Neural Network Language Modeling

Yogarshi Vyas

02/24/16

LING773

Topics for Today

- Language Modeling recap
- Basics of Neural Networks
- Feed forward Neural Network Language Model
- Recurrent Neural Network Language Model
- Word Embeddings
- Word2Vec – Context Bag-of-Words and Skip-gram

Language Modeling Recap

- Assign a probability to a sentence
  - P (I went to the bank) >> P (I goes to the bank)

- Can be calculated using Chain Rule

- P(I went to the bank) = P(bank | I went to the) * P(the | I went to) * P(to | I went) * P(went | I) * P(I)

LM – Estimating Probabilities

- Count and divide

– P (bank | I went to the)
  = Count (I went to the bank) / Count (I went to the)

- But these estimates will be poor!
  – So we make Markov assumptions

- P (bank | I went to the) ≈ P (bank | to the)

  or

- P (bank | I went to the) ≈ P (bank | to)

Language Modeling Recap

- Assign a probability to a sentence
  - P (I went to the bank) >> P (Eye vent to the bank)

- Can be calculated using Chain Rule

- P(I went to the bank) = P(bank | I went to the) *
  P(the | I went to) *
  P(to | I went) *
  P(went | I) *
  P(I)

Language Modeling Recap

- Assign a probability to a sentence
  - P (I went to the bank) >> P (Eye vent to the bank)

- Can be calculated using Chain Rule

- P(I went to the bank) = P(bank | ~~I went~~ to the) *

  P(the | ~~I~~ went to) *

  P(to | I went) *

  P(went | I) *

  P(I)

Smoothing

- Zeros are bad!
- Give artificial counts to unseen words, by taking away counts from seen words
  - Laplace
  - Good Turing
  - Kneser Ney

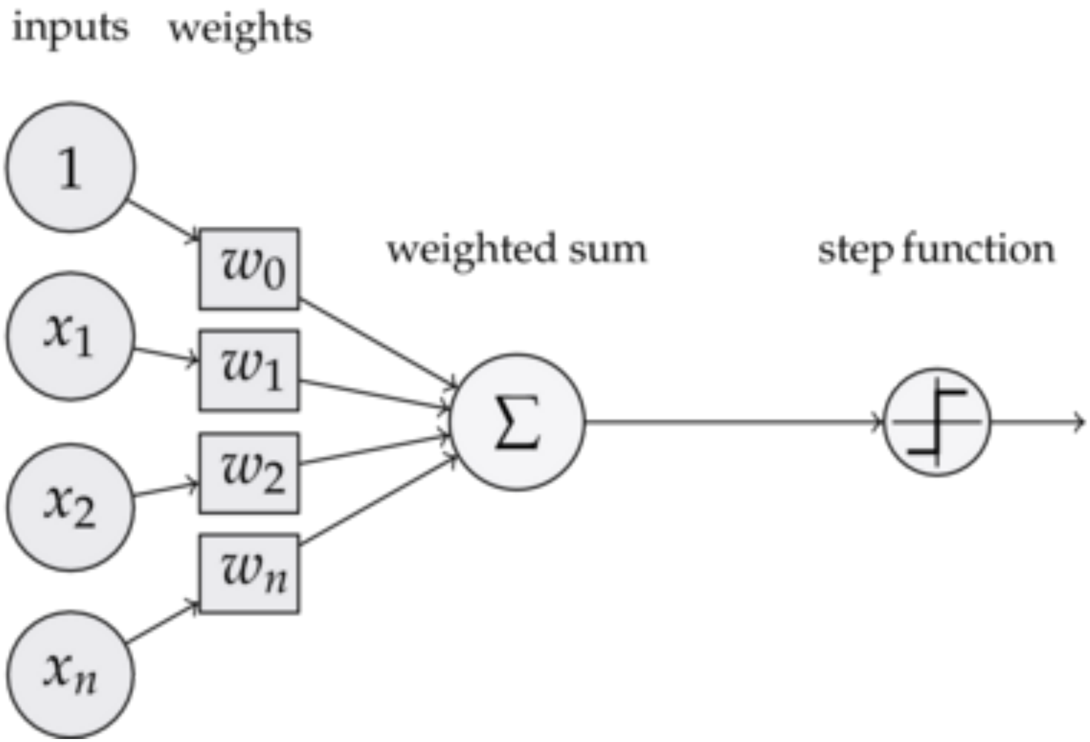
Issues with n-gram LMs

- State of the art applications of LMs (say in MT), use 4/5-grams.
  – Parameters grow exponentially with n.
- Similarity between words is not taken into account.
  – P("The cat is walking in the bedroom") should give some indication about P("A dog was running in a room")

Neural Net Recap

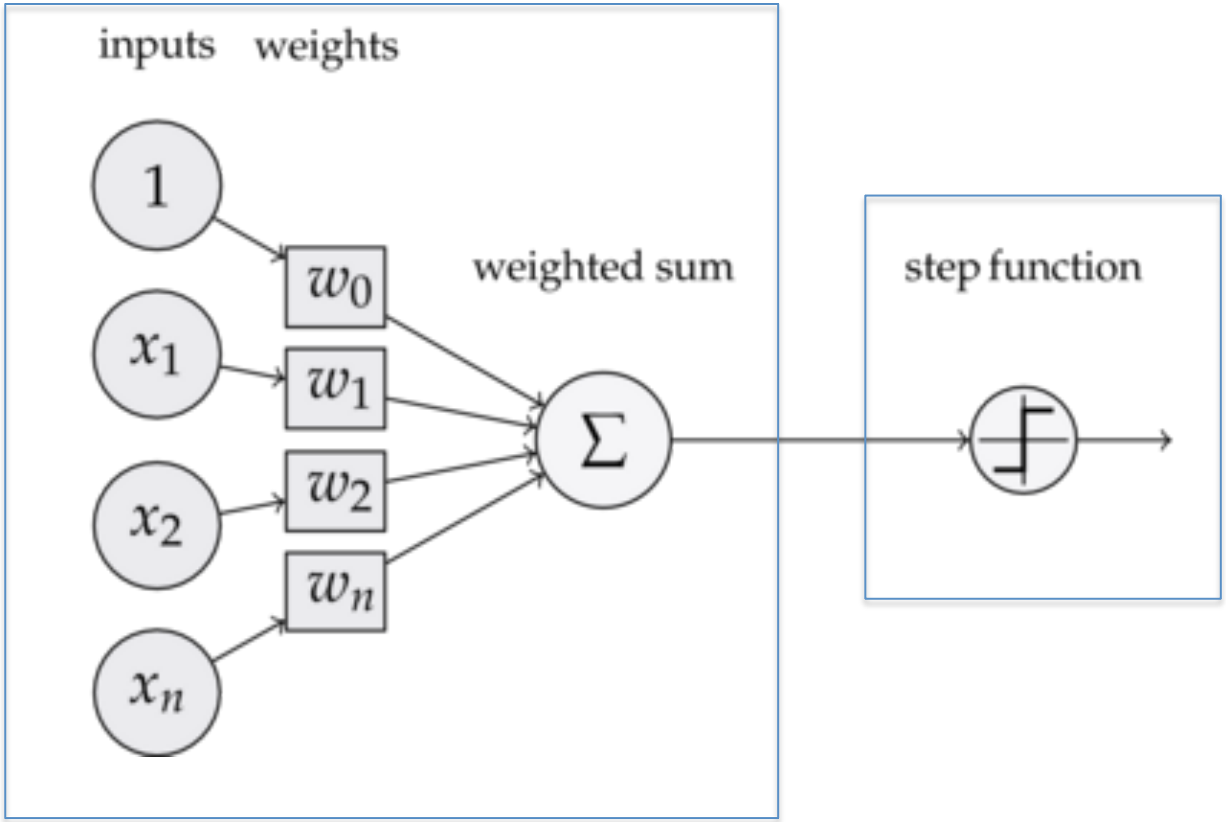
- Linear Classifier/Perceptron
- Sigmoid/Softmax
- Multilayer Perceptron / Feed forward NN
- Recurrent NN

