香港科技大學
THE HONG KONG
UNIVERSITY OF SCIENCE
AND TECHNOLOGY

Introduction to Deep Learning

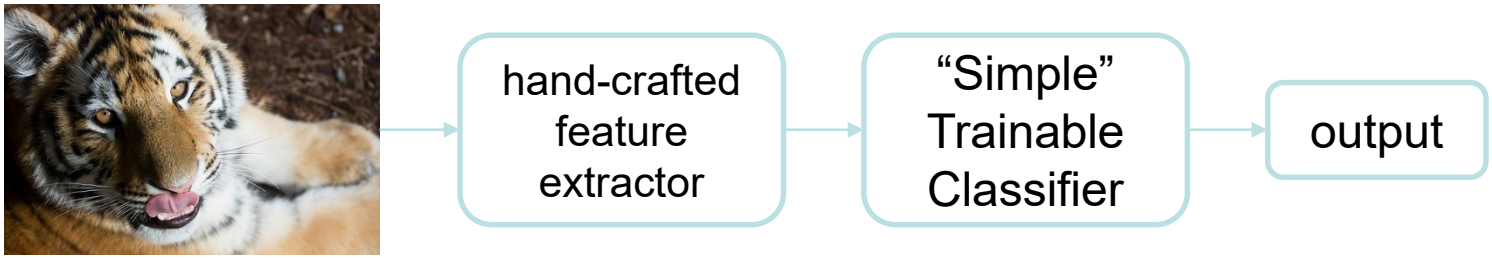
Professor Qiang Yang

Outline

- Introduction
- Supervised Learning
  - Convolutional Neural Network
  - Sequence Modelling: RNN and its extensions
- Unsupervised Learning
  - Autoencoder
  - Stacked Denoising Autoencoder
- Reinforcement Learning
  - Deep Reinforcement Learning
  - Two applications: Playing Atari & AlphaGo

Introduction

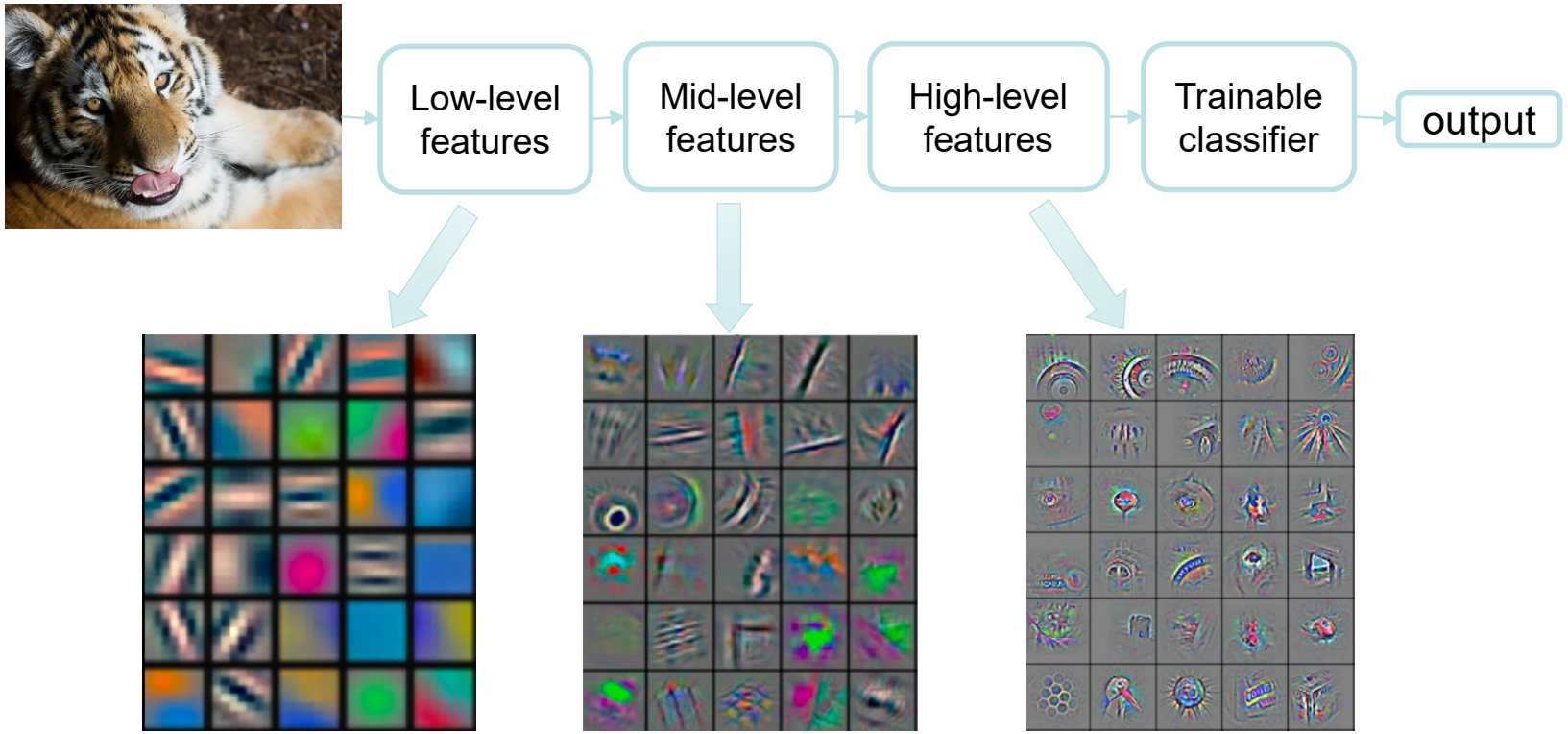
- Traditional pattern recognition models use hand-crafted features and relatively simple trainable



- This approach has the following limitations:
  - It is very tedious and costly to develop hand-crafted features
  - The hand-crafted features are usually highly dependents on one application, and cannot be transferred easily to other applications

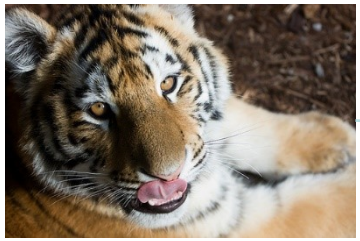
Deep Learning

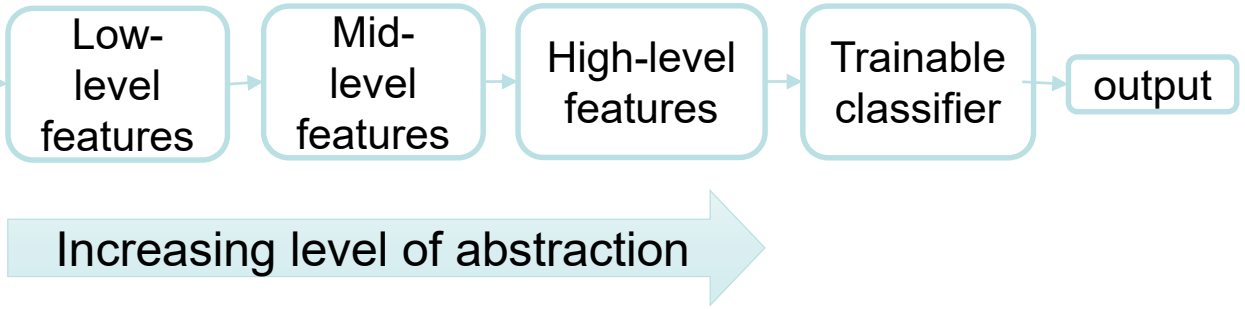
- Deep learning (a.k.a. representation learning) seeks to learn rich hierarchical representations (i.e. features) automatically through multiple stage of feature learning process.



Feature visualization of convolutional net trained on ImageNet (Zeiler and Fergus, 2013)

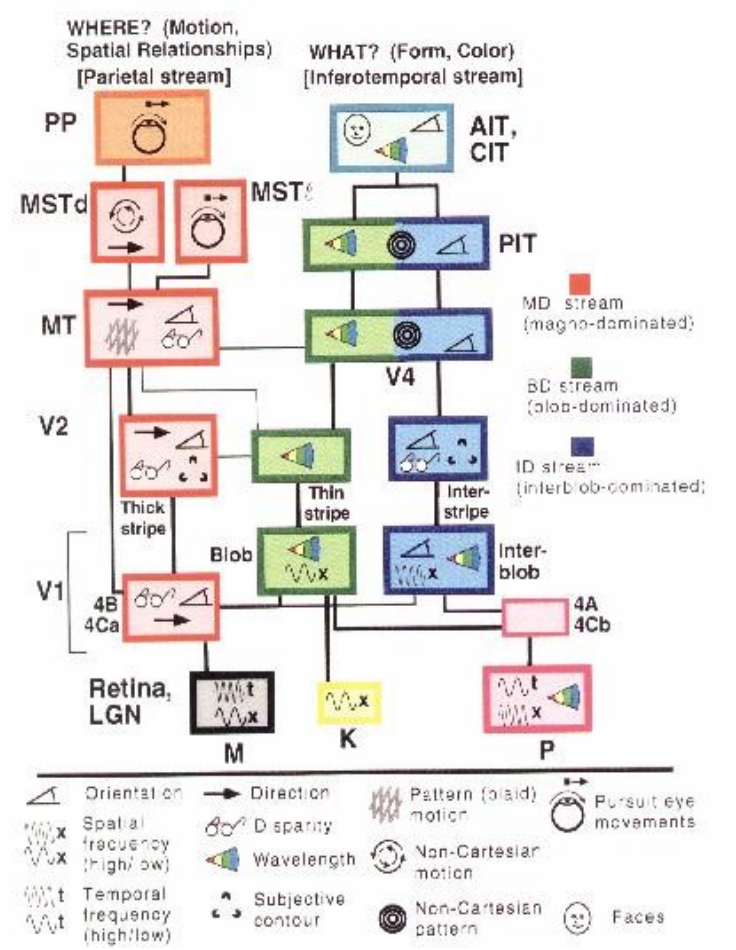
Learning Hierarchical Representations





- Hierarchy of representations with increasing level of abstraction. Each stage is a kind of trainable nonlinear feature transform
- Image recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story

The Mammalian Visual Cortex is Hierarchical



(van Essen and Gallant, 1994)

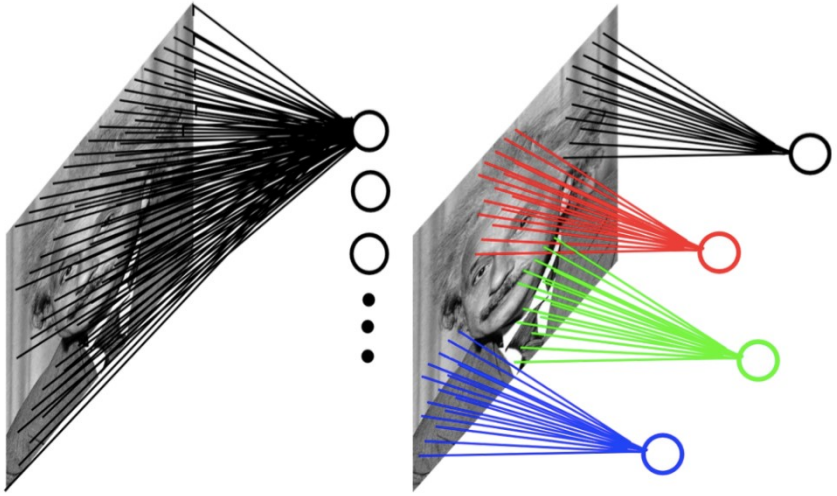
- It is good to be inspired by nature, but not too much.
- We need to understand which details are important, and which details are merely the result of evolution.
- Each module in Deep Learning transforms its input representation into a higher-level one, in a way similar to human cortex.

Supervised Learning

- Convolutional Neural Network
- Sequence Modelling
  - Why do we need RNN?
  - What are RNNs?
  - RNN Extensions
  - What can RNNs can do?

