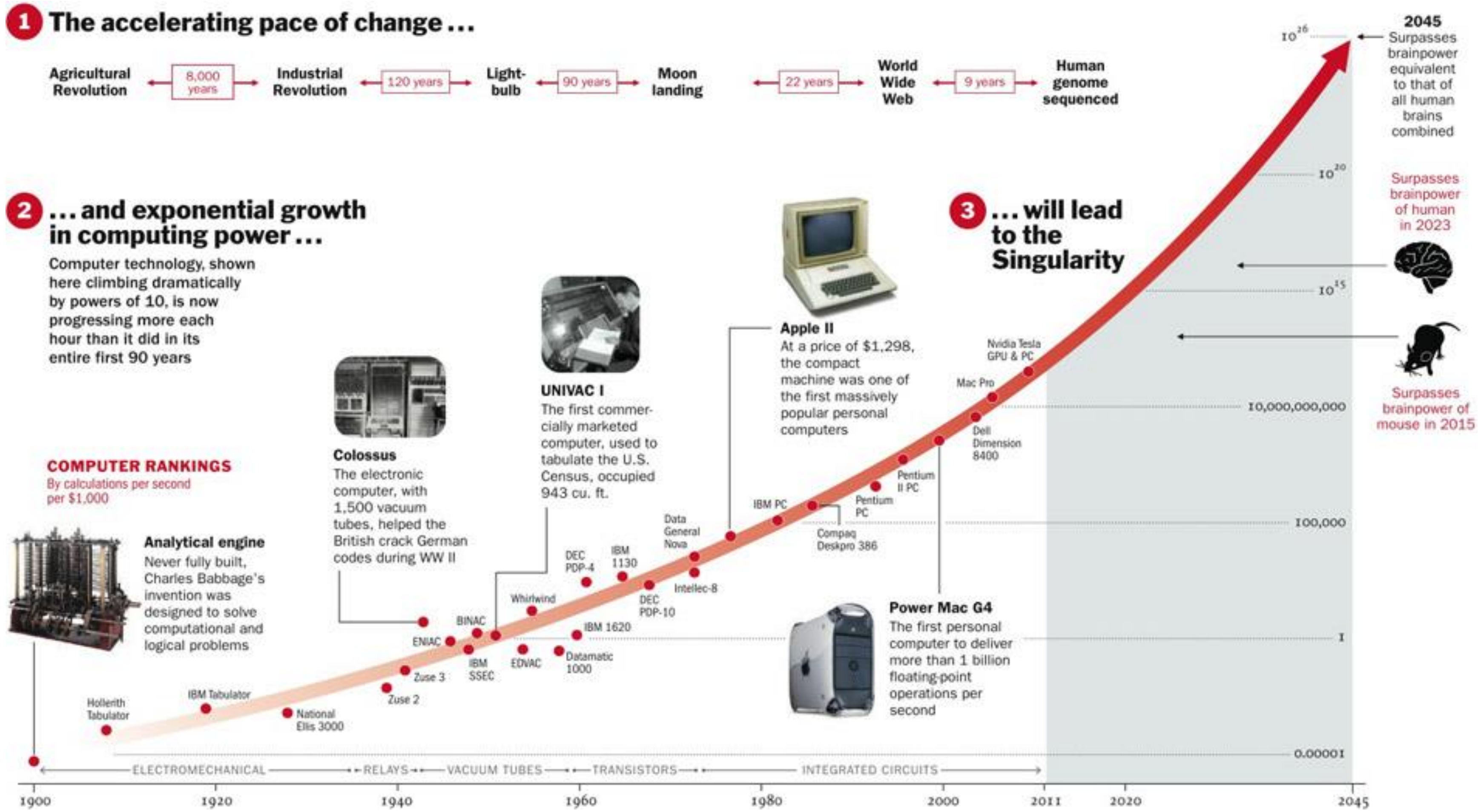
Fundamentals of Deep (Artificial) Neural Networks (DNN)

Greg Tsagkatakis

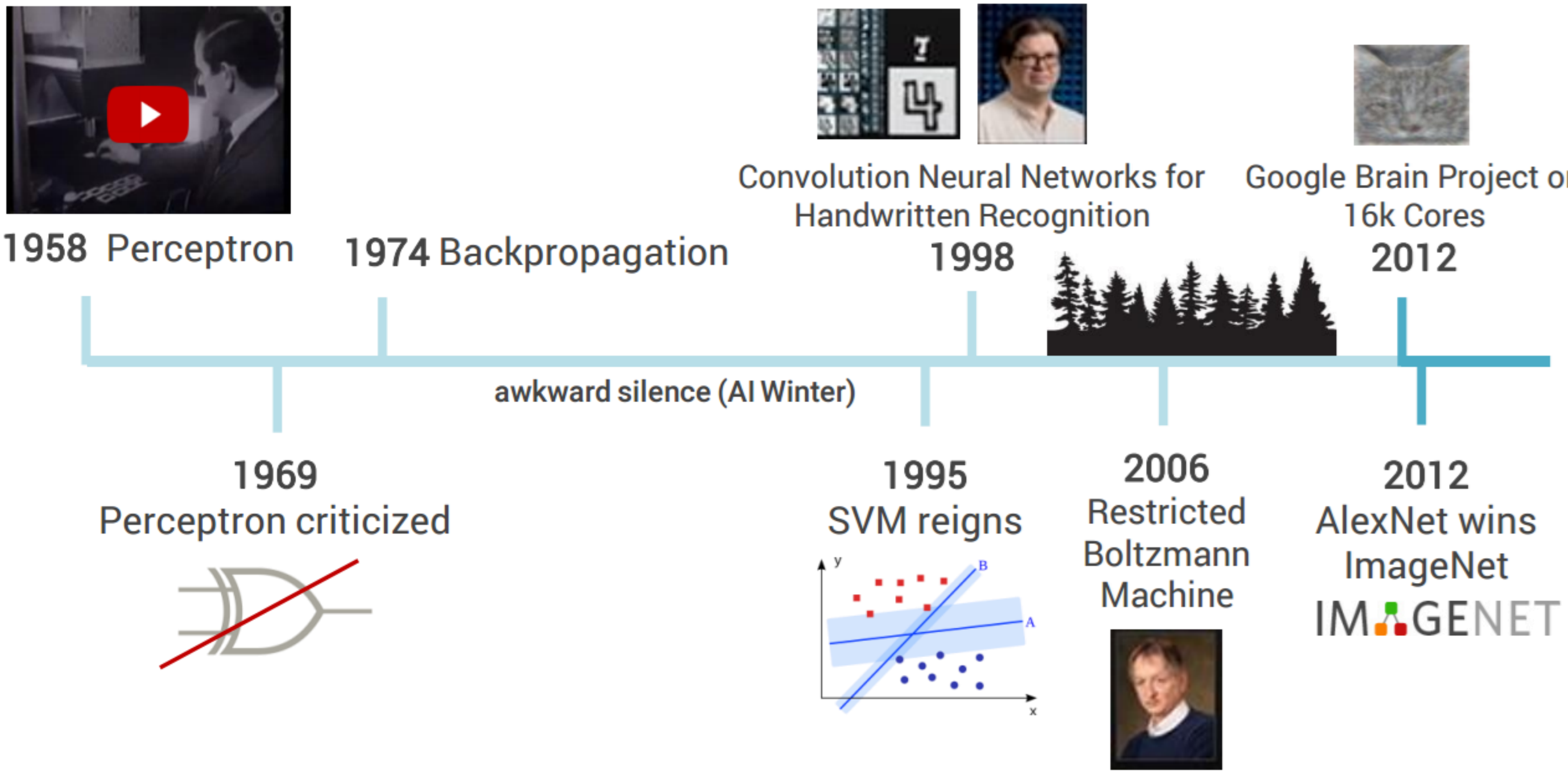
CSD - UOC

ICS - FORTH

Accelerated growth

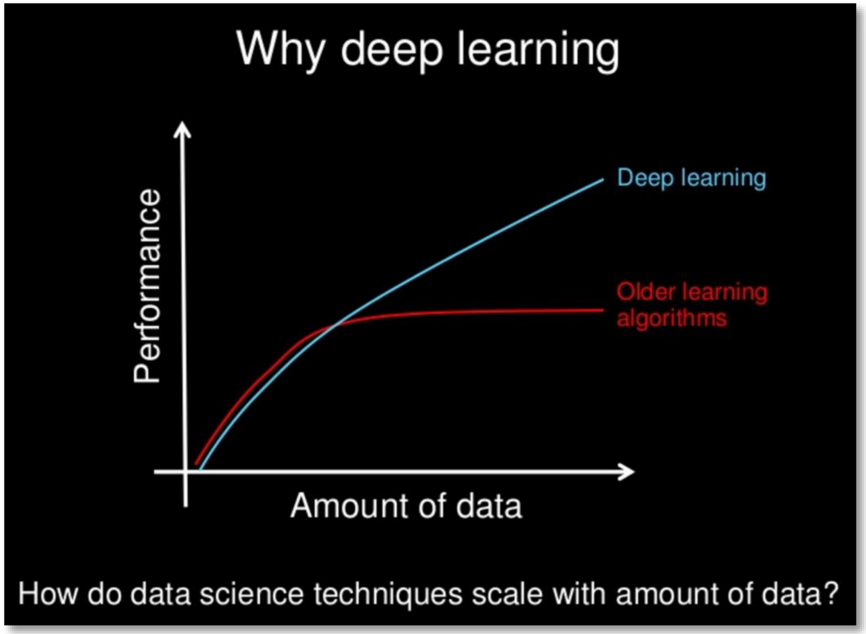


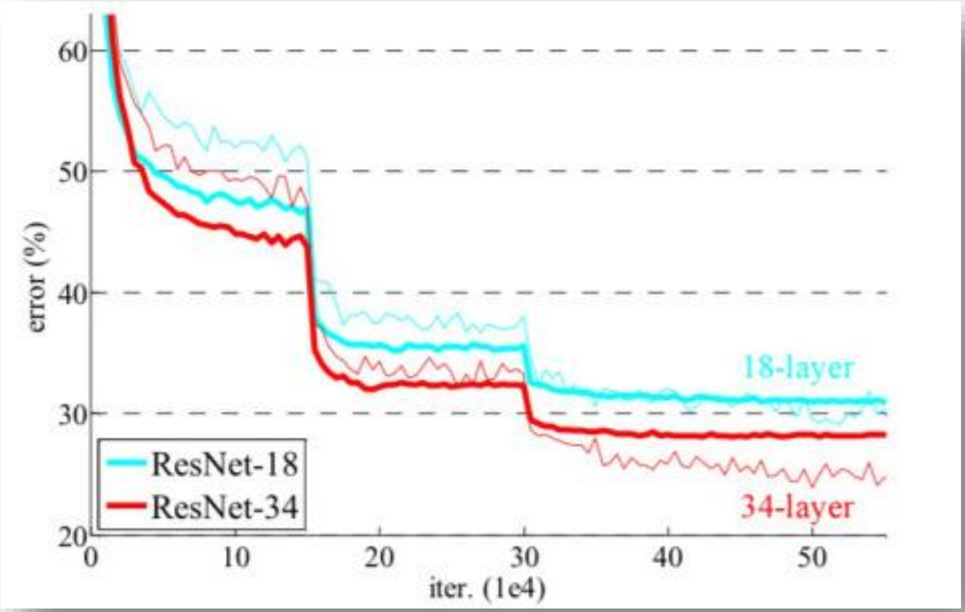
Brief history of DL



Why Today?

Lots of Data

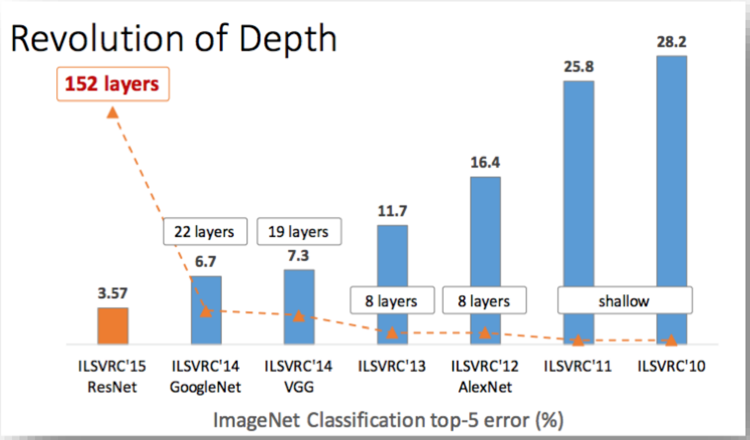


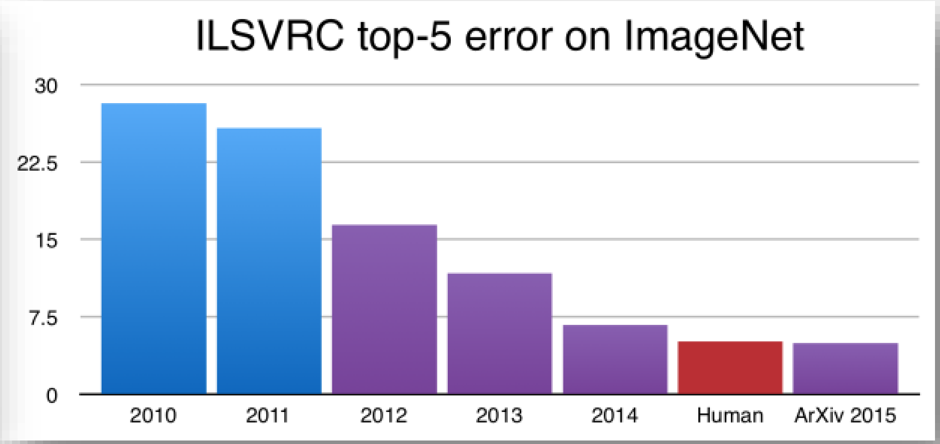


Why Today?

Lots of Data

Deeper Learning



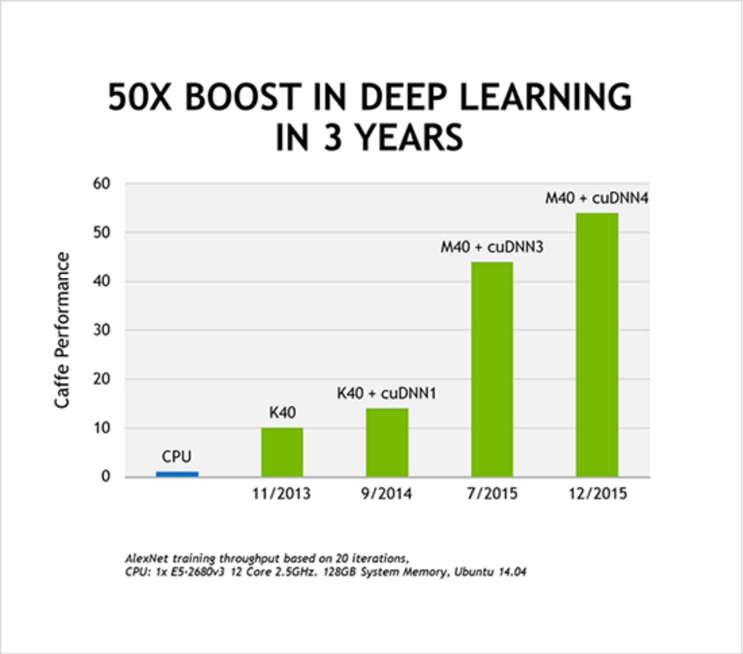


Why Today?

Lots of Data

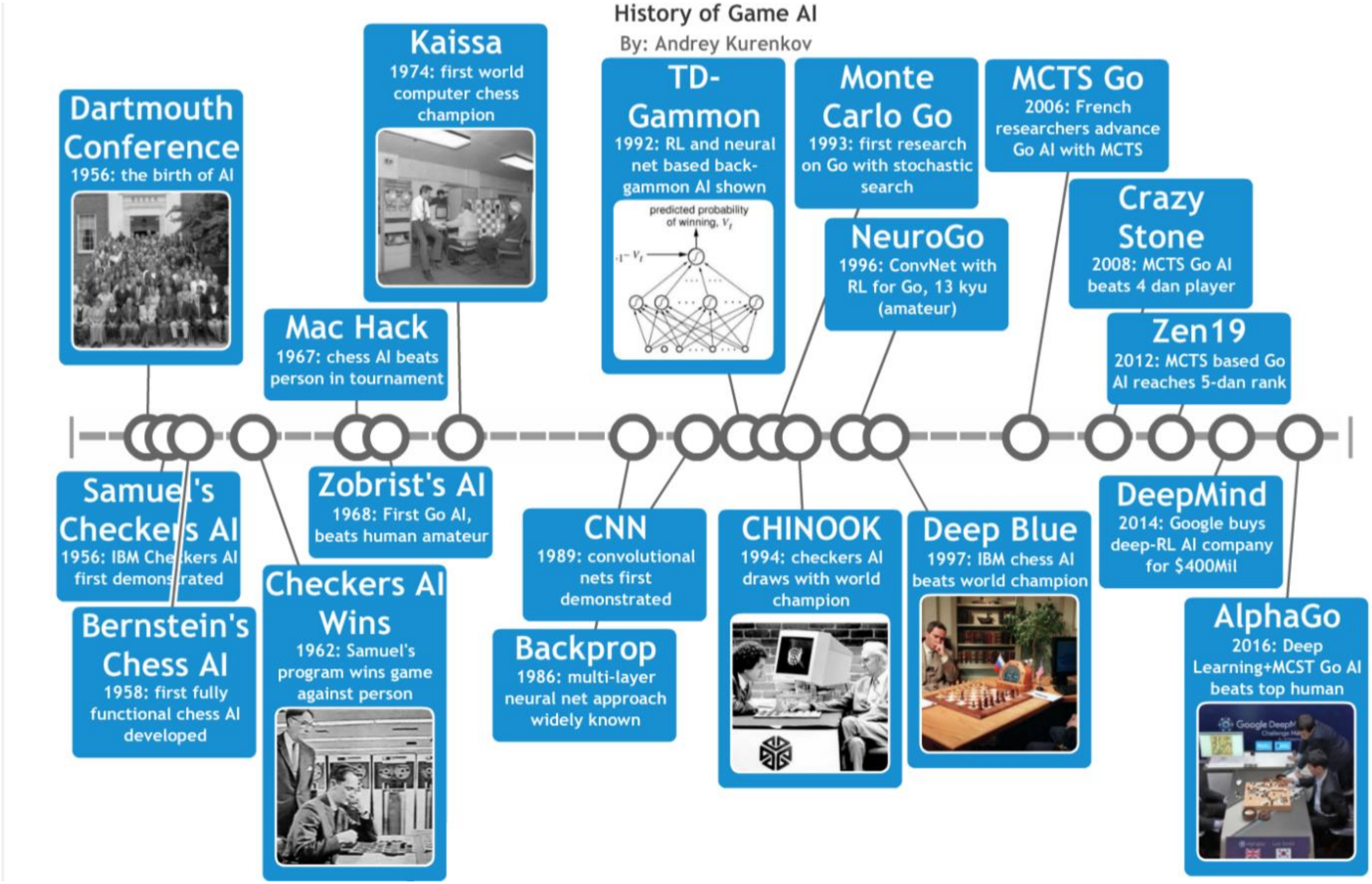
Deep Learning

More Power



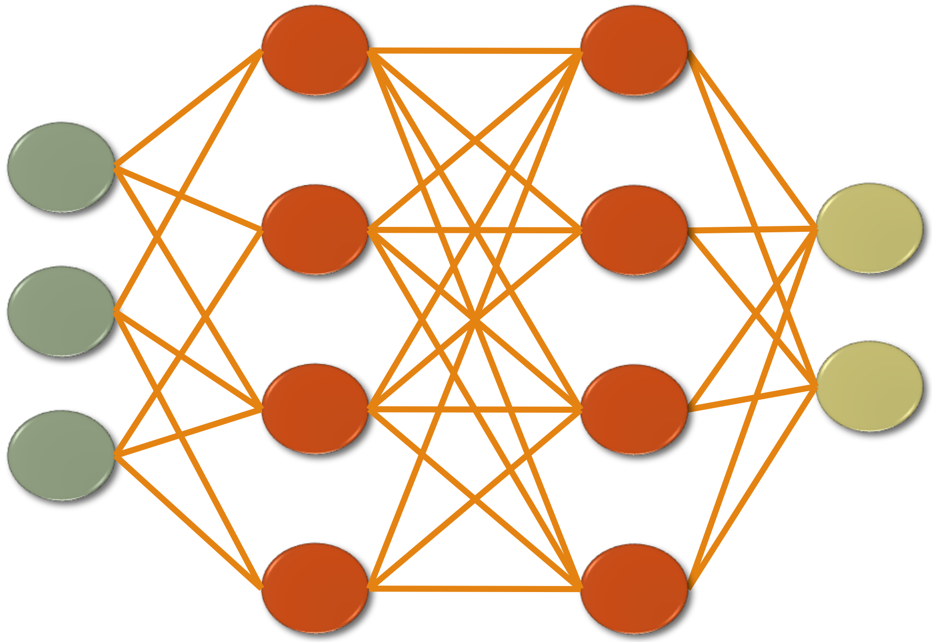
https://blogs.nvidia.com/blog/2016/01/12/accelerating-ai-artificial-intelligence-gpus/
https://www.slothparadise.com/what-is-cloud-computing/

Apps: Gaming


History of Game AI
By: Andrey Kurenkov
Dartmouth Conference
1956: the birth of AI
Kaissa
1974: first world computer chess champion
TD-Gammon
1992: RL and neural net based back-gammon AI shown
predicted probability of winning, V_t
Monte Carlo Go
1993: first research on Go with stochastic search
MCTS Go
2006: French researchers advance Go AI with MCTS
Crazy Stone
2008: MCTS Go AI beats 4 dan player
NeuroGo
1996: ConvNet with RL for Go, 13 kyu (amateur)
Mac Hack
1967: chess AI beats person in tournament
Zen19
2012: MCTS based Go AI reaches 5-dan rank
Samuel's Checkers AI
1956: IBM Checkers AI first demonstrated
Zobrist's AI
1968: First Go AI, beats human amateur
CNN
1989: convolutional nets first demonstrated
CHINOOK
1994: checkers AI draws with world champion
Deep Blue
1997: IBM chess AI beats world champion
DeepMind
2014: Google buys deep-RL AI company for $400Mil
Bernstein's Chess AI
1958: first fully functional chess AI developed
Checkers AI Wins
1962: Samuel's program wins game against person
Backprop
1986: multi-layer neural net approach widely known
AlphaGo
2016: Deep Learning+MCST Go AI beats top human

