Introduction to Deep Learning

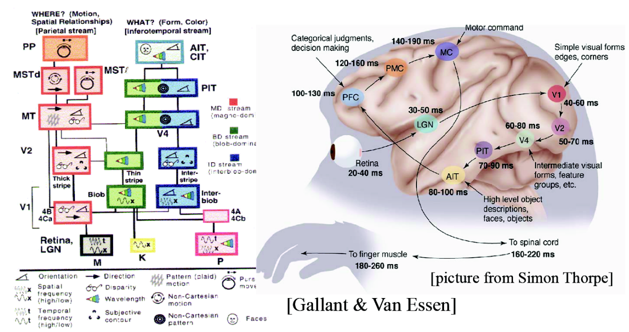
625 Lecture, Spring 2020

Yoonsuck Choe, Ph.D.

Professor, Texas A&M University

What Is Deep Learning?

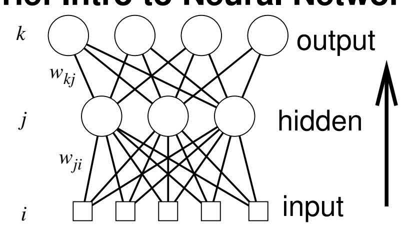
- Learning higher level abstractions/representations from data.
- Motivation: how the brain represents and processes sensory information in a hierarchical manner.
  - The ventral (recognition) pathway in the visual cortex has multiple stages
  - Retina - LGN - V1 - V2 - V4 - PIT - AIT ....

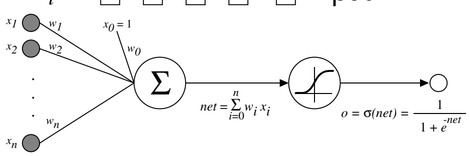


[Gallant & Van Essen] [picture from Simon Thorpe]

From LeCun's Deep Learning Tutorial

Brief Intro to Neural Networks



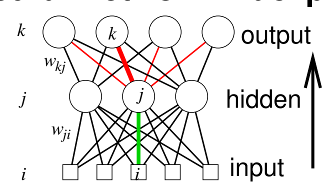


Deep learning is based on neural networks.

- Weighted sum followed by nonlinear activation function.
- Weights changed w/ *gradient descent* (η = learning rate, E=err):

$$w_{ij} \leftarrow w_{ij} - \eta \frac{\partial E}{\partial w_{ij}}$$

Intro to Neural Network: Backpropagation



Weight w_{ji} is updated as: $w_{ji} \leftarrow w_{ji} + \eta \delta_j a_i$, where

- a_i : activity at input side of weight w_{ji}.
- Hidden to output weights (thick red weight). T_k is target value.

$$\delta_k = (T_k - a_k)\sigma'(net_k)$$

- Deeper weights (green line in figure above).

$$\delta_j = \left[\sum_k w_{kj}\delta_k\right]\sigma'(net_j)$$

What Neurons Do in a Neural Network

Two points of view (both are valid):

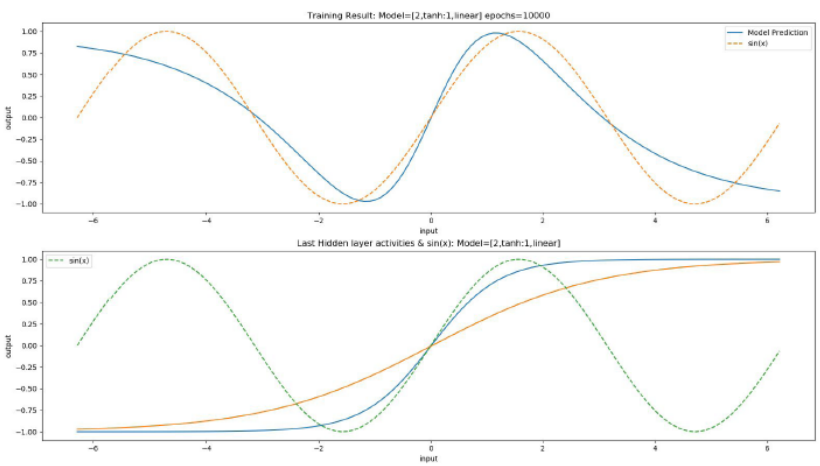
- Function approximation
- Decision boundary

* Represent input features – more on this later.

Function Approximation

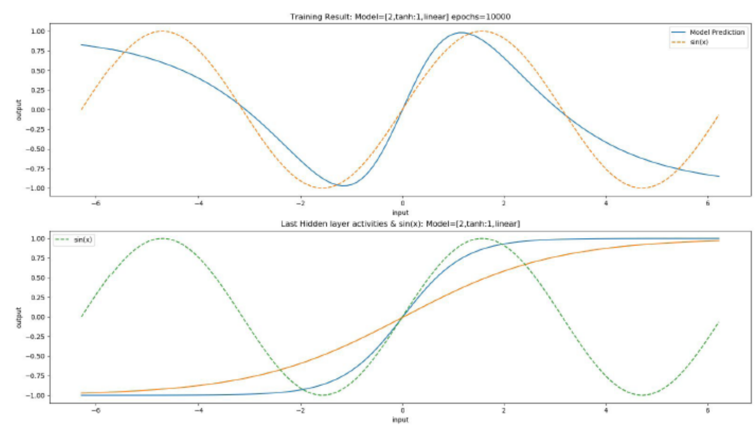
- Assume one input unit (scalar value).
- Depending on # of hidden layers, # of hidden units, etc., function with any complex shape can be learned. Ex: $y = \sin(x)$.

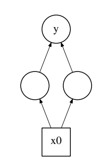
Example: $y = \sin(x)$



- Top: $\sin(x)$ nnet: Model=[# of units, activation func, [next layer spec], ...]
- Bottom: $\sin(x)$ vs. the hidden unit's output of last hidden layer.

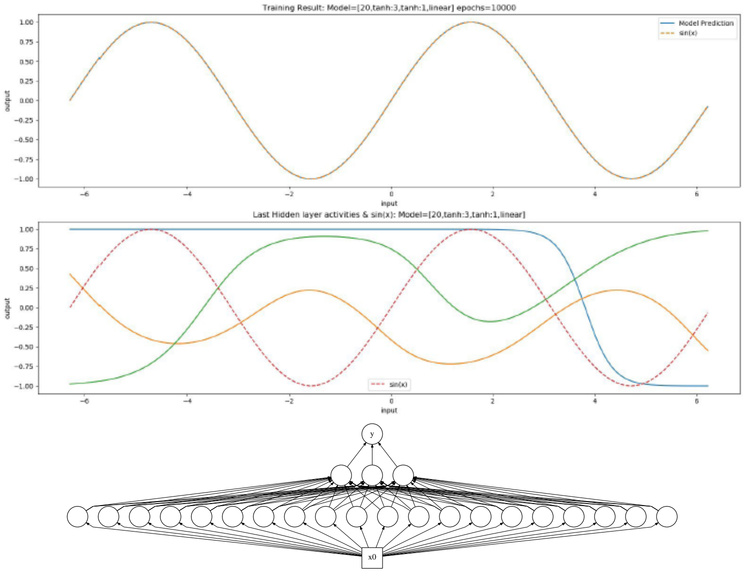
Ex: $y = \sin(x)$ Model=[2,tanh:1,linear]





