

Deep Learning and Lexical, Syntactic and Semantic Analysis

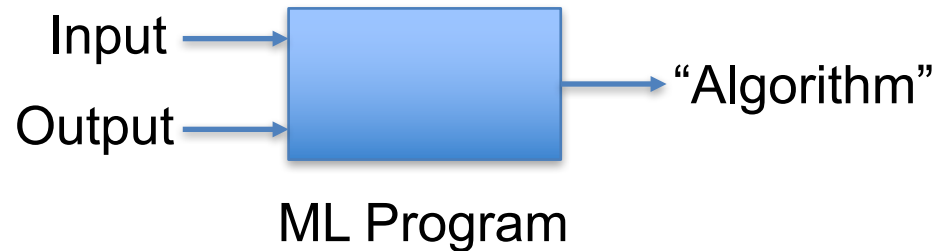
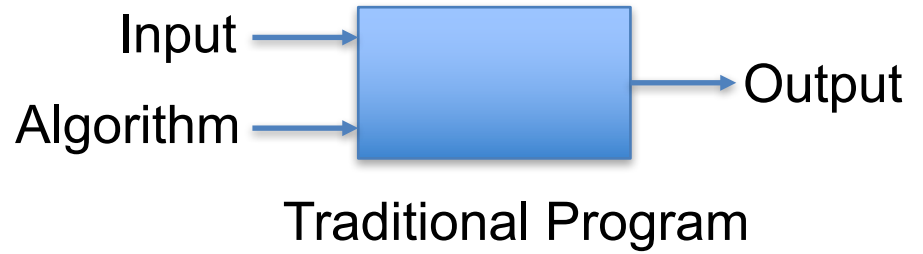
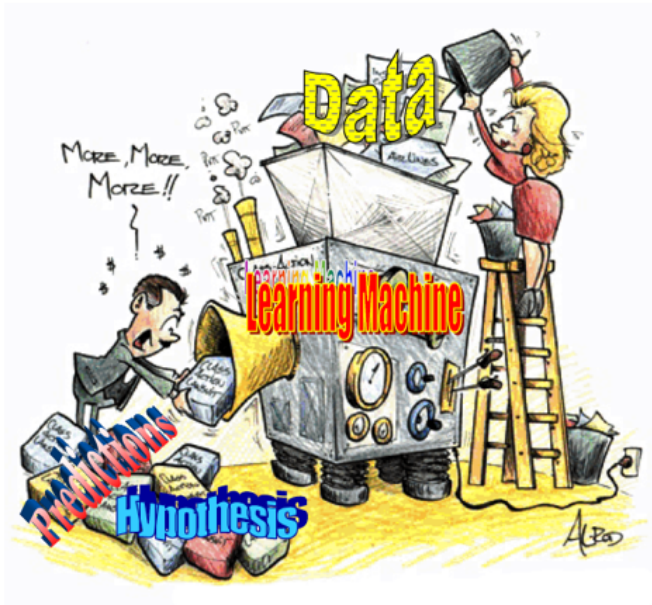
Wanxiang Che and Yue Zhang
2016-10

Part 2: Introduction to Deep Learning

Part 2.1: Deep Learning Background

What is Machine Learning?

- From Data to Knowledge



Very hard to say what makes a 2

0 0 0 1 1 1 1 1 2

2 2 2 2 2 2 2 3 2 3

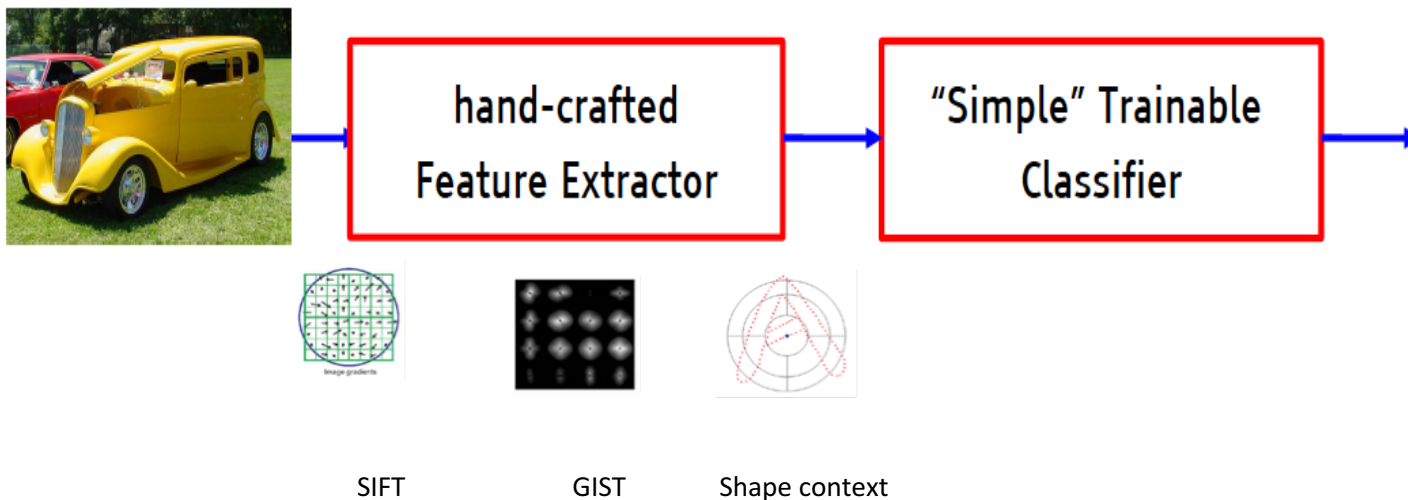
3 4 4 4 4 4 5 5 5 5

6 6 7 7 7 7 8 8 8

8 8 8 8 8 9 9 9 9

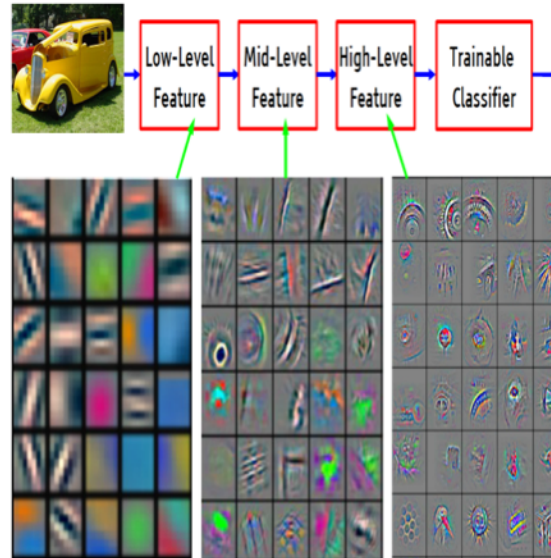
Traditional Model (before 2012)

- Fixed/engineered features + trainable classifier (分类器)
 - Designing a feature extractor requires considerable efforts by experts



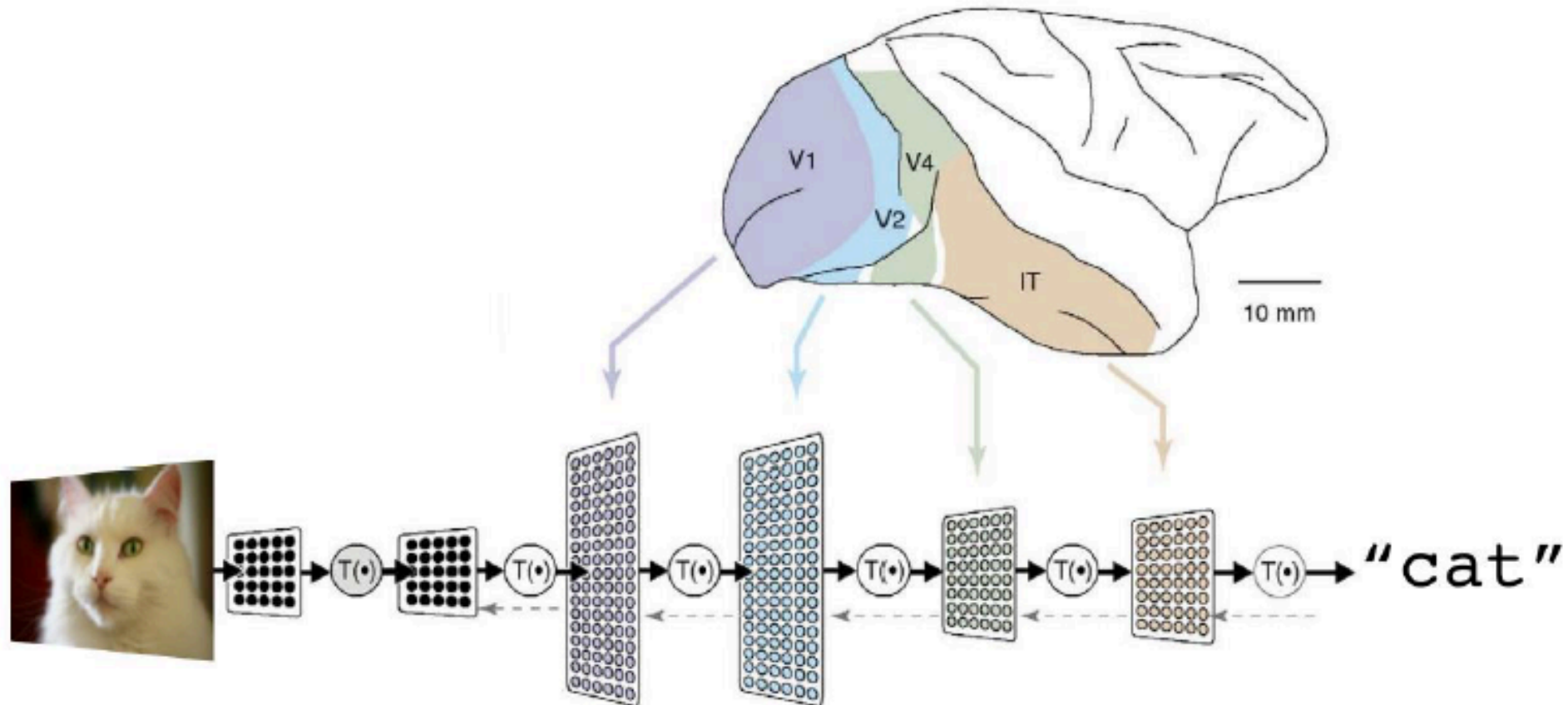
Deep Learning (after 2012)

- Learning Hierarchical Representations
- DEEP means **more than one** stage of **non-linear** feature transformation



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Deep Learning Architecture



Deep Learning is Not New

- 1980s technology (Neural Networks)

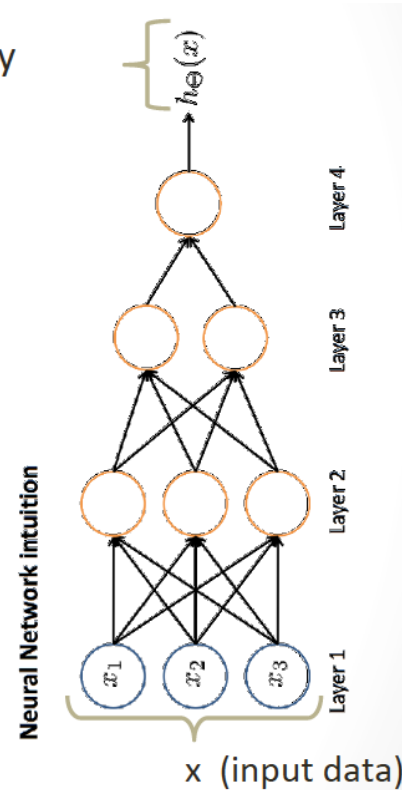
Supervised learning

- Given x and y , learn $p(y|x)$
- Is this photo, x , a “cat”, y ?

$x =$



(label) y



About Neural Networks

- Pros
 - Simple to learn $p(y|x)$
 - Results OK for shallow nets
- Cons
 - Does not learn $p(x)$
 - Trouble with > 3 layers
 - Overfitts
 - Slow to train

Deep Learning beats NN

- Pros

- Simple to learn $p(y|x)$
- Results OK for shallow nets

- Cons

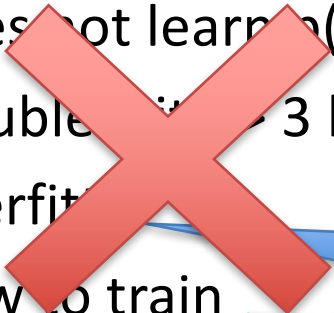
- Does not learn $p(x)$
- Trouble with > 3 layers
- Overfitting
- Slow to train

Unsupervised feature learning: RBMs, DAEs, ...

