Deep Learning: An Overview

-- Nguyen Hung Son --

Acknowledgements

Many of the pictures, results, and other materials are taken from:

Aarti Singh, Carnegie Mellon University

Andrew Ng, Stanford University

Barnabas Poczos, Carnegie Mellon University

Christopher Manning, Stanford University

Geoffrey Hinton, Google & University of Toronto

Richard Socher, MetaMind

Richard Turner, University of Cambridge

Yann LeCun, New York University

Yoshua Bengio, Universite de Montreal

Weifeng Li, Victor Benjamin, Xiao Liu, and Hsinchun Chen, University of Arizona

Outline

- Introduction
  - Motivation
  - Deep Learning Intuition
- Neural Network
  - Neuron, Feedforward algorithm, and Backpropagation algorithm
  - Limitations
- Deep Learning
  - Deep Belief Network: Autoencoder & Restricted Boltzman Machine
  - Convolutional Neural Network
- Deep Learning for Text Mining
  - Word Embedding
  - Recurrent Neural Network
  - Recursive Neural Network
- Conclusions

Outline

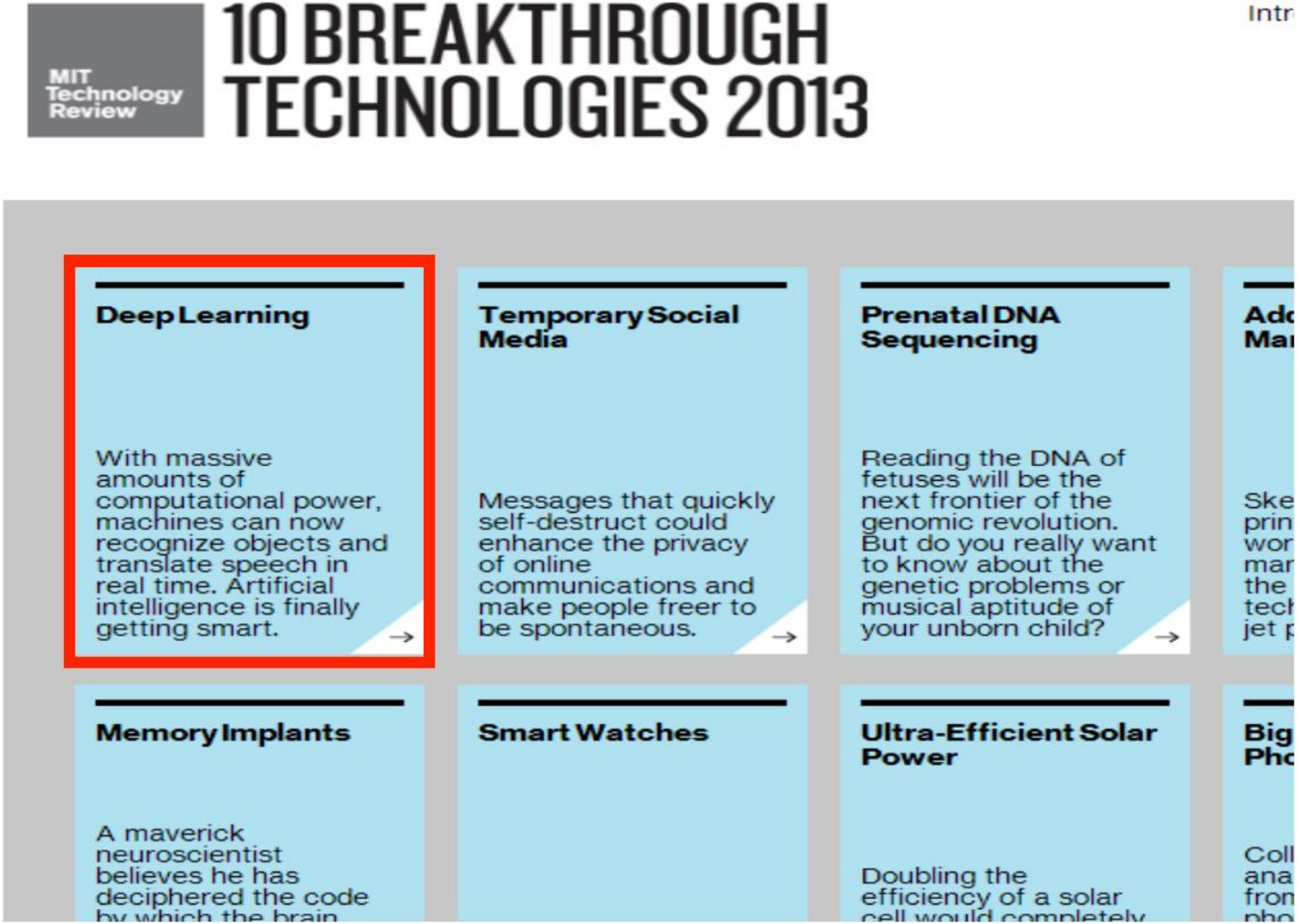
- Introduction
  - Motivation
  - Deep Learning Intuition
- Neural Network
  - Neuron, Feedforward algorithm, and Backpropagation algorithm
  - Limitations
- Deep Learning
  - Deep Belief Network: Autoencoder & Restricted Boltzman Machine
  - Convolutional Neural Network
- Deep Learning for Text Mining
  - Word Embedding
  - Recurrent Neural Network
  - Recursive Neural Network
- Conclusions

In "Nature" 27 January 2016:

"DeepMind's program AlphaGo beat Fan Hui, the European Go champion, five times out of five in tournament conditions..."

"AlphaGo was not preprogrammed to play Go: rather, it learned using a general-purpose algorithm that allowed it to interpret the game's patterns."

"...AlphaGo program applied **deep learning** in neural networks (convolutional NN) — brain-inspired programs in which connections between layers of simulated neurons are strengthened through examples and experience."


MIT Technology Review
10 BREAKTHROUGH TECHNOLOGIES 2013
Intr
Deep Learning
With massive amounts of computational power, machines can now recognize objects and translate speech in real time. Artificial intelligence is finally getting smart.
Temporary Social Media
Messages that quickly self-destruct could enhance the privacy of online communications and make people freer to be spontaneous.
Prenatal DNA Sequencing
Reading the DNA of fetuses will be the next frontier of the genomic revolution. But do you really want to know about the genetic problems or musical aptitude of your unborn child?
Memory Implants
A maverick neuroscientist believes he has deciphered the code
Smart Watches
Ultra-Efficient Solar Power
Doubling the efficiency of a solar

Deep Learning Today

- Advancement in speech recognition in the last 2 years
  - A few long-standing performance records were broken with deep learning methods
  - Microsoft and Google have both deployed DL-based speech recognition systems in their products
- Advancement in Computer Vision
  - Feature engineering is the bread-and-butter of a large portion of the CV community, which creates some resistance to feature learning
  - But the record holders on ImageNet and Semantic Segmentation are convolutional nets
- Advancement in Natural Language Processing
  - Fine-grained sentiment analysis, syntactic parsing
  - Language model, machine translation, question answering

Motivations for Deep Architectures

Insufficient depth can hurt

With shallow architecture (SVM, NB, KNN, etc.), the required number of nodes in the graph (i.e. computations, and also number of parameters, when we try to learn the function) may grow very large.

Many functions that can be represented efficiently with a deep architecture cannot be represented efficiently with a shallow one.

The brain has a deep architecture

The visual cortex shows a sequence of areas each of which contains a representation of the input, and signals flow from one to the next.

Note that representations in the brain are in between dense distributed and purely local: they are **sparse**: about 1% of neurons are active simultaneously in the brain.

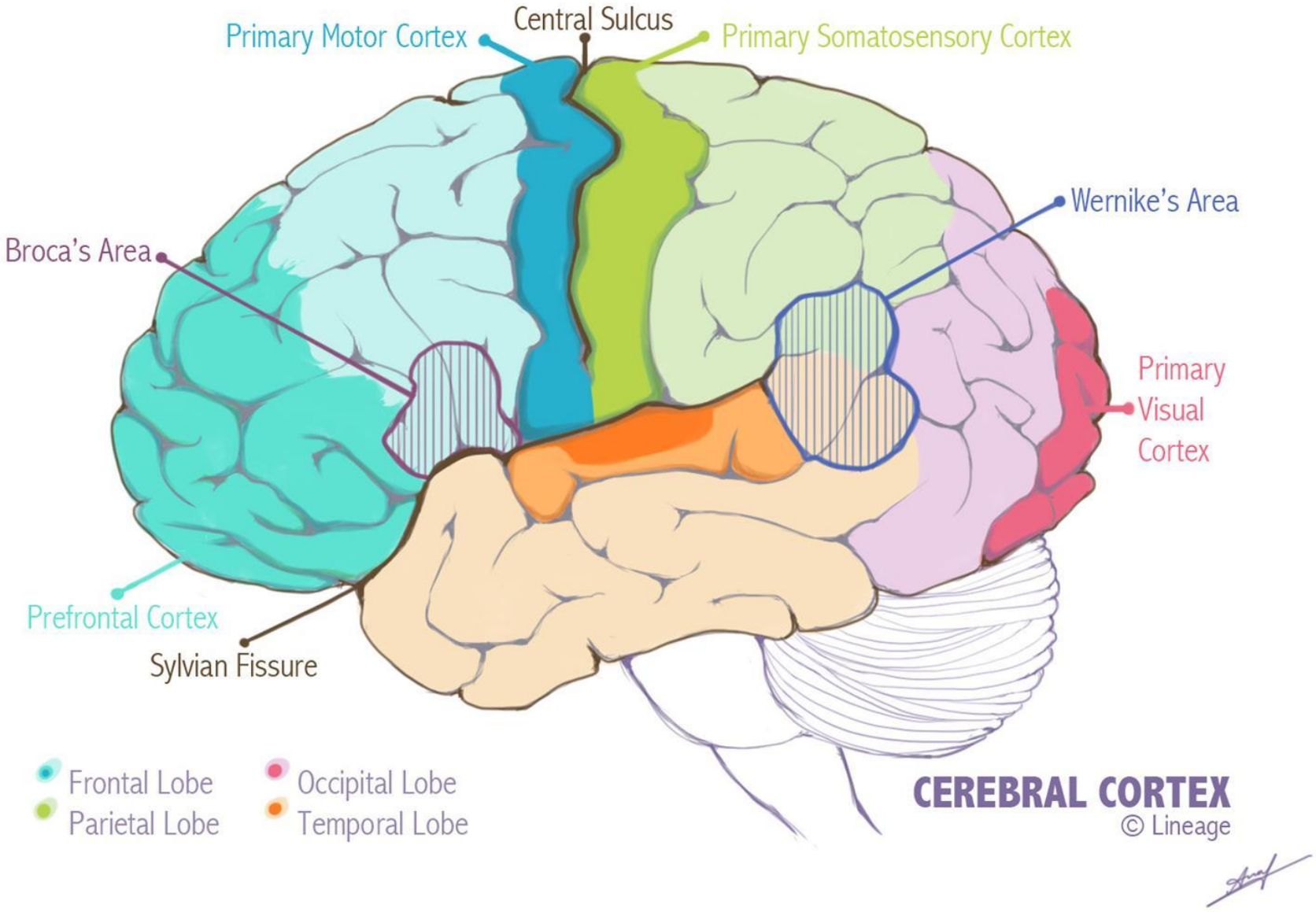
Cognitive processes seem deep

Humans organize their ideas and concepts hierarchically.

Humans first learn simpler concepts and then compose them to represent more abstract ones.

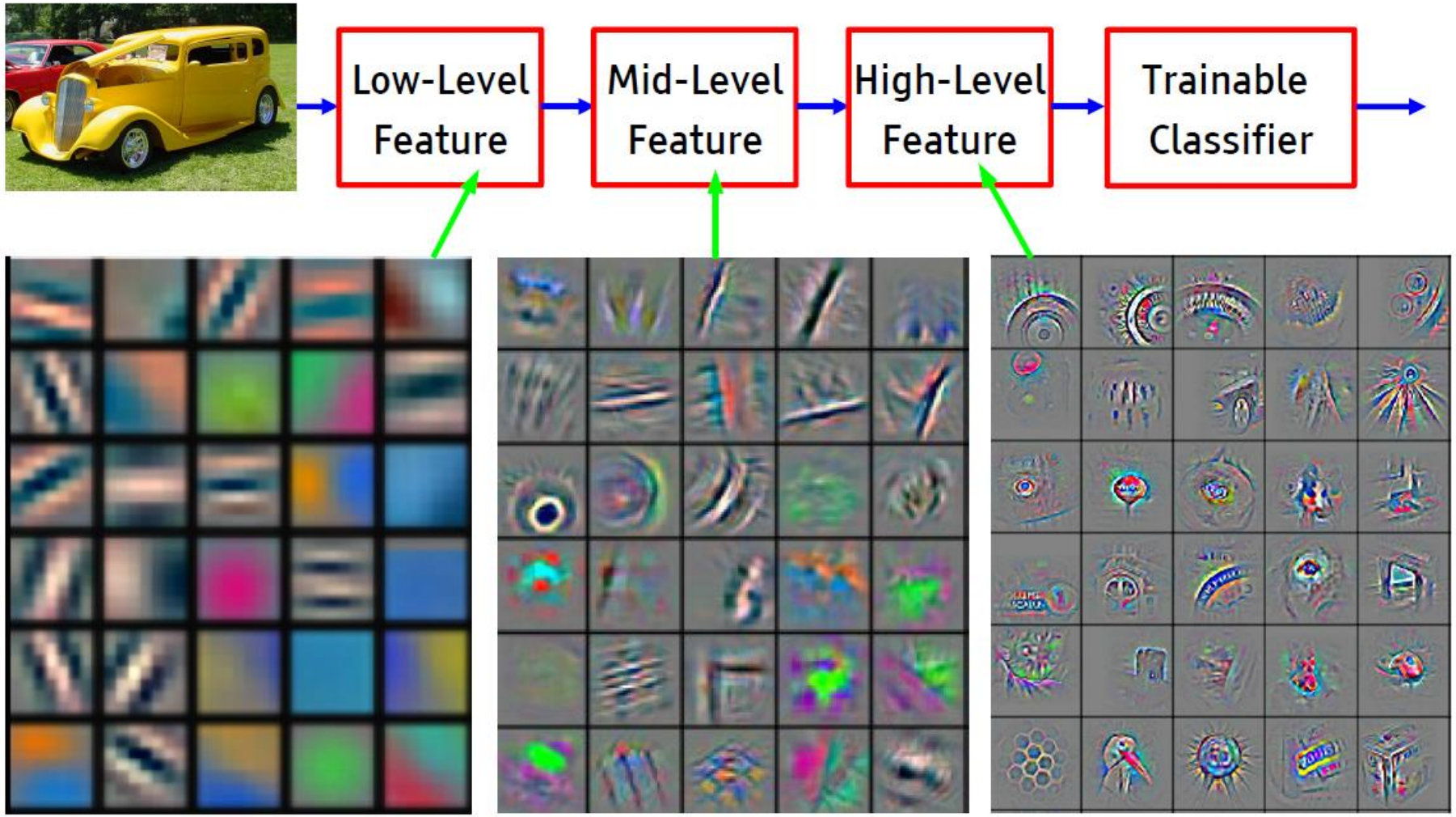
Engineers break-up solutions into multiple levels of abstraction and processing

The brain has a deep architecture



Deep Learning = Learning Hierarchical Representations

It's deep if it has more than one stage of non-linear feature transformation

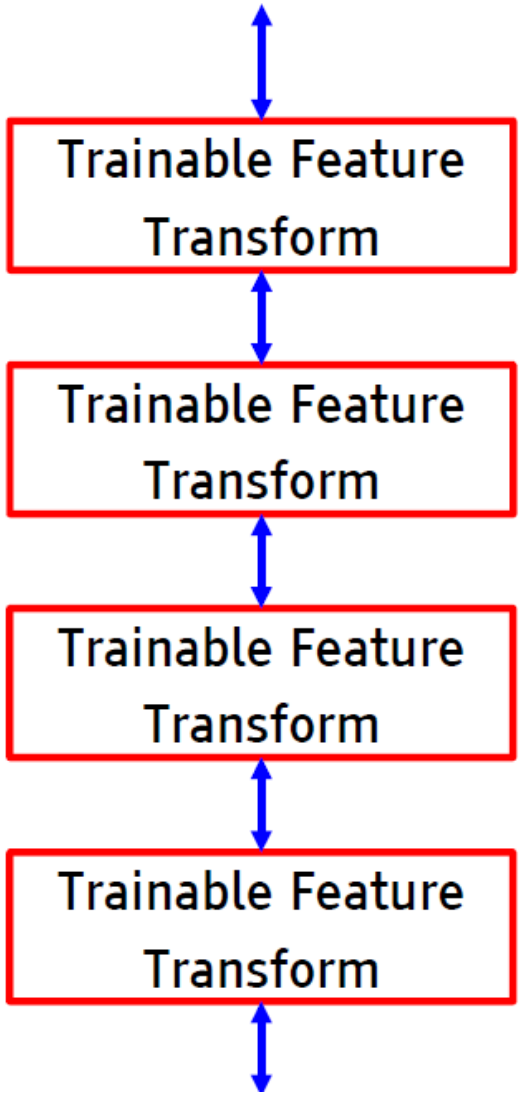


eature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Learning Representations: a challenge for ML, CV, AI, Neuroscience, Cognitive Science...

- How do we learn representations of the perceptual world?
  - How can a perceptual system build itself by looking at the world?
  - How much prior structure is necessary
- ML/AI: how do we learn features or feature hierarchies?
  - What is the fundamental principle? What is the learning algorithm? What is the architecture?
- Neuroscience: how does the cortex learn perception?
  - Does the cortex "run" a single, general learning algorithm? (or a small number of them)
- CogSci: how does the mind learn abstract concepts on top of less abstract ones?