Linear classifiers



$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Linear Classifiers

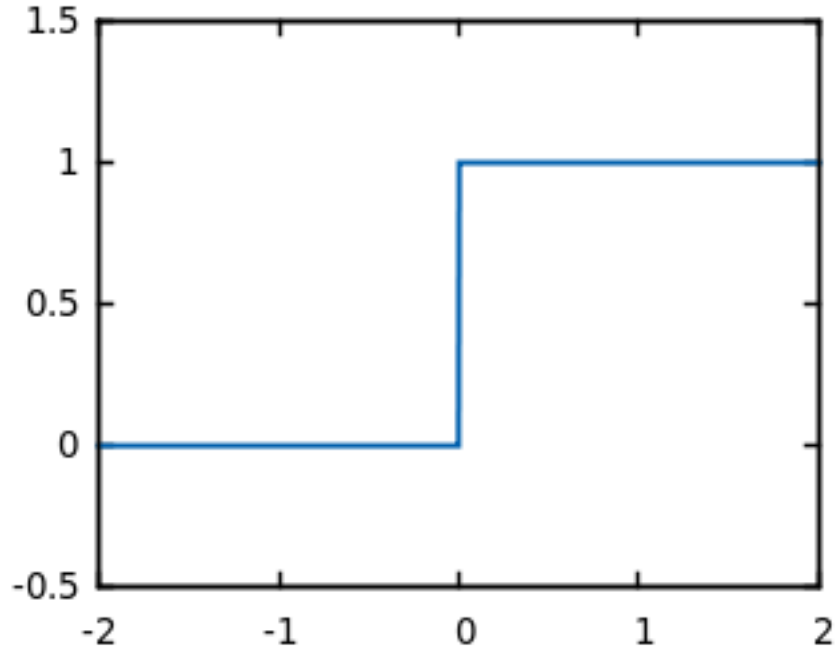


$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

Non linearity – Heavyside Step Function

$$f(x) = 0 \text{ if } x < 0$$
$$= 1 \text{ if } x > 0$$

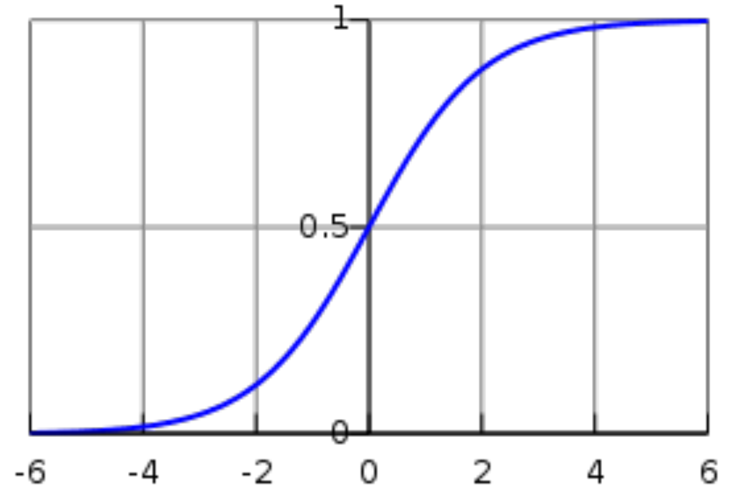
Output is either 0 or 1



Non linearity – Logistic Function

$f(x) = 1/(1 + e^{-x})$

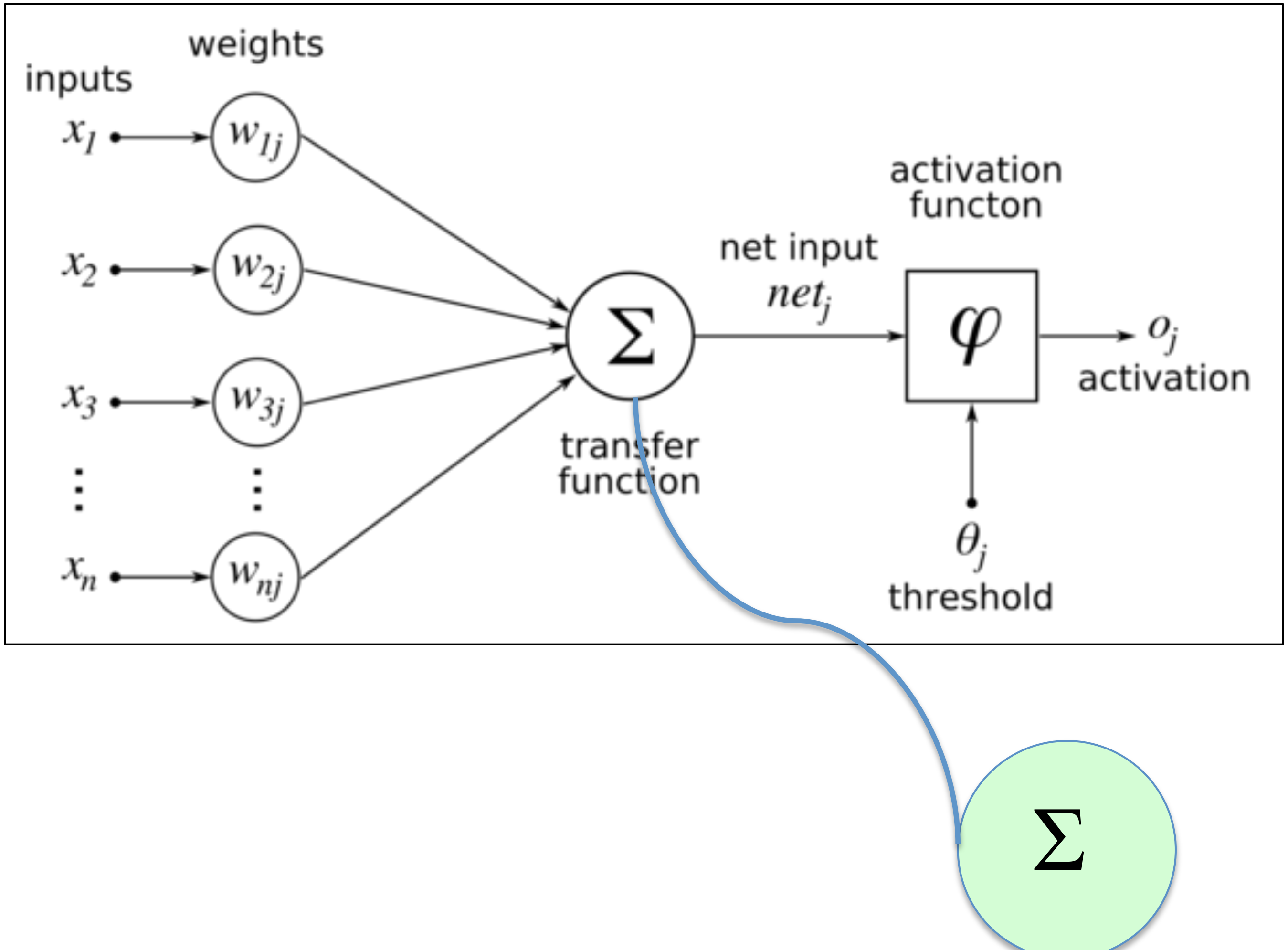
Output is a real value between 0 and 1



Softmax

- Multivariate version of Logistic Function
- Takes an arbitrary vector and squashes all values between 0 and 1 such that they sum to 1
- From a K-dimensional vector, you can get a K-class classifier

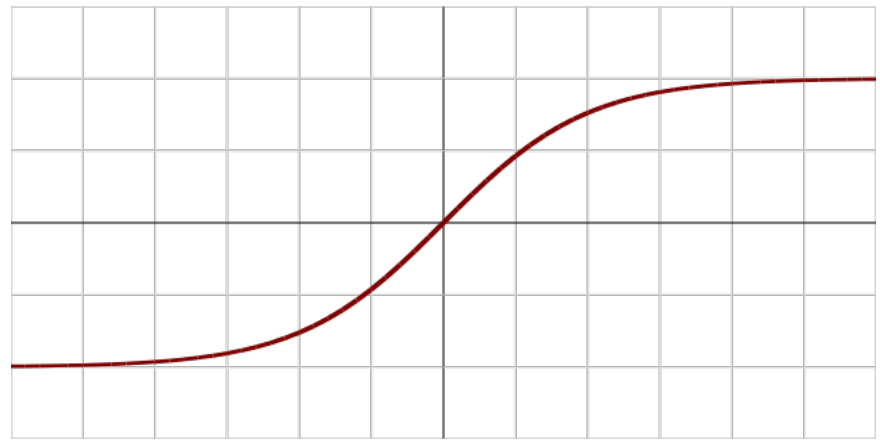
A neuron



weights
inputs
x_1
w_{1j}
x_2
w_{2j}
x_3
w_{3j}
x_n
w_{nj}
Σ
transfer
function
net input
net_j
activation
functon
φ
o_j
activation
θ_j
threshold
Σ

