Neural Language Modeling

Katharina Kann — CSCI/LING5832

Evaluation of language models

Model Evaluation

- How do we know if our models are any good?
  - …and how do we know if one model is better than another?
- Well Shannon's game gives us an intuition.
  - The generated texts from the higher order models surely sound better.
    - That is, they sound more like the text the model was obtained from.
- But what does that mean? How can we make that notion operational?

Evaluating N-Gram Models

- Best evaluation for a language model
  - Put model A into an application
    - For example, a machine translation system
  - Evaluate the performance of the application with model A
  - Put model B into the application and evaluate
  - Compare performance of the application with the two models
    - **Extrinsic** evaluation

Evaluation

- Extrinsic evaluation
  - This is really time-consuming and can be hard
  - Not something you want to do unless you’re pretty sure you’ve got a good solution

Evaluation

- Extrinsic evaluation
  - This is really time-consuming and can be hard
  - Not something you want to do unless you're pretty sure you've got a good solution
- So
  - As a intermediate evaluation, in order to run rapid experiments we evaluate N-grams with an **intrinsic** evaluation
  - An evaluation that tries to capture how good the model is intrinsically, not how much it improves performance in some larger system

Evaluation

- Standard method
  - Train parameters of our model on a training set.
  - Evaluate the model on some new data: a test set.
    - A dataset which is different from our training set, but drawn from the same source

Perplexity

- The intuition behind perplexity as a measure is the notion of surprise.
  - How surprised is the language model when it sees the test set?
    - The more surprised the model is, the lower the probability it assigned to the test set.
    - The higher the probability, the less surprised it was.

Perplexity

- Perplexity is just the probability of a test set (assigned by the language model), as normalized by the number of words:

$$
\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N}\,)}
\end{aligned}
$$

- Chain rule:

$$
\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}
$$

- For bigrams:

$$
\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}
$$

Perplexity

- Perplexity is just the probability of a test set (assigned by the language model), as normalized by the number of words:

$$\begin{aligned}\mathrm{PP}(W) &= P(w_1 w_2 \ldots w_N)^{-\frac{1}{N}} \\ &= \sqrt[N]{\frac{1}{P(w_1 w_2 \ldots w_N}})\end{aligned}$$

- Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1 \ldots w_{i-1})}}$$

- For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_{i-1})}}$$

- Minimizing model perplexity is the same as maximizing probability of a test set.

Lower Perplexity is Better

- Training 38 million words, test 1.5 million words, WSJ

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Practical Issues

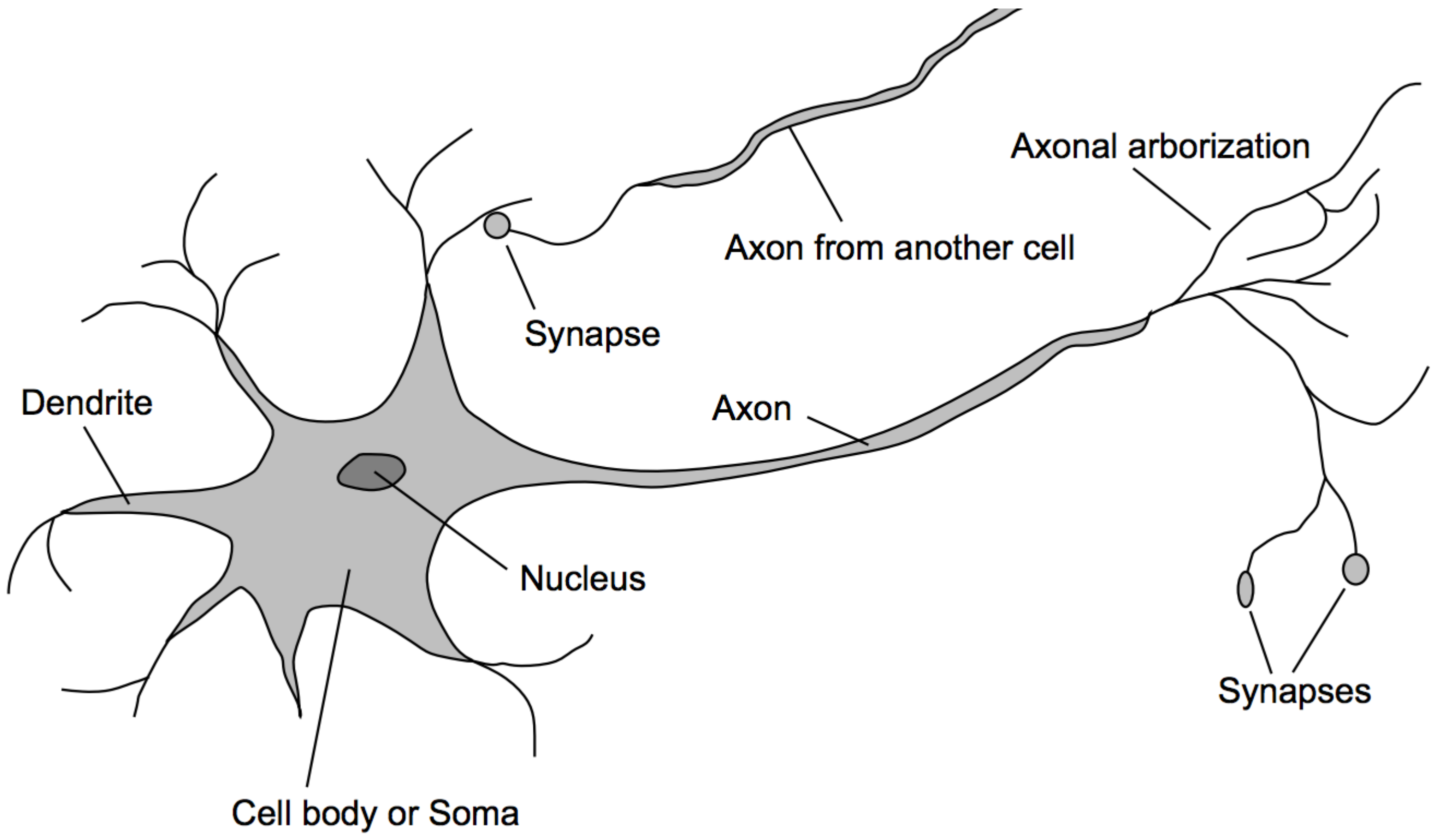
- Once we start looking at test data, we'll run into words that we haven't seen before. So our models won't work. Standard solution:
  - Given a corpus
    - Create a fixed lexicon L, of size V
      - Say from a dictionary or
      - A subset of terms from the training set
    - Any word not in L is changed to <UNK>
    - Collect counts for that as for any normal word
  - At test time
    - Use <UNK> counts for any word not seen in training

Practical Issues

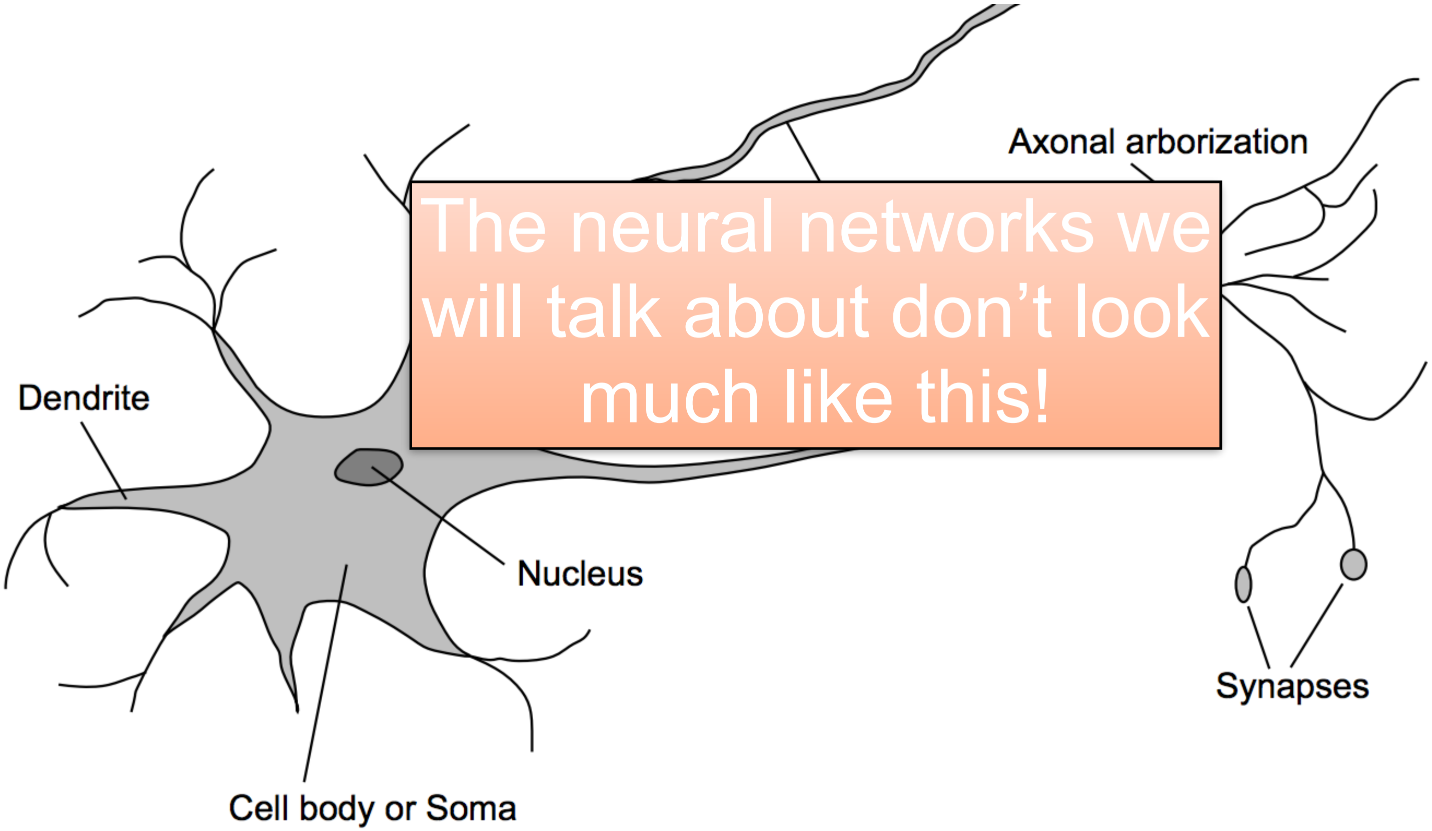
- Multiplying a bunch of really small numbers < 0 is a really bad idea.
  - Underflow is likely
- So do everything in log space
  - Avoids underflow (and adding is faster than multiplying)

