An Introduction to Convolutional Neural Networks

1

Yuan YAO

HKUST

Summary

- We had covered so far
  - Linear models (linear and logistic regression) – always a good start, simple yet powerful
  - Model Assessment and Selection – basics for all methods
  - Trees, Random Forests, and Boosting – good for high dim mixed-type heterogeneous features
  - Support Vector Machines – good for small amount of data but high dim geometric features
- Next, neural networks for unstructured data (image, language etc.):
  - **Convolutional Neural Networks** – image data
  - Generative models and GANs – new unsupervised learning for image, etc.
  - Recurrent Neural Networks, LSTM – sequence data
  - Transformer, BERT – machine translation etc.
 - Reinforcement Learning – Markov decision process, playing games, etc.

Kaggle survey: Top ML Methods

https://www.kaggle.com/surveys/2017
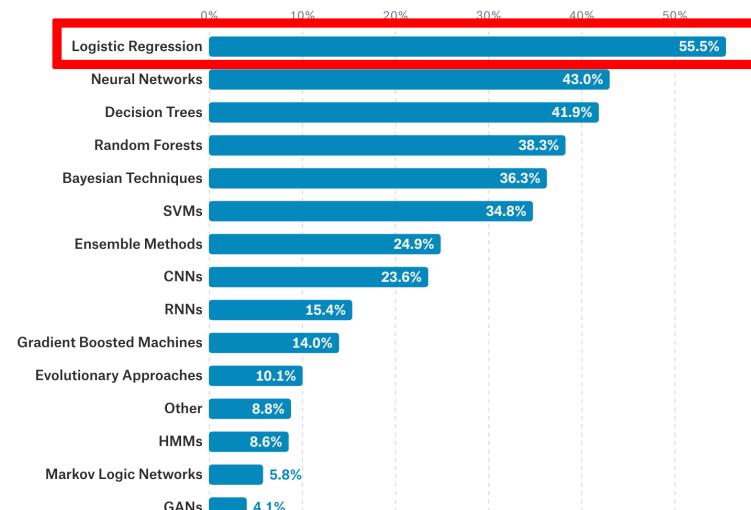
Academic

What data science methods are used at work?

Logistic regression is the most commonly reported data science method used at work for all industries *except* Military and Security where Neural Networks are used slightly more frequently.

Method	Percentage
Logistic Regression	55.5%
Neural Networks	43.0%
Decision Trees	41.9%
Random Forests	38.3%
Bayesian Techniques	36.3%
SVMs	34.8%
Ensemble Methods	24.9%
CNNs	23.6%
RNNs	15.4%
Gradient Boosted Machines	14.0%
Evolutionary Approaches	10.1%
Other	8.8%
HMMs	8.6%
Markov Logic Networks	5.8%
GANs	4.1%

1,201 responses

View code in Kaggle Kernels

Industry

What data science methods are used at work?

Logistic regression is the most commonly reported data science method used at work for all industries *except* Military and Security where Neural Networks are used slightly more frequently.

Method	Percentage
Logistic Regression	63.5%
Decision Trees	49.9%
Random Forests	46.3%
Neural Networks	37.6%
Bayesian Techniques	30.6%
Ensemble Methods	28.5%
SVMs	26.7%
Gradient Boosted Machines	23.9%
CNNs	18.9%
RNNs	12.3%
Other	8.3%
Evolutionary Approaches	5.5%
HMMs	5.4%
Markov Logic Networks	4.9%
GANs	2.8%

7,301 responses

View code in Kaggle Kernels

What type of data is used at work?
https://www.kaggle.com/surveys/2017
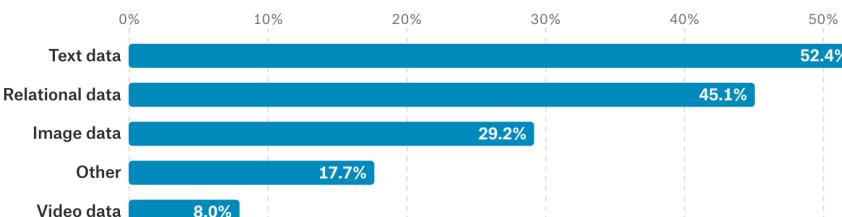
Academic

What type of data is used at work?

Relational data is the most commonly reported type of data used at work for all industries except for Academia and the Military and Security industry where text data's used more.

| Company Size ⇅ | Academic ⇅ | Job Title ⇅ |

Text data — 52.4%
Relational data — 45.1%
Image data — 29.2%
Other — 17.7%
Video data — 8.0%
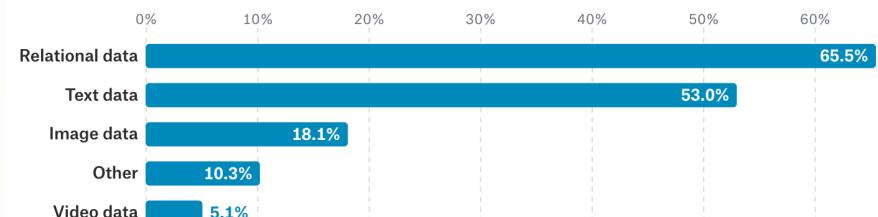
1,277 responses

Industry

What type of data is used at work?

Relational data is the most commonly reported type of data used at work for all industries except for Academia and the Military and Security industry where text data's used more.

| Company Size ⇅ | Industry ⇅ | Job Title ⇅ |

Relational data — 65.5%
Text data — 53.0%
Image data — 18.1%
Other — 10.3%
Video data — 5.1%
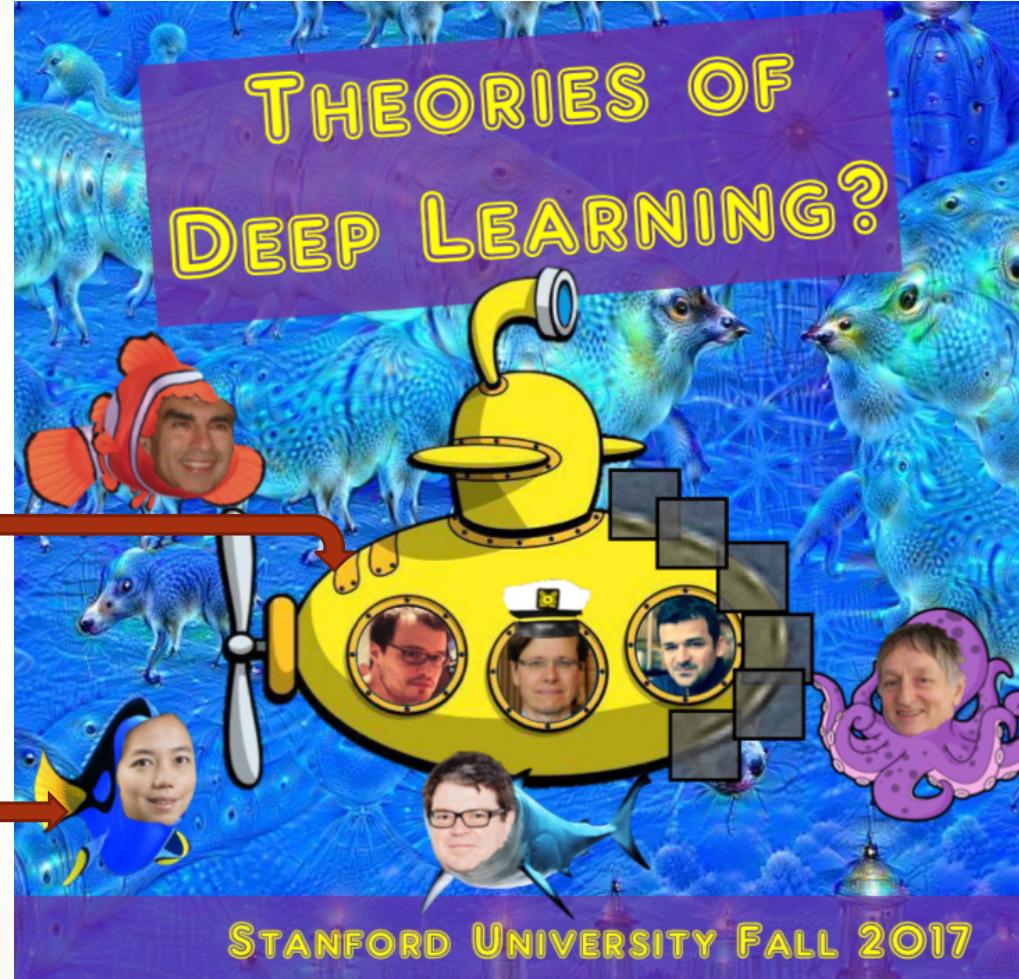
8,024 responses

Acknowledgement

https://stats385.github.io/

http://cs231n.github.io/
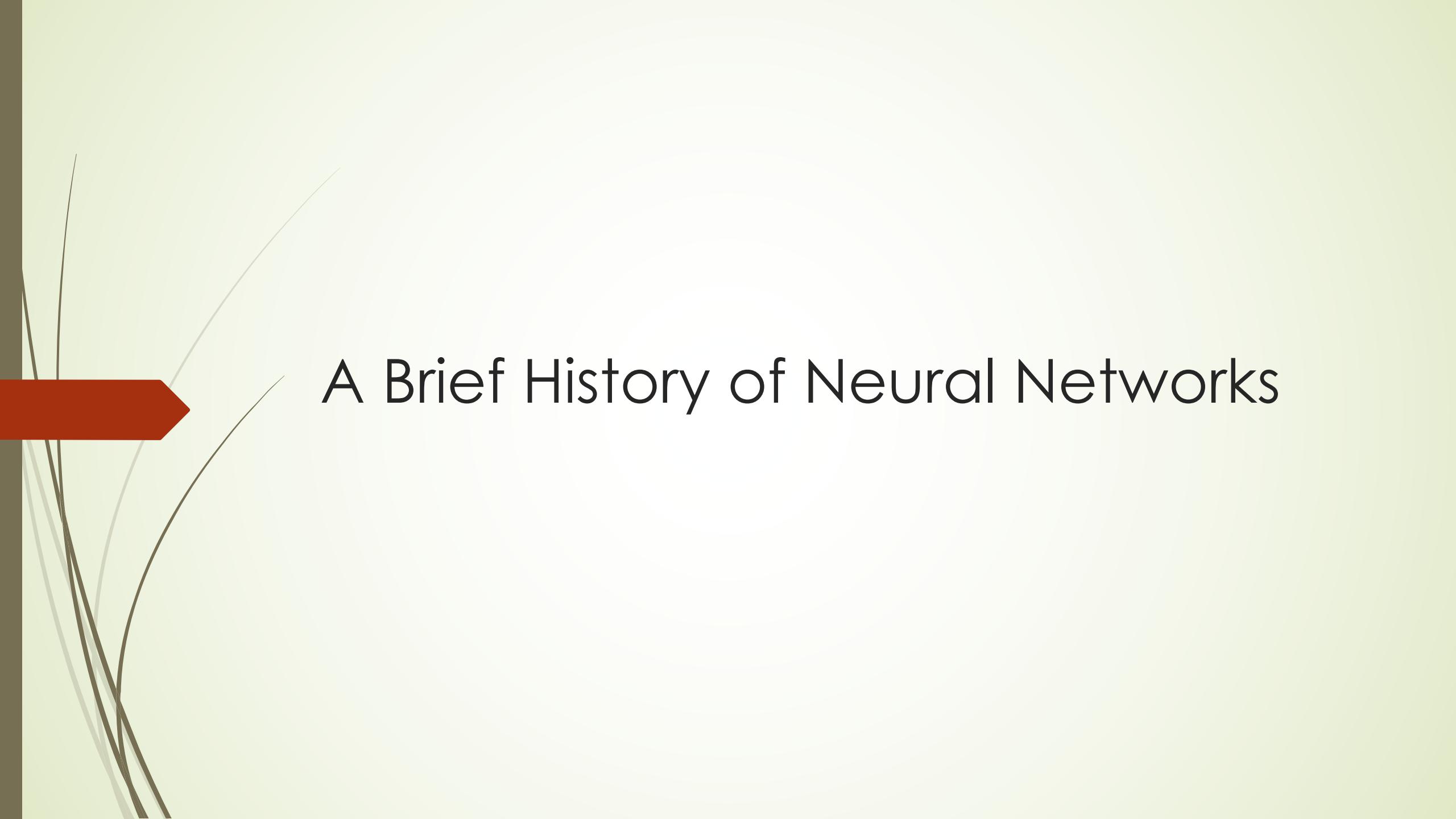
A following-up course at HKUST: https://deeplearning-math.github.io/

