CSCE 421: Machine Learning

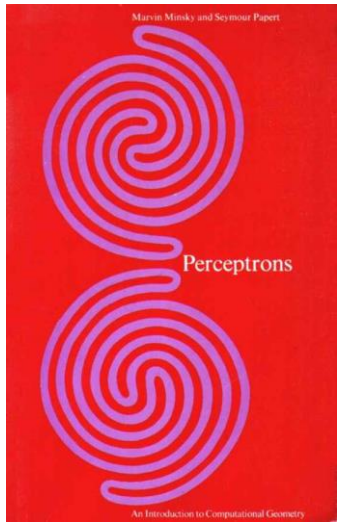
Lecture 22: Deep Learning Models

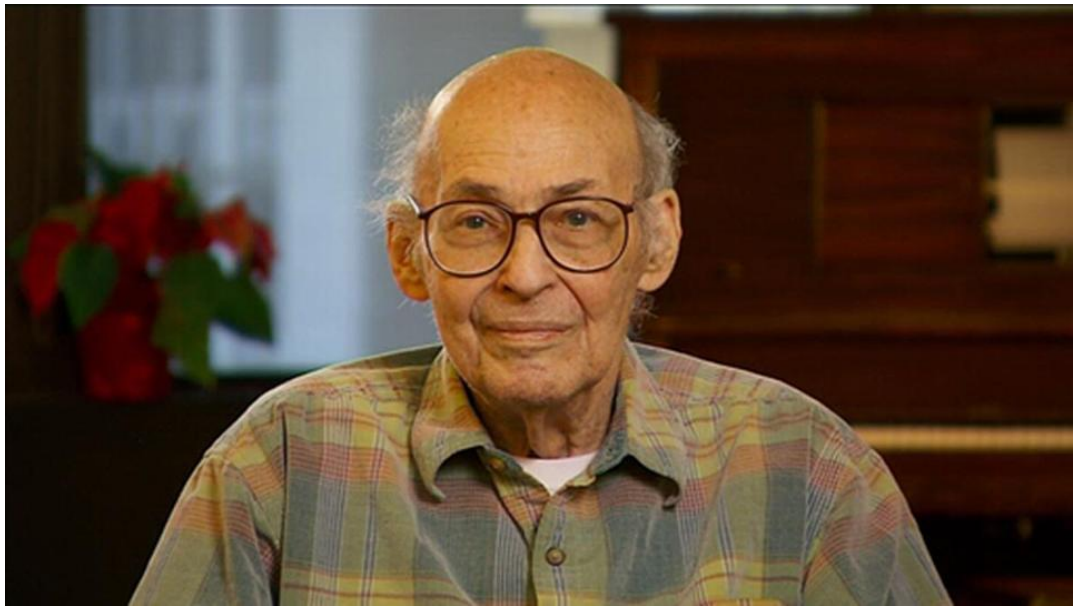
Texas A&M University

Outline

- History of Deep Learning
- Auto Encoding
- Convolutional Neural Networks
- Recurrent Neural Networks
- Transfer Learning
- Multi-task Learning

Deep Learning (i): History and Basics

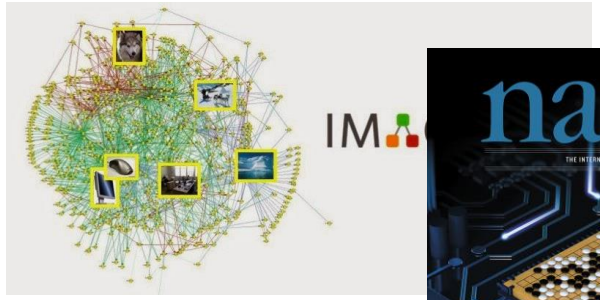




Marvin L. Minsky
Turing Awardee 1969, "Father of Artificial Intelligence"
(August 9, 1927 – January 24, 2016)
Who co-founded MIT's CSAIL, MIT Media Lab, and "killed neural networks once" in 1970s

A Triumph of Deep Learning: 2012 - present

Top-performers in many tasks, over many domains



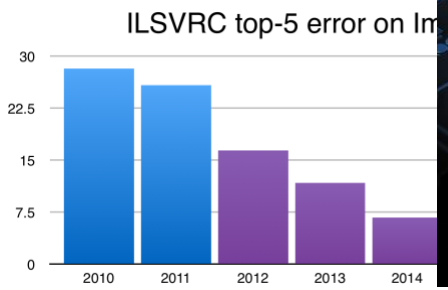
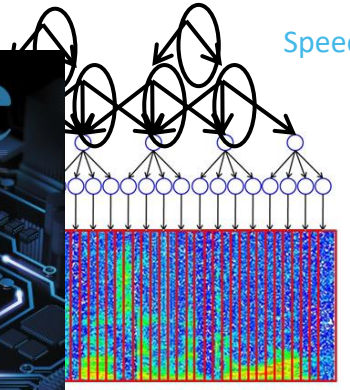


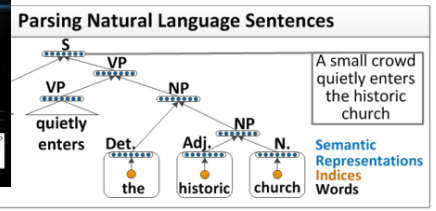
Image classification, detection, localization…



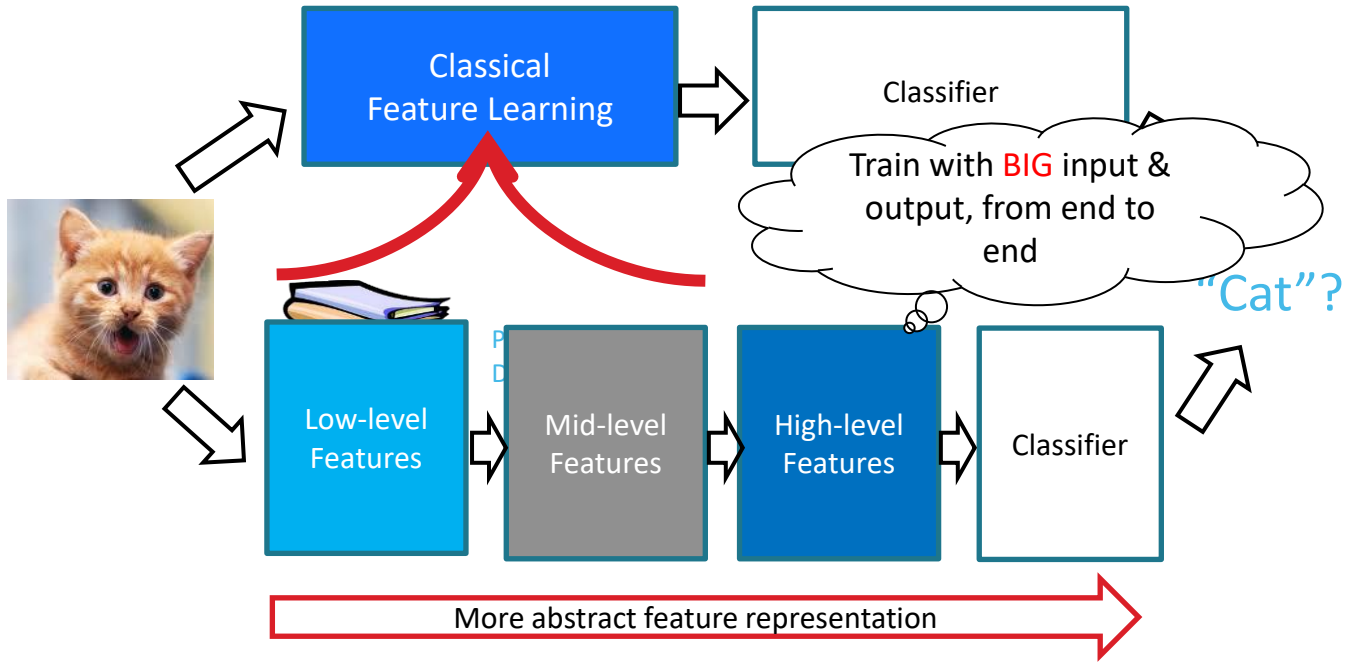
Speech Recognition

Spectrogram

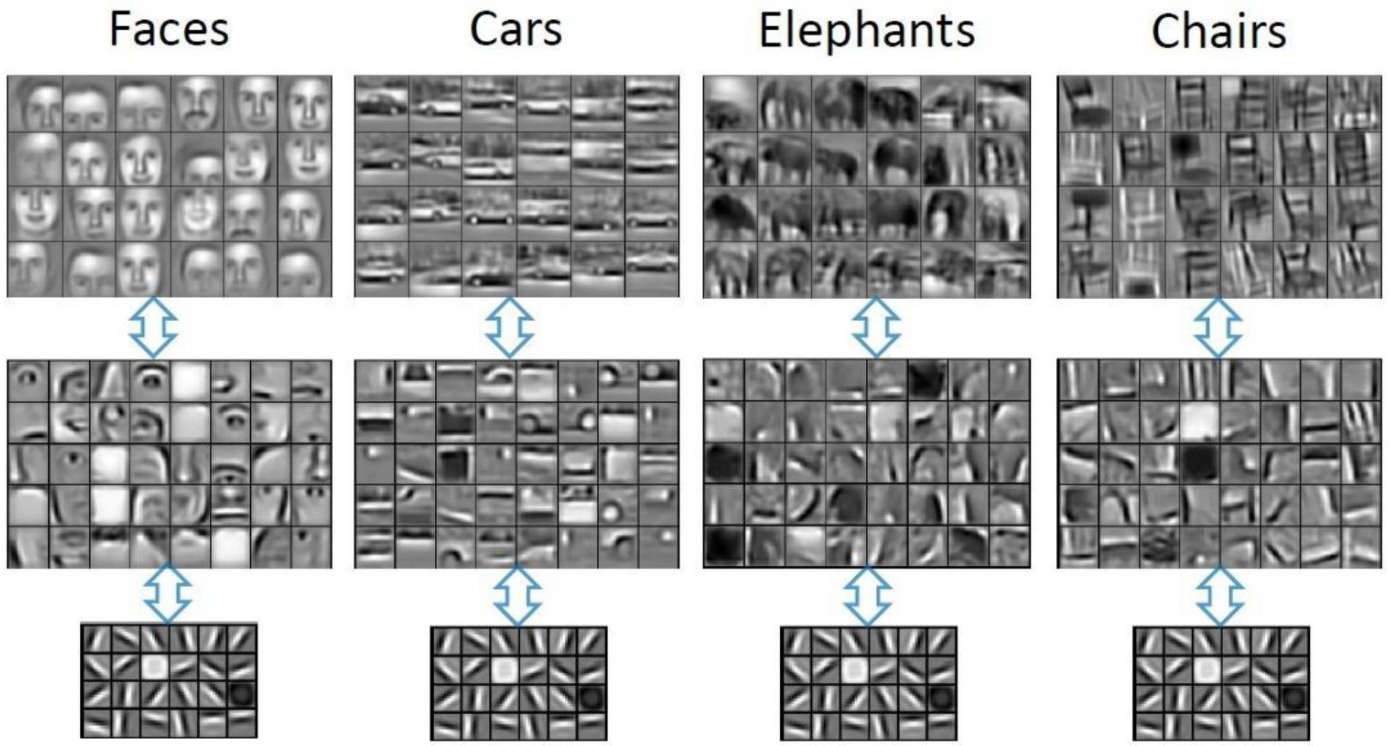
NLP



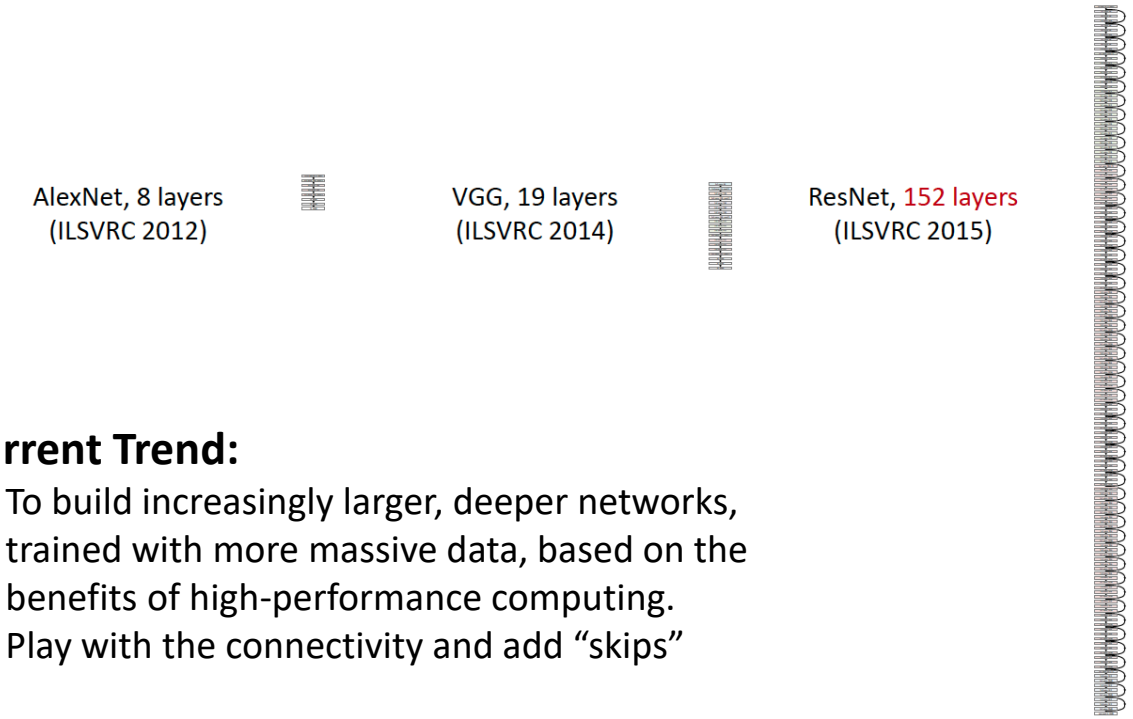
Feature learning: Going Deep



Deep Features (May) Learn Semantic Hierarchy



Status Quo

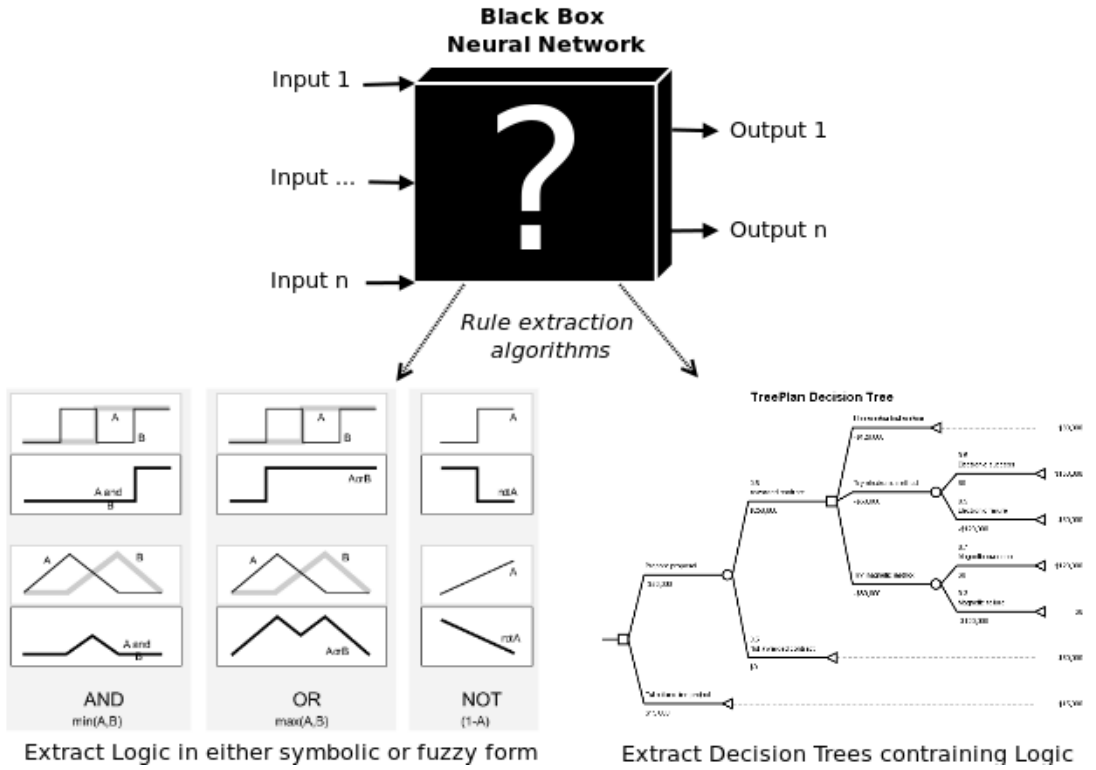


Current Trend:

- To build increasingly larger, deeper networks, trained with more massive data, based on the benefits of high-performance computing.
- Play with the connectivity and add “skips”

Grand Challenges

- Why/how deep learning works?
  - Gap between engineering (or art) and science: Lack of theoretical understandings & guarantees, and analytical tools
  - Training is computationally expensive and difficult, relying on many “magics”
  - No principled way to incorporate domain expertise
  - No principled way to interpret the model behaviors



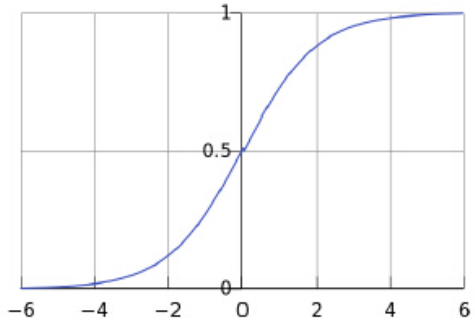
Extract Logic in either symbolic or fuzzy form

Extract Decision Trees contraining Logic

Building Blocks

Function model: $f(x) = \sigma(w^T \cdot x + b)$ **Terminology: "Weight" and "Neuron"**

- Parameters: vector $w \in R^d$, b is scalar bias term
- σ is a non-linearity, e.g. sigmoid: $\sigma(z) = 1/(1 + \exp(-z))$
- For simplicity, sometimes write $f(x) = \sigma(w^T x)$ where $w = [w; b]$ and $x = [x; 1]$

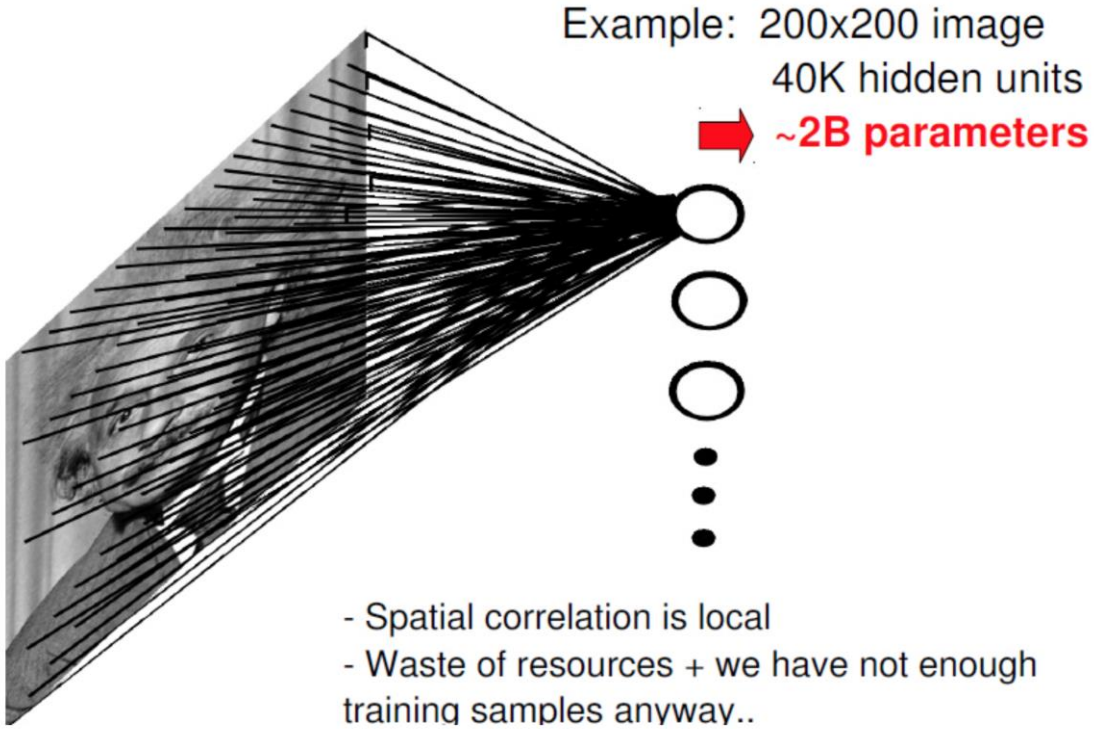


Assume Squared-Error* $Loss(w) = \frac{1}{2}\sum_m (\sigma(w^T x^{(m)}) - y^{(m)})^2$

Gradient: $\nabla_w Loss = \sum_m \left[\sigma(w^T x^{(m)}) - y^{(m)}\right] \sigma'(w^T x^{(m)}) x^{(m)}$

- General form of gradient: $\sum_m Error^{(m)} * \sigma'(in^{(m)}) * x^{(m)}$

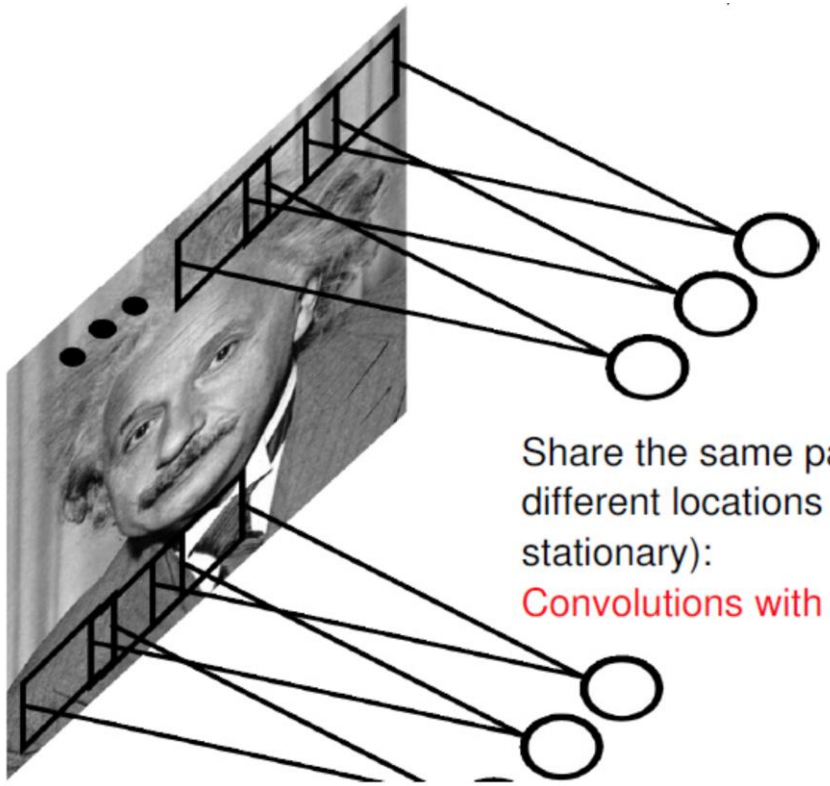
Fully Connected Layer



Outline

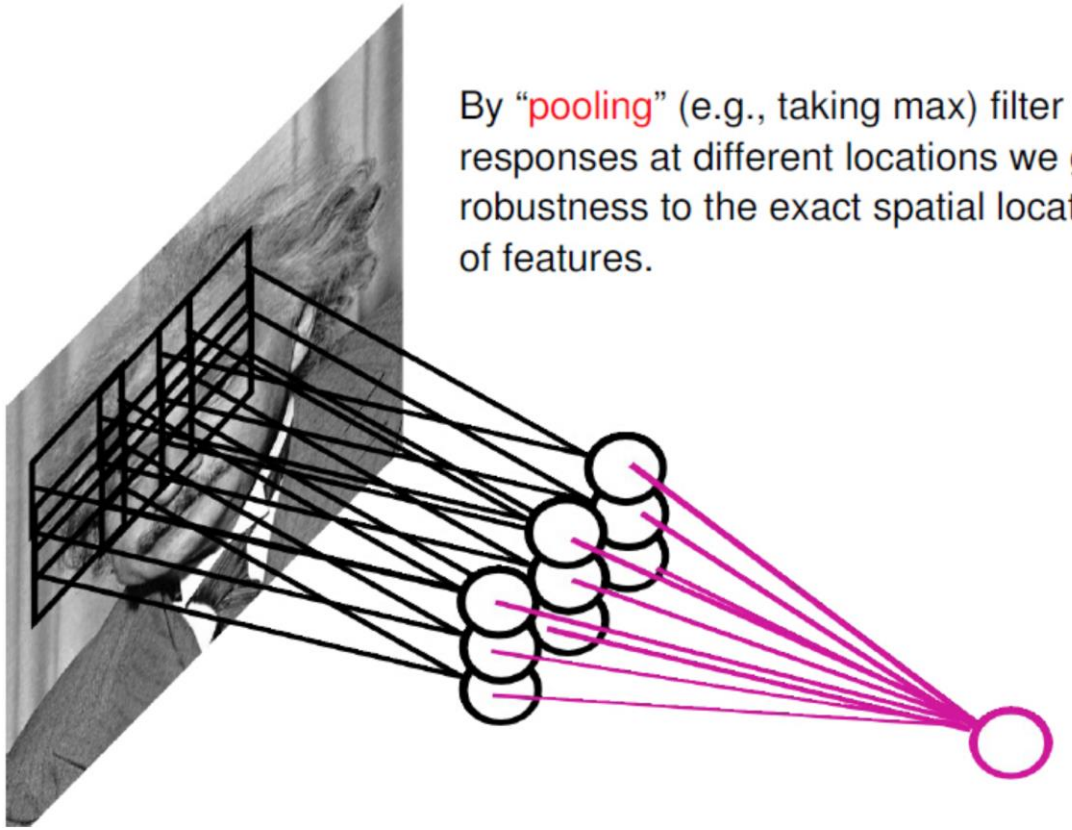
- History of Deep Learning
- Auto Encoding
- Convolutional Neural Networks
- Recurrent Neural Networks
- Transfer Learning
- Multi-task Learning

Convolutional Layer



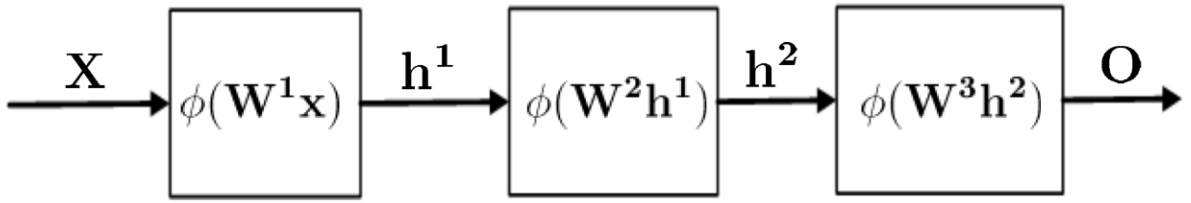
Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels

Pooling Layer

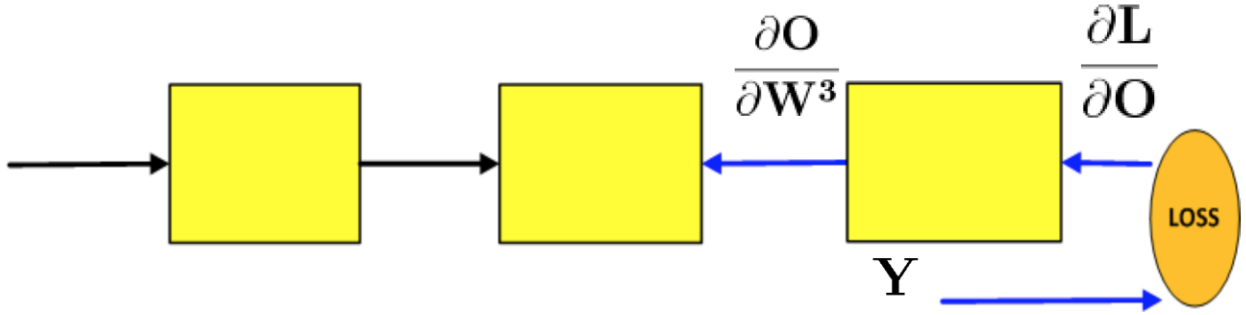


By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

Forward-Backward Propagation

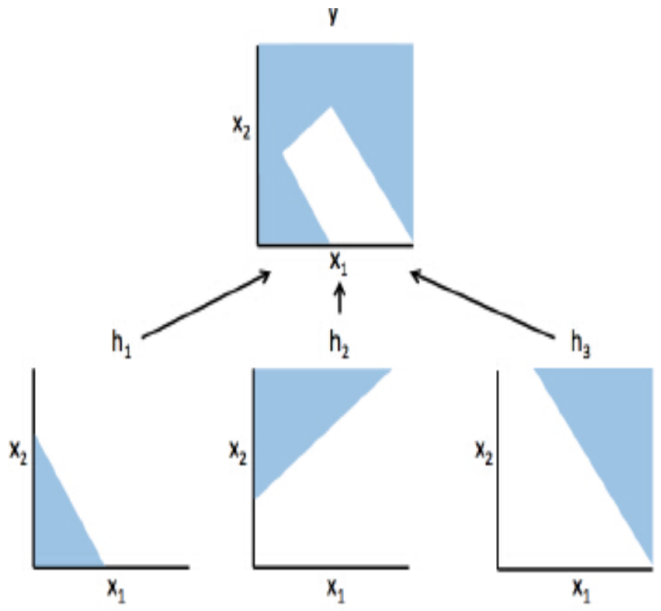


Forward propagation: $h(\mathbf{x}) = \phi(\mathbf{W}\mathbf{x})$



Backward propagation: $\frac{\partial \mathbf{L}}{\partial \mathbf{W^3}} = \frac{\partial \mathbf{L}}{\partial \mathbf{O}} \frac{\partial \mathbf{O}}{\partial \mathbf{W^3}}$ **(Chain Rule)**

Universal Approximation Theorem



Stacking *Linear* Classifiers (Perceptron)

- 1-layer only model liner hyperplanes
- 2-layer with infinite hidden nodes can express any continuous functions
- >2-layer can do this with less nodes

Universal Approximation Theorem

Let $\varphi(\cdot)$ be a nonconstant, bounded, and monotonically-increasing continuous function. Let I_m denote the m-dimensional unit hypercube $[0,1]^m$. The space of continuous functions on I_m is denoted by $C(I_m)$. Then, given any $\varepsilon > 0$ and any function $f \in C(I_m)$, there exist an integer N, real constants $v_i, b_i \in \mathbb{R}$ and real vectors $w_i \in \mathbb{R}^m$, where $i = 1, \cdots, N$, such that we may define:

$$F(x) = \sum_{i=1}^{N} v_i \varphi\left(w_i^T x + b_i\right)$$

as an approximate realization of the function f where f is independent of φ; that is,

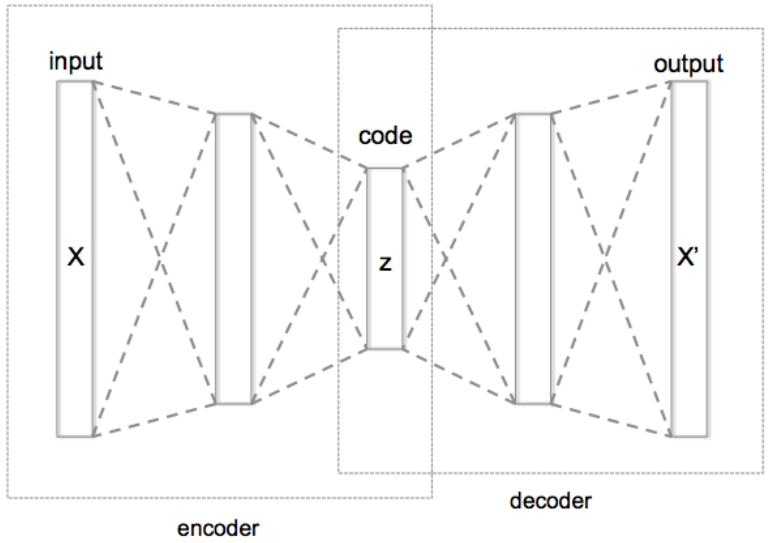
$$|F(x) - f(x)| < \varepsilon$$

for all $x \in I_m$. In other words, functions of the form $F(x)$ are dense in $C(I_m)$.