Convolutional Neural Network

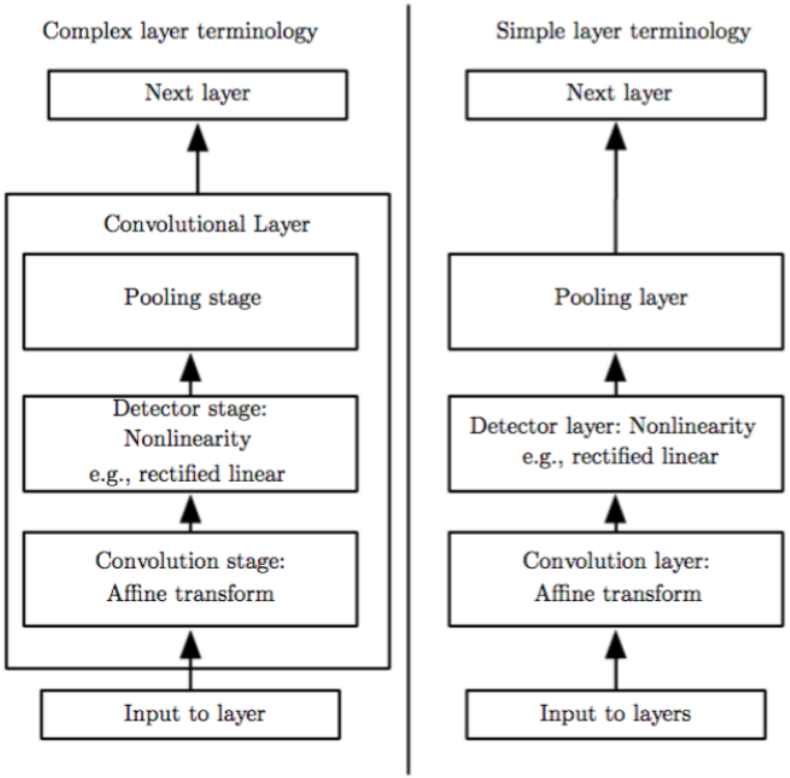
- Input can have very high dimension. Using a fully-connected neural network would need a large amount of parameters.
- Inspired by the neurophysiological experiments conducted by [Hubel & Wiesel 1962], CNNs are a special type of neural network whose hidden units are only connected to local receptive field. The number of parameters needed by CNNs is much smaller.



Example: 200x200 image

a) fully connected: 40,000 hidden units => 1.6 billion parameters
b) CNN: 5x5 kernel, 100 feature maps => 2,500 parameters

Three Stages of a Convolutional Layer

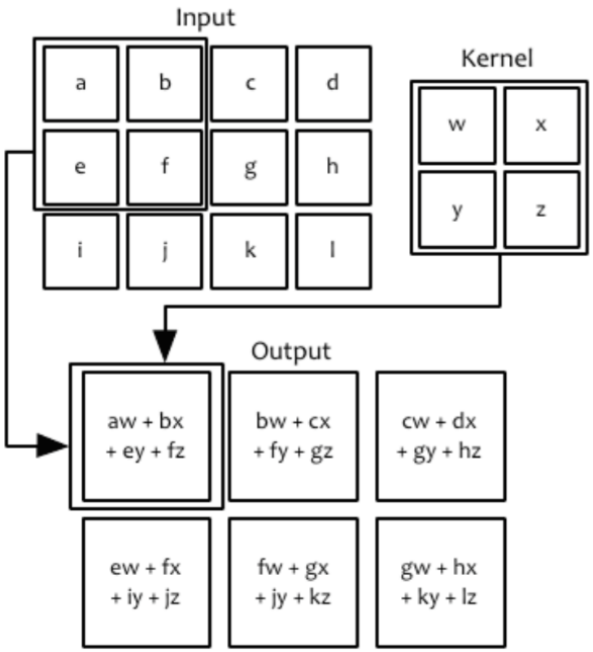


1. Convolution stage
2. Nonlinearity: a nonlinear transform such as rectified linear or $\tanh$
3. Pooling: output a summary statistics of local input, such as max pooling and average pooling

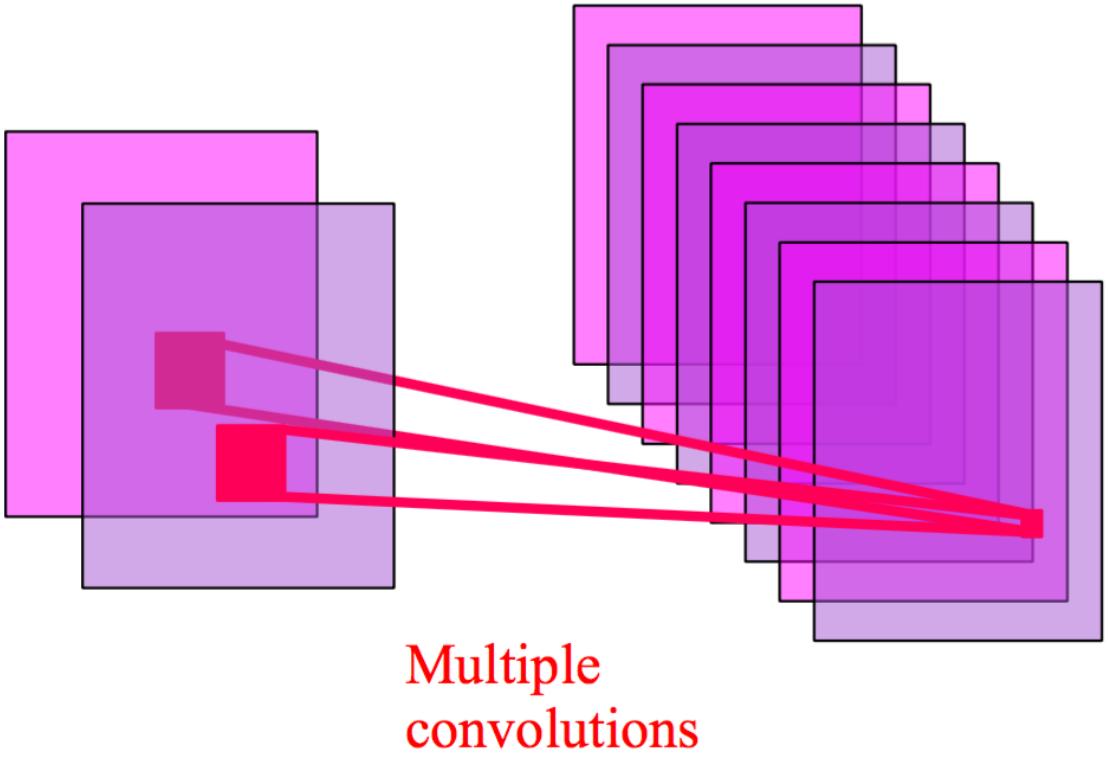
Convolution Operation in CNN

- Input: an image (2-D array) x
- Convolution kernel/operator(2-D array of learnable parameters): w
- Feature map (2-D array of processed data): s
- Convolution operation in 2-D domains:

$$s[i,j] = (x * w)[i,j] = \sum_{m=-M}^{M} \sum_{n=-N}^{N} x[i+m, j+n]\, w[m,n]$$



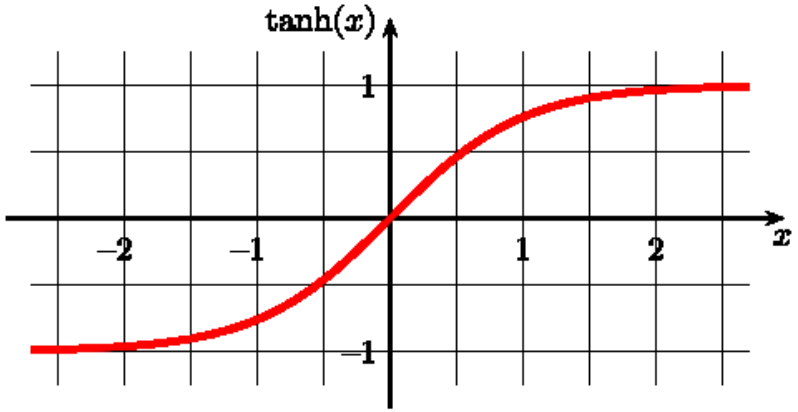
Multiple Convolutions



Usually there are multiple feature maps, one for each convolution operator.

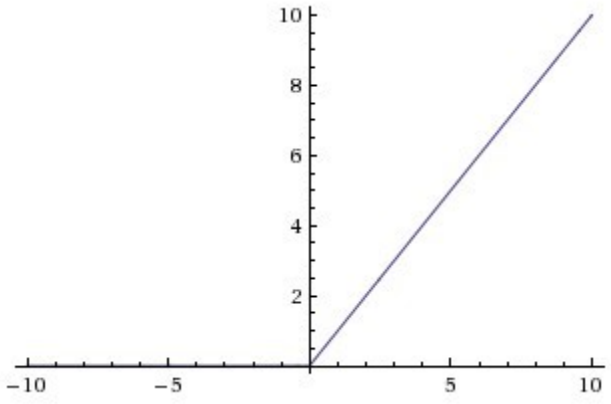
Non-linearity

Tanh(x)



$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

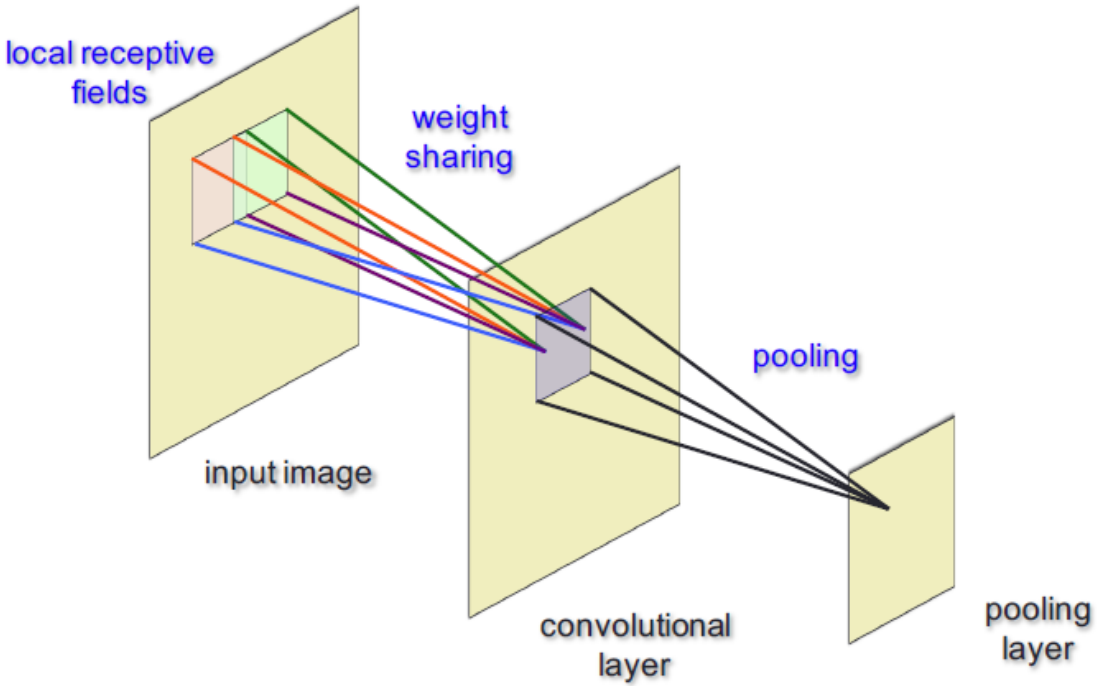
ReLU



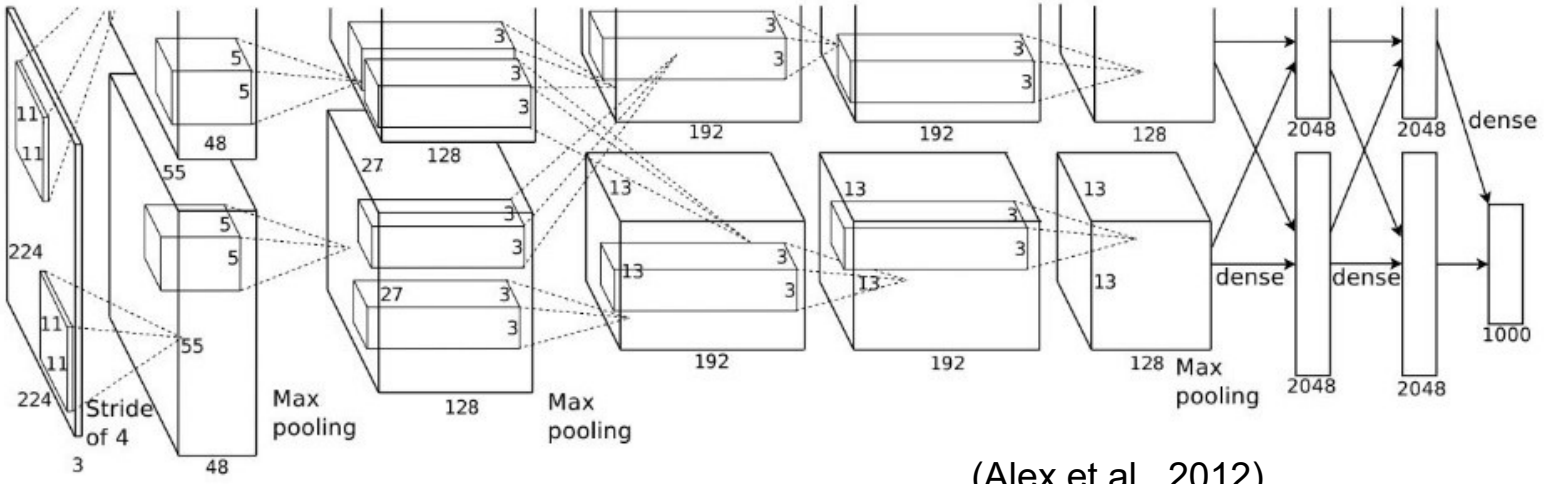
$$f(x) = \max(0, x)$$

Pooling

- Common pooling operations:
  - Max pooling: reports the maximum output within a rectangular neighborhood.
  - Average pooling: reports the average output of a rectangular neighborhood (possibly weighted by the distance from the central pixel).



