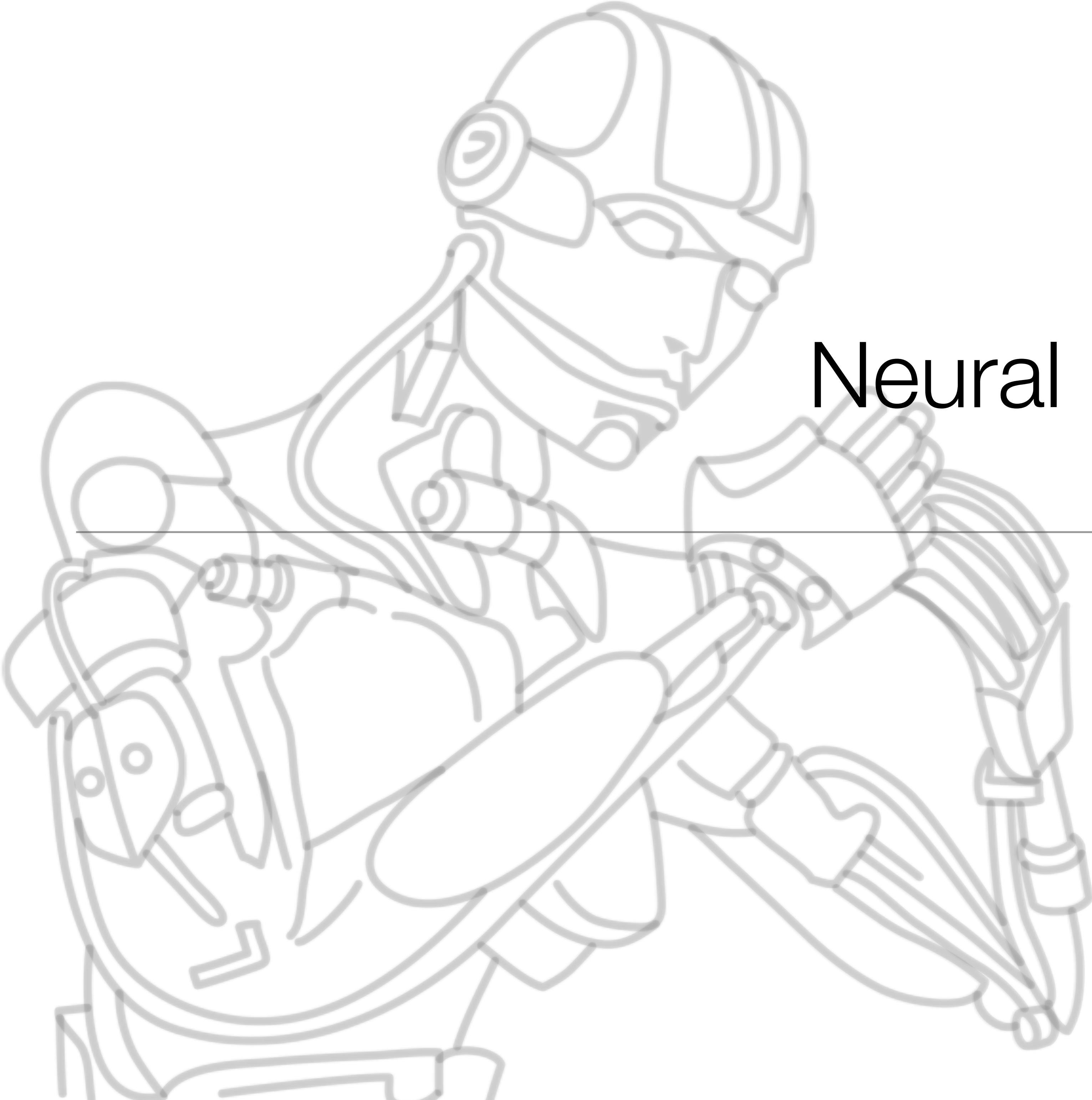


Neural Networks

Jan Peters
Filipe Veiga
Simone Parisi

TECHNISCHE UNIVERSITÄT DARMSTADT

Today’s agenda!

- Learn about Neural Networks!
- Covered Topics:
  - Single-Layer Perceptrons
  - Multi-Layer Perceptrons
  - Backpropagation Algorithm

- *Reading assignment:* Bishop 5.1-5.3, or Murphy 16.5.1-4

Questions which you need to be able to answer...

- How does logistic regression relate to neural networks?
- How do neural networks relate to the brain?
- What kind of functions can single layer neural networks learn?
- Why do two layers help?
- How many layers do you need to represent arbitrary functions?
- Why did they make such splash in the late 1980s?
- Why were Neural Networks abandoned in the 1970s? Why did that somewhat happen again in the mid-1990s?
- Why did they re-awaken in the 2010s?
- What is the biggest problem of neural networks?

From Linear Classification to Single Layer Neural Networks

1. From Linear Classification to Single Layer Neural Networks

2. Multi-Layer Perceptrons

TECHNISCHE UNIVERSITÄT DARMSTADT

Remember Logistic Regression?

- Model the class-posterior as:

$$p(C_1|\mathbf{x}) = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0)$$

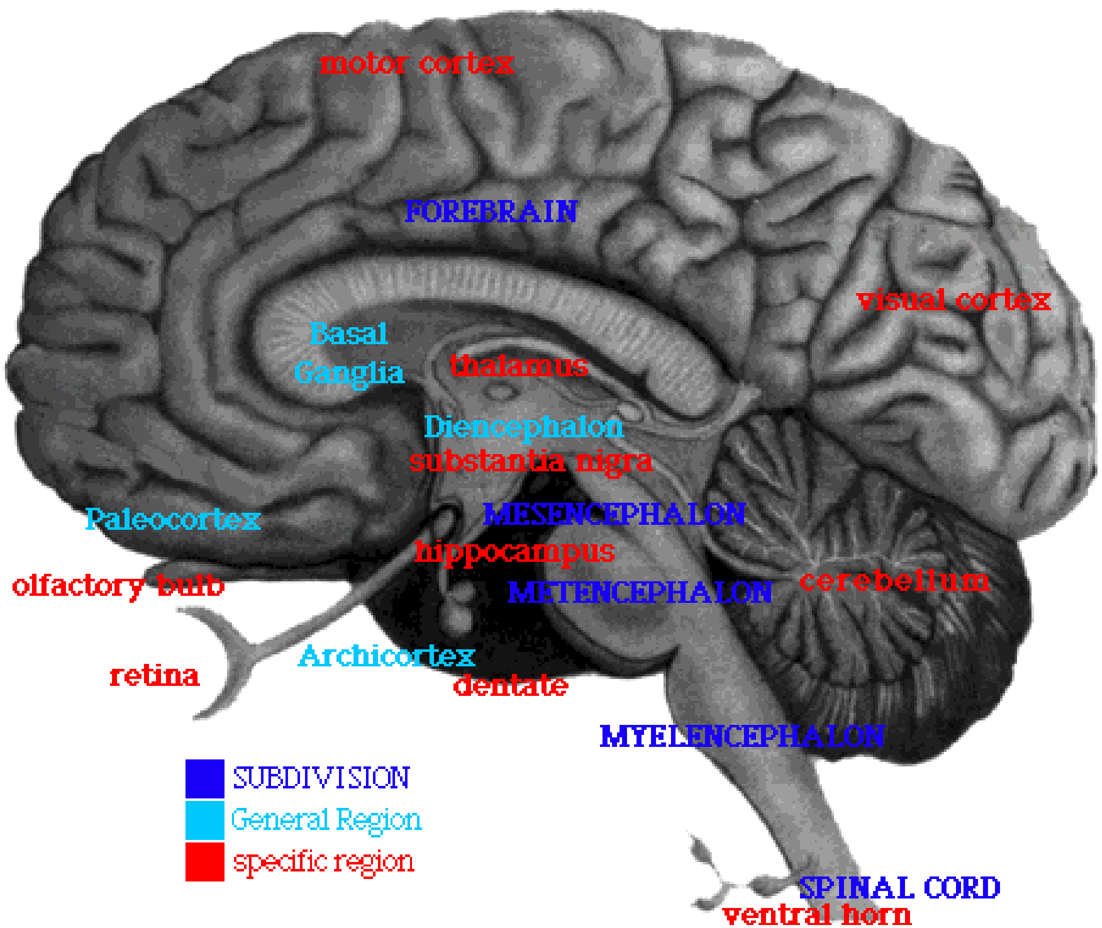
- Maximize the likelihood:

Assumption

$$y_i = \begin{cases} 1, & \mathbf{x}_i \text{ belongs to } C_2 \\ 0, & \mathbf{x}_i \text{ belongs to } C_1 \end{cases}$$

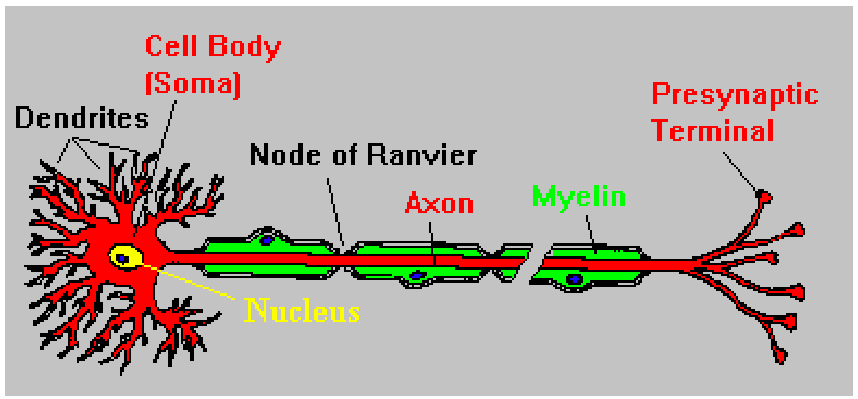
$$\begin{aligned} p(Y|X;\mathbf{w},w_0) &= \prod_{i=1}^{N} p(y_i|\mathbf{x}_i;\mathbf{w},w_0) \\ &= \prod_{i=1}^{N} p(C_1|\mathbf{x}_i;\mathbf{w},w_0)^{1-y_i} p(C_2|\mathbf{x}_i;\mathbf{w},w_0)^{y_i} \\ &= \prod_{i=1}^{N} \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + w_0)^{1-y_i} (1 - \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x}_i + w_0))^{y_i} \end{aligned}$$

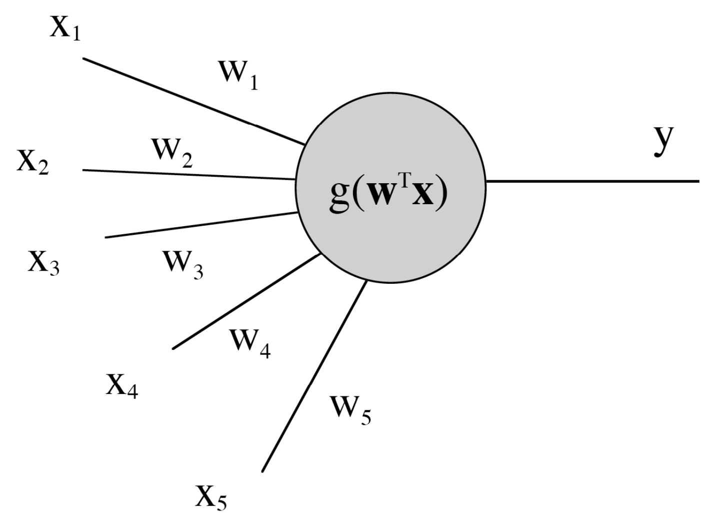
The Neural Network Metaphor

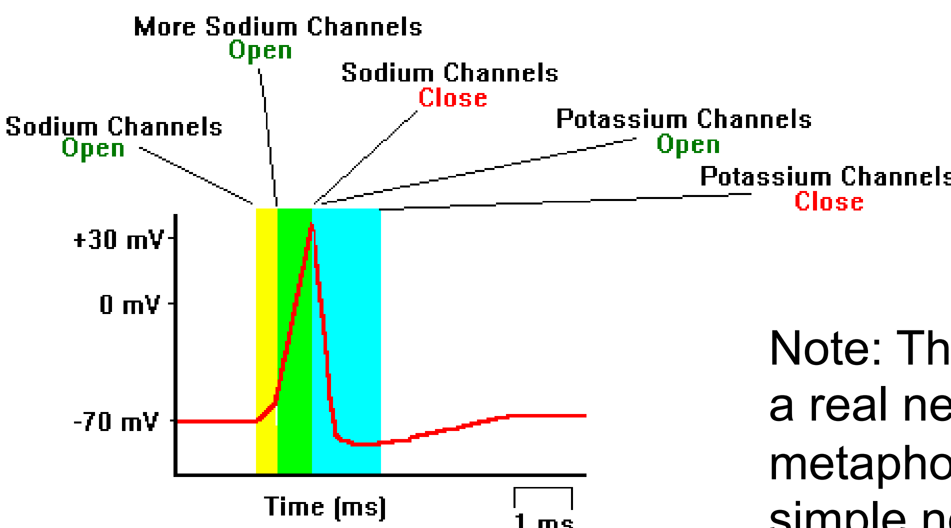


10^{11} neurons (processors), each with unknown computational power, and on average 1000-10000 connections

The Neural Network Metaphor







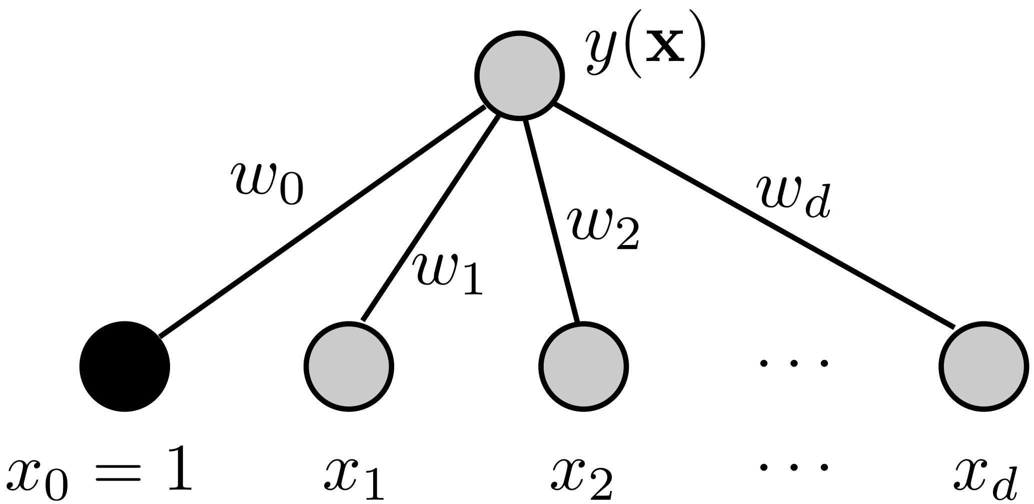
Note: This is a VERY simplified sketch of a real neuron–the connection to biology is more metaphorical than realistic. But even these simple neurons can do amazing computation!