- A feed-forward network with **a single hidden layer** containing **a finite number of *nonlinear* neurons**, can approximate **any continuous function on compact subsets of R^n**, under mild assumptions.
- It is not the specific choice of the activation function, but rather the **multilayer feedforward architecture** itself which gives neural networks the potential of being universal approximators.
- It does not touch upon the **algorithmic learnability** of those parameters.

Auto-Encoder

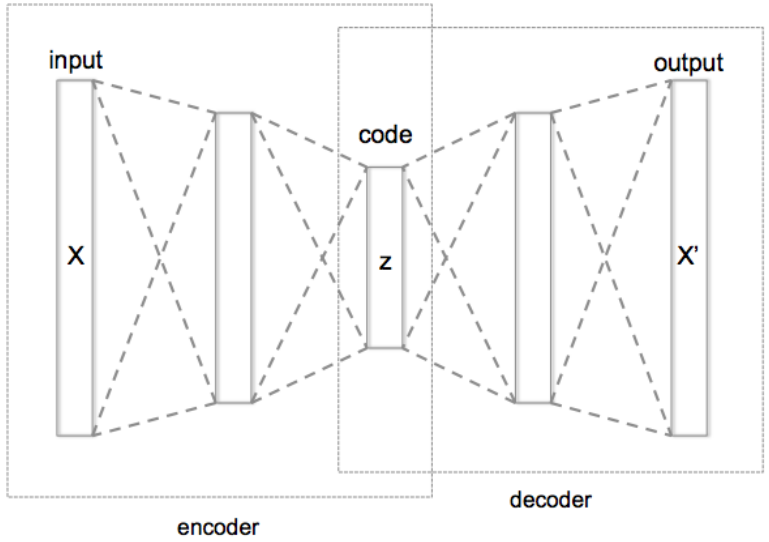
- Unsupervised feature extraction
- Reconstruct the input from itself via using “bottleneck”



X = X’

Denoising Auto-Encoder

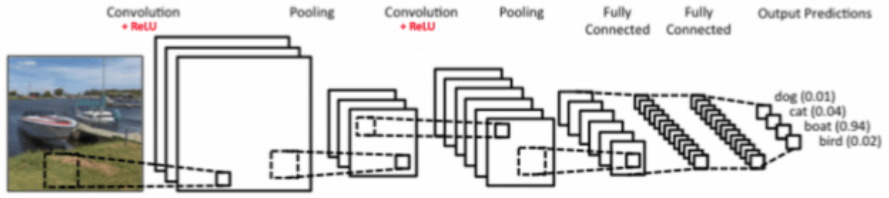
- Reconstruct the input from a slightly corrupted “noisy” version
- Purpose: learning robust features for better generalization



X = X’ + noise

Convolutional Neural Networks

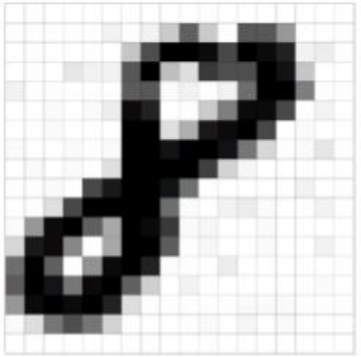
- Similar to regular neural networks
  - made up of neurons, each with an input and an activation function
  - have weights and biases to be learned
  - have a loss function on the last (fully-connected) layer
- Explicit assumption that the inputs are images
  - explicit assumption that the inputs are images
  - vastly reduce the amount of parameters in the network



Convolutional Neural Networks

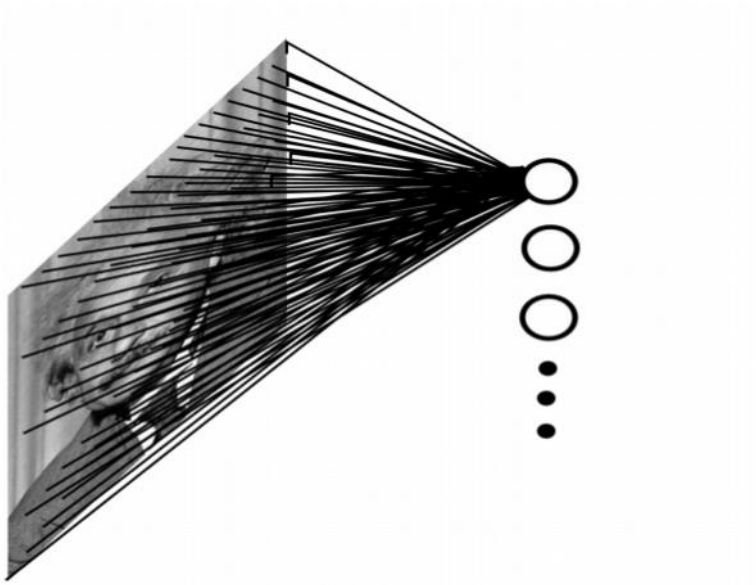
Image representation

- Grayscale image (1-channel)
  - 2d-matrix
  - each pixel ranges from 0 to 255 - 0: black, 255: white
- Color image (3-channel, RGB)
  - three 2d-matrices stacked over each other
  - each with pixel values ranging between 0 and 255



Convolutional Neural Networks

A **fully** connected neural network with image input

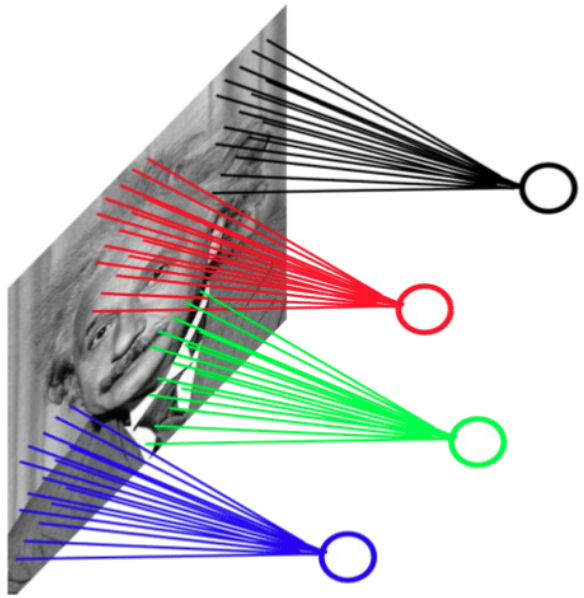


- 1000×1000 image, 1M hidden units $\rightarrow 10^{12}$ parameters
- Since spatial correlation is local, we can significantly simplify this

Convolutional Neural Networks

Idea 1: Convolution

A **locally** connected neural network with image input

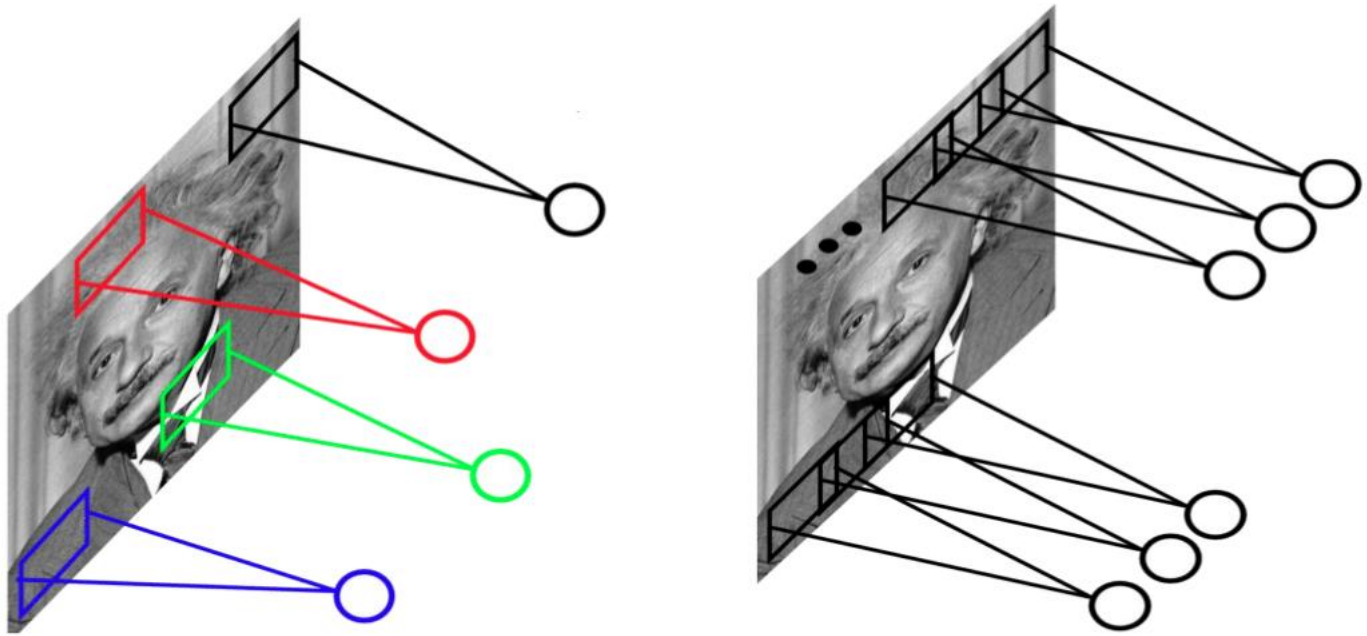


- 1000×1000 image, 1M hidden units, 10×10 filter size $\rightarrow$ 10^8 parameters
- Since spatial correlation is local, we can significantly simplify this

Convolutional Neural Networks

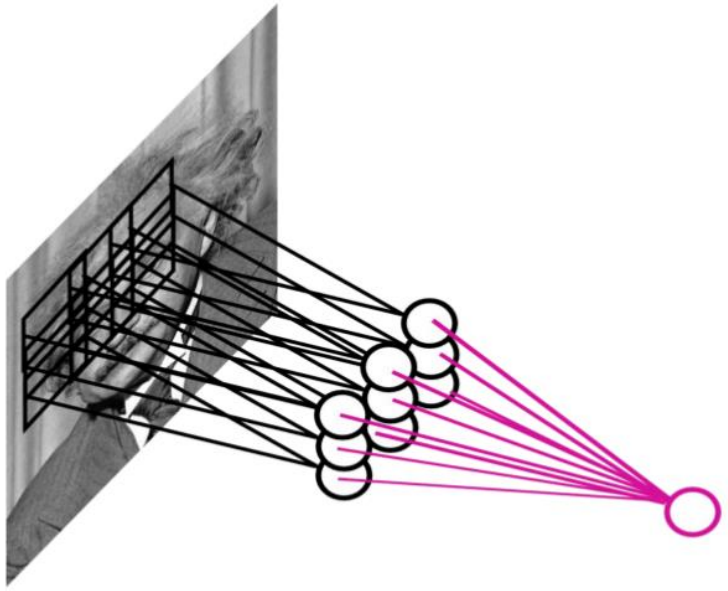
Idea 2: Weight sharing

- Stationarity: Statistics are similar at different locations
- Share the same parameters across different locations



Convolutional Neural Networks

Idea 3: Max-pooling

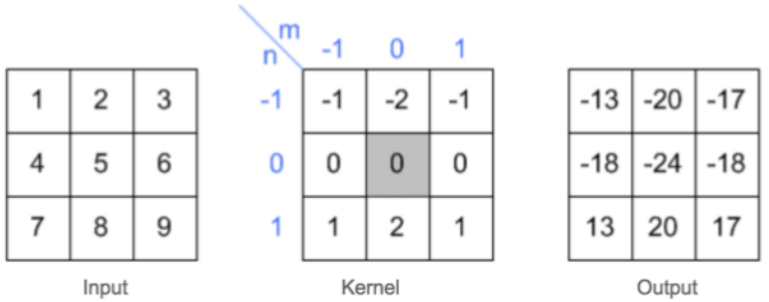


- Let us assume filter is an "eye" detector
- How can we make the detection robust to the exact location of the eye?
- By pooling (e.g., max or average) filter responses at different locations we gain robustness to the exact spatial location of features

Convolution 2d-example

- Convolution is the mathematical operation that implements filtering
- Given an input image $x[m, n]$ and an impulse response $h[m, n]$ (filter or kernel), the convolution output can be written as

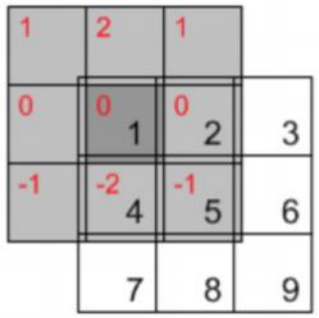
$$y[m, n] = x[m, n] * h[m, n] = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} x[i, j]h[m - i, n - j]$$



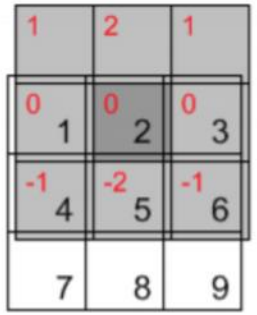
http://www.songho.ca/dsp/convolution/convolution.html#convolution_2d

Convolutional Neural Networks

Convolution 2d-example



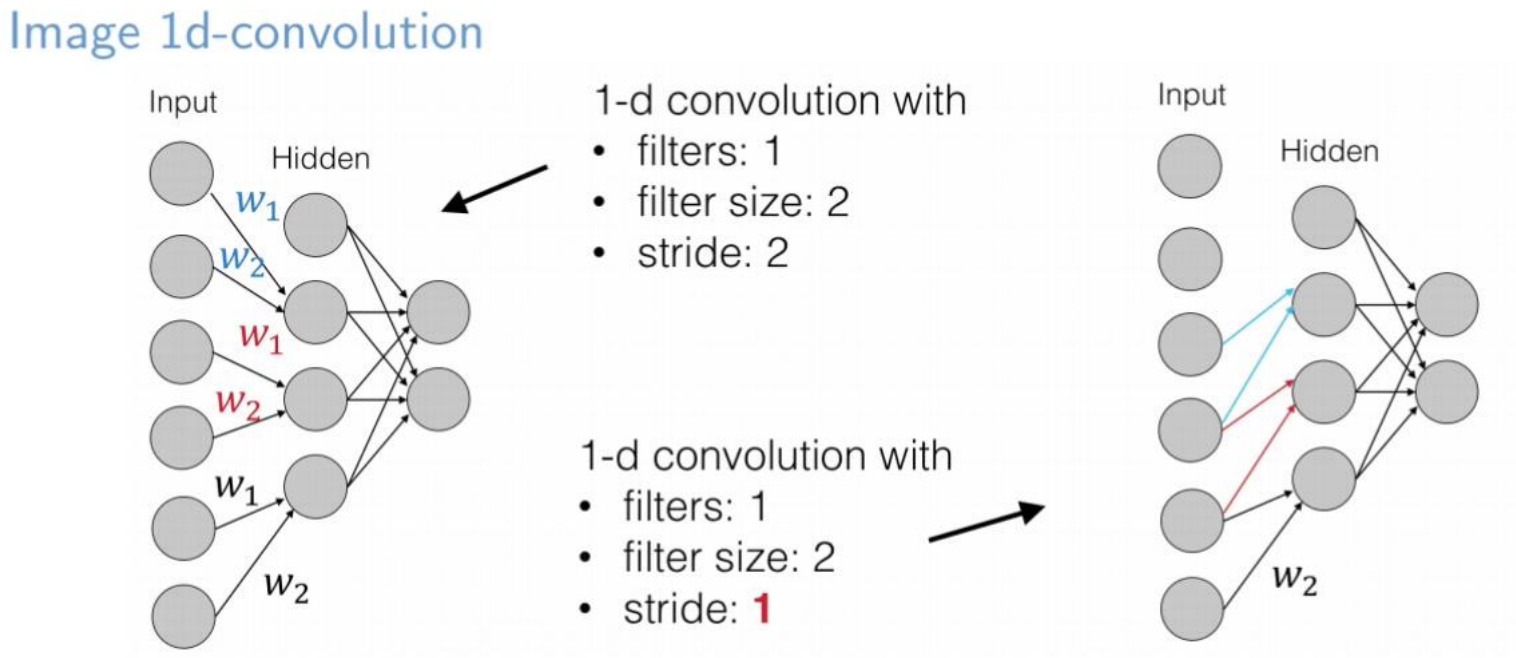
$$y[0,0] = x[-1,-1]\cdot h[1,1] + x[0,-1]\cdot h[0,1] + x[1,-1]\cdot h[-1,1]$$
$$+ x[-1,0]\cdot h[1,0] + x[0,0]\cdot h[0,0] + x[1,0]\cdot h[-1,0]$$
$$+ x[-1,1]\cdot h[1,-1] + x[0,1]\cdot h[0,-1] + x[1,1]\cdot h[-1,-1]$$
$$= 0\cdot 1 + 0\cdot 2 + 0\cdot 1 + 0\cdot 0 + 1\cdot 0 + 2\cdot 0 + 0\cdot(-1) + 4\cdot(-2) + 5\cdot(-1) = -13$$



$$y[1,0] = x[0,-1]\cdot h[1,1] + x[1,-1]\cdot h[0,1] + x[2,-1]\cdot h[-1,1]$$
$$+ x[0,0]\cdot h[1,0] + x[1,0]\cdot h[0,0] + x[2,0]\cdot h[-1,0]$$
$$+ x[0,1]\cdot h[1,-1] + x[1,1]\cdot h[0,-1] + x[2,1]\cdot h[-1,-1]$$
$$= 0\cdot 1 + 0\cdot 2 + 0\cdot 1 + 1\cdot 0 + 2\cdot 0 + 3\cdot 0 + 4\cdot(-1) + 5\cdot(-2) + 6\cdot(-1) = -20$$

