Deep Learning Architectures and Algorithms

In-Jung Kim
Handong Global University
2016. 12. 2.

Agenda

- Introduction to Deep Learning
- RBM and Auto-Encoders
- Convolutional Neural Networks
- Recurrent Neural Networks
- Reinforcement Learning
- Deep Reinforcement Learning
- Q&A

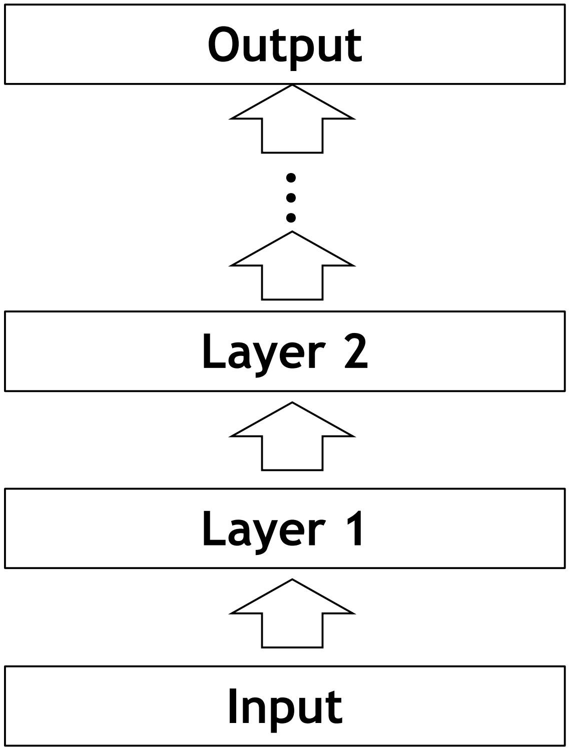
Deep Learning

- A branch of machine learning based on a set of algorithms that attempt to model **high-level abstractions in data**, mostly, based on **deep networks**.

  - Each layer combines input features to produce high-level features

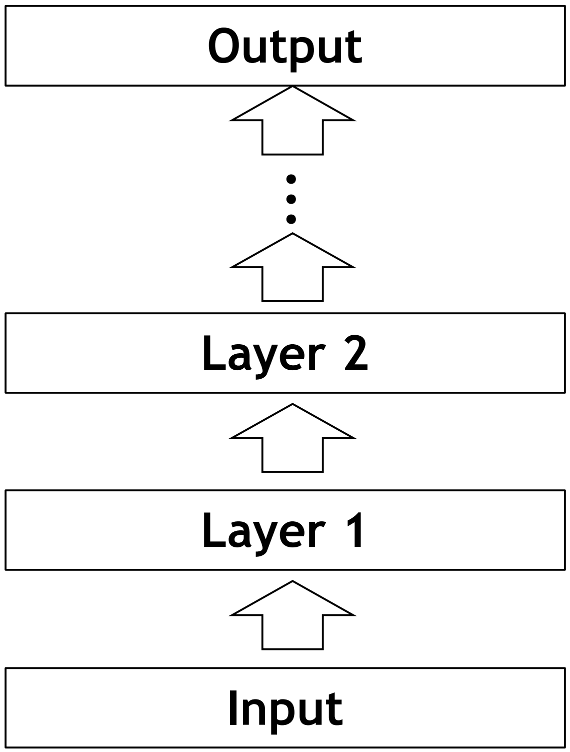
$$o = f(\sum_{i=1}^{n} w_i x_i + \theta)$$

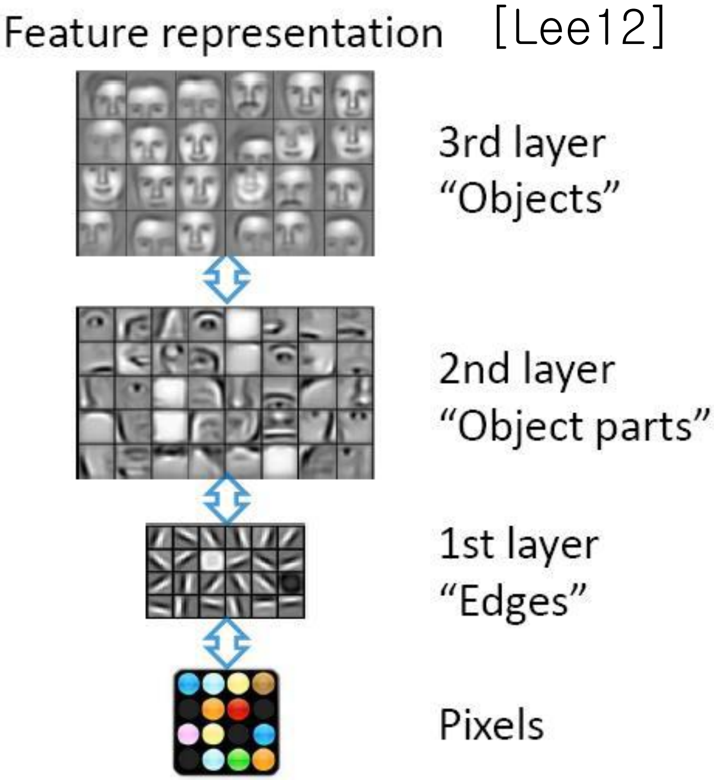
  - A neural network with many layers can extract high-level features



Stepwise Abstraction

- Effective in learning **high-level representation** by step-wise abstraction





Popular Deep Learning Architectures

- Traditional neural networks and their extensions
  - MLP, RBF, auto-encoders, …
- Stochastic models
  - Boltzmann machine, RBM, DBN, DBM, …
- Convolutional neural networks (CNN)
  - Learns position independent local features
  - Spatially shared connections
  - Combines heterogeneous layers
- Recurrent neural networks (RNN)
  - Neural network with memory
  - Model for dynamic process
  - Temporarily shared connections

Popular Deep Learning Architectures

- Hybrid models
  - Convolutional RBM
    - CNN + contrastive divergence
  - Predictive sparse decomposition
    - Sparse coding + deconvolution
  - Recurrent convolutional neural networks (RCNN)
    - Recurrent convolution layer
  - Long-term recurrent convolutional network
    - LSTM + CNN
  - Attention models
    - CNN + glimpse network (RNN)
  - Adversarial neural networks
    - Generative model + discriminative model

Learning Strategies

- Supervised learning: “learning with teacher”
  - Adjust model to produce desired outputs (label)
  - Optimize network for a specific task
- Unsupervised learning: “learning without teacher”
  - Clustering
  - Reproduction
    - Feature extraction, data compression
    - Layer-wise unsupervised pre-training
  - Latent variable models
    - Hidden cause

Learning Strategies

- Semi-supervised learning
  - Learn from <small volume of labeled data> + <large volume of unlabeled data>
  - Improves generalization
- Reinforcement learning: “learning from critics”
  - Interaction between agent and environment
    - Action: agent → environment
    - Reward: environment → agent
  - Adjust model to maximize reward

Why Deep Networks?

- Efficient in learning high-level feature
  - High-level features are more informative and robust than lower-level features
- Integrated learning
  - DNN integrates feature extractor and classifier in a single network
- Efficient in modeling of highly varying functions
- Large capacity
  - DNN can learn very well from a huge volume of samples
- Framework that embraces various methodologies and techniques

Challenges with Deep Networks

- Hard to optimize
  - Conventional learning algorithm does not work well for deep fully connected networks starting from random weights

  ➔ New learning algorithms

- A large number of parameters

  ➔ A huge volume of training samples is now available.

  ➔ Techniques to improve generalization ability

  Ex) sparse coding, virtual sample generation, dropout

- Requires heavy computation

  ➔ Acceleration H/W (GPU, cluster, ASIC, FPGA)

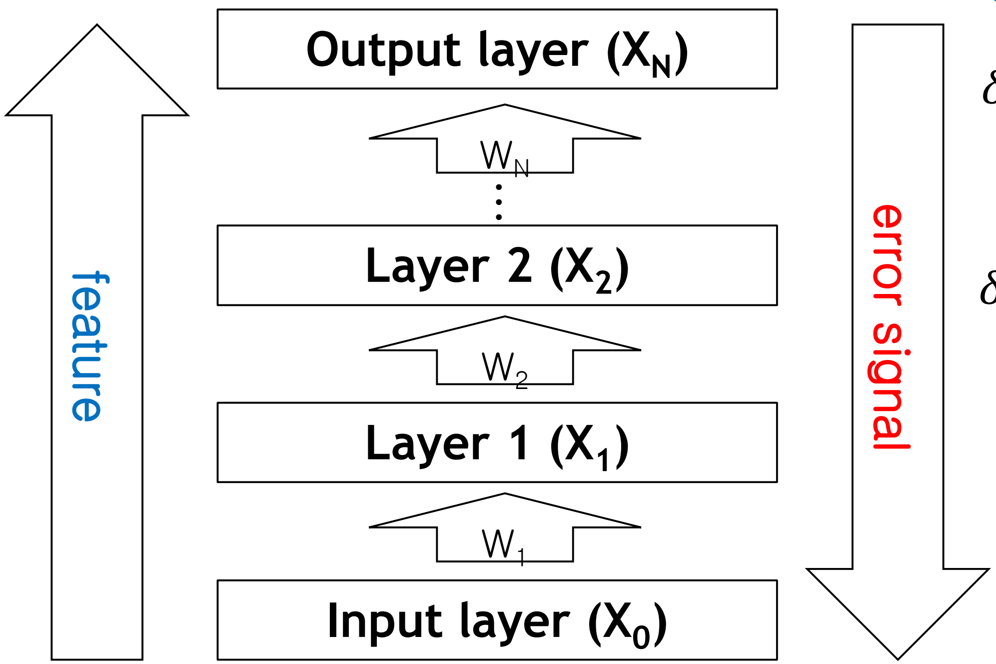
The Back-Propagation Algorithm

- Gradient descent algorithm to minimize error E.

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}} \qquad \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \delta_j x_i$$

$$\delta_j = \frac{\partial E}{\partial net_j}$$

$$x_j = \mathrm{f}(\underbrace{\textstyle\sum_i w_{ij} x_i + \theta}_{net_j})$$



$$\delta_i = f'(net_i) \sum_j w_{ij} \delta_j$$

Diminishing Gradient Problem

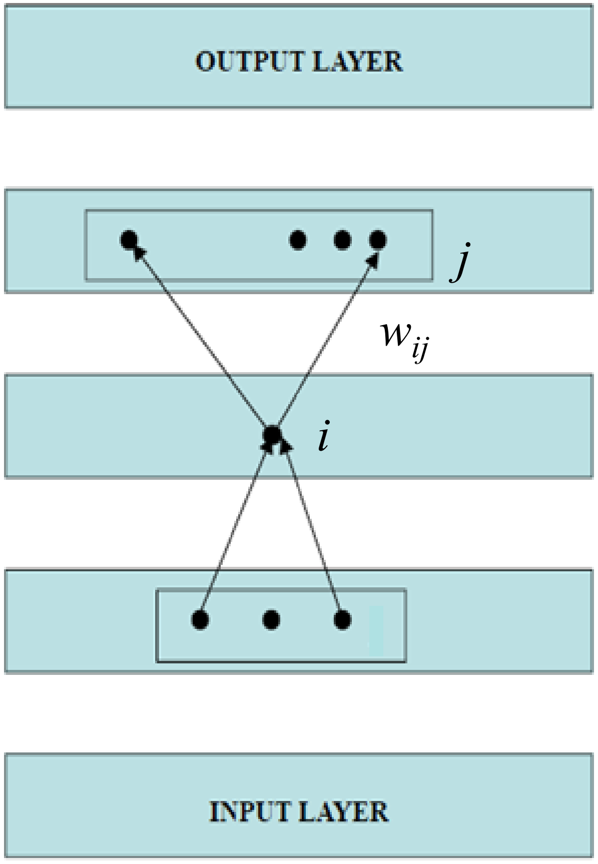
- BP does not work on deep networks
  - Error signals from many nodes are blended together.

  ➔ become dim and vague on bottom layers

  - Error signal (δ_i)
    - Signal that guides learning
  - Error signal

    at a non-output node i

$$\delta_i = f'(net_i) \sum_j w_{ij} \delta_j$$



Agenda

- Introduction to Deep Learning
- <u>RBM and Auto-Encoders</u>
- Convolutional Neural Networks
- Recurrent Neural Networks
- Reinforcement Learning
- Deep Reinforcement Learning
- Q&A

Layer-wise Unsupervised Pre-training

- Conventional back-propagation algorithm does not work well for deep neural networks starting from random weights.

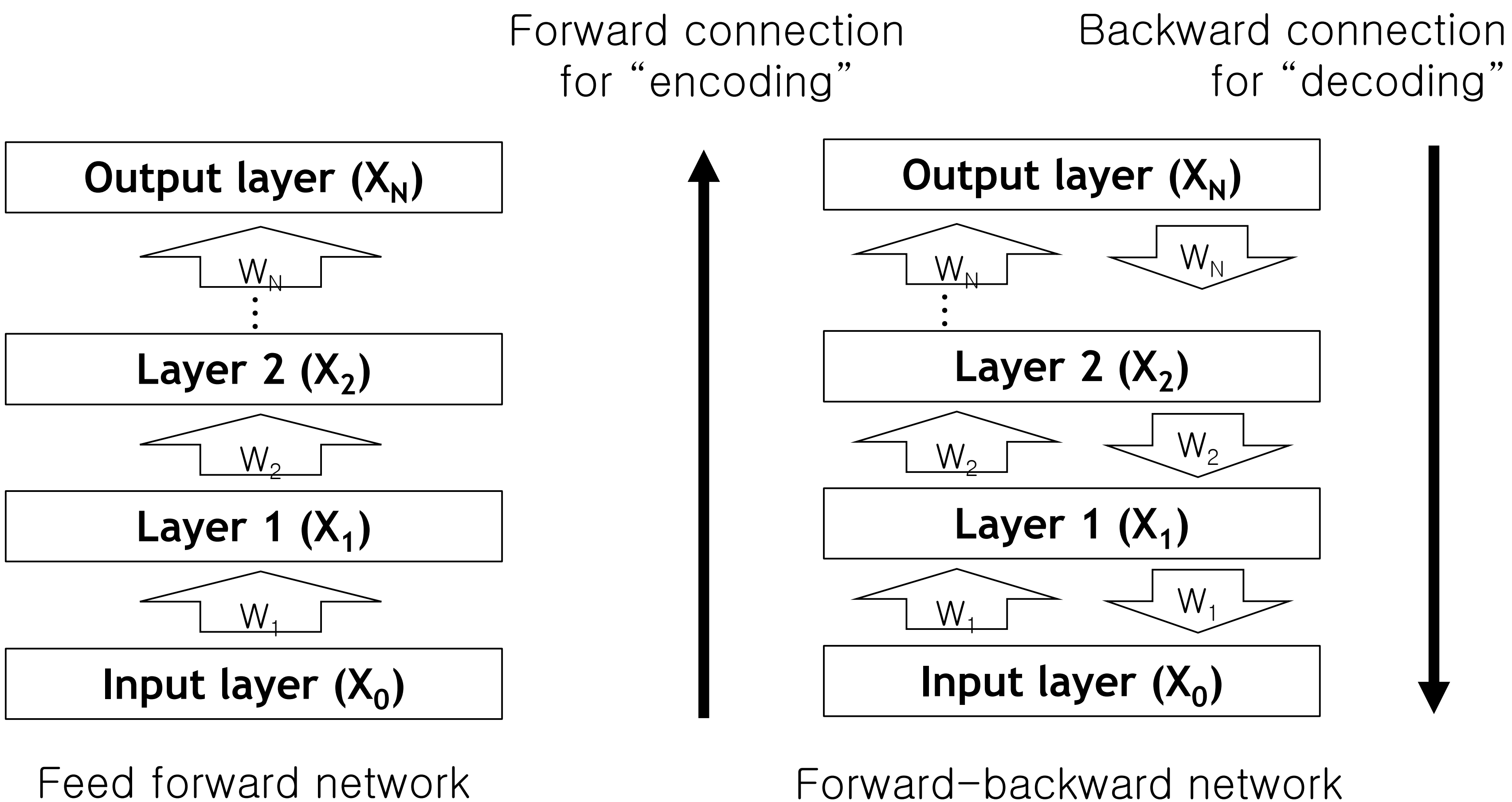
- **Layer-wise unsupervised pre-training algorithm**