Brief History of Neural Networks

- William James (1890): Describes (in words and figures) simple distributed networks and Hebbian Learning.
- McCulloch & Pitts (1943): Binary threshold units that perform logical operations (they proof universal computation!).
- Hebb (1949): Formulation of a physiological (local) learning rule
- Rosenblatt (1958): The Perceptron—a first real learning machine
- Widrow & Hoff (1960): ADALINE and the Widrow-Hoff supervised learning rule.
- Minsky & Papert (1969): The limitations of perceptron—the beginning of the "Neural Winter"
- [Outliers: v.d.Malsburg (1973): Selforganizing Maps, Grossberg (1980): Adaptive Resonance Theory, Hopfield (1982/84): Attractor Networks: A clean theory of pattern association and memory, Kohonen (1982): Self-organizing maps].

We can re-interpret it as a Neural Network!

- Single-layer network:



output layer (here: single node)

weights

input layer

Linear outputs (linear regression function):

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0 = \sum_{i=1}^{d} w_i x_i + w_0$$

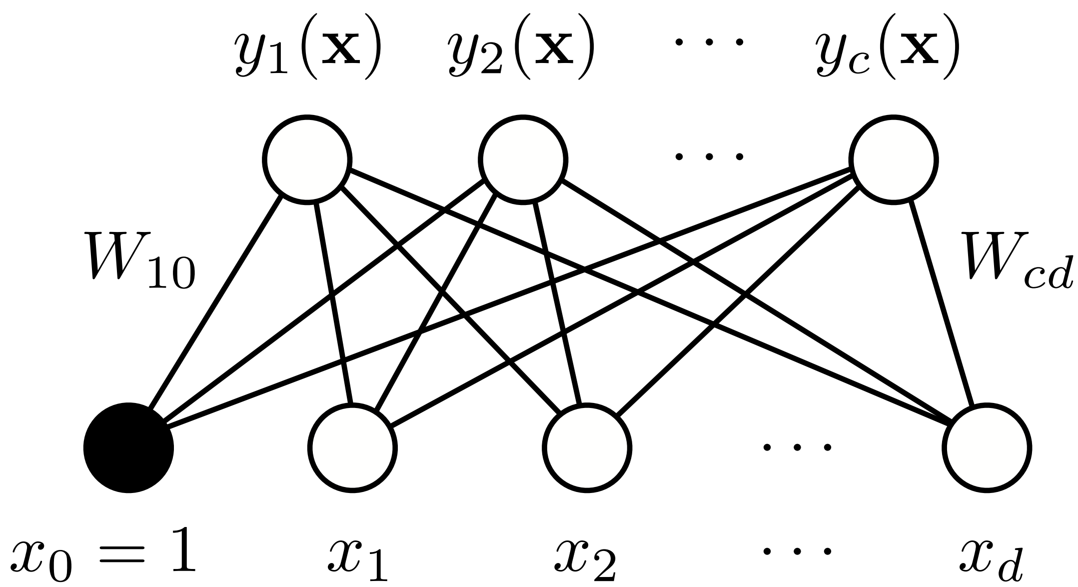
Logistic outputs:

$$y(\mathbf{x}) = \sigma(\mathbf{w}^{\mathrm{T}}\mathbf{x} + w_0)$$

Neural Networks

- Also called single-layer perceptron.
- 2 variants:
  - If we use a linear output node, we get a linear regression function.
  - If we use a sigmoid output node, we get something similar to logistic regression.
  - In either case, a classification can be obtained by taking the sign.
  - Nonetheless: At least classically, we don’t use maximum likelihood, but a different learning criterion.
- But the actual power comes from extensions:
  - Multi-class case
  - Multi-layer perceptron

Multi-Class Network



$$y_k(\mathbf{x}) = \sum_{i=0}^{d} W_{ki} x_i$$

or

$$y_k(\mathbf{x}) = \sigma\left(\sum_{i=0}^{d} W_{ki} x_i\right)$$

- Can be used to do multidimensional linear regression.
- But also multi-class linear classification.
- Nonlinear extension is straightforward.

Least-Squares Techniques

- Supervised learning of the weights W:
  - N training data points: $X = [\mathbf{x}^1, \dots, \mathbf{x}^N]$
  - c target values for each data point: $T_k = [t_k^1, \dots, t_k^N]$
  - Compute c outputs of the network: $y_k(\mathbf{x}^n; W)$
  - Minimize error function:

$$E(W) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} (y_k(\mathbf{x}^n; W) - t_k^n)^2$$

$$= \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} \left(\sum_{i=1}^{d} W_{ki} \phi_i(\mathbf{x}^n) - t_k^n \right)^2$$

assume arbitrary feature transformation

Gradient Descent

- Training a single-layer neural net with linear activation:

$$E(W) = \sum_{n=1}^{N} E^n(W) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} \left(\sum_{i=1}^{d} W_{ki} \phi_i(\mathbf{x}^n) - t_k^n \right)^2$$

with

$$E^n(W) = \frac{1}{2} \sum_{k=1}^{c} \left(\sum_{i=1}^{d} W_{ki} \phi_i(\mathbf{x}^n) - t_k^n \right)^2$$

$$\frac{\partial E^n(W)}{\partial W_{lj}} = \left(\sum_{i=1}^{d} W_{li} \phi_i(\mathbf{x}^n) - t_l^n \right) \phi_j(\mathbf{x}^n)$$

$$= \left(y_l(\mathbf{x}^n) - t_l^n \right) \phi_j(\mathbf{x}^n)$$

Gradient Descent

- "Batch learning":

$$W_{lj}^{(t+1)} = W_{lj}^{(t)} - \eta \left. \frac{\partial E(W)}{\partial W_{lj}} \right|_{W^{(t)}}$$

learning rate

  - The gradient is computed using all training data points:

$$\frac{\partial E(W)}{\partial W_{lj}} = \sum_{n=1}^{N} \frac{\partial E^n(W)}{\partial W_{lj}}$$

  - Computationally expensive!

Gradient Descent

- Sequential or pattern based update:

$$W_{lj}^{(t+1)} = W_{lj}^{(t)} - \eta \left. \frac{\partial E^n(W)}{\partial W_{lj}} \right|_{W^{(t)}}$$

learning rate (smaller)

where $$E(W) = \sum_{n=1}^{N} E^n(W)$$

- Computation of the gradient based on a single training data point:

$$\frac{\partial E^n(W)}{\partial W_{lj}}$$

- More efficient, but the gradient can be “noisy”.
- Intermediate solution: Use small training “batches”.