- Deep Learning addresses the problem of learning hierarchical representations with a single algorithm
  - or perhaps with a few algorithms



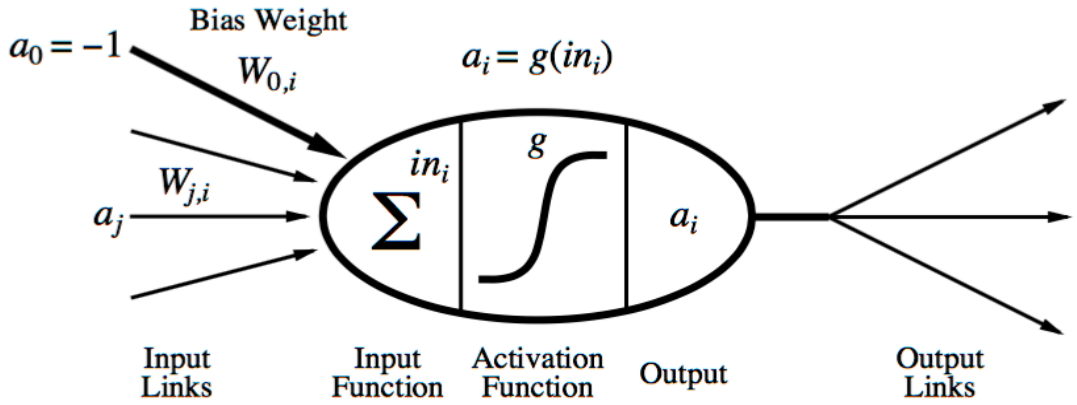
Outline

- Introduction
  - Motivation
  - Deep Learning Intuition
- Neural Network
  - Neuron, Feedforward algorithm, and Backpropagation algorithm
  - Limitations
- Deep Learning
  - Deep Belief Network: Autoencoder & Restricted Boltzman Machine
  - Convolutional Neural Network
- Deep Learning for Text Mining
  - Word Embedding
  - Recurrent Neural Network
  - Recursive Neural Network
- Conclusions

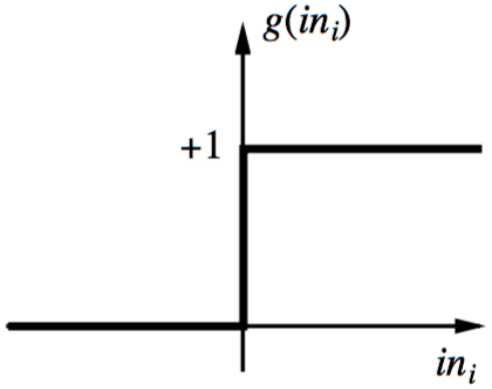
Neural Network: A Neuron

A neuron is a computational unit in the neural network that exchanges messages with each other.

$a_i \leftarrow g(in_i) = g\left(\sum_j W_{j,i} a_j\right)$

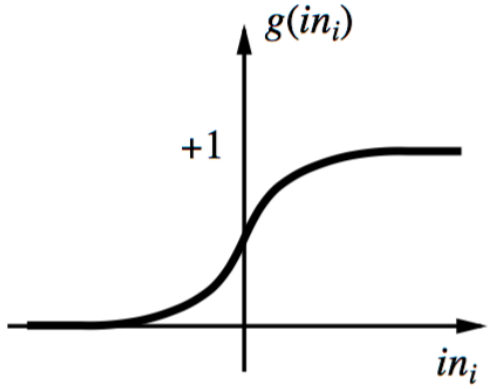


Possible activation functions:



(a)

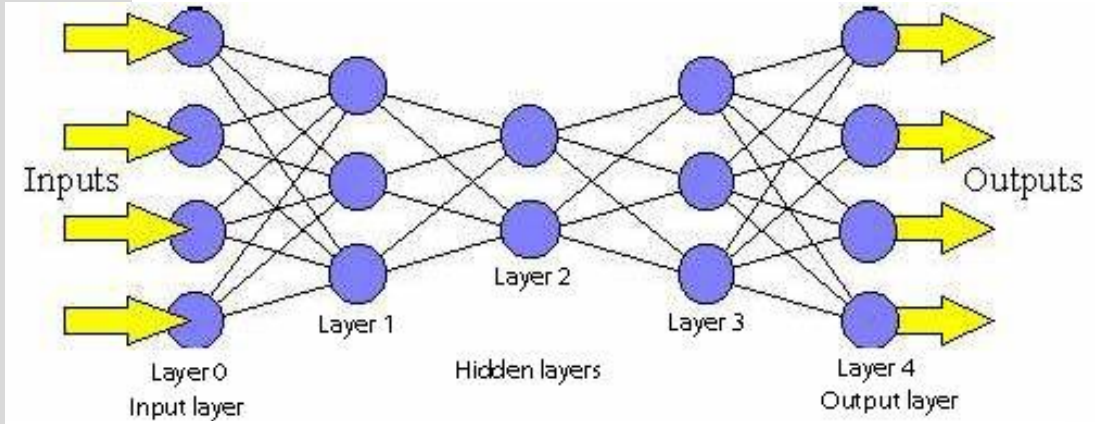
Step function/threshold function



(b)

Sigmoid function (a.k.a, logistic function)

Feed forward/Backpropagation Neural Network

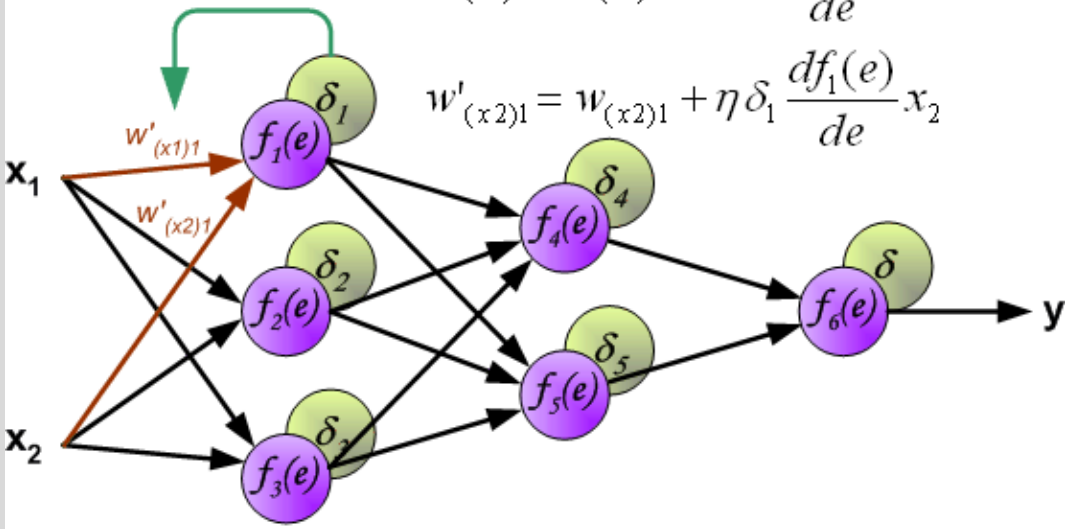


Feed forward algorithm:

Activate the neurons from the input to the output.

$$w'_{(x1)1} = w_{(x1)1} + \eta\delta_1 \frac{df_1(e)}{de} x_1$$

$$w'_{(x2)1} = w_{(x2)1} + \eta\delta_1 \frac{df_1(e)}{de} x_2$$



Backpropagation:

- Randomly initialize the parameters
- Calculate total error δ at the output, $f_6(e)$
- Then calculate contributions to error, δ_n, at each step going backwards.

Limitations of Neural Networks

Random initialization + **densely connected networks** lead to:

High cost

- Each neuron in the neural network can be considered as a logistic regression.
- Training the entire neural network is to train all the interconnected logistic regressions.

Difficult to train as the number of hidden layers increases

- Recall that logistic regression is trained by gradient descent.
- In backpropagation, gradient is progressively getting more dilute. That is, below top layers, the correction signal δ_n is minimal.

Stuck in local optima

- The objective function of the neural network is usually not convex.
- The random initialization does not guarantee starting from the proximity of global optima.

Solution:

- Deep Learning/Learning multiple levels of representation

Outline

- Introduction
  - Motivation
  - Deep Learning Intuition
- Neural Network
  - Neuron, Feedforward algorithm, and Backpropagation algorithm
  - Limitations
- Deep Learning
  - Deep Belief Network: Autoencoder & Restricted Boltzman Machine
  - Convolutional Neural Network
- Deep Learning for Text Mining
  - Word Embedding
  - Recurrent Neural Network
  - Recursive Neural Network
- Conclusions

Restricted Boltzmann machines

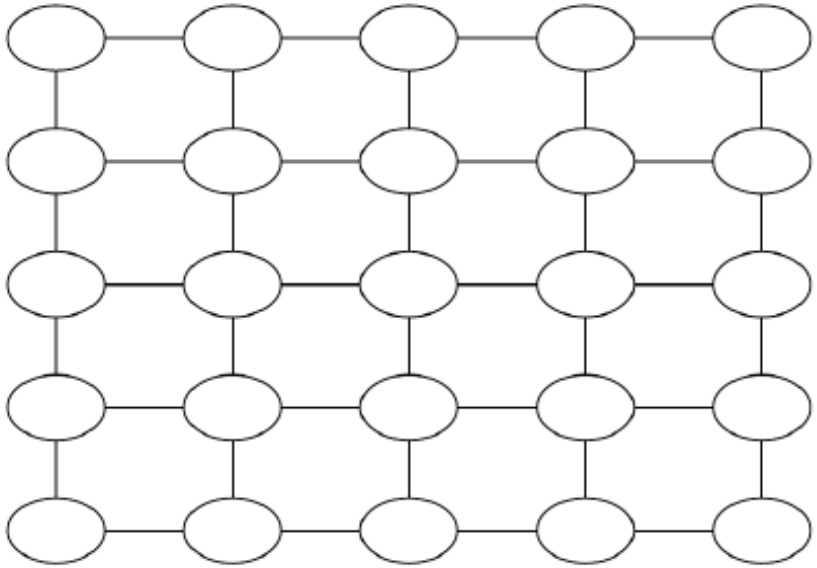
- Unsupervised learning
- Given i.i.d. samples $\{v^{(1)}, \ldots, v^{(n)}\}$, learn the joint distribution $\mu(v)$
- motivation for studying deep learning
  - concrete example of parameter leaning
  - applications in dimensionality reduction, classification, collaborative filtering, feature learning, topic modeling
  - successful in vision, language, speech
  - unsupervised learning: learn a generative model or a distribution
- running example: learn distribution over hand-written digits (cf. character recognition)
  - 32×32 binary images $v^{(\ell)} \in \{0,1\}^{32 \times 32}$

Graphical model

- grid

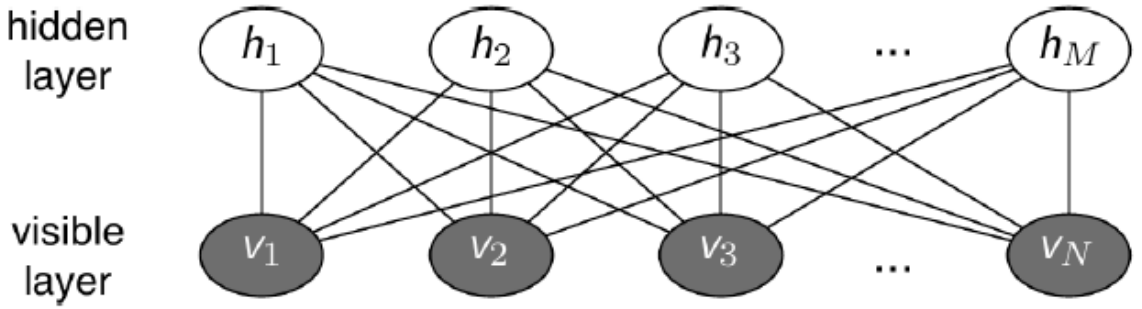
$$\mu(v) = \frac{1}{Z} \exp\left\{ \sum_i \theta_i v_i + \sum_{(i,j)\in E} \theta_{ij} v_i v_j \right\}$$

a sample $v^{(\ell)}$



Restricted Boltzmann machine [Smolensky 1986] "harmoniums"

- undirected graphical model
- two layers: visible layer $v \in \{0,1\}^N$ and hidden layer $h \in \{0,1\}^M$
- fully connected bipartite graph

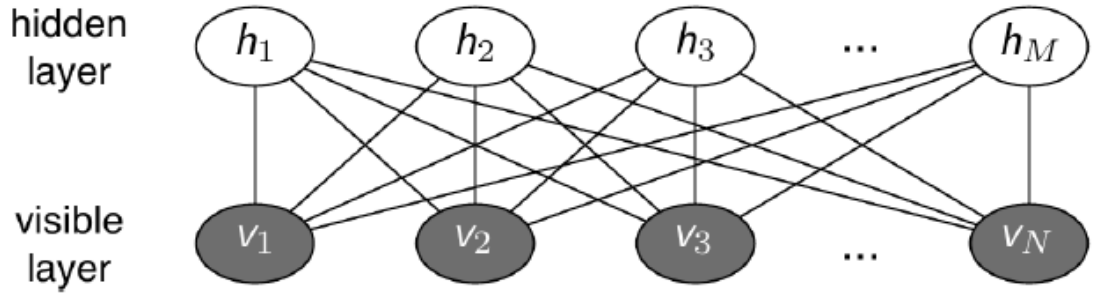


- RBMs represented by parameters $a \in \mathbb{R}^N$, $b \in \mathbb{R}^M$, and $W \in \mathbb{R}^{N \times M}$ such that

$$\mu(v,h) = \frac{1}{Z} \exp\left\{ a^T v + b^T h + v^T W h \right\}$$

- consider the marginal distribution $\mu(v)$ of the visible nodes, then any distribution on v can be modeled arbitrarily well by RBM with $k-1$ hidden nodes, where k is the cardinality of the support of the target distribution

Restricted Boltzmann machine

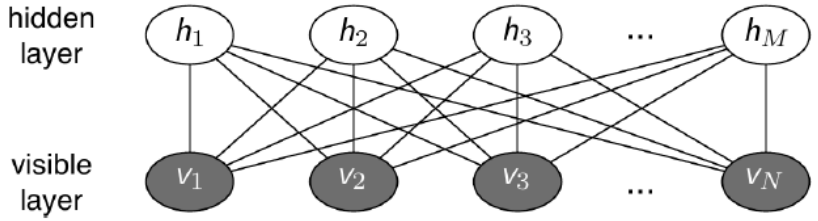


define energy

$$E(v,h) = -a^T v - b^T h - v^T W h$$

$$\begin{aligned} \mu(v,h) &= \frac{1}{Z}\exp\{-E(v,h)\} \\ &= \frac{1}{Z}\prod_{i\in[N]} e^{a_i v_i} \prod_{j\in[M]} e^{b_j h_j} \prod_{(i,j)\in E} e^{v_i W_{ij} h_j} \end{aligned}$$

note that RBM has no intra-layer edges (hence the name ***restricted***)
(cf. Boltzmann machine [Hinton,Sejnowski 1985])



it follows from the Markov property of MRF that the conditional distribution a product distribution

$$\mu(v|h) \propto \exp\left\{(a + Wh)^T v\right\} = \prod_i \exp\left\{(a_i + \langle W_{i\cdot}, h\rangle) v_i\right\}$$

$$= \prod_{i\in[N]} \mu(v_i|h)$$

$$\mu(h|v) = \prod_{j\in[M]} \mu(h_j|v)$$

hence, inference in the conditional distribution is efficient
e.g. compute a conditional marginal $\mu(h_i|v)$

since $\mu(h|v) = \prod \mu(h_j|v)$,

$$\mu(h_j = 0|v) = \frac{\mu(h_j = 0|v)}{\mu(h_j = 0|v) + \mu(h_j = 1|v)}$$

$$= \frac{1}{1 + e^{b_j + \langle W_{\cdot j}, v\rangle}}$$

$$\mu(h_j = 1|v) = \underbrace{\frac{1}{1 + e^{-(b_j + \langle W_{\cdot j}, v\rangle)}}}_{\triangleq \sigma(b_j + \langle W_{\cdot j}, v\rangle) \text{ is a sigmiod}}$$

similarly,

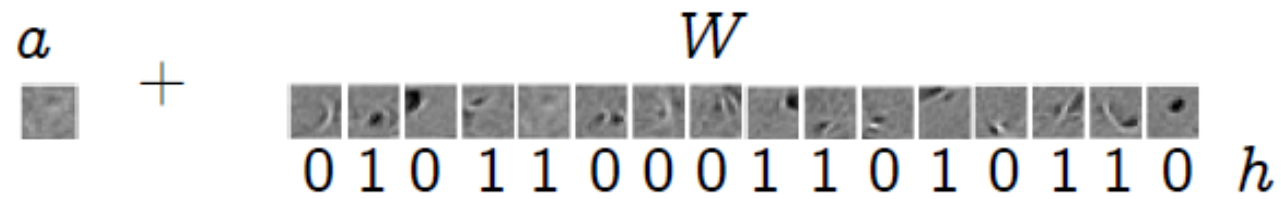
$$\mu(v_i = 1|h) = \sigma(a_i + \langle W_{i\cdot}, h\rangle)$$

where $W_{i\cdot}$ and $W_{\cdot j}$ are i-th row and j-th column of W resepectively, and $\langle \cdot, \cdot \rangle$ denotes the inner product

- one interpretation of RBM is to think of it as a stochastic version of a neural network, where nodes and edges correspond to neurons and synaptic connections, and a single node fires (i.e. $v_i = +1$) stochastically from a sigmoid activation function $\sigma(x) = 1/(1 + e^{-x})$,

$$\mu(v_i = +1|h) = \frac{e^{a_i + \langle W_{i\cdot}, h \rangle}}{1 + e^{a_i + \langle W_{i\cdot}, h \rangle}} = \sigma(a_i + \langle W_{i\cdot}, h \rangle)$$

- another interpretation is to think of the bias a and the components $W_{\cdot j}$ connected to a hidden node h_j encode higher level structure



- if the goal is to sample from this distribution (perhaps to generate images that resemble hand-written digits), the lack of inter-layer connection makes Gibbs sampling particularly easy, since one can apply block Gibbs sampling for each layer all together from $\mu(v|h)$ and $\mu(h|v)$ iteratively

- What about computing the marginal $\mu(v)$?