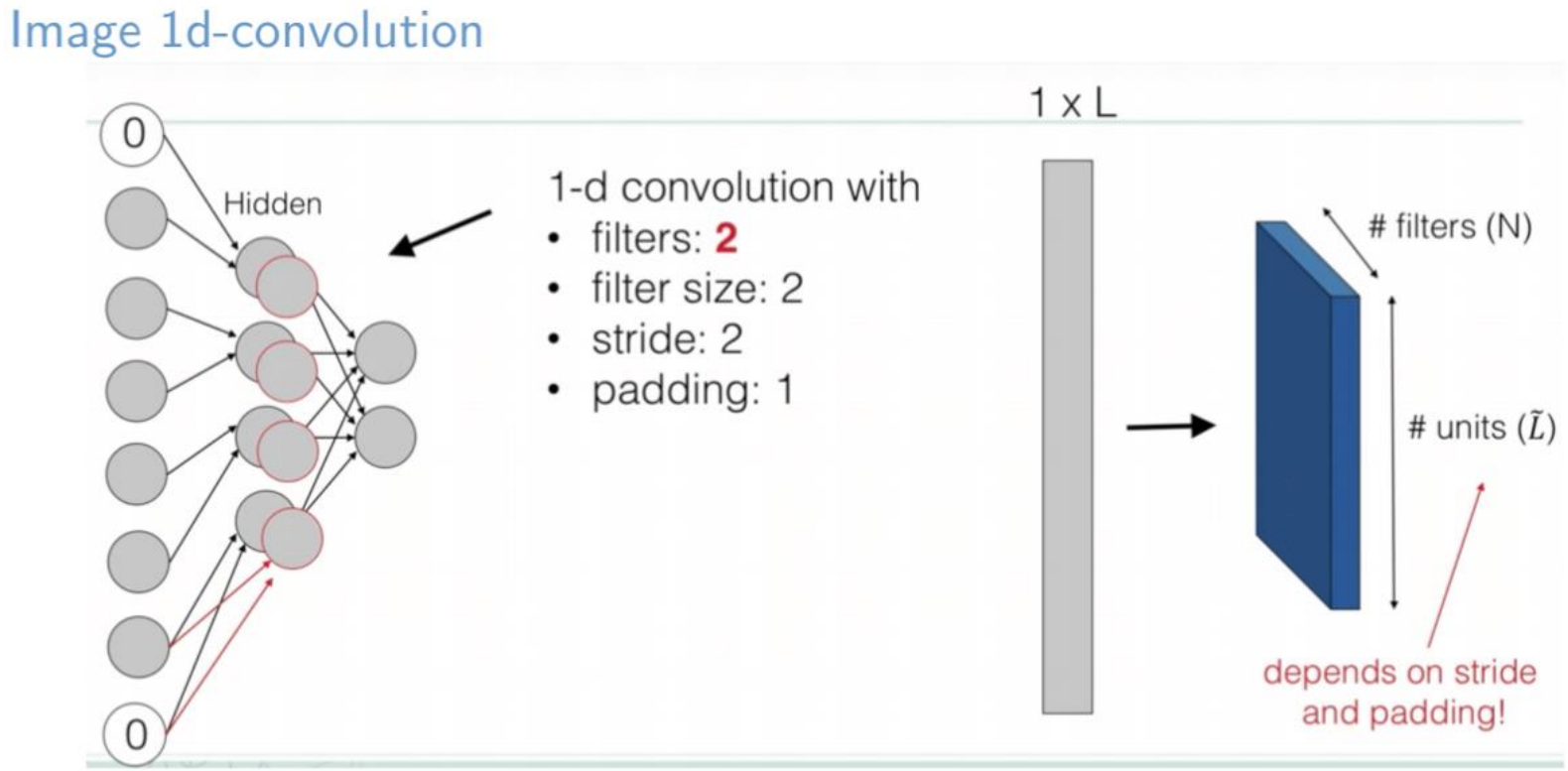
http://www.songho.ca/dsp/convolution/convolution.html#convolution_2d

Convolutional Neural Networks



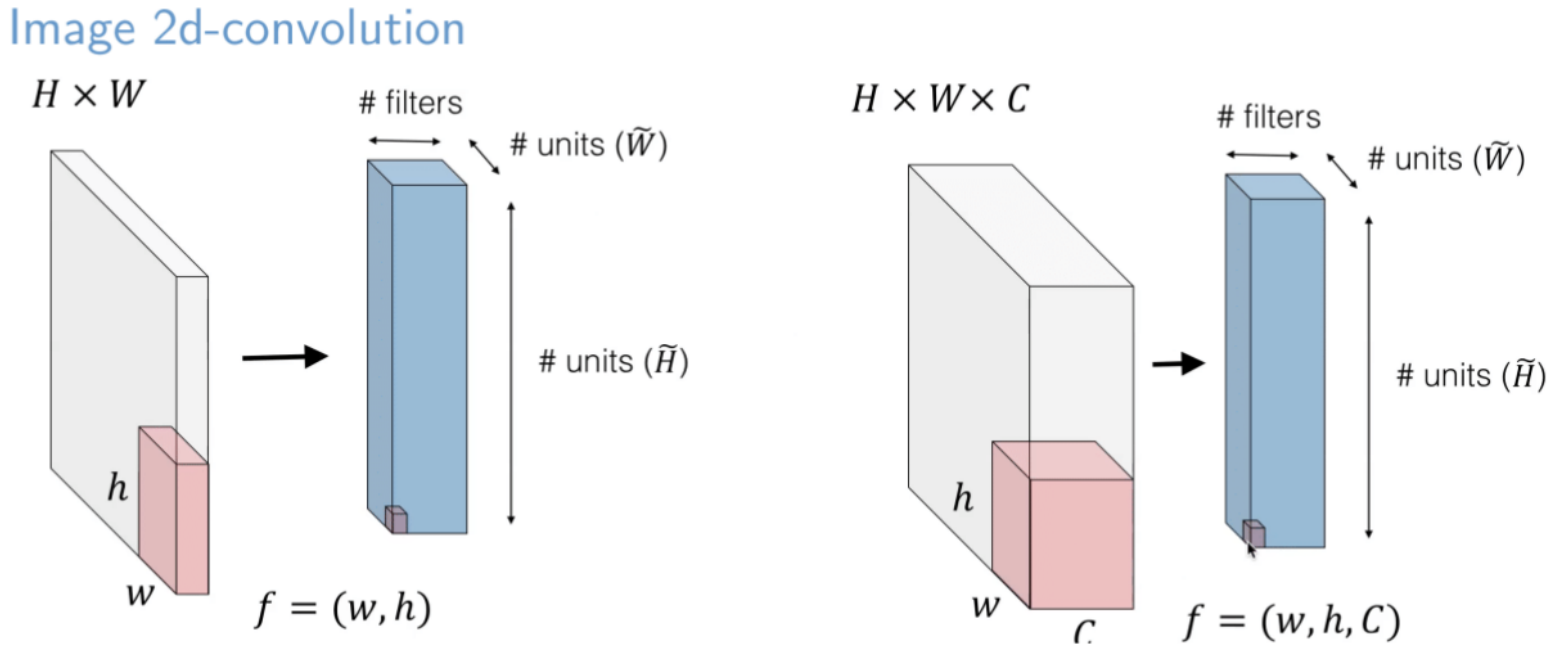
https://www.nervanasys.com/convolutional-neural-networks/

Convolutional Neural Networks



https://www.nervanasys.com/convolutional-neural-networks/

Convolutional Neural Networks



https://www.nervanasys.com/convolutional-neural-networks/

Also check:

http://cs231n.github.io/assets/conv-demo/index.html

https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/ (figure 6)

Convolutional Neural Networks

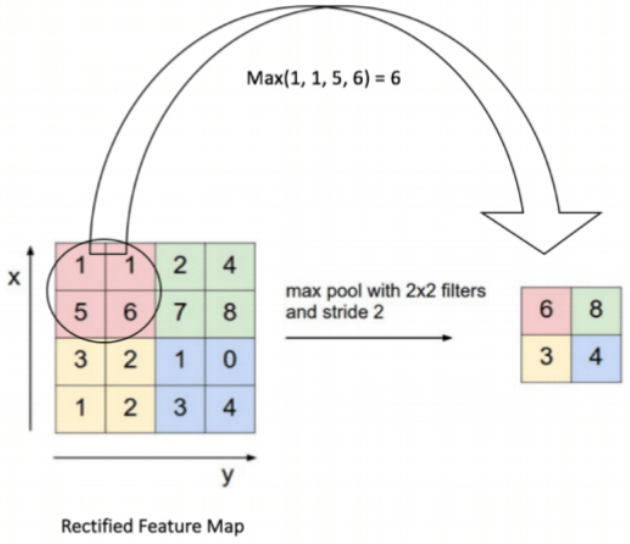
Image 2d-convolution hyperparameters

- Depth: the number of filters we use for the convolution operation
- Stride: the number of pixels by which we slide our filter matrix over the input
- Zero-padding: padding the input matrix with zeros around the border, so that we can apply the filter to bordering elements of our input image matrix

Convolutional Neural Networks

Spatial Pooling (also called subsampling or downsampling)

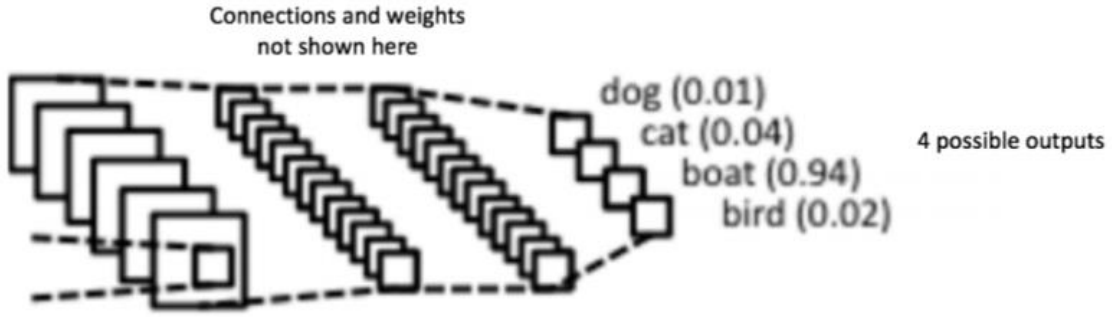
- Reduces the dimensionality of each feature map but retains the most important information
- Can be of different types: Max, Average, Sum etc.
- Makes the input representations (feature dimension) smaller and more manageable
- Promotes an almost scale invariant representation of the image



Convolutional Neural Networks

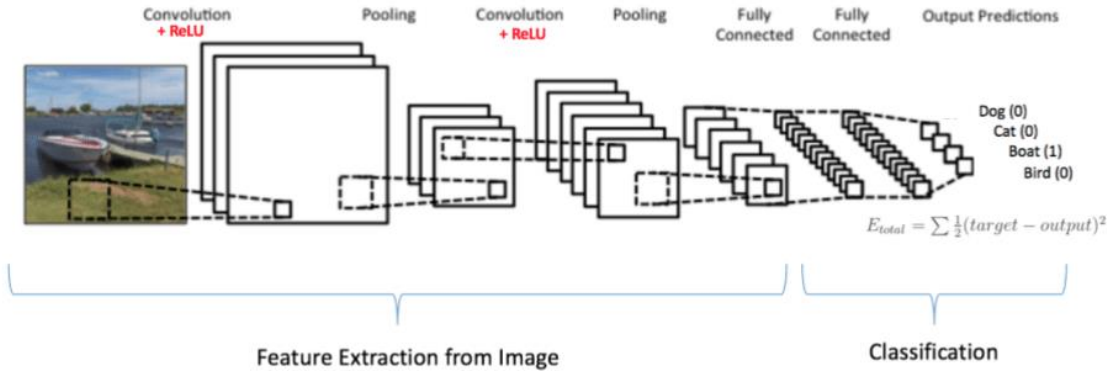
Final fully connected layer

- Traditional multilayer perceptron
- Yields the classification/regression result

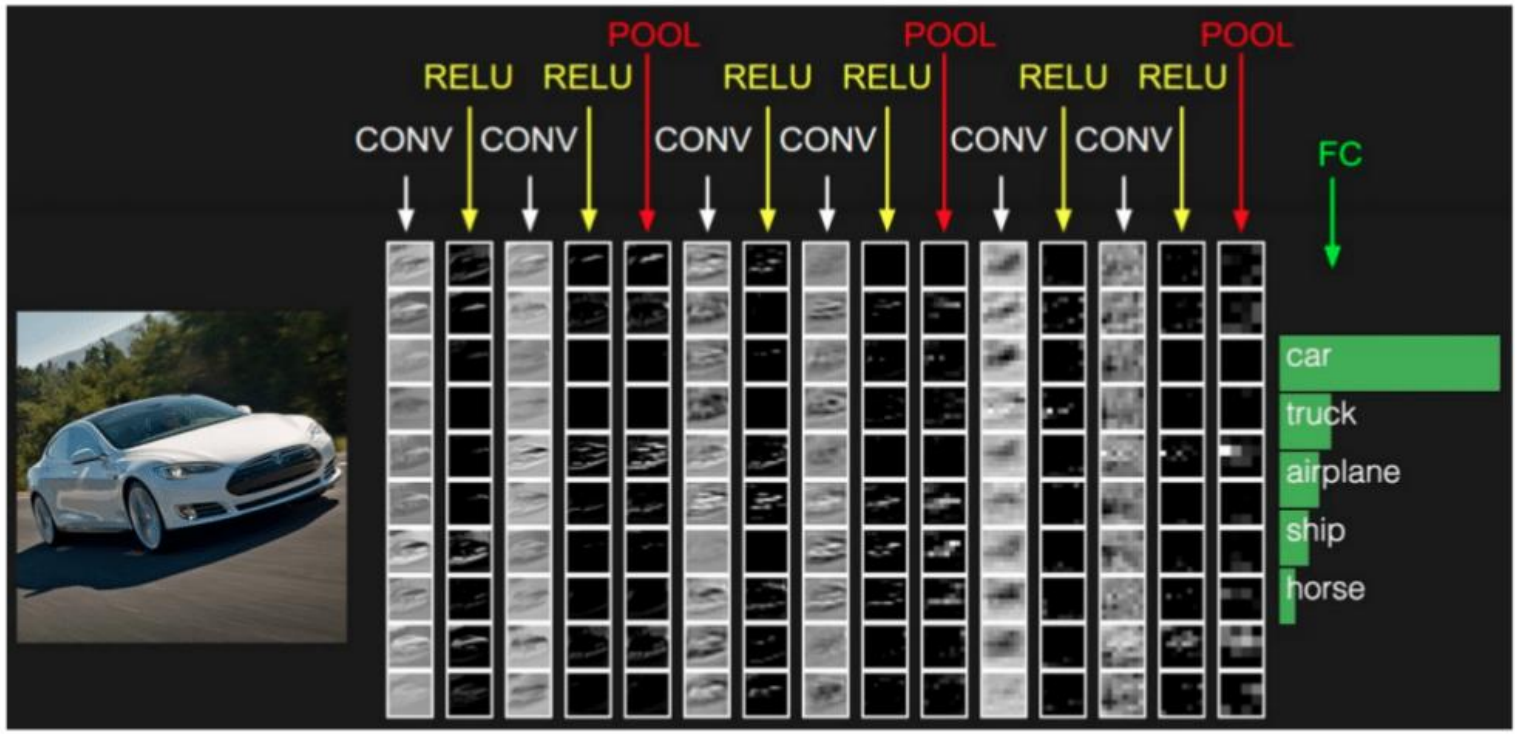


Convolutional Neural Networks

- **Step 1:** Initialize weights
- **Step 2:** Take first image as input and go through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the fully connected layer) and finds the output probabilities for each class
- **Step 3:** Calculate the total error at the output layer
- **Step 4:** Use backpropagation to update the weights, which are adjusted in proportion to their contribution to the total error
- **Step 5:** Repeat Steps 1-4 for all train images