Deep CNN: winner of ImageNet 2012



(Alex et al., 2012)

- Multiple feature maps per convolutional layer.
- Multiple convolutional layers for extracting features at different levels.
- Higher-level layers take the feature maps in lower-level layers as input.

Deep CNN for Image Classification

Classification

Click for a Quick Example

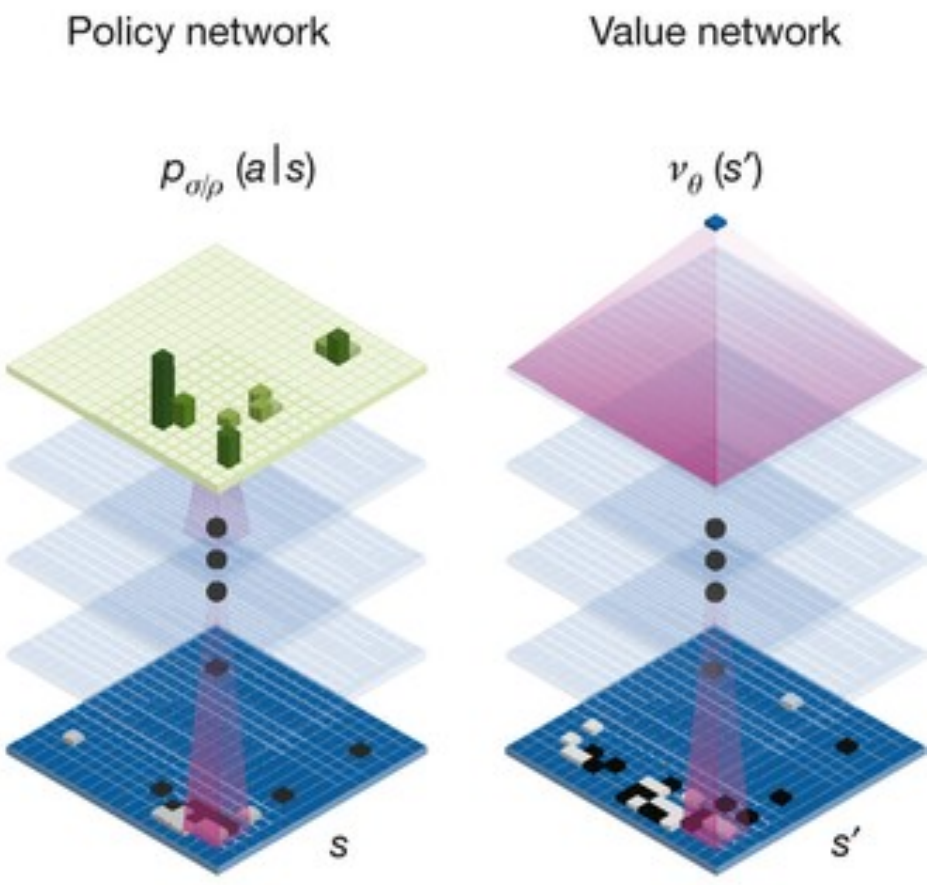
Maximally accurate | Maximally specific

cat	1.79306
feline	1.74269
domestic cat	1.70760
tabby	0.94807
domestic animal	0.76846

CNN took 0.064 seconds.

Try out a live demo at
http://demo.caffe.berkeleyvision.org/

Deep CNN in AlphaGO



(Silver et al, 2016)

Policy network:

- Input: 19x19, 48 input channels
- Layer 1: 5x5 kernel, 192 filters
- Layer 2 to 12: 3x3 kernel, 192 filters
- Layer 13: 1x1 kernel, 1 filter

Value network has similar architecture to policy network

Sequence Modelling

- Why do we need RNN?
- What are RNNs?
- RNN Extensions
- What can RNNs can do?

Why do we need RNNs?

The limitations of the Neural network (CNNs)

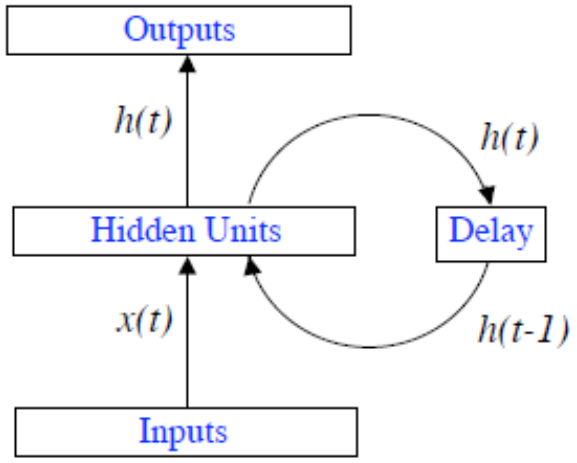
- Rely on the assumption of independence among the (training and test) examples.
  - After each data point is processed, the entire state of the network is lost
- Rely on examples being vectors of fixed length

We need to model the data with temporal or sequential structures and varying length of inputs and outputs

- Frames from video
- Snippets of audio
- Words pulled from sentences

What are RNNs?

Recurrent neural networks (RNNs) are connectionist models with the ability to selectively pass information across sequence steps, while processing sequential data one element at a time.



The simplest form of ***fully recurrent neural network*** is an MLP with the previous set of hidden unit activations feeding back into the network along with the inputs

Allow a 'memory' of previous inputs to persist in the network's internal state, and thereby influence the network output

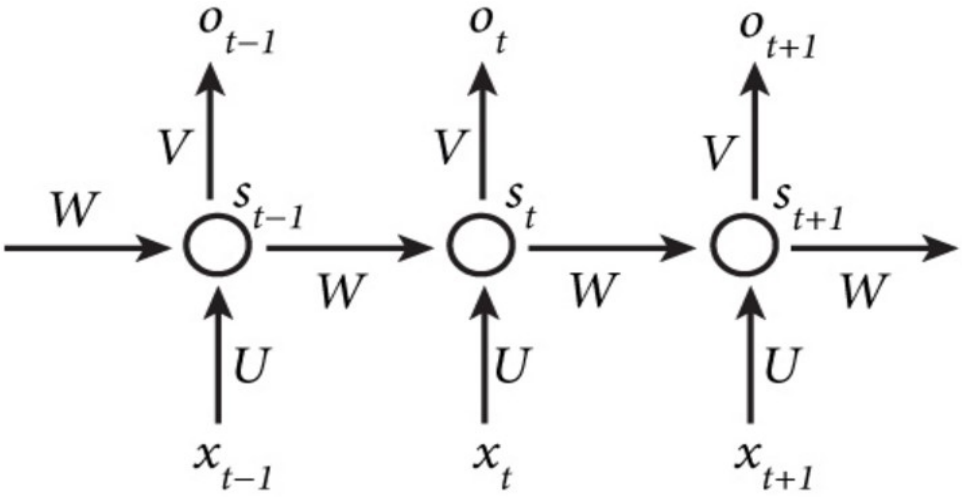
$$h(t) = f_H(W_{IH}x(t) + W_{HH}h(t-1))$$

$$y(t) = f_O(W_{HO}h(t))$$

f_H and f_O are the activation function for hidden and output unit; W_{IH}, W_{HH}, and W_{HO} are connection weight matrices which are learnt by training

What are RNNs?

- The recurrent network can be converted into a feed-forward network by ***unfolding over time***

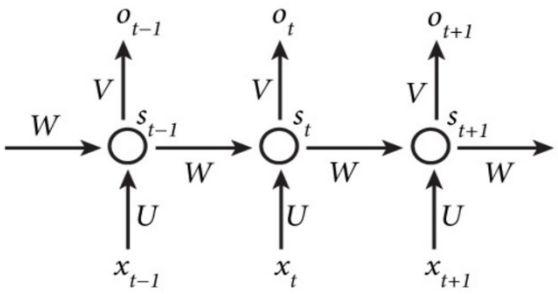


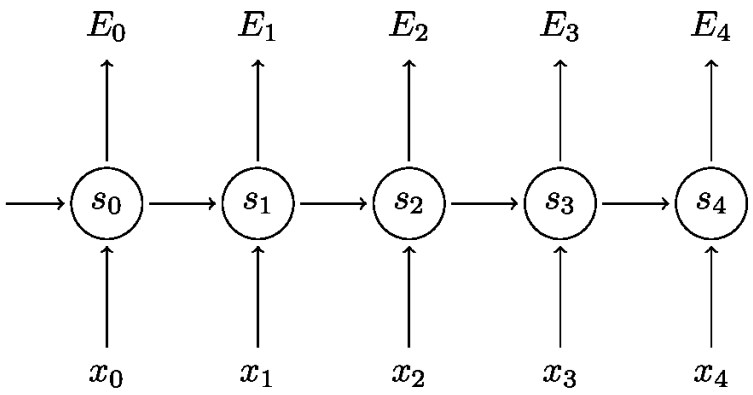
An unfolded recurrent network. Each node represents a layer of network units at a single time step. The weighted connections from the input layer to hidden layer are labelled ‘w1’, those from the hidden layer to itself (i.e. the recurrent weights) are labelled ‘w2’ and the hidden to output weights are labelled‘w3’. Note that the same weights are reused at every time step. Bias weights are omitted for clarity.

What are RNNs?

- Training RNNs (determine the parameters)

Back Propagation Through Time (BPTT) is often used to learn the RNN
BPTT is an extension of the back-propagation (BP)





- The output of this RNN is $\hat{y}_t$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = softmax(Vs_t)$$

- The loss/error function of this network is

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$ → The error at each time step

$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t)$$ → the total loss is the sum of the errors at each time step

What are RNNs?

- Training RNNs (determine the parameters)

✓ The gradients of the error with respect to our parameters

Just like we sum up the errors, we also sum up the gradients at each time step for one training example. For parameter W, the gradient is

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

✓ The gradient at each time step

we use time 3 as an example

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W} \longrightarrow \text{Chain Rule}$$

$$s_3 = \tanh(Ux_1 + Ws_2) \longrightarrow s_3 \text{ depends on } W \text{ and } s_1\text{, we cannot simply treat } s_2 \text{ a constant}$$

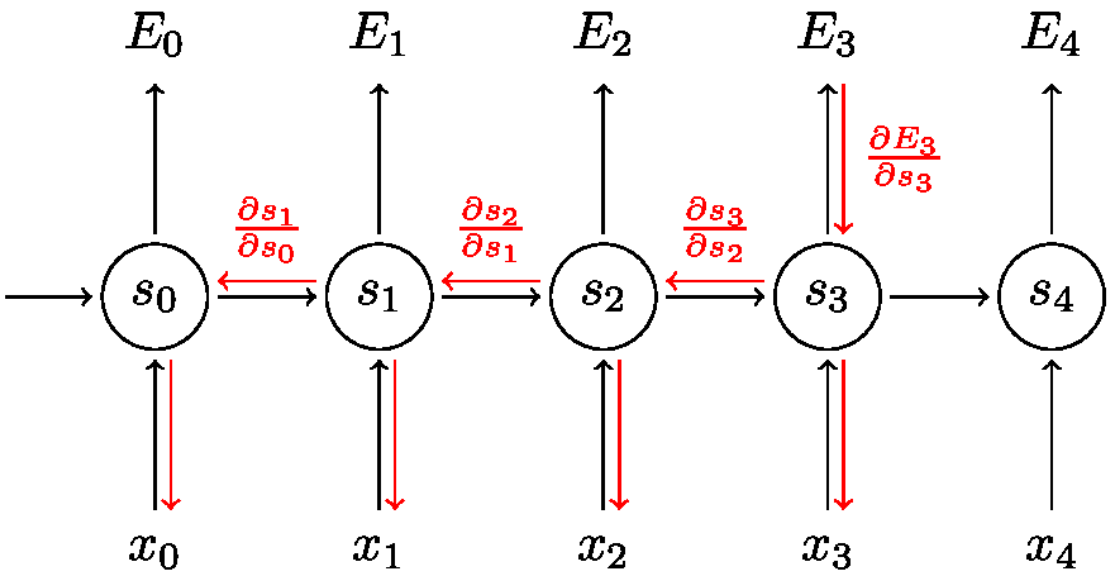
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W} \longrightarrow \text{Apply Chain Rule again on } s_k$$

What are RNNs?

- Training RNNs (determine the parameters)

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

Becaise W is used in every step up to the output we care about, we need to back-propagate gradients from $t = 3$ through the network all the way to $t = 0$



What are RNNs?

- The vanishing gradient problem

To understand why, let's take a closer look at the gradient we calculated above:

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} \longrightarrow \frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \frac{\partial s_k}{\partial W}$$

Because the layers and time steps of deep neural networks relate to each other through multiplication, derivatives are susceptible to vanishing

Gradient contributions from "far away" steps become zero, and the state at those steps doesn't contribute to what you are learning: You end up not learning long-range dependencies.

Proper initialization of the W matrix can reduce the effect of vanishing gradient

What are RNNs?

