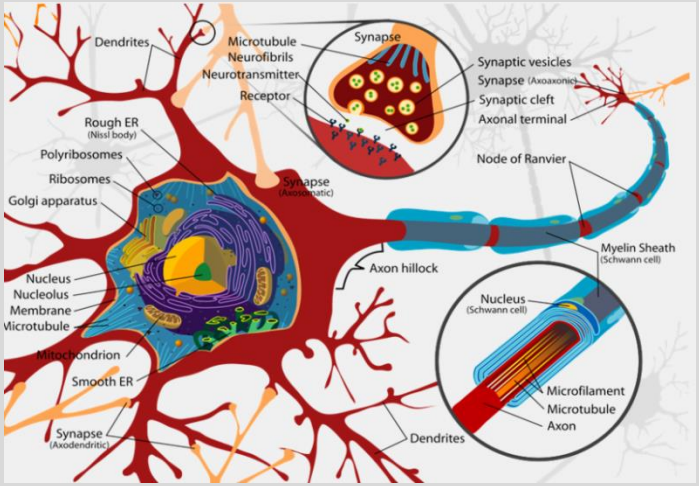
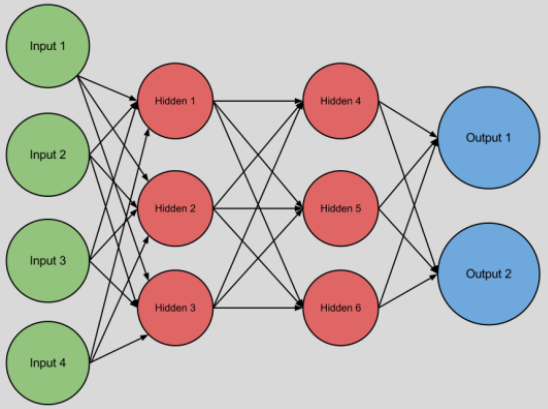
# Нейросети и глубокогое обучение (Deep Learning)

# Neural Networks

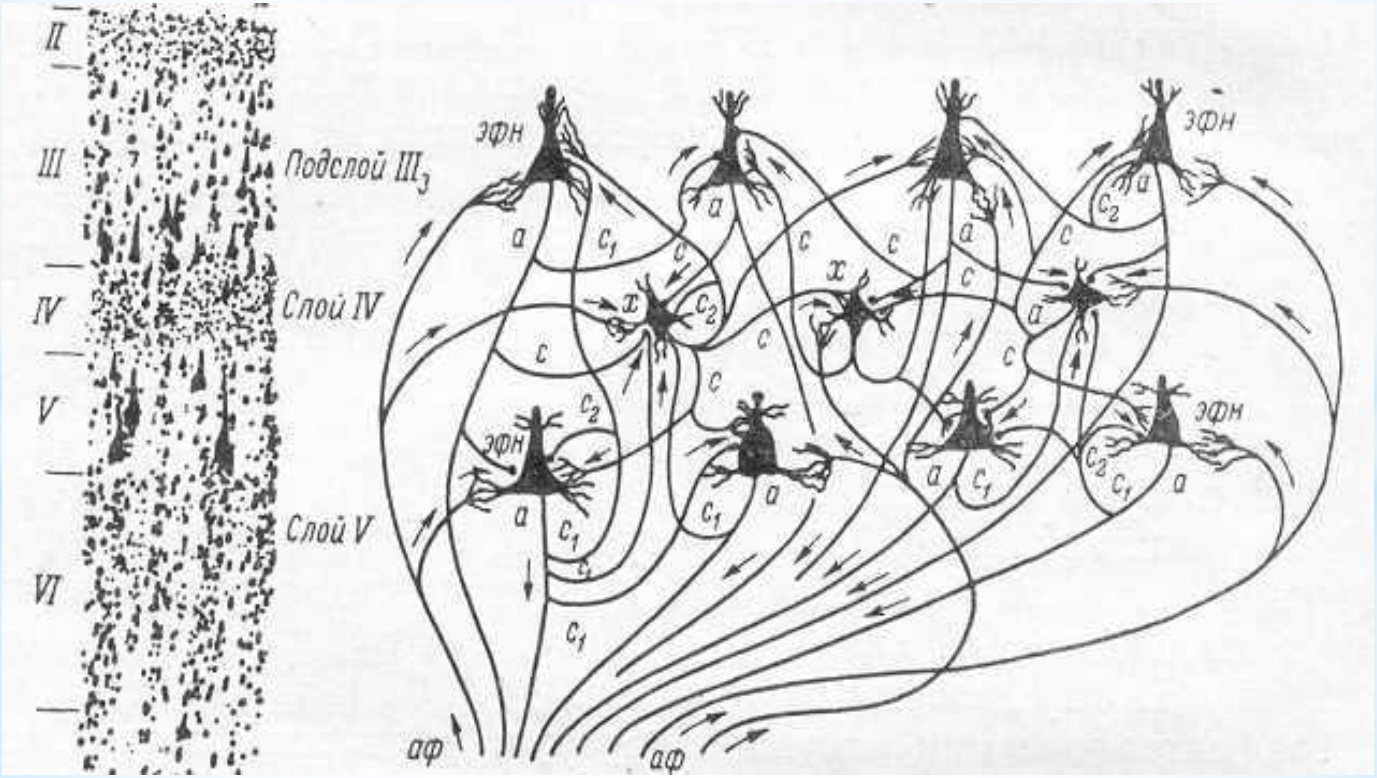
## Neuroscience/



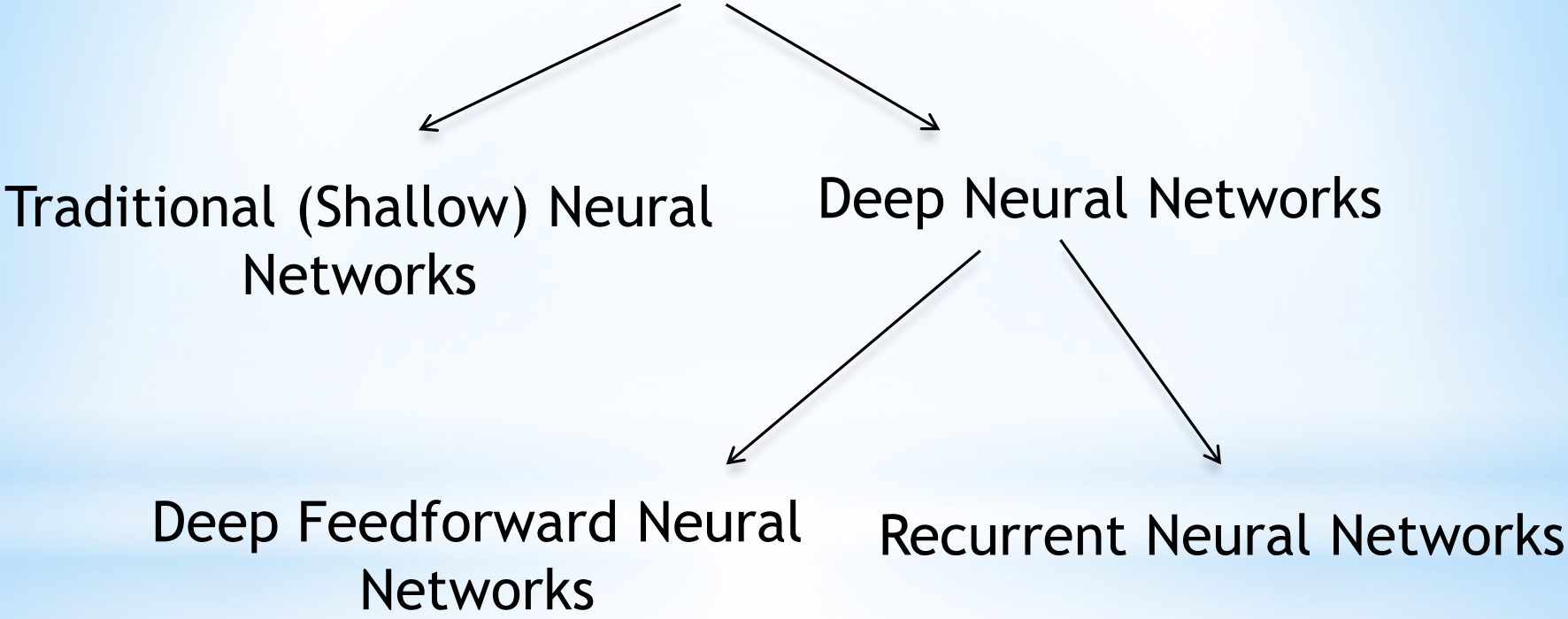

## Machine Learning



# Biological Neural Networks



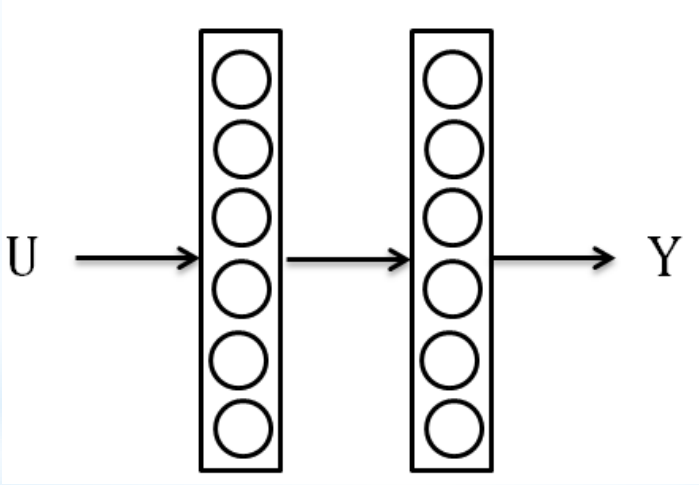
# Artificial Neural Networks




*Y. LeCun, Y. Bengio, G. Hinton. Deep learning // Nature, |Vol. 521, p. 436 – 444..*

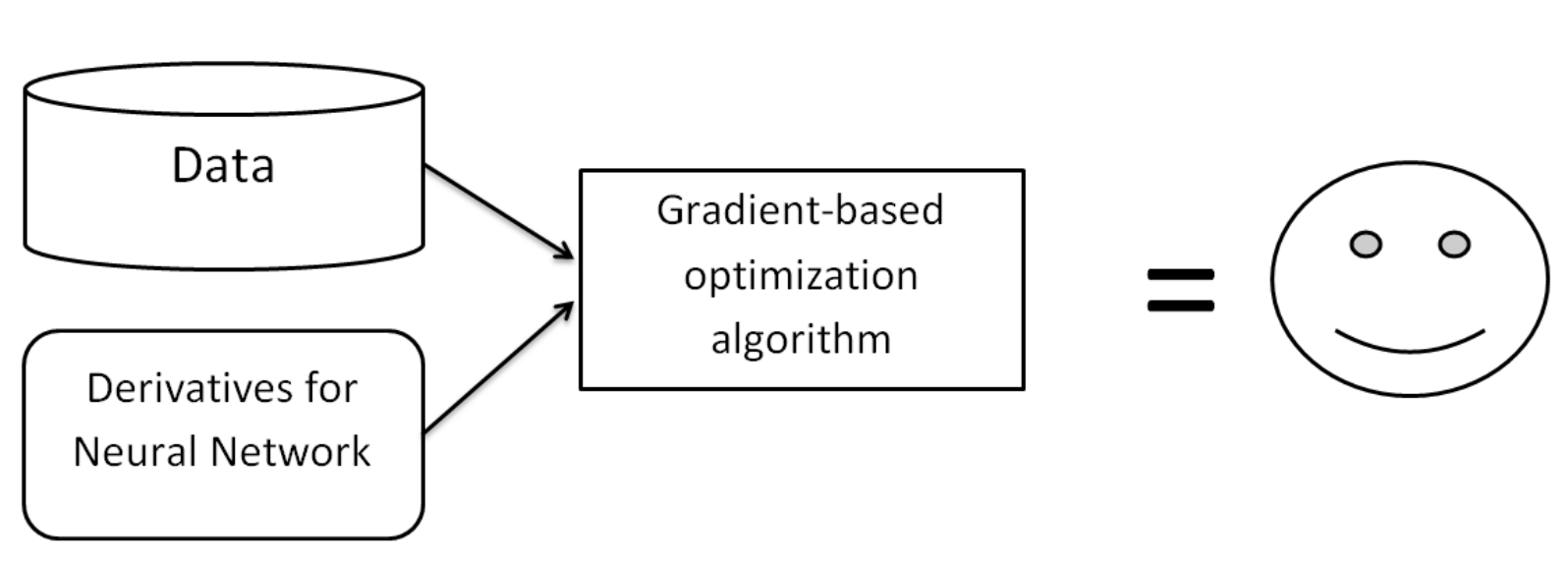
*J. Schmidhuber, Deep Learning in Neural Networks: An Overview // Technical Report IDSIA-03-14, arXiv:1404.7828 v4, 88 pages, 888 references, 8 October 2014.*

# Classic Feedforward Neural Networks (before 2006).

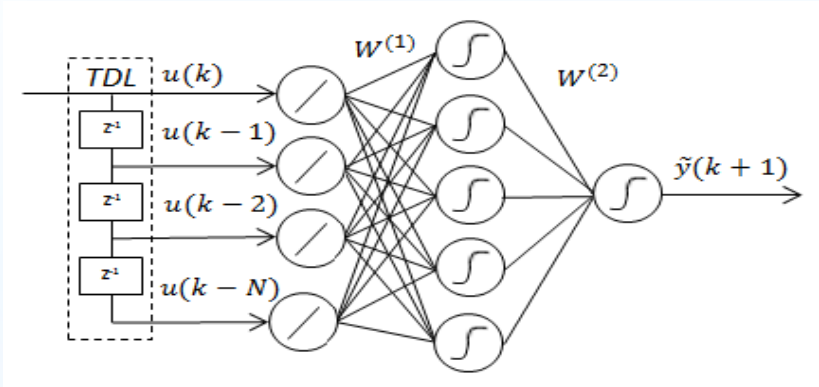



- Single hidden layer (Kolmogorov-Cybenko Universal Approximation Theorem as the main hope).
- Vanishing gradients effect prevents using more layers.
- Less than 10K free parameters.
- Feature preprocessing stage is often critical.

# Training the traditional (shallow) Neural Network: derivative + optimization



# 1) forward propagation pass

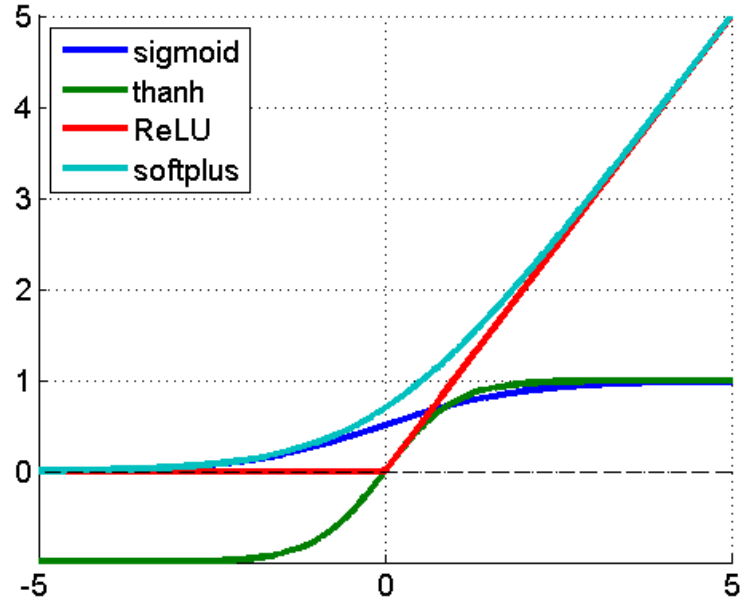



$$z_j = f(\sum_i w_{ji}^{(1)} x_i),$$

$$\tilde{y}(k+1) = g(\sum_j w_j^{(2)} z_j),$$

where $z_j$ is the postsynaptic value for the *j*-th hidden neuron, $w^{(1)}$ are the hidden layer's weights, *f()* are the hidden layer's activation functions, $w^{(2)}$ are the output layer's weights, and *g()* are the output layer's activation functions.