Some reference books on Deep Learning

- Deep Learning with Python, Manning Publications 2017
  - by François Chollet
  - https://www.manning.com/books/deep-learning-with-python?a_aid=keras&a_bid=76564dff
- Deep Learning, MIT Press 2016
  - By Ian Goodfellow, Yoshua Bengio, and Aaron Courville,
  - http://www.deeplearningbook.org/
- Many other public resources
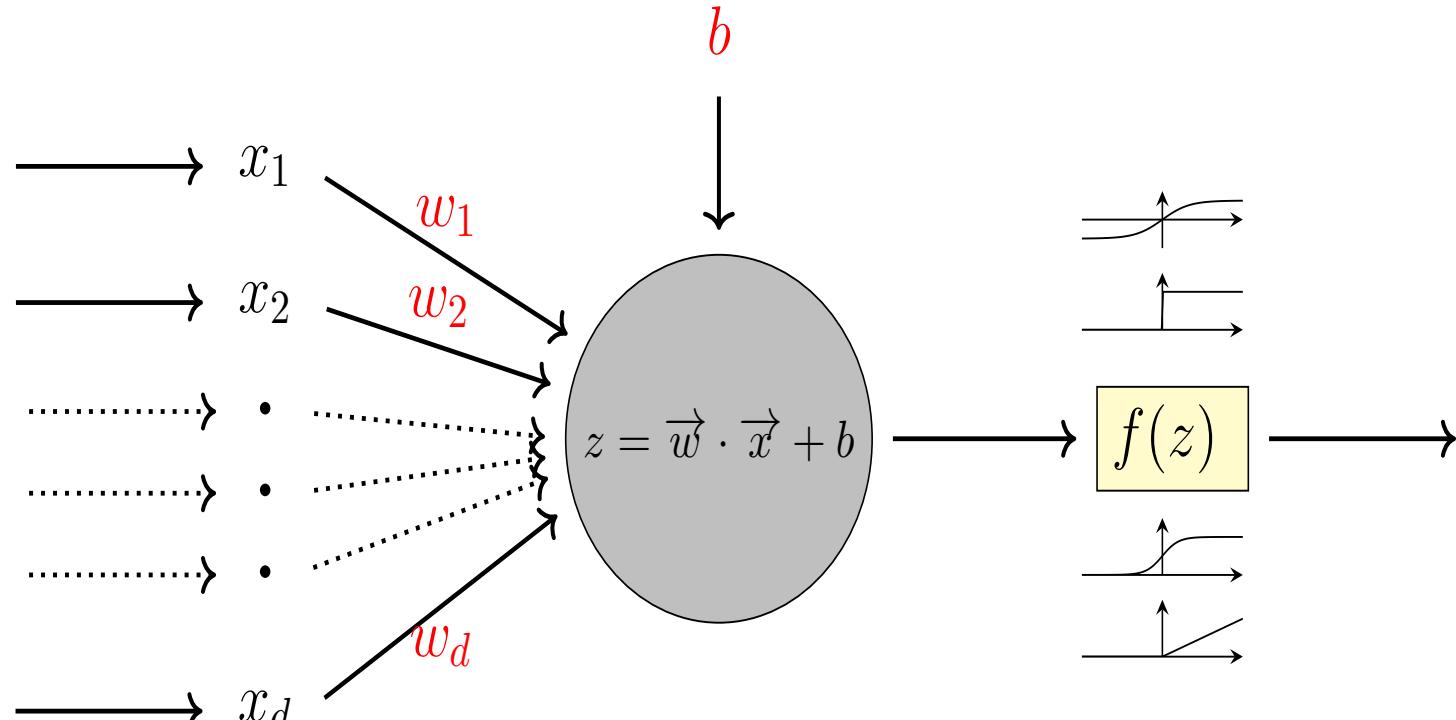
A Brief History of Neural Networks

Perceptron: single-layer

- Invented by Frank Rosenblatt (1957)

The Perceptron Algorithm

$$\ell(w) = -\sum_{i \in \mathcal{M}_w} y_i \langle w, \mathbf{x}_i \rangle, \quad \mathcal{M}_w = \{i : y_i \langle \mathbf{x}_i, w \rangle < 0, y_i \in \{-1, 1\}\}.$$

The Perceptron Algorithm is a *Stochastic Gradient Descent* method (Robbins–Monro 1950; Kiefer-Wolfowitz 1951) :

$$
\begin{aligned}
w_{t+1} \quad &= \quad w_t - \eta_t \nabla_i \ell(w) \\
&= \quad \begin{cases} w_t - \eta_t y_i \mathbf{x}_i, & \text{if } y_i w_t^T \mathbf{x}_i < 0, \\ w_t, & \text{otherwise.} \end{cases}
\end{aligned}
$$

Finite Stop of Perceptron for Separable Data
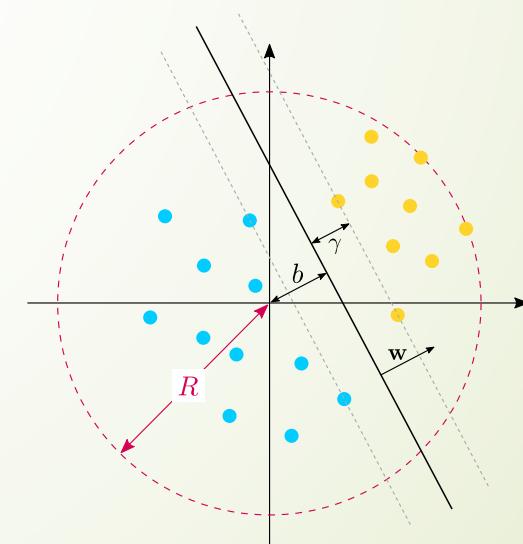
The perceptron convergence theorem was proved by Block (1962) and Novikoff (1962). The following version is based on that in Cristianini and Shawe-Taylor (2000).

Theorem 1 (Block, Novikoff). *Let the training set* $S = \{(\mathbf{x}_1, t_1), \ldots, (\mathbf{x}_n, t_n)\}$ *be contained in a sphere of radius R about the origin. Assume the dataset to be linearly separable, and let* $\mathbf{w}_{\text{opt}}$, $\|\mathbf{w}_{\text{opt}}\| = 1$, *define the hyperplane separating the samples, having functional margin* $\gamma > 0$. *We initialise the normal vector as* $\mathbf{w}_0 = \mathbf{0}$. *The number of updates, k, of the perceptron algorithms is then bounded by*

$$k \leq \left(\frac{2R}{\gamma}\right)^2.$$

(10)

Input ball: $\quad R = \max_i \|\mathbf{x}_i\|.$

Margin: $\quad \gamma := \min_i y_i f(x_i)$

Proof.

Proof. Though the proof can be done using the augmented normal vector and samples defined in the beginning, the notation will be a lot easier if we introduce a different augmentation: $\hat{\mathbf{w}} = (\mathbf{w}^\mathsf{T}, b/R)^\mathsf{T} = (w_1, \ldots, w_D, b/R)^\mathsf{T}$ and $\hat{\mathbf{x}} = (\mathbf{x}^\mathsf{T}, R)^\mathsf{T} = (x_1, \ldots, x_D, R)^\mathsf{T}$.

Proof (continued, growth of $|w_k|$)

We first derive an upper bound on how fast the normal vector grows. As the hyperplane is unchanged if we multiply $\hat{\mathbf{w}}$ by a constant, we can set $\eta = 1$ without loss of generality. Let $\hat{\mathbf{w}}_{k+1}$ be the updated (augmented) normal vector after the kth error has been observed.

$$\|\hat{\mathbf{w}}_{k+1}\|^2 = (\hat{\mathbf{w}}_k + t_i\hat{\mathbf{x}}_i)^\mathsf{T}(\hat{\mathbf{w}}_k + t_i\hat{\mathbf{x}}_i) \tag{11}$$

$$= \hat{\mathbf{w}}_k^\mathsf{T}\hat{\mathbf{w}}_k + \hat{\mathbf{x}}_i^\mathsf{T}\hat{\mathbf{x}}_i + 2t_i\hat{\mathbf{w}}_k^\mathsf{T}\hat{\mathbf{x}}_i \tag{12}$$

$$= \|\hat{\mathbf{w}}_k\|^2 + \|\hat{\mathbf{x}}_i\|^2 + 2t_i\hat{\mathbf{w}}_k^\mathsf{T}\hat{\mathbf{x}}_i. \tag{13}$$

Since an update was triggered, we know that $t_i\hat{\mathbf{w}}_k^\mathsf{T}\hat{\mathbf{x}}_i \leq 0$, thus

$$\|\hat{\mathbf{w}}_k\|^2 + \|\hat{\mathbf{x}}_i\|^2 + 2t_i\hat{\mathbf{w}}_k^\mathsf{T}\hat{\mathbf{x}}_i \leq \|\hat{\mathbf{w}}_k\|^2 + \|\hat{\mathbf{x}}_i\|^2 \tag{14}$$

$$= \|\hat{\mathbf{w}}_k\|^2 + (\|\mathbf{x}_i\|^2 + R^2) \tag{15}$$

$$\leq \|\hat{\mathbf{w}}_k\|^2 + 2R^2. \tag{16}$$

This implies that $\|\hat{\mathbf{w}}_k\|^2 \leq 2kR^2$, thus

$$\|\hat{\mathbf{w}}_{k+1}\|^2 \leq 2(k+1)R^2. \tag{17}$$

Proof (continued, projection on w$_{opt}$)

We then proceed to show how the inner product between an update of the normal vector and $\hat{\mathbf{w}}_{opt}$ increase with each update:

$$\hat{\mathbf{w}}_{opt}^{\mathsf{T}}\hat{\mathbf{w}}_{k+1} = \hat{\mathbf{w}}_{opt}^{\mathsf{T}}\hat{\mathbf{w}}_k + t_i\hat{\mathbf{w}}_{opt}^{\mathsf{T}}\hat{\mathbf{x}}_i \tag{18}$$

$$\geq \hat{\mathbf{w}}_{opt}^{\mathsf{T}}\hat{\mathbf{w}}_k + \gamma \tag{19}$$

$$\geq (k+1)\gamma, \tag{20}$$

since $\hat{\mathbf{w}}_{opt}^{\mathsf{T}}\hat{\mathbf{w}}_k \geq k\gamma$. We therefore have

$$k^2\gamma^2 \leq (\hat{\mathbf{w}}_{opt}^{\mathsf{T}}\hat{\mathbf{w}}_k)^2 \leq \|\hat{\mathbf{w}}_{opt}\|^2\|\hat{\mathbf{w}}_k\|^2 \leq 2kR^2\|\hat{\mathbf{w}}_{opt}\|^2, \tag{21}$$

where we have made use of the Cauchy-Schwarz inequality. As $k^2\gamma^2$ grows faster than $2kR^2$, Eq. (21) can hold if and only if

$$k \leq 2\|\hat{\mathbf{w}}_{opt}\|^2\frac{R^2}{\gamma^2}. \tag{22}$$

Proof (continued, combined bounds)

As $b \leq R$, we can rewrite the norm of the normal vector:

$$\|\hat{\mathbf{w}}_{\text{opt}}\|^2 = \|\mathbf{w}_{\text{opt}}\|^2 + \frac{b^2}{R^2} \leq \|\mathbf{w}_{\text{opt}}\|^2 + 1 = 2. \tag{23}$$

The bound on k now becomes

$$k \leq 4\frac{R^2}{\gamma^2} = \left(\frac{2R}{\gamma}\right)^2, \tag{24}$$
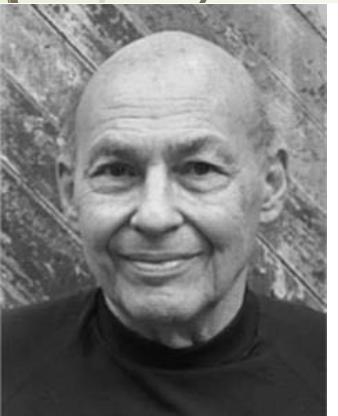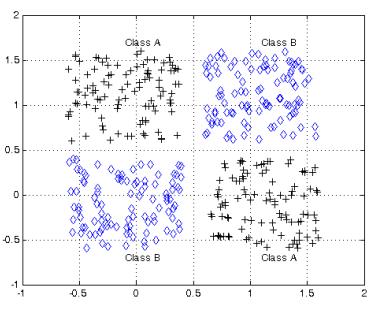
which therefore bounds the number of updates necessary to find the separating hyper-plane. $\square$

Locality or Sparsity of Computation
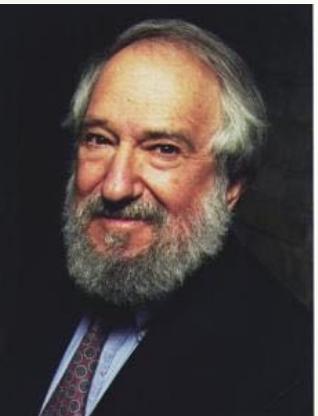
Minsky and Papert, 1969
Perceptron can't do **XOR** classification
Perceptron needs infinite global
information to compute **connectivity**
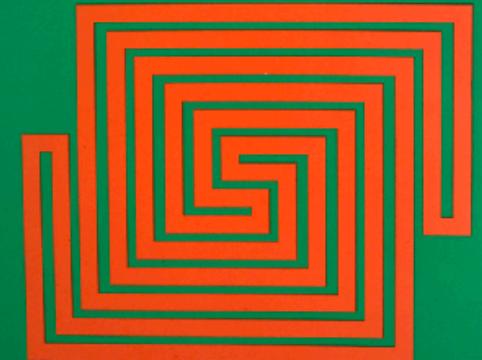
Locality or **Sparsity** is important:
Locality in time?
Locality in space?