Gradient Descent

- Delta learning rule:

$$W_{lj}^{(t+1)} = W_{lj}^{(t)} - \eta(y_l(\mathbf{x}^n) - t_l^n)\phi_j(\mathbf{x}^n)$$

$$= W_{lj}^{(t)} - \eta\delta_l^n\phi_j(\mathbf{x}^n)$$

with $\delta_l^n = y_l(\mathbf{x}^n) - t_l^n$

- Other names:
  - LMS rule (least mean squares)
  - adaline rule
  - Widrow-Hoff rule

This is just like the algorithm for the classical perceptron!

Hence single-layer perceptron!

Gradient Descent

- Neural networks with non-linear, differentiable activation function (e.g. logistic networks):

$$y_k(\mathbf{x}^n) = g(a_k) = g\left(\sum_{i=1}^{d} W_{ki}\phi_i(\mathbf{x}^n)\right)$$

- Gradient descent:

$$\frac{\partial E^n(W)}{\partial W_{lj}} = g'(a_l)\left(y_l(\mathbf{x}^n) - t_l^n\right)\phi_j(\mathbf{x}^n)$$

  - Logistic neural network:

$$\sigma'(a) = \sigma(a)(1 - \sigma(a))$$

Gradient Descent

- Modified delta rule:

$$W_{lj}^{(t+1)} = W_{lj}^{(t)} - \eta g'(a_l)(y_l(\mathbf{x}^n) - t_l^n)\phi_j(\mathbf{x}^n)$$

$$= W_{lj}^{(t)} - \eta \delta_l^n \phi_j(\mathbf{x}^n)$$

with $$\delta_l^n = g'(a_l)(y_l(\mathbf{x}^n) - t_l^n)$$

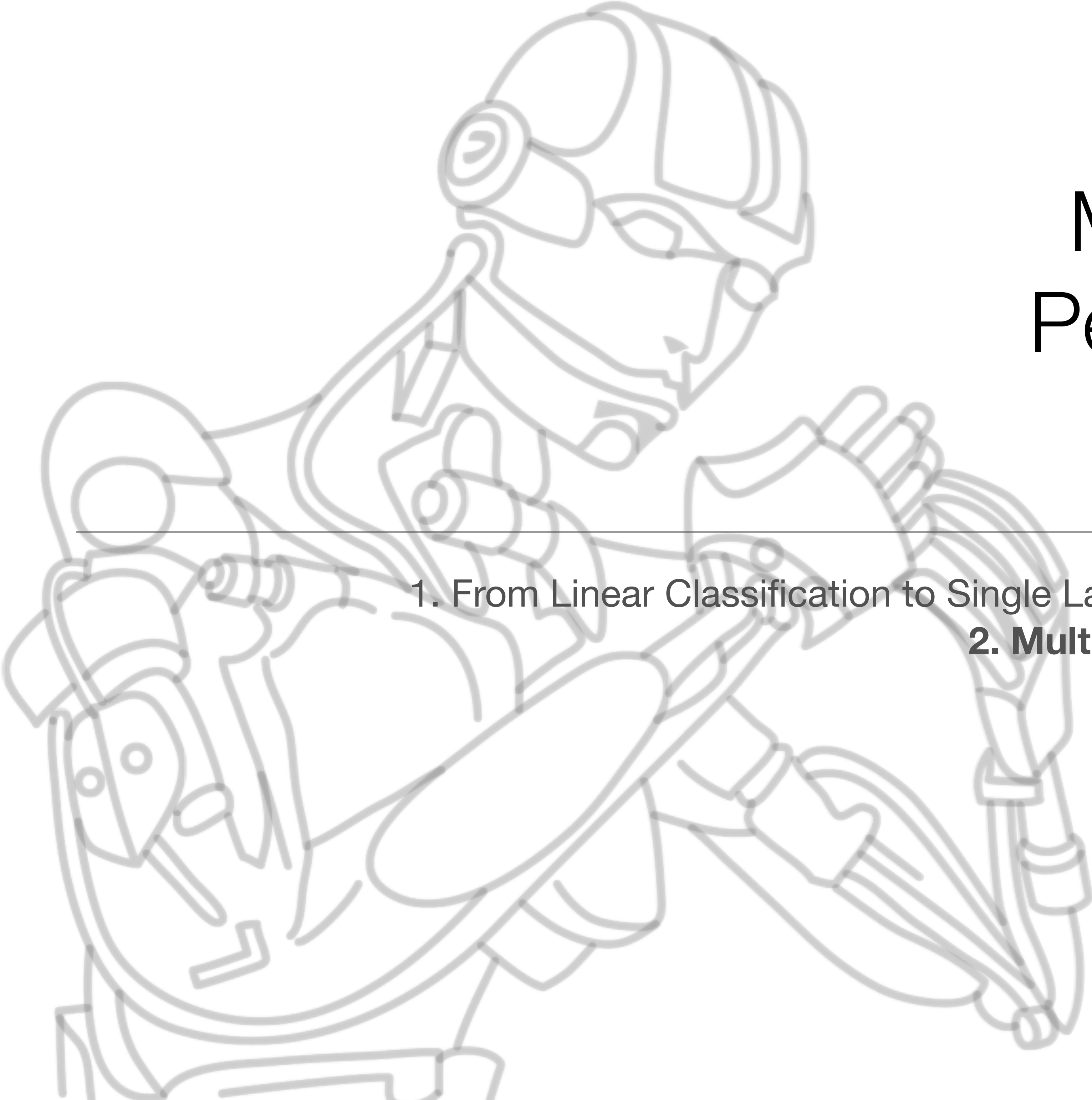
If you use the techniques from Lecture 4, you can be much more efficient!

Some Observations

- Once again, we are implicitly assuming a Gaussian distribution over the predictions:

$$p(t_k^n|\mathbf{x}^n, \mathbf{W}, \beta) = \mathcal{N}(t_k^n|y_k(\mathbf{x}^n; W), \beta^{-1})$$

- With a nonlinear activation function, the error function we minimize is non-convex:
  - Multiple local minima (often many).
  - We may get trapped in poor local optima.