Neural networks

Motivation


Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

Motivation



The neural networks we will talk about don't look much like this!

Overview

- Advantage: no feature engineering needed.
- Neural networks detect on their own what parts of the raw input are important.
- They are really good at that!

Applications (Some Examples)

- Sequence classification
  - Sentiment analysis
  - Natural language inference

Applications (Some Examples)

- Sequence classification
  - Sentiment analysis
  - Natural language inference
- Sequence labeling
  - Part-of-speech tagging
  - Named entity recognition

Applications (Some Examples)

- Sequence classification
  - Sentiment analysis
  - Natural language inference
- Sequence labeling
  - Part-of-speech tagging
  - Named entity recognition
- Sequence generation
  - Language modeling
  - Machine translation

Perceptron

- Single neural unit
- Binary output
- Linear classifier

$$z = b + \Sigma_0^i w_i z_i$$

$$z > 0 \rightarrow y = 1$$
$$z \leq 0 \rightarrow y = 0$$

Perceptron - Example

$$z = b + \Sigma_0^i w_i x_i$$

$$z > 0 \rightarrow y = 1$$
$$z \leq 0 \rightarrow y = 0$$

- Assume
  - weights w=[2, 4, 8, 3], bias b = 0
  - Inputs x=[0, 0.2, 1, 0]
- What is the output?

Perceptron - Example

$$z = b + \Sigma_0^i w_i x_i$$

$$z > 0 \rightarrow y = 1$$
$$z \leq 0 \rightarrow y = 0$$

- Assume
  - weights w=[2, 4, 8, 3], bias b = 0
  - Inputs x=[0, 0.2, 1, 0]
- What is the output?
  - z = 0 + 0.8 + 8 + 0 + 0 = 8.8

Perceptron - Example

$$z = b + \Sigma_0^i w_i x_i$$

$$z > 0 \rightarrow y = 1$$
$$z \leq 0 \rightarrow y = 0$$

- Assume
  - weights w=[2, 4, 8, 3], bias b = 0
  - Inputs x=[0, 0.2, 1, 0]
- What is the output?
  - z = 0 + 0.8 + 8 + 0 + 0 = 8.8
  - y = 1

Perceptrons for AND, OR, and XOR

- 2 binary inputs (1 or 0; TRUE or FALSE)
- Which parameters (w and b) give you
  - the AND function?
  - the OR function?
  - the XOR function?

AND

$$z = b + \Sigma_0^i w_i z_i \qquad \begin{array}{l} z > 0 \rightarrow y = 1 \\ z \leq 0 \rightarrow y = 0 \end{array}$$

- 2 binary inputs (1 or 0; TRUE or FALSE)
- We want:
  - [0, 0] -> 0
  - [0, 1] -> 0
  - [1, 0] -> 0
  - [1, 1] -> 1

$$z = b + \Sigma_0^i w_i z_i \qquad \begin{array}{l} z > 0 \rightarrow y = 1 \\ z \leq 0 \rightarrow y = 0 \end{array}$$

AND

- 2 binary inputs (1 or 0; TRUE or FALSE)
- We want:
  - [0, 0] -> 0
  - [0, 1] -> 0
  - [1, 0] -> 0
  - [1, 1] -> 1
- Possible answer: w = [1, 1]; b = -1

AND

$$z = b + \Sigma_0^i w_i z_i \qquad \begin{array}{l} z > 0 \rightarrow y = 1 \\ z \leq 0 \rightarrow y = 0 \end{array}$$

- 2 binary inputs (1 or 0; TRUE or FALSE)
- We want:
  - [0, 0] -> 0 $\quad b \leq 0$
  - [0, 1] -> 0 $\quad b + w_2 \leq 0$
  - [1, 0] -> 0 $\quad b + w_1 \leq 0$
  - [1, 1] -> 1 $\quad b + w_1 + w_2 > 0$
- Possible answer: w = [1, 1]; b = -1

$$z = b + \Sigma_0^i w_i z_i \qquad \begin{array}{l} z > 0 \rightarrow y = 1 \\ z \leq 0 \rightarrow y = 0 \end{array}$$

AND

- 2 binary inputs (1 or 0; TRUE or FALSE)

$$\begin{aligned} b &\leq 0 \\ b + w_2 &\leq 0 \\ b + w_1 &\leq 0 \\ b + w_1 + w_2 &> 0 \end{aligned}$$

AND

$$z = b + \Sigma_0^i w_i z_i \qquad \begin{array}{l} z > 0 \rightarrow y = 1 \\ z \leq 0 \rightarrow y = 0 \end{array}$$

- 2 binary inputs (1 or 0; TRUE or FALSE)

$$\begin{aligned} b &\leq 0 \\ b + w_2 &\leq 0 \\ b + w_1 &\leq 0 \\ b + w_1 + w_2 &> 0 \end{aligned}$$

Set $b = -1.$

$$\begin{aligned} b &\leq 0 \\ w_2 &\leq 1 \\ w_1 &\leq 1 \\ w_1 + w_2 &> 1 \end{aligned}$$

$$z = b + \Sigma_0^i w_i z_i \qquad \begin{array}{l} z > 0 \rightarrow y = 1 \\ z \leq 0 \rightarrow y = 0 \end{array}$$

AND

- 2 binary inputs (1 or 0; TRUE or FALSE)

$$\begin{aligned} b &\leq 0 \\ b + w_2 &\leq 0 \\ b + w_1 &\leq 0 \\ b + w_1 + w_2 &> 0 \end{aligned}$$

Set $b = -1$.

$$\begin{aligned} b &\leq 0 \\ w_2 &\leq 1 \\ w_1 &\leq 1 \\ w_1 + w_2 &> 1 \end{aligned}$$

$$1 \geq w_1 > 1 - w_2 \geq 0$$

$$z = b + \Sigma_0^i w_i z_i \qquad \begin{array}{l} z > 0 \rightarrow y = 1 \\ z \leq 0 \rightarrow y = 0 \end{array}$$

AND

- 2 binary inputs (1 or 0; TRUE or FALSE)

$$\begin{aligned} b &\leq 0 \\ b + w_2 &\leq 0 \\ b + w_1 &\leq 0 \\ b + w_1 + w_2 &> 0 \end{aligned}$$

Set $b = -1.$

$$\begin{aligned} b &\leq 0 \\ w_2 &\leq 1 \\ w_1 &\leq 1 \\ w_1 + w_2 &> 1 \end{aligned}$$

Set $w_1 = 1.$

$$1 \geq w_2 > 1 - 1 = 0$$

$$1 \geq w_1 > 1 - w_2 \geq 0$$

$$z = b + \Sigma_0^i w_i z_i \qquad \begin{array}{l} z > 0 \rightarrow y = 1 \\ z \leq 0 \rightarrow y = 0 \end{array}$$

AND

- 2 binary inputs (1 or 0; TRUE or FALSE)

$$\begin{aligned} b &\leq 0 \\ b + w_2 &\leq 0 \\ b + w_1 &\leq 0 \\ b + w_1 + w_2 &> 0 \end{aligned}$$

Set $b = -1.$

$$\begin{aligned} b &\leq 0 \\ w_2 &\leq 1 \\ w_1 &\leq 1 \\ w_1 + w_2 &> 1 \end{aligned}$$

Set $w_1 = 1.$

$$1 \geq w_2 > 1 - 1 = 0$$

Set $w_2 = 1.$

$$1 \geq w_1 > 1 - w_2 \geq 0$$

In-class Exercise

- 2 binary inputs (1 or 0; TRUE or FALSE)
- Find parameters for
  - the OR function
  - the XOR function

OR			XOR		
x1	x2	y	x1	x2	y
0	0	0	0	0	0
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