- New activation functions: ReLU, ...
- Gated mechanism

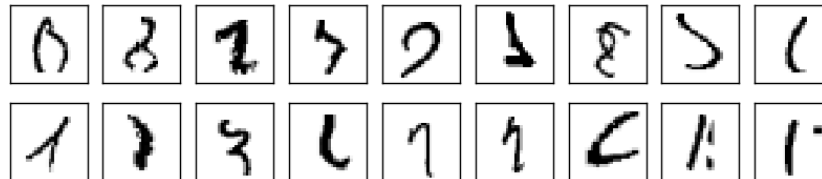
- Dropout
- Maxout
- Stochastic Pooling

GPU



Results on MNIST

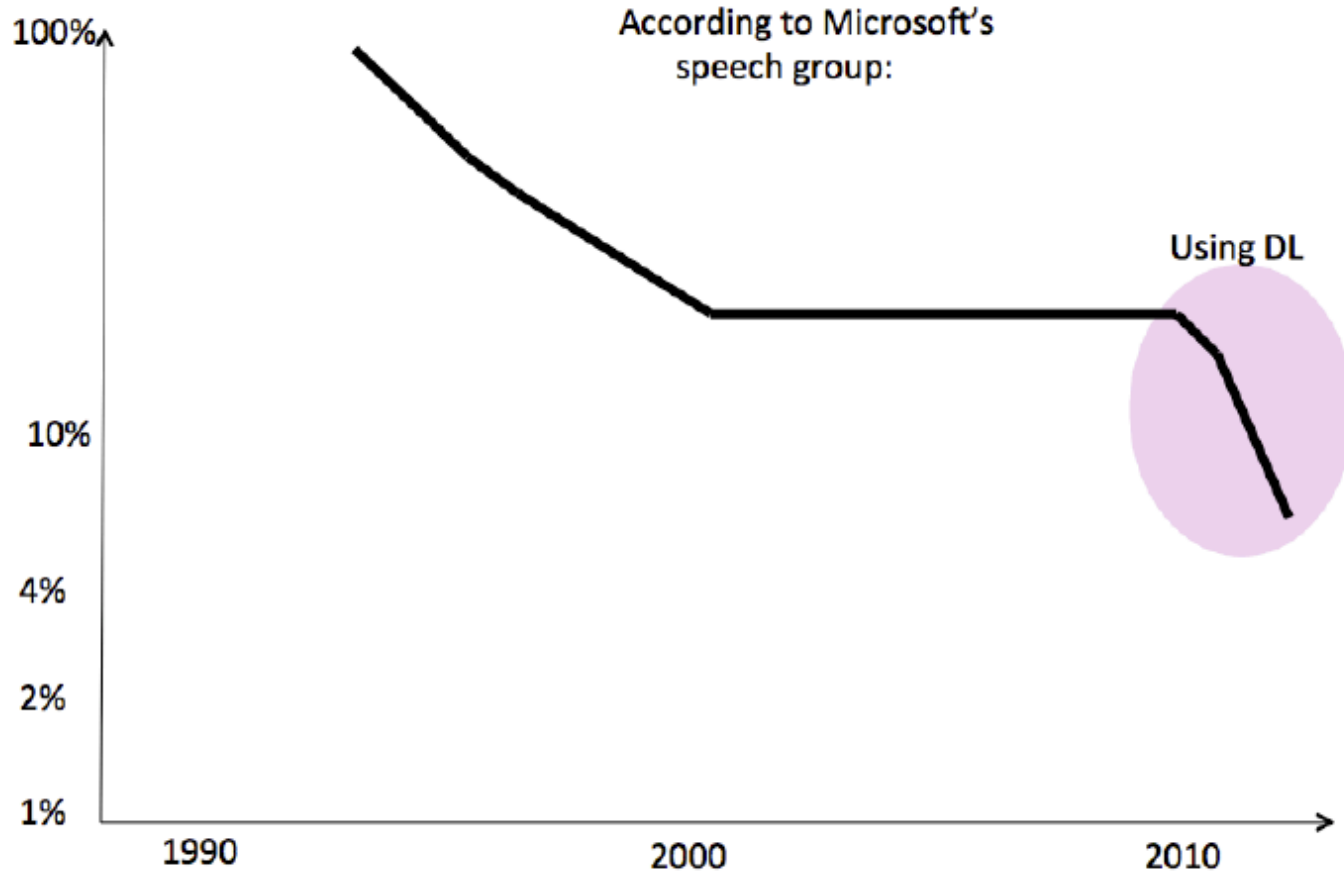
- Naïve Neural Network
 - 96.59%
- SVM (default settings for libsvm)
 - 94.35%
- Optimal SVM [Andreas Mueller]
 - 98.56%
- The state of the art: Convolutional NN (2013)
 - 99.79%



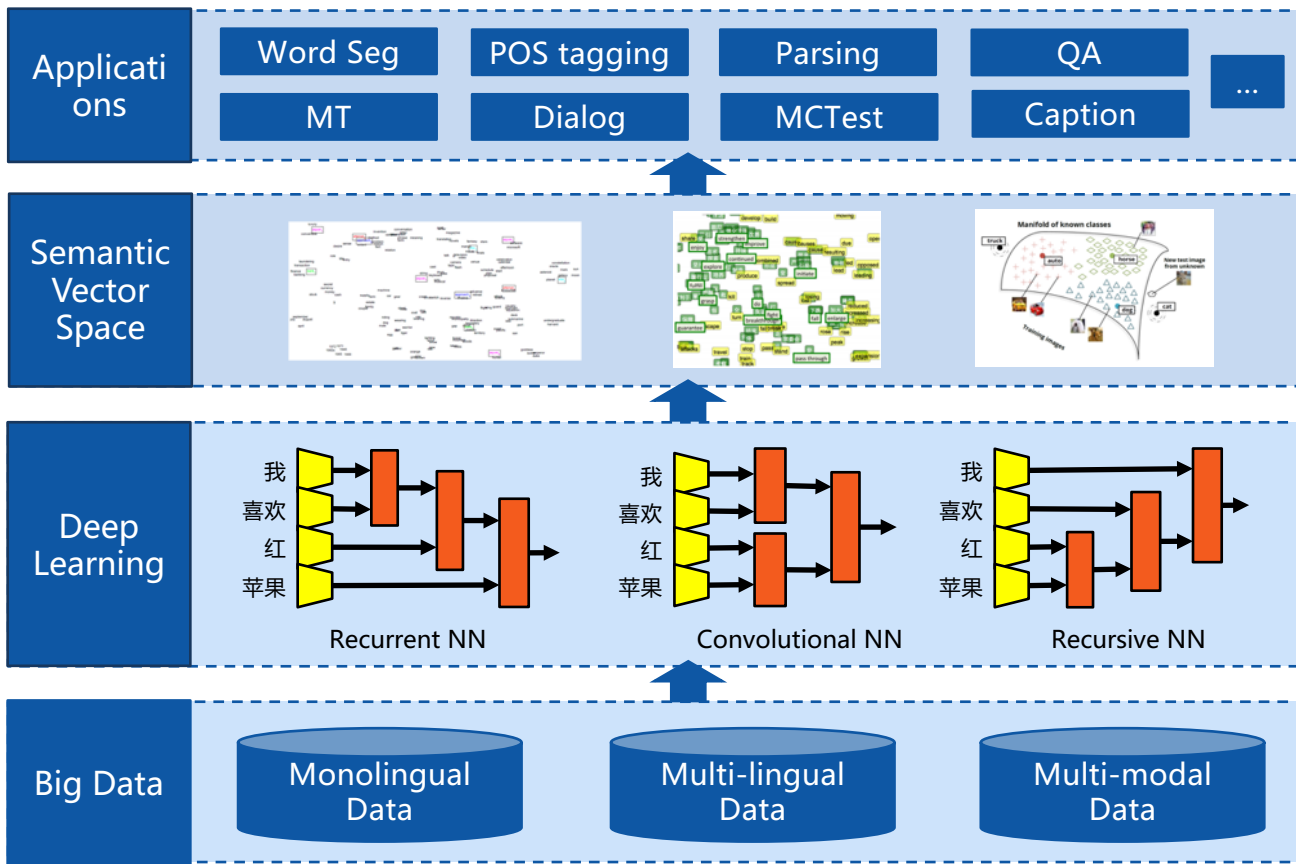
Deep Learning Wins

9. MICCAI 2013 Grand Challenge on Mitosis Detection
8. ICPR 2012 Contest on Mitosis Detection in Breast Cancer Histological Images
7. ISBI 2012 Brain Image Segmentation Challenge (with superhuman pixel error rate)
6. IJCNN 2011 Traffic Sign Recognition Competition (only our method achieved superhuman results)
5. ICDAR 2011 offline Chinese Handwriting Competition
4. Online German Traffic Sign Recognition Contest
3. ICDAR 2009 Arabic Connected Handwriting Competition
2. ICDAR 2009 Handwritten Farsi/Arabic Character Recognition Competition
1. ICDAR 2009 French Connected Handwriting Competition. Compare the overview page on handwriting recognition.
 - <http://people.idsia.ch/~juergen/deeplearning.html>

Deep Learning for Speech Recognition



Deep Learning for NLP

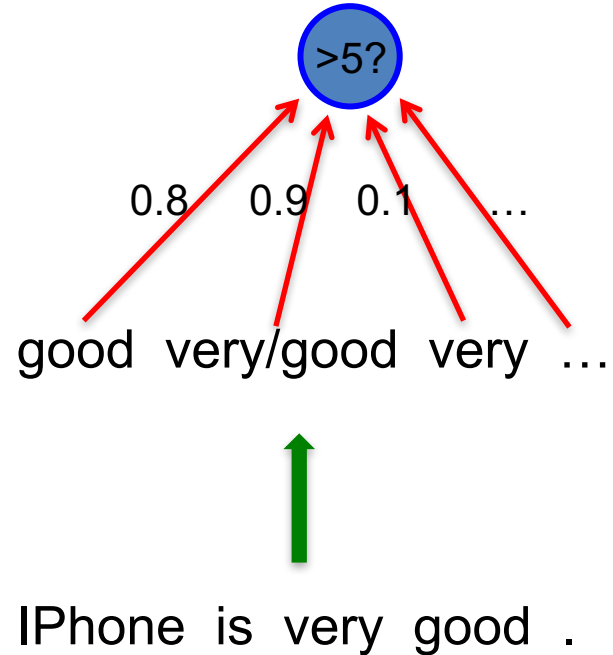
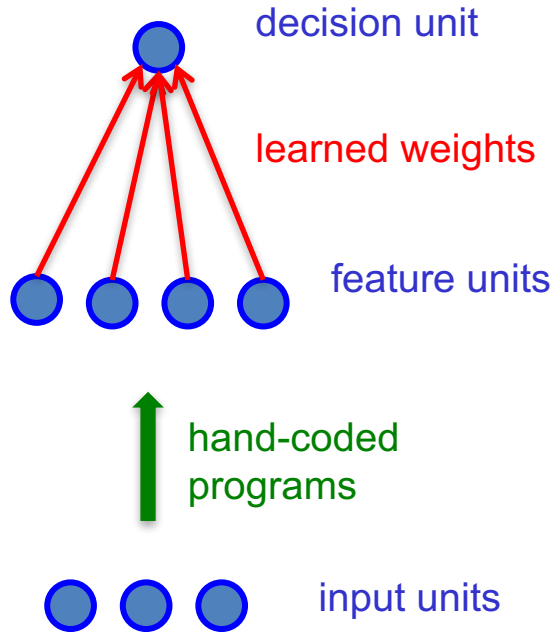


