[80245013 Machine Learning, Fall, 2020]

Deep Learning (deep neural nets)

Jun Zhu
dcszj@mail.tsinghua.edu.cn
http://bigml.cs.tsinghua.edu.cn/~jun
State Key Lab of Intelligent Technology & Systems
Tsinghua University

October 13, 2020

Content

- Overview
- Backpropagation
- Convolutional Neural Networks
- Practical CNN Architectures

What is Deep Learning?

Artificial Intelligence

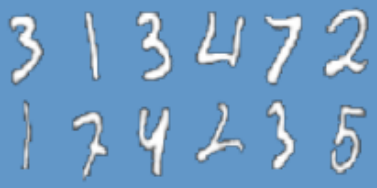
Any technique that enables computers to mimic human behavior

Machine Learning

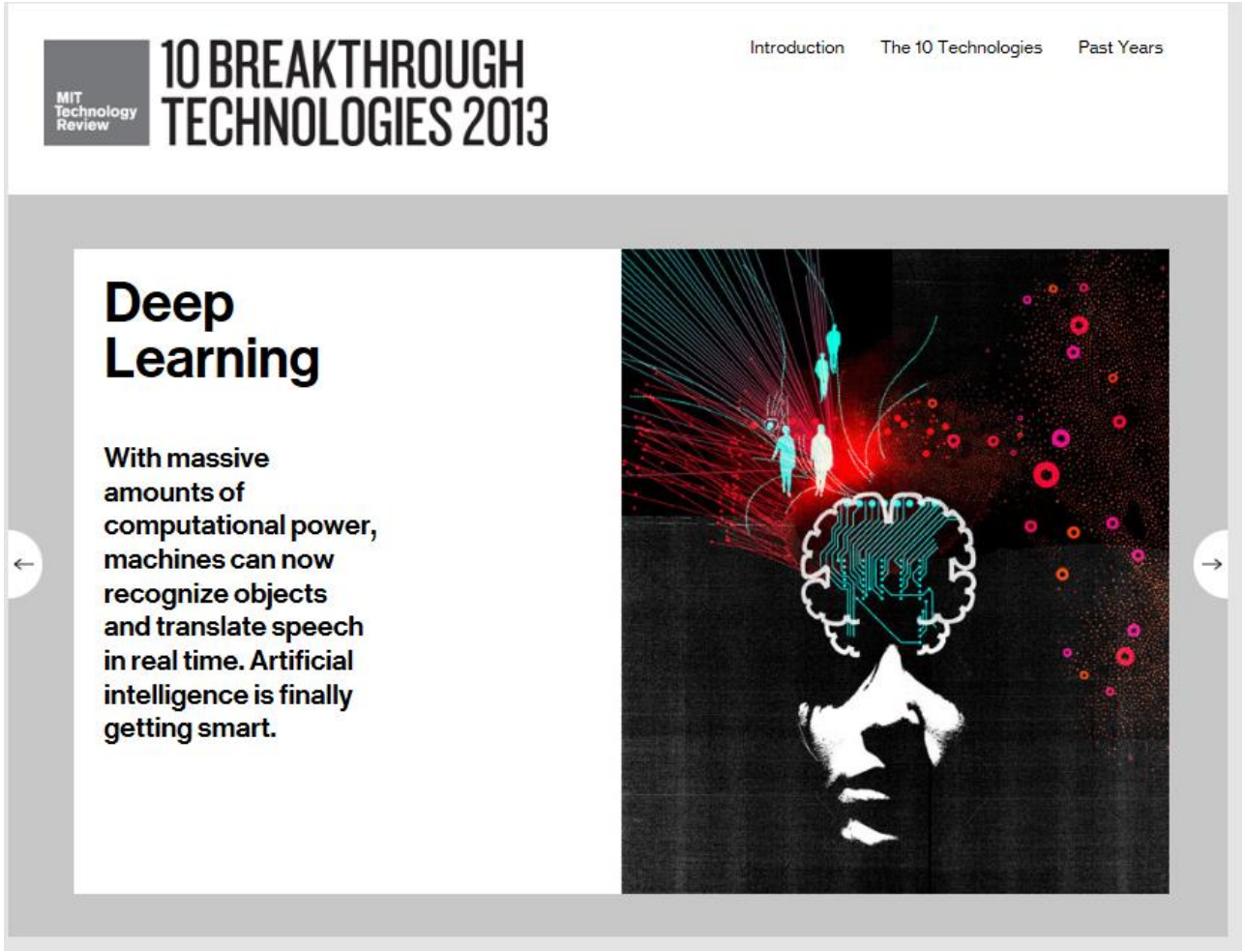
Ability to learn without explicitly being programmed

Deep Learning

Learn underlying features in data using neural networks

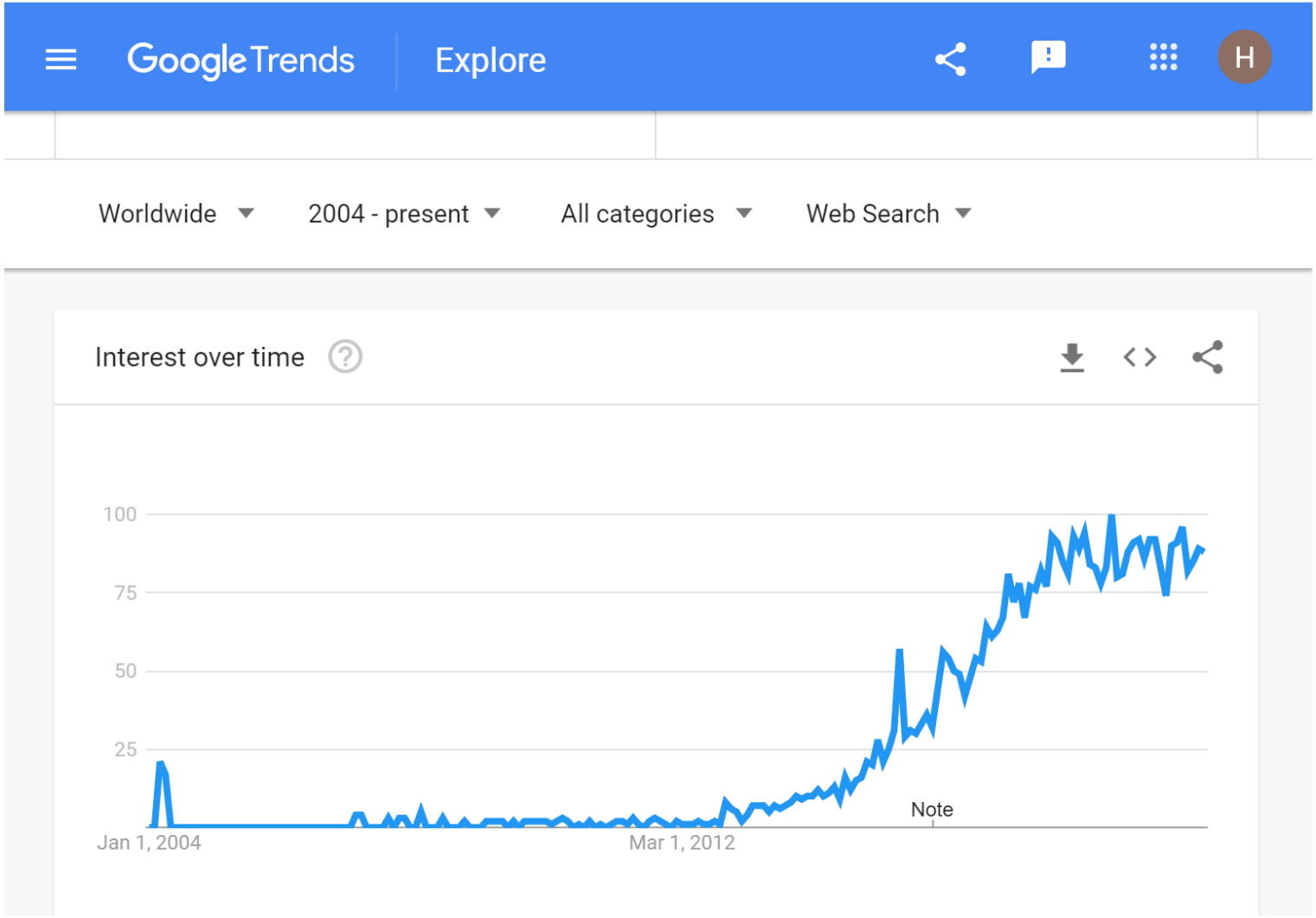


MIT 10 Breakthrough Tech 2013



http://www.technologyreview.com/featuredstory/513696/deep-learning/

Deep Learning is popular



Deep Learning in industry

Driverless car

Face identification

Speech recognition

Web search

...

Google

Alibaba.com

HUAWEI

intel

...

Microsoft

Baidu 百度

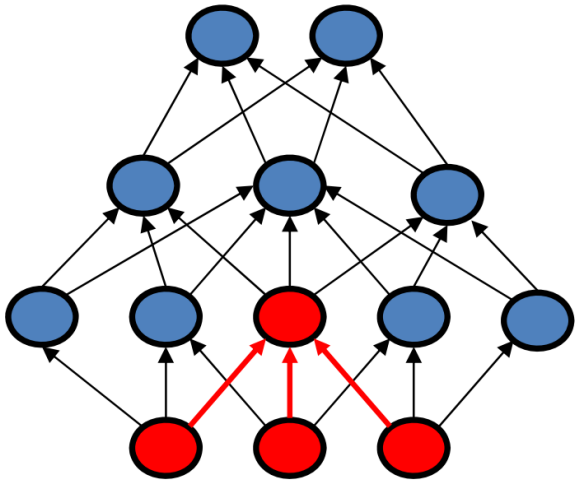
Tencent 腾讯

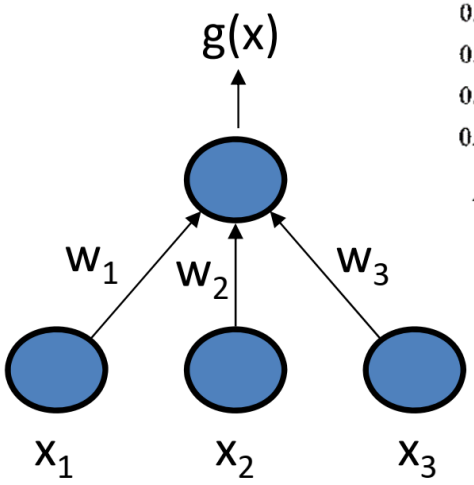
AMD

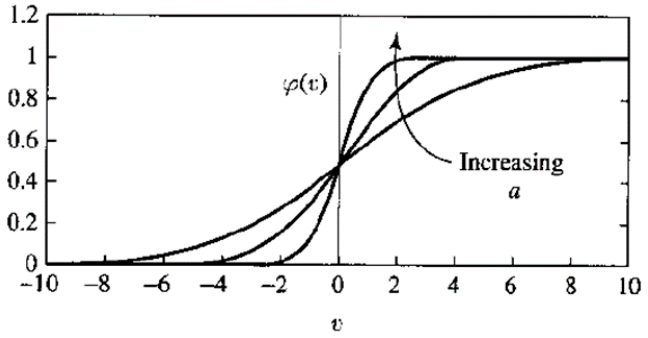
Deep Neural Networks

Neural network
Back propagation
Nature

1986





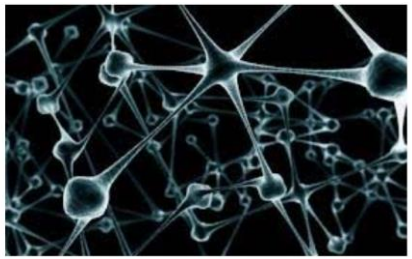


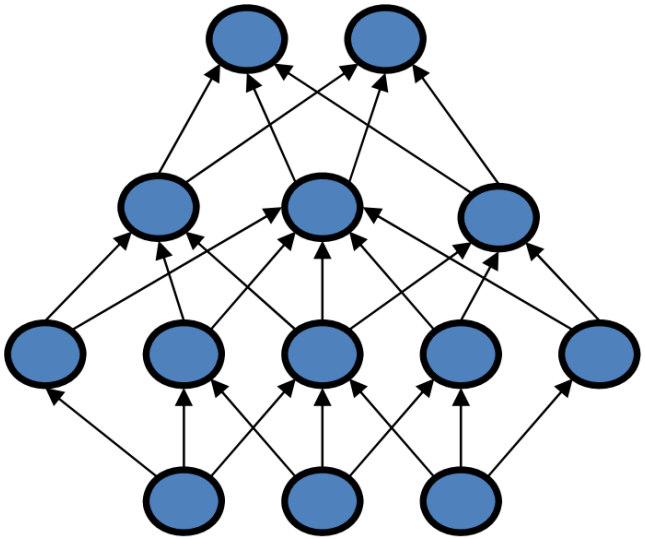
$$g(\mathbf{x}) = f\left(\sum_{i=1}^{d}(x_i w_i + w_0)\right) = f(\mathbf{w}^\top \mathbf{x})$$

Deep Neural Networks

Neural network
Back propagation

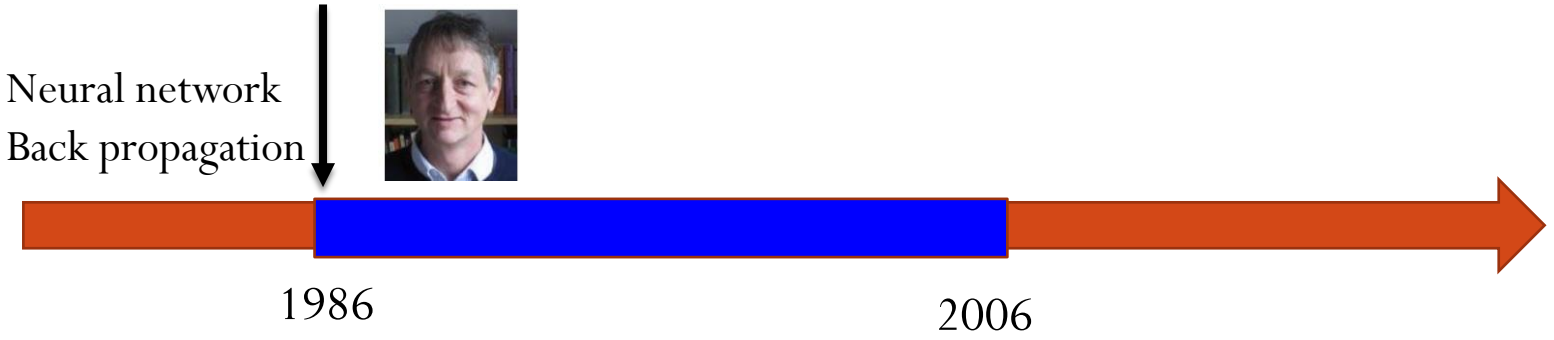
1986





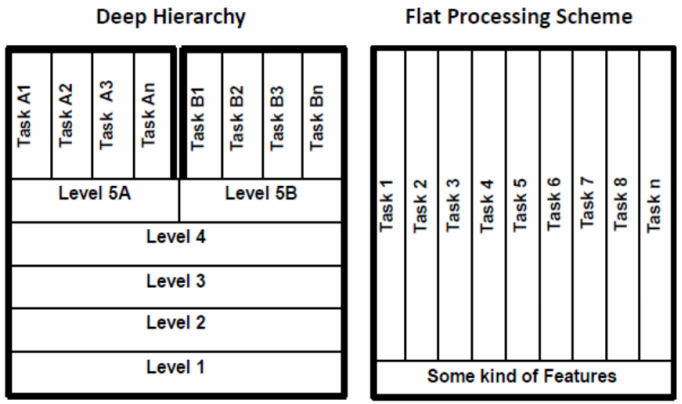
- Solve general learning problems
- Tided with biological system
- **But it is given up…**
  - Hard to train
  - Insufficient computational resources
  - Small training sets
  - Does not work well

Deep Neural Networks



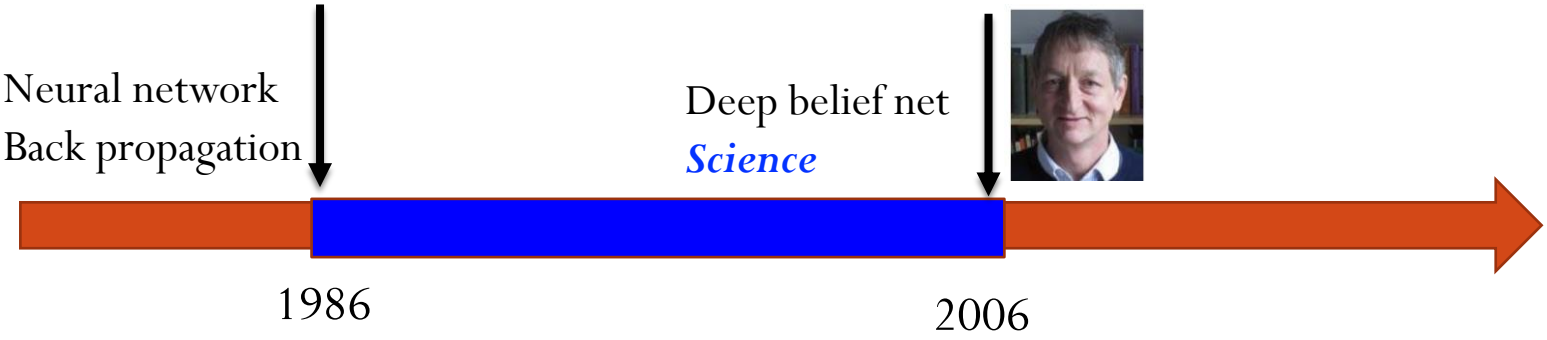
- SVM
- Boosting
- Decision tree
- KNN
-

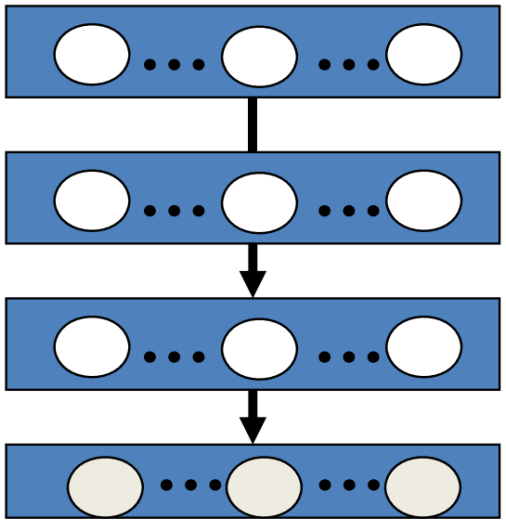
- Flat structures
- Loose tided with biological system
- Specific methods for specific tasks
  - Hand crafted features (SIFT, LBP, HOG)



Kruger et al. TPAMI'13

Deep Neural Networks



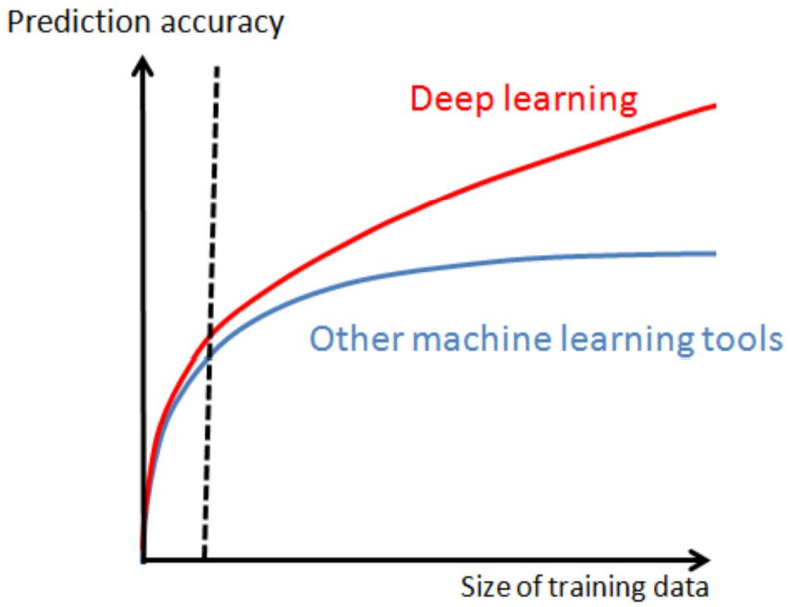


- Unsupervised & Layer-wised pretraining
- Better designs for modeling and training (normalization, nonlinearity, dropout)
- New development of computer architectures
  - GPU
- Large scale databases

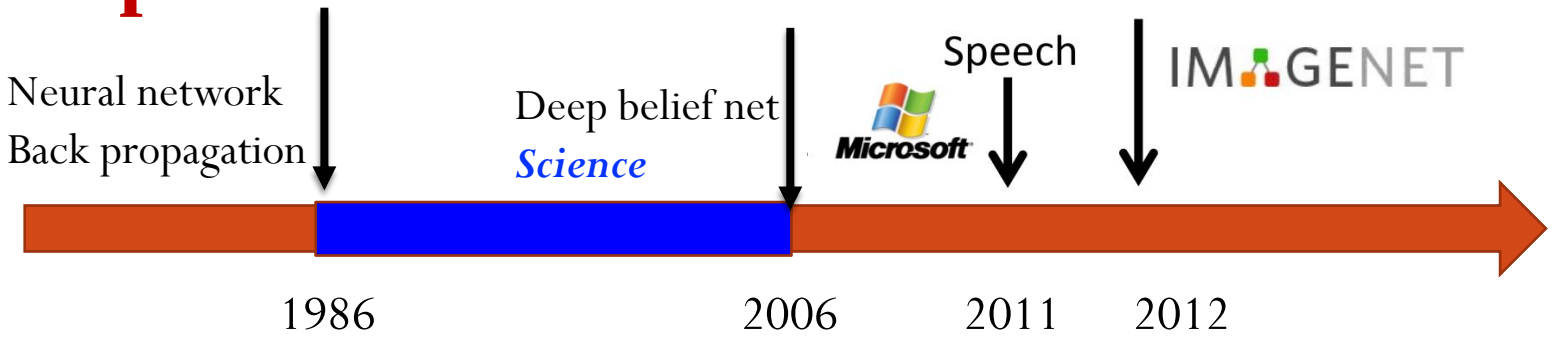
Big Data!

Machine Learning with Big Data

- **Machine learning with small data**: overfitting, reducing model complexity (capacity)
- **Machine learning with big data**: underfitting, increasing model complexity, optimization, computation resource



Deep Neural Networks



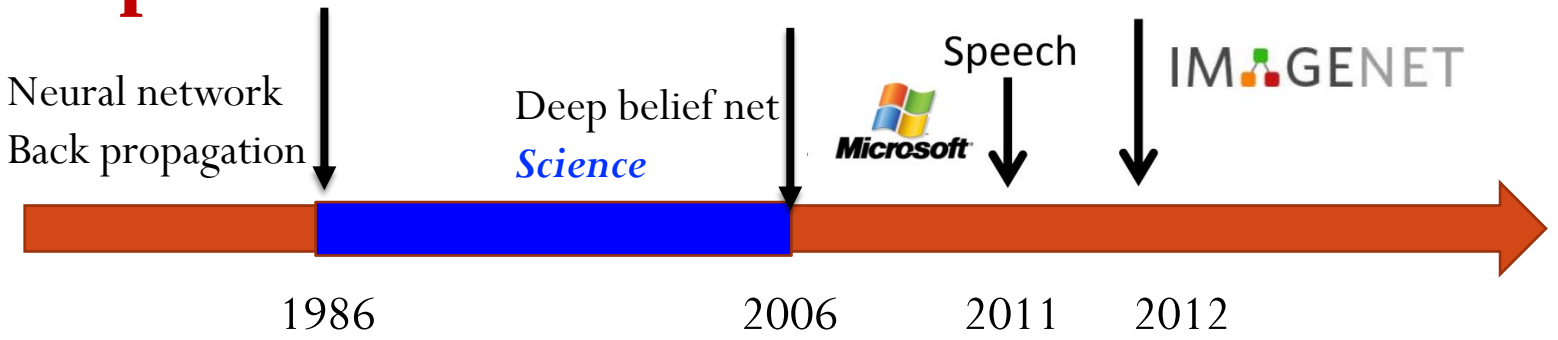
- ImageNet 2013 – image classification challenge

Rank	Name	Error rate	Description
1	NYU	0.11197	Deep learning
2	NUS	0.12535	Deep learning
3	Oxford	0.13555	Deep learning

- ImageNet 2013 – object detection challenge

Rank	Name	Mean Average Precision	Description
1	UvA-Euvision	0.22581	Hand-crafted features
2	NEC-MU	0.20895	Hand-crafted features
3	NYU	0.19400	Deep learning

Deep Neural Networks



- ImageNet 2014 – image classification challenge

Rank	Name	Error rate	Description
1	Google	0.06656	Deep learning
2	Oxford	0.07325	Deep learning
3	MSRA	0.08062	Deep learning

- ImageNet 2014 – object detection challenge

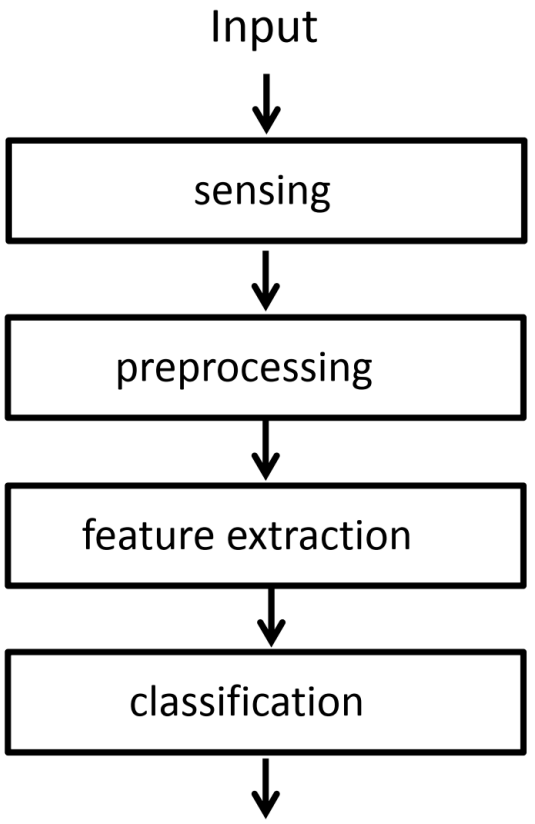
Rank	Name	Mean Average Precision	Description
1	Google	0.43933	Deep learning
2	CUHK	0.40656	Deep learning
3	DeepInsight	0.40452	Deep learning
4	UvA-Euvision	0.35421	Deep learning
5	Berkley Vision	0.34521	Deep learning

Deep Learning Changes the Design Cycle

- How do machine learning, computer vision, and speech recognition people study pattern recognition problems in different ways?

- How does deep learning change the design cycle of a AI system?