XOR Function

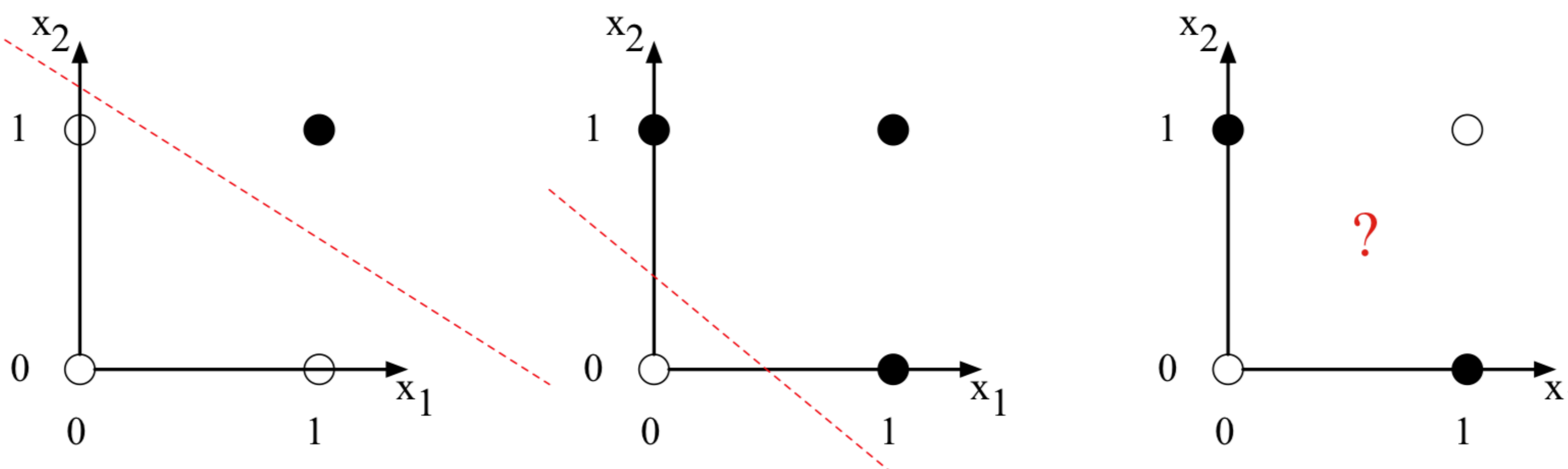
- The perceptron cannot compute XOR
  - The perceptron is a linear classifier.
  - Decision boundary is a line.

$$w_1 x_1 + w_2 x_2 + b = 0$$

$$\rightarrow x_2 = -\frac{w_1}{w_2} x_1 - b$$

XOR Function

- XOR is not linearly separable

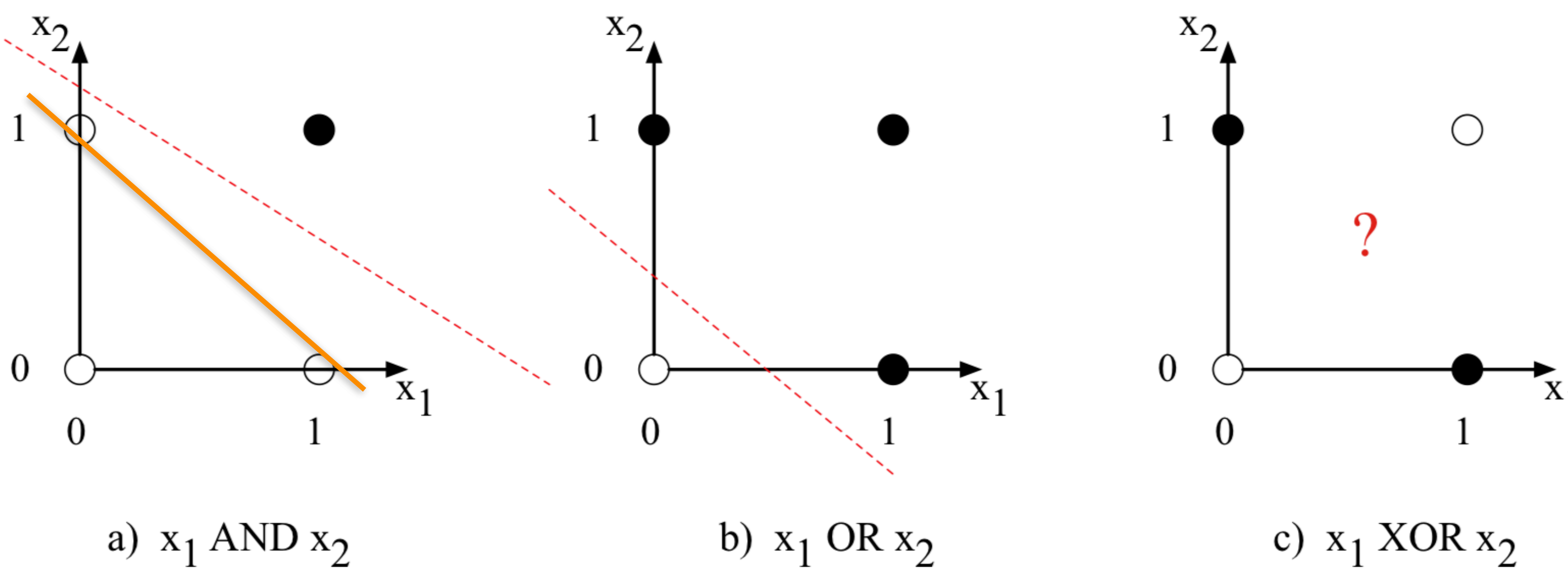


a) x_1 AND x_2 b) x_1 OR x_2 c) x_1 XOR x_2

J&M, Ch. 7

XOR Function

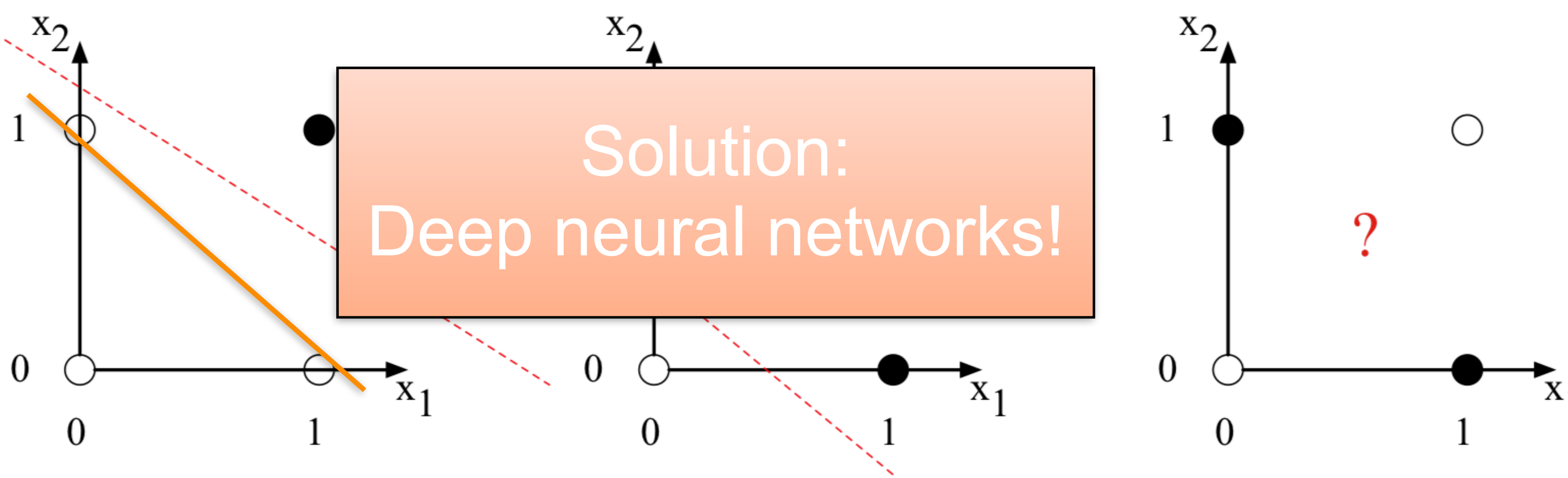
- XOR is not linearly separable



J&M, Ch. 7

XOR Function

- XOR is not linearly separable



Solution:
Deep neural networks!