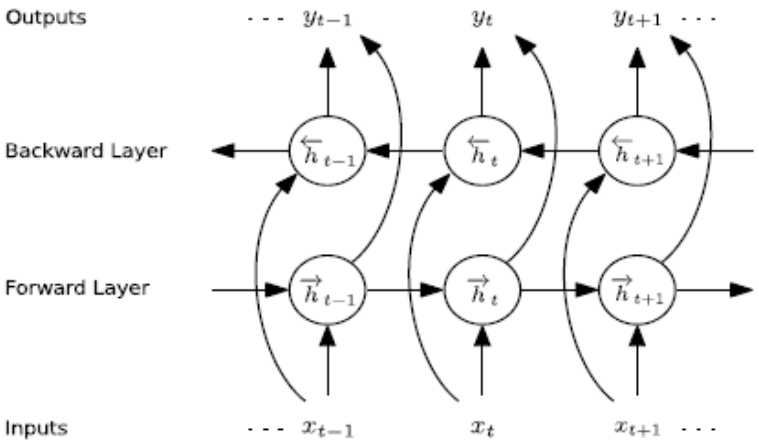
- How to sole the vanishing gradient problem?
  - Proper initialization of the W matrix can reduce the effect of vanishing gradients
  - Use ReLU instead of tanh or sigmoid activation function

    ReLU derivate is a constant of either 0 or 1, so it isn't likely to suffer from vanishing gradients
  - Use Long Short-Term Memory or Gated Recurrent unit architectures

    LSTM will be introduced later

RNN Extensions: Bidirectional Recurrent Neural Networks

Traditional RNNs only model the dependence of the current state on the previous state, BRNNs (*Schuster* and *Paliwal*, 1997) extend to model dependence on both past states and future states.
For example: predicting a missing word in a sequence you want to look at both the left and the right context.



An unfolded BRNN

$$\vec{h}_t = f(W_{x\vec{h}}x_t + W_{\vec{h}\vec{h}}\vec{h}_{t-1} + b_{\vec{h}})$$
$$\overleftarrow{h}_t = f(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t-1} + b_{\overleftarrow{h}})$$

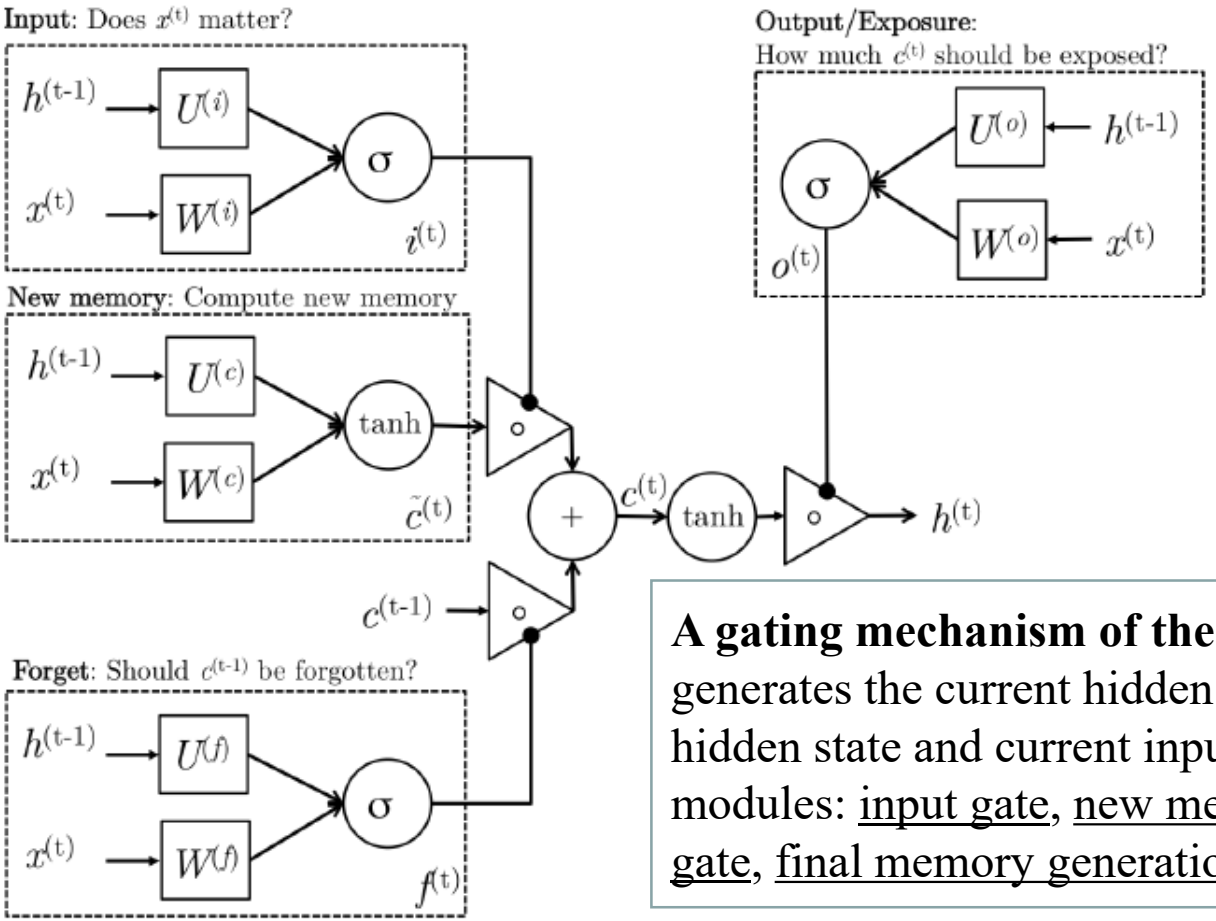
training sequence forwards and backwards to two separate recurrent hidden layers

$$y_t = W_{\vec{h}y}\vec{h}_t + W_{\overleftarrow{h}y}\overleftarrow{h}_t + b_y$$

past and future context determines the output

RNN Extensions: Long Short-term Memory

The vanishing gradient problem prevents standard RNNs from learning long-term dependencies. LSTMs (Hochreiter and Schmidhuber, 1997) were designed to combat vanishing gradients through a *gating* mechanism.

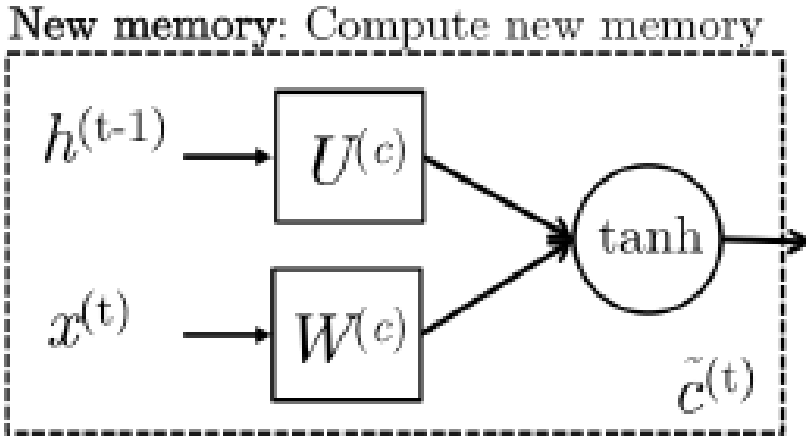


A gating mechanism of the LSTM , which generates the current hidden state by the paste hidden state and current input **..**It contains five modules: input gate, new memory cell, forget gate, final memory generation, and output gate.

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

New memory cell



$$\tilde{c}_t = \tanh(W^c x_t + U^c h_{t-1})$$

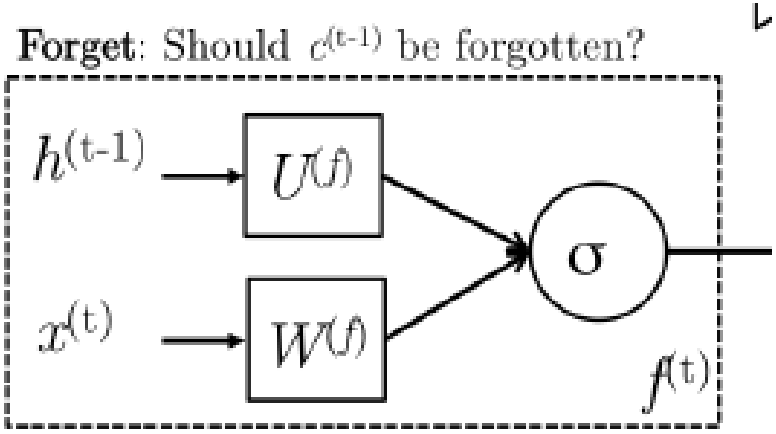
New memory

use the input word and the past hidden state to generate a new memory which includes aspects of the new input

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

Forget gate



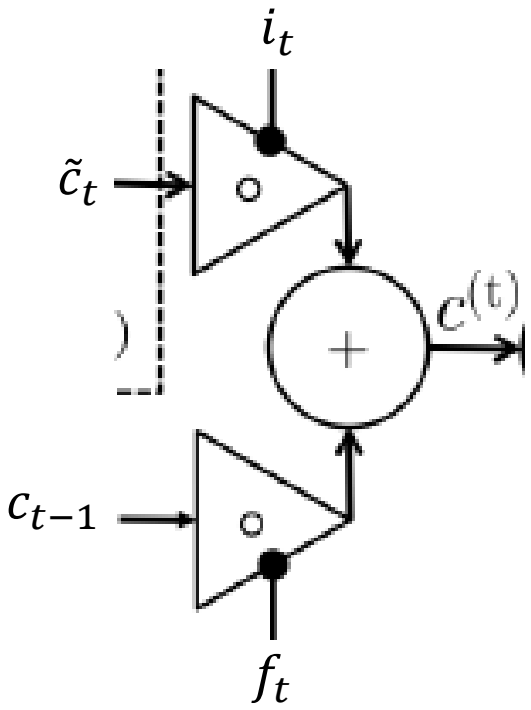
$$f_t = \sigma(W^f x_t + U^f h_{t-1})$$

The forget gate looks at the input word and the past hidden state and makes an assessment on whether the past memory cell is useful for the computation of the current memory cell

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

Final memory cell



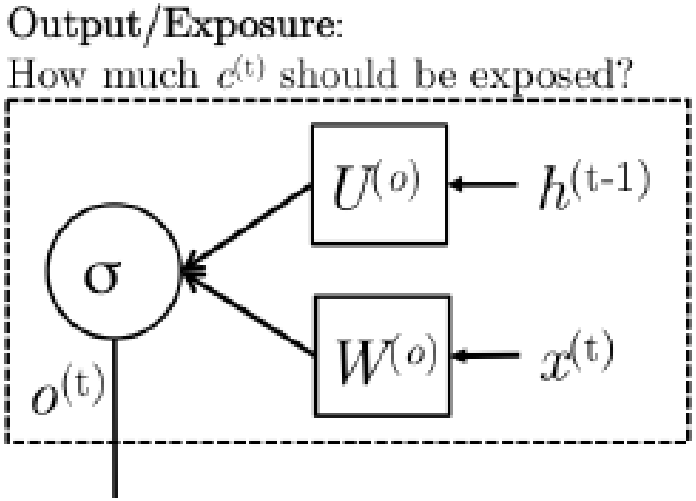
$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t$$

This stage first takes the advice of the forget gate f_t and accordingly forgets the past memory c_{t-1}. Similarly, it takes the advice of the input gate i_t and accordingly gates the new memory. It then sums these two results to produce the final memory

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

Output gate



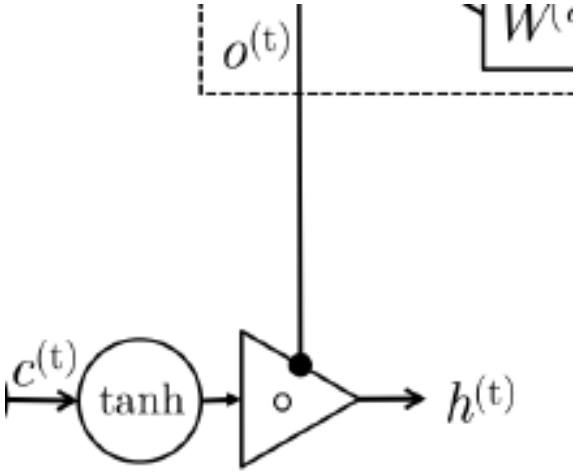
$$o_t = \sigma(W_o x_t + U_o h_{t-1})$$

This gate makes the assessment regarding what parts of the memory c_t needs to be exposed/present in the hidden state h_t.

RNN Extensions: Long Short-term Memory

A gating mechanism of the LSTM

The hidden state



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

RNN extensions: Long Short-term memory

Conclusions on LSTM

LSTMs contain information outside the normal flow of the recurrent network in a gated cell. Information can be stored in, written to, or read from a cell, much like data in a computer's memory. The cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, back-propagating error, and adjusting weights via gradient descent.