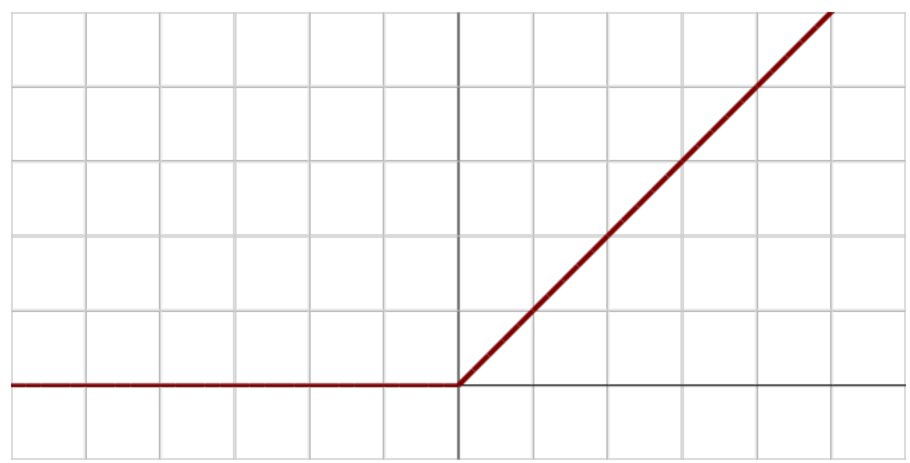
Other non-linearities

- Tanh

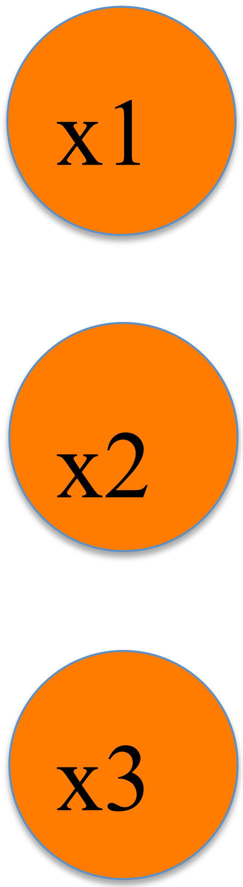


- ReLU

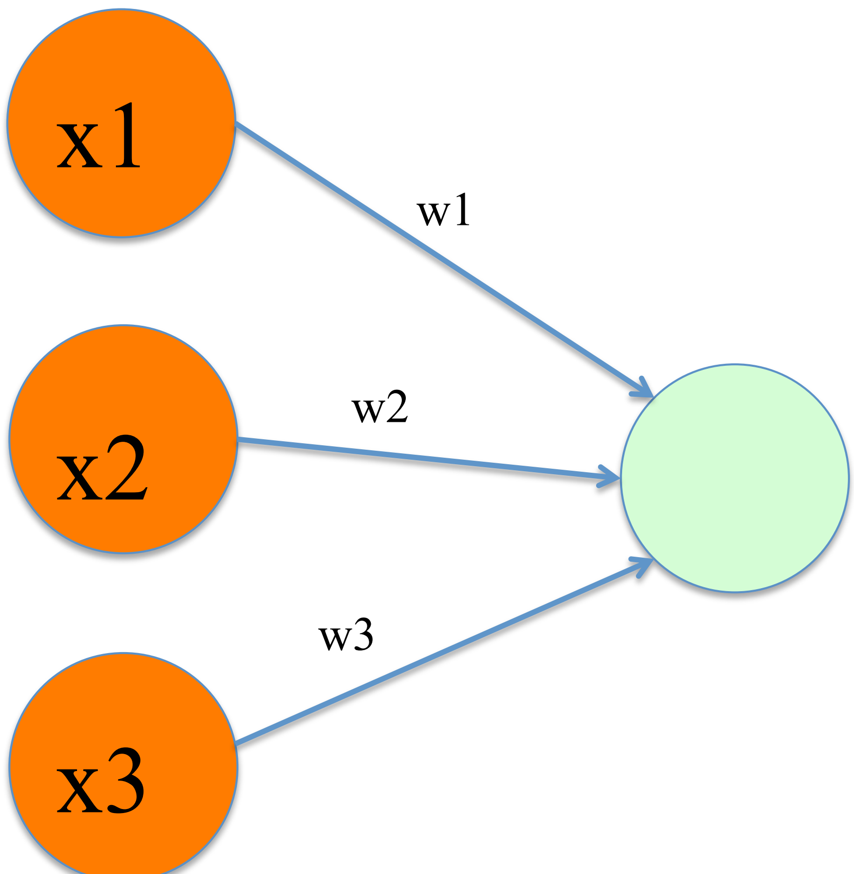
(Rectified Linear Unit)