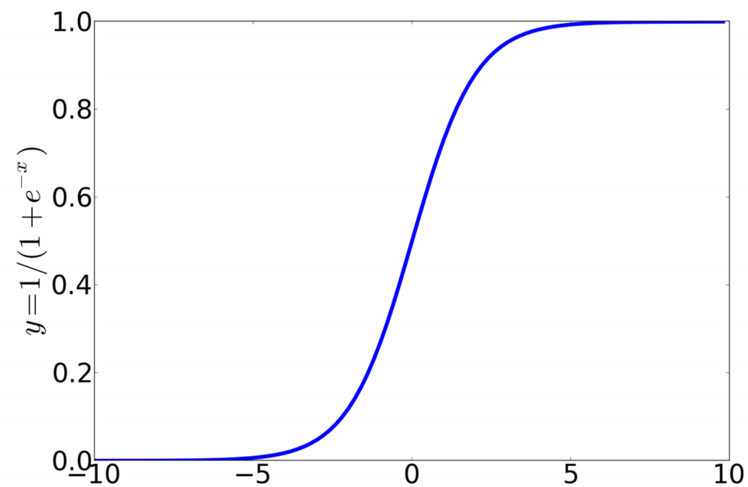
a) x_1 AND x_2 b) x_1 OR x_2 c) x_1 XOR x_2

J&M, Ch. 7

Non-linear output functions

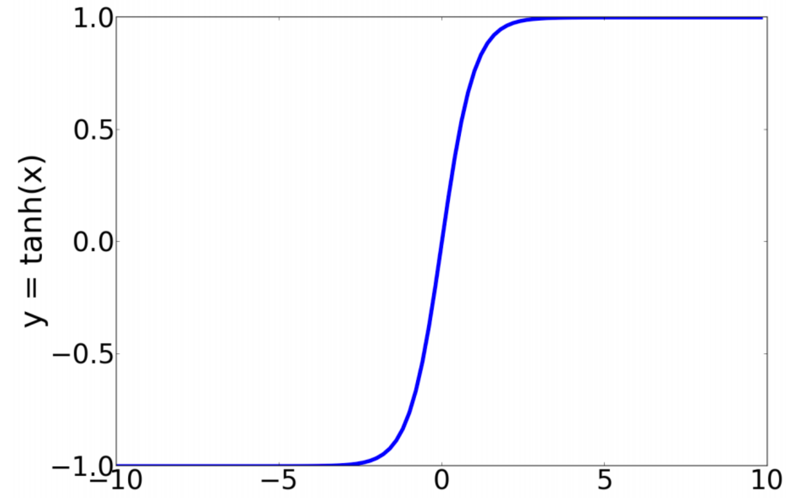
Sigmoid Function

- Maps output to a range between 0 and 1
  - Useful for outliers!
- Can represent probabilities.
- Is differentiable.
  - Handy for learning!



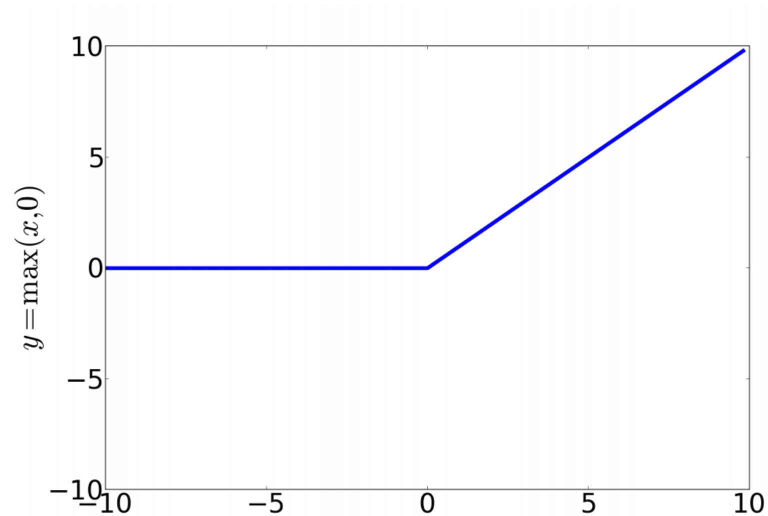
Tanh Function

- More commonly used as an activation function
- In practice often better results than sigmoid



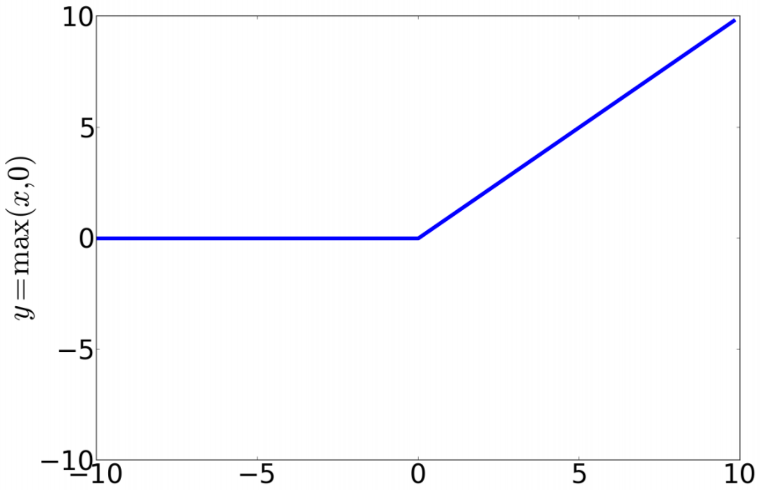
ReLU Function

- Simplest activation function
- Most commonly used
- Nearly linear



ReLU Function

- Simplest activation function
- Most commonly used
- Nearly linear



If in doubt, explore different options using your development set!

Feedforward networks

Feedforward Networks

- Also called multi-layer perceptrons.
- Multiple layers of perceptrons with non-linear functions after each layer.
- Can represent more complex decision boundaries.
    - (They can compute XOR!)