  Ex) DBN[Hinton2006], stacked auto-encoders[Bengio2006]
  - First, place the weights near a local optimal position by unsupervised learning algorithm
  - Then, conventional supervised learning algorithms work fine

- Based on generative neural networks

Generative Neural Networks

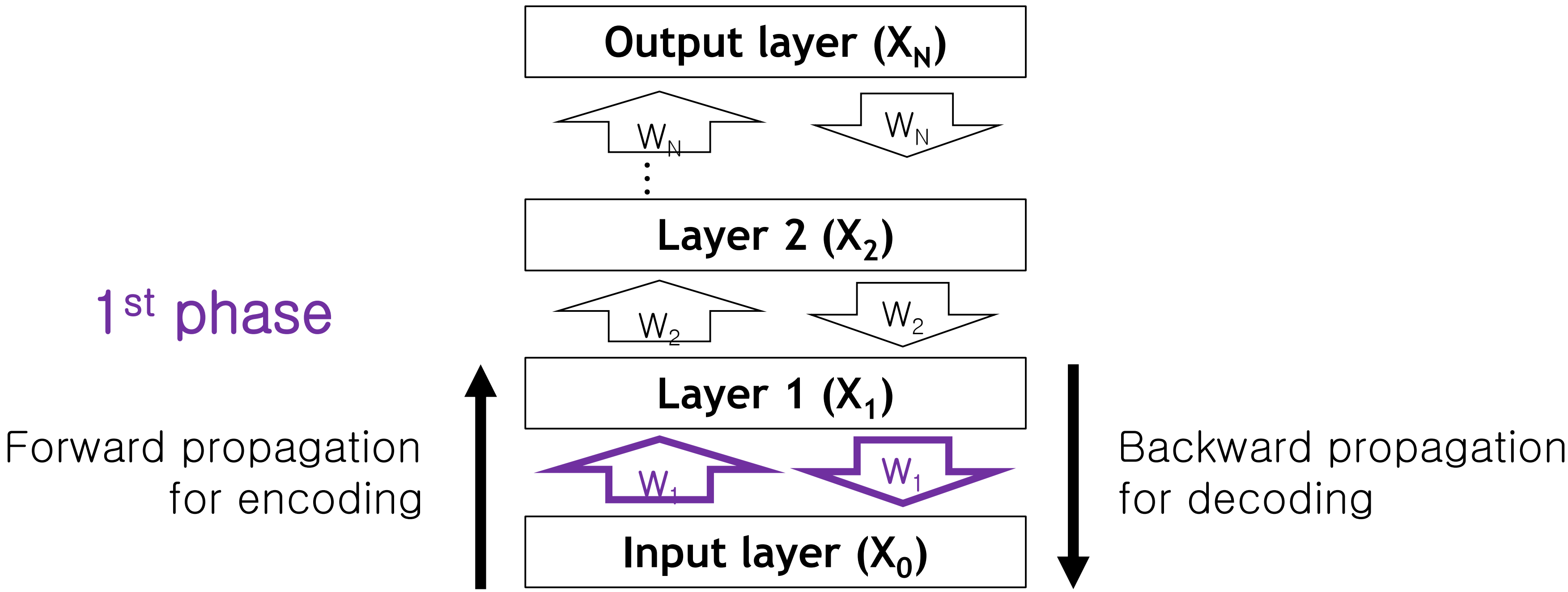
- Neural networks with forward-backward connections



Layer-wise Unsupervised Pre-training

- Starting from bottom layer, train each layer to reproduce the input

  Input ➔ encoding ➔ hidden ➔ decoding ➔ reprod. of input

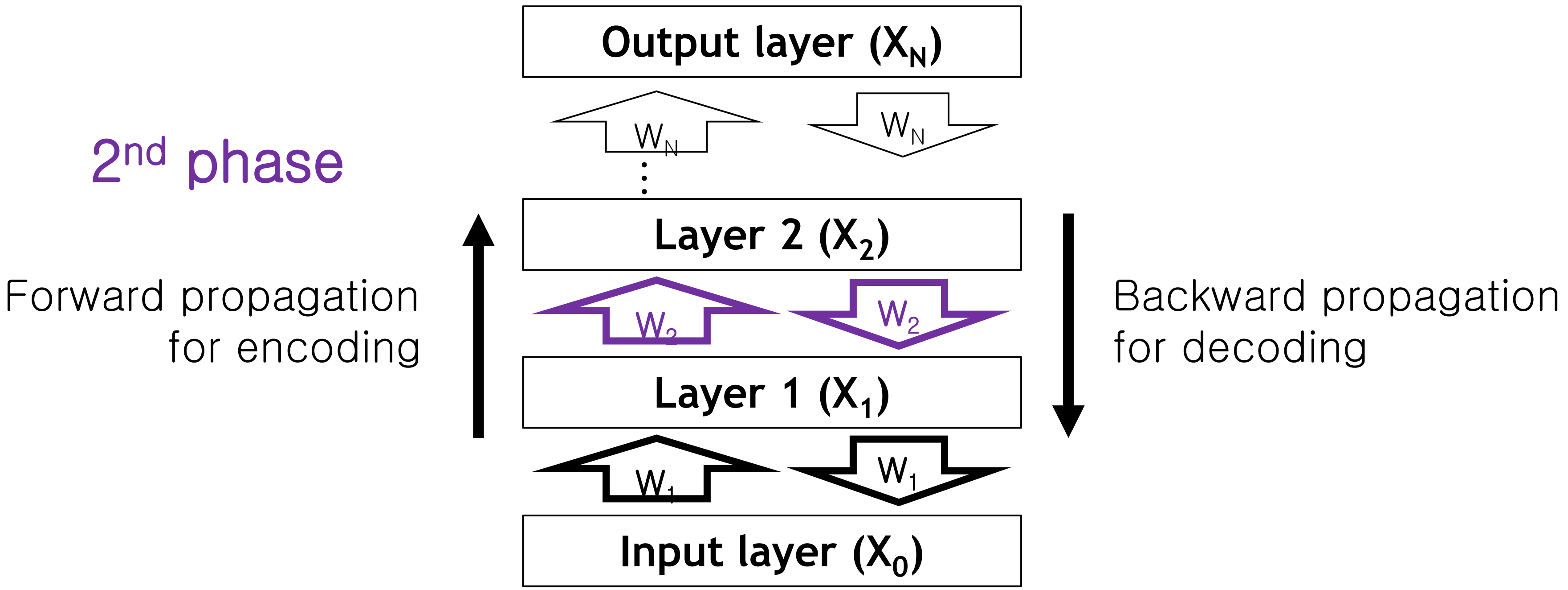


Forward-backward network

Layer-wise Unsupervised Pre-training

- Starting from bottom layer, train each layer to reproduce the input

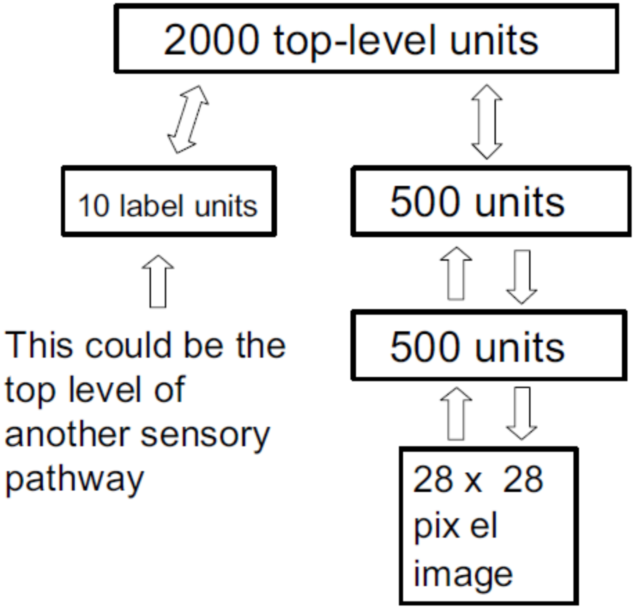
  Input ➔ encoding ➔ hidden ➔ decoding ➔ reprod. of input

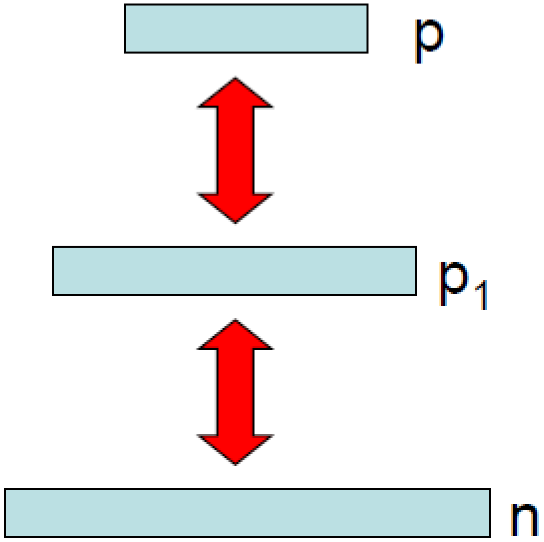


Forward-backward network

Pretraining-based Methods

- Deep belief network [Hinton2006]
  - Stacked RBM
- Stacked Autoencoders [Bengio2006]





Energy Based Models

- Energy-based models: probabilistic models that associate scalar energy to configuration of variables
  Ex) Boltzmann machine, MRF, …

- Probability of energy-based models
  - $P(X) = \frac{e^{-Energy(X)}}{Z} = \frac{e^{-Energy(X)}}{\sum_X e^{-Energy(X)}}$
    - $Z = \sum_X e^{-Energy(X)}$ is called partition function

- Probability of energy-based models with hidden nodes
  - $P(V, H) = \frac{e^{-Energy(V,H)}}{Z} = \frac{e^{-Energy(V,H)}}{\sum_{(V,H)} e^{-Energy(V,H)}}$

Restricted Boltzmann Machine

- Energy of RBM
  - $Energy(V,H) = -b^T V - c^T H - H^T W V$

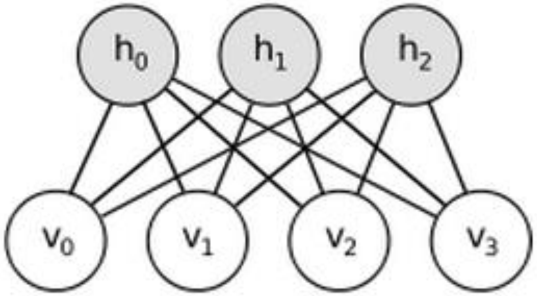
- $\boldsymbol{b}, \boldsymbol{c}$: bias vectors
- $\boldsymbol{W}$: weight matrix

- Probability of (V,H)

$$P(V,H) = \frac{e^{-Energy(V,H)}}{Z} = \frac{e^{-Energy(V,H)}}{\sum_{(V,H)} e^{-Energy(V,H)}}$$

Then,

- $P(H|V) = \prod_j P(h_j|V)$
  - $P\left(h_j^{t+1} = 1|V^t\right) = sigmoid(W_j V^t + c_j)$
- $P(V|H) = \prod_i P(v_i|H)$
  - $P\left(v_i^{t+1} = 1|H^{t+1}\right) = sigmoid(W_i^T H^{t+1} + b_i)$



Restricted Boltzmann Machine

- Probability of visible variables
  - $P(V) = \sum_H P(V, H) = \sum_H \frac{e^{-Energy(V,H;\theta)}}{Z}$
- Free energy of visible variables
  - Marginalization of energies in log domain

$$FreeEnergy(V) = -\log(\sum_H e^{-Energy(V,H)})$$

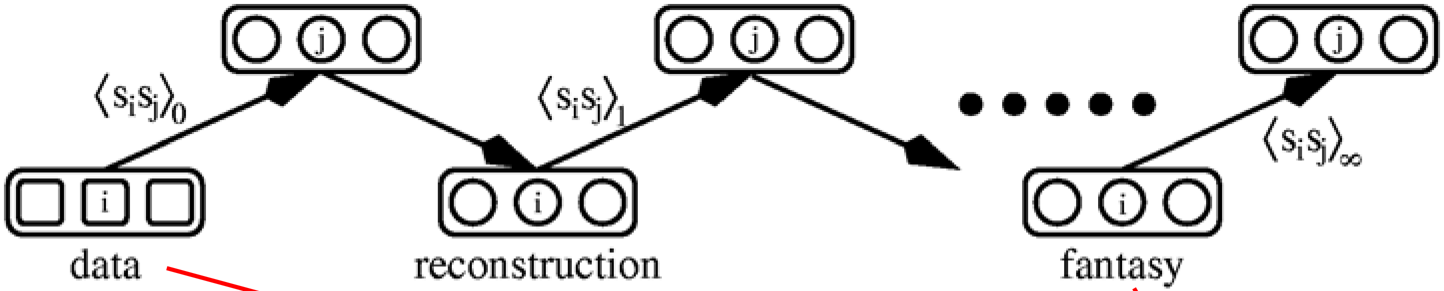
  - Then,

$$P(V) = \frac{e^{-FreeEnergy(V)}}{Z}$$

$$\log P(V) = -FreeEnergy(V) - \log Z$$

Training of RBM [Hinton02]

- Train weights to minimize Contrastive Divergence
  1. Run MCMC chain $V_0, V_1, V_2, \cdots, V_n$ for n steps starting from a training sample V_0