Part 2.2: Feedforward Neural Networks

The Traditional Paradigm for ML

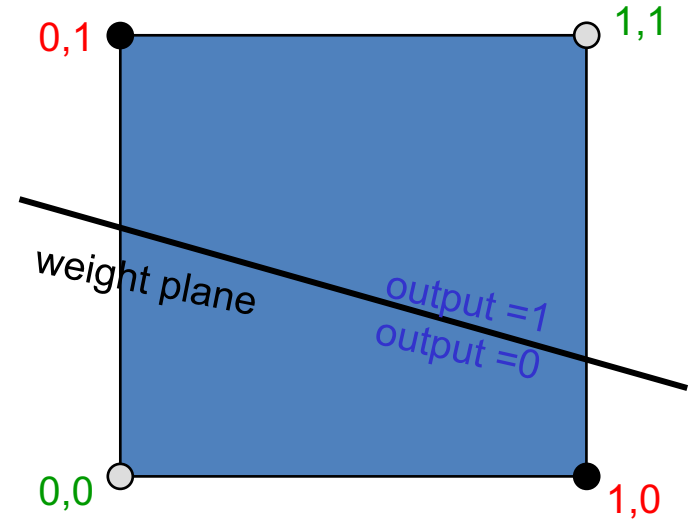
1. Convert the raw input vector into a vector of feature activations
 - Use hand-written programs based on common-sense to define the features
2. Learn how to weight each of the feature activations to get a single scalar quantity
3. If this quantity is above some threshold, decide that the input vector is a positive example of the target class

The Standard Perceptron Architecture



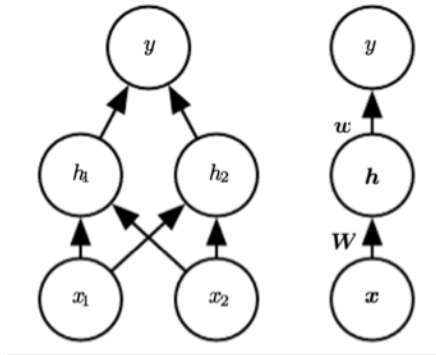
The Limitations of Perceptrons

- The **hand-coded features**
 - Great influence on the performance
 - Need lots of cost to find suitable features
- A **linear classifier** with a hyperplane
 - Cannot separate non-linear data, such as XOR function cannot be learned by a single-layer perceptron



The **positive** and **negative** cases cannot be separated by a plane

Learning with Non-linear Hidden Layers

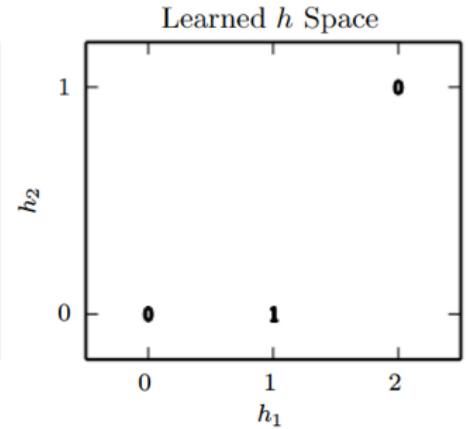
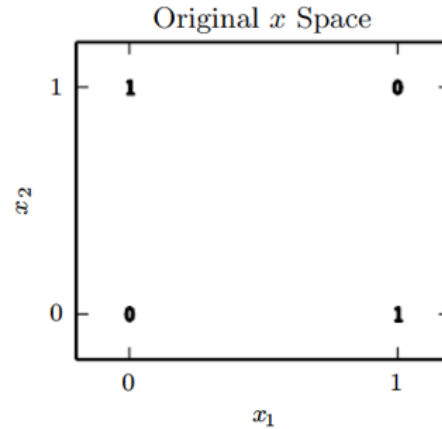


$$\mathbf{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

$$\mathbf{c} = \begin{bmatrix} 0 \\ -1 \end{bmatrix},$$

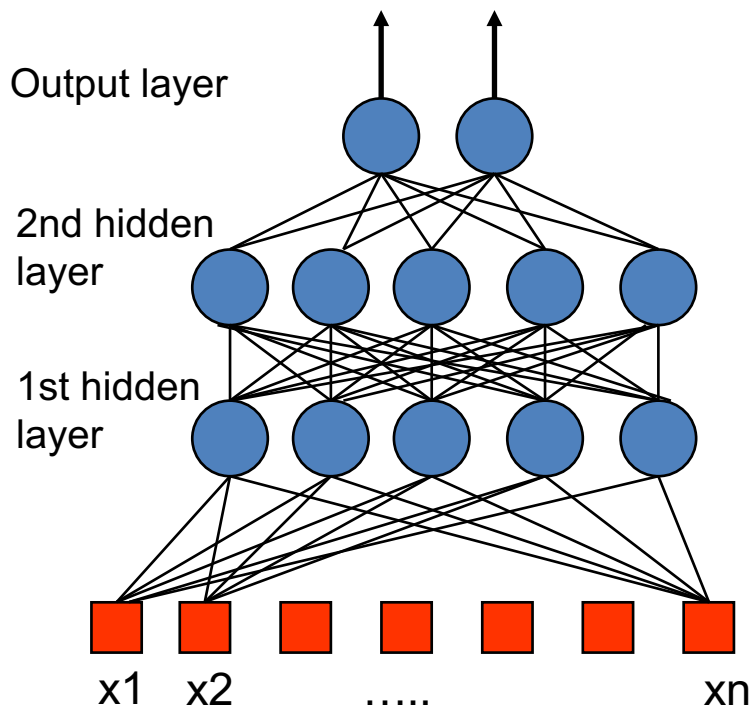
$$\mathbf{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

$$b = 0.$$



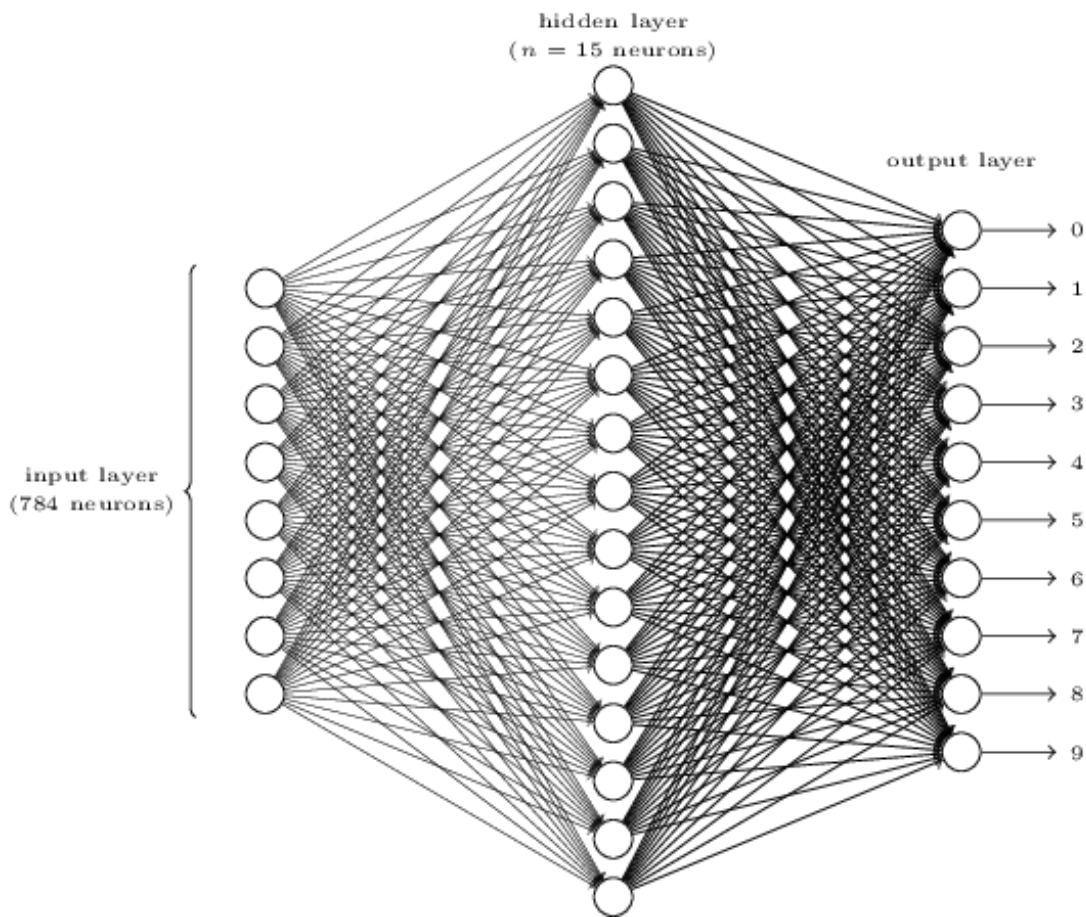
$$f(\mathbf{x}; \mathbf{W}, \mathbf{c}, \mathbf{w}, b) = \mathbf{w}^\top \max\{0, \mathbf{W}^\top \mathbf{x} + \mathbf{c}\} + b.$$

Feedforward Neural Networks

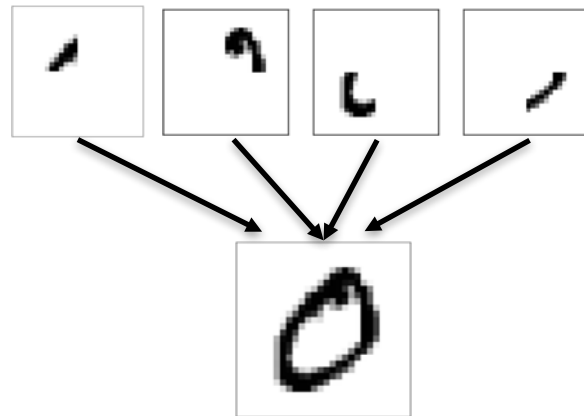


- The information is propagated from the inputs to the outputs
- Time has no role (NO cycle between outputs and inputs)
- Multi-layer Perceptron (**MLP**)?
- Learning the weights of hidden units is equivalent to **learning features**
- Networks without hidden layers are very limited in the input-output mappings
 - More layers of linear units do not help. Its still linear
 - Fixed output non-linearities are not enough

Multiple Layer Neural Networks



- What are those hidden neurons doing?
 - Maybe represent outlines



General Optimizing (Learning) Algorithms

- Gradient Descent

$$\theta \leftarrow \theta + \epsilon \nabla_{\theta} \sum_t L(f(\mathbf{x}^{(t)}; \theta), \mathbf{y}^{(t)}; \theta)$$

- Stochastic Gradient Descent (SGD)

- Minibatch SGD ($m > 1$), Online GD ($m = 1$)

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k .

Require: Initial parameter θ

while stopping criterion not met **do**

 Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

 Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 Apply update: $\theta \leftarrow \theta - \epsilon \hat{\mathbf{g}}$