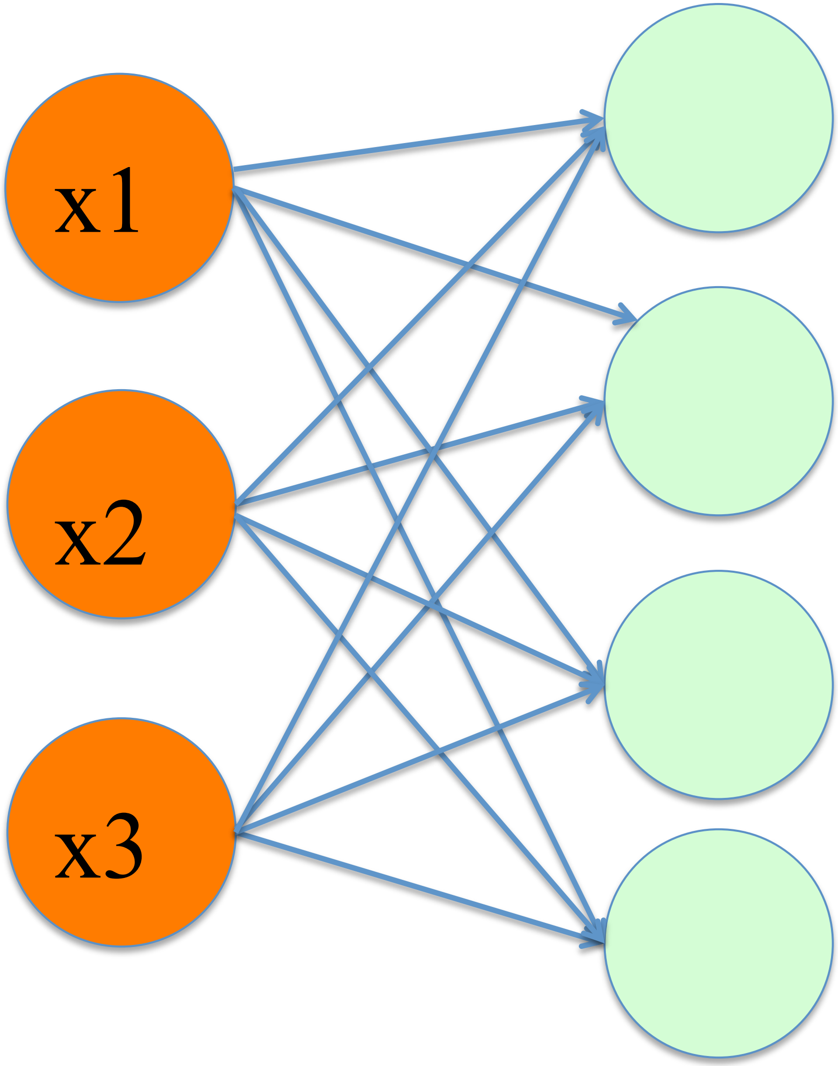
Stacking up neurons…



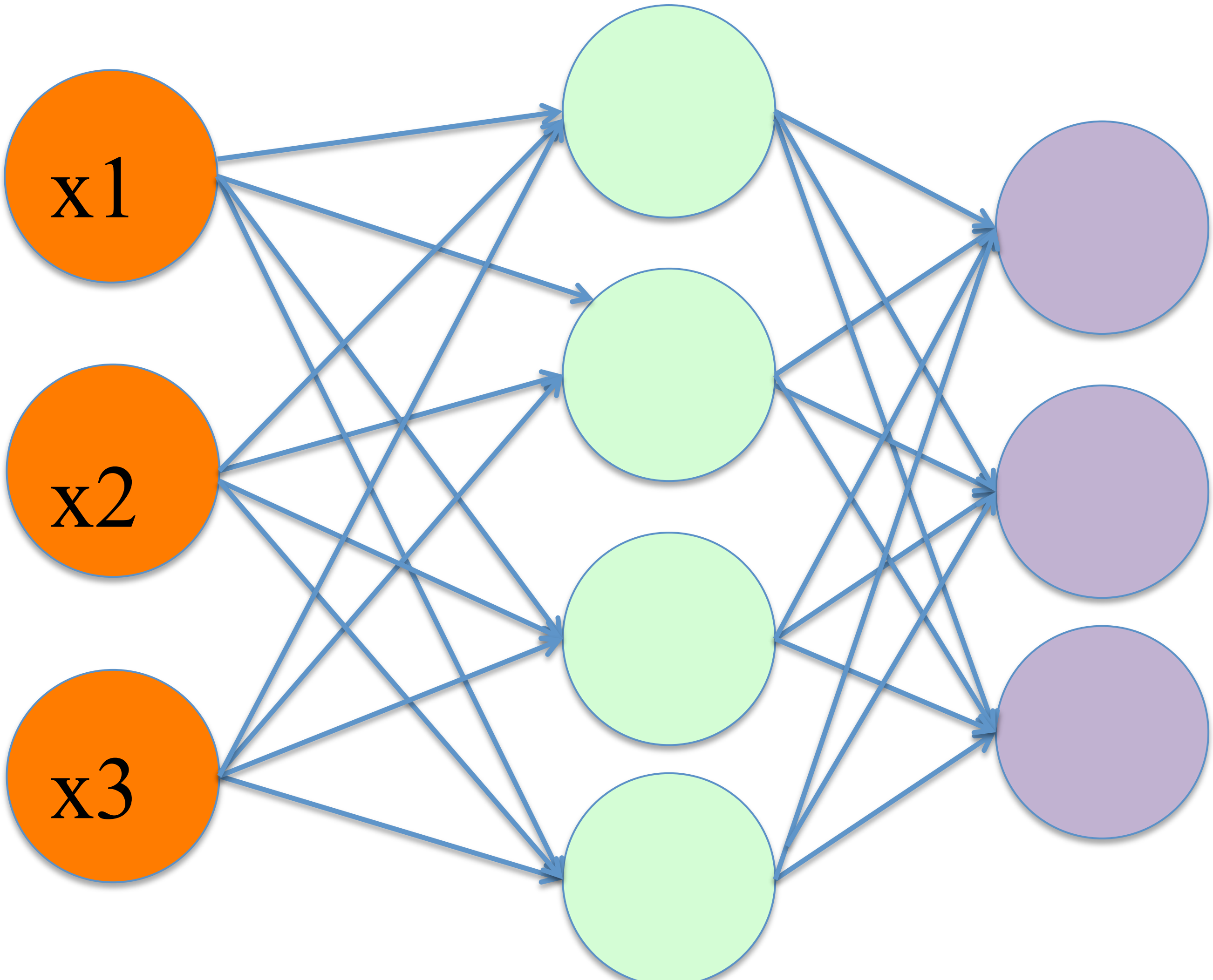
Stacking up neurons…



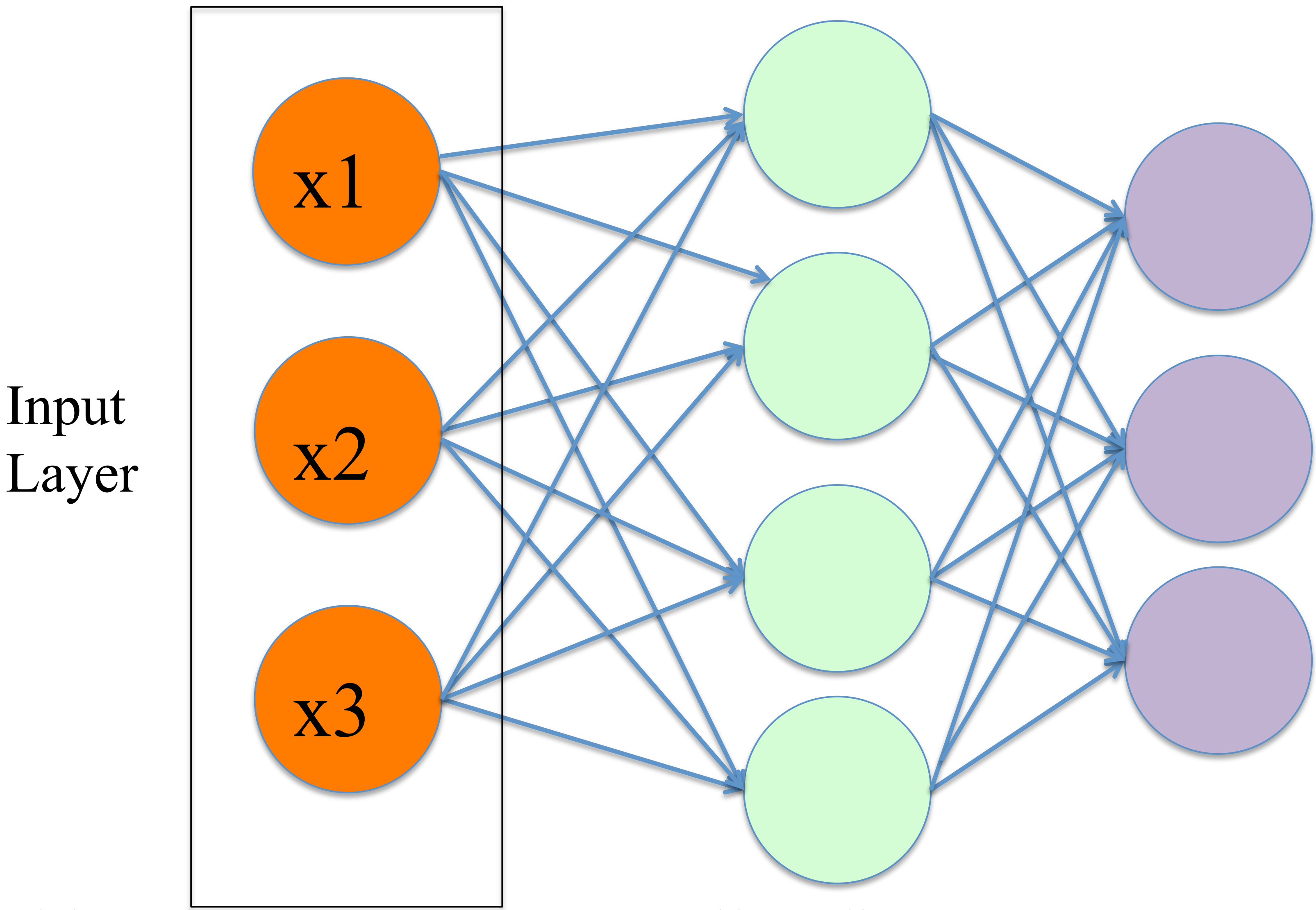
Stacking up neurons..