# Activation functions




## ReLU activation function

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$$

*Andrej Karpathy and* Fei-Fei. *CS231n: Convolutional Neural Networks for Visual Recognition* http://cs231n.github.io/convolutional-networks

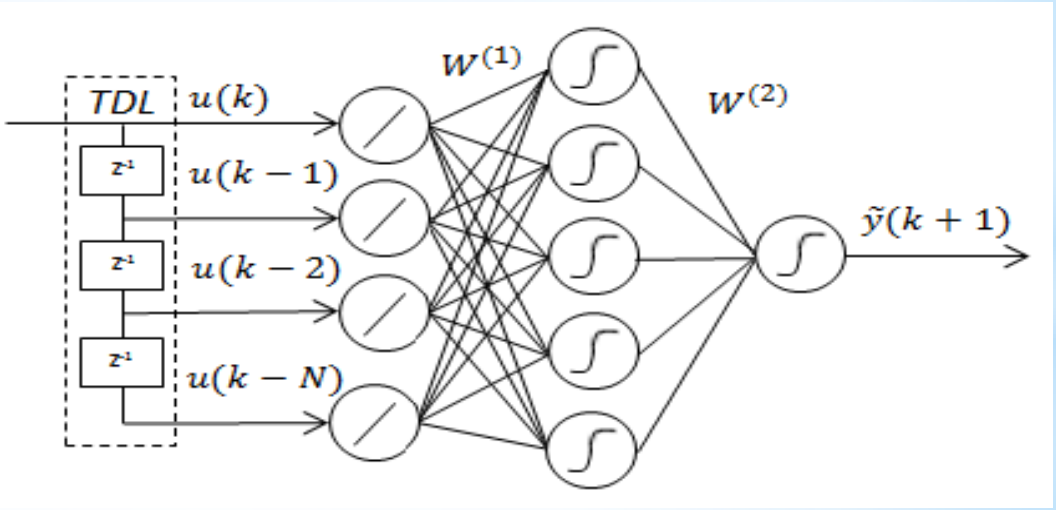
*Yoshua Bengio, Ian Goodfellow and Aaron Courville. Deep Learning // An MIT Press book in preparation* http://www-labs.iro.umontreal.ca/~bengioy/DLbook

# 2) backpropagation pass

Local gradients calculation:

$$\delta^{OUT} = t(k+1) - \tilde{y}(k+1),$$

$$\delta_j^{HID} = f'(z_j) w_j^{(2)} \delta^{OUT}.$$

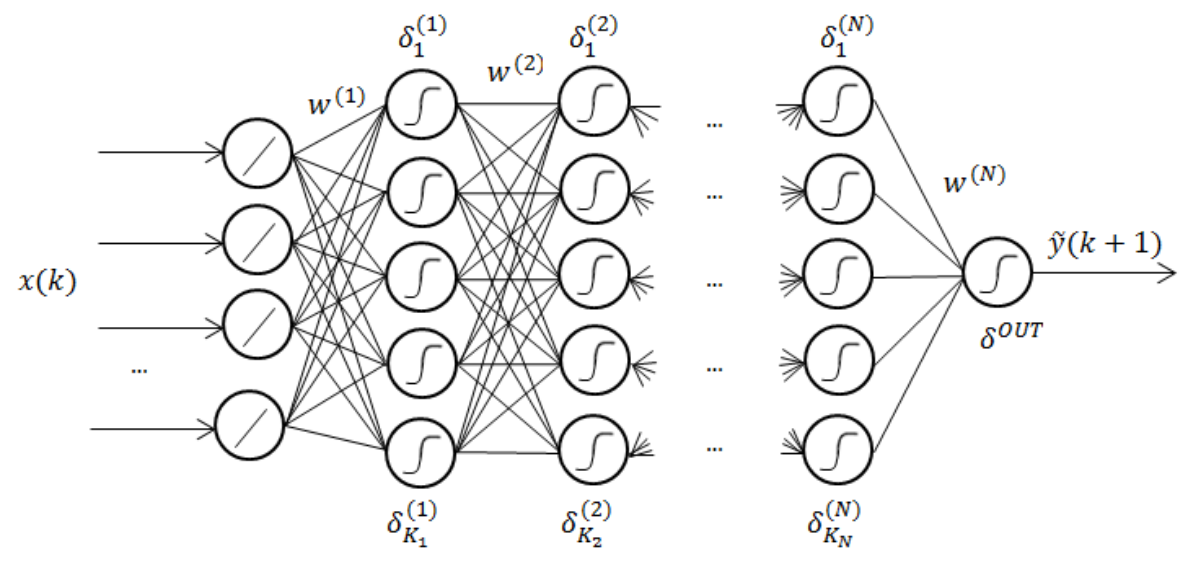



Derivatives calculation:

$$\frac{\partial E(k)}{\partial w_j^{(2)}} = \delta^{OUT} z_j,$$

$$\frac{\partial E(k)}{\partial w_{ji}^{(1)}} = \delta_j^{IN} x_i.$$

# Backpropagation mechanics



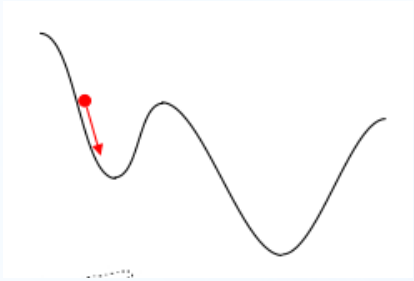

$$\delta_j^{(m)} = f_j^{(m-1)\prime} \sum_i w_{ij}^{(m)} \delta_i^{(m+1)},$$

$$\frac{\partial E(k)}{\partial w_{ji}^{(m)}} = \delta_j^{(m)} z_i^{(m-1)},$$

$$\frac{\partial E(k)}{\partial w_{ji}^{(m)}} \to 0 \quad for\ m \to 1$$

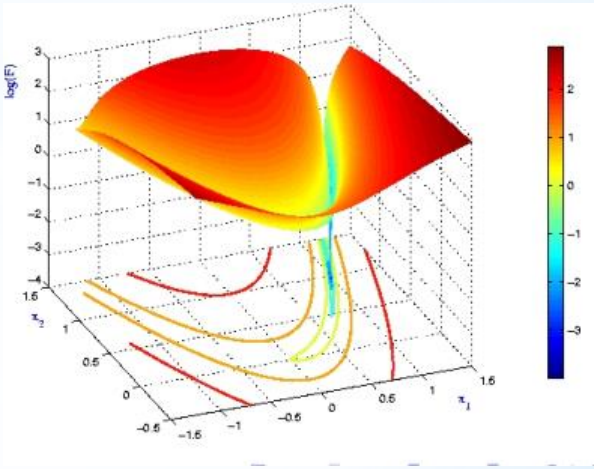
# Bad effect of vanishing (exploding) gradients: two hypotheses

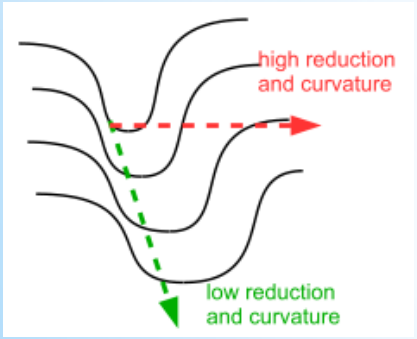
1) increased frequency and severity of bad local minima

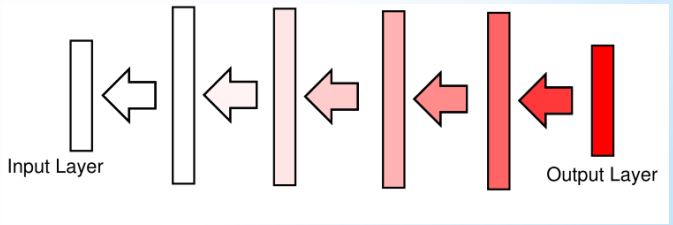


2) pathological curvature, like the type seen in the well-known Rosenbrock function:

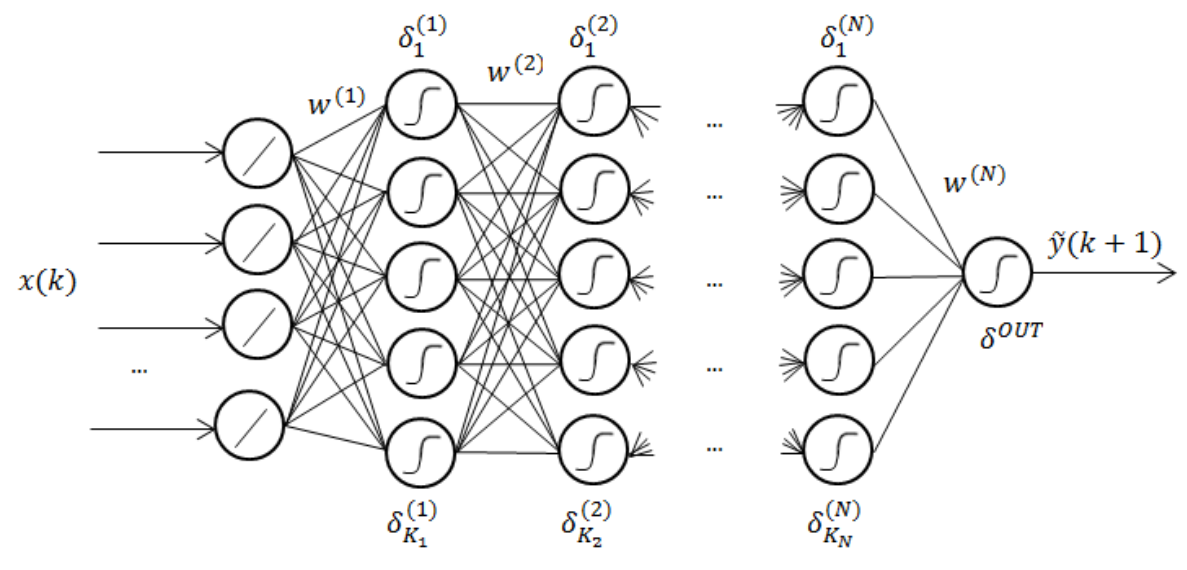
$$f(x,y) = (1-x^2) + 100(y-x^2)^2$$



# Backpropagation mechanics: again




$$\delta_j^{(m)} = f_j^{(m-1)\prime} \sum_i w_{ij}^{(m)} \delta_i^{(m+1)},$$

$$\frac{\partial E(k)}{\partial w_{ji}^{(m)}} = \delta_j^{(m)} z_i^{(m-1)},$$

$$\frac{\partial E(k)}{\partial w_{ji}^{(m)}} \to 0 \quad for\ m \to 1$$

# Backpropagation mechanics: vector form

$$\boldsymbol{\delta}(m-1) = \boldsymbol{\delta}(m)\mathbf{W}_m diag(f'(\mathbf{a}(m-1)))$$

Observations:

$\left\|\mathbf{W}_m\right\| < 1$ - robustness (weights decay)

$\left\|diag(f'(\mathbf{a}(m-1)))\right\| < 1$ - *max*(f') = ¼ for sigmoid

# Backpropagation mechanics as product of Jacobians

Jacobian of m-th layer:

$$\mathbf{J}(n) = \mathbf{W}_n \, diag(\, f'(\mathbf{a}(n-1))).$$

Jacobian of (n-h)-th layer:

$$\boldsymbol{\delta}(n-1) = \boldsymbol{\delta}(n)\mathbf{J}(n),$$

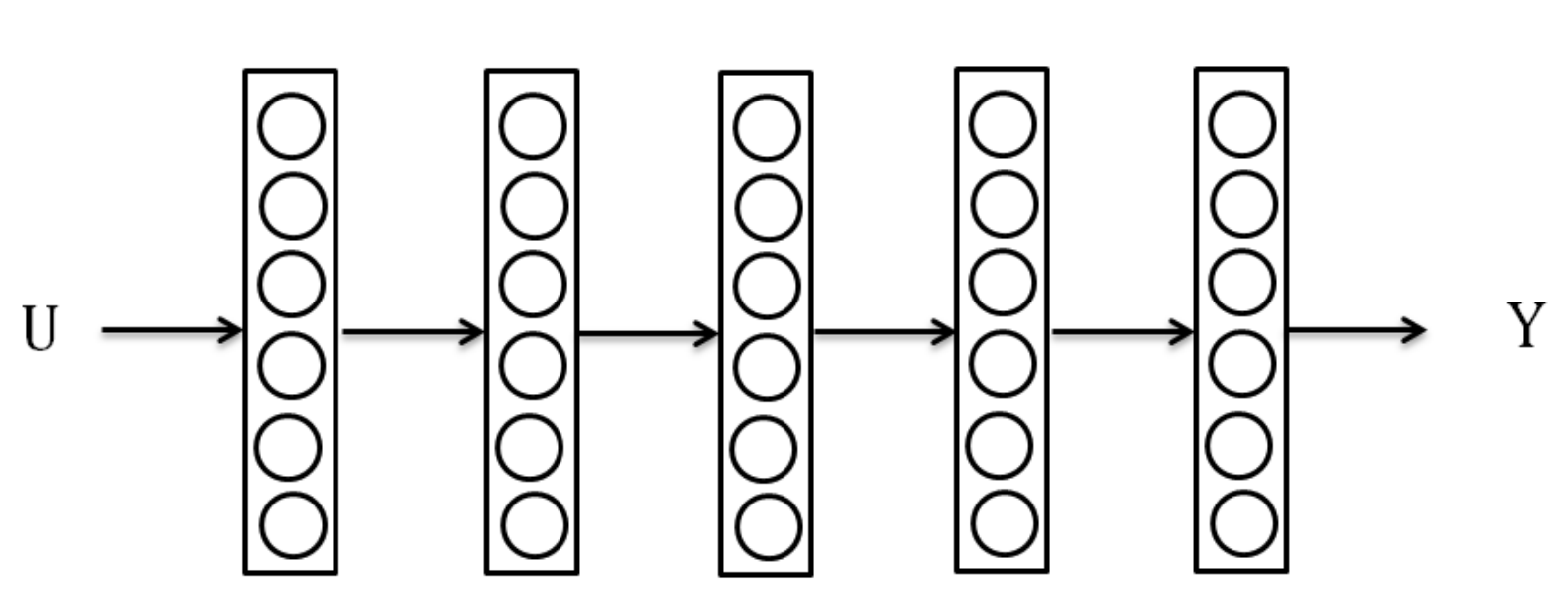
$$\boldsymbol{\delta}(n-2) = \boldsymbol{\delta}(n)\mathbf{J}(n)\mathbf{J}(n-1),$$

$$\boldsymbol{\delta}(n-h) = \boldsymbol{\delta}(n)\mathbf{J}(n)\mathbf{J}(n-1)\ldots\mathbf{J}(n-h+1).$$

Is it possible to make the Jacobians the orthogonal matrices?