end while

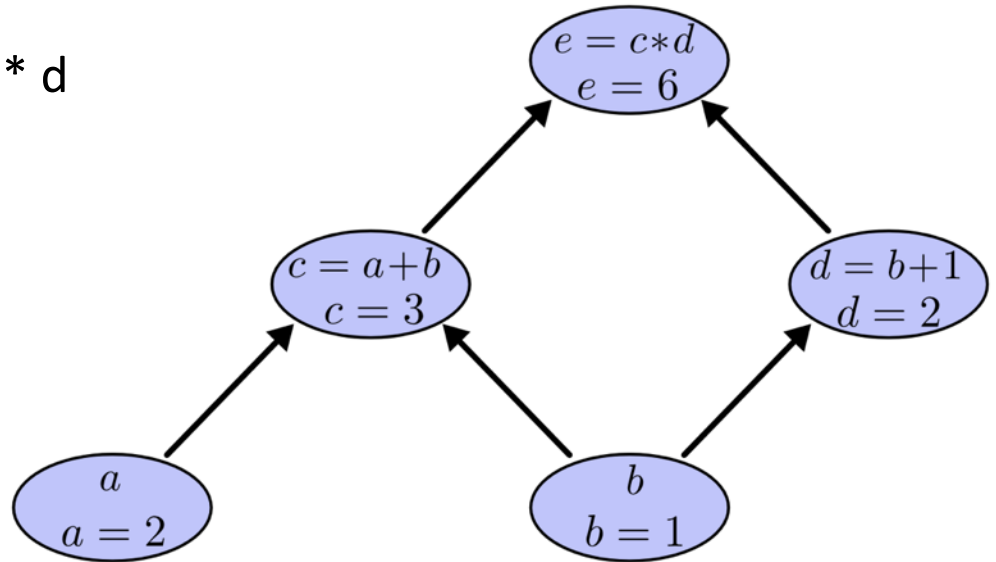
Computational/Flow Graphs

- Describing Mathematical Expressions
- For example

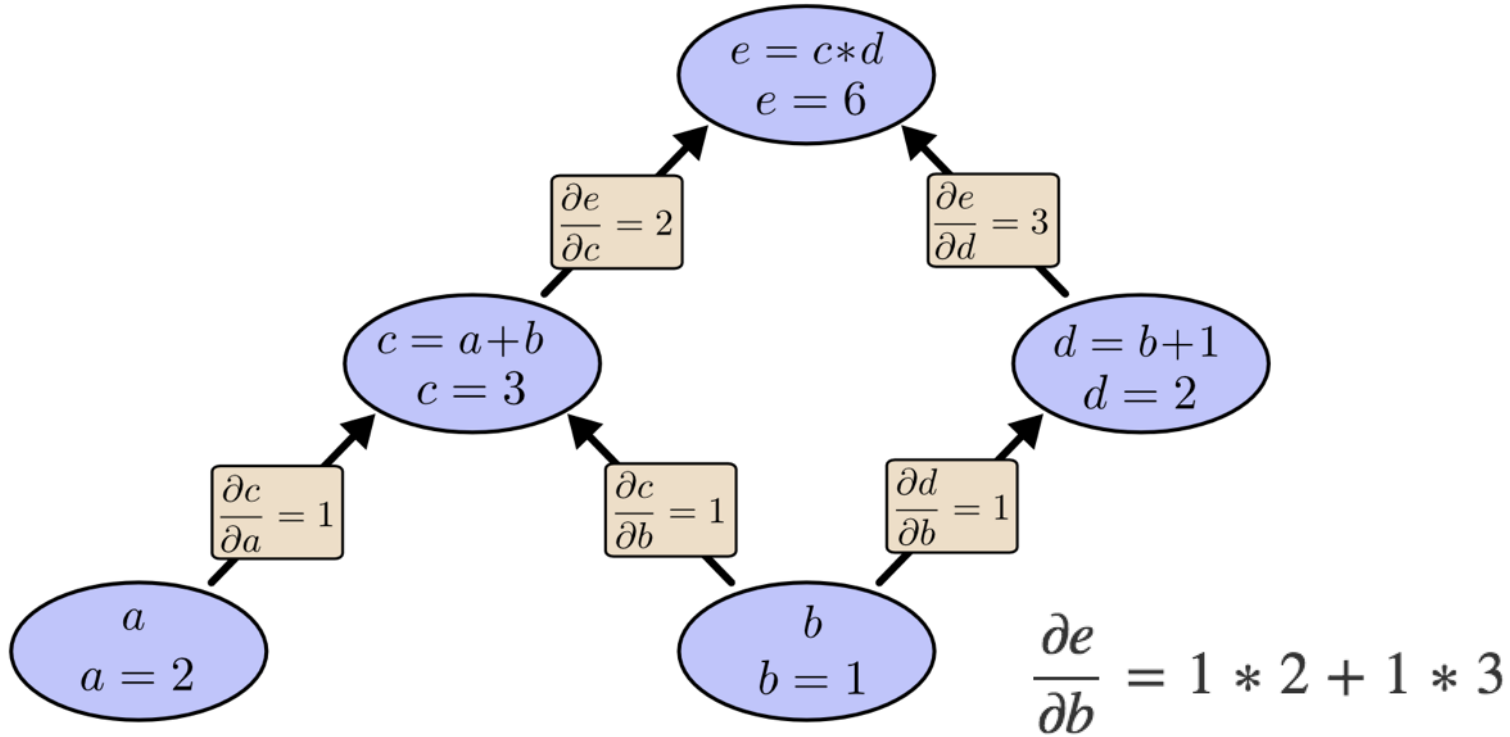
– $e = (a + b) * (b + 1)$

- $c = a + b, d = b + 1, e = c * d$

– If $a = 2, b = 1$

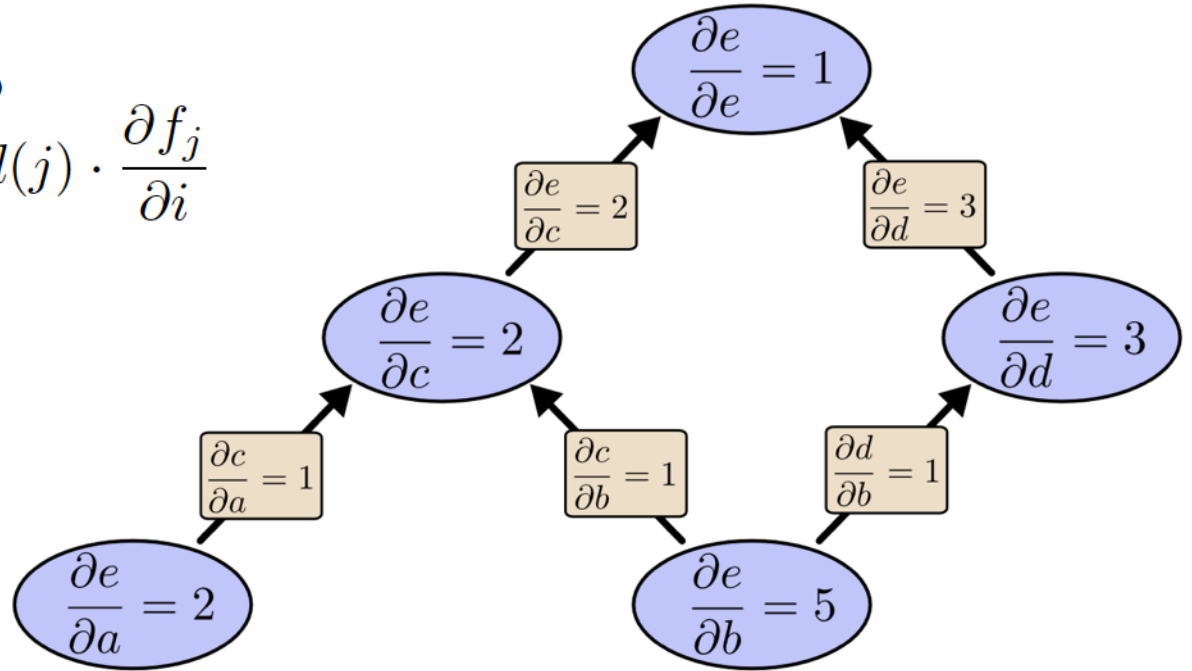


Derivatives on Computational Graphs



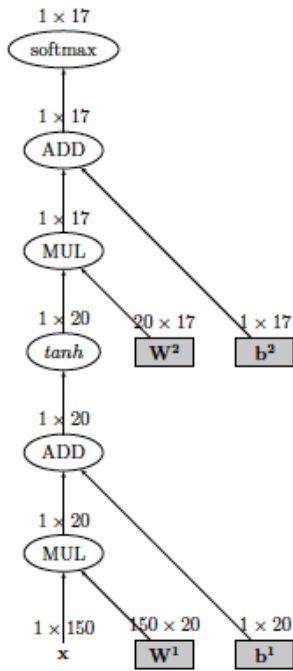
Computational Graph Backward Pass (Backpropagation)

- 1: $d(N) \leftarrow 1$
- 2: **for** $i = N-1$ to 1 **do**
- 3: $d(i) \leftarrow \sum_{j \in \pi(i)} d(j) \cdot \frac{\partial f_j}{\partial i}$

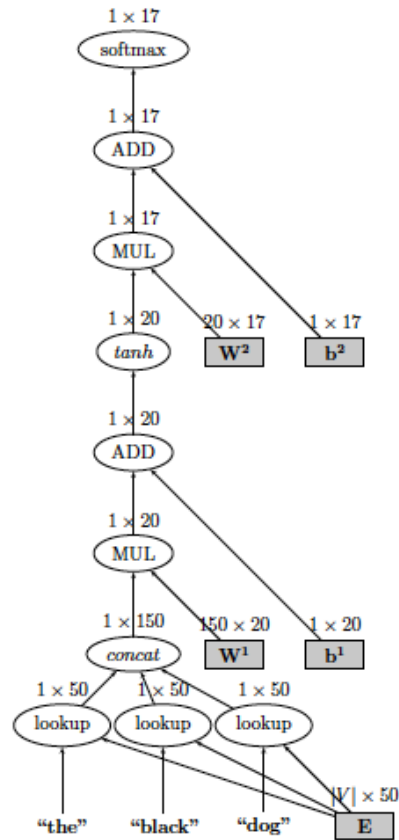


An FNN POS Tagger

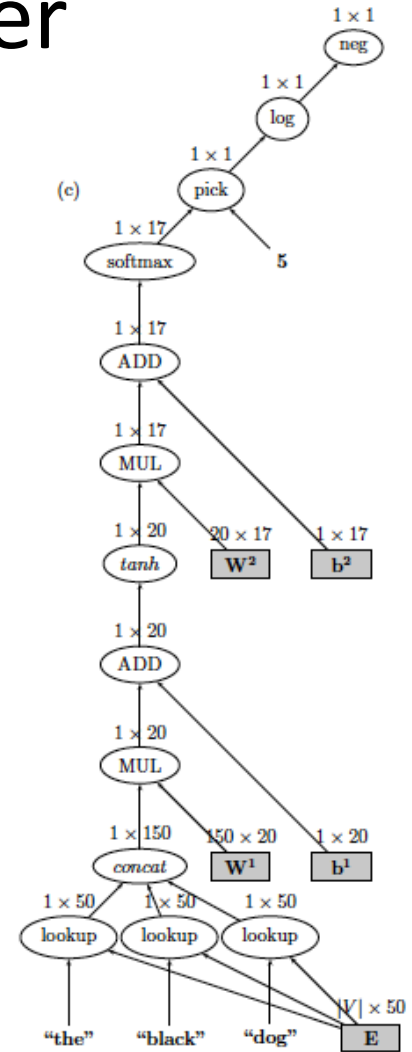
(a)



(b)



(c)




Part 2.3: Word Embeddings

Typical Approaches for Word Representation

- 1-hot representation (orthogonality)
 - bag-of-words model

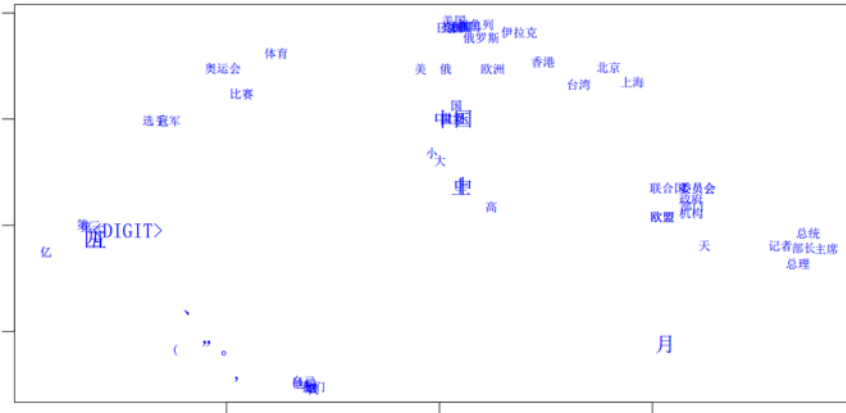
star [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, ...]

sun [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, ...]