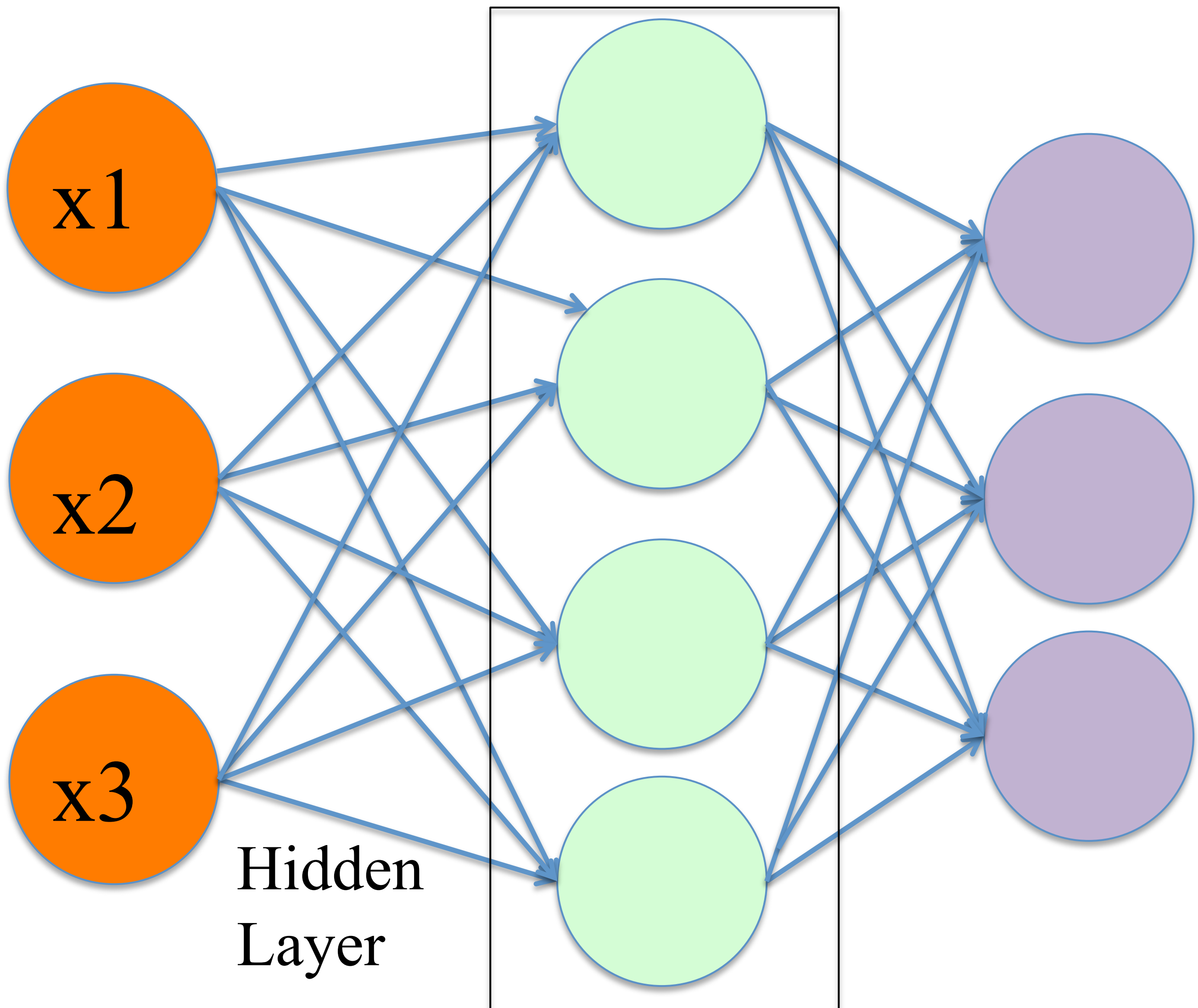
Feed-forward Neural Network!



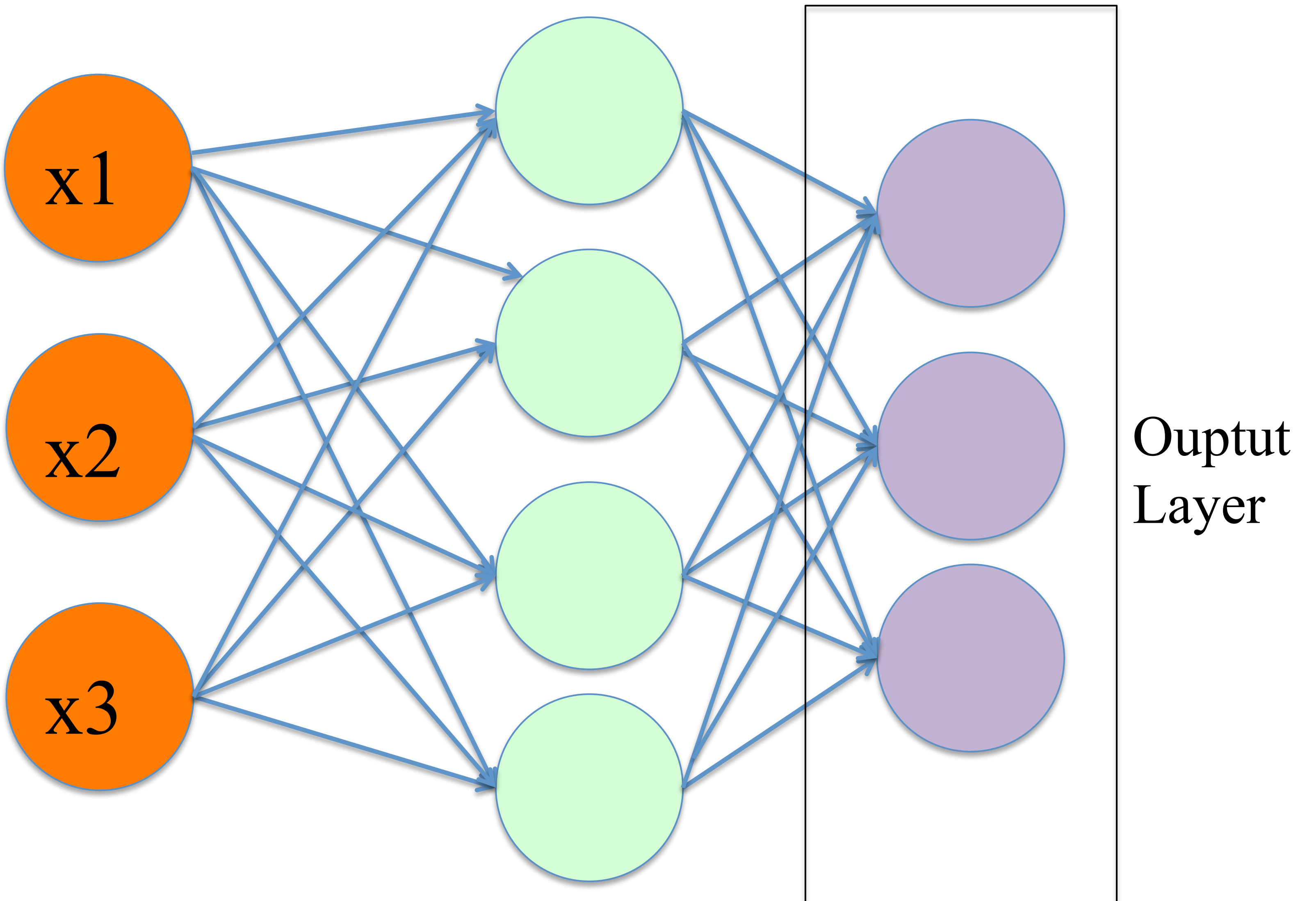
Feed-forward Neural Network!



Feed-forward Neural Network!



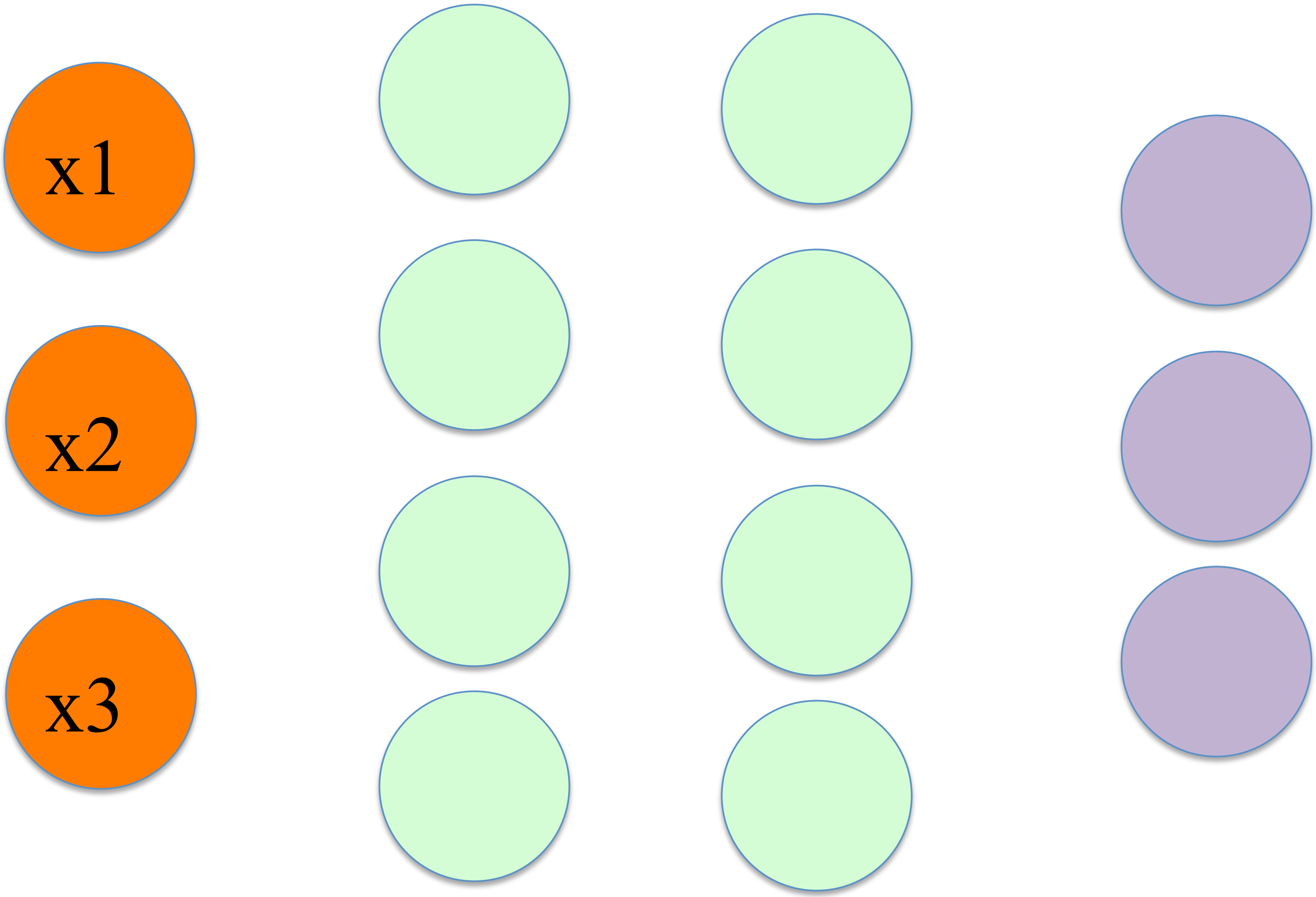
Feed-forward Neural Network!