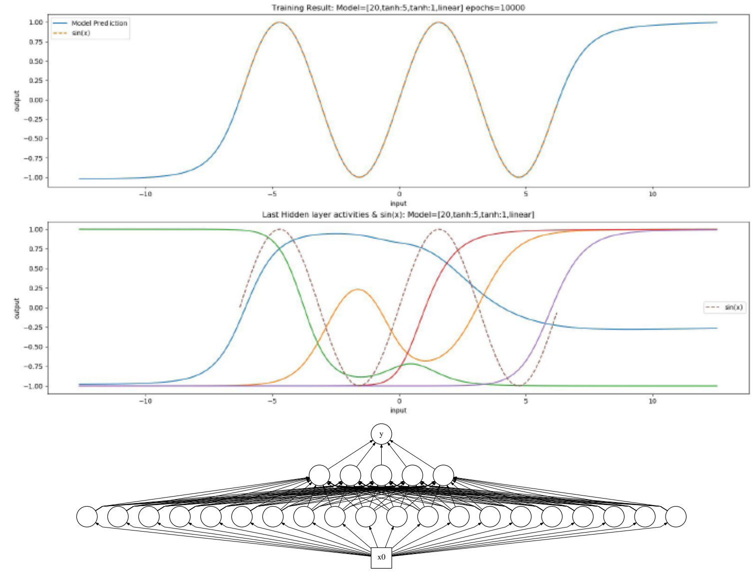
- One hidden layer with 2 units, One output unit. [2,tanh:1,linear]
- Bottom plot: Hidden neurons represent sigmoids.
- Top plot: Output unit is a linear combination of two sigmoids.

Ex: $y = \sin(x)$ Model=[20,tanh:3,tanh:1,linear]



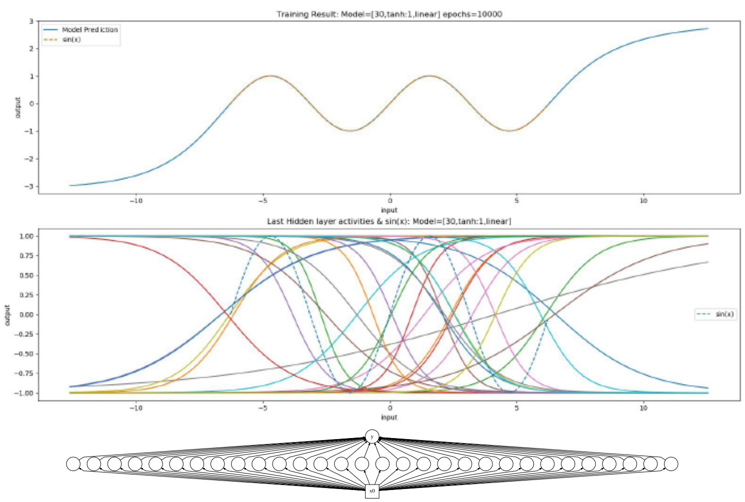
- 2nd hidden layer represents linear combination of 20 sigmoids.

Ex: $y = \sin(x)$ Model=[20,tanh:5,tanh:1,linear]



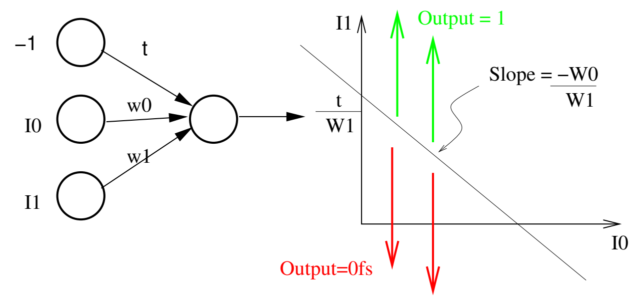
- Out-of-range inputs illustrate the limitation of DL.

Ex: $y = \sin(x)$ Model=[30,tanh:1,linear]



- Does a single hidden layer suffice? – Yes, with enough neurons.

Decision Boundary



Perceptrons (step function activation) can only represent **linearly separable** functions.

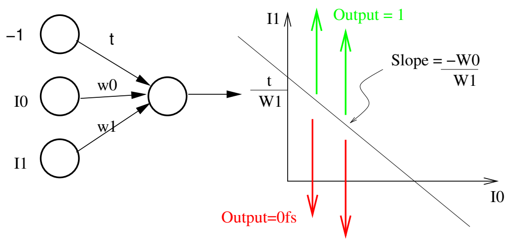
- Output of the perceptron:

$$W_0 \times I_0 + W_1 \times I_1 - t > 0, \text{ then output is } 1$$

$$W_0 \times I_0 + W_1 \times I_1 - t \leq 0, \text{ then output is } -1$$

If activation function is sigmoid, decision is a smooth ramp.

Decision Boundary



- Rearranging

$$W_0 \times I_0 + W_1 \times I_1 - t > 0, \text{ then output is } 1,$$

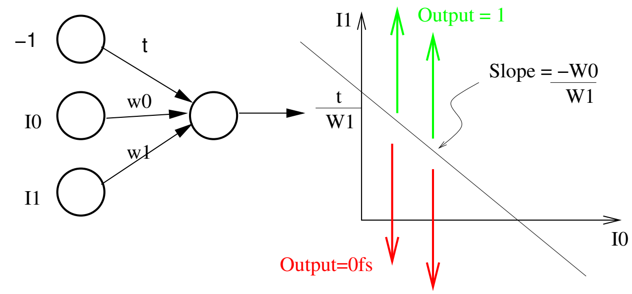
we get (if $W_1 > 0$)

$$I_1 > \frac{-W_0}{W_1} \times I_0 + \frac{t}{W_1},$$

where points above the line, the output is 1, and -1 for those below the line. Compare with

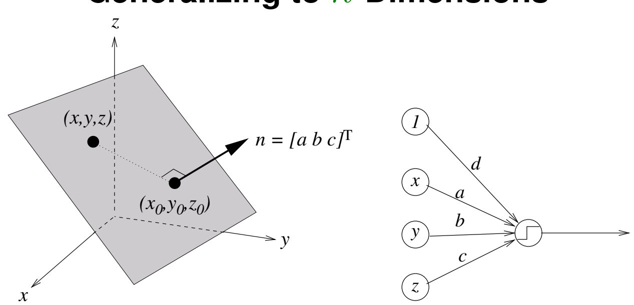
$$y = \frac{-W_0}{W_1} \times x + \frac{t}{W_1}.$$

Limitation of Perceptrons



- Only functions where the -1 points and 1 points are clearly separable can be represented by perceptrons.
- The geometric interpretation is generalizable to functions of n arguments, i.e. perceptron with n inputs plus one threshold (or bias) unit.

