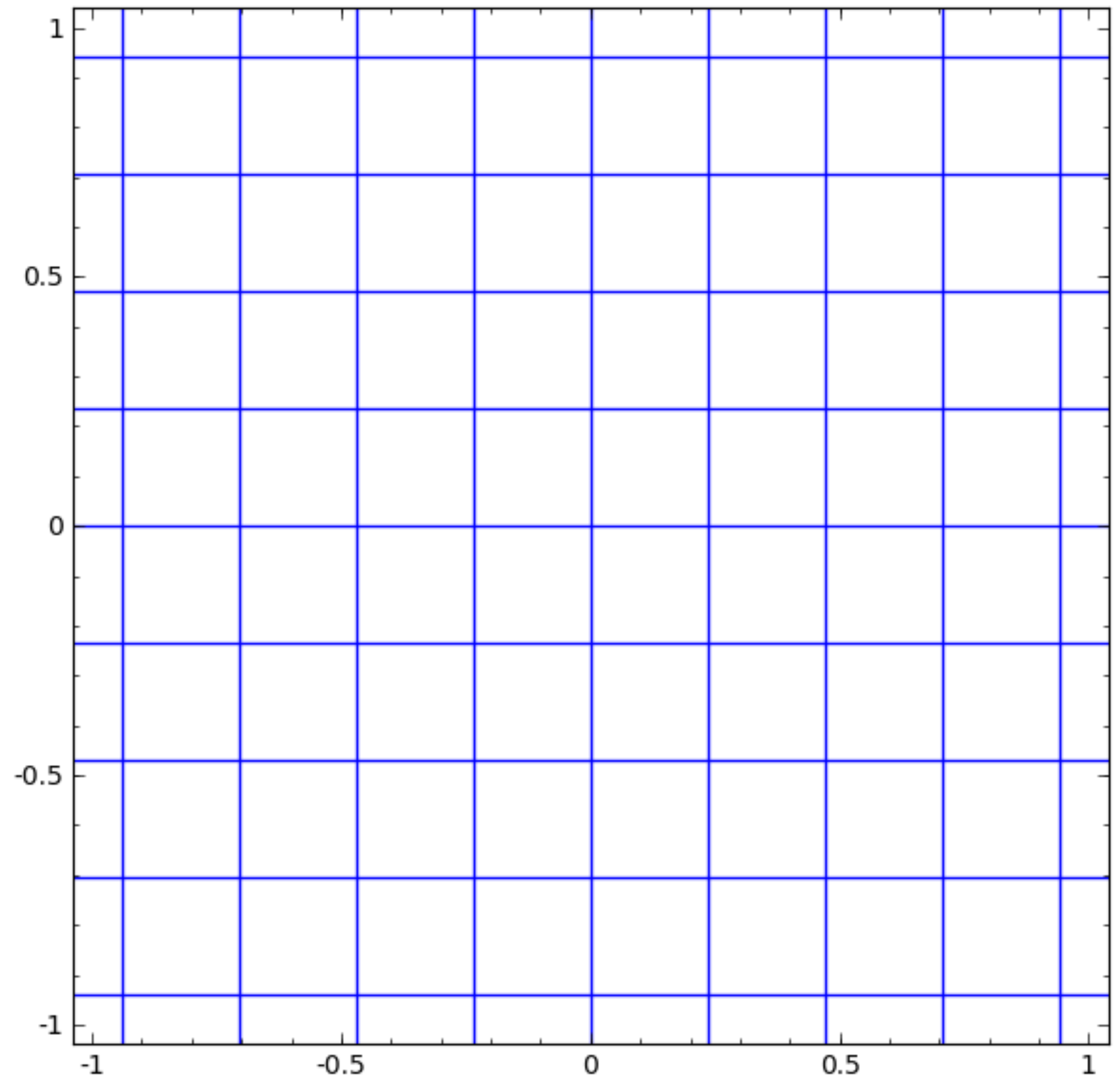
$$y = g(\mathbf{w} \cdot \mathbf{x} + b)$$

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

Nonlinear transformation

Linear transf.