Feedforward Networks

$$k = b + \Sigma_0^i w_i z_i$$

$$l = \text{tanh}(k)$$

$$m = b + \Sigma_0^i w_i l_i$$

$$y = \text{sigmoid}(m)$$

Feedforward Networks

$$k = b + \Sigma_0^i w_i z_i$$

$$l = \tanh(k)$$

i = 1

$$m = b + \Sigma_0^i w_i l_i$$

$$y = \text{sigmoid}(m)$$

Feedforward Networks

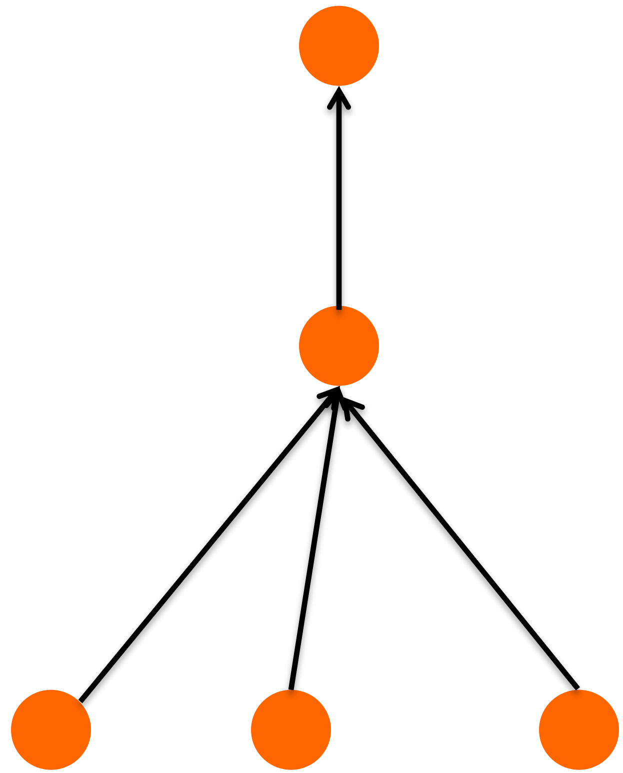
$$k = b + \Sigma_0^i w_i z_i$$

$$l = \tanh(k)$$

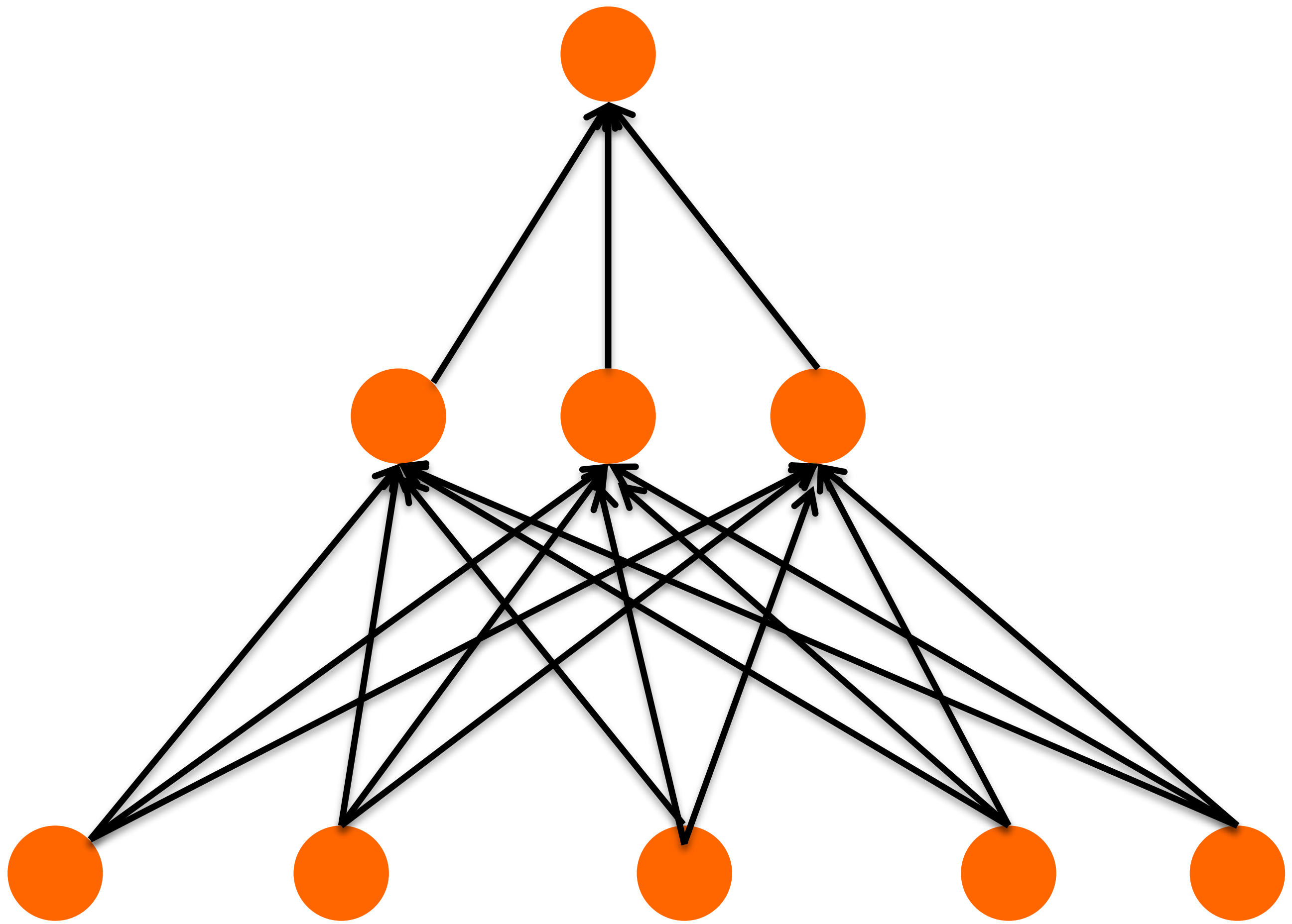
i = 1

$$m = b + \Sigma_0^i w_i l_i$$

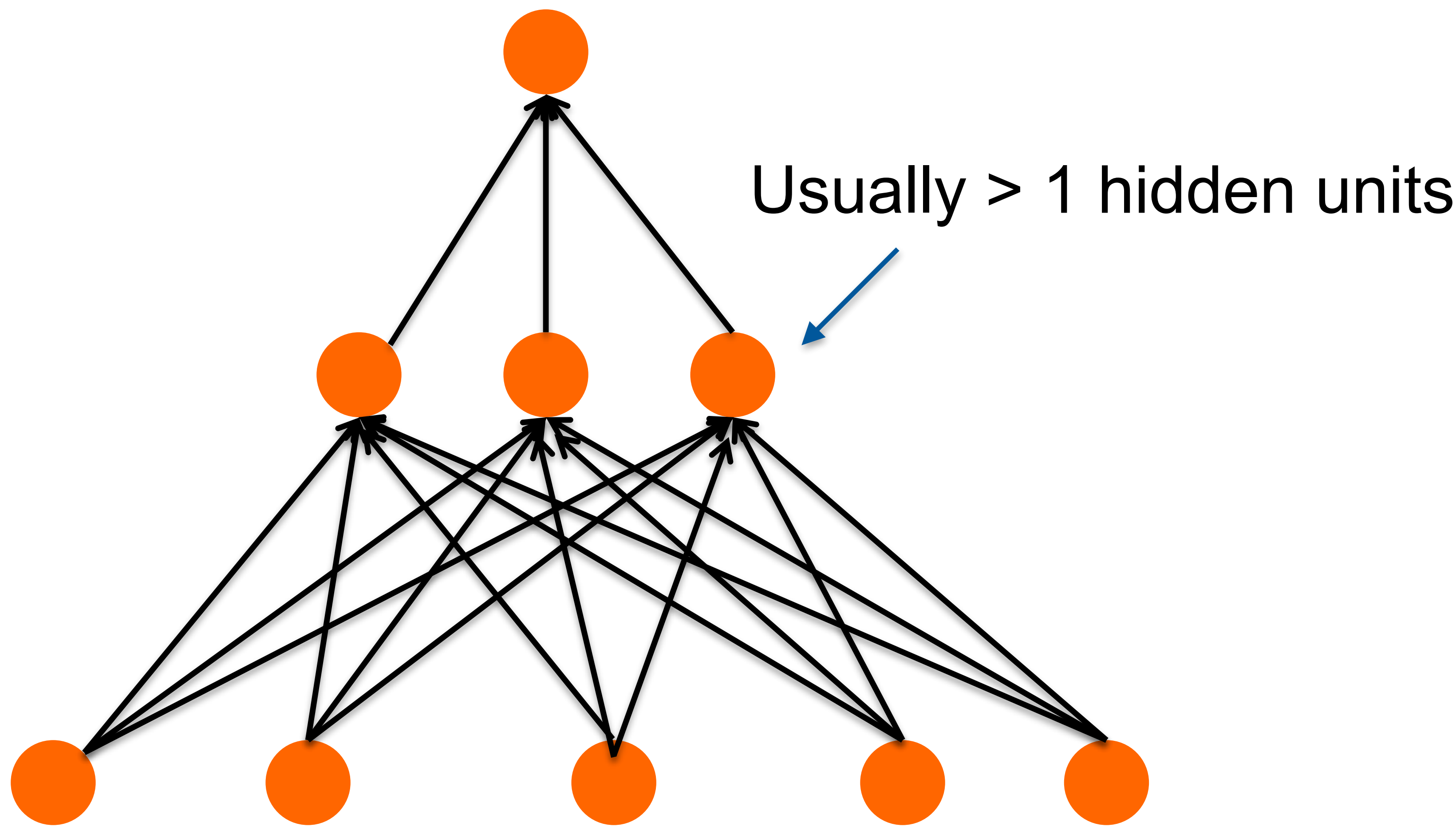
$$y = \text{sigmoid}(m)$$



Feedforward Networks



Feedforward Networks



Feedforward Network - Example

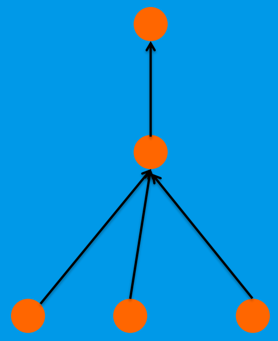
- Weights w_1=[2, 4, 8]
- Bias b_1 = 0
- Weights w_2 = [3]
- Bias b_2 = 1
- Inputs x=[0, 0.2, 1]
- What is the output?

$$k = b + \Sigma_0^i w_i z_i$$

$$l = \text{tanh}(k)$$

$$m = b + \Sigma_0^i w_i l_i$$

$$y = \text{sigmoid}(m)$$



Feedforward Network - Example

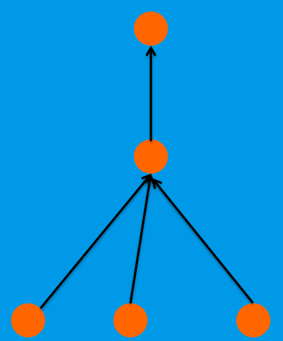
- Weights w_1=[2, 4, 8]
- Bias b_1 = 0
- Weights w_2 = [3]
- Bias b_2 = 1
- Inputs x=[0, 0.2, 1]
- What is the output?
  - k = 0 + 2*0 + 4*0.2 + 8*1 = 8.8

$$k = b + \Sigma_0^i w_i z_i$$

$$l = \tanh(k)$$

$$m = b + \Sigma_0^i w_i l_i$$

$$y = \text{sigmoid}(m)$$



Feedforward Network - Example

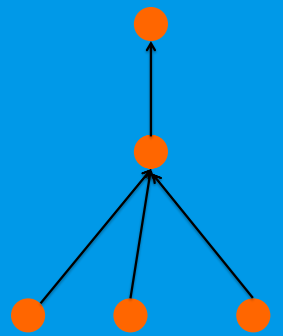
- Weights w_1=[2, 4, 8]
- Bias b_1 = 0
- Weights w_2 = [3]
- Bias b_2 = 1
- Inputs x=[0, 0.2, 1]
- What is the output?
  - k = 0 + 2*0 + 4*0.2 + 8*1 = 8.8
  - l = tanh(8.8) = 1.0

$$k = b + \Sigma_0^i w_i z_i$$

$$l = \text{tanh}(k)$$

$$m = b + \Sigma_0^i w_i l_i$$

$$y = \text{sigmoid}(m)$$



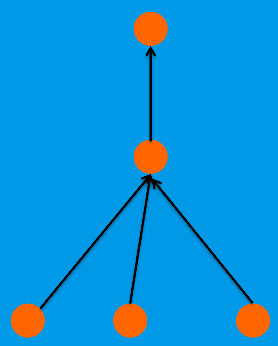
Feedforward Network - Example

$$k = b + \Sigma_0^i w_i z_i$$

$$l = \text{tanh}(k)$$

$$m = b + \Sigma_0^i w_i l_i$$

$$y = \text{sigmoid}(m)$$



- Weights w_1=[2, 4, 8]
- Bias b_1 = 0
- Weights w_2 = [3]
- Bias b_2 = 1
- Inputs x=[0, 0.2, 1]
- What is the output?
  - k = 0 + 2*0 + 4*0.2 + 8*1 = 8.8
  - l = tanh(8.8) = 1.0
  - m = 1 + 3*1 = 4