Marvin Minsky   **Seymour Papert**
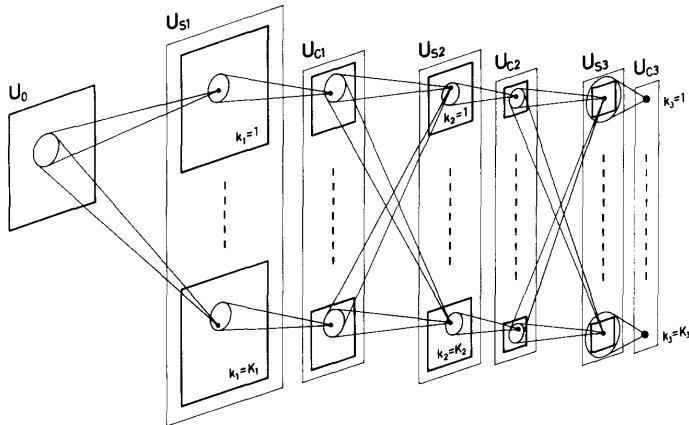
Convolutional Neural Networks: shift invariances and locality

Biol. Cybernetics 36, 193–202 (1980)
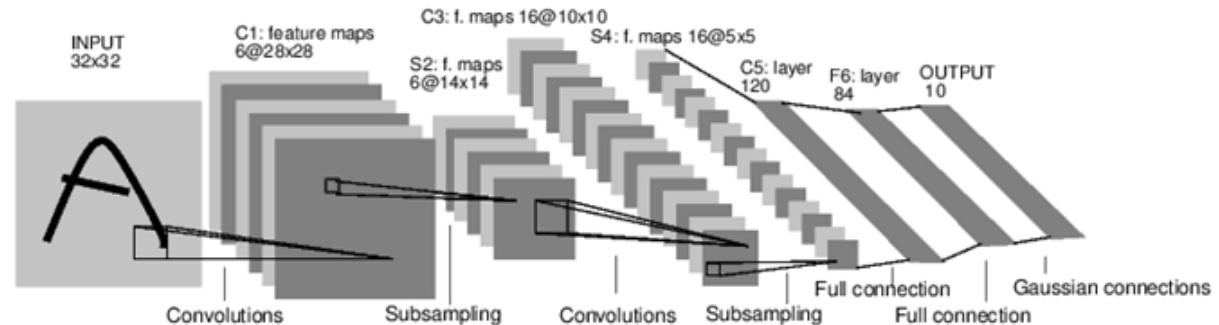
Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position

Kunihiko Fukushima

NHK Broadcasting Science Research Laboratories, Kinuta, Setagaya, Tokyo, Japan

- Can be traced to *Neocognitron* of Kunihiko Fukushima (1979)
- Yann LeCun combined convolutional neural networks with back propagation (1989)
- Imposes **shift invariance** and **locality** on the weights
- Forward pass remains similar
- Backpropagation slightly changes – need to sum over the gradients from all spatial positions
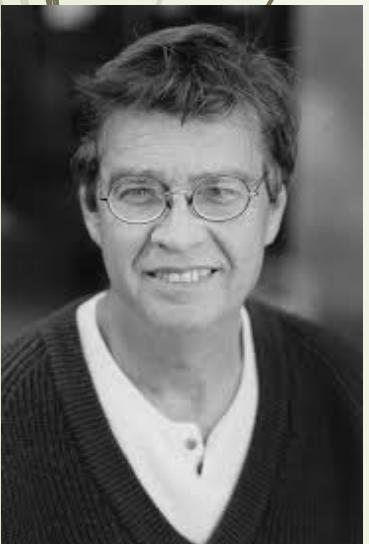
Multilayer Perceptrons (MLP) and Back-Propagation (BP) Algorithms

Rumelhart, Hinton, Williams (1986)
Learning representations by back-propagating errors, Nature, 323(9): 533-536
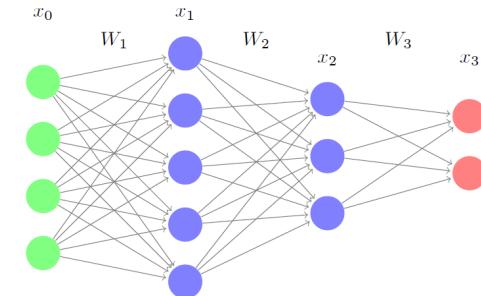
BP algorithms as **stochastic gradient descent algorithms (Robbins–Monro 1950; Kiefer-Wolfowitz 1951)** with Chain rules of Gradient maps

MLP classifies **XOR**, but the global hurdle on topology (connectivity) computation still exists

BP Algorithm: Forward Pass
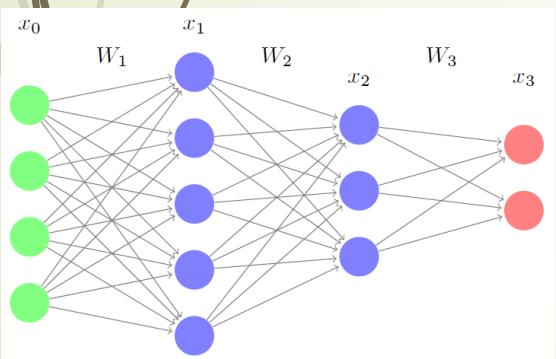
- Cascade of repeated [linear operation followed by coordinatewise nonlinearity]'s
- Nonlinearities: sigmoid, hyperbolic tangent, (recently) ReLU.


Algorithm 1 Forward pass
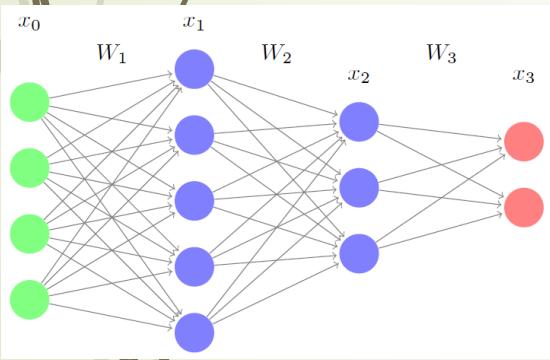
Input: x_0
Output: x_L

1: **for** $\ell = 1$ to L **do**
2: $\quad x_\ell = f_\ell(W_\ell x_{\ell-1} + b_\ell)$
3: **end for**

BP algorithm = Gradient Descent Method

- Training examples $\{x_0^i\}_{i=1}^n$ and labels $\{y^i\}_{i=1}^n$
- Output of the network $\{x_L^i\}_{i=1}^m$
- Objective

$$J(\{W_l\}, \{b_l\}) = \frac{1}{n} \sum_{i=1}^{n} \frac{1}{2} \|y^i - x_L^i\|_2^2 \qquad (1)$$

Other losses include cross-entropy, logistic loss, exponential loss, etc.

- Gradient descent

$$W_l = W_l - \eta \frac{\partial J}{\partial W_l}$$

$$b_l = b_l - \eta \frac{\partial J}{\partial b_l}$$

In practice: use Stochastic Gradient Descent (SGD)

Derivation of BP: Lagrangian Multiplier

Given n training examples $(I_i, y_i) \equiv$ (input,target) and L layers

- Constrained optimization

$$\min_{W,x} \quad \sum_{i=1}^{n} \|x_i(L) - y_i\|_2$$

$$\text{subject to} \quad x_i(\ell) = f_\ell\Big[W_\ell x_i(\ell-1)\Big],$$

$$i = 1, \dots, n, \quad \ell = 1, \dots, L, \ x_i(0) = I_i$$

- Lagrangian formulation (Unconstrained)

$$\min_{W,x,B} \mathcal{L}(W, x, B)$$

$$\mathcal{L}(W, x, B) = \sum_{i=1}^{n} \left\{ \|x_i(L) - y_i\|_2^2 + \right.$$

$$\left. \sum_{\ell=1}^{L} B_i(\ell)^T \Big(x_i(\ell) - f_\ell\big[W_\ell x_i(\ell-1)\big]\Big) \right\}$$

back-propagation – derivation

- $\frac{\partial \mathcal{L}}{\partial B}$

Forward pass

$$x_i(\ell) = f_\ell\Big[\underbrace{W_\ell x_i\,(\ell-1)}_{A_i(\ell)}\Big] \quad \ell = 1,\ldots,L, \quad i = 1,\ldots,n$$

- $\frac{\partial \mathcal{L}}{\partial x}, z_\ell = [\nabla f_\ell]B(\ell)$

Backward (adjoint) pass

$$z(L) = 2\nabla f_L\Big[A_i(L)\Big]\,(y_i - x_i(L))$$

$$z_i(\ell) = \nabla f_\ell\Big[A_i(\ell)\Big]W_{\ell+1}^T z_i(\ell+1) \quad \ell = 0,\ldots,L-1$$

- $W \leftarrow W + \lambda\frac{\partial \mathcal{L}}{\partial W}$
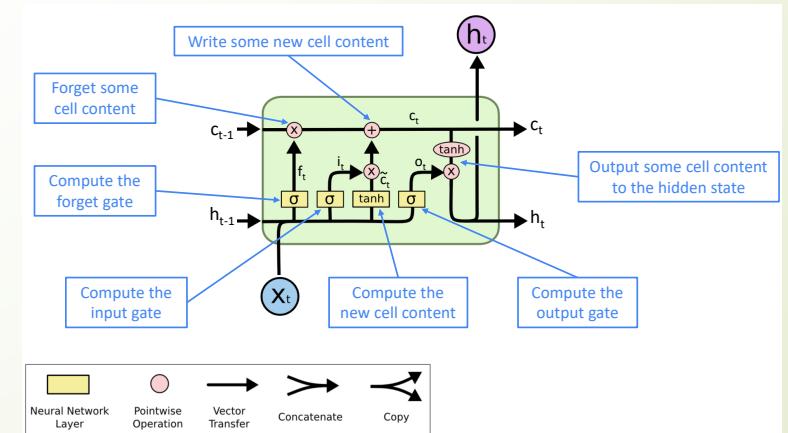
Weight update

$$W_\ell \leftarrow W_\ell + \lambda\sum_{i=1}^n z_i(\ell)x_i^T(\ell-1)$$
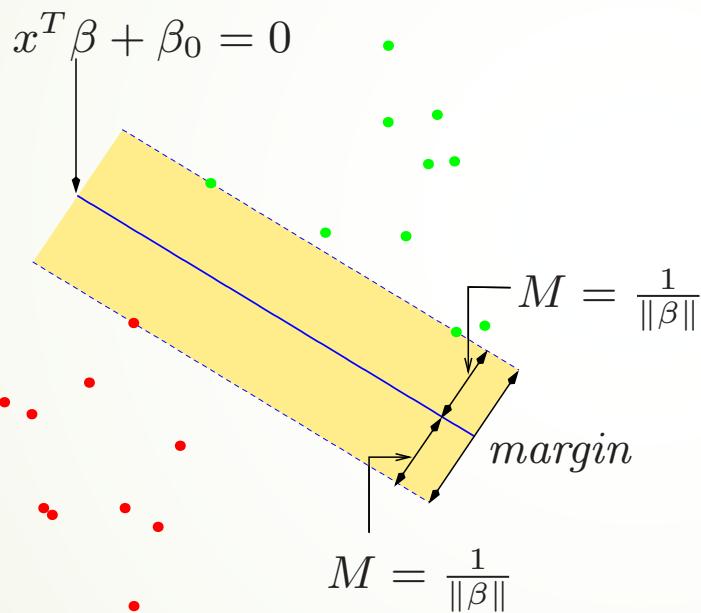
Long-Short-Term-Memory (LSTM)

- Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory". *Neural Computation*. **9** (8): 1735–1780. (https://www.bioinf.jku.at/publications/older/2604.pdf)

- BP can not train deep networks due to gradient vanishing problem etc.

- Introduction of short path to train deep networks without vanishing gradient problem.

