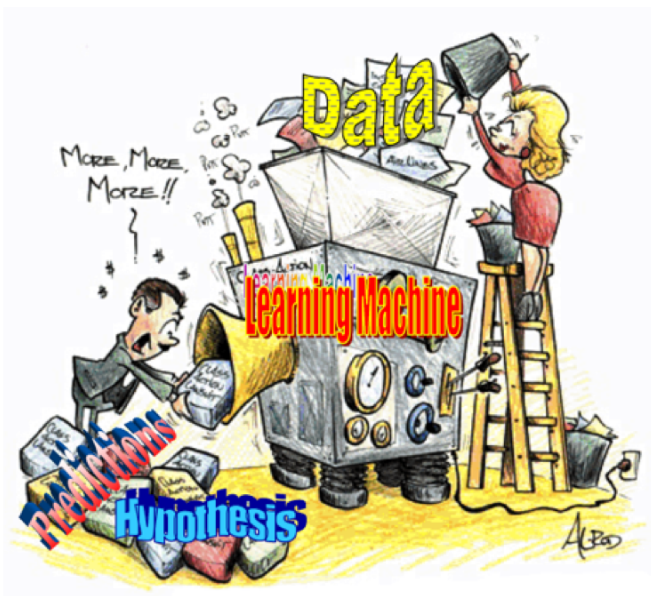
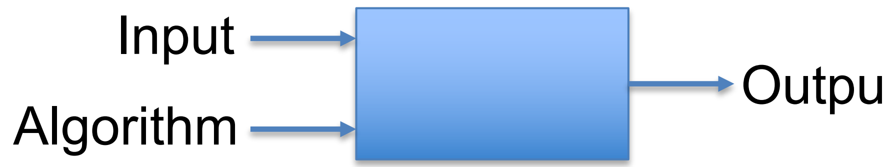
Part 2: Deep Learning

Part 2.1: Deep Learning Background

What is Machine Learning?

- From Data to Knowledge



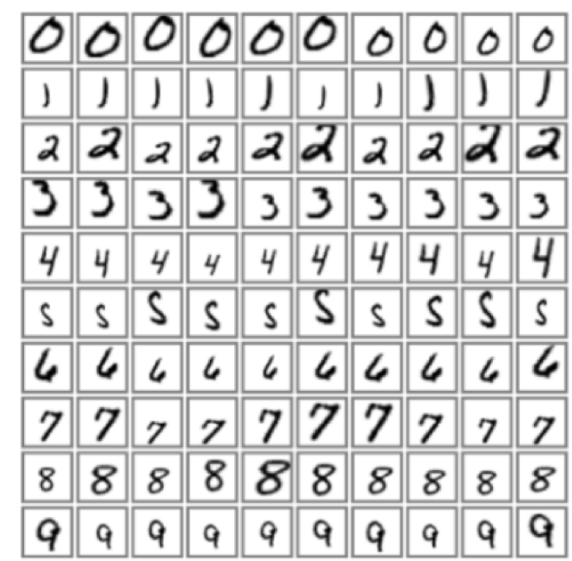


Traditional Program

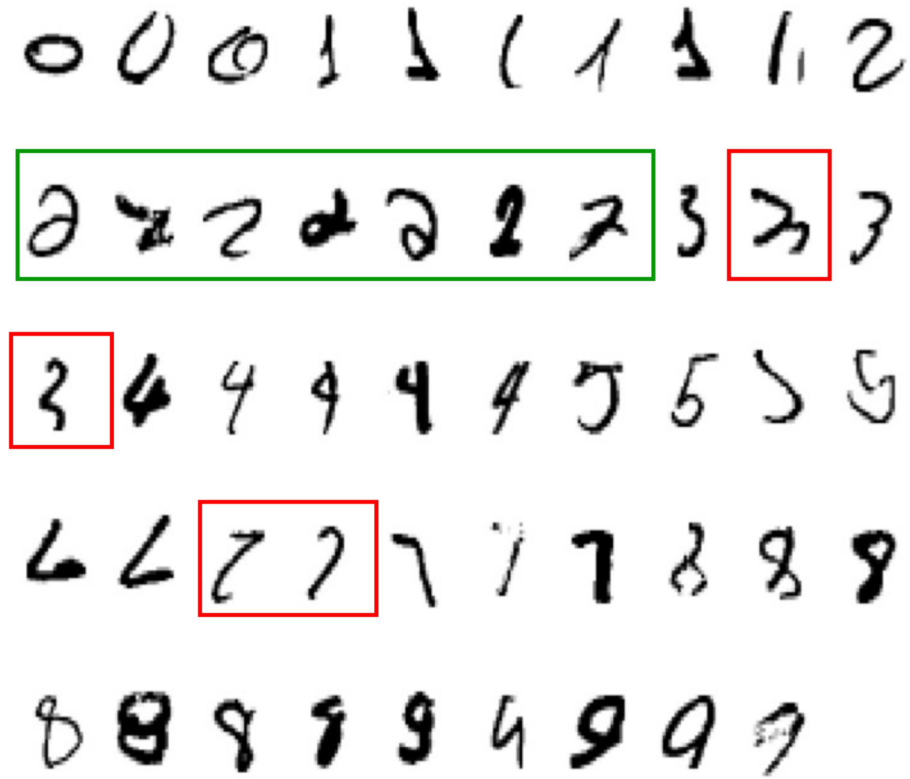
ML Program

A Standard Example of ML

- The MNIST (Modified NIST) database of hand-written digits recognition
  - Publicly available
  - A huge amount about how well various ML methods do on it
  - 60,000 + 10,000 hand-written digits (28x28 pixels each)

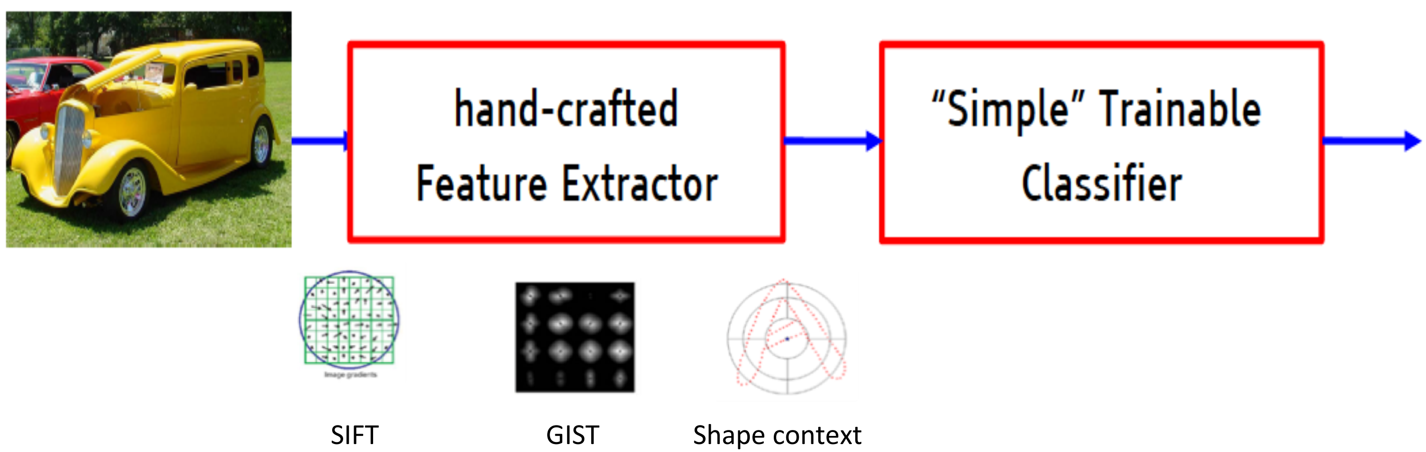


Very hard to say what makes a 2



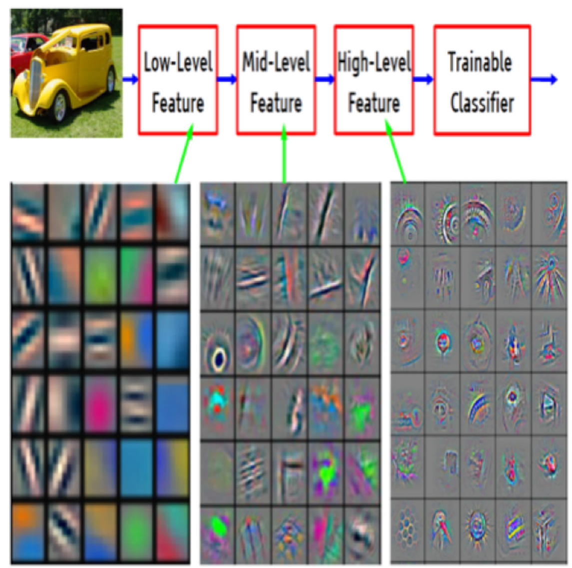
Traditional Model (before 2012)