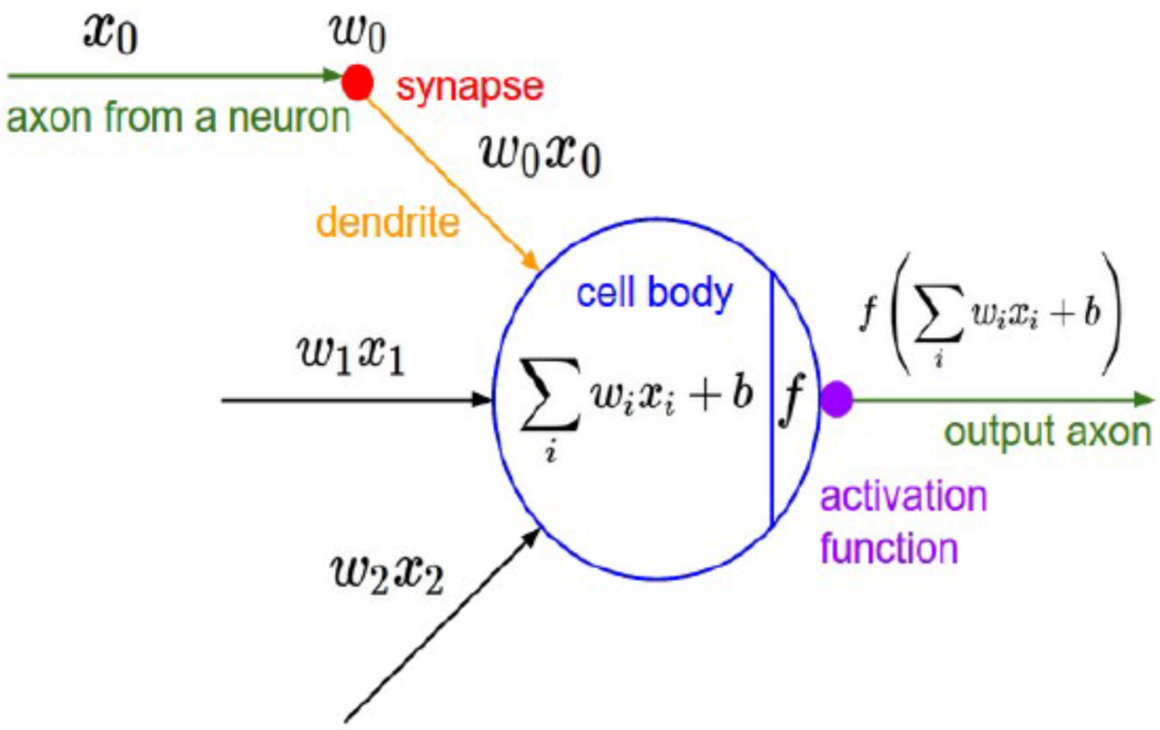
Key components of ANN

- Architecture (input/hidden/output layers)



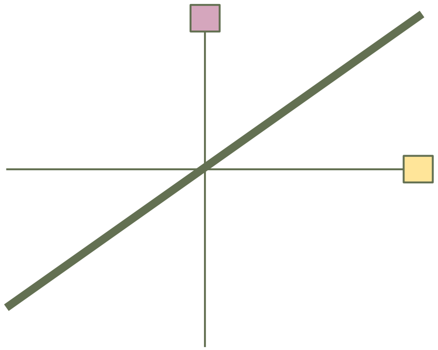
Key components of ANN

- Architecture (input/hidden/output layers)
- Weights

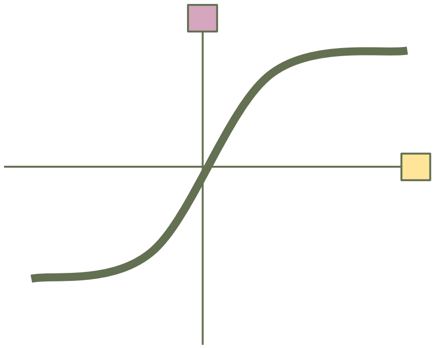


Key components of ANN

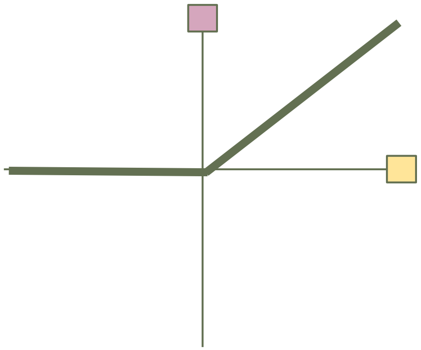
- Architecture (input/hidden/output layers)
- Weights
- Activations



LINEAR



LOGISTIC / SIGMOIDAL / TANH



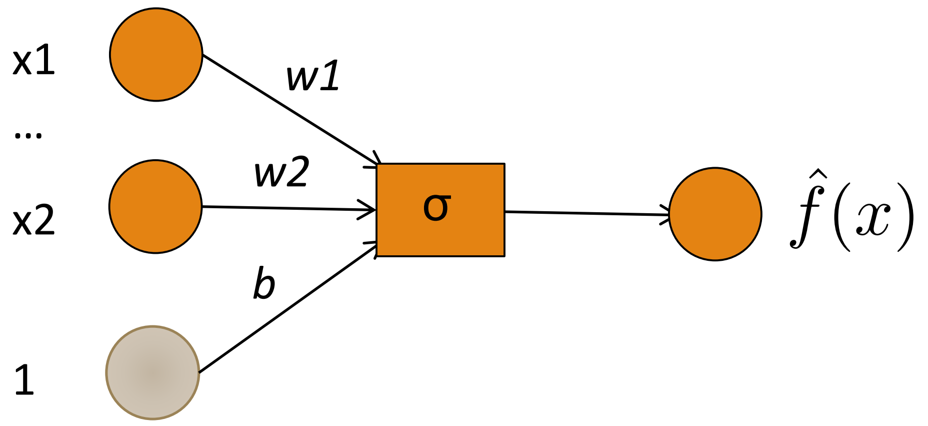
RECTIFIED LINEAR (ReLU)

Perceptron: an early attempt

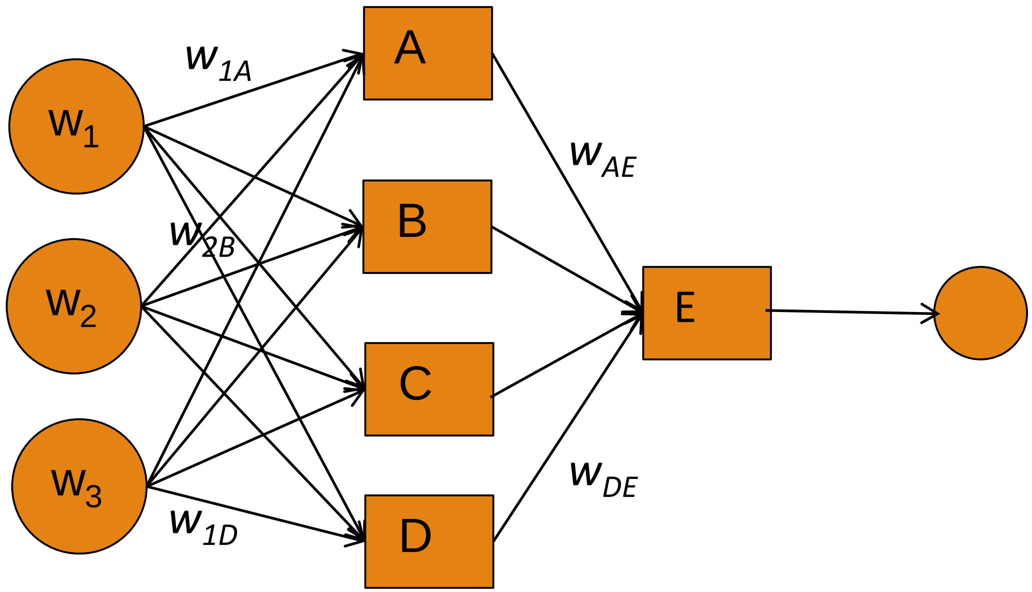
Activation function

$$\hat{f}(x) = \sigma(w \cdot x + b) \qquad \sigma(y) = \begin{cases} 1, & y > 0 \\ 0, & o/w \end{cases}$$

Need to tune w and b



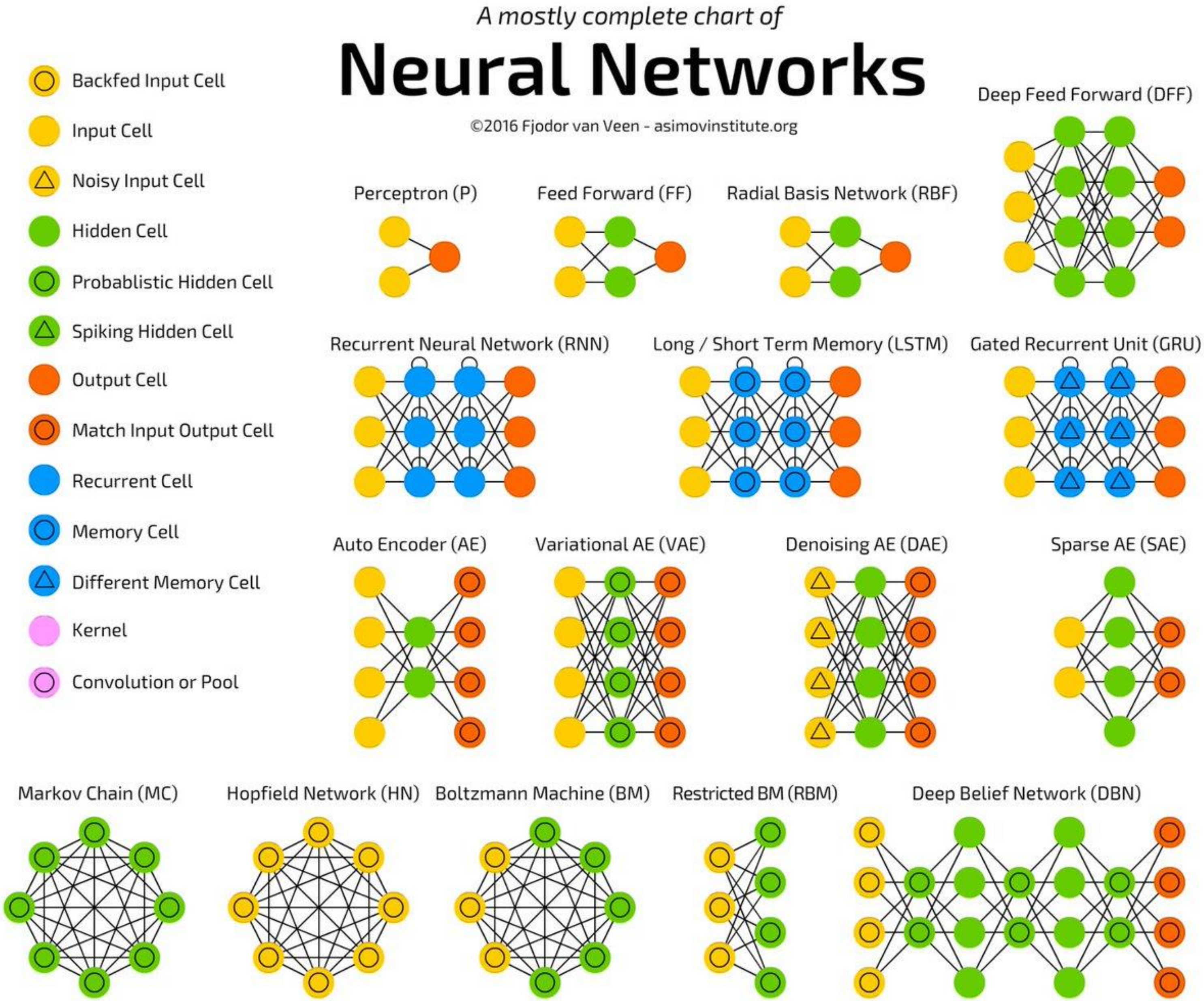
Multilayer perceptron



We just added a neuron layer!

We just introduced non-linearity!

A neuron is of the form **σ(w.x + b)** where **σ** is an *activation* function



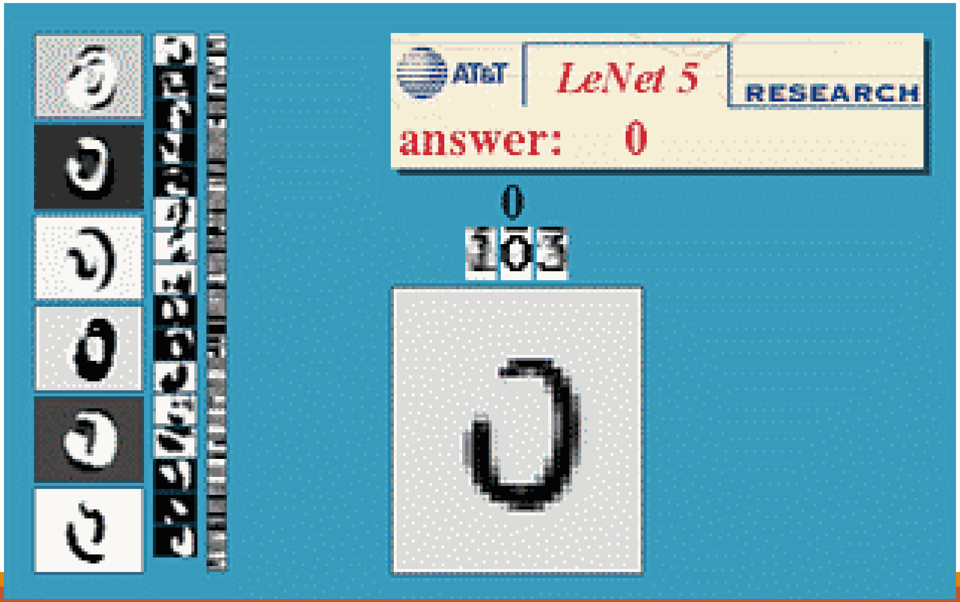
[Sasen Cain (@spectralradius)]

Training & Testing

Training: determine weights

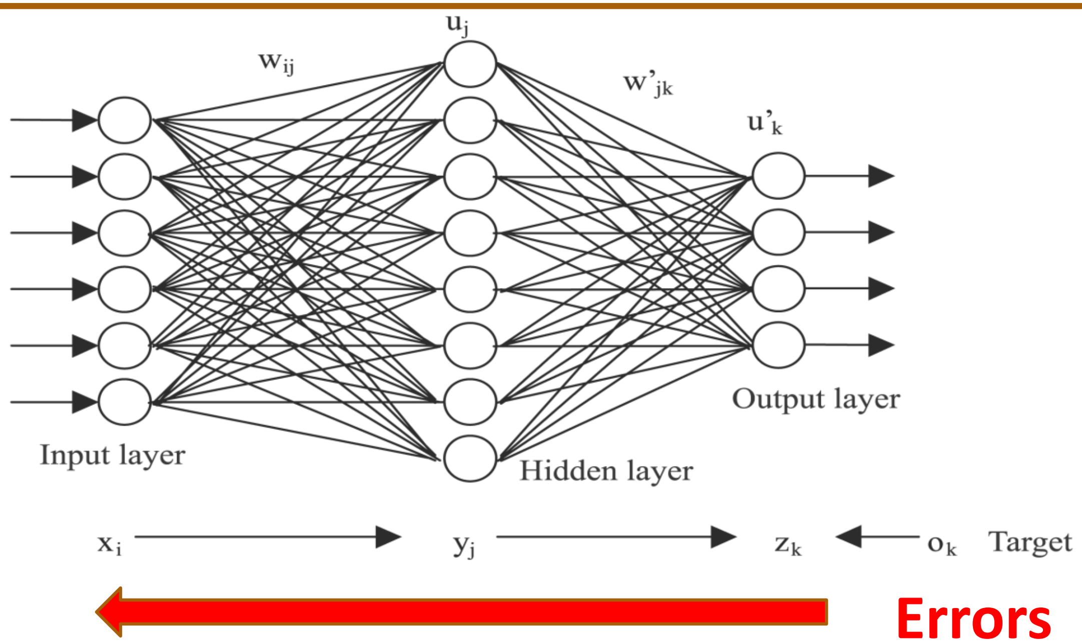
- Supervised: labeled training examples
- Unsupervised: no labels available
- Reinforcement: examples associated with rewards

Testing (Inference): apply weights to new examples



Training DNN

1. Get batch of data
2. Forward through the network -> estimate loss
3. Backpropagate error
4. Update weights based on gradient



BackPropagation

Chain Rule in Gradient Descent: Invented in 1969 by Bryson and Ho

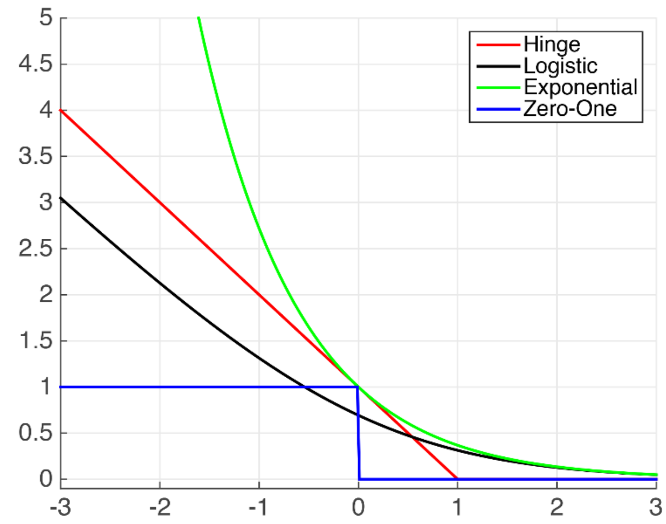
Defining a loss/cost function

Assume a function $J(x, y; \theta) = \frac{1}{2} \sum (y - f(x; \theta))^2$

$$f(x; \theta) = w^T x + b \quad , \quad \theta = \{w, b\}$$

Types of Loss function

- Hinge $J(x, y) = max\{0, 1 - xy\}$
- Exponential $J(x, y) = exp(-xy)$
- Logistic $J(x, y) = log_2(1 + exp(-xy))$



Gradient Descent

- Minimize function J w.r.t. parameters θ

New weights

$$\theta^* = \theta - n * \nabla J(y, x; \theta)$$

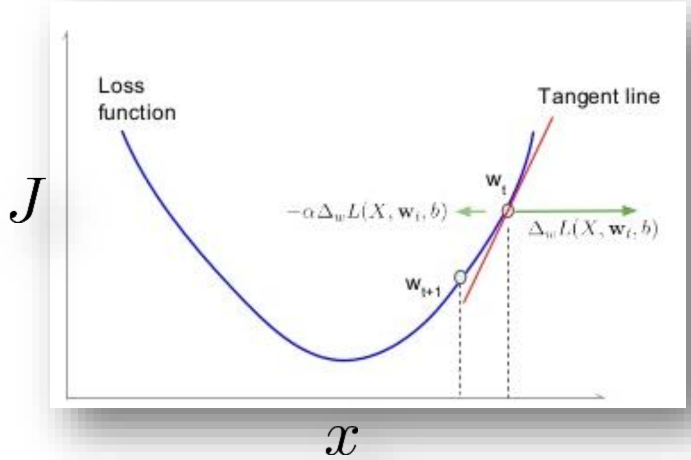
Gradient

Old weights

Learning rate

- Gradient

$$\nabla J(x) = \left(\frac{\partial J(x)}{\partial x_1}, \frac{\partial J(x)}{\partial x_2}, \ldots, \frac{\partial J(x)}{\partial x_n}\right)$$



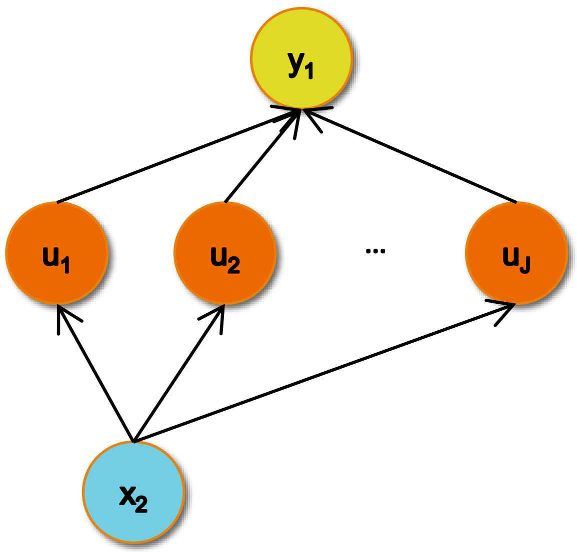
- Chain rule

BackProp

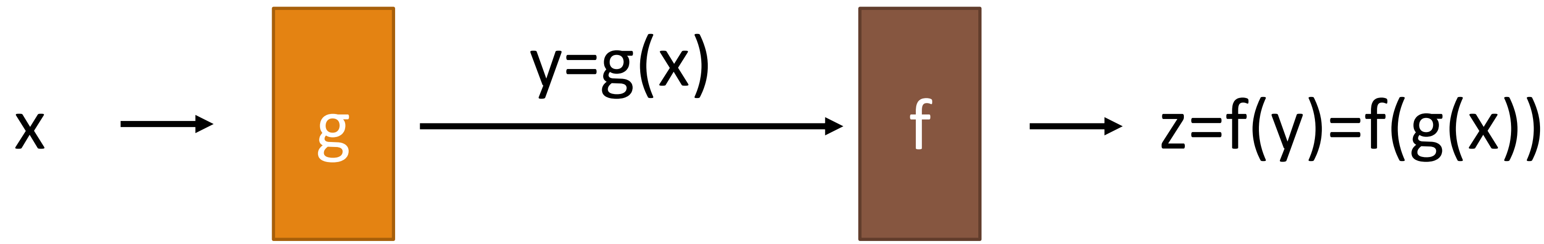
Given: $\boldsymbol{y} = g(\boldsymbol{u})$ and $\boldsymbol{u} = h(\boldsymbol{x})$.

