COMP3411: Artificial Intelligence

Extension 10. Deep Learning

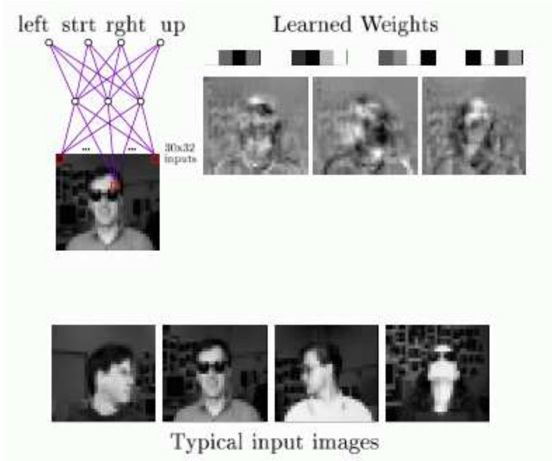
Outline

- Image Processing
  - Convolutional Networks
- Language Processing
  - Recurrent Networks
  - Long Short Term Memory
  - Word Embeddings
- Deep Reinforcement Learning
  - Deep Q-Learning
  - Policy Gradients
  - Asynchronous Advantage Actor Critic

Image Processing Tasks

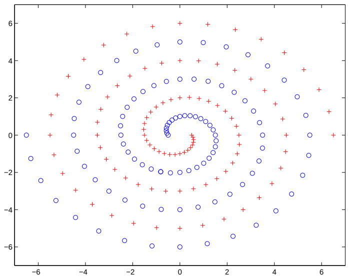
- image classification
- object detection
- object segmentation
- style transfer
- generating images
- generating art

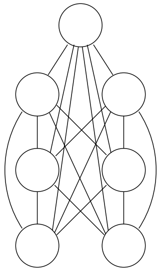
Learning Face Direction



Limitations of Two-Layer Neural Networks

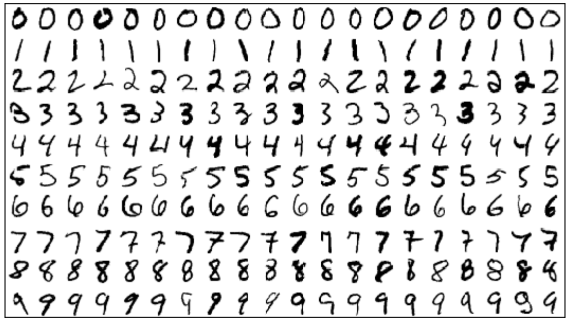
Some functions cannot be learned with a 2-layer sigmoidal network.





For example, this Twin Spirals problem cannot be learned with a 2-layer network, but it can be learned using a 3-layer network if we include shortcut connections between non-consecutive layers.

MNIST Handwritten Digit Dataset

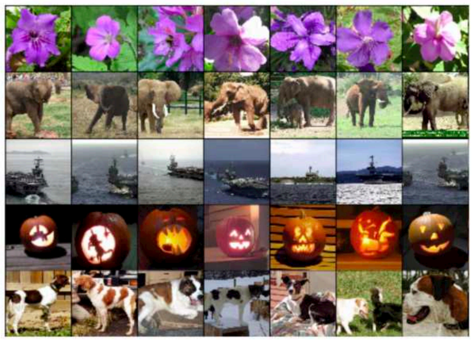


- black and white, resolution 28×28
- 60,000 images
- 10 classes $(0, 1, 2, 3, 4, 5, 6, 7, 8, 9)$

CIFAR Image Dataset

- color, resolution 32×32
- 50,000 images
- 10 classes

ImageNet LSVRC Dataset



- color, resolution 227×227
- 1.2 million images
- 1000 classes

Vanishing / Exploding Gradients

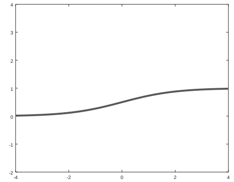
Training by backpropagation in networks with many layers is difficult.

When the weights are small, the differentials become smaller and smaller as we backpropagate through the layers, and end up having no effect.

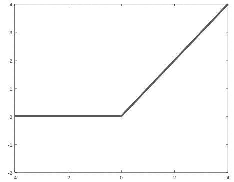
When the weights are large, the activations in the higher layers will saturate to extreme values. As a result, the gradients at those layers will become very small, and will not be propagated to the earlier layers.