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



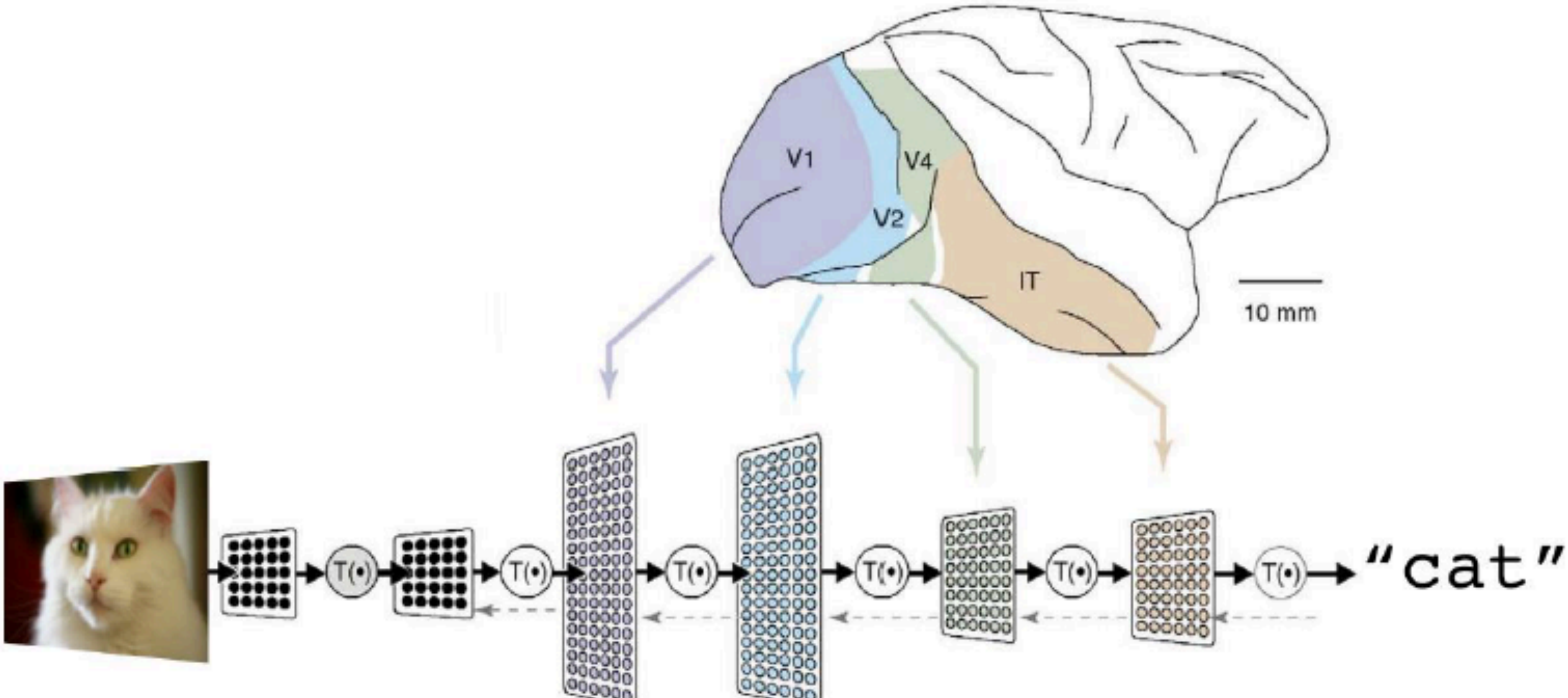
Deep Learning (after 2012)

- 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

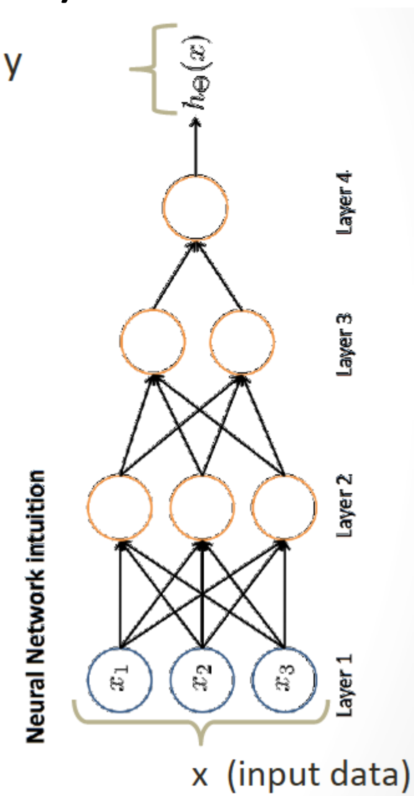
- 1980s technology (Neural Networks)

(label) y

Supervised learning

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

x = 



About Neural Networks

- Pros
  - Simple to learn $p(y|x)$
  - Performance is OK for shallow nets
- Cons
  - Trouble with > 3 layers
  - Overfitts
  - Slow to train

Deep Learning beats NN

- Pros
  - Simple to learn p(y|x)
  - Performance is OK for shallow nets
- Cons
  - Trou with layers
  - Overfitt
  - Slow trai