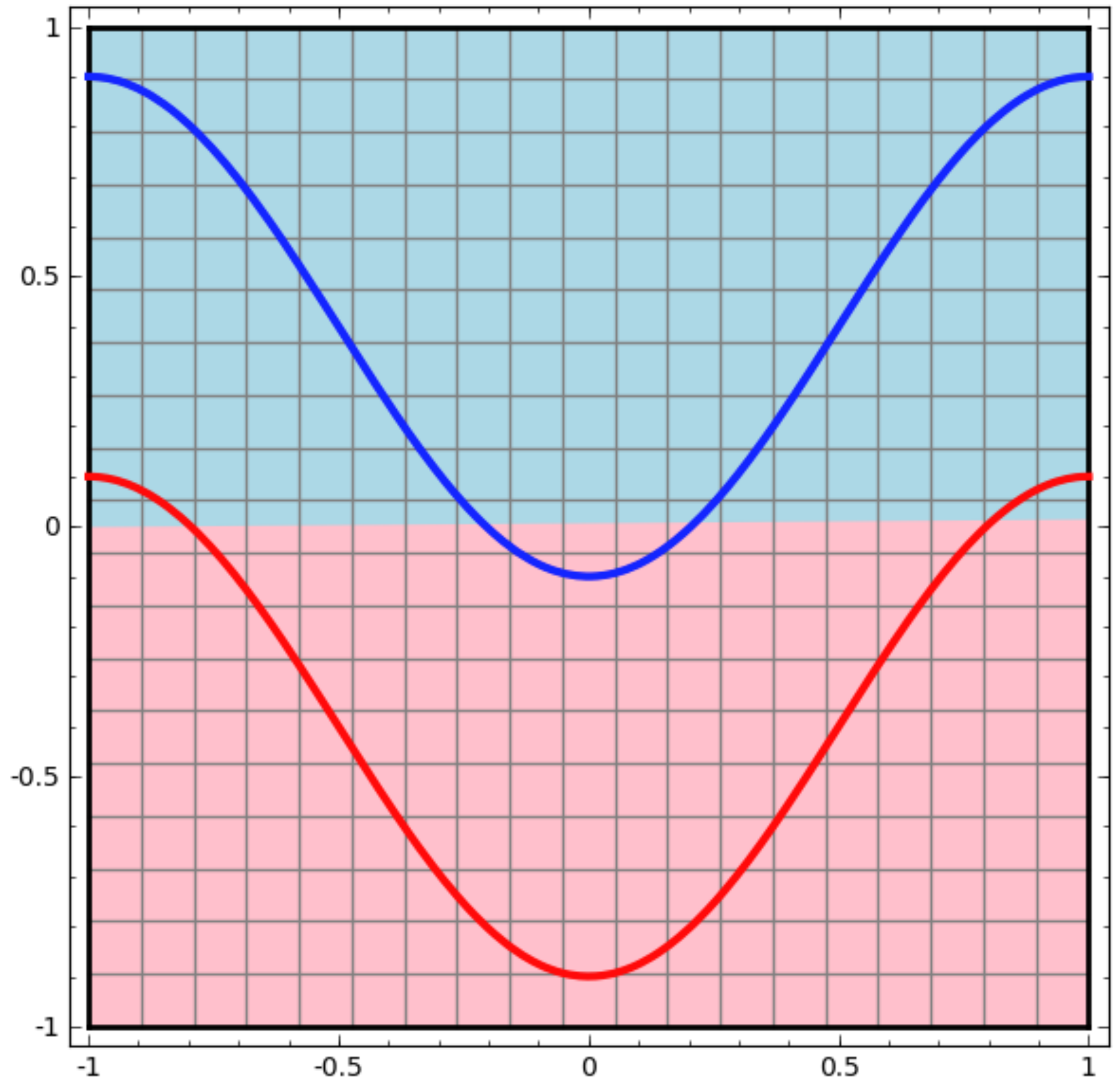
Shift



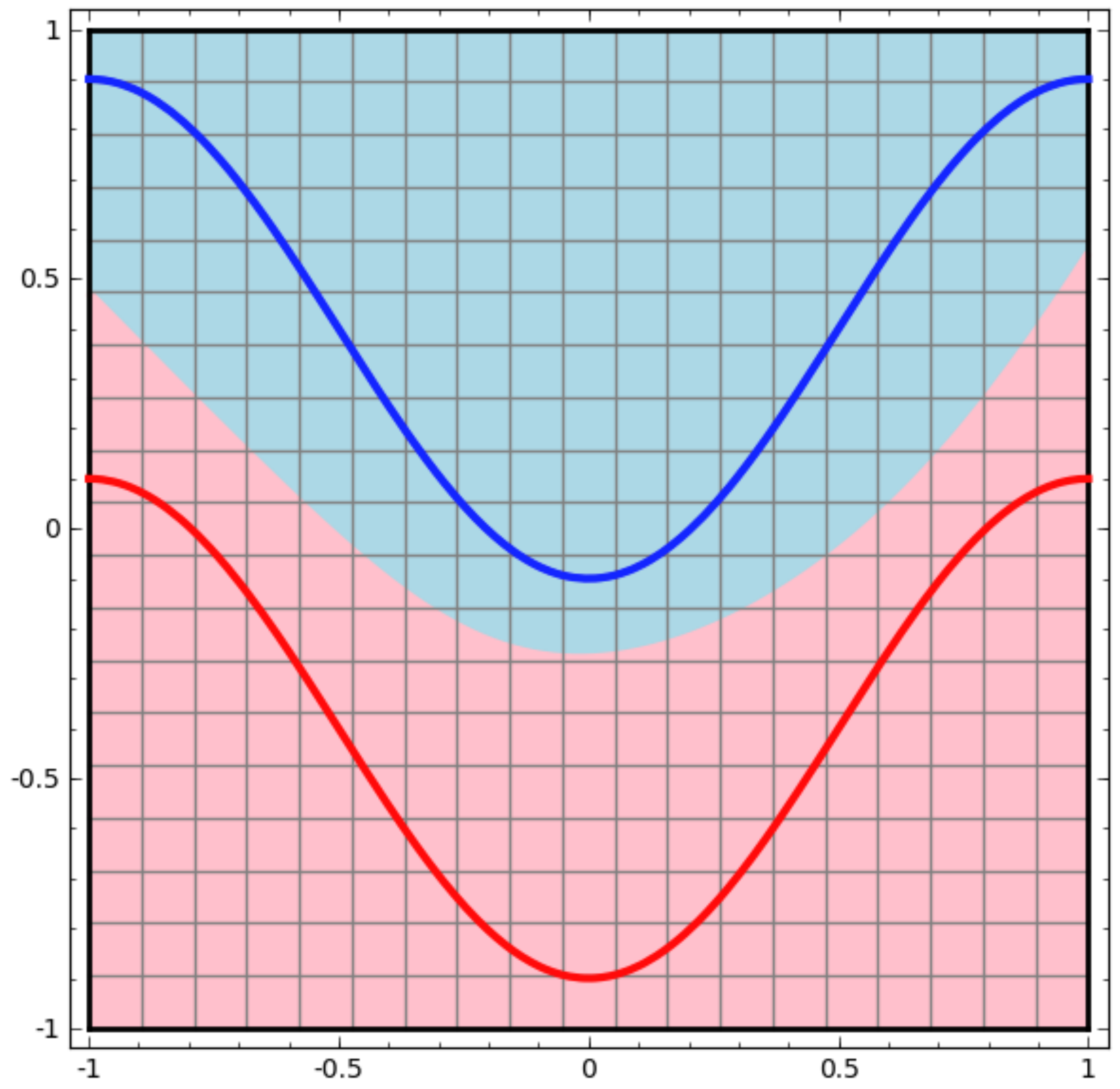

Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

# Neural Networks

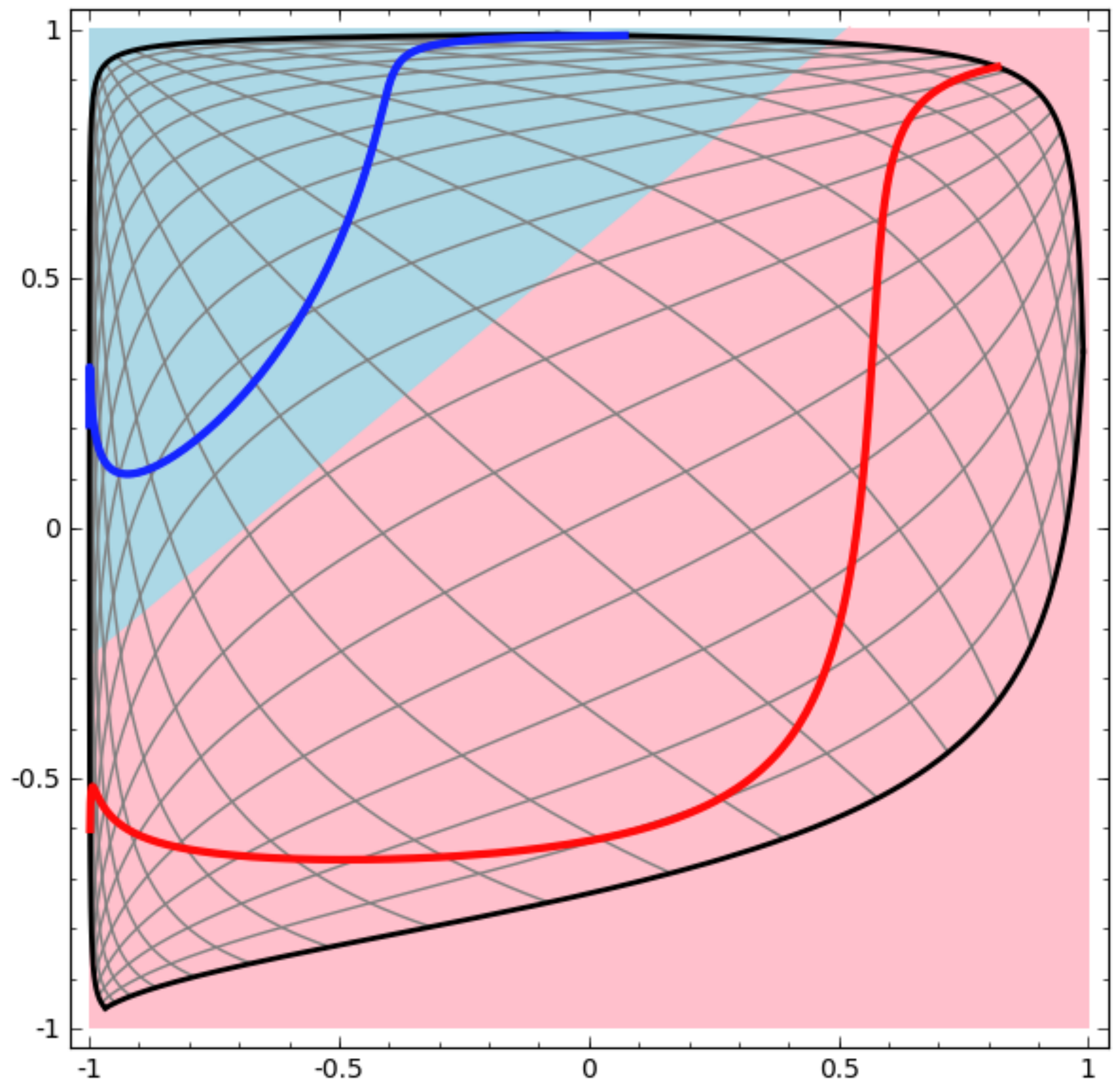
Linear classifier




Neural network




…possible because we transformed the space!



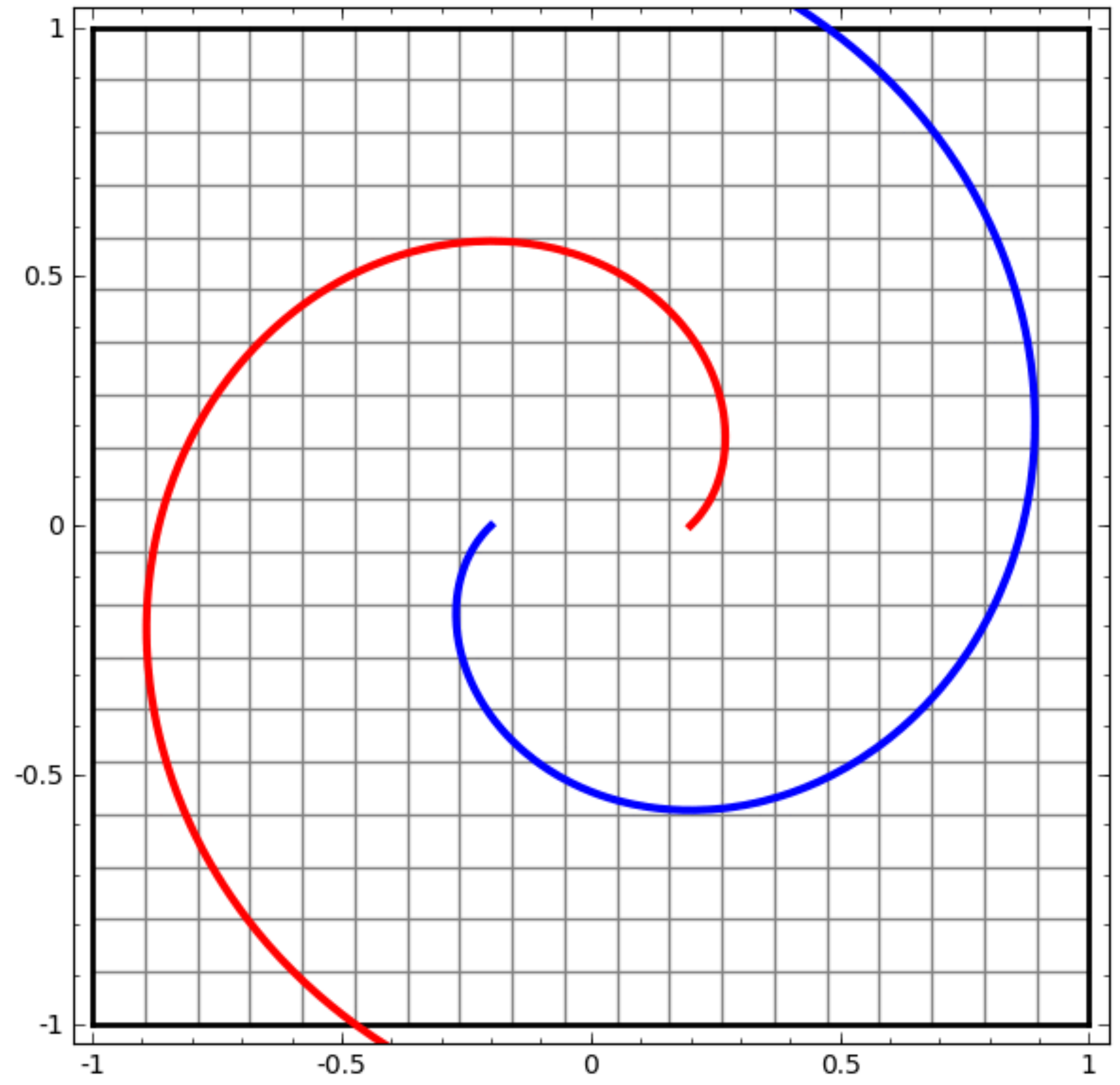

Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

# Deep Neural Networks

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

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

$$\mathbf{z} = g(\mathbf{V}\underbrace{g(\mathbf{W}\mathbf{x} + \mathbf{b})}_{\text{output of first layer}} + \mathbf{c})$$




Check: what happens if no nonlinearity?

More powerful than basic linear models?