Chain Rule:

$$\frac{dy_i}{dx_k} = \sum_{j=1}^{J} \frac{dy_i}{du_j}\frac{du_j}{dx_k}, \quad \forall i, k$$



BackProp



Chain rule:

- **Single variable**

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}.$$

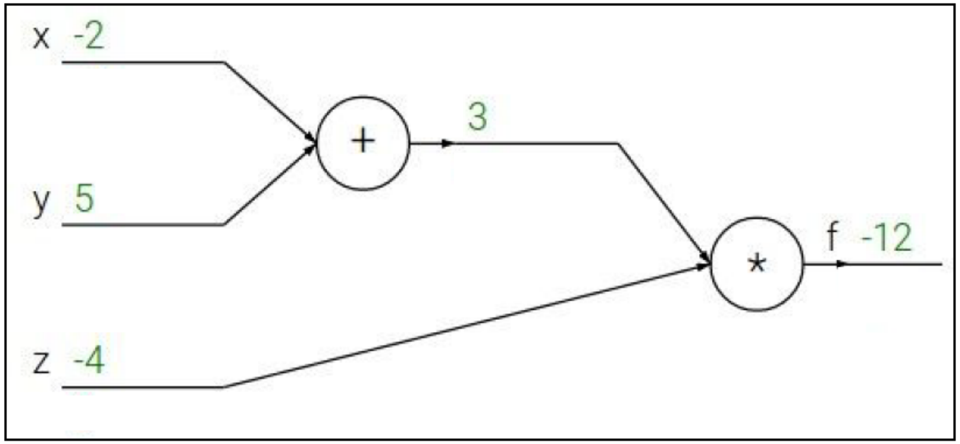
- **Multiple variables**

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j}\frac{\partial y_j}{\partial x_i}.$$

Backpropagation: a simple example

$$f(x,y,z) = (x+y)z$$

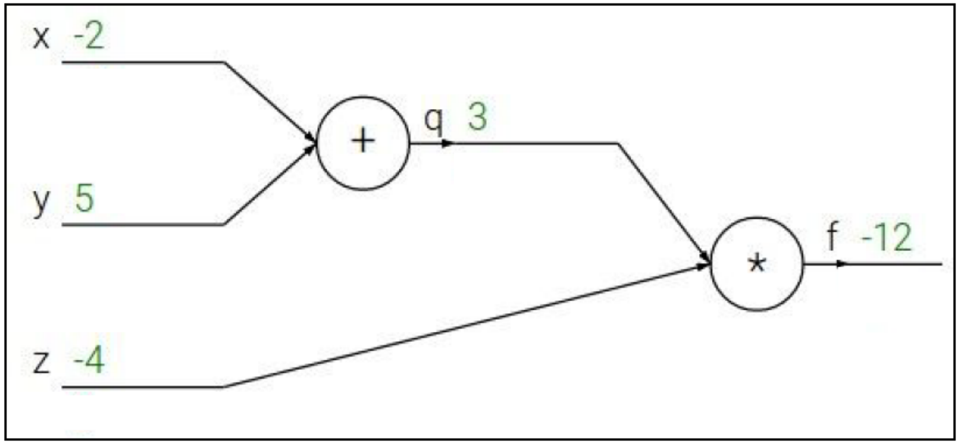
e.g. x = -2, y = 5, z = -4



Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation: a simple example

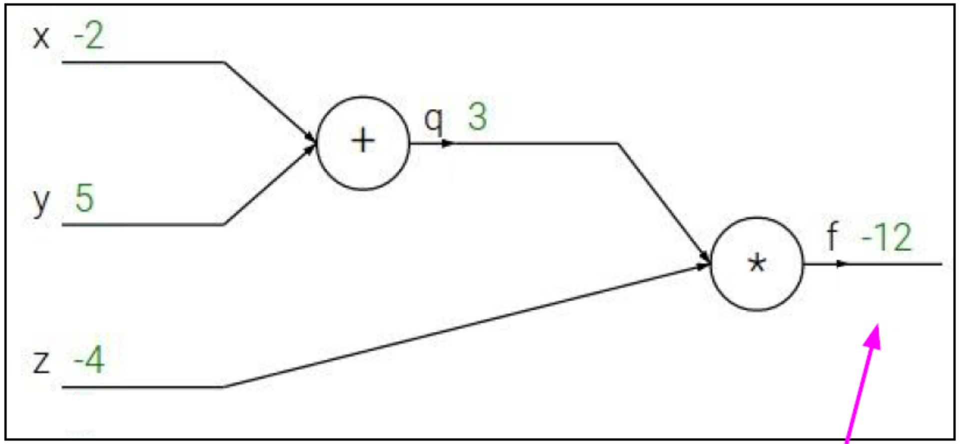
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$\boxed{\frac{\partial f}{\partial f}}$

Backpropagation: a simple example

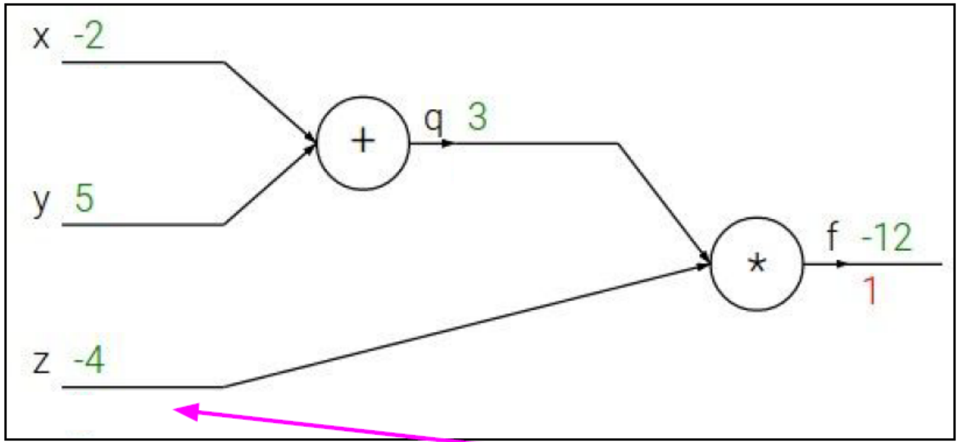
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

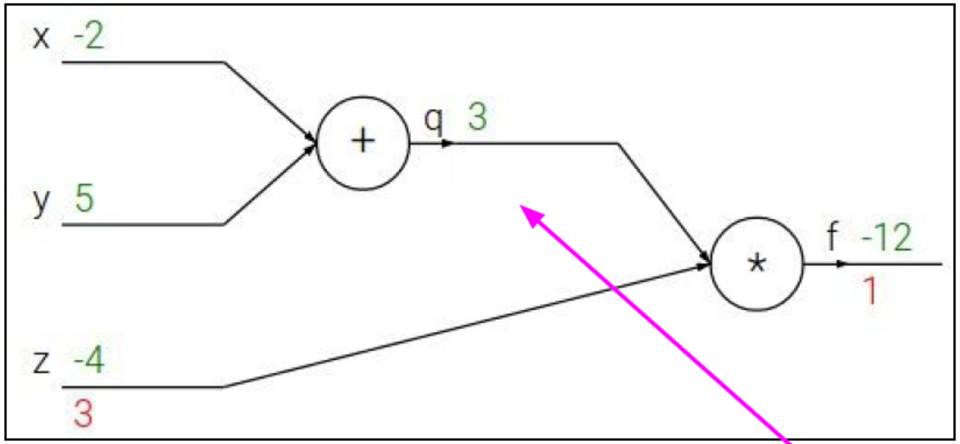


$$\frac{\partial f}{\partial z}$$

Backpropagation: a simple example

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



$$\boxed{q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1}$$

$$\boxed{f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q}$$

$$\boxed{\frac{\partial f}{\partial q}}$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

Backpropagation: a simple example

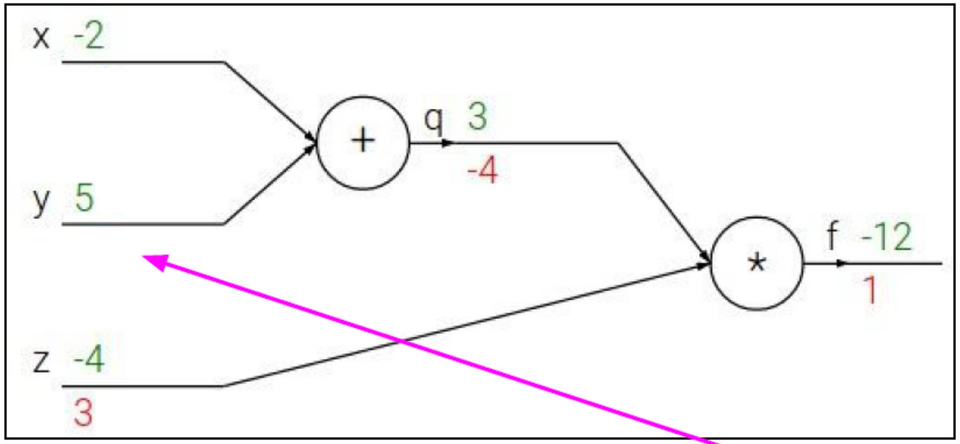
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial y}$$

Backpropagation: a simple example

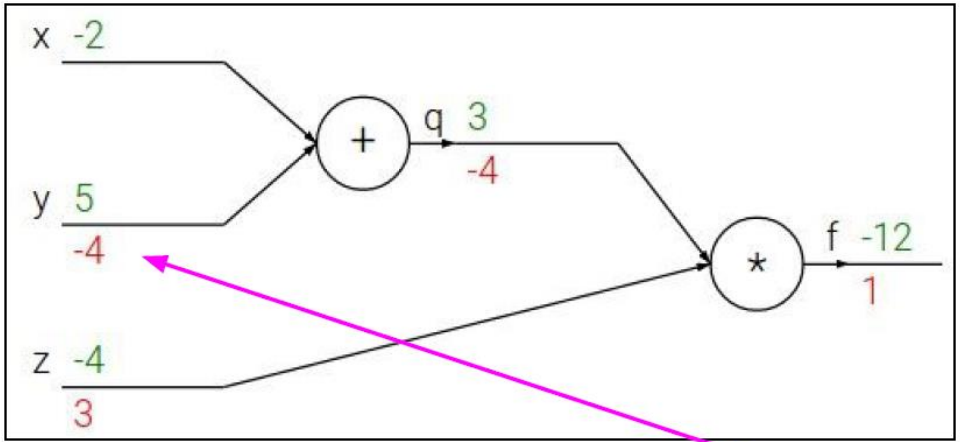
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$

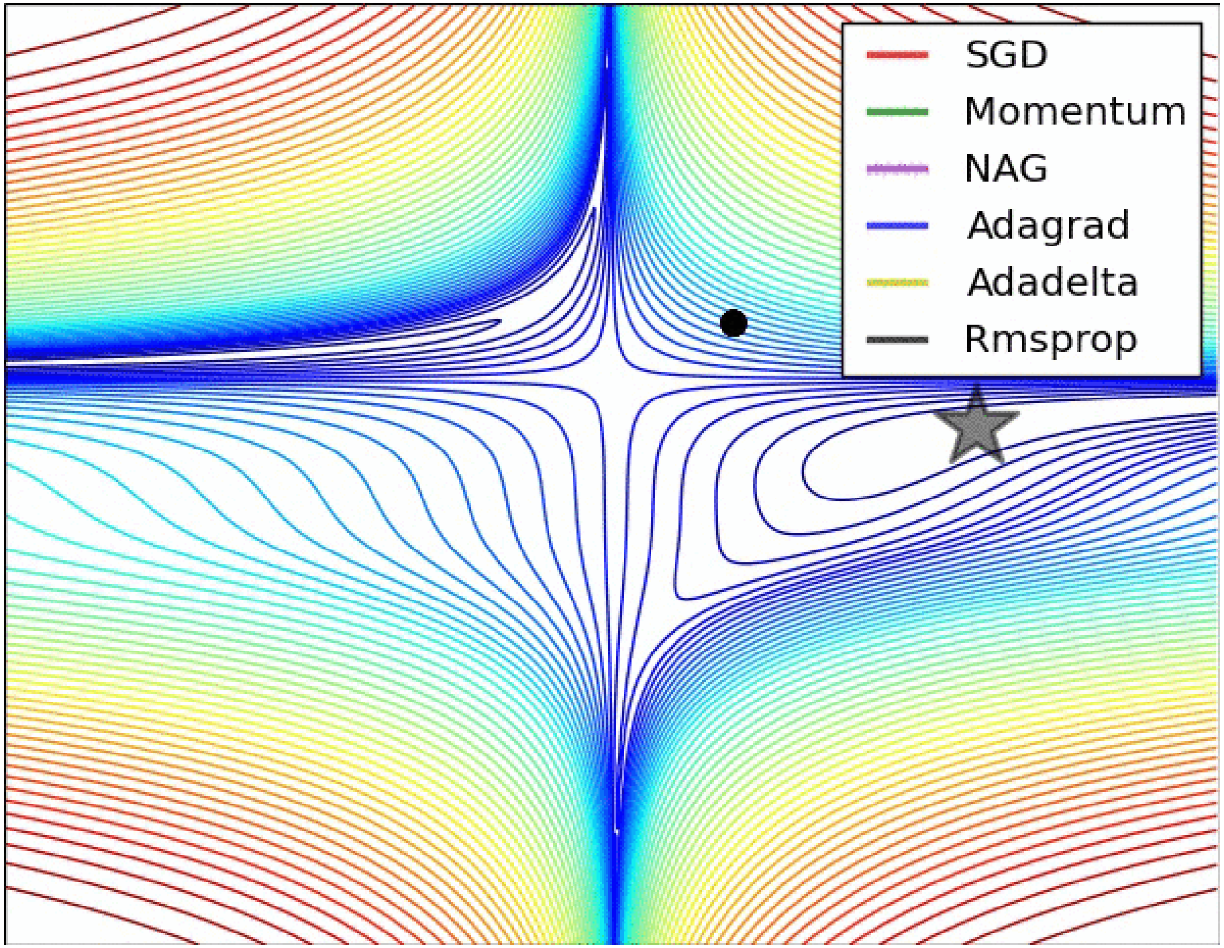


$\frac{\partial f}{\partial y}$

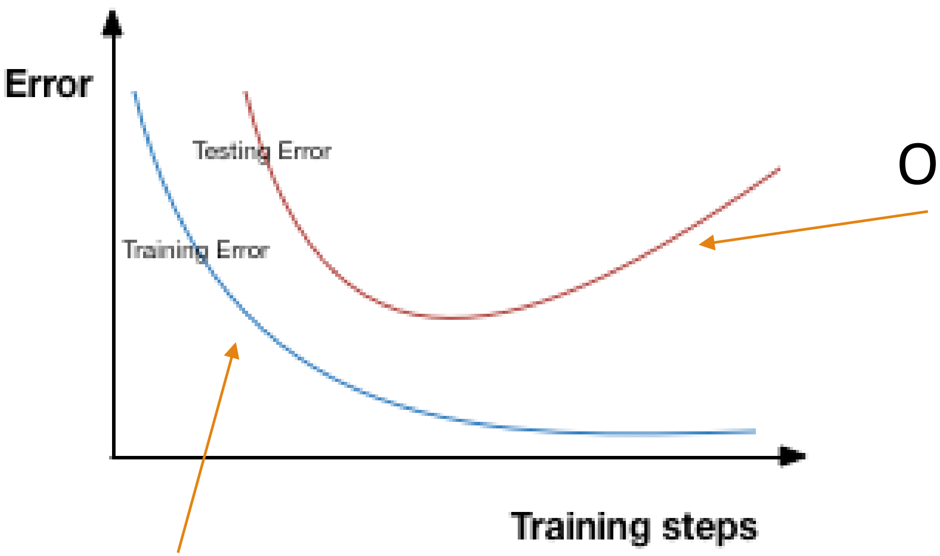
Chain rule:

$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial y}$$

Visualization



Training Characteristics



Over-fitting

Under-fitting

Supervised Learning

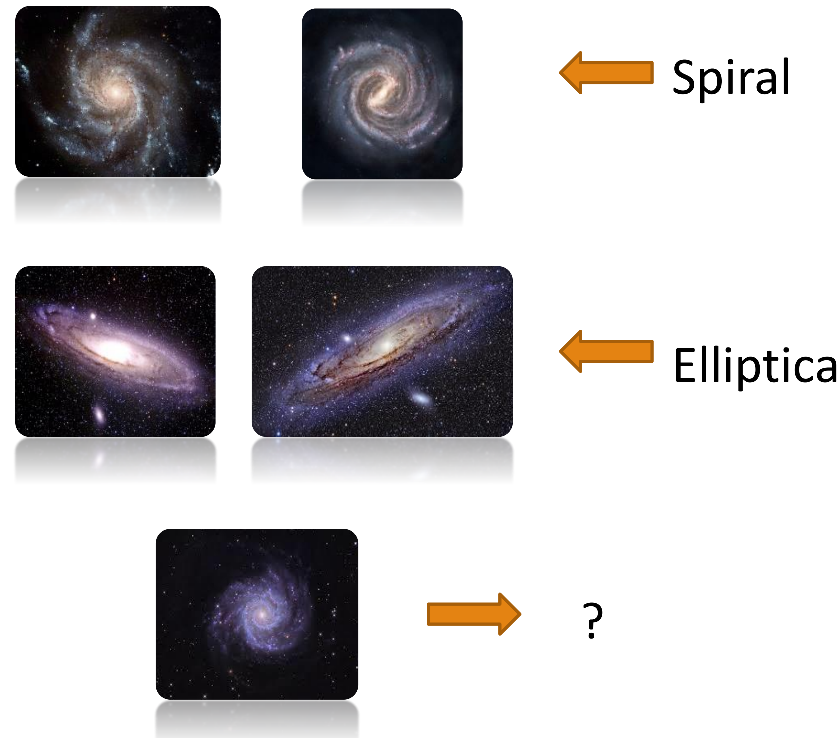
Supervised Learning

Data
Labels

Model
Prediction

Exploiting prior knowledge

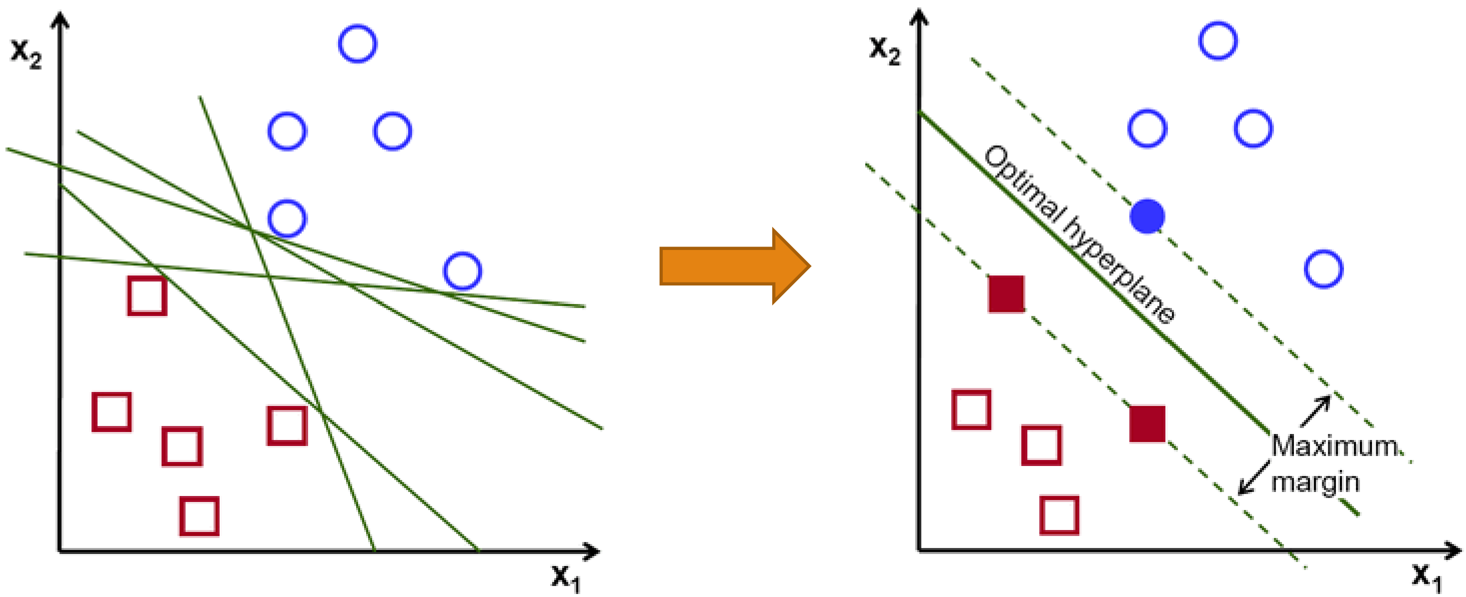
- Expert users
- Crowdsourcing
- Other instruments



State-of-the-art (before Deep Learning)

Support Vector Machines

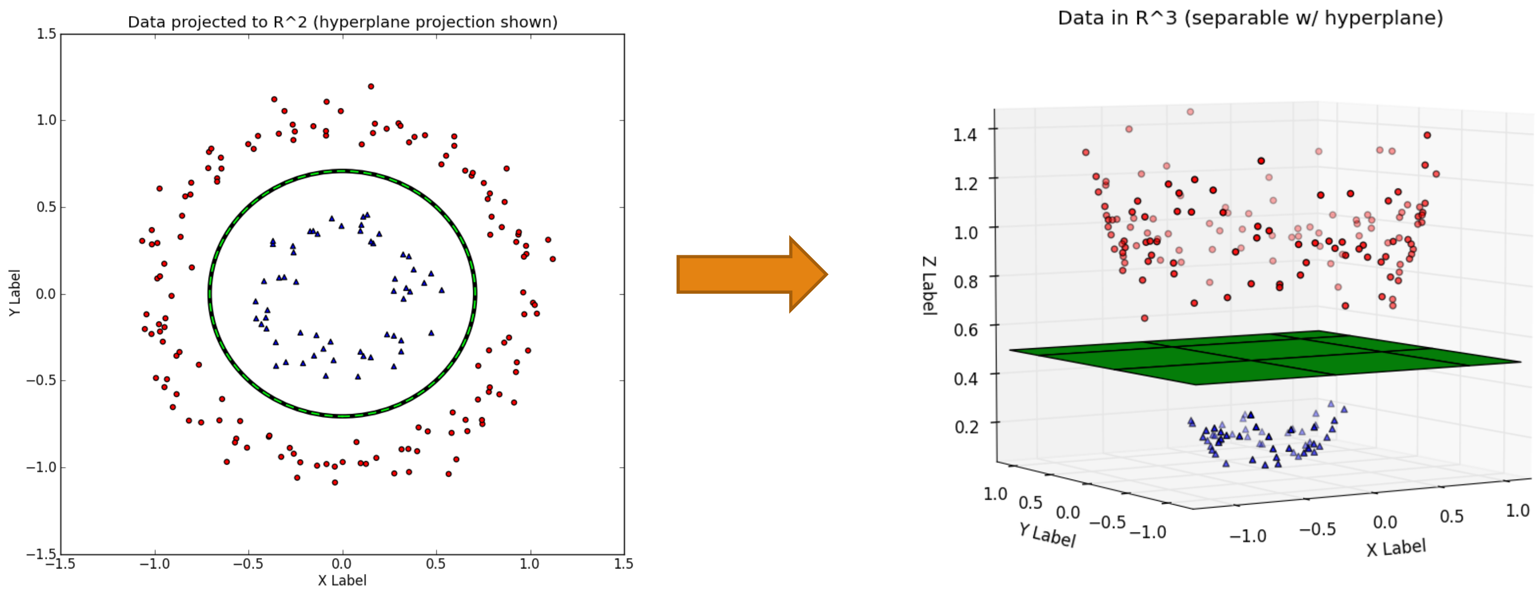
- Binary classification



State-of-the-art (before Deep Learning)