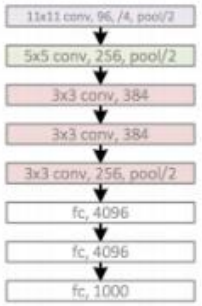


Convolutional Neural Networks

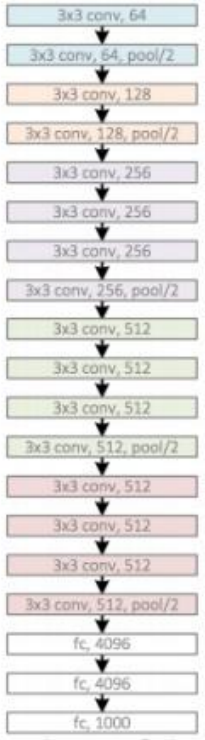


Convolutional Neural Networks

AlexNet, 8 layers
(ILSVRC 2012)



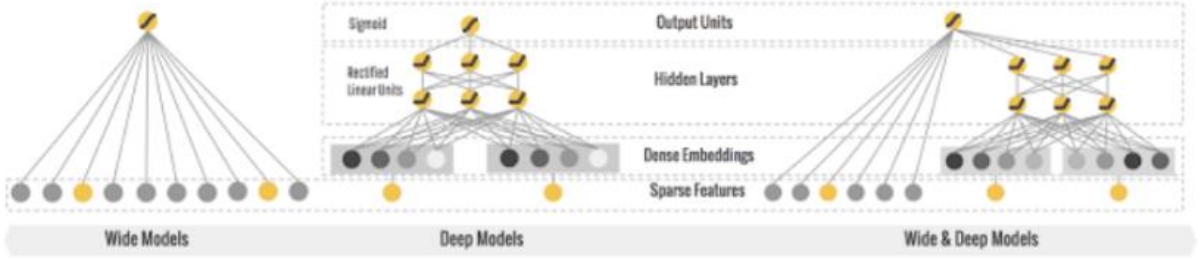
VGG, 19 layers
(ILSVRC 2014)



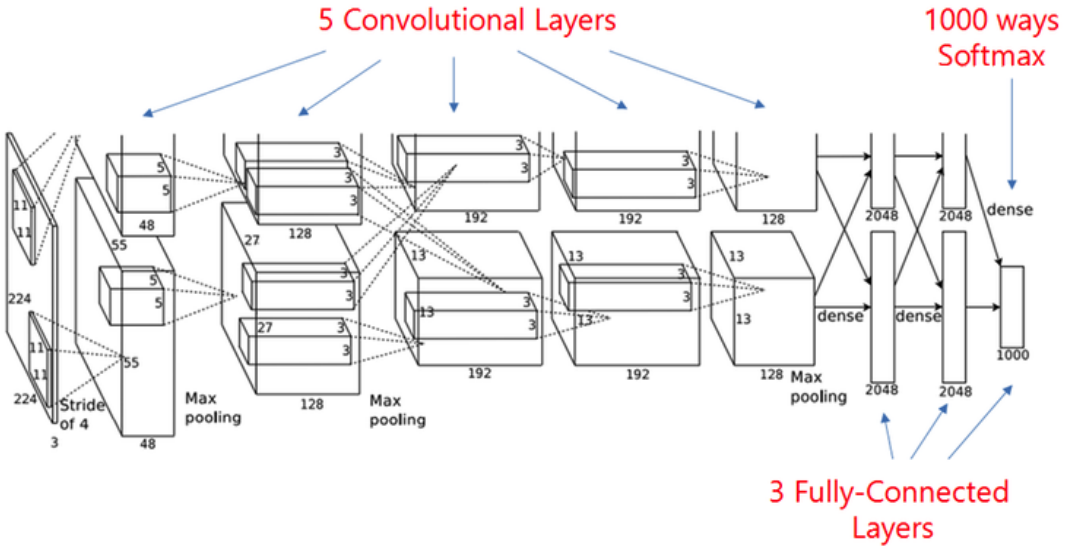
GoogleNet, 22 layers
(ILSVRC 2014)

Wide and deep models

- Google: Wide & Deep Learning (2016)
- generic large-scale regression and classification problems with sparse inputs (categorical features with a large number of possible feature values), such as recommender systems, search, and ranking problems
- The wide model takes into account a wide set of cross-product feature transformations to capture how the co-occurrence of a query-item feature pair correlates with the target label
- The deep feed-forward neural network learns lower-dimensional dense representations for every query and item.



AlexNet, 2012



- The **FIRST** winner deep model in computer vision, and one of the most classical choices for domain experts to adapt for their applications
- 5 convolutional layers + 3 fully-connected layers + softmax classifier
- **Key Technical Features**: ReLU, dropout, data augmentation

VGG-Net, 2014

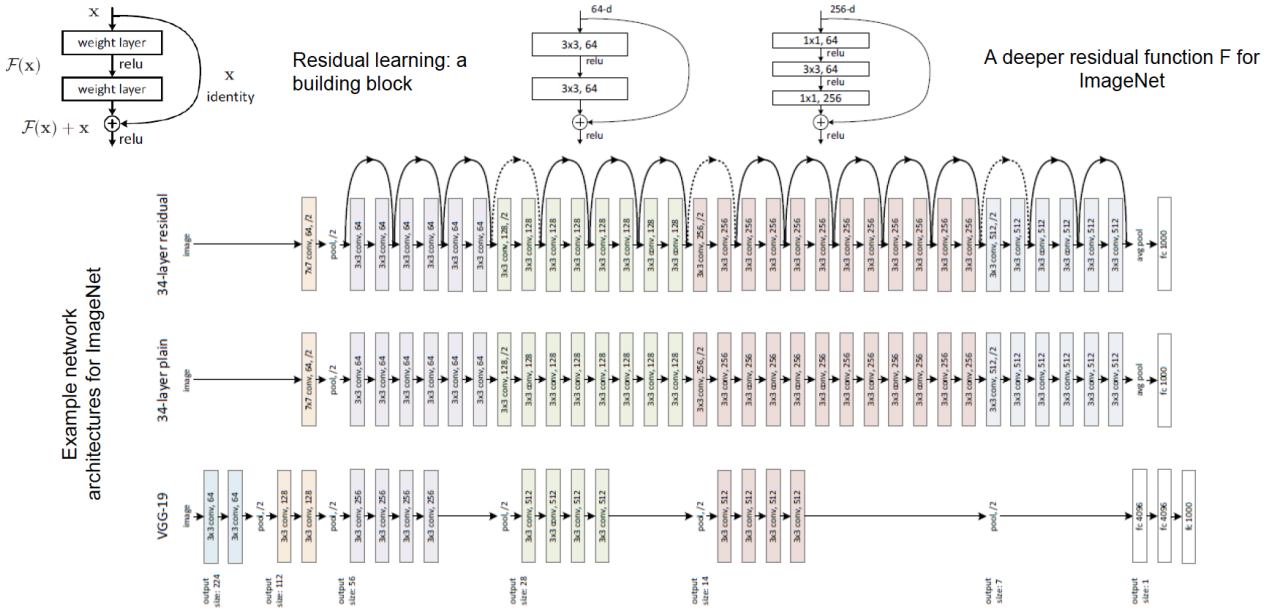
ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Key Technical Features:

- Increase depth (up to 19)
- Smaller filter size (3)

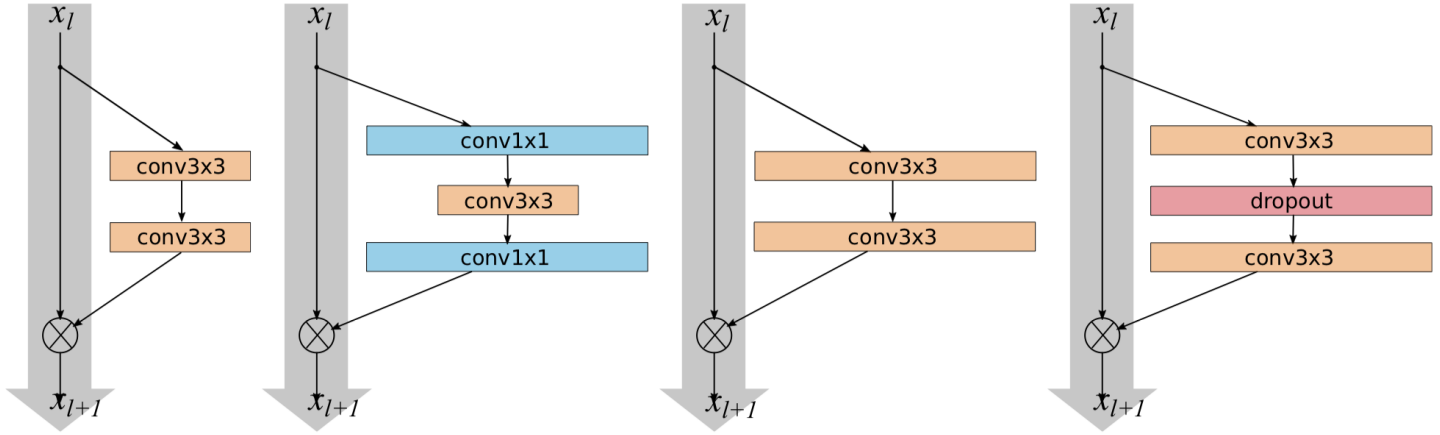
Configurations D and E are widely used for various tasks, called *VGG-16* and *VGG-19*

Deep Residual Network (ResNet), 2015



Key Technical Features: skip connections for residual mapping, up to > 1000 layers

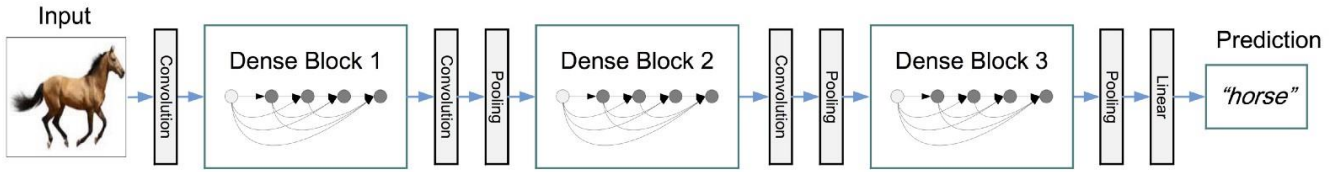
Wide ResNet, 2016



(a) basic (b) bottleneck (c) basic-wide (d) wide-dropout

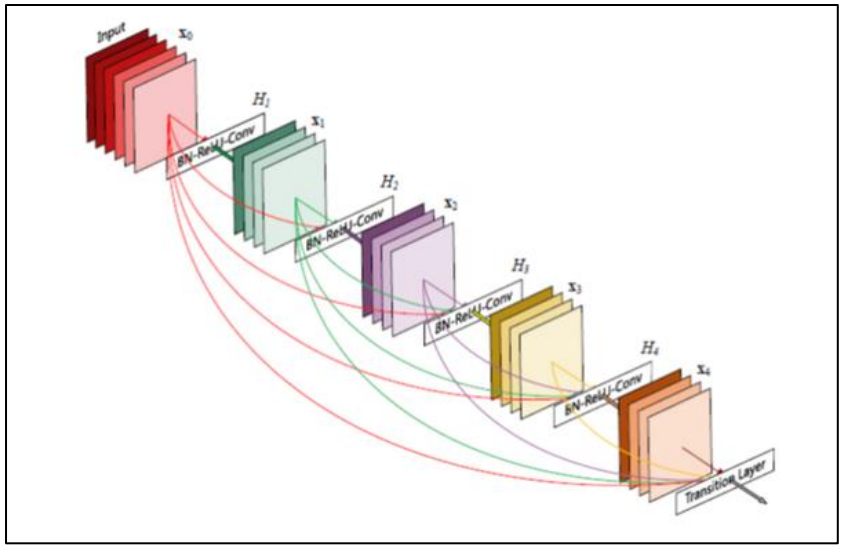
- Widening of ResNet blocks (if done properly) provides a more effective way of improving performance of residual networks compared to increasing their depth.
- A wide 16-layer deep network has the same accuracy as a 1000-layer thin deep network and a comparable number of parameters, although being several times faster to train.

Densely Connected Convolutional Networks (DenseNet), 2017

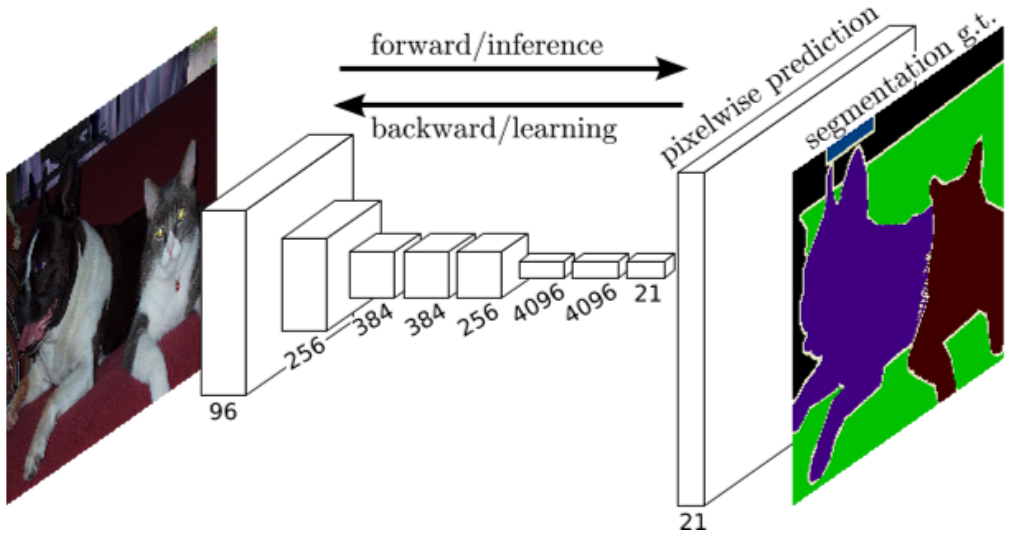


Key Technical Features:

- Finer combination of multi-scale features (or whatever…)



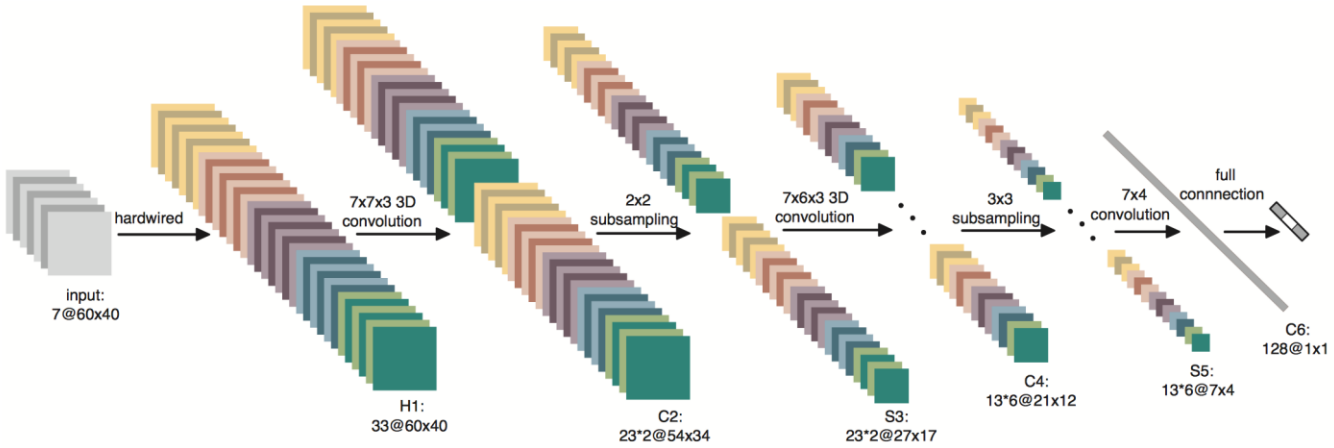
Fully Convolutional Network (FCN)