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

Taken from http://colah.github.io/posts/2014-03-NN-Manifolds-Topology/

# Feedforward Networks, Backpropagation

# Logistic Regression with NNs

$$P(y|\mathbf{x}) = \frac{\exp(w^\top f(\mathbf{x}, y))}{\sum_{y'} \exp(w^\top f(\mathbf{x}, y'))}$$

- Single scalar probability

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}\left([w^\top f(\mathbf{x}, y)]_{y \in \mathcal{Y}}\right)$$

- Compute scores for all possible labels at once (returns vector)

$$\text{softmax}(p)_i = \frac{\exp(p_i)}{\sum_{i'} \exp(p_{i'})}$$

- softmax: exps and normalizes a given vector

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(W f(\mathbf{x}))$$

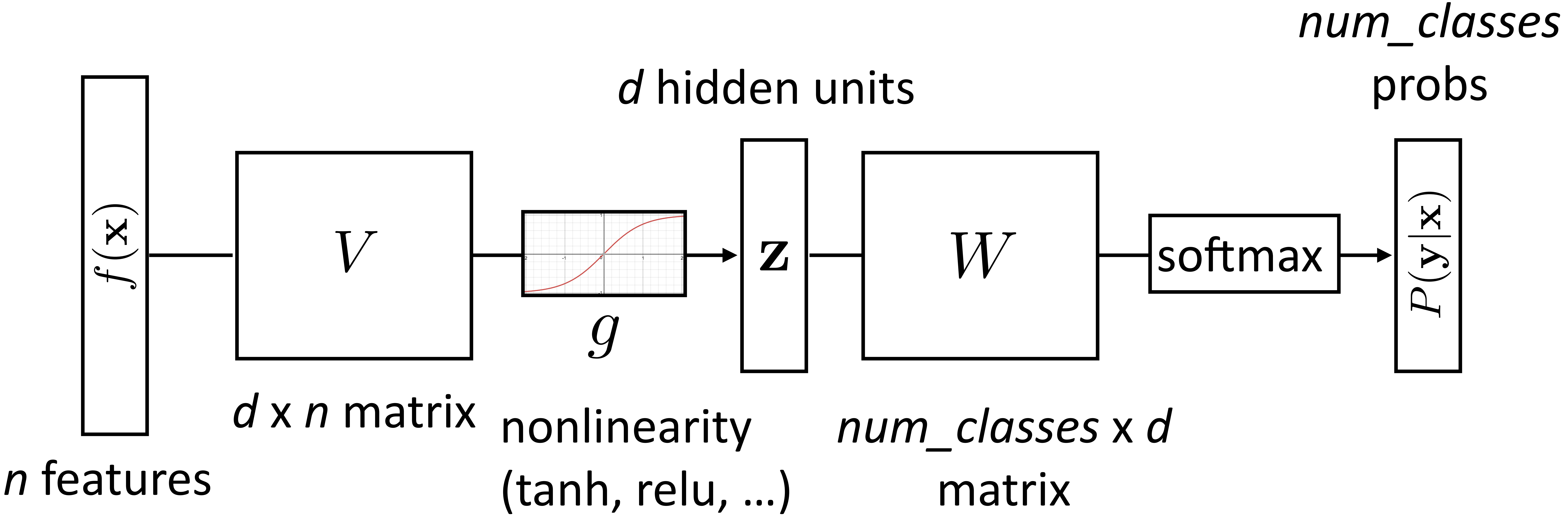
- Weight vector per class;
  W is [num classes x num feats]

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(W g(V f(\mathbf{x})))$$

- Now one hidden layer

# Neural Networks for Classification

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$



# Training Neural Networks

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(W\mathbf{z}) \qquad \mathbf{z} = g(Vf(\mathbf{x}))$$

- Maximize log likelihood of training data

$$\mathcal{L}(\mathbf{x}, i^*) = \log P(y = i^*|\mathbf{x}) = \log\left(\text{softmax}(W\mathbf{z}) \cdot e_{i^*}\right)$$

- *i**: index of the gold label

- $e_i$: 1 in the *i*th row, zero elsewhere. Dot by this = select *i*th index

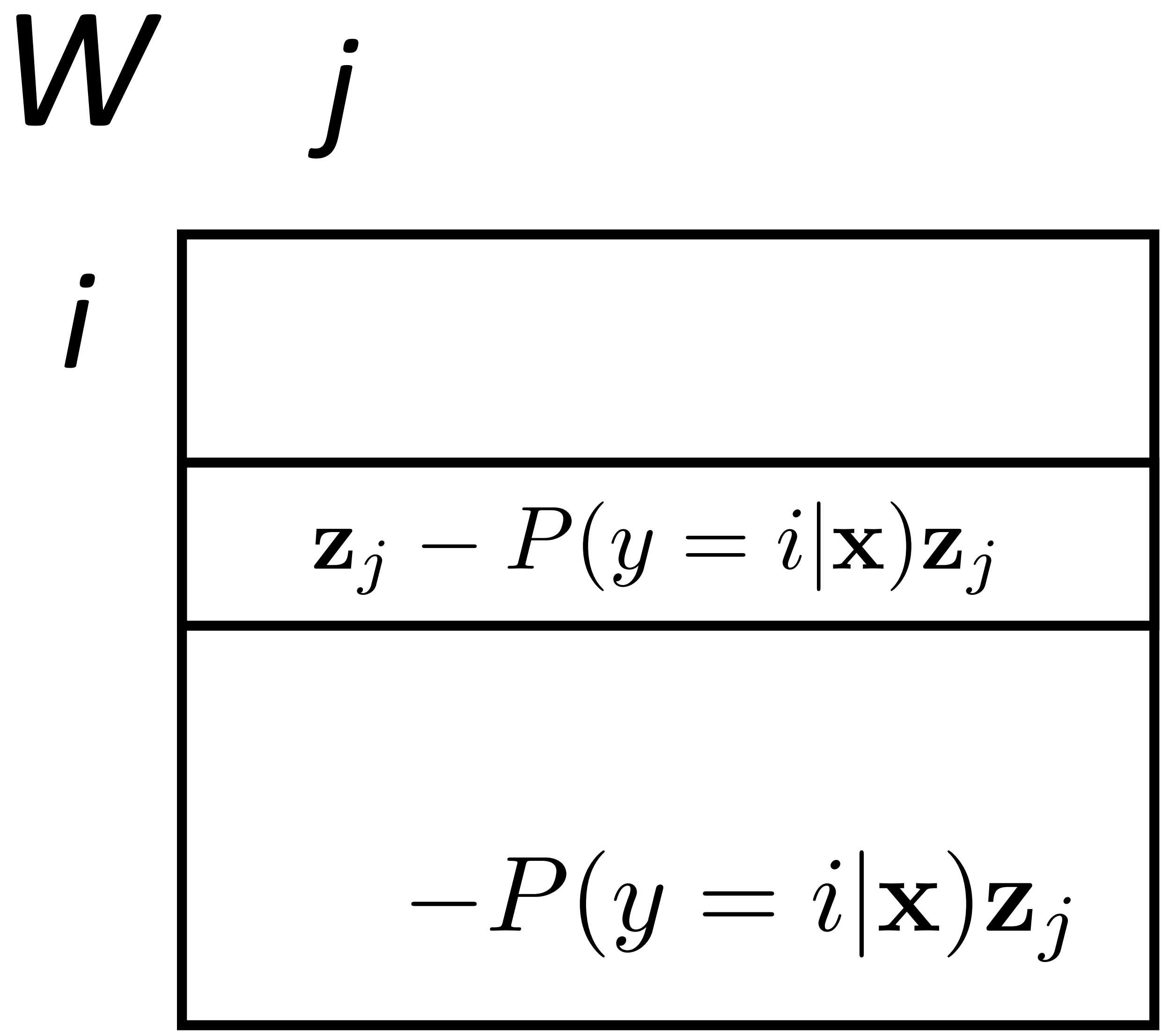
$$\mathcal{L}(\mathbf{x}, i^*) = W\mathbf{z} \cdot e_{i^*} - \log \sum_j \exp(W\mathbf{z}) \cdot e_j$$