Support Vector Machines

- Binary classification
- Kernels <-> non-linearities



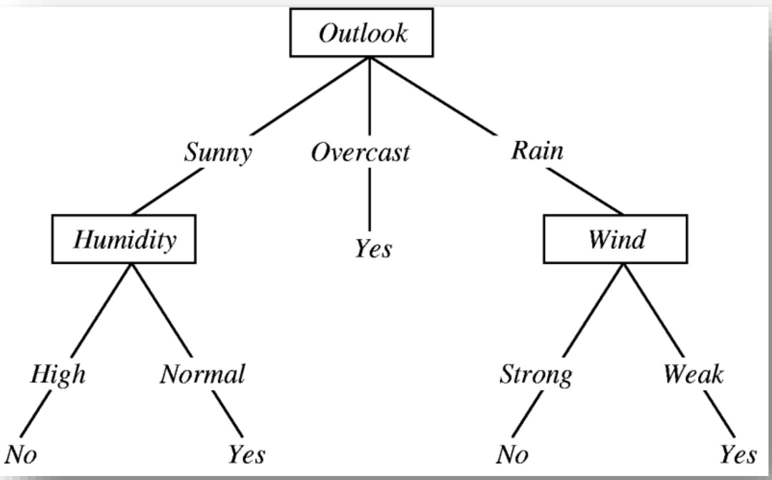
State-of-the-art (before Deep Learning)

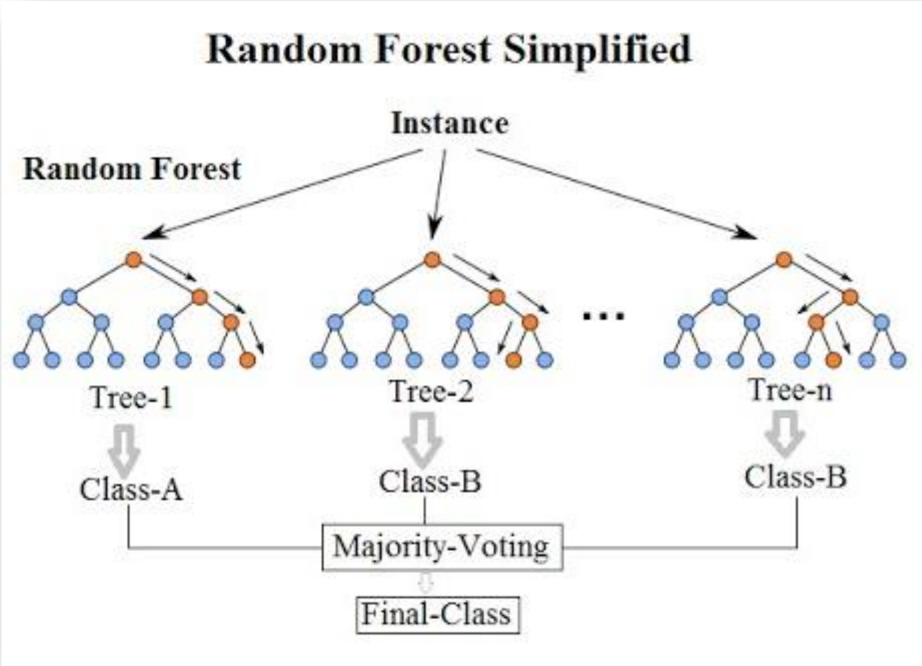
Support Vector Machines

- Binary classification
- Kernels <-> non-linearities

Random Forests

- Multi-class classification





State-of-the-art (before Deep Learning)

Support Vector Machines

- Binary classification
- Kernels <-> non-linearities

Random Forests

- Multi-class classification

Markov Chains/Fields

- Temporal data

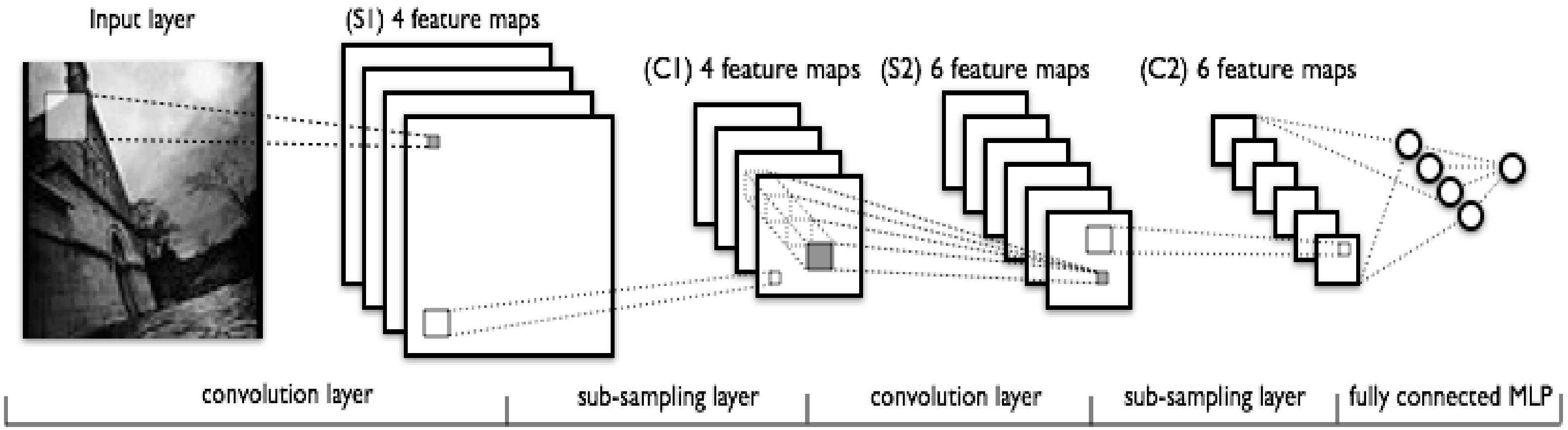
State-of-the-art (since 2015)

Deep Learning (DL)

Convolutional Neural Networks (CNN) <-> Images

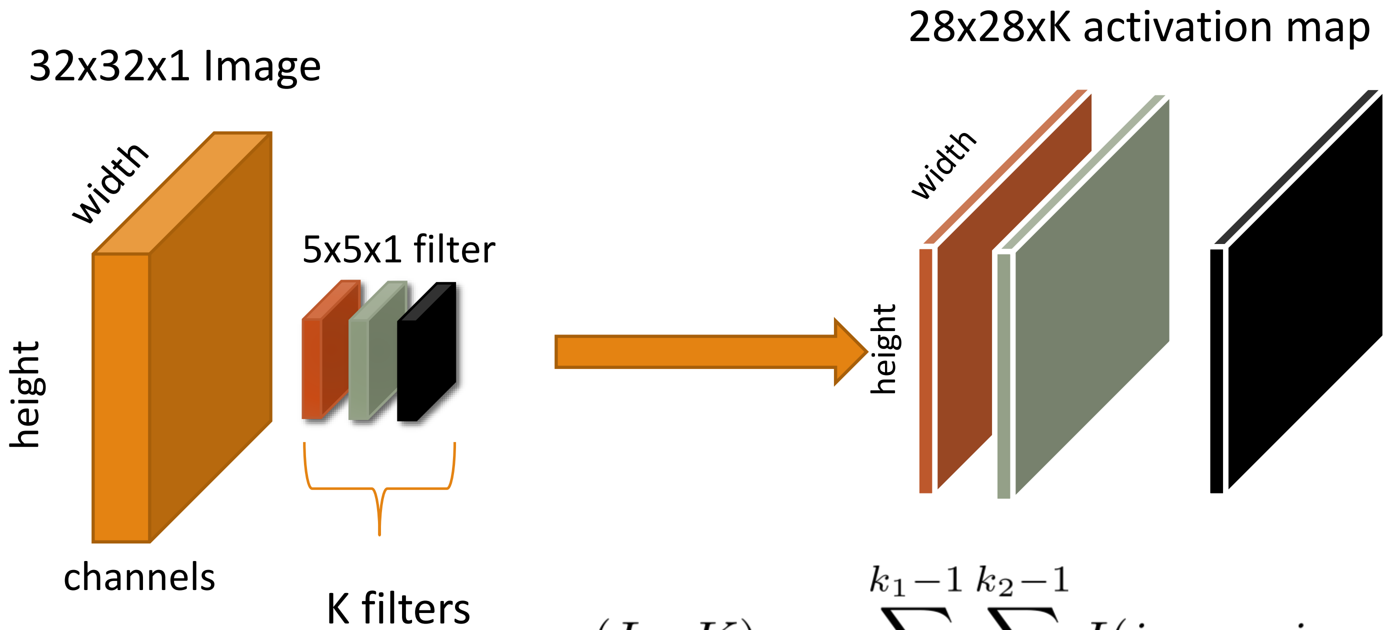
Recurrent Neural Networks (RNN) <-> Audio

Convolutional Neural Networks



(Convolution + Subsampling) + () … + Fully Connected

Convolutional Layers



$$(I * K)_{ij} = \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i-m, j-n) K(m,n)$$
$$= \sum_{m=0}^{k_1-1} \sum_{n=0}^{k_2-1} I(i+m, j+n) K(-m,-n)$$

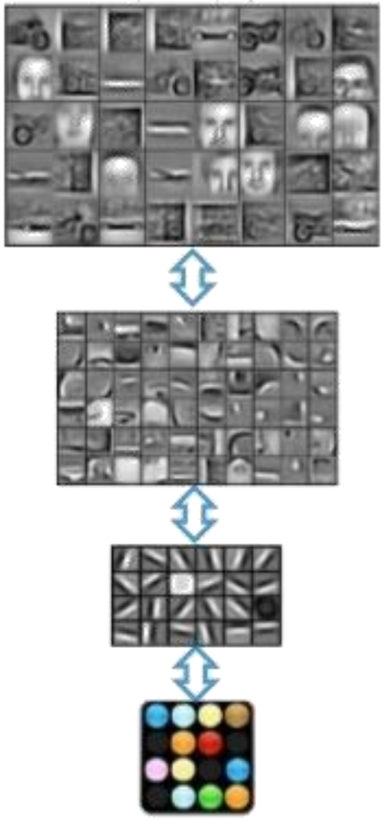
Convolutional Layers

Characteristics

- Hierarchical features
- Location invariance

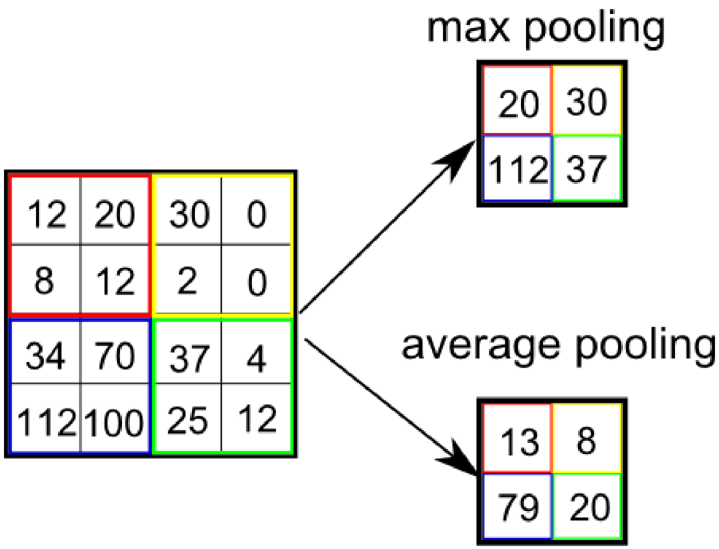
Parameters

- Number of filters (32,64…)
- Filter size (3x3, 5x5)
- Stride (1)
- Padding (2,4)



“Machine Learning and AI for Brain Simulations” – Andrew Ng Talk, UCLA, 2012

Subsampling (pooling) Layers



<-> downsampling

➢ Scale invariance

Parameters

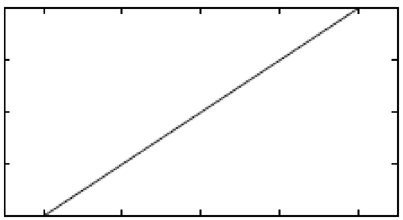
- Type
- Filter Size
- Stride

Activation Layer

Introduction of non-linearity

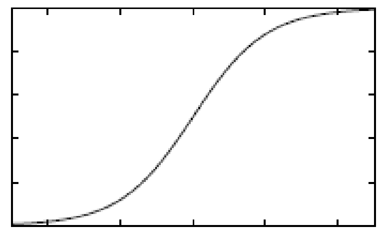
- Brain: thresholding -> spike trains

Identity (Linear)



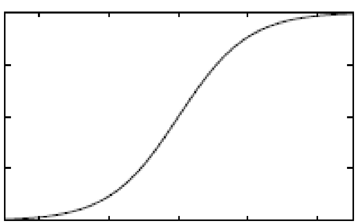
$$identity(x) = x$$

Sigmoid



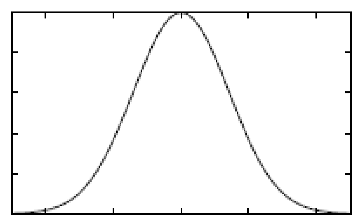
$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

Tanh (Hypertangent)



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

Gaussian



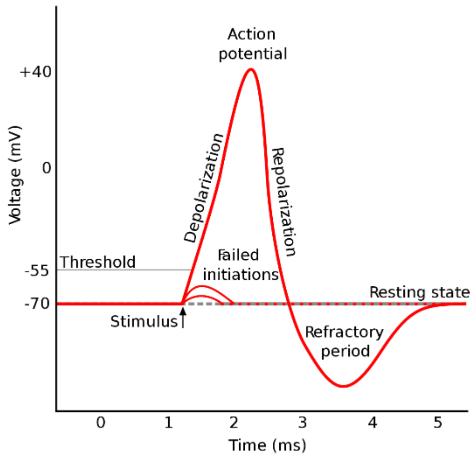
$$gaussian(x) = e^{-x^2/\sigma^2}$$

Activation Layer

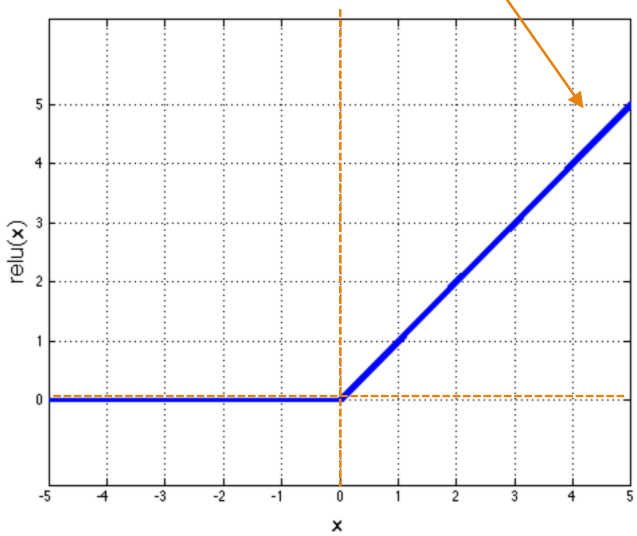
ReLU: x=max(0,x)

- ✓ Simplifies backprop
- ✓ Makes learning faster
- ✓ Avoids saturation issues
- ✓ ~ non-negativity constraint

(Note: The brain)



No saturated gradients

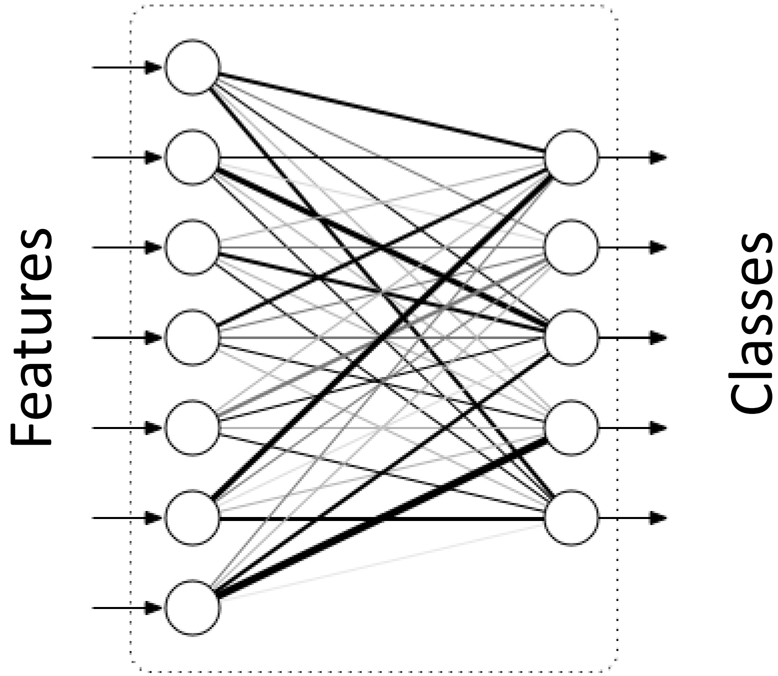


Fully Connected Layers

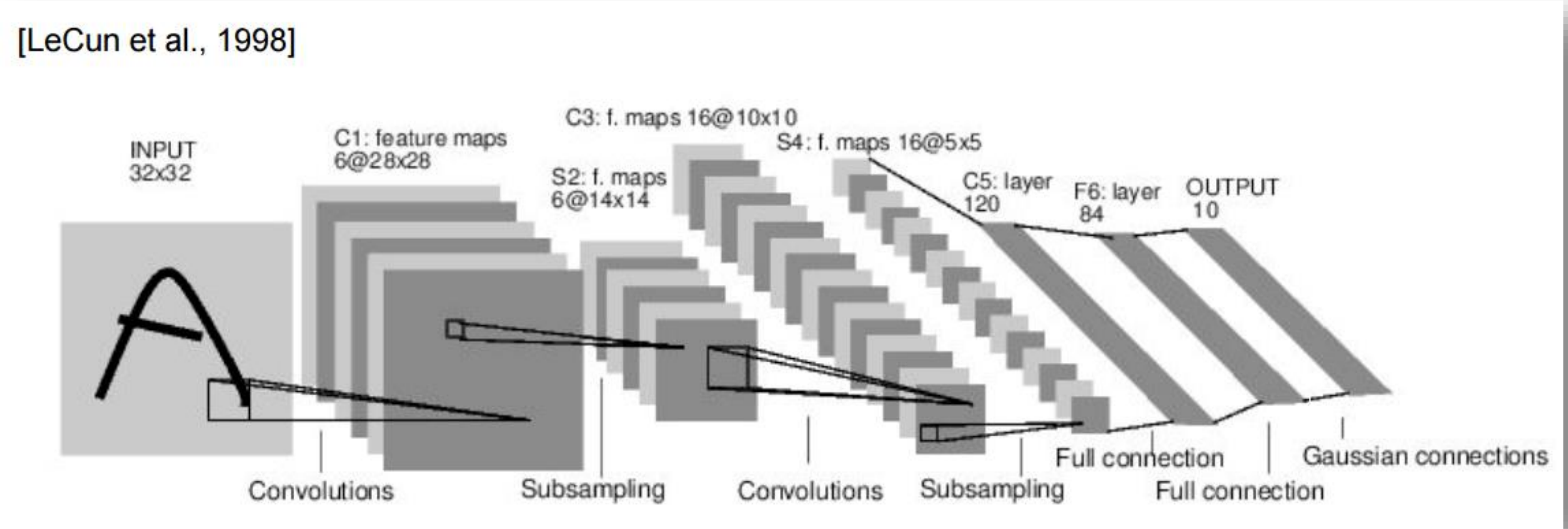
Full connections to all activations in previous layer

Typically at the end

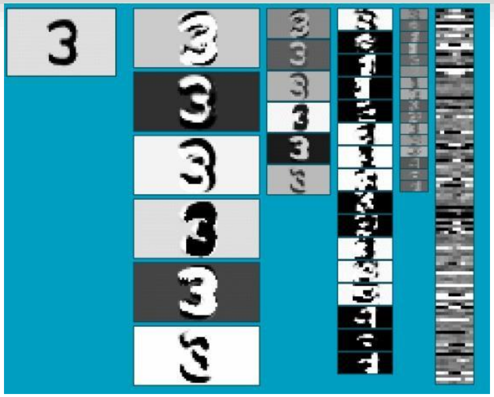
Can be replaced by conv



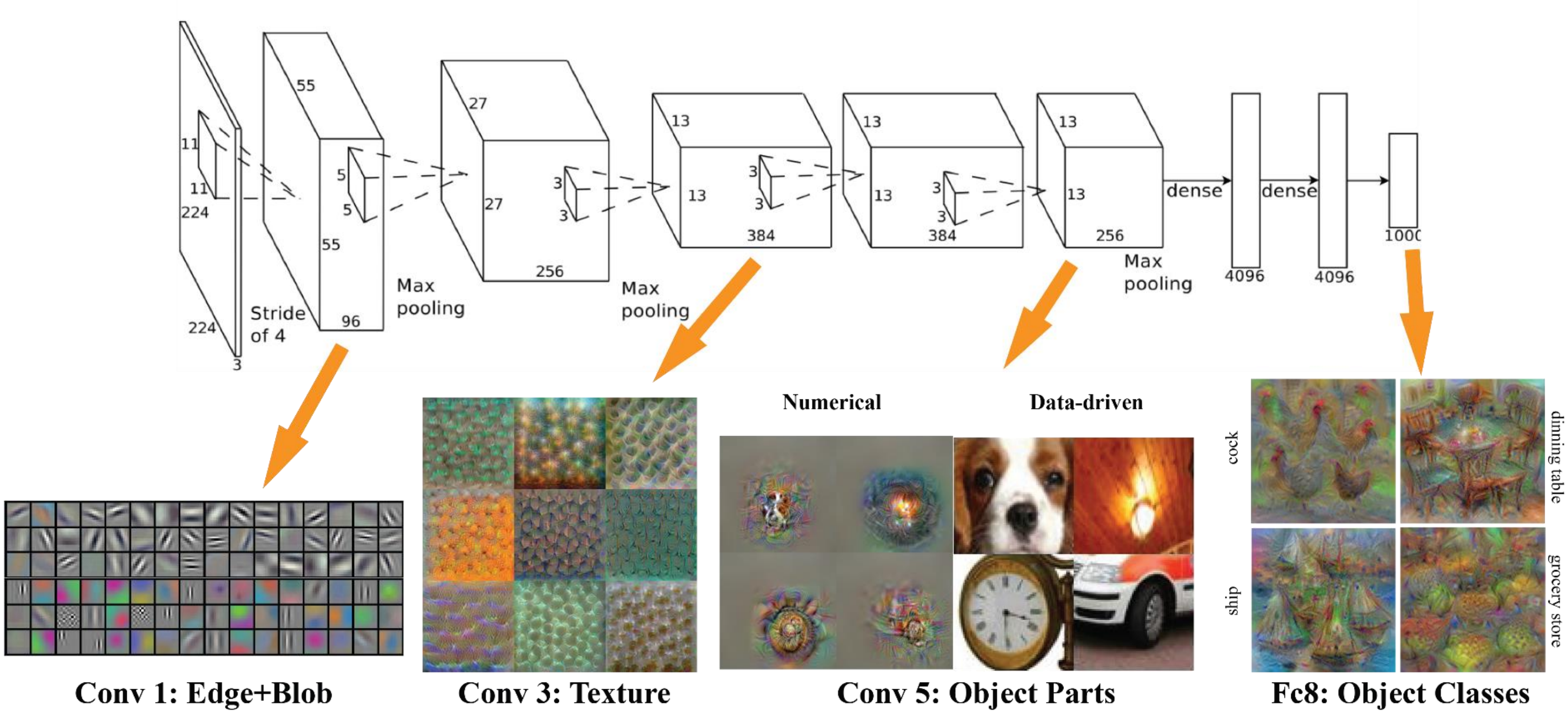
LeNet [1998]