AI System

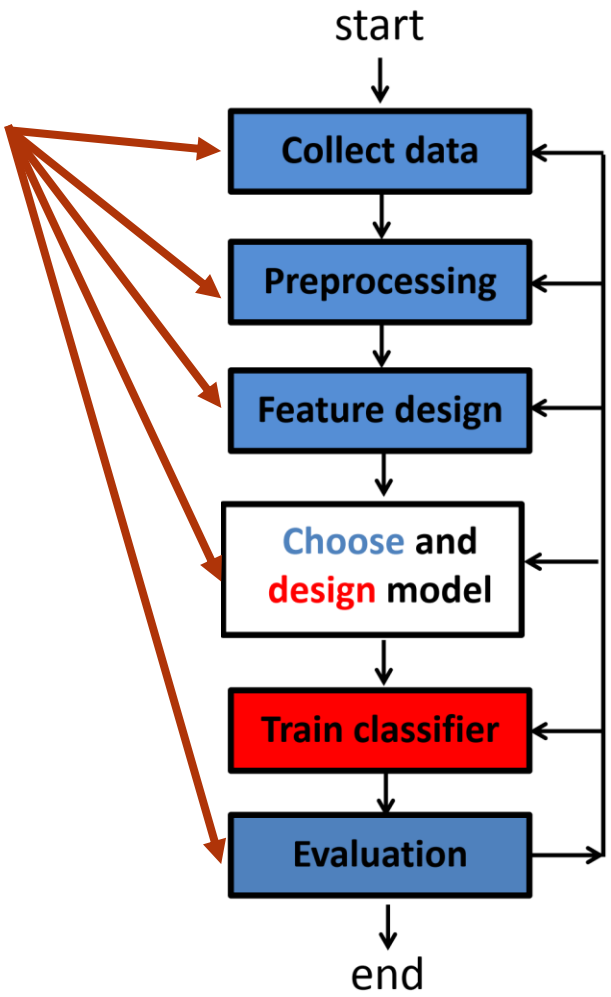


Decision: "salmon" or "sea bass"

Design Cycle

Domain knowledge

- Preprocessing and feature ***design*** may lose useful information and not be optimized, since they are not parts of an end-to-end learning system



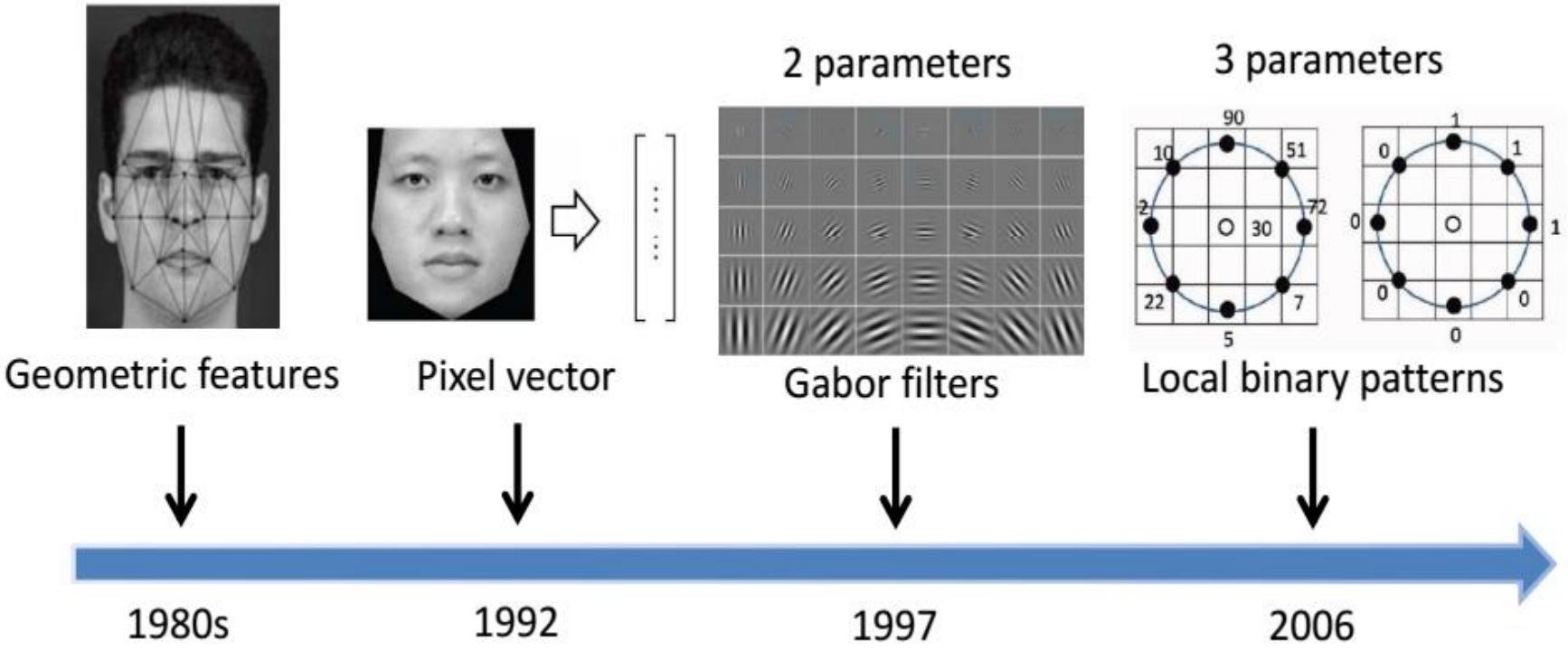
Interest of people working on computer vision, speech recognition, medical image processing,...

Interest of people working on machine learning

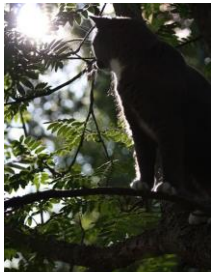
Interest of people working on machine learning and computer vision, speech recognition, medical image processing,...

Feature Engineering

- Handcrafted Features for Face Recognition

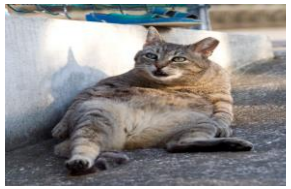


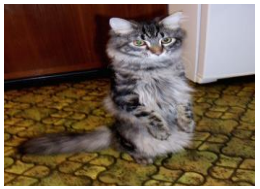
Challenges of Feature Engineering



Illumination

Deformation





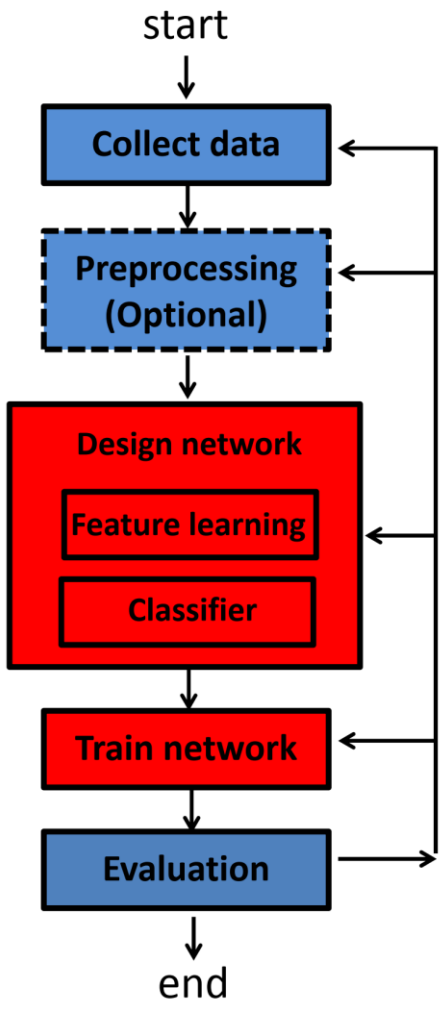
Occlusion

Background Clutter

Intraclass variation

Design Cycle with Deep Learning

- Learning plays a bigger role in the design circle
- Feature learning becomes part of the end-to-end learning system
- Preprocessing becomes optional means that several pattern recognition on steps can merge into one end-to-end learning system
- Feature learning make the key difference
- We underestimated the importance of data collection and evaluation



Deep Learning = The Entire Machine is Trainable

- Traditional Pattern Recognition: Fixed/Handcrafted Feature Extractor

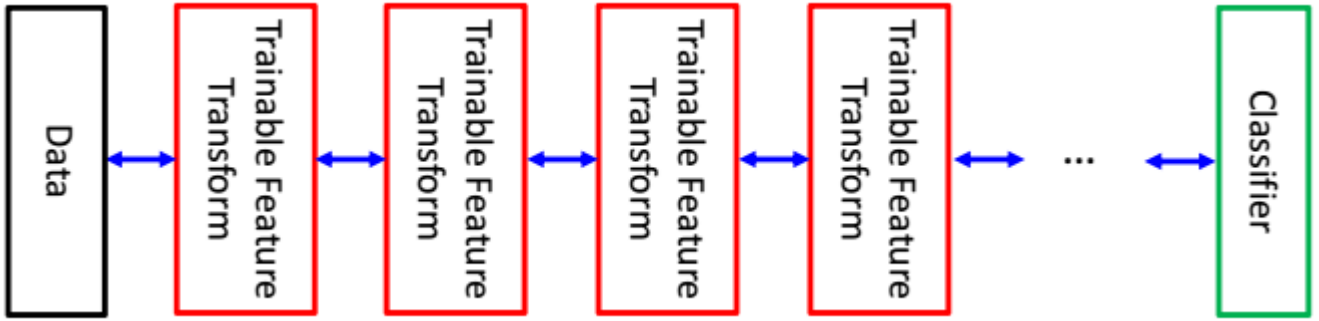
- Mainstream Modern Pattern Recognition: Unsupervised mid-level features

- Deep Learning: Representations are hierarchical and trained

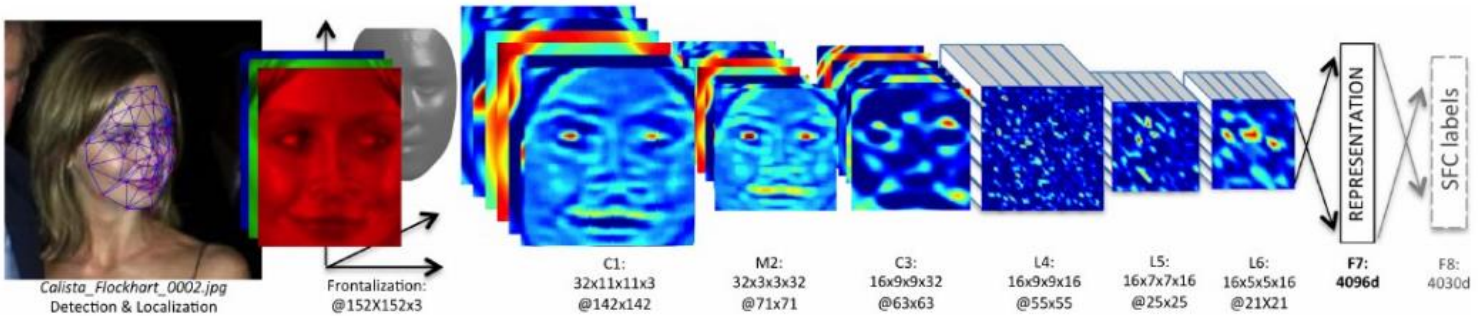
Deep Learning Means Feature Learning

- Deep learning is about learning hierarchical feature representations

$$\mathbf{y} = F(\mathbf{W}^k(F(\mathbf{W}^{k-1} \cot F(\cdots F(\mathbf{W}^0 \cdot \mathbf{x})))))$$

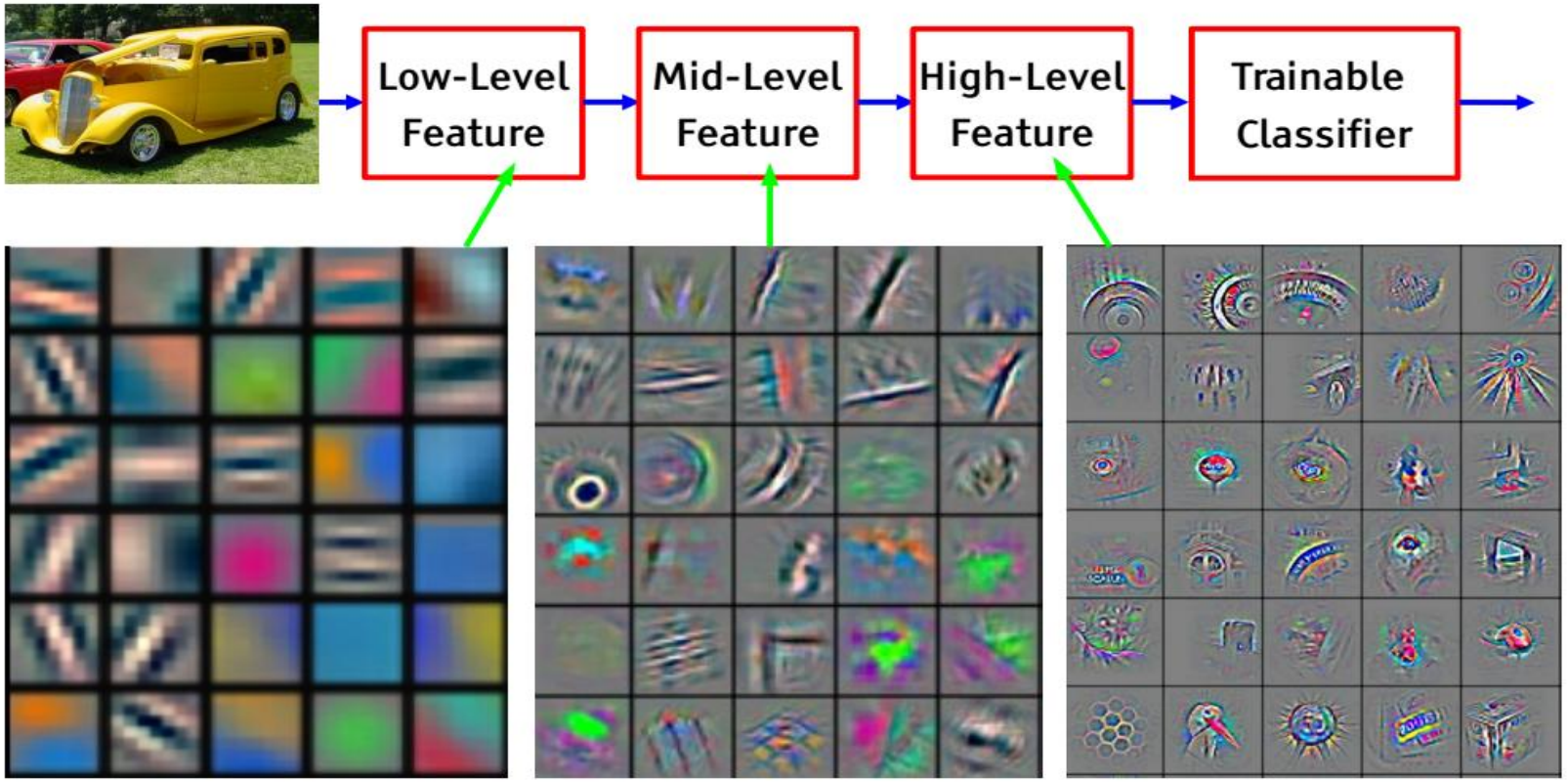


- Deep learning: Hierarchical; nonlinear



DL= Learning Hierarchical Representations

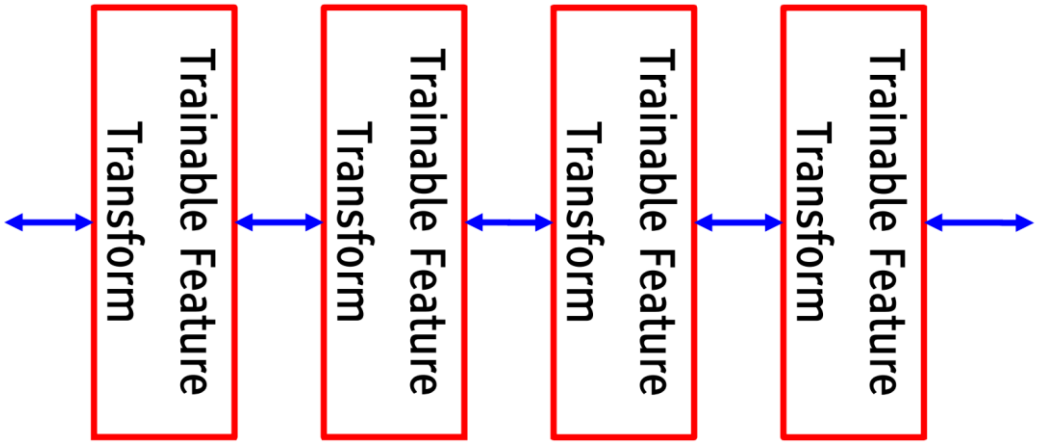
- It's deep if it has more than one stage of non-linear feature transformation



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

Trainable Feature Hierarchy

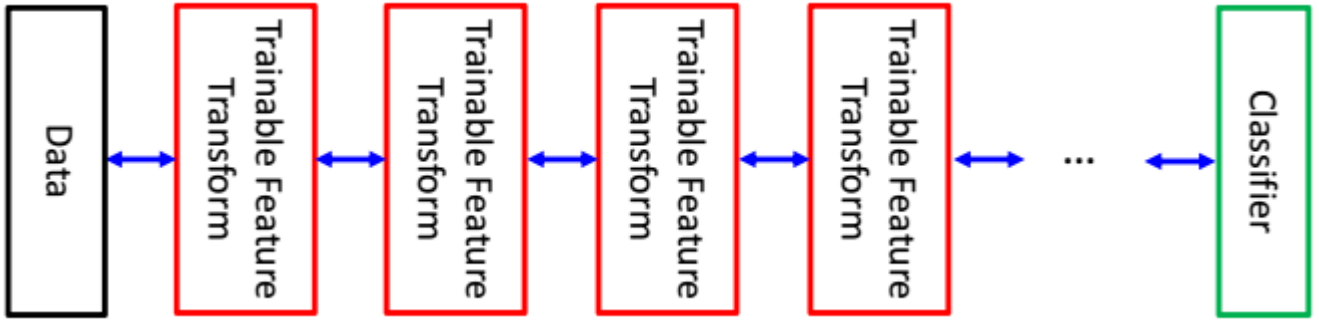
- Hierarchy of representations with increasing level of abstraction
- Each stage is a kind of trainable feature transform
  - Image recognition
    - Pixel → edge → texton → motif → part → object
  - Text
    - Character → word → word group → clause → sentence → story



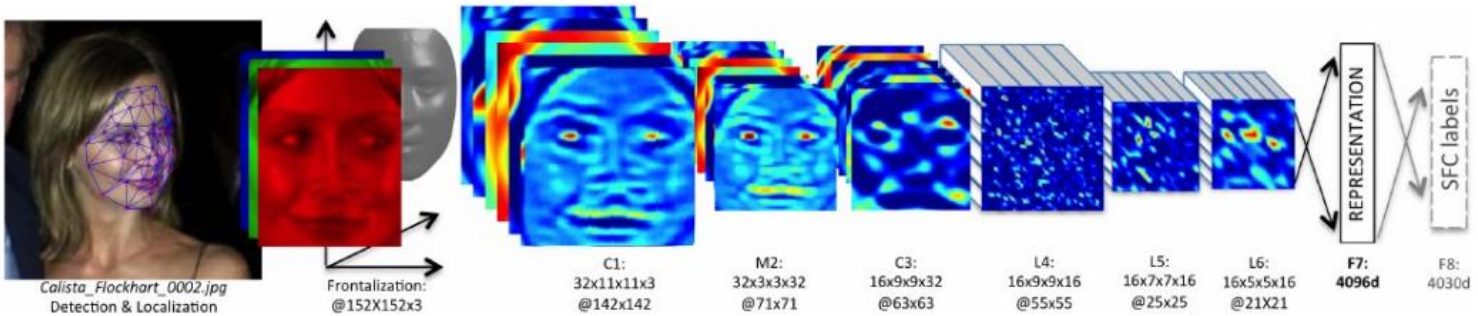
Deep Learning means Feature Learning

- Deep learning is about learning hierarchical feature representations

$$\mathbf{y} = F(\mathbf{W}^k(F(\mathbf{W}^{k-1}F(\cdots F(\mathbf{W}^0 \cdot \mathbf{x})))))$$

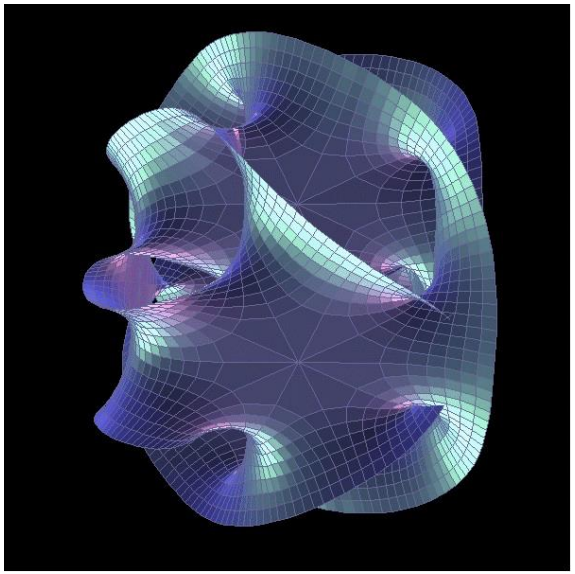


- Deep learning: Hierarchical; nonlinear



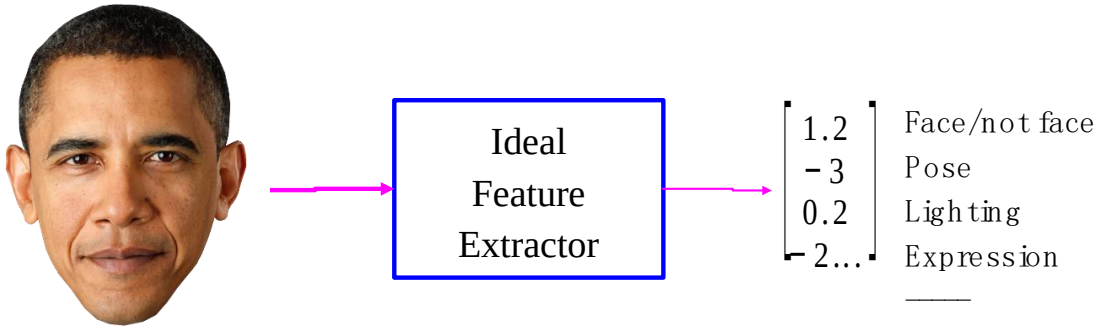
What are good features?

- Learning Representations of Data:
  - Discovering & disentangling the independent explanatory factors
- The Manifold Hypothesis:
  - Natural data lives in a low-dimensional (non-linear) manifold



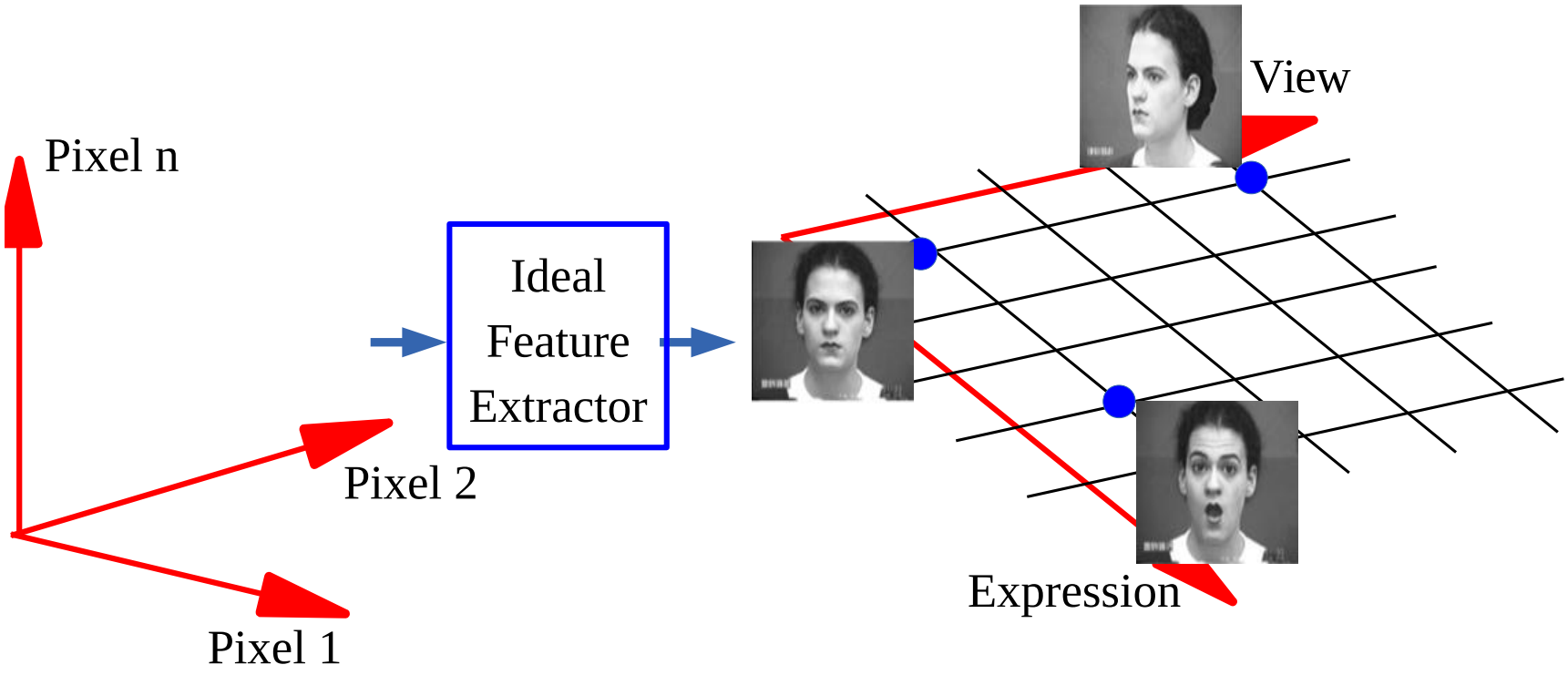
Discovering the Hidden Structure in High-Dimensional Data

- Example: all face images of a person
  - 1000x1000 pixels = 1,000,000 dimensions
  - But the face has 3 cartesian coordinates and 3 Euler angles
  - And humans have less than about 50 muscles in the face
  - Hence the manifold of face images for a person has <56 dimensions
- The perfect representations of a face image:
  - Its coordinates on the face manifold