RNN extensions: Long Short-term Memory

Why LSTM can combat the vanish gradient problem?

LSTMs help preserve the error that can be back-propagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely

What can RNNs can do?

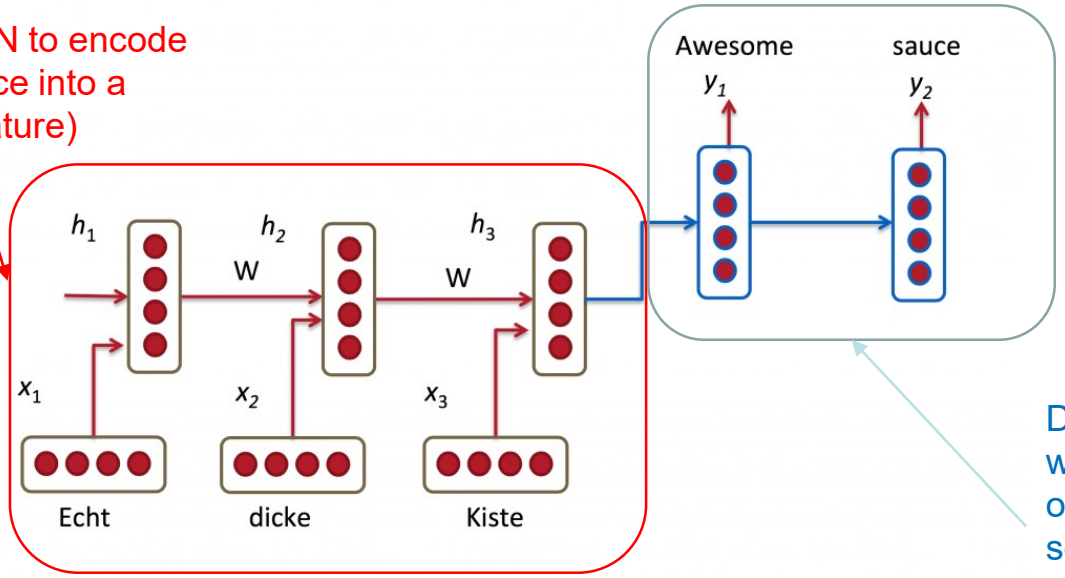
Machine Translation

Visual Question Answering

Machine Translation

In machine translation, the input is a sequence of words in source language, and the output is a sequence of words in target language.

Encoder: An RNN to encode the input sentence into a hidden state (feature)



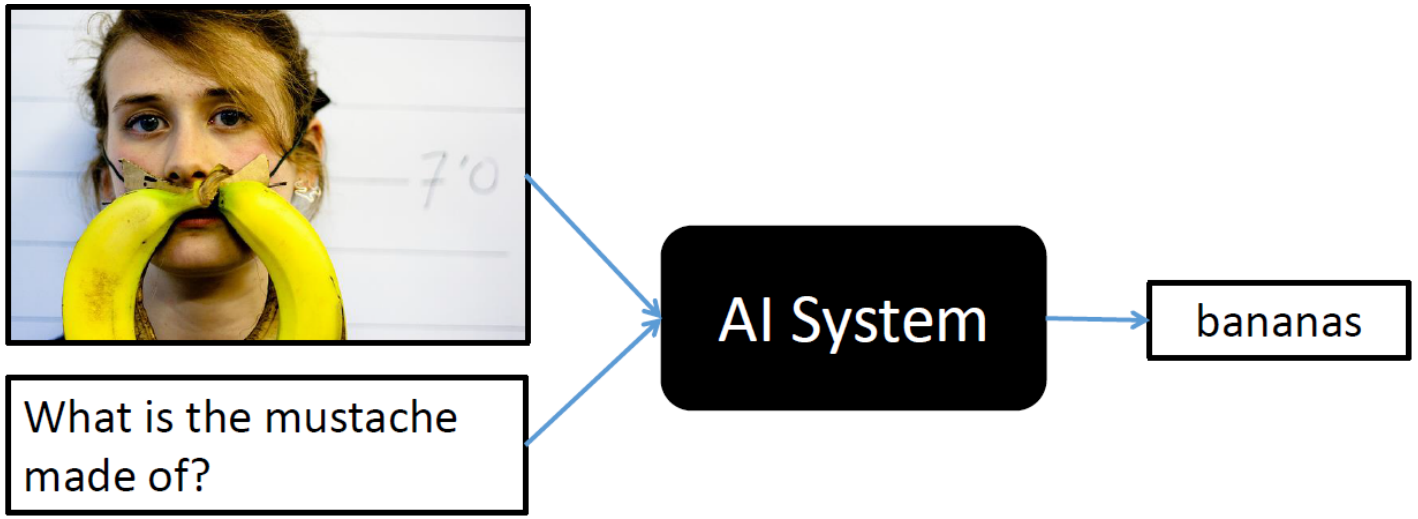
Encoder-decoder architecture for machine translation

Decoder: An RNN with the hidden state of the sentence in source language as the input and output the translated sentence

Visual Question Answering (VQA)

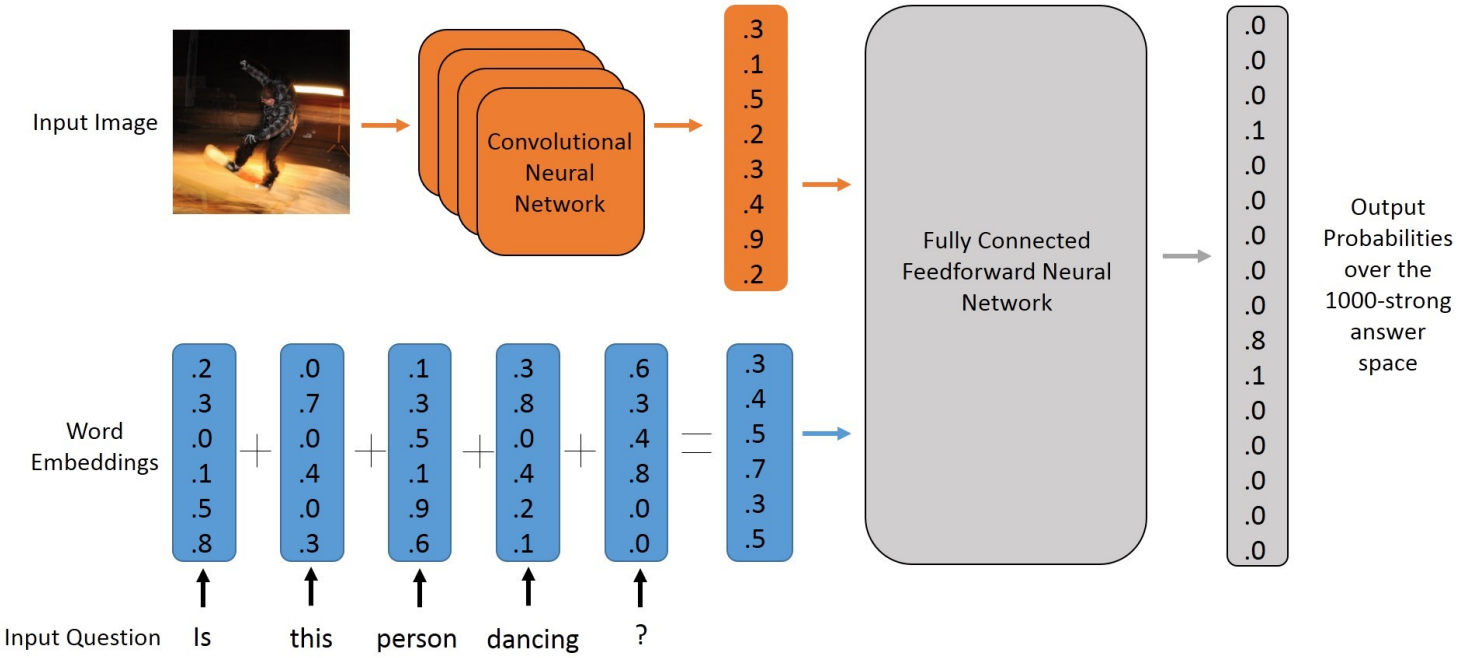
Demo Website

VQA: Given an image and a natural language question about the image, the task is to provide an accurate natural language answer



Picture from (Antol et al., 2015)

Visual Question Answering

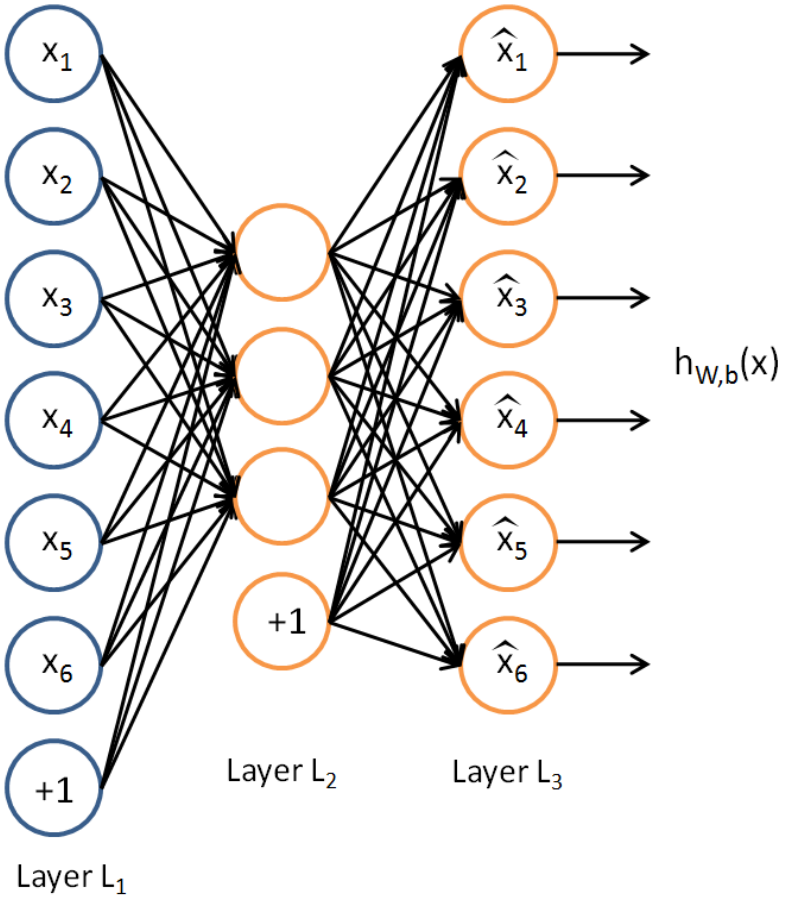


The output is to be conditioned on both image and textual inputs. A CNN is used to encode the image and a RNN is implemented to encode the sentence.

Unsupervised Learning

- Autoencoders
- Deep Autoencoders
- Denoising Autoencoders
- Stacked Denoising Autoencoders

Autoencoders



An Autoencoder is a feedforward neural network that learns to predict the input itself in the output.

$$y^{(i)} = x^{(i)}$$

- The input-to-hidden part corresponds to an encoder
- The hidden-to-output part corresponds to a decoder.

Deep Autoencoders

- A deep Autoencoder is constructed by extending the encoder and decoder of autoencoder with multiple hidden layers.
- Gradient vanishing problem: the gradient becomes too small as it passes back through many layers

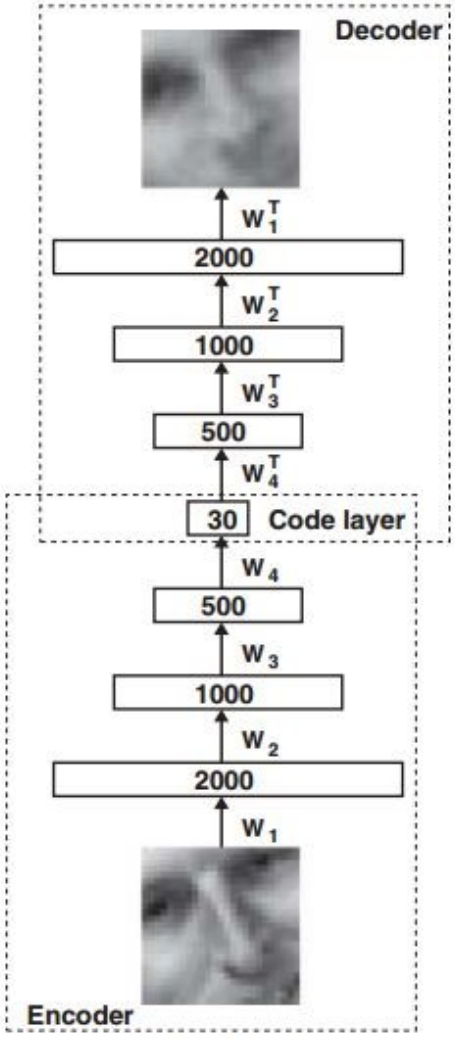


Diagram from (Hinton and Salakhutdinov, 2006)