When the weights have intermediate values, the differentials will sometimes get multiplied many times is places where the transfer function is steep, causing them to blow up to large values.

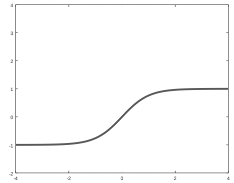
Activation Functions (6.3)



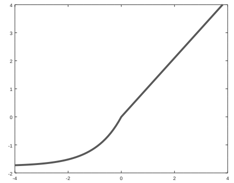
Sigmoid



Rectified Linear Unit (ReLU)

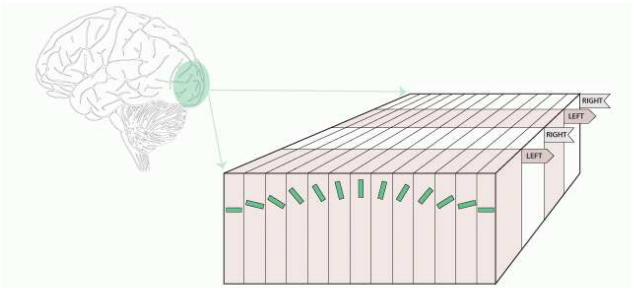


Hyperbolic Tangent



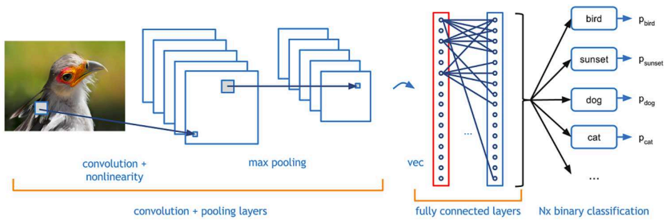
Scaled Exponential Linear Unit (SELU)

Hubel and Weisel – Visual Cortex



- cells in the visual cortex respond to lines at different angles
- cells in V2 respond to more sophisticated visual features
- Convolutional Neural Networks are inspired by this neuroanatomy
- CNN's can now be simulated with massive parallelism, using GPU's

Convolutional Networks

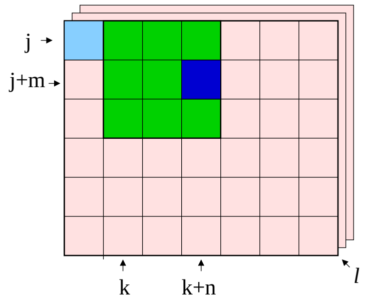


Suppose we want to classify an image as a bird, sunset, dog, cat, etc.

If we can identify features such as feather, eye, or beak which provide useful information in one part of the image, then those features are likely to also be relevant in another part of the image.

We can exploit this regularity by using a convolution layer which applies the same weights to different parts of the image.

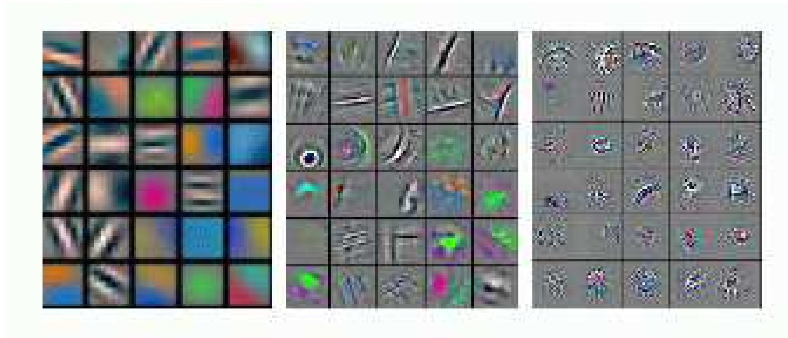
Convolutional Neural Networks



$$Z^i_{j,k} = g\left(b^i + \sum_l \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} K^i_{l,m,n} V^l_{j+m,k+n}\right)$$

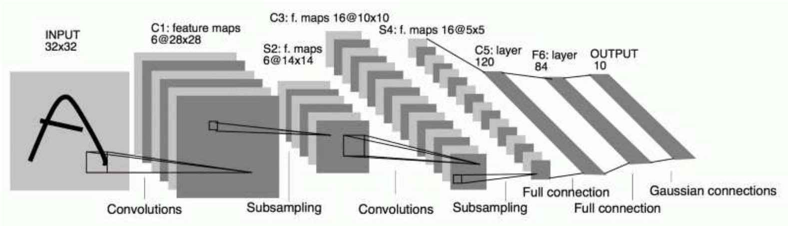
The same weights are applied to the next $M \times N$ block of inputs, to compute the next hidden unit in the convolution layer ("weight sharing").