# Computing Gradients

$$\mathcal{L}(\mathbf{x}, i^*) = W\mathbf{z} \cdot e_{i^*} - \log \sum_j \exp(W\mathbf{z}) \cdot e_j$$

- Gradient with respect to *W*

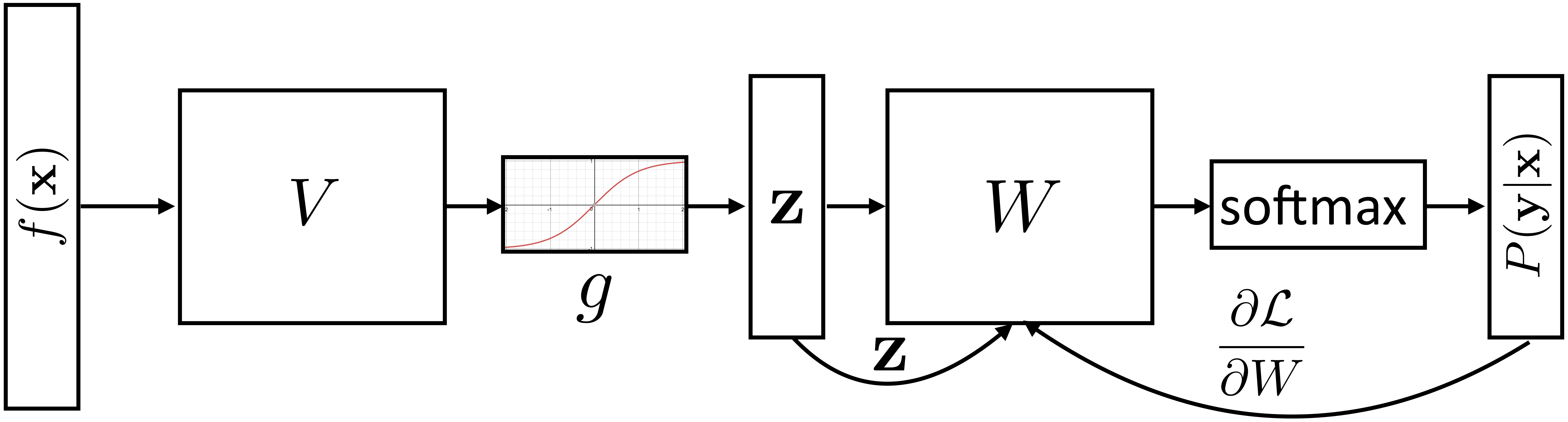
$$\frac{\partial}{\partial W_{ij}} \mathcal{L}(\mathbf{x}, i^*) = \begin{cases} \mathbf{z}_j - P(y = i|\mathbf{x})\mathbf{z}_j & \text{if } i = i^* \\ -P(y = i|\mathbf{x})\mathbf{z}_j & \text{otherwise} \end{cases}$$




- Looks like logistic regression with ***z*** as the features!

# Neural Networks for Classification

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$



# Computing Gradients: Backpropagation

$$\mathcal{L}(\mathbf{x}, i^*) = W\mathbf{z} \cdot e_{i^*} - \log \sum_j \exp(W\mathbf{z}) \cdot e_j$$

$$\mathbf{z} = g(Vf(\mathbf{x}))$$

Activations at hidden layer

- Gradient with respect to *V*: apply the chain rule

$$\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial V_{ij}} = \boxed{\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}}} \frac{\partial \mathbf{z}}{\partial V_{ij}}$$

[some math...]

$$err(\text{root}) = e_{i^*} - P(\mathbf{y}|\mathbf{x})$$

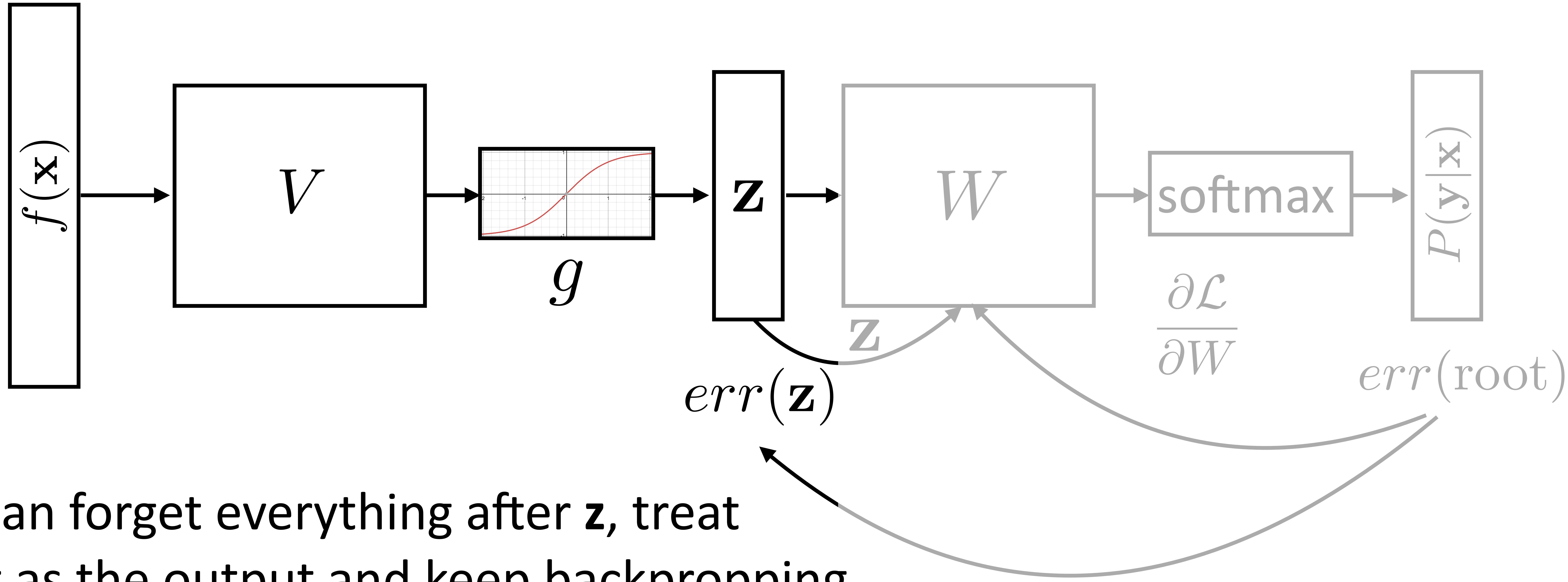
dim = m

$$\boxed{\frac{\partial \mathcal{L}(\mathbf{x}, i^*)}{\partial \mathbf{z}} = err(\mathbf{z}) = W^\top err(\text{root})}$$