*A.M. Saxe, J.L. McClelland, S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks // Proceedings of International Conference on Learning Representations (ICLR) 2014, Banff, Canada, 14-16 April 2014*

# Deep Feedforward Neural Networks

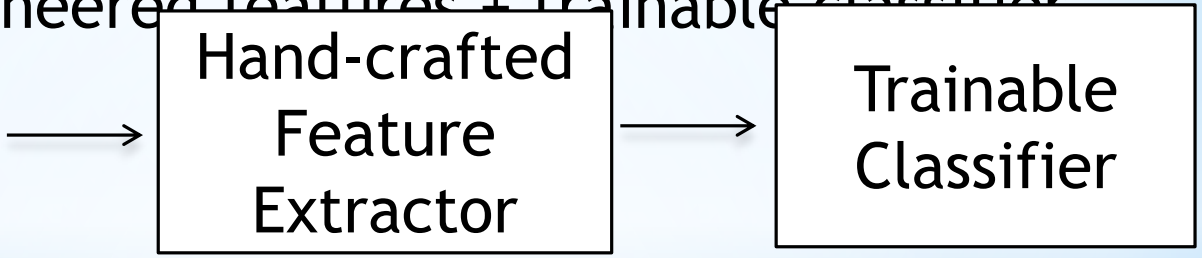



- 2-stage training process: i) unsupervised pre-training; ii) fine tuning (vanishing gradients problem is beaten!).
- Number of hidden layers > 1 (usually 6-9).
- No (or less) feature preprocessing stage.
- 100K – 100M free parameters.

# Deep Learning = Learning of Representations (Features)

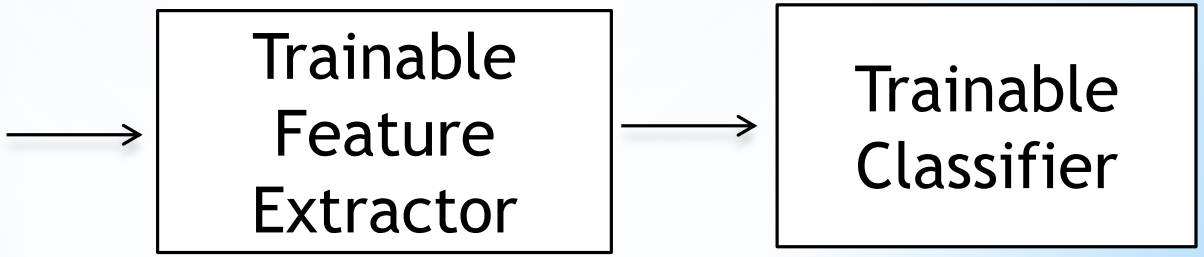
The traditional model of pattern recognition (since the late 50's):

neered features + trainable classifier




End-to-end learning / Feature learning / Deep learning:

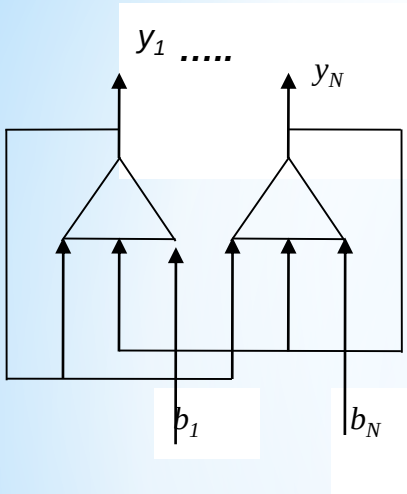
trainable features + trainable classifier



# Unsupervised pre-training

- Restricted Boltzmann Machines (RBM)
- Denoising Autoencoders
- Sparse Autoencoders

# Simple Neural associative memory




Hopfield's network

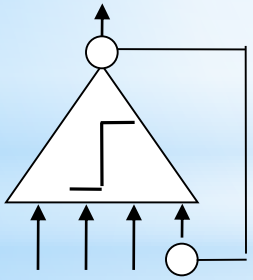
$$W_{ij} = \frac{1}{N}\sum_{m=1}^{M} y_i^m y_j^m$$

Memory capacity: M< 0.14N

Pseudo-inverse learning:

Memory capacity: M< 0,25N

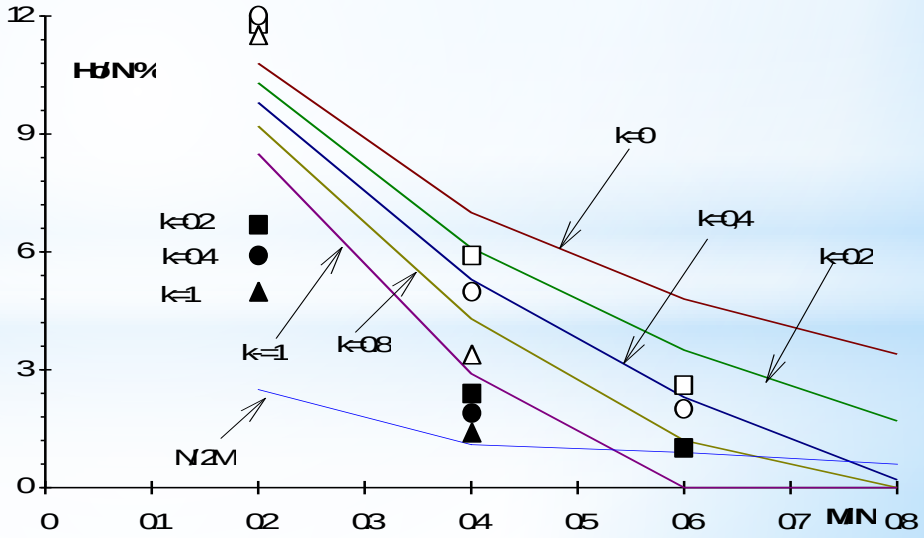
Synaptic matrix desaturation:



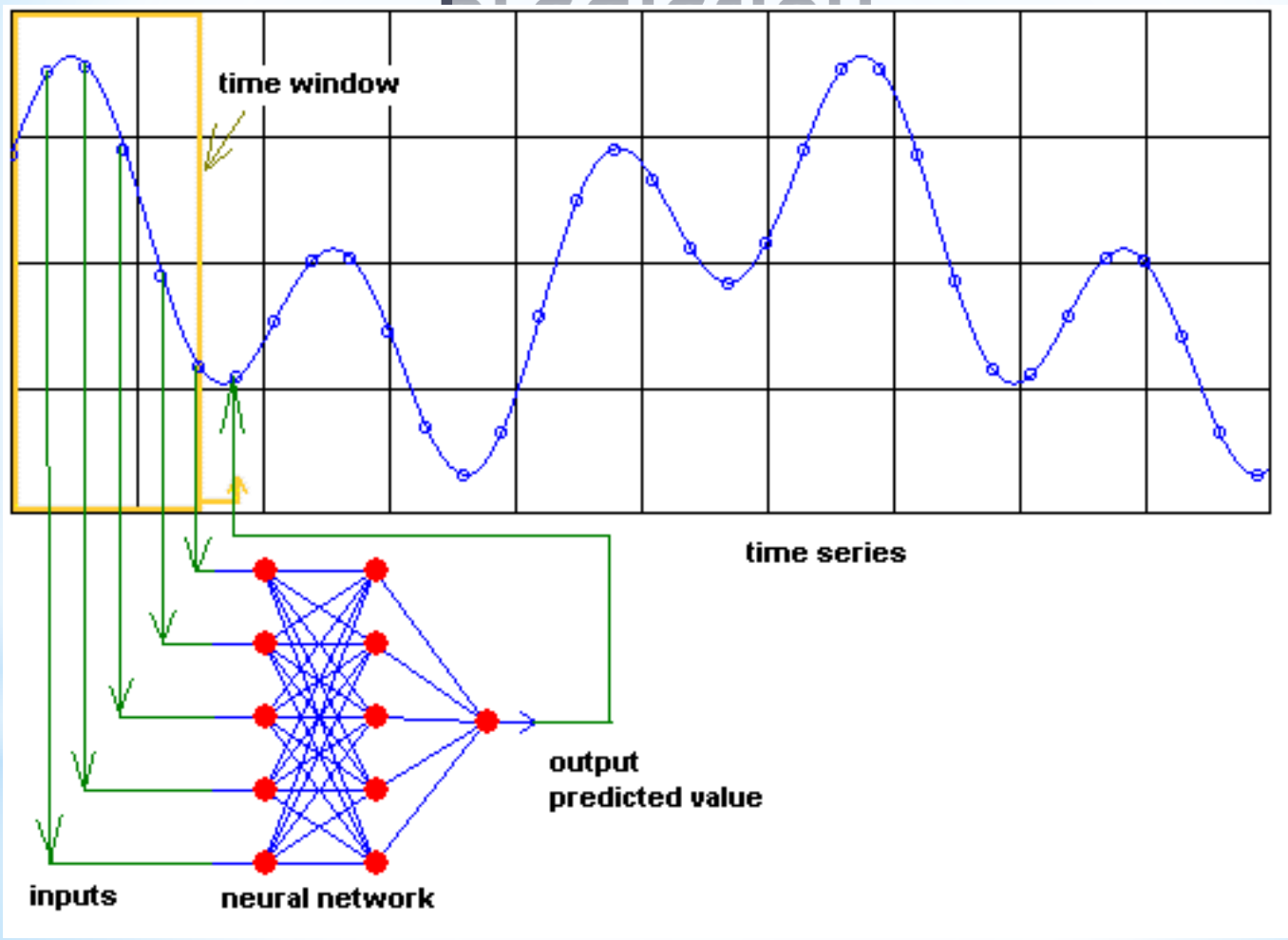
$$W_{ij} = \begin{cases} W_{ij}, i \neq j \\ kW_{ij}, i = j \end{cases}$$

$0 < k < 1$

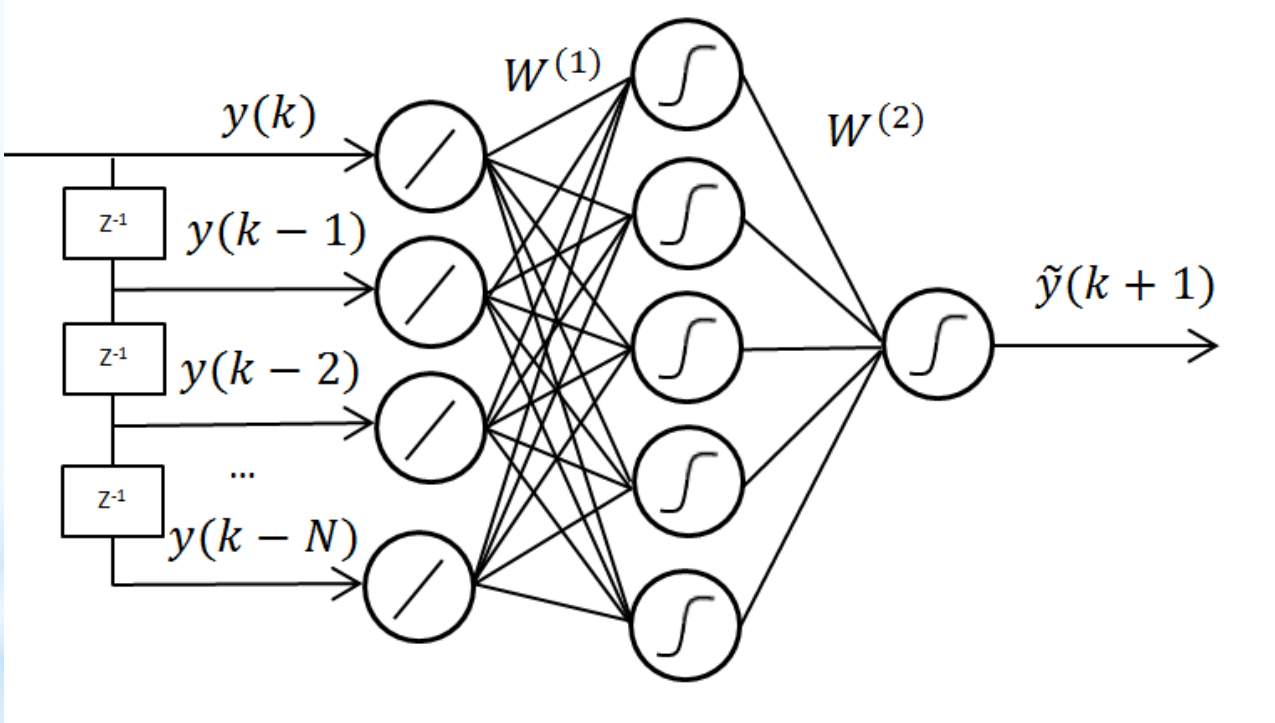
Memory capacity: M<0,7N



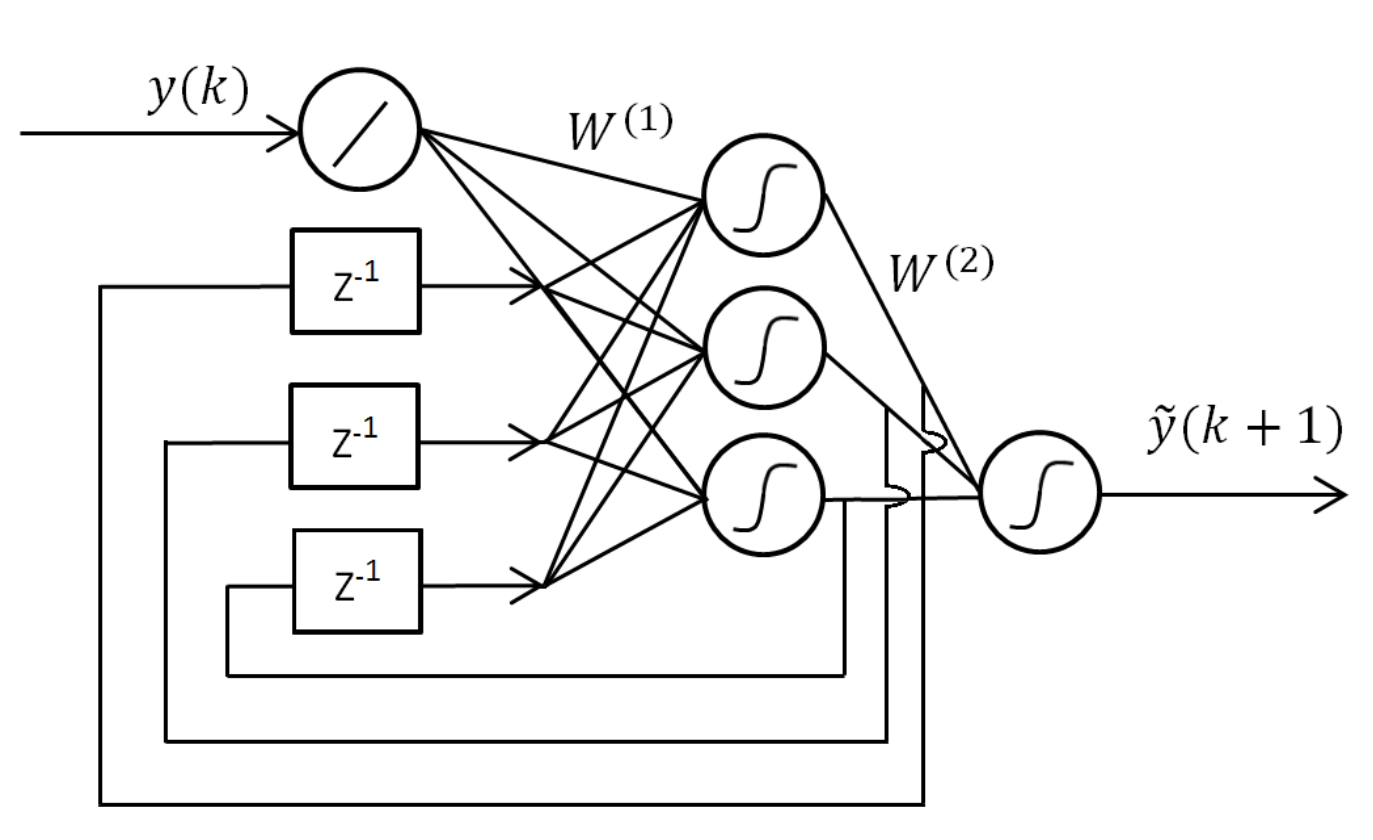
# Simple MLP setup for time series prediction