Convolutional Filters



First Layer Second Layer Third Layer

LeNet trained on MNIST

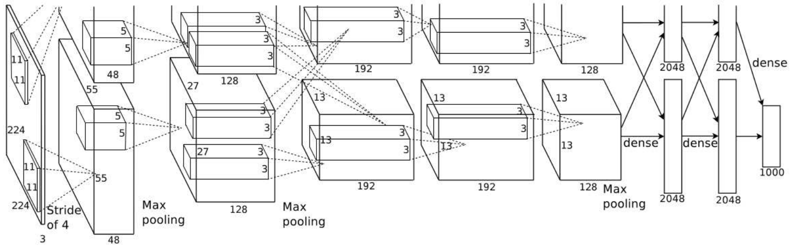


The 5×5 window of the first convolution layer extracts from the original 32×32 image a 28×28 array of features. Subsampling then halves this size to 14×14. The second Convolution layer uses another 5×5 window to extract a 10×10 array of features, which the second subsampling layer reduces to 5×5. These activations then pass through two fully connected layers into the 10 output units corresponding to the digits '0' to '9'.

ImageNet Architectures

- LeNet, 5 layers (1998)
- AlexNet, 8 layers (2012)
- VGG, 19 layers (2014)
- GoogleNet, 22 layers (2014)
- ResNets, 152 layers (2015)
- DenseNets, 160 layers (2017)

AlexNet Details

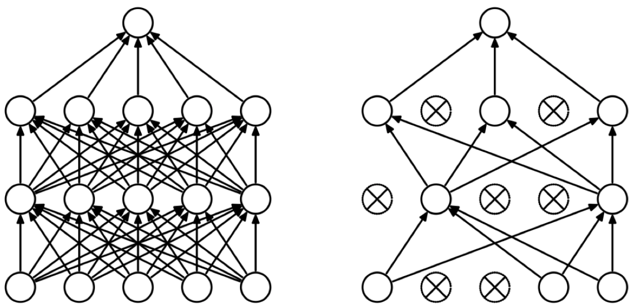


- 650K neurons
- 630M connections
- 60M parameters
- more parameters that images $\rightarrow$ danger of overfitting

Enhancements

- Rectified Linear Units (ReLUs)
- overlapping pooling (width $= 3$, stride $= 2$)
- stochastic gradient descent with momentum and weight decay
- data augmentation to reduce overfitting
- 50% dropout in the fully connected layers

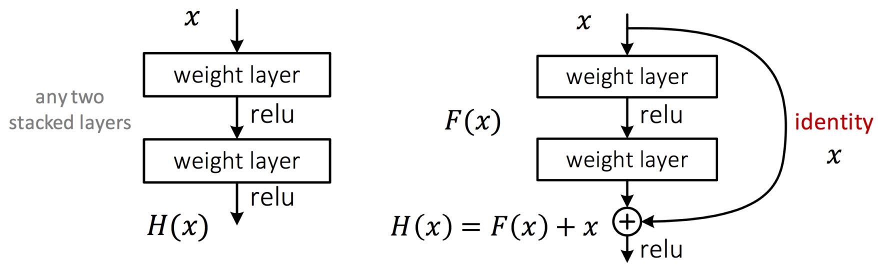
Dropout (7.12)



(a) Standard Neural Net (b) After applying dropout.

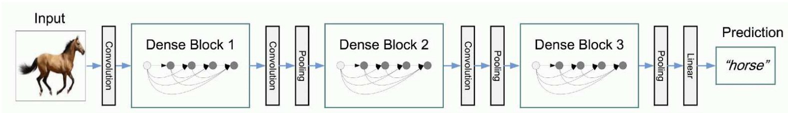
Nodes are randomly chosen to not be used, with some fixed probability (usually, one half).

Residual Networks



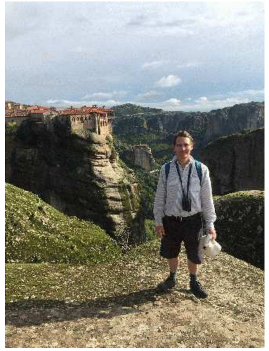
Idea: Take any two consecutive stacked layers in a deep network and add a "skip" connection which bipasses these layers and is added to their output.