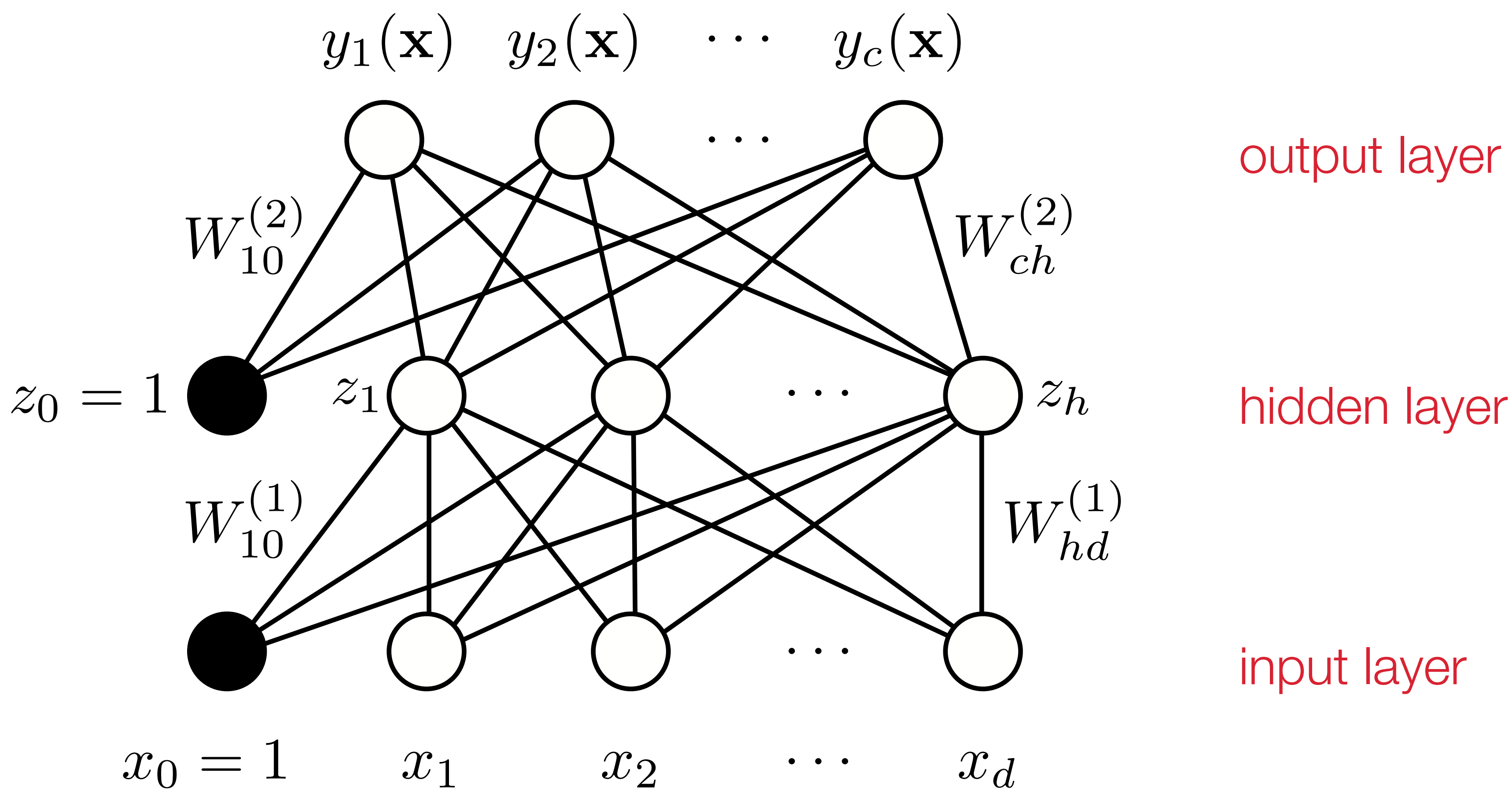


Multi-Layer Perceptrons

1. From Linear Classification to Single Layer Neural Networks
2. Multi-Layer Perceptrons

TECHNISCHE UNIVERSITÄT DARMSTADT

Multi-Layer Perceptron



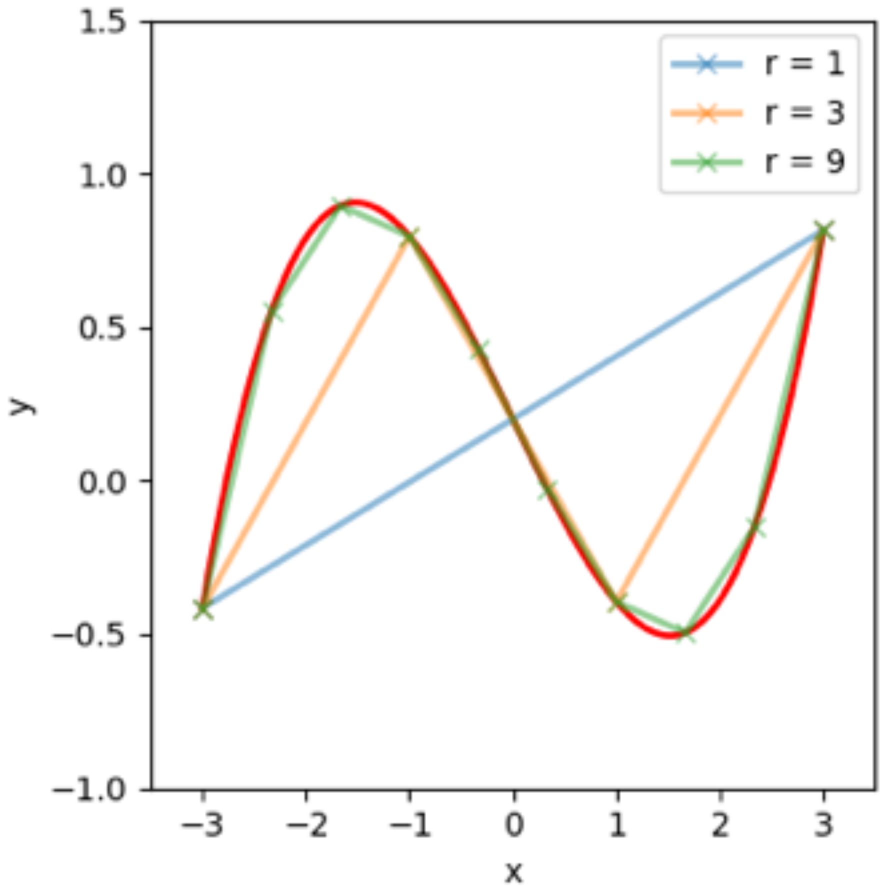
$$y_k(\mathbf{x}) = g^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)\right)$$

Multi-Layer Perceptron

$$y_k(\mathbf{x}) = g^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)\right)$$

- Activation functions $g^{(k)}$:
  - For example $g^{(2)}(a) = \sigma(a), \quad g^{(1)}(a) = a$
- The hidden layer can have an arbitrary number of nodes h.
  - There can also be multiple hidden layers.
- Universal approximators:
  - A 2-layer network (1 hidden layer) can approximate any continuous function of a compact domain arbitrarily well!
    (assuming sufficient hidden nodes)

Universal Approximation Theorem



$$O\left(\binom{n}{d}^{d(l-1)} n^d\right)$$

n = Number of Neurons per Laye
l = Number of Hidden Layers
d = Number of Inputs

$$O\left(\binom{n}{1}^{1(1-1)} n^1\right) = O(n)$$

$l = 1$
$d = 1$

$$O\left(\binom{n}{1}^{1(2-1)} n^1\right) = O(n^2)$$

$l = 2$
$d = 1$

$$O\left(\binom{n}{1}^{1(k-1)} n^1\right) = O(n^k)$$

$l = k$
$d = 1$

Kurt Hornik et. al., “Multilayer feedforward networks are universal approximators”, 1989
Guido Montufar et.al., “On the Number of Linear Regions of Deep Neural Networks”, 2014

Gradient Descent

- Squared error:

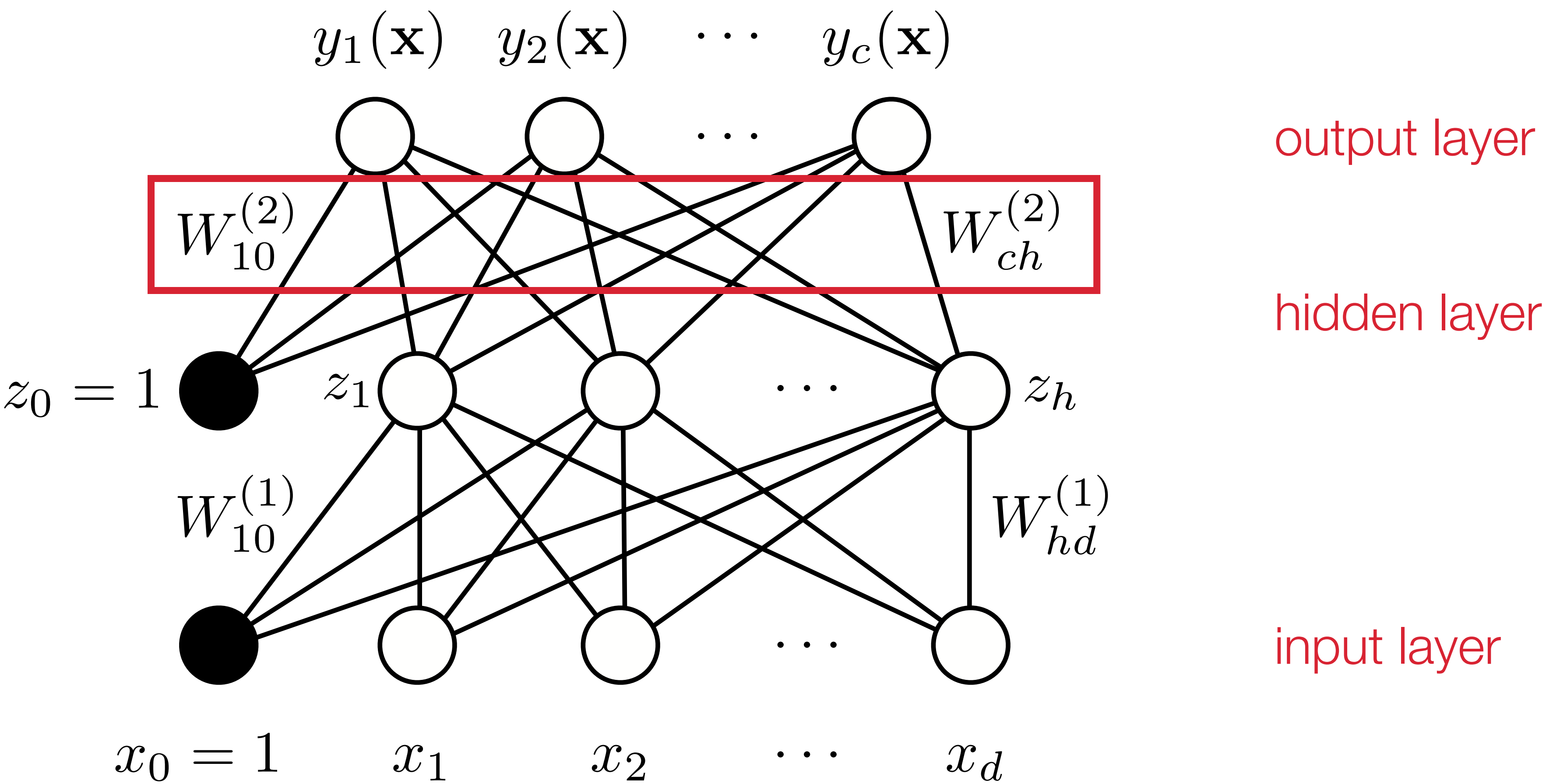
$$E(W) = \sum_{n=1}^{N} E^n(W) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{c} (y_k(\mathbf{x}^n) - t_k^n)^2$$

$$y_k(\mathbf{x}^n) = g^{(2)} \left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)} \left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j^n \right) \right)$$

$$= g^{(2)} \left(\sum_{i=0}^{h} W_{ki}^{(2)} z_i(\mathbf{x}^n) \right)$$

with $$z_i(\mathbf{x}^n) = g^{(1)} \left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j^n \right)$$

Multi-Layer Perceptron



$$y_k(\mathbf{x}) = g^{(2)}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j\right)\right)$$

Gradient Descent

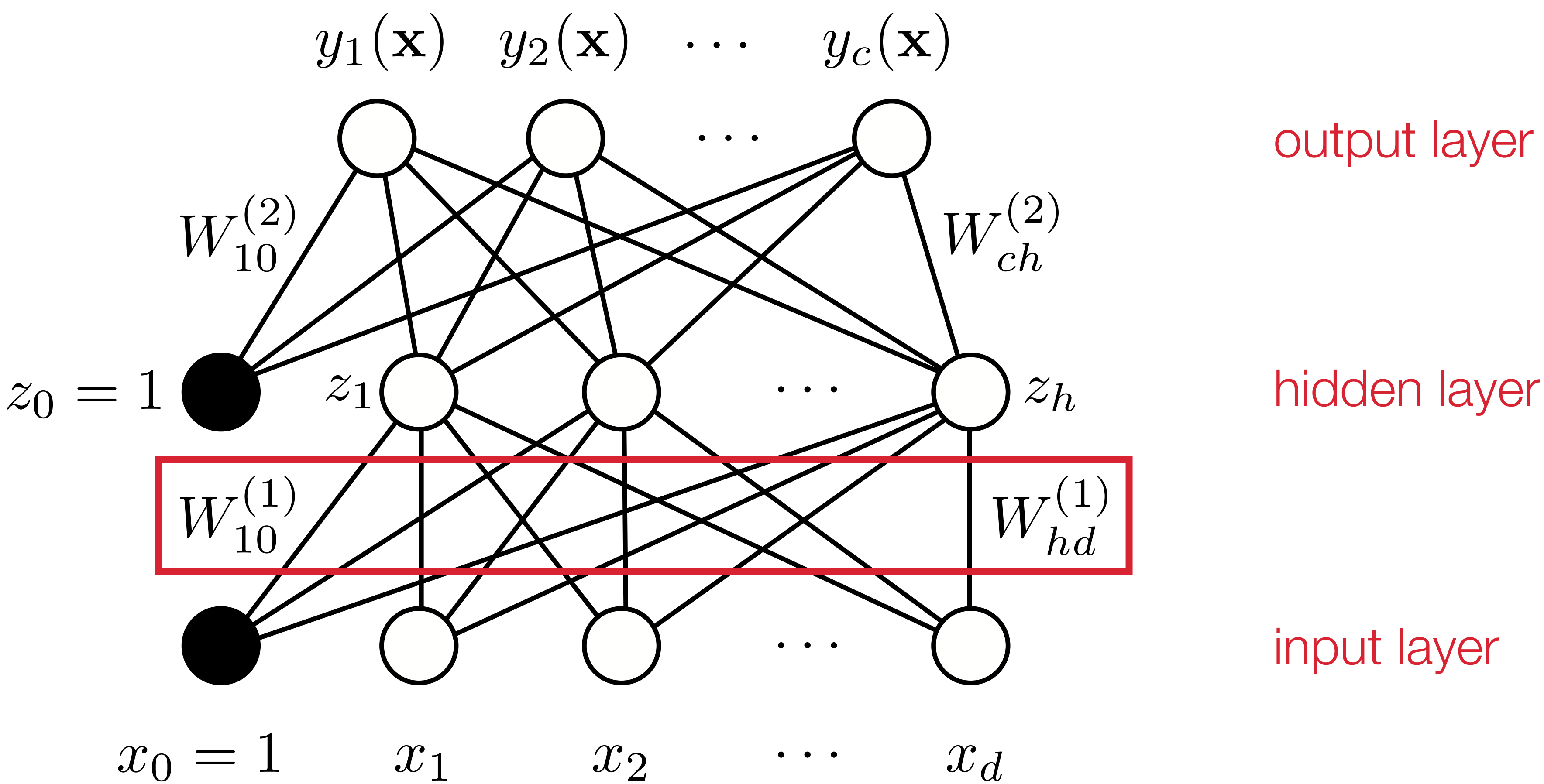
- Assuming linear activation $g^{(2)}(a) = a$:

$$E^n(W) = \frac{1}{2}\sum_{k=1}^{c}\left(\sum_{i=1}^{h} W_{ki}^{(2)} z_i(\mathbf{x}^n) - t_k^n\right)^2$$

$$\begin{aligned}
\frac{\partial E^n(W)}{\partial W_{lj}^{(2)}} &= \left(\sum_{i=1}^{h} W_{li}^{(2)} z_i(\mathbf{x}^n) - t_l^n\right) z_j(\mathbf{x}^n) \\
&= \left(y_l(\mathbf{x}^n) - t_l^n\right) z_j(\mathbf{x}^n) \\
&= \delta_l^n z_j(\mathbf{x}^n)
\end{aligned}$$

with $\delta_l^n = y_l(\mathbf{x}^n) - t_l^n$

Multi-Layer Perceptron



$$y_k(\mathbf{x}) = g^{(2)} \left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)} \left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j \right) \right)$$

Gradient Descent

$$E^n(W) = \frac{1}{2}\sum_{k=1}^{c}\left(\sum_{i=0}^{h} W_{ki}^{(2)} g^{(1)}\left(\sum_{j=0}^{d} W_{ij}^{(1)} x_j^n\right) - t_k^n\right)^2$$

$$\frac{E^n(W)}{\partial W_{lm}^{(1)}} = x_m^n z_l'(\mathbf{x}^n)\sum_{k=1}^{c}\delta_k^n W_{kl}^{(2)}$$

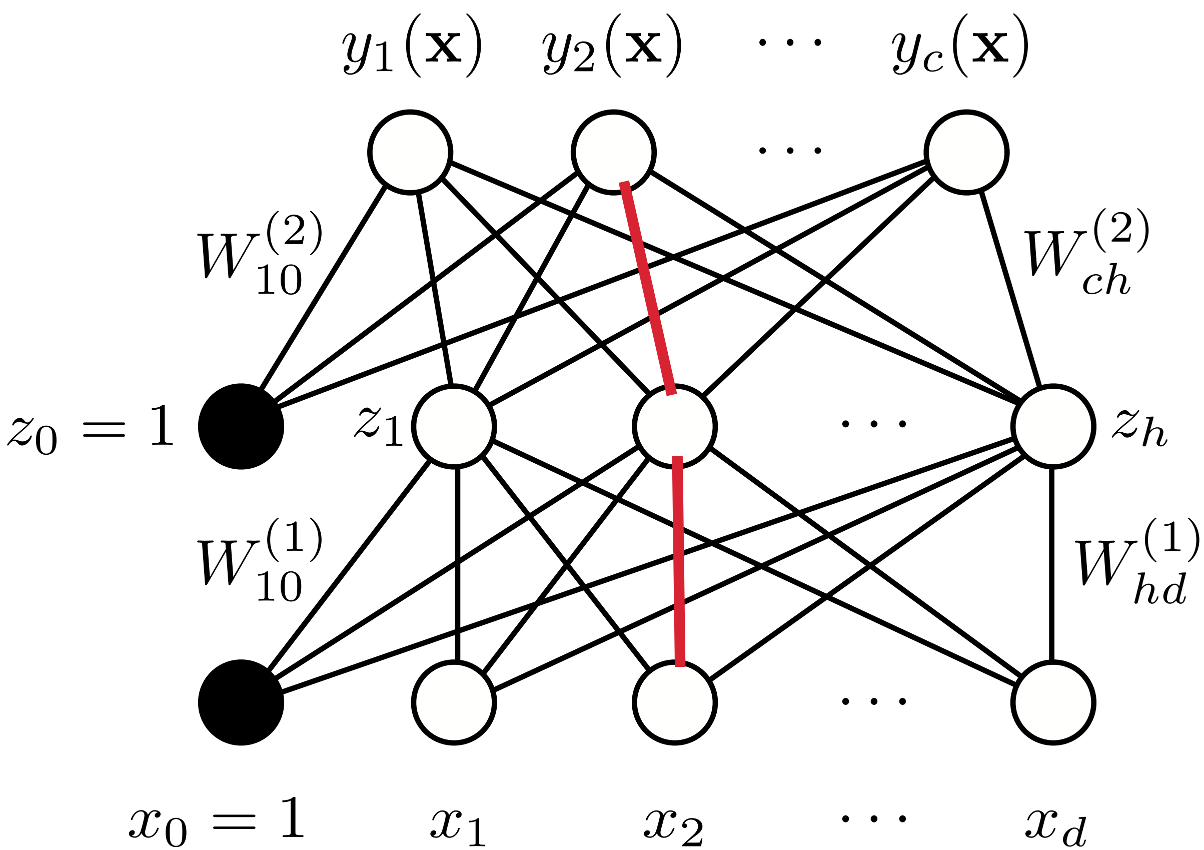
$$= x_m^n z_l'(\mathbf{x}^n)\hat{\delta}_l^n$$

with $\hat{\delta}_l^n = \sum_{k=1}^{c}\delta_k^n W_{kl}^{(2)}$

Gradient Descent

- Intuitively:
  - Step 1: Forward pass

Forward propagation



Compute output unit activations:

$$y_k(\mathbf{x}^n)$$

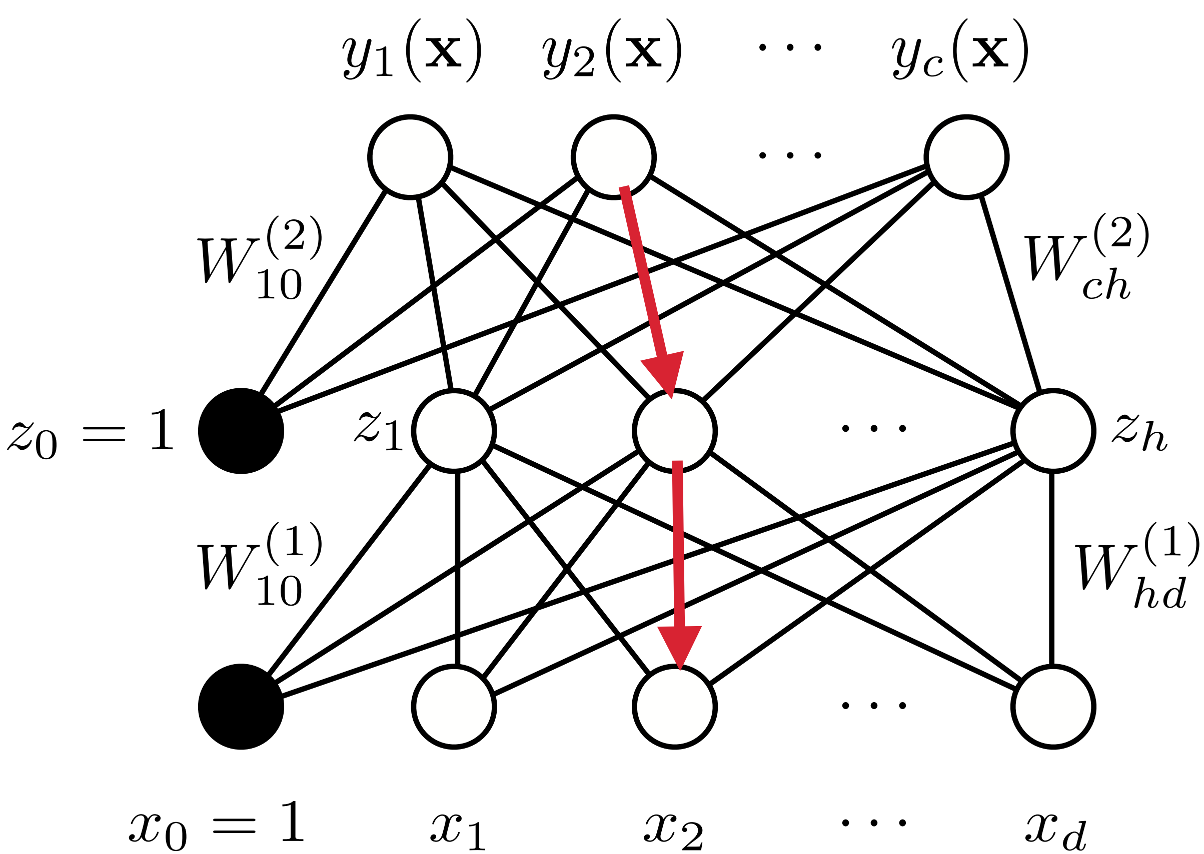
Compute hidden unit activations:

$$z_i(\mathbf{x}^n)$$

Gradient Descent

- Intuitively:
  - Step 2: Backward pass

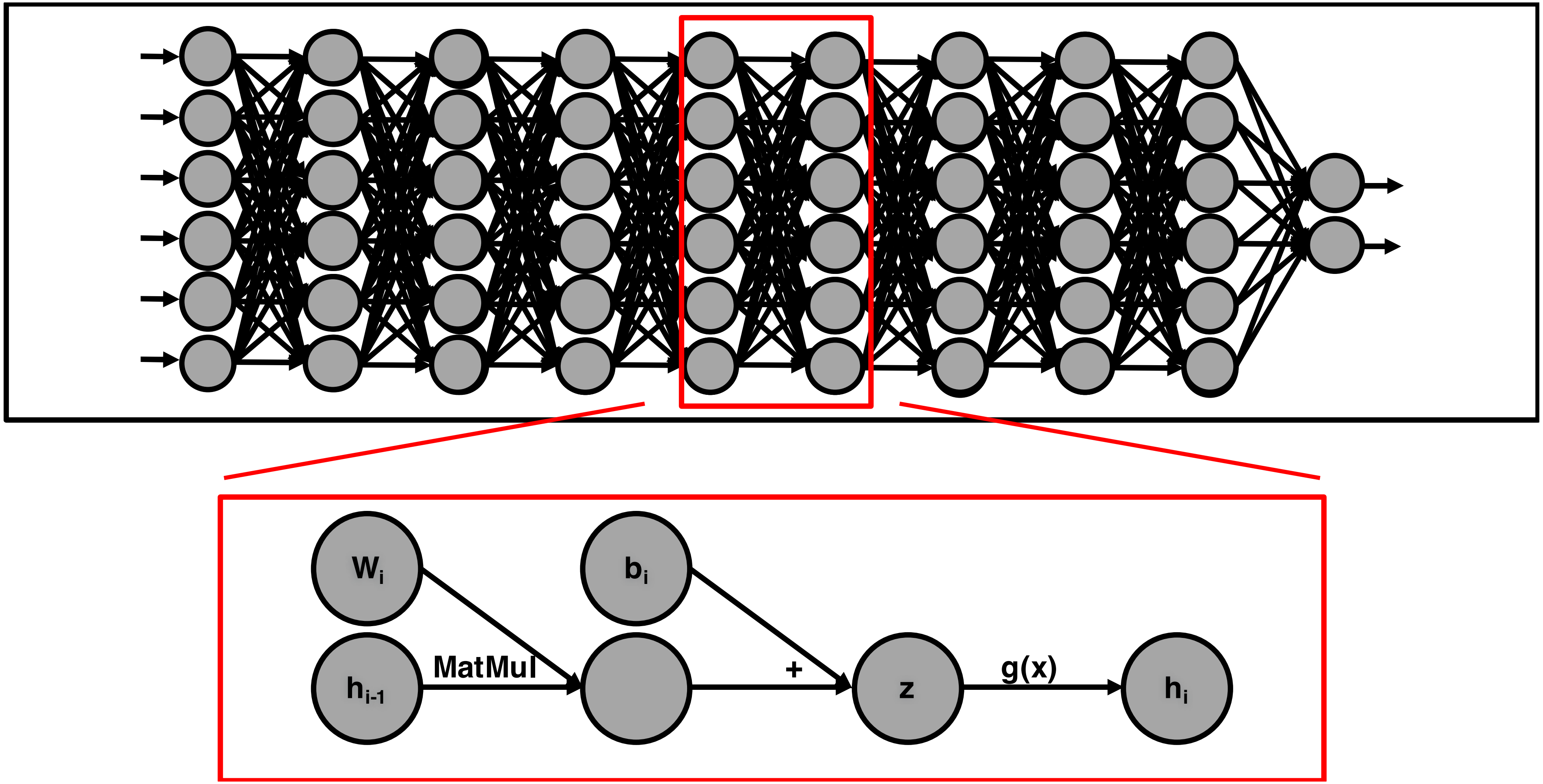
Backward propagation
"Backprop"



Compute output error:
δ_k^n

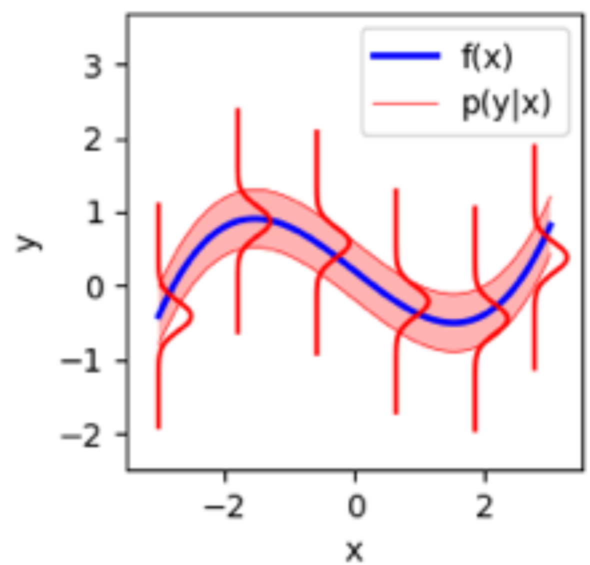
Compute hidden error:
$\hat{\delta}_i^n$

Computational Graphs



Output Neuron Types