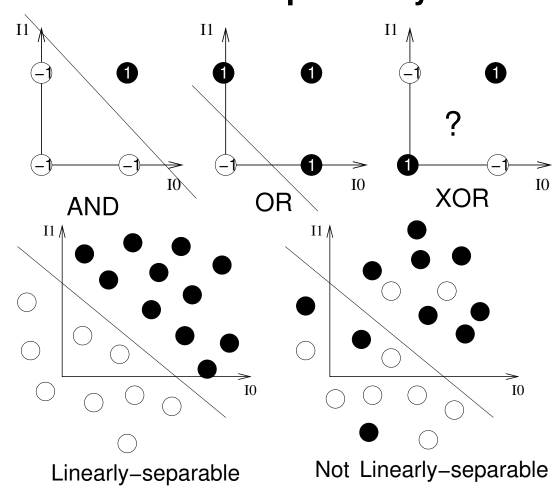
Generalizing to n-Dimensions



http://mathworld.wolfram.com/Plane.html

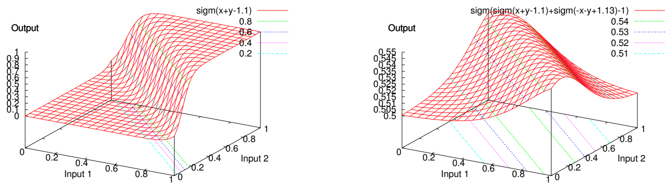
- $\vec{n} = (a, b, c), \vec{x} = (x, y, z), \vec{x_0} = (x_0, y_0, z_0)$.
- Equation of the plane: $\vec{n} \cdot (\vec{x} - \vec{x_0}) = 0$
- In short, $ax + by + cz + d = 0$, where a, b, c can serve as the weight, and $d = -\vec{n} \cdot \vec{x_0}$ as the bias.
- For n-D input space, the decision boundary becomes a $(n-1)$-D hyperplane (1-D less than the input space).

Linear Separability



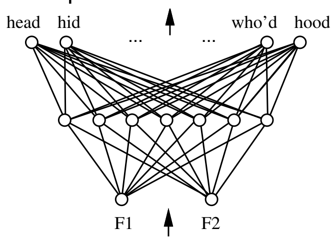
- Functions/Inputs that can or cannot be separated by a linear boundary.

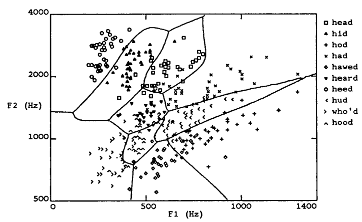
Decision Boundary in Multilayer Networks



(a) One output (b) Two hidden, one output

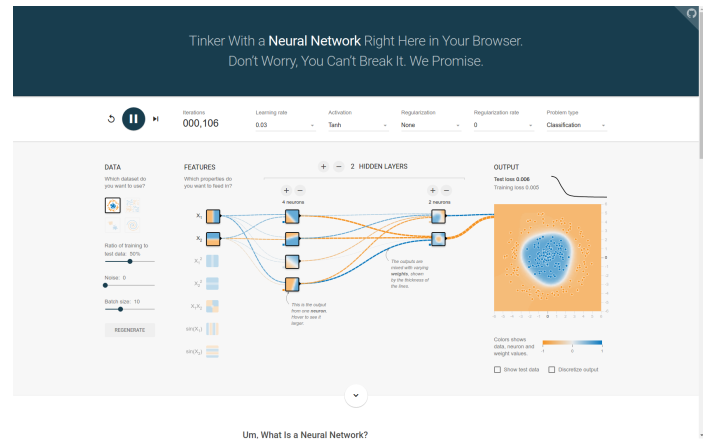
- Example: XOR





- Multiple decision regions.

Decision Boundary Demo with Tensorflow Playground



- http://playground.tensorflow.org

Deep Learning

- Complex models with large number of parameters
  - Hierarchical representations
  - More parameters = more accurate on training data
  - Simple learning rule for training (gradient-based).
- Lots of data
  - Needed to get better generalization performance.
  - High-dimensional input need exponentially many inputs (curse of dimensionality).
- Lots of computing power: GPGPU, etc.
  - Training large networks can be time consuming.

Deep Learning, in the Context of AI/ML

Deep Learning: Automating Feature Discovery

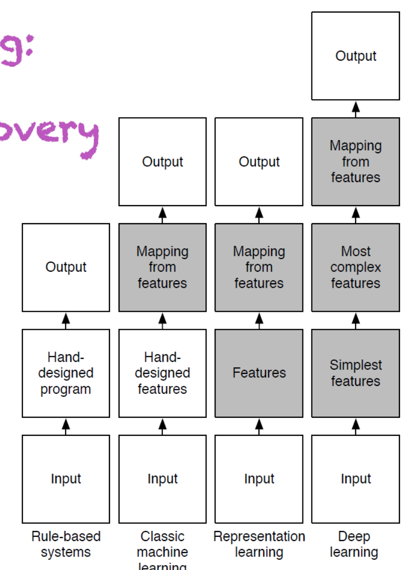


Fig: I. Goodfellow

From LeCun's Deep Learning Tutorial

The Rise of Deep Learning

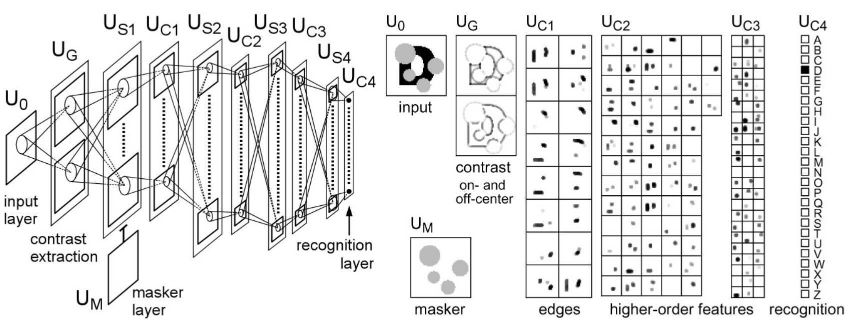
Made popular in recent years

- Geoffrey Hinton et al. (2006).
- Andrew Ng & Jeff Dean (Google Brain team, 2012).
- Schmidhuber et al.'s deep neural networks (won many competitions and in some cases showed super human performance; 2011–). Recurrent neural networks using LSTM (Long Short-Term Memory).
- Google Deep Mind: Atari 2600 games (2015), AlphaGo (2016).
- ICLR, International Conference on Learning Representations: First meeting in 2013.