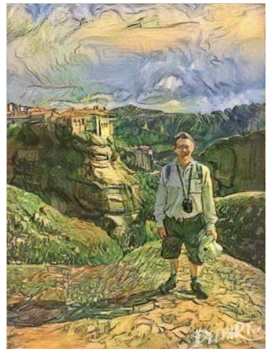
Dense Networks



Recently, good results have been achieved using networks with densely connected blocks, within which each layer is connected by shortcut connections to all the preceding layers.

Neural Style Transfer

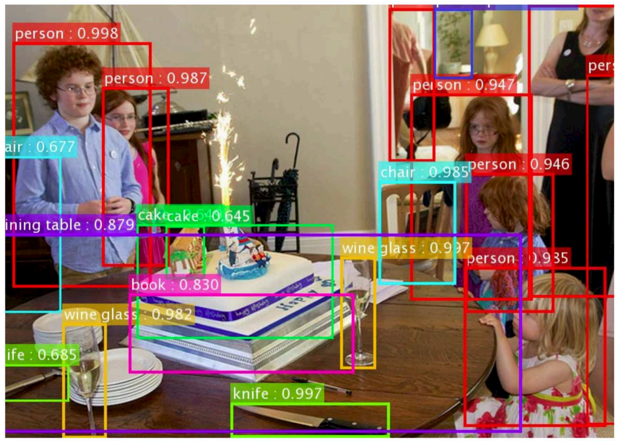




content + style → new image

Neural Style Transfer

Object Detection



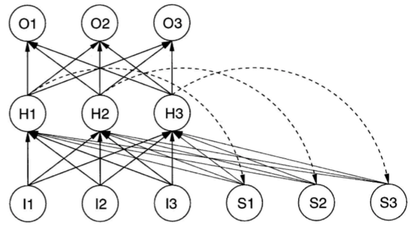
Processing Temporal Sequences

There are many tasks which require a sequence of inputs to be processed rather than a single input.

- speech recognition
- time series prediction
- machine translation
- handwriting recognition
- image captioning

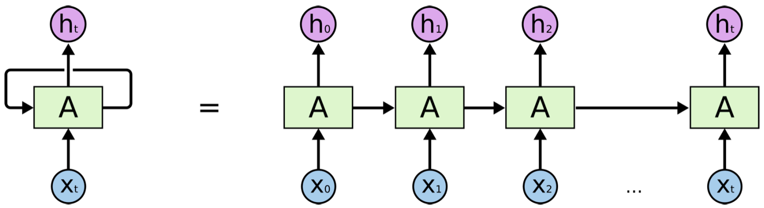
How can neural network models be adapted for these tasks?

Simple Recurrent Network (Elman, 1990)



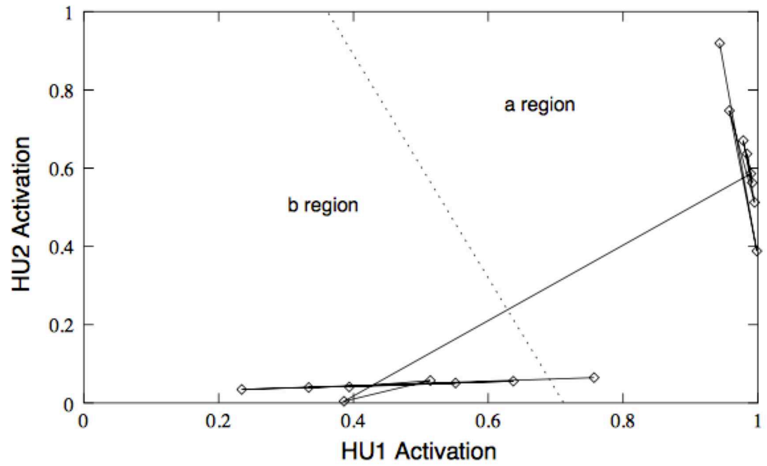
- at each time step, hidden layer activations are copied to “context” layer
- hidden layer receives connections from input and context layers
- the inputs are fed one at a time to the network, it uses the context layer to “remember” whatever information is required for it to produce the correct output