dim = d

# Backpropagation: Picture

$$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(Wg(Vf(\mathbf{x})))$$




- Can forget everything after **z**, treat it as the output and keep backpropping

# Backpropagation: Takeaways

- Gradients of output weights *W* are easy to compute — looks like logistic regression with hidden layer ***z*** as feature vector
- Can compute derivative of loss with respect to ***z*** to form an "error signal" for backpropagation
- Easy to update parameters based on "error signal" from next layer, keep pushing error signal back as backpropagation
- Need to remember the values from the forward computation

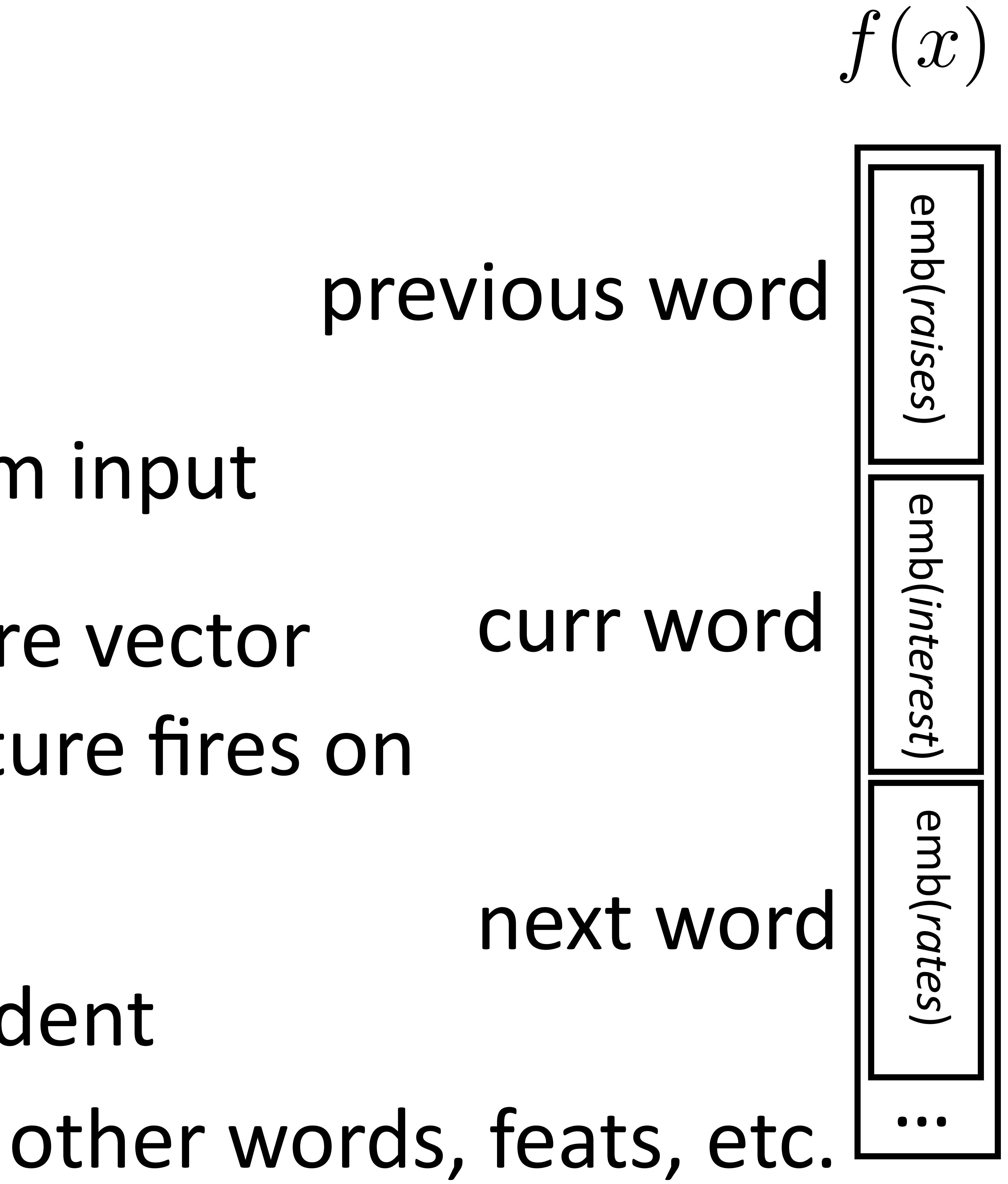
# Applications

# NLP with Feedforward Networks

- Part-of-speech tagging with FFNNs

??