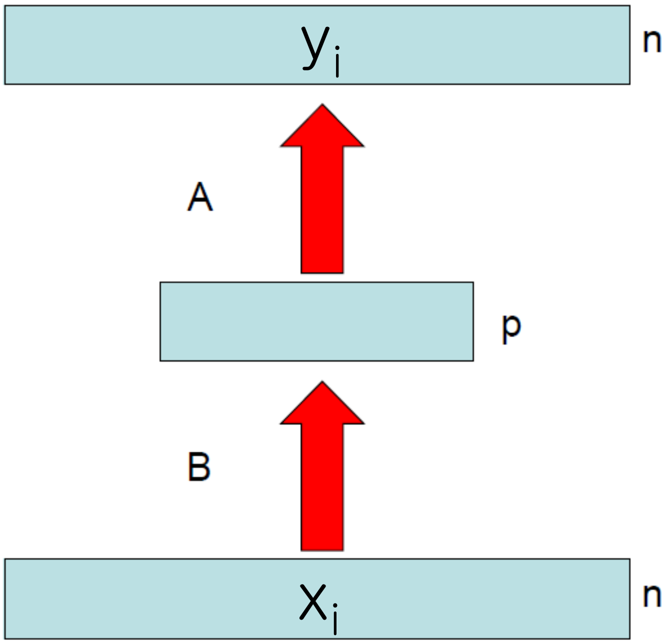
  2. CD_n update after seeing sample V_n

$$\theta^{t+1} = \theta^t + \eta\left(-\frac{\partial FreeEnergy(V_0;\theta)}{\partial\theta} + \frac{\partial FreeEnergy(V_n;\theta)}{\partial\theta}\right)$$

$$\approx \theta^t + \eta\left(-\frac{\partial Energy(V_0, H_0;\theta)}{\partial\theta} + \frac{\partial Energy(V_n, H_n;\theta)}{\partial\theta}\right)$$

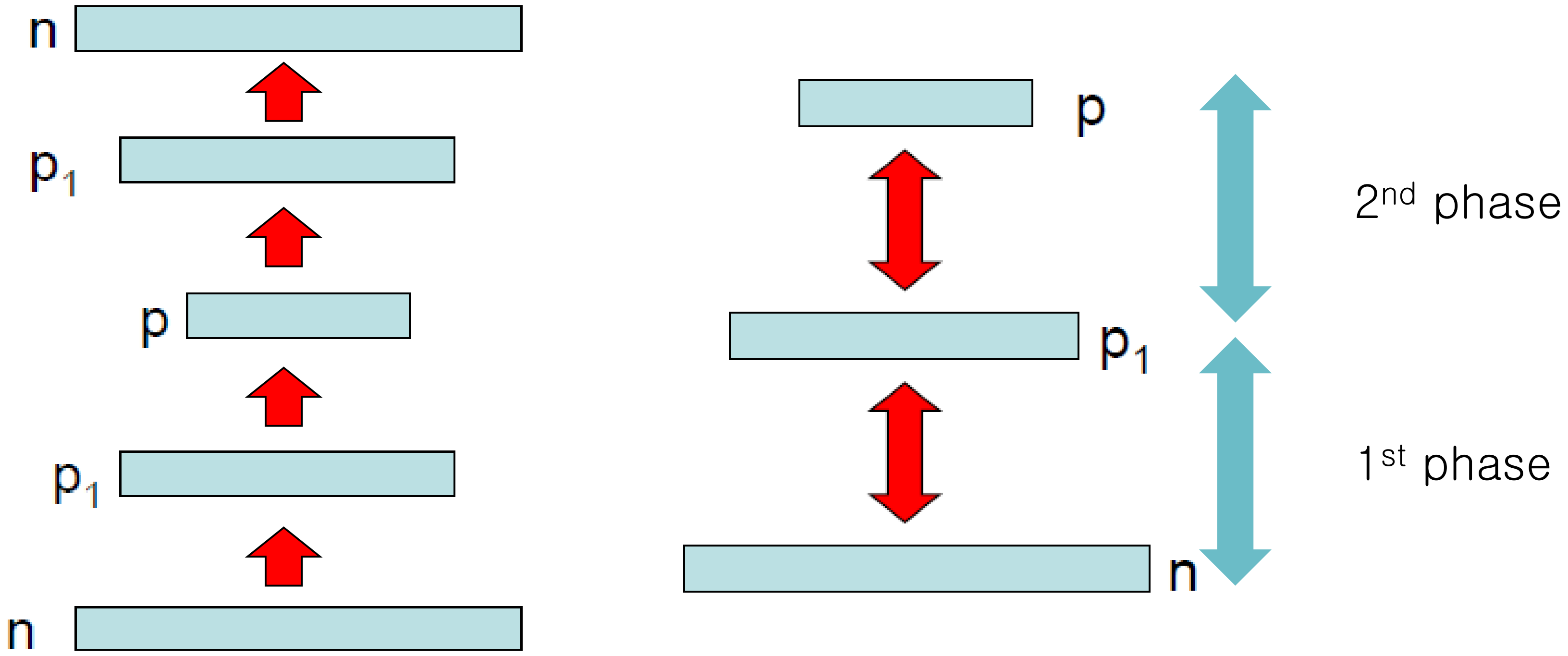
Autoencoder

- Auto-encoder is an ANN whose **desired output is the same as the input**.
  - The aim of an auto-encoder is to learn a **compressed representation** (encoding) for a set of data.
- Training algorithm
  - Given $x_1, \cdots, x_m$ training vectors over $\mathbb{R}^N$,
  - Find weight vectors A and B that minimize: $\Sigma_i (y_i - x_i)^2$



Stacked Autoencoders

- After training, hidden nodes extract **features** from the input nodes.
- Stacking autoencoders constructs a deep network



Sparse Autoencoder [LeCun07]

- Encoder/decoder paradigm
  - Encoder: $f_{enc}(Y) = W^T Y + b_{enc}$
  - Decoder: $f_{dec}(Z) = Wl(Z) + b_{enc}$

  Y: input vector, Z: code vector,

  W: weight matrix, $l(.)$: activation function

- Energy-based model

- The loss function to minimize

  $L(Y, Z) = \alpha_e \|Z - f_{enc}(Y)\|_2^2 + \|Y - f_{dec}(Z)\|_2^2 + \alpha_s h(Z) + \alpha_r \|W\|_1$
  - Compatibility between Y and Z: first two terms
  - Sparsity: $h(Z) = \sum_i \log(1 + l^2(z_i))$
  - Regularization: $\alpha_r \|W\|_1$

Denoising Autoencoder [Vincent08]

- Denoising autoencoder

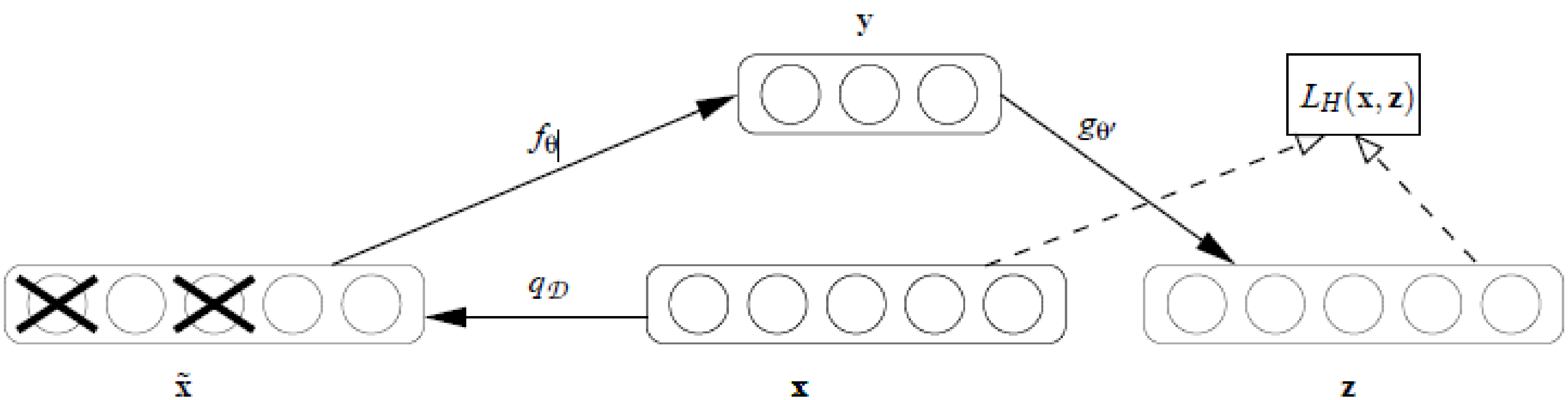


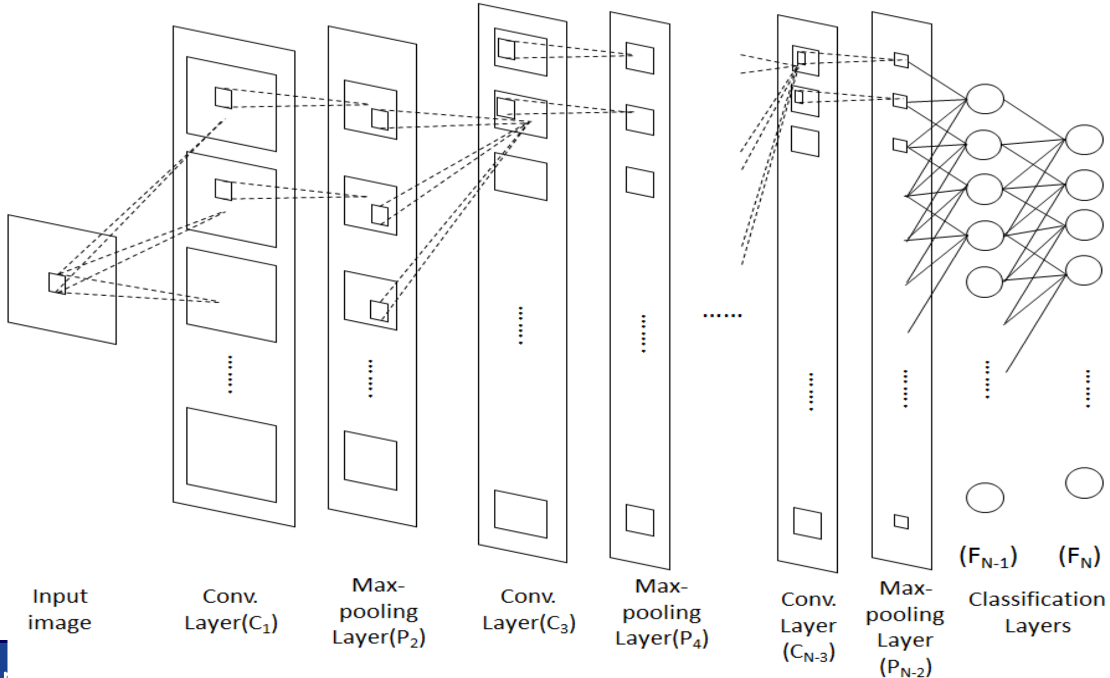
Figure 1: The denoising autoencoder architecture. An example $\mathbf{x}$ is stochastically corrupted (via $q_\mathcal{D}$) to $\tilde{\mathbf{x}}$. The autoencoder then maps it to $\mathbf{y}$ (via encoder f_θ) and attempts to reconstruct $\mathbf{x}$ via decoder $g_{\theta'}$, producing reconstruction $\mathbf{z}$. Reconstruction error is measured by loss $L_H(\mathbf{x}, \mathbf{z})$.

Agenda

- Introduction to Deep Learning
- RBM and Auto-Encoders
- <u>Convolutional Neural Networks</u>
- Recurrent Neural Networks
- Reinforcement Learning
- Deep Reinforcement Learning
- Q&A

Convolutional Neural Networks

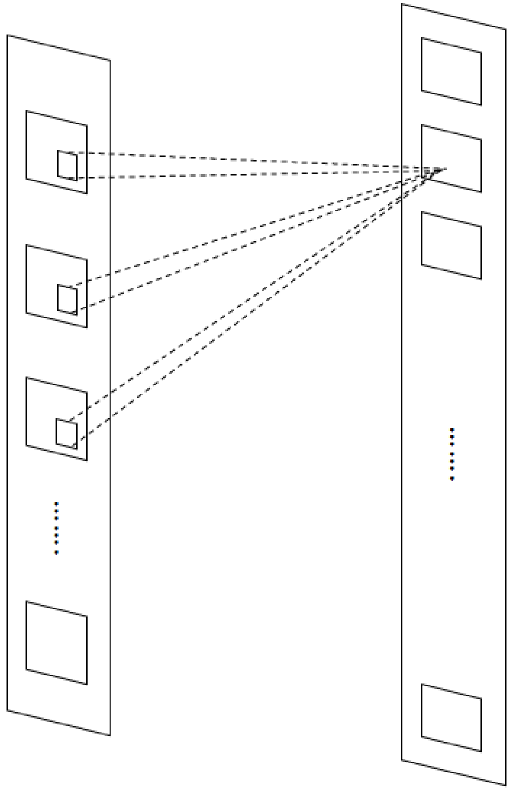
- Designed to learn position-independent local features
  - Spatially shared connections
- Combines heterogeneous layers
  - Convolution, max-pooling, fully-connected, …



Convolution Layers

- Odd-numbered layers in low/middle-level of CNN
- Nodes on each layer are grouped into 2D planes (or feature maps)
- Each plane is connected to one or more input planes
- Each node computes weighted sum of input nodes in a small region
- All nodes on a plane share weight set

➔ Extract feature by convolution operation



Convolution Layers

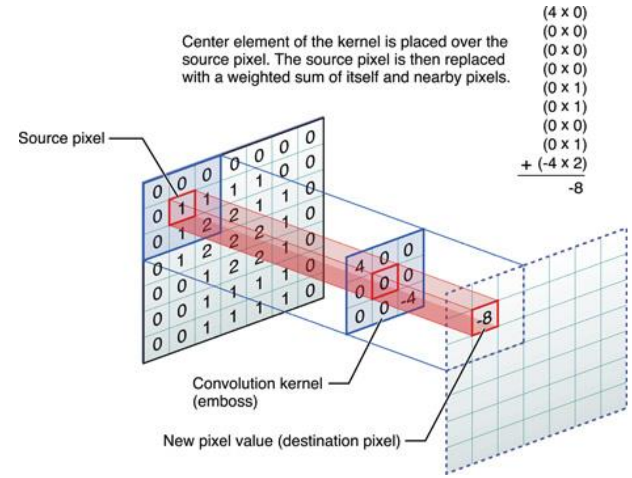
- Propagation formula

$$X^{n}_{(p,i,j)} = f\left(\sum_{q\in C^{n}_{p}} \sum_{0\le u,v\le M_{n}-1} w^{n}_{(q,p,u,v)} X^{n-1}_{(q,iS_{n}+u,jS_{n}+v)} + \theta^{n}_{p}\right)$$