Long History (in Hind Sight)

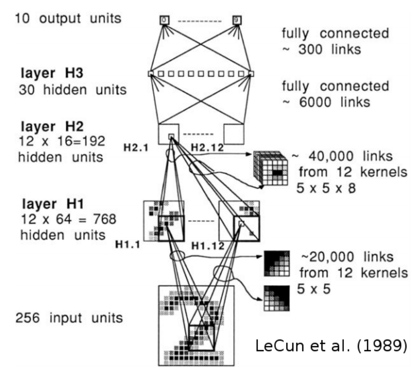
- Fukushima's Neocognitron (1980).
- LeCun et al.'s Convolutional neural networks (1989).
- Schmidhuber's work on stacked recurrent neural networks (1993). Vanishing gradient problem.
- See Schmidhuber's extended review: Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61, 85-117.

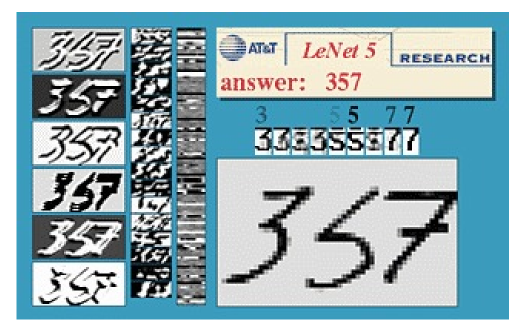
History: Fukushima's Neocognitron



- Appeared in journal *Biological Cybernetics* (1980).
- Multiple layers with local receptive fields.
- S cells (trainable) and C cells (fixed weight).
- Deformation-resistent recognition.

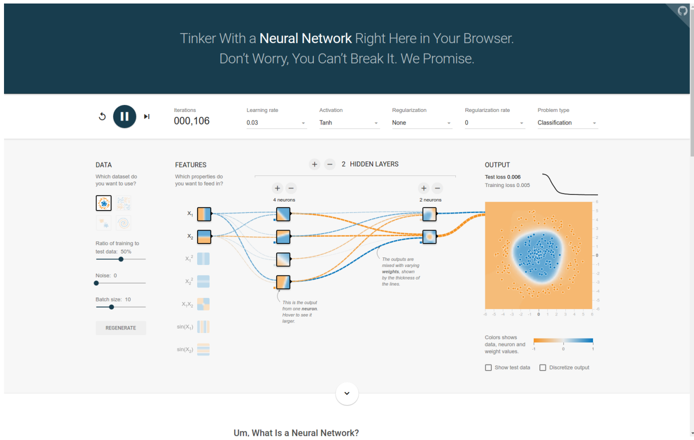
History: LeCun's Colvolutional Neural Nets





- Convolution kernel (weight sharing) + Subsampling
- Fully connected layers near the end.
- Became a main-stream method in deep learning.

Motivating Deep Learning: Tensorflow Demo



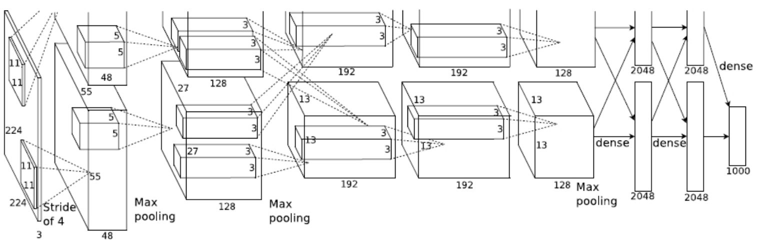
- http://playground.tensorflow.org
- Demo to explore why deep nnet is powerful and how it is limited.

Current Trends

Focusing on ground-breaking works in Deep Learning:

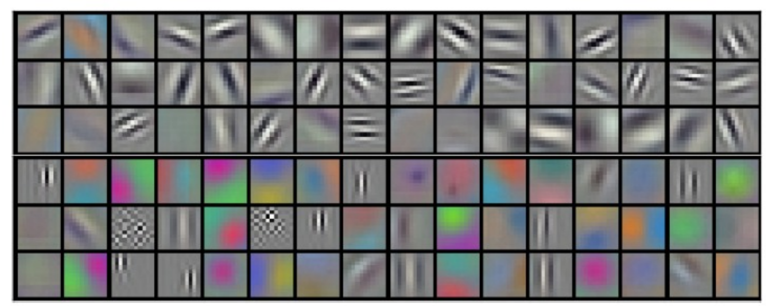
- Convolutional neural networks
- Deep Q-learning Network (extensions to reinforcement learning)
- Deep recurrent neural networks using (LSTM)
- Applications to diverse domains.
  - Vision, speech, video, NLP, etc.
- Lots of open source tools available.

Deep Convolutional Neural Networks (1)



- Krizhevsky et al. (2012)
- Applied to ImageNet competition (1.2 million images, 1,000 classes).
- Network: 60 million parameters and 650,000 neurons.
- Top-1 and top-5 error rates of 37.5% and 17.0%.
- Trained with backprop.

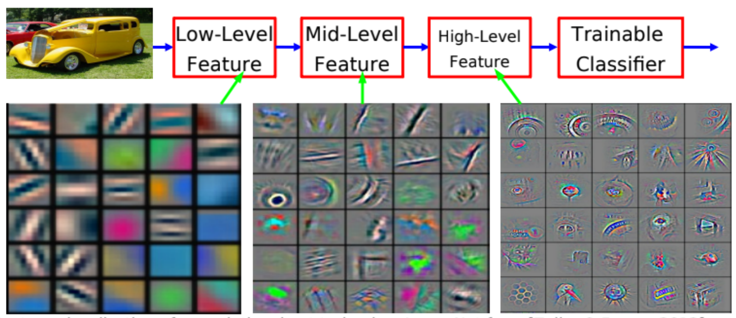
Deep Convolutional Neural Networks (2)



- Learned kernels (first convolutional layer).
- Resembles mammalian RFs: oriented Gabor patterns, color opponency (red-green, blue-yellow).

Deep Convolutional Neural Networks (3)

Natural is data is compositional => it is efficiently representable hierarchically

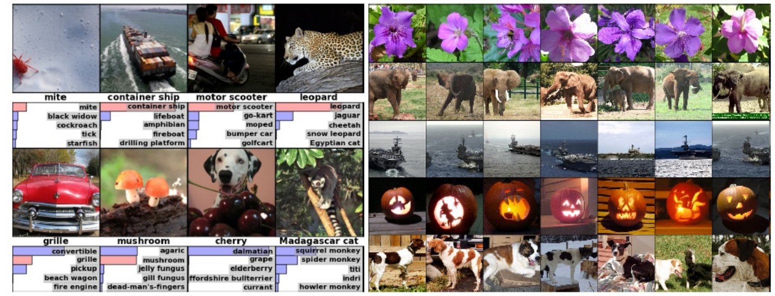


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

- Higher layers represent progressively more complex features.