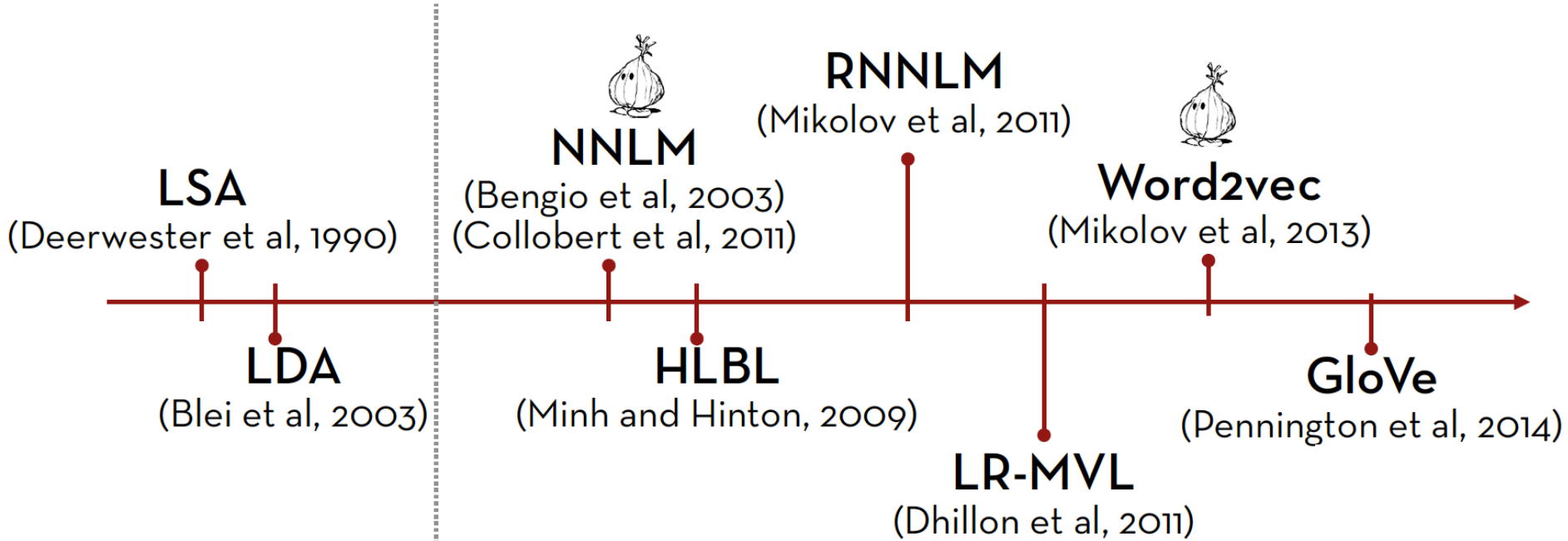
$\text{sim}(\text{star}, \text{sun}) = 0$ 

Distributed Word Representation

- Each word is associated with a **low-dimension** (compressed, 50-1000), **density** (non-sparse) and **real** (continuous) vector (**word embedding**)
 - Learning word vectors through **supervised** models
- Nature
 - Semantic similarity as vector similarity

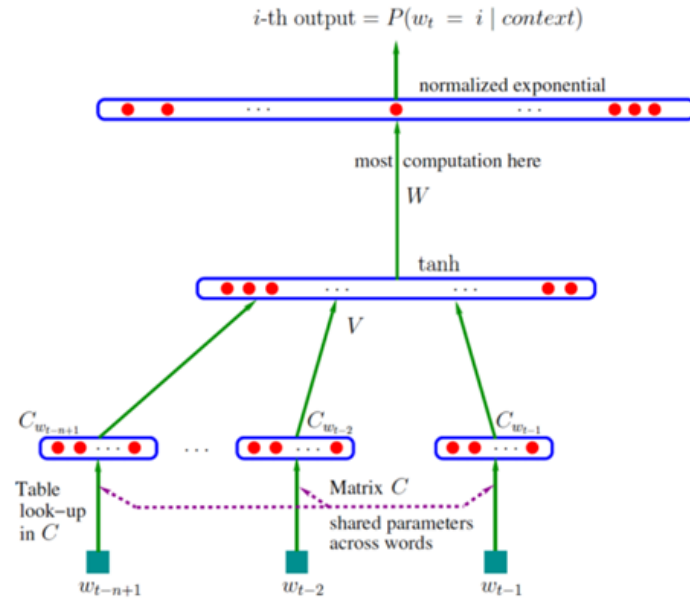


How to obtain Word Embedding



Neural Network Language Models

- Neural Network Language Models (NNLM)
 - Feed Forward (Bengio et al. 2003)



- **Maximum-Likelihood Estimation**
- **Back-propagation**
- Input: $(n - 1)$ embeddings

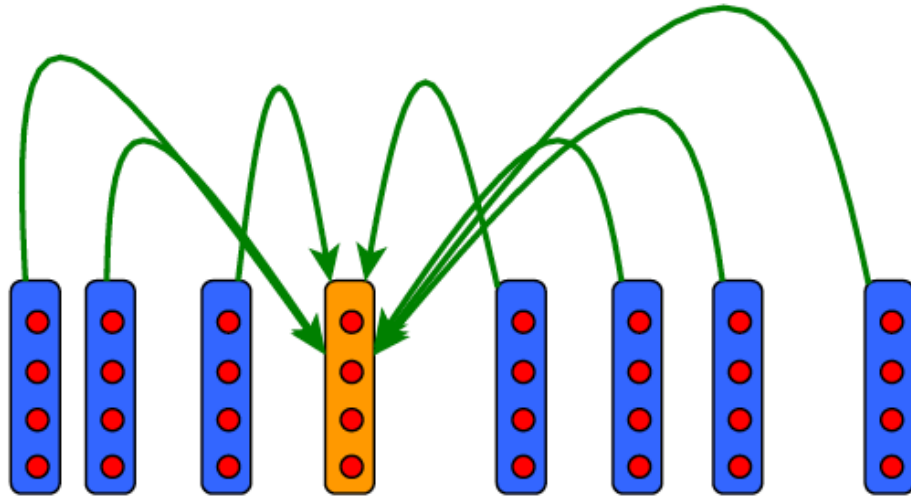
$$P(w_t = k | w_{t-n+1}, \dots, w_{t-1}) = \frac{e^{a_k}}{\sum_{l=1}^N e^{a_l}}$$

$$a_k = b_k + \sum_{i=1}^h W_{ki} \tanh(c_i + \sum_{j=1}^{(n-1)d} V_{ij} x_j)$$

$$L(\theta) = \sum_t \log P(w_t | w_{t-n+1}, \dots, w_{t-1})$$

Predict Word Vector Directly

- SENNA (Collobert and Weston, 2008)
- word2vec (Mikolov et al. 2013)



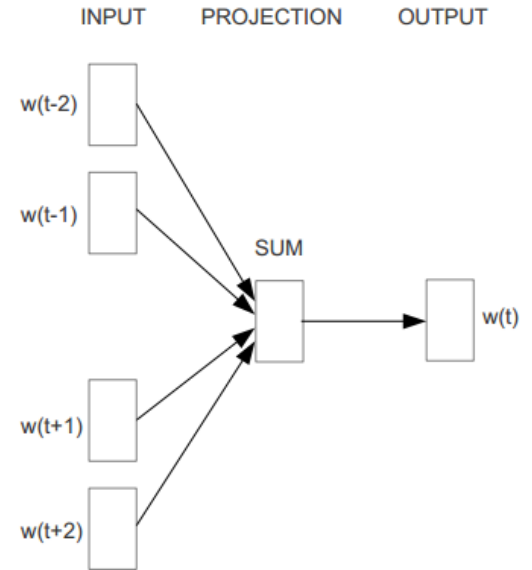
I got the shotgun. You got the briefcase.

Word2vec: CBOW (Continuous Bag-of-Word)

- Add inputs from words within short window to predict the current word
- The weights for different positions are shared
- Computationally much more efficient than normal NNLM
- The hidden layer is just linear
- Each word is an embedding $v(w)$
- Each context is an embedding $v'(c)$

$$r(c) = v'(c_{-2}) + v'(c_{-1}) + v'(c_1) + v'(c_2)$$

$$p(v(w) | r(c)) = \frac{\exp(r(c) \cdot v(w))}{\sum_{w^*} \exp(r(c) \cdot v(w^*))}$$



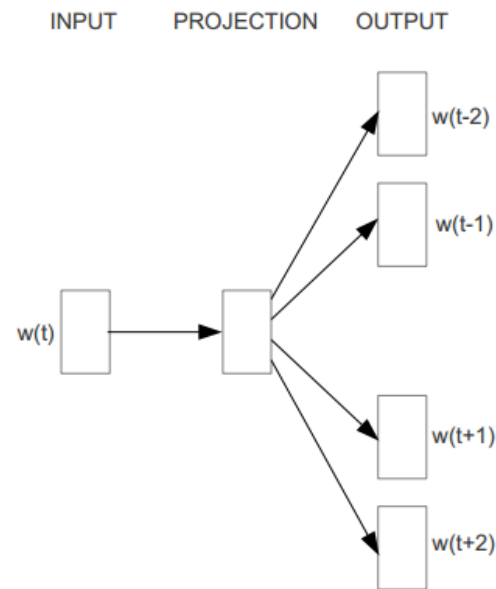
CBOW

Word2vec: Skip-Gram

- Predicting surrounding words using the current word
- Similar performance with CBOW
- Each word is an embedding $v(w)$
- Each context is an embedding $v'(c)$

$$\frac{1}{|\mathcal{C}|} \sum_{(w,c) \in \mathcal{C}} \log p(v'(c) | v(w))$$

$$p(v'(c) | v(w)) = \frac{\exp(v'(c) \cdot v(w))}{\sum_{c^*} \exp(v'(c^*) \cdot v(w))}$$



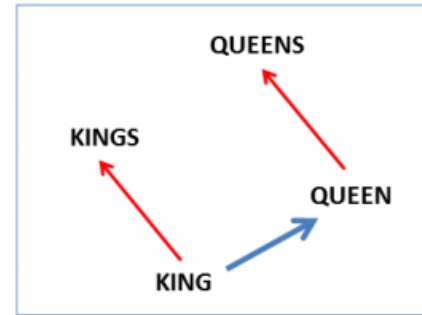
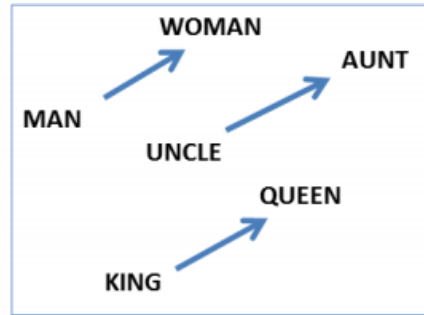
Skip-gram

Word2vec Training

- SGD + backpropagation
- Most of the computational cost is a function of the size of the vocabulary (millions)
- Training accelerating
 - Negative Sampling
 - Mikolov et al. 2013
 - Hierarchical Decomposition
 - Morin and Bengio 2005. Mnih and Hinton 2008. Mikolov et al. 2013
 - Graph Processing Unit (GPU)

Word Analogy

$$v(\text{king}) - v(\text{queen}) \approx v(\text{man}) - v(\text{woman})$$



Part 2.4: Recurrent and Other Neural Networks

Language Models

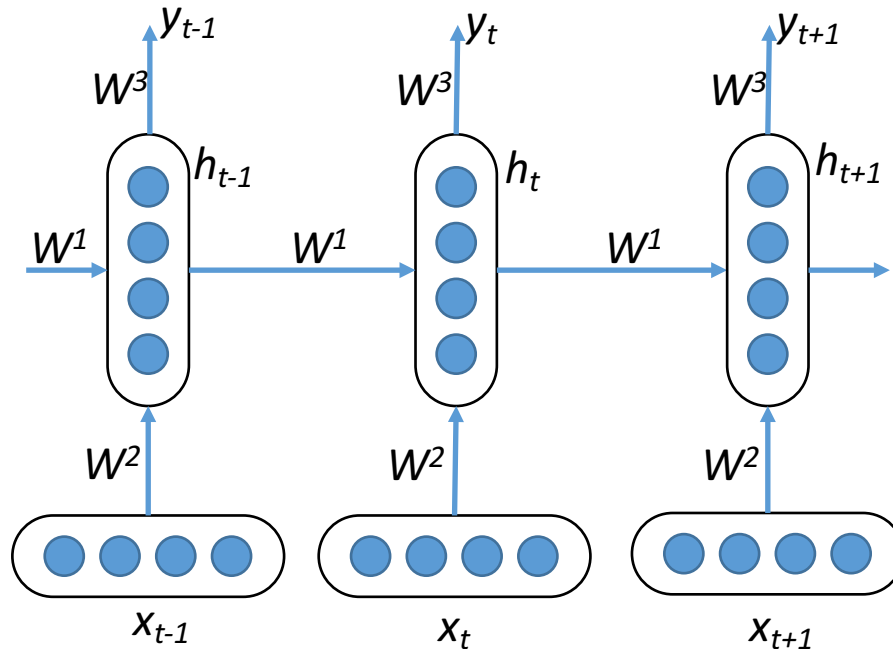
- A language model computes a probability for a sequence of word: $P(w_1, \dots w_n)$ or predicts a probability for the next word: $P(w_{n+1} | w_1, \dots w_n)$
- Useful for machine translation, speech recognition, and so on
 - Word ordering
 - $P(\text{the cat is small}) > P(\text{small the is cat})$
 - Word choice
 - $P(\text{there are **four** cats}) > P(\text{there are **for** cats})$