[LeCun et al., 1998]
INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

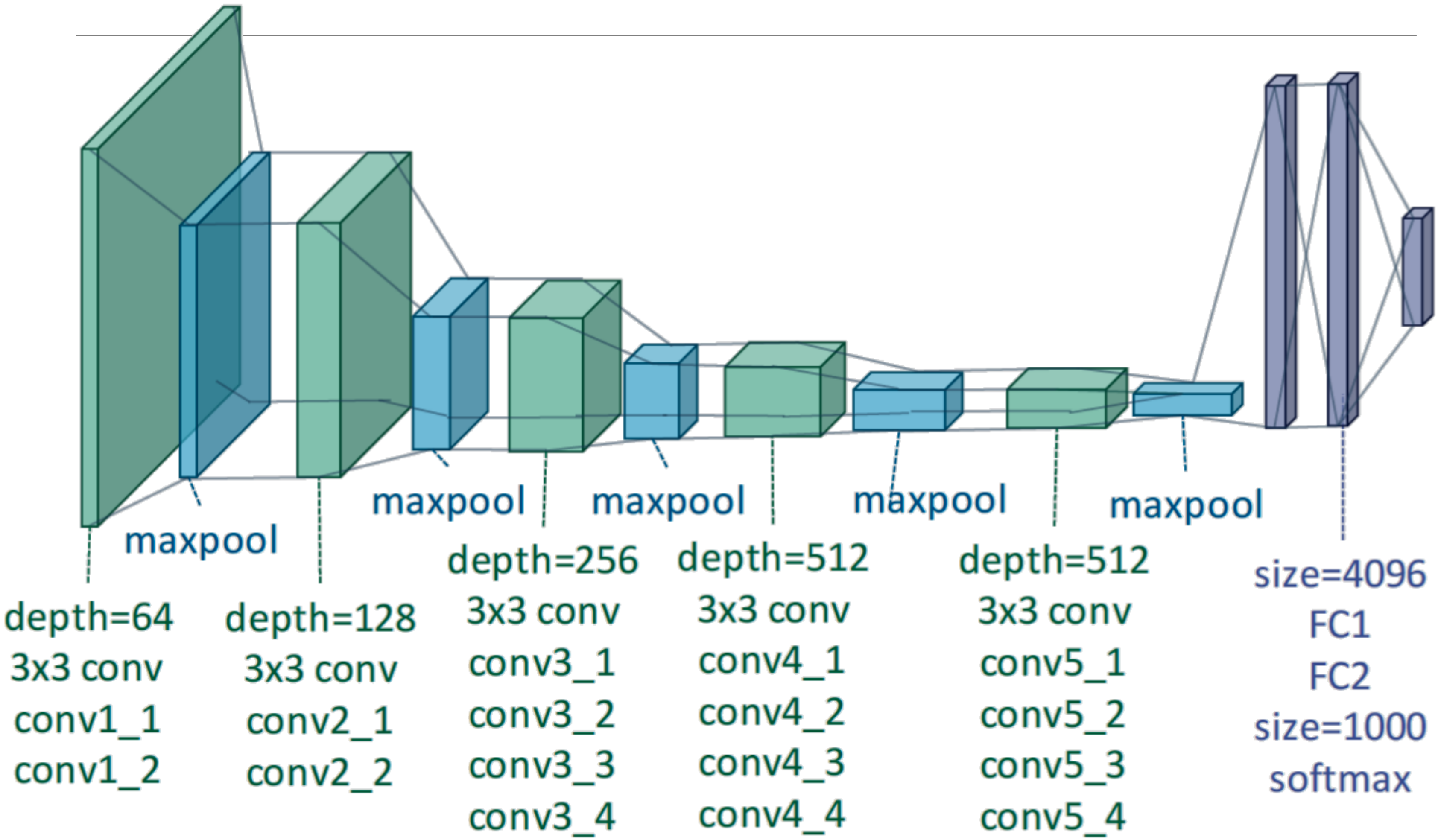


AlexNet [2012]



Alex Krizhevsky, Ilya Sutskever and Geoff Hinton, ImageNet ILSVRC challenge in 2012
http://vision03.csail.mit.edu/cnn_art/data/single_layer.png

VGGnet [2014]



K. Simonyan, A. Zisserman Very Deep Convolutional Networks for Large-Scale Image Recognition, arXiv technical report, 2014

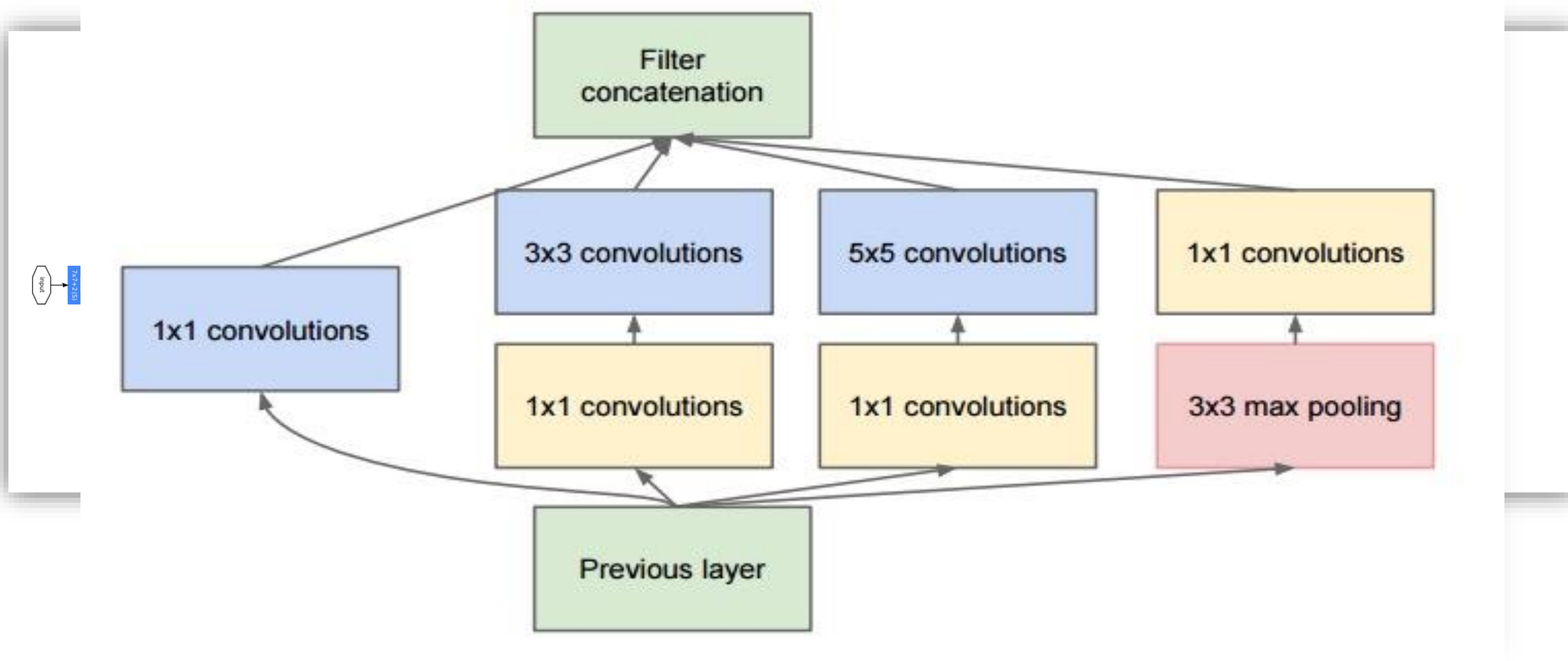
VGGnet

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					

D: VGG16
E: VGG19
All filters are 3x3

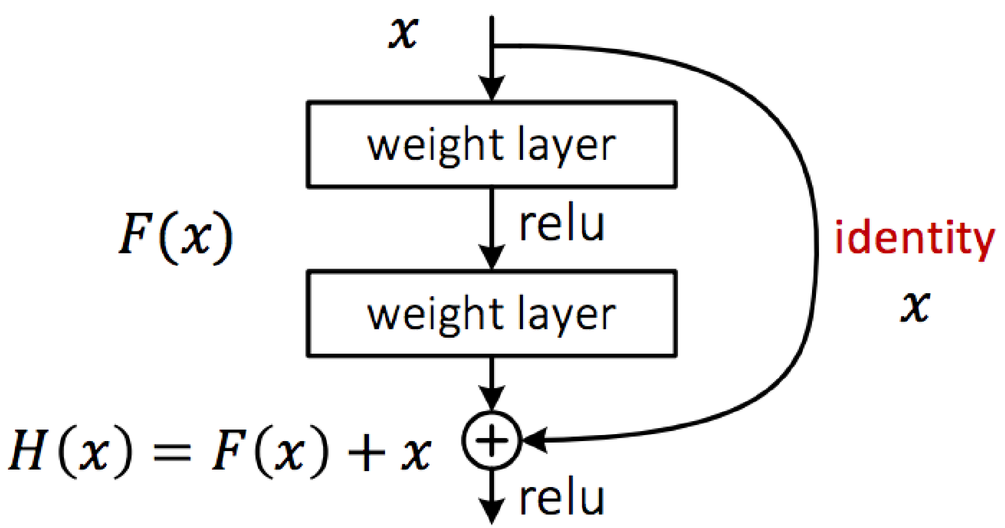
More layers
smaller filters

Inception (GoogLeNet, 2014)

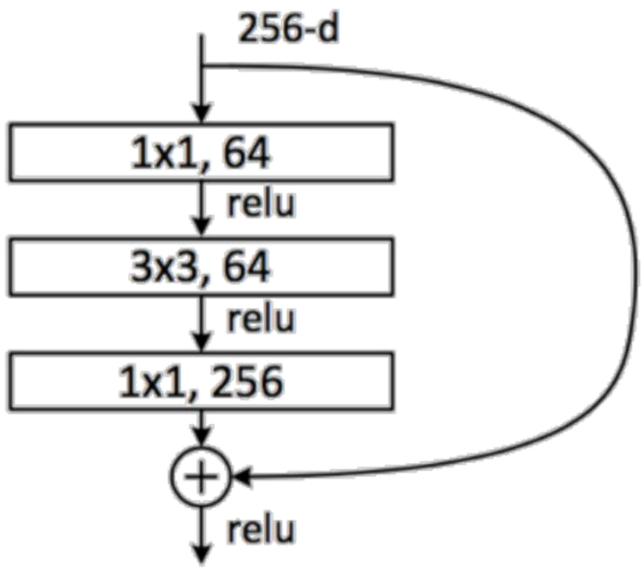


Inception module with dimensionality reduction

Residuals

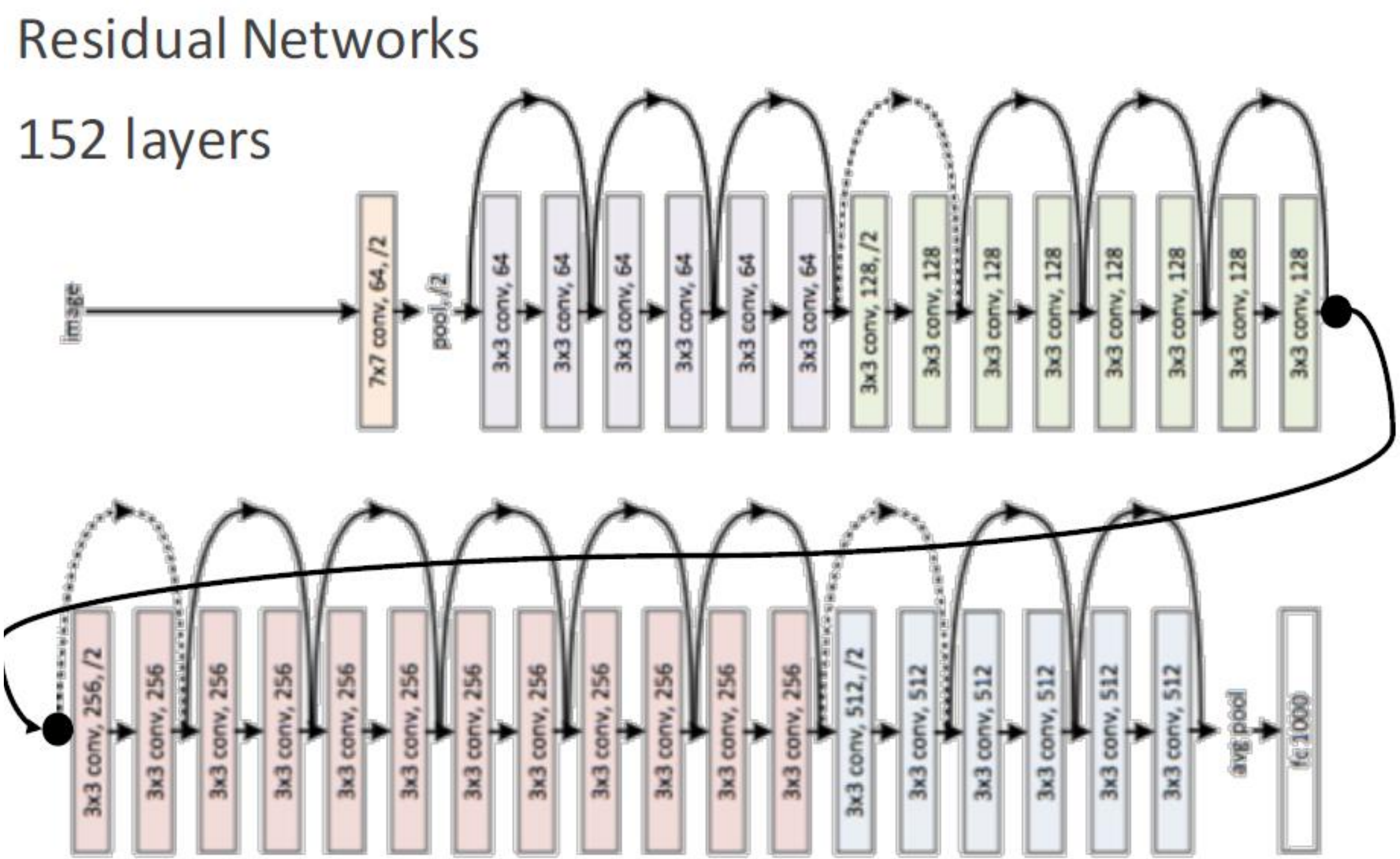


x
weight layer
$F(x)$
relu
weight layer
identity
x
$H(x) = F(x) + x$
relu



256-d
1x1, 64
relu
3x3, 64
relu
1x1, 256
relu

ResNet, 2015



He, Kaiming, et al. "Deep residual learning for image recognition." *IEEE CVPR*. 2016.

Training protocols

Fully Supervised

- Random initialization of weights
- Train in supervised mode (example + label)

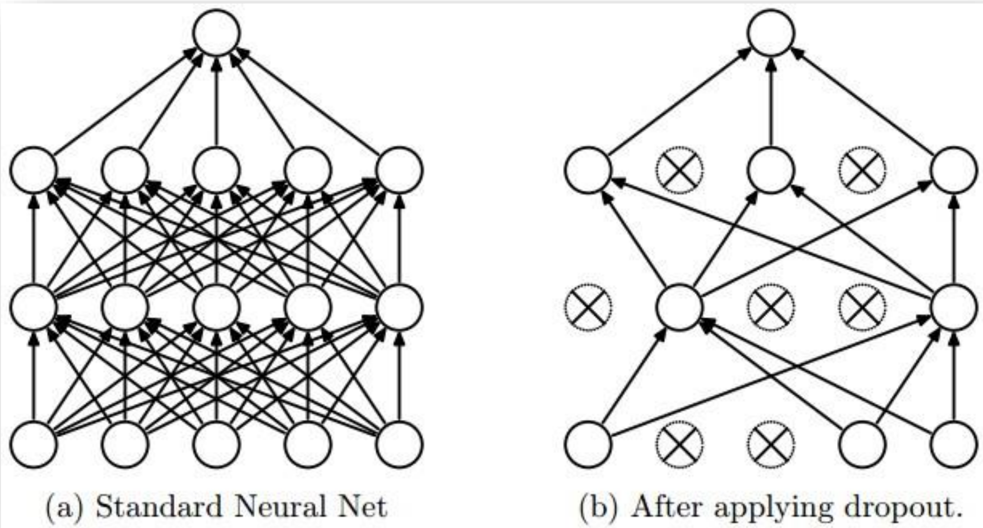
Unsupervised pre-training + standard classifier

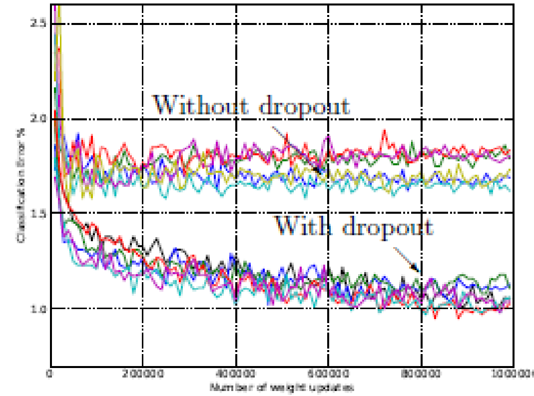
- Train each layer unsupervised
- Train a supervised classifier (SVM) on top

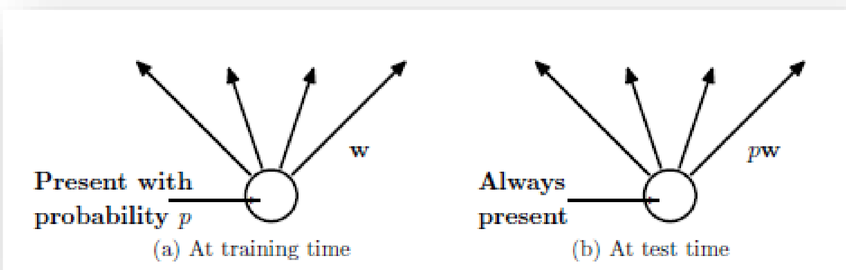
Unsupervised pre-training + supervised fine-tuning

- Train each layer unsupervised
- Add a supervised layer

Dropout







Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." *Journal of machine learning research*15.1 (2014): 1929-1958.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

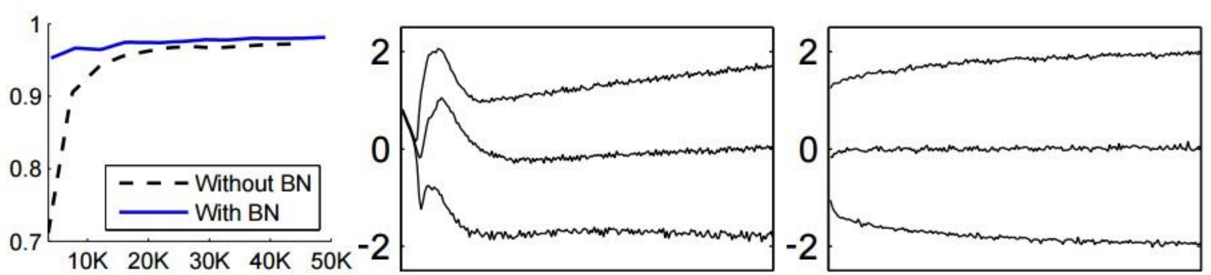
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

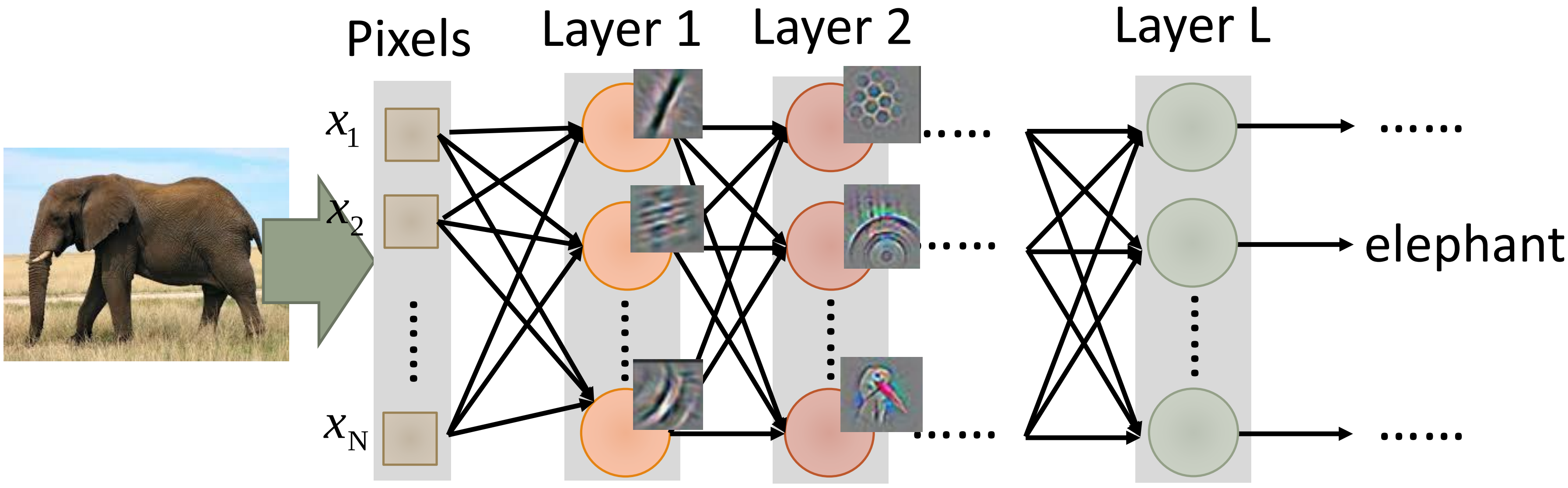
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