Back Propagation Through Time

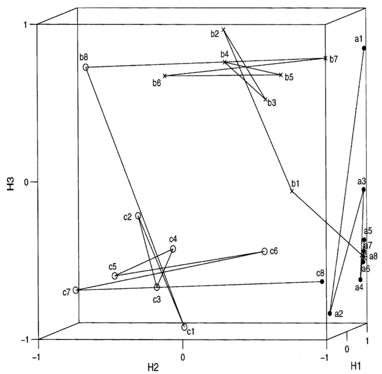


- we can “unroll” a recurrent architecture into an equivalent feedforward architecture, with shared weights
- applying backpropagation to the unrolled architecture is reffered to as “backpropagation through time”
- we can backpropagate just one timestep, or a fixed number of timesteps, or all the way back to beginning of the sequence

Oscillating Solution for $a^n b^n$

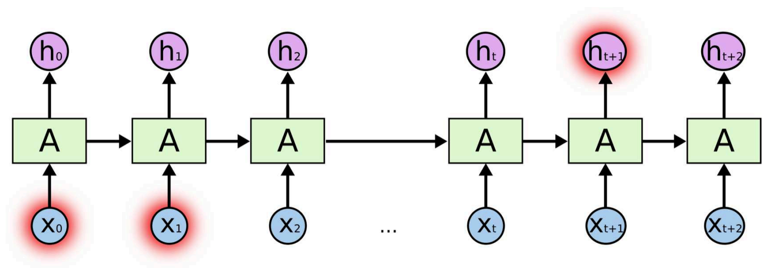


Hidden Unit Dynamics for $a^n b^n c^n$



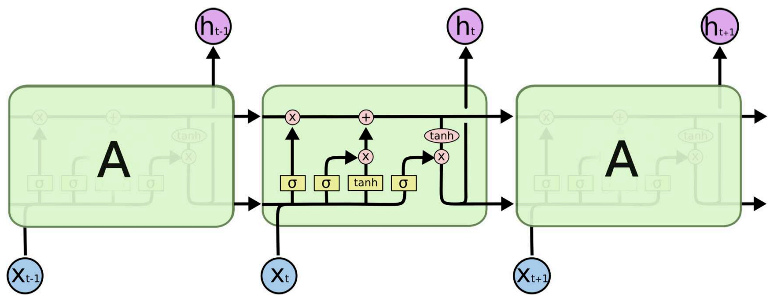
- SRN with 3 hidden units can learn to predict $a^n b^n c^n$ by counting up and down simultaneously in different directions, thus producing a star shape.

Long Range Dependencies



- Simple Recurrent Networks (SRNs) can learn medium-range dependencies but have difficulty learning long range dependencies
- Long Short Term Memory (LSTM) and Gated Recurrent Units (GRU) can learn long range dependencies better than SRN

Long Short Term Memory



LSTM – context layer is modulated by three gating mechanisms: forget gate, input gate and output gate.

`http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Statistical Language Processing

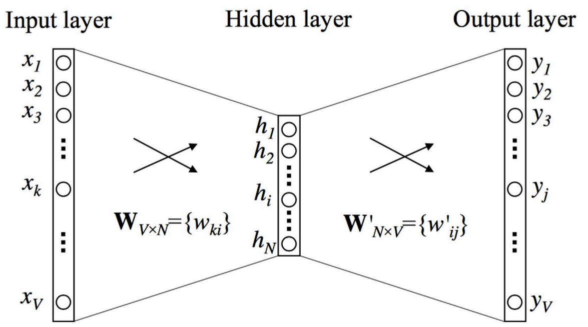
Synonyms for “elegant”

> stylish, graceful, tasteful, discerning, refined, sophisticated, dignified, cultivated, distinguished, classic, smart, fashionable, modish, decorous, beautiful, artistic, aesthetic, lovely; charming, polished, suave, urbane, cultured, dashing, debonair; luxurious, sumptuous, opulent, grand, plush, high-class, exquisite

Synonyms, antonyms and taxonomy require human effort, may be incomplete and require discrete choices. Nuances are lost. Words like “king”, “queen” can be similar in some attributes but opposite in others.

Could we instead extract some statistical properties automatically, without human involvement?

word2vec 1-Word Context Model



The k^{th} row $\mathbf{v}_k$ of $\mathbf{W}$ is a representation of word k.

The j^{th} column $\mathbf{v}'_j$ of $\mathbf{W}'$ is an (alternative) representation of word j.