- New activation functions: ReLU, …
- Gated mechanism

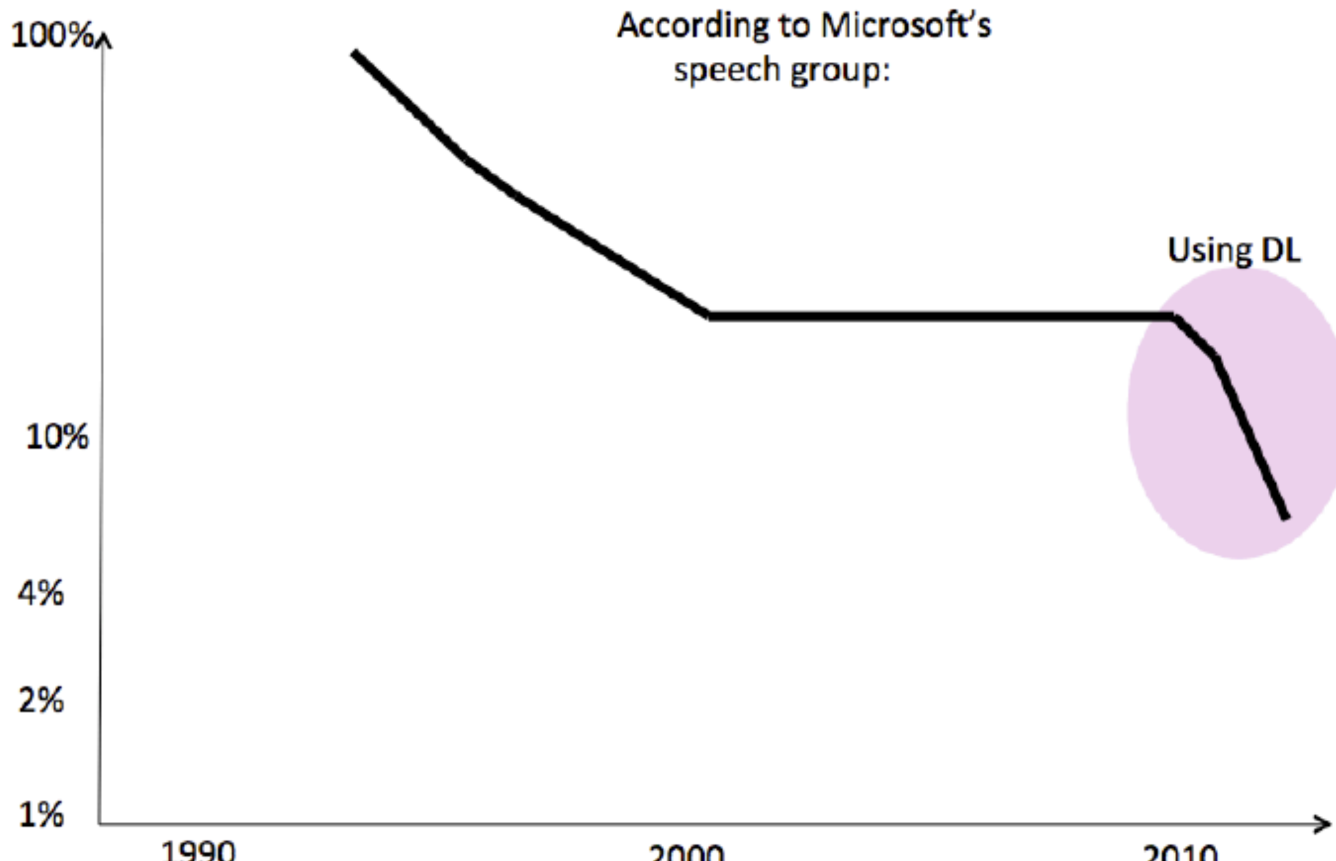
- Dropout
- Maxout
- Stochastic Pooling

GPU

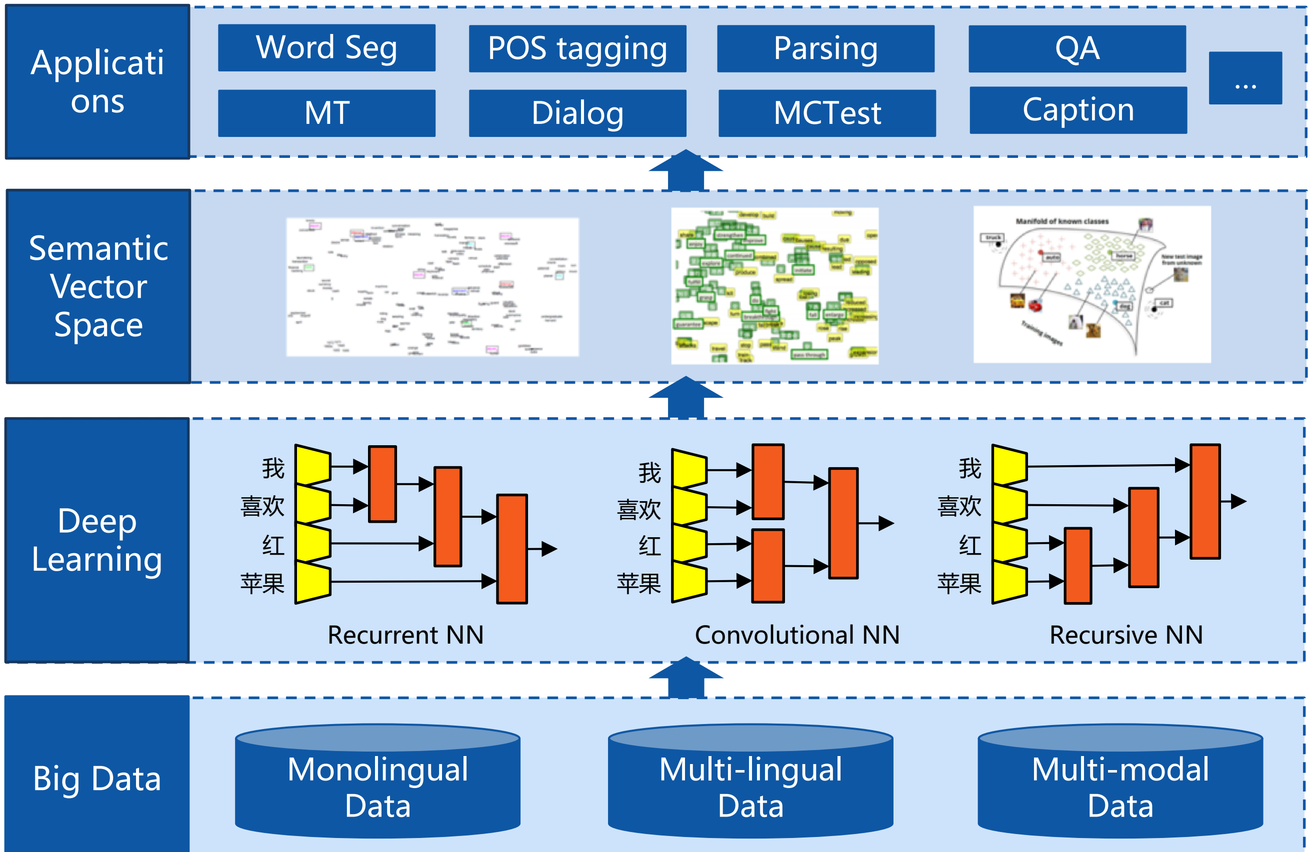
Results on MNIST

- 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 for Speech Recognition

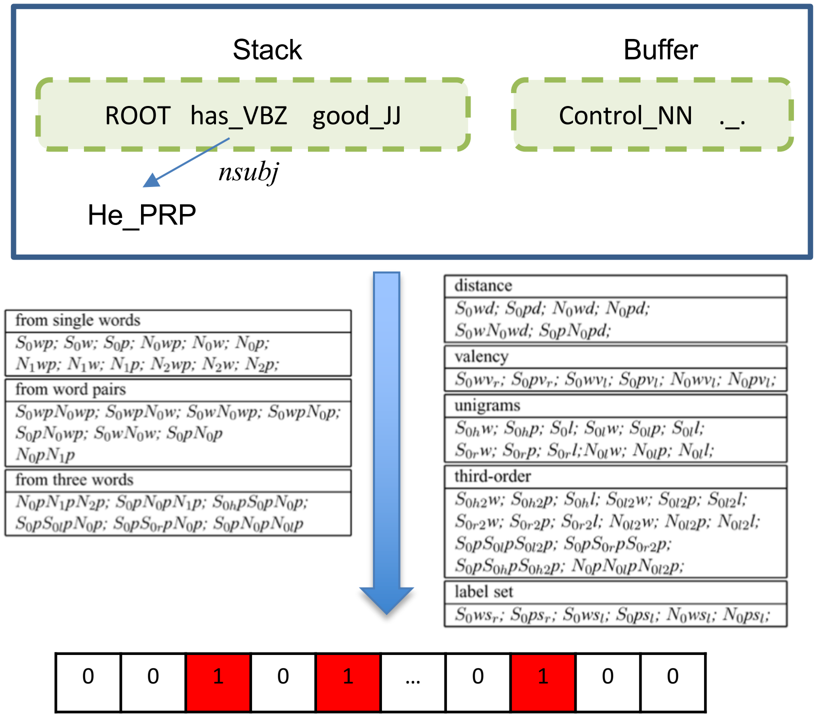


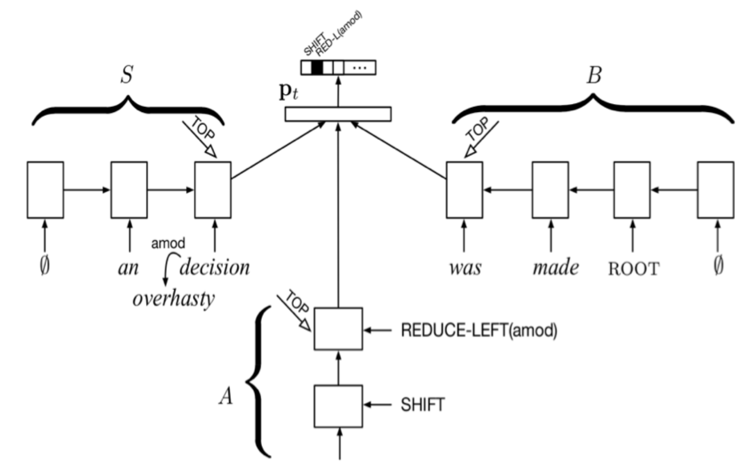
DL for NLP: Representation Learning



DL for NLP: End-to-End Learning

Configuration



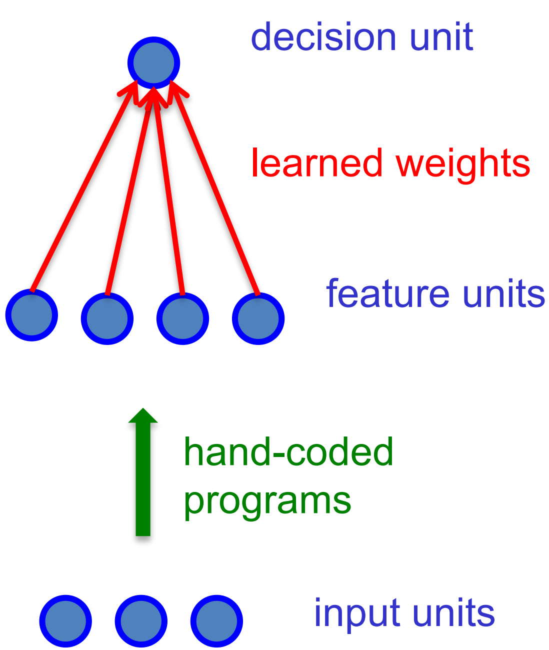


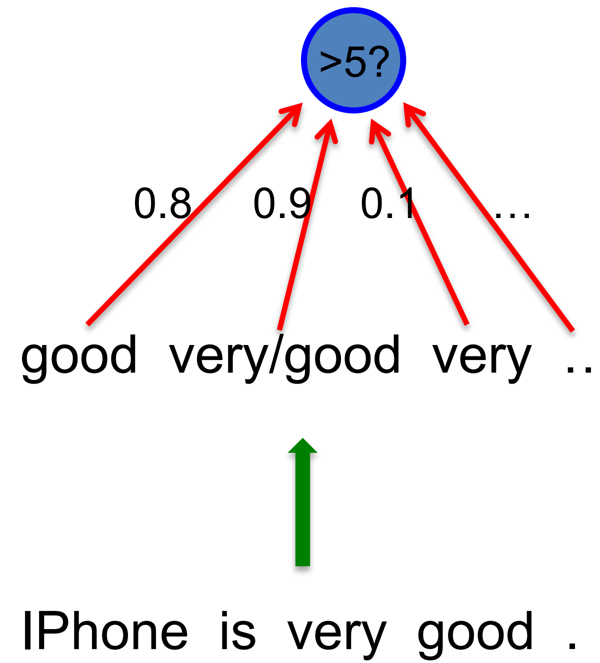
Traditional Parser

Stack-LSTM Parser

Part 2.2: Feedforward Neural Networks

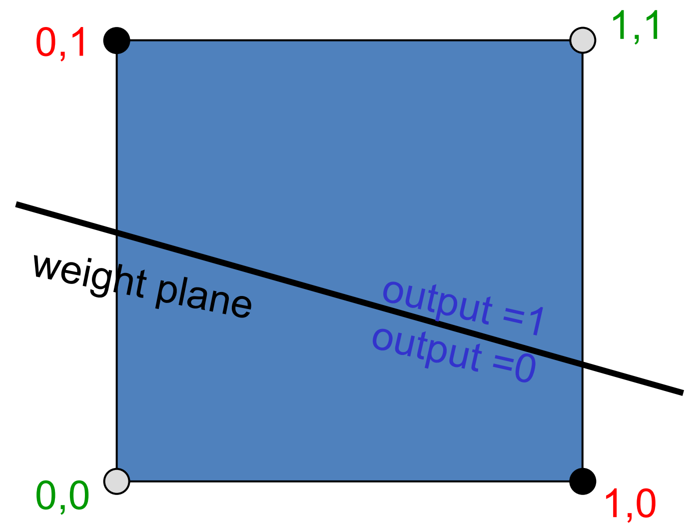