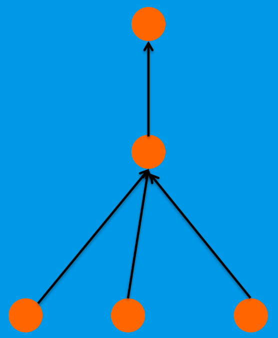
Feedforward Network - Example

$$k = b + \Sigma_0^i w_i z_i$$

$$l = \tanh(k)$$

$$m = b + \Sigma_0^i w_i l_i$$

$$y = \text{sigmoid}(m)$$



- Weights w_1=[2, 4, 8]
- Bias b_1 = 0
- Weights w_2 = [3]
- Bias b_2 = 1
- Inputs x=[0, 0.2, 1]
- What is the output?
  - k = 0 + 2*0 + 4*0.2 + 8*1 = 8.8
  - l = tanh(8.8) = 1.0
  - m = 1 + 3*1 = 4
  - y = sigmoid(4) = 0.98

Multi-Class Outputs

- What if you have more than two output classes?
  - Add more output units (one for each class)
  - Use a softmax layer:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \le i \le D$$

Softmax

$$\text{softmax}(z_i) \;=\; \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq D$$

- Use for one-of-many predictions
- Gives you a probability distribution
  - Values are all between 0 and 1
  - Values add up to 1

Softmax

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq D$$

- Example:
  - 3 classes
  - z = [130, 34, 91]

Softmax

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \leq i \leq D$$

- Example:
  - 3 classes
  - z = [130, 34, 91]

- $softmax(z_i) = \dfrac{e^{130}}{e^{130} + e^{34} + e^{91}}$

Neural language models

Neural Language Models

- Back to language modeling: Calculating the probability of the next word in a sequence given some history.

$$P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$$

Neural Language Models

- Back to language modeling: Calculating the probability of the next word in a sequence given some history.
  $P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$
- How to handle variable lengths?

Neural Language Models

- Back to language modeling: Calculating the probability of the next word in a sequence given some history.
  $P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$
- How to handle variable lengths?
  - Sliding windows (of fixed length)
  - Recurrent neural networks

Neural Language Models

- Back to language modeling: Calculating the probability of the next word in a sequence given some history.
  $P(w_t|w_1^{t-1}) \approx P(w_t|w_{t-N+1}^{t-1})$
- How to handle variable lengths?
  - Sliding windows (of fixed length)
  - Recurrent neural networks

Issues

- How do we represent our input?

Issues

- How do we represent our input?
- Option 1: one-hot vectors
  - Potentially really large (depending on the vocabulary size)
  - No information

Issues

- How do we represent our input?
- Option 1: one-hot vectors
    - Potentially really large (depending on the vocabulary size)
    - No information
- Option 2: word embeddings
    - Dense vectors, so not too many weight parameters needed
    - Pretrained embeddings already contain useful information!

Issues

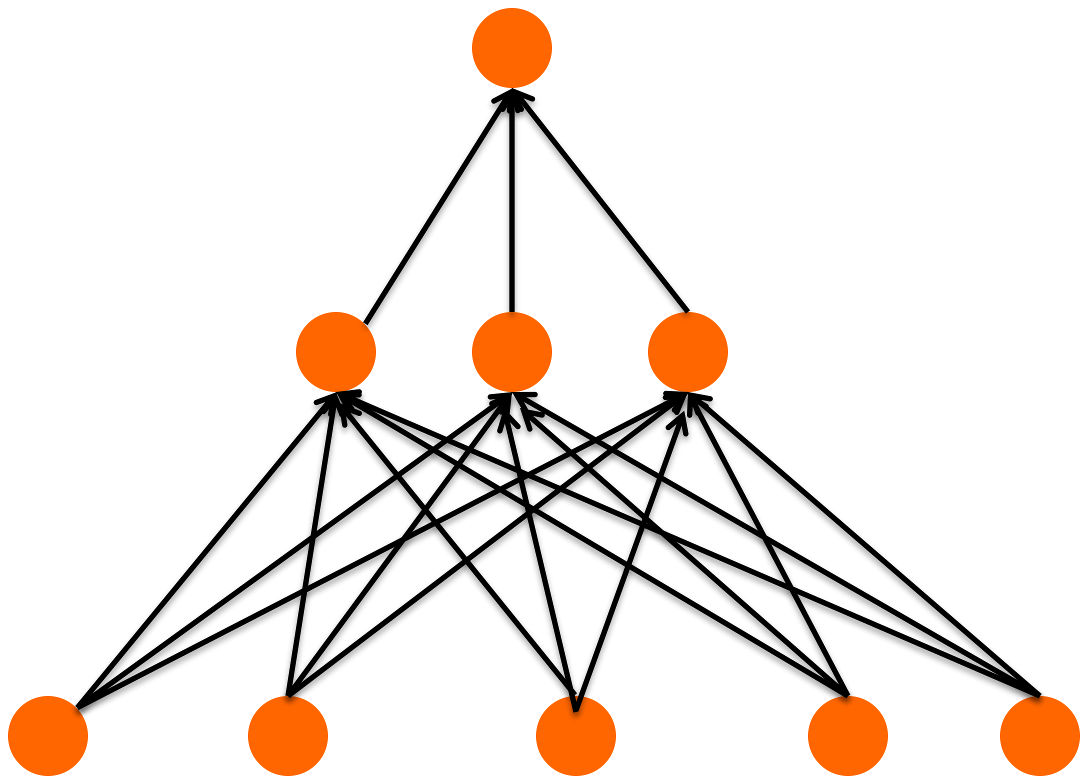
- How many possible classes are there?

Issues

- How many possible classes are there?
- |V|: one for each word in the vocabulary
    - Training takes forever.
    - So does forward inference.
    - In most applications we don’t need the full distribution, just the probability of a small candidate set.
    - Only use the most frequent words!

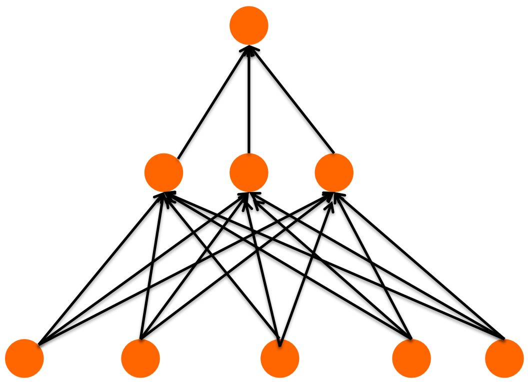
On the Number of Parameters

- How many parameters does the following network contain?



On the Number of Parameters

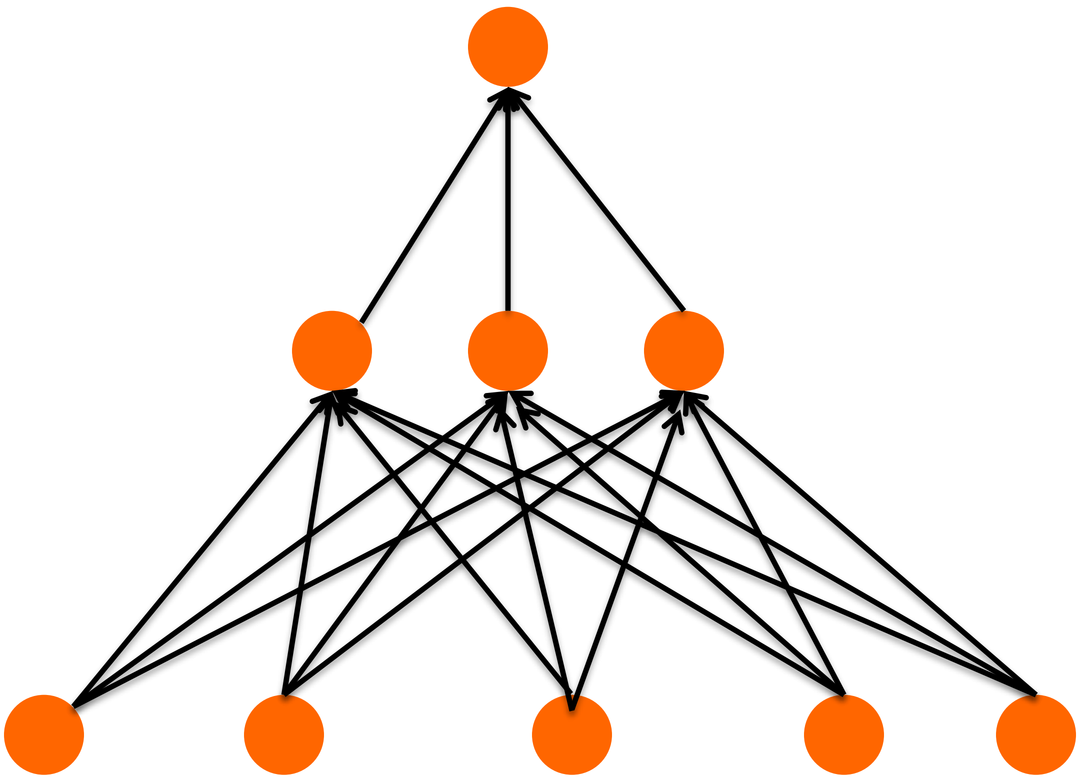
- How many parameters does the following network contain?