If the (1-hot) input is k, the linear sum at each output will be $u_j = \mathbf{v}'^{\,\mathrm{T}}_j \mathbf{v}_k$

Linguistic Regularities

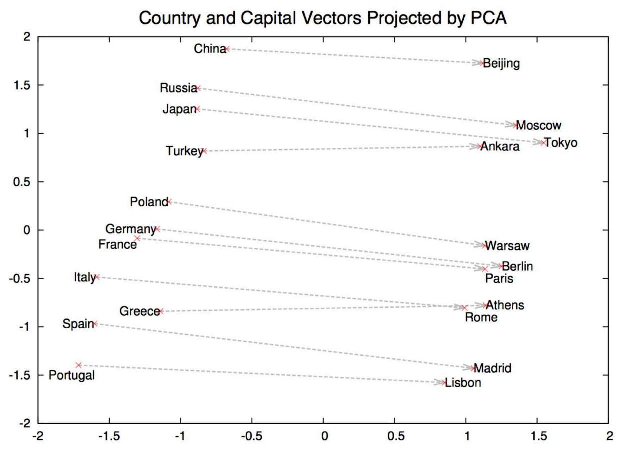
King + Woman - Man $\simeq$ Queen

More generally,

A is to B as C is to ??

$$d = \mathrm{argmax}_x \frac{(v_c + v_b - v_a)^{\mathrm{T}} v_x}{||v_c + v_b - v_a||}$$

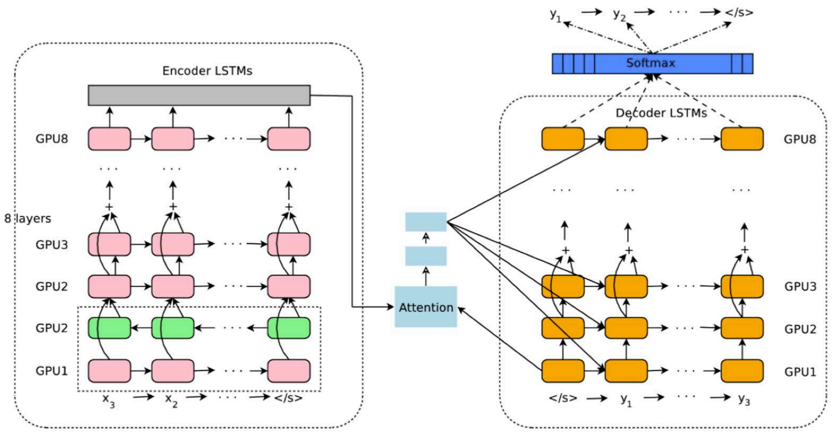
Capital Cities



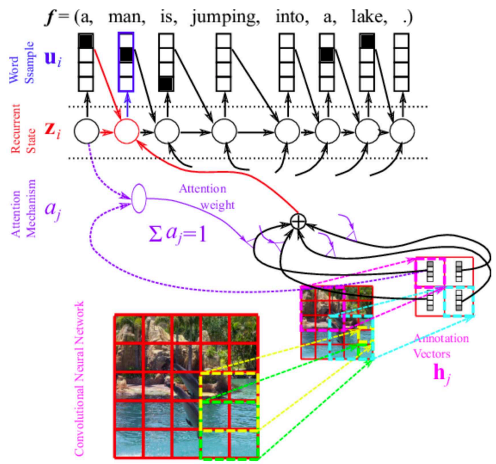
Word Relationships

Relationship	Example 1	Example 2	Example 3
France - Paris	Italy: Rome	Japan: Tokyo	Florida: Tallahassee
big - bigger	small: larger	cold: colder	quick: quicker
Miami - Florida	Baltimore: Maryland	Dallas: Texas	Kona: Hawaii
Einstein - scientist	Messi: midfielder	Mozart: violinist	Picasso: painter
Sarkozy - France	Berlusconi: Italy	Merkel: Germany	Koizumi: Japan
copper - Cu	zinc: Zn	gold: Au	uranium: plutonium
Berlusconi - Silvio	Sarkozy: Nicolas	Putin: Medvedev	Obama: Barack
Microsoft - Windows	Google: Android	IBM: Linux	Apple: iPhone
Microsoft - Ballmer	Google: Yahoo	IBM: McNealy	Apple: Jobs
Japan - sushi	Germany: bratwurst	France: tapas	USA: pizza