The Standard Perceptron Architecture





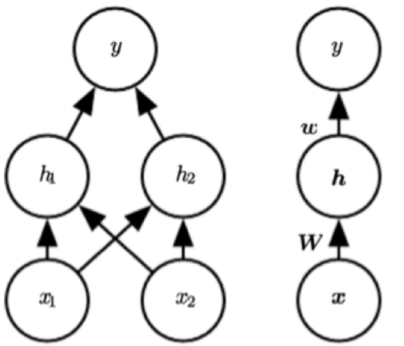
The Limitations of Perceptrons

- 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

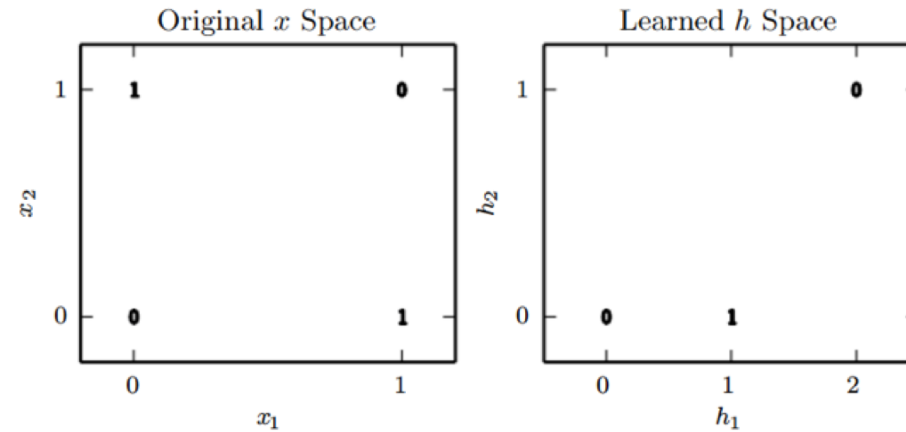


$$\boldsymbol{W} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix},$$

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

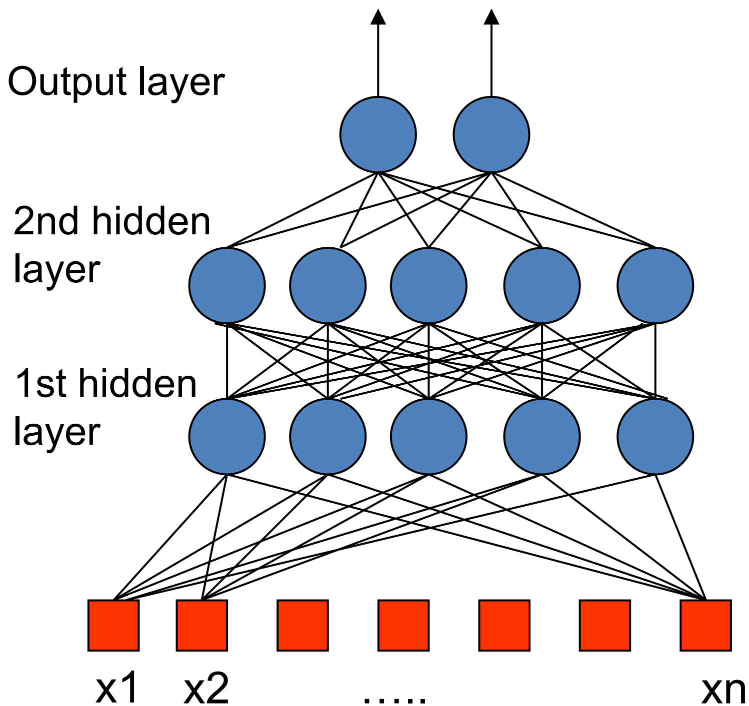
$$\boldsymbol{w} = \begin{bmatrix} 1 \\ -2 \end{bmatrix},$$

$$b = 0.$$



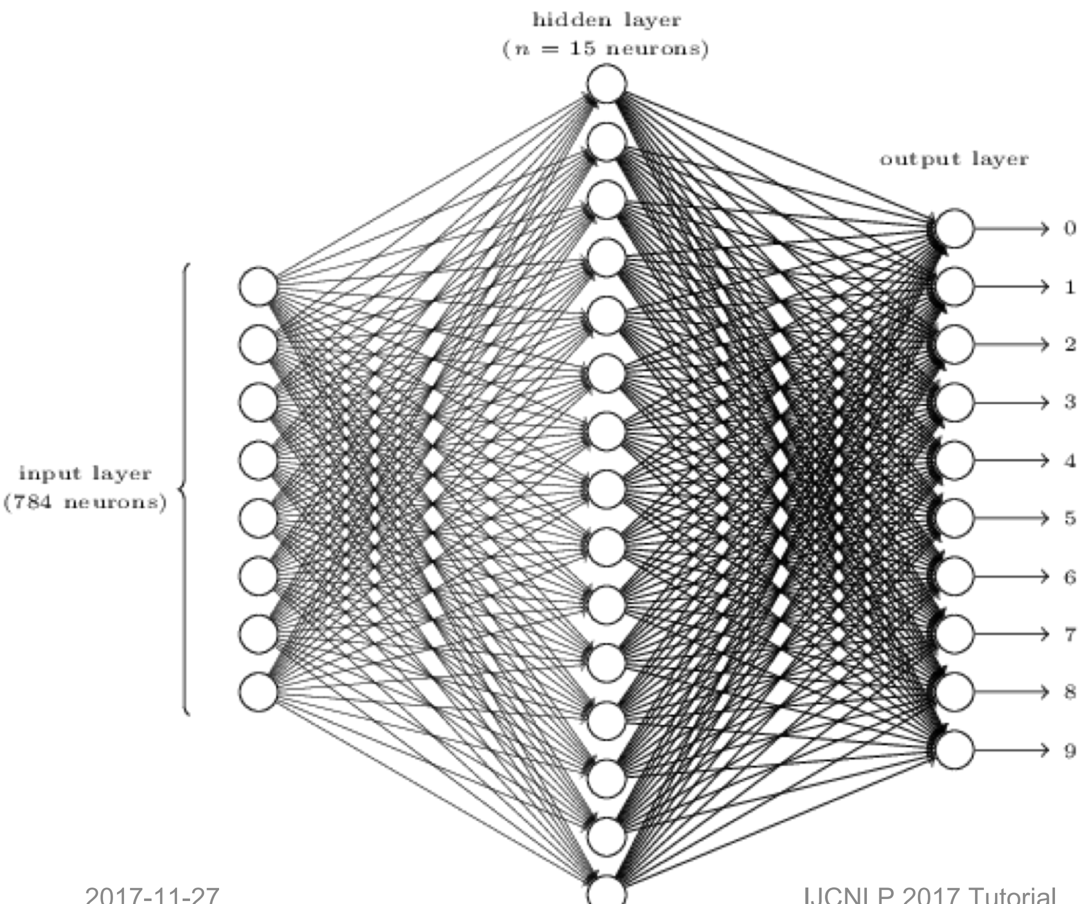
$$f(\boldsymbol{x}; \boldsymbol{W}, \boldsymbol{c}, \boldsymbol{w}, b) = \boldsymbol{w}^\top \max\{0, \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{c}\} + b.$$