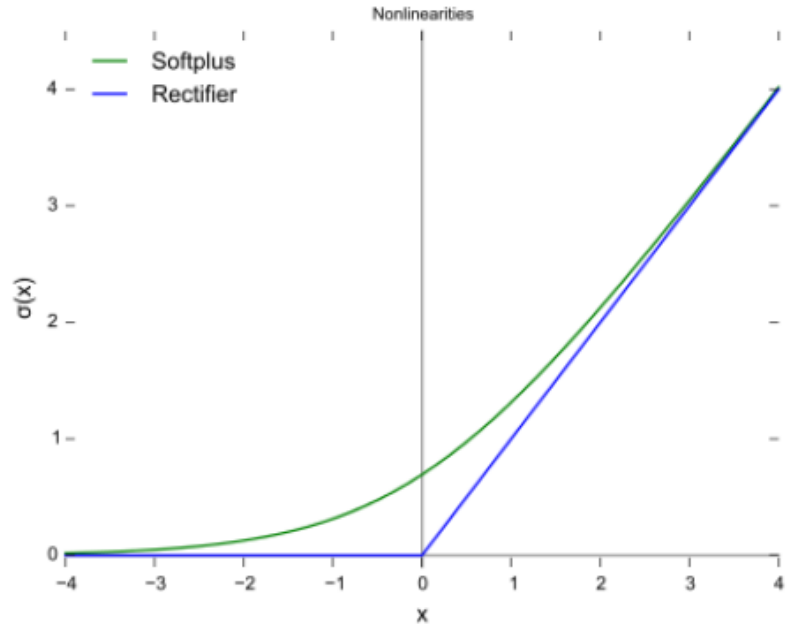
$$\begin{aligned}
\mu(v) &= \sum_{h \in \{0,1\}^M} \mu(v,h) \\
&= \frac{1}{Z} \sum_{h} \exp\{-E(v,h)\} \\
&= \frac{1}{Z} \exp\left\{ a^T v + \sum_{j=1}^{M} \log(1 + e^{b_j + \langle W_{\cdot j}, v \rangle}) \right\}
\end{aligned}$$

because

$$\begin{aligned}
\sum_{h \in \{0,1\}^M} e^{a^T v + b^T h + v^T W h} &= e^{a^T v} \sum_{h \in \{0,1\}^M} e^{b^T h + v^T W h} \\
&= e^{a^T v} \sum_{h \in \{0,1\}^M} \prod_{j=1}^{M} e^{b_j h_j + \langle v, W_{\cdot j} \rangle h_j} \\
&= e^{a^T v} \prod_{j=1}^{M} \sum_{h_j \in \{0,1\}} e^{b_j h_j + \langle v, W_{\cdot j} \rangle h_j}
\end{aligned}$$

$$
\begin{aligned}
\mu(v) &= \frac{1}{Z}\exp\left\{a^T v + \sum_{j=1}^{M}\log(1+e^{b_j+\langle W_{\cdot j}, v\rangle})\right\} \\
&= \frac{1}{Z}\exp\left\{a^T v + \sum_{j=1}^{M}\text{softmax}(b_j+\langle W_{\cdot j}, v\rangle)\right\}
\end{aligned}
$$

a_i is the bias in v_i, b_j is the bias in h_j, and $W_{\cdot j}$ is the output "image" v resulting from the hidden node h_j



- in the spirit of unsupervised learning, how do we extract features such that we achieve, dimensionality reduction?
  - suppose we have learned the RBM, and have a, b, W at our disposal
  - we want to compute features of a new sample $v^{(\ell)}$
  - we use the conditional distribution of the hidden layers as features

$$\mu(h|v^{(\ell)}) = [\mu(h_1|v^{(\ell)}), \cdots, \mu(h_M|v^{(\ell)})]$$

a sample $v^{(\ell)}$

[0,.2,0,.3,0, 0,0,0,0,.8,0,0,0,.4,1,0]

extracted features

Parameter learning

- given i.i.d. samples $v^{(1)}, \ldots, v^{(n)}$ from the marginal $\mu(v)$, we want to learn the RBM using maximum likelihood estimator

$$\mu(v) = \frac{1}{Z} \sum_{h \in \{0,1\}^M} \exp\left\{ a^T v + b^T h + v^T W h \right\}$$

$$\mathcal{L}(a, b, W) = \frac{1}{n} \sum_{\ell=1}^{n} \log \mu(v^{(\ell)})$$

- Gradient ascent (although not concave, and multi-modal)
  - ★ consider a single sample likelihood

$$\log \mu(v^{(\ell)}) = \log \sum_h \exp(a^T v^{(\ell)} + b^T h + (v^{(\ell)})^T W h) - \log Z$$

$$\frac{\partial \log \mu(v^{(\ell)})}{\partial b_i} = \frac{\sum_h h_i \, e^{a^T v^{(\ell)} + b^T h + (v^{(\ell)})^T W h}}{\sum_h e^{a^T v^{(\ell)} + b^T h + (v^{(\ell)})^T W h}} - \frac{\sum_{v,h} h_i \, e^{a^T v + b^T h + v^T W h}}{Z}$$

$$= \underbrace{\mathbb{E}_{\mu(h|v^{(\ell)})}[h_i]}_{\text{data-dependent expectation}} - \underbrace{\mathbb{E}_{\mu(v,h)}[h_i]}_{\text{model expectation}}$$

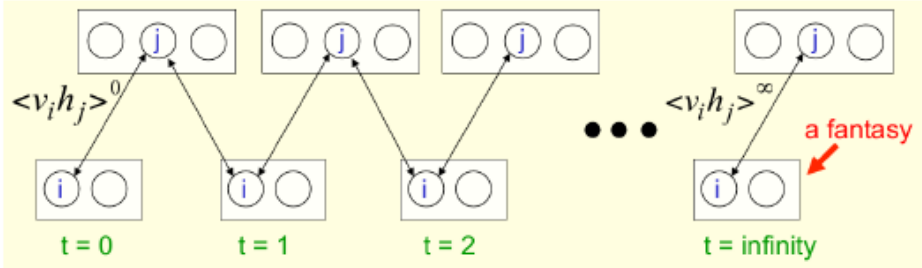
$$\frac{\partial \log \mu(v^{(\ell)})}{\partial a_i} = v_i^{(\ell)} - \mathbb{E}_{\mu(v,h)}[v_i]$$

$$\frac{\partial \log \mu(v^{(\ell)})}{\partial W_{ij}} = \mathbb{E}_{\mu(h|v^{(\ell)})}[v_i^{(\ell)} h_j] - \mathbb{E}_{\mu(v,h)}[v_i h_j]$$

- once we have the gradient update a, b, and W as

$$W_{ij}^{(t+1)} = W_{ij}^{(t)} + \alpha \underbrace{\left(\mathbb{E}_{\mu(h|v)q(v)}[v_i h_j] - \mathbb{E}_{\mu(v,h)}[v_i h_j]\right)}_{\frac{\partial \log \mu}{\partial W_{ij}}}$$

  $q(v)$ is the empirical distribution of the training samples $v^{(1)}, \ldots, v^{(n)}$
- to compete the gradient, we need to compute the second term $\mathbb{E}_{\mu(v,h)}[v_i h_j]$ requires inference
  - ★ one approach is to approximately compute the expectation using samples from MCMC, but in general can be quite slow for large network
  - ★ custom algorithms designed for RBMs: contrastive divergence, persistent contrastive divergence, parallel tempering
- MCMC (block Gibbs sampling) for computing $\mathbb{E}_{\mu(v,h)}[v_i h_j]$ where μ is defined by current parameters $a^{(t)}$, $b^{(t)}$, $W^{(t)}$



  - ★ start with samples $\{v_k^{(0)}\}_{k\in[K]}$ drawn from the data (i.i.d. uniformly at random with replacement) and repeat
  - ★ sample $\{h_k^{(t)}\}_{k\in[K]}$ with $\mu(h_k^{(t)}|v_k^{(t)})$, and
  - ★ sample $\{v_k^{(t+1)}\}_{k\in[K]}$ with $\mu(v_k^{(t+1)}|h_k^{(t)})$

- practical parameter learning algorithms for RBM use heuristics to approximate the log-likelihood gradient to speed up
- use persistent Markov chains to speed up the Markov chain
  - jointly update *fantasy particles* $\{(h, v)_k^{(t)}\}_{k\in[K]}$ and parameters $(a^{(t)}, b^{(t)}, W^{(t)})$ by repeating
    - ★ fix $(a^{(t)}, b^{(t)}, W^{(t)})$ and sample the next fantasy particles $\{(h, v)_k^{(t+1)}\}_{k\in[K]}$ according to a Markov chain
    - ★ update $(a^{(t)}, b^{(t)}, W^{(t)})$

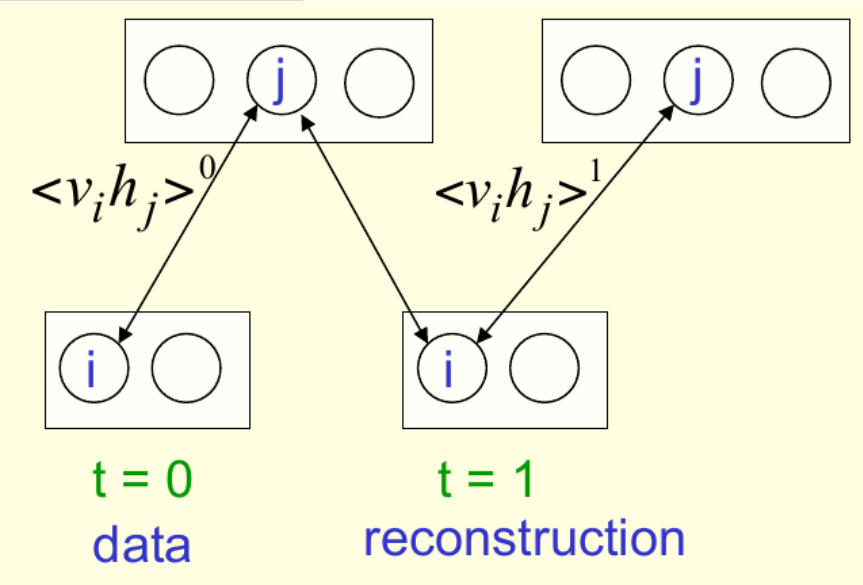
$$W_{ij}^{(t+1)} = W_{ij}^{(t)} + \alpha_t \left(\mathbb{E}_{\mu(h|v)q(v)}[v_i h_j] - \mathbb{E}_{\mu(v,h)}[v_i h_j]\right)$$

by computing the model expectation (the second term) via $\{(h, v)_k^{(t+1)}\}_{k\in[K]}$

Contrastive Divergence [Hinton 2002]

- a standard approach for learning RBMs
- k-step contrastive divergence
  - ★ **Input:** Graph G over v, h, training samples $S = \{v^{(1)}, \ldots, v^{(n)}\}$
  - ★ **Output:** gradient $\{\Delta w_{ij}\}_{i\in[N],j\in[M]}, \{\Delta a_i\}_{i\in[N]}, \{\Delta b_j\}_{j\in[M]}$

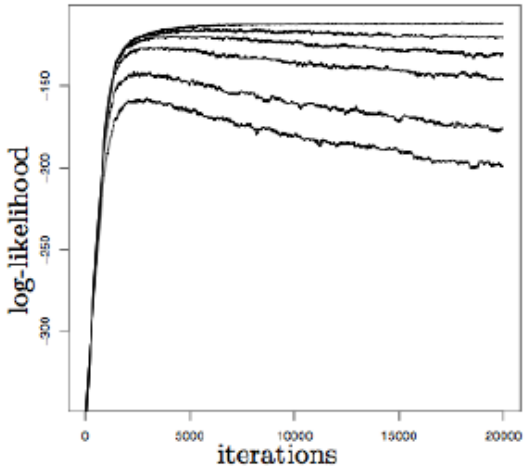
1. initialize $\Delta w_{ij}, \Delta a_i, \Delta b_j = 0$
2. **Repeat**
3. **for all** $v^{(\ell)} \in S$
4. $v(0) \leftarrow v^{(\ell)}$
5. **for** $t = 0, \ldots, k-1$ **do**
6. **for** $i = 1, \ldots, N$ **do** sample $h(t)_i \sim \mu(h_i|v(t))$
7. **for** $j = 1, \ldots, M$ **do** sample $v(t+1)_j \sim \mu(v_j|h(t))$
8. **for** $i = 1, \ldots, N, j = 1, \ldots, M$ **do**
9. $\Delta w_{ij} \leftarrow \Delta w_{ij} + \mathbb{E}_{\mu(h_i|v(0))}[h_i v(0)_j] - \mathbb{E}_{\mu(h_i|v(k))}[h_i v_j]$
10. $\Delta a_i \leftarrow \Delta a_i + v(0)_j - v(k)_j$
11. $\Delta b_j \leftarrow \Delta b_j + \mathbb{E}_{\mu(h_i|v(0))}[h_i] - \mathbb{E}_{\mu(h_i|v(k))}[h_i]$
12. **End for.**

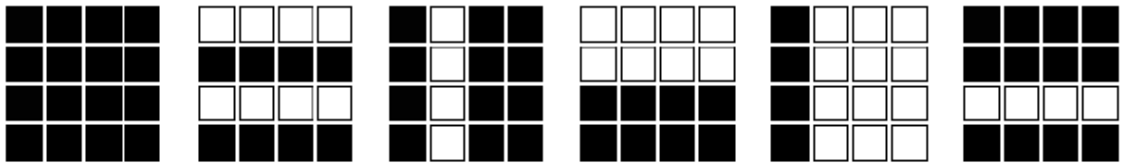


- since the Markov chain is run for finite k (often k=1 in practice), the resulting approximation of the gradient is biased (is dependent on the training samples $\{v^{(\ell)}\}$)
- **Theorem.**

$$\frac{\log \mu(v(0))}{\partial w_{ij}} = \Delta w_{ij} + \mathbb{E}_{\mu(h|v)P(v(k))}[v(k)_i h_j] - \mathbb{E}_{\mu(v,h)}[v_i h_j]$$

- the gap vanishes as $k \to \infty$ and $p(v(k)) \to \mu(v)$
- in general for finite k, the bias can lead to contrastive divergence converging to a solution that is not the maximum-likelihood

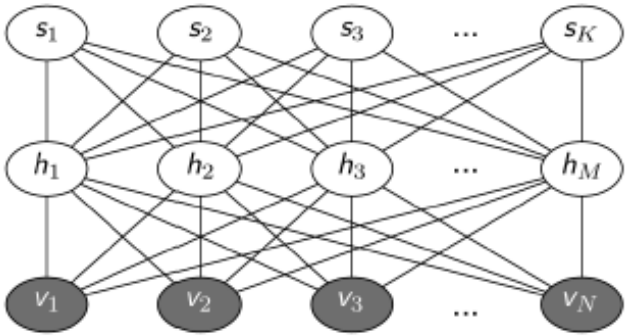




contrastive divergence with $k = 1, 2, 5, 10, 20, 100$ and 16 hidden nodes and training data from 4×4 bars-and-stripes example [2]

Deep Boltzmann machine

- deep Boltzmann machine

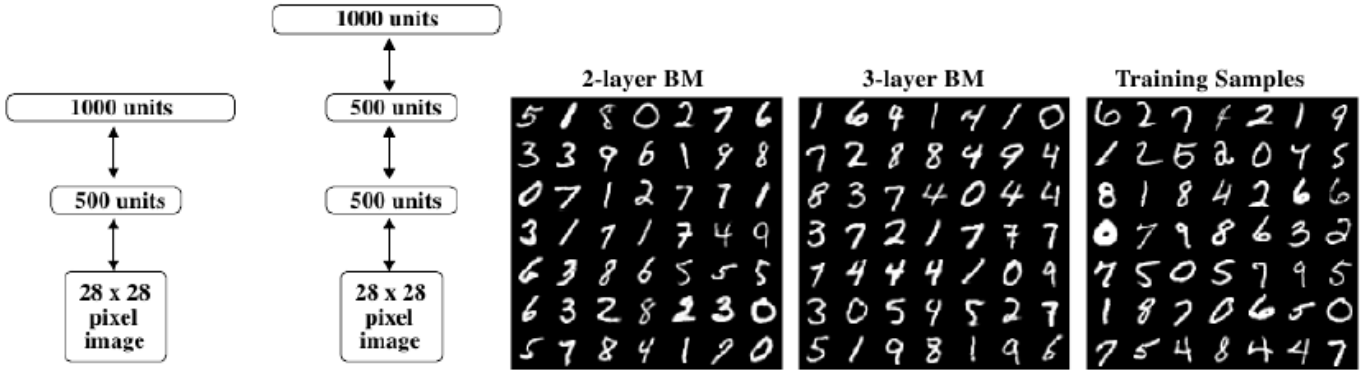


$$\mu(v, h, s) = \frac{1}{Z} \exp\left\{ a^T v + b^T h + c^T s + v^T W^1 h + h^T W^2 s \right\}$$