* From Yann LeCun's Harvard lecture (2019)

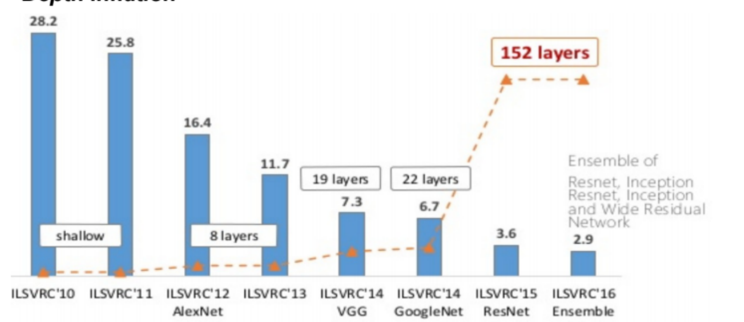
Deep Convolutional Neural Networks (4)



- Left: Bold = correct label. 5 ranked labels: model's estimation.
- Right: Test (1st column) vs. training images with closest hidden representation to the test data.

Deep Convolutional Neural Networks (5)

Depth inflation

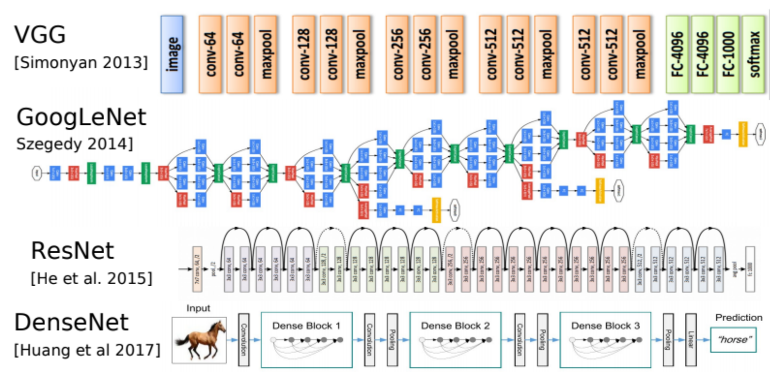


(Figure: Anirudh Koul)

- Depth inflation: Deeper is better!

* From Yann LeCun's Harvard lecture (2019)

Deep Convolutional Neural Networks (6)

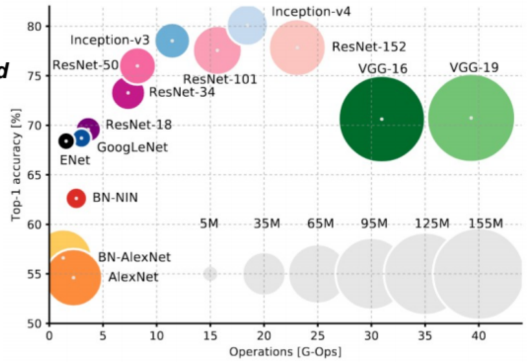


- Not just depth but architecture also matters!

* From Yann LeCun's Harvard lecture (2019)

Deep Convolutional Neural Networks (7)

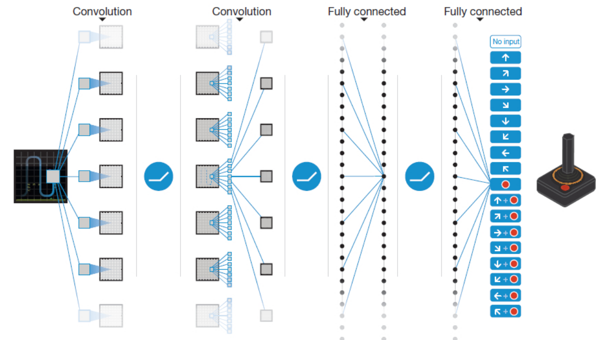
- ***[Canziani 2016]***
- ***ResNet50 and ResNet 100 are used routinely in production.***



- Computation vs. performance

* From Yann LeCun's Harvard lecture (2019)

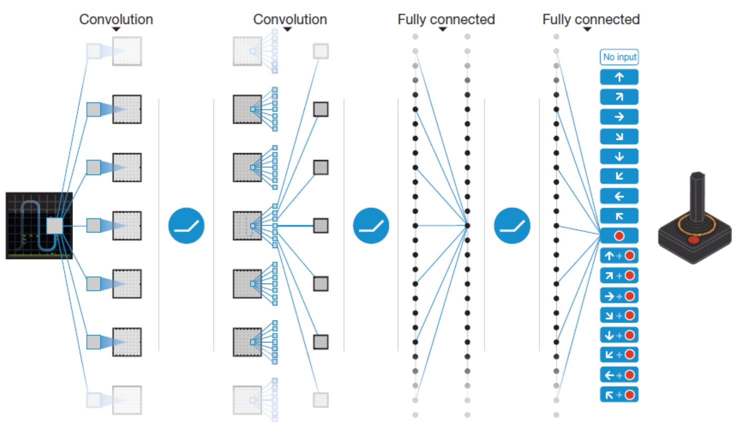
Deep Q-Network (DQN)



Google Deep Mind (Mnih et al. *Nature* 2015).

- Latest application of deep learning to a *reinforcement learning* domain (Q as in Q-learning).
- Applied to *Atari 2600* video game playing.

DQN Overview



- Input: video screen; Output: $Q(s, a)$; Reward: game score.
- $Q(s, a)$: action-value function
  - Value of taking action a when in state s.

DQN Overview

- Input preprocessing
- Experience replay (collect and replay state, action, reward, and resulting state)
- Delayed (periodic) update of Q.
- Moving target $\hat{Q}$ value used to compute error (loss function L, parameterized by weights θ_i).
  - Gradient descent:

$$\frac{\partial L}{\partial \theta_i}$$

DQN Algorithm

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^- = \theta$

For episode $= 1, M$ **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For $t = 1$,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \operatorname{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^-\right) & \text{otherwise} \end{cases}$

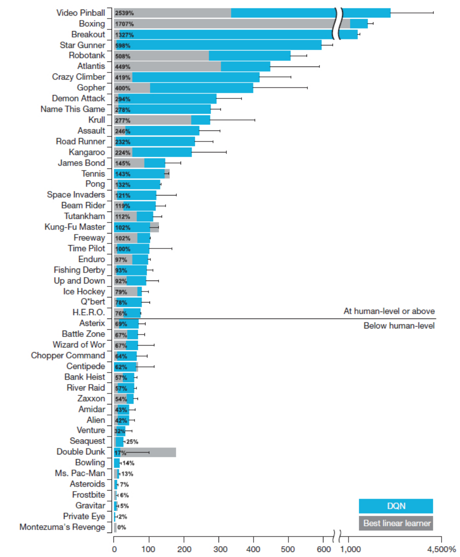
    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