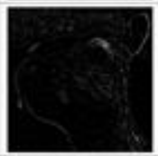
  - q: input plane, p: output plane, M_n: mask width/height
  - C^{n}_{p}: # of input planes connected to p^{th} output plane
  - $w^{n}_{(q,p,u,v)}$: weight at (u,v) on the mask from q^{th} plane to p^{th} plane
  - $X^{n}_{(p,i,j)}$: feature at (i,j) on p^{th} plane of layer n
  - S_n: stride (horizontal/vertical distance between adjacent windows)
  - θ^{n}_{p}: bias

Convolution Operation

- Convolution operation



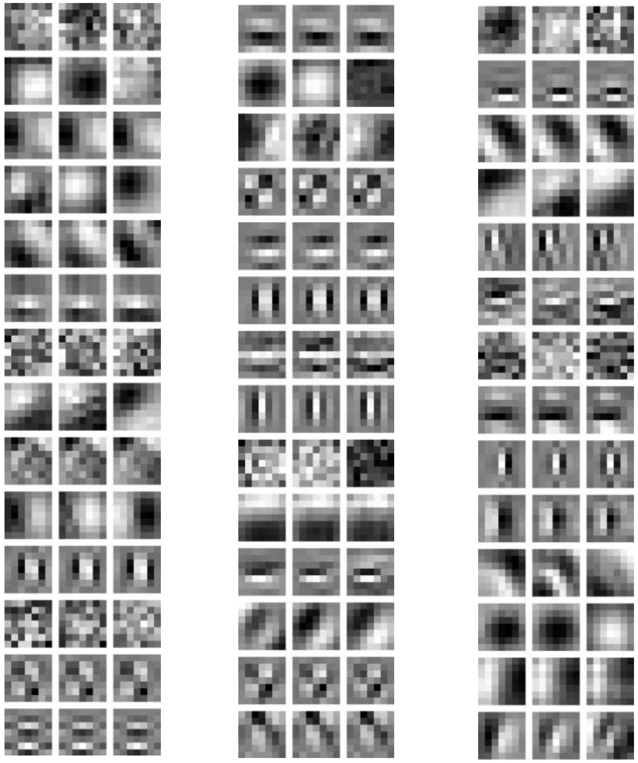
- Convolution filters

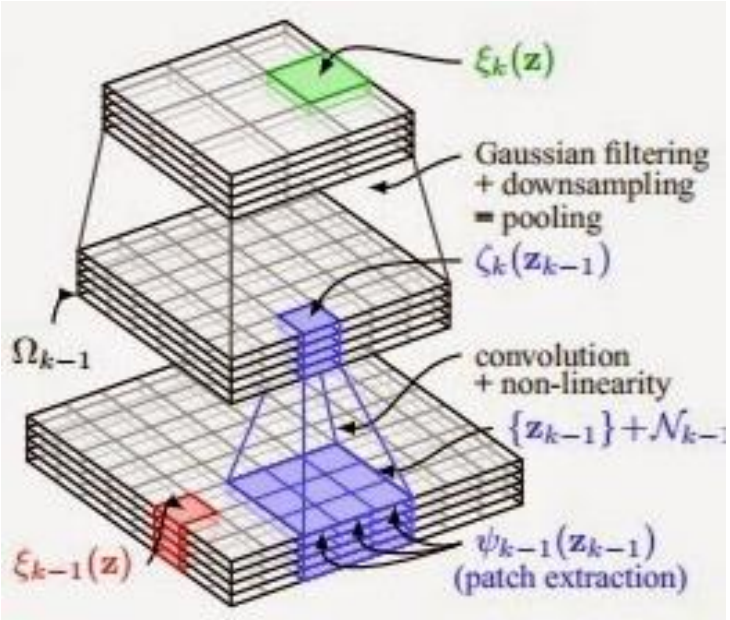
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

Sharpen	$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}$	
Box blur (normalized)	$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$	
Gaussian blur (approximation)	$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$	

Convolution Layers

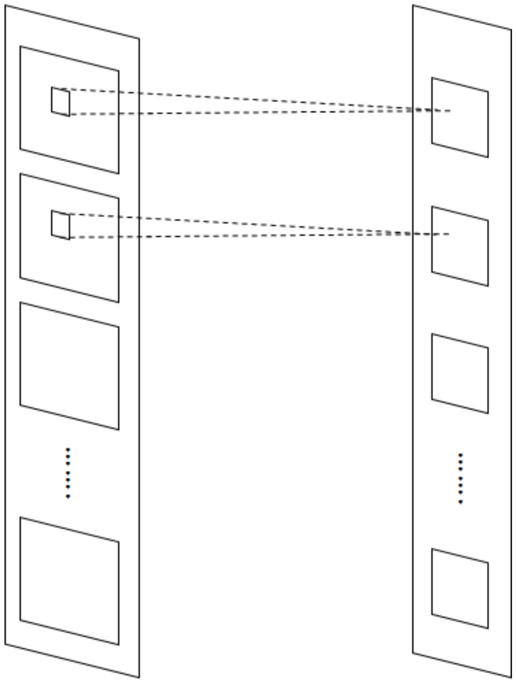
- Learning filters from data
- Multi-channel convolution





Max-Pooling Layers

- Even-numbered layers in low/middle-level of CNN
- Nodes on each layer are grouped into planes
- Each plane is connected to only one input plane
- Each node chooses maximum among input nodes in a small region



➔ Abstract features

- Reduces feature dimension
- Ignores positional variation of feature elements

Max-Pooling Layers

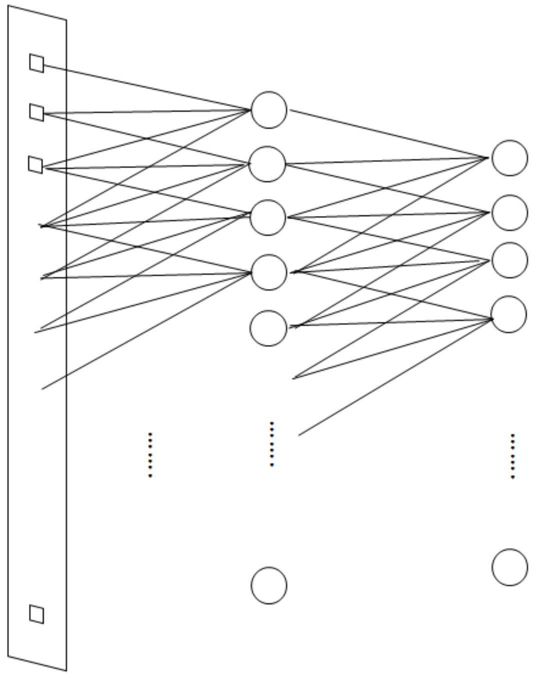
- Propagation formula

$$X_{(p,i,j)}^{n} = f\left(\max_{0\le u,v\le M_n-1} X_{(p,iS_n+u,jS_n+v)}^{n-1}\right)$$

  - p: output plane, M_n: window width/height
  - $X_{(p,i,j)}^{n}$: feature at (i,j) on p^{th} plane of layer n
  - S_n: stride (horizontal/vertical distance between adjacent windows)

Fully-connected Layers

- Top 2~3 layers of CNN
- 1D structure
- Each node is fully connected to all input nodes
- Each node computes weighted sum of all input nodes



➔ Classify input pattern with high-level features extracted by previous layers

Fully-connected Layers

- Propagation formula

$$X_p^n = f\left(\sum_q w_{(q,p)}^n X_q^{n-1} + \theta_p^n\right)$$

  - p: output node
  - X_p^n: feature at on p^{th} node of layer n
  - $w_{(q,p)}^n$: connection weight between X_q^{n-1} and X_p^n
  - θ_p^n: bias

Gradient-based Learning [LeCun98]

- Trains the whole network to minimize a single error function *E*.

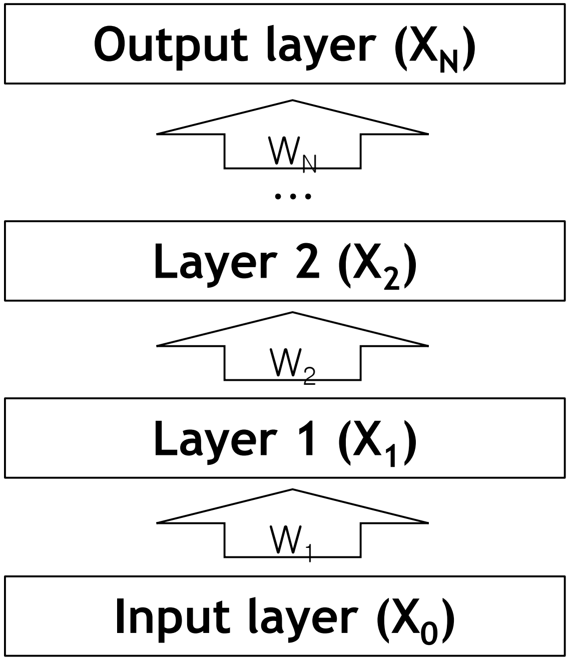
$$W \leftarrow W - \eta \frac{\partial E}{\partial W}$$

- At layer *n*

$$\frac{\partial E}{\partial W_n} = \frac{\partial X_n}{\partial W_n} \frac{\partial E}{\partial X_n} \qquad \frac{\partial E}{\partial X_{n-1}} = \frac{\partial X_n}{\partial X_{n-1}} \frac{\partial E}{\partial X_n}$$

- At layer *n−1*

$$\frac{\partial E}{\partial W_{n-1}} = \frac{\partial X_{n-1}}{\partial W_{n-1}} \frac{\partial E}{\partial X_{n-1}} \qquad \frac{\partial E}{\partial X_{n-2}} = \frac{\partial X_{n-1}}{\partial X_{n-2}} \frac{\partial E}{\partial X_{n-1}}$$



LeCun's Algorithm vs. Conventional BP

- $\frac{\partial E}{\partial W^n} = \frac{\partial E}{\partial X^n}\frac{\partial X^n}{\partial W^n} = \frac{\partial E}{\partial X^n}\frac{\partial X^n}{\partial NET^n}\frac{\partial NET^n}{\partial W^n} = \frac{\partial E}{\partial NET^n}\frac{\partial NET^n}{\partial W^n} = \Delta^n X^{n-1}$
  - $\Delta^n = \frac{\partial E}{\partial NET^n} = \frac{\partial E}{\partial X^n}\frac{\partial X^n}{\partial NET^n} = \overline{\Delta}^n F'(NET^n)$
    - Let $\overline{\Delta}^n \equiv \frac{\partial E}{\partial X^n}$
- $\frac{\partial E}{\partial X^{n-1}} = \frac{\partial E}{\partial X^n}\frac{\partial X^n}{\partial X^{n-1}} = \frac{\partial E}{\partial X^n}\frac{\partial X^n}{\partial NET^n}\frac{\partial NET^n}{\partial X^{n-1}} = \frac{\partial E}{\partial NET^n}\frac{\partial NET^n}{\partial X^{n-1}} = \Delta^n W^n$
  - $\Delta^{n-1} = \overline{\Delta}^{n-1} F'(NET^{n-1}) = \Delta^n W^n F'(NET^{n-1})$

  This is matrix notation of conventional BP formula

  - $\delta_i^{n-1} = f'(net_i^{n-1}) \sum_j W_{ij}^n \delta_j^n$

LeCun's Backpropagation Algorithm (1/2)

Given an input vector X^0, desired output D, and an error criterion

$$E_{MSE} = \frac{1}{2}\frac{\sum_c (X_c^N - d_c)^2}{C}$$

- Propagate feature from 1st to Nth layers

$$X_j^n = f\left(\sum_{i=0}^{I-1} W_{ij}^n X_i^{n-1} + \theta_j\right)$$

- Compute $\overline{\Delta}^N = \frac{\partial E}{\partial X^N}$ of the output layer

$$\bar{\delta}_c^N = \frac{\partial E}{\partial X_c^N} = \frac{(X_c^N - d_c)}{C}$$

LeCun's Backpropagation Algorithm (2/2)

- Repeat for each n=N, N−1, ⋯, 1
  - Compute $\delta_j^n = \bar{\delta}_j^n f'(net_j^n)$
  - Compute gradient
    $$\frac{\partial E}{\partial W_{ij}^n} = \delta_j^n X_i^{n-1} = \bar{\delta}_j^n f'(net_j^n)\, X_i^{n-1}$$
  - Update weights (can be delayed in batch mode training)
    $$W_{ij}^n \leftarrow W_{ij}^n - \eta \frac{\partial E}{\partial W_{ij}^n}$$
  - Compute $\bar{\Delta}^{n-1} = \frac{\partial E}{\partial X^{n-1}}$ for the preceding layer
    $$\bar{\delta}_i^{n-1} = \sum_j W_{ij}^n \delta_j^n$$

Agenda

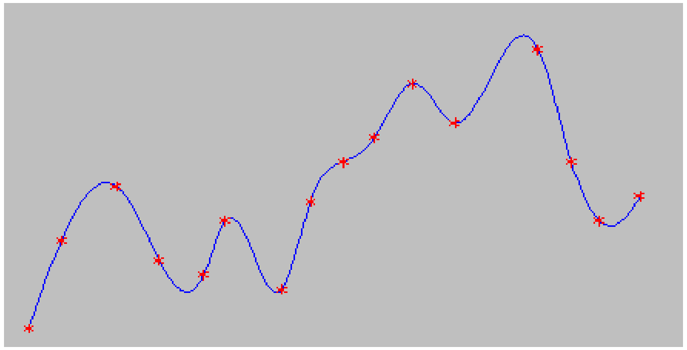
- Introduction to Deep Learning
- RBM and Auto-Encoders
- Convolutional Neural Networks
- <u>Recurrent Neural Networks</u>
- Reinforcement Learning
- Deep Reinforcement Learning
- Q&A

References

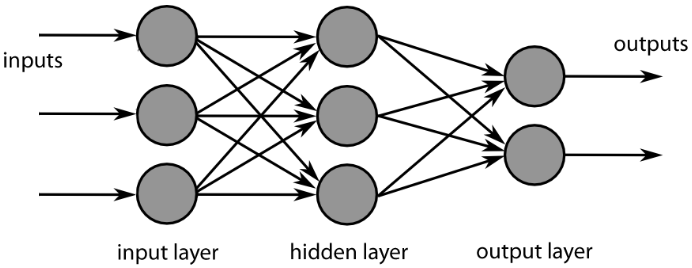
- RNN tutorial
  - http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/
- LSTM tutorial
  - http://colah.github.io/posts/2015-08-Understanding-LSTMs

Recurrent Neural Networks

- Motivation: analyzing time series data
  - Many real world data are dependent on the previous or next data.