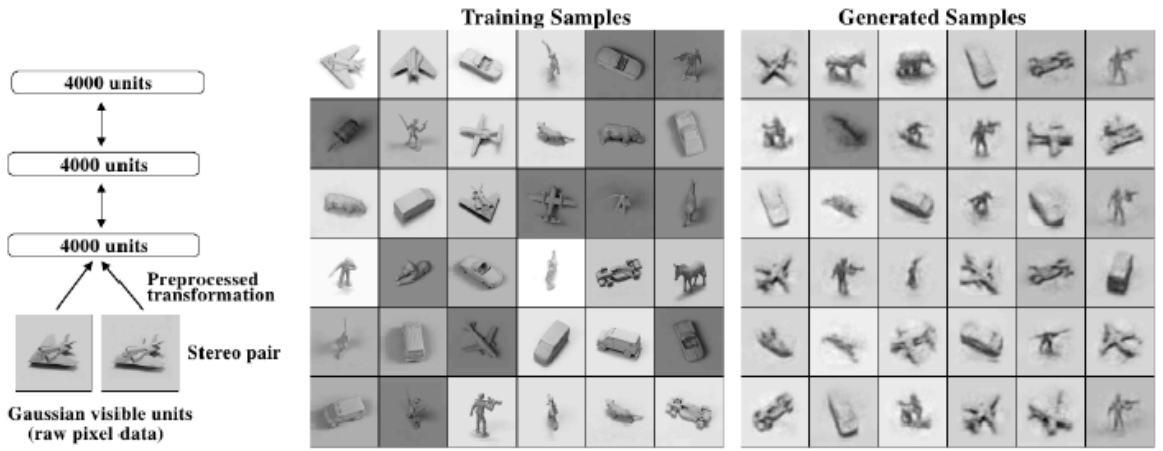
- capable of learning higher level and more complex representations

Deep Boltzmann machine [Salakhutdinov, Hinton '09]

- MNIST dataset with 60,000 training set
  - Samples generated by running Gibbs sampler for 100,000 steps



- NORB dataset with 24,300 stereo image pairs training set
  - 25 toy objects in 5 classes (car, truck, plane, animal, human)



- (stochastic) gradient ascent (drop a, b, c for simplicity)

$$\begin{aligned}\mathcal{L}(W^1, W^2) &= \log \mu(v^{(\ell)}) \\ &= \log \sum_{h,s} \exp\left\{(v^{(\ell)})^T W^1 h + h^T W^2 s\right\} - \log Z\end{aligned}$$

- gradient

$$\frac{\partial \log \mu(v^{(\ell)})}{\partial W^1_{ij}} =$$

$$\frac{\sum_{h,s}(v_i^{(\ell)} h_j)\, e^{(v^{(\ell)})^T W^1 h + h^T W^2 s}}{\sum_{h,s} e^{(v^{(\ell)})^T W^1 h + h^T W^2 s}} - \frac{\sum_{v,h,s}(v_i^{(\ell)} h_j)\, e^{(v^{(\ell)})^T W^1 h + h^T W^2 s}}{Z}$$

$$= \mathbb{E}_{\mu(h|v^{(\ell)})}[v_i^{(\ell)} h_j] - \mathbb{E}_{\mu(v,h)}[v_i h_j]$$

$$\frac{\partial \log \mu(v^{(\ell)})}{\partial W^2_{ij}} = \mathbb{E}_{\mu(h,s|v^{(\ell)})}[h_i s_j] - \mathbb{E}_{\mu(h,s)}[h_i s_j]$$

- problem: now even the data-dependent expectation is difficult to compute

Variational method

- use naive mean-field approximation to get a lower bound on the objective function

$$\begin{aligned}
\mathcal{L}(W^1, W^2) &= \log \mu(v^{(\ell)}) \\
&\geq \log \mu(v^{(\ell)}) - D_{\mathrm{KL}}\big(b(h,s) \,\big\|\, \mu(h,s|v^{(\ell)})\big) \\
&= \sum_{h,s} b(h,s) \log \mu(h,s,v^{(\ell)}) + H(b) \\
&\equiv \mathbb{G}(b, W^1, W^2, v^{(\ell)})
\end{aligned}$$

- instead of maximizing $\mathcal{L}(W^1, W^2)$, maximize $\mathbb{G}(b, W^1, W^2, v^{(\ell)})$ to get approximately optimal W^1 and W^2
- constrain $b(h,s) = \prod_i p_i(h_i) \prod_j q_j(s_j)$ for simplicity
- and let $p_i = p_i(h_i = 1)$ and $q_j = q_j(s_j = 1)$

$$\mathbb{G} = \sum_{k,i} W^1_{ki} v^{(\ell)}_k p_i + \sum_{i,j} W^2_{ij} p_i q_j - \log Z + \sum_i H(p_i) + \sum_j H(q_j)$$

  - fix (W^1, W^2) and find maximizers $\{p_i^*\}, \{q_j^*\}$
  - fix $\{p_i^*\}, \{q_i^*\}$ and update (W^1, W^2) according to gradient ascent

$$W^2_{ij}(t+1) = W^2_{ij}(t) + \alpha(p_i^* \, q_j^* - \mathbb{E}_{\mu(h,s)}[h_i s_j])$$

$$\mathbb{G} = \sum_{k,i} W^1_{ki} v^{(\ell)}_k p_i + \sum_{i,j} W^2_{ij} p_i q_j - \log Z + \sum_i H(p_i) + \sum_j H(q_j)$$

- how do we find $\{p^*_i\}$ and $\{q^*_j\}$?
- gradient

$$\begin{aligned}
\frac{\partial \mathbb{G}}{\partial p_i} &= \sum_k W^1_{ki} v^{(\ell)}_k + \sum_j W^2_{ij} q_j + \log p_i - \log(1 - p_i) = 0 \\
\frac{\partial \mathbb{G}}{\partial q_j} &= \sum_i W^2_{ij} p_i + \log q_j - \log(1 - q_j) = 0
\end{aligned}$$

$$\begin{aligned}
p^*_i &= \frac{1}{1 + e^{-\sum_k W^1_{ki} v^{(\ell)}_k - \sum_j W^2_{ij} q^*_j}} \\
q^*_j &= \frac{1}{1 + e^{-\sum_i W^2_{ij} p^*_i}}
\end{aligned}$$

Convolutional Neural Network (CNN)

Convolutional Neural Networks are inspired by mammalian visual cortex.

The visual cortex contains a complex arrangement of cells, which are sensitive to small sub-regions of the visual field, called a receptive field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.

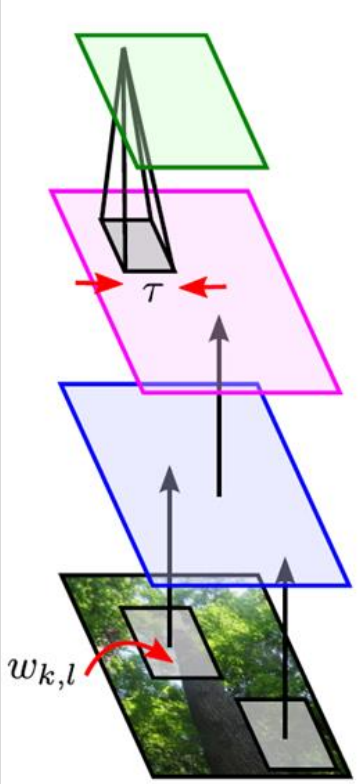
Two basic cell types:

Simple cells respond maximally to specific edge-like patterns within their receptive field.

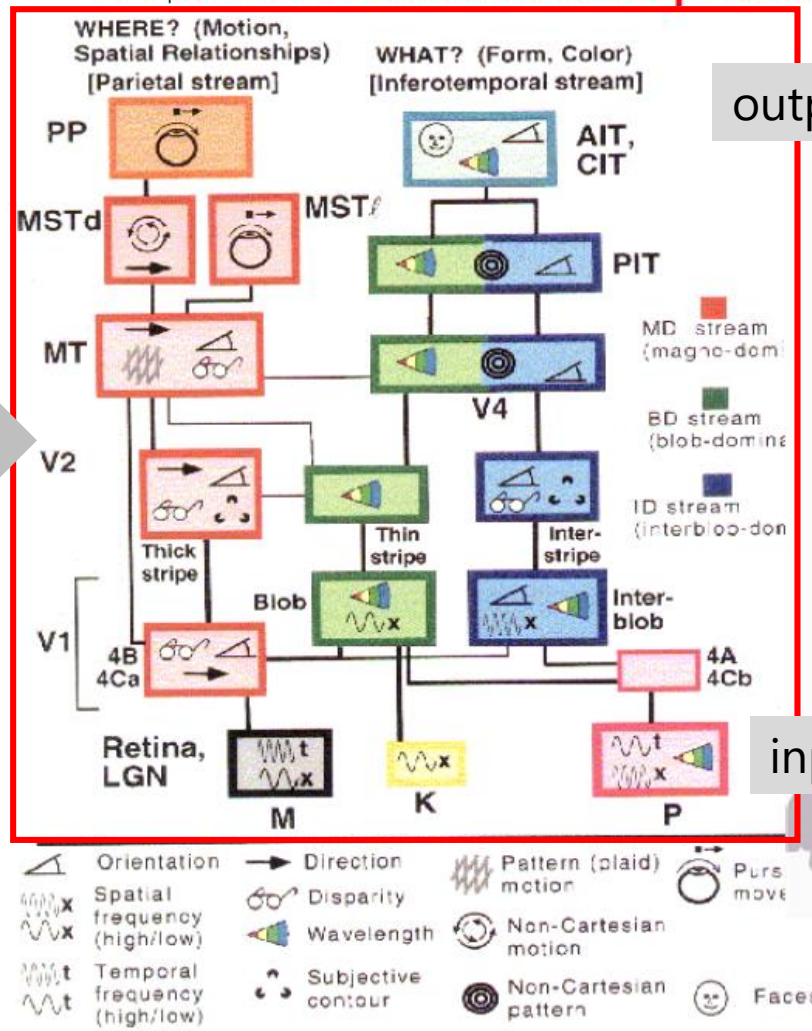
Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern.

The Mammalian Visual Cortex Inspires CNN

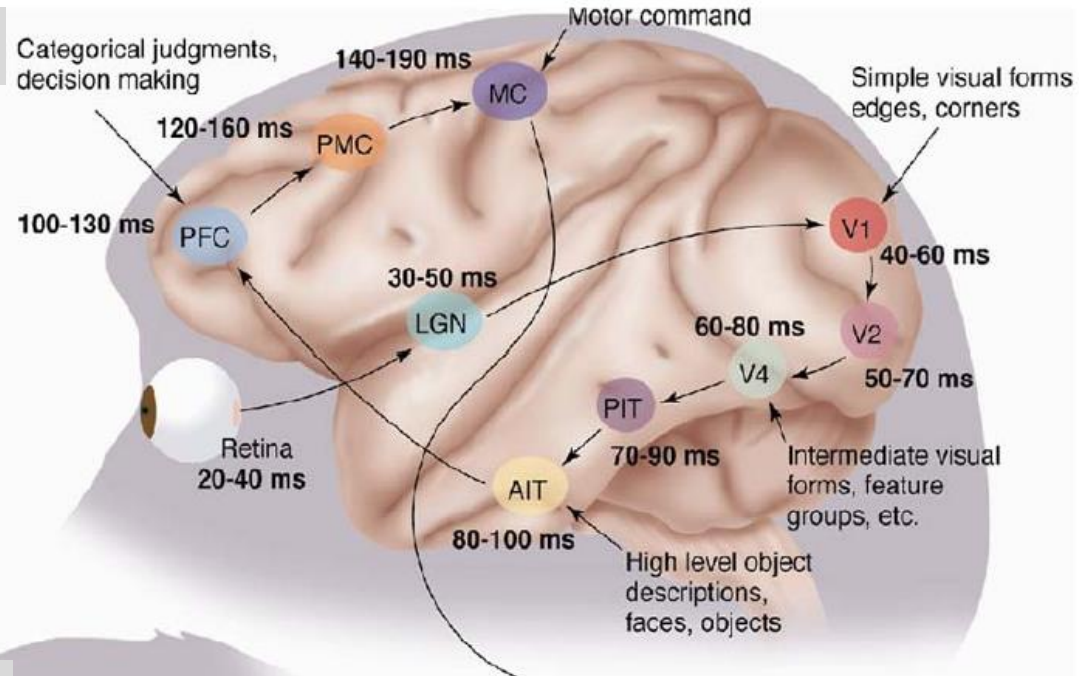
Convolutional Neural Net



- The ventral (recognition) pathway in the visual cortex has multiple stages
- Retina - LGN - V1 - V2 - V4 - PIT - AIT ....
- Lots of intermediate representations



output



input

[picture from Simon Thorpe]

[Gallant & Van Essen]

CNN Architecture

Intuition: Neural network with specialized connectivity structure,

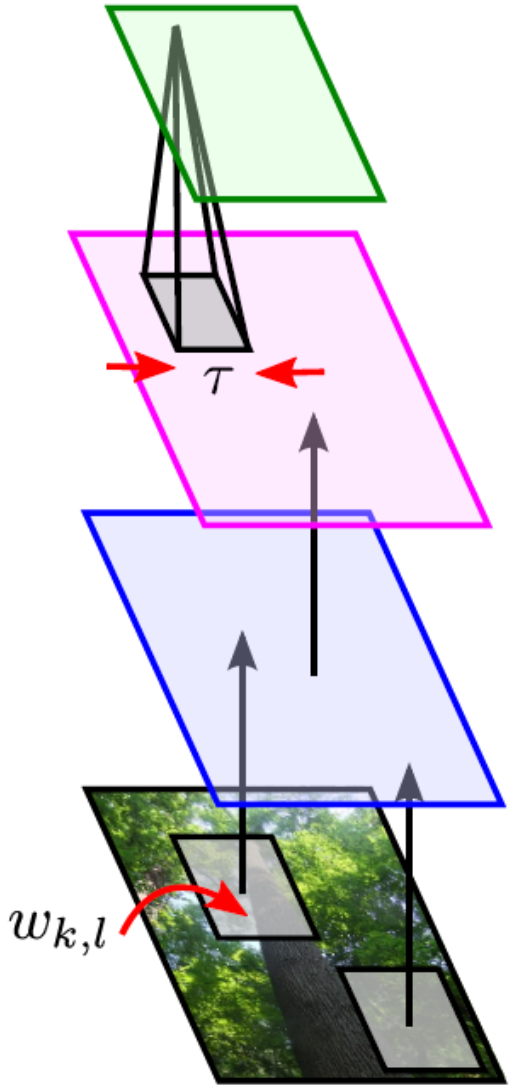
- Stacking multiple layers of feature extractors
- Low-level layers extract local features.
- High-level layers extract learn global patterns.

A CNN is a list of layers that transform the input data into an output class/prediction.

There are a few distinct types of layers:

- Convolutional layer
- Non-linear layer
- Pooling layer

Building-blocks for CNN's



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

Feature maps of a larger region are combined.

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

Feature maps are trained with neurons.

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

Shared weights

convolutional stage

Each sub-region yields a feature map, representing its feature.

$z_{i,j}$

Images are segmented into sub-regions.

input image

CNN Architecture: Convolutional Layer

The core layer of CNNs

The convolutional layer consists of a set of filters.

- Each filter covers a spatially small portion of the input data.

Each filter is convolved across the dimensions of the input data, producing a multidimensional feature map.

- As we convolve the filter, we are computing the dot product between the parameters of the filter and the input.

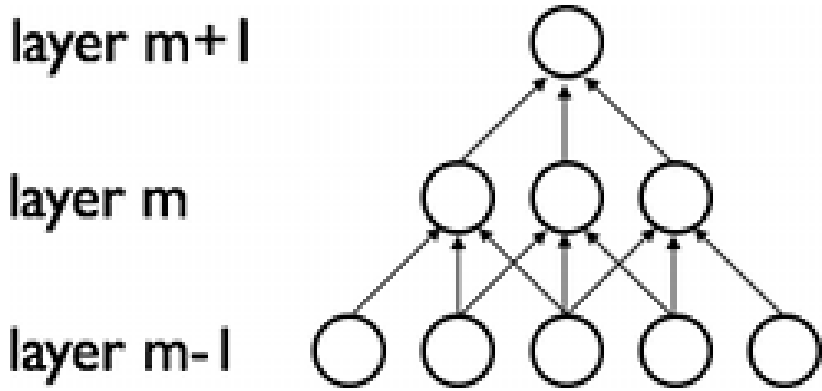
Intuition: the network will learn filters that activate when they see some specific type of feature at some spatial position in the input.

The key architectural characteristics of the convolutional layer is local connectivity and shared weights.

CNN Convolutional Layer: Local Connectivity

- Neurons in layer **m** are only connected to 3 adjacent neurons in the **m-1** layer.
- Neurons in layer **m+1** have a similar connectivity with the layer below.
- Each neuron is unresponsive to variations outside of its *receptive field* with respect to the input.

  Receptive field: small neuron collections which process portions of the input data
- The architecture thus ensures that the learnt feature extractors produce the strongest response to a spatially local input pattern.



CNN Convolutional Layer: Shared Weights

We show 3 hidden neurons belonging to the same feature map (the layer right above the input layer).

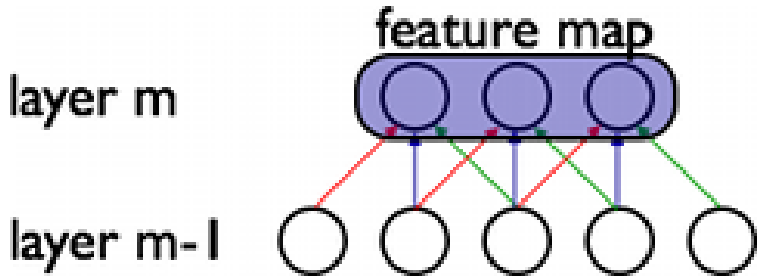
Weights of the same color are shared—constrained to be identical.

Gradient descent can still be used to learn such shared parameters, with only a small change to the original algorithm.

The gradient of a shared weight is simply the sum of the gradients of the parameters being shared.

Replicating neurons in this way allows for features to be detected regardless of their position in the input.

Additionally, weight sharing increases learning efficiency by greatly reducing the number of free parameters being learnt.