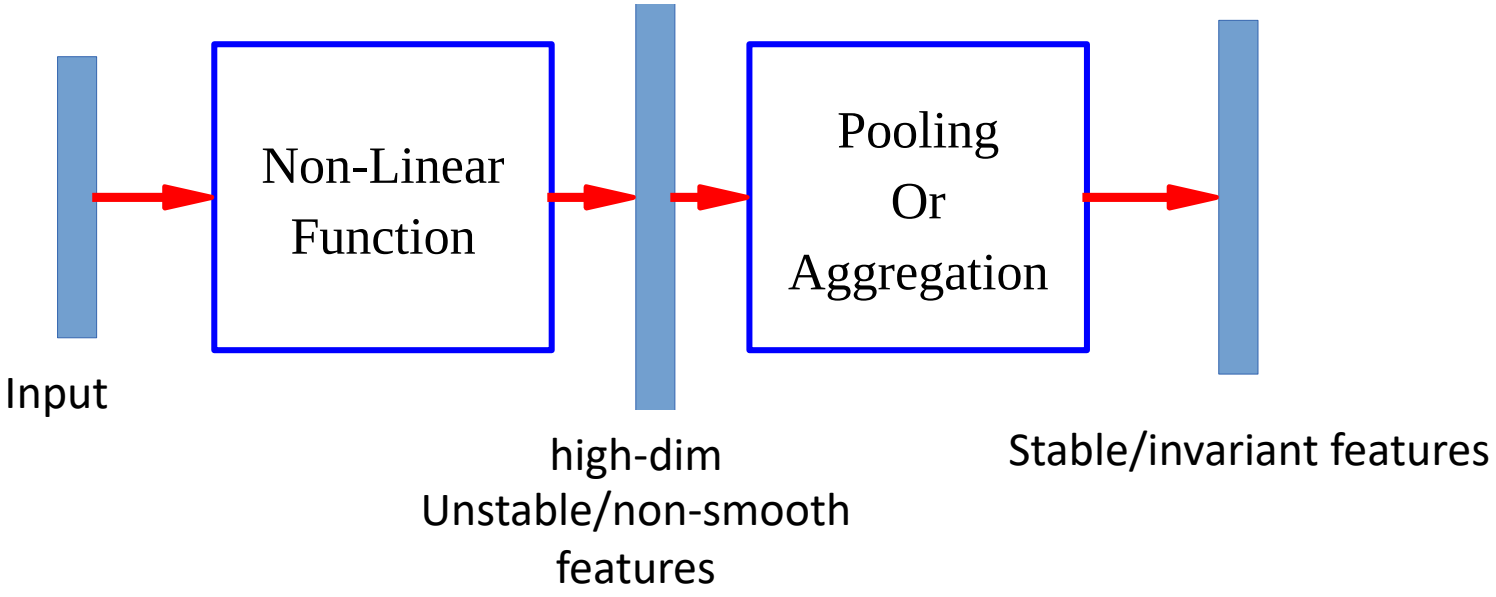
Disentangling factors of variation

- The Ideal Disentangling Feature Extractor



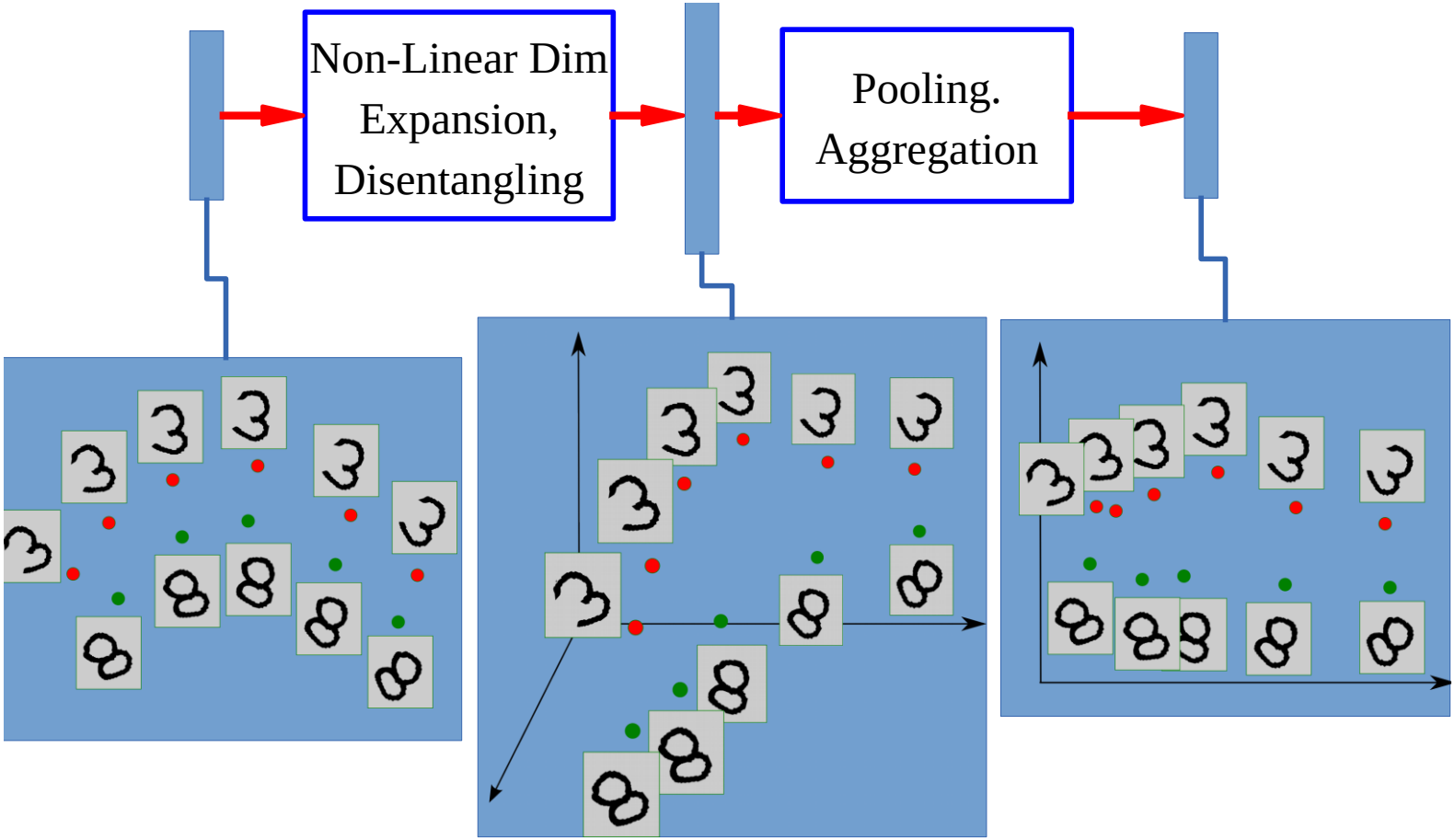
Basic Idea for Invariant Feature Learning

- Embed the input non-linearly into a high dimensional space
  - In the new space, things that were non separable may become separable
- Pool regions of the new space together
  - Bringing together things that are semantically similar, e.g., pooling



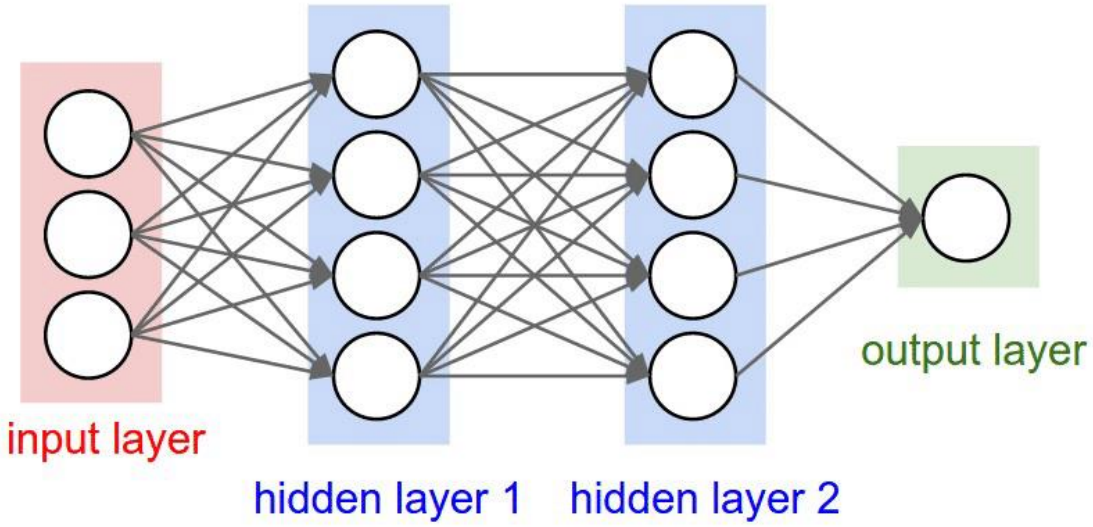
Basic Idea for Invariant Feature Learning

- Entangled data manifolds



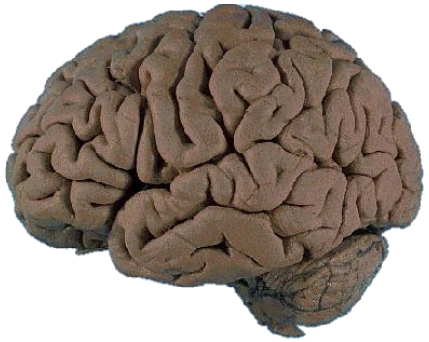
Questions

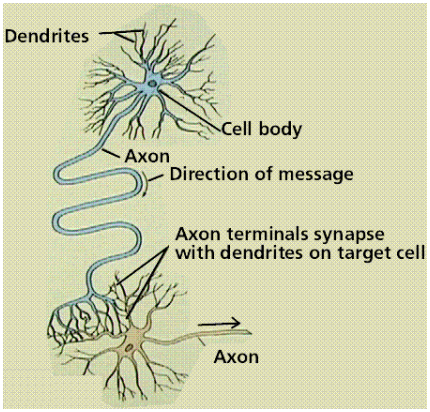
- How do we compose the network that performs the requested function?

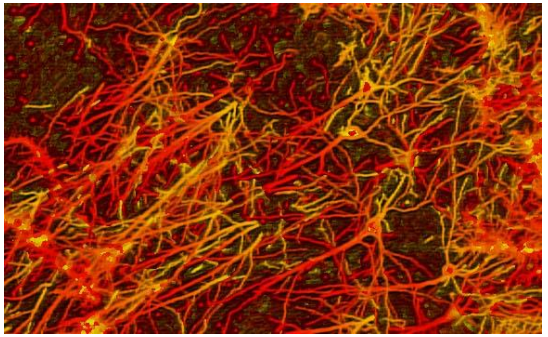


Deep Learning Models

How the human brain learns?







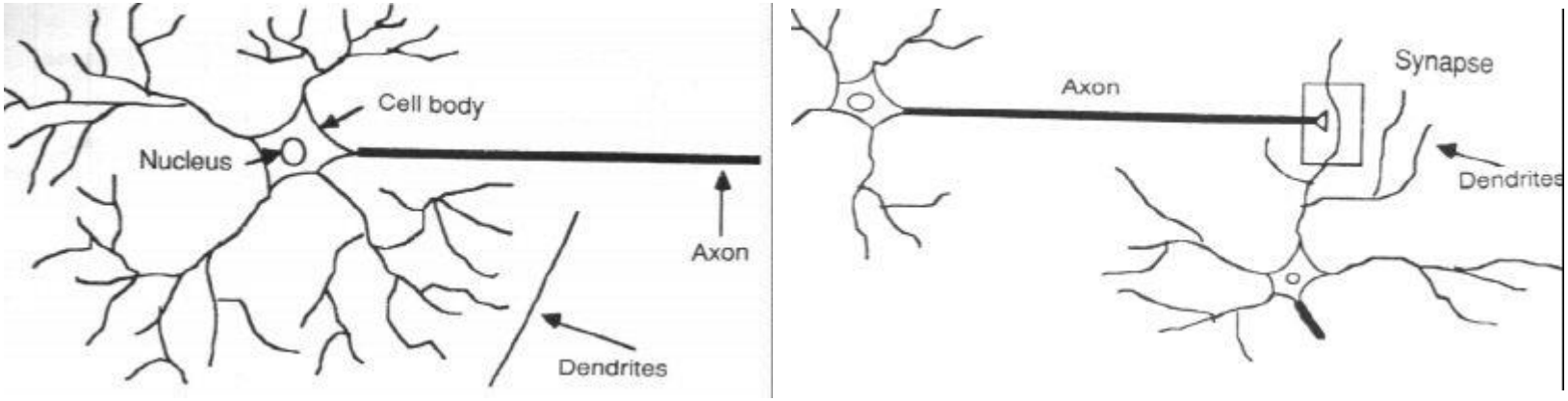
The business end of this is made of lots of these joined in networks like this

Much of our own “computations” are performed in/by this network

Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes

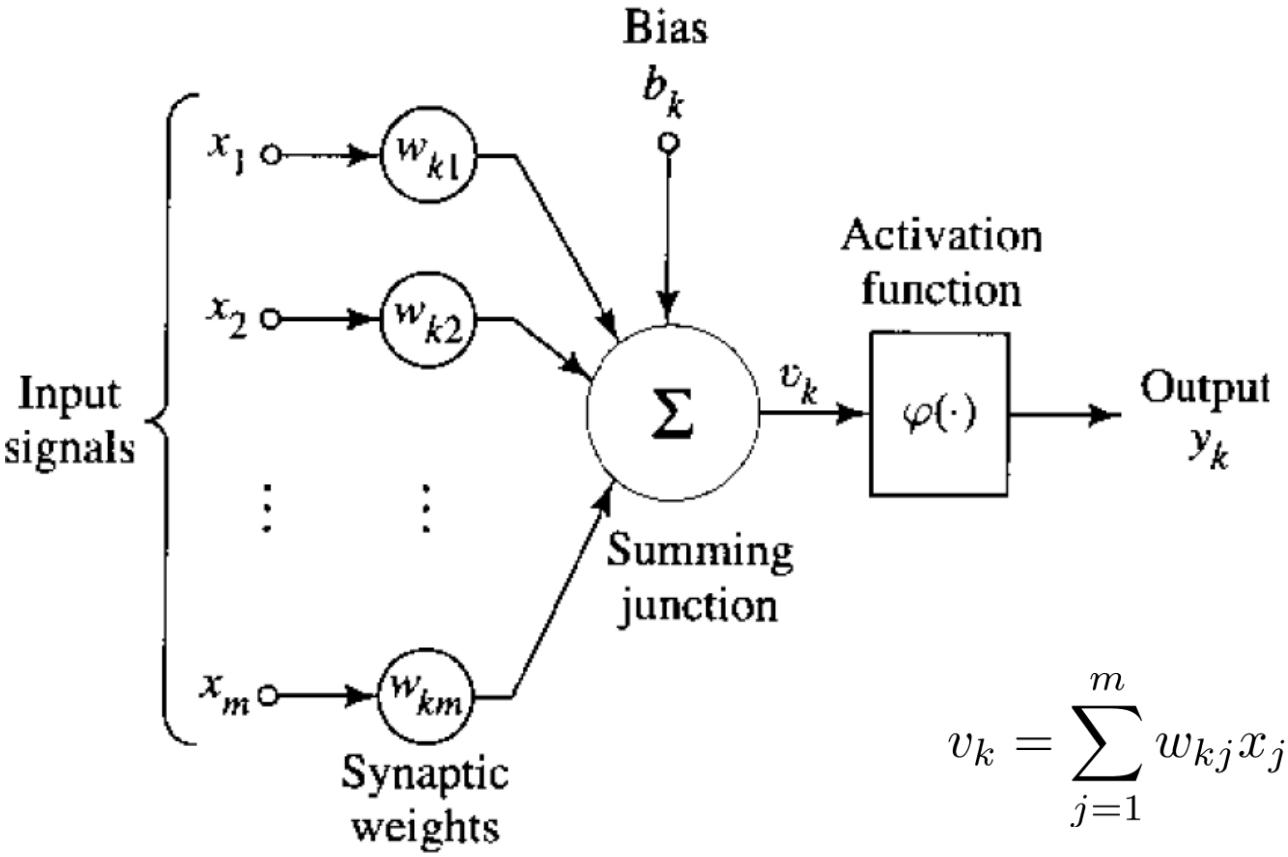
How the human brain learns?

- A typical neuron



- Learning occurs by changing the effectiveness of the synapses so that the influence of one neuron on another changes

Model of a neuron

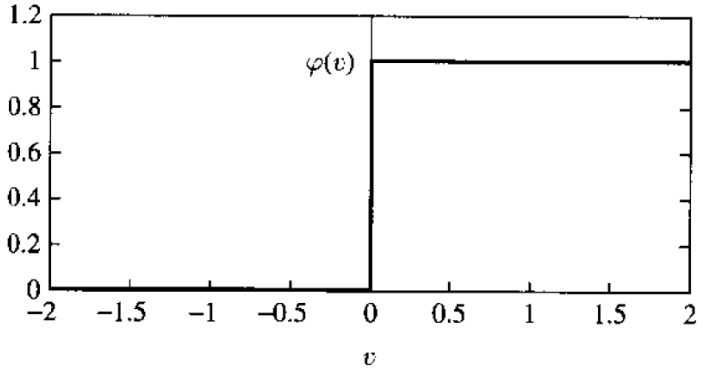


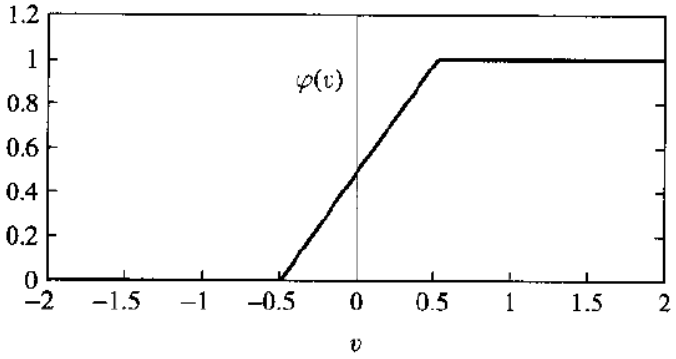
$$v_k = \sum_{j=1}^{m} w_{kj} x_j + b_k$$

$$y_k = \psi(v_k)$$

Activation function

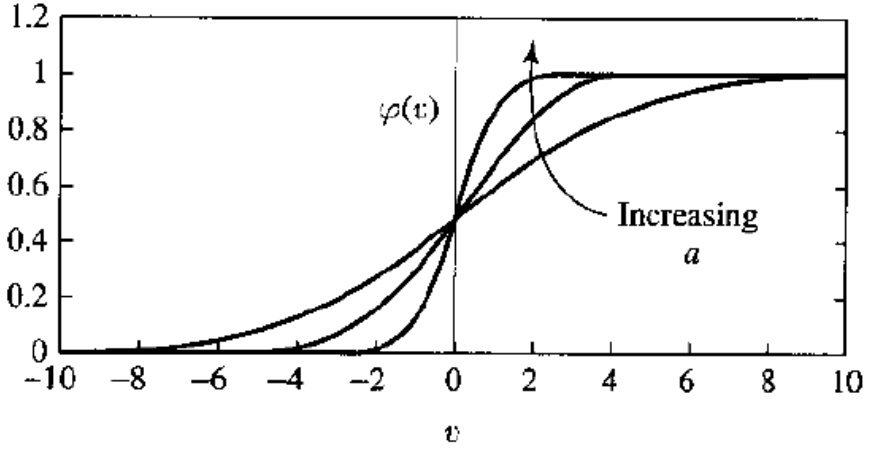
- Threshold function & piecewise linear function:





- Sigmoid function

$$\psi_\alpha(v) = \frac{1}{1+\exp(-\alpha v)}$$



$a \to \infty$: step function

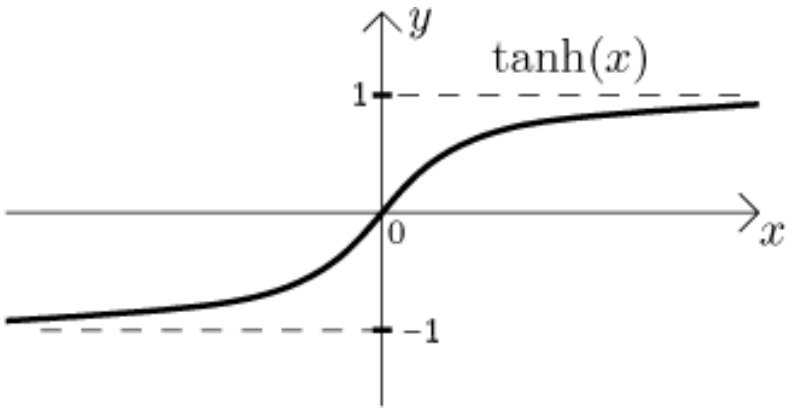
Activation function with negative values

- Threshold function & piecewise linear function:

$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

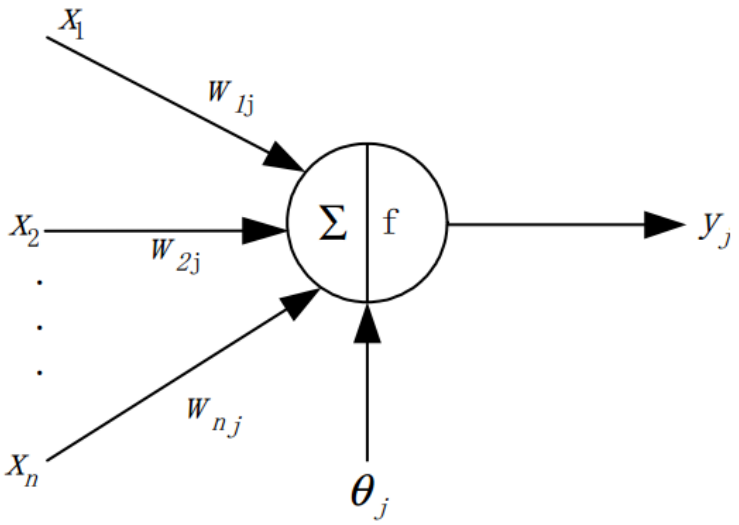
- Hyperbolic tangent function

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



McCulloch & Pitts's Artificial Neuron

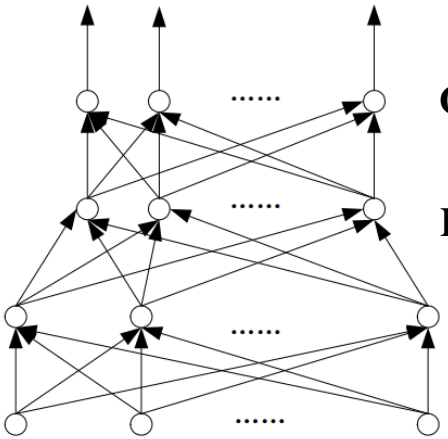
- The first model of artificial neurons in 1943
  - Activation function: a threshold function



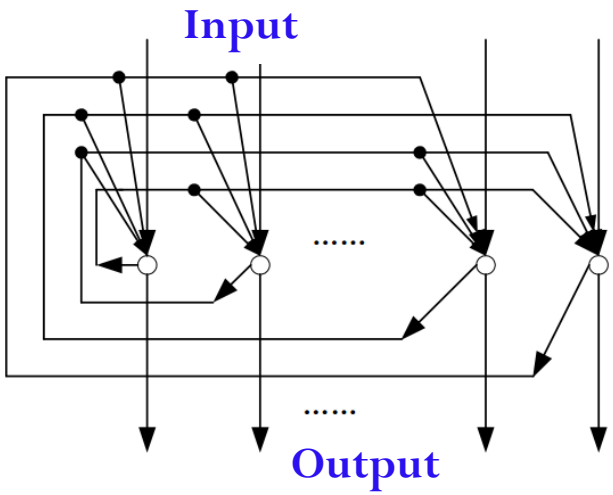
$$y_j = \mathrm{sgn}\left(\sum_i w_{ij} x_i - \theta_j\right)$$

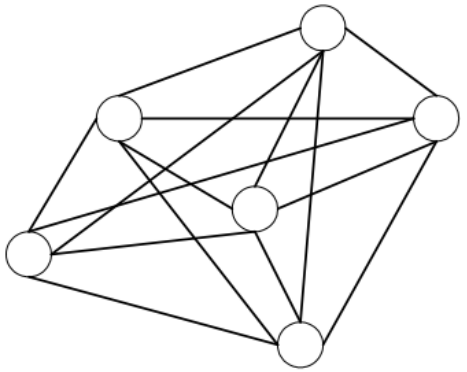
Network Architecture

- Feedforward networks



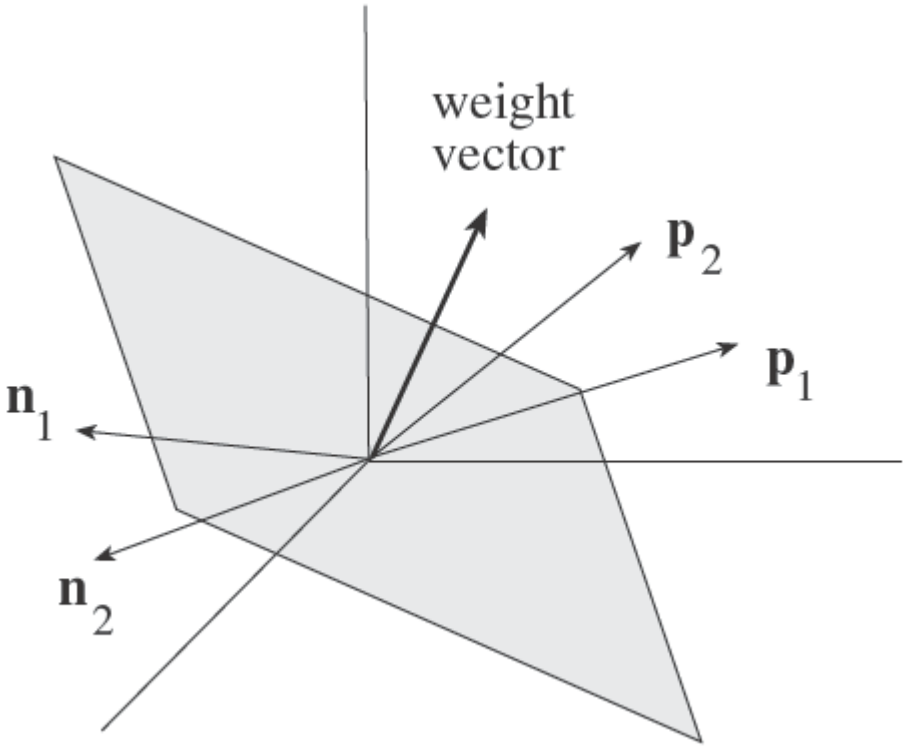
- Recurrent networks





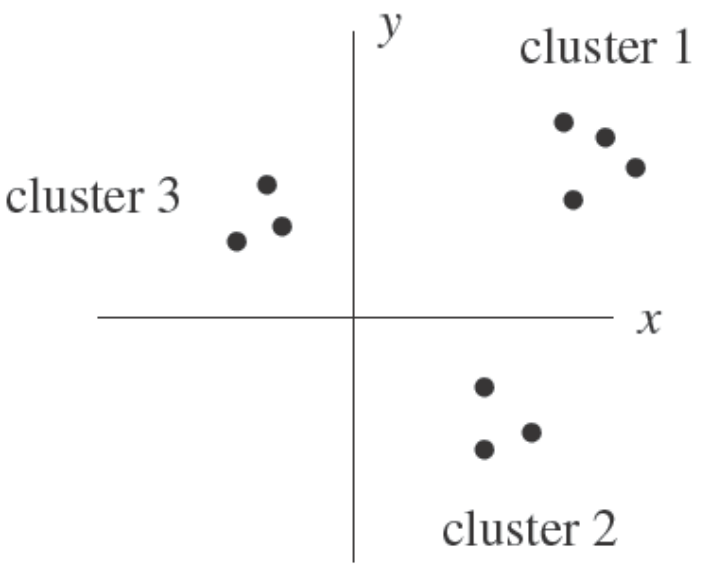
Learning Paradigms

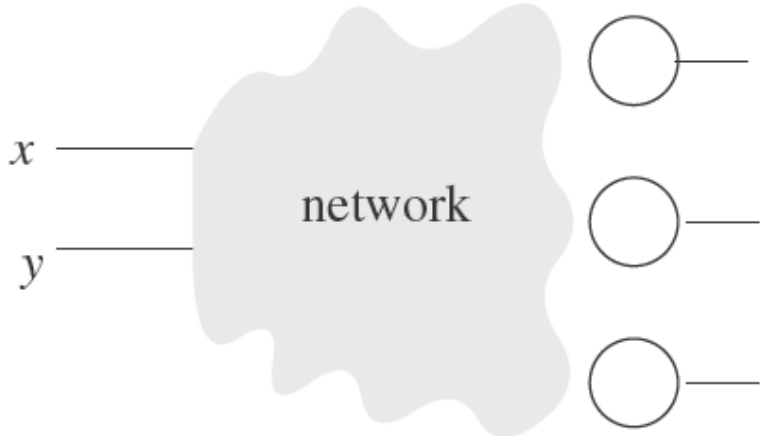
- Supervised Learning (learning with a teacher)
  - For example, classification: learns a separation plane