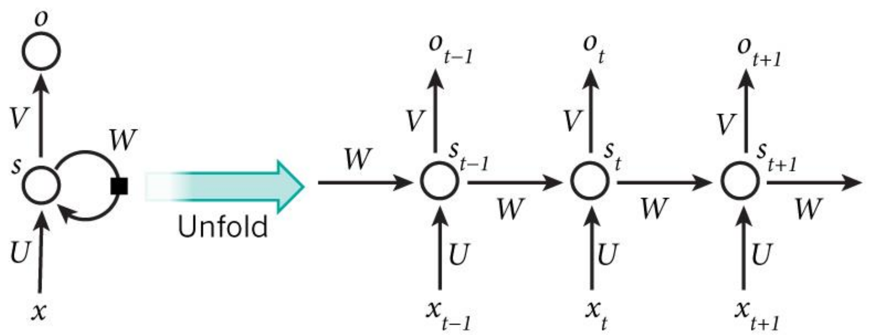


  - Feed forward networks assumes all inputs are independent from each other



Recurrent Neural Networks

- Recurrent neural network
  - Neural networks with recurrent connection
  - State of nodes affect the output and the next state
  - Model for dynamic process
  - Temporarily shared connections

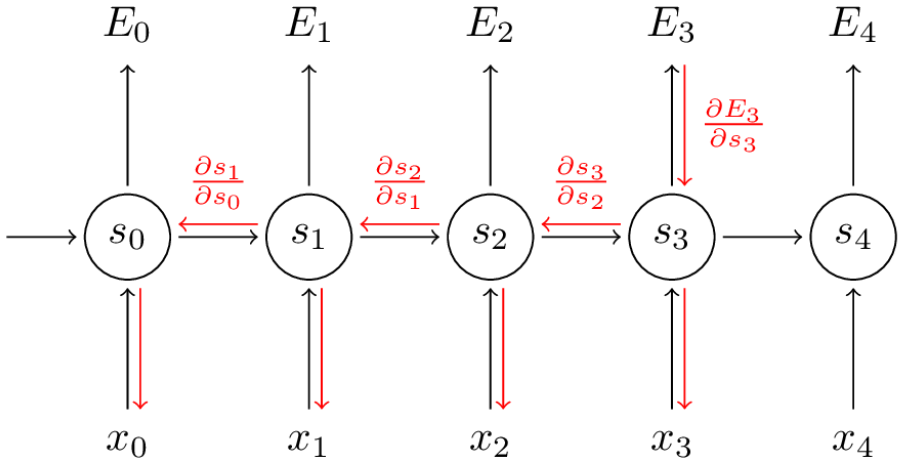


- Currently, the most promising architecture for NLP, speech recognition, handwriting recognition, automatic image captioning

Training RNN

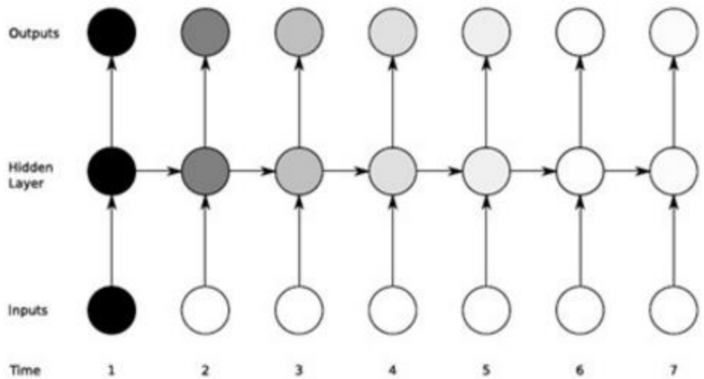
- BPTT (back-propagation through time)
  - Back-propagation algorithm applied to unfolded RNN
  - For each training sample, sum up the gradient at each time step

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

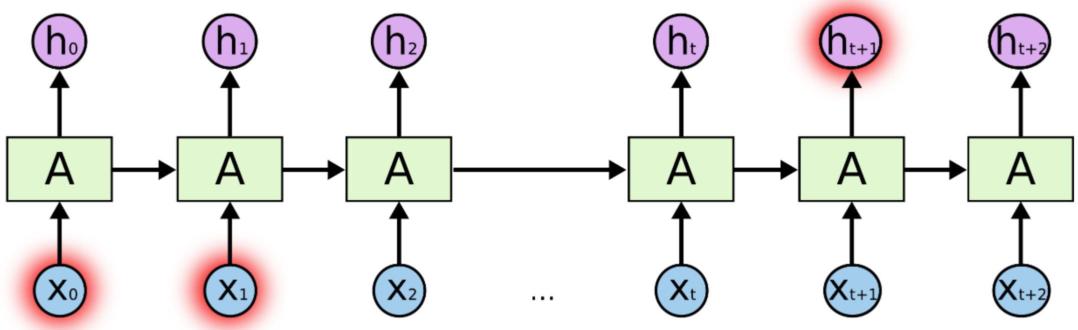


Problems in RNN

- Vanishing gradient problem

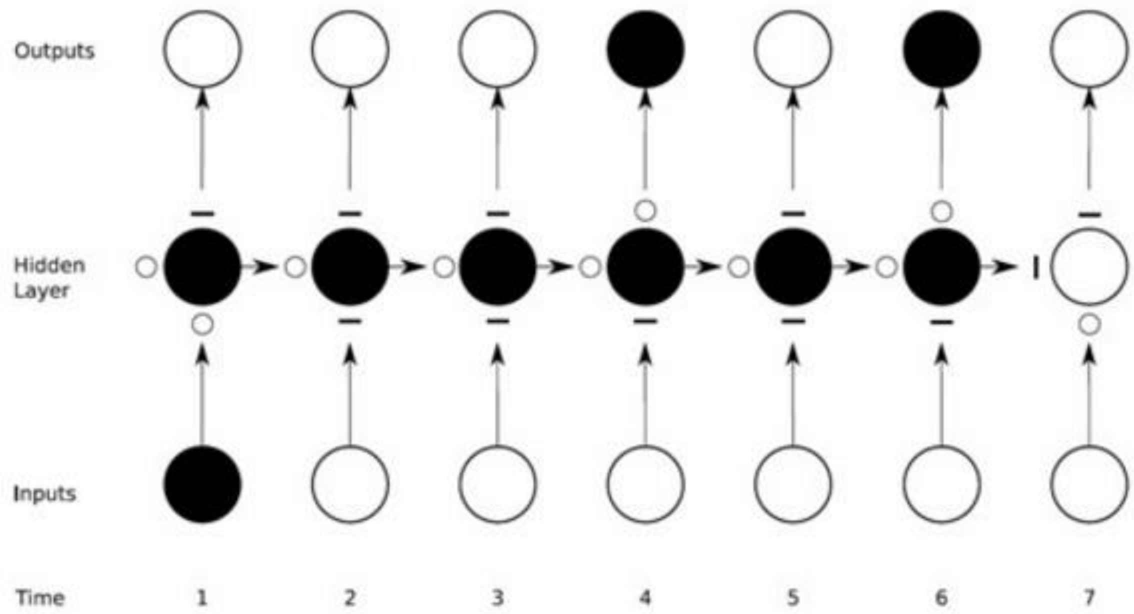


- Learning long-term dependency



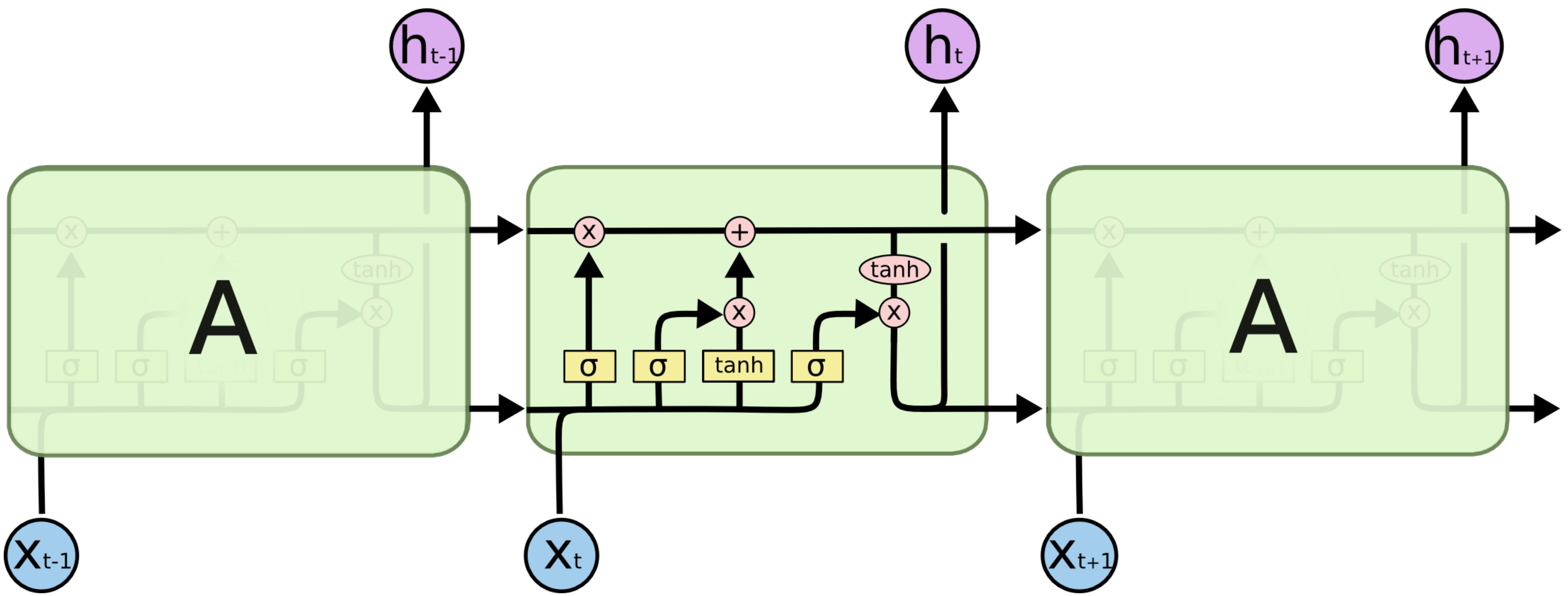
LSTM Networks

- LSTM: Long Short-Term Memory
  - Designed to learning long-term dependency
  - RNN with explicit gate to control data flow
    - Input/output/forget gates



LSTM Networks

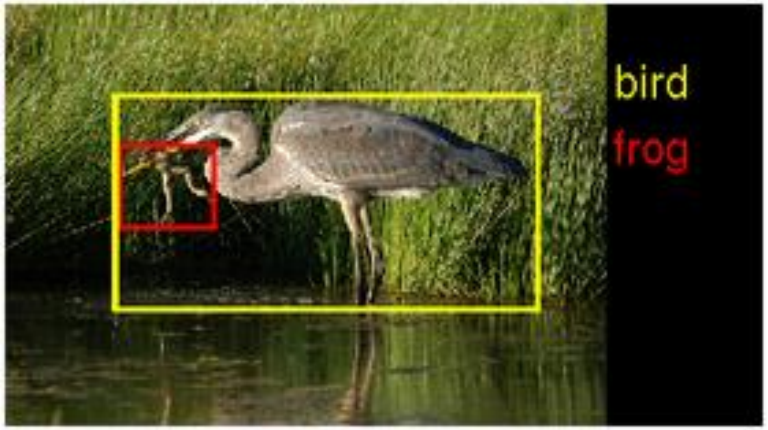
- Structure of LSTM networks
  - σ: gate networks

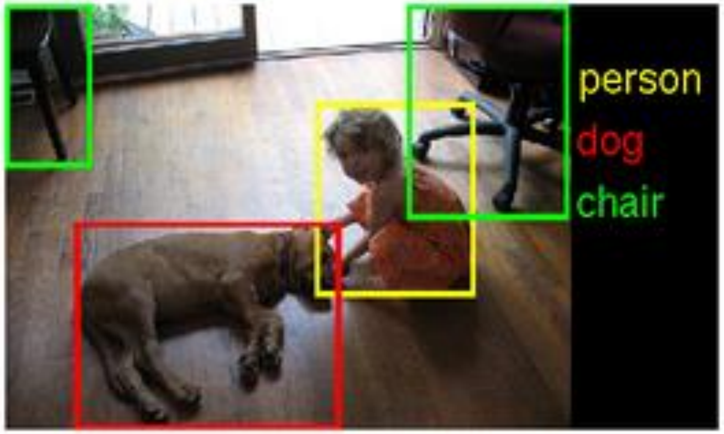


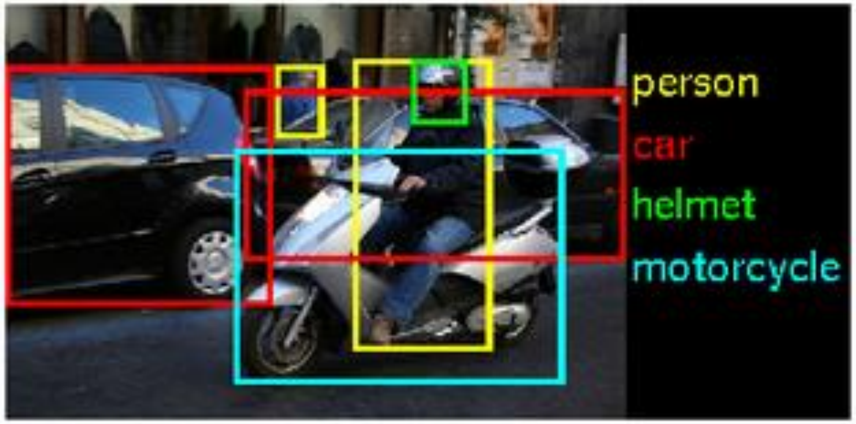
Object Image Recognition

- ImageNet Large Scale Visual Recognition Challenge (http://www.image-net.org)
  - 1000 object categories
  - Training set: 1,281,167 images
  - Validation set: 50,000 images
  - Test set: 100,000 images

Examples of ImageNet Images







ILSVRC2012 Results

- CNN defeated other systems by large margin in ILSVRC2012

CNN

Team name	Filename	Error (5 guesses)	Description
SuperVision	test-preds-141-146.2009-131-137-145-146.2011-145f.	0.15315	Using extra training data from ImageNet Fall 2011 release
SuperVision	test-preds-131-137-145-135-145f.txt	0.16422	Using only supplied training data
ISI	pred_FVs_wLACs_weighted.txt	0.26172	Weighted sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and CSIFT+FV, respectively.
ISI	pred_FVs_weighted.txt	0.26602	Weighted sum of scores from classifiers using each FV.
ISI	pred_FVs_summed.txt	0.26646	Naive sum of scores from classifiers using each FV.