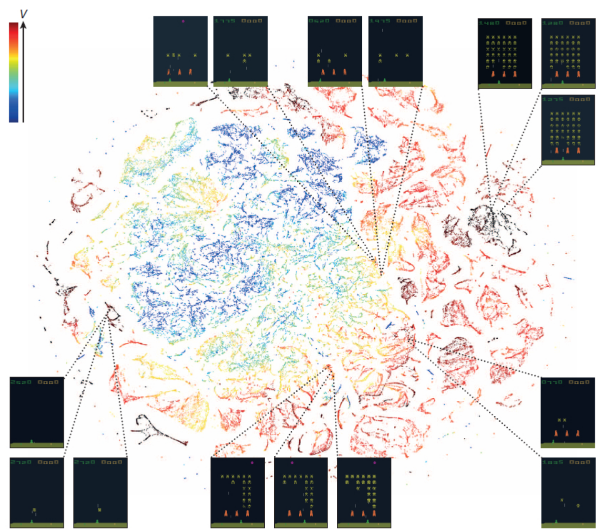
End For

DQN Results



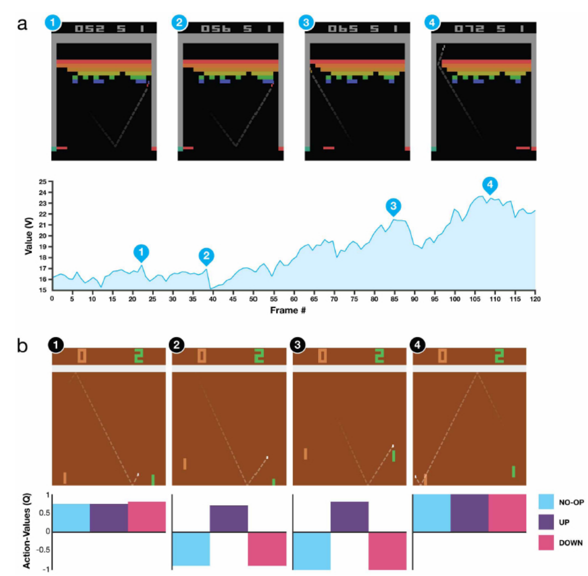
- Superhuman performance on over half of the games.

DQN Hidden Layer Representation (t-SNE map)



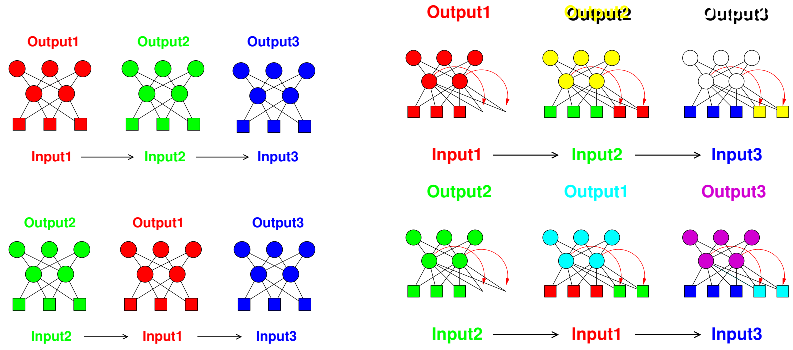
- Similar perception, similar reward clustered.

DQN Operation



- Value vs. game state; Game state vs. action value.

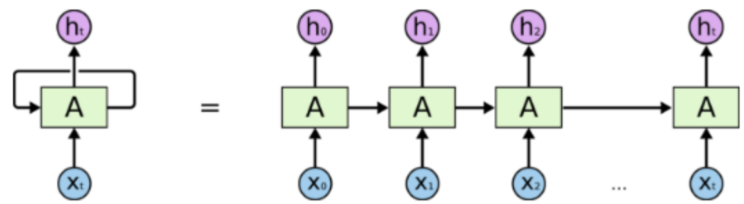
Deep Recurrent Neural Networks



Feedforward **Recurrent**

- Feedforward networks: No memory of past input.
- Recurrent networks:
  - Good: Past input affects present output.
  - Bad: Cannot remember far into the past.

RNN Training: Backprop in Time



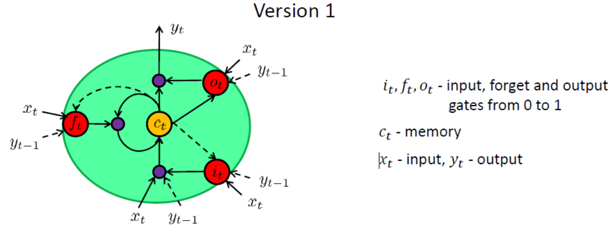
An unrolled recurrent neural network.

- Can unfold recurrent loop: Make it into a feedforward net.
- Use the same backprop algorithm for training.
- Again, cannot remember too far into the past.

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

Long Short-Term Memory

Version 1

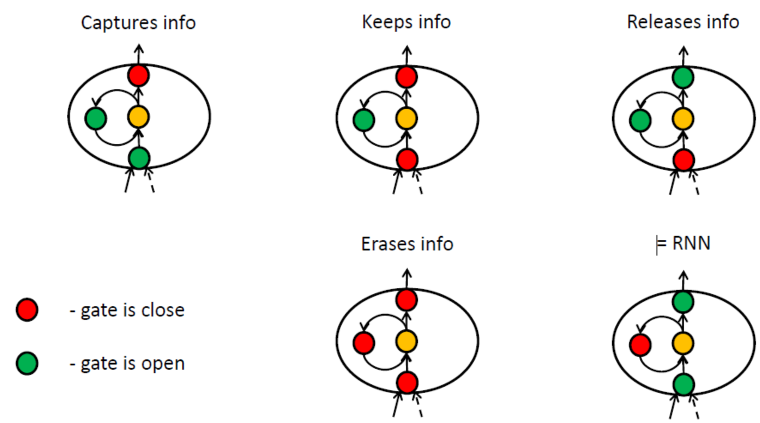


$$i_t = \sigma(w_{ix}x_t + w_{ic}c_{t-1} + w_{iy}y_{t-1} + b_i)$$
$$f_t = \sigma(w_{fx}x_t + w_{fc}c_{t-1} + w_{fy}y_{t-1} + b_f)$$
$$o_t = \sigma(w_{ox}x_t + w_{oc}c_t + w_{oy}y_{t-1} + b_o)$$
$$c_t = f_t c_{t-1} + i_t \cdot tanh(w_{cx}x_t + w_{cy}y_{t-1}) \qquad y_t = o_t \cdot tanh(c_t)$$

- LSTM to the rescue (Hochreiter and Schmidhuber, 1997).
- Built-in recurrent memory that can be written (Input gate), reset (Forget gate), and outputted (Output gate).

From http://www.machinelearning.ru/wiki/images/6/6c/RNN_and_LSTM_16102015.pdf

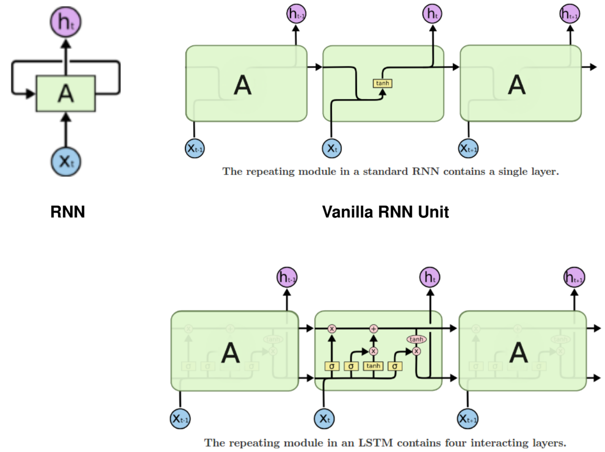
Long Short-Term Memory



- Long-term retention possible with LSTM.