- This idea will come back to Convolutional Networks as ResNet in 2015.
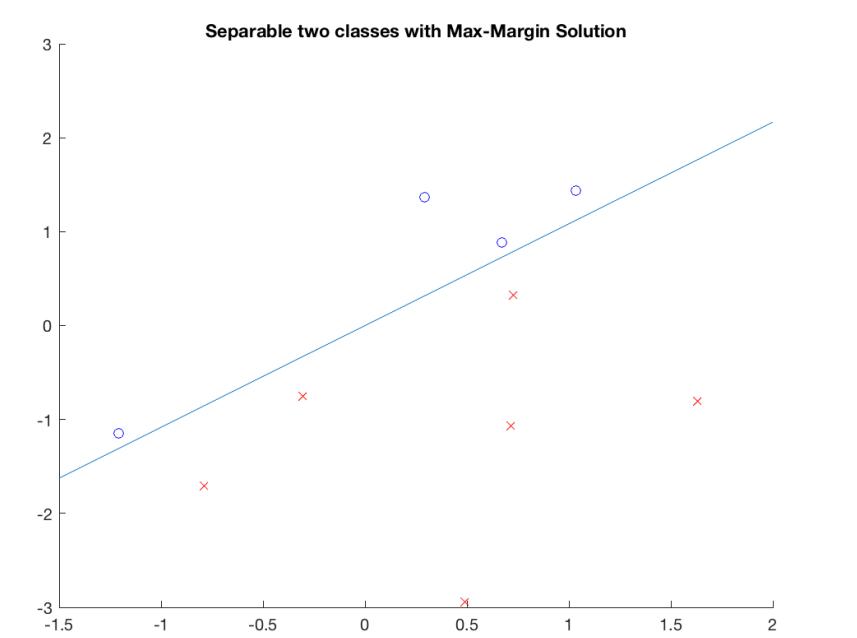
Support Vector Machine (Max-Margin Classifier)

$$\text{minimize}_{\beta_0, \beta_1, \ldots, \beta_p} \|\beta\|^2 := \sum_j \beta_j^2$$

$$\text{subject to } y_i(\beta_0 + \beta_1 x_{i1} + \ldots + \beta_p x_{ip}) \geq 1 \text{ for all } i$$

$$x^T \beta + \beta_0 = 0$$

$$M = \frac{1}{\|\beta\|}$$

$$M = \frac{1}{\|\beta\|}$$

margin

Separable two classes with Max-Margin Solution

Vladmir Vapnik, 1994

Convex optimization + Reproducing Kernel Hilbert Spaces (Grace Wahba etc.)
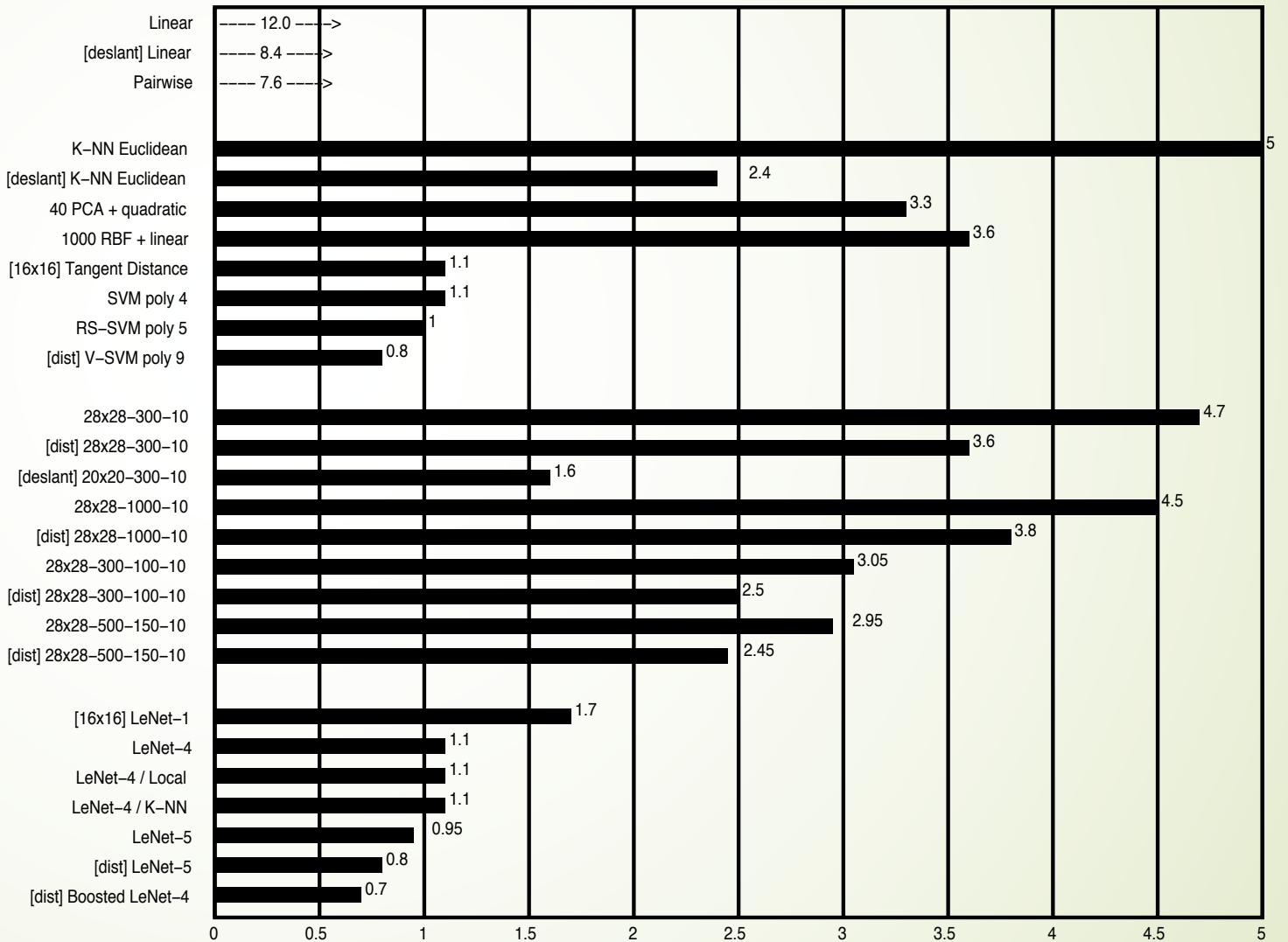
MNIST Challenge Test Error: SVM vs. CNN LeCun et al. 1998


Simple SVM performs as well as Multilayer Convolutional Neural Networks which need careful tuning (LeNets)

Second dark era for NN: 2000s

Linear	---- 12.0 ---->
[deslant] Linear	---- 8.4 ---->
Pairwise	---- 7.6 ---->
K–NN Euclidean	5
[deslant] K–NN Euclidean	2.4
40 PCA + quadratic	3.3
1000 RBF + linear	3.6
[16x16] Tangent Distance	1.1
SVM poly 4	1.1
RS–SVM poly 5	1
[dist] V–SVM poly 9	0.8
28x28–300–10	4.7
[dist] 28x28–300–10	3.6
[deslant] 20x20–300–10	1.6
28x28–1000–10	4.5
[dist] 28x28–1000–10	3.8
28x28–300–100–10	3.05
[dist] 28x28–300–100–10	2.5
28x28–500–150–10	2.95
[dist] 28x28–500–150–10	2.45
[16x16] LeNet–1	1.7
LeNet–4	1.1
LeNet–4 / Local	1.1
LeNet–4 / K–NN	1.1
LeNet–5	0.95
[dist] LeNet–5	0.8
[dist] Boosted LeNet–4	0.7

LeNet

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   Gaussian connections

Full connection

Fig. 2.   Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

http://blog.csdn.net/Chenyukuai6625

- **Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner**. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, november 1998.
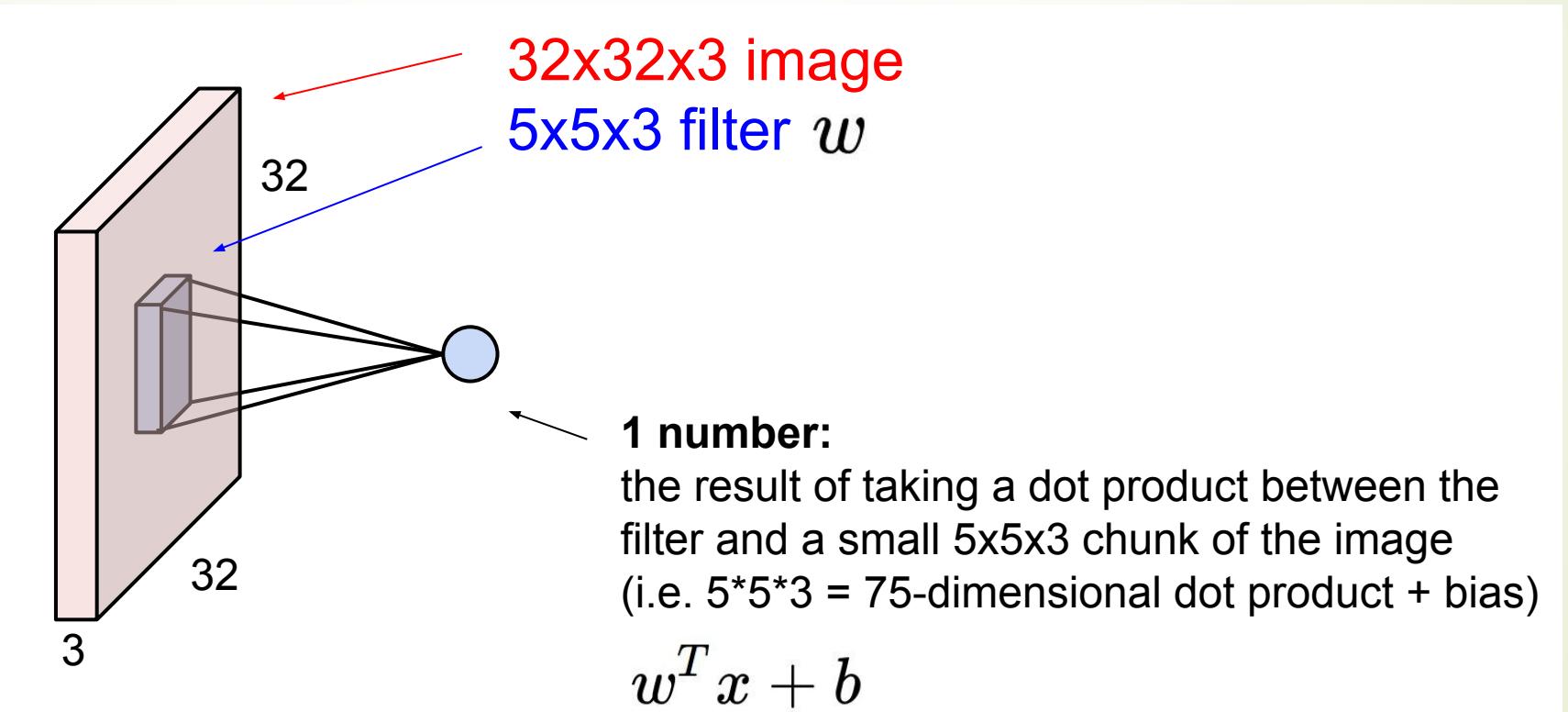
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

input

1

3072

Wx

10 x 3072
weights

activation

1

10
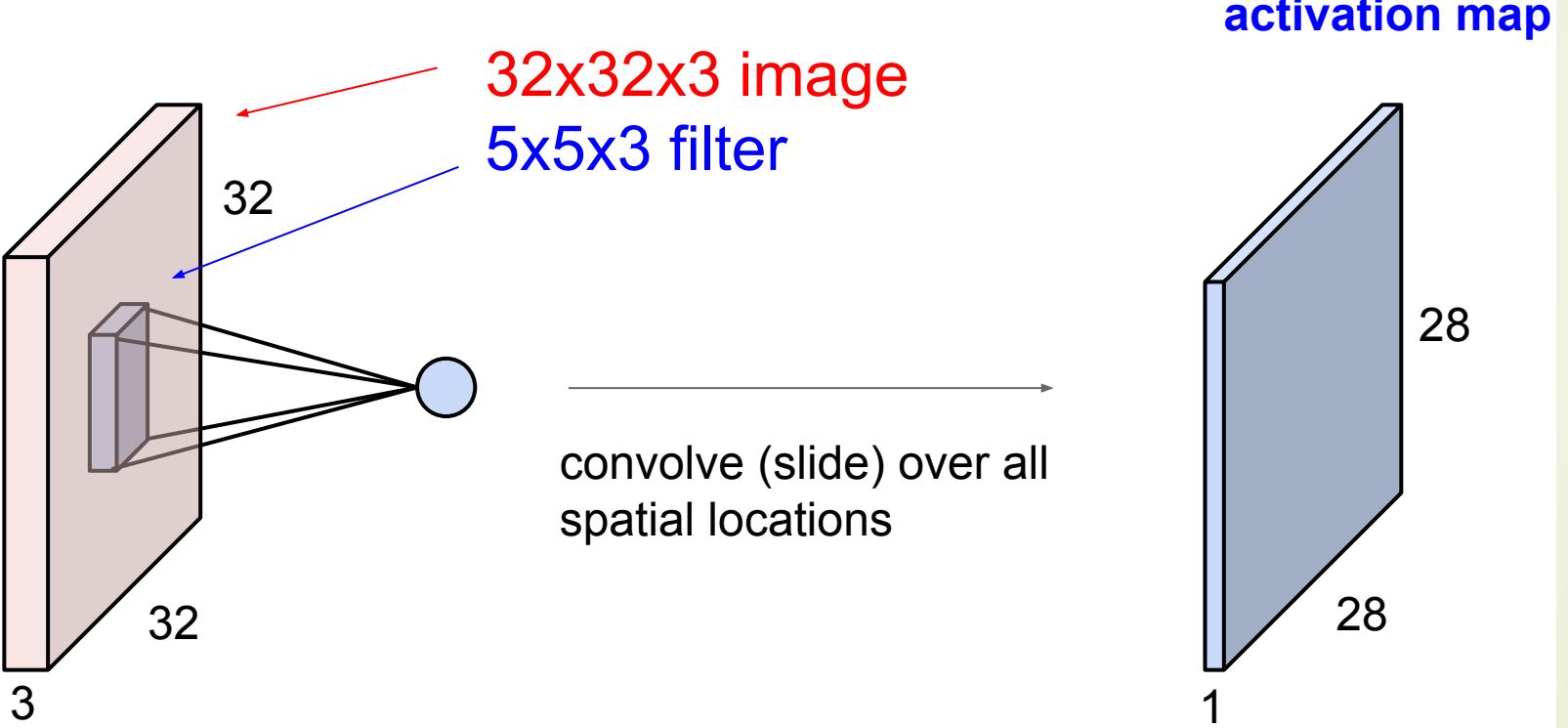
1 number:
the result of taking a dot product
between a row of W and the input
(a 3072-dimensional dot product)