Training Deep Autoencoders

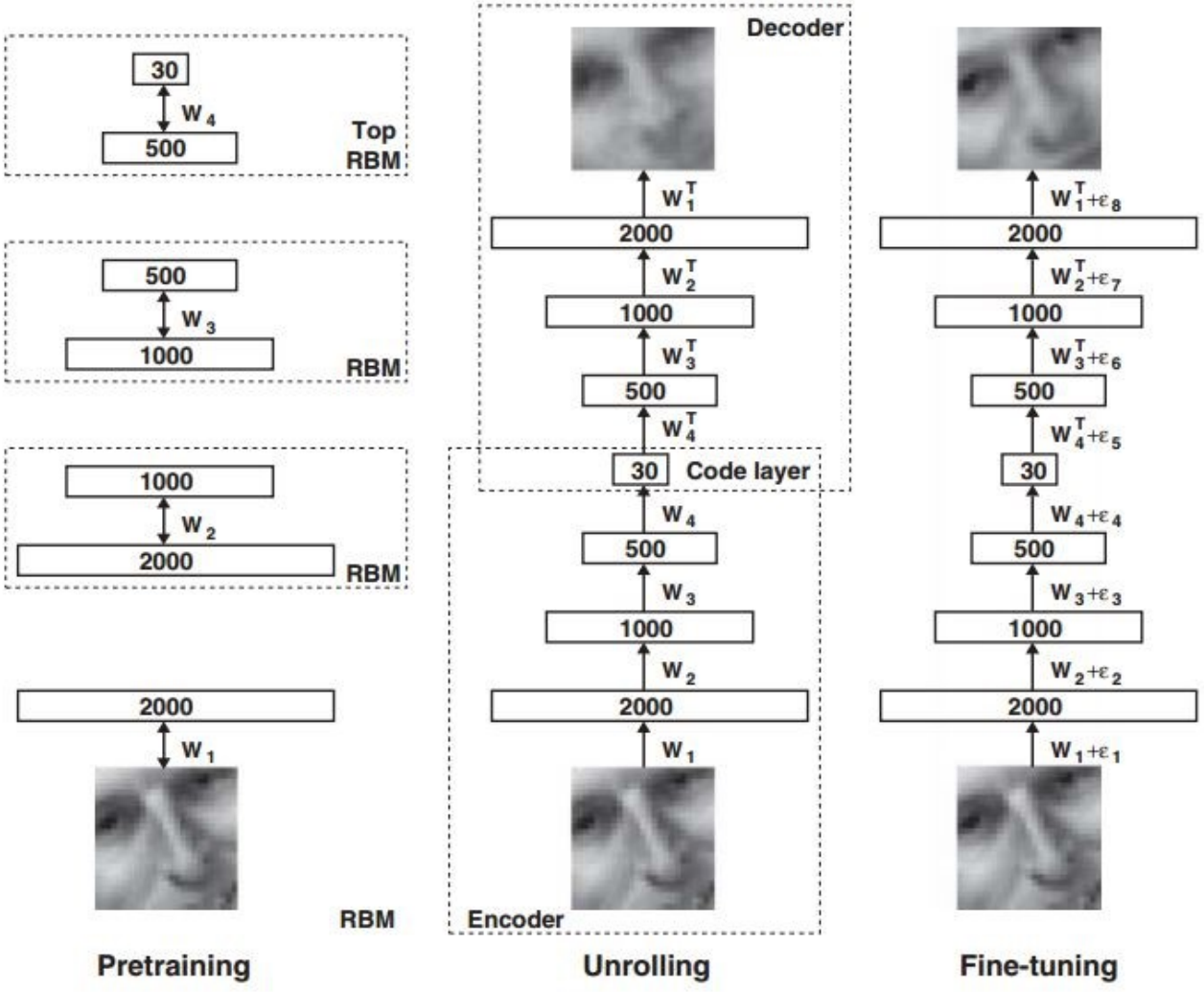
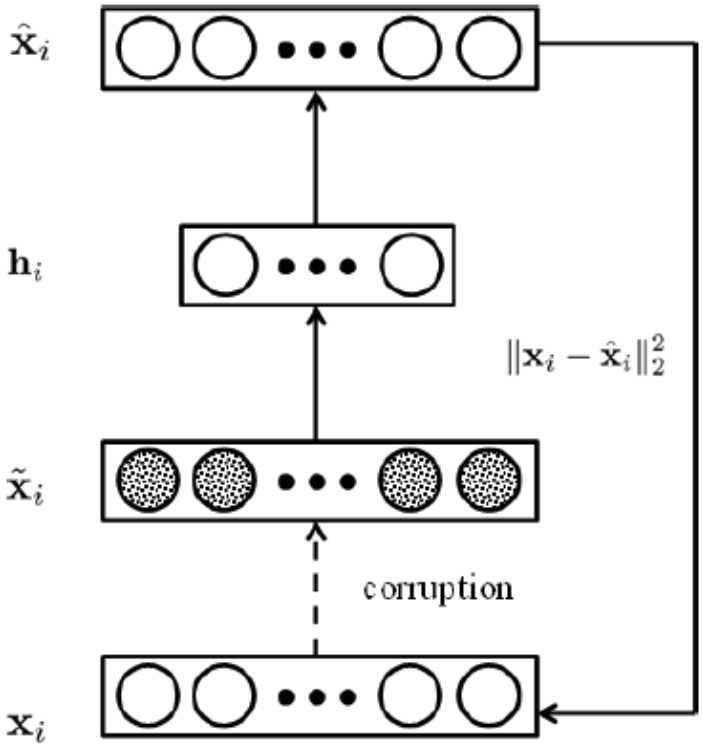


Diagram from (Hinton and Salakhutdinov, 2006)

Denoising Autoencoders

- By adding stochastic noise to the, it can force Autoencoder to learn more robust features.



Training Denoising Autoencoder

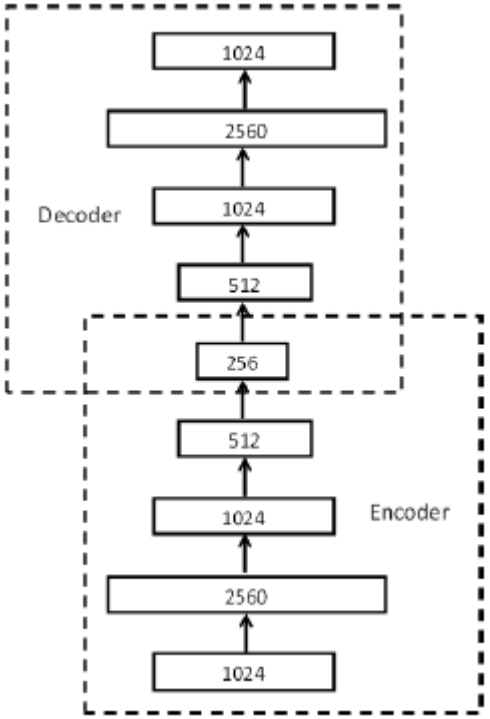
The loss function of Denoising autoencoder:

$$\min_{\mathbf{W}_1,\mathbf{W}_2,\mathbf{b}_1,\mathbf{b}_2} \sum_{\ell} \|\mathbf{x}^{(\ell)} - \hat{\mathbf{x}}^{(\ell)}\|_2^2 + \lambda\left(\|\mathbf{W}_1\|_F^2 + \|\mathbf{W}_2\|_F^2\right)$$

where

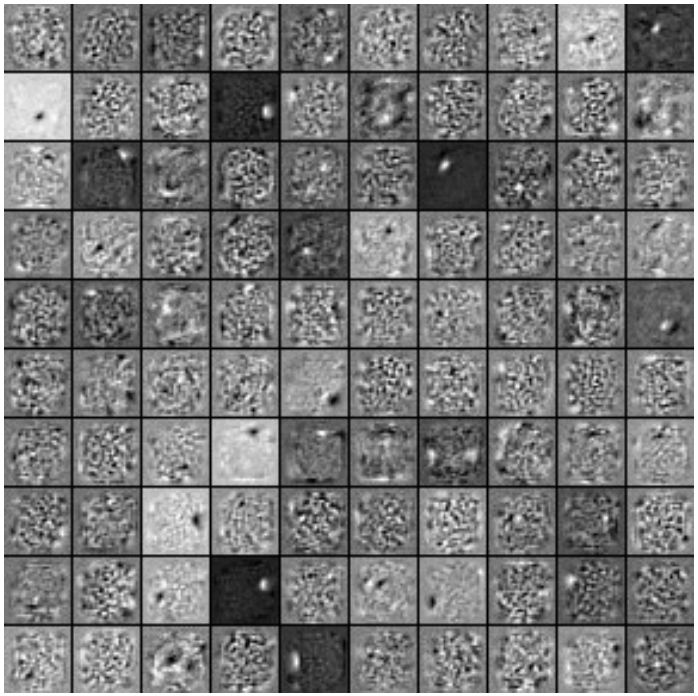
$$\mathbf{h}^{(\ell)} = \sigma(\mathbf{W}_1\tilde{\mathbf{x}}^{(\ell)} + \mathbf{b}_1)$$
$$\hat{\mathbf{x}}^{(\ell)} = \sigma(\mathbf{W}_2\mathbf{h}^{(\ell)} + \mathbf{b}_2)$$

Like deep Autoencoder, we can stack multiple denoising autoencoders layer-wisely to form a Stacked Denoising Autoencoder.

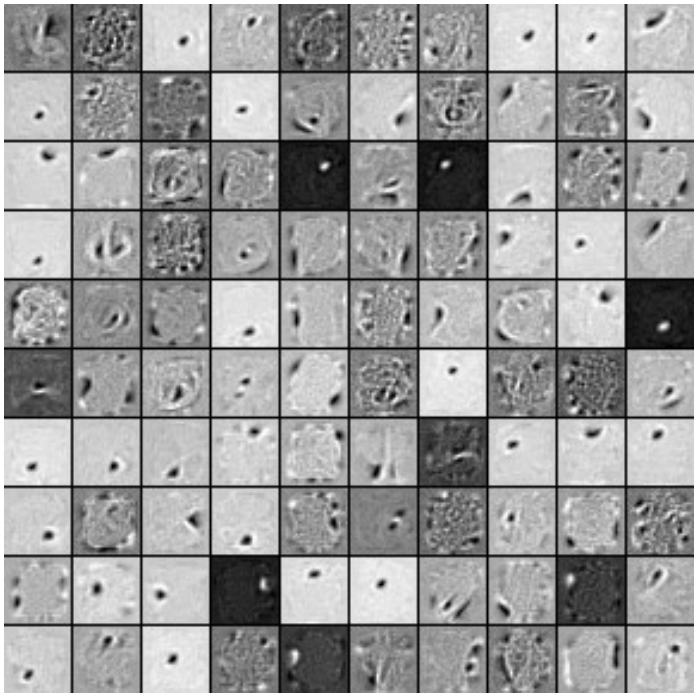


Training Denoising Autoencoder on MNIST

- The following pictures show the difference between the resulting filters of Denoising Autoencoder trained on MNIST with different noise ratios.



No noise (noise ratio=0%)



noise ratio=30%

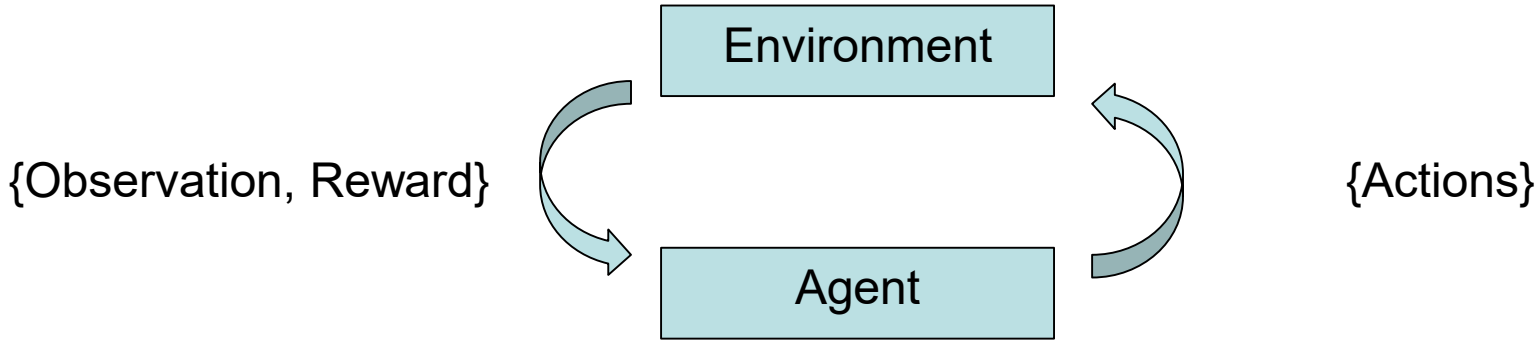
Diagram from (Hinton and Salakhutdinov, 2006)

Deep Reinforcement Learning

- Reinforcement Learning
- Deep Reinforcement Learning
- Applications
  - Playing Atari Games
  - AlphaGO

Reinforcement Learning

- What's Reinforcement Learning?