Feedforward Neural Networks

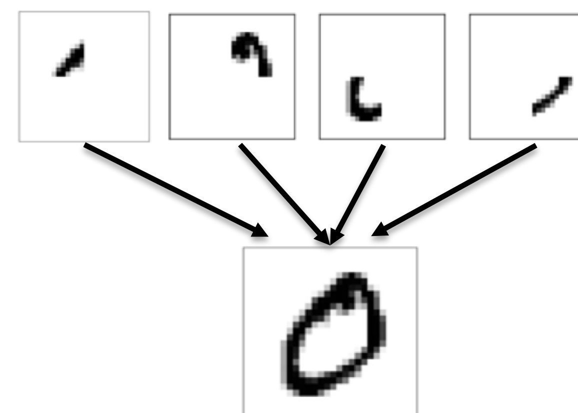


- Multi-layer Perceptron (**MLP**)
- The information is propagated from the inputs to the outputs
- NO cycle between outputs and inputs
- Learning the weights of hidden units is equivalent to **learning features**
- Networks without hidden layers are very limited in the input-output mappings
  - More 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

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

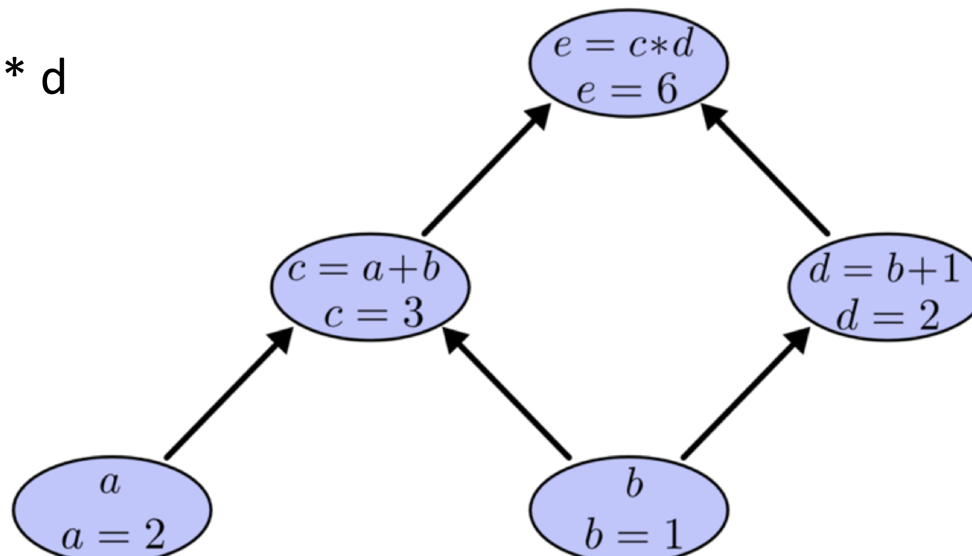
- Stochastic Gradient Descent (SGD)
  - Minibatch SGD (m > 1), Online GD (m = 1)

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

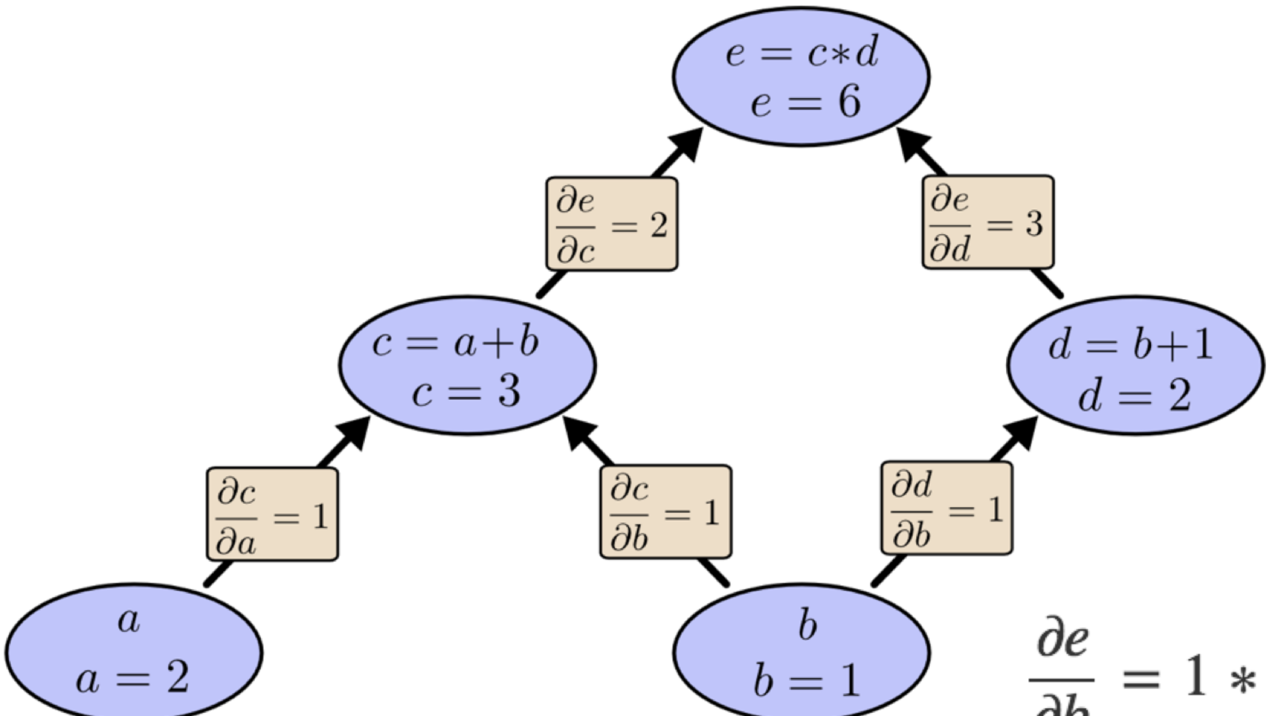
Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \boxed{\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
  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



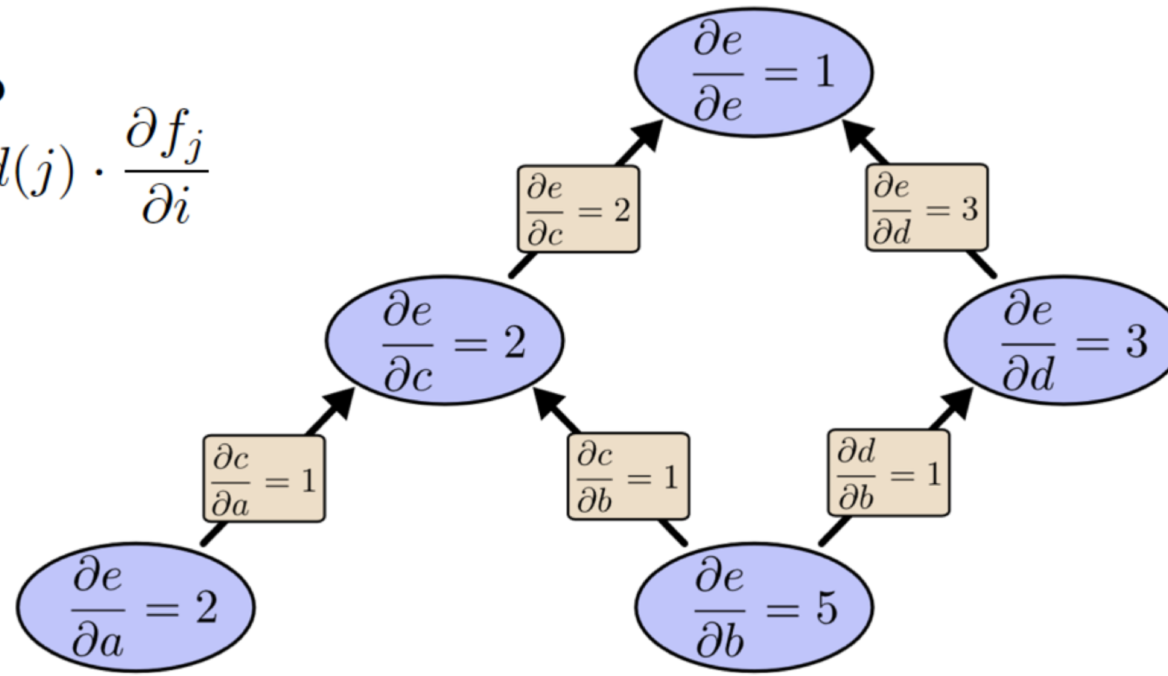
Derivatives on Computational Graphs



$$\frac{\partial e}{\partial b} = 1 * 2 + 1 * 3$$

Computational Graph Backward Pass (Backpropagation)