Key Technical Features:

- No fully-connected layer **->** No fixed requirement on input size
- Widely adopted in pixel-to-pixel prediction tasks, e.g., image segmentation

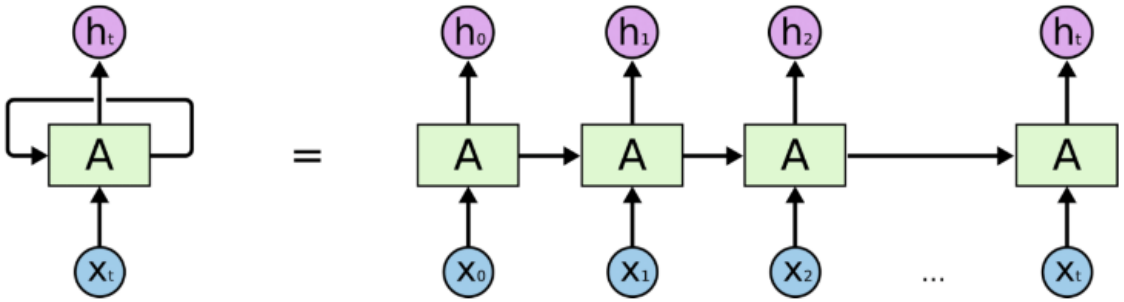
3D Convolutional Network (3D CNN)



Key Technical Features:

- Going from 2D convolutional filters to 3D filters, to take temporal coherence into consideration

Recurrent Neural Network

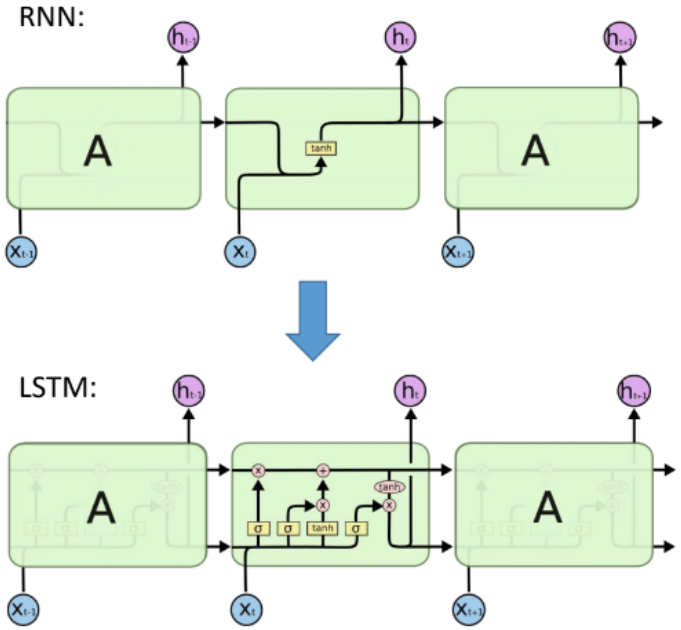


- A RNN is **unfolded** its forward and backward computations.
- Backpropagation Through Time (**BPTT**): Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps
- **Vanishing/Exploding Gradients**: Difficulty in learning long-term dependency

An intro article for RNN/LSTM: "Understanding LSTM Networks":
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Recurrent Neural Network

- Long short-term Memory



- A Long Short Term Memory (**LSTM**) combats vanishing gradients through a **gating** mechanism, thus capturing **long-term dependency** better.
- A LSTM does the exact same thing as a RNN, just in a different way!
- **Key Idea:** the gating functions are learned together with weights, and determine how much information we would like keep from last state and current computation, etc.

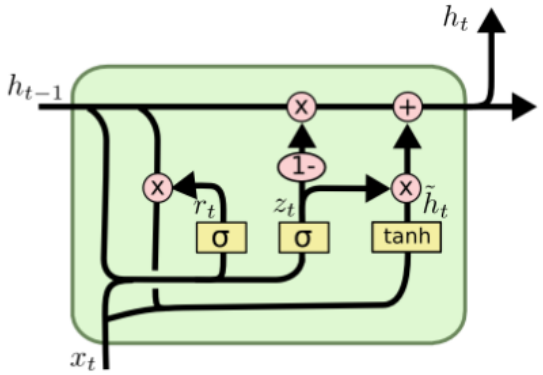
Recurrent Neural Network

- Long short-term Memory

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Recurrent Neural Network

- Gated Recurrent Unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

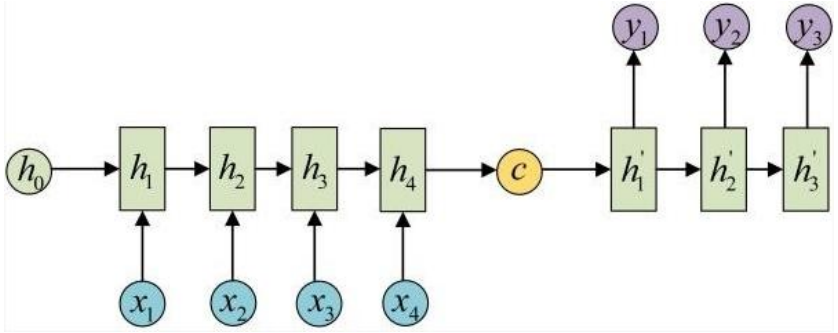
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Practice:

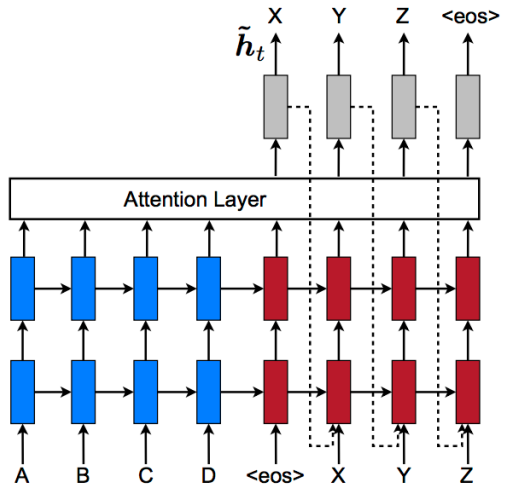
Implement LSTM without calling Tensorflow/PyTorch/Keras LSTM module.

Recurrent Neural Network

- Sequence to sequence (encoder decoder)

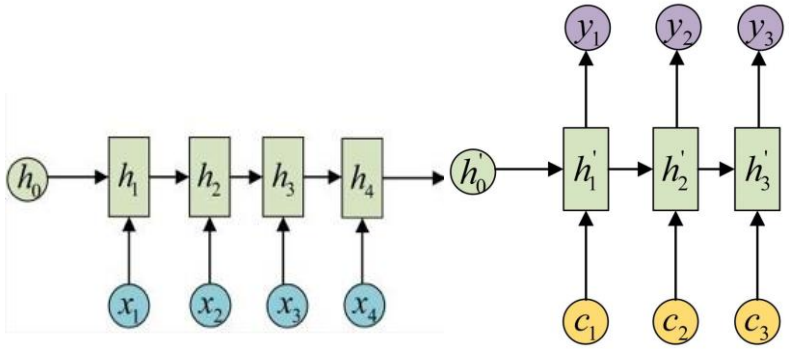


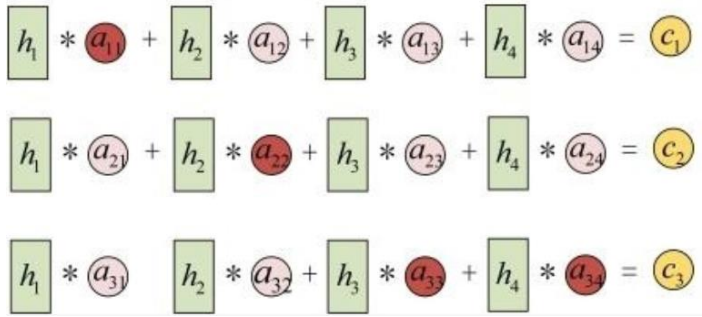
- Attention

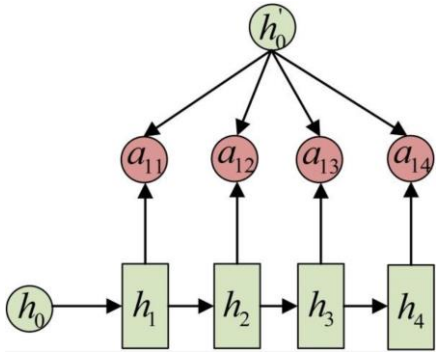


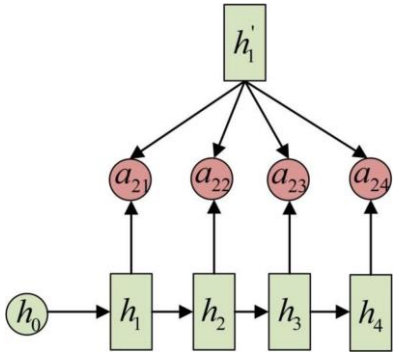
Recurrent Neural Network

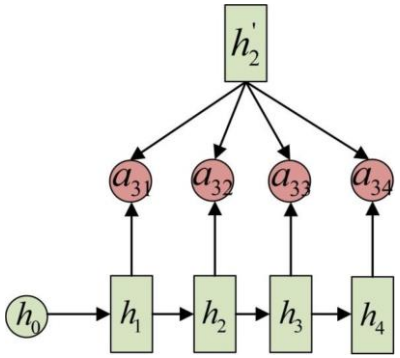
- Attention





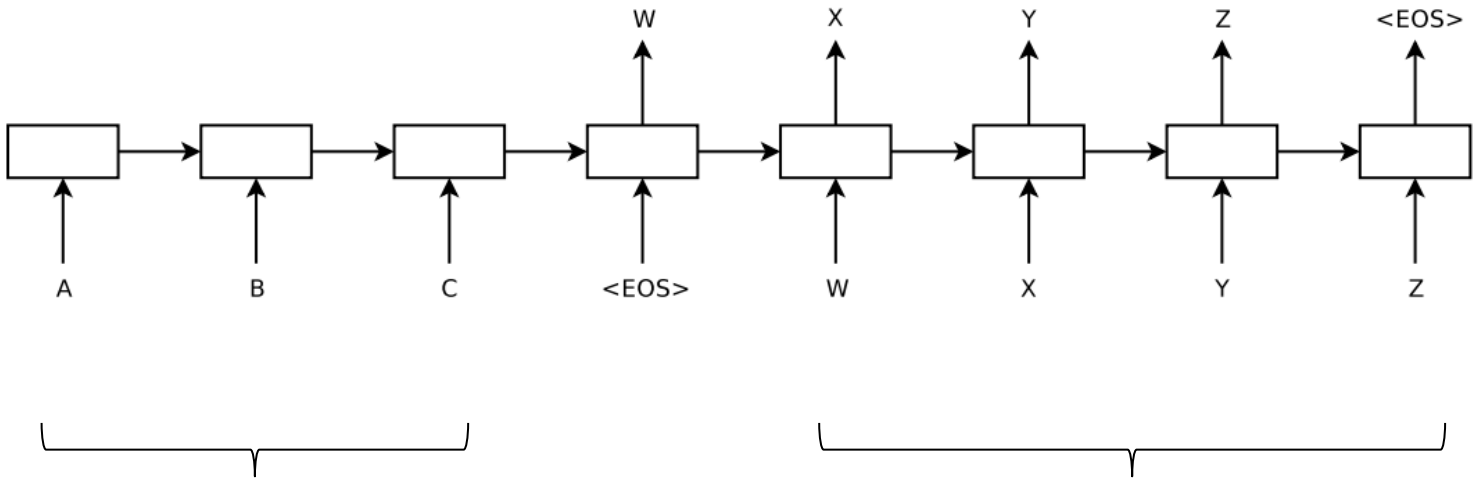






Recurrent Neural Network

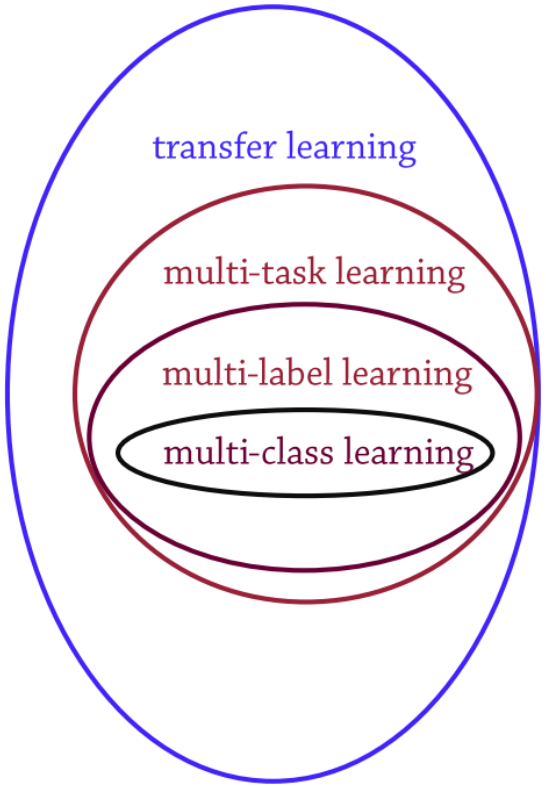
- Sequence to sequence (encoder-decoder)



Encoder:
- Sequentially deliver inputs
- Has no output

Decoder:
- Has no input (inputs are from outputs of previous step)
- Sequentially produce outputs

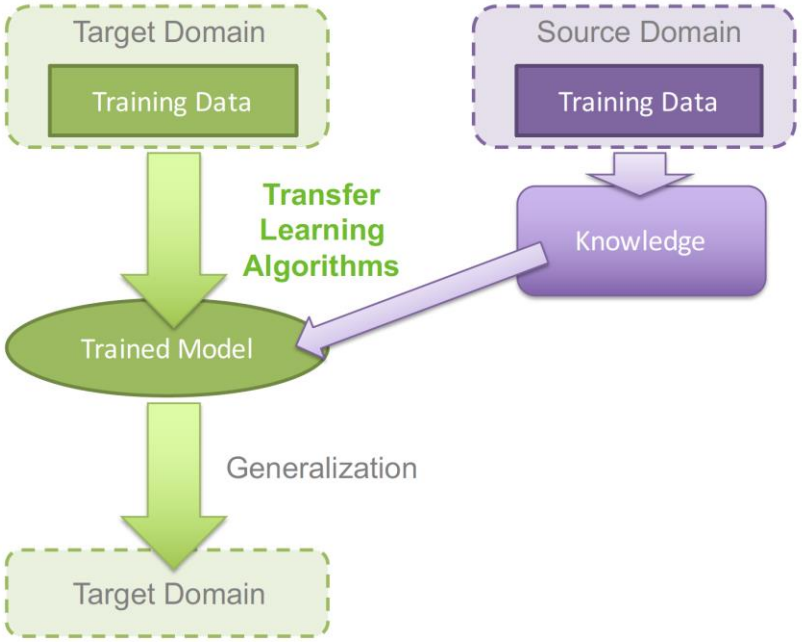
Learning Methods



- Transfer Learning
  - Define source & target domains
  - Learn on the source domain
  - Generalize on the target domain
- Multi-task Learning
  - Model the task relatedness
  - Learn all tasks simultaneously
  - Tasks may have different data/features
- Multi-label Learning
  - Model the label relatedness
  - Learn all labels simultaneously
  - Labels share the same data/features
- Multi-class Learning
  - Learn the classes independently
  - All classes are exclusive

Transfer Learning

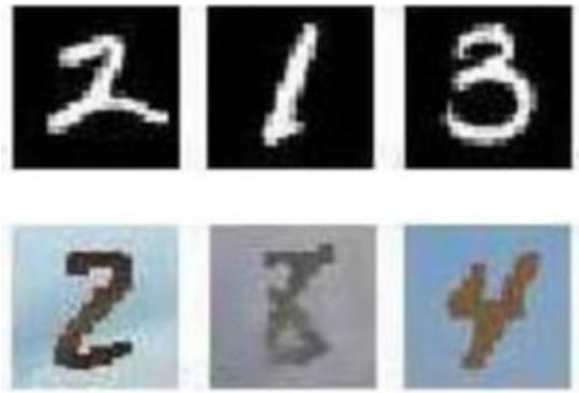
Transfer learning: A model trained on one task is reused as the initialization for another task.