ILSVRC2013 Results

ImageNet Large Scale Vis

www.image-net.org/challenges/LSVRC/2013/results.php

UIUC-IFP	Convnet for object detection.	0.010489	--	0

Task 2: Classification

Legend:
Dark grey background = outside training data

All high rankers use CNNs

Team name	Comment	Error
Clarifai	Multiple models trained on the original data plus an additional model trained on 5000 categories.	0.11197
Clarifai	Multiple models trained on the original data plus an additional model trained on other 1000 category data.	0.11537
Clarifai	Average of multiple models on original training data.	0.11743
Clarifai	Another attempt at multiple models on original training data.	0.1215
Clarifai	Single model trained on original data.	0.12535
NUS	adaptive non-parametric rectification of all outputs from CNNs and refined PASCAL VOC12 winning solution, with further retraining on the validation set.	0.12953
NUS	adaptive non-parametric rectification of all outputs from CNNs and refined PASCAL VOC12 winning solution.	0.13303
ZF	5 models (4 different architectrues) trained on original data.	0.13511
Andrew Howard	This is an ensemble of convolutional neural networks combining multiple transformations for training and testing and models operating at different resolutions.	0.13555
Andrew Howard	This method explores re weighting the predictions from different data transformation and ensemble members in the previous submission.	0.13564
ZF	5 models trained on original data, 1 big.	0.13748

Face Recognition

- Taigman, et al, “DeepFace: Closing the Gap to Human-Level Performance in Face Verification”, 2014
  - 97.25% on LFW (Labeled Faces in the Wild)
- Fan, et al, “Learning Deep Face Representation”, 2014
  - 97.30% on LFW
- Sun, et al, “Deep Learning Face Representation from Predicting 10,000 Classes”, 2014
  - 99.15 on LFW
- Shroff, et al, “FaceNet: A Unified Embedding for Face Recognition and Clustering”
  - 99.63% on LFW
  - 95.12% on YouTube Face DB

DeepFace [Taigman2014]

- Feature extraction by CNN
  - Train a CNN-based face recognizer
  - Represent the input face image by the output of $(N-1)^{th}$ layer

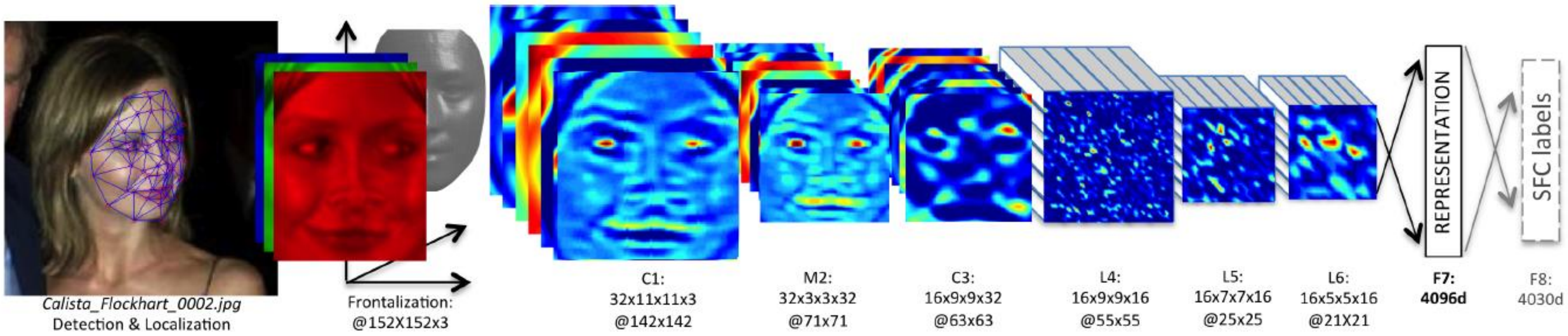


Figure 2. **Outline of the *DeepFace* architecture**. A front-end of a single convolution-pooling-convolution filtering on the rectified input, followed by three locally-connected layers and two fully-connected layers. Colors illustrate outputs for each layer. The net includes more than 120 million parameters, where more than 95% come from the local and fully connected layers.

Deep Learning in Speech Recognition

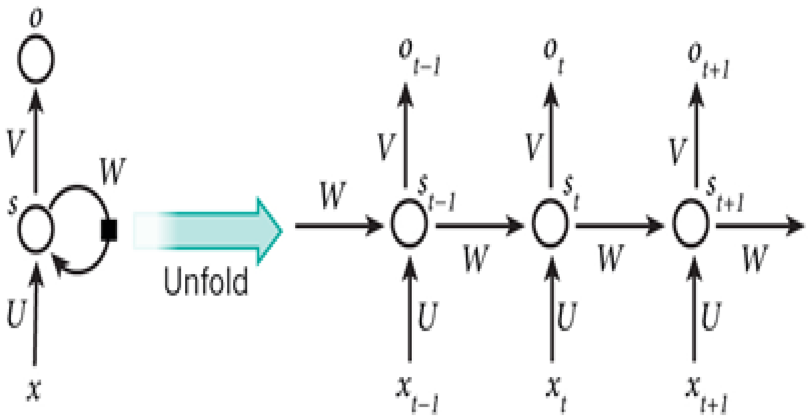
- Deep Neural Networks for Acoustic Modeling in Speech Recognition [Hinton2012]

[TABLE 1] COMPARISONS AMONG THE REPORTED SPEAKER-INDEPENDENT (SI) PHONETIC RECOGNITION ACCURACY RESULTS ON TIMIT CORE TEST SET WITH 192 SENTENCES.

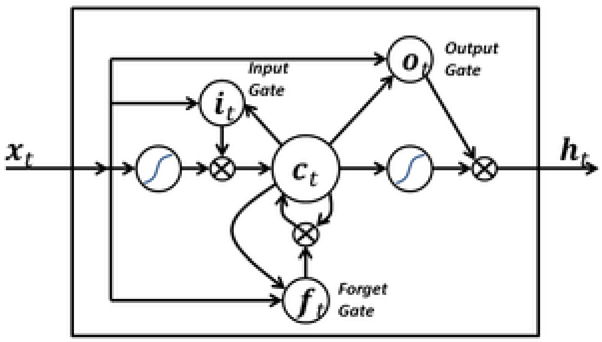
	METHOD	PER
	CD-HMM [26]	27.3%
	AUGMENTED CONDITIONAL RANDOM FIELDS [26]	26.6%
	RANDOMLY INITIALIZED RECURRENT NEURAL NETS [27]	26.1%
	BAYESIAN TRIPHONE GMM-HMM [28]	25.6%
	MONOPHONE HTMS [29]	24.8%
	HETEROGENEOUS CLASSIFIERS [30]	24.4%
Deep learning	MONOPHONE RANDOMLY INITIALIZED DNNs (SIX LAYERS) [13]	23.4%
	MONOPHONE DBN-DNNs (SIX LAYERS) [13]	22.4%
	MONOPHONE DBN-DNNs WITH MMI TRAINING [31]	22.1%
	TRIPHONE GMM-HMMs DT W/ BMMI [32]	21.7%
	MONOPHONE DBN-DNNs ON FBANK (EIGHT LAYERS) [13]	20.7%
	MONOPHONE MCRBM-DBN-DNNs ON FBANK (FIVE LAYERS) [33]	20.5%

Deep Learning in Speech Recognition

- LSTM (long short-term memory)
  - Recurrent neural network (RNN) architecture
  - Achieved 17.7% on TIMIT dataset



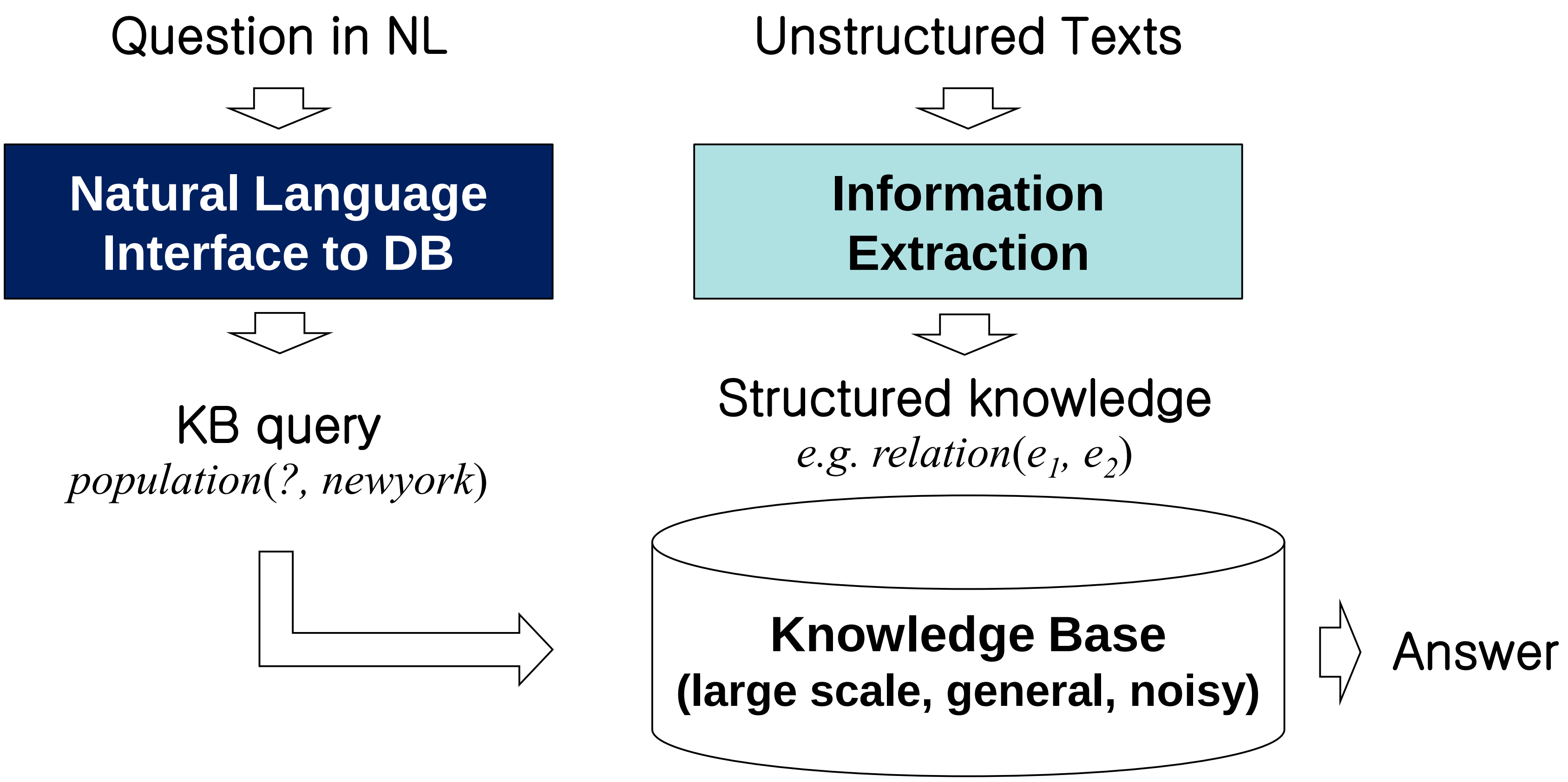
Recurrent neural networks



LSTM

Natural Language Processing

- Open-domain question answering



Semantic Parsing using CNN [Yih2014]

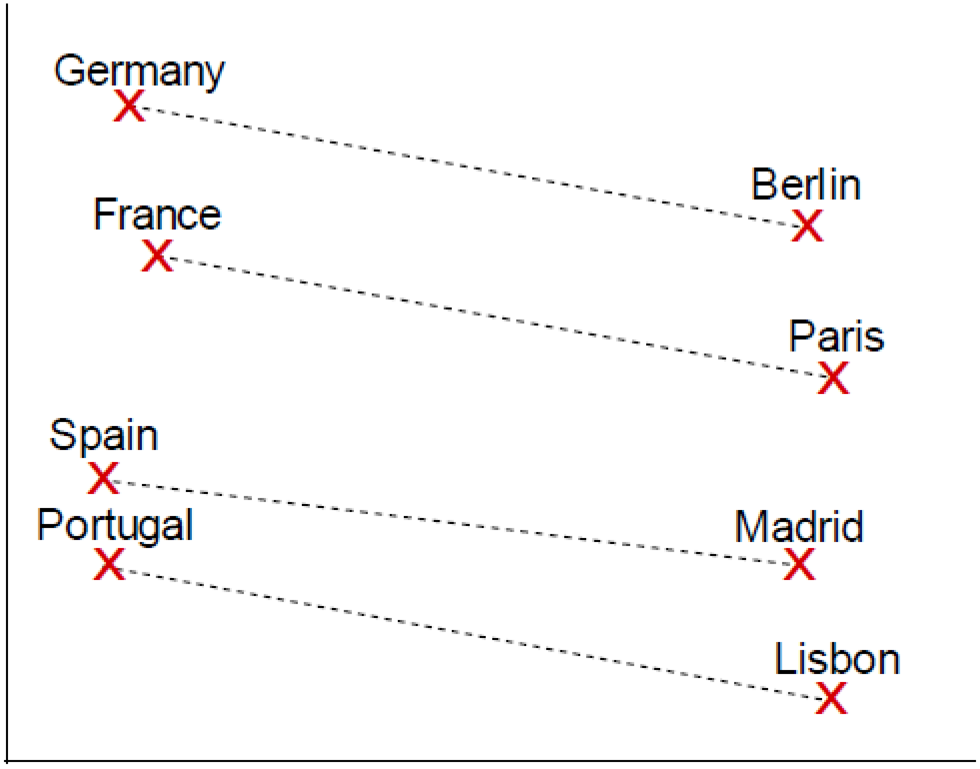
- Semantic parsing
  - Entity mention <-> KB entity
  - Relation pattern <-> KB relation

- CNN-based semantic similarity model (CNNSM)
  - Maps variable-length word sequence to low-dimensional vector
  - Compares word sequences by cosine distance.