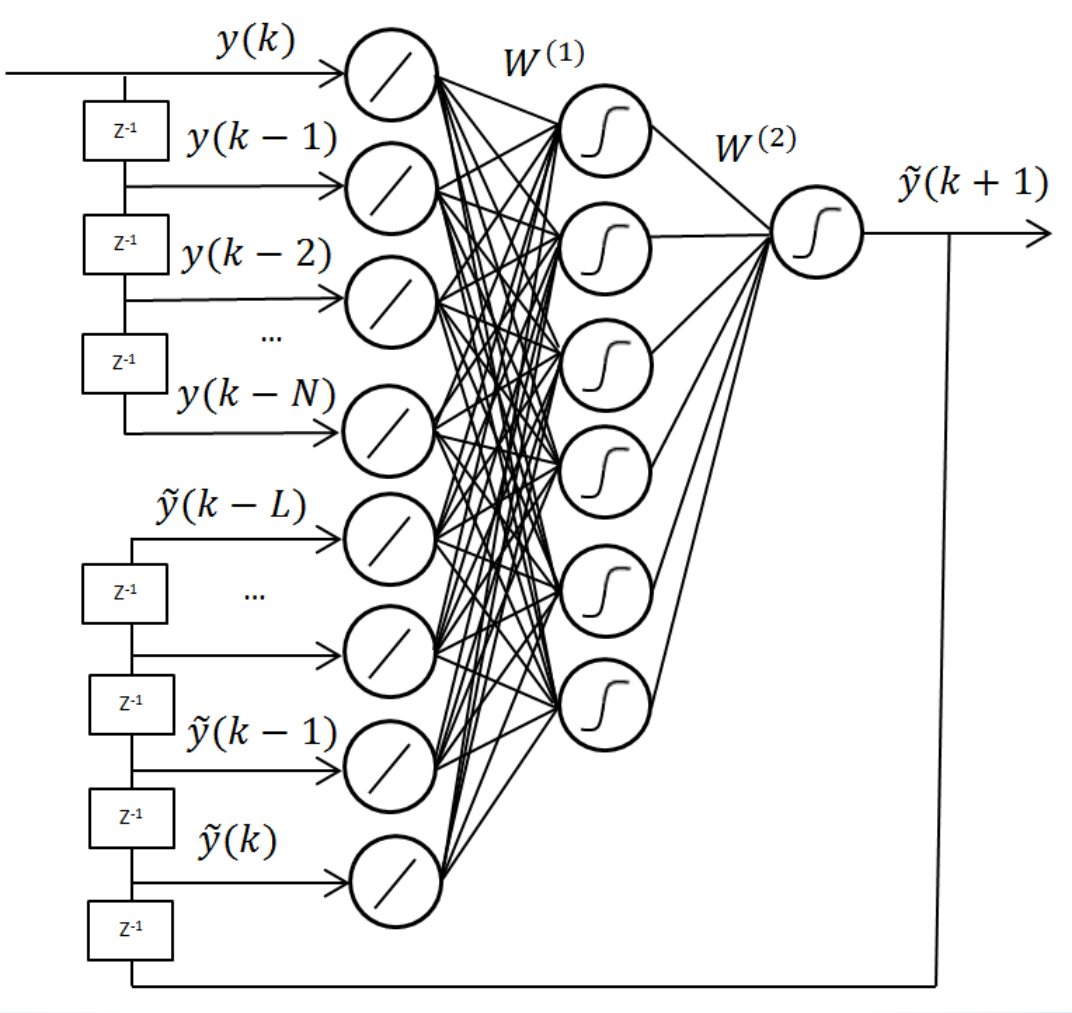
# Multilayer perceptron with time-delay line (TDNN)



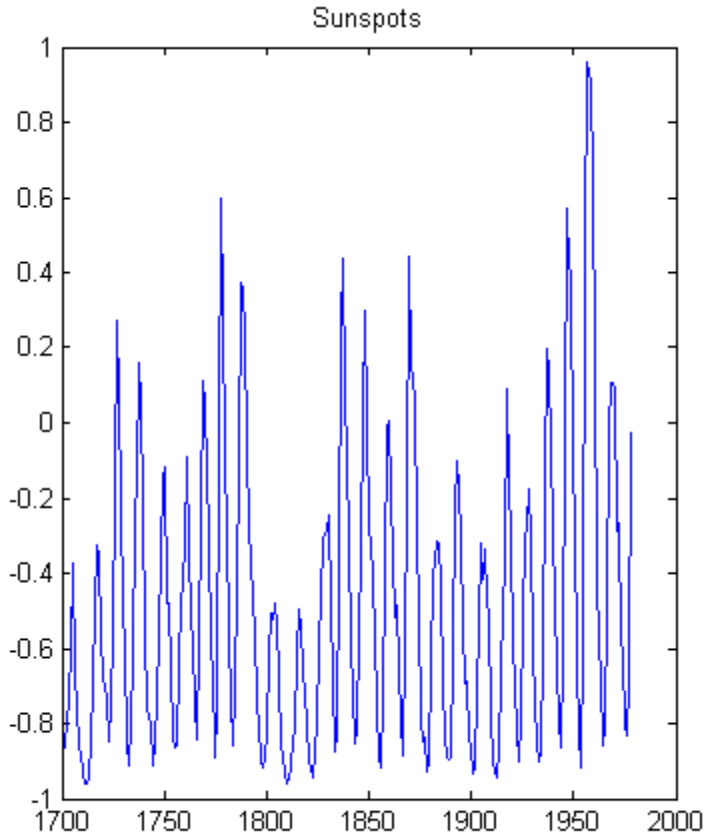
# Simple Recurrent Network (SRN)

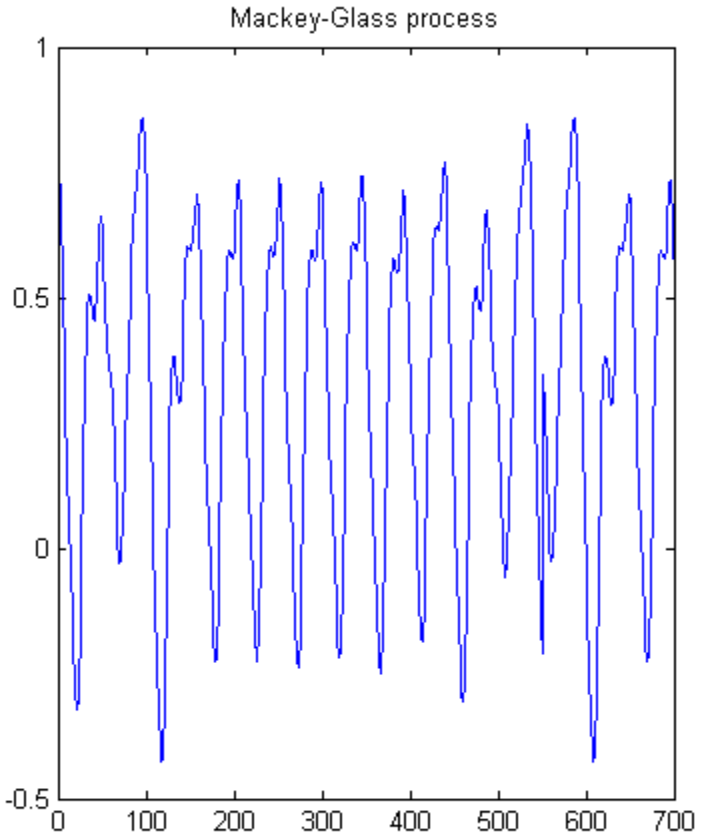


# Nonlinear Autoregressive Neural Networks with eXternal input (NARX)



# Training data: Sunspots and Mackey-Glass process



# *Training Neuroemulators using Extended Kalman Filter (Prokhorov, 2008)

Problem of estimation of some unknown "true" state of "ideal" neural network that provides "zero" residual. Network's weights w(k) is considered as states and training error e(k) as residual.

1) Next value $\tilde{y}(k)$ is calculated ("forward pass").

2) Backward pass is made: derivatives $\frac{\partial \tilde{y}}{\partial w_i}$, are calculated using backpropagation method. Observation matrix H(k) is to be filled.

$$H(k) = \left[ \frac{\partial \tilde{y}}{\partial w_1} \quad \frac{\partial \tilde{y}}{\partial w_2} \quad \dots \quad \frac{\partial \tilde{y}}{\partial w_N} \right]^T.$$

3) Current residual e(k) is calculated, residuals matrix is filled E(k):

$$E(k) = [e(k)].$$

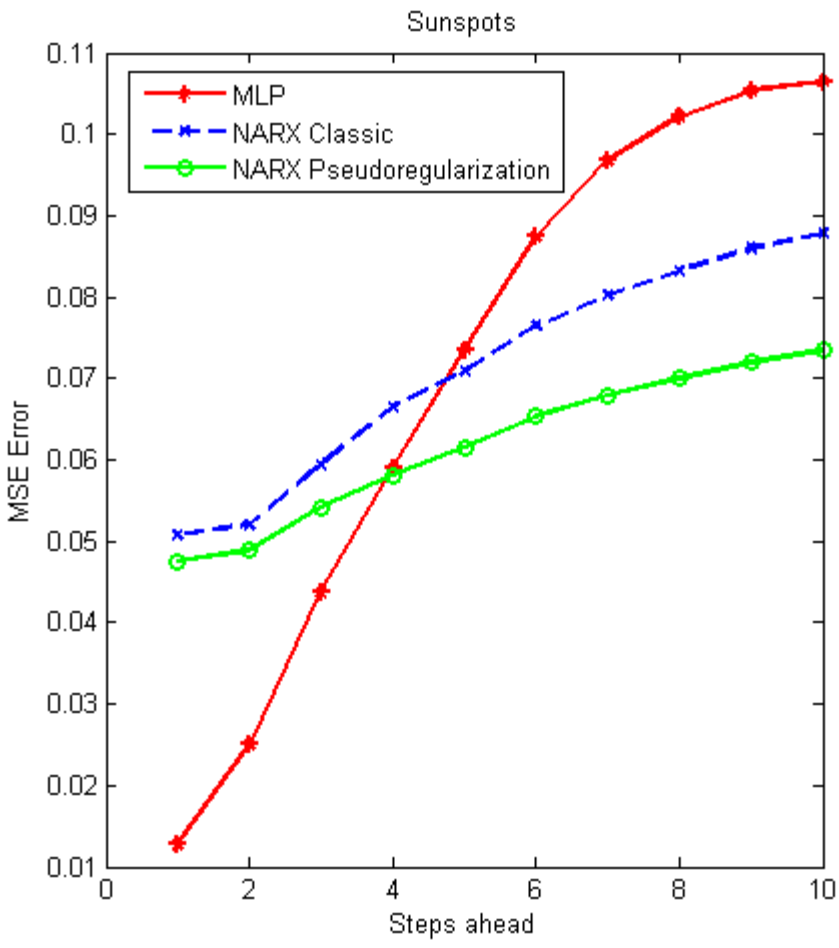
4) New values of neural network's weights are calculated:

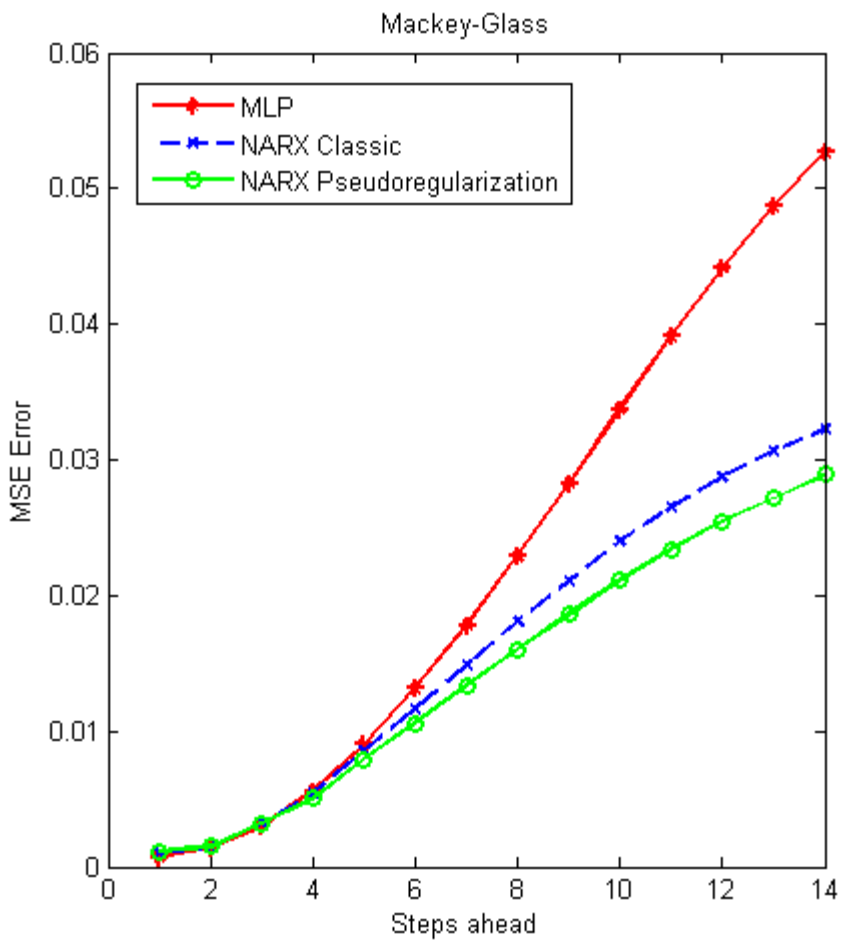
$$K(k) = P(k)H(k)^T [H(k)P(k)H(k)^T + R]^{-1},$$
$$P(k+1) = P(k) - K(k)H(k)P(k) + Q,$$
$$w(k+1) = w(k) + K(k)e(k).$$



5) Steps 1) – 4) are to be performed until Neural Network will be trained.

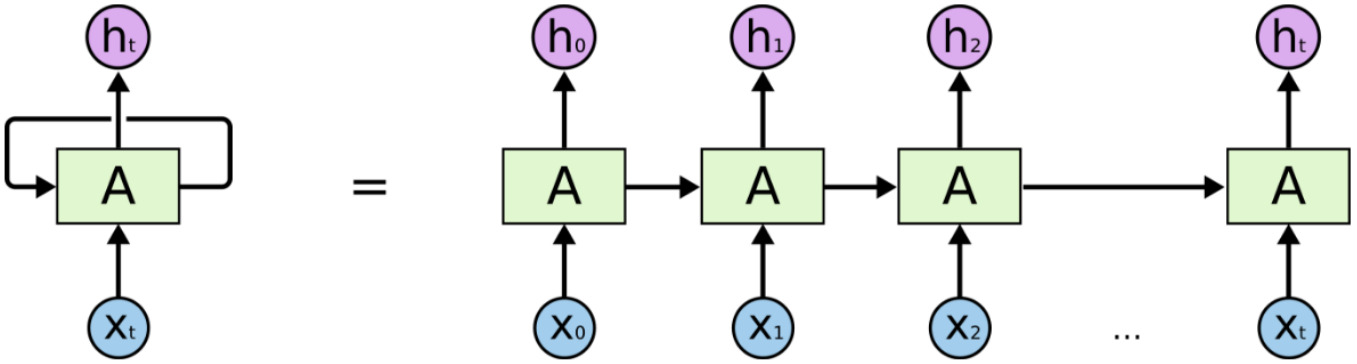
# Experiments: Sunspots and Mackey-Glass process prediction



- I. Vanilla RNN

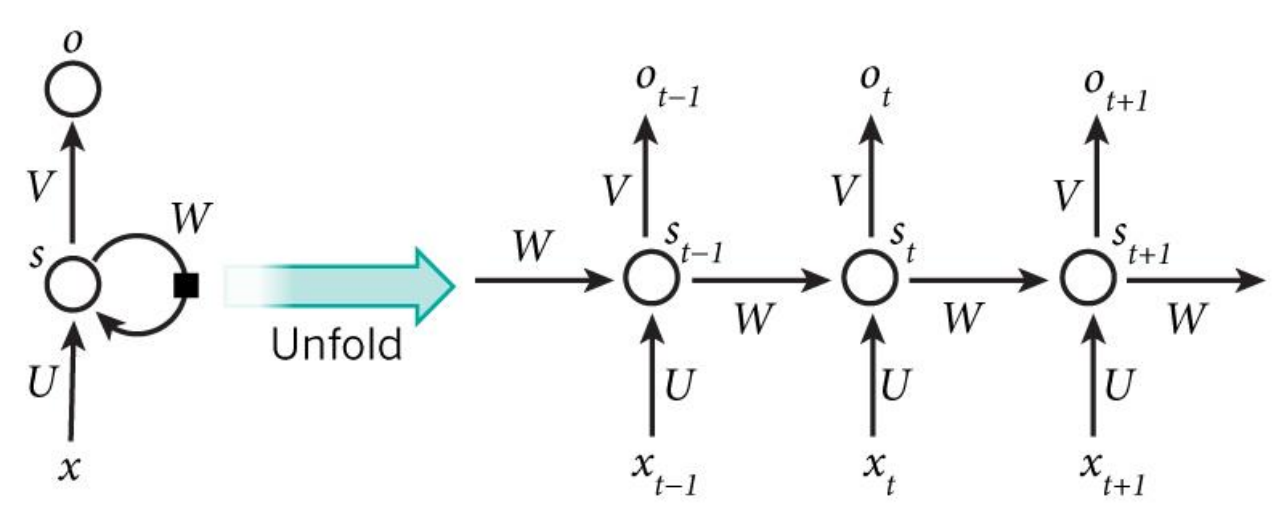
## GREAT Intro: Understanding LSTM Networks




An unrolled recurrent neural network.

In theory, RNNs are absolutely capable of handling such “long-term dependencies.” A human could carefully pick parameters for them to solve toy problems of this form. Sadly, in practice, RNNs don’t seem to be able to learn them.

- I. Vanilla RNN




$$s_t = \tanh(Ux_t + Ws_{t-1})$$

$$\hat{y}_t = \mathrm{softmax}(Vs_t)$$

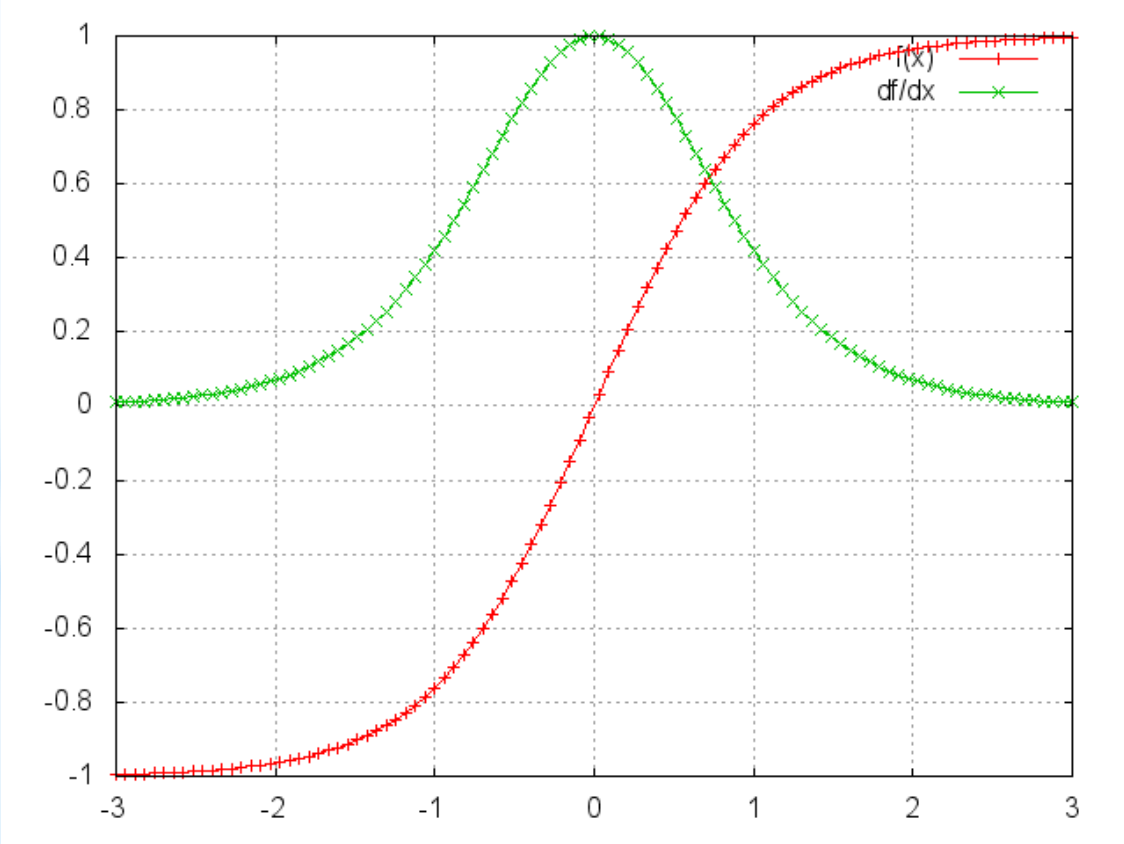
WILDML has a series of articles to introduce RNN (4 articles, 2 GitHub repos).

# • I. Vanilla RNN

- Gradient Failure Problem
  (vanishing or explosion)

RNNs tend to be very deep

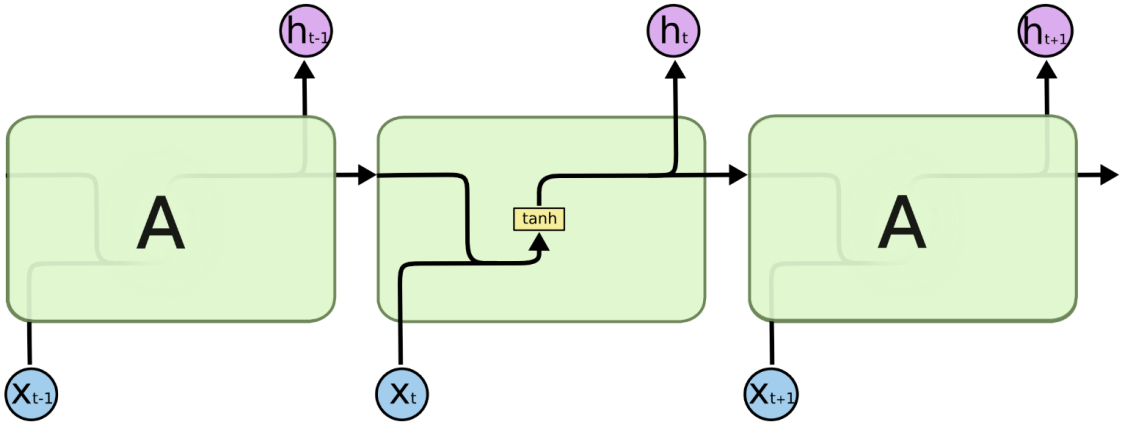
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left( \prod_{j=k+1}^{3} \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$$



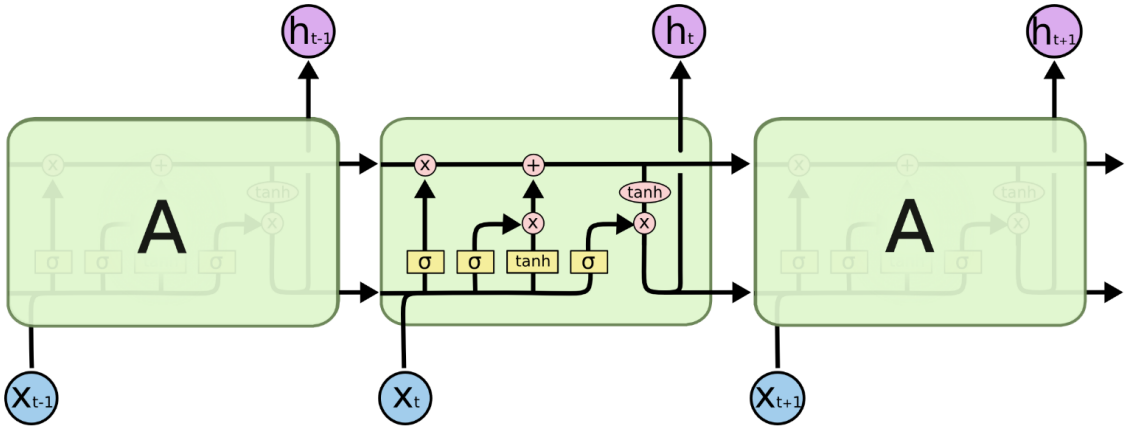

*tanh and derivative. Source:*

# • II. LSTM

- Differences of LSTM and Vanilla RNN



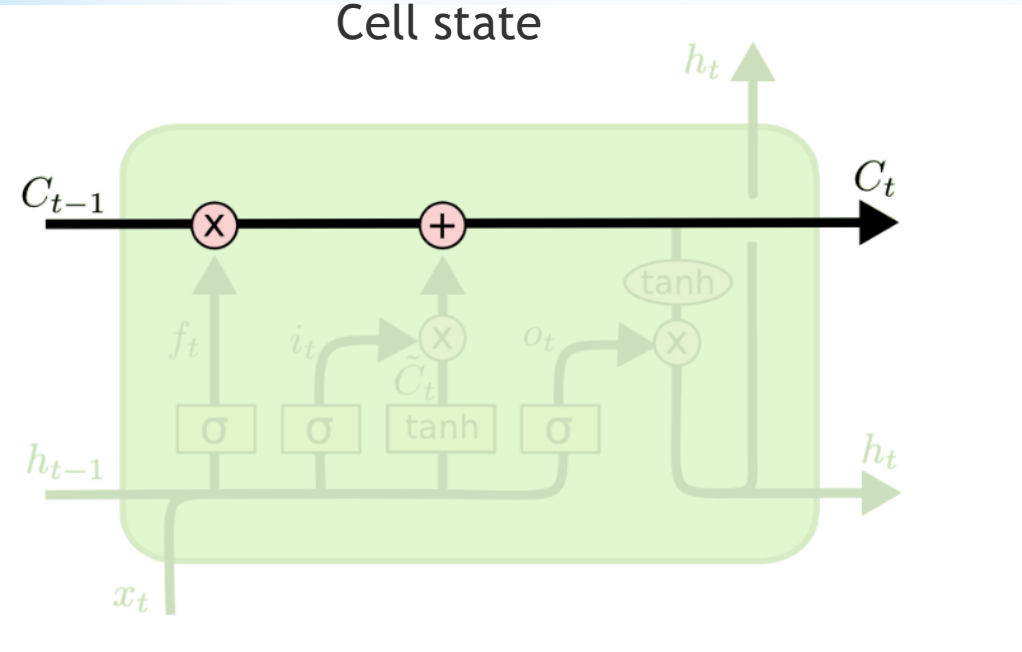

The repeating module in a standard RNN contains a single layer.

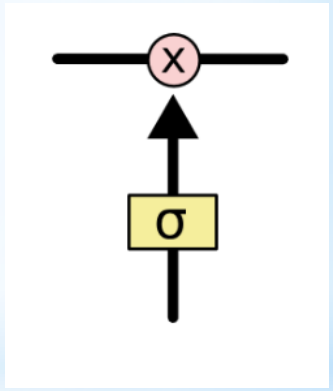



The repeating module in an LSTM contains four interacting layers.

# II. LSTM

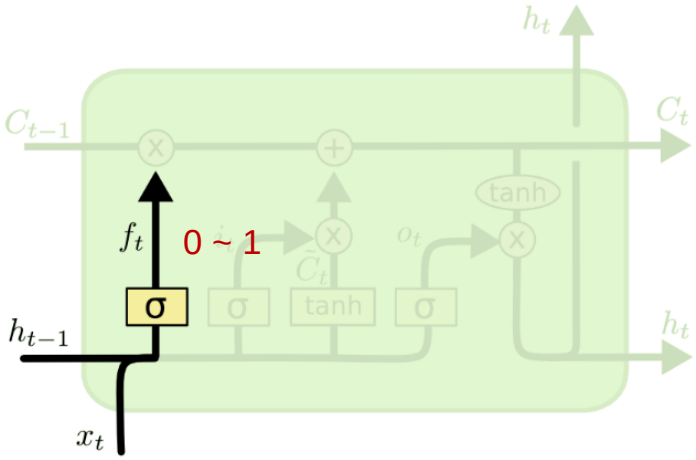
- Core Idea Behind LSTMs



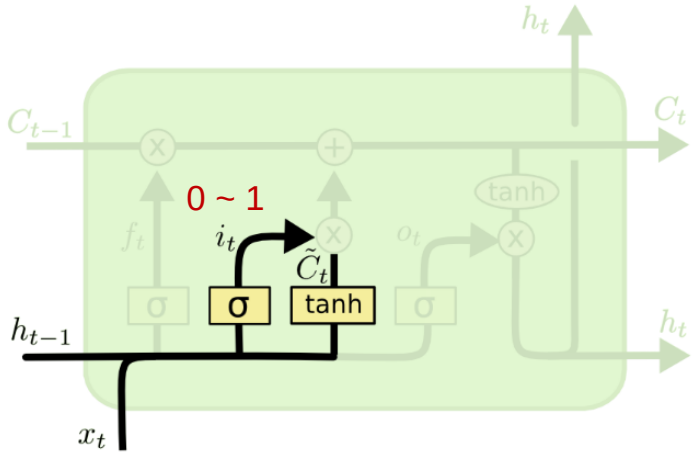



# II. LSTM

- Step-by-Step Walk Through




$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

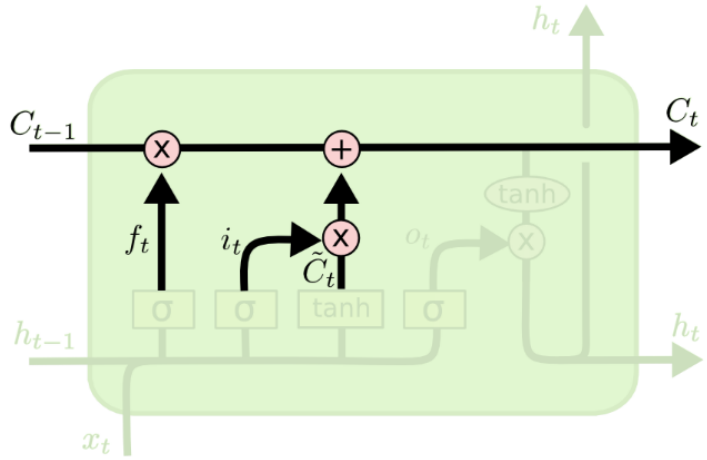



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$

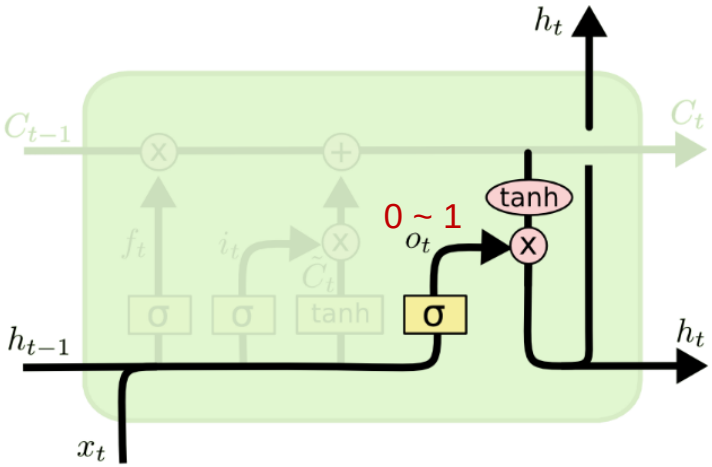
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

# II. LSTM

- Step-by-Step Walk Through




$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

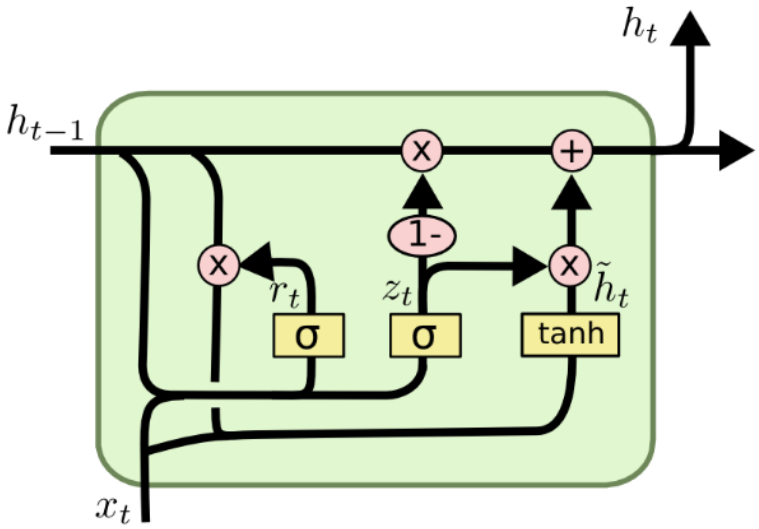



$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right)$$

$$h_t = o_t * \tanh\left(C_t\right)$$

# • III. GRU and other structures

- Gated Recurrent Unit (GRU)




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

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

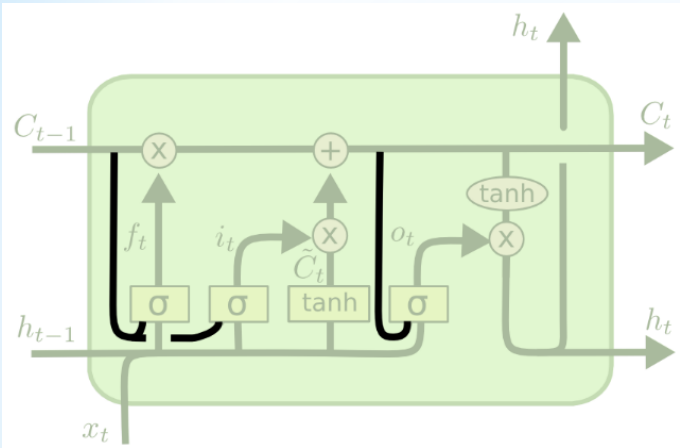
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

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

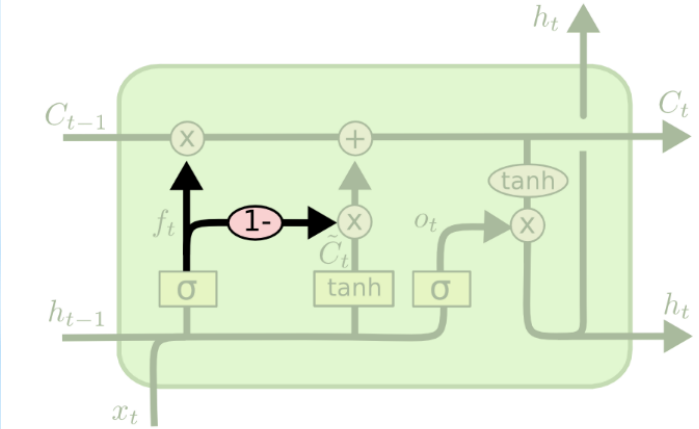
- Combines the forget and input gates into a single “update gate.”
- Merges the cell state and hidden state
- Other changes

# III. GRU and other structures

- Variants on Long Short Term Memory




$$f_t = \sigma\left(W_f \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_f\right)$$
$$i_t = \sigma\left(W_i \cdot [\boldsymbol{C_{t-1}}, h_{t-1}, x_t] \; + \; b_i\right)$$
$$o_t = \sigma\left(W_o \cdot [\boldsymbol{C_t}, h_{t-1}, x_t] \; + \; b_o\right)$$



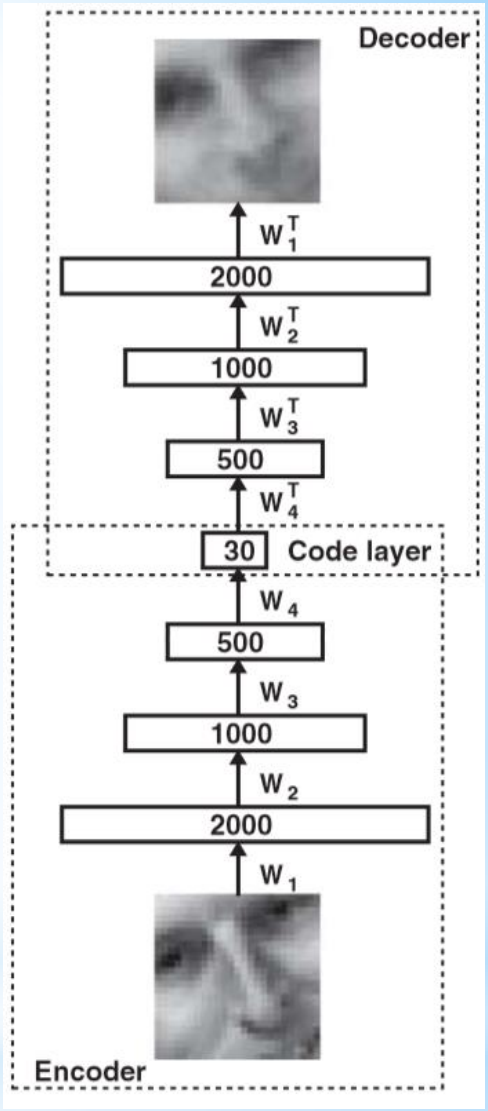

$$C_t = f_t * C_{t-1} + (\mathbf{1} - \boldsymbol{f_t}) * \tilde{C}_t$$

Greff, et al. (2015) do a nice comparison of popular variants, finding that they're all about the same.

# Bibliography


- [1] Understanding LSTM Networks
- [2] Back Propagation Through Time and Vanishing Gradients

# *Dimensionality reduction




* Use a stacked RBM as deep auto-encoder

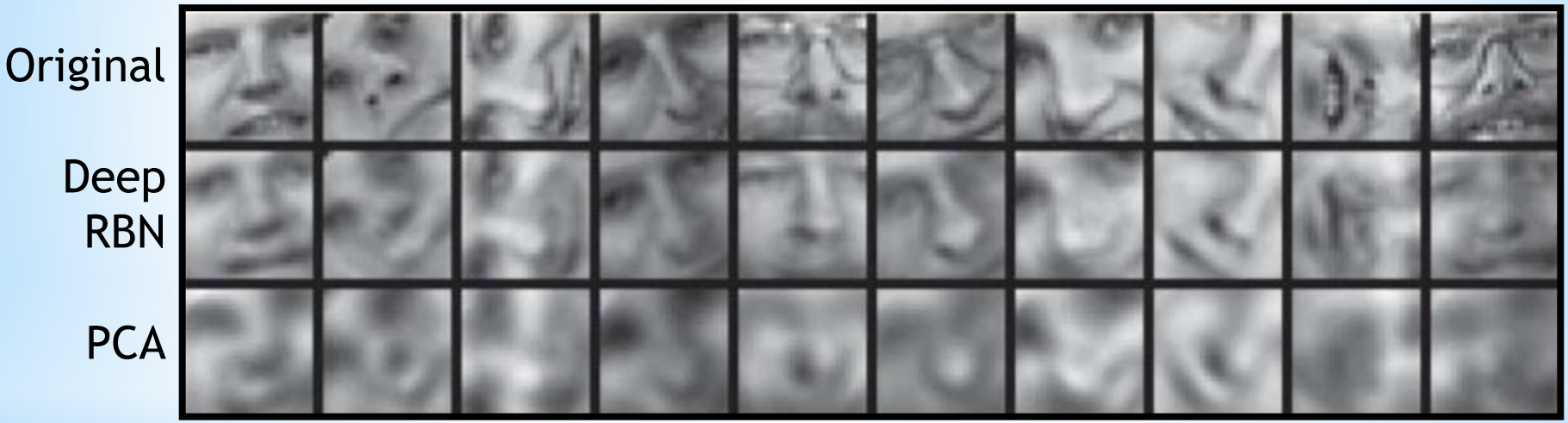
1. Train RBM with images as input & output
2. Limit one layer to few dimensions

→ Information has to pass through middle layer

*G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks // Science 313 (2006), p. 504 – 507.*

# *Dimensionality reduction

Olivetti face data, 25x25 pixel images reconstructed from 30 dimensions (625 → 30)




*G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks // Science 313 (2006), p. 504 – 507.*

Unlabeled data is readily availabl

Example: Images from the web

1. Download 10’000’000 images
2. Train a 9-layer DNN
3. Concepts are formed by DNN

*Unlabele

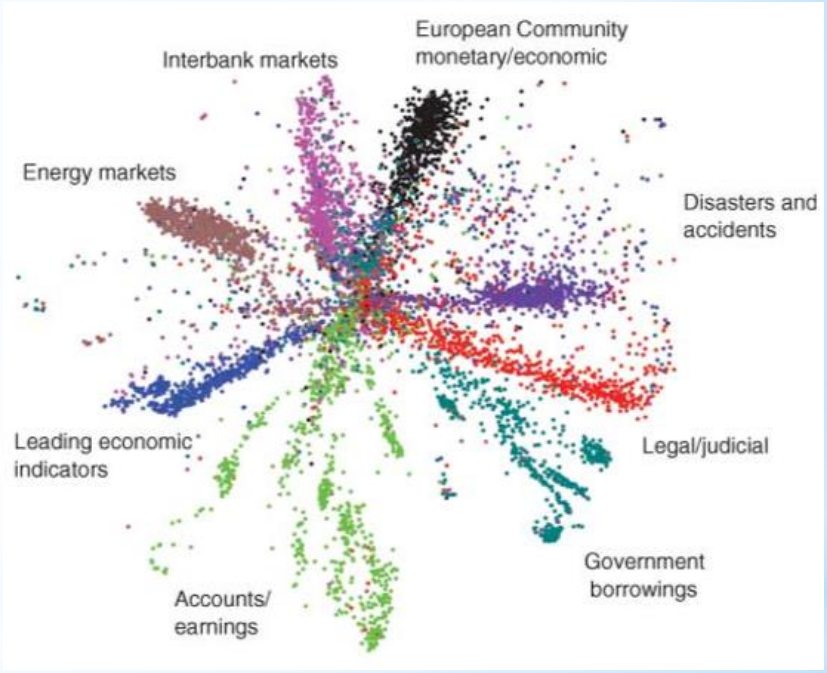
*G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks // Science 313 (2006), p. 504 – 507.*

# *Dimensionality reduction

804’414 Reuters news stories, reduction to 2 dimensions

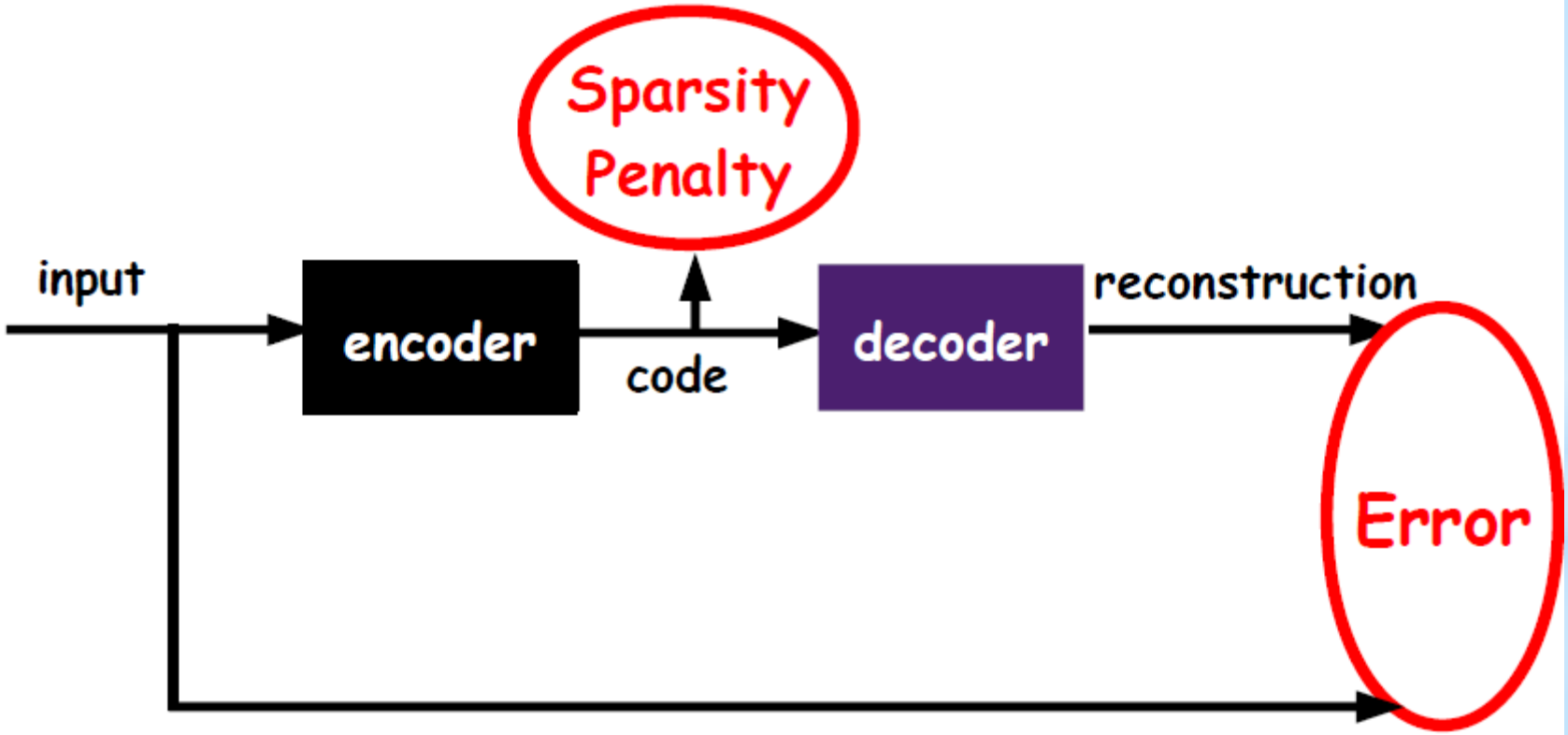
PCA

Deep RBN




*G. E. Hinton and R. R. Salakhutdinov. Reducing the Dimensionality of Data with Neural Networks // Science 313 (2006), p. 504 – 507.*

# Sparse Autoencoders

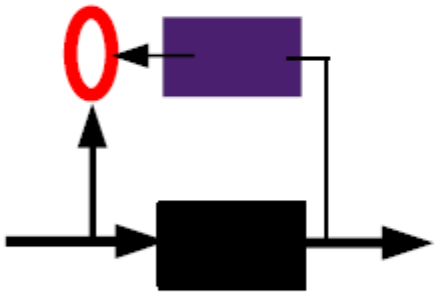



- input: $X$ code: $h = W^T X$

- loss: $L(X; W) = \| W h - X \|^2 + \lambda \sum_j |h_j|$

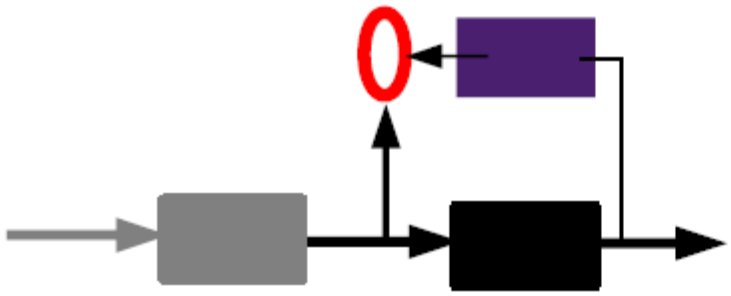
# How to use unsupervised pre-training stage / 1

1) Given unlabeled data, learn features



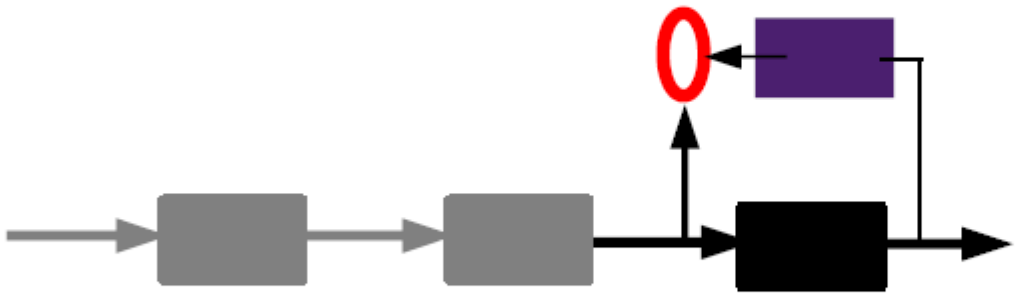
# How to use unsupervised pre-training stage / 2

1) Given unlabeled data, learn features
2) Use encoder to produce features and train another layer on the top



# How to use unsupervised pre-training stage / 3

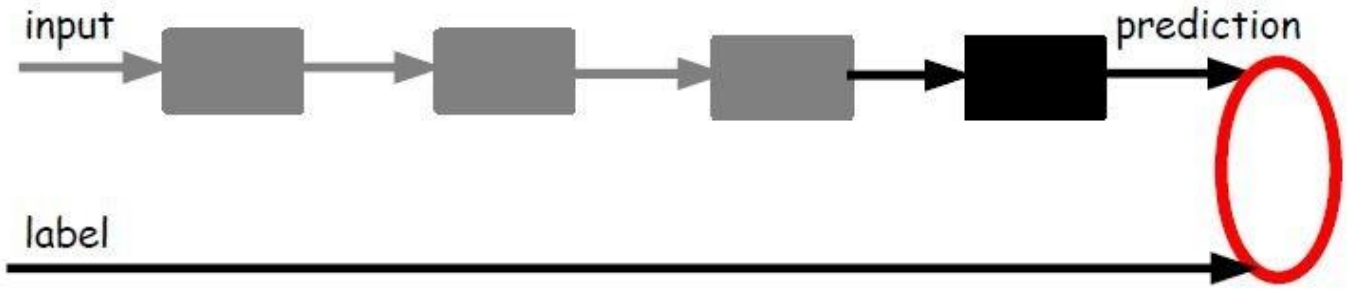
1) Given unlabeled data, learn features
2) Use encoder to produce features and train another layer on the top



Layer-wise training of a feature hierarchy

# How to use unsupervised pre-training stage / 4

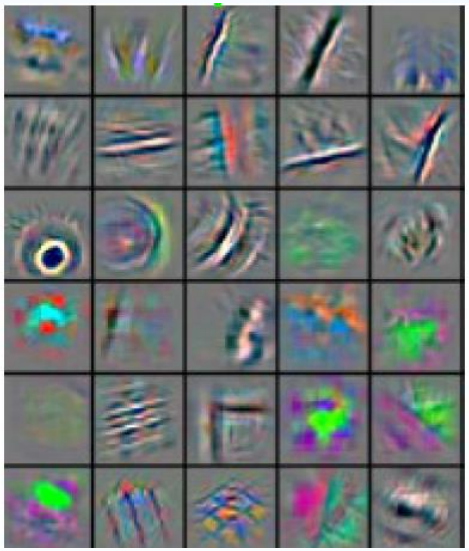
1) Given unlabeled data, learn features
2) Use encoder to produce features and train another layer on the top
3) feed features to classifier & train just the classifier




Reduced overfitting since features are learned in unsupervised way!

# Hierarchy of trained representations

Low-level feature → Middle-level feature → Top-level feature



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

# FLOPS comparison

| Type | Name | Flops | Cost |
|---|---|---|---|
| Mobile | Raspberry Pi 1st Gen, 700 Mhz | 0,04 Gflops | $35 |
| Mobile | Apple A8 | 1,4 Gflops | $700 (in iPhone 6) |
| CPU | Intel Core i7-4930K (Ivy Bridge), 3.7 GHz | 140 Gflops | $700 |
| CPU | Intel Core i7-5960X (Haswell), 3.0 GHz | 350 Gflops | $1300 |
| GPU | NVidia GTX 980 | 4612 Gflops (single precision), 144 Gflops (double precision) | $600 + cost of PC (~$1000) |
| GPU | NVidia Tesla K80 | 8740 Gflops (single precision), 2910 Gflops (double precision) | $4500 + cost of PC (~1500) |

# Deep Networks Training time using GPU


- Pretraining – from  2-3 weeks to 2-3 months.
- Fine-tuning (final supervised training) – from 1 day to 1 week.

# Tools for training Deep Neural Networks

| *Name* | *Language* | *OS* | *FC NN* | *CNN* | *AE* | *RBM* |
|---|---|---|---|---|---|---|
| DeepLearnToolbox | Matlab | Win, Lin | + | + | + | + |
| Theano | Python | Win, Lin, Mac | + | + | + | + |
| Pylearn2 | Python | Lin, Vagrant | + | + | + | + |
| Deepnet | Python | Lin | + | + | + | + |
| Deepmat | Matlab | ? | + | + | + | + |
| Torch | Lua, C | Lin, iOS, Android | + | + | + | + |
| Darch | R | Win, Lin | + | – | + | + |
| Caffe | C++, Python, Matlab | Lin, Win | + | + | – | – |
| nnForge | C++ | Lin | + | + | – | – |
| CXXNET | C++ | Lin | + | + | – | – |
| Cuda-convnet | C++ | Win, Lin | + | + | – | – |
| Cuda CNN | Matlab | Win, Lin | + | + | – | – |

*D. Kruchinin, E. Dolotov, K. Kornyakov, V. Kustikova, P. Druzhkov. The Comparison of Deep Learning Libraries on the Problem of Handwritten Digit Classication // Analysis of Images, Social Networks and Texts (AIST), 2015, April, 9-11th, Yekaterinburg.*

# TIMIT Phoneme Recognition

| Model | # of parameters | Accuracy |
|---|---|---|
| Hidden Markov Model, HMM | N / A | 27,3% |
| Deep Belief Network, DBN | ~ 4M | 26,7% |
| Deep RNN | 4,3M | 17.7% |

Training data: 462 speakers train / 24 speakers test, 3.16 / 0.14 hrs.

*Graves, A., Mohamed, A.-R., and Hinton, G. E. (2013). Speech recognition with deep recurrent neural networks // IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 6645-6649. IEEE.*

*Mohamed, A. and Hinton, G. E. (2010). Phone recognition using restricted Boltzmann machines // IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 4354-4357.*

# Google Large Vocabulary Speech Recognition

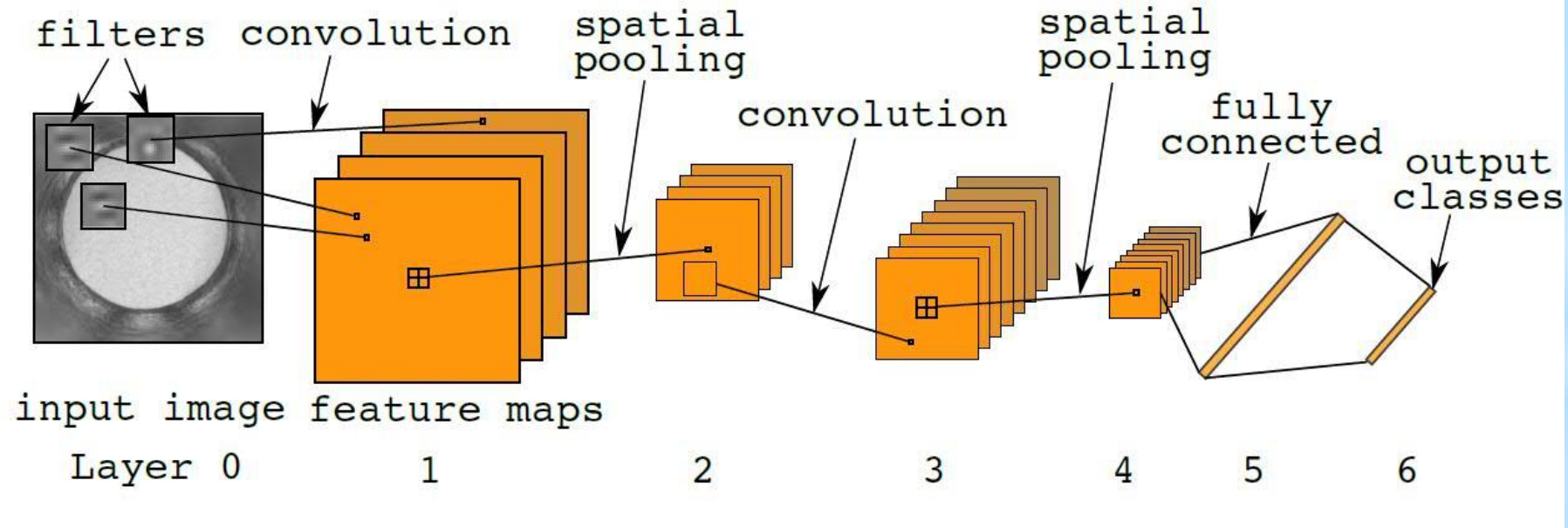
| Model | # of parameters | Cross-entropy |
|---|---|---|
| ReLU DNN | 85M | 11.3 |
| Deep Projection LSTM RNN | 13M | 10.7 |

Training data: 3M utterances (1900 hrs).

*H. Sak, A. Senior, F. Beaufays. Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling // INTERSPEECH'2014.*

*K. Vesely, A. Ghoshal, L. Burget, D. Povey. Sequence-discriminative training of deep neural networks // INTERSPEECH'2014.*

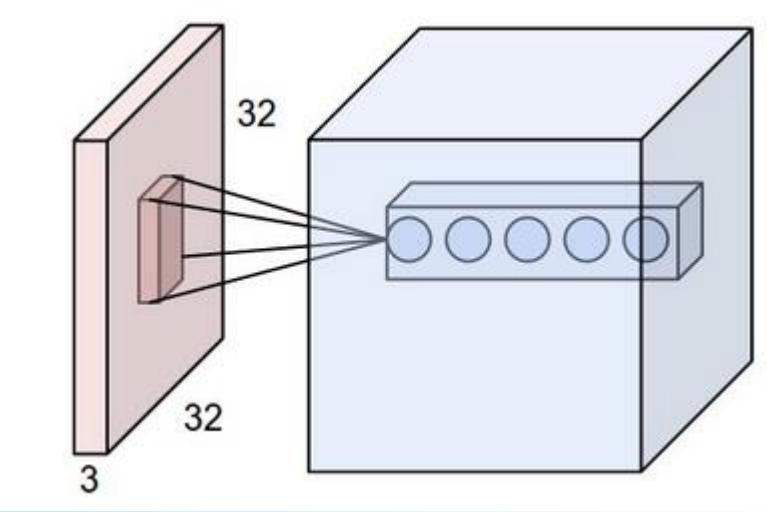
# Convolutional Neural Networks:

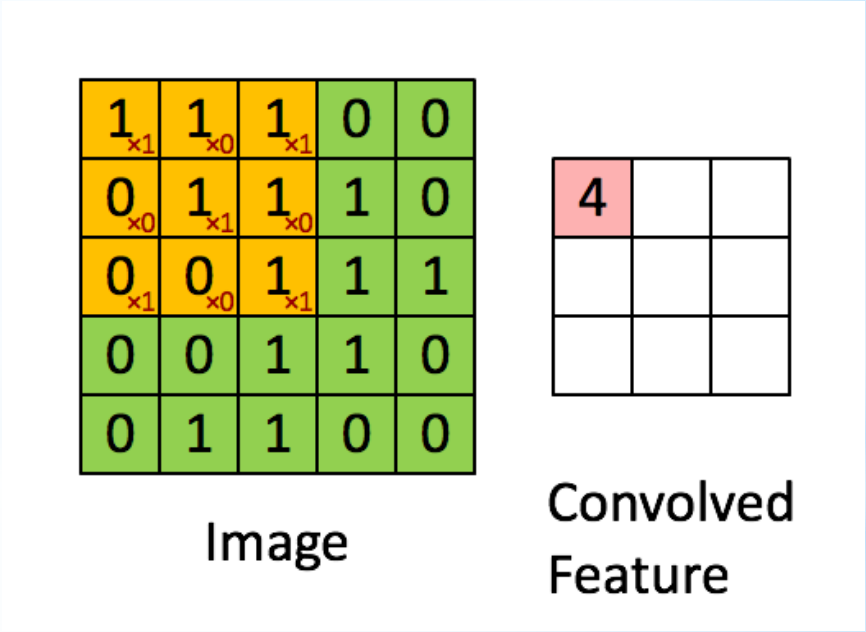



*Andrej Karpathy and* Fei-Fei. *CS231n: Convolutional Neural Networks for Visual Recognition* http://cs231n.github.io/convolutional-networks

*Yoshua Bengio, Ian Goodfellow and Aaron Courville. Deep Learning // An MIT Press book in preparation* http://www-labs.iro.umontreal.ca/~bengioy/DLbook

# Convolution Layer

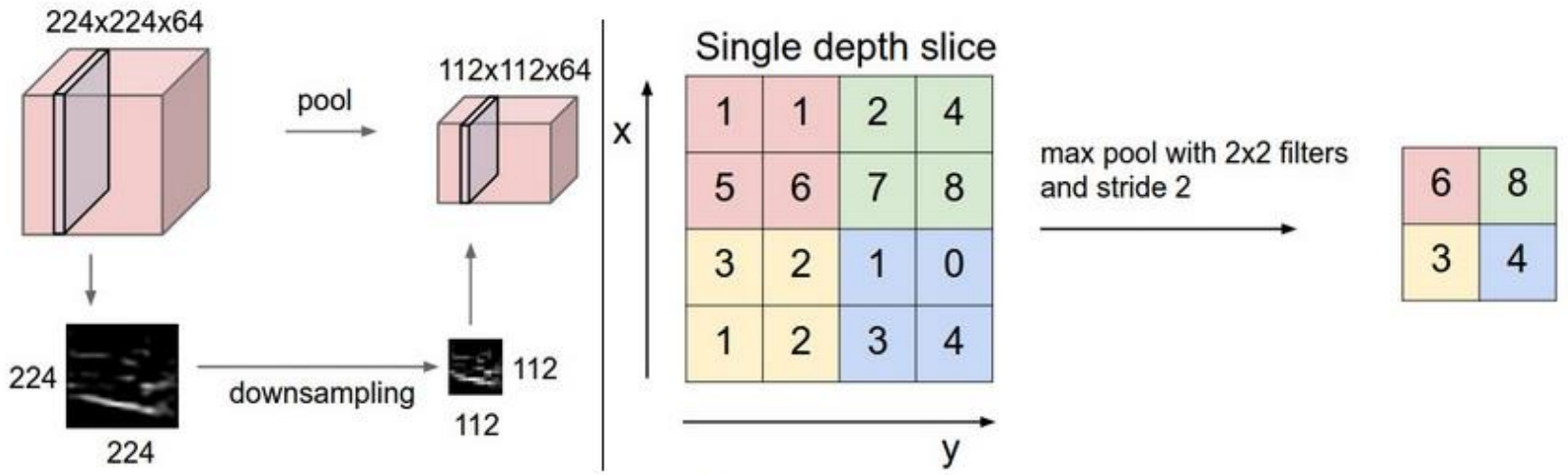



*Andrej Karpathy and* Fei-Fei. *CS231n: Convolutional Neural Networks for Visual Recognition* http://cs231n.github.io/convolutional-networks

*Yoshua Bengio, Ian Goodfellow and Aaron Courville. Deep Learning // An MIT Press book in preparation* http://www-labs.iro.umontreal.ca/~bengioy/DLbook

# Pooling layer

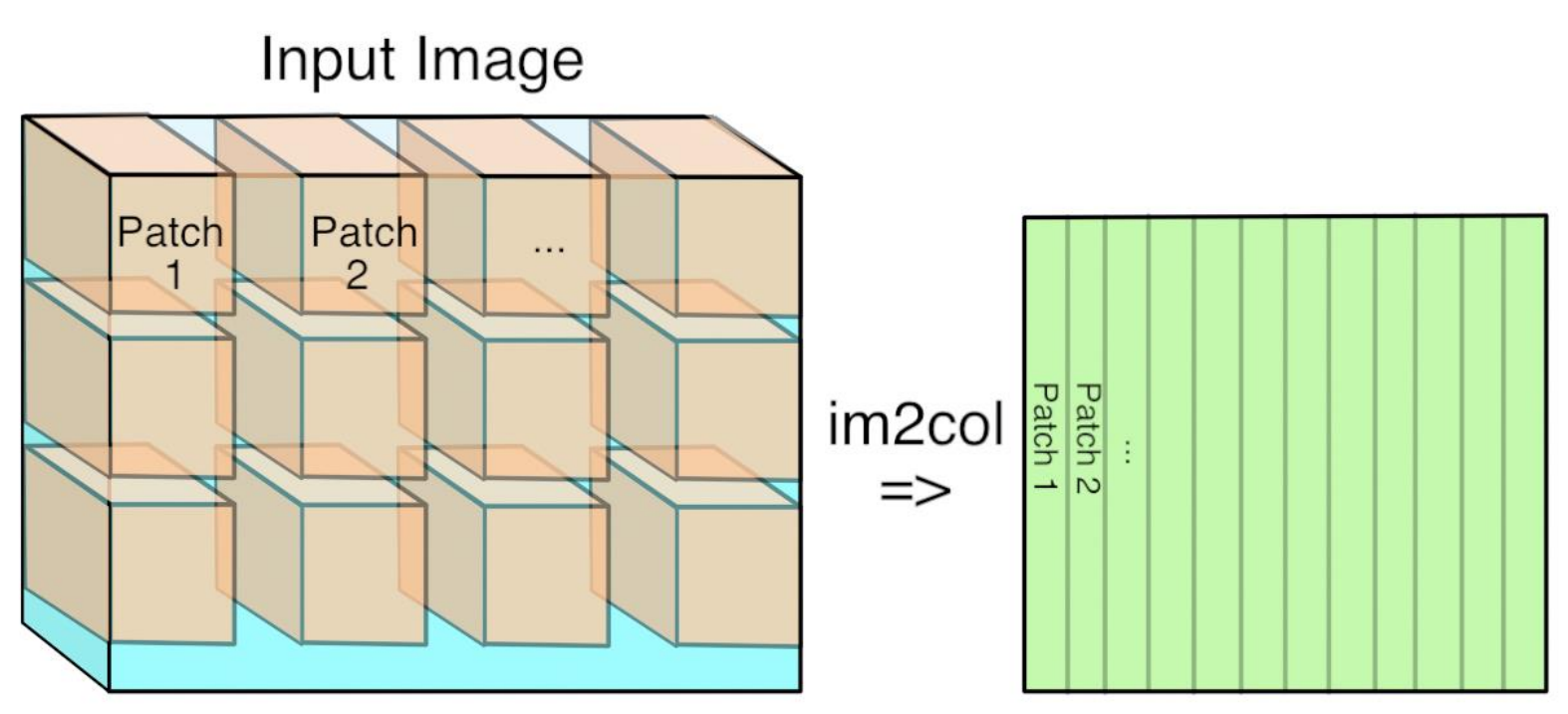



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

*Andrej Karpathy and* Fei-Fei. *CS231n: Convolutional Neural Networks for Visual Recognition* http://cs231n.github.io/convolutional-networks

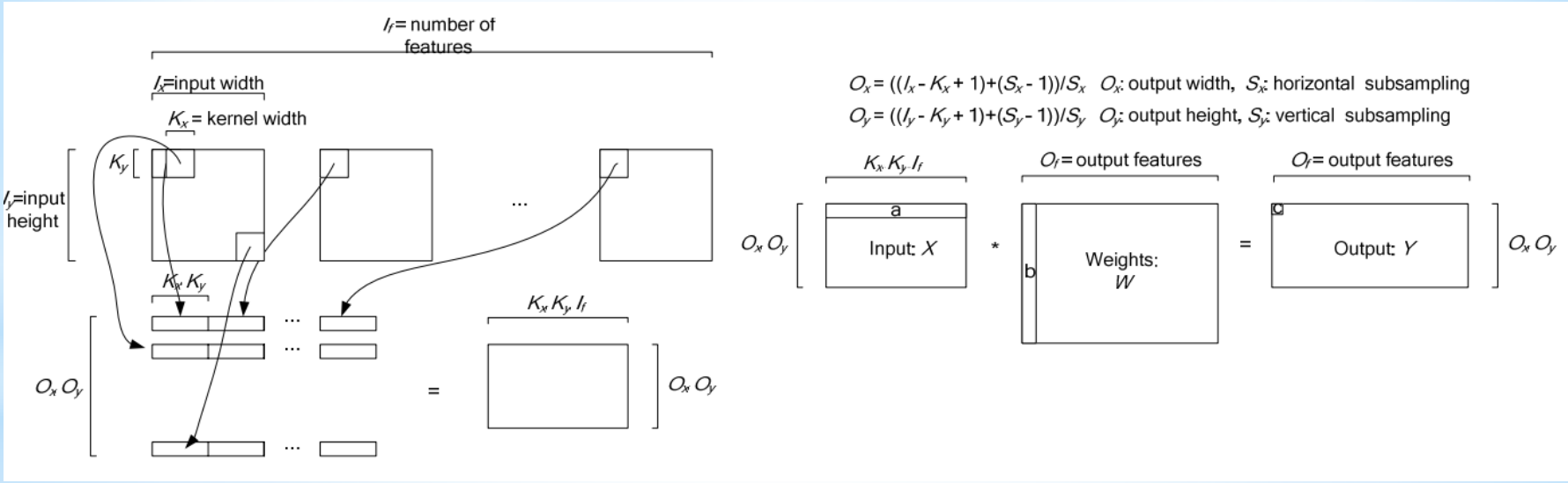
*Yoshua Bengio, Ian Goodfellow and Aaron Courville. Deep Learning // An MIT Press book in preparation* http://www-labs.iro.umontreal.ca/~bengioy/DLbook

# Implementation tricks: im2col




*K. Chellapilla, S. Puri, P. Simard. High Performance Convolutional Neural Networks for Document Processing // International Workshop on Frontiers in Handwriting Recognition, 2006.*

# Implementation tricks: im2col for convolution




*K. Chellapilla, S. Puri, P. Simard. High Performance Convolutional Neural Networks for Document Processing // International Workshop on Frontiers in Handwriting Recognition, 2006.*

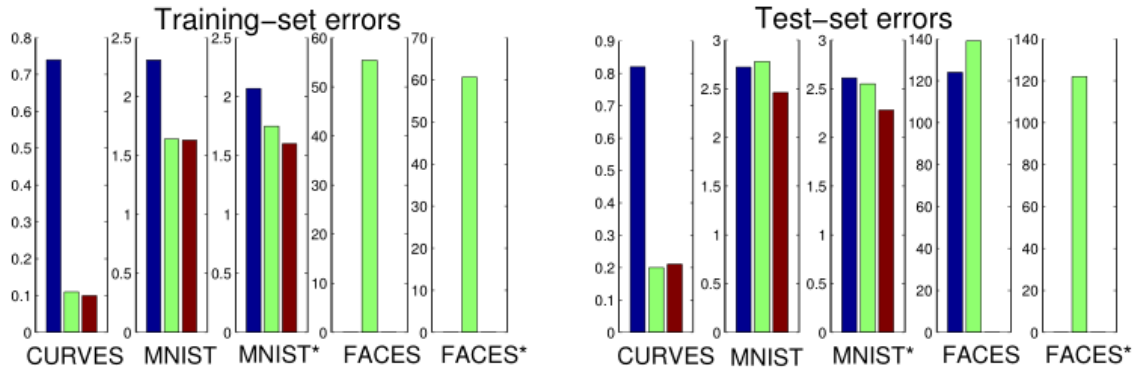
# Deep, Big, Simple Neural Nets

| ID | architecture (number of neurons in each layer) | test error for best validation [%] | best test error [%] | simulation time [h] | weights [milions] |
|---|---|---|---|---|---|
| 1 | 1000, 500, 10 | **0.49** | 0.44 | 23.4 | 1.34 |
| 2 | 1500, 1000, 500, 10 | **0.46** | 0.40 | 44.2 | 3.26 |
| 3 | 2000, 1500, 1000, 500, 10 | **0.41** | 0.39 | 66.7 | 6.69 |
| 4 | 2500, 2000, 1500, 1000, 500, 10 | **0.35** | 0.32 | 114.5 | 12.11 |
| 5 | $9 \times 1000$, 10 | **0.44** | 0.43 | 107.7 | 8.86 |

*D. Ciresan, U. Meier, L. M. Gambardella, J. Schmidhuber - Deep, Big, Simple Neural Nets for Handwritten Digit Recognition (Neural Computation, December 2010*

# Hessian-Free optimization: Deep Learning with no pre-training stage

- **PT + NCG** = pre-trained initialization with non-linear CG optimizer
- **RAND+HF** = random initialization with our Hessian-free method
- **PT + HF** = pre-trained initialization with our Hessian-free method
- * indicates an $\ell_2$ prior was used




| | PT + NCG | RAND+HF | PT + HF | NO EARLY STOP |
|---|---|---|---|---|
| **CURVES** | 0.74, 0.82 | 0.11, 0.20 | 0.10, 0.21 | 0.1 |
| **MNIST** | 2.31, 2.72 | 1.64, 2.78 | 1.63, 2.46 | 1.4 |
| **MNIST*** | 2.07, 2.61 | 1.75, 2.55 | 1.60, 2.28 | |
| **FACES** | -, 124 | 55.4, 139 | -,- | 12.9! |
| **FACES*** | -,- | 60.6, 122 | -,- | |

*J. Martens. Deep Learning via Hessian-free Optimization // Proceedings of the 27th International Conference on Machine Learning (ICML), 2010.*

# *Thanks!