CNN Architecture: Non-linear Layer

Intuition: Increase the nonlinearity of the entire architecture without affecting the receptive fields of the convolution layer

A layer of neurons that applies the non-linear activation function, such as,

$$f(x) = \max(0, x)$$

$$f(x) = \tanh x$$

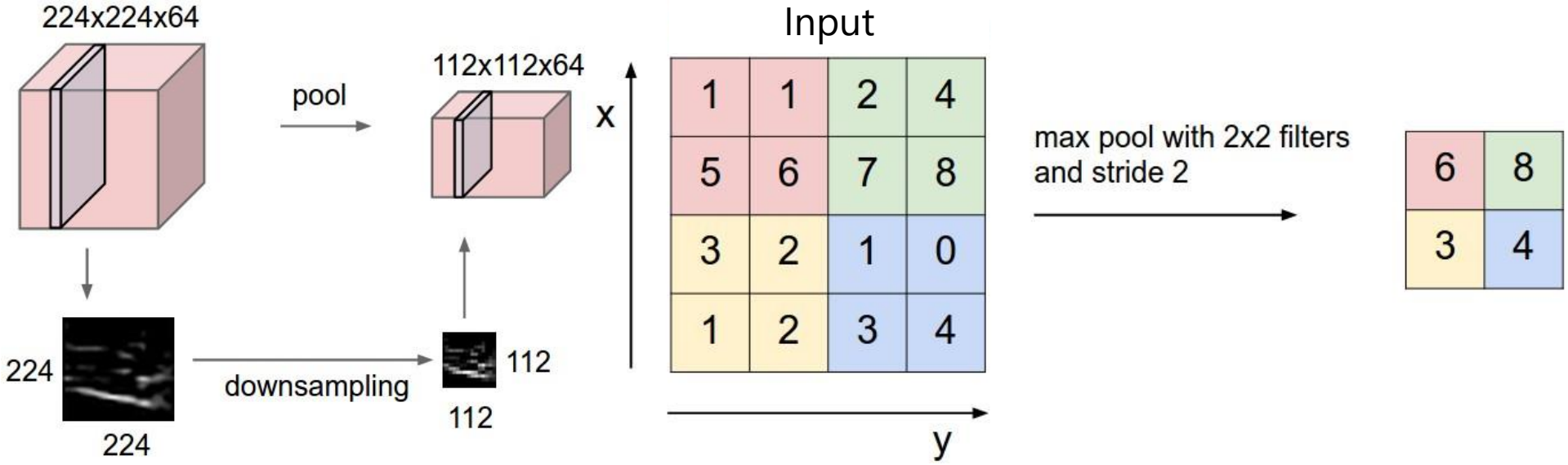
$$f(x) = |\tanh x|$$

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

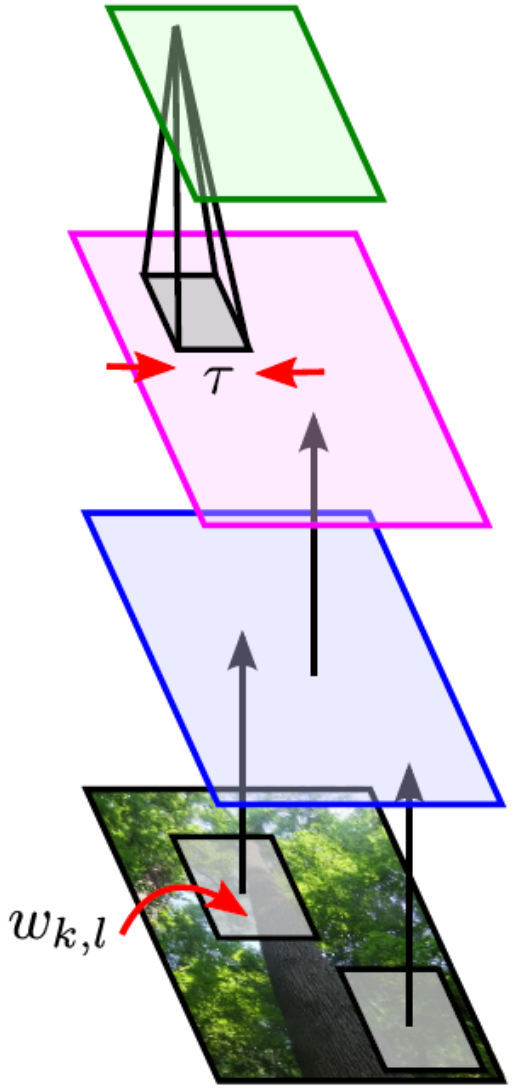
CNN Architecture: Pooling Layer

Intuition: to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting

Pooling partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum value of the features in that region.



Building-blocks for CNN's



$$x_{i,j} = \max_{|k|<\tau,|l|<\tau} y_{i-k,j-l}$$

mean or subsample also used

pooling stage

Feature maps of a larger region are combined.

$$y_{i,j} = f(a_{i,j})$$

e.g. $f(a) = [a]_+$

$f(a) = \text{sigmoid}(a)$

non-linear stage

Feature maps are trained with neurons.

$$a_{i,j} = \sum_{k,l} w_{k,l} z_{i-k,j-l}$$

Shared weights

convolutional stage

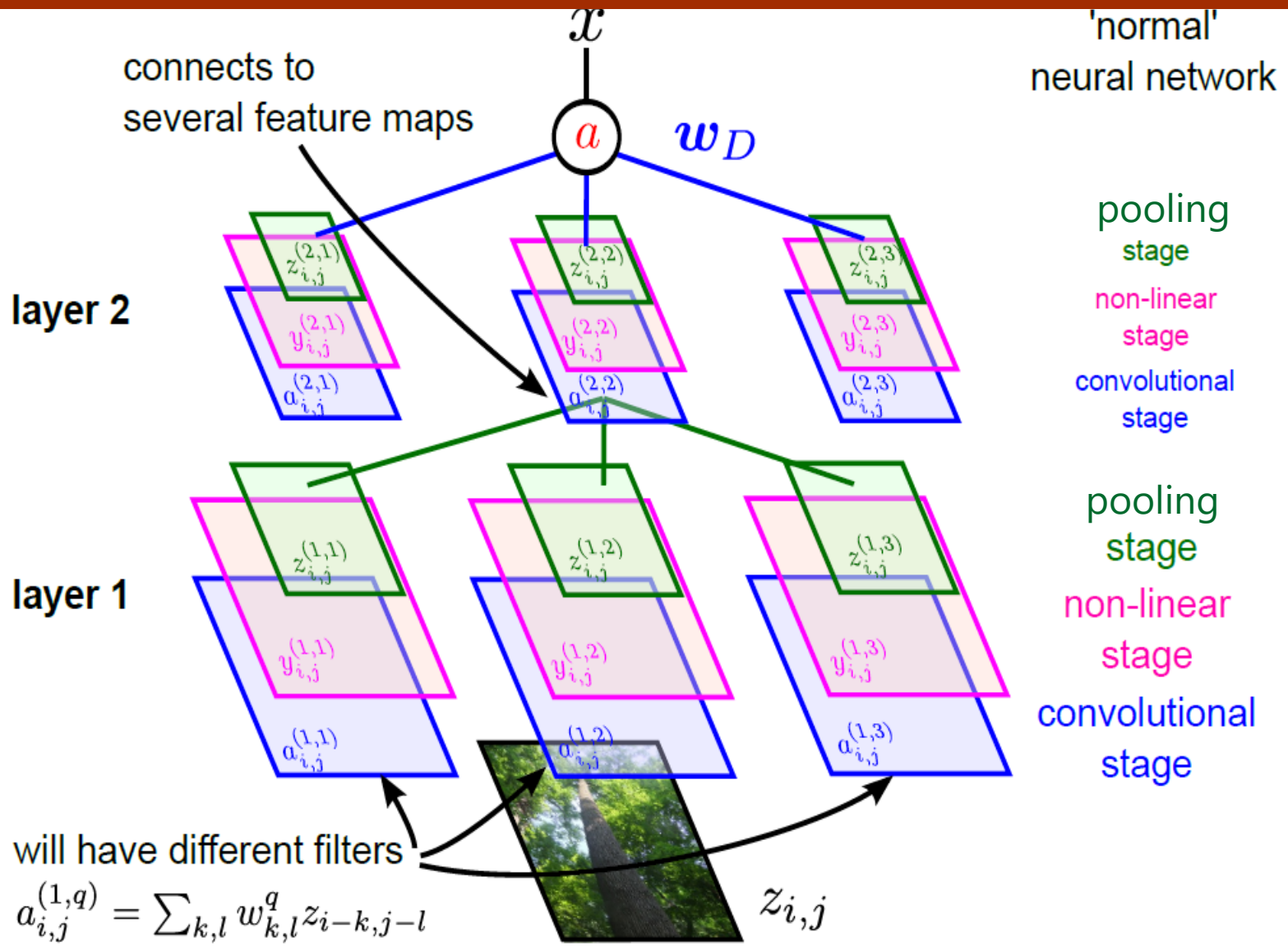
Each sub-region yields a feature map, representing its feature.

$z_{i,j}$

Images are segmented into sub-regions.

input image

Full CNN


x
'normal'
neural network
connects to
several feature maps
a
w_D
pooling
stage
non-linear
stage
convolutional
stage
layer 2
layer 1
pooling
stage
non-linear
stage
convolutional
stage
will have different filters
$a_{i,j}^{(1,q)} = \sum_{k,l} w_{k,l}^{q} z_{i-k,j-l}$
$z_{i,j}$

Outline

- Introduction
  - Motivation
  - Deep Learning Intuition
- Neural Network
  - Neuron, Feedforward algorithm, and Backpropagation algorithm
  - Limitations
- Deep Learning
  - Deep Belief Network: Autoencoder & Restricted Boltzman Machine
  - Convolutional Neural Network
- Deep Learning for Text Mining
  - Recurrent Neural Network
  - Recursive Neural Network
  - Word Embedding
- Conclusions

Deep Learning for Text Mining

Building deep learning models on textual data requires:

Representation of the basic text unit, word.

Neural network structure that can hierarchically capture the sequential nature of text.

Deep learning models for text mining use:

Vector representation of words (i.e., word embeddings)

Neural network structures

- *Recurrent* Neural Network
- *Recursive* Neural Network

Recurrent Neural Networks

The applications of standard Neural Networks (and also Convolutional Networks) are limited due to:

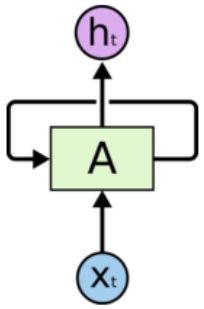
- They only accepted a fixed-size vector as input (e.g., an image) and produce a fixed-size vector as output (e.g., probabilities of different classes).
- These models use a fixed amount of computational steps (e.g. the number of layers in the model).

Recurrent Neural Networks are unique as they allow us to operate over sequences of vectors.

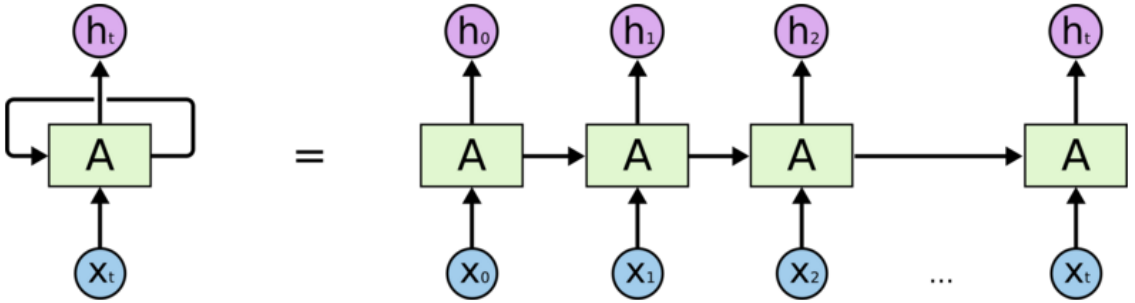
- Sequences in the input, the output, or in the most general case both

Recurrent Neural Networks

Recurrent Neural Networks are networks with loops in them, allowing information to persist.



Recurrent Neural Networks have loops.



An unrolled recurrent neural network.

In the above diagram, a chunk of neural network, ***A***, looks at some input x_t and outputs a value h_t.
A loop allows information to be passed from one step of the network to the next.

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor.
The diagram above shows what happens if we unroll the loop.

Recurrent Neural Networks

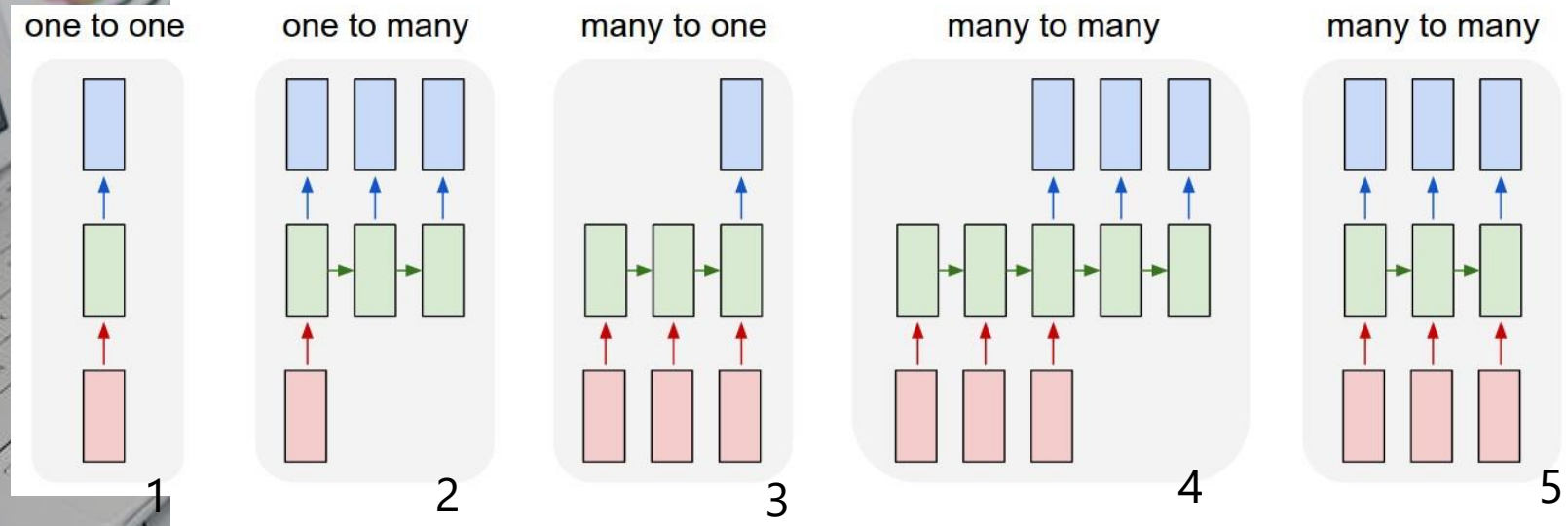
Intuition of Recurrent Neural Networks

- Human thoughts have persistence; humans don't start their thinking from scratch every second.
  - As you read this sentence, you understand each word based on your understanding of previous words.

- One of the appeals of RNNs is the idea that they are able to connect previous information to the present task
  - E.g., using previous video frames to inform the understanding of the present frame.
  - E.g., a language model tries to predict the next word based on the previous ones.

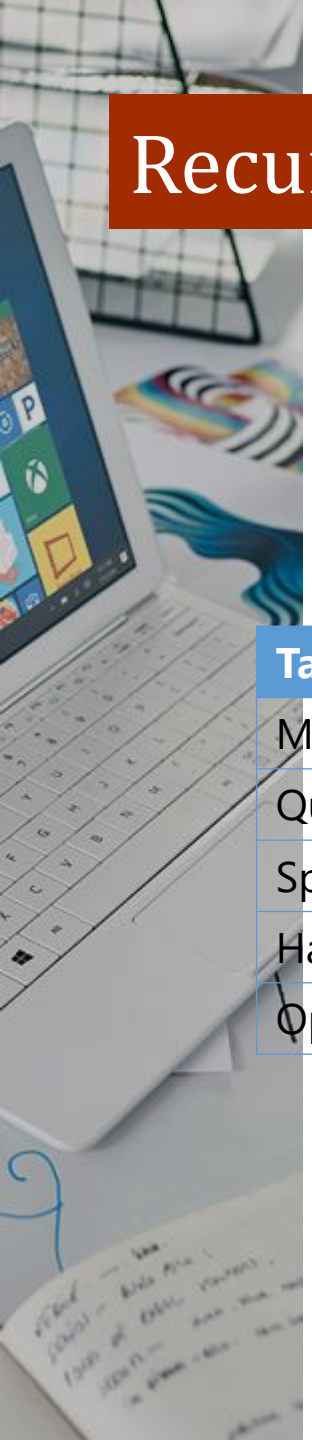
Recurrent Neural Networks

Examples of Recurrent Neural Networks



- Each rectangle is a vector and arrows represent functions (e.g. matrix multiply).
- Input vectors are in red, output vectors are in blue and green vectors hold the RNN's state

(1) Standard mode of processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification).
(2) Sequence output (e.g. image captioning takes an image and outputs a sentence of words).
(3) Sequence input (e.g. sentiment analysis where a given sentence is classified as expressing positive or negative sentiment).
(4) Sequence input and sequence output (e.g. Machine Translation: an RNN reads a sentence in English and then outputs a sentence in French).
(5) Synced sequence input and output (e.g. video classification where we wish to label each frame of the video).