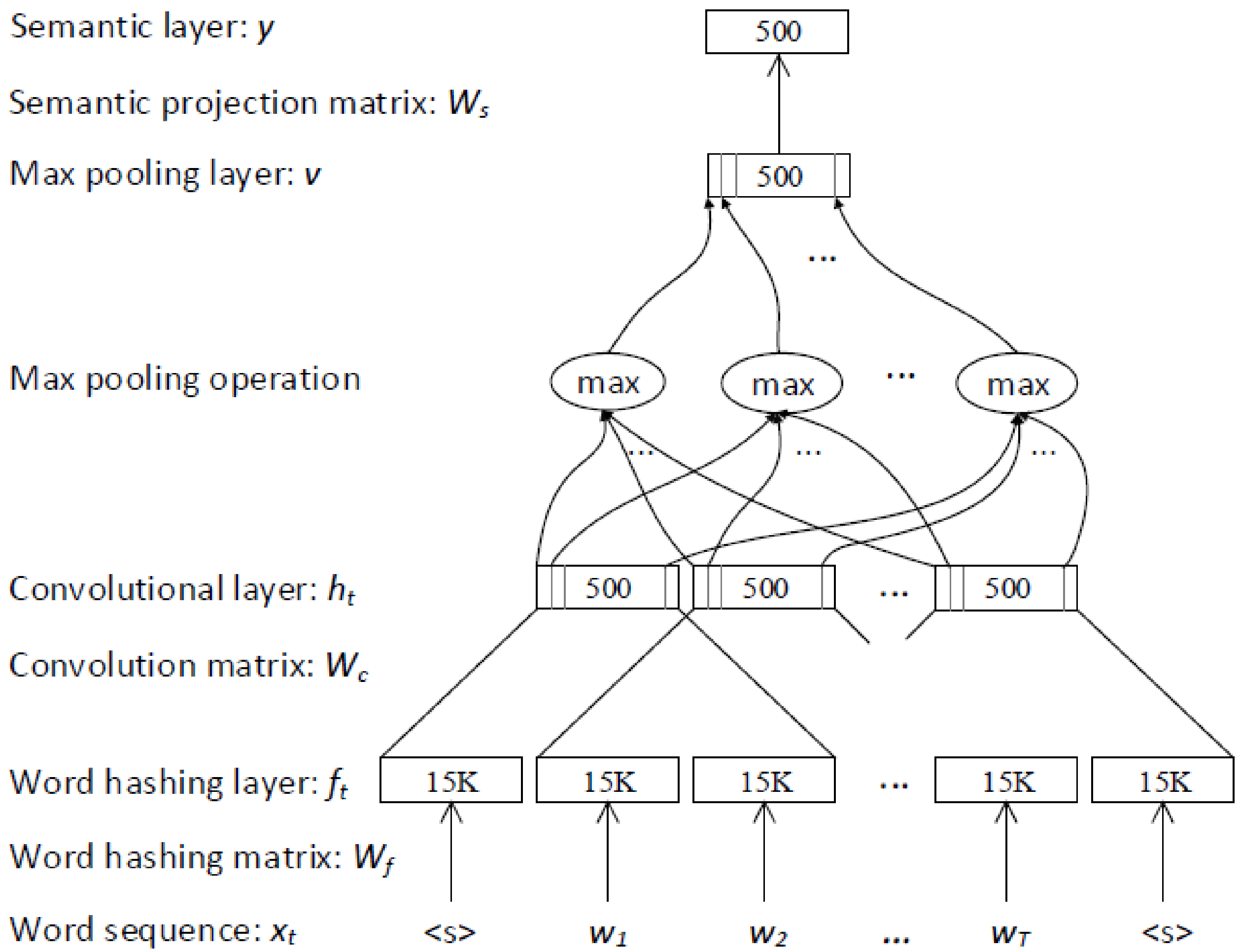
Word Embedding

- Embedded vector space

  Ex) ‘Paris – France + Berlin’ provide a vector near ‘Germany’



CNN-based Semantic Model [Yih2014]



count vector of
letter-trigram

Agenda

- Introduction to Deep Learning
- RBM and Auto-Encoders
- Convolutional Neural Networks
- Recurrent Neural Networks
- <u>Reinforcement Learning</u>
- Deep Reinforcement Learning
- Q&A

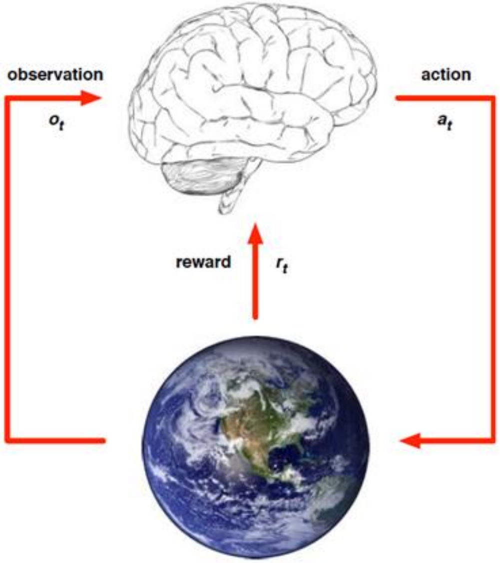
References

- Deep Reinforcement Learning [Silver16]
- Human-level control through deep reinforcement learning [Mnih15]
- Mastering the game of Go with deep neural networks and tree search [Silver16]
- Introduction to Reinforcement Learning [K.Kim13]

Reinforcement Learning

RL is a general-purpose framework for decision-making

- RL is for an agent with the capacity to act
- Each action influences the agent's future state
- Success is measured by a scalar reward signal
- Goal: select actions to maximise future reward

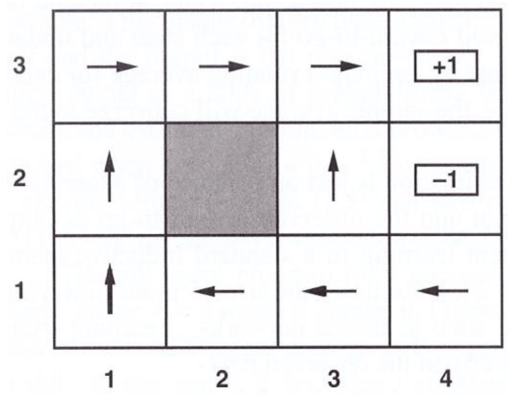


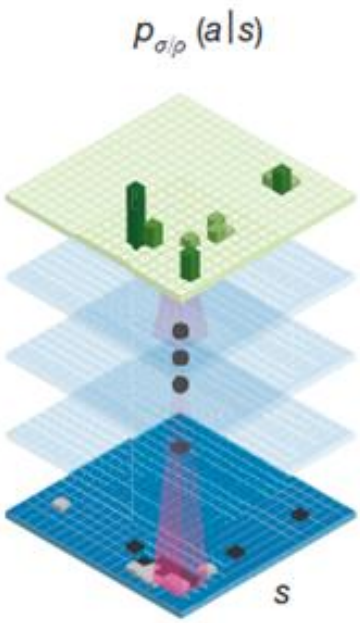
Major Components of RL Agents

- An RL agent may include one or more of these components
  - Policy: agent's behavior function for each state
  - Value function: how good is each state and/or action
  - Model: agent's representation of environment

Policy

- A policy is the agent's behaviour
- It is a map from state to action:
  - Deterministic policy: $a = \pi(s)$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}[a|s]$





Value Function

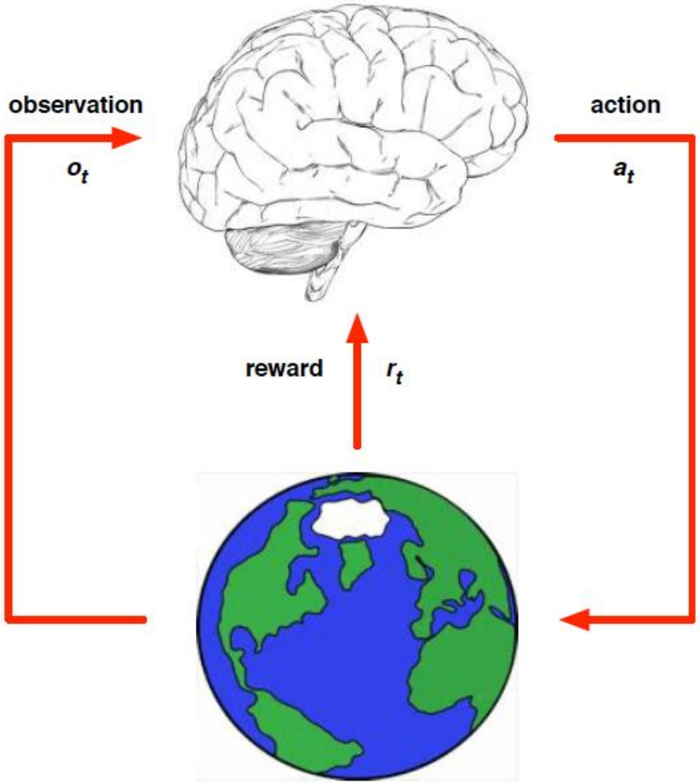
- A value function is a prediction of future reward
  - "How much reward will I get from action a in state s?"
- Q-value function gives expected total reward
  - from state s and action a
  - under policy π
  - with discount factor γ

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s, a\right]$$

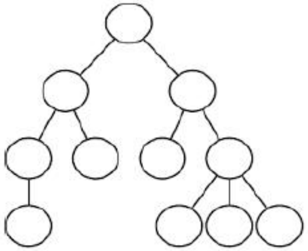
- Value functions decompose into a Bellman equation

$$Q^{\pi}(s,a) = \mathbb{E}_{s',a'}\left[r + \gamma Q^{\pi}(s',a') \mid s, a\right]$$

Model



- Model is learnt from experience
- Acts as proxy for environment
- Planner interacts with model
- e.g. using lookahead search



Approaches to Reinforcement Learning

Value-based RL

- Estimate the optimal value function $Q^*(s, a)$
- This is the maximum value achievable under any policy

Policy-based RL

- Search directly for the optimal policy π^*
- This is the policy achieving maximum future reward

Model-based RL

- Build a model of the environment
- Plan (e.g. by lookahead) using model

Agenda

- Introduction to Deep Learning
- RBM and Auto-Encoders
- Convolutional Neural Networks
- Recurrent Neural Networks
- Reinforcement Learning
- <u>Deep Reinforcement Learning</u>
- Q&A

Deep Reinforcement Learning

We seek a single agent which can solve any human-level task

- RL defines the objective
- DL gives the mechanism
- RL + DL = general intelligence

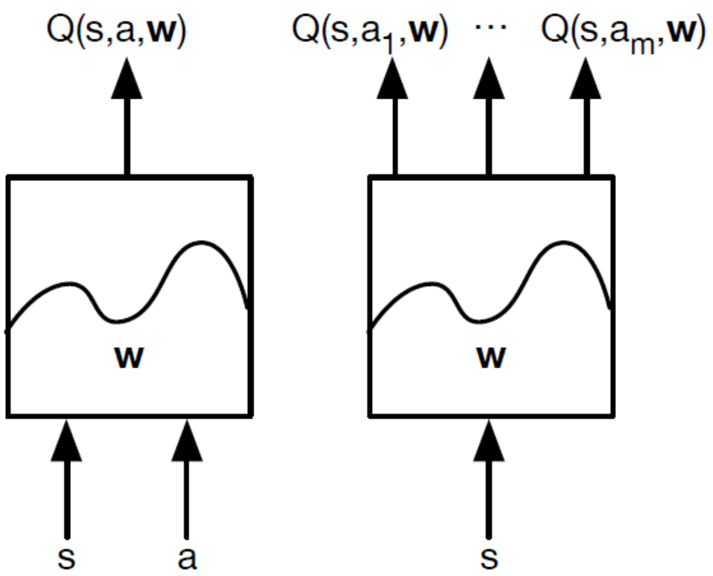
Deep Reinforcement Learning

- Use deep neural networks to represent
  - Value function
  - Policy
  - Model
- Optimise loss function by stochastic gradient descent

