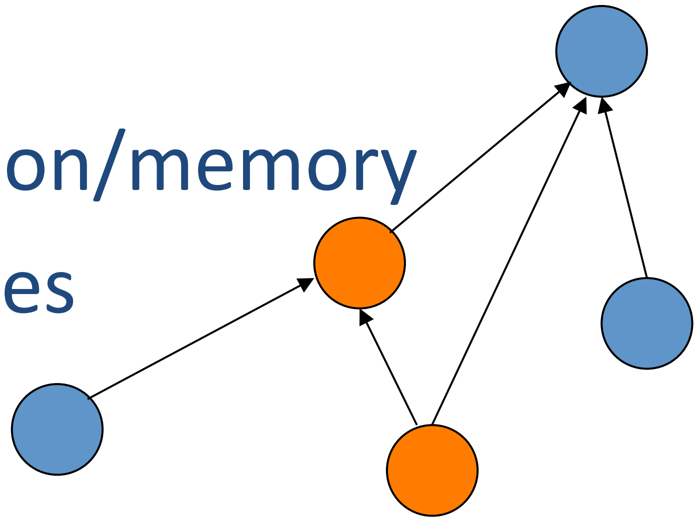
- Basic Concepts
  - supervised, unsupervised and semi-supervised learning
  - density estimation, classification, regression
  - error measures
  - optimization
  - linear and nonlinear models
- Perceptrons and Multilayer Perceptrons
  - perceptron
  - multilayer perceptron
  - training
  - overfitting and its prevention
  - learning from data and hints
  - neural networks for time series
- Deep Neural Networks
  - shallow networks' problems
  - optimization
  - regularization
  - convolution
  - hardware

Neural Networks

- Networks of processing units (neurons) with connections (synapses) between them
- Large number of neurons: 10^{10}
- Large connectitivity: 10^5
- Parallel processing
- Distributed computation/memory
- Robust to noise, failures

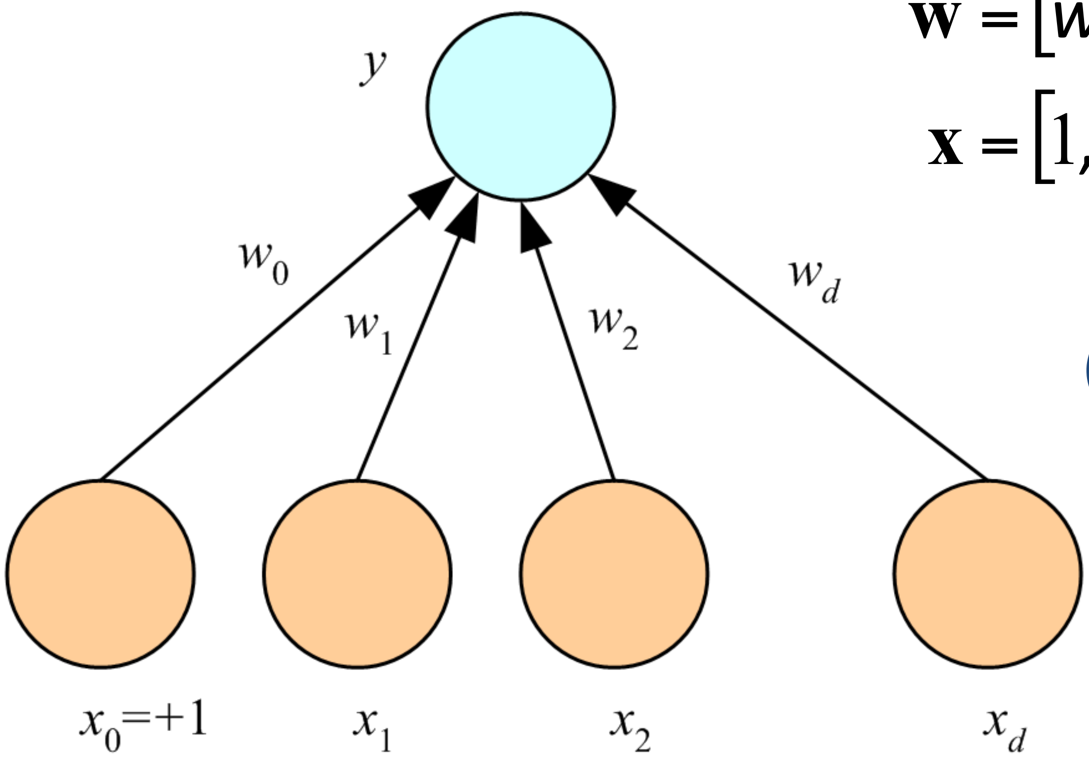


Perceptron

$$y = \sum_{j=1}^{d} w_j x_j + w_0 = \mathbf{w}^T \mathbf{x}$$

$$\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$$

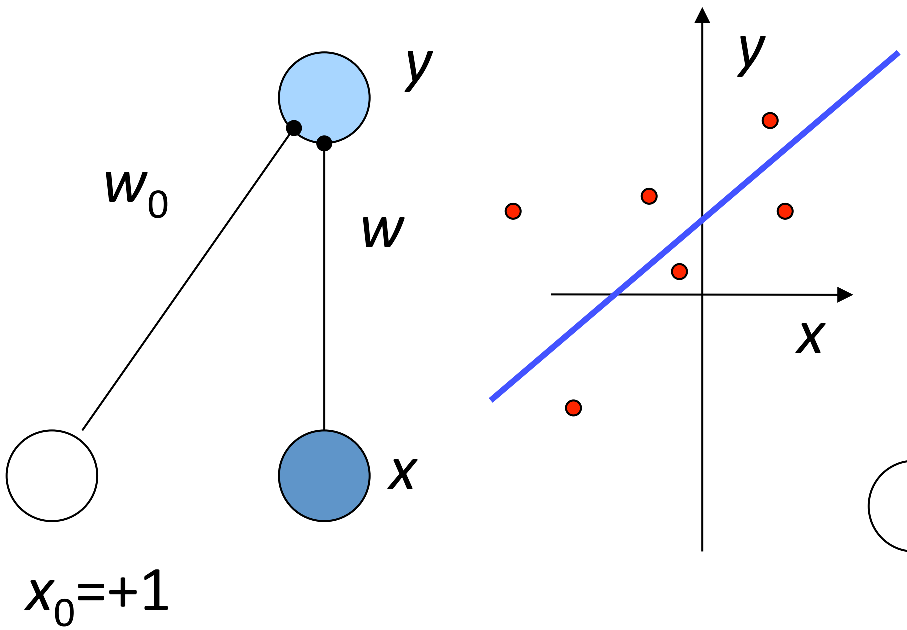
$$\mathbf{x} = [1, x_1, \ldots, x_d]^T$$



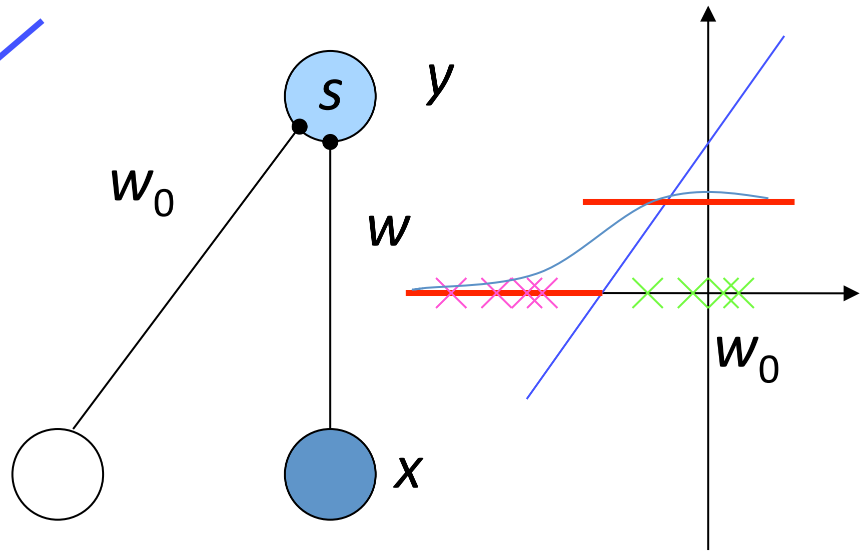
(Rosenblatt, 1962)

What a Perceptron Does

- Regression: $y=wx+w_0$



- Classification: $y=1(wx+w_0>0)$



$$y = \text{sigmoid}\ (o) = \frac{1}{1+\exp\left[-\mathbf{w}^T\mathbf{x}\right]}$$

Training a Perceptron

- Regression:

$$E^t\left(\mathbf{w} \mid \mathbf{x}^t, r^t\right) = \frac{1}{2}\left(r^t - y^t\right)^2 = \frac{1}{2}\left[r^t - \left(\mathbf{w}^T \mathbf{x}^t\right)\right]^2$$

$$\Delta w_j^t = \eta\left(r^t - y^t\right)x_j^t$$

- Classification:

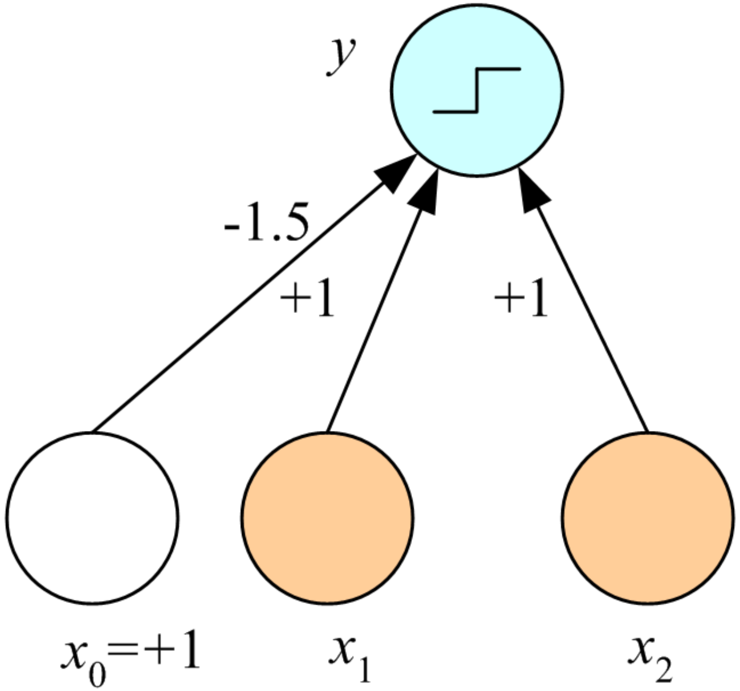
$$y^t = \text{sigmoid}\left(\mathbf{w}^T \mathbf{x}^t\right)$$

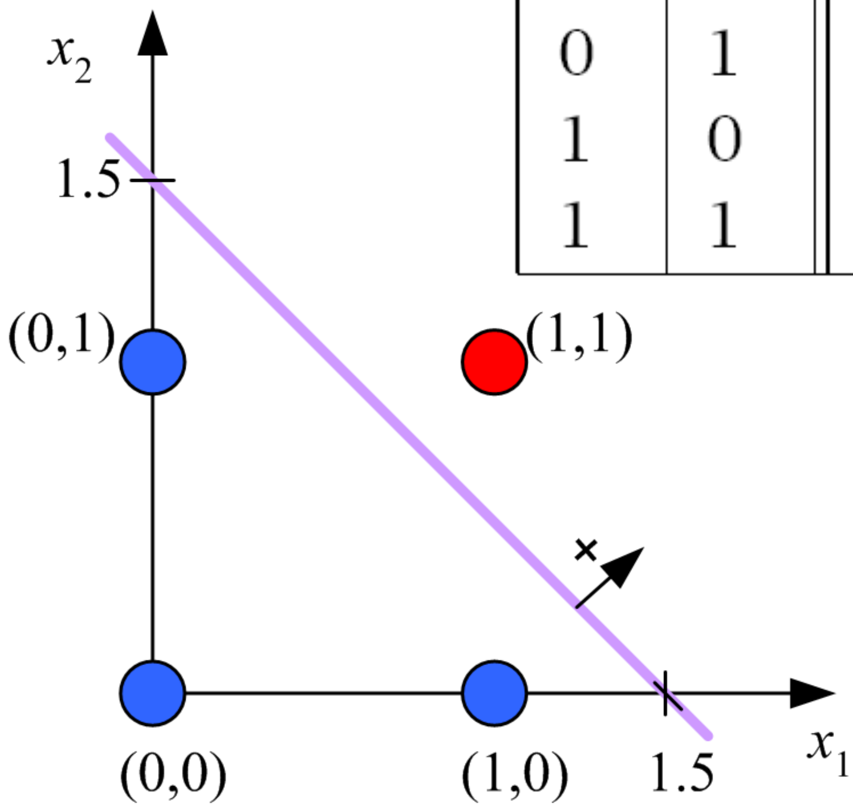
$$E^t\left(\mathbf{w} | \mathbf{x}^t, \mathbf{r}^t\right) = -r^t \log y^t - \left(1 - r^t\right) \log\left(1 - y^t\right)$$

$$\Delta w_j^t = \eta\left(r^t - y^t\right)x_j^t$$

Learning Boolean AND

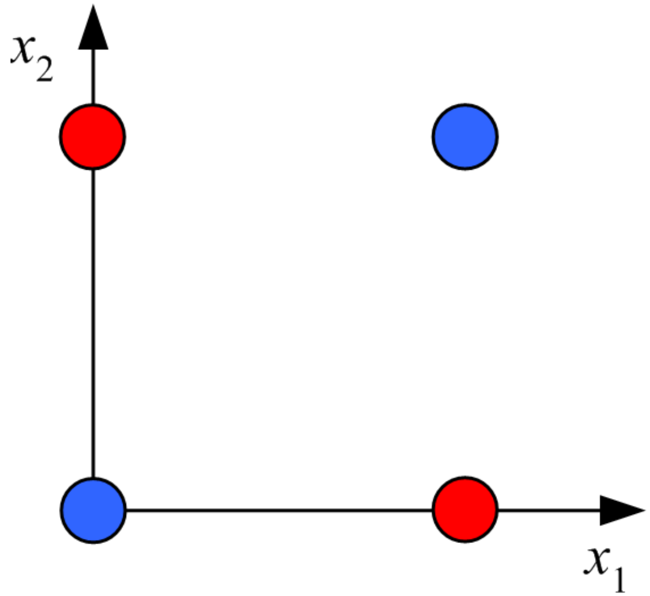
x_1	x_2	r
0	0	0
0	1	0
1	0	0
1	1	1





XOR

x_1	x_2	r
0	0	0
0	1	1
1	0	1
1	1	0

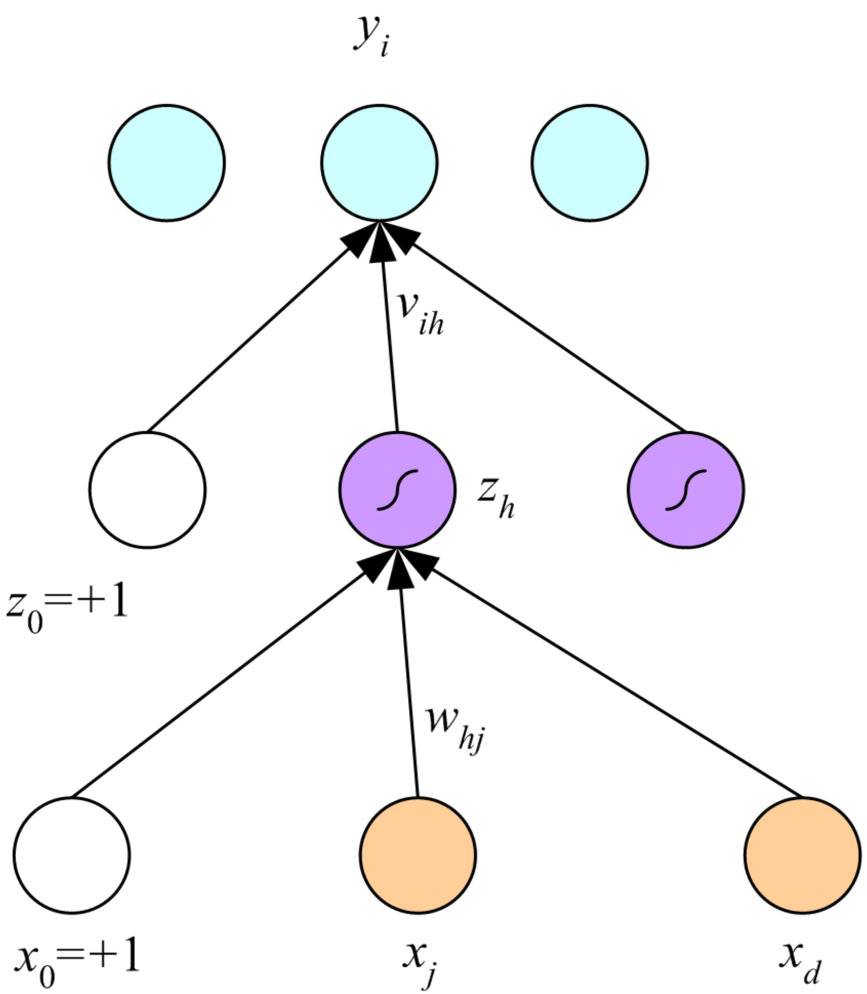


- No w_0, w_1, w_2 satisfy:

$$
\begin{aligned}
w_0 &\le 0 \\
w_2 + w_0 &> 0 \\
w_1 + w_0 &> 0 \\
w_1 + w_2 + w_0 &\le 0
\end{aligned}
$$

(Minsky and Papert, 1969)