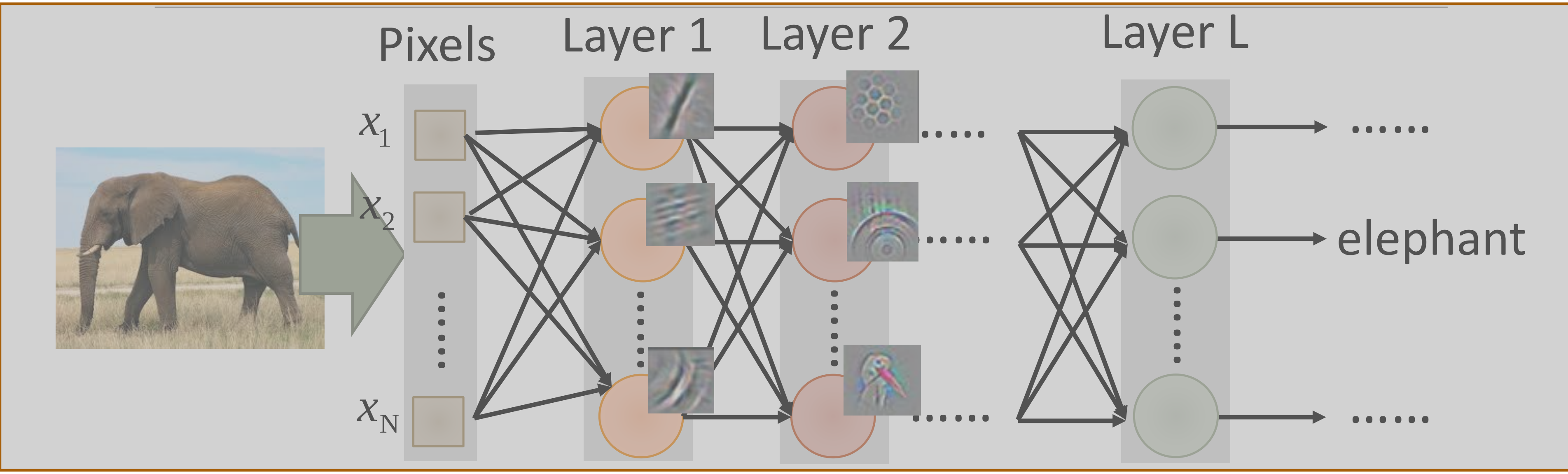


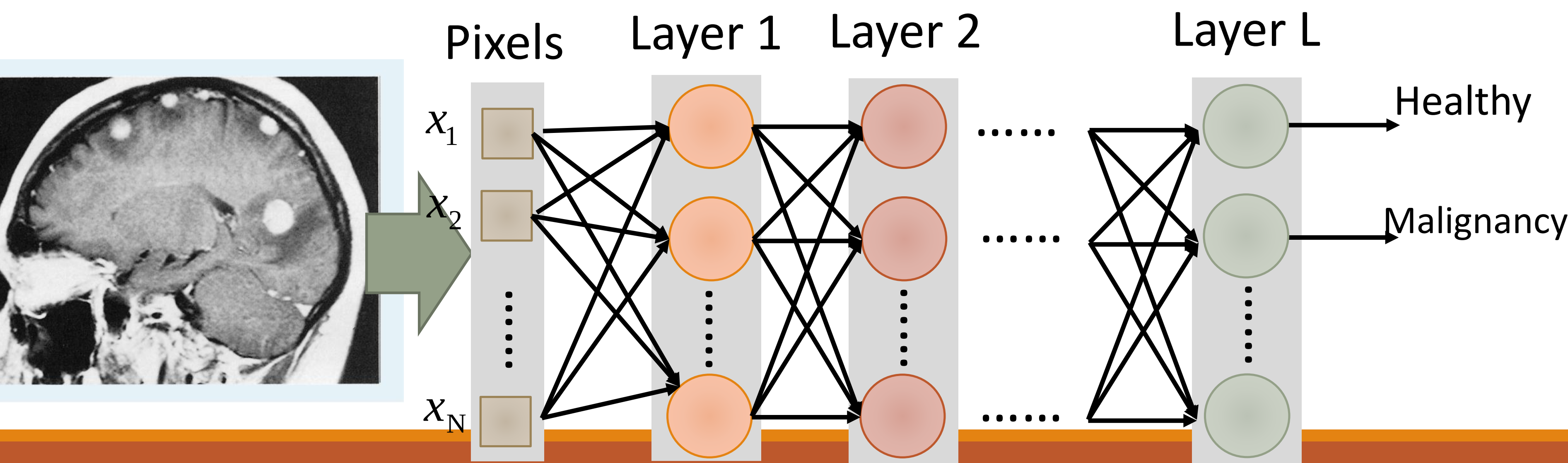
(a) (b) Without BN (c) With BN

Transfer Learning

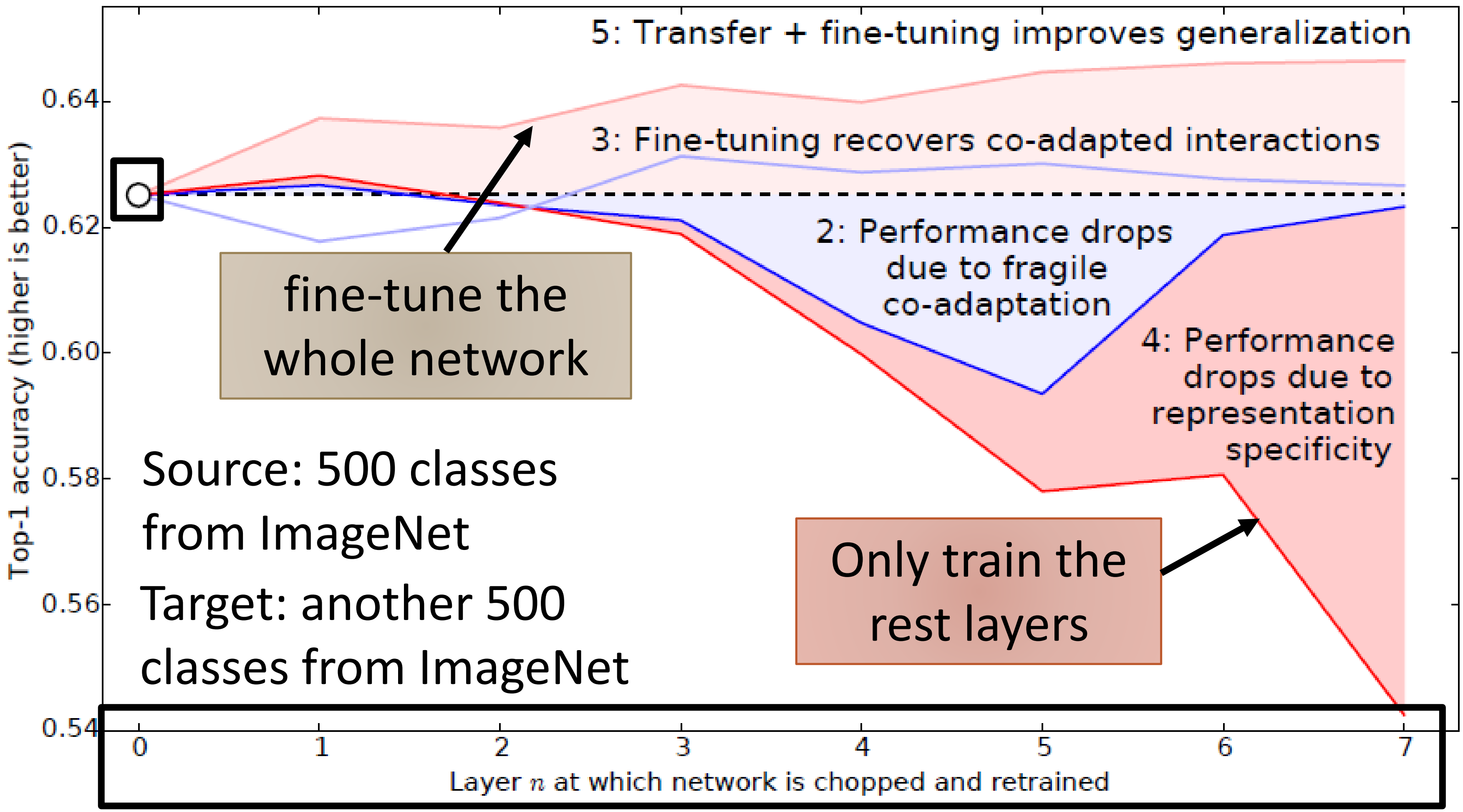


Transfer Learning



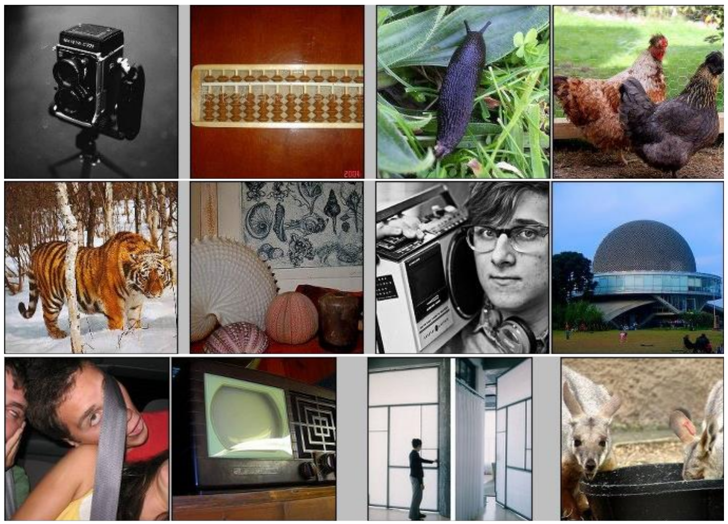


Layer Transfer - Image



Jason Yosinski, Jeff Clune, Yoshua Bengio, Hod Lipson, "How transferable are features in deep neural networks?", NIPS, 2014

ImageNET



- ~14 million labeled images, 20k classes
- Images gathered from Internet
- Human labels via Amazon MTurk
- ImageNet Large-Scale Visual Recognition Challenge (ILSVRC):
  1.2 million training images, 1000 classes

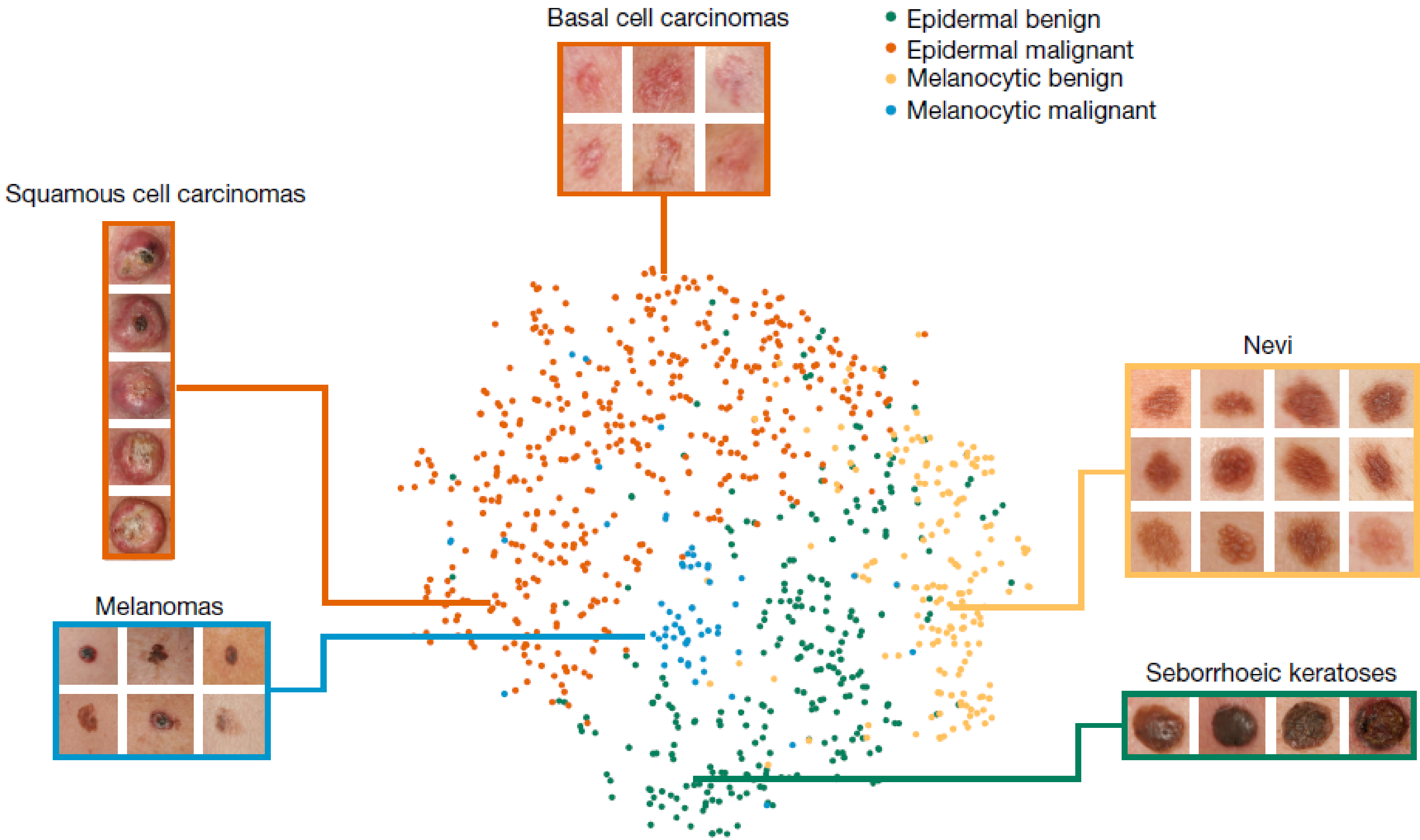
www.image-net.org/challenges/LSVRC/

Summary: ILSVRC 2012-2015

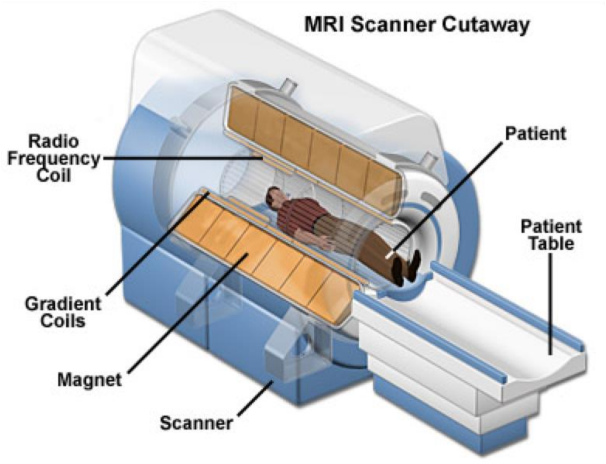
Team	Year	Place	Error (top-5)	External data
(AlexNet, 7 layers)	2012	-	16.4%	no
SuperVision	2012	1st	15.3%	ImageNet 22k
Clarifai – NYU (7 layers)	2013	-	11.7%	no
Clarifai	2013	1st	11.2%	ImageNet 22k
VGG – Oxford (16 layers)	2014	2nd	7.32%	no
GoogLeNet (19 layers)	2014	1st	6.67%	no
ResNet (152 layers)	2015	1st	3.57%	
Human expert*			5.1%	

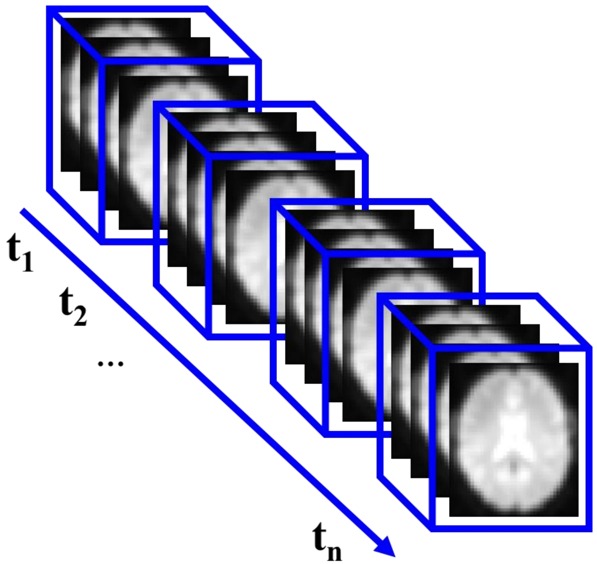
http://karpathy.github.io/2014/09/02/what-i-learned-from-competing-against-a-convnet-on-imagenet/

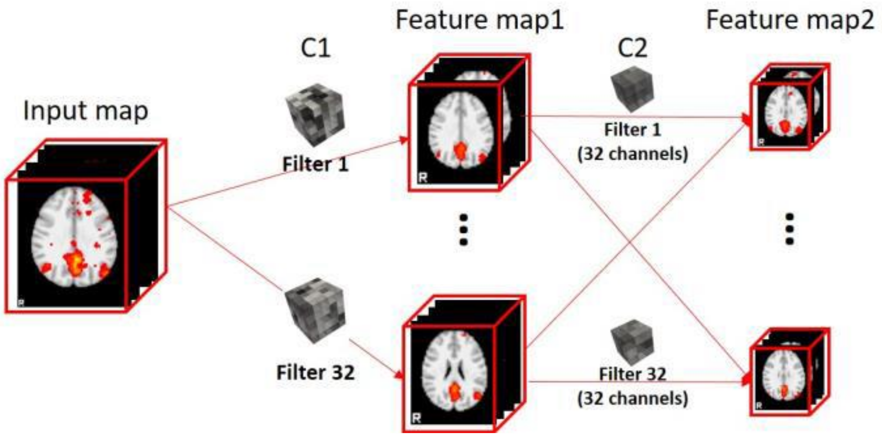
Skin cancer detection



CNN & FMRI


MRI Scanner Cutaway
Radio Frequency Coil
Patient
Patient Table
Gradient Coils
Magnet
Scanner


t_1
t_2
...
t_n


Input map
C1
Filter 1
Filter 32
Feature map1
C2
Filter 1
(32 channels)
Filter 32
(32 channels)
Feature map2

Different types of mapping

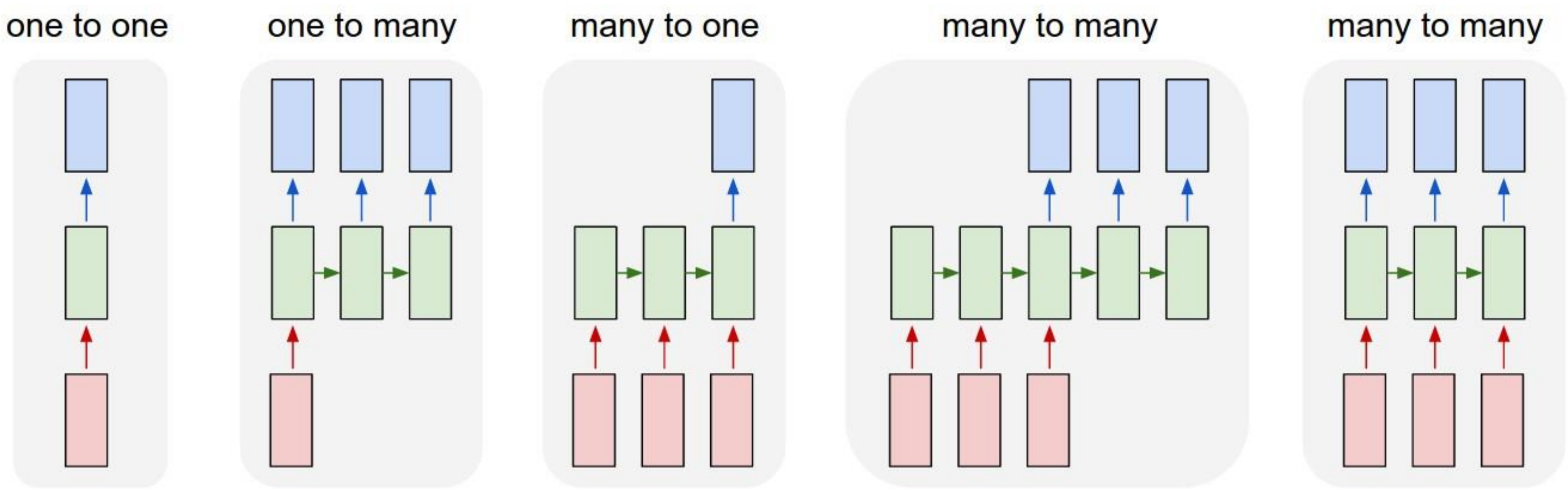


Image classification

Image captioning

Sentiment analysis

Machine translation

Synced sequence (video classification)

Recurrent Neural Networks

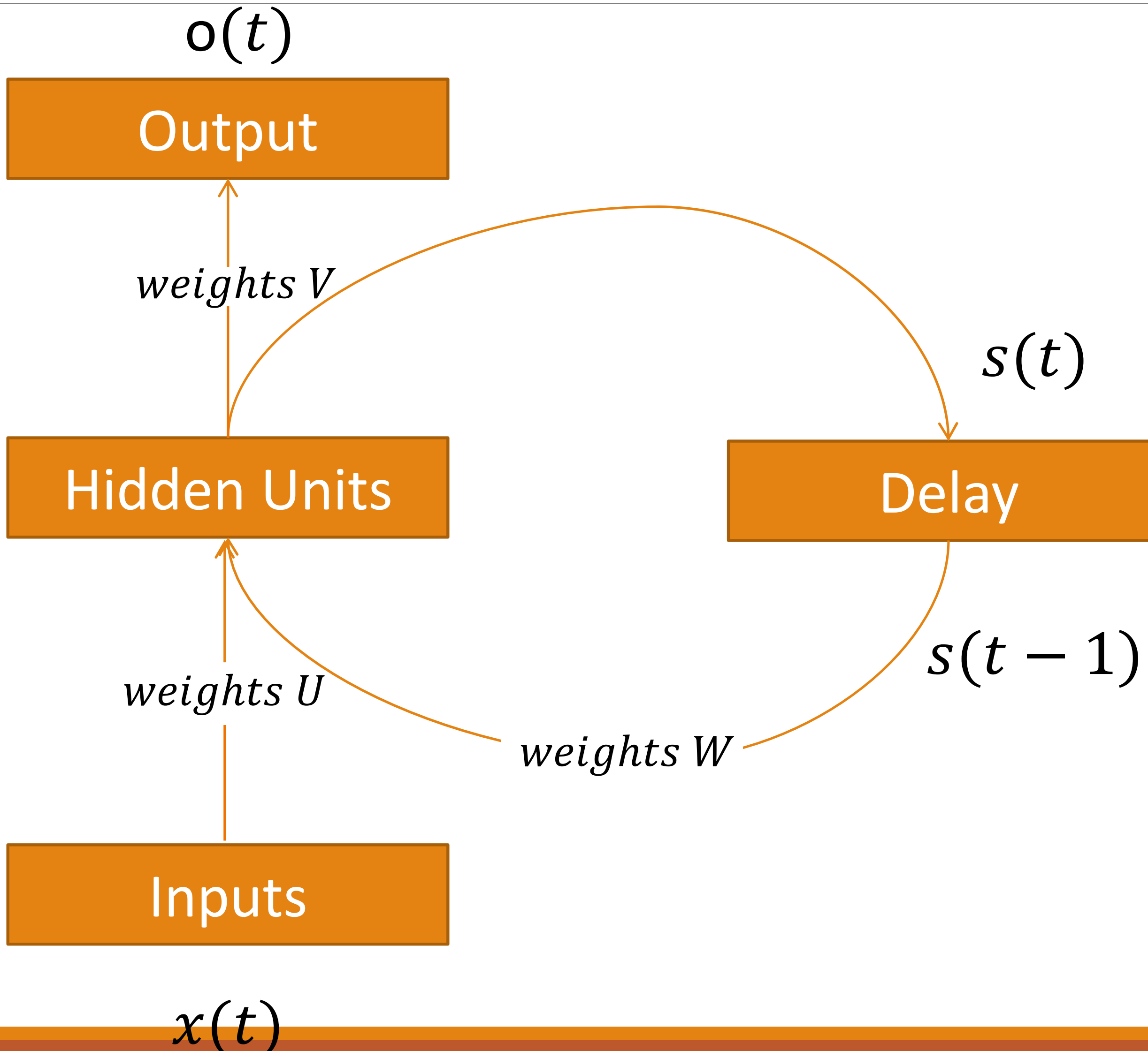
Motivation

- Feed forward networks accept a fixed-sized vector as input and produce a fixed-sized vector as output
- fixed amount of computational steps
- recurrent nets allow us to operate over *sequences* of vectors

Use cases

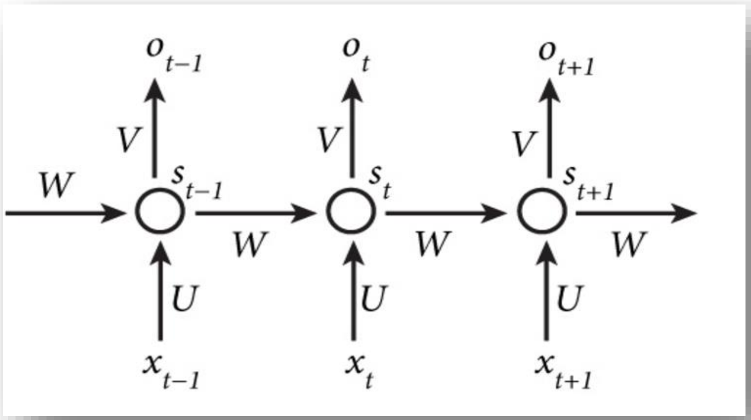
- Video
- Audio
- Text

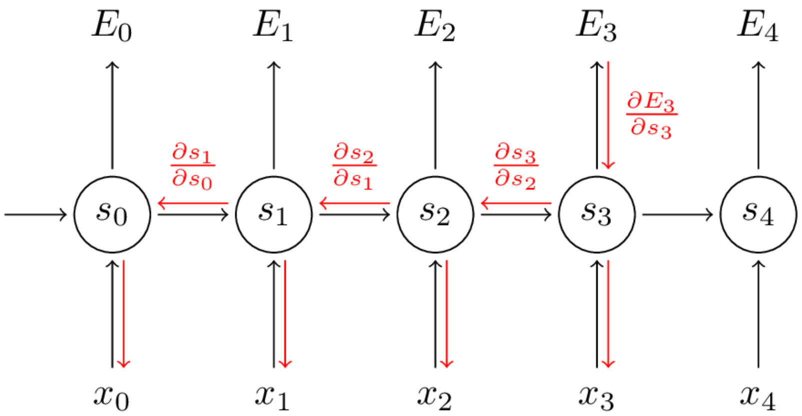
RNN Architecture



Unfolding RNNs

- Each node represents a layer of network units at a single time step.
- The same weights are reused at every time step.