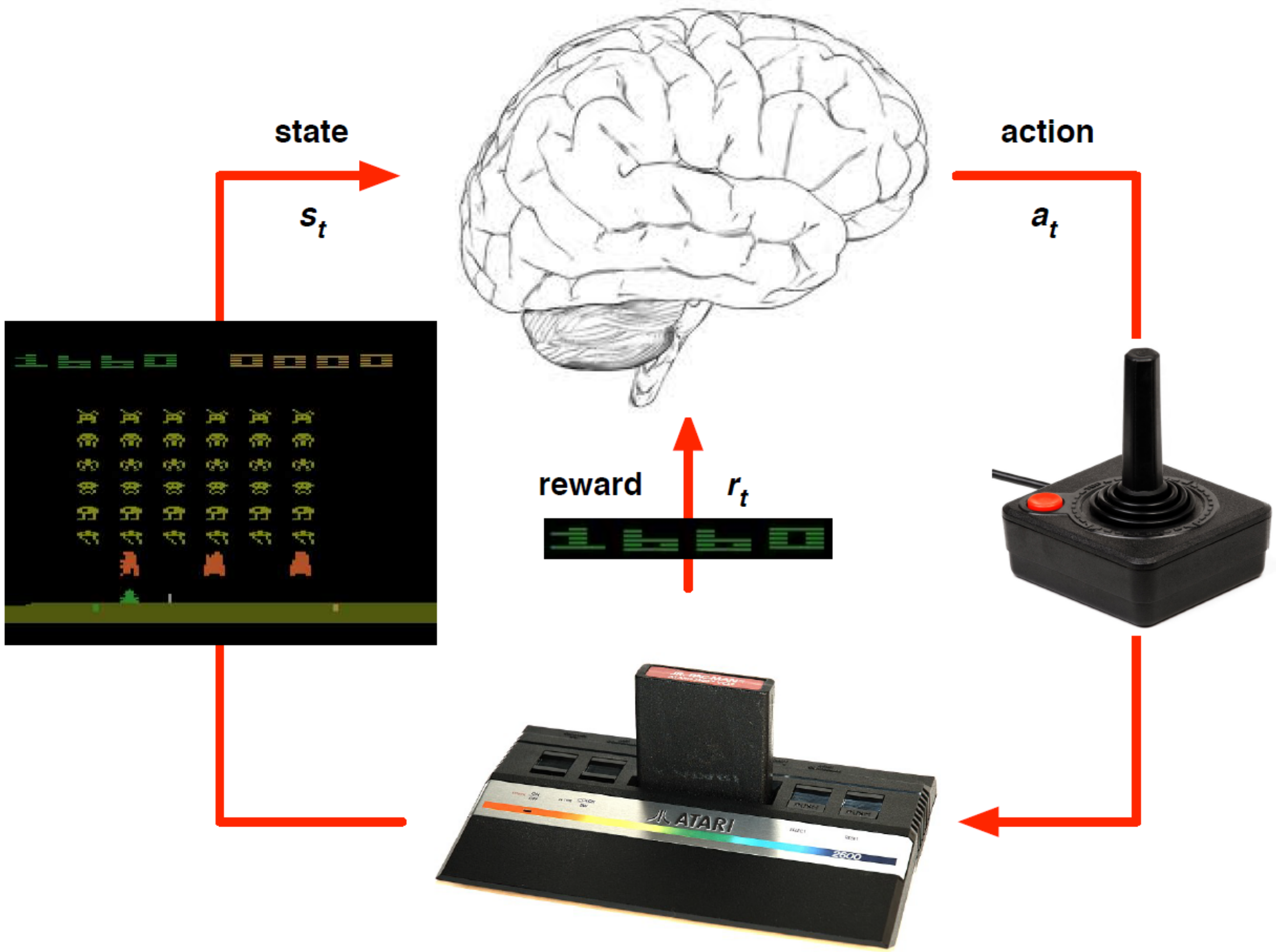
Q–Networks

Represent value function by Q-network with weights **w**

$$Q(s, a, \mathbf{w}) \approx Q^*(s, a)$$

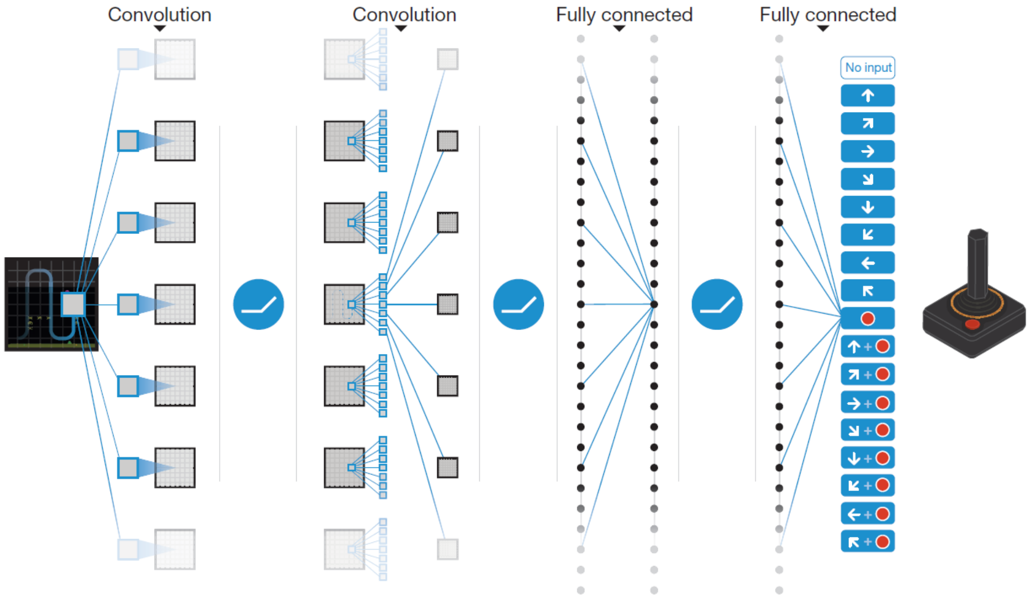


Deep Reinforcement Learning in Atari



DQN in Atari

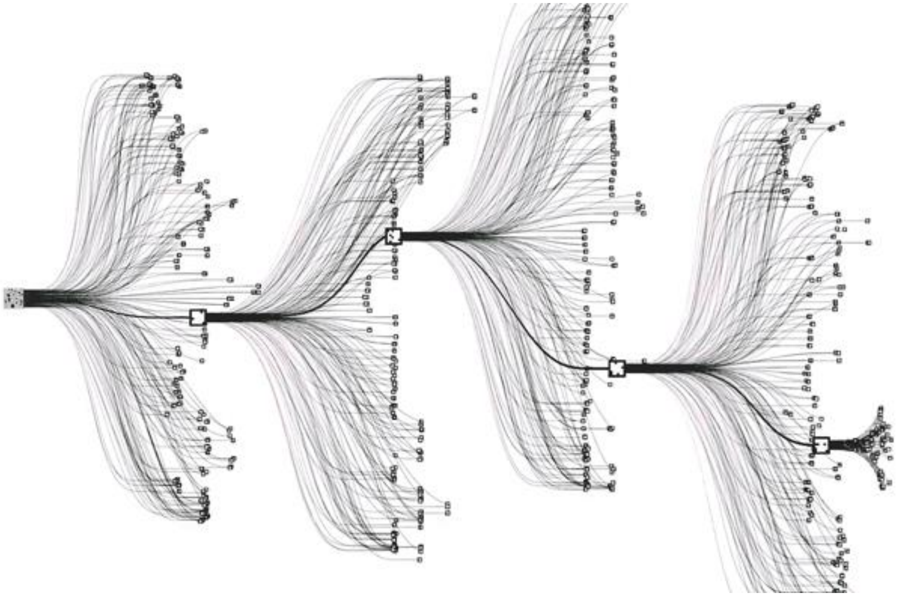
- End-to-end learning of values $Q(s, a)$ from pixels s
- Input state s is stack of raw pixels from last 4 frames
- Output is $Q(s, a)$ for 18 joystick/button positions
- Reward is change in score for that step



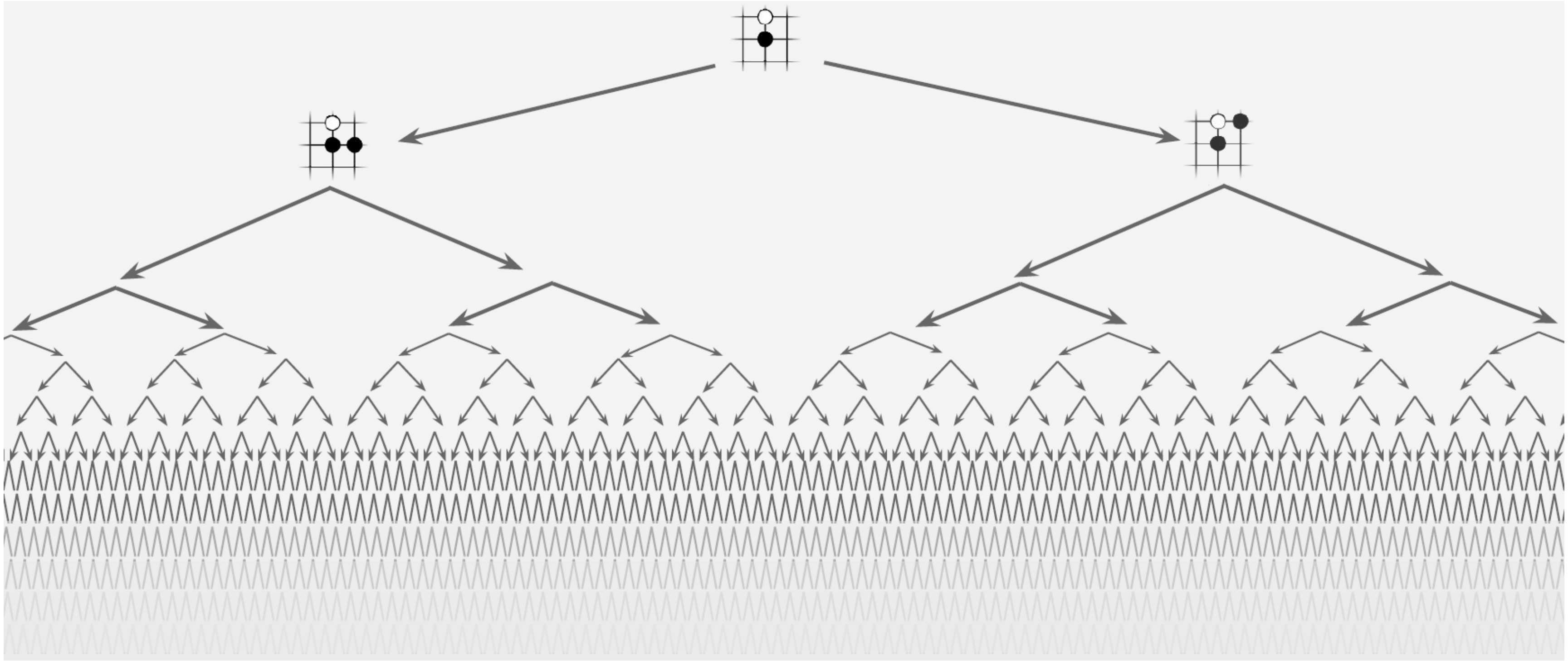
Network architecture and hyperparameters fixed across all games

AlphaGo

- Conventional tree search
  - Infeasible for huge search space
- Monte Carlo Tree Search (MCTS)
  - Random sampling
  - Policy network, value networks guide random sampling

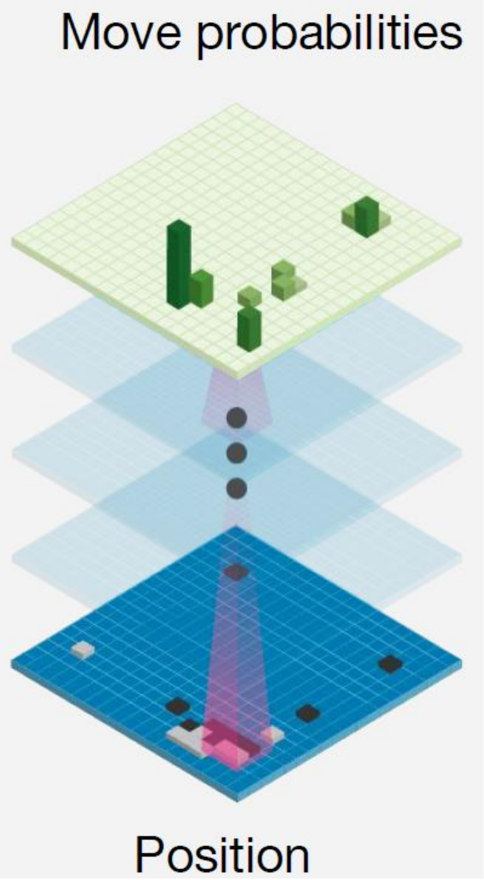


Exhaustive Search

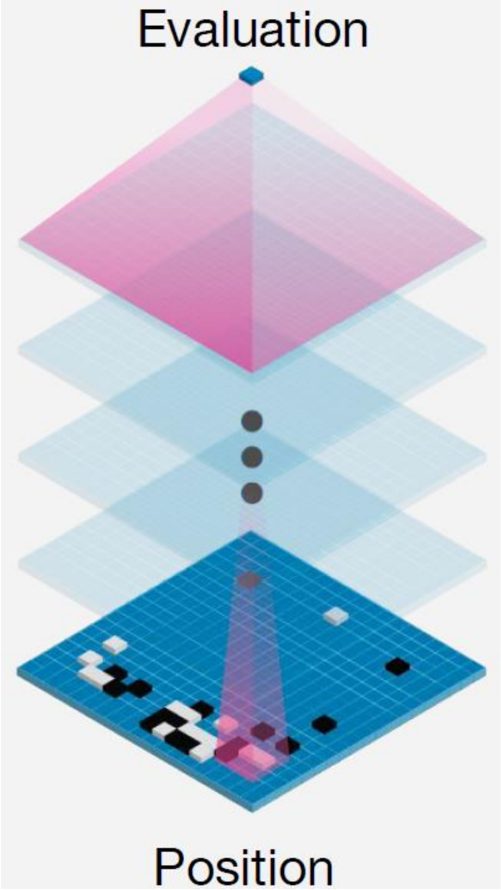


CNNs in AlphaGo

- Policy network

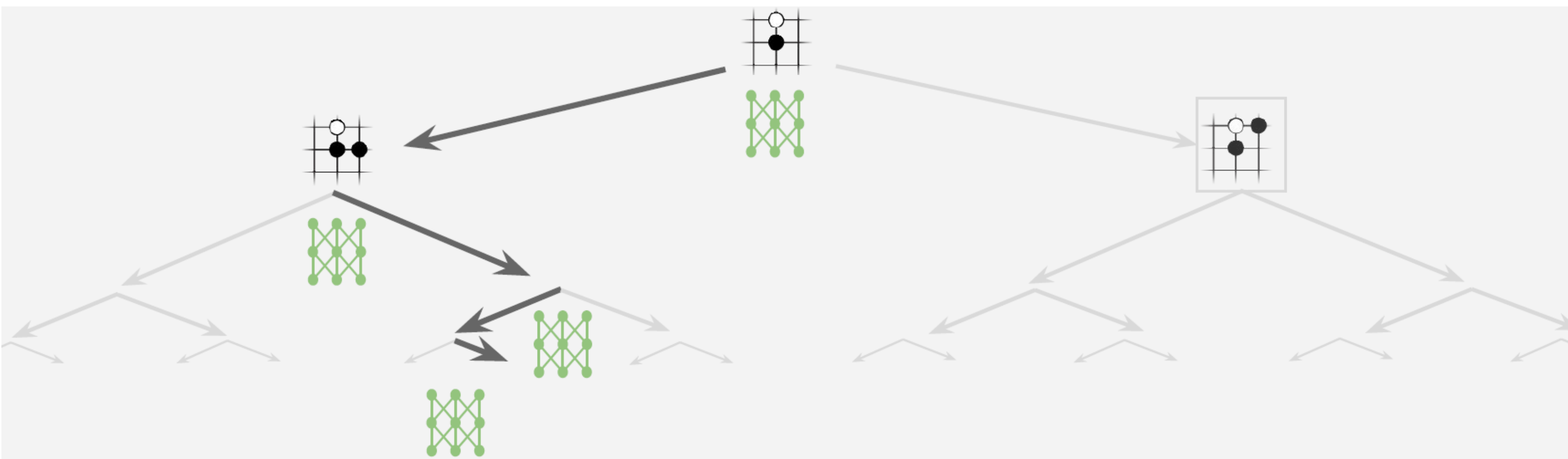


- Value network



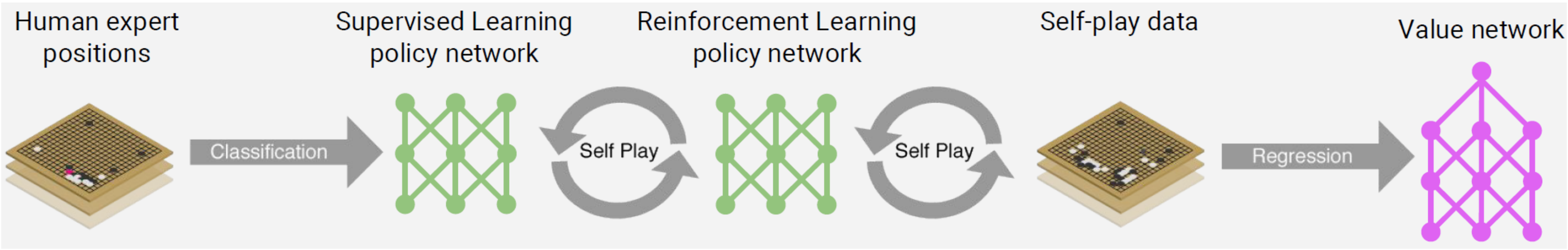
Reducing Search Space

- Policy network reduces breath
- Value network reduces depth



Neural Network Training Pipeline

- Supervised learning of policy network
- Reinforcement learning of policy network
- Reinforcement learning of value network



Supervised Learning of Policy Network

- **Policy network:** 12 layer convolutional neural network
- **Training data:** 30M positions from human expert games (KGS 5+ dan)
- **Training algorithm:** maximize likelihood by stochastic gradient descent

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma}$$

- **Training time:** 4 weeks on 50 GPUs using Google Cloud
- **Results:** 57% accuracy on held out test data (state-of-the art was 44%)

Reinforcement Learning of Policy Network

- **Policy network:** 12 layer convolutional neural network
- **Training data:** games of self-play between policy network
- **Training algorithm:** maximize wins *z* (1 or −1)by policy gradient reinforcement learning

$$\Delta\sigma \propto \frac{\partial \log p_\sigma(a|s)}{\partial \sigma} z$$

- **Training time:** 1 week on 50 GPUs using Google Cloud
- **Results:** 80% vs supervised learning. Raw network ~3 amateur dan

Reinforcement Learning of Value Network

- **Value network:** 12 layer convolutional neural network
- **Training data:** 30 million games of self-play
- **Training algorithm:** minimize MSE by stochastic gradient descent

$$\Delta\theta \propto \frac{\partial v_\theta(s)}{\partial \theta}(z - v_\theta(s))$$

- **Training time:** 1 week on 50 GPUs using Google Cloud
- **Results:** First strong position evaluation function – previously thought impossible