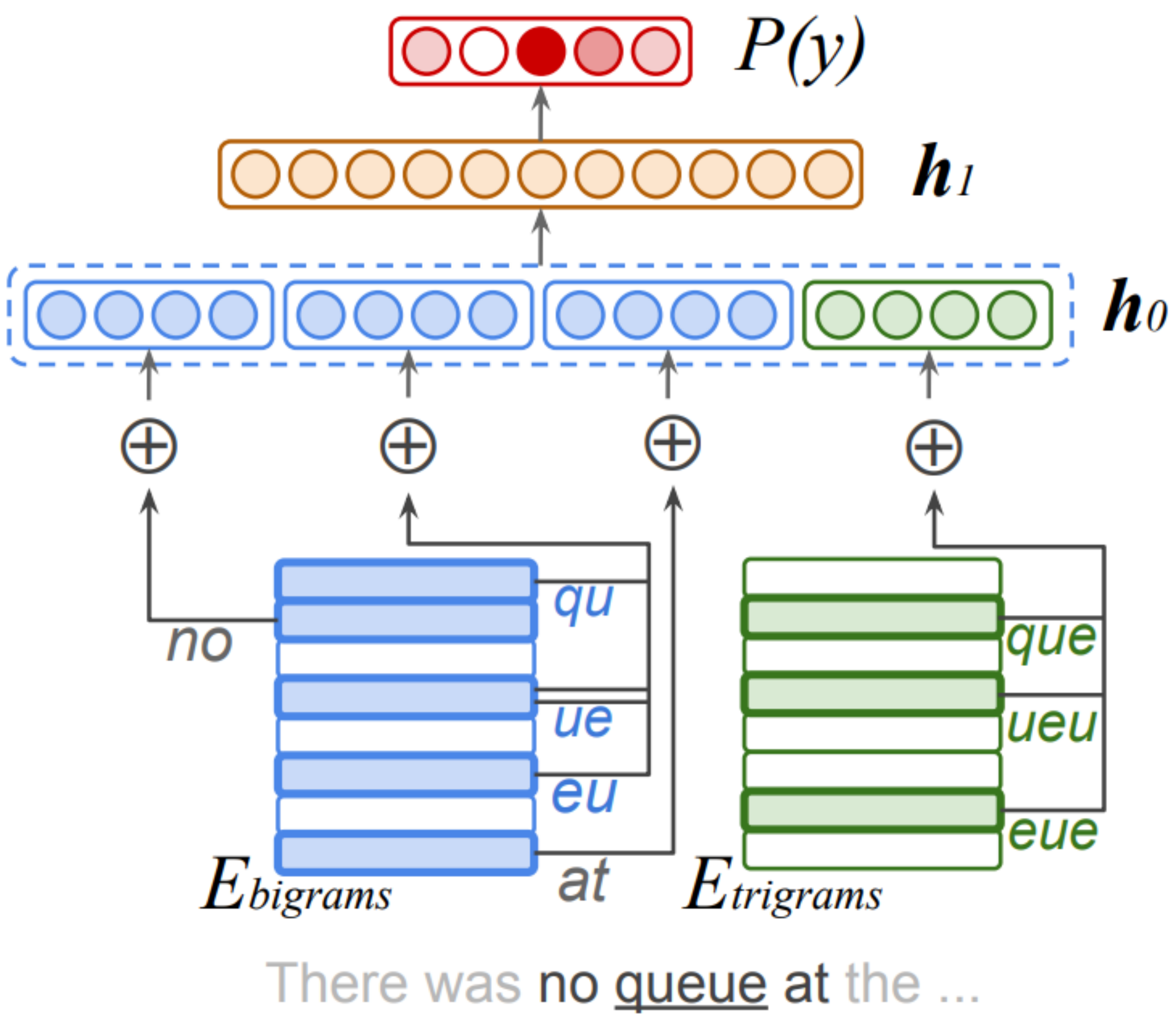
*Fed raises* ***interest*** *rates in order to …*




- Word embeddings for each word form input
- ~1000 features here — smaller feature vector than in sparse models, but every feature fires on every example
- Weight matrix learns position-dependent processing of the words

Botha et al. (2017)

# NLP with Feedforward Networks




- Hidden layer mixes these different signals and learns feature conjunctions

Botha et al. (2017)

# NLP with Feedforward Networks

- Multilingual tagging results:

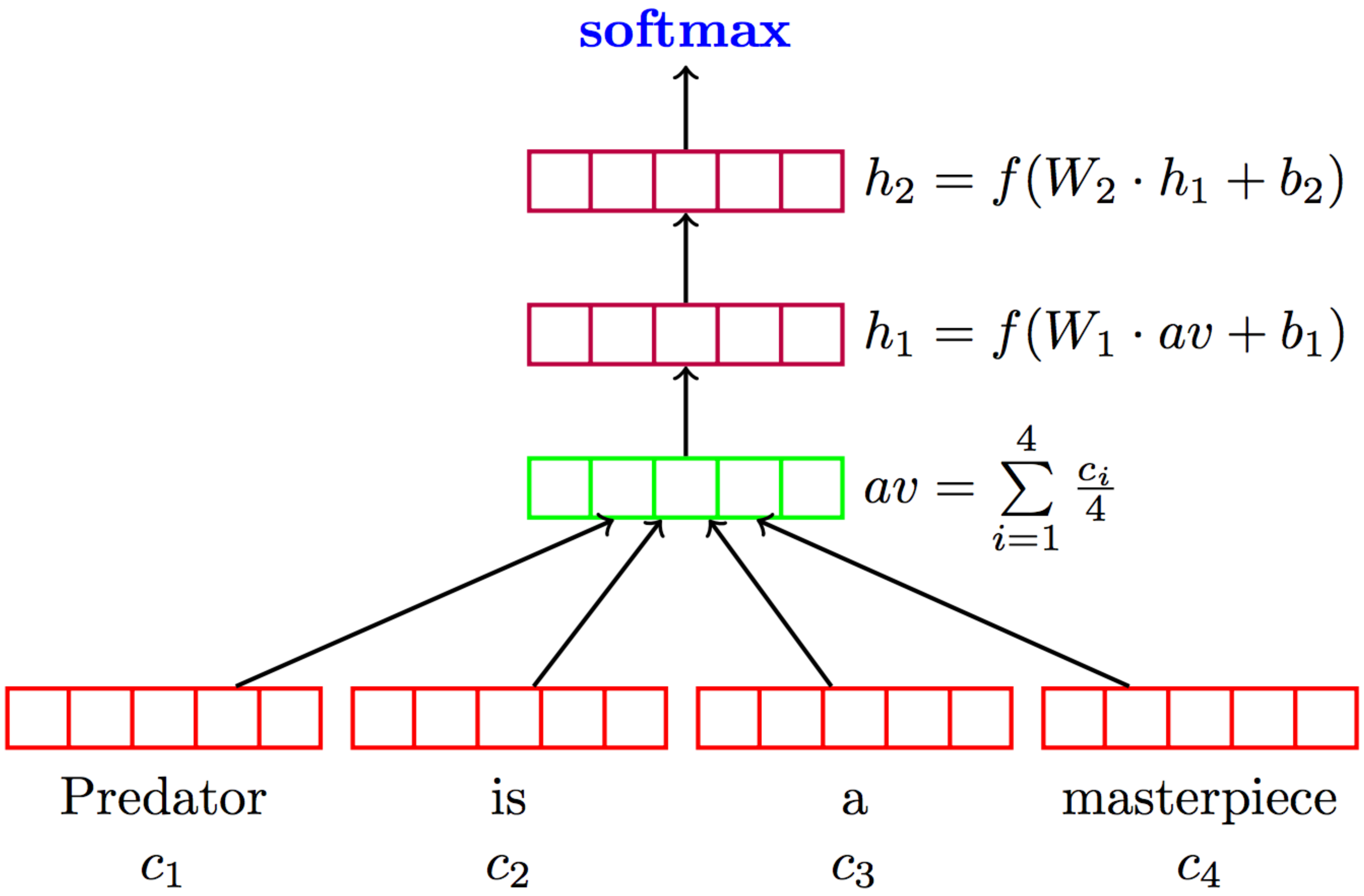
| Model | Acc. | Wts. | MB | Ops. |
|---|---|---|---|---|
| Gillick et al. (2016) | 95.06 | 900k | - | 6.63m |
| Small FF | 94.76 | 241k | 0.6 | 0.27m |
| +Clusters | 95.56 | 261k | 1.0 | 0.31m |
| $\frac{1}{2}$ Dim. | 95.39 | 143k | 0.7 | 0.18m |