Recurrent Neural Networks

Incredible success applying RNNs to language modeling and sequence learning problems

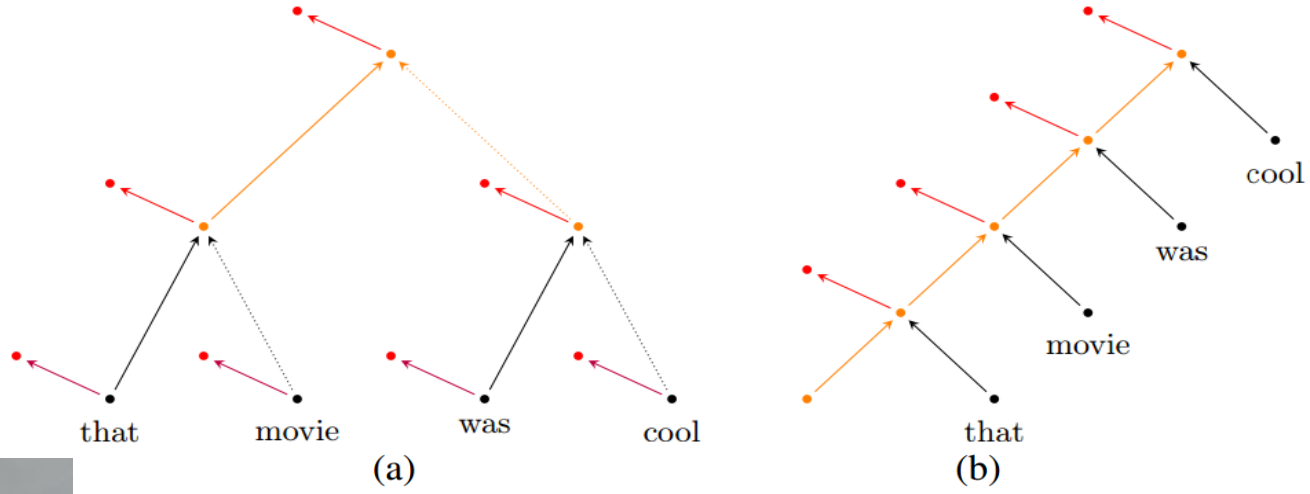
Task	Input Sequence	Output Sequence
Machine translation (Sutskever et al. 2014)	English	French
Question answering (Bordes et al. 2014)	Question	Answer
Speech recognition (Graves et al. 2013)	Voice	Text
Handwriting prediction (Graves 2013)	Handwriting	Text
Opinion mining (Irsoy et al. 2014)	Text	Opinion expression

Recursive Neural Networks

A recursive NN can be seen as a generalization of the recurrent NN

A recurrent neural network is in fact a recursive neural network with the structure of a linear chain

Recursive NNs operate on hierarchical structure, Recurrent NN operate on progression of time



- Operation of a recursive net (a), and a recurrent net (b) on an example sentence. Note the linear chain in (b)
- Black, orange and red dots represent input, hidden and output layers, respectively.
- Directed edges having the same color-style combination denote shared connections.

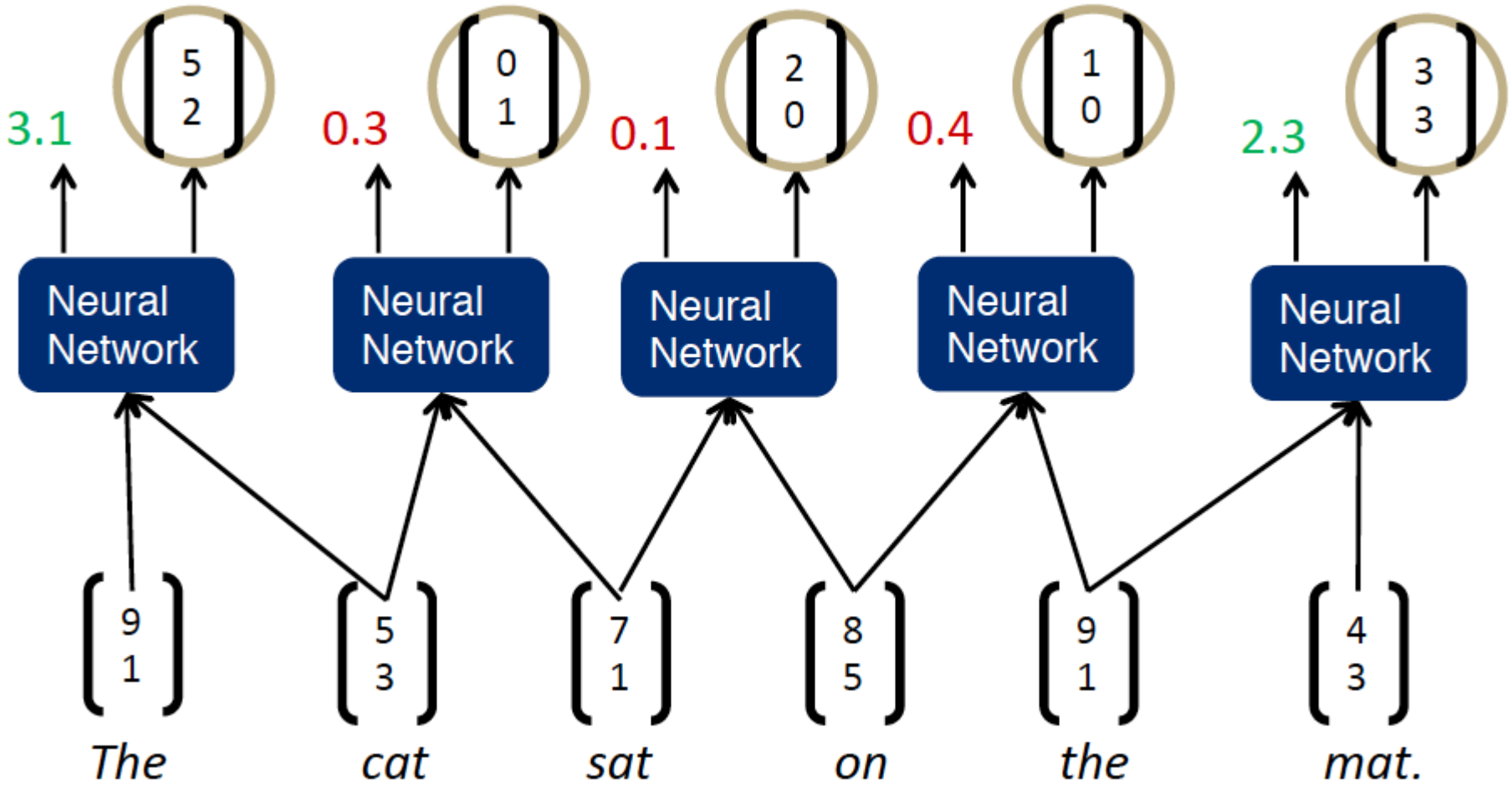
Applied to parsing, sentence-level sentiment analysis, and paraphrase detection.

Recursive Neural Networks

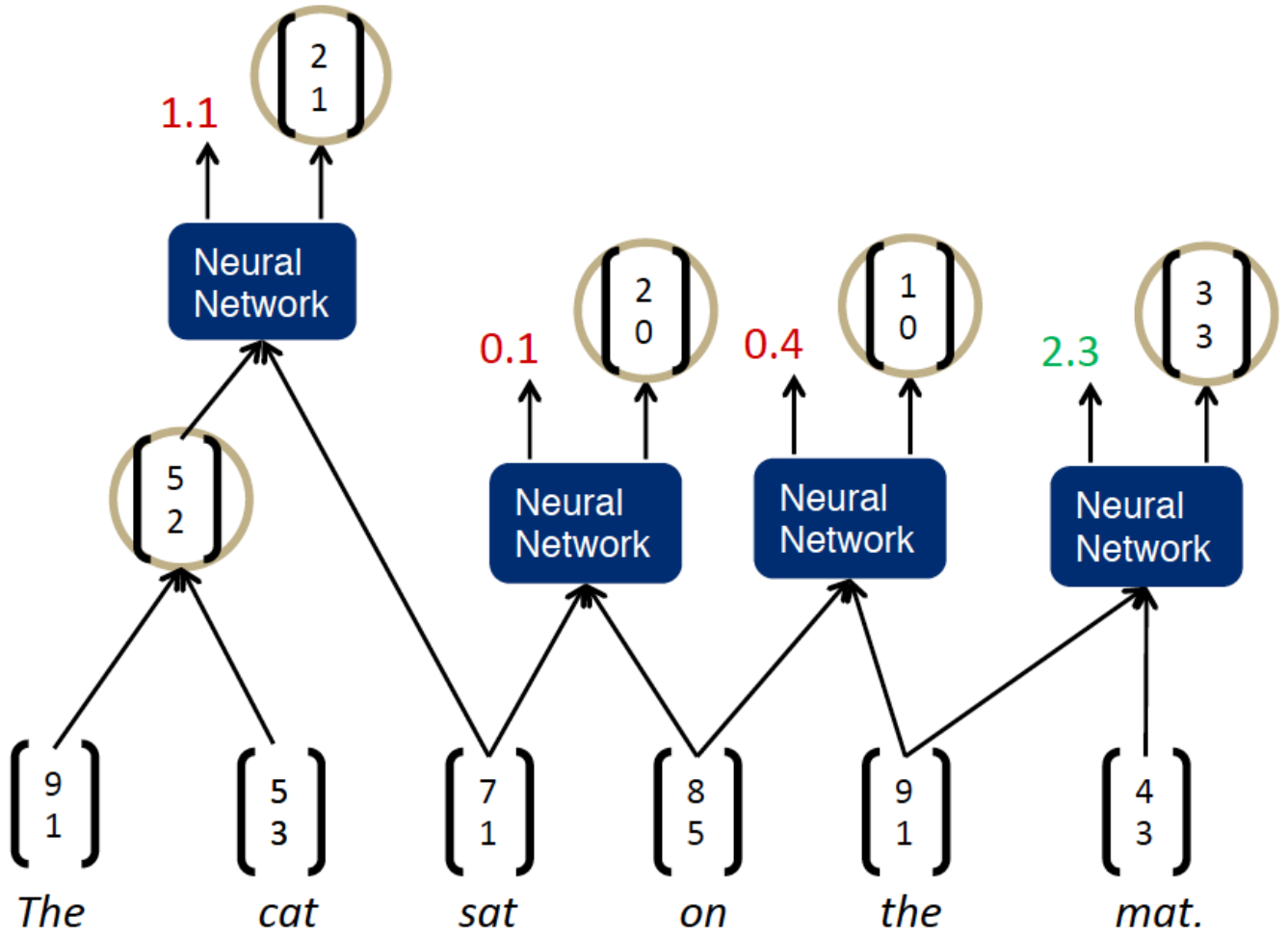
Given the structural representation of a sentence, e.g. a parse tree:

- Recursive Neural Networks recursively generate parent representations in a bottom-up fashion,
  - By combining tokens to produce representations for phrases, eventually producing the whole sentence.
- The sentence-level representation (or, alternatively, its phrases) can then be used to make a final classification for a given input sentence — e.g. whether it conveys a positive or a negative sentiment.

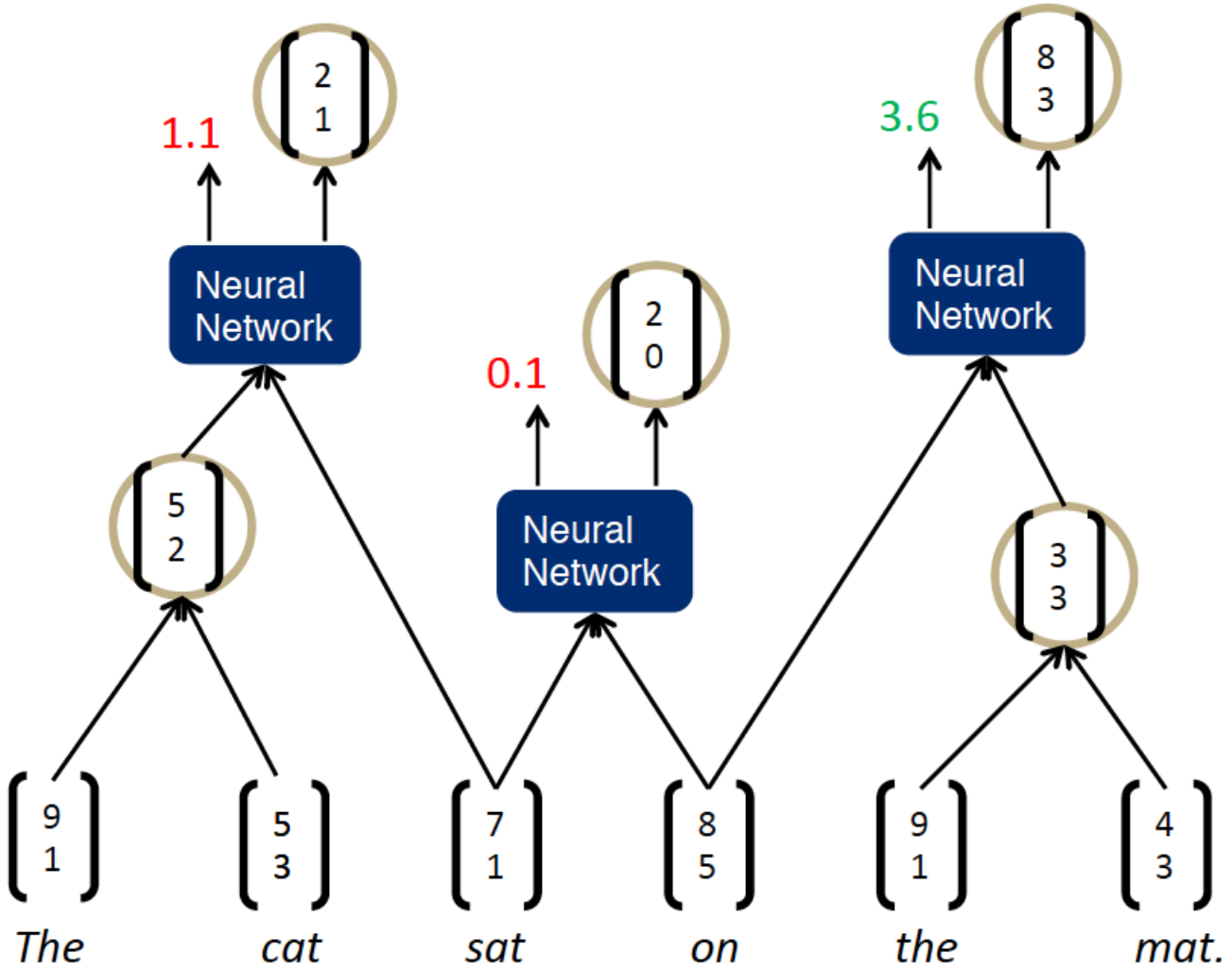
Recursive Neural Network Illustration (Animation)



Recursive Neural Network Illustration (Animation)



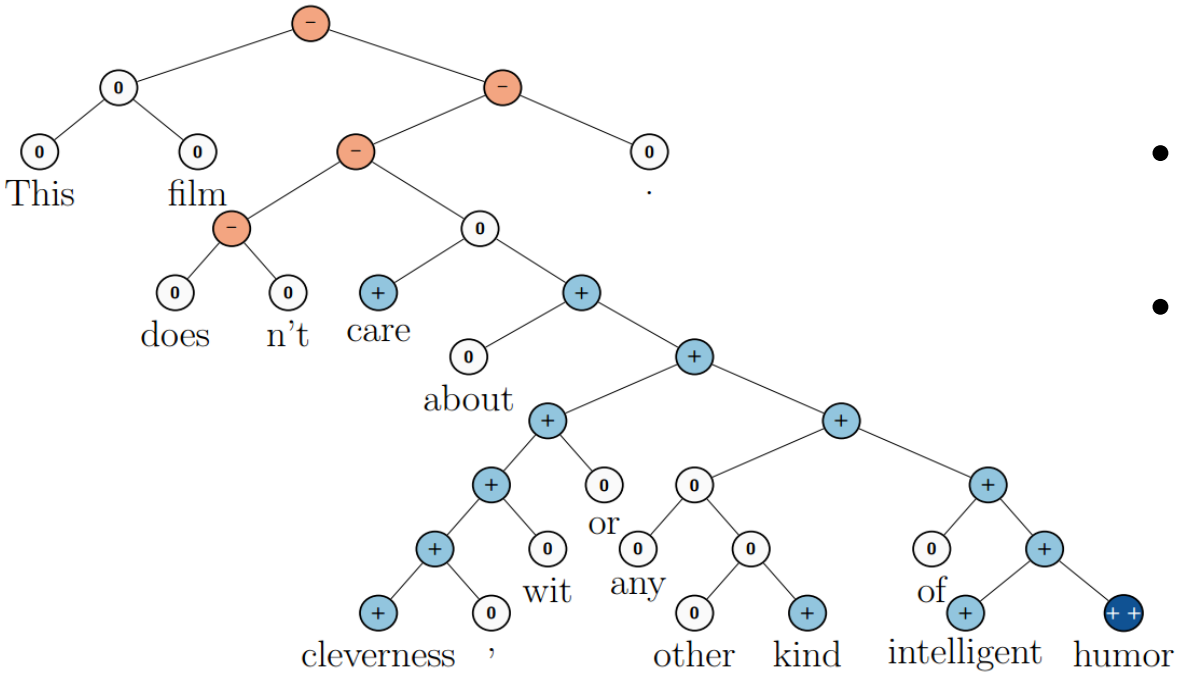
Recursive Neural Network Illustration (Animation)



Recursive Neural Networks

Recursive neural networks have had significant successes in a number of NLP tasks.

Socher *et al.* (2013) uses a recursive neural network to predict sentence sentiment:



- Fine-grained sentiment analysis
- Allocate positive and negative sentiment from words to phrases and to the entire sentence

Recursive Neural Networks

SemEval 2014: Predicting semantic relatedness of two sentences (Tai et al. 2015)

Ranking by mean word vector cosine similarity	Score
a woman is slicing potatoes	
a woman is cutting potatoes	0.96
a woman is slicing herbs	0.92
a woman is slicing tofu	0.92
a boy is waving at some young runners from the ocean	
a man and a boy are standing at the bottom of some stairs , which are outdoors	0.92
a group of children in uniforms is standing at a gate and one is kissing the mother	0.90
a group of children in uniforms is standing at a gate and there is no one kissing the mother	0.90
two men are playing guitar	
some men are playing rugby	0.88
two men are talking	0.87
two dogs are playing with each other	0.87

Ranking by Dependency Tree LSTM model	Score
a woman is slicing potatoes	
a woman is cutting potatoes	4.82
potatoes are being sliced by a woman	4.70
tofu is being sliced by a woman	4.39
a boy is waving at some young runners from the ocean	
a group of men is playing with a ball on the beach	3.79
a young boy wearing a red swimsuit is jumping out of a blue kiddies pool	3.37
the man is tossing a kid into the swimming pool that is near the ocean	3.19
two men are playing guitar	
the man is singing and playing the guitar	4.08
the man is opening the guitar for donations and plays with the case	4.01
two men are dancing and singing in front of a crowd	4.00

Table 4: Most similar sentences from a 1000-sentence sample drawn from the SICK test set. The Tree-LSTM model is able to pick up on more subtle relationships, such as that between "beach" and "ocean" in the second example.