Learning Paradigms

- Unsupervised learning (learning without a teacher)
  - Example: clustering



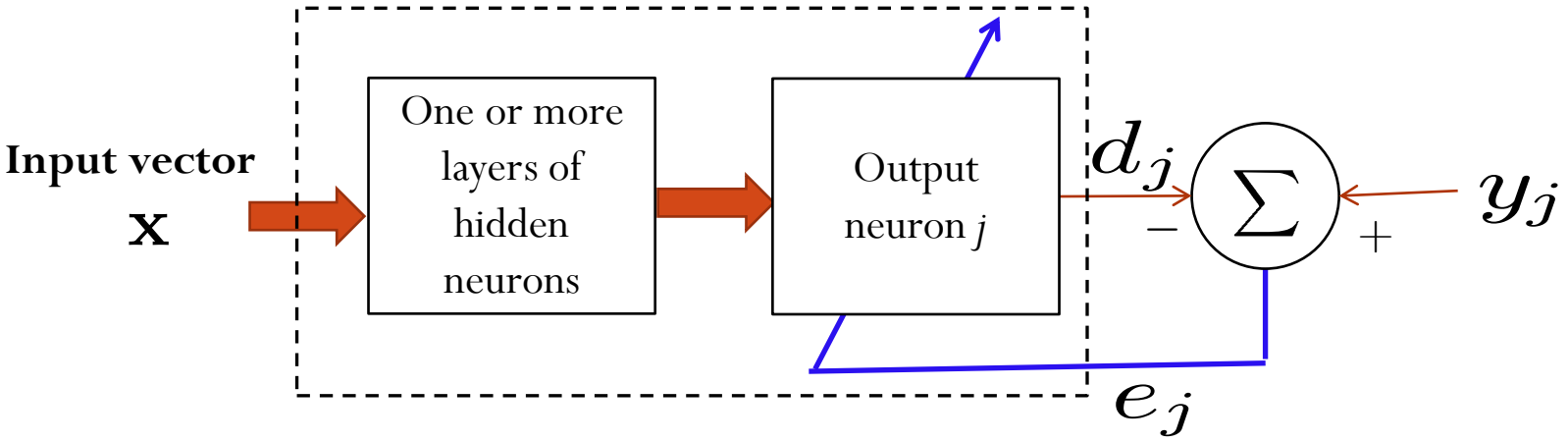


Learning Rules

- Error-correction learning
- Competitive learning
- Hebbian learning
- Boltzmann learning
- Memory-based learning
  - Nearest neighbor, radial-basis function network

Error-correction learning

- The generic paradigm:



  - Error signal:

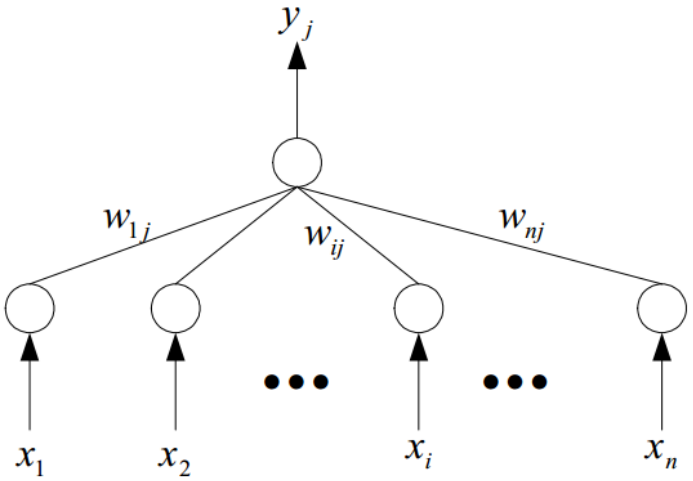
$$e_j = y_j - d_j$$

  - Learning objective:

$$\min_{\mathbf{w}} R(\mathbf{w}; \mathbf{x}) := \frac{1}{2} \sum_j e_j^2$$

Example: Perceptron

- One-layer feedforward network based on error-correction learning (no hidden layer):



  - Current output (at iteration t):

$$d_j = (\mathbf{w}_t^j)^\top \mathbf{x}$$

  - Update rule (*exercise?*):

$$\mathbf{w}_{t+1}^j = \mathbf{w}_t^j + \eta(y_j - d_j)\mathbf{x}$$

Perceptron for classification

- Consider a single output neuron
- Binary labels:

$$y \in \{+1, -1\}$$

- Output function:

$$d = \text{sgn}\left(\mathbf{w}_t^\top \mathbf{x}\right)$$

- Apply the error-correction learning rule, we get … (next slide)

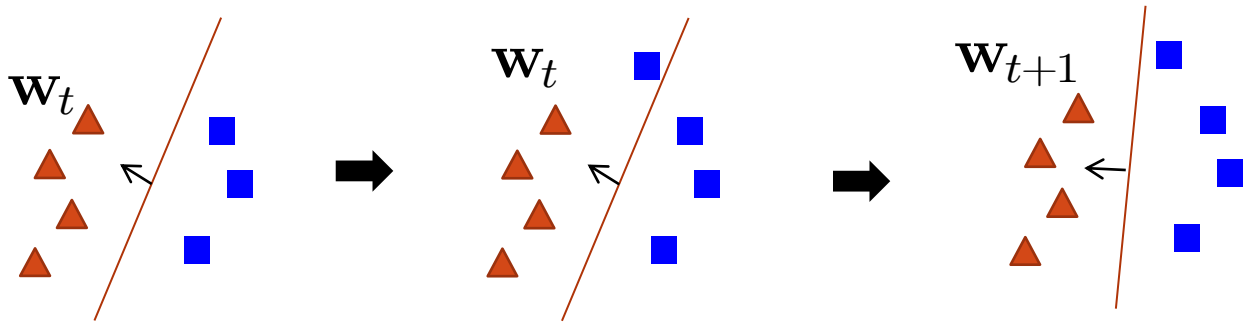
Perceptron for Classification

- Set $\mathbf{w}_1 = 0$ and t=1; scale all examples to have length 1 (doesn't affect which side of the plane they are on)
- Given example x, predict positive *iff*

$$\mathbf{w}_t^\top \mathbf{x} > 0$$

- If a mistake, update as follows
  - Mistake on positive: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta_t \mathbf{x}$
  - Mistake on negative: $\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta_t \mathbf{x}$

$t \leftarrow t + 1$



Convergence Theorem

- For linearly separable case, the perceptron algorithm will converge in a finite number of steps

Mistake Bound

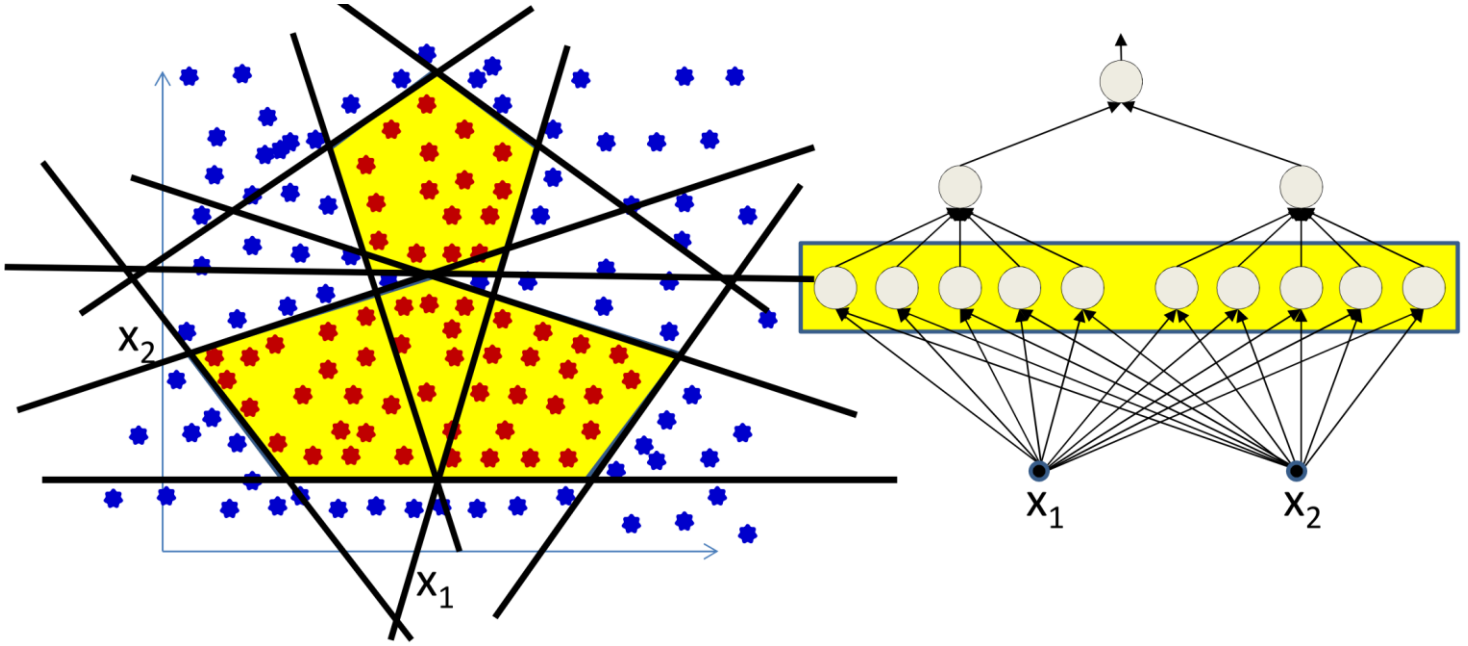
- Theorem:
  - *Let $\mathcal{S}$ be a sequence of labeled examples consistent with a linear threshold function $\mathbf{w}_*^\top \mathbf{x} > 0$, where $\mathbf{w}_*$ is a unit-length vector.*
  - *The number of mistakes made by the online Perceptron algorithm is at most $(1/\gamma)^2$, where*

$$\gamma = \min_{\mathbf{x} \in \mathcal{S}} \frac{|\mathbf{w}_*^\top \mathbf{x}|}{\|\mathbf{x}\|}$$

  - *i.e.: if we scale examples to have length 1, then γ is the minimum distance of any example to the plane $\mathbf{w}_*^\top \mathbf{x} = 0$*
  - *γ is often called the "margin" of $\mathbf{w}_*$; the quantity $\frac{\mathbf{w}_*^\top \mathbf{x}}{\|\mathbf{x}\|}$ is the cosine of the angle between $\mathbf{x}$ and $\mathbf{w}_*$*

More complex decision boundaries

- Individual neurons represent one of the lines that compose the figure (linear classifiers)
- This must be done for *every* neuron. Getting any of them wrong will result in incorrect output!

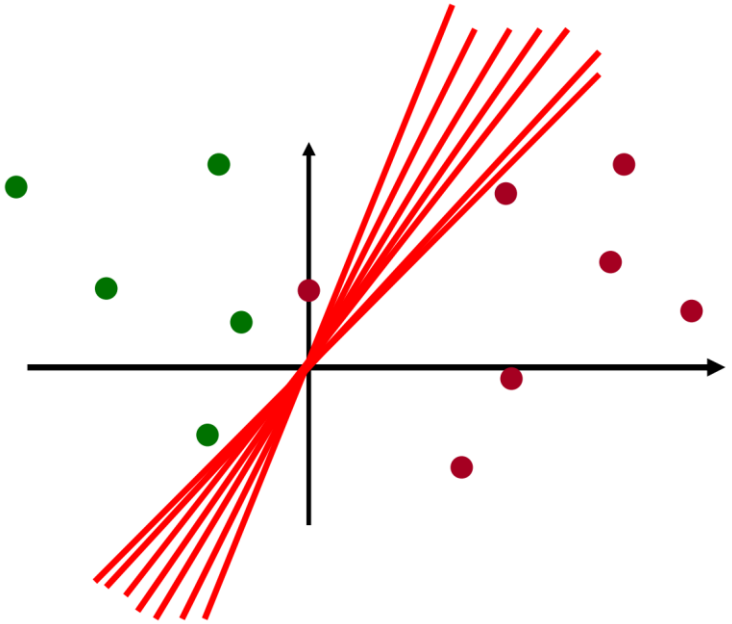


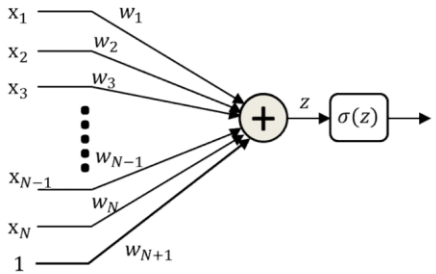
Learning a multilayer perceptron

- Training data only specifies input and output of network
- Intermediate outputs (outputs of individual neurons) are not specified
- Training this network using the perceptron rule is a combinatorial optimization problems
- **Must also determine the correct output for *each* neuron for *every* training instance**
- **NP! Exponential complexity**

Optimization problem

- The perceptron is a flat function with zero derivative everywhere, except at 0 where it is non-differentiable
  - You can vary the weights a lot without changing the error
  - There is no indication of which direction to change the weights to reduce error

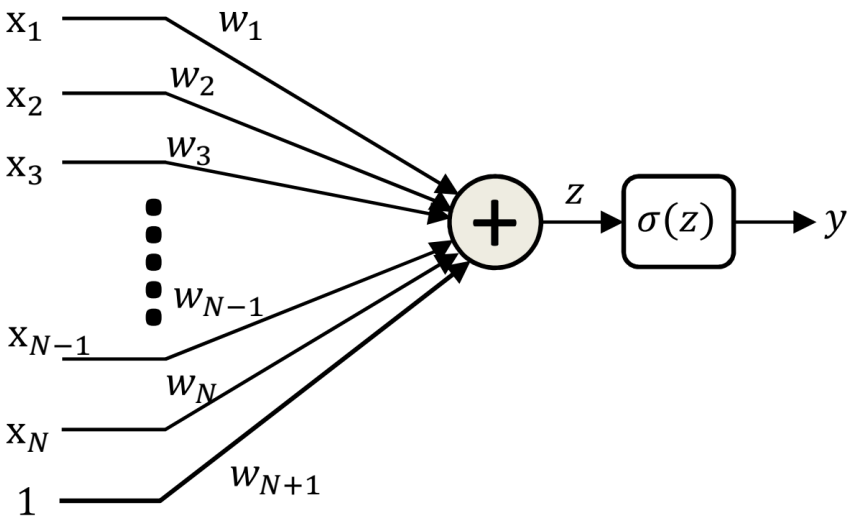


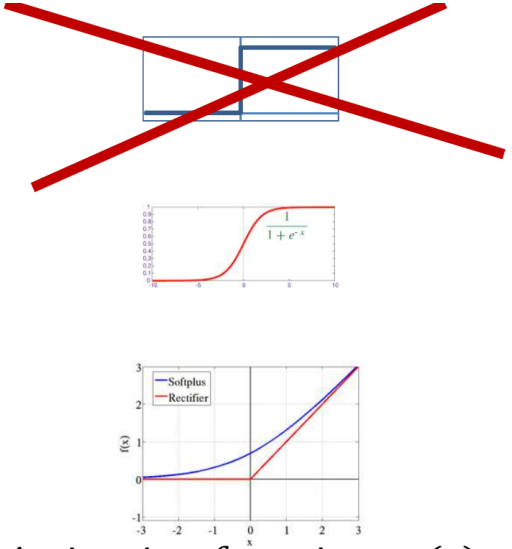


$\sigma(z) =$ 

Solution

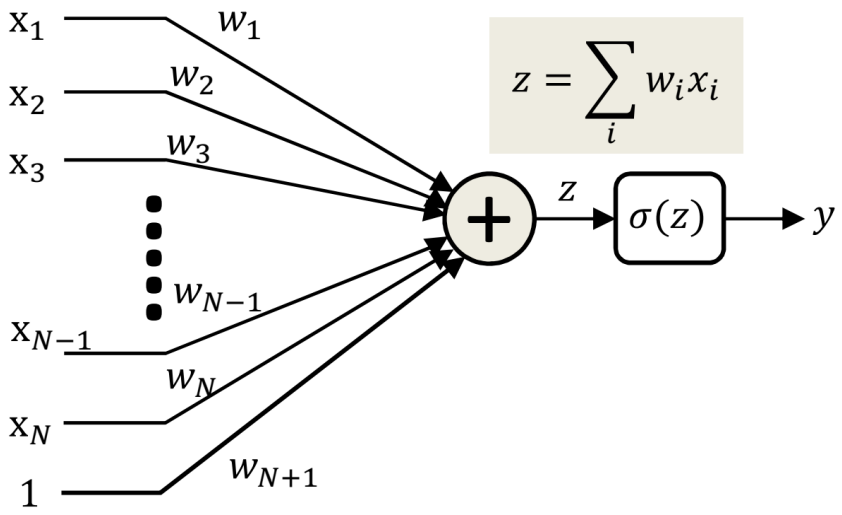
- Let's make the neuron differentiable
  - Small changes in weight can result in non-negligible changes in output
  - This enables us to estimate the parameters using gradient descent techniques.





Activation functions $\sigma(z)$

Perceptrons with differentiable activation functions



$$z = \sum_i w_i x_i$$

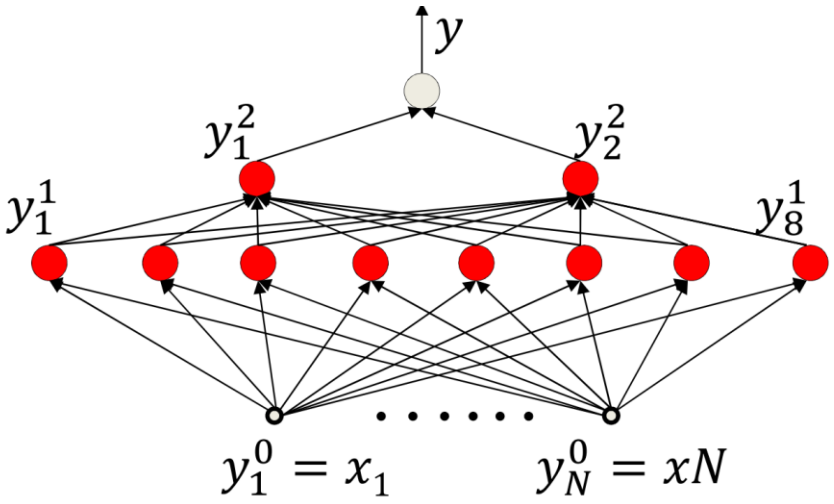
$$\frac{dy}{dw_i} = \frac{dy}{dz}\frac{dz}{dw_i} = \sigma'(z)x_i$$

$$\frac{dy}{dx_i} = \frac{dy}{dz}\frac{dz}{dx_i} = \sigma'(z)w_i$$

- $\sigma(z)$ is a differentiable function
- Using the chain rule, y is a differentiable function of both inputs x and weights w
- we can compute the change in the output for small changes in either the input or the weights

Overall network is differentiable

- The overall function is differentiable w.r.t every parameter
  - Small changes in the parameters result in measurable changes in the output
  - We will derive the actual derivatives using the chain rule later



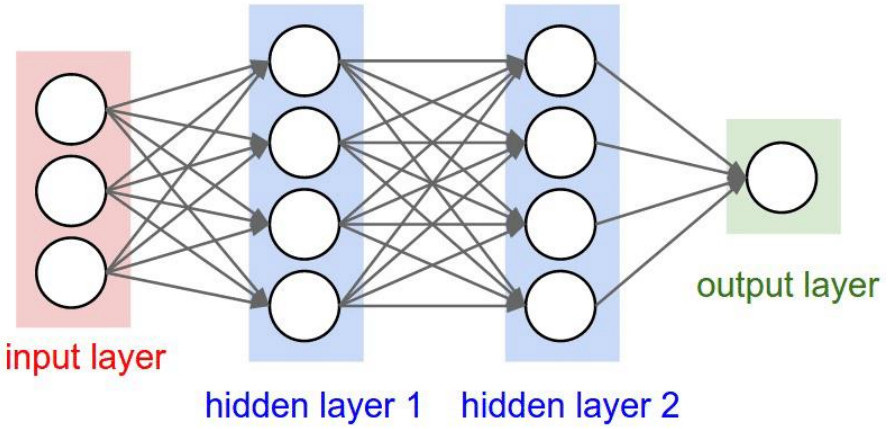
$$y_j^k = \sigma\left(\sum_i w_{i,j}^{k-1} y_i^{k-1}\right)$$

How to learn the function

- Given a training set of input-output pairs $(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$
- Minimize the empirical estimate of expected error

$$Err(W) = \frac{1}{T} \sum_i div\Big(f(x_i; W), y_i\Big)$$

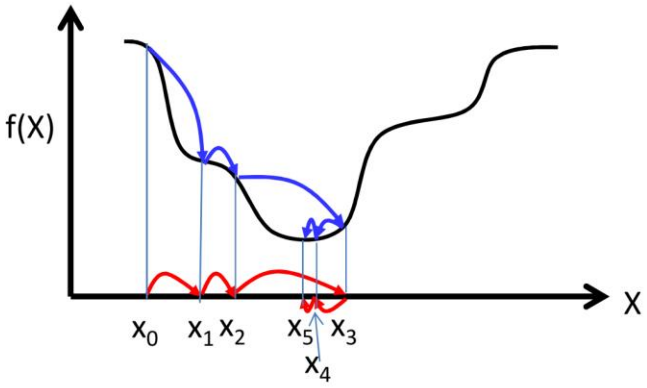
  - div() is a divergence function that goes to zero when

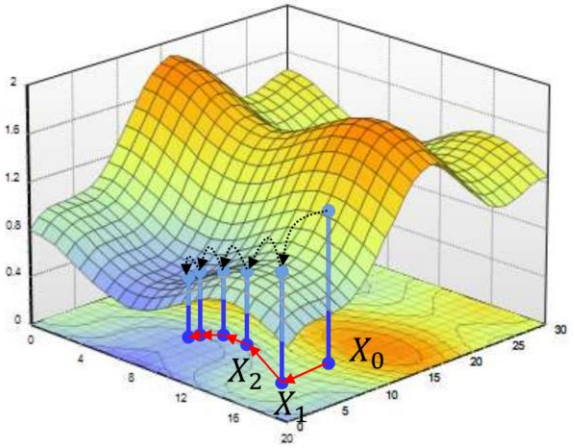


$$y = f(x; W)$$

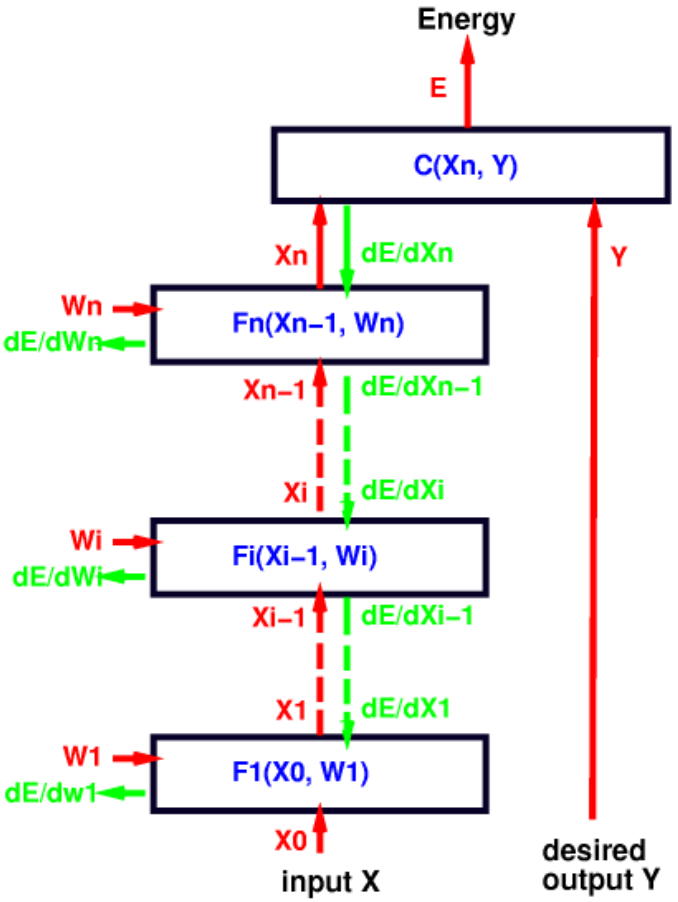
Closed Form Solutions are not always available

- Often it is not possible to simply solve $\nabla f(x) = 0$
  - The function to minimize may have an intractable form
- Iterative solutions are used
  - Start from an initial guess X_0 for the optimal X
  - Update the guess towards a (hopefully) "better" value of $f(X)$
  - Stop when $f(X)$ no longer decreases





Multimodule Systems: Cascade



- Assembling modules into networks
- Forward Propagation:

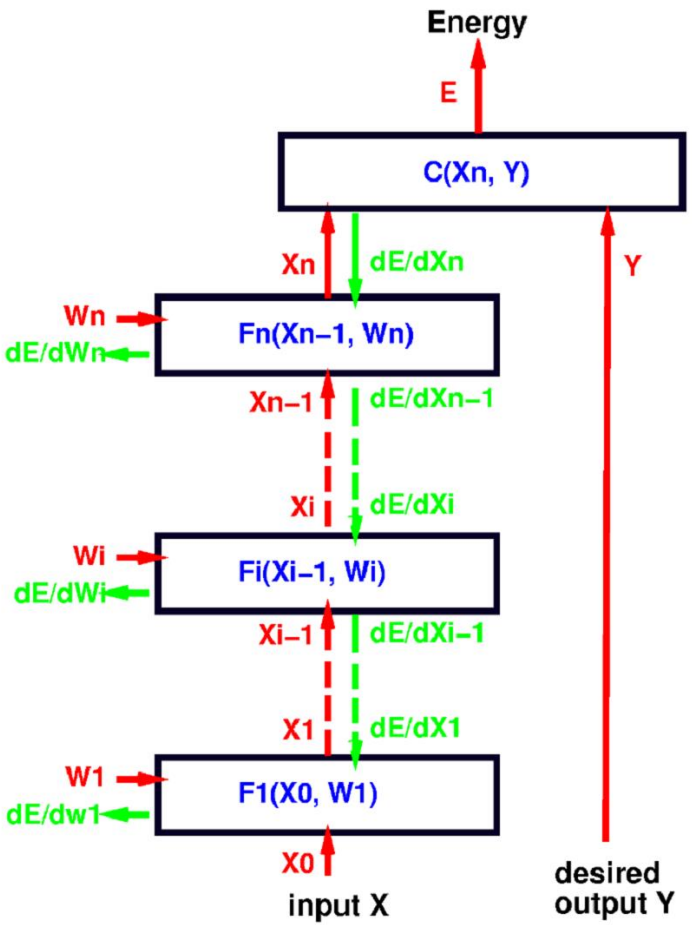
let $X = X_0$,

$$X_i = F_i(X_{i-1}, W_i) \quad \forall i \in [1, n]$$

$$E(Y, X, W) = C(X_n, Y)$$

- Each module is an object
  - Contains trainable parameters
  - Inputs are arguments
  - Output is returned

Computing the Gradient

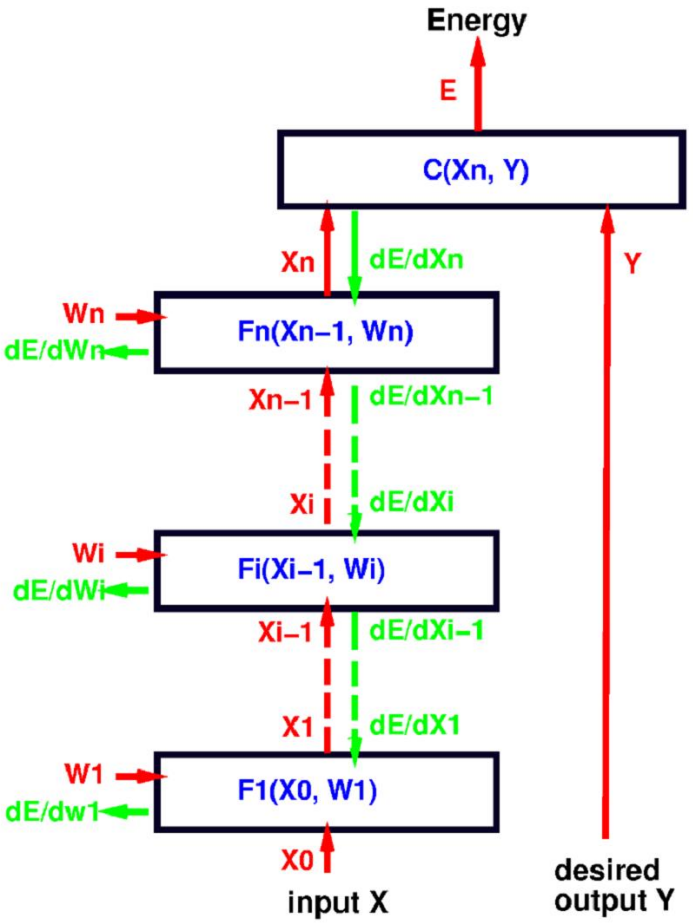


- To train a multi-module system, we must compute the gradient of $E(X, Y, W)$ with respect to all the parameters in the system (all the W_k).
- Let's consider module k

$$X_k = F_k(X_{k-1}, W_k)$$

- We can assume that we can calculate $\frac{\partial E_k}{\partial X_k}$, i.e., for each component of vector X_k we know how much E would wiggle if we wiggled that component of X_k

Computing the Gradient



- We can apply chain rule to compute $\frac{\partial E_k}{\partial X_k}$

$$\frac{\partial E}{\partial W_k} = \frac{\partial E}{\partial X_k}\frac{\partial F_k(X_{k-1}, W_k)}{\partial W_k}$$

$$[1 \times N_w] = [1 \times N_x].[N_x \times N_w]$$

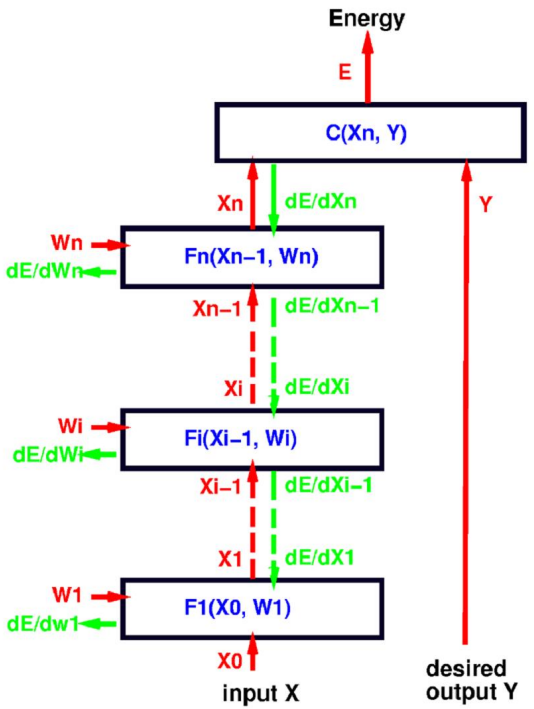
- *Jacobian matrix* of F_k with respect to W_k

$$\left[\frac{\partial F_k(X_{k-1}, W_k)}{\partial W_k}\right]_{ij} = \frac{\partial [F_k(X_{k-1}, W_k)]_i}{\partial [W_k]_j}$$

- Indicate how much the i-th output wiggles when we wiggle the j-th weight.
- A recurrence equation!

Back Propagation

- To compute all the derivatives, we use a backward sweep called the back-propagation algorithm that uses the recurrence equation for $\frac{\partial E_k}{\partial X_k}$



$$\frac{\partial E}{\partial X_n} = \frac{\partial C(X_n, Y)}{\partial X_n}$$

$$\frac{\partial E}{\partial X_{n-1}} = \frac{\partial E}{\partial X_n}\frac{F_n(X_{n-1}, W_n)}{X_{n-1}}$$

$$\frac{\partial E}{\partial W_n} = \frac{\partial E}{\partial X_n}\frac{F_n(X_{n-1}, W_n)}{W_n}$$

$$\frac{\partial E}{\partial X_{n-2}} = \frac{\partial E}{\partial X_{n-1}}\frac{F_{n-1}(X_{n-2}, W_{n-1})}{X_{n-2}}$$

$$\frac{\partial E}{\partial W_{n-1}} = \frac{\partial E}{\partial X_{n-1}}\frac{F_{n-1}(X_{n-2}, W_{n-1})}{W_{n-1}}$$

- until we reach the first module

Overall Approach

- For each data instance
  - Forward pass: Pass instance forward through the net. Store all intermediate outputs of all computation
  - Backward pass: Sweep backward through the net, iteratively compute all derivatives w.r.t weights
- Actual Error is the sum of the error over all training instances

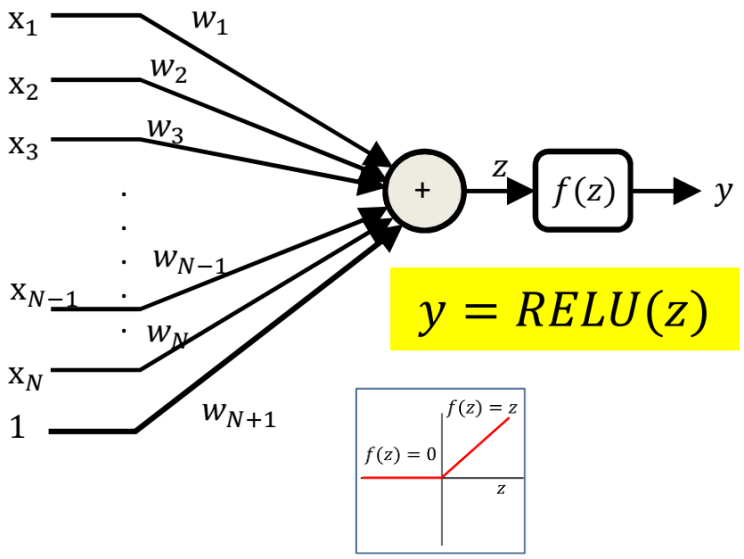
$$Err(W) = \frac{1}{T}\sum_i div\Big(f(x_i; W), y_i\Big)$$

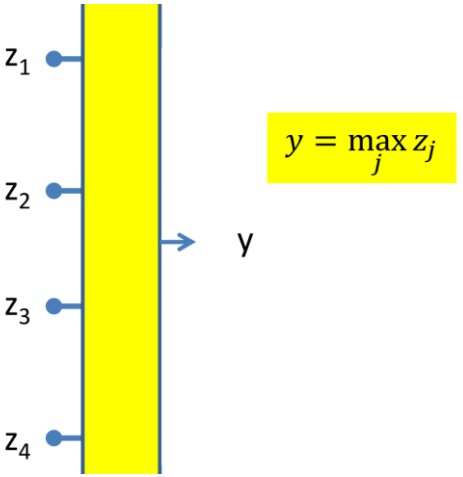
- Actual gradient is the sum or average of the derivatives computed for each training instance

$$W \leftarrow W - \nabla_W Err$$

Non-differentiable functions

- Functions are sometimes not actually differentiable
  - E.g. The RELU (Rectified Linear Unit); and its variants such as leaky RELU, randomized leaky RELU
- Must use "subgradients" where available



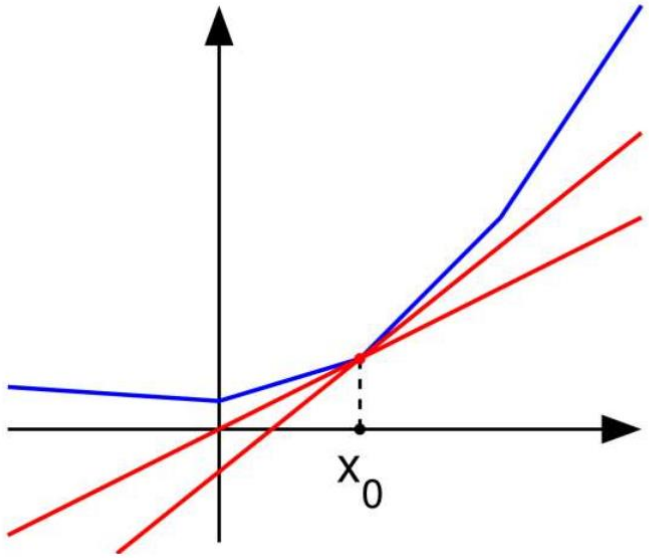


The subgradient

- A subgradient of a function at a point x_0 is any vector such that

$$f(x) - f(x_0) \geq v^\top (x - x_0)$$

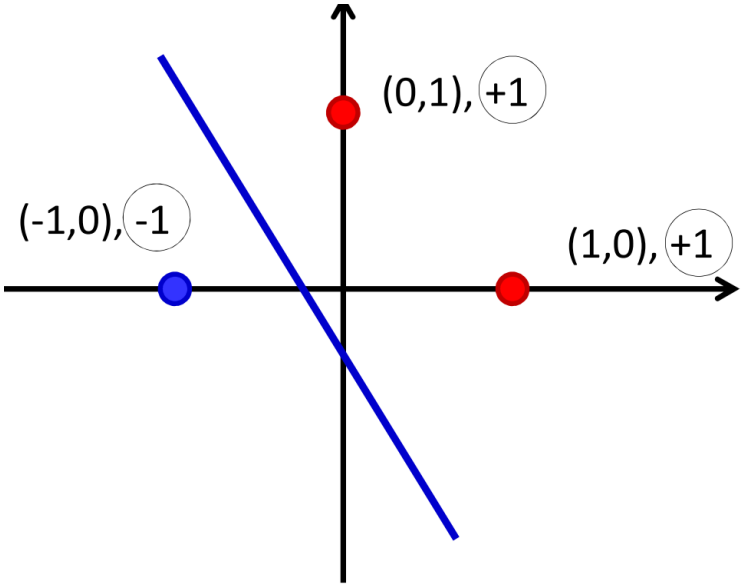
- Guaranteed to exist only for convex functions
- The subgradient is a direction in which the function is guaranteed to increase

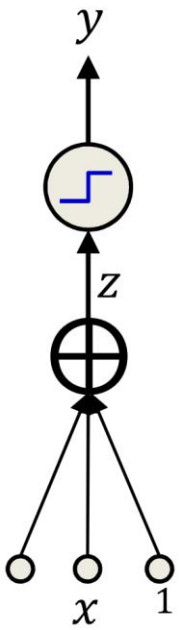


Does backprop do the right thing?

- Is backprop always right?
  - Assuming it actually find the global minimum of the divergence function?
- In classification problems, the classification error is a non-differentiable function of weights
- The divergence function minimized is only a proxy for classification error
- Minimizing divergence may not minimize classification error

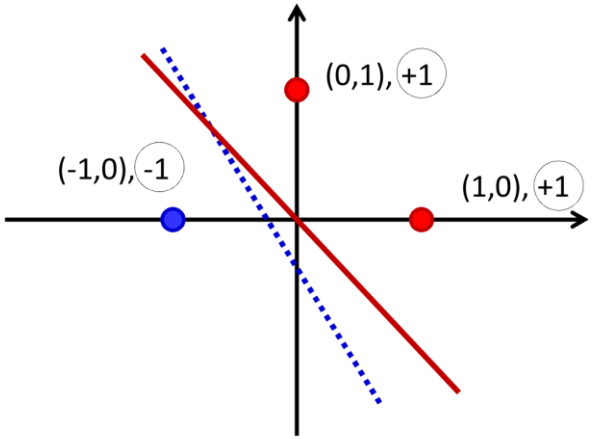
Single neuron perceptron

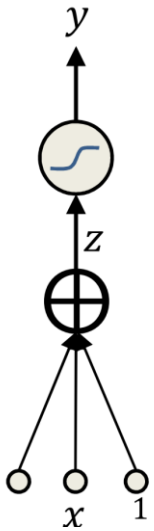


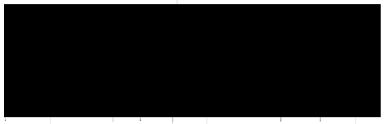


- Simple problem, 3 training instances, single neuron
- Perceptron training rule trivially find a perfect solution

Backprop vs. Perceptron





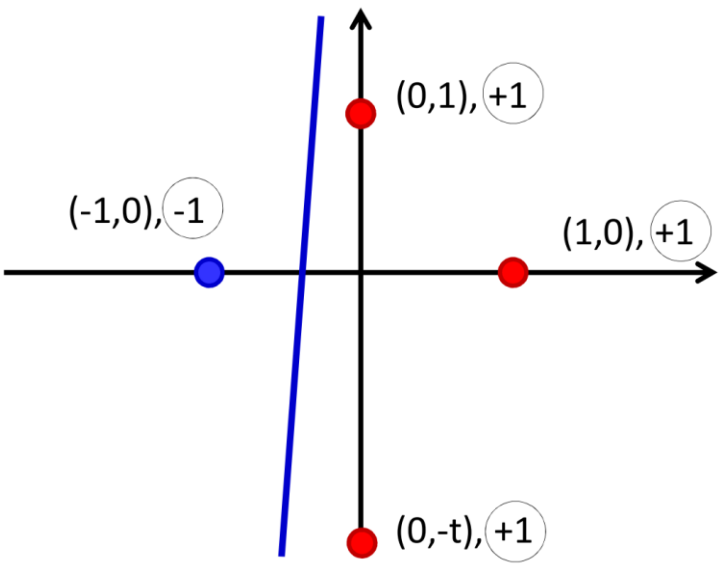


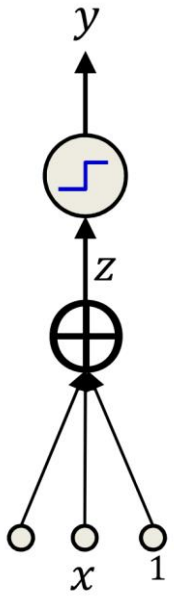
- Back propagation using logistic function

$$err = (f(x) - y)^2$$

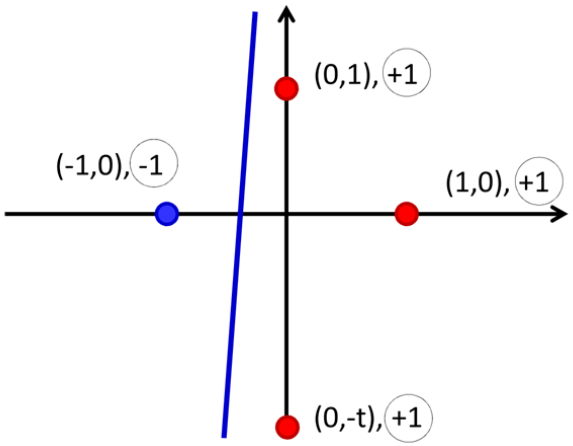
- Unique minimum trivially proved to exist, Backpropagation finds it

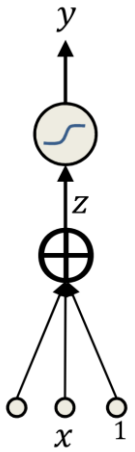
Backprop vs. Perceptron





- Now add a fourth point and t is very large (point near $-\infty$)
- Perceptron trivially finds a solution





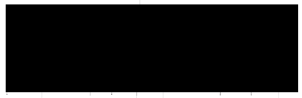
- Backprop: Contribution of fourth point to derivative of L

$$div_4 = (1 - \sigma(-w_y t + b))^2$$

$$\frac{div_4}{dw_y} = 2(1 - \sigma(-w_y t + b))\sigma'(-w_y t + b)t$$

$$\frac{div_4}{db} = -2(1 - \sigma(-w_y t + b))\sigma'(-w_y t + b)$$

Backprop



$$\frac{div_4}{dw_y} = 2(1-\sigma(-w_y t+b))\sigma'(-w_y t+b)t$$

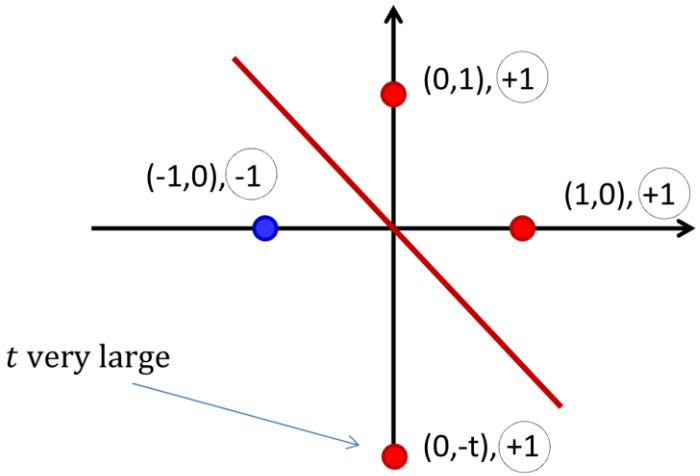
$$div_4 = (1-\sigma(-w_y t+b))^2 \quad \frac{div_4}{db} = -2(1-\sigma(-w_y t+b))\sigma'(-w_y t+b)$$

- For very large positive t,
- $\sigma'(-w_y t+b) \to 0$ *exponentially* as $t \to \infty$
- Therefore, for very large positive $t \to \infty$

$$\frac{div_4}{dw_y} = \frac{div_4}{db} = 0$$

Backprop

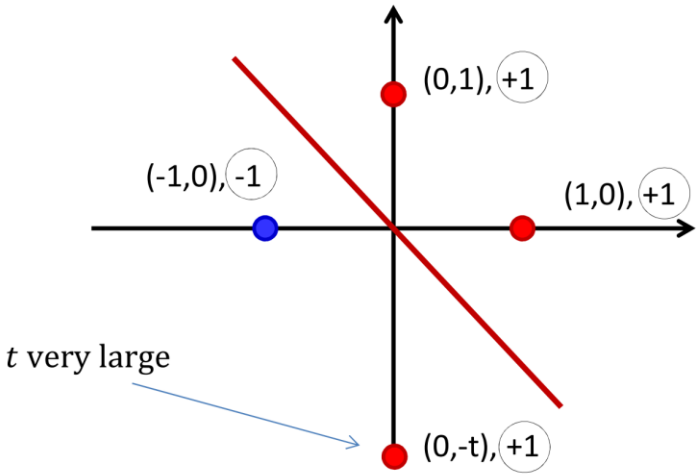
- The fourth point at (0, -t) does not change the gradient of the divergence near the optimal solution for 3 points



- The optimum solution for 3 points is also a broad local minimum (0 gradient) for the 4-point problem!
  - Will be trivially found by backprop nearly all the time

Backprop

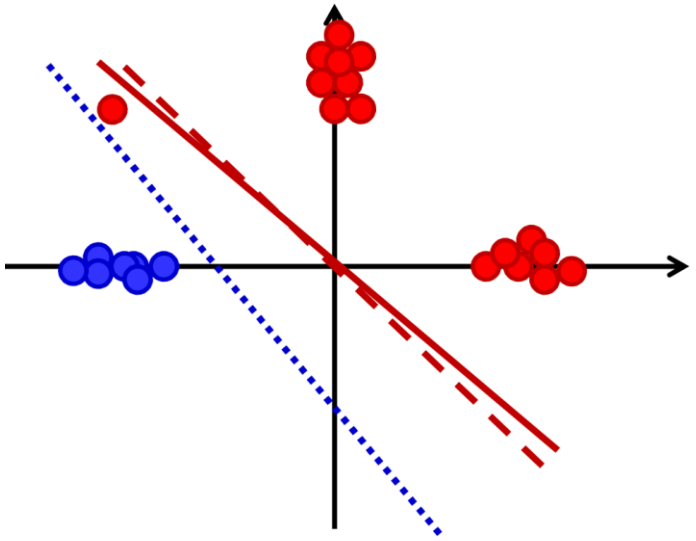
- The fourth point at does not change the gradient of the divergence near the optimal solution for 3 points



- **Local optimum** solution found by backprop
- Does not separate the points even though the points are linearly separable!
- Compare to the perceptron: *Backpropagation fails to separate where the perceptron succeeds*

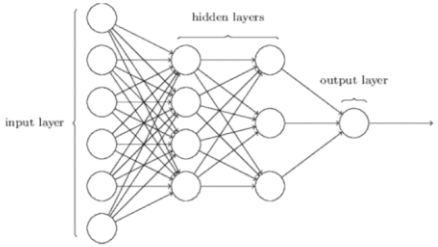
A more complex problem

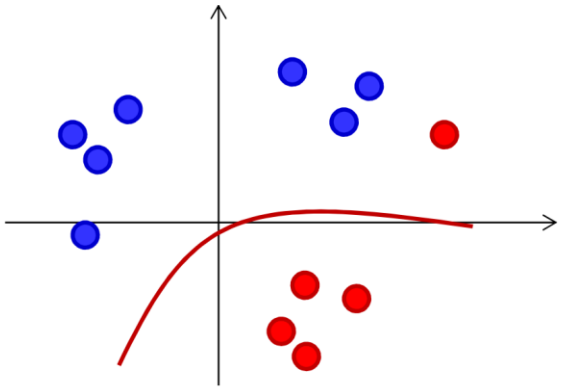
- The fourth point at does not change the gradient of the divergence near the optimal solution for 3 points



- Adding a "spoiler" (or a small number of spoilers)
  - Perceptron finds the linear separator
  - Backprop does not find a separator
    - A single additional input does not change the loss function significantly

Backprop fails to separate even when possible





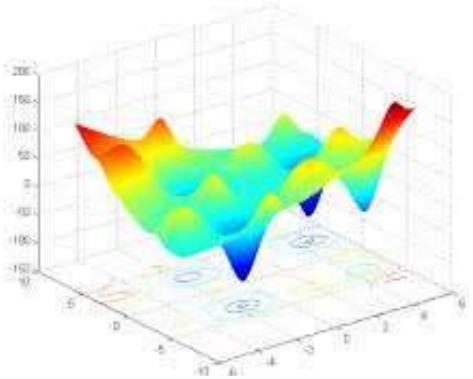
- This is not restricted to single perceptrons
  - In an MLP the lower layers "learn a representation" that enables linear separation by higher layers
- Adding a few "spoilers" will not change their behavior

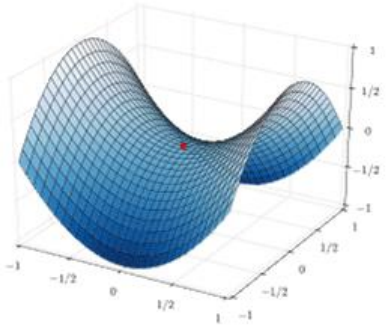
So what is happening here?