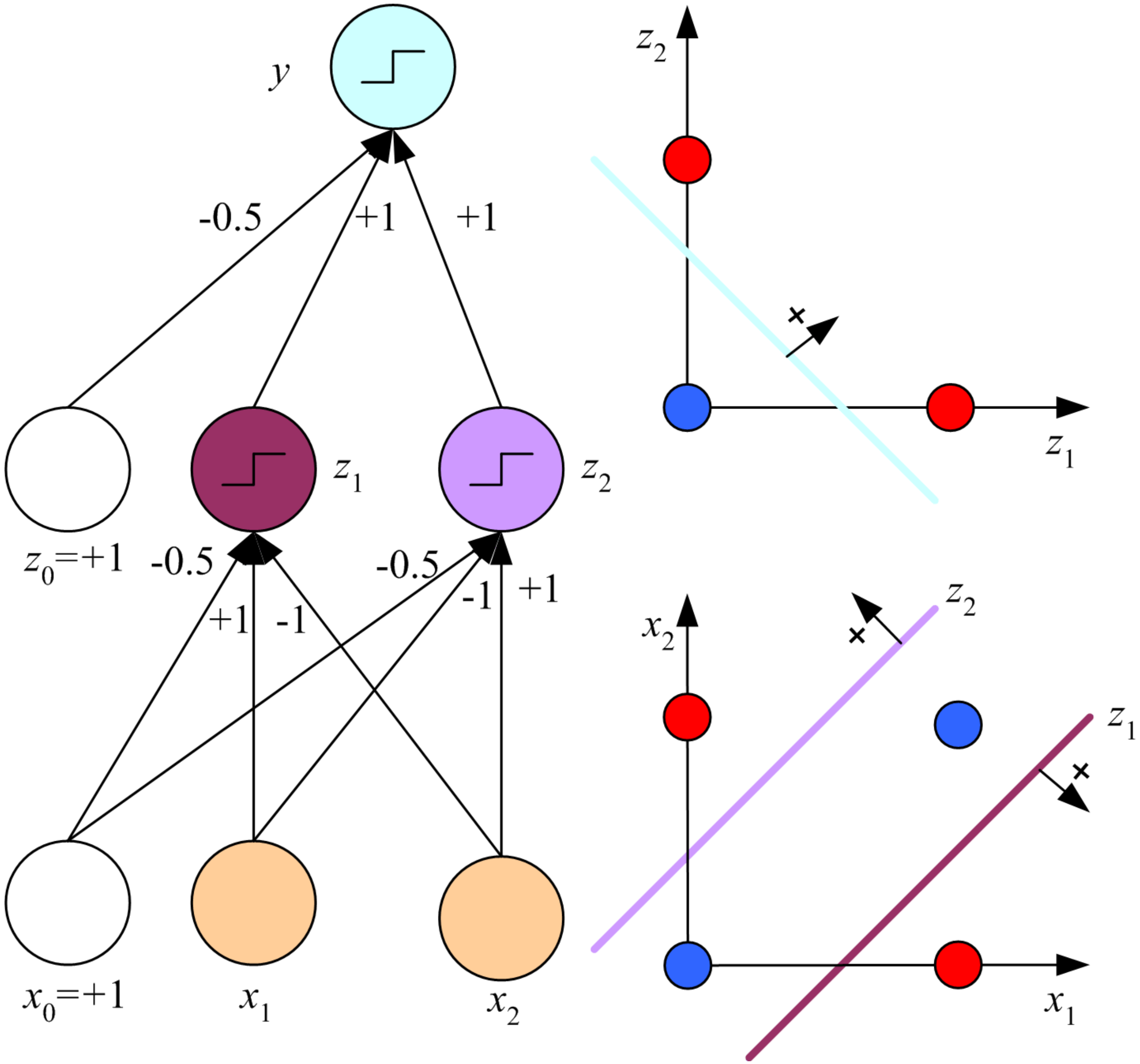
Multilayer Perceptrons



$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

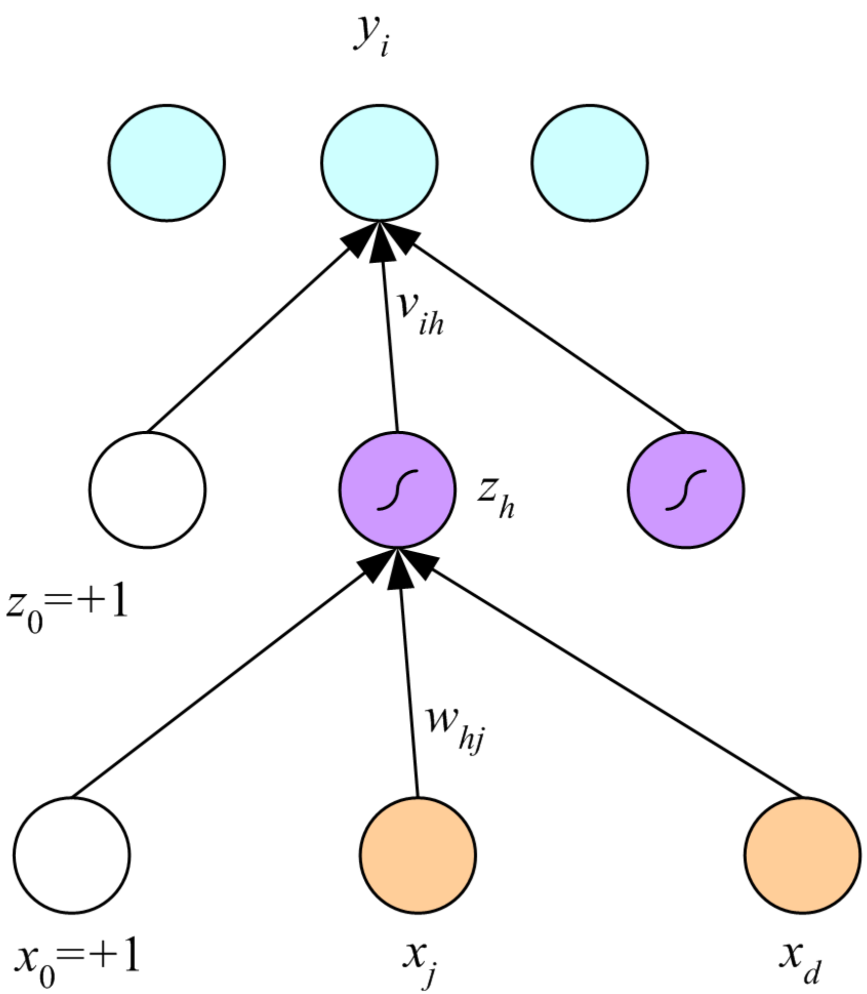
$$z_h = \text{sigmoid}\left(\mathbf{w}_h^T \mathbf{x}\right) = \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

(Rumelhart et al., 1986)



x_1 XOR x_2 = (x_1 AND ~x_2) OR (~x_1 AND x_2)

Backpropagation



$$y_i = \mathbf{v}_i^T \mathbf{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

$$z_h = \text{sigmoid}\left(\mathbf{w}_h^T \mathbf{x}\right)$$

$$= \frac{1}{1 + \exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

$$\boxed{\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial z_h} \frac{\partial z_h}{\partial w_{hj}}}$$

Regression

$$E(\mathbf{W},\mathbf{v}\mid\mathcal{X})=\frac{1}{2}\sum_t\left(r^t-y^t\right)^2$$

$$y^t=\sum_{h=1}^{H}v_h z_h^t+v_0$$

$$\boxed{\Delta v_h=\sum_t\left(r^t-y^t\right)z_h^t}$$

Forward

Backward

$$z_h=\boxed{\text{sigmoid}}\left(\mathbf{w}_h^T\mathbf{x}\right)$$

$\boldsymbol{x}$

$$\begin{aligned}\Delta w_{hj}&=-\eta\frac{\partial E}{\partial w_{hj}}\\&=-\eta\sum_t\frac{\partial E}{\partial y^t}\frac{\partial y^t}{\partial z_h^t}\frac{\partial z_h^t}{\partial w_{hj}}\\&=-\eta\sum_t-\left(r^t-y^t\right)v_h\boxed{z_h^t\left(1-z_h^t\right)}x_j^t\\&=\eta\sum_t\left(r^t-y^t\right)v_h z_h^t\left(1-z_h^t\right)x_j^t\end{aligned}$$

Initialize all v_{ih} and w_{hj} to rand$(-0.01, 0.01)$

Repeat

    For all $(\boldsymbol{x}^t, r^t) \in \mathcal{X}$ in random order

        For $h = 1, \ldots, H$

            $z_h \leftarrow \text{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x}^t)$

        For $i = 1, \ldots, K$

            $y_i = \boldsymbol{v}_i^T \boldsymbol{z}$

        For $i = 1, \ldots, K$

            $\Delta \boldsymbol{v}_i = \eta(r_i^t - y_i^t)\boldsymbol{z}$

        For $h = 1, \ldots, H$

            $\Delta \boldsymbol{w}_h = \eta(\sum_i (r_i^t - y_i^t) v_{ih}) z_h (1 - z_h) \boldsymbol{x}^t$