Transfer Learning

Why transfer learning?

- Data distribution changes across different domains or vary over time.
- Data dependencies can be different.



Black and white → color image

Bird recognition

Airplane recognition

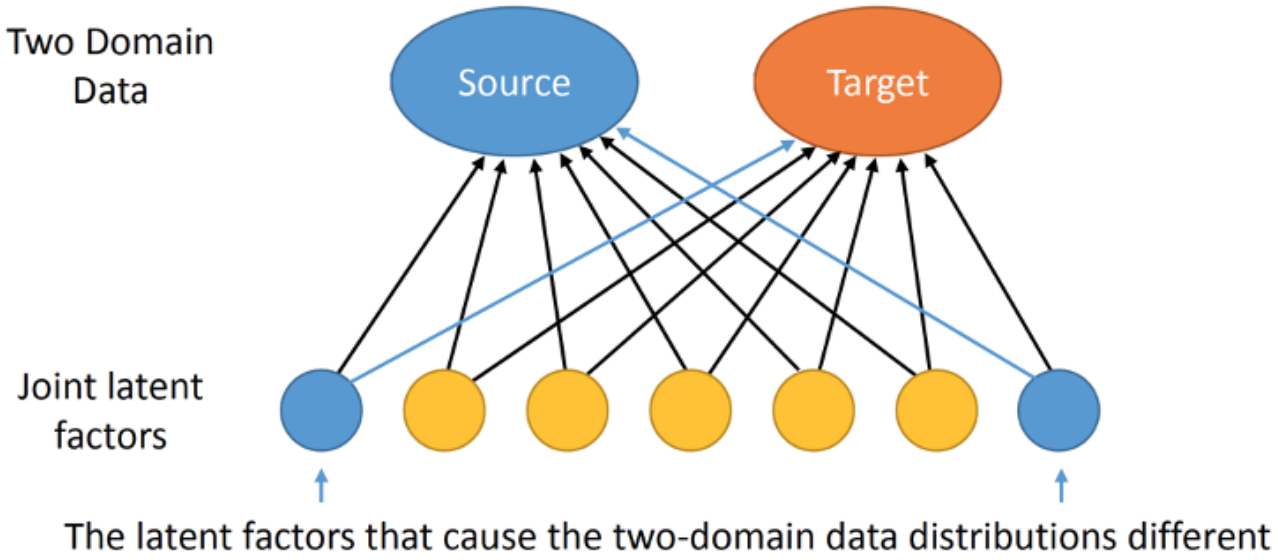
Transfer Learning

- **Instance (Data) Transfer**
  - Reweight instances of target data according to source
  - Example: importance sampling
- **Feature Transfer**
  - Mapping features of source and target data in a common space
  - Example: pre-training + tuning; Transfer Component Analysis; Domain adversarial neural networks
- **Parameter Transfer**
  - Learn target model parameters according to source model
  - Example: Net2Net; Multi-task learning

Transfer Learning

Transfer Component Analysis (TCA)

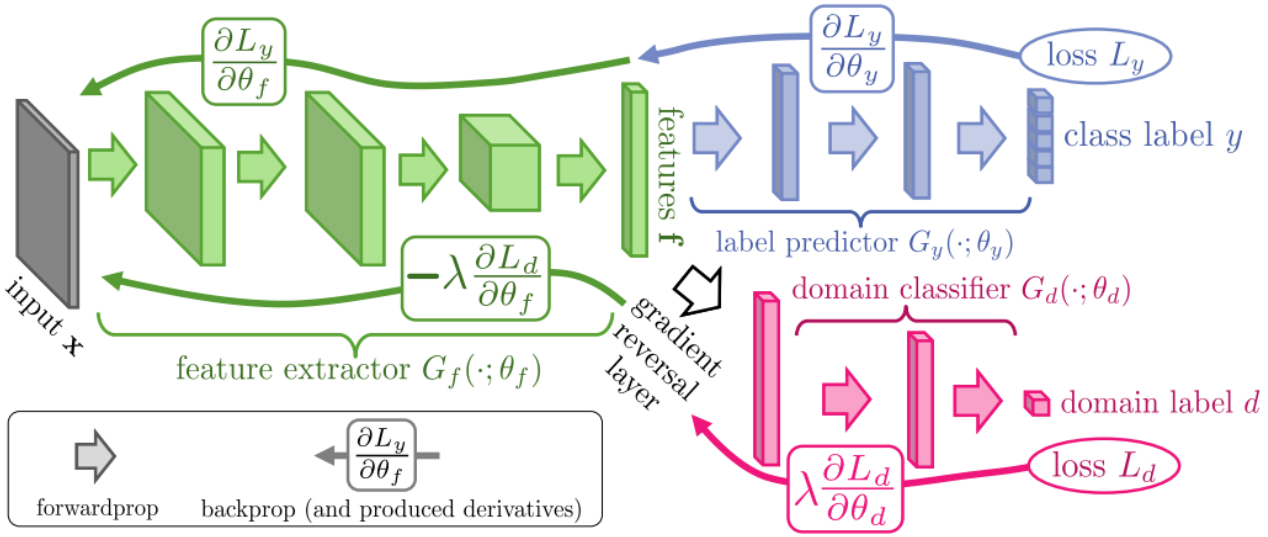
- Minimize the distance between domain distributions by projecting data onto the learned transfer components.



Transfer Learning

Domain adversarial neural networks

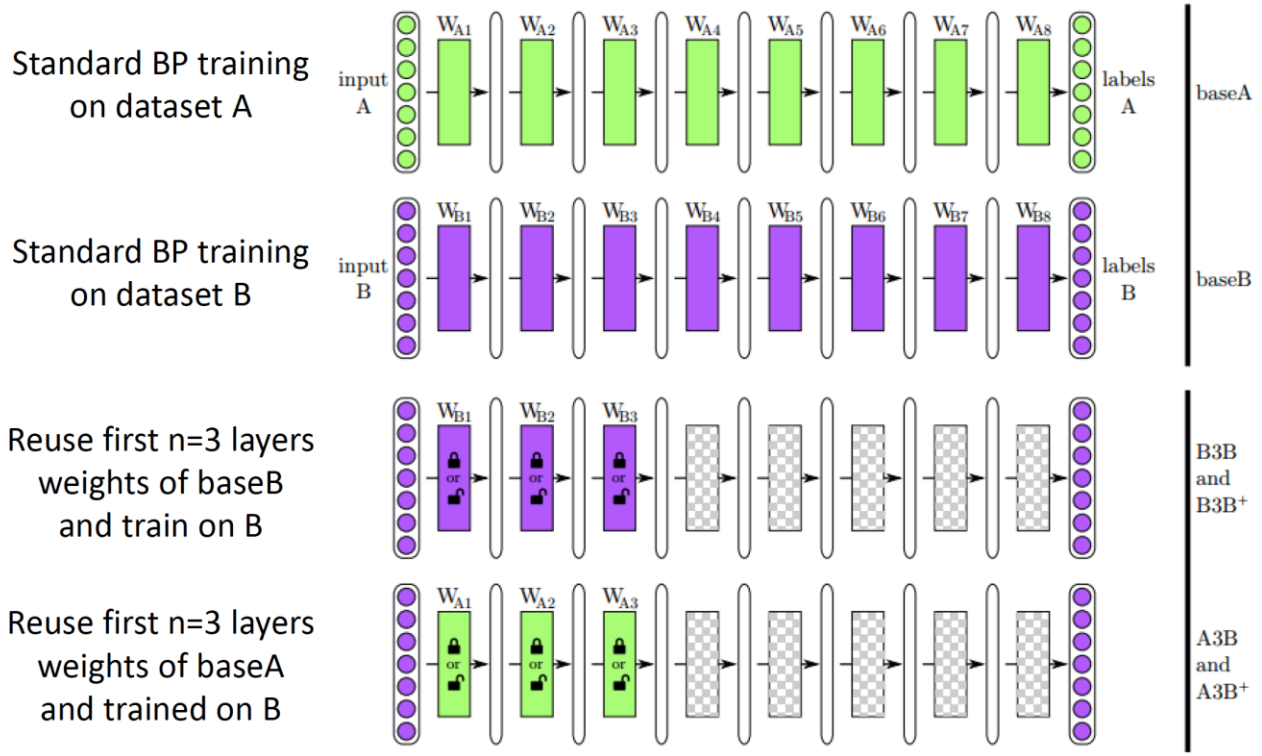
- Feature extractor and domain classifier are adversarially trained, so that feature extractor is blind to domains.



Ganin, Yaroslav, et al. "Domain-adversarial training of neural networks." *The Journal of Machine Learning Research* 17.1 (2016): 2096-2030.

Transfer Learning

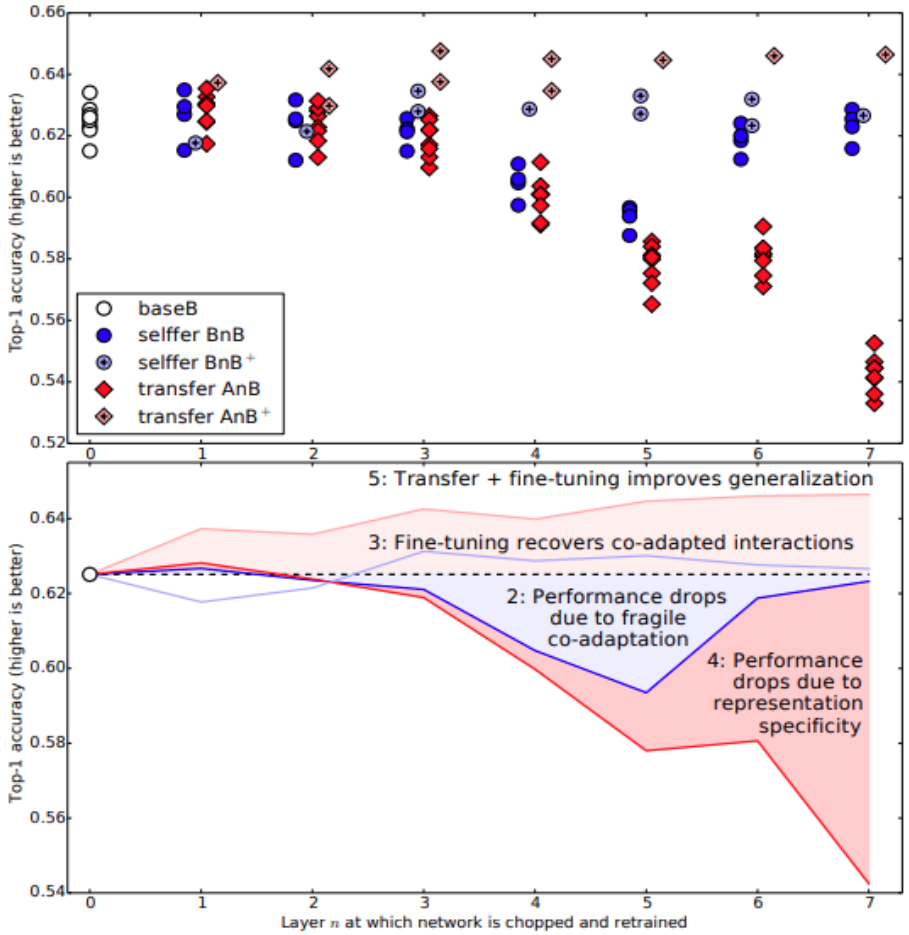
How transferable are features in deep neural networks?



Yosinski, Jason, et al. "How transferable are features in deep neural networks?." *Advances in neural information processing systems*. 2014.

Transfer Learning

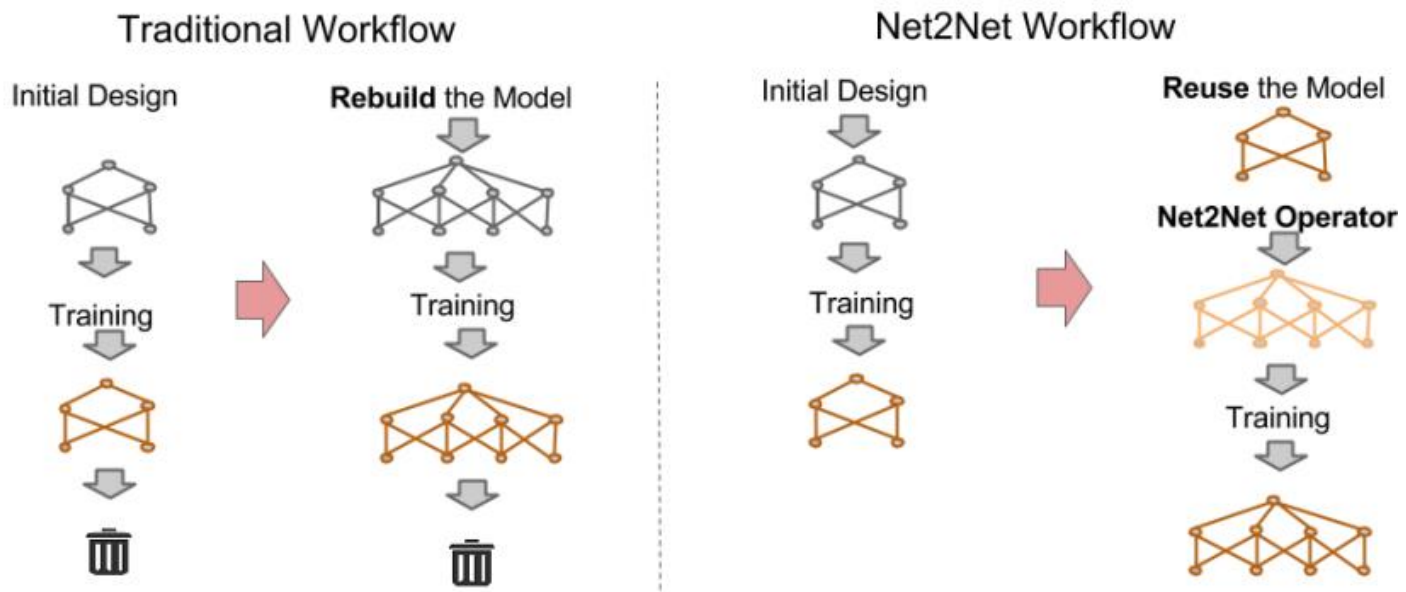
How transferable are features in deep neural networks?



Transfer Learning

Net2Net

- Net2Net reuses information of a already trained deep model to speedup training of a new model.

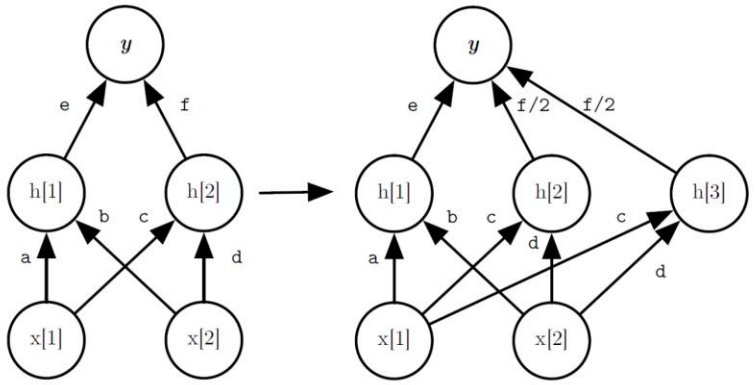


Chen, Tianqi, Ian Goodfellow, and Jonathon Shlens. "Net2net: Accelerating learning via knowledge transfer." *arXiv preprint arXiv:1511.05641* (2015).