- Agent interacts with an environment and learns by maximizing a scalar reward signal
- No labels or any other supervision signal.
- Previously suffering from hand-craft states or representation.

Policies and Value Functions

- Policy π is a behavior function selecting actions given states

$$a = \pi(\mathrm{s})$$

- Value function Q^{π}(s,a) is expected total reward r from state s and action a under policy π

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

"How good is action a in state s?"

Approaches To Reinforcement Learning

- Policy-based RL
  - Search directly for the optimal policy π^*
  - Policy achieving maximum future reward
- Value-based RL
  - Estimate the optimal value function Q^*(s,a)
  - Maximum value achievable under any policy
- Model-based RL
  - Build a transition model of the environment
  - Plan (e.g. by look-ahead) using model

Bellman Equation

- Value function can be unrolled recursively

$$Q^{\pi}(s,a) = \mathbb{E}\left[r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \ldots \mid s,a\right]$$
$$= \mathbb{E}_{s'}\left[r + \gamma Q^{\pi}(s',a') \mid s,a\right]$$

- Optimal value function $Q*$ (s, a) can be unrolled recursively

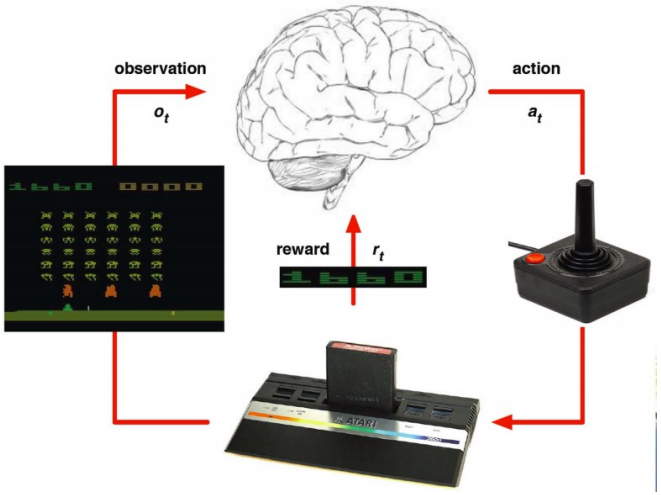
$$Q^{*}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q^{*}(s',a') \mid s,a\right]$$

- Value iteration algorithms solve the Bellman equation

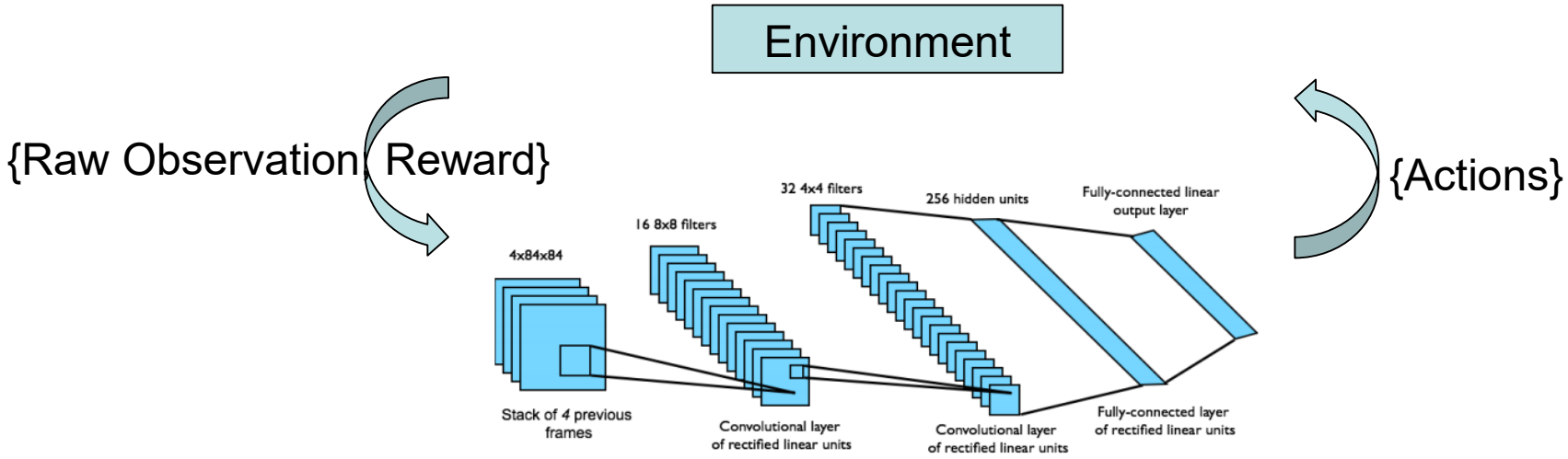
$$Q_{i+1}(s,a) = \mathbb{E}_{s'}\left[r + \gamma \max_{a'} Q_i(s',a') \mid s,a\right]$$

Deep Reinforcement Learning

- Human



- So what's **DEEP** RL?



Deep Reinforcement Learning

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

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

- Define objective function by mean-squared error in Q-values

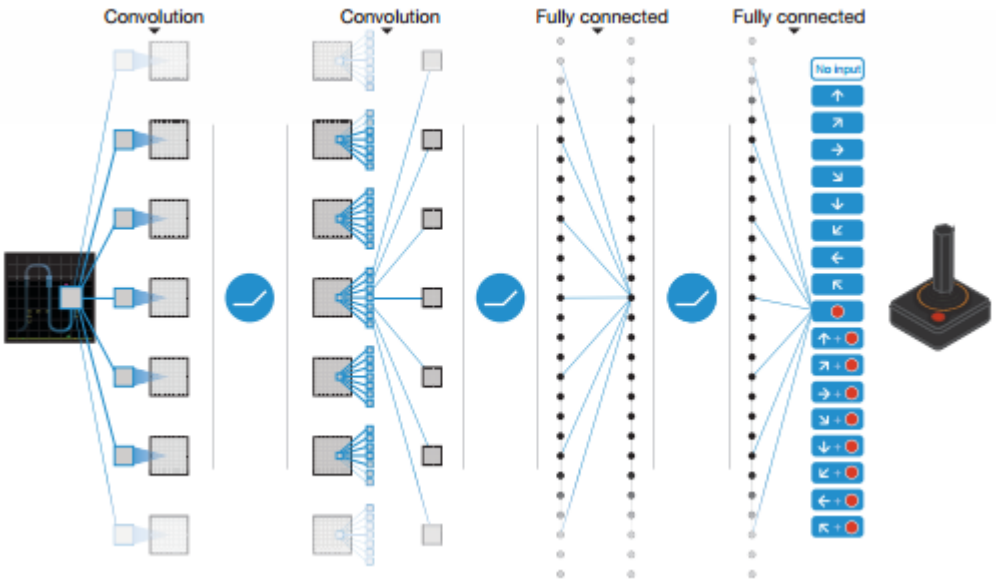
$$\mathcal{L}(w) = \mathbb{E}\left[\left(\underbrace{r + \gamma \max_{a'} Q(s', a', w)}_{\text{target}} - Q(s, a, w)\right)^2\right]$$

- Leading to the following Q-learning gradient

$$\frac{\partial \mathcal{L}(w)}{\partial w} = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a', w) - Q(s, a, w)\right) \frac{\partial Q(s, a, w)}{\partial w}\right]$$

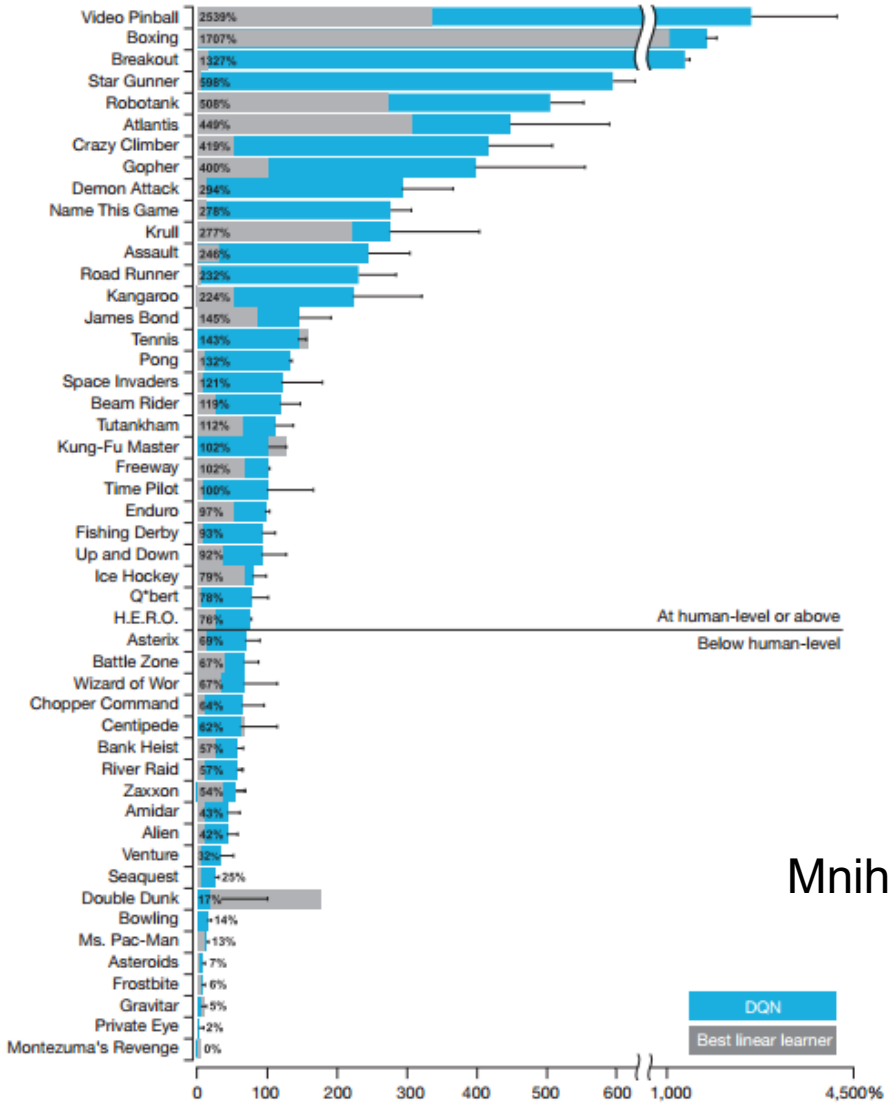
DQN in Atari

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



Mnih, Volodymyr, et al. 2015.

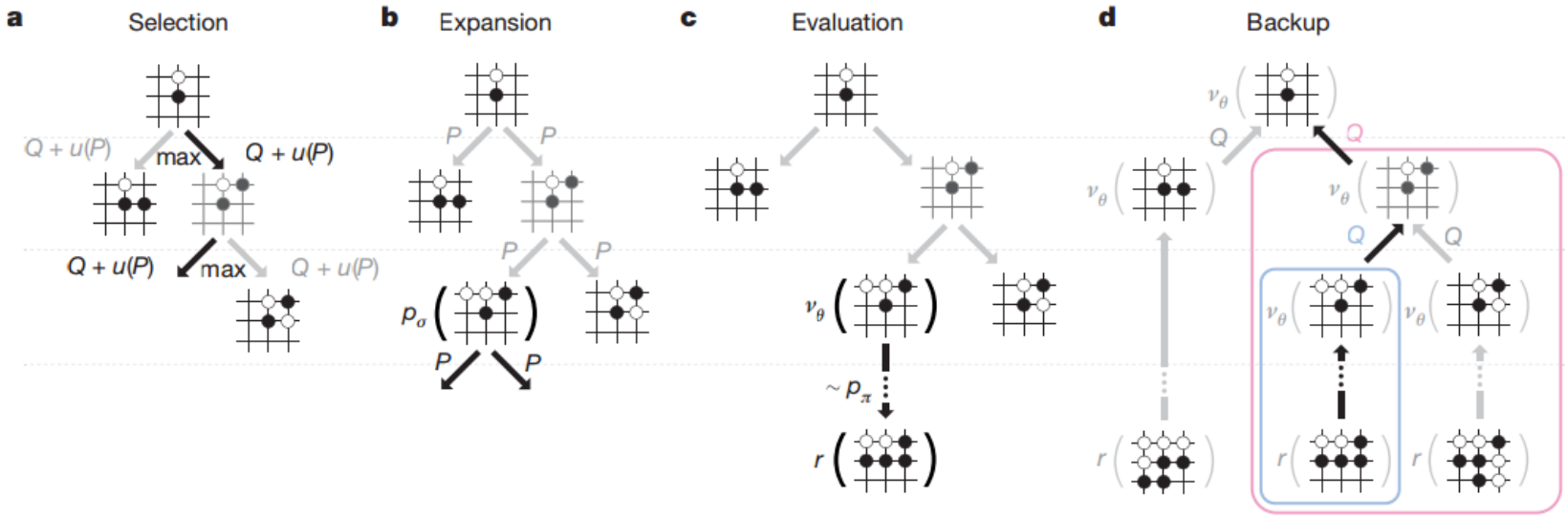
DQN in Atari : Human Level Control



Mnih, Volodymyr, et al. 2015.