Training a Neural Net

- What are the parameters of a neural net?
  - Architecture (Number of layers, size, etc.)
  - Activation function
  - Input features
  - **Weight**
- Train weights using **Backpropagation**
  - Gradient descent + Chain Rule
  - Loss functions – Cross entropy, Log likelihood

Adding layers – making it deep

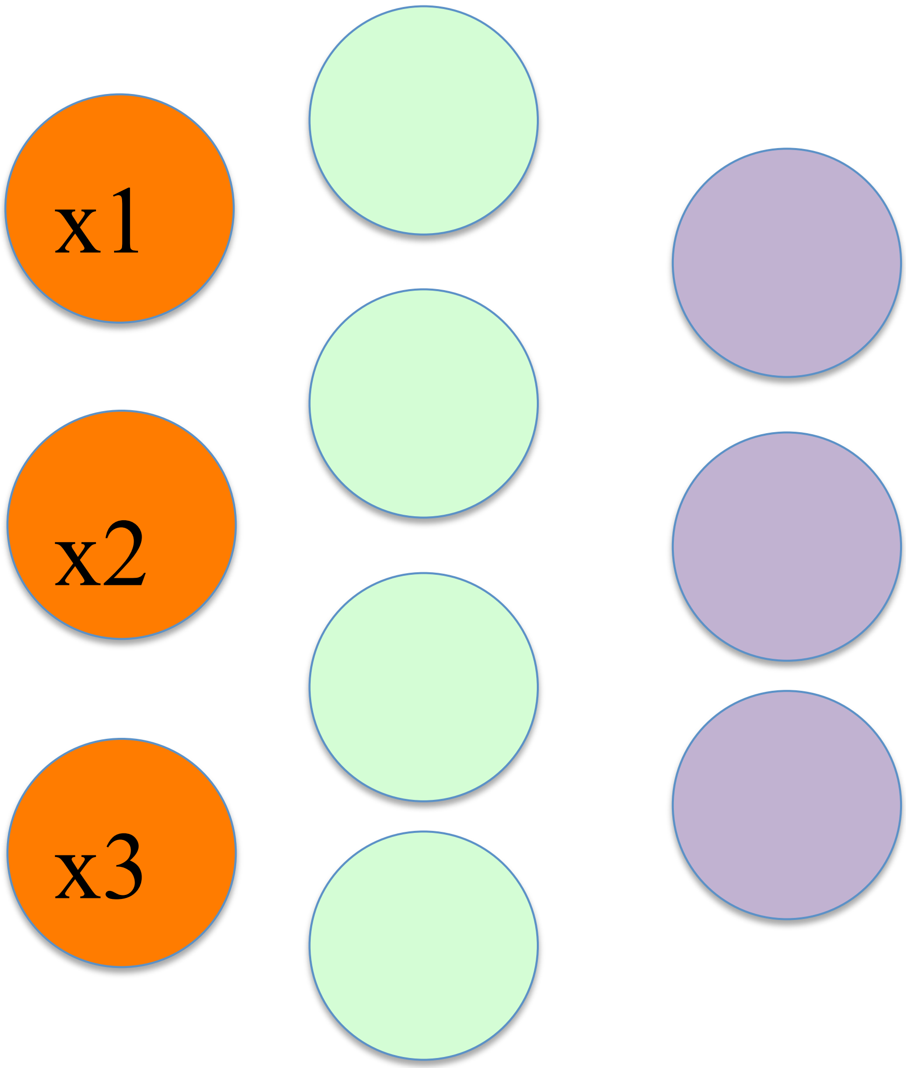


Matrix multiplication view

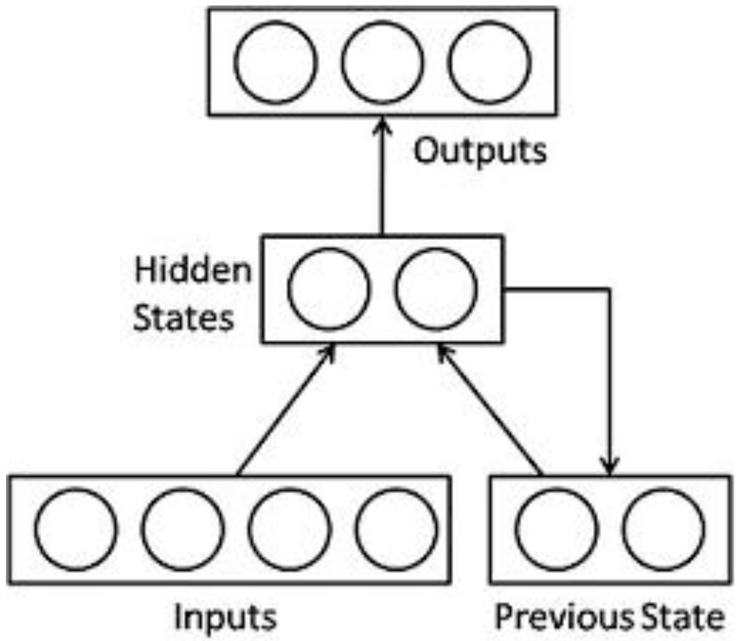
- Input
  3d vector **x**

- Hidden layer
  $\mathbf{y} = f_1(\mathbf{W_1 x})$
  $\mathbf{W_1}$ is 3x4

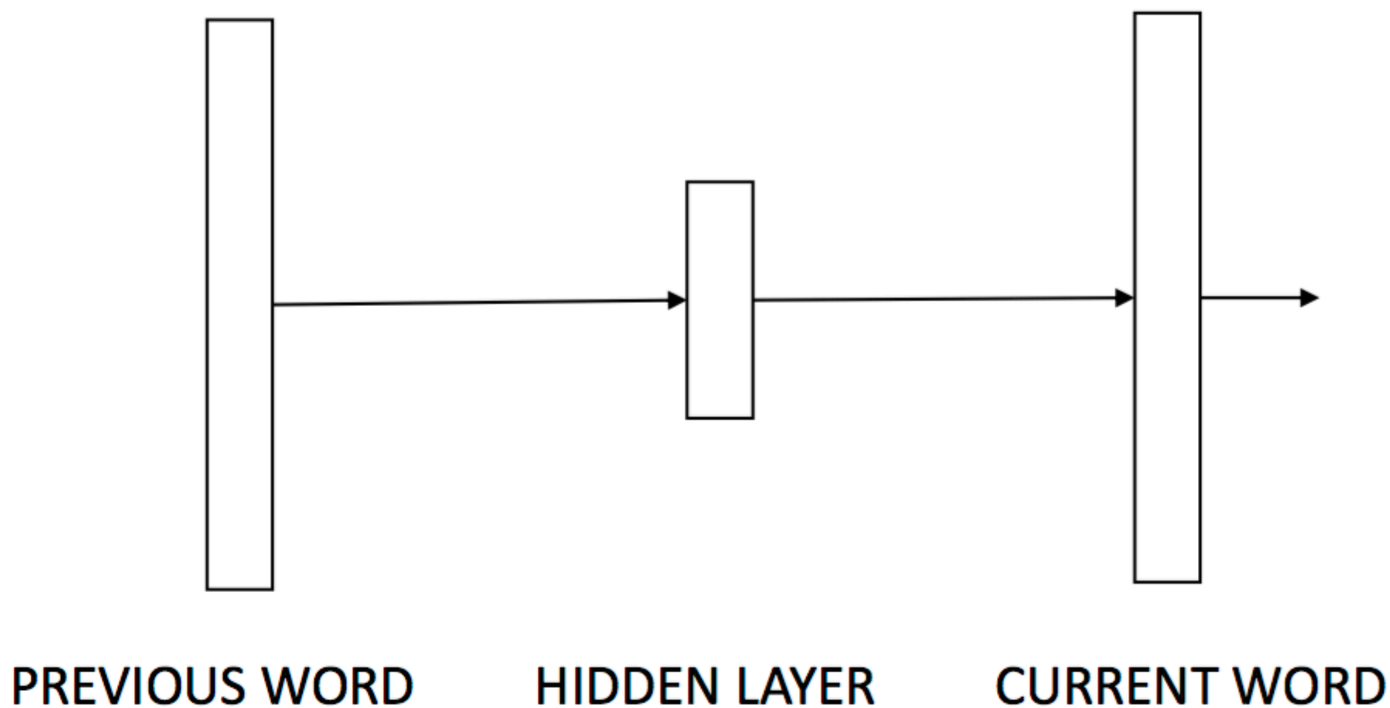
- Output layer
  $z = \mathbf{W_2 y}$
  $\mathbf{W_2}$ is 4x3