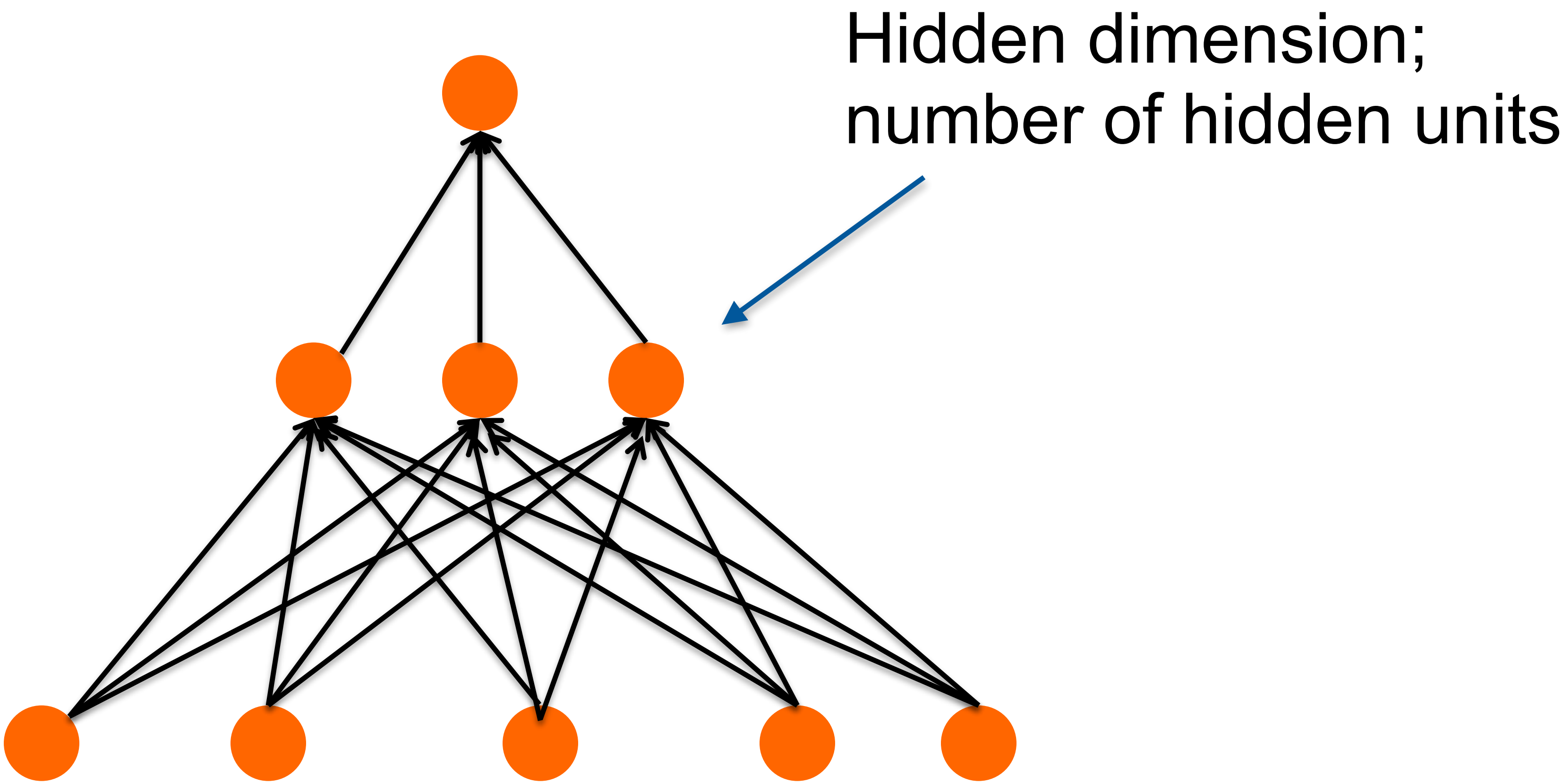
- One for each connection.
- One bias for each neuron.
- 5*3 + 3 for the first layer: 18
- 3*1 + 1 for the second layer: 4

- Total of 22 parameters. This is a tiny network!