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



Part 2.3: Recurrent and Other Neural Networks

Language Models

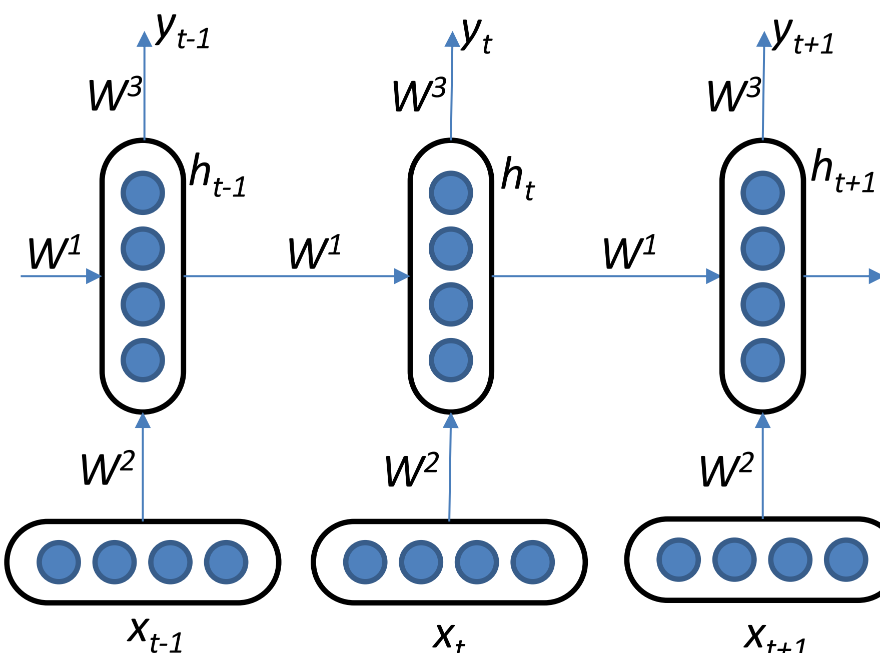
- A language model computes a probability for a sequence of word: $P(w_1, \cdots w_n)$ or predicts a probability for the next word: $P(w_{n+1}|w_1, \cdots w_n)$
- Useful for machine translation, speech recognition, and so on
  - Word ordering
    - *P*(the cat is small) > *P*(small the is cat)
  - Word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

Traditional Language Models

- An incorrect but necessary **Markov assumption**!
  - Probability is usually conditioned on *n* previous words
  - $P(w_1, \cdots w_n) = \prod_{i=1}^{m} P(w_i | w_1, \cdots, w_{i-1}) \approx \prod_{i=1}^{m} P(w_i | w_{i-(n-1)}, \cdots, w_{i-1})$
- Disadvantages
  - There are A LOT of n-grams!
  - Cannot see too long history

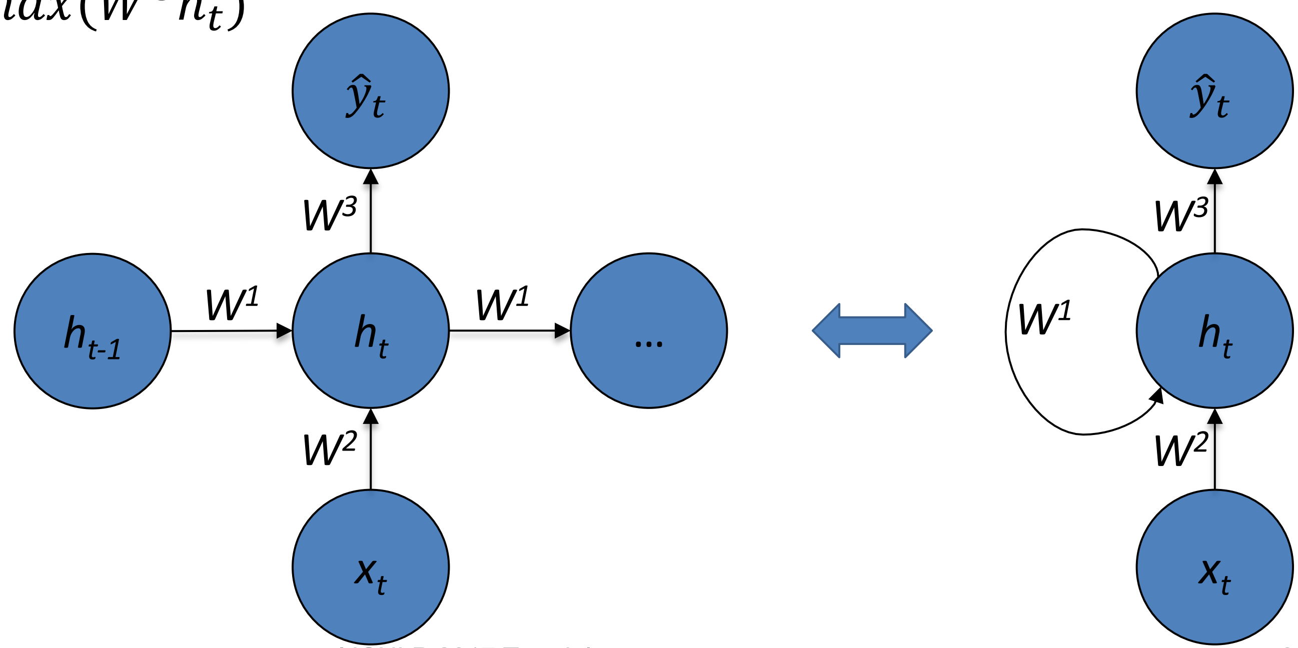
Recurrent Neural Networks (RNNs)

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



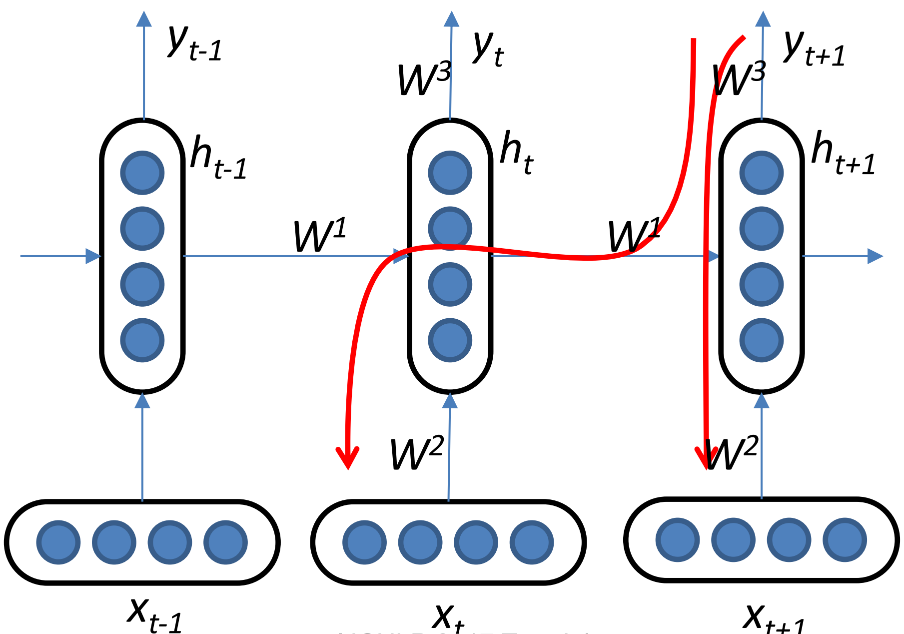
Recurrent Neural Networks (RNNs)

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

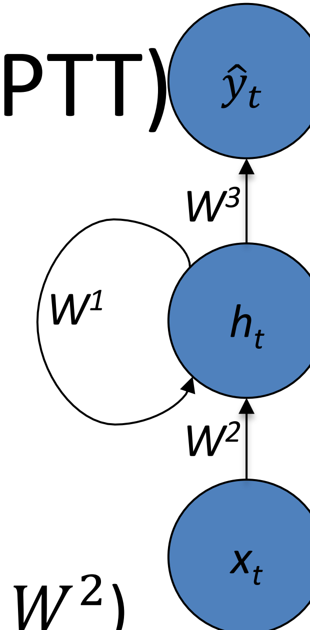


Training RNNs is hard

- 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

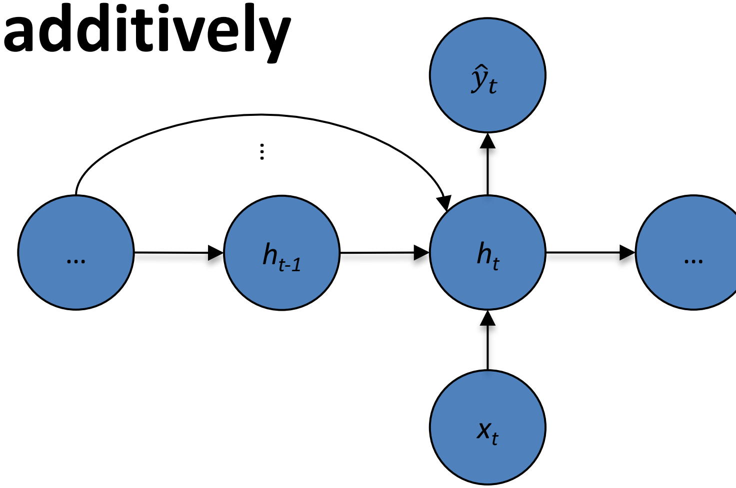
- $\frac{\partial E_t}{\partial W} = \sum_{k=1}^{t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \boxed{\frac{\partial h_t}{\partial h_k}} \frac{\partial h_k}{\partial W}$, $h_t = \tanh(W^1 h_{t-1} + W^2 x_t)$
- $\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^{t} \frac{\partial h_j}{\partial h_{j-1}} = \prod_{j=k+1}^{t} W^1 \mathrm{diag}[\tanh'(\cdots)]$
- $\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \le \gamma \|W^1\| \le \gamma \lambda_1$
  - where γ is bound $\|\mathrm{diag}[\tanh'(\cdots)]\|$, $\lambda_1$is the largest singular value of W^1
- $\left\| \frac{\partial h_t}{\partial h_k} \right\| \le (\gamma \lambda_1)^{t-k}$→0
  - if $\gamma\lambda_1 < 1$, this can become very small (vanishing gradient)
  - if $\gamma\lambda_1 > 1$, this can become very large (exploding gradient)
    - Trick for exploding gradient: clipping trick (set a threshold)