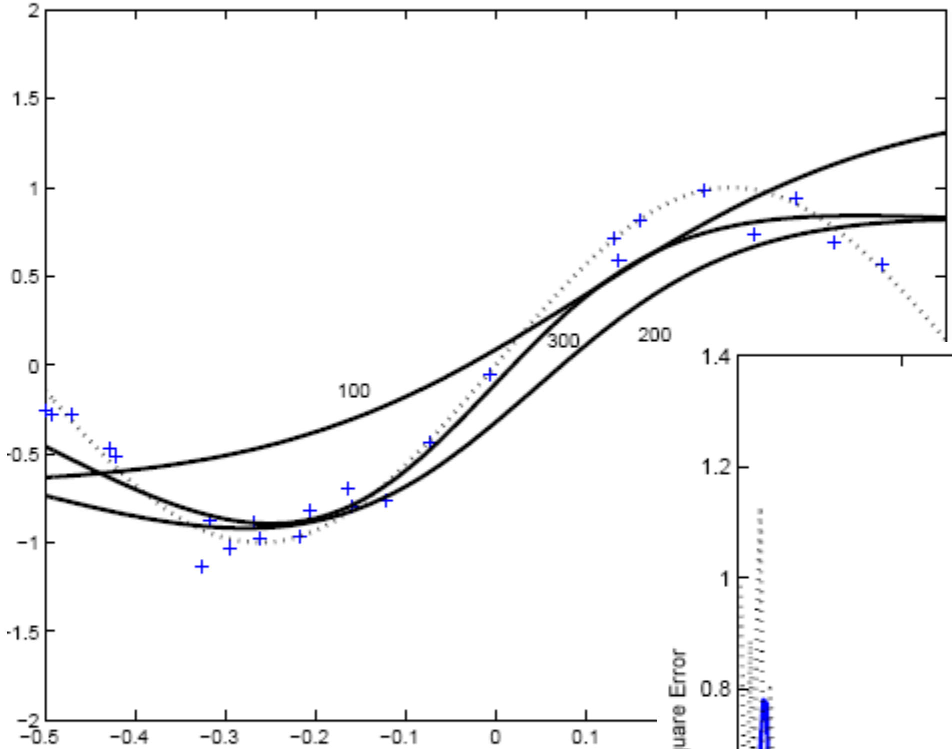
        For $i = 1, \ldots, K$

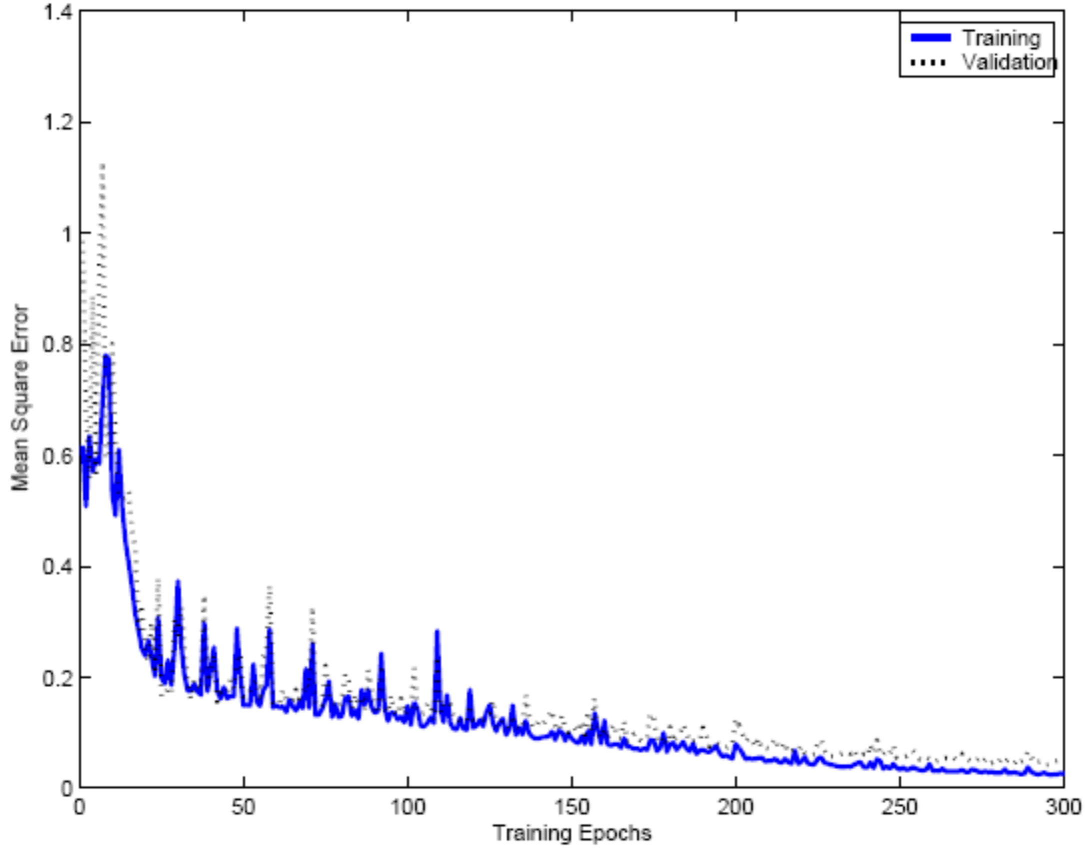
            $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \Delta \boldsymbol{v}_i$

        For $h = 1, \ldots, H$

            $\boldsymbol{w}_h \leftarrow \boldsymbol{w}_h + \Delta \boldsymbol{w}_h$

Until convergence


100
300
200


Training
Validation
Mean Square Error
Training Epochs

Improving Convergence

- Momentum

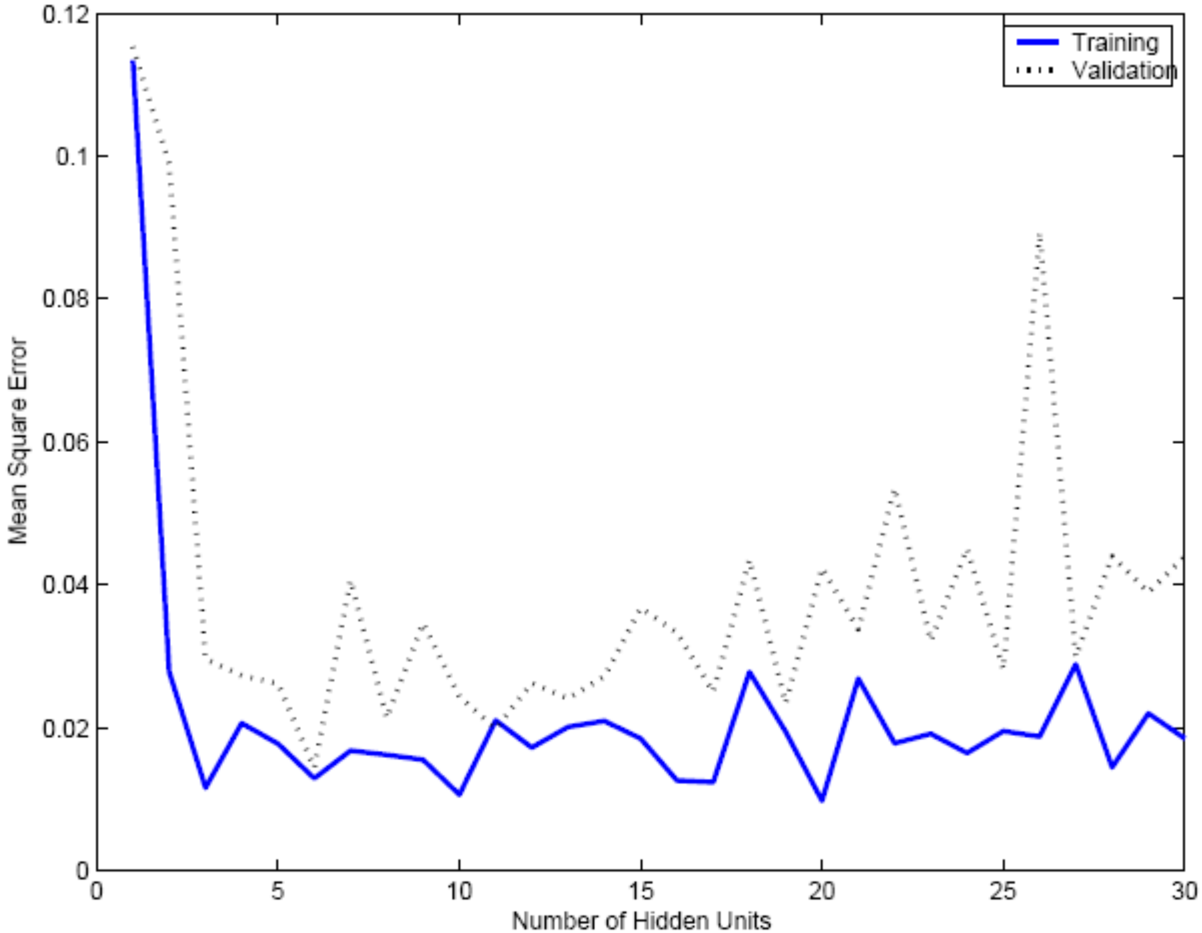
$$\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$$