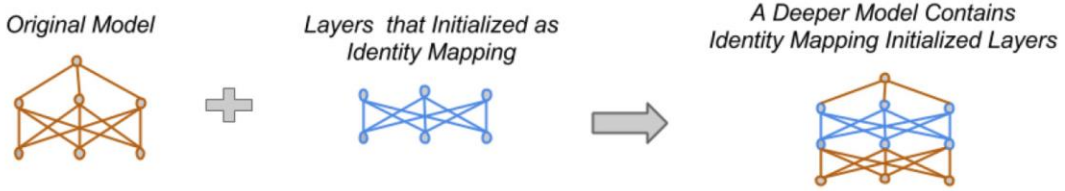
Transfer Learning

Net2Net

- Wider



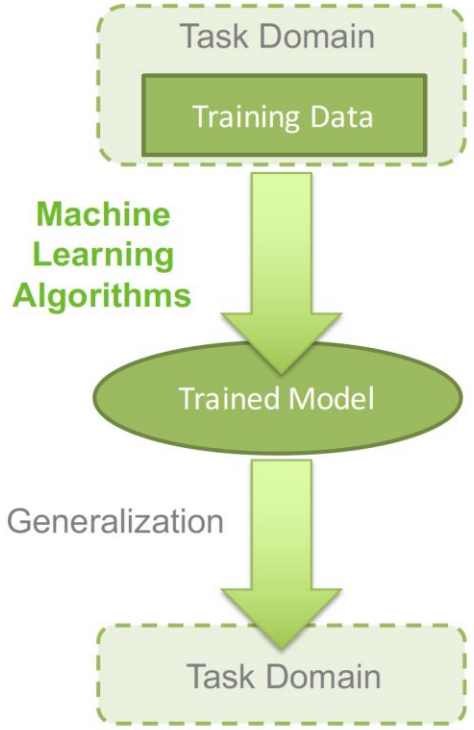
- Deeper



Multi-task Learning

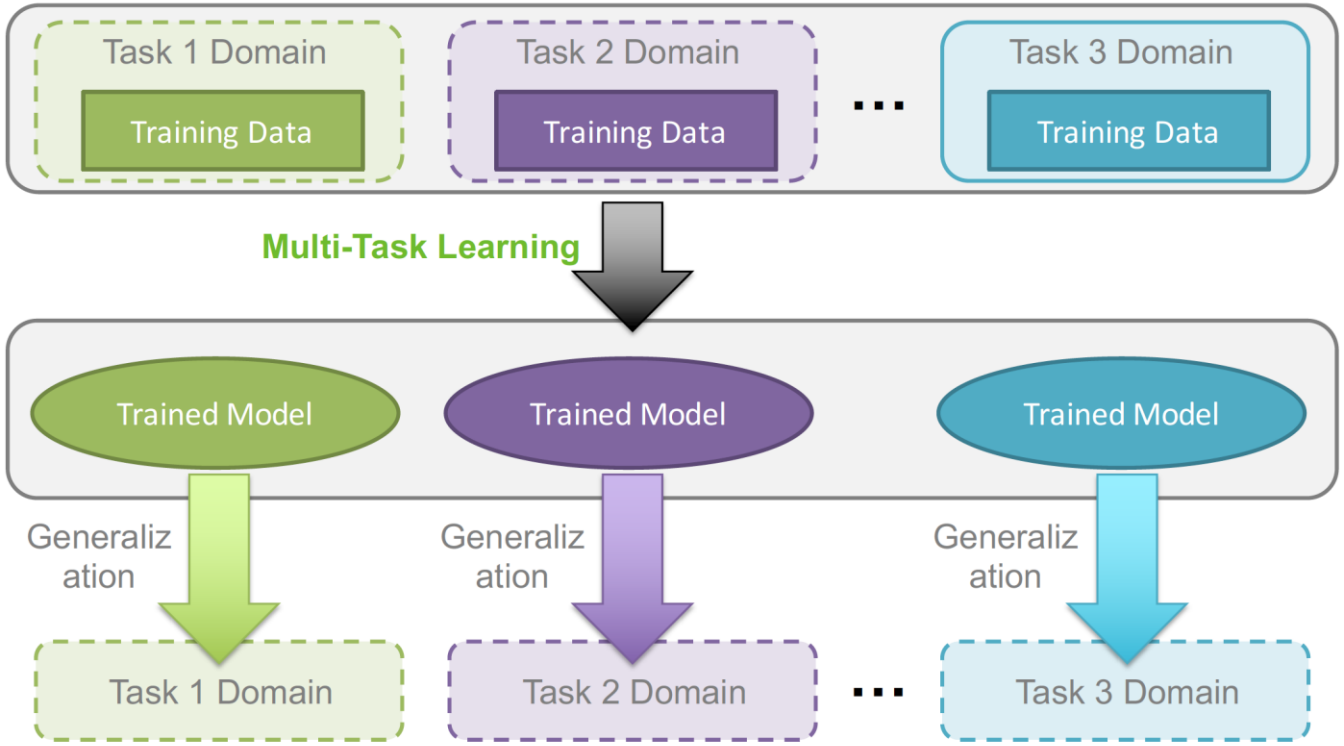
Machine learning for single task

- Elements of machine learning on single task
  - The problem (task/domain)
  - Training data
  - Learning algorithms
  - Trained model
  - Applying model on unseen data (generalization)



Multi-task Learning

Multi-task learning

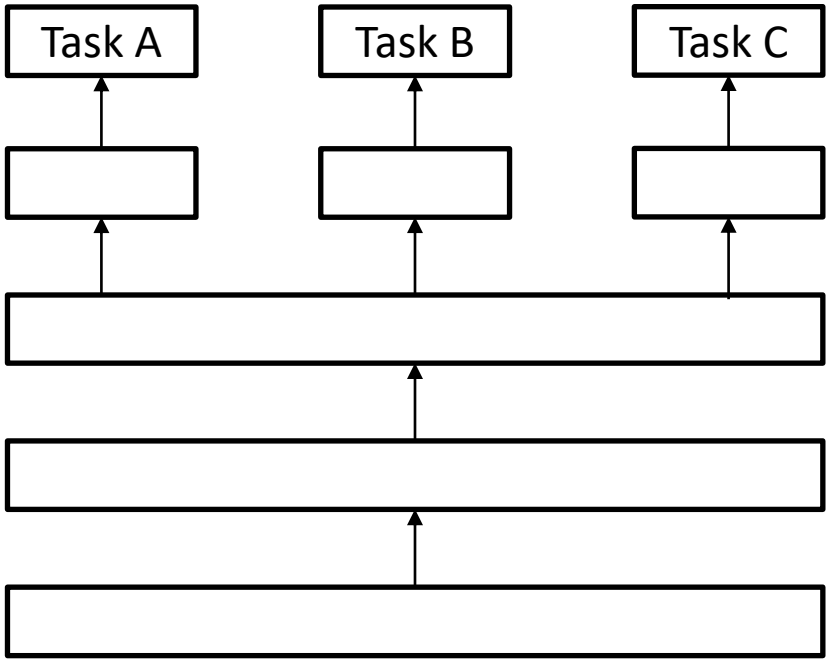


Multi-task Learning

- **Direct Parameter Sharing**
  - Examples: shared weights or activations in neural networks; shared parameters in Gaussian process
- **Structural Regularization**
  - Can be designed to incorporate various assumptions and domain knowledge
  - Can be trained using large-scale optimization algorithms on big data
  - The key is to design the regularization term that couples the tasks.

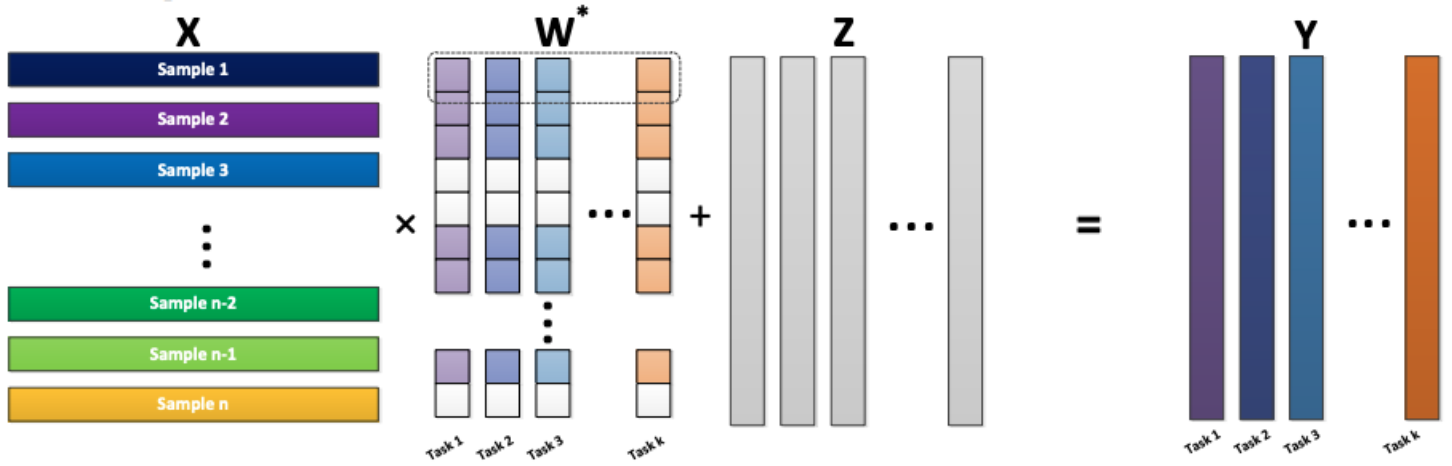
Multi-task Learning

- Direct Parameter Sharing



Multi-task Learning

- Multi-Task Learning with Joint Sparsity
  - Group sparsity: ℓ_1/ℓ_2-norm regularization
  - When $q > 1$ we have group sparsity

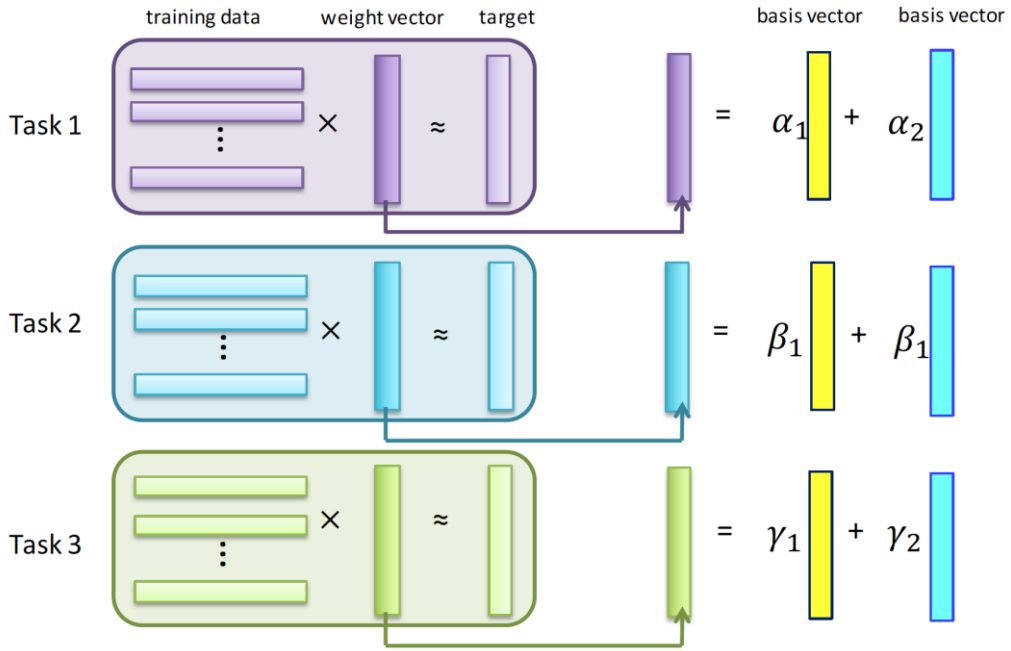


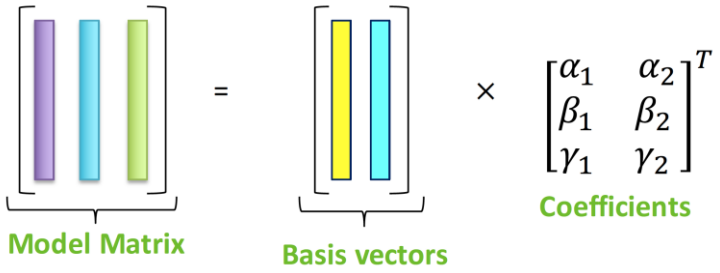
Input: m × n Weight: n × d Bias: m × k Output: m × k

$$\min_W \frac{1}{2}\|XW - Y\|_F^2 + \lambda\|W\|_{1,q} \qquad \|W\|_{1,q} = \sum_{i=1}^{d}\|\boldsymbol{w}_i\|_q$$

Multi-task Learning

- Multi-Task Learning with Low-Rank Parameters
  - Capture task relatedness via a shared low-rank structure



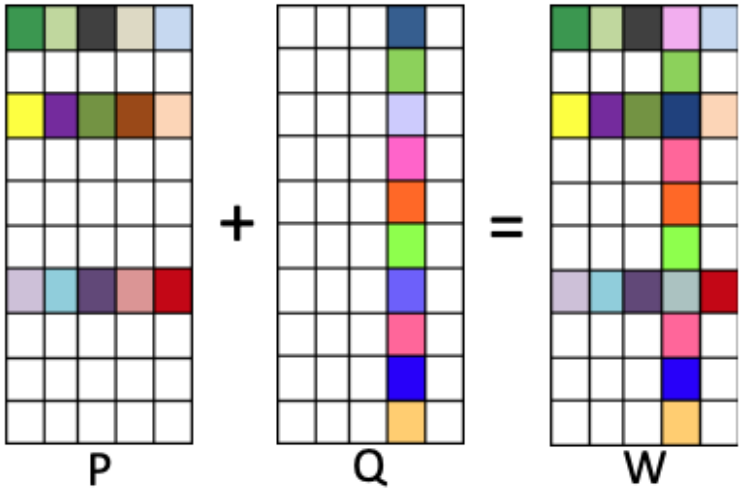


Rank minimization formulation

- $\min_W \text{Loss}(W) + \lambda \times \text{Rank}(W)$

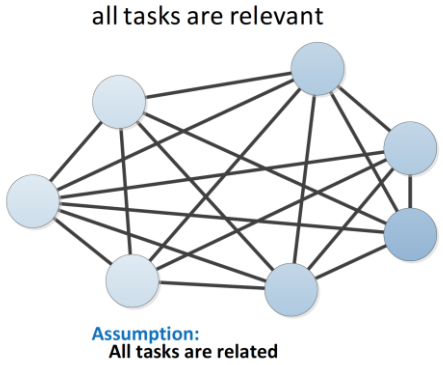
Multi-task Learning

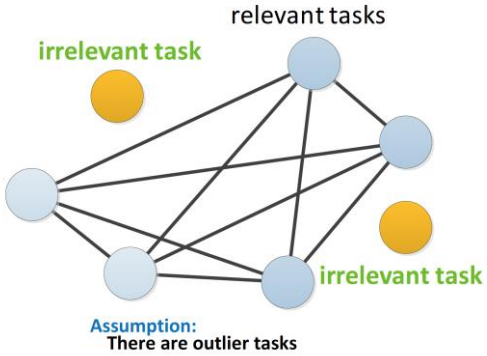
- Robust Multi-Task Learning



$$\min_{W,P,Q} \sum_{i=1}^{m} \frac{1}{mn_i} \left\| X_i^T \mathbf{w}_i - \mathbf{y}_i \right\|^2 + \lambda_1 \|P\|_{1,2} + \lambda_2 \left\| Q^T \right\|_{1,2}$$

$$s.t.\ W = P + Q,$$





Gong, Pinghua, Jieping Ye, and Changshui Zhang. "Robust multi-task feature learning." *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 2012.

Multi-task Learning

- Clustered Multi-Task Learning
  - Not all tasks are related
  - Clustered MTL
    - Tasks are clustered into groups.
    - Tasks within a group have similar models.
  - Use regularization to capture clustered structures.

Jacob, Laurent, Jean-philippe Vert, and Francis R. Bach. "Clustered multi-task learning: A convex formulation." *Advances in neural information processing systems*. 2009.