Traditional Language Models

- An incorrect but necessary Markov assumption!
 - Probability is usually conditioned on window of n previous words
 - $P(w_1, \dots, w_n) = \prod_{i=1}^m P(w_i | w_1, \dots, w_{i-1}) \approx \prod_{i=1}^m P(w_i | w_{i-(n-1)}, \dots, w_{i-1})$
- How to estimate probabilities
 - $p(w_2 | w_1) = \frac{\text{count}(w_1, w_2)}{\text{count}(w_1)}$ $p(w_3 | w_1, w_2) = \frac{\text{count}(w_1, w_2, w_3)}{\text{count}(w_1, w_2)}$
- Performance improves with keeping around higher n-grams counts and doing smoothing, such as backoff (e.g. if 4-gram not found, try 3-gram, etc)
- Disadvantages
 - There are A LOT of n-grams!
 - Cannot see too long history
 - $P(\text{坐/作 了一整天的 火车/作业})$

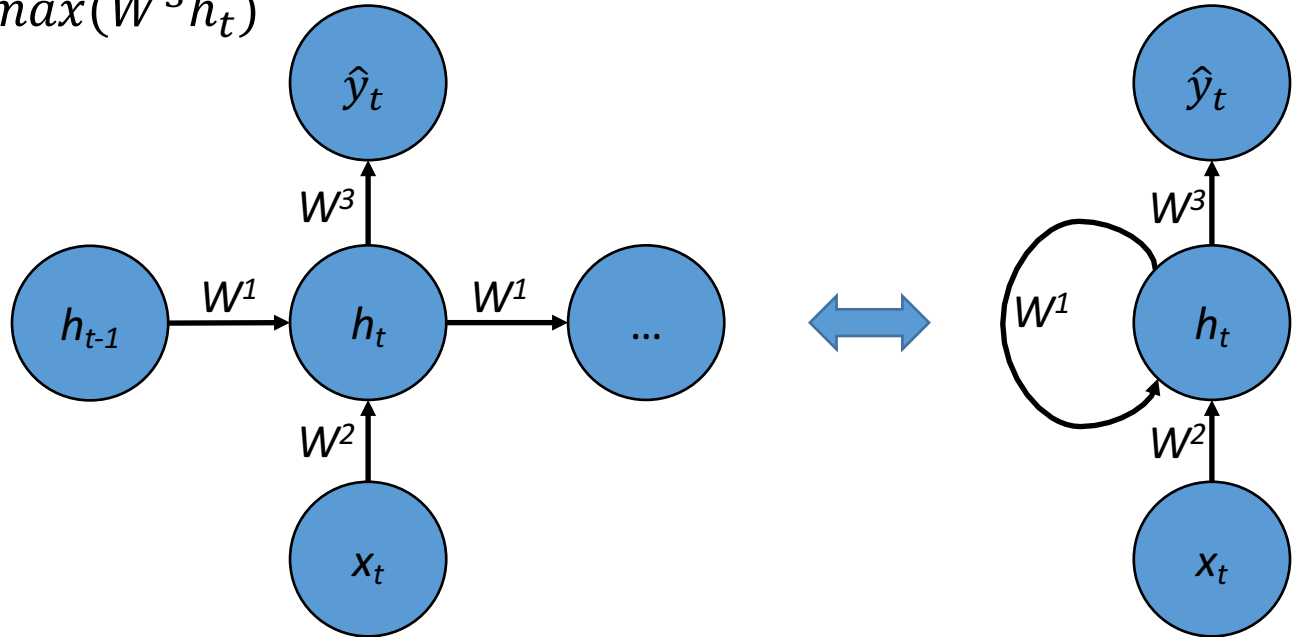
Recurrent Neural Networks (RNNs)

- Condition the neural network on all previous inputs
- RAM requirement only scales with number of inputs



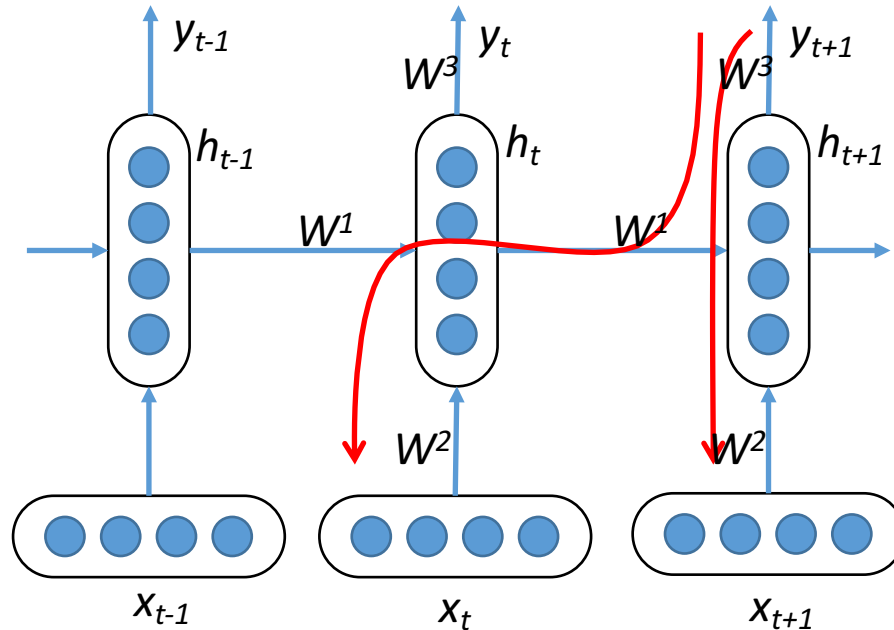
Recurrent Neural Networks (RNNs)

- At a single time step t
 - $h_t = \tanh(W^1 h_{t-1} + W^2 x_t)$
 - $\hat{y}_t = \text{softmax}(W^3 h_t)$



Training RNNs is hard

- Ideally inputs from many time steps ago can modify output y
- For example, with 2 time steps



BackPropagation Through Time (BPTT)

- Total error is the sum of each error at time step t

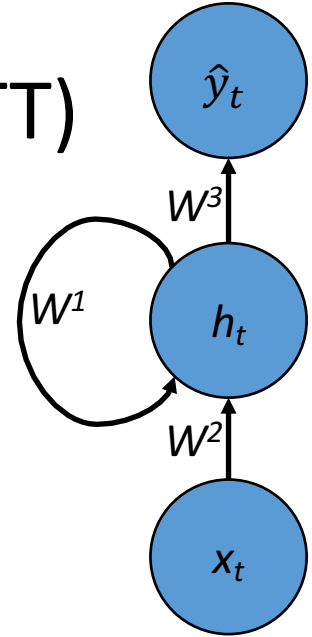
- $\frac{\partial E}{\partial W} = \sum_{t=1}^T \frac{\partial E_t}{\partial W}$

- $\frac{\partial E_t}{\partial W^3} = \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial W^3}$ is easy to be calculated

- But to calculate $\frac{\partial E_t}{\partial W^1} = \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial W^1}$ is hard (also for W^2)

- Because $h_t = \tanh(W^1 h_{t-1} + W^2 x_t)$ depends on h_{t-1} , which depends on W^1 and h_{t-2} , and so on.

- So $\frac{\partial E_t}{\partial W^1} = \sum_{k=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W^1}$



The vanishing gradient problem

- $\frac{\partial E_t}{\partial W} = \sum_{k=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}$, $h_t = \tanh(W^1 h_{t-1} + W^2 x_t)$

- $\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^t W^1 \text{diag}[\tanh'(\dots)]$

- $\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\| \leq \gamma \|W^1\| \leq \gamma \lambda_1$

- where γ is bound $\|\text{diag}[\tanh'(\dots)]\|$, λ_1 is the largest singular value of W^1

- $\left\| \frac{\partial h_t}{\partial h_k} \right\| \leq (\gamma \lambda_1)^{t-k} \rightarrow 0$, if $\gamma \lambda_1 < 1$

- This can become very small or very large quickly \rightarrow Vanishing or exploding gradient

- Trick for exploding gradient: clipping trick (set a threshold)

A “solution”

- Intuition
 - Ensure $\gamma\lambda_1 \geq 1 \rightarrow$ to prevent vanishing gradients
- So ...
 - Proper initialization of the W
 - To use ReLU instead of tanh or sigmoid activation functions

A better “solution”

- Recall the original transition equation

- $h_t = \tanh(W^1 h_{t-1} + W^2 x_t)$

- We can instead update the state **additively**

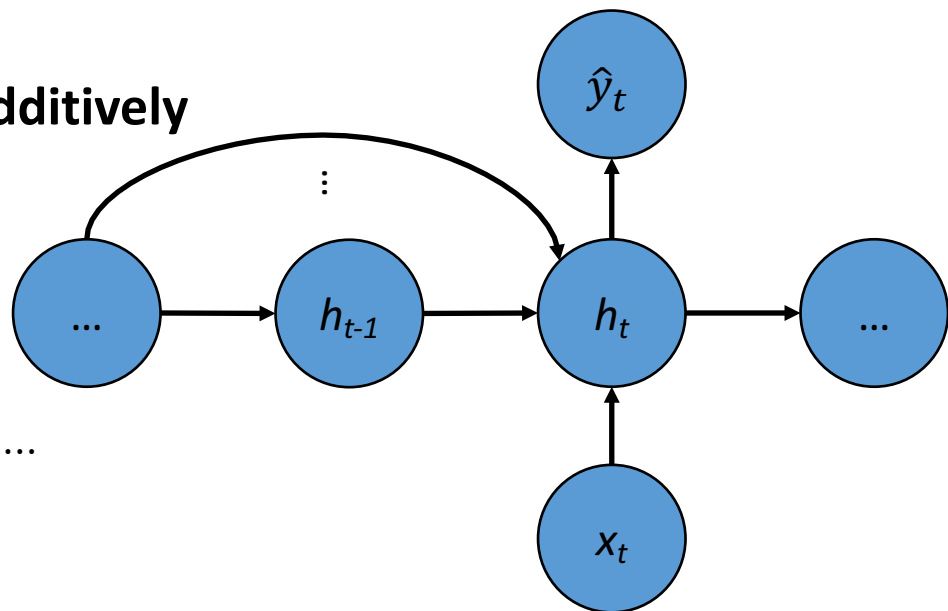
- $u_t = \tanh(W^1 h_{t-1} + W^2 x_t)$

- $h_t = h_{t-1} + u_t$

- then, $\left\| \frac{\partial h_t}{\partial h_{t-1}} \right\| = 1 + \left\| \frac{\partial u_t}{\partial h_{t-1}} \right\| \geq 1$

- On the other hand

- $h_t = h_{t-1} + u_t = h_{t-2} + u_{t-1} + u_t = \dots$



A better “solution” (cont.)