- The perceptron may change greatly upon adding just a single new training instance
  - But it fits the training data well
    - The perceptron rule has low bias, i.e., makes no errors if possible
    - But high variance, i.e., swings wildly in response to small changes
- Backpropagation will often not find a separating solution even though the solution is within the class of functions learnable by the network
- Backprop is minimally changed by new training instances
  - Prefers consistency over perfection
  - It is a low-variance estimator, at the potential cost of bias

The Error Surface

- Previous example (and statements) assumed the loss objective had a single global optimum that could be found
- What about local optima ?
  - **Popular hypothesis**:
    - In large networks, saddle points are far more common than local minima
      - Frequency exponential in network size
    - Most local minima are equivalent
      - And close to global minimum
    - This is not true for small networks





Saddle point: A point where the slope is zero

- The surface increases in some directions, but decreases in others

Story so far

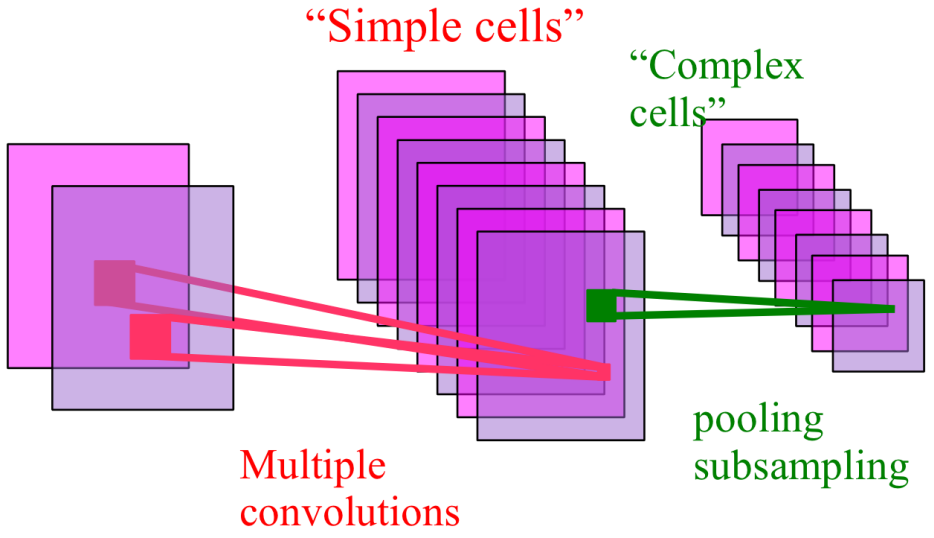
- Neural nets can be trained via gradient descent that minimizes a loss function
- Backpropagation can be used to derive the derivatives of the loss
- Backprop *is not guaranteed* to find a "true" solution, even if it exists, and lies within the capacity of the network to model
  - The optimum for the loss function may not be the "true" solution
- For large networks, the loss function may have a large number of unpleasant saddle points
  - Which backpropagation may find

Content

- Backpropagation
- Convolutional Neural Networks (ResNet)
- Advanced topics

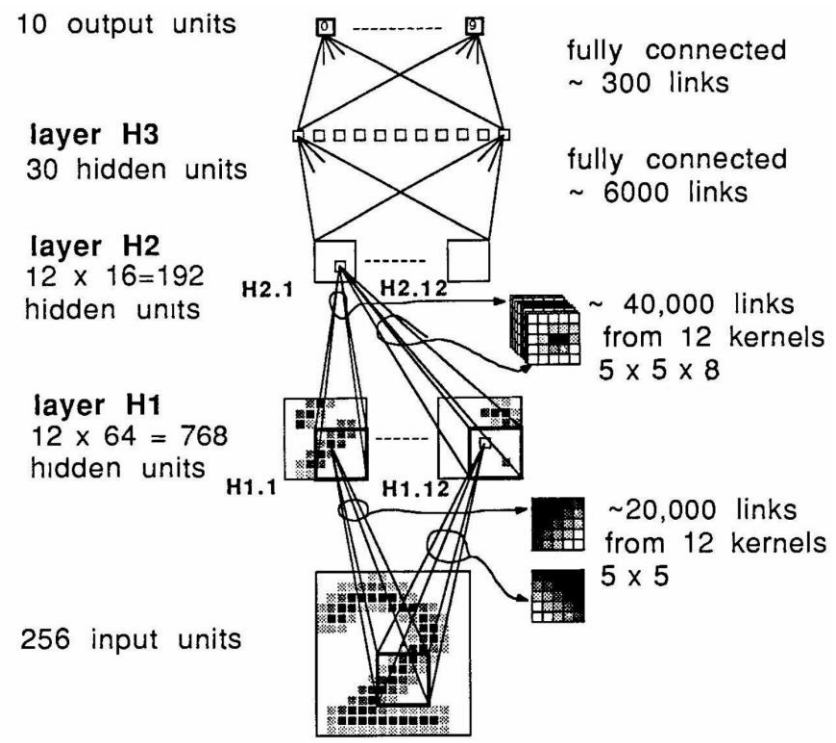
Hubel & Wiesel's Model

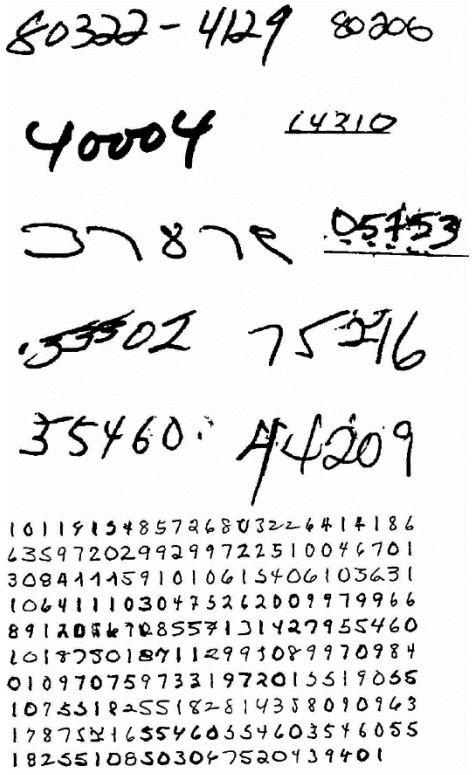
- [Hubel & Wiesel 1962]:
  - Simple cells detect local features
  - Complex cells "pool" the outputs of simple cells within a retinotopic neighborhood.



First "Real" ConvNets [LeCun et al 89]

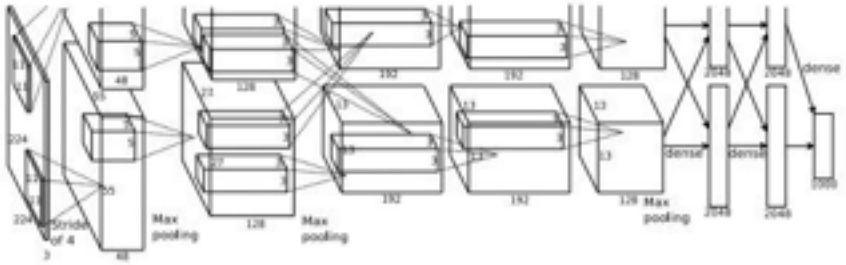
- Trained with Backprop
  - USPS Zipcode digits: 7300 training, 2000 test.
  - Convolution with stride. No separate pooling.



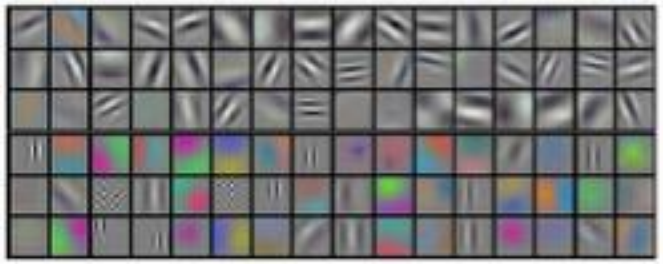


First strong results

- ***Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition***
  George Dahl, Dong Yu, Li Deng, Alex Acero, 2012
- ***Imagenet classification with deep convolutional neural networks***
  Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton, 2012



"AlexNet"

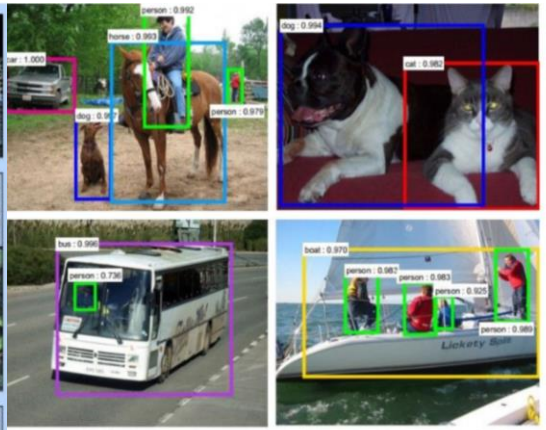


Fast-forward to today: ConvNets are everywhere

Classification

Figures copyright Alex Krizhevsky, Ilya Sutskever,

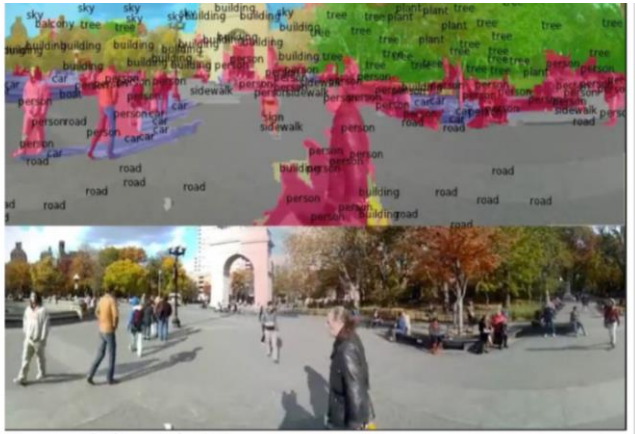
Detection



Figures copyright Shaoqing Ren, Kaiming He, Ross Girschick, Jian Sun, 2015. Reproduced with permission.

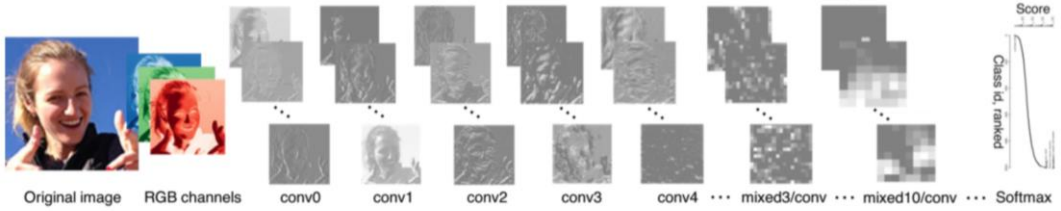
[Faster R-CNN: Ren, He, Girshick, Sun 2015]

Segmentation



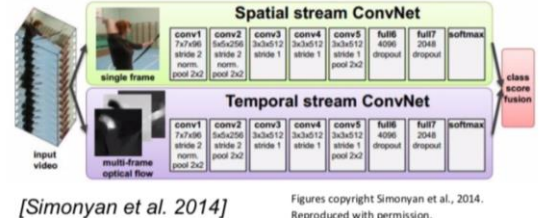
Figures copyright Clement Farabet, 2012.
Reproduced with permission.

[Farabet et al., 2012]



[Taigman et al. 2014]

Activations of inception-v3 architecture [Szegedy et al. 2015] to image of Emma McIntosh, used with permission. Figure and architecture not from Taigman et al. 2014.



[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014.
Reproduced with permission.

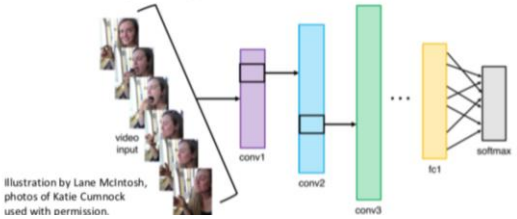
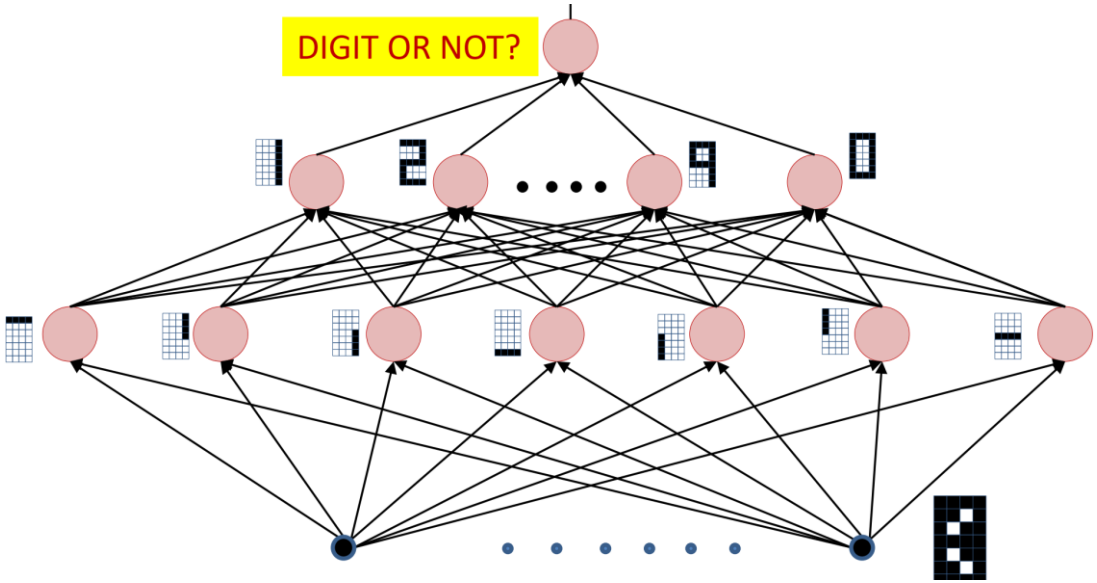


Illustration by Lane McIntosh, photos of Katie Cumnock used with permission.

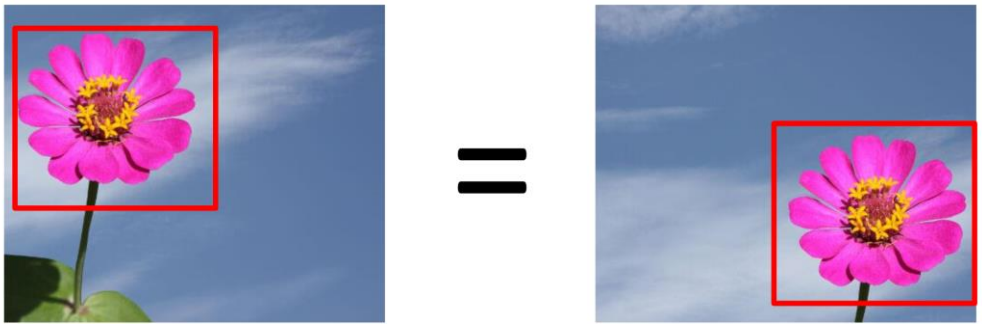
Action Recognition

The MLP as a cascade of feature detectors



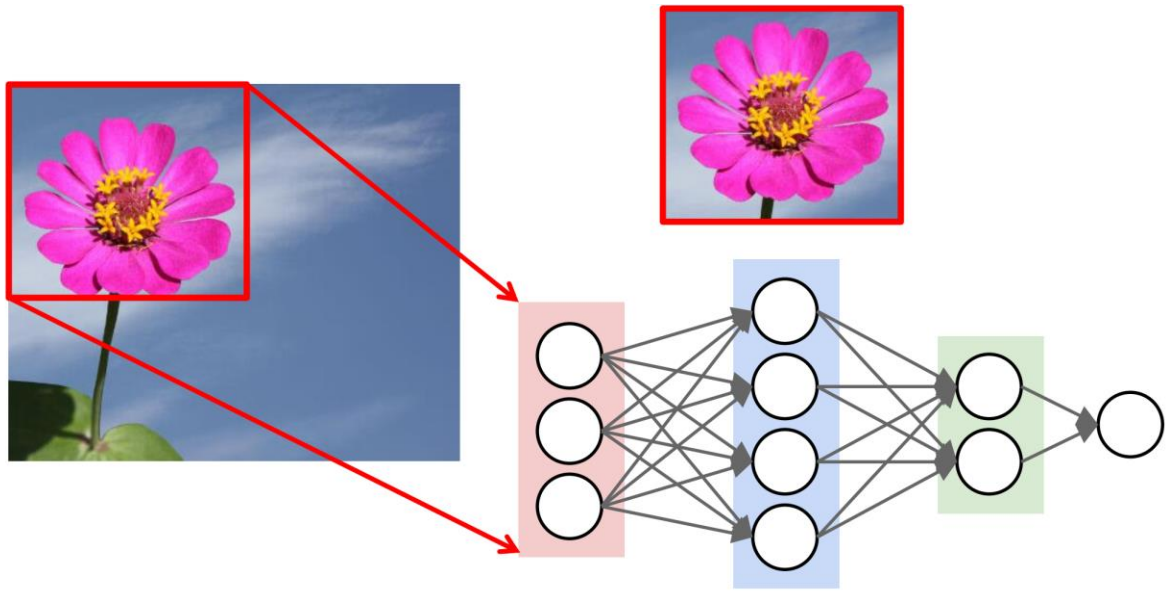
- The network is a cascade of feature detectors
  - Higher level neurons compose complex templates from features represented by lower-level neurons

The need for shift invariance



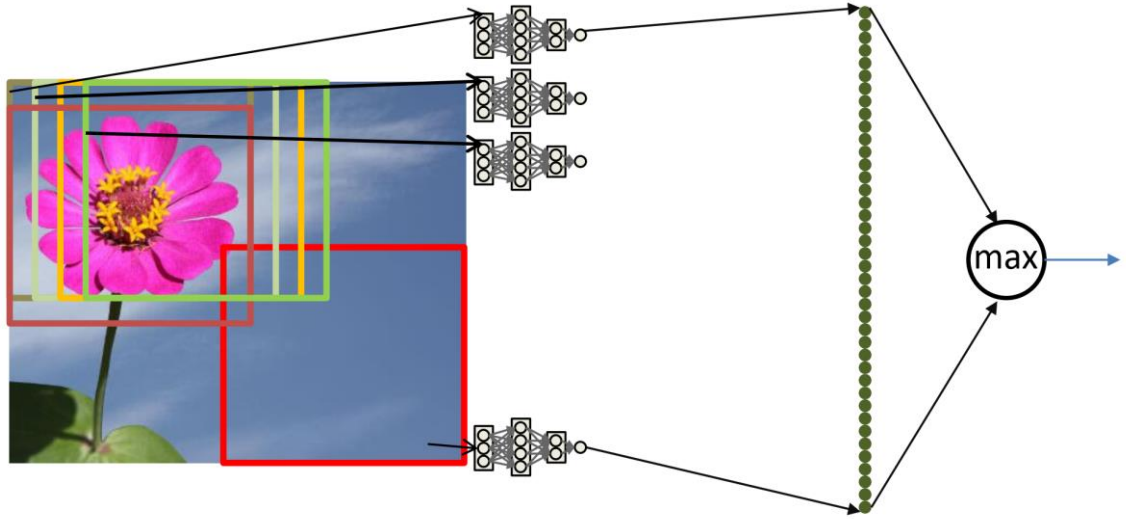
- In many problems the location of a pattern is not important
  - Only the presence of the pattern
- Conventional MLPs are sensitive to the location of the pattern
- Moving it by one component results in an entirely different input that the MLP wont recognize
- **Requirement: Network must be shift invariant**

Solution: Scan



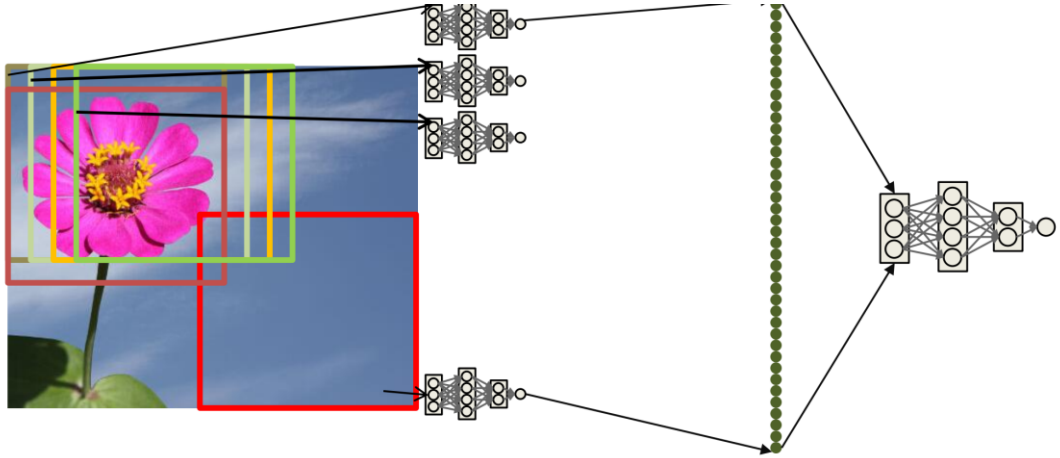
- Scan for the desired object
  - "Look" for the target object at each position
- At each location, the entire region is sent through an MLP

Scanning the picture to find a flower



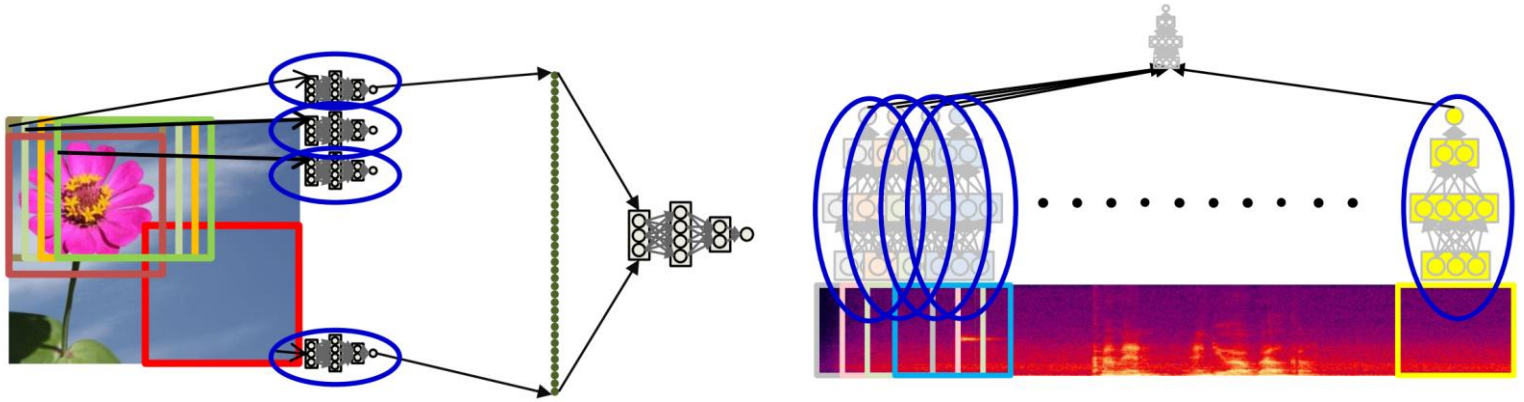
- Determine if any of the locations had a flower
  - We get one classification output per scanned location
- The score output by the MLP
- Look at the maximum value

Its just a giant network with common subnets



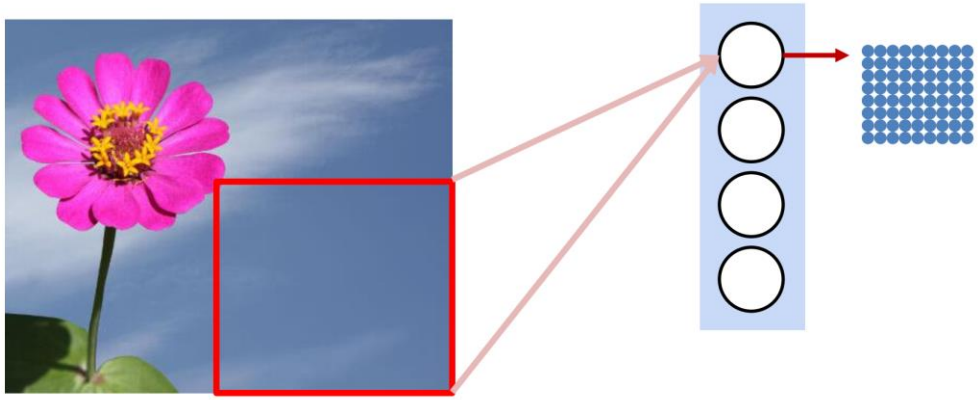
- The entire operation can be viewed as a single giant network
  - Composed of many "subnets" (one per window)
  - With one key feature: all subnets are identical

Learning in shared parameter networks



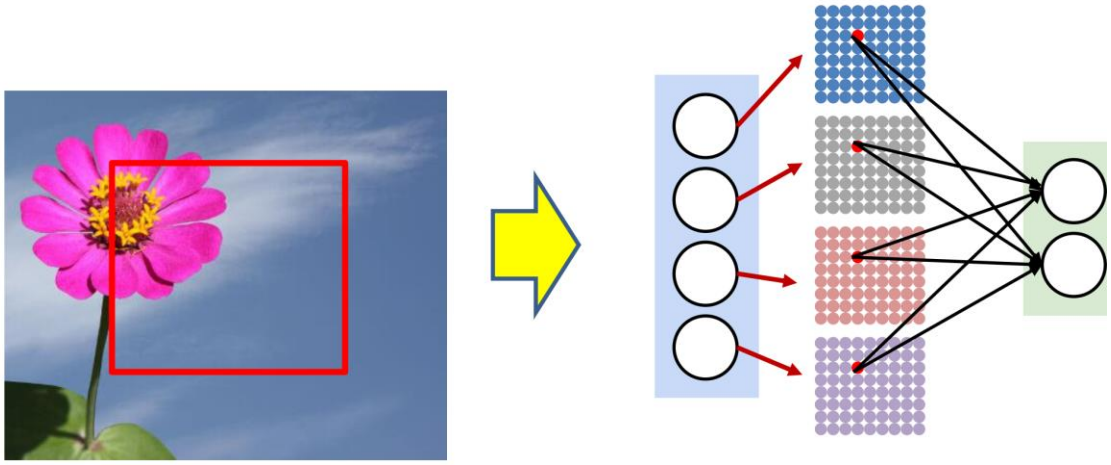
- These are shared parameter networks
  - All lower-level subnets are identical
- Are all searching for the same pattern
  - Any update of the parameters of one copy of the subnet must equally update all copies

Scanning: A closer look



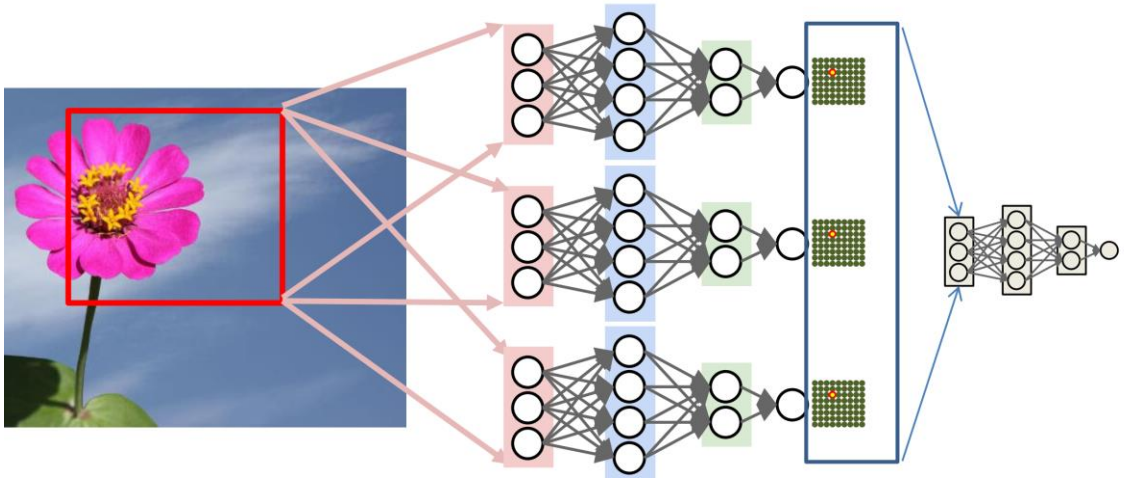
- Consider a single perceptron, at each position of the box, the perceptron is evaluating the picture as part of the classification for that region
  - We could arrange the outputs of the neurons for each position correspondingly to the original picture
- We can arrange the outputs from the response at each scanned position into a rectangle that's proportional in size to the original picture