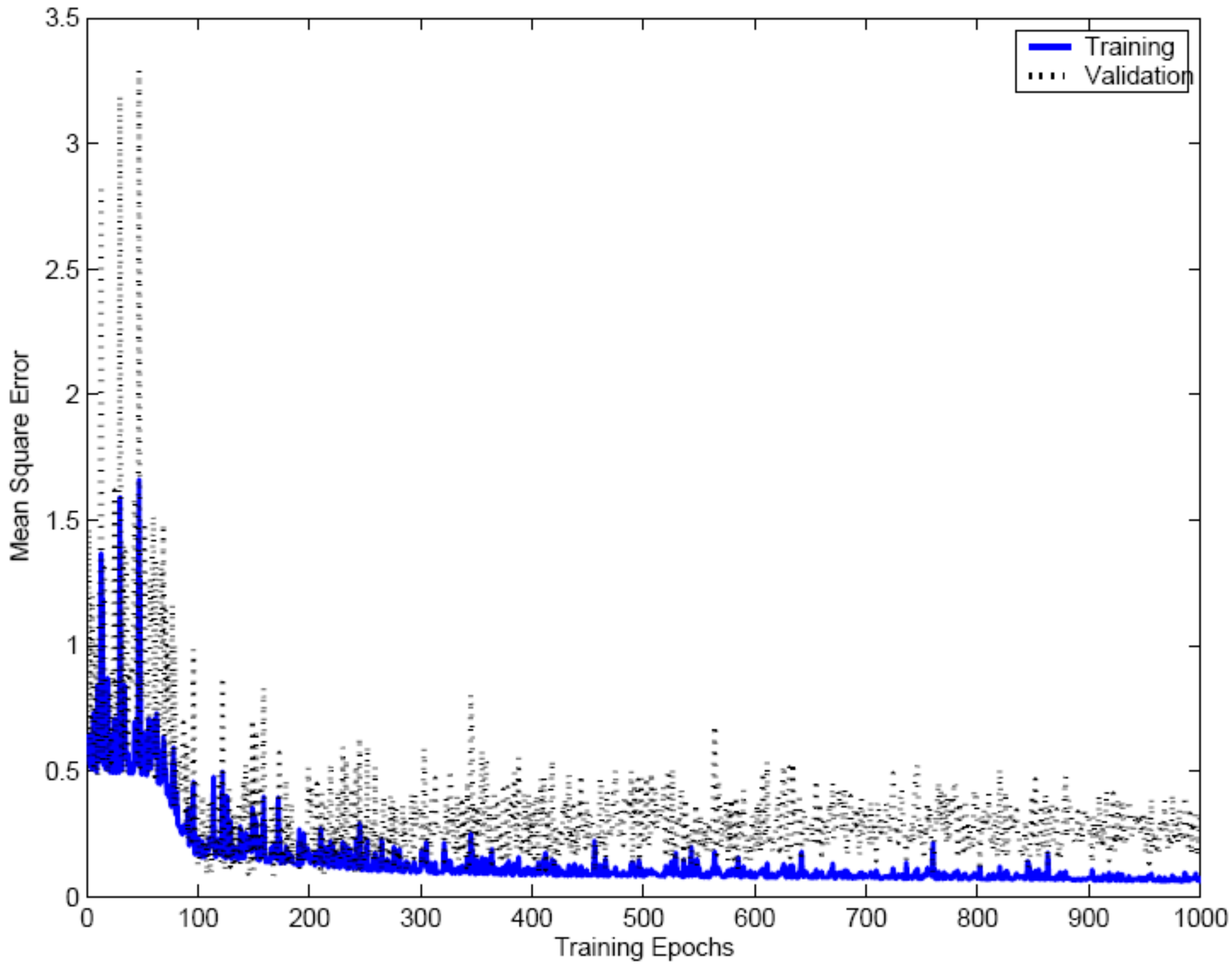
- Adaptive learning rate

$$\Delta \eta = \begin{cases} +a & \text{if } E^{t+\tau} < E^t \\ -b\eta & \text{otherwise} \end{cases}$$

Overfitting/Overtraining

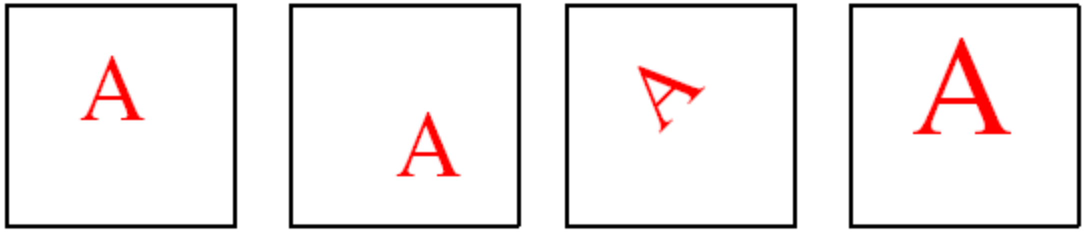
Number of weights: $H\,(d+1)+(H+1)K$




3.5
3
2.5
2
1.5
1
0.5
0
Mean Square Error
0
100
200
300
400
500
600
700
800
900
1000
Training Epochs
Training
Validation

Hints (Abu-Mostafa, 1995)

- Invariance to translation, rotation, size



- Virtual examples
- Augmented error: $E' = E + \lambda_h E_h$

If $\boldsymbol{x'}$ and $\boldsymbol{x}$ are the "same": $E_h = [g(x|\theta) - g(x'|\theta)]^2$

Approximation hint:

$$E_h = \begin{cases} 0 & \text{if } g(x|\theta) \in [a_x, b_x] \\ (g(x|\theta) - a_x)^2 & \text{if } g(x|\theta) < a_x \\ (g(x|\theta) - b_x)^2 & \text{if } g(x|\theta) > b_x \end{cases}$$

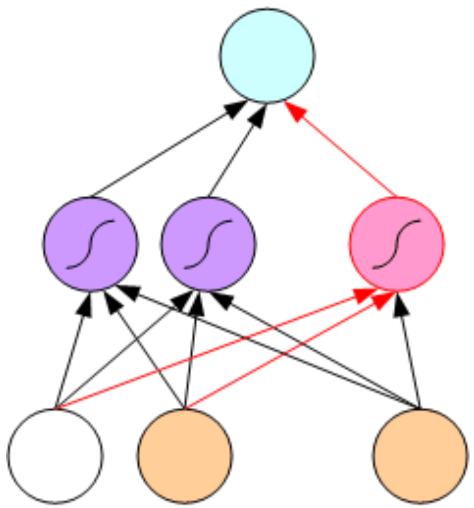
Tuning the Network Size

- Destructive
- Weight decay:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda w_i$$

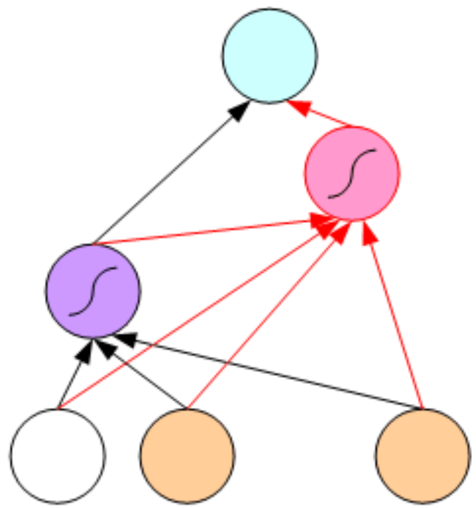
$$E' = E + \frac{\lambda}{2} \sum_i w_i^2$$