- Interpolate between old state and new state (“choosing to **forget**”)
 - $f_t = \sigma(W^f x_t + U^f h_{t-1})$
 - $h_t = f_t \odot h_{t-1} + (1 - f_t) \odot u_t$
- Introduce a separate **input gate** i_t
 - $i_t = \sigma(W^i x_t + U^i h_{t-1})$
 - $h_t = f_t \odot h_{t-1} + i_t \odot u_t$
- Selectively expose memory cell c_t with an **output gate** o_t
 - $o_t = \sigma(W^o x_t + U^o h_{t-1})$
 - $c_t = f_t \odot c_{t-1} + i_t \odot u_t$
 - $h_t = o_t \odot \tanh(c_t)$

Long Short-Term Memory (LSTM)

$$u_t = \tanh(W h_{t-1} + V x_t)$$

$$f_t = \text{sigmoid}(W_f h_{t-1} + V_f x_t)$$

$$i_t = \text{sigmoid}(W_i h_{t-1} + V_i x_t)$$

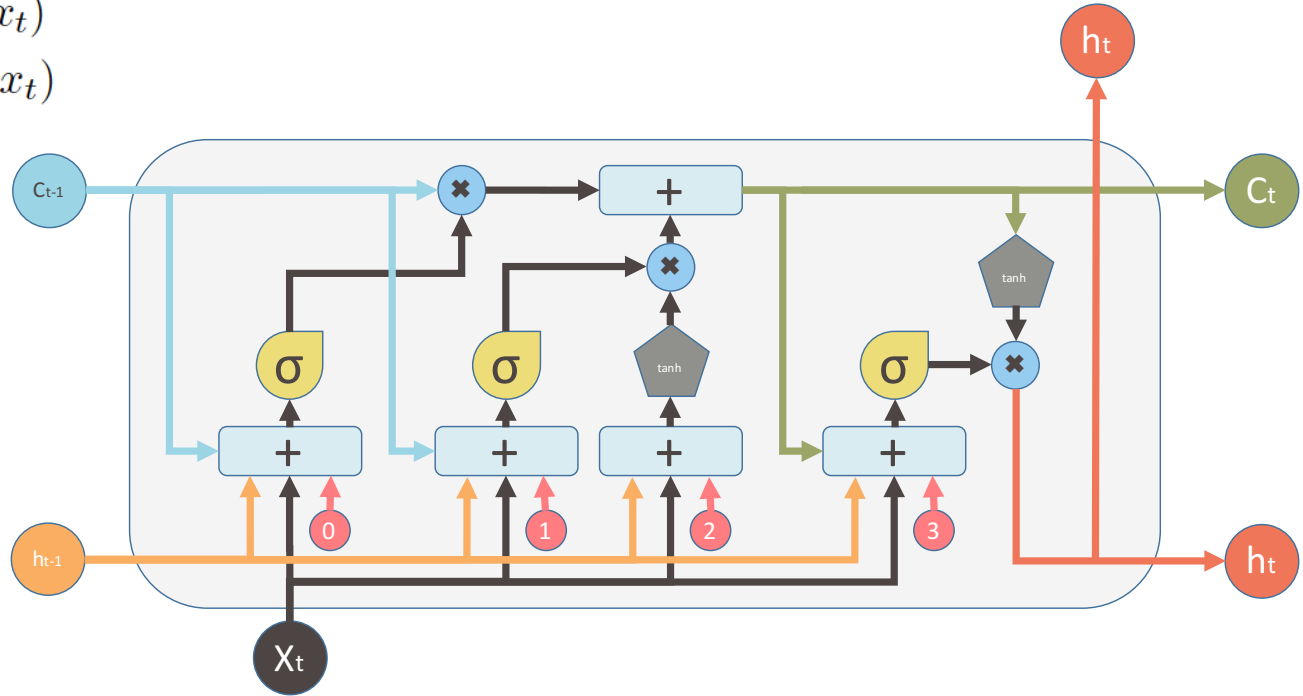
$$o_t = \text{sigmoid}(W_o h_{t-1} + V_o x_t)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot u_t$$

$$h_t = o_t \odot \tanh(c_t)$$

$$y_t = U h_t$$

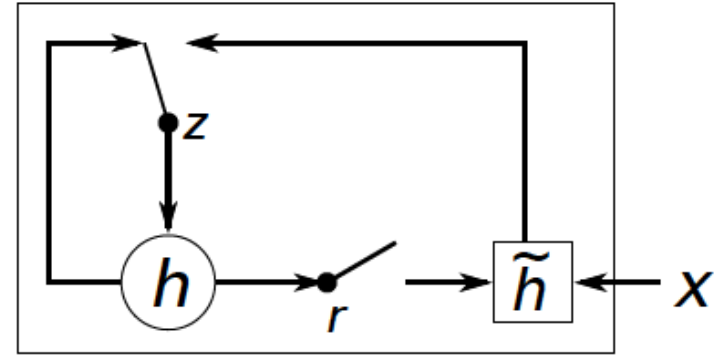
- Hochreiter & Schmidhuber, 1997
- LSTM = additive updates + gating



Gated Recurrent Units, GRU (Cho et al. 2014)

- Main ideas
 - Keep around memories to capture long distance dependencies
 - Allow error messages to flow at different strengths depending on the inputs
- Update gate
 - Based on current input and hidden state
 - $z_t = \sigma(W^z x_t + U^z h_{t-1})$
- Reset gate
 - Similarly but with different weights
 - $r_t = \sigma(W^r x_t + U^r h_{t-1})$

GRU



- New memory content

- $\tilde{h}_t = \tanh(Wx_t + r_t \odot Uh_{t-1})$

- Update gate z controls how much of past state should matter now

- If z closed to 1, then we can copy information in that unit through many time steps \rightarrow less vanishing gradient!

- If reset gate r unit is close to 0, then this ignores previous memory and only stores the new input information \rightarrow allows model to drop information that is irrelevant in the future

- Units with long term dependencies have active update gates z

- Units with short-term dependencies often have reset gates r very active

- Final memory at time step combines current and previous time steps

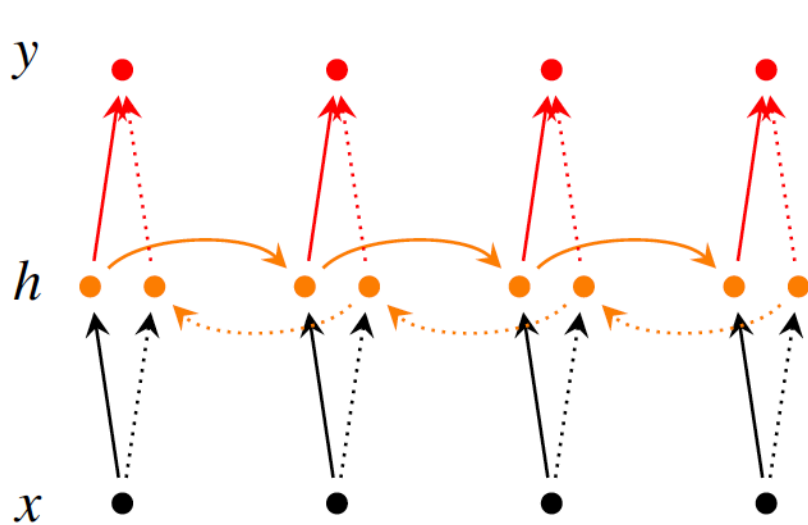
- $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}$

LSTM vs. GRU

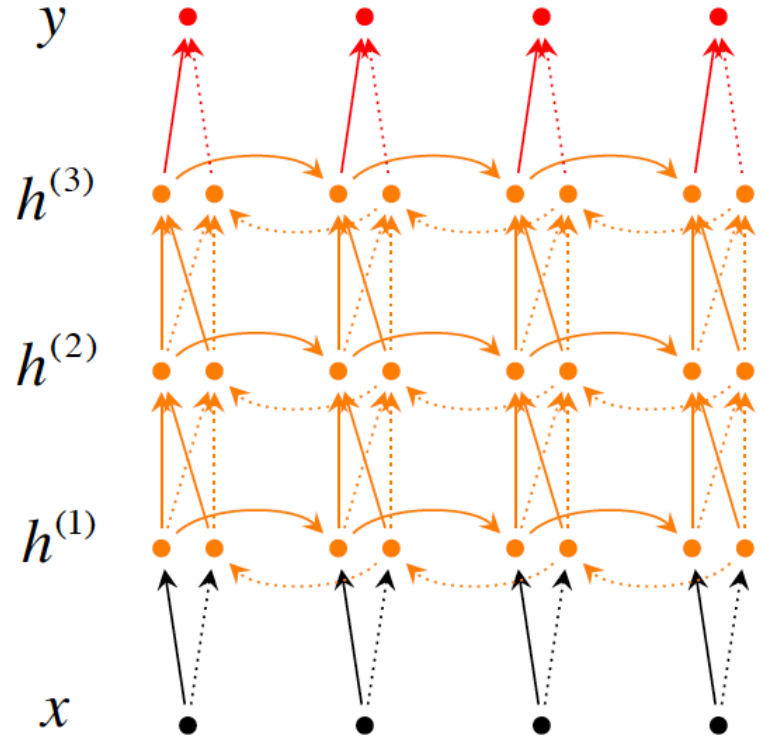
- No clear winner!
- Tuning hyperparameters like layer size is probably more important than picking the ideal architecture
- GRUs have fewer parameters and thus may train a bit faster or need less data to generalize
- If you have enough data, the greater expressive power of LSTMs may lead to better results.