Convolution

32x32x3 image

5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

Convolution Layer: a first (blue) filter

32x32x3 image
5x5x3 filter

activation map

convolve (slide) over all spatial locations
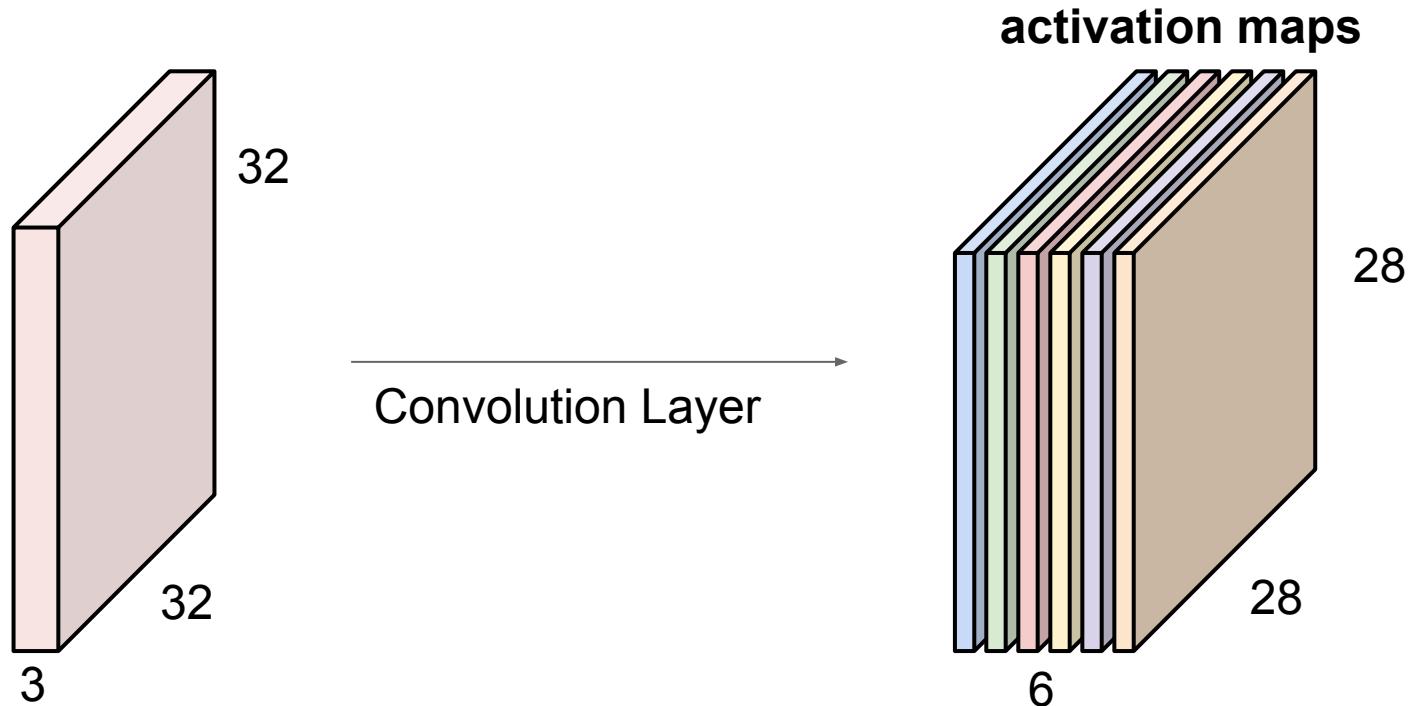
Convolution Layer: a second (green) filter

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations
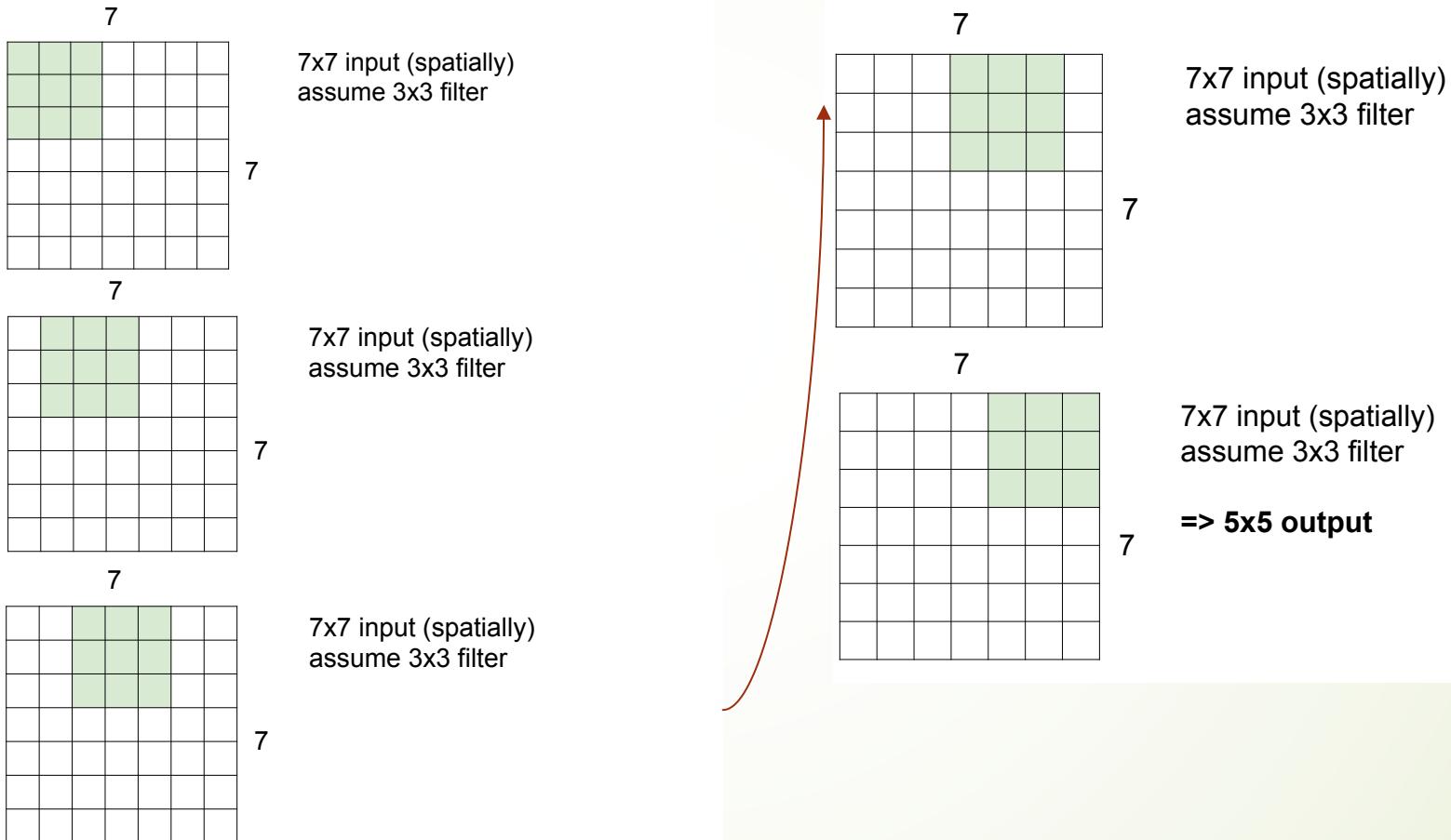
activation maps

Convolution Layer

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

activation maps

32

32

3

Convolution Layer

28

28

6
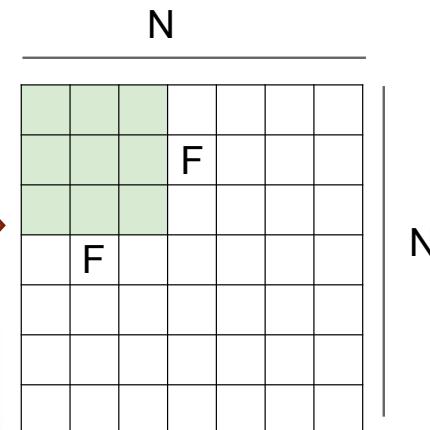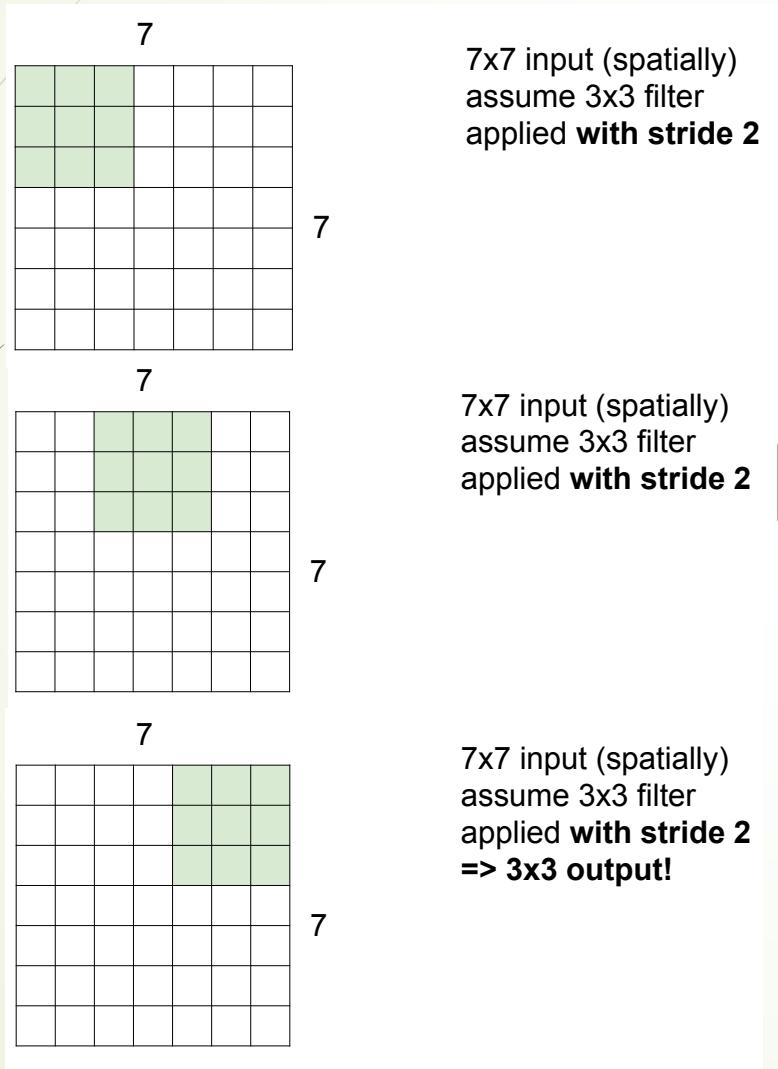
We stack these up to get a "new image" of size 28x28x6!

A Closer Look at Convolution: stride=1

7x7 input (spatially)
assume 3x3 filter

7x7 input (spatially)
assume 3x3 filter

7x7 input (spatially)
assume 3x3 filter

7x7 input (spatially)
assume 3x3 filter

7x7 input (spatially)
assume 3x3 filter

**=> 5x5 output**

A Closer Look at Convolution: stride=2

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

A Closer Look at Convolution: Padding

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7
3x3 filter, applied with **stride 1**
**pad with 1 pixel** border => what is the output?
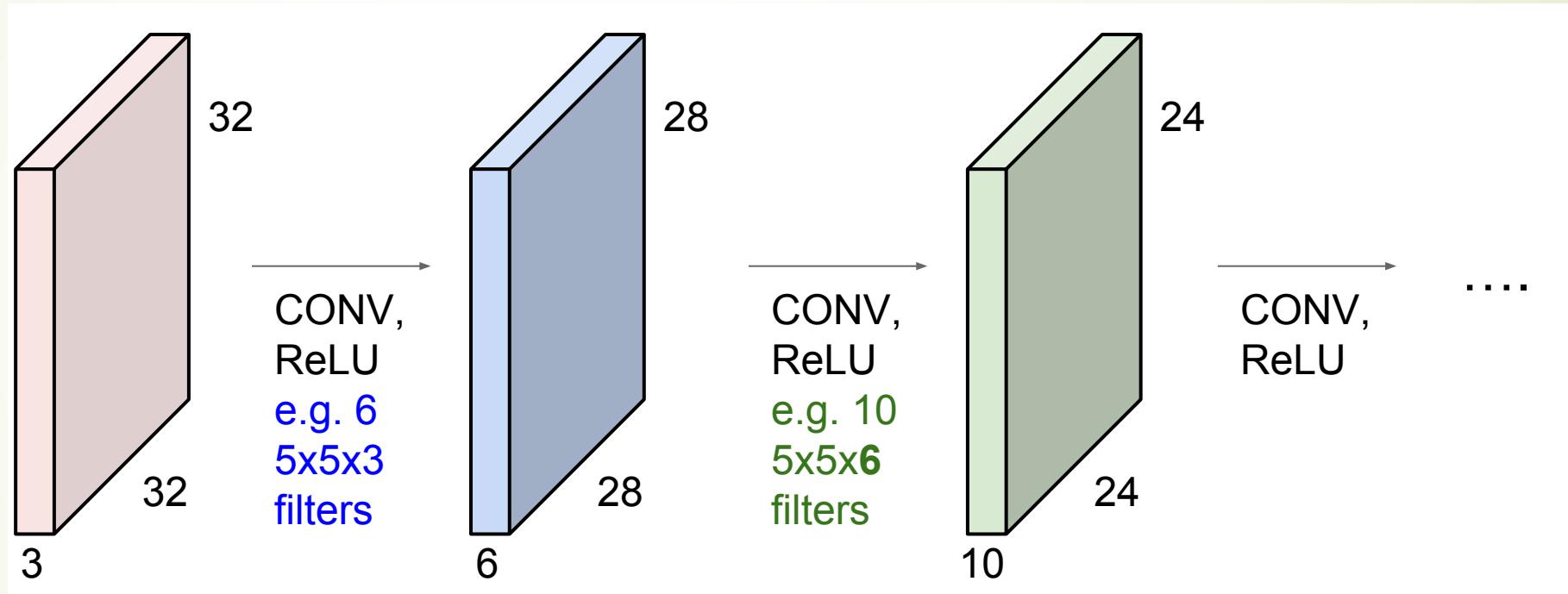
7x7 output!
in general, common to see CONV layers with
stride 1, filters of size FxF, and zero-padding with
(F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
    F = 5 => zero pad with 2
    F = 7 => zero pad with 3