Recurrent NN



A simple NN bigram LM



Input layer - 1-of-V Vector
Hidden layer - Real valued vector in R^m, m << V
Output layer - Probability distribution over V words

A hint of Word Embeddings..

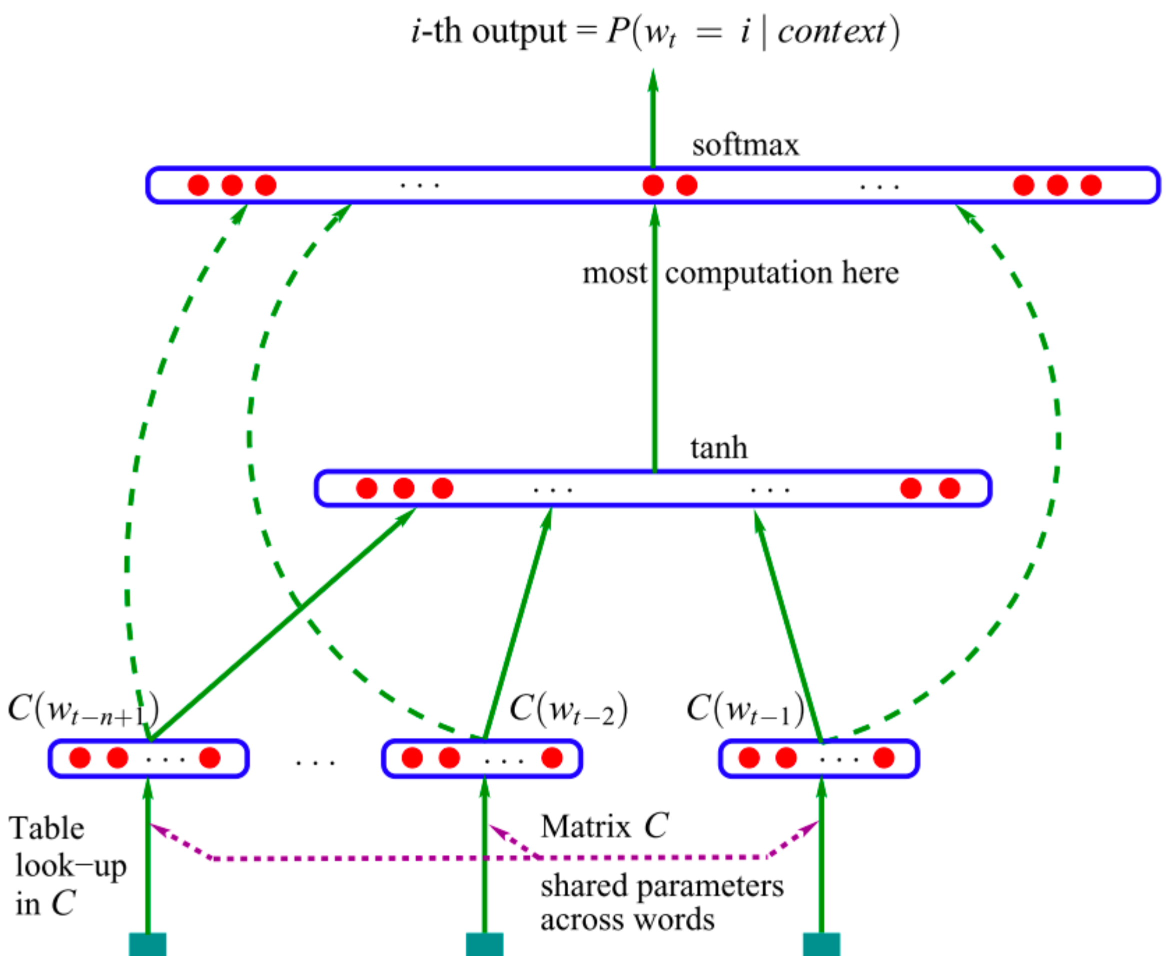
- Matrix between input and hidden layer
  - Vxm matrix
  - Vector representations for each word

Feedforward NN Language Model

- Approach summary –
  - Represent each word in the vocabulary as a **feature vector** (a real-valued vector in R^m)
  - Express the **joint probability function** of a sequence of words in terms of these vectors
  - Learn **probability function** and **vectors** simultaneously

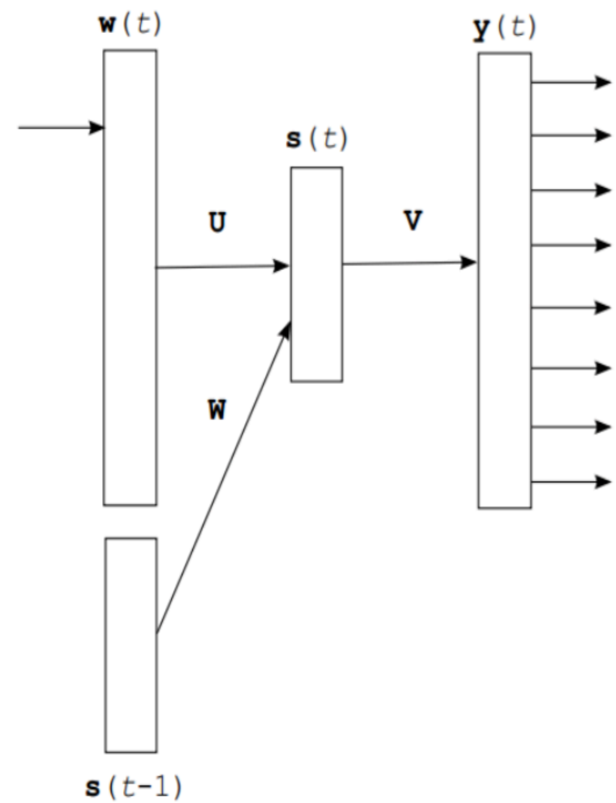
(Bengio, Yoshua, et al. "A Neural Probabilistic Language Model." *Journal Of Machine Learning Research*. 2003.)

Bengio et al - Details



Recurrent NN Language Model

- Recurrent NN
- Bengio et al still use fixed length context. By plugging in a recurrent NN, you can theoretically have infinite context!

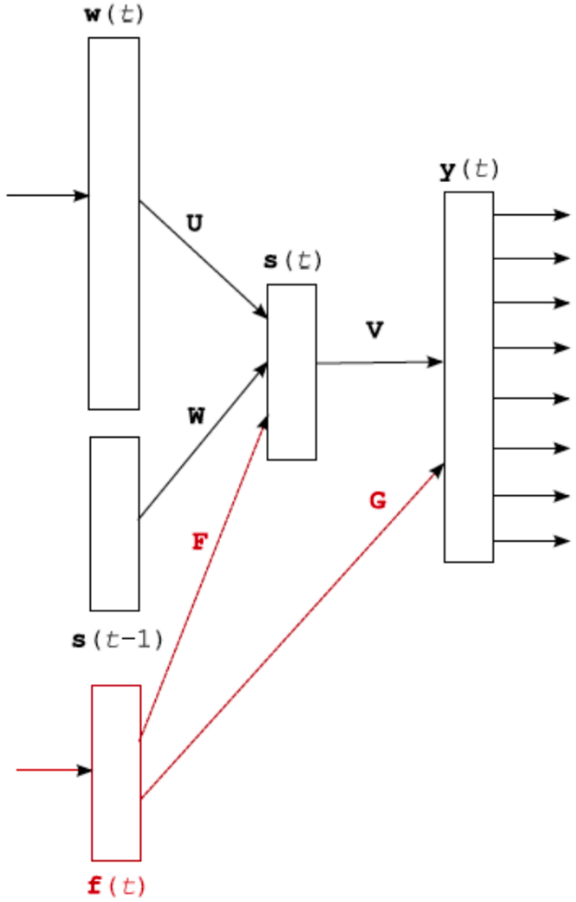


Mikolov, Tomas, et al. "Recurrent neural network based language model." INTERSPEECH. Vol. 2. 2010.