More RNNs

- Bidirectional RNN

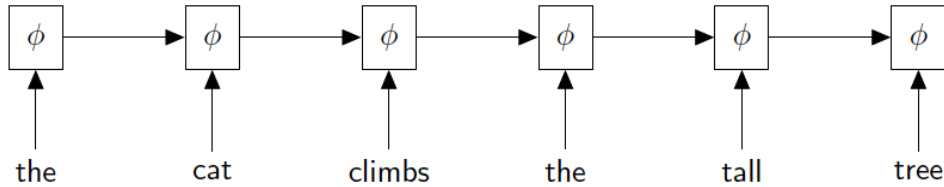


- Stack Bidirectional RNN

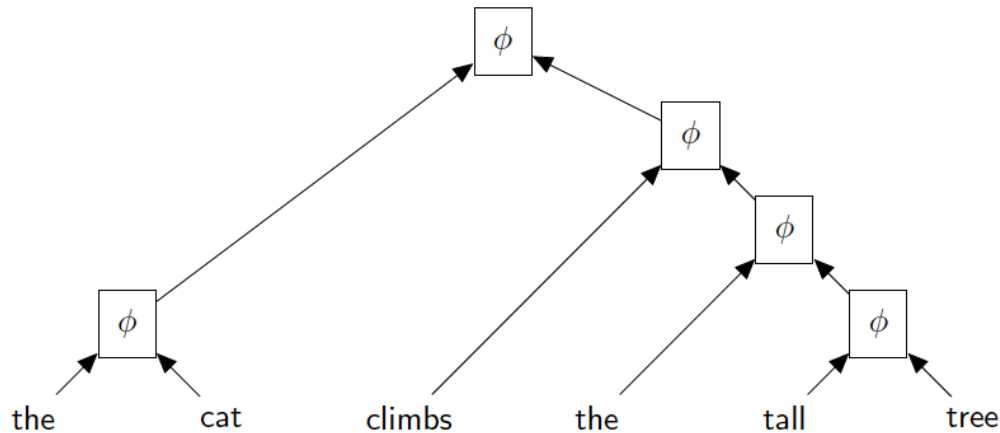


Tree-LSTMs

- Traditional Sequential Composition



- Tree-Structured Composition

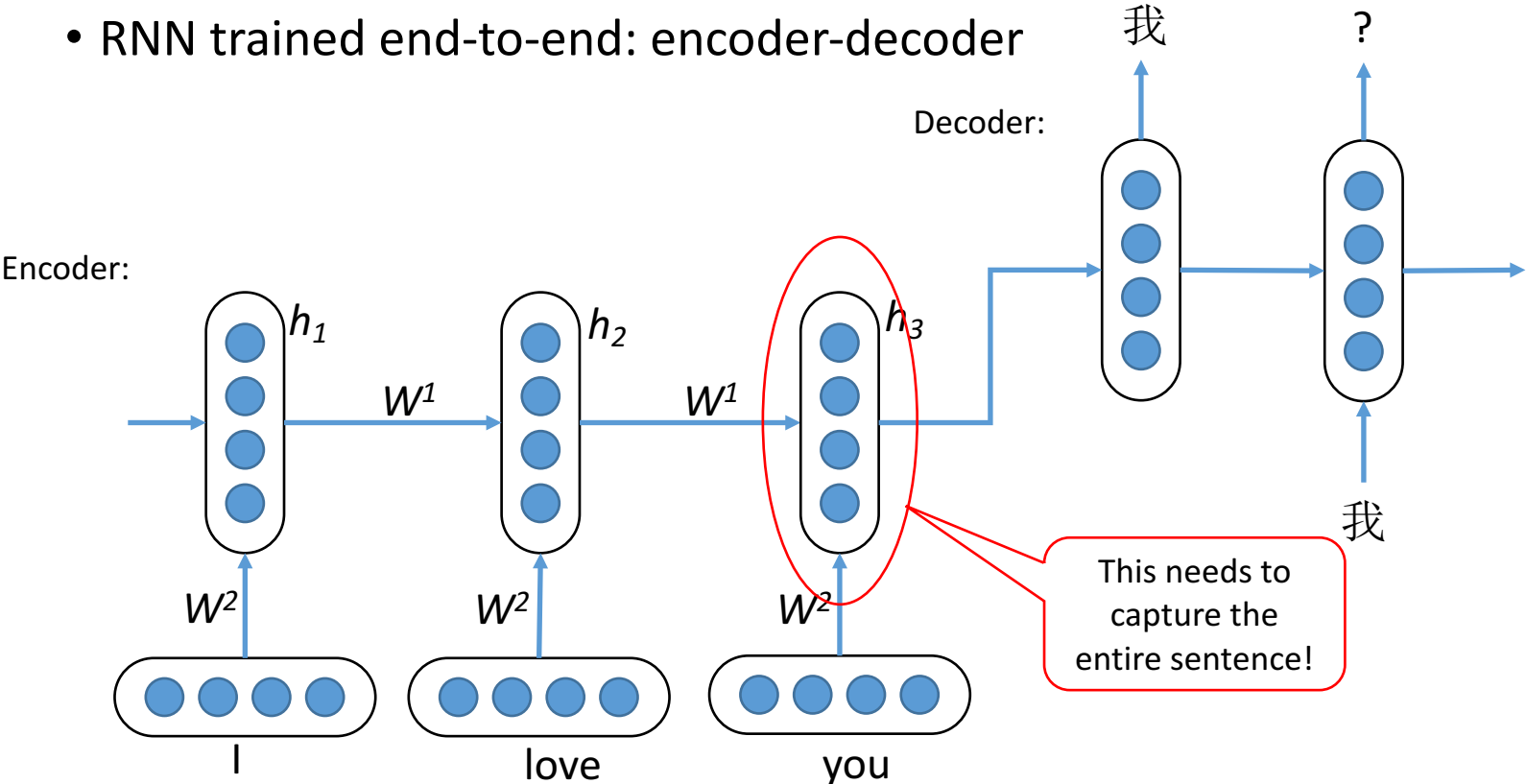


More Applications of RNN

- Neural Machine Translation
- Handwriting Generation
- Image Caption Generation
-

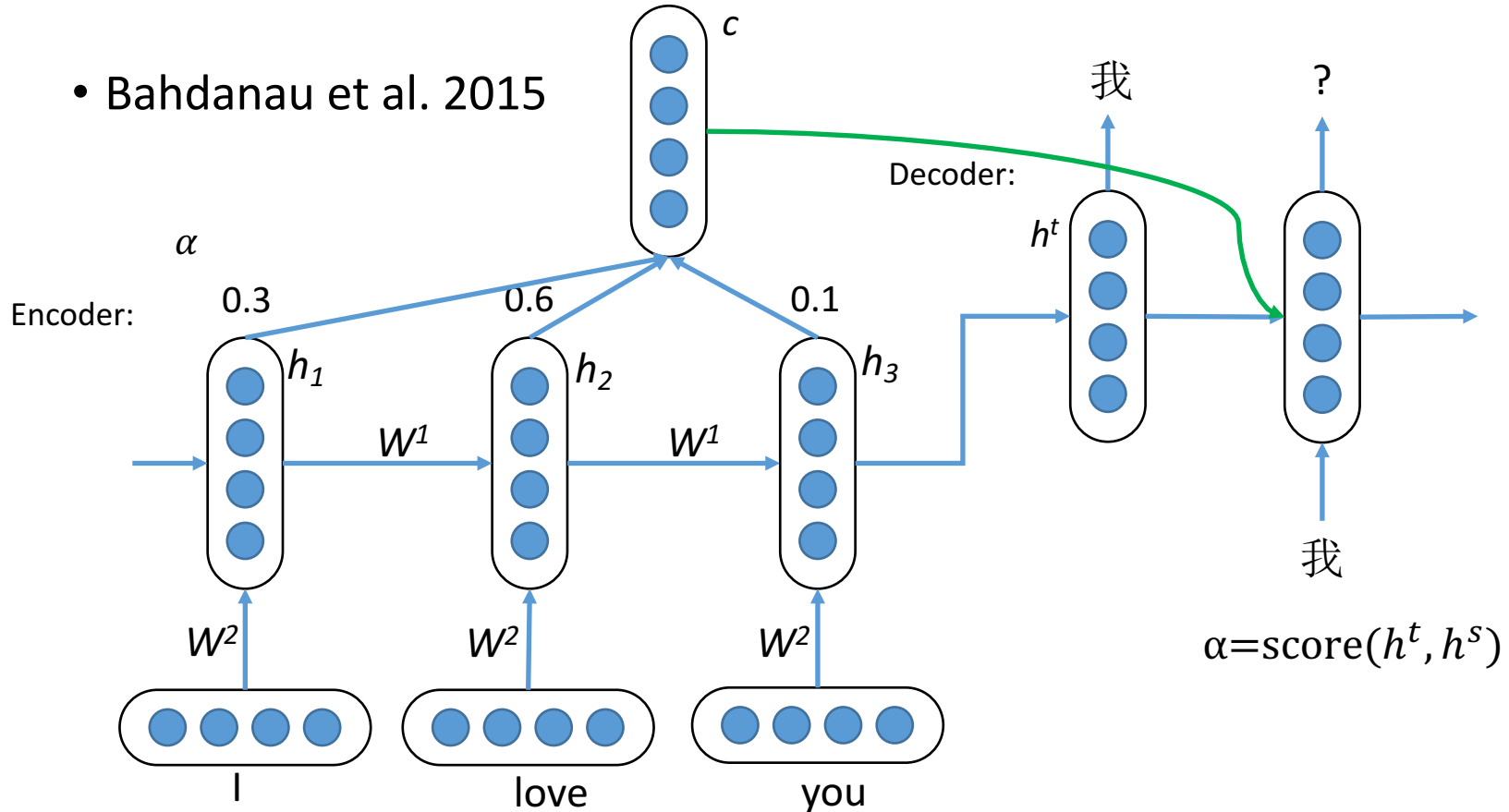
Neural Machine Translation

- RNN trained end-to-end: encoder-decoder



Attention Mechanism – Scoring

- Bahdanau et al. 2015



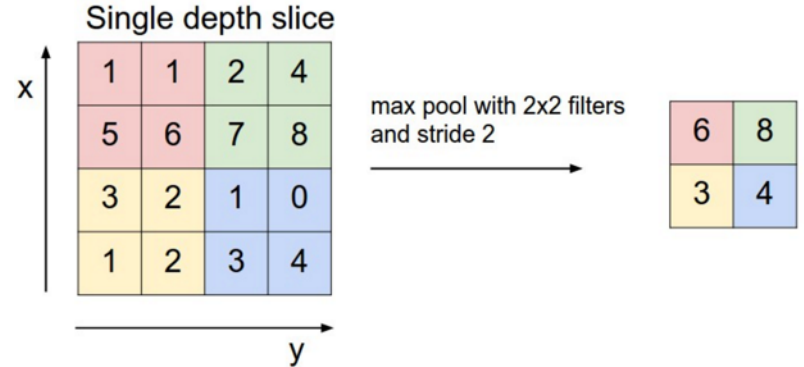
Convolution Neural Network

1 _{x1}	1 _{x0}	1 _{x1}	0	0
0 _{x0}	1 _{x1}	1 _{x0}	1	0
0 _{x1}	0 _{x0}	1 _{x1}	1	1
0	0	1	1	0
0	1	1	0	0

Image

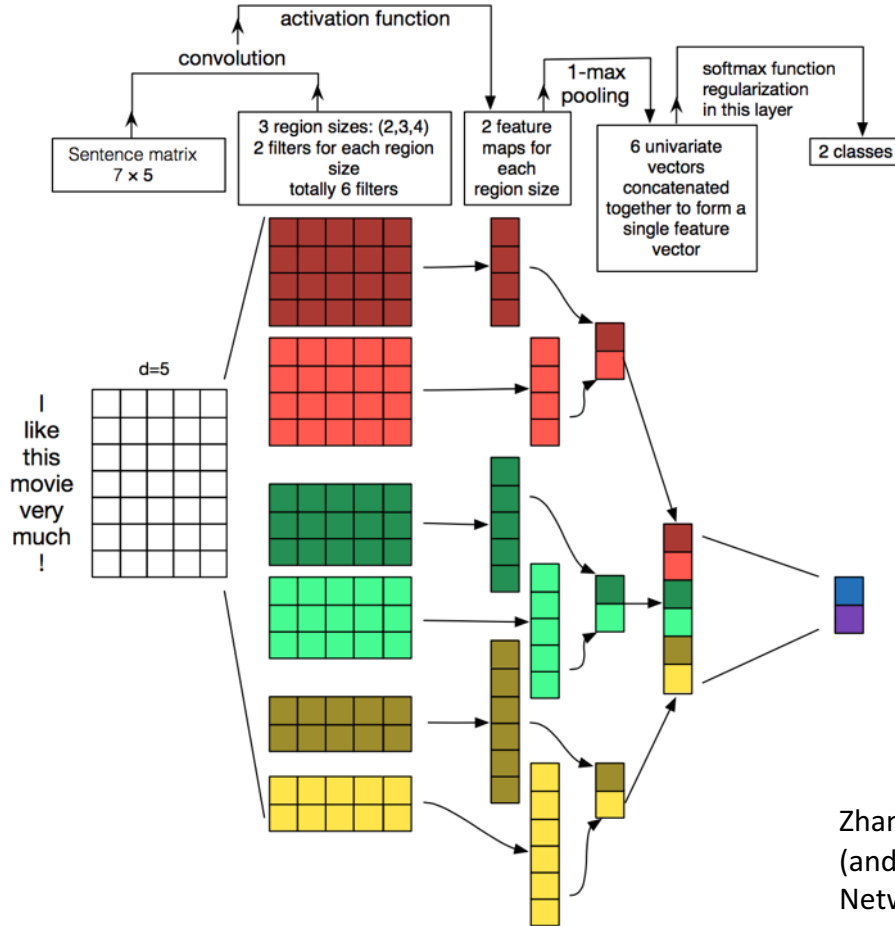
4		

Convolved
Feature



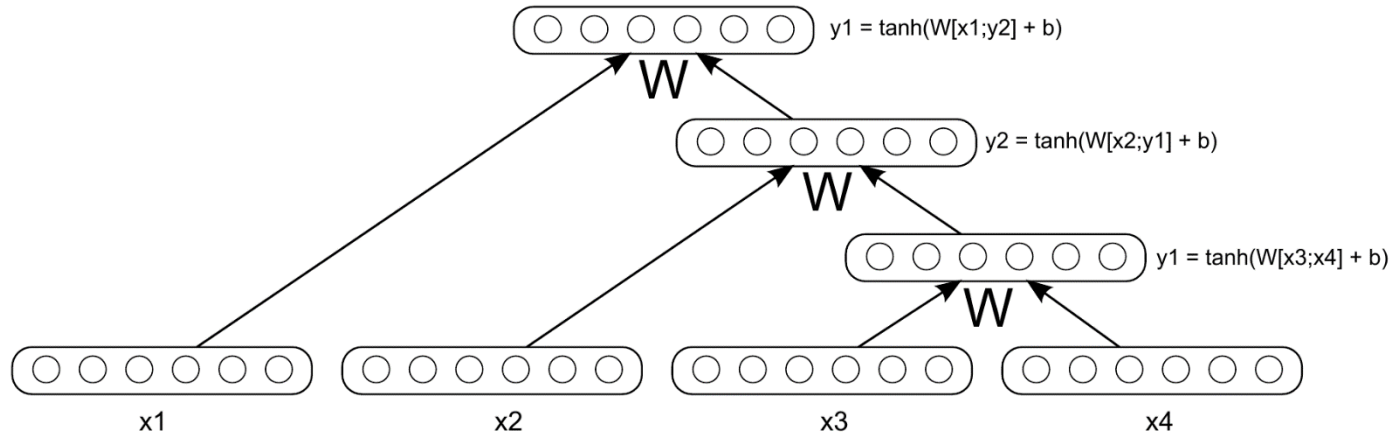
Pooling

CNN for NLP



Zhang, Y., & Wallace, B. (2015). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification.

Recursive Neural Network



Socher, R., Manning, C., & Ng, A. (2011). Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Network. NIPS.