ConvNet:

32 × 32 × 3

CONV, ReLU
e.g. 6
5x5x3
filters

28 × 28 × 6

CONV, ReLU
e.g. 10
5x5x**6**
filters

24 × 24 × 10

CONV, ReLU

....
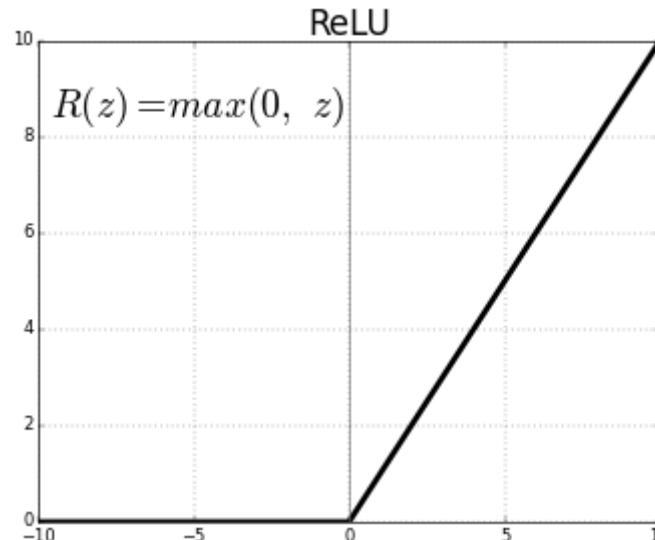
Stride = 1
Padding = 0

Formula: NewImageSize = floor((ImageSize – Filter + 2*Padding)/Stride + 1)

Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

ReLU
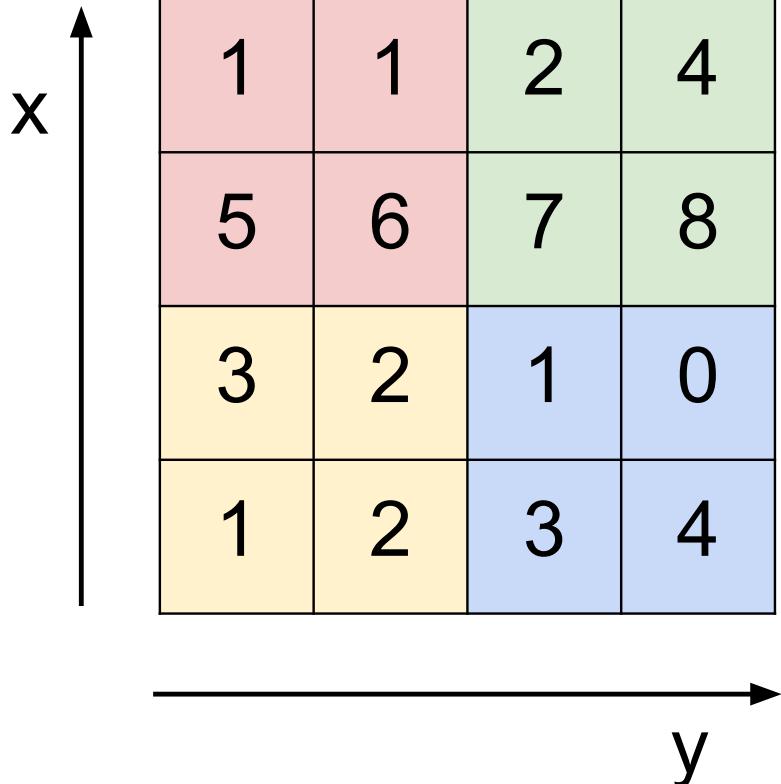
- Non-saturating function and therefore faster convergence when compared to other nonlinearities
- Problem of dying neurons

Source: `https://ml4a.github.io/ml4a/neural_networks/`

Max Pooling

Single depth slice

x

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

y

max pool with 2x2 filters and stride 2

→
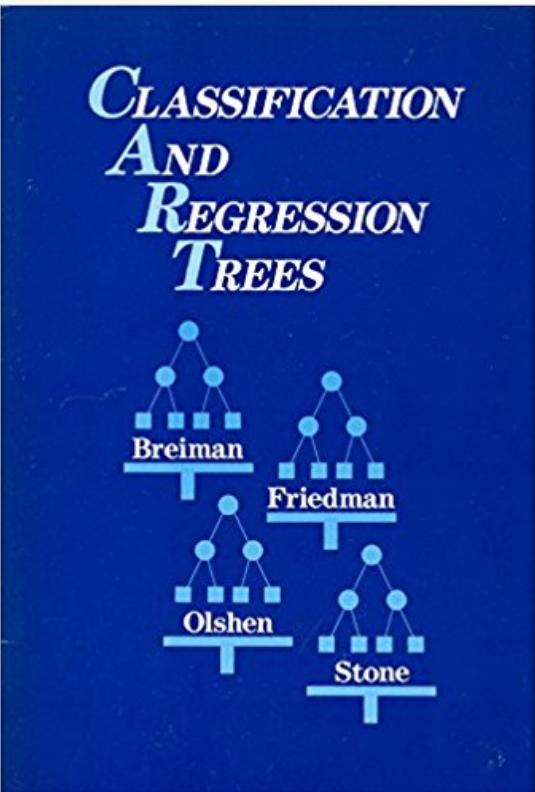
6	8
3	4

2000-2010: The Era of SVM, Boosting, ... as nights of Neural Networks

Decision Trees and Boosting
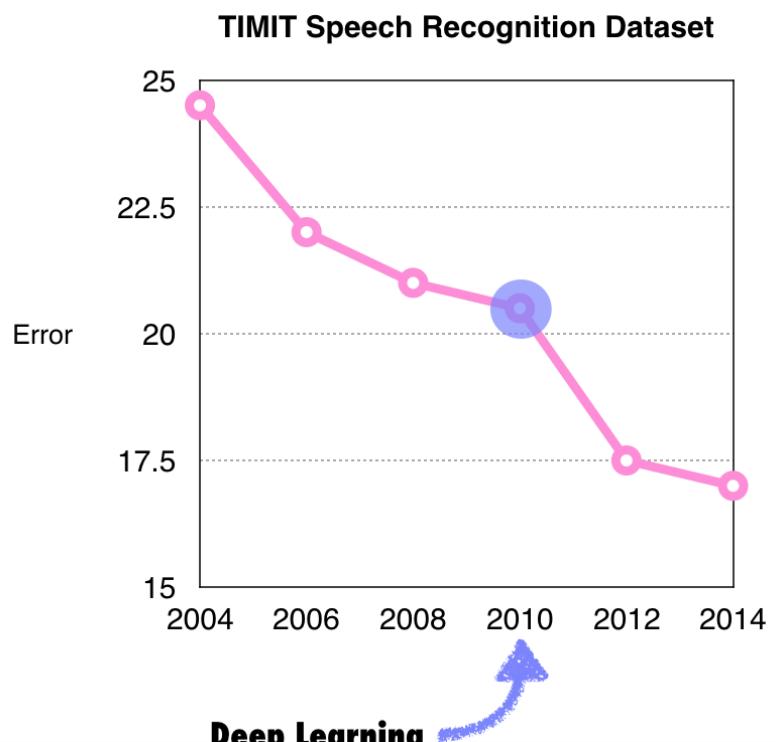
- Breiman, Friedman, Olshen, Stone, (1983): CART
- ``The Boosting problem'' (M. Kearns & L. Valiant): **Can a set of weak learners create a single strong learner**? (三个臭皮匠顶个诸葛亮？)
- Breiman (1996): Bagging
- Freund, Schapire (1997): AdaBoost
- Breiman (2001): Random Forests
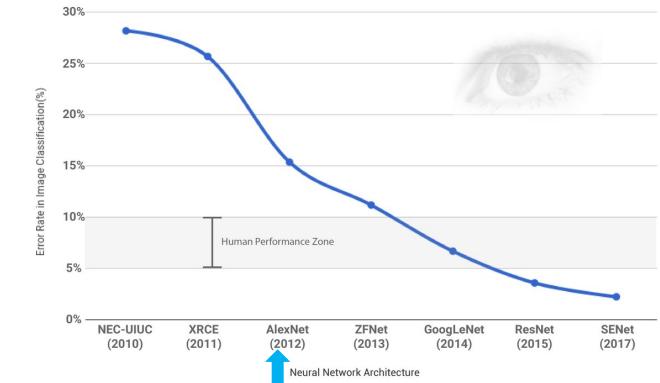
Around the year of 2012…

Speech Recognition: TIMIT

TIMIT Speech Recognition Dataset

Error

Deep Learning

Computer Vision: ImageNet
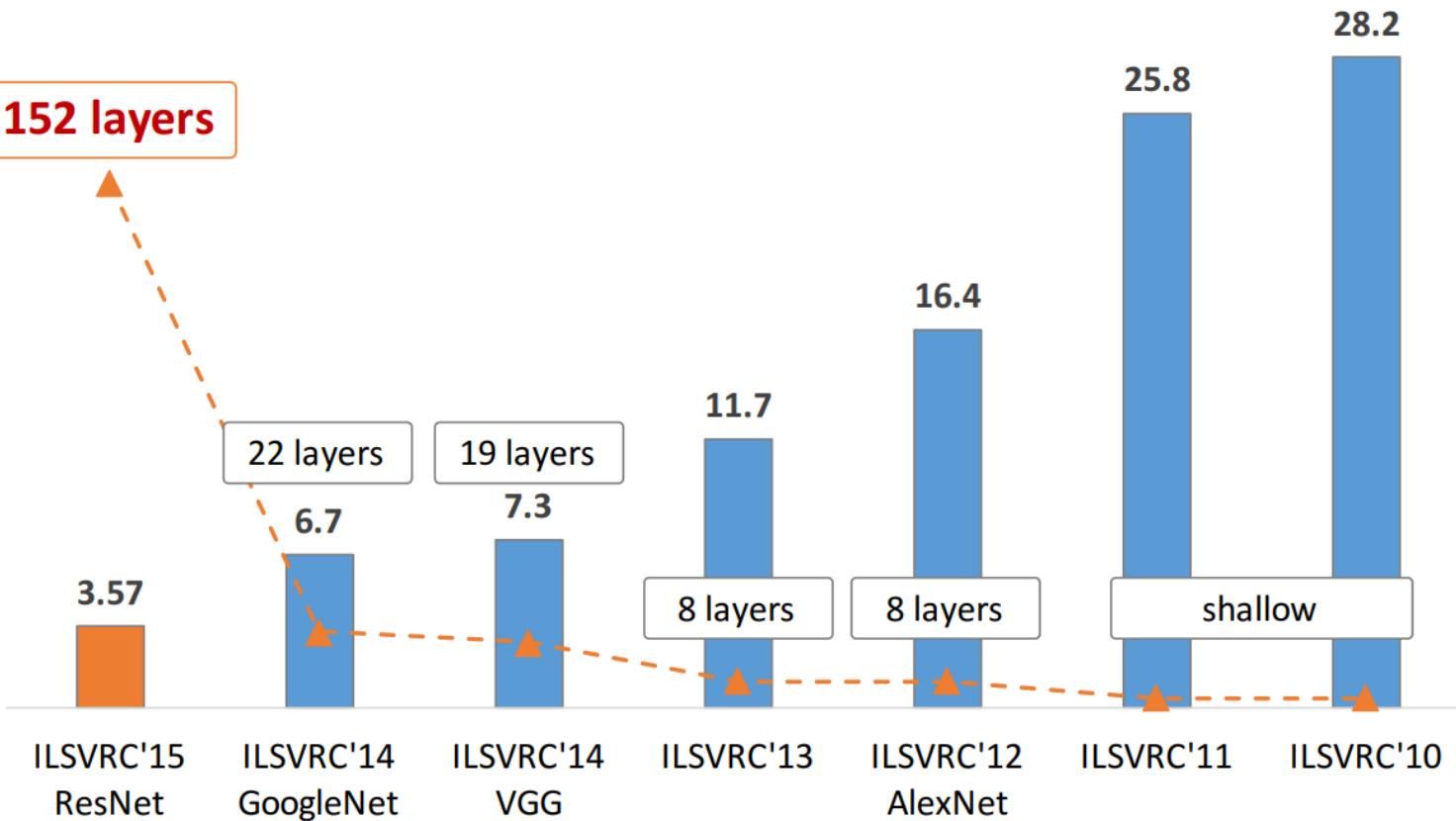
- ImageNet (subset):
  - 1.2 million training images
  - 100,000 test images
  - 1000 classes
- ImageNet large-scale visual recognition Challenge

source: https://www.linkedin.com/pulse/must-read-path-breaking-papers-image-classification-muktabh-mayank