Scanning: A closer look



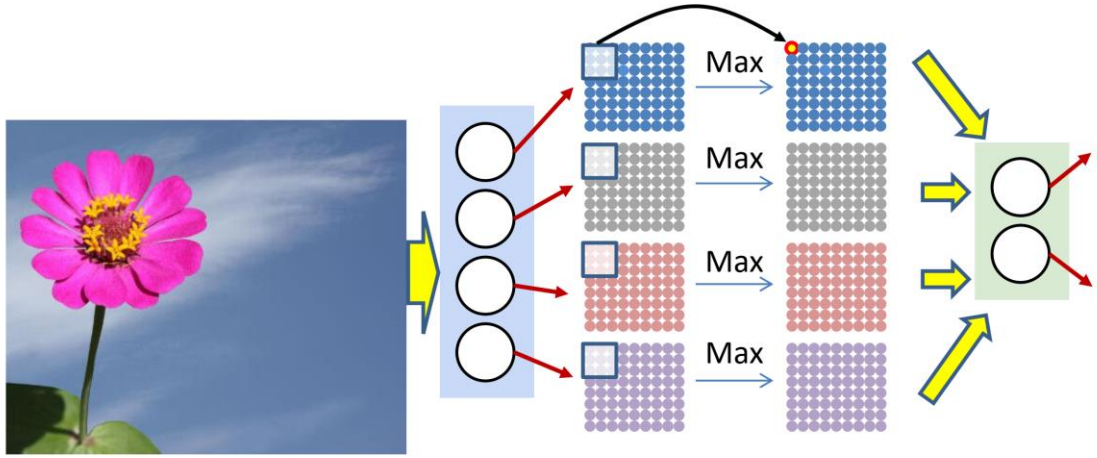
- Each perceptron's outputs from each of the scanned positions can be arranged as a rectangular pattern
- We can recurse the logic
  - The second level neurons too are "scanning" the rectangular outputs of the first-level neurons

Generalizing a bit



- At each location, the net searches for a flower
- The entire map of outputs is sent through a follow-up perceptron (or MLP) to determine if there really is a flower in the picture

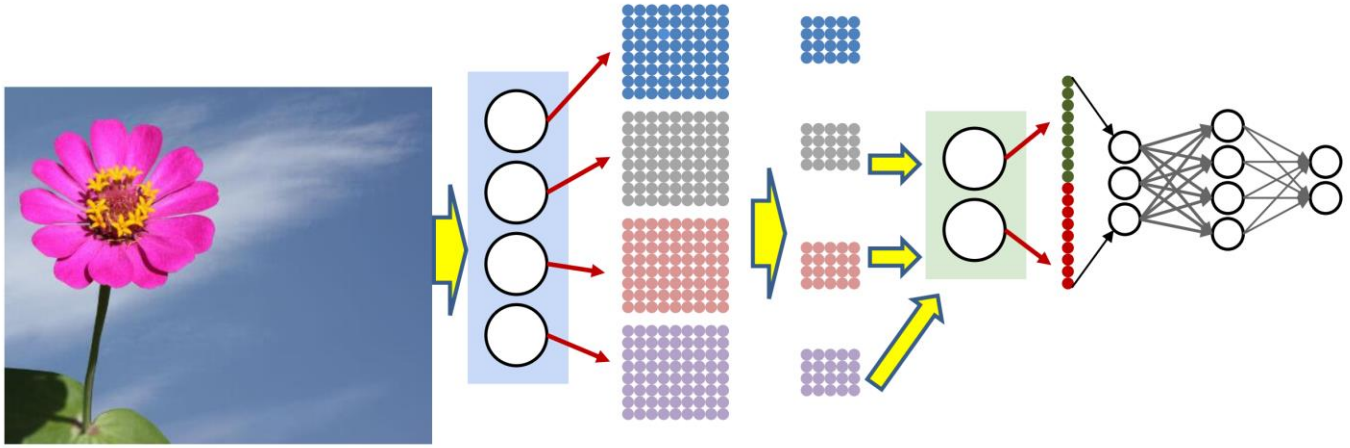
Accounting for jitter



- We would like to account for some jitter
  - If a pattern shifts by one pixel, is it still a petal?
  - A small jitter is acceptable
- Replace each value by the maximum of the values within a small region around it
  - Max filtering or Max pooling

The overall structure

- In reality we can have many layers of "convolution" followed by max pooling (and reduction) before the final MLP
  - The individual perceptrons at any "scanning" or "convolutive" layer are called "filters"
  - max pooling
  - Fully connected



Story so far

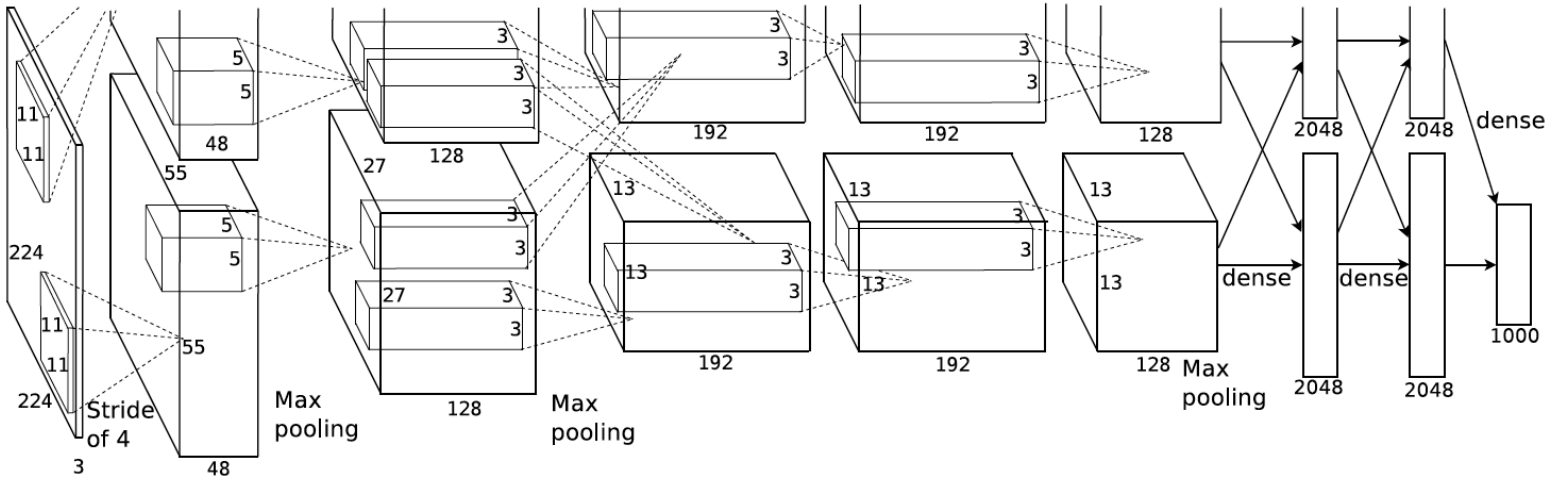
- Neural networks learn patterns in a hierarchical manner
- Pattern classification tasks such as "does this picture contain a cat" are best performed by scanning for the target pattern
- Scanning for patterns can be viewed as classification with a large shared parameter network
- Scanning an input with a network and combining the outcomes is equivalent to scanning with individual neurons
- At each layer, a scan by a neuron may optionally be followed by a "max" (or any other) "pooling" operation to account for deformation

Practical CNN Architectures

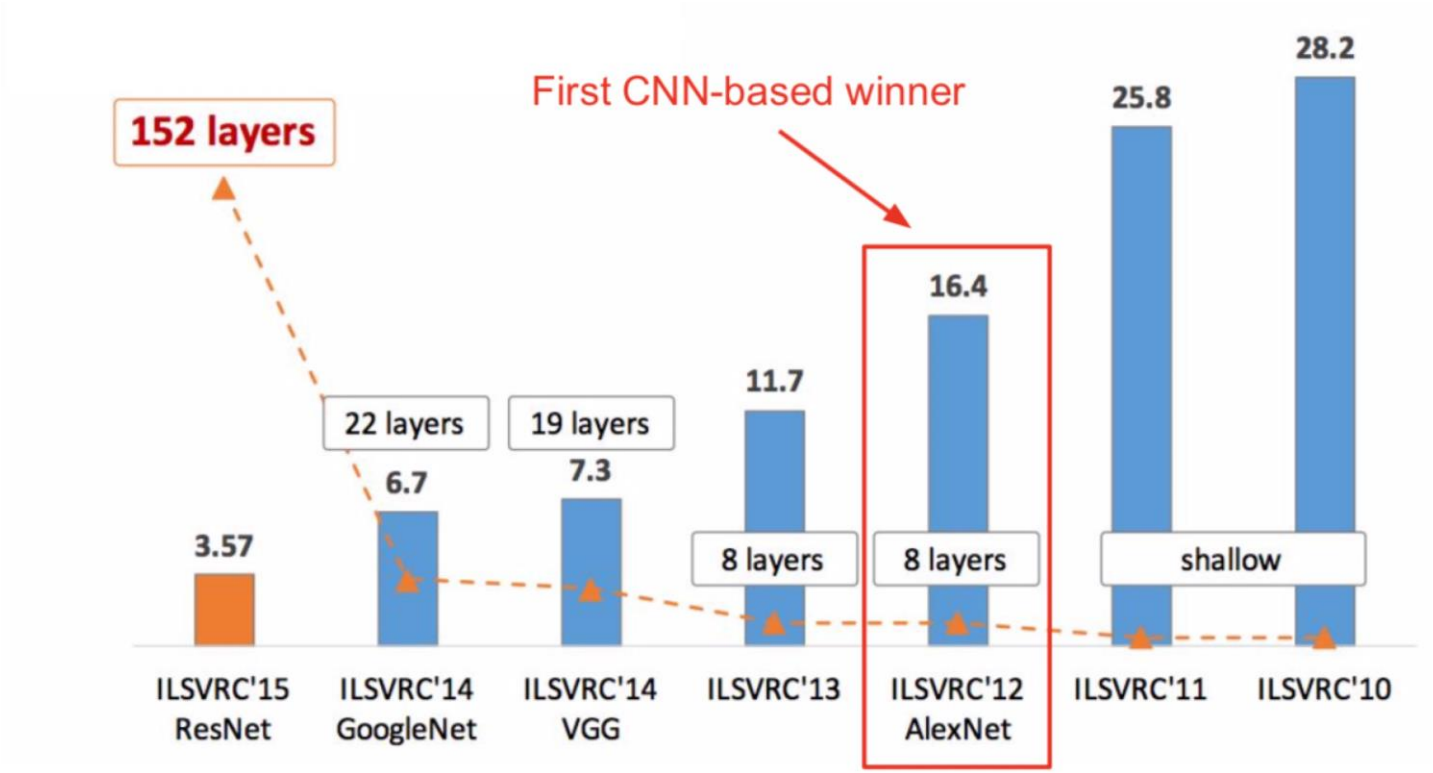
- AlexNet
- Vgg
- GoogleNet
- ResNet

AlexNet

- Architecture:
  - CONV1 MAX POOL1 NORM1
  - CONV2 MAX POOL2 NORM2
  - CONV3 CONV4 CONV5
  - Max POOL3
  - FC6 FC7 FC8



First CNN-based Winners

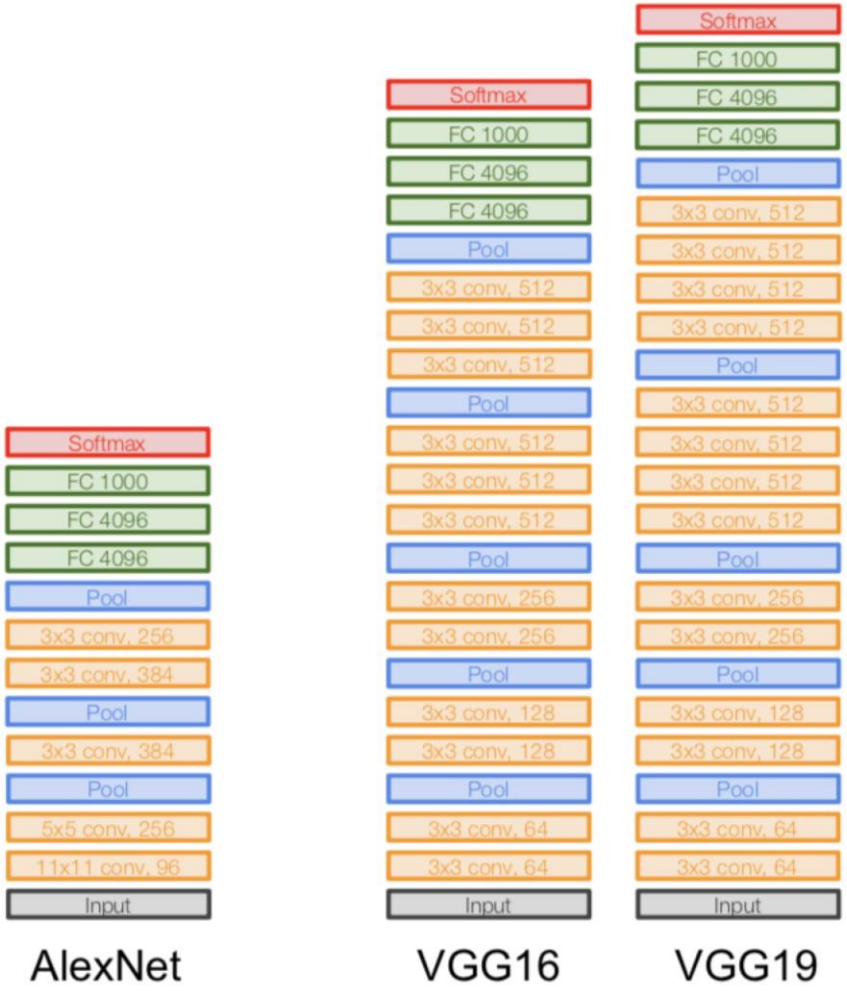


AlexNet

- **Details/Retrospectives:**
  - first use of ReLU
  - used Norm layers (not common anymore)
  - heavy data augmentation
  - 7 CNN ensemble: 18.2% -> 15.4%

VGGNet *[Simonyan and Zisserman, 2014]*

- Small filters, Deeper networks
  - 8 layers (AlexNet)->
    16 - 19 layers (VGG16Net)
  - Only 3x3 CONV stride 1, pad 1 and 2x2 MAX POOL
  - 15.4% top 5 error for AlexNet
    -> 7.3% top 5 error

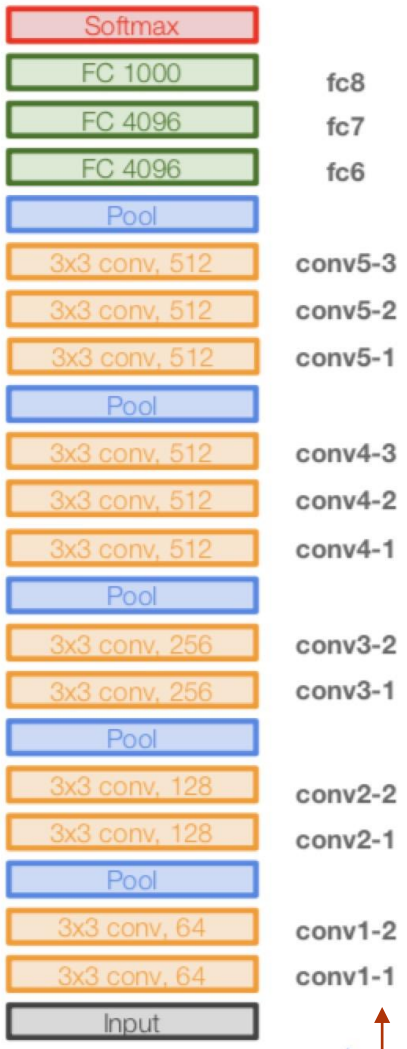


Why use smaller filters?

- Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer
- But deeper, more non-linearities
- And fewer parameters: 3
  - * (3^2C^2) vs. 7^2C^2 for C channels per layer

VGGNet

- Details:
  - Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
  - Use ensembles for best results
  - FC7 features generalize well to other tasks

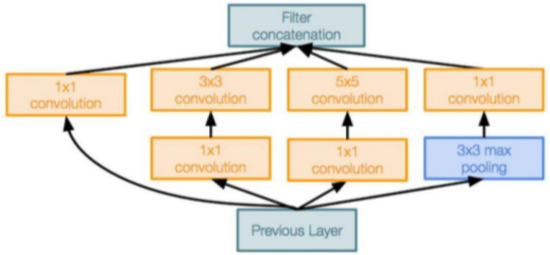


VGG16

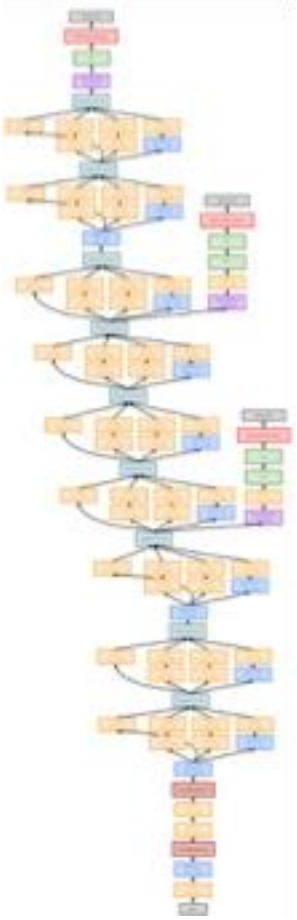
Common names

GoogleNet

- Deeper networks, with computational efficiency
  - 22 layers
  - Efficient “Inception” module
  - No FC layers
  - Only 5 million parameters!
  - 12x less than AlexNet
  - ILSVRC’14 classification winner
  - 6.7% top 5 error

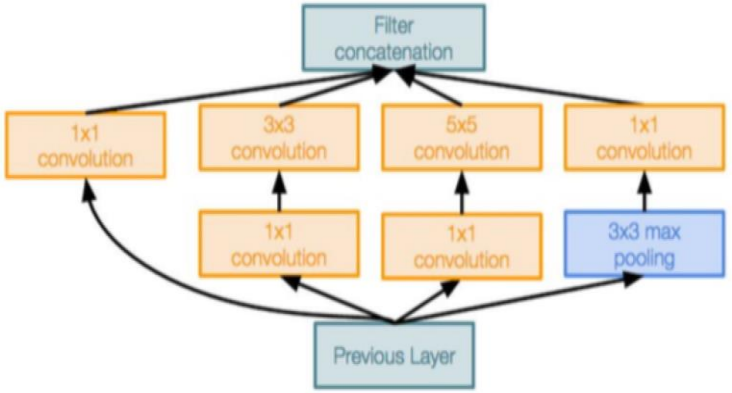


Inception module



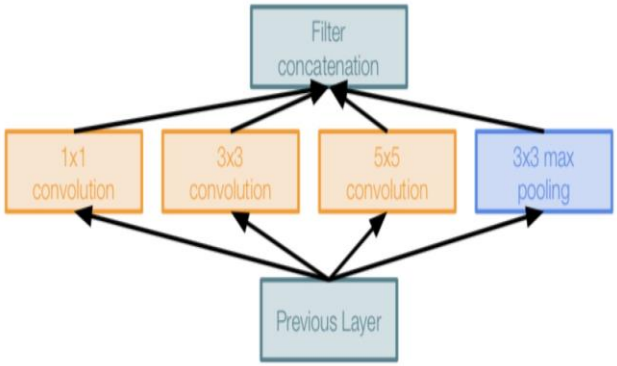
Inception module

- design a good local network topology (network within a network)
- stack these modules on top of each other

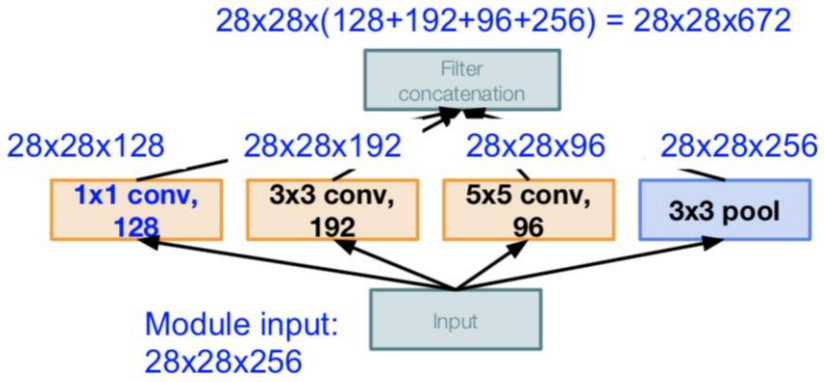


Naive Inception module

- Apply parallel filter operations on the input from previous layer:
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation
- Concatenate all filter outputs together depth-wise



Naive Inception module



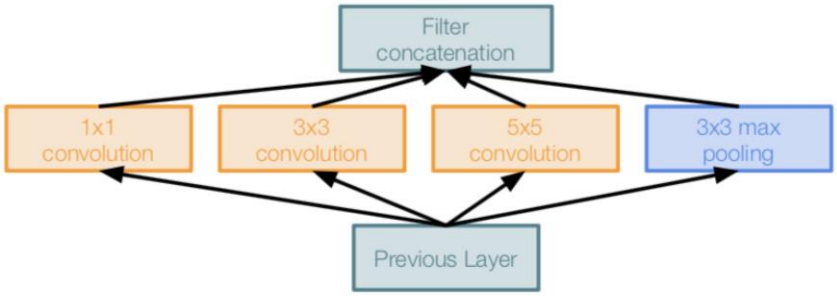
- **Conv Ops:**
- [1x1 conv, 128] 28x28x128x1x1x256
- [3x3 conv, 192] 28x28x192x3x3x256
- [5x5 conv, 96] 28x28x96x5x5x256
- **Total: 854M ops**

Very expensive compute

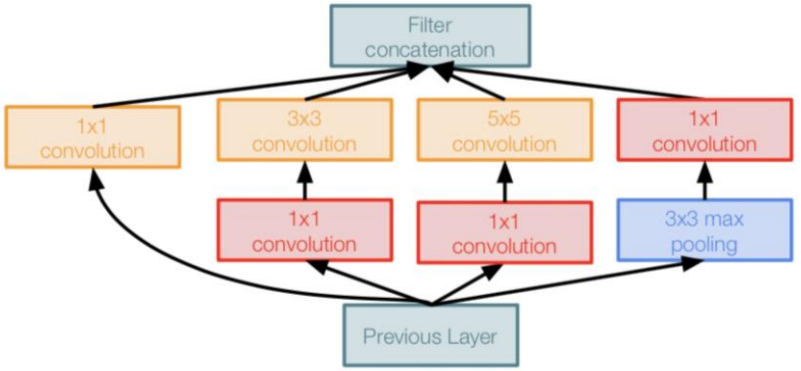
1x1 convolutions

Preserves spatial dimensions, reduces depth!