Recursive NN based model can pick up more subtle semantic relatedness in sentences compared to word vector model, e.g., ocean and beach.

Vector Representation of Words

Vector space models (VSMs) represent (embed) words in a continuous vector space

- Theoretical foundation in Linguistics: Distributional Hypothesis
  - Words with similar meanings will occur with similar neighbors if enough text material is available (Rubenstein et al. 1967).

Approaches that leverage VSMs can be divided into two categories

Approach	Example	Description
Count-based methods	Latent semantic analysis	Compute how often some word co-occurs with its neighbor words in a large text corpus, and then map these count-statistics down to a small, dense vector for each word
Predictive methods	Neural probabilistic language model	Directly predict a word from its neighbors in terms of learned small, dense embedding vectors (considered parameters of the model)

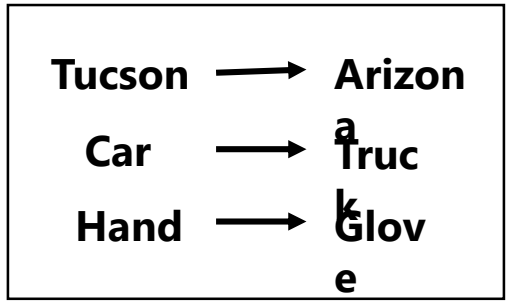
Word2vec –Vector Representation of Words (Mikolov et al. 2013)

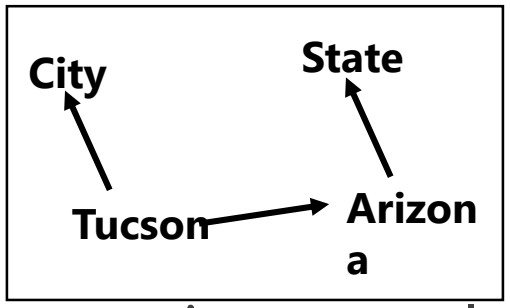
Word2vec: computationally-efficient, 2-layer predictive NN for learning word embeddings from raw text

- Considered deep for its ability to digest expansive data sets quickly

Can be used for unsupervised learning of words

- Relationships between different words
- Ability to abstract higher meaning between words (e.g., Tucson is a city in the state of Arizona)





Useful for language modeling, sentiment analysis, and more

Word2vec –Vector Representation of Words (Mikolov et al. 2013)

Its input is a text corpus and its output is a set of vectors or "embeddings" (feature vectors for words in that corpus)

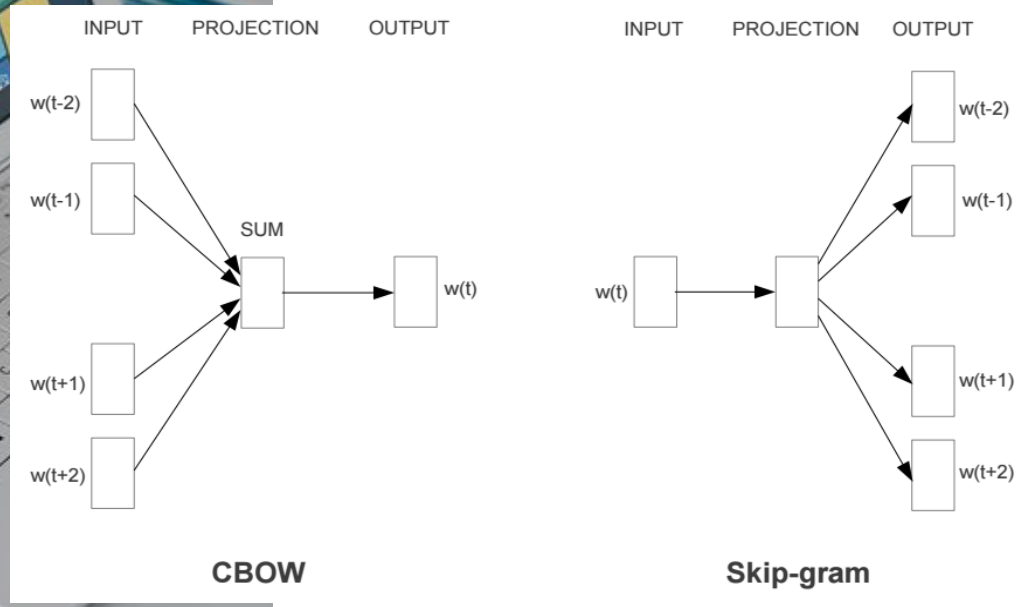
- Similarity between two embeddings represents conceptual similarity of words

Example results: words associated with *Sweden*, in order of proximity:

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

Word2vec –Vector Representation of Words (Mikolov et al. 2013)

Word2vec comes with two models:



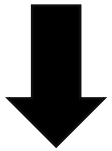
Model	Approach	Speed and Performance	Use case
Continuous Bag-of-Words model (CBOW)	The CBOW predicts the current word based on the context.	Faster to train than the skip-gram model	Predicts frequent words better
Skip-Gram model	Skip-gram predicts surrounding words given the current word.	Usually performs better than CBOW	Predicts rare words better

Word2vec –Vector Representation of Words (Mikolov et al. 2013)

Skip-gram learning:

Given $w_{0,}$ predict w_{-2}, w_{-1}, w_1, and w_2

w_{-2}	w_{-1}	w_0	w_1	w_2
?	?	Network	?	?



w_{-2}	w_{-1}	w_0	w_1	w_2
Recurrent	Neural		Language	Model

Conversely, CBOW tries to predict w_0 when given w_{-2}, w_{-1}, w_1, and w_2

Running Word2Vec

Download: https://code.google.com/archive/p/word2vec/

Word2Vec comes bundled with many files. Two important ones:

Word2vec.c - the actual Word2Vec program written in C; is executed in command line

Demo-word.sh - shell script containing example of how to run *Word2Vec.c* on test data

To use Word2Vec, you need:

1. A corpus (e.g., collection of tweets, news articles, product reviews)

Word2Vec expects a sequence of sentences as input

One input file containing many sentences, with one sentence per line

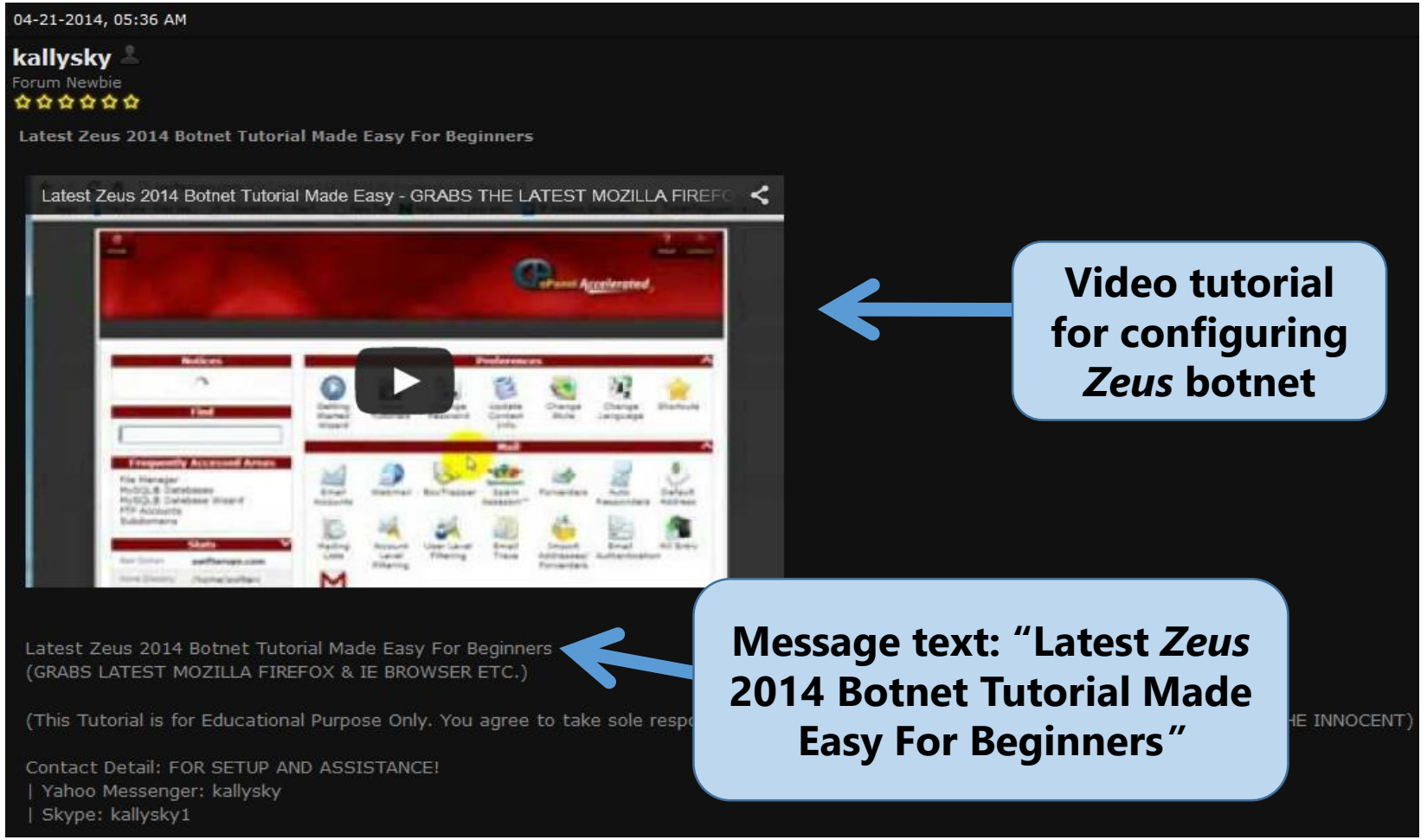
2. A C programming language compiler

Unix environments are easiest - Linux generally ships with 'gcc' pre-installed, OSX can use Xcode

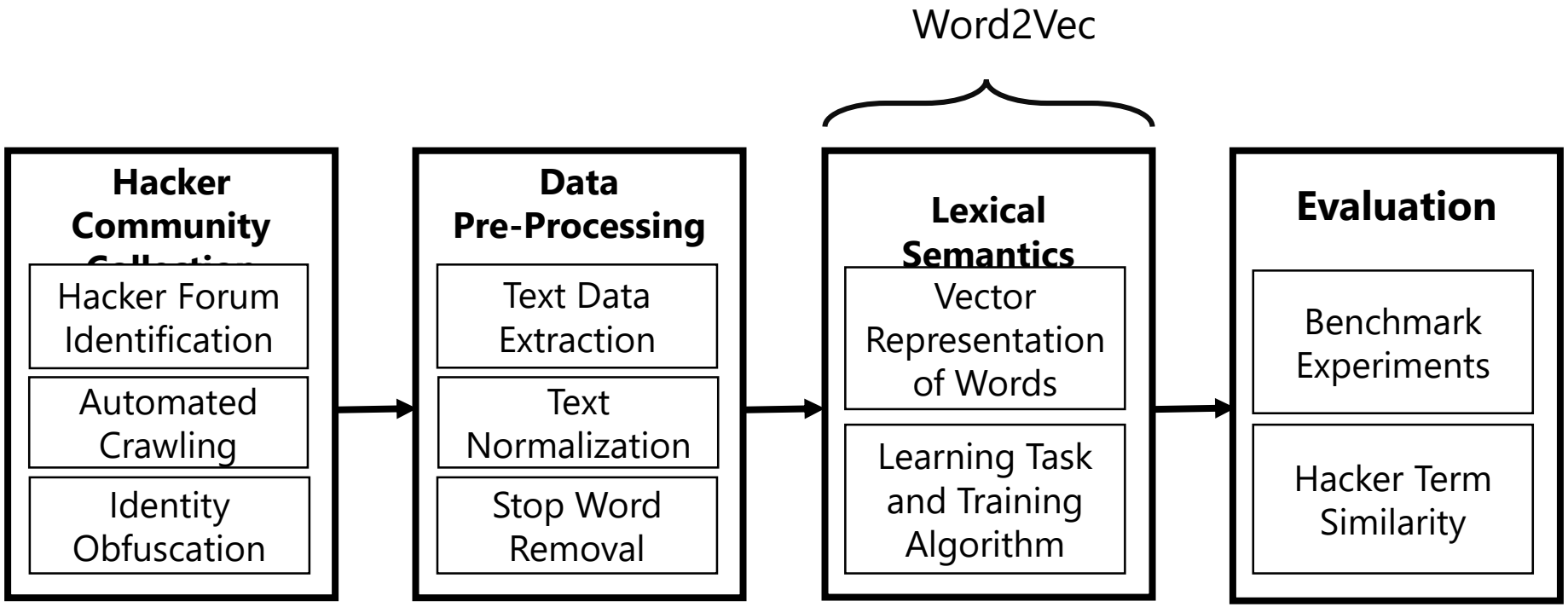
Word2Vec is useful for language modeling tasks,

Word2Vec Example

- Example: *Zeus* refers to a botnet and not Greek mythology
- Word2Vec can provide automated understanding of unfamiliar terms and language
- We further explore this use-case as an illustrative example



Word2Vec Example

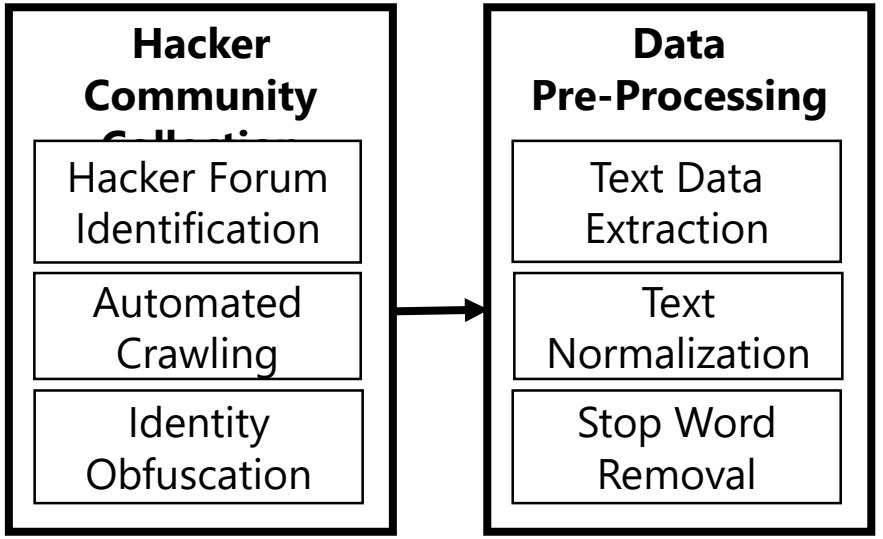


Benjamin, V., & Chen, H. (2015). Developing understanding of hacker language through the use of lexical semantics. In *2015 IEEE International Conference on Intelligence and Security Informatics (ISI),* (pp. 79-84).