From http://www.machinelearning.ru/wiki/images/6/6c/RNN_and_LSTM_16102015.pdf

Long Short-Term Memory in Action



RNN **Vanilla RNN Unit**

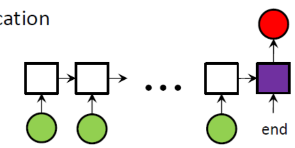
LSTM Unit

- Unfold in time and use backprop as usual.

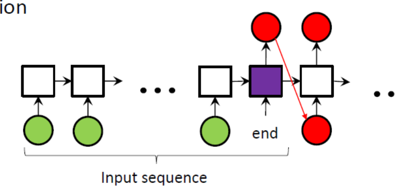
Fig from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Applications

- Sequence classification



- Sequence translation



- Applications: Sequence classification, Sequence translation.

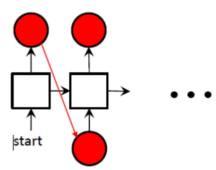
From http://machinelearning.ru

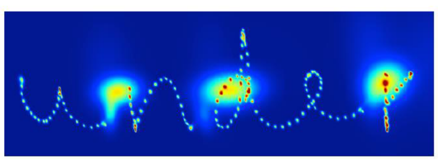
LSTM Applications

handwriting -> handwriting

Next pen position (we predict parameters):
x1,x2 - mixture of bivariate Gaussians
x3 - Bernoulli distribution

Current pen position:
x1,x2 – pen offset
x3 – is it end of the stroke





- Applications: Sequence prediction

From http://machinelearning.ru

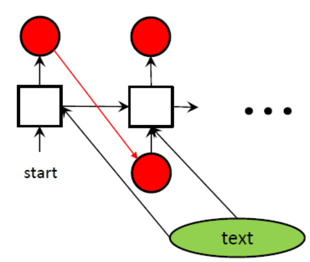
LSTM Applications

text -> handwriting

Next pen position

Current pen position

Which letter we write now



- Applications: Sequence classification, Sequence prediction, Sequence translation.

From http://machinelearning.ru

Deep Learning Applications: Vision

- Give the name of the dominant object in the image
- Top-5 error rates: if correct class is not in top 5, count as error
  - Red:ConvNet, blue: no ConvNet

2012 Teams	%error
Supervision (Toronto)	15.3
ISI (Tokyo)	26.1
VGG (Oxford)	26.9
XRCE/INRIA	27.0
UvA (Amsterdam)	29.6
INRIA/LEAR	33.4

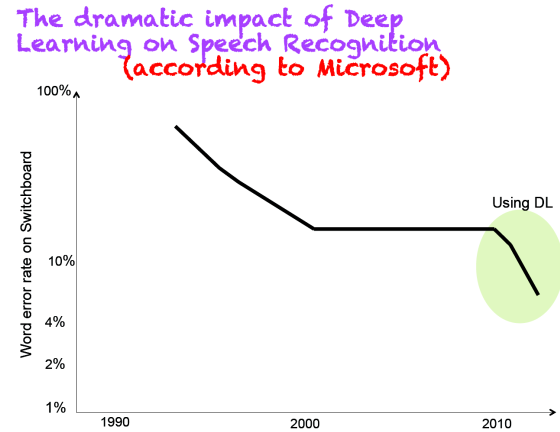
2013 Teams	%error
Clarifai (NYU spinoff)	11.7
NUS (singapore)	12.9
Zeiler-Fergus (NYU)	13.5
A. Howard	13.5
OverFeat (NYU)	14.1
UvA (Amsterdam)	14.2
Adobe	15.2
VGG (Oxford)	15.2
VGG (Oxford)	23.0

2014 Teams	%error
GoogLeNet	6.6
VGG (Oxford)	7.3
MSRA	8.0
A. Howard	8.1
DeeperVision	9.5
NUS-BST	9.7
TTIC-ECP	10.2
XYZ	11.2
UvA	12.1

- ConvNet sweeping image recognition challenges.

From LeCun's Deep Learning Tutorial

Deep Learning Applications: Speech

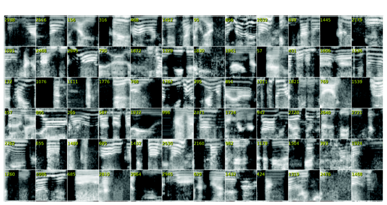


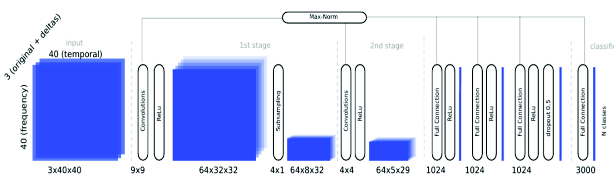
- Deep learning led to major improvement in speech recognition.

From LeCun's Deep Learning Tutorial

Deep Learning Applications: Speech

Training samples.
- 40 MEL-frequency Cepstral Coefficients
- Window: 40 frames, 10ms each



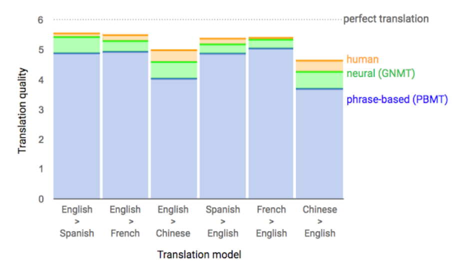


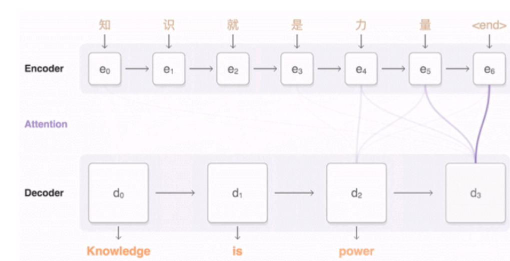
- Acoustic Model: ConvNet with 7 layers. 54.4 million parameters.
- Classifies acoustic signal into 3000 context-dependent subphones categories
- ReLU units + dropout for last layers
- Trained on GPU. 4 days of training

- ConvNet can also be applied to speech recognition.
- Use spectrogram and treat it like a 2D image.
- SOTA: end-to-end attention-based RNN (w/ LSTM, GRU, ...)