Inception module with dimension reduction

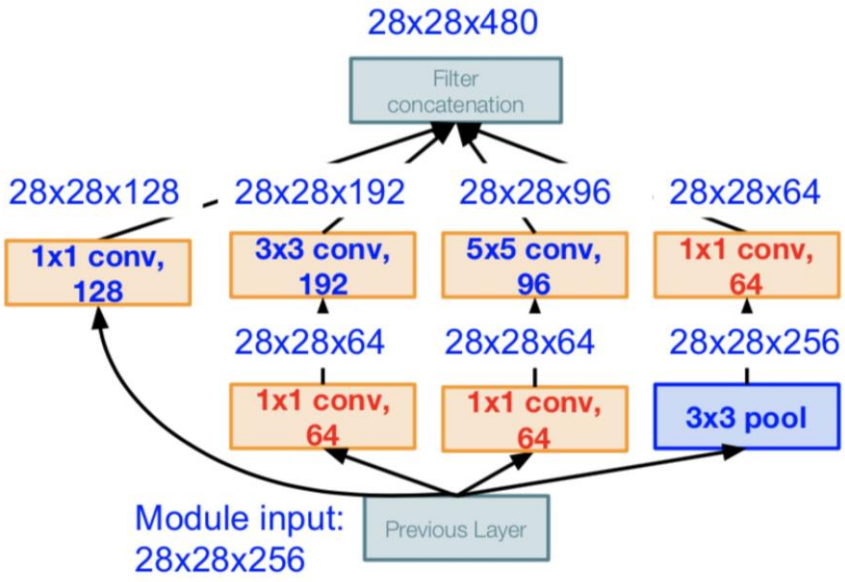


Naive Inception module



Inception module with dimension reduction

Inception module with dimension reduction

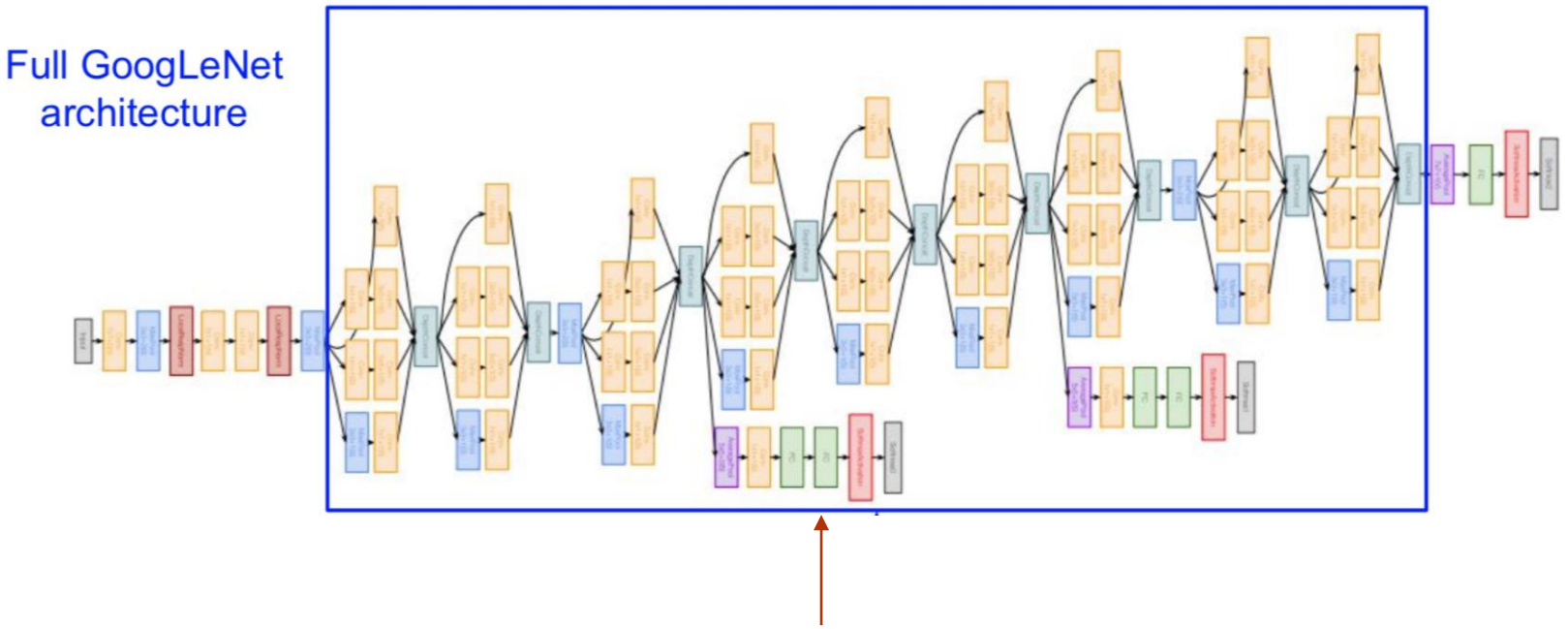


- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 128] 28x28x128x1x1x256
- [3x3 conv, 192] 28x28x192x3x3x64
- [5x5 conv, 96] 28x28x96x5x5x64
- [1x1 conv, 64] 28x28x64x1x1x256

Total: 358M ops

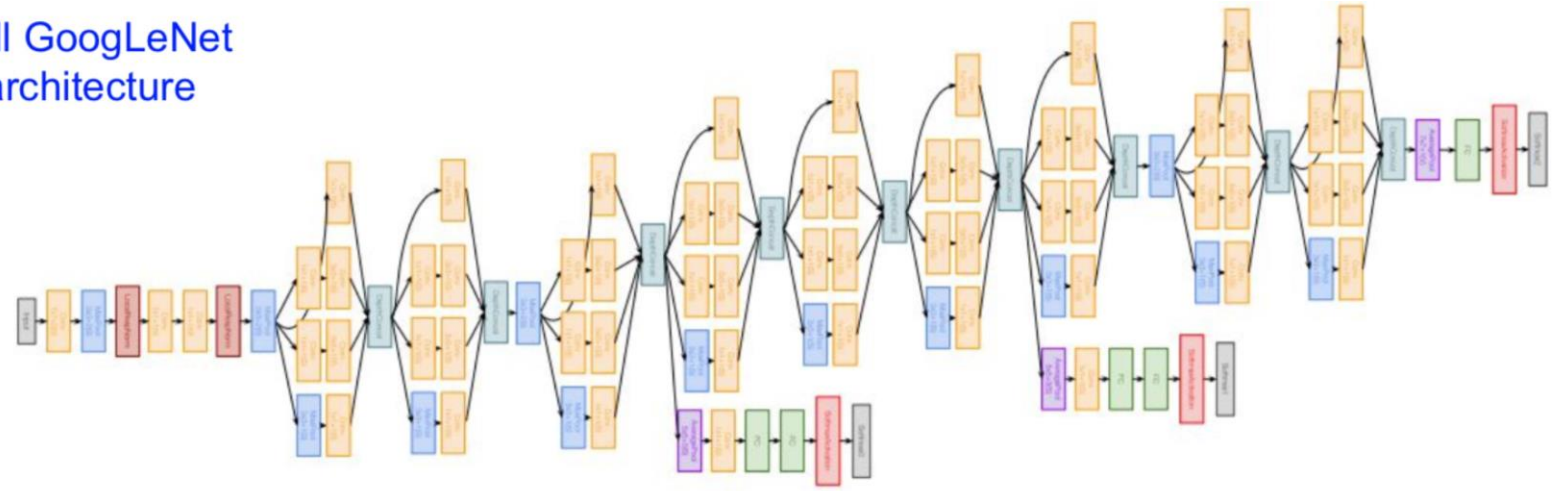
854M ops for naive version

GoogLeNet



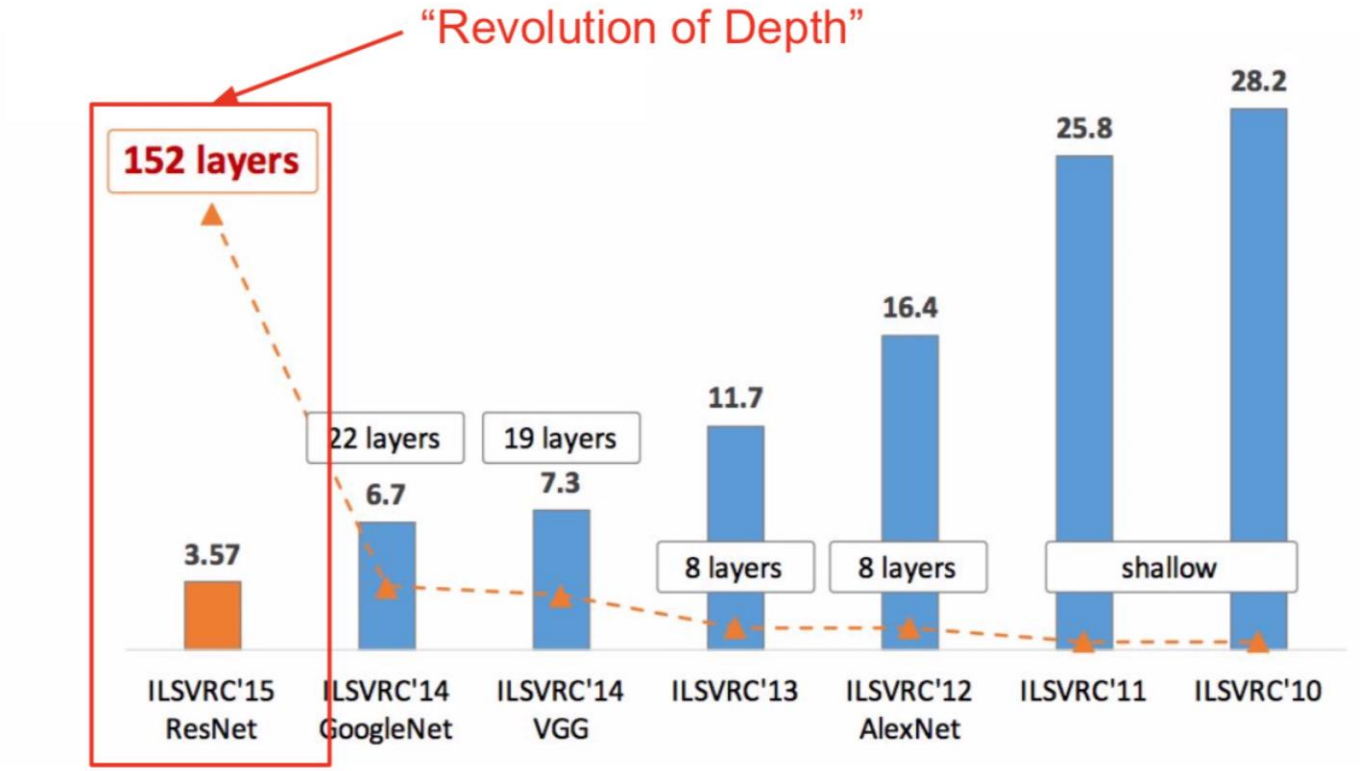
GoogLeNet

Full GoogLeNet architecture



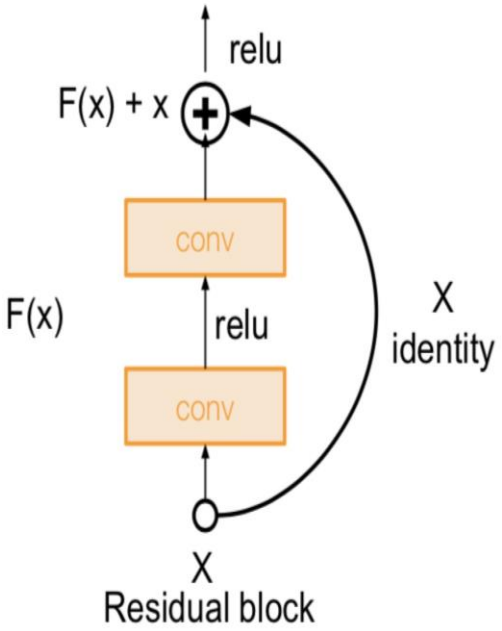
22 total layers with weights (including each parallel layer in an Inception module)

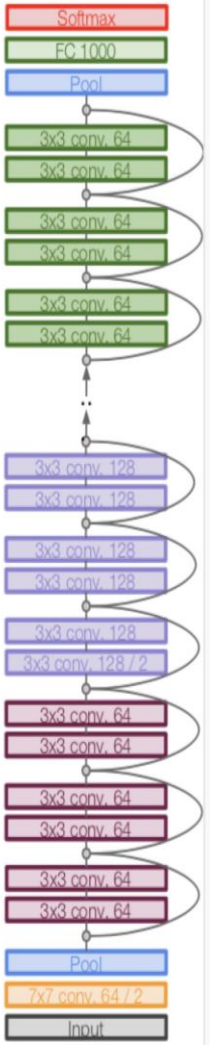
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners



ResNet [He et al., 2015]

- Very deep networks using residual connections
  - 152-layer model for ImageNet
  - -ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!





ResNet [He et al., 2015]

- What happens when we continue stacking deeper layers on a "plain" convolutional neural network?

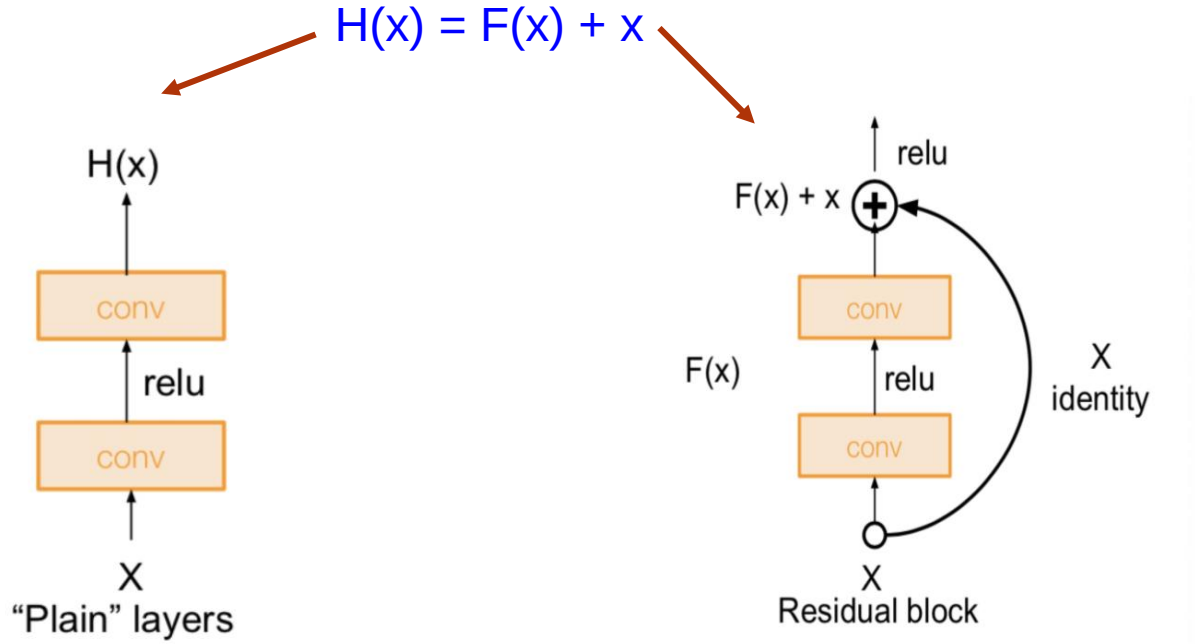
- 56-layer model performs worse on both training and test error
- The deeper model performs worse, but it's not caused by overfitting!

ResNet [He et al., 2015]

- Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize
  - The deeper model should be able to perform at least as well as the shallower model.
  - A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.

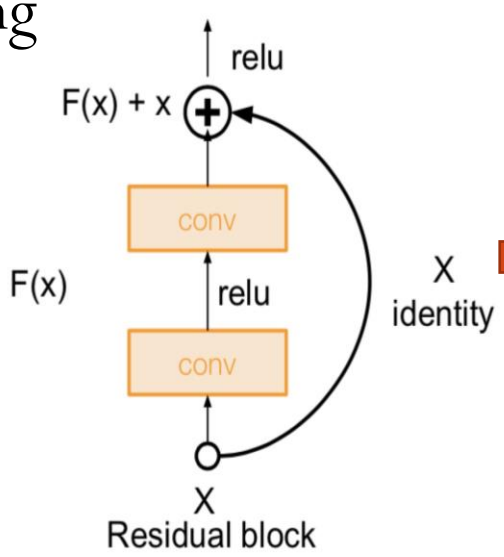
ResNet [He et al., 2015]

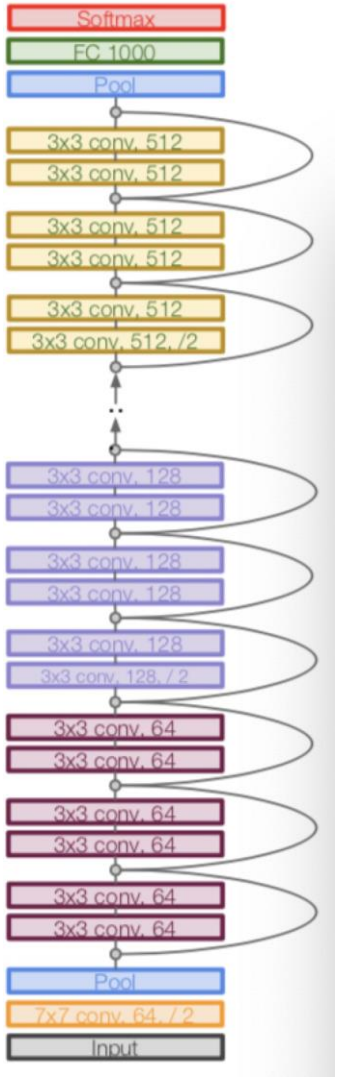
- Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping
  - If identity were optimal, easy to set weights as 0
  - If optimal mapping is closer to identity, easier to find small fluctuations



ResNet [He et al., 2015]

- Stack residual blocks
- Able to train very deep networks without degrading
- Deeper networks now achieve lowing training error as expected

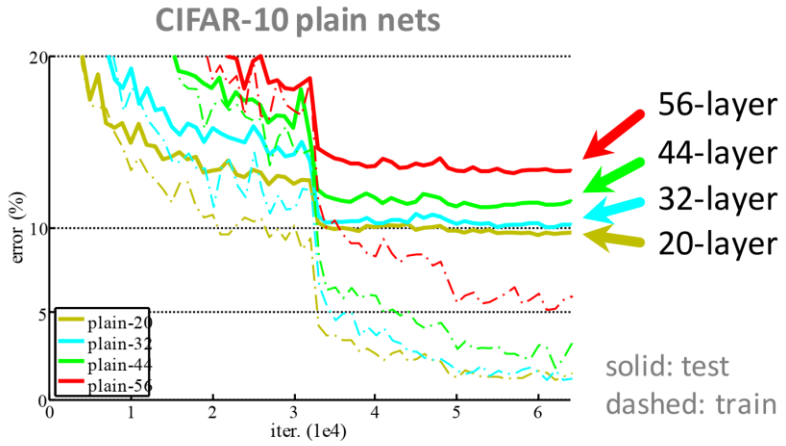


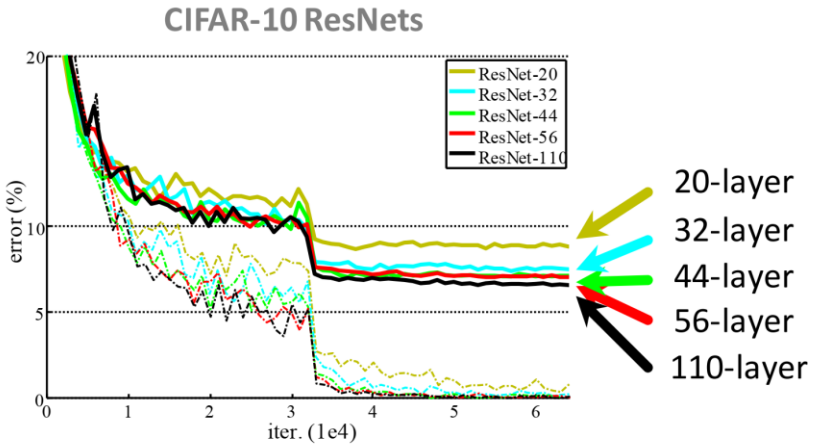


ILSVRC 2015 classification winner (3.6% top 5 error) -- better than “human performance”! (Russakovsky 2014)

CIFAR-10 experiments

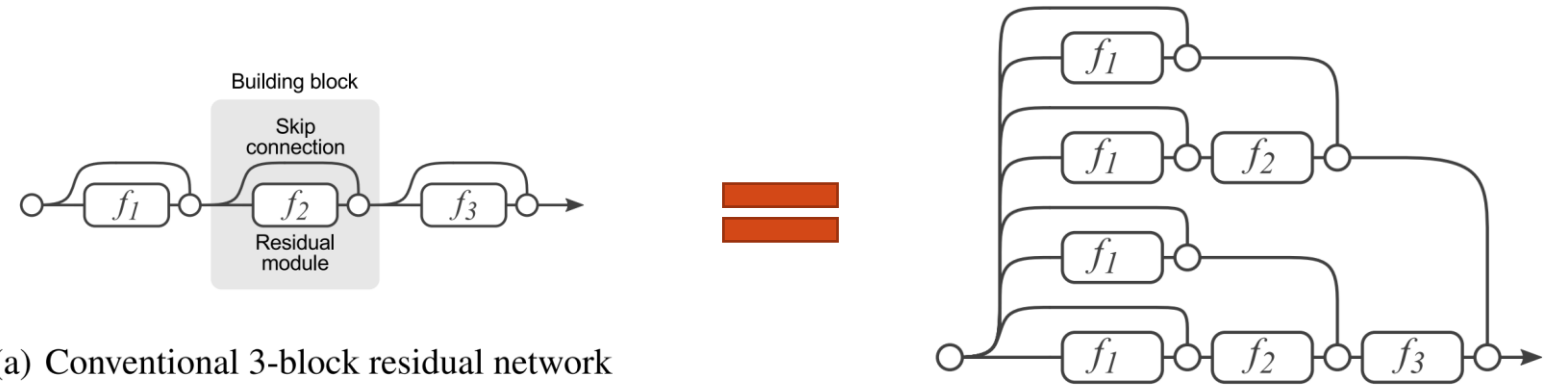
- Deep ResNets can be trained without difficulties
- Deeper ResNets have **lower training error**, and also lower test error





ResNets as Ensembles

- Can think of ResNets as ensembling subsets of residual modules
  - For each residual module, we can choose whether we include it
  - There are 2 options per module (include/exclude) for L modules
  - Total of 2^L modules in the implicit ensemble

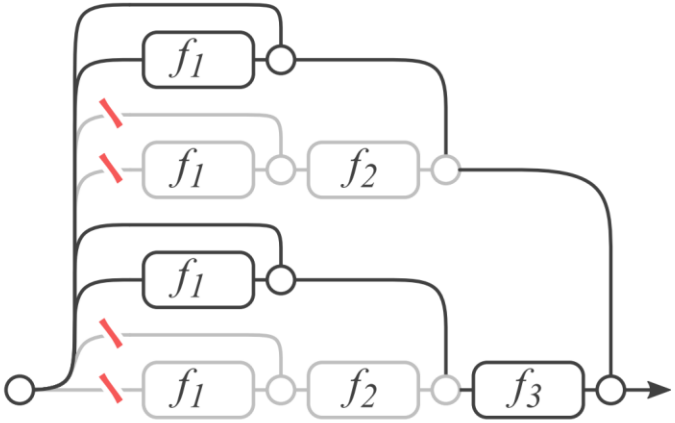


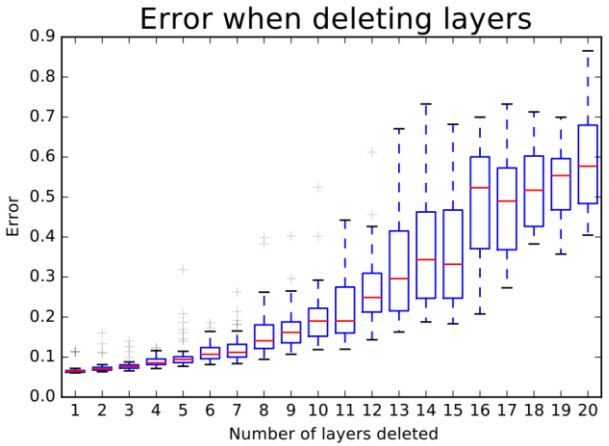
(a) Conventional 3-block residual network

(b) Unraveled view of (a)

ResNets as Ensembles

- If one of modules is removed, there are still 2^{L-1} possible subsets of modules
- Tried dropping and reordering layers





Issues on learning deep models

- Representation ability
  - Ability of model to fit training data, if optimum could be found
  - If model A's solution space is a superset of B's, A should be better
- Optimization ability
  - Feasibility of finding an optimum
  - Not all models are equally easy to optimize
- Generalization ability
  - Once training data is fit, how good is the test performance

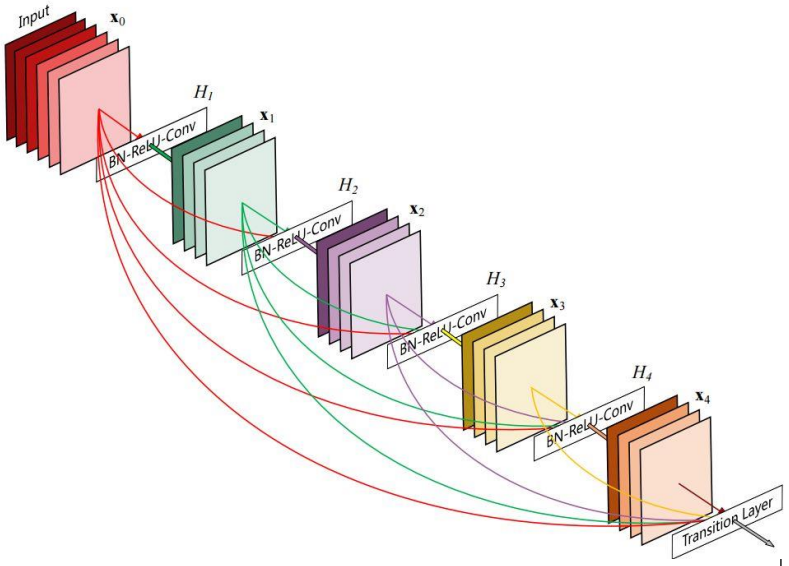
How do ResNets address these issues?

- Representation ability
  - Allow models to go deeper
- Optimization ability
  - Enable very smooth forward/backward prop
  - Greatly ease optimizing deeper models
- Generalization ability
  - Deeper + thinner is good generalization

DenseNet

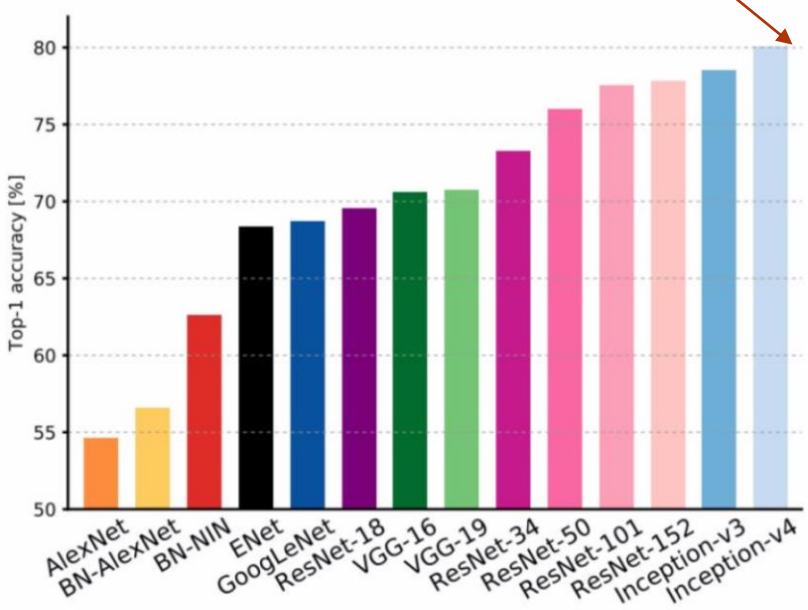
- Every layer is connected to all other layers
- For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers.

- Alleviate the vanishing-gradient problem.
- Strengthen feature propagation.
- Encourage feature reuse.
- Substantially reduce the number of parameters



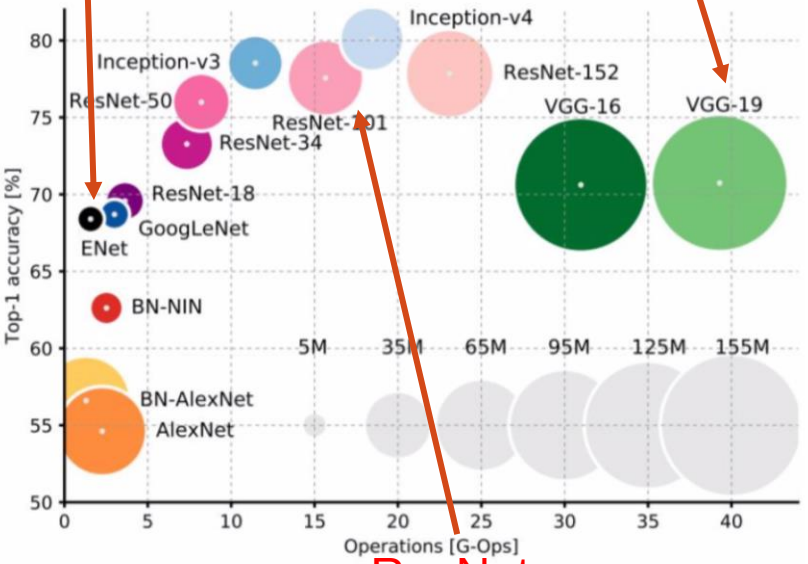
Comparison

Inception-v4: Resnet + Inception!



GoogleNet:
most efficient

VGG: Highest memory,
most operations



ResNet:
Moderate efficiency depending on model, highest accuracy

An Analysis of Deep Neural Network Models for Practical Applications, 2017

Summary: CNN Architectures

- Neural network with specialized connectivity structure
  - Feed-forward:
  - Convolve input
  - Non-linearity (rectified linear)
  - Pooling (local max)
- Supervised training
- Train convolutional filters by back-propagating error
- VGG, GoogLeNet, ResNet all widely used
- ResNet current best default

Summary

- At surface level, there's tons of new architectures that are very different
- Upon closer inspection, most of them are reapplying well established principles
  - Reduce filter sizes and factorize filters aggressively
  - Use 1x1 convolutions to reduce and expand the number of feature maps judiciously
  - Use skip connections and/or create multiple paths through the network
  - Identity propagation (Residuals, Dense Blocks) seem to make training easier

Thank You!