Word2Vec Example

We identify and collect two hacker forums:

Forum	Members	Threads	Posts	Time Span
HackFive	947	1,108	5,334	1/24/2013 –12/30/2014
HackHound	633	507	3,622	12/10/2012 –12/30/2014



Text Data Extraction

Extract forum message data from raw webpages

Thread titles, message bodies

Text Normalization

Strip punctuation from words

Remove stop words

Scrubbed data is fed into Word2Vec one sentence at a time

Word2Vec's input is a single text file

Each row of the input file represents one sentence

Word2Vec Example

Many parameters can be altered to change Word2Vec behavior

- *Architecture*: Skip-gram (slower, better for infrequent words) vs CBOW (fast)
- *Training algorithm*: Hierarchical Softmax (better for infrequent words) vs Negative Sampling (better for frequent words, better with low dimensional vectors)
- *Dimensionality of the word vectors*: How many elements per word vector
- *Context (window) size*: Size of sliding window used to iterate through words for training

Skip-gram with negative sampling configuration generally provides best performance (Mikolov et al., 2013)

- Additionally, Word2Vec comes default with recommended parameters
- Default size for word vector dimensionality is 200 words
- Default context window size of 10 words

Lexical Semantics

Learning Task and Training Algorithm

Word Embeddings

Word2Vec Example

Lexical Semantics

Learning Task and Training Algorithm

Word Embeddings

The output of Word2Vec is a file called *Vectors.bin*

- Can be opened and viewed in plaintext
- Contains embeddings of each word in corpus

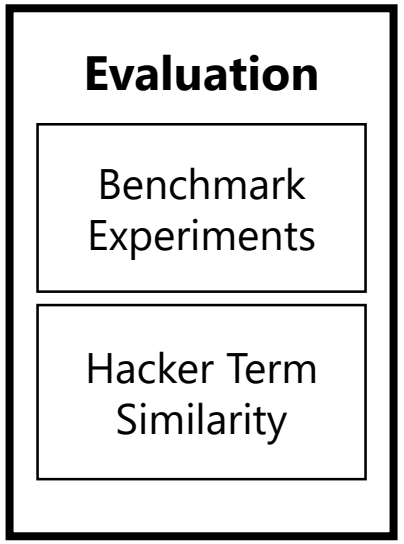
Generated embeddings can be used in two ways:

- Directly evaluated to better understand underlying corpus
- Fed into other models and deep learning algorithms as features

Word2Vec Example

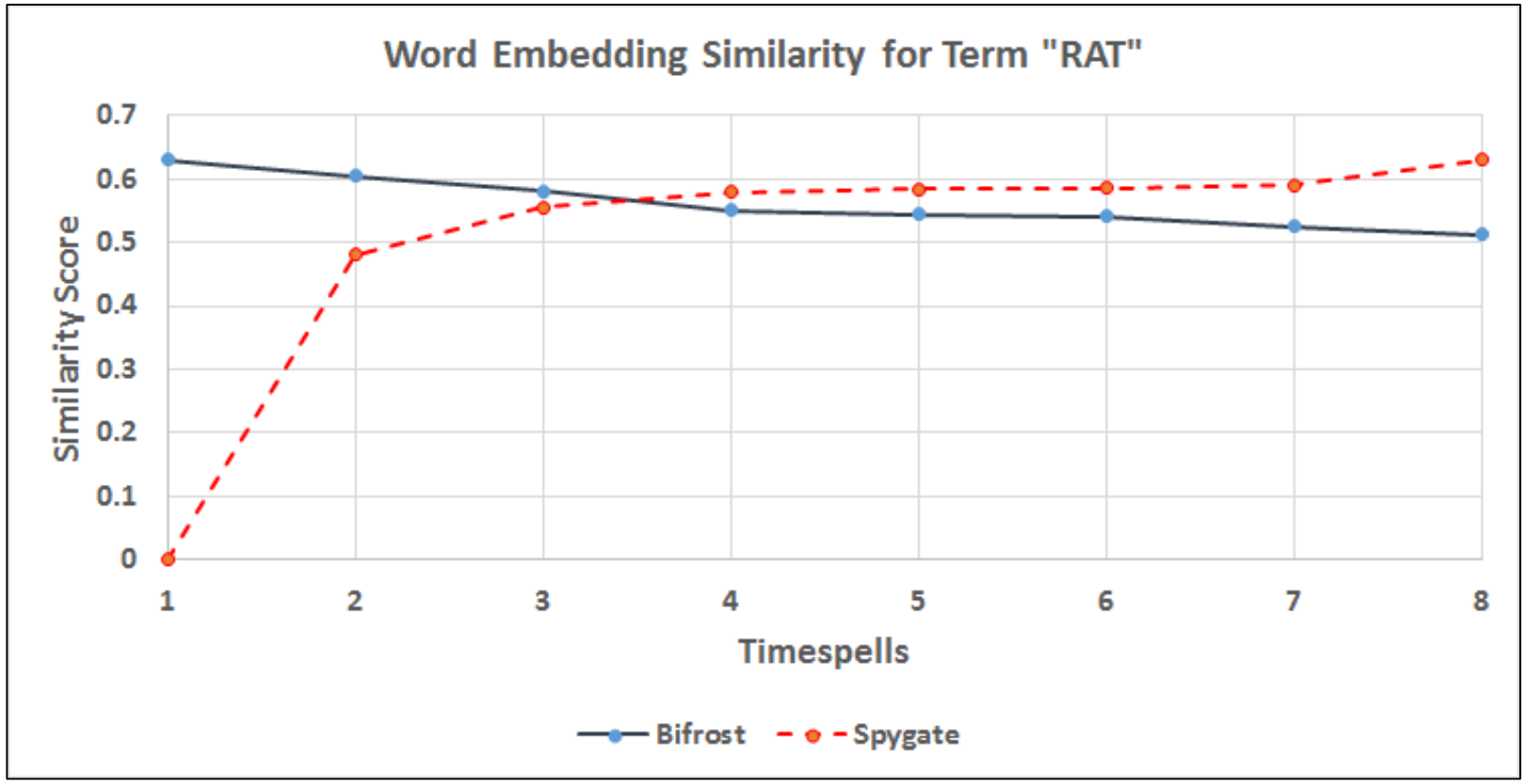
- We directly evaluate word embeddings in this study
  - Embeddings are vectors, can use cosine similarity to find similar words
  - Useful in hacker context to discover new hacker terms, tool names, etc.

	Most Similar Embeddings for "Botnet"	
	Word	*Similarity Score*
1	**Citadel**	**0.561456**
2	**Zeus**	**0.554653**
3	Partners	0.548900
4	**Pandemiya**	**0.545221**
5	**Mailer**	**0.540075**
6	Panel	0.524557
7	Linksys	0.498224
8	**Cythosia**	**0.480465**
9	**Phase**	**0.464738**
10	**Spyeye**	**0.459695**
P@10	70%	



Word2Vec Example

- *Bifrost* and *Spygate* are remote administration tools (RATs) that grant hackers backdoor access to victim computers
  - Can look at their similarity with word RAT over time to assess evolving significance in discussions concerning RATS



Evaluation

Benchmark Experiments

Hacker Term Similarity

Recursive Neural Network Example – Identifying Key Carding Sellers

The carding community rely heavily on the online black market for exchanging malwares and stolen data.

Online black market for carders:

- *Sellers* advertise their malwares and/or stolen data by posting a thread in the carding community
- *Buyers* leave feedback commenting the their evaluation of the seller.

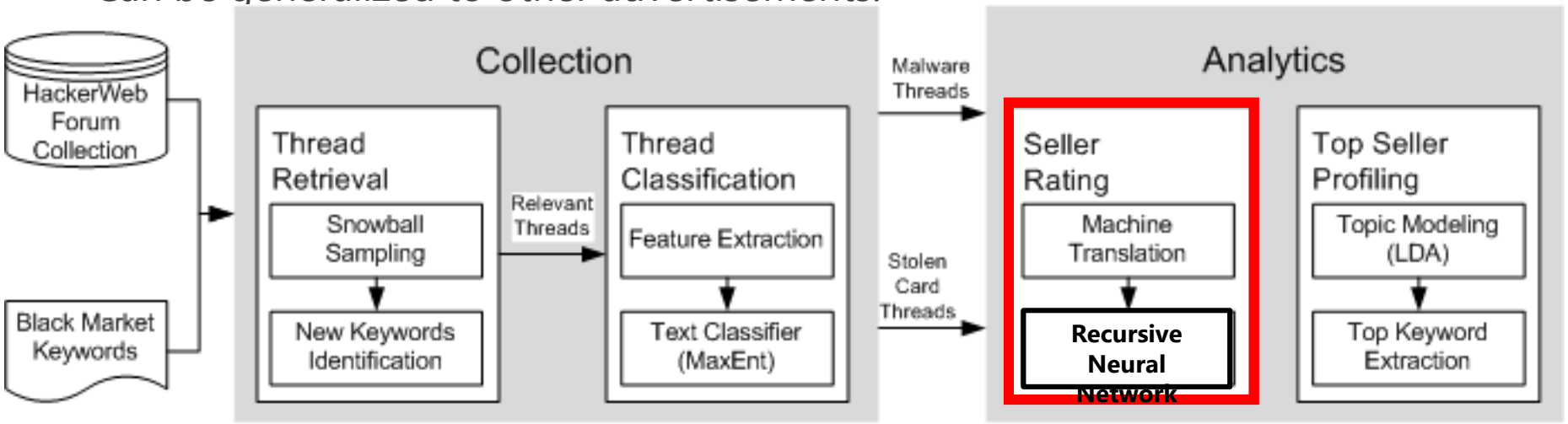
Objective: to identify key carding sellers through evaluating buyers feedback.

Recursive Neural Network Example – Identifying Key Carding Sellers

Input: Original threads from hacker forums

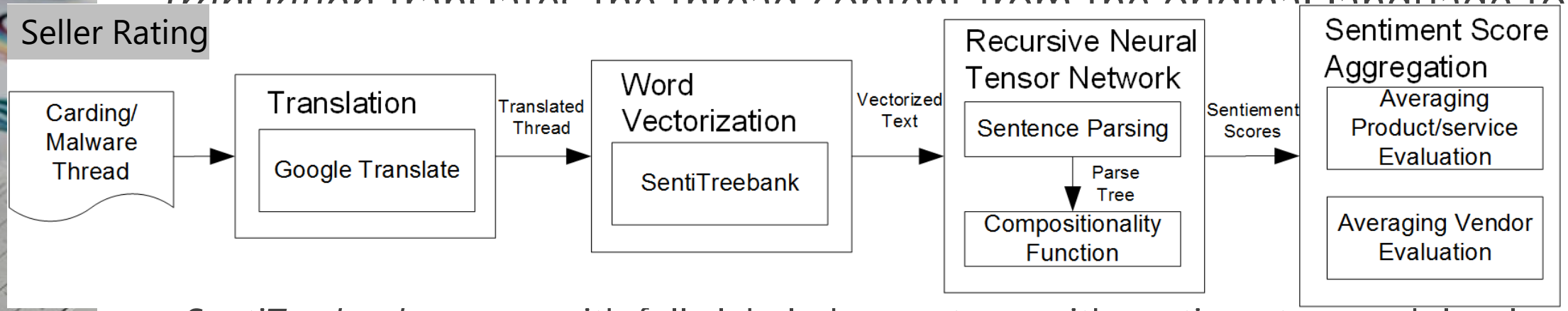
Preprocessing:

- Thread Retrieval: Identifying threads related to the underground economy by conducting snowball sampling-based keywords search
- Thread Classification: Identifying advertisement threads using MaxEnt classifier
  - Focusing on malware advertisements and stolen card advertisement
  - Can be generalized to other advertisements.



Recursive Neural Network Example – Identifying Key Carding Sellers

Translation translates the thread content from the original language to



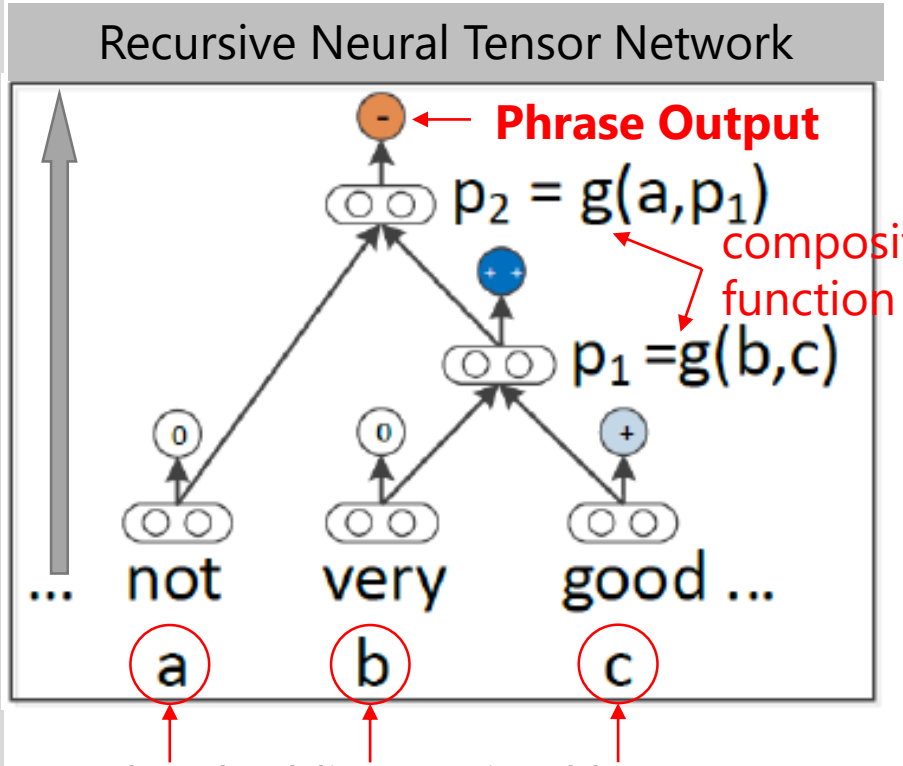
SentiTreebank: corpus with fully labeled parse trees with sentiment on each level

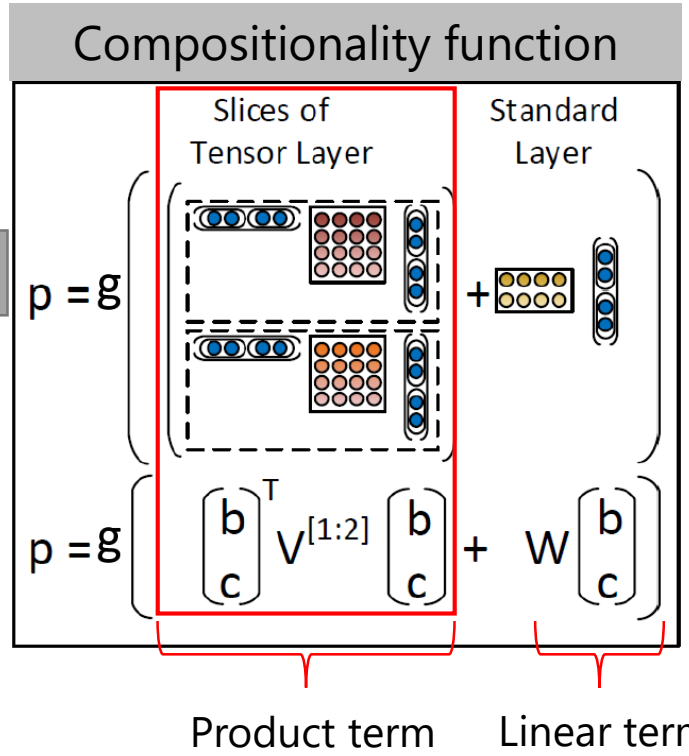
Recursive Neural Tensor Network parses the sentence into a binary tree and aggregate semantics from constituents recursively.

Sentiment Score Aggregation averages the sentiment measures for each advertisement and seller.

Recursive Neural Network Example – Identifying Key Carding Sellers

Example feedback: "*The dump is not very good*"





The additional *tensor layer* makes the compositional function more powerful by introducing the product of the constituent word embeddings.

Output of the example feedback: **Negative**

Recursive Neural Network Example – Identifying Key Carding Sellers

Top 3 Best/Worst Malware and Stolen Data Sellers for Each Forum								
	Top 3 Best				Top 3 Worst			
	Malware		Stolen Data		Malware		Stolen Data	
Rank	User	Score	User	Score	User	Score	User	Score
	Antichat							
1	L**G	5	i**o	3.6	N**g	1.8	l**s	2.3
2	V**U	4.5	a**s	3.5	k**a	2	P**A	2.3
3	g**l	4	D**R	3.4	D**i	2	s**8	2.4
	CrdPro							
1	H**l	4	R**r	4.4	N**0	2	f**4	1.3
2	b**t	4	F**x	4	1**4	2	s**3	1.3
3	S**r	4	R**c	4	M**D	2	l**u	1.3
	Zloy							
1	P**t	5	B**r	4	r**t	1.5	r**y	1
2	D**n	4	B**1	4	w**0	1.6	m**n	1
3	D**f	4	s**c	4	g**n	2	j**a	2

Outline

- Introduction
  - Motivation
  - Deep Learning Intuition
- Neural Network
  - Neuron, Feedforward algorithm, and Backpropagation algorithm
  - Limitations
- Deep Learning
  - Deep Belief Network: Autoencoder & Restricted Boltzman Machine
  - Convolutional Neural Network
- Deep Learning for Text Mining
  - Word Embedding
  - Recurrent Neural Network
  - Recursive Neural Network
- Conclusions

Conclusion

Deep learning = Learning Hierarchical Representations

Deep learning is thriving in big data analytics, including *image processing*, *speech recognition*, and *natural language processing*.

Deep learning has matured and is very promising as an artificial intelligence method.

Still has room for improvement:

- Scaling computation
- Optimization
- Bypass intractable marginalization
- More disentangled abstractions
- Reasoning from incrementally added facts

Deep Learning Resources

Name	Language	Link	Note
Pylearn2	Python	http://deeplearning.net/software/pylearn2/	A machine learning library built on Theano
Theano	Python	http://deeplearning.net/software/theano/	A python deep learning library
Caffe	C++	http://caffe.berkeleyvision.org/	A deep learning framework by Berkeley
Torch	Lua	http://torch.ch/	An open source machine learning framework
Overfeat	Lua	http://cilvr.nyu.edu/doku.php?id=code:start	A convolutional network image processor
Deeplearning4j	Java	http://deeplearning4j.org/	A commercial grade deep learning library
Word2vec	**C**	**https://code.google.com/p/word2vec/**	**Word embedding framework**
GloVe	C	http://nlp.stanford.edu/projects/glove/	Word embedding framework
Doc2vec	C	https://radimrehurek.com/gensim/models/doc2vec.html	Language model for paragraphs and documents
StanfordNLP	**Java**	**http://nlp.stanford.edu/**	**A deep learning-based NLP package**
TensorFlow	**Python**	**http://www.tensorflow.org**	**A deep learning based python library**

References

Bordes, A., Chopra, S., & Weston, J. (2014). Question answering with subgraph embeddings. *arXiv preprint arXiv:1406.3676*.

Graves, A., Mohamed, A. R., & Hinton, G. (2013, May). Speech recognition with deep recurrent neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on* (pp. 6645-6649). IEEE.

Graves, A. (2013). Generating sequences with recurrent neural networks.*arXiv preprint arXiv:1308.0850*.

Irsoy, O., & Cardie, C. (2014, October). Opinion Mining with Deep Recurrent Neural Networks. In *EMNLP* (pp. 720-728).

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Rubenstein, H., & Goodenough, J. B. (1965). Contextual correlates of synonymy. *Communications of the ACM*, *8*(10), 627-633.

Socher, R., Perelygin, A., Wu, J. Y., Chuang, J., Manning, C. D., Ng, A. Y., & Potts, C. (2013, October). Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*(Vol. 1631, p. 1642).

Sutskever, I., Vinyals, O., & Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems* (pp. 3104-3112).

Tai, K. S., Socher, R., & Manning, C. D. (2015). Improved semantic representations from tree-structured long short-term memory networks. *arXiv preprint arXiv:1503.00075*.