Deep Learning
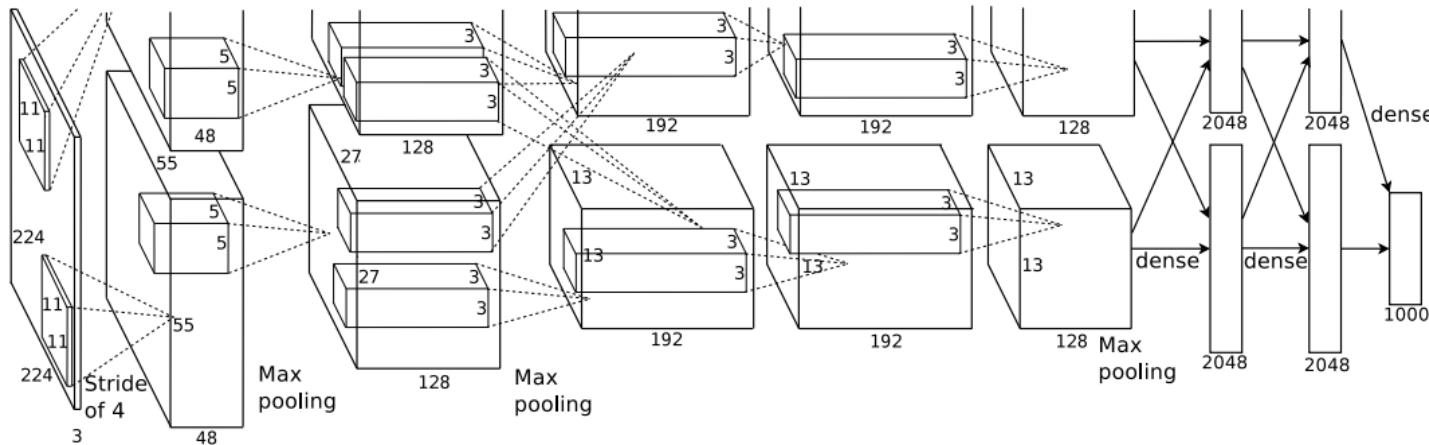
Depth as function of year


[He et al., 2016]
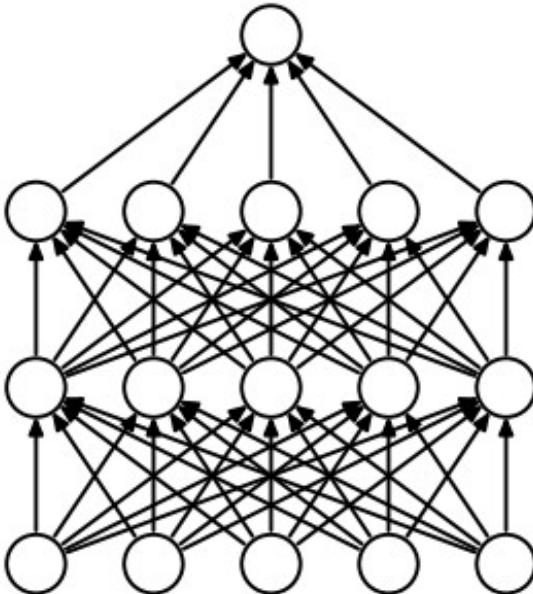
AlexNet (2012): Architecture

- 8 layers: first 5 convolutional, rest fully connected
- ReLU nonlinearity
- Local response normalization
- Max-pooling
- Dropout
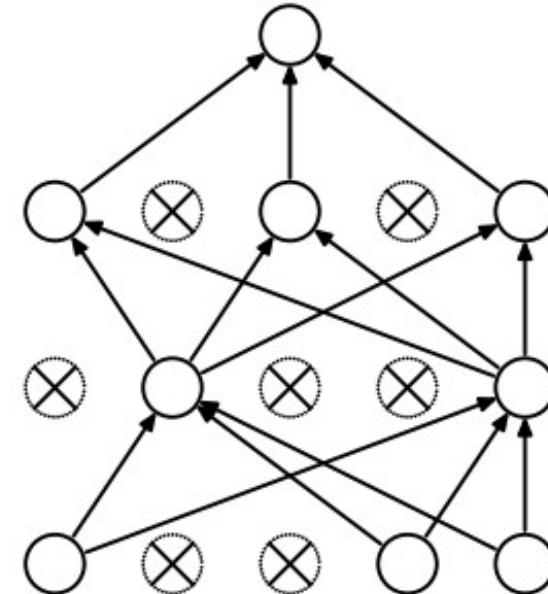
Source: [Krizhevsky et al., 2012]

# AlexNet (2012): Dropout

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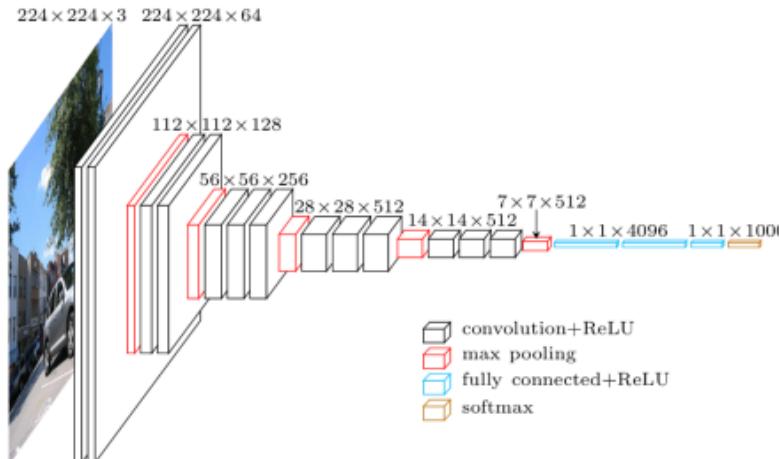
(b) After applying dropout.

Source: [Srivastava et al., 2014]

- Zero every neuron with probability $1 - p$
- At test time, multiply every neuron by p

VGG (2014) [Simonyan-Zisserman'14]

- Deeper than AlexNet: 11-19 layers versus 8
- No local response normalization
- Number of filters multiplied by two every few layers
- Spatial extent of filters 3×3 in all layers
- Instead of 7×7 filters, use three layers of 3×3 filters
  - Gain intermediate nonlinearity
 - Impose a regularization on the 7×7 filters

Source: https://blog.heuritech.com/2016/02/29/
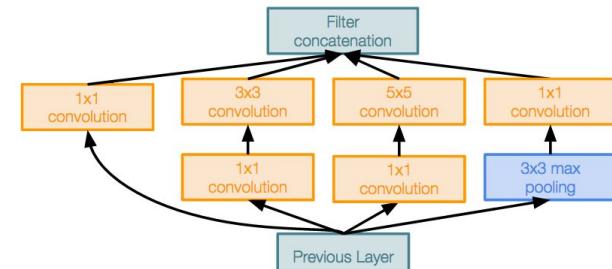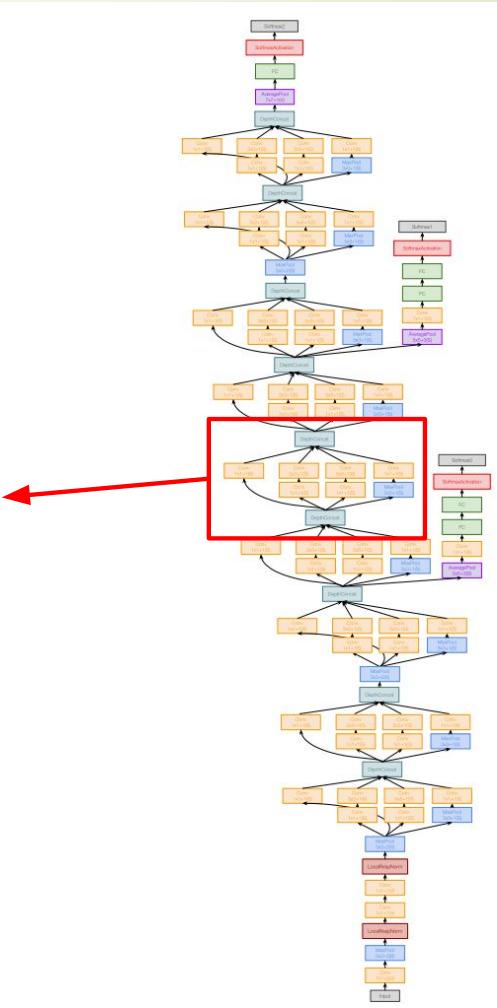
GoogLeNet *[Szegedy et al., 2014]*

- 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)
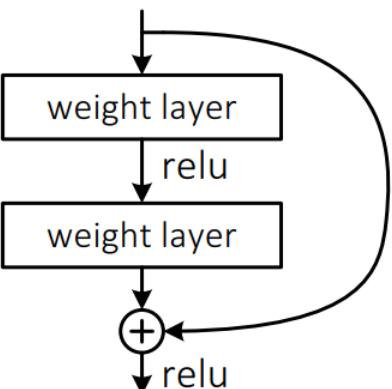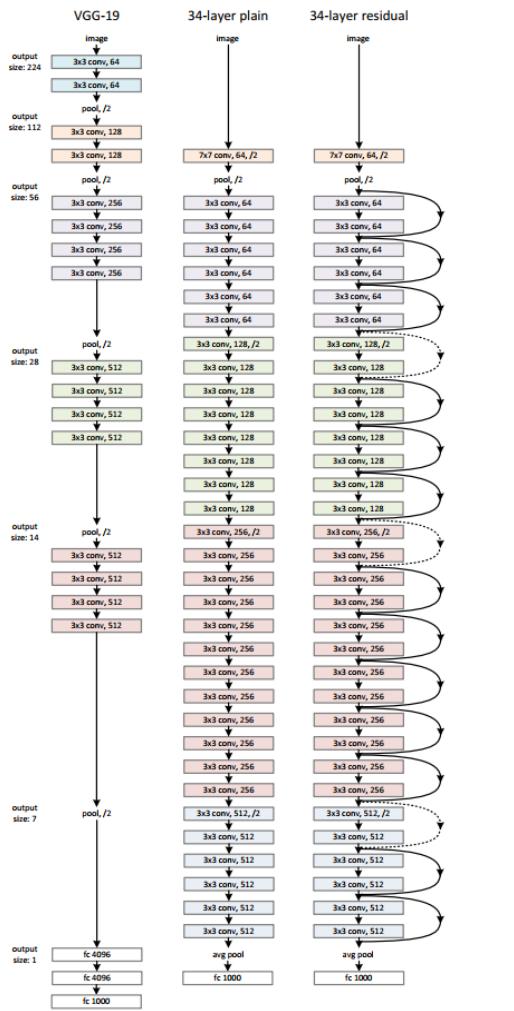
ResNet (2015) [HGRS-15]
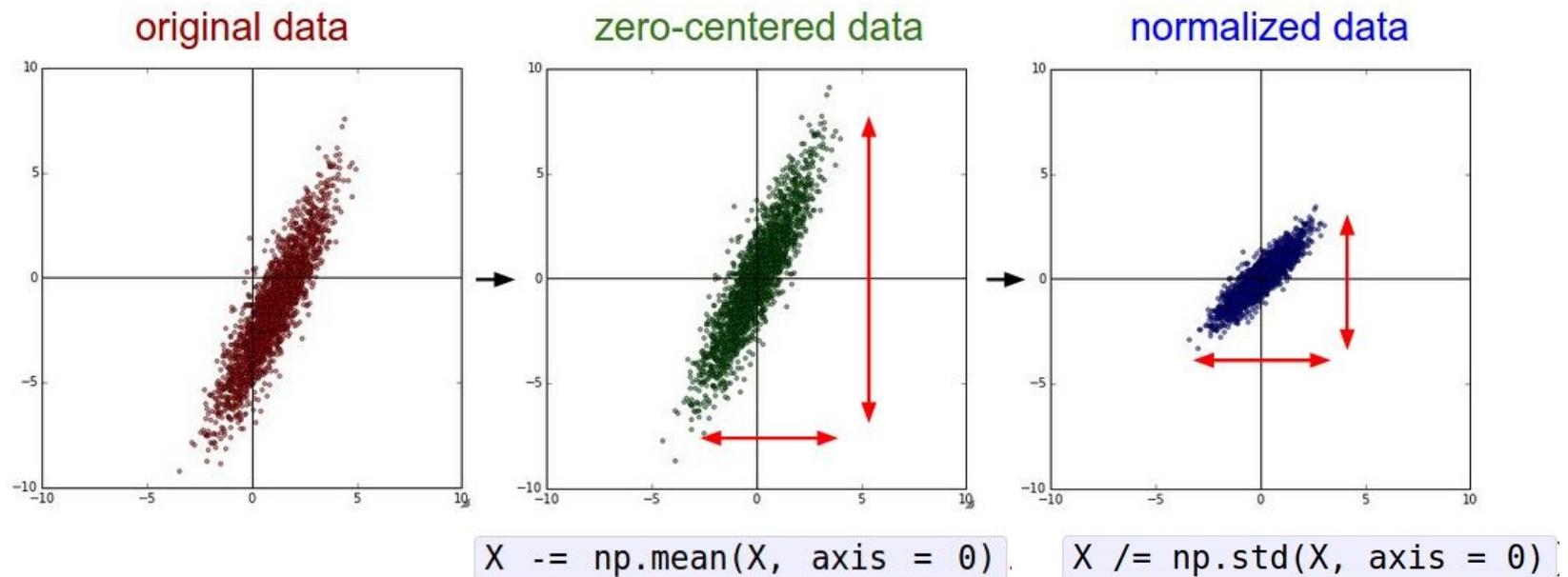
- Solves problem by adding skip connections
- Very deep: 152 layers
- No dropout
- Stride
- Batch normalization