A “solution”

- Intuition
  - Ensure $\gamma\lambda_1 \geq 1$ → 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
    - $h_t = h_{t-1} + u_t = h_{t-2} + u_{t-1} + u_t = \cdots$



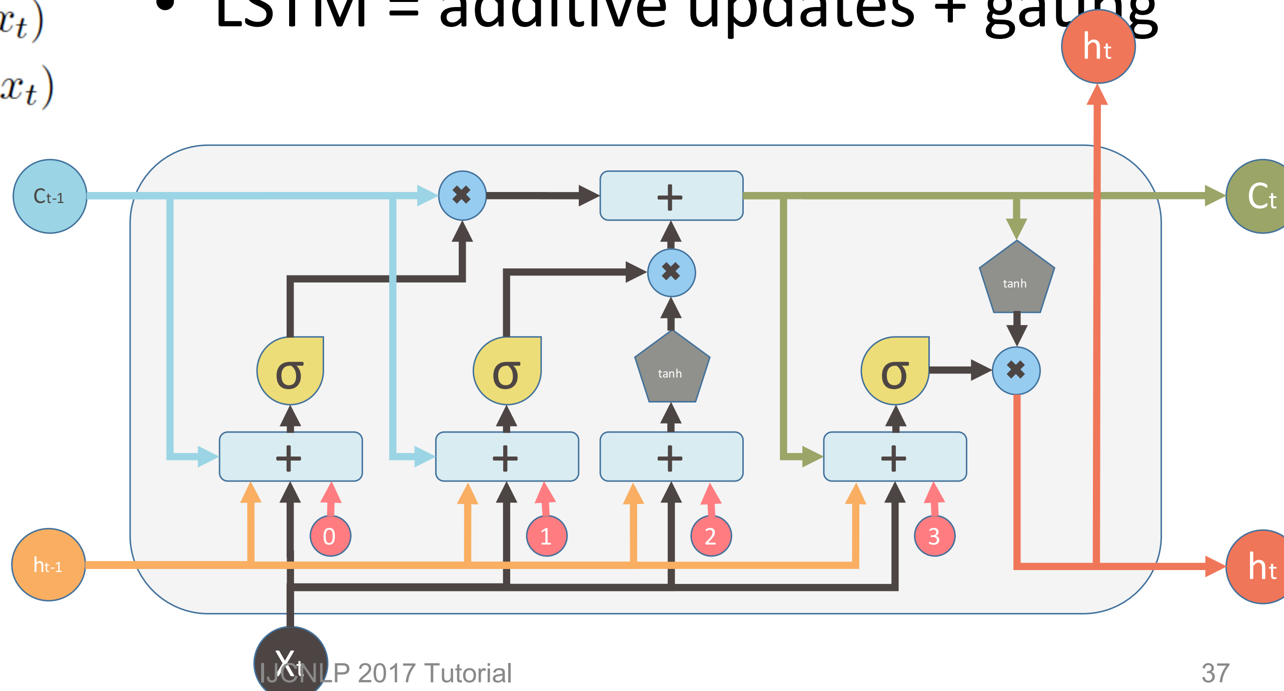
A better "solution" (cont.)

- Interpolate between old state and new state ("choosing to **forget**")
  - $f_t = \sigma\left(W^f x_t + U^f h_{t-1}\right)$
  - $h_t = f_t \odot h_{t-1} + (1 - f_t) \odot u_t$
- Introduce a separate **input gate** i_t
  - $i_t = \sigma\left(W^i x_t + U^i h_{t-1}\right)$
  - $h_t = f_t \odot h_{t-1} + i_t \odot u_t$
- 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)

$$u_t = \tanh\left(W h_{t-1} + V x_t\right)$$
$$f_t = \text{sigmoid}\left(W_f h_{t-1} + V_f x_t\right)$$
$$i_t = \text{sigmoid}\left(W_i h_{t-1} + V_i x_t\right)$$
$$o_t = \text{sigmoid}\left(W_o h_{t-1} + V_o x_t\right)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot u_t$$
$$h_t = o_t \odot \tanh(c_t)$$
$$y_t = U h_t$$

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



Gated Recurrent Unites, 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

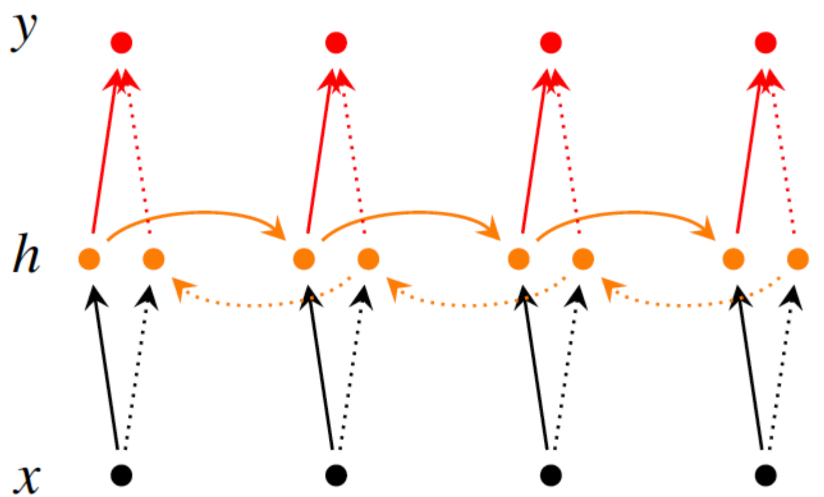
- Memory at time step combines current and previous time steps
  - $h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}$
  - Update gate *z* controls how much of past state should matter now
    - If *z* closed to 1, then we can copy information in that unit through many time steps → less vanishing gradient!
- New memory content
  - $\tilde{h}_t = \tanh(Wx_t + r_t \odot Uh_{t-1})$
  - If reset gate *r* unit is close to 0, then this ignores previous memory and only stores the new input information → allows model to drop information that is irrelevant in the future

LSTM vs. GRU

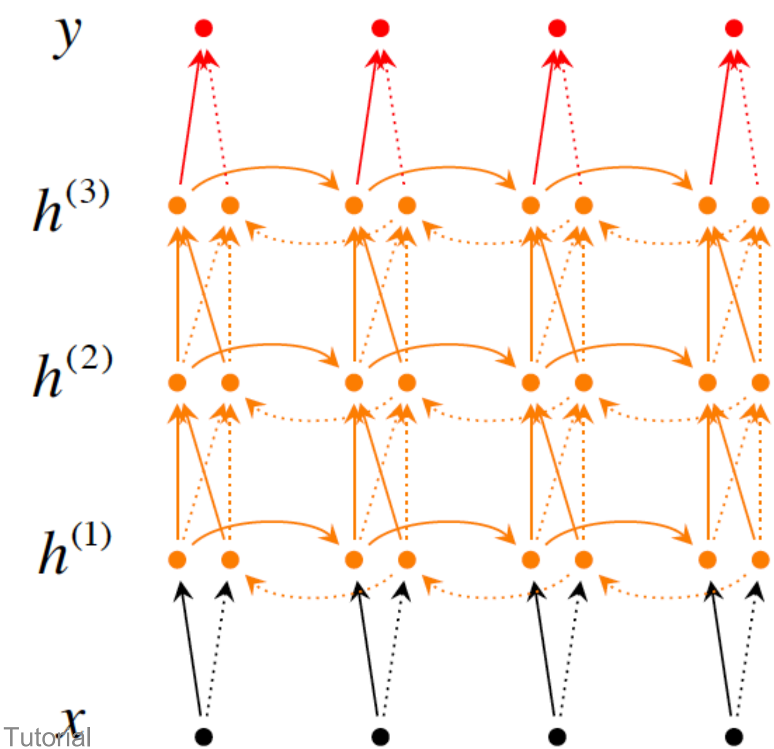
- No clear winner!
- Tuning hyperparameters like layer size is probably more important than picking the ideal architecture
- GRUs have fewer parameters and thus may train a bit faster or need less data to generalize
- If you have enough data, the greater expressive power of LSTMs may lead to better results.