Unsupervised Learning

Agenda

- Autoencoders
- Sparse coding
- Generative Adversarial Networks

Autoencoders

Unsupervised feature learning

Network is trained to output the input (learn identify function).

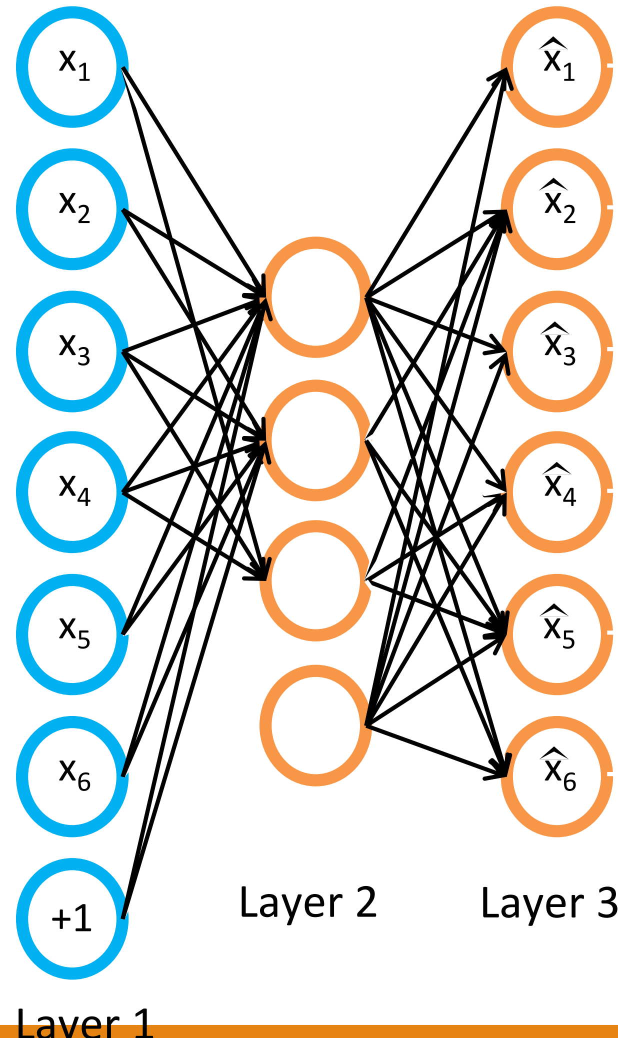
$$J = \frac{1}{m} \sum_{i=1}^{m} \|\hat{x} - x\|_2$$

Encoder

$$f(x) = \mathbf{h} = z(\mathbf{W}_1 \mathbf{x} + \mathbf{b}_1)$$

Decoder

$$g(f(x)) = \hat{\mathbf{x}} = z(\mathbf{W}_2 \mathbf{h} + \mathbf{b_2})$$



Regularized Autoencoders

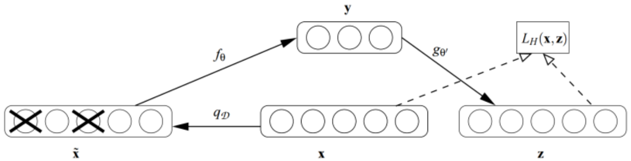
Sparse neuron activation

$$J_{sparse} = \sum \|\hat{\mathbf{x}} - \mathbf{x}\|_2 + \beta \sum KL(p, \hat{p})$$

Contractive auto-encoders

$$J_{contractive} = \sum \|\hat{\mathbf{x}} - \mathbf{x}\|_2 + \beta \| \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} \|_F$$

Denoising auto-encoders



Convolutional AE

$$f(x) = \mathbf{h} = z(\mathbf{W}_1 * \mathbf{x} + \mathbf{b}_1)$$

Stacked Autoencoders

Extended AE with multiple layers of hidden units

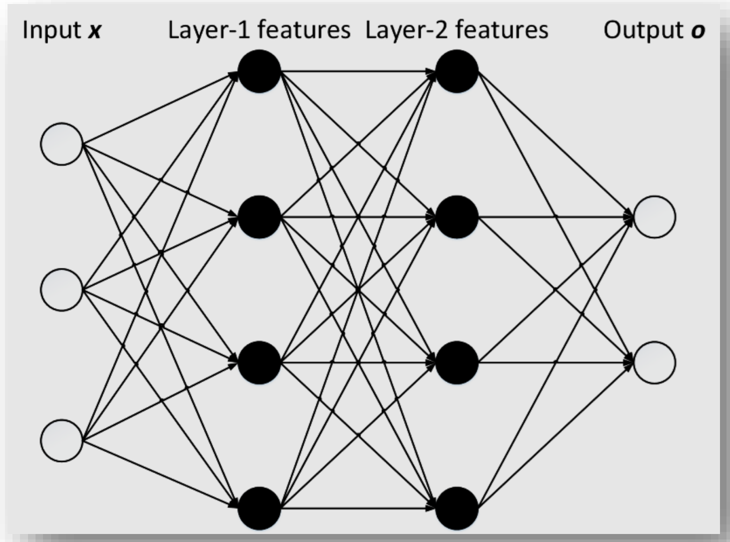
Challenges of Backpropagation

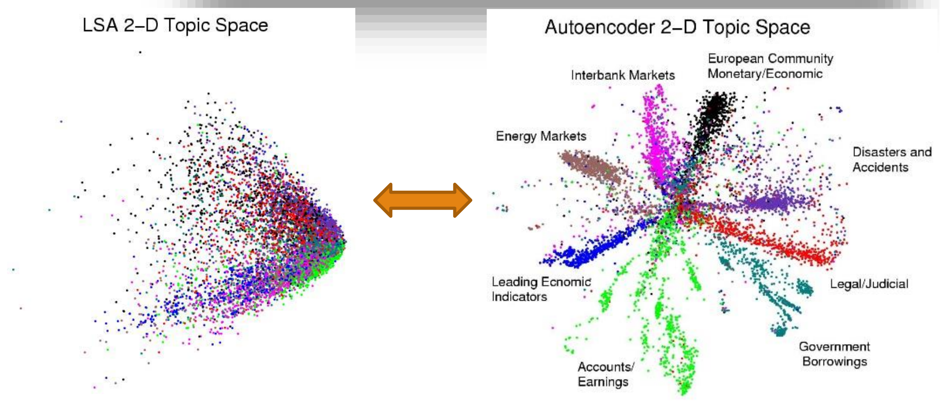
Efficient training

- Normalization of input

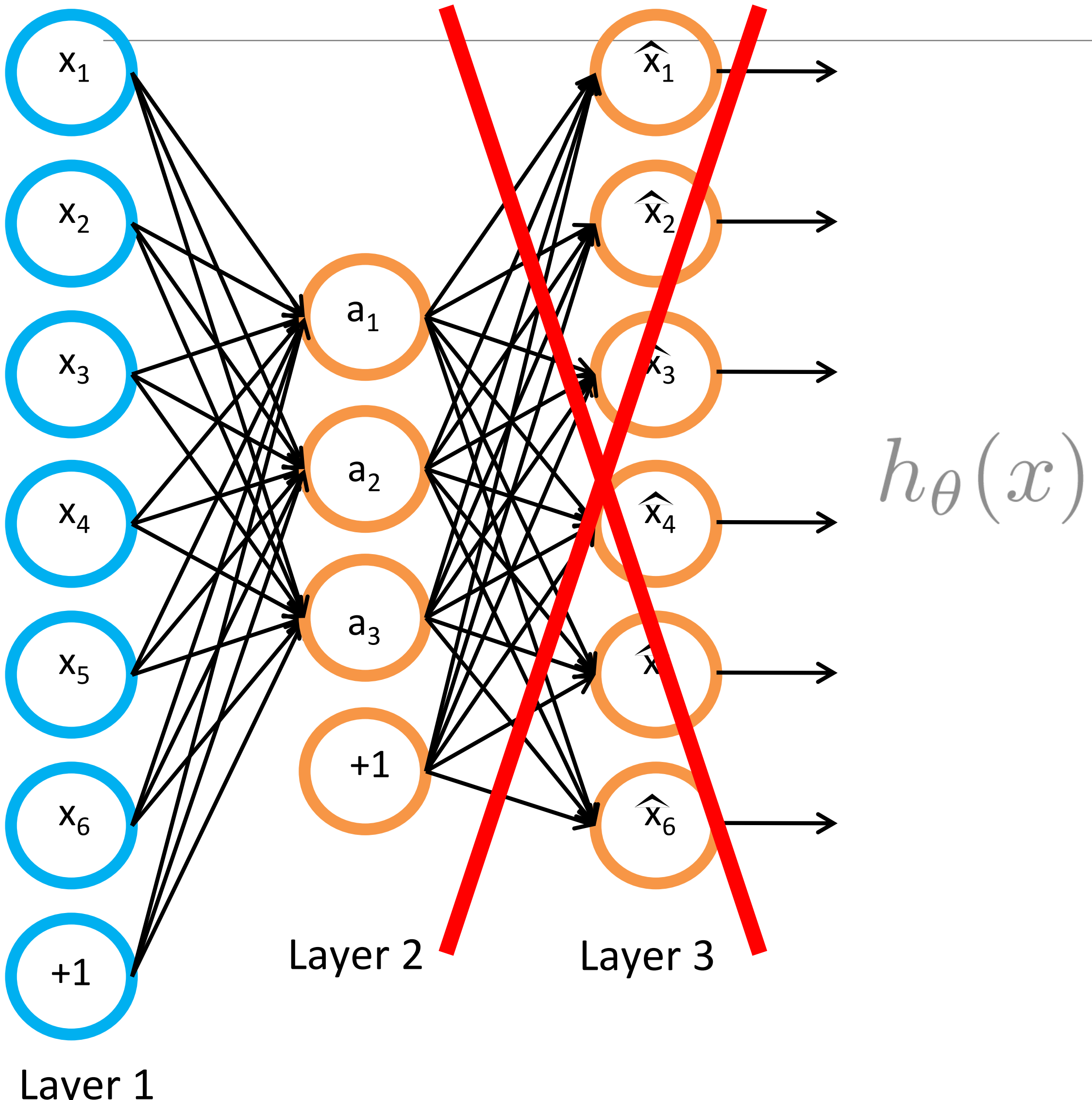
Unsupervised pre-training

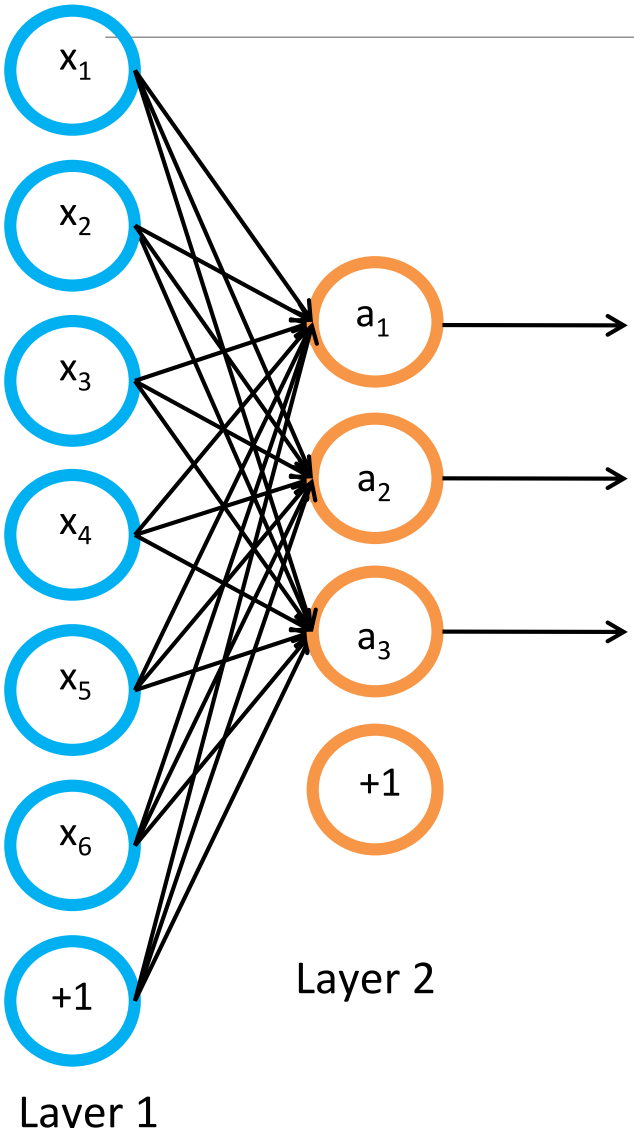
- Greedy layer-wise training
- Fine-tune w.r.t criterion





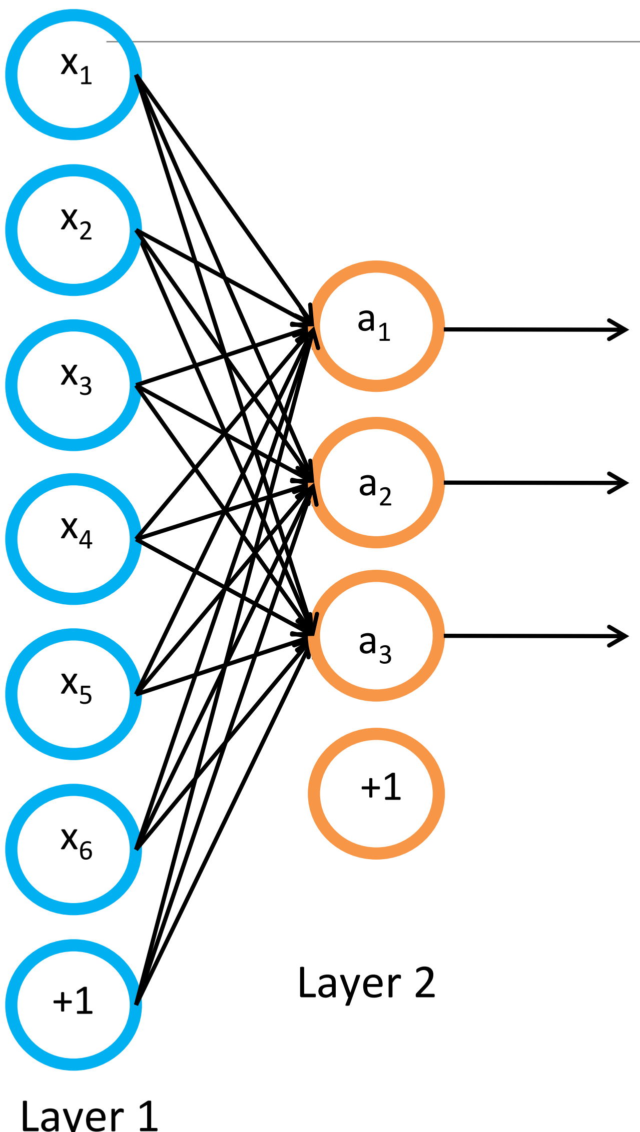
Bengio, Learning deep architectures for AI, Foundations and Trends in Machine Learning ,2009


x_1
x_2
x_3
x_4
x_5
x_6
+1
Layer 1
a_1
a_2
a_3
+1
Layer 2
$\hat{x}_1$
$\hat{x}_2$
$\hat{x}_3$
$\hat{x}_4$
$\hat{x}_6$
Layer 3
$h_\theta(x)$

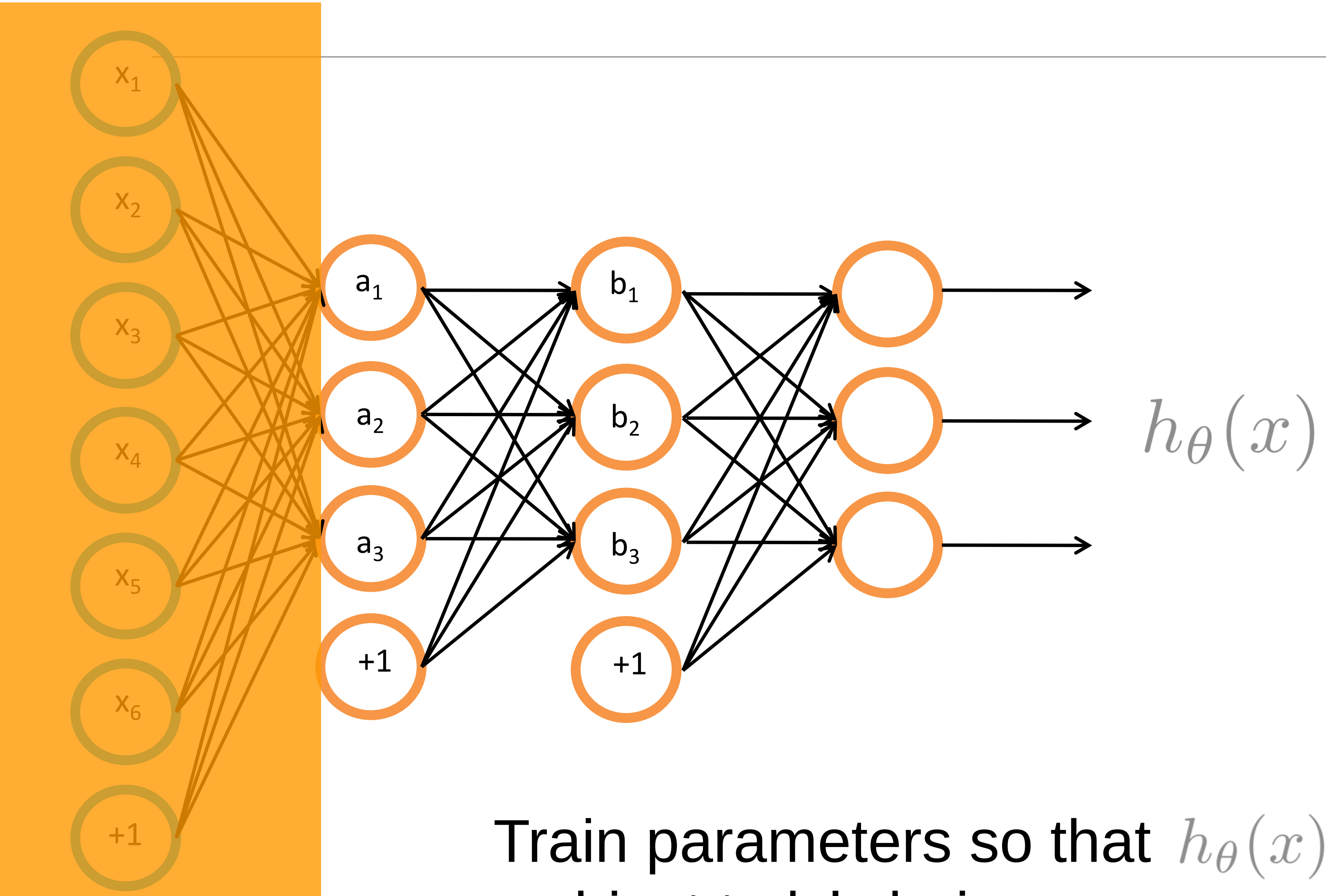


$$\begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

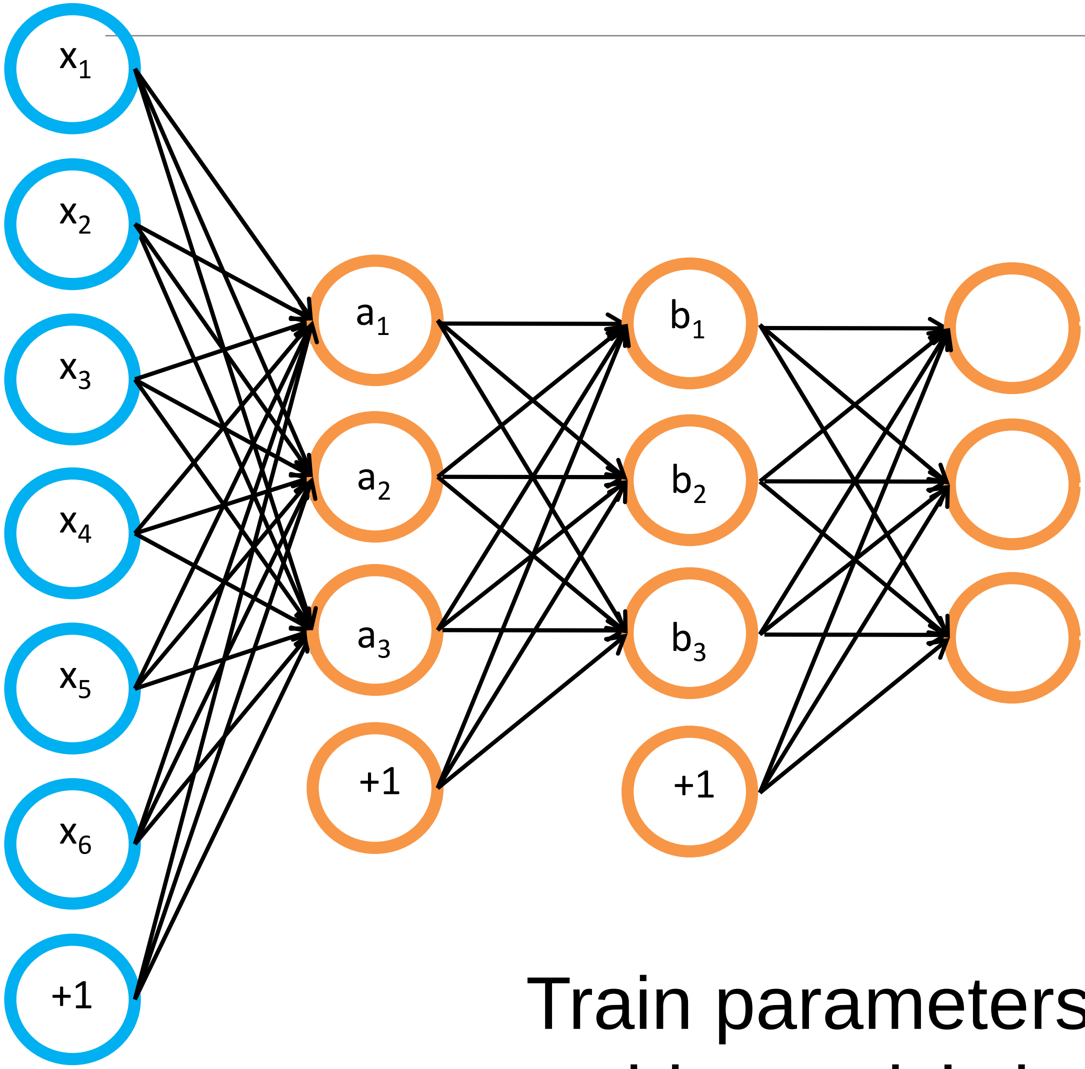
New representation for input.