From LeCun's Deep Learning Tutorial

Deep Learing Applications: NLP

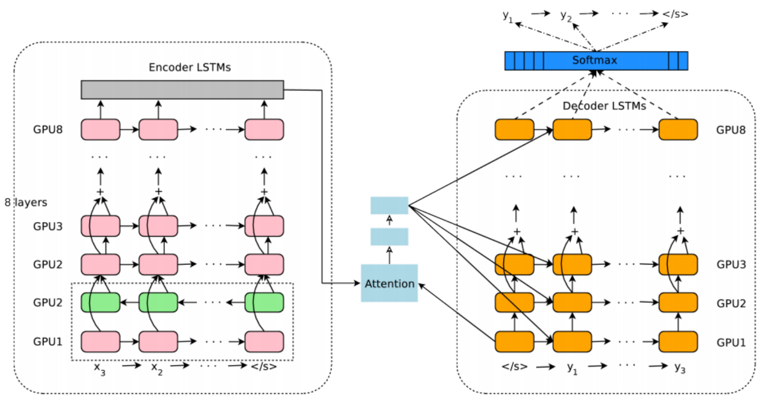




- Based on encoding/decoding and attention.

From https://research.googleblog.com/2016/09/a-neural-network-for-machine.html

Deep Learing Applications: NLP

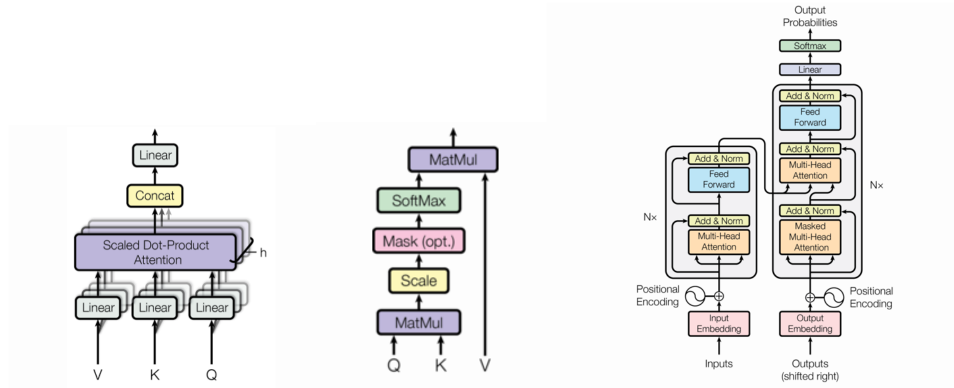


- Google's LSTM-based machine translation.

Wu et al. *arXiv:1609.08144* (2016).

How attention works: https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attent:

Deep Learning for NLP: Transformers

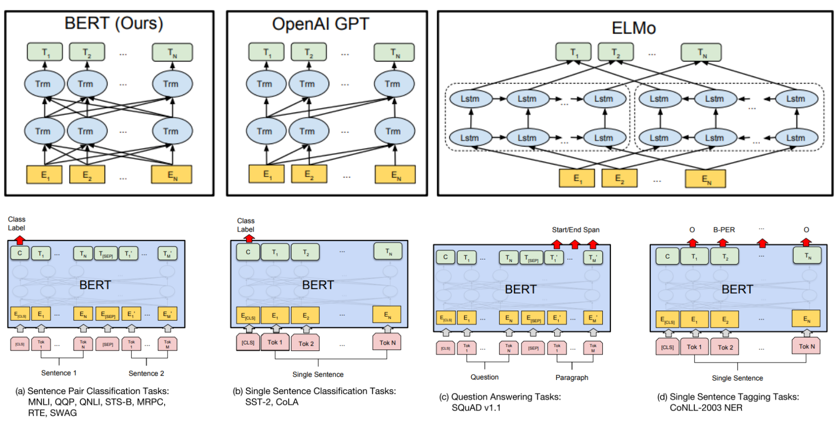


Multihead Self-attention Scaled Dot-Product Attention Transformer

- Highly parallelizable, Reduces serial computation
- Multi-head self-attention + position-encoding/position-wise FFW
- Organized over Query, Key, Value (Q,K,V)

https://medium.com/@adityathiruvengadam/transformer-architecture-attention-is-all-you-need-aeccd9f50d09

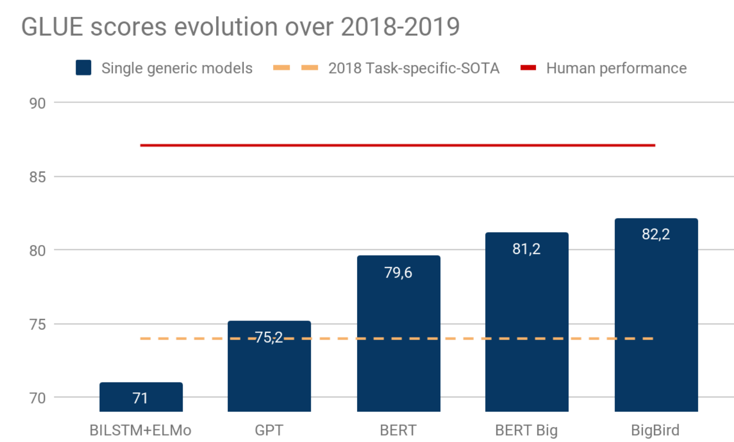
Deep Learning for NLP: Transformers & BERT



from Devlin et al. 2018

- BERT, based on Transformer: Powerful new approach for NLP

Deep Learning for NLP: Transformers & BERT



- Transformer-based NLP led to big leap in performance.

https://medium.com/synapse-dev/understanding-bert-transformer-attention-isnt-all-you-need-5839ebd396db

Limitations of Deep Learning

- Requires massive amounts of (labeled) data.
- Long training time. Large trained models.
- Catastrophic forgetting.
- Designing good model is done mostly manually.
- Vulnerable to adversarial inputs.
- Hard to explain how it works / what it learned.

Overcoming Limitations of DL

Pretty much well known problems, and solutions emerging.

- Data: Active learning, Core sets, data augmentation, etc.
- Computing time: Train with reduced data. Compact models.
- Large trained models: Compression, distillation
- Catastrophic forgetting: Various approaches, not perfect yet.
- Issue of manual design: AutoML, NAS, ENAS, Evolution, etc.
- Adversarial inputs: Adversarial training, defensive distillation, ...
- Explainability: DARPA XAI effort - explanation generation, Bayesian program induction, semantic associations, etc.

Advanced/Fundamental Issues in Deep Learning

- Reasoning, Common-sense reasoning
- Unsupervised, self-supervised learning
- Human-like learning
- Meaning/semantic-level processing
- Problem posing, Coping with new tasks
- Tool construction and tool use

Summary

- Deep convolutional networks: High computational demand, over the board great performance.
- Deep Q-Network: unique apporach to reinforcement learning. End-to-end machine learning. Super-human performance.
- Deep recurrent neural networks: sequence learning. LSTM is a powerful mechanism.
- Diverse applications. Top performance.
- Lots of practical and fundamental limits
- Flood of deep learning tools available.