- Constructive
- Growing networks



Dynamic Node Creation

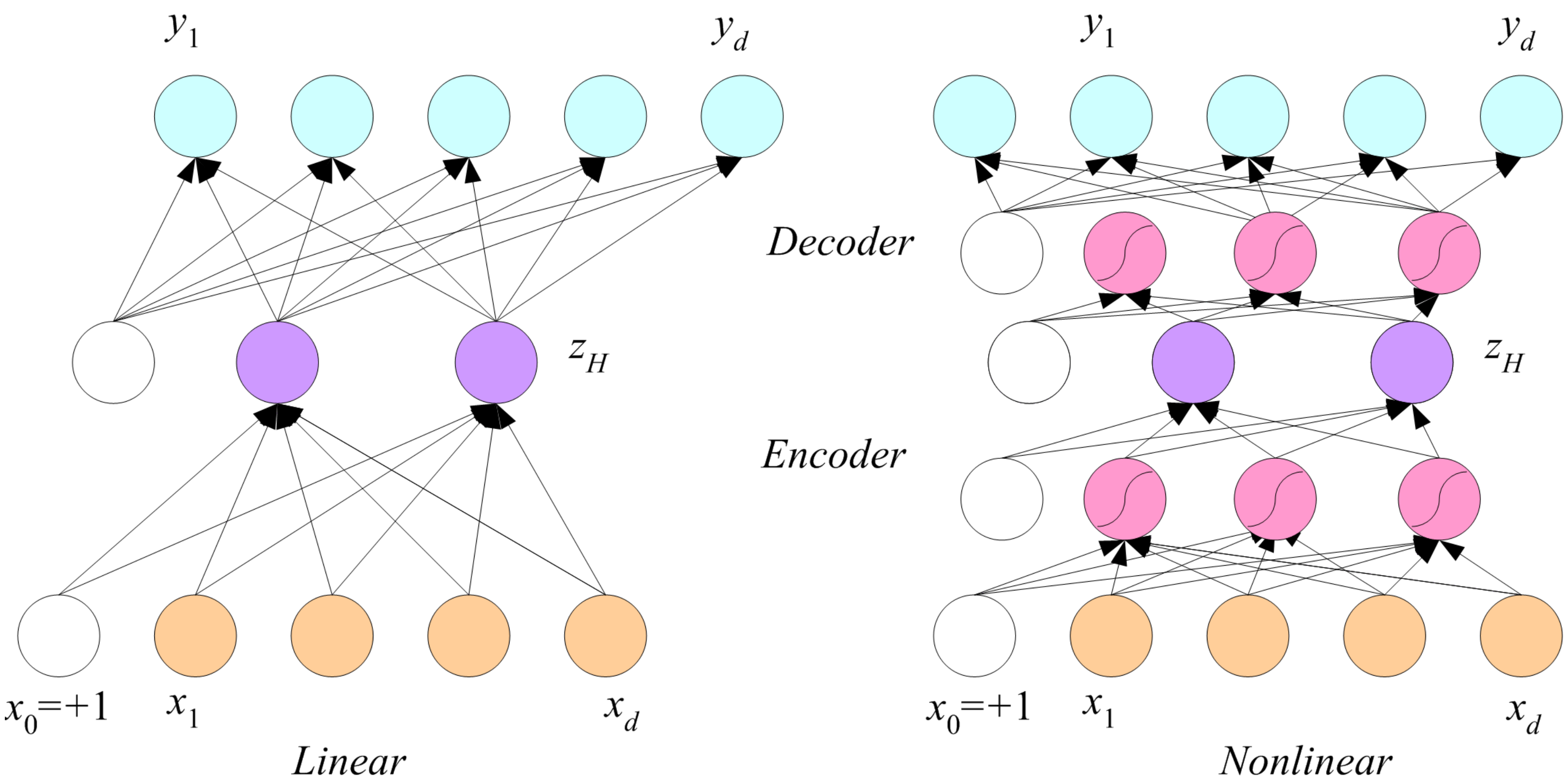
(Ash, 1989)



Cascade Correlation

(Fahlman and Lebiere, 1989)

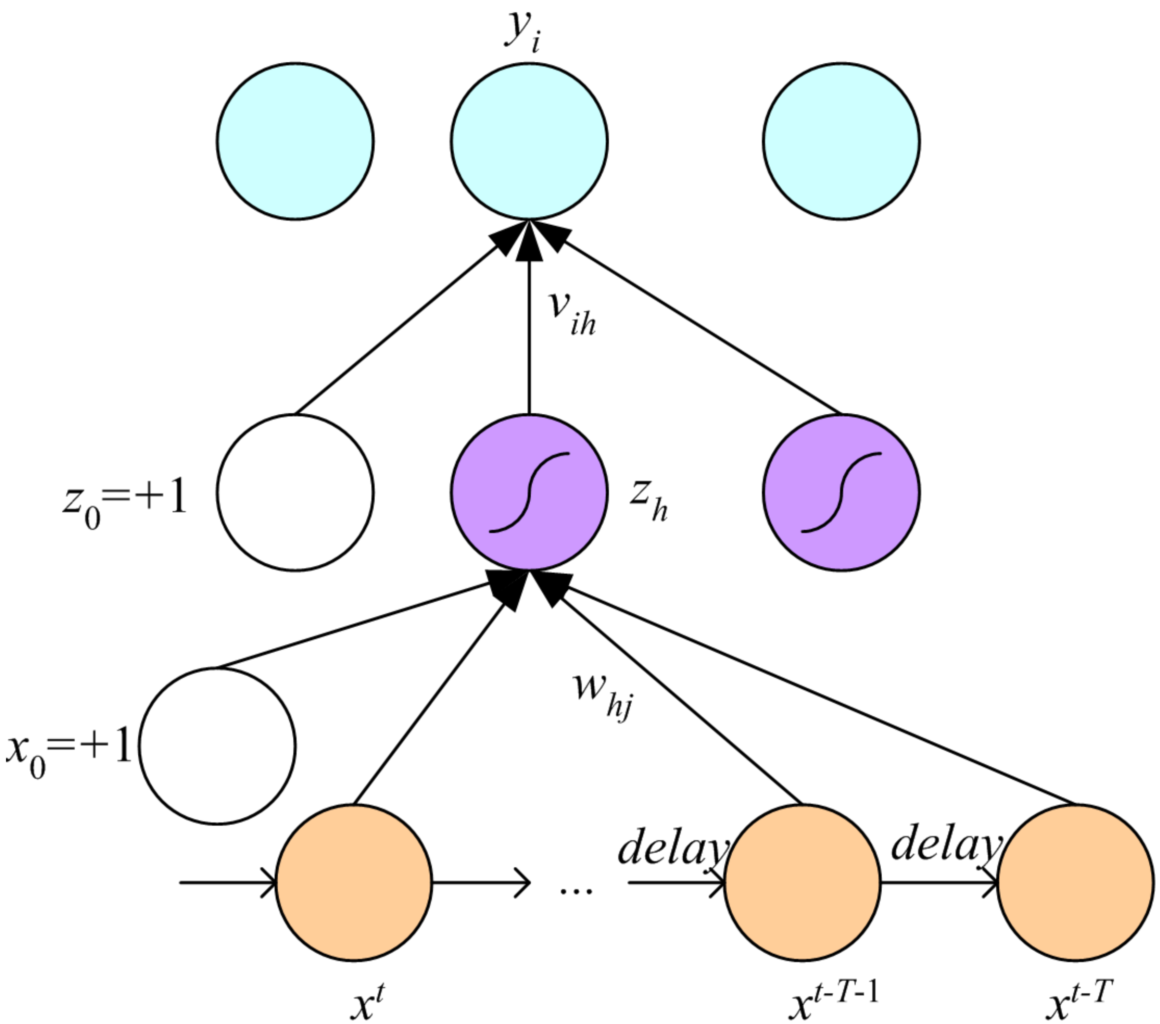
Dimensionality Reduction



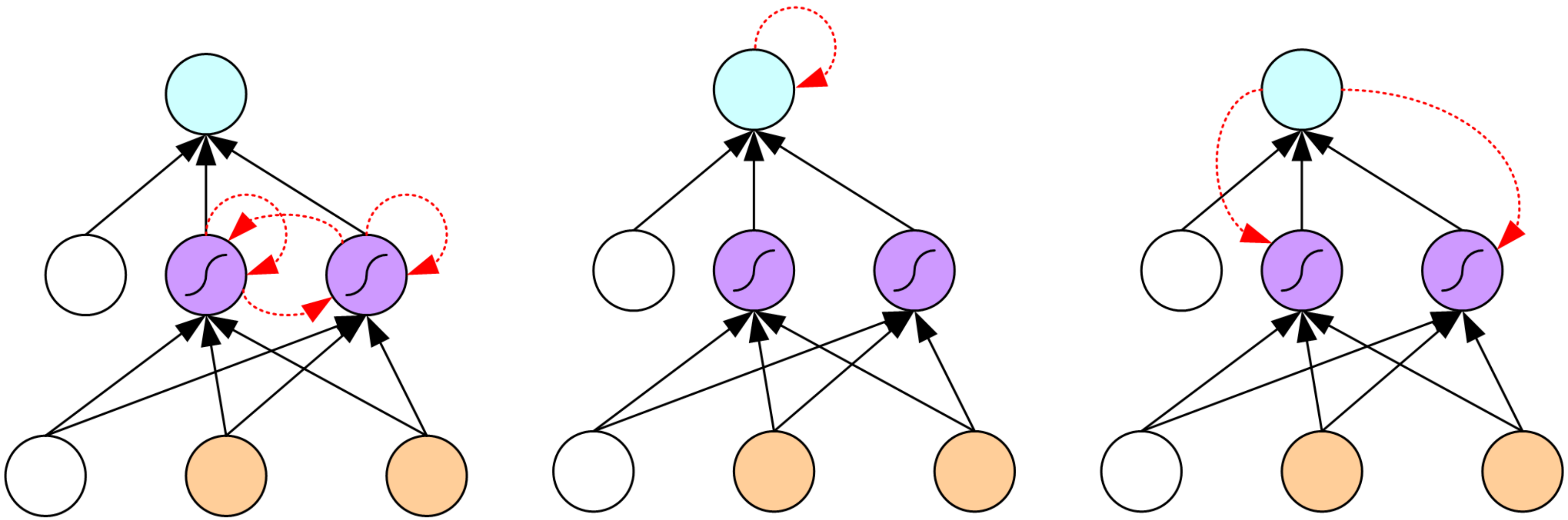
Learning Time

- Applications:
  - Sequence recognition: Speech recognition
  - Sequence reproduction: Time-series prediction
  - Sequence association
- Network architectures
  - Time-delay networks (Waibel et al., 1989)
  - Recurrent networks (Rumelhart et al., 1986)