Extend RNNLM

- Add more features

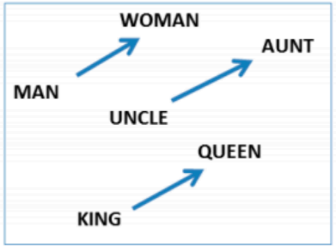
- What are some features you can add?

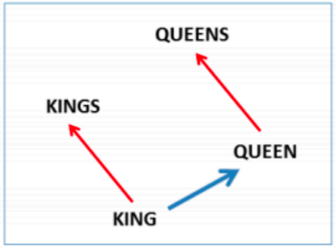


Word Embeddings

- Learn real valued vector space representations of words from large corpora

- Word vectors capture many linguistic properties
  - Syntactic - gender, tense, plurality
  - Sematic - “capital city of”





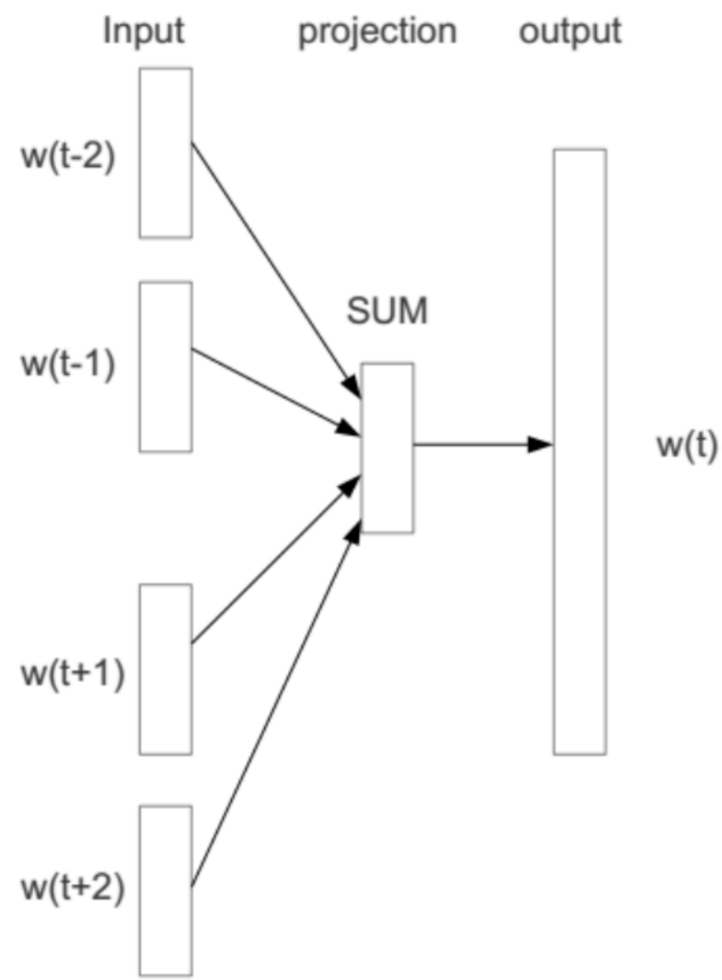
- We can do nearest neighbor search around result of vector operation “King - man + woman” and obtain “Queen”

(Linguistic regularities in continuous space word representations (Mikolov et al, 2013))

Expression	*Nearest token*
Paris - France + Italy	Rome
bigger - big + cold	colder
sushi - Japan + Germany	bratwurst
Cu - copper + gold	Au
Windows - Microsoft + Google	Android
Montreal Canadiens - Montreal + Toronto	Toronto Maple Leafs

CBoW

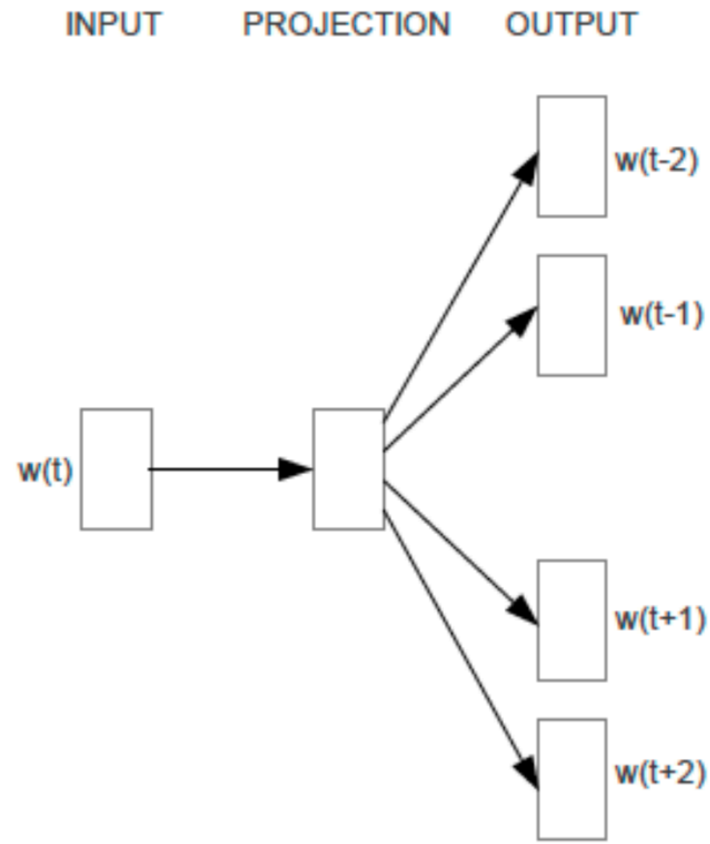
- Language modelling – use previous *n* words to predict the next word
- CBoW – Use previous ***and next*** words to predict current word
- Same as Feedforward NN, but no hidden layer, thus less expensive to compute



Mikolov, Tomas, et al. "Efficient estimation of word representations in vector space." *arXiv preprint arXiv:1301.3781* (2013).

SkipGram

- CBoW – Use context to predict the word
- SkipGram – Use word to predict the context



Tricks of the Trade

- Hierarchical softmax
  - The output softmax layer is huge (size of vocabulary)
  - Do it hierarchically, by predicting only word classes, then predicting subclasses, then subsubclasses... And so on until you get to leaves.

- Negative Sampling
  - Pick random word context pairs as negative examples

Whistles and Bells - Compositionality

- How do you get representations for phrases/ sentences?
  - Average!

- Skip-thought
  - Basically skip-gram over sentences

Kiros, Ryan, et al. "Skip-thought vectors." *Advances in Neural Information Processing Systems*. 2015.

Whistles and Bells – Multilingual Representations

- Independently learn vectors for English and French, learn a mapping (usually linear) between the two spaces
- Train an objective jointly over two languages
- In both cases, parallel data is used as crosslingual signal

Shortcomings

- Very little to no theoretical grounding
  - People have offered explanations for SGNS that draw parallels to LSA [(1)]
- Polysemy
  - One vector for the word 'bank'
    - Does it capture properties of the financial institution? Or the river bank? Or both? How do you disentangle? [(2)]
- Interpretability
  - 100 dimensional vector representations for each word – but what does each dimension capture?
  - By adding sparsity and non-negativity, you can get interpretability + some cognitive plausibility [(3)]

(1) - Levy, Omer, and Yoav Goldberg. "Neural word embedding as implicit matrix factorization." *NIPS* 2014.
(2) - Huang, Eric H., et al. "Improving word representations via global context and multiple word prototypes." *ACL* 2012.
(3) - Murphy, Brian, Partha Pratim Talukdar, and Tom M. Mitchell. "Learning Effective and Interpretable Semantic Models using Non-Negative Sparse Embedding." *COLING*. 2012.