More RNNs

- Bidirectional RNN

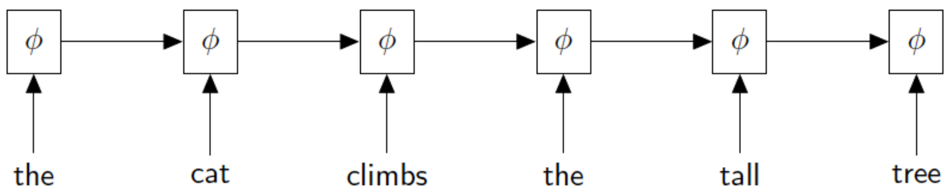


- Stack Bidirectional RNN

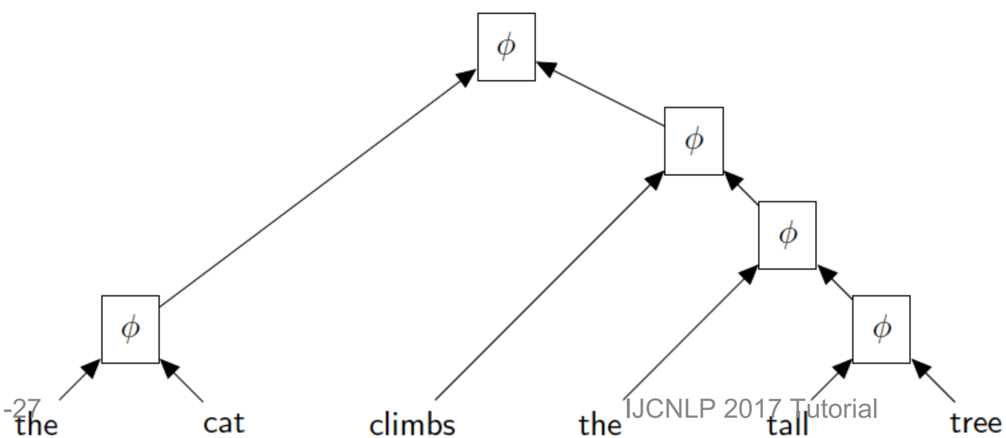


Tree-LSTMs

- Traditional Sequential Composition



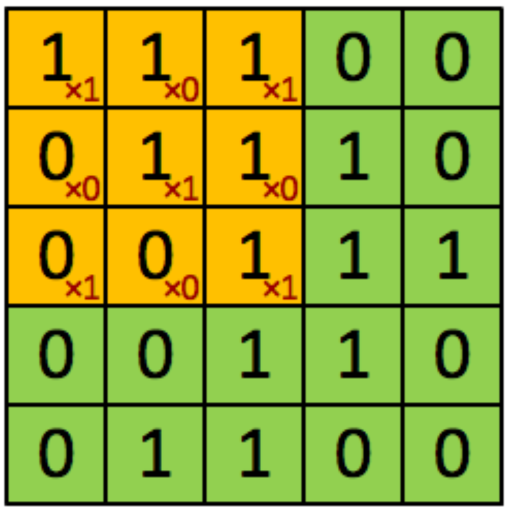
- Tree-Structured Composition



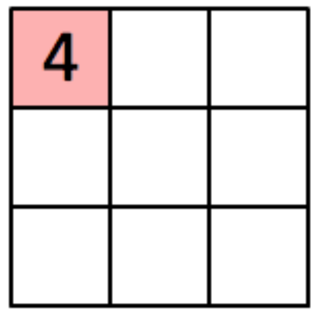
More Applications of RNN

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

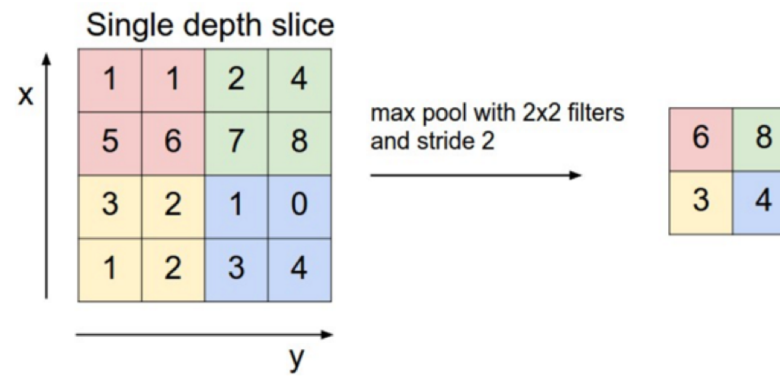
Convolution Neural Network



Image



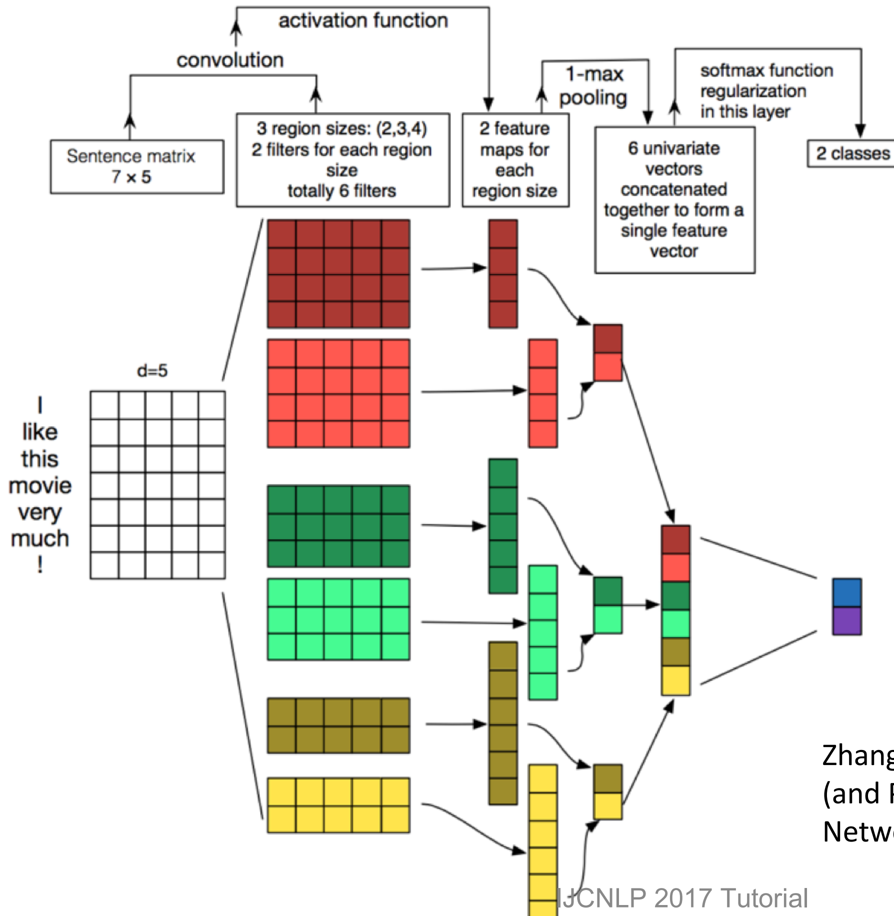
Convolved Feature



Pooling

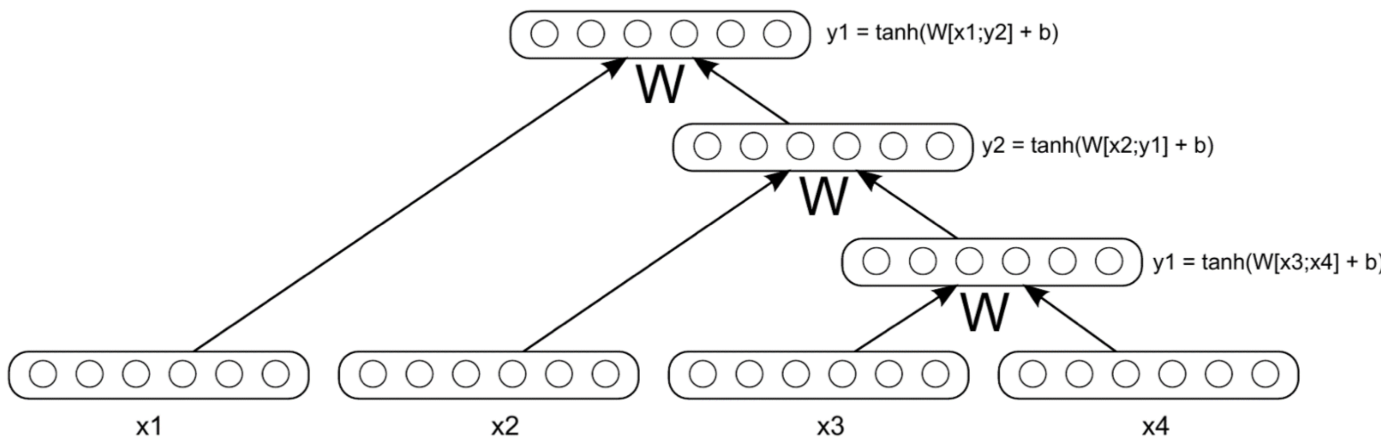
CS231n Convolutional Neural Network for Visual Recognition.

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.

Summary

- Deep Learning
  - Representation Learning
  - End-to-end Learning
- Popular Networks
  - Feedforward Neural Networks
  - Recurrent Neural Networks
  - Convolutional Neural Networks