Time-Delay Neural Networks

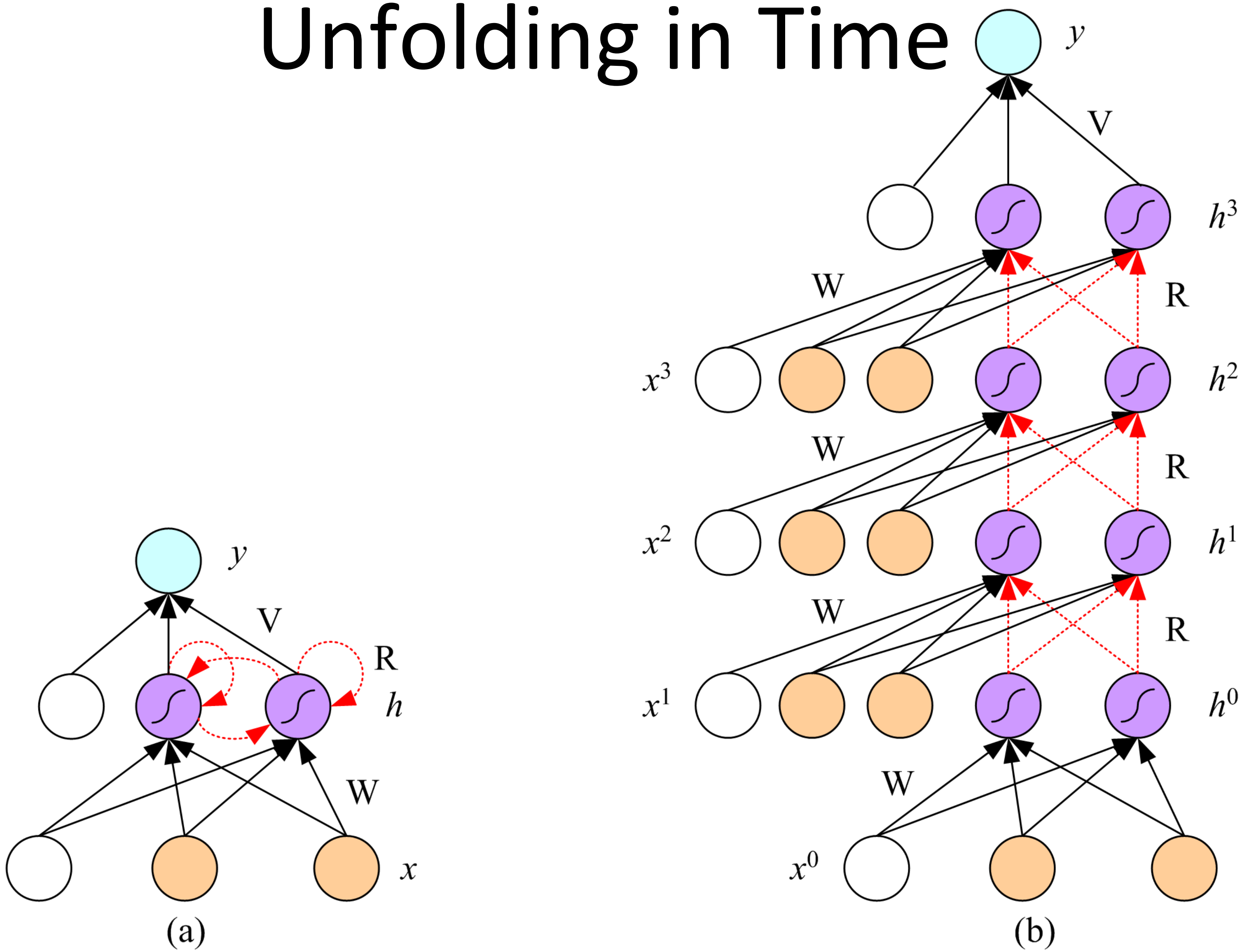


Recurrent Networks



(a) (b) (c)

Unfolding in Time

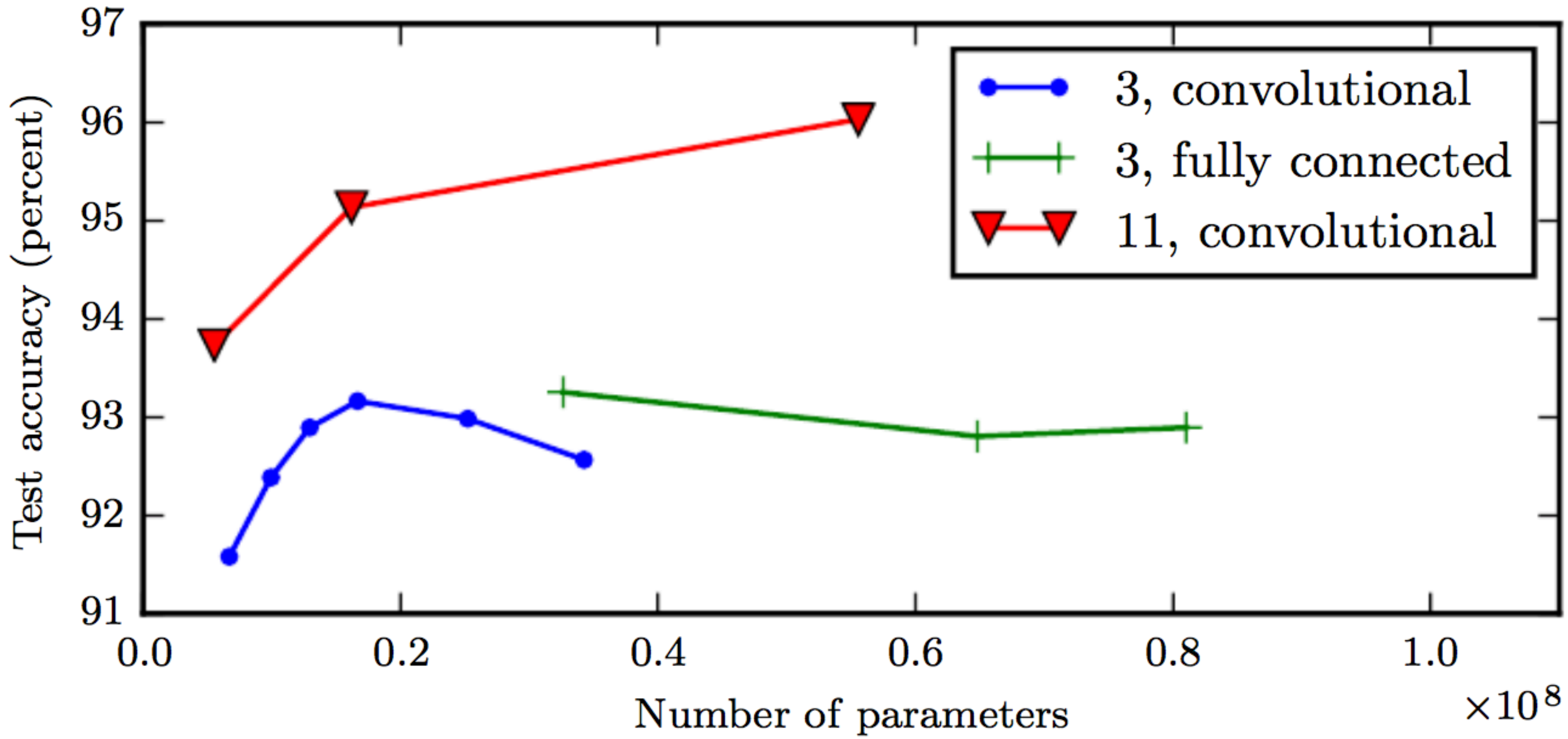


- Basic Concepts
  - supervised, unsupervised and semi-supervised learning
  - density estimation, classification, regression
  - error measures
  - optimization
  - linear and nonlinear models
- Perceptrons and Multilayer Perceptrons
  - perceptron
  - multilayer perceptron
  - training
  - overfitting and its prevention
  - learning from data and hints
  - neural networks for time series
- Deep Neural Networks
  - shallow networks' problems
  - optimization
  - regularization
  - convolution
  - hardware

Universal Approximation Theorem

- MLP with one hidden layer is a universal approximator (Hornik et al., 1989)
- But using multiple layers may lead to simpler networks
- There is an MLP that fits the data, but it might not be possible to find that MLP using the training algorithms we have.

How to Train a MLP with Lots of Data



Having a shallow network with a lot of hidden units may not learn as easy as a deep network.