- Gillick used LSTMs; this is smaller, faster, and better

Botha et al. (2017)

# Sentiment Analysis

- Deep Averaging Networks: feedforward neural network on average of word embeddings from input




Iyyer et al. (2015)

# Sentiment Analysis

| | Model | RT | SST fine | SST bin | IMDB | Time (s) | |
|---|---|---|---|---|---|---|---|
| | DAN-ROOT | — | 46.9 | 85.7 | — | **31** | |
| | DAN-RAND | 77.3 | 45.4 | 83.2 | 88.8 | 136 | |
| | DAN | 80.3 | 47.7 | 86.3 | 89.4 | 136 | Iyyer et al. (2015) |
| Bag-of-words | NBOW-RAND | 76.2 | 42.3 | 81.4 | 88.9 | 91 | |
| | NBOW | 79.0 | 43.6 | 83.6 | 89.0 | 91 | |
| | BiNB | — | 41.9 | 83.1 | — | — | |
| | NBSVM-bi | 79.4 | — | — | 91.2 | — | Wang and Manning (2012) |
| Tree RNNs / CNNS / LSTMS | RecNN* | 77.7 | 43.2 | 82.4 | — | — | |
| | RecNTN* | — | 45.7 | 85.4 | — | — | |
| | DRecNN | — | 49.8 | 86.6 | — | 431 | |
| | TreeLSTM | — | **50.6** | 86.9 | — | — | |
| | DCNN* | — | 48.5 | 86.9 | 89.4 | — | |
| | PVEC* | — | 48.7 | 87.8 | **92.6** | — | |
| | CNN-MC | **81.1** | 47.4 | **88.1** | — | 2,452 | Kim (2014) |
| | WRRBM* | — | — | — | 89.2 | — | |

# Implementation Details

# Computation Graphs

- Computing gradients is hard! Computation graph abstraction allows us to define a computation symbolically and will do this for us

- Automatic differentiation: keep track of derivatives / be able to backpropagate through each function:

```
y = x * x   --codegen-->   (y,dy) = (x * x, 2 * x * dx)
```

- Use a library like Pytorch or Tensorflow. This class: Pytorch

# Computation Graphs in Pytorch

- Define forward pass for $P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$

```
class FFNN(nn.Module):
    def __init__(self, inp, hid, out):
        super(FFNN, self).__init__()
        self.V = nn.Linear(inp, hid)
        self.g = nn.Tanh()
        self.W = nn.Linear(hid, out)
        self.softmax = nn.Softmax(dim=0)

    def forward(self, x):
        return self.softmax(self.W(self.g(self.V(x))))
```

# Computation Graphs in Pytorch

$$P(\mathbf{y}|\mathbf{x}) = \mathrm{softmax}(Wg(Vf(\mathbf{x})))$$

ei*: one-hot vector of the label (e.g., [0, 1, 0])

```
ffnn = FFNN()
def make_update(input, gold_label):
   ffnn.zero_grad() # clear gradient variables
   probs = ffnn.forward(input)
   loss = torch.neg(torch.log(probs)).dot(gold_label)
   loss.backward()
   optimizer.step()
```

# Training a Model

Define a computation graph

For each epoch:

    For each batch of data:

        Compute loss on batch

        Autograd to compute gradients

        Take step with optimizer

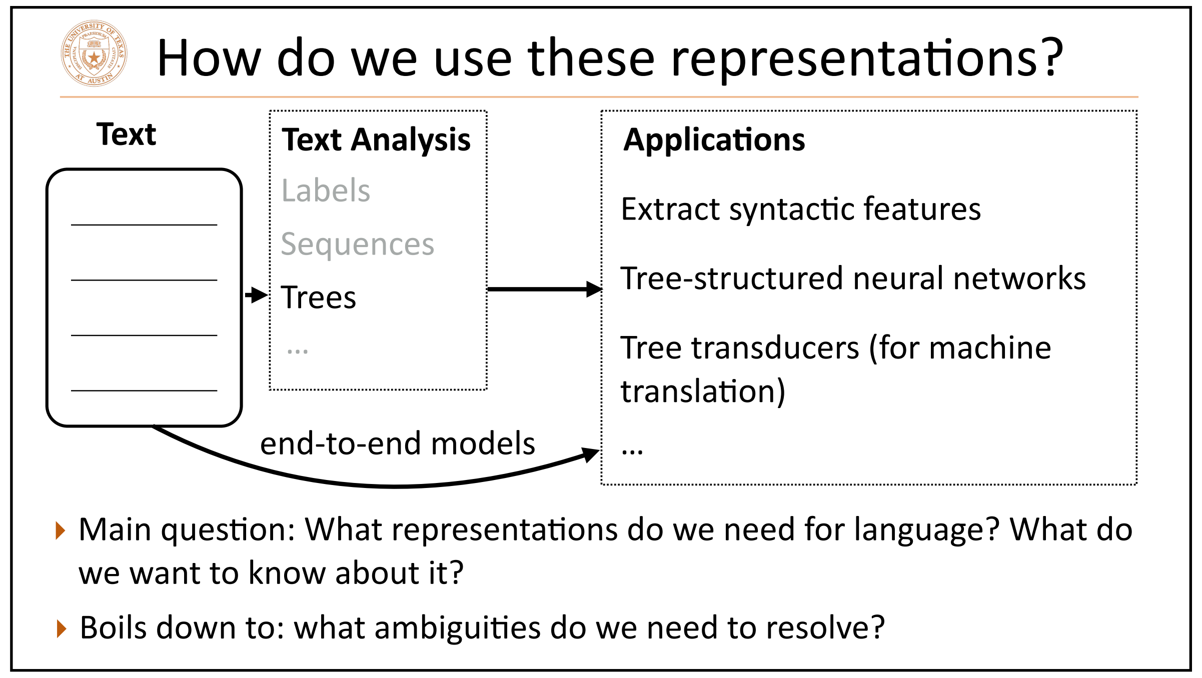
Decode test set

# Next Time

- additional network architectures (convolutional, recurrent)

# slide credits

slides that look like this




come from

CS388 given by Greg Durrett at U Texas, Austin

and their use is gratefully acknowledged. I try to make any modifications obvious, but if there are errors on a slide, assume that I added them.