Source: Deep Residual Learning for Image Recognition

Batch Normalization

(Assume X [NxD] is data matrix,
each example in a row)

Batch Normalization

Algorithm 2 Batch normalization [Ioffe and Szegedy, 2015]

Input: Values of x over minibatch $x_1 \ldots x_B$, where x is a certain channel in a certain feature vector

Output: Normalized, scaled and shifted values $y_1 \ldots y_B$

1: $\mu = \frac{1}{B} \sum_{b=1}^{B} x_b$

2: $\sigma^2 = \frac{1}{B} \sum_{b=1}^{B} (x_b - \mu)^2$

3: $\hat{x}_b = \frac{x_b - \mu}{\sqrt{\sigma^2 + \epsilon}}$

4: $y_b = \gamma \hat{x}_b + \beta$

- Accelerates training and makes initialization less sensitive
- Zero mean and unit variance feature vectors

BatchNorm at Test

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

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

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

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

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$
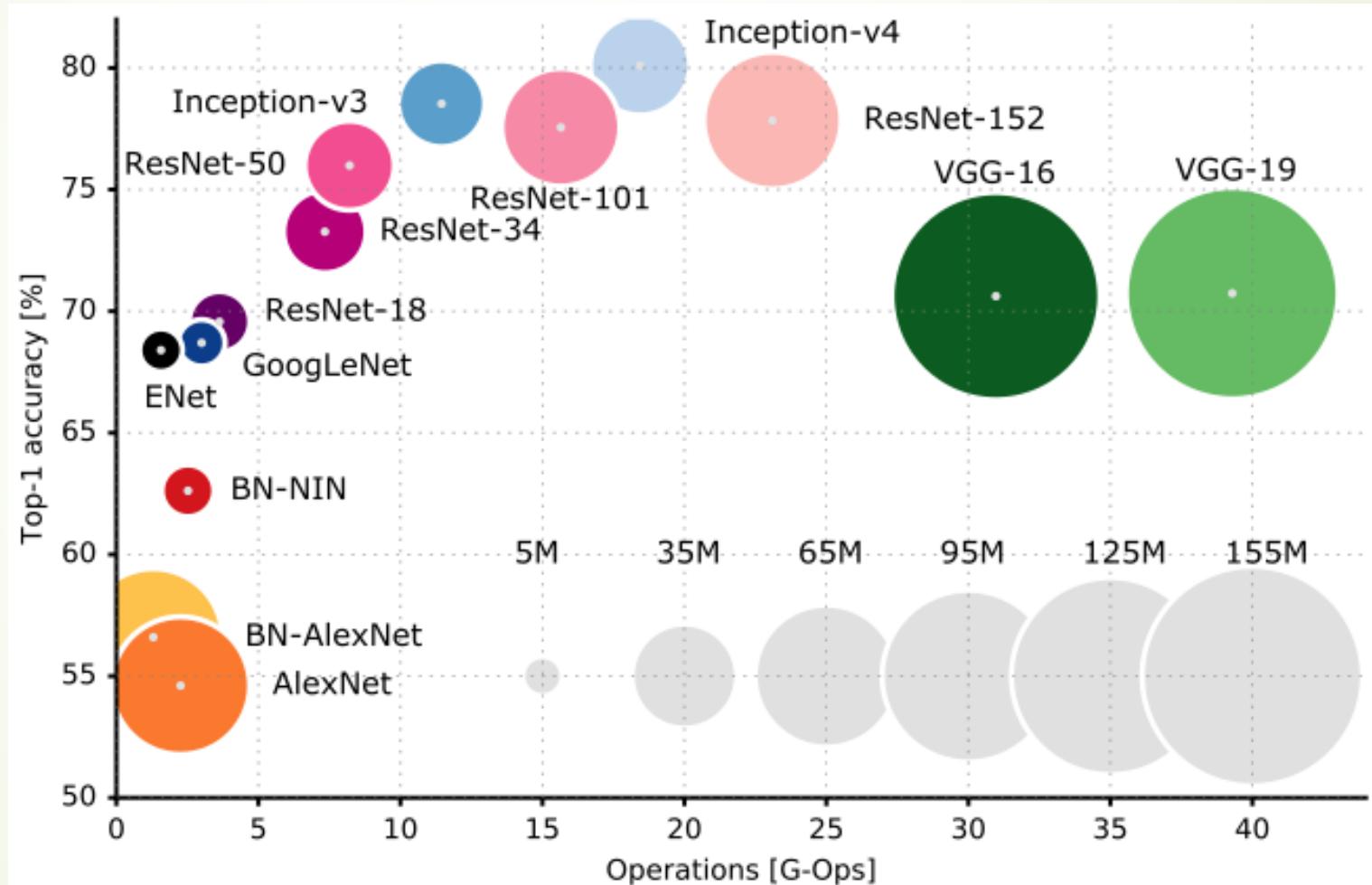
$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Note: at test time BatchNorm layer functions differently:

The mean/std are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.
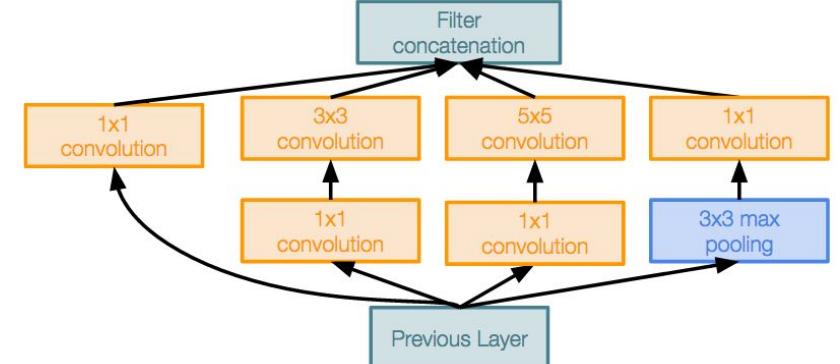
(e.g. can be estimated during training with running averages)

Complexity vs. Accuracy of Different Networks
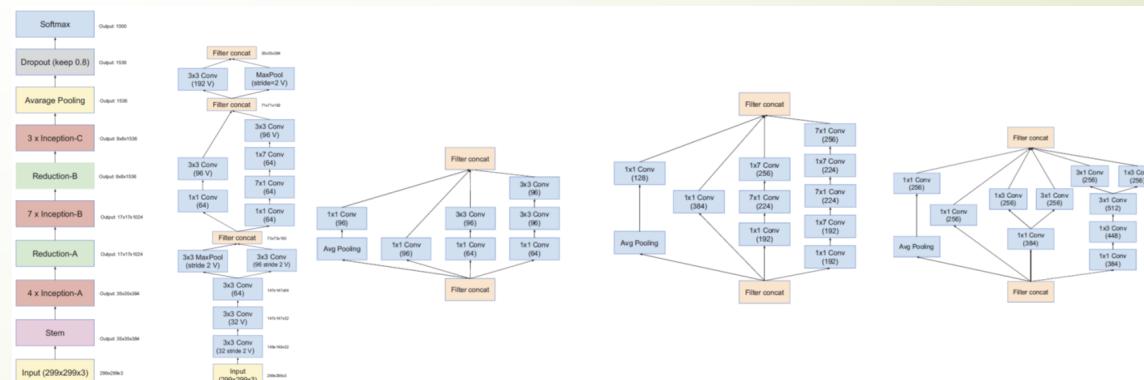
Inception-v4 = ResNet + Inception

- "**Inception**" module:
  - Introduced by *Szegedy et al., 2014* in ***GoogLeNet***
  - ILSVRC'14 classification winner (6.7% top 5 error)
- Apply parallel filter operations on the input from previous layer:
  - Dimensionality reduction (1x1 conv)
  - Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
  - Pooling operation (3x3)
- Concatenate all filter outputs together depth-wise
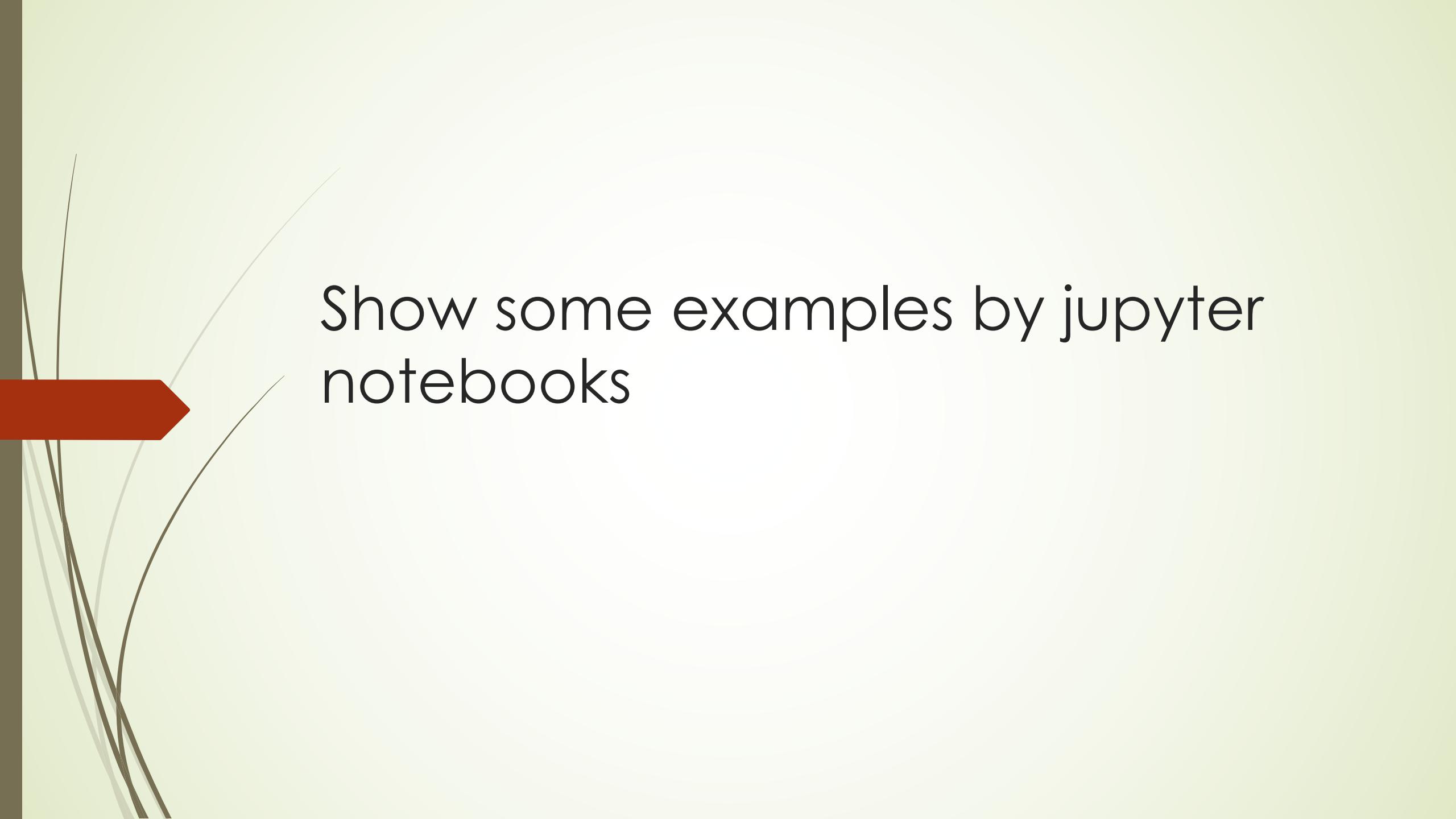
Inception module

Deep Learning Softwares

- **Pytorch** (developed by Yann LeCun and Facebook):
  - http://pytorch.org/tutorials/
- Tensorflow (developed by Google based on Caffe)
  - https://www.tensorflow.org/tutorials/
- Theano (developed by Yoshua Bengio)
  - http://deeplearning.net/software/theano/tutorial/
- **Keras (based on Tensorflow or Pytorch)**
  - https://www.manning.com/books/deep-learning-with-python?a_aid=keras&a_bid=76564dff

Show some examples by jupyter notebooks

Thank you!