Google Neural Machine Translation



Captioning, with Attention



Reinforcement Learning Framework

- An agent interacts with its environment.
- There is a set $\mathcal{S}$ of *states* and a set $\mathcal{A}$ of *actions*.
- At each time step t, the agent is in some state s_t.
  It must choose an action a_t, whereupon it goes into state $s_{t+1} = \delta(s_t, a_t)$ and receives reward $r_t = \mathcal{R}(s_t, a_t)$
- Agent has a *policy* $\pi : \mathcal{S} \to \mathcal{A}$. We aim to find an optimal policy π^* which maximizes the cumulative reward.
- In general, δ, $\mathcal{R}$ and π can be multi-valued, with a random element, in which case we write them as probability distributions

$$\delta(s_{t+1} = s \,|\, s_t, a_t) \quad \mathcal{R}(r_t = r \,|\, s_t, a_t) \quad \pi(a_t = a \,|\, s_t)$$

Q-Learning

For a deterministic environment, π^*, Q^* and V^* are related by

$$\pi^*(s) = \text{argmax}_a\, Q^*(s,a)$$
$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma V^*(\delta(s,a))$$
$$V^*(s) = \max_b Q^*(s,b)$$

So

$$Q^*(s,a) = \mathcal{R}(s,a) + \gamma \max_b Q^*(\delta(s,a),b)$$

This allows us to iteratively approximate Q by

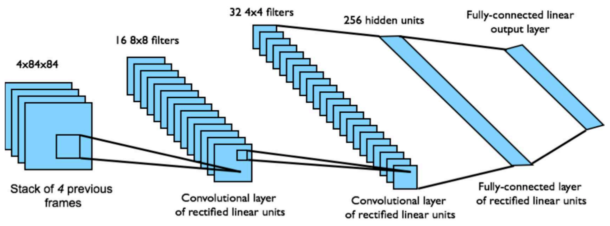
$$Q(s_t,a_t) \leftarrow r_t + \gamma \max_b Q(s_{t+1},b)$$

If the environment is stochastic, we instead write

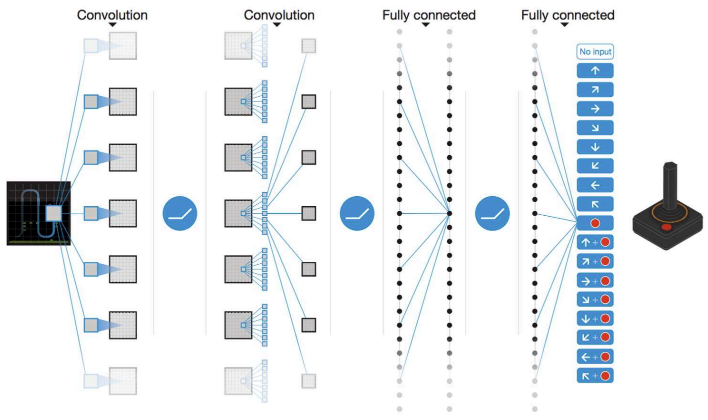
$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \eta\,[r_t + \gamma \max_b Q(s_{t+1},b) - Q(s_t,a_t)]$$

Deep Q-Learning for Atari Games

- end-to-end learning of values $Q(s,a)$ from pixels s
- input state s is stack of raw pixels from last 4 frames
  - 8-bit RGB images, 210×160 pixels
- output is $Q(s,a)$ for 18 joystick/button positions
- reward is change in score for that timestep



Deep Q-Network



Asynchronous Advantage Actor Critic

- use policy network to choose actions
- learn a parameterized Value function $V_u(s)$ by TD-Learning
- estimate Q-value by n-step sample

$$Q(s_t,a_t) = r_{t+1} + \gamma r_{t+2} + \ldots + \gamma^{n-1} r_{t+n} + \gamma^n V_u(s_{t+n})$$

- update policy by

$$\theta \leftarrow \theta + \eta_\theta [Q(s_t,a_t) - V_u(s_t)] \nabla_\theta \log \pi_\theta(a_t \mid s_t)$$

- update Value function my minimizing

$$[Q(s_t,a_t) - V_u(s_t)]^2$$

Other Deep Learning Topics

- Hopfield Networks
- Restricted Boltzmann Machines
- Autoencoders
- Generative Adversarial Networks