AlphaGO:
Monte Carlo Tree Search

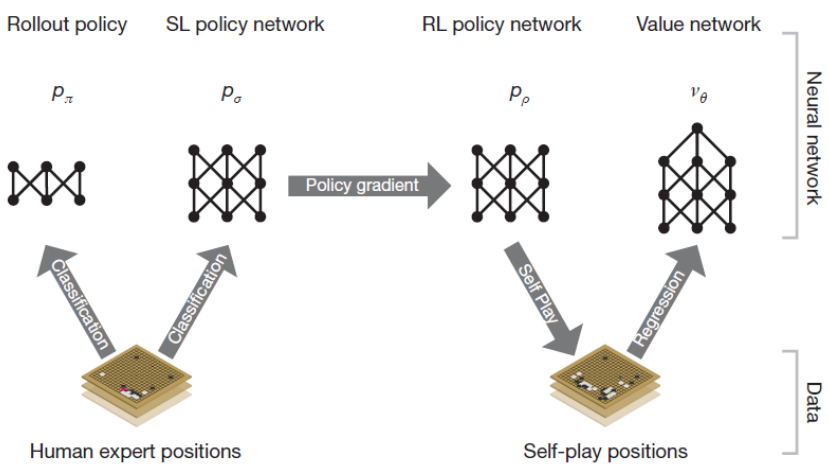
- MCTS: Model look ahead to reduce searching space by predicting opponent's moves



Silver, David, et al. 2016.

AlphaGO: Learning Pipeline

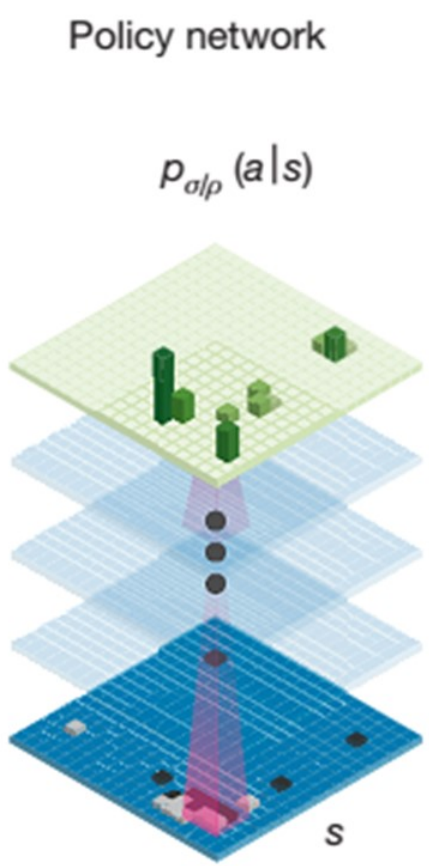
- Combine SL and RL to learn the search direction in MCTS



Silver, David, et al. 2016.

- SL policy Network
  - Prior search probability or potential
- Rollout:
  - combine with MCTS for quick simulation on leaf node
- Value Network:
  - Build the Global feeling on the leaf node situation

Learning to Prune: SL Policy Network



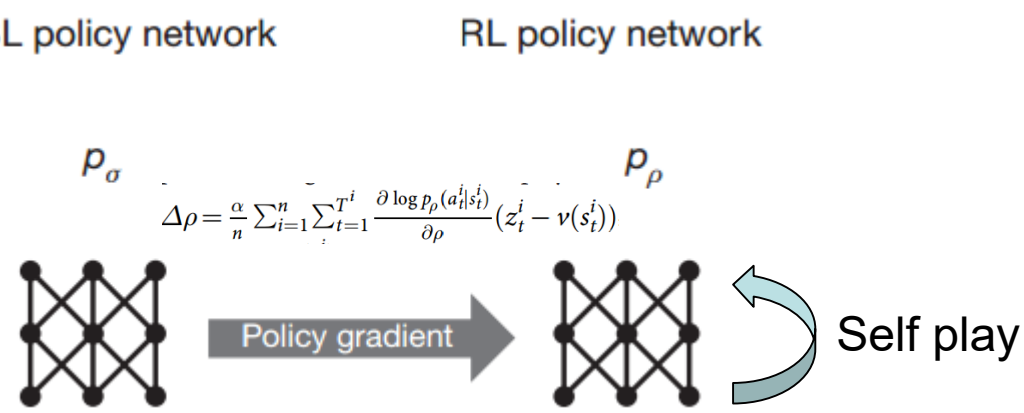
- 13-layer CNN
- Input board position s
- Output: $\mathrm{p}_{\sigma}(a|s)$, where a is the next move

Extended Data Table 2 | Input features for neural networks

Feature	# of planes	Description
Stone colour	3	Player stone / opponent stone / empty
Ones	1	A constant plane filled with 1
Turns since	8	How many turns since a move was played
Liberties	8	Number of liberties (empty adjacent points)
Capture size	8	How many opponent stones would be captured
Self-atari size	8	How many of own stones would be captured
Liberties after move	8	Number of liberties after this move is played
Ladder capture	1	Whether a move at this point is a successful ladder capture
Ladder escape	1	Whether a move at this point is a successful ladder escape
Sensibleness	1	Whether a move is legal and does not fill its own eyes
Zeros	1	A constant plane filled with 0
Player color	1	Whether current player is black

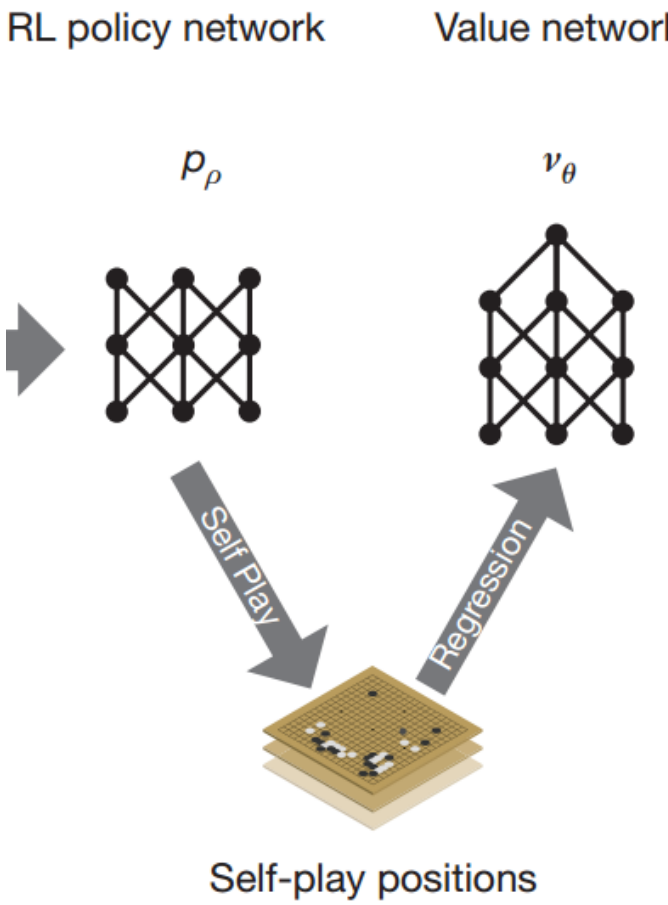
Feature planes used by the policy network (all but last feature) and value network (all features).

Learning to Prune：RL Policy Network



- 1 Million samples are used to train.
- RL-Policy network *VS* SL-Policy network.
  - RL-Policy alone wins 80% games against SL-Policy.
  - Combined with MCTS, SL-Policy network is better
- Used to derive the Value Network as the ground truth
  - Making enough data for training

Learning to Prune：Value Network

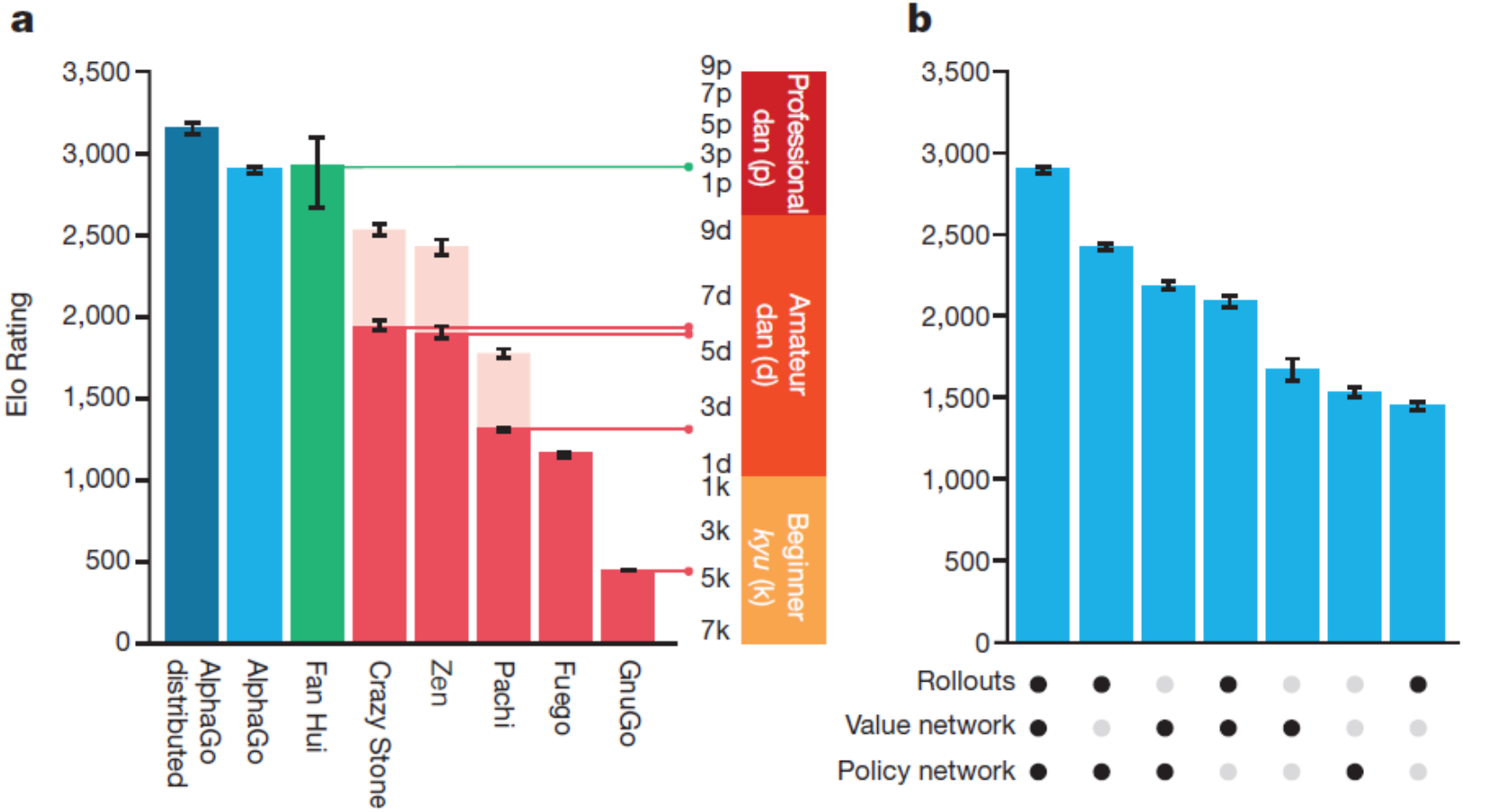


- Regression: Similar architecture

$$v^p(s) = \mathbb{E}[z_t | s_t = s, a_{t...T} \sim p]$$

- SL Network: Sampling to generate a unique game.
- RL Network: Simulate to get the game's final result.
- Train: 50 million mini-batches of 32 positions(30 million unique games)

AlphaGO：Evaluation



Silver, David, et al. 2016.

The version solely using the policy network does not perform any search

References

- Antol, S., Agrawal, A., Lu, J., Mitchell, M., Batra, D., Lawrence Zitnick, C., & Parikh, D. (2015). VQA: Visual question answering. In *Proceedings of the IEEE International Conference on Computer Vision* (pp. 2425-2433).
- Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science*, *313*(5786), 504-507.
- Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *Signal Processing, IEEE Transactions on*, *45*(11), 2673-2681.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).
- Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735-1780.
- Mnih, Volodymyr, et al. "Playing atari with deep reinforcement learning."*arXiv preprint arXiv:1312.5602* (2013).
- Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529-533.
- Silver, David, et al. "Mastering the game of Go with deep neural networks and tree search." *Nature* 529.7587 (2016): 484-489.
- Silver, D. (2015). Deep Reinforcement Learning [Powerpoint slides]. Retrieve from http://www.iclr.cc/lib/exe/fetch.php?media=iclr2015:silver-iclr2015.pdf
- Lecun, Y., & Ranzato, M. (2013). Deep Learning Tutorial [Powerpoint slides]. Retrieved from http://www.cs.nyu.edu/~yann/talks/lecun-ranzato-icml2013.pdf