x_1
x_2
x_3
x_4
x_5
x_6
+1
Layer 1
a_1
a_2
a_3
+1
Layer 2

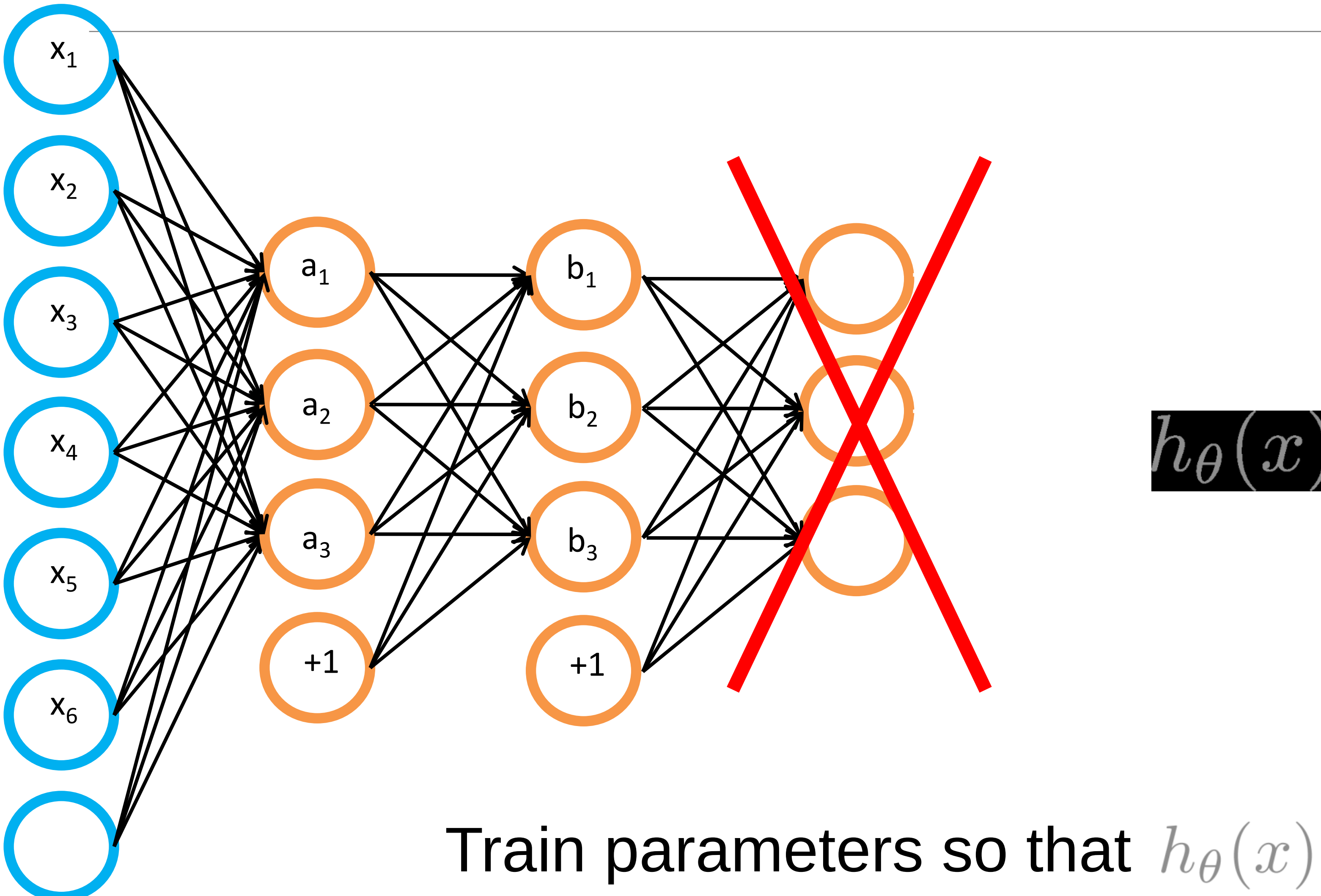


Train parameters so that $h_\theta(x) \approx a$, subject to b_i's being sparse.

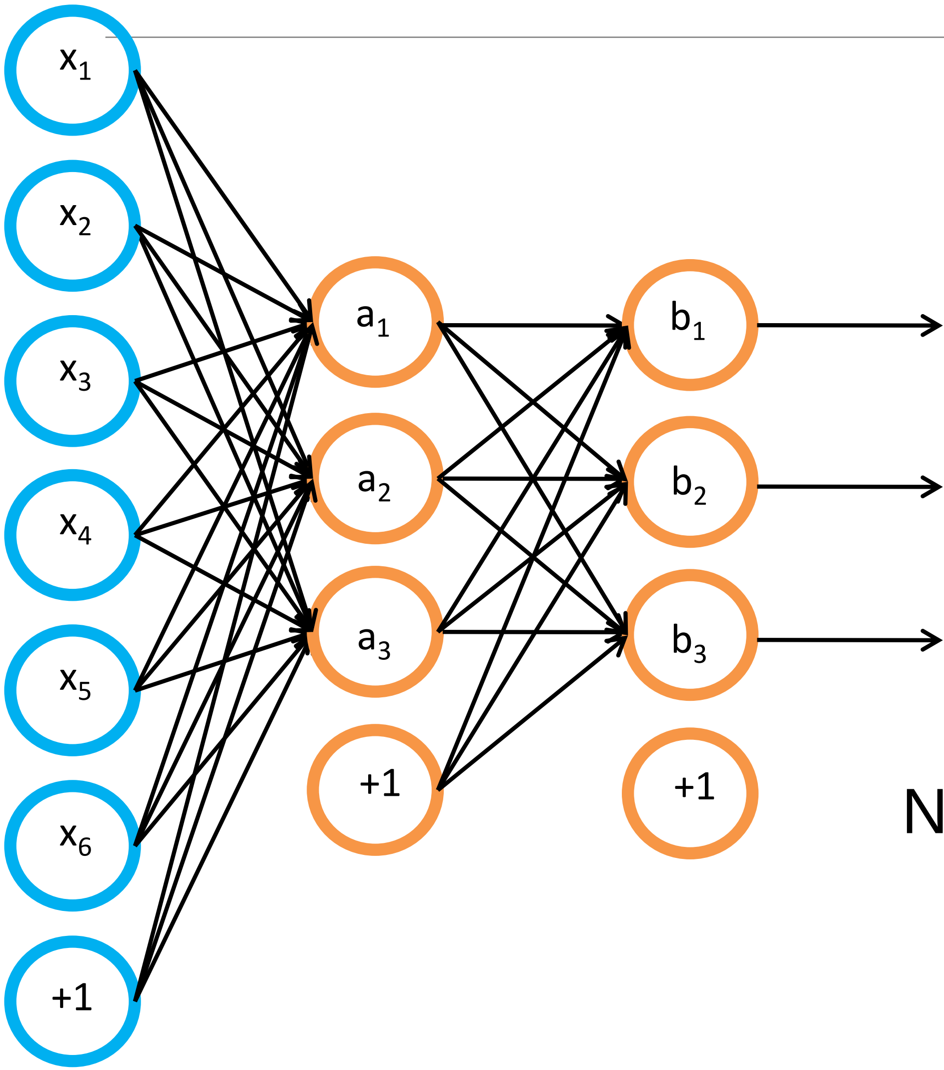


$h_\theta(x)$

Train parameters so that $h_\theta(x) \approx a$, subject to b_i's being sparse.

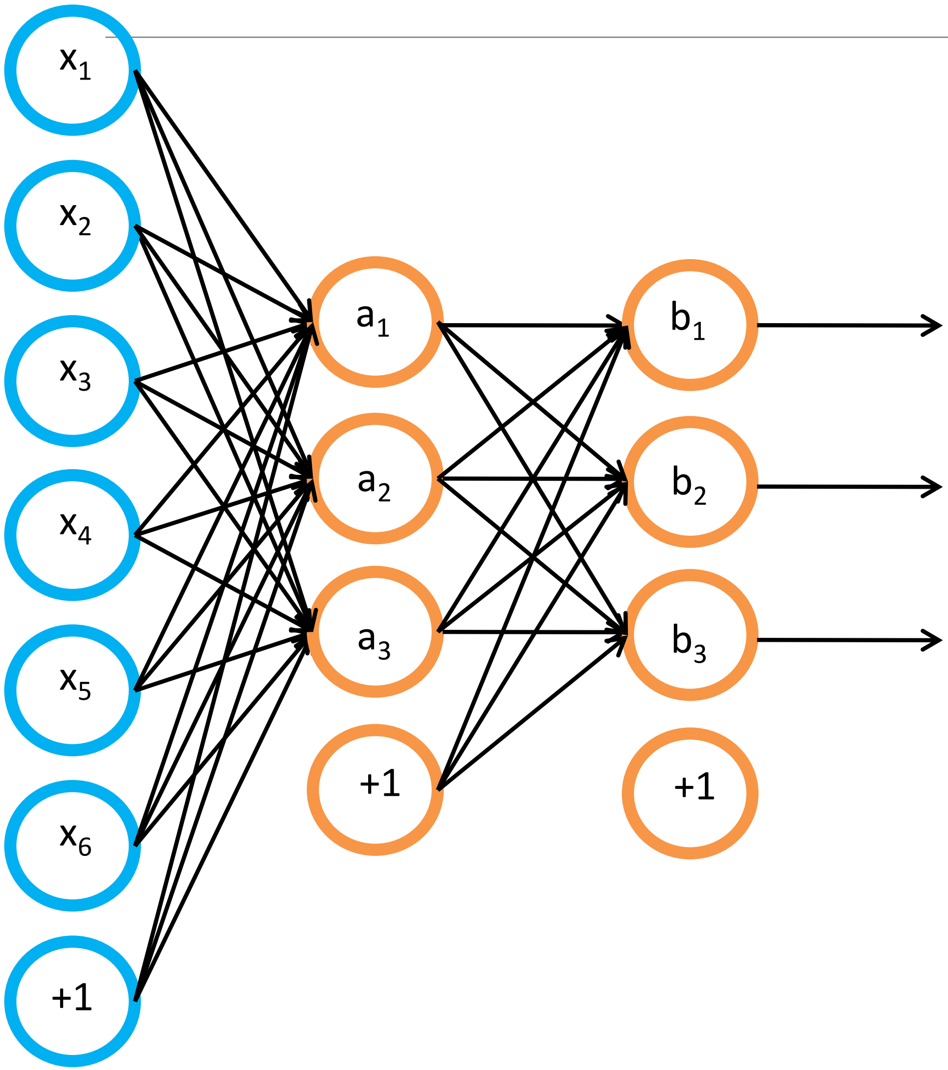


Train parameters so that $h_\theta(x) \approx a$, subject to b_i's being sparse.

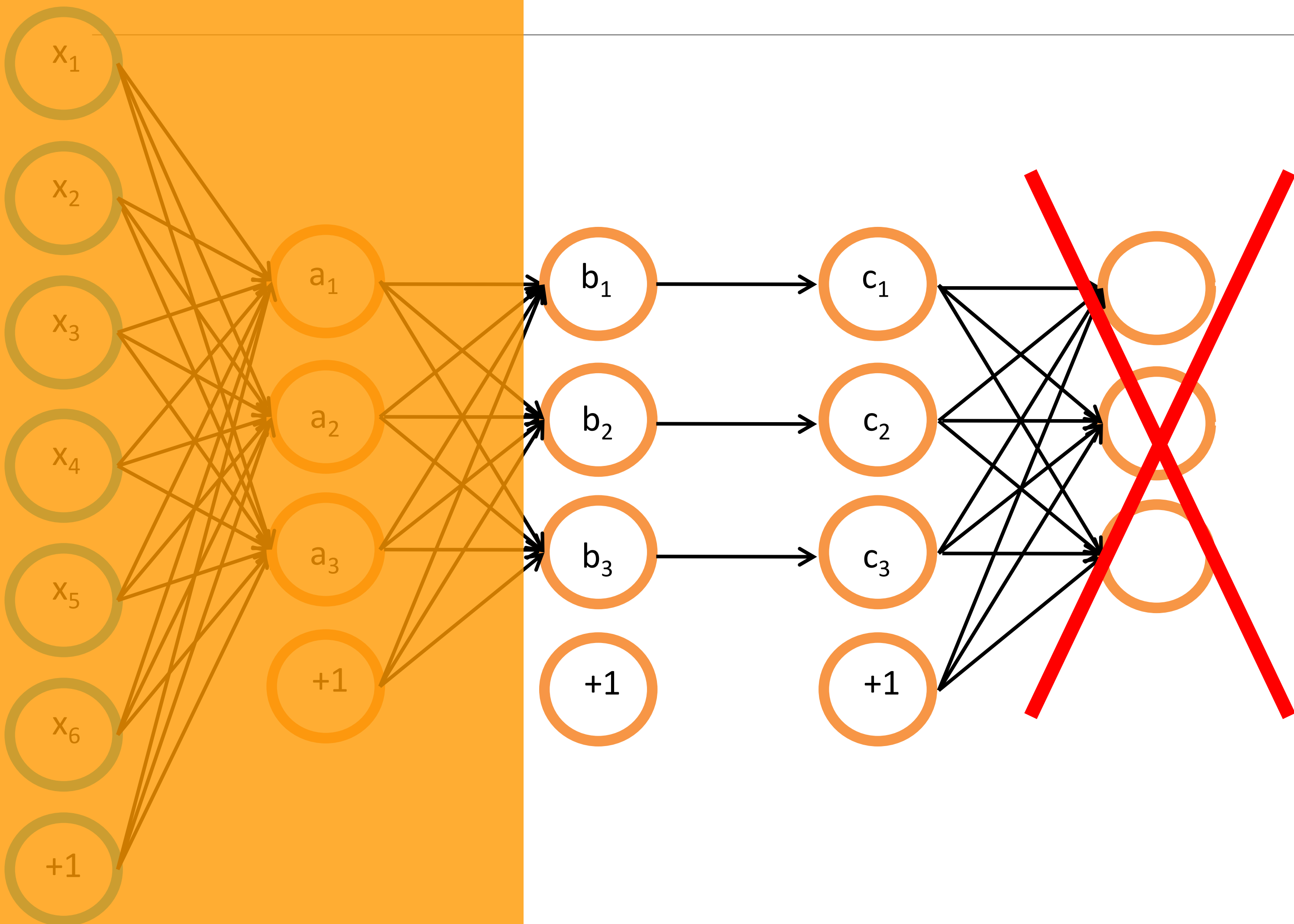


$$\begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

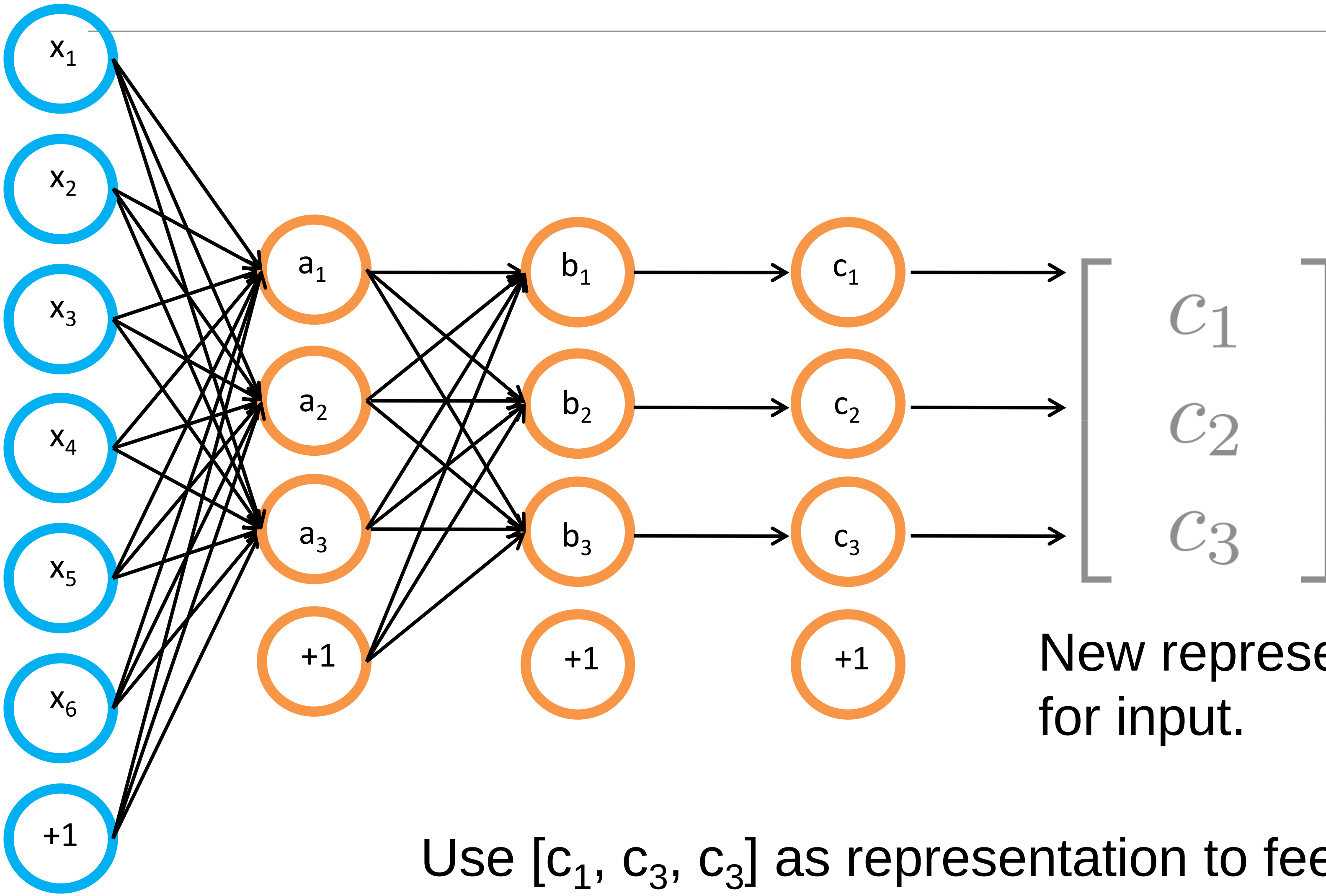
New representation for input.



x1
x2
x3
x4
x5
x6
+1
a1
a2
a3
+1
b1
b2
b3
+1



$$h_\theta(x) \approx b$$



New representation for input.

Use [c_1, c_3, c_3] as representation to feed to learning algorithm.

TensorFlow

Deep learning library, open-sourced by Google (11/2015)

TensorFlow provides primitives for

- defining functions on tensors
- automatically computing their derivatives

What is a tensor

What is a computational graph

Material from lecture by Bharath Ramsundar, March 2018, Stanford

Introduction to Keras

Official high-level API of TensorFlow

- Python
- 250K developers

Same front-end <-> Different back-ends

- TensorFlow (Google)
- CNTK (Microsoft)
- MXNet (Apache)
- Theano (RIP)

Hardware

- GPU (Nvidia)
- CPU (Intel/AMD)
- TPU (Google)

Companies: Netflix, Uber, Google, Nvidia…

Material from lecture by Francois Chollet, 2018, Stanford

Keras models

Installation
- Anaconda -> Tensorflow -> Keras

Build-in
- Conv1D, Conv2D, Conv3D…
- MaxPooling1D, MaxPooling2D, MaxPooling3D…
- Dense, Activation, RNN…

The Sequential Model
- Very simple
- Single-input, Single-output, sequential layer stacks

The functional API
- Mix & Match
- Multi-input, multi-output, arbitrary static graph topologies

Sequential

>>**from** keras.models **import** Sequential

>>model = Sequential()

>> **from** keras.layers **import** Dense

>> model.add(Dense(units=64, activation='relu', input_dim=100))

>> model.add(Dense(units=10, activation='softmax'))

>> model.compile(loss='categorical_crossentropy', optimizer='sgd', metrics=['accuracy'])

>> model.fit(x_train, y_train, epochs=5, batch_size=32)

>> loss_and_metrics = model.evaluate(x_test, y_test, batch_size=128)

>> classes = model.predict(x_test)

Functional

```
>> from keras.layers import Input, Dense

>> from keras.models import Model

>> inputs = Input(shape=(784,))

>> x = Dense(64, activation='relu')(inputs)

>> x = Dense(64, activation='relu')(x)

>> predictions = Dense(10, activation='softmax')(x)

>> model = Model(inputs=inputs, outputs=predictions)

>> model.compile(optimizer='rmsprop', loss='categorical_crossentropy',
metrics=['accuracy'])

>> model.fit(data, labels)
```

References

Stephens, Zachary D., et al. "Big data: astronomical or genomical?." *PLoS biology* 13.7 (2015): e1002195.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep learning.“ *Nature* 521.7553 (2015): 436-444.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Kietzmann, Tim Christian, Patrick McClure, and Nikolaus Kriegeskorte. "Deep Neural Networks In Computational Neuroscience. " *bioRxiv* (2017): 133504.