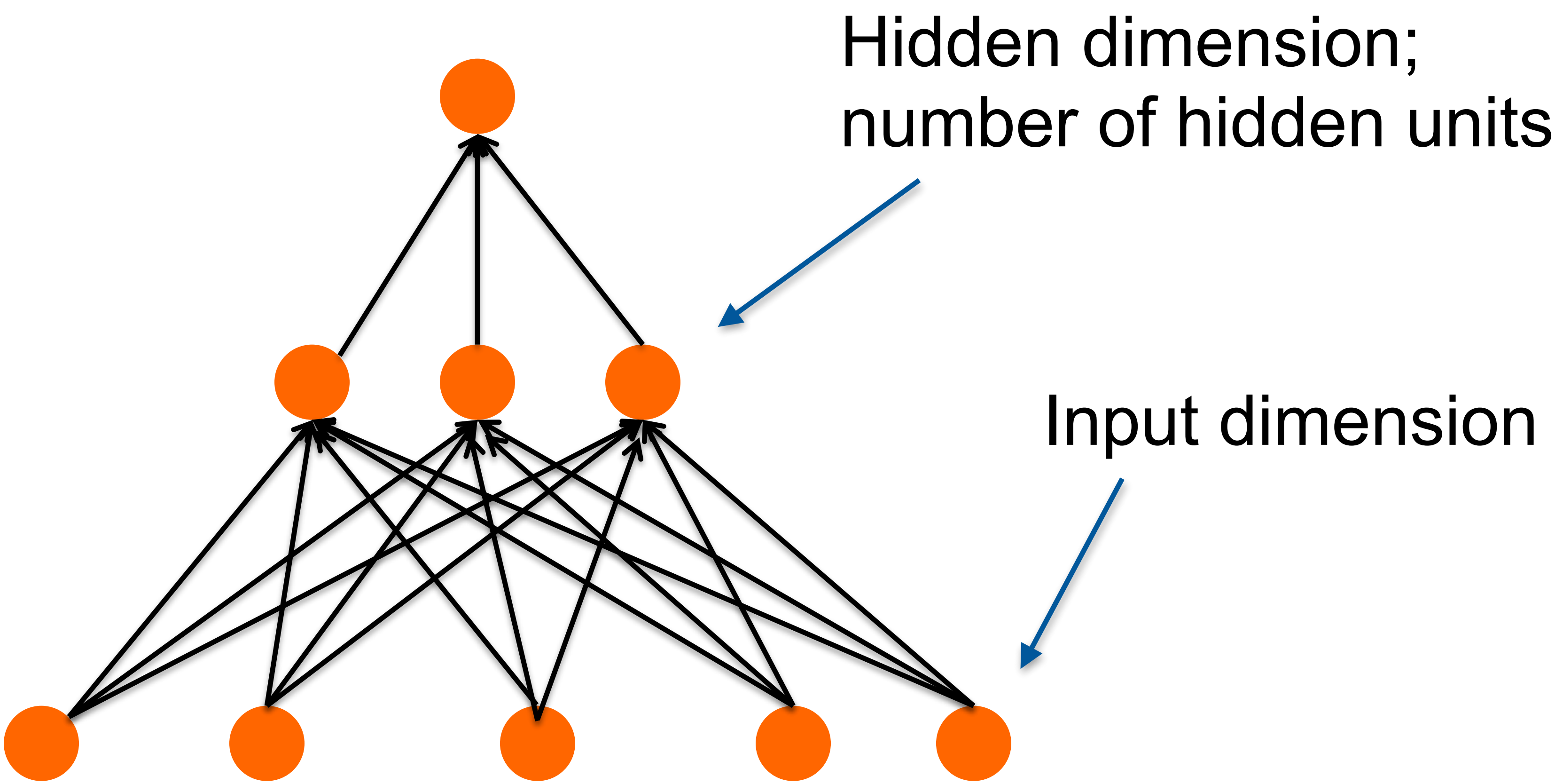
On Names



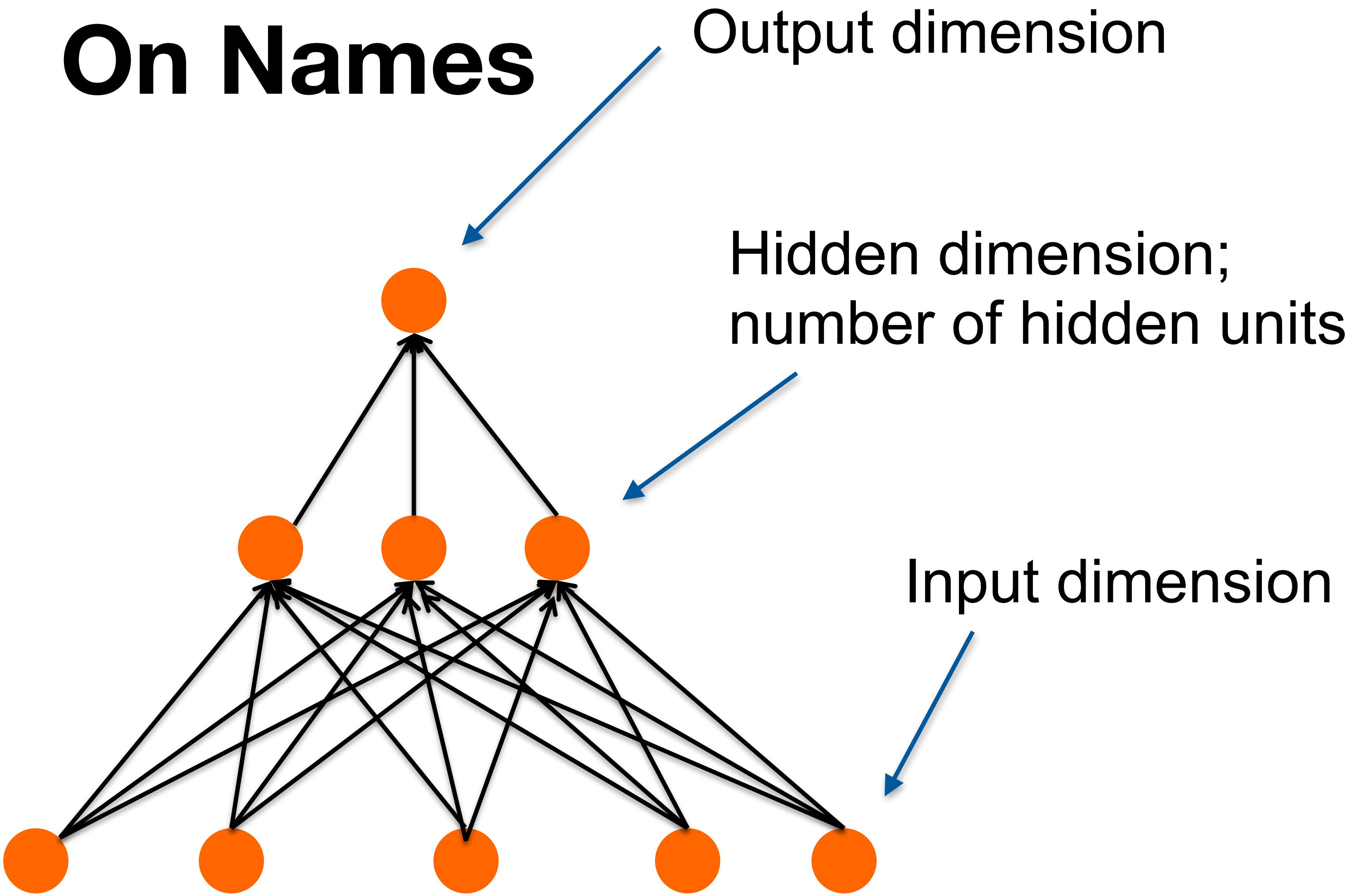
On Names



On Names



On Names


Output dimension
Hidden dimension;
number of hidden units
Input dimension

Back to Neural Language Models

- Our vocabulary is *Sam*, *likes*, *egg*, and *cheese*.
  - Plus <s> and </s>!
- How many dimensions do our (co-occurrence) word embeddings have?
- If we want to make a neural bigram language model, how many dimensions does our input have?

- How many layers do we use? How many dimensions each?
- How many dimensions does our output have?

Back to Neural Language Models

- Our vocabulary is *Sam*, *likes*, *egg*, and *cheese*.
  - Plus <s> and </s>!
- How many dimensions do our (co-occurrence) word embeddings have? (Answer: 6)
- If we want to make a neural bigram language model, how many dimensions does our input have?

- How many layers do we use? How many dimensions each?
- How many dimensions does our output have?

Back to Neural Language Models

- Our vocabulary is *Sam*, *likes*, *egg*, and *cheese*.
  - Plus <s> and </s>!
- How many dimensions do our (co-occurrence) word embeddings have? (Answer: 6)
- If we want to make a neural bigram language model, how many dimensions does our input have? (Answer: 6)
- How many layers do we use? How many dimensions each?
- How many dimensions does our output have?

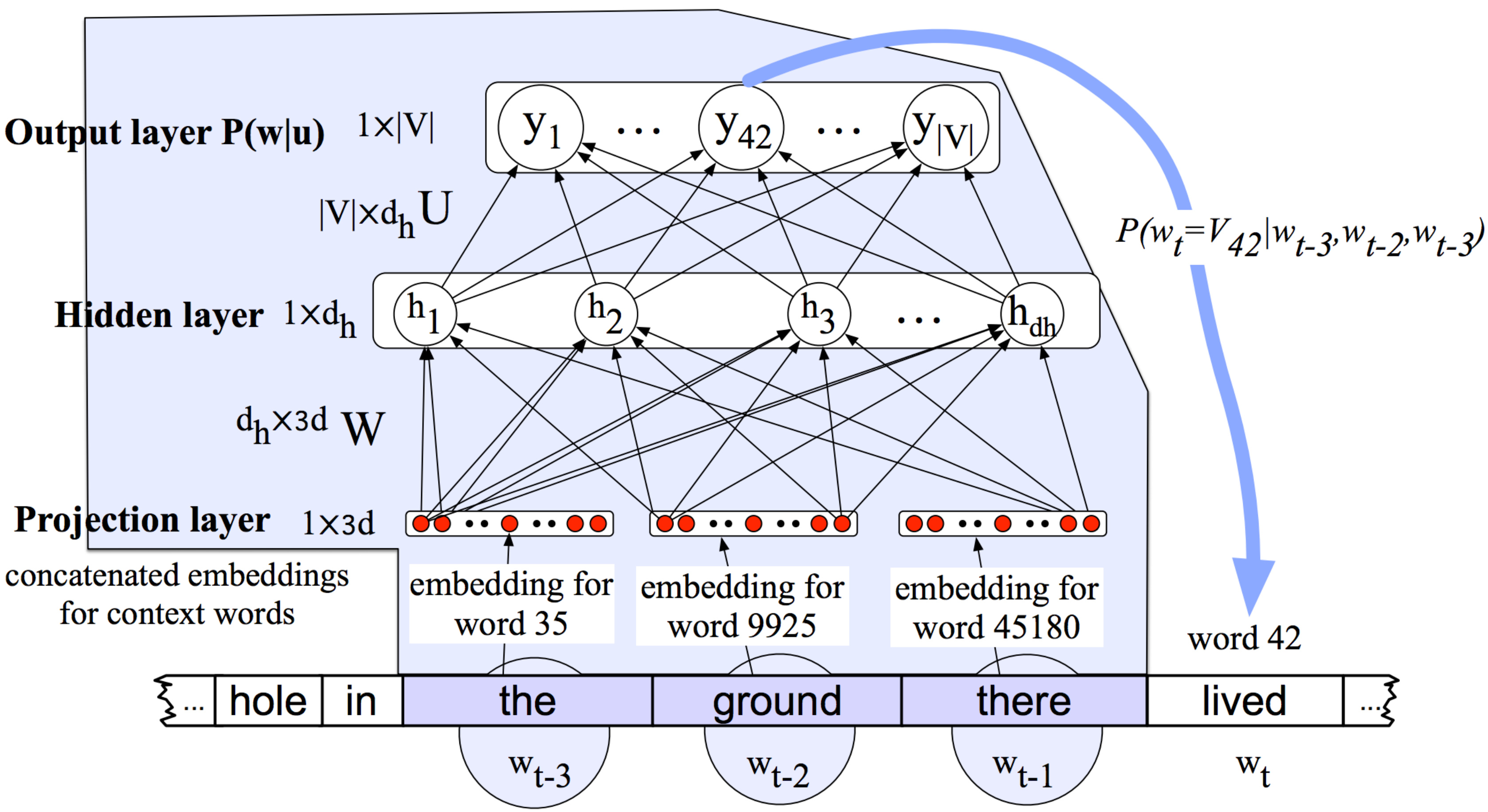
Back to Neural Language Models

- Our vocabulary is *Sam*, *likes*, *egg*, and *cheese*.
  - Plus <s> and </s>!
- How many dimensions do our (co-occurrence) word embeddings have? (Answer: 6)
- If we want to make a neural bigram language model, how many dimensions does our input have? (Answer: 6)
- How many layers do we use? How many dimensions each? (Answer: we can choose!)
- How many dimensions does our output have?

Back to Neural Language Models

- Our vocabulary is *Sam*, *likes*, *egg*, and *cheese*.
  - Plus <s> and </s>!
- How many dimensions do our (co-occurrence) word embeddings have? (Answer: 6)
- If we want to make a neural bigram language model, how many dimensions does our input have? (Answer: 6)
- How many layers do we use? How many dimensions each? (Answer: we can choose!)
- How many dimensions does our output have? (Answer: 6)

Back to Neural Language Models



Open Question

- Where do we get the network's weights from?
  - Will be discussed in some weeks!

Wrapping up

- Discussed today:
  - Evaluation of language models
  - Perceptrons
  - Feedforward networks
  - Neural language models based on feedforward networks
- Next Monday: Sentiment analysis: Feature-based methods