Deep network may partition tasks into subtasks and learn easier.

(Goodfellow, 2014)

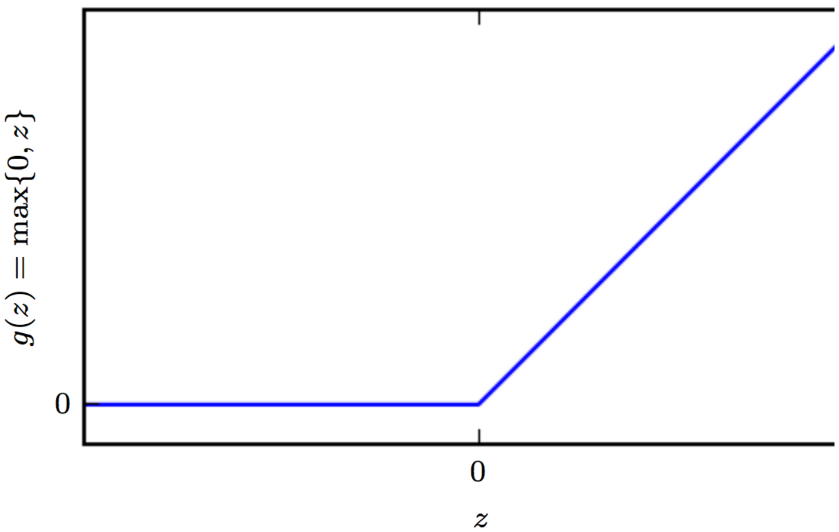
Vanishing Gradients

- Learning with a lot of layers is difficult.
- Since gradients are computed based on gradients of layers closer to the outputs, using chain rule, the gradients of weights closer to the inputs get very small.

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_h}\frac{\partial z_h}{\partial w_{hj}}$$

- Instead of sigmoid, RelU

(Rectified Linear Activation Function is used. RelU outputs are linear for positive inputs and 0 for negative inputs



$$\mathrm{Re}\,lU(z) = \max(0, z)$$

Deep Learning Libraries

torch | Caffe | theano

TensorFlow | NVIDIA CUDA | cuDNN

- When computing gradients, use graph representation of the MLP.
- Instead of single weights, use vectors/tensors of weights.
- Different libraries (Torch,Caffe vs Theano, TensorFlow) approaches to the gradient computation.

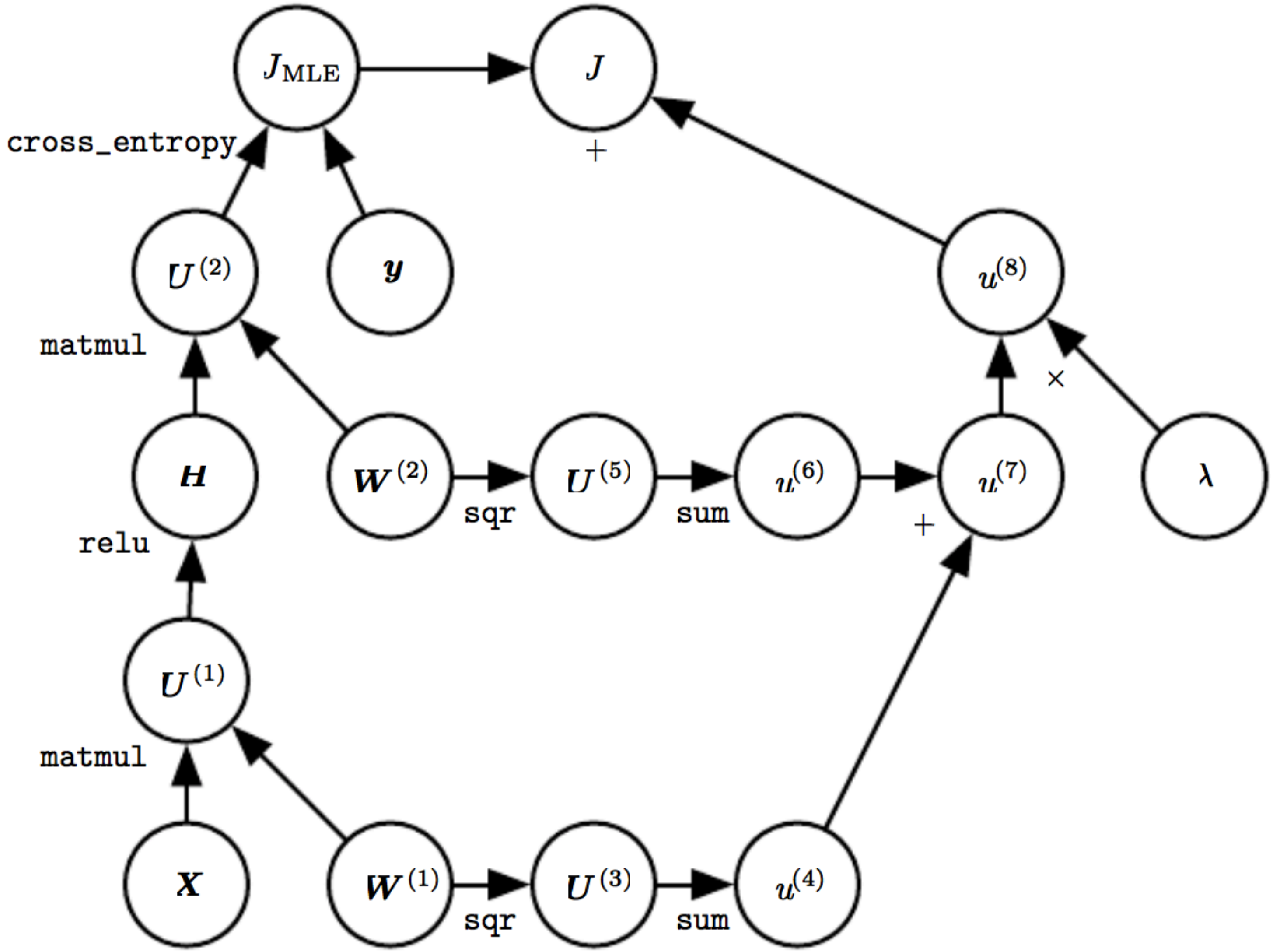


Figure 6.11: The computational graph used to compute the cost used to train our example of a single-layer MLP using the cross-entropy loss and weight decay.

Optimization

- Theano and Tensorflow approximate the Hessian Matrix (higher order derivatives) using Krylov Methods. Matrix inversions (therefore computations of eigenvectors/values are also approximated).
- Nesterov Momentum: similar to momentum, but evaluate gradient after applying the current velocity.
- Minibatches (in order to optimize for the hardware also)
- Conjugate gradient
- Polyak averaging
- Greedy supervised pretraining

Regularization for Deep Learning

- L1 and L2 (ridge regression/Tikhonov regularization) regularization
- Feature selection (results in weight elimination
- Early stopping
- Dataset augmentation: e.g. add rotated, blurred, scaled etc. images

Regularization for Deep Learning

- Ability to penalize weights at different layers differently (Srebro 2005, constrain norm of each column of the weight matrix of a neural network, rather than constrain the whole weight matrix)
- Noise robustness: dropout algorithm (approximates bagging, zero random some weights at each minibatch)
- Parameter tying and parameter sharing: reduce the number of parameters used by clustering/forcing similar valued parameters together and representing them using smaller number of bits. CNNs already do parameter sharing
- Adversarial training: training on perturbed images

Convolution

- Sparse interactions, parameter sharing and equivariant representations
- Parallel convolutions (feature extraction units), folllowed by RelU, followed by pooling to to obtain summary statistics of nearby cells.

Hardware: GPUs and TPUs to make matrix/weight operations faster

LAMBDA BLADE

If you're looking for an enterprise solution for cutting-edge Deep Learning performance and parallelization, the Lambda Blade is a necessity. Experience how profitable of an investment Deep Learning hardware can be for your team.

Designed for deployment in data centers, our 8 GPU configurations significantly boost Deep Learning performance and are ideal for remote access from multiple concurrent users.

$16,500

Includes FREE shipping

Ships in 3 days

source: https://lambdal.com

l at 1

LAMBDA SINGLE

SINGLE is an introduction to high performance Deep Learning hardware. Pre-installed is every framework you'll need — including Tensorflow, Torch, and Caffe.

Have any questions? Call us: 1 (650) 479-5530.

GPU

1x 1080 Ti

NVIDIA GeForce GTX 1080 Ti or Titan Xp. Pascal architecture

PROCESSOR

Intel i5

Intel™ Core i5-6500, 6M Cache, up to 3.60 GHz

STORAGE

250GB SSD

Configurable to 500 GB SATA SSD

RAM

16GB DDR4

Configurable to 32 GB DDR4 RAM.

$2,599

Includes FREE shipping

Ships in 3 days

source: https://lambdal.com

Teşekkürler.
Questions?

cataltepe@itu.edu.tr
zehra@tazi.io

[1] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

http://www.deeplearningbook.org

Alpaydin, Ethem. Introduction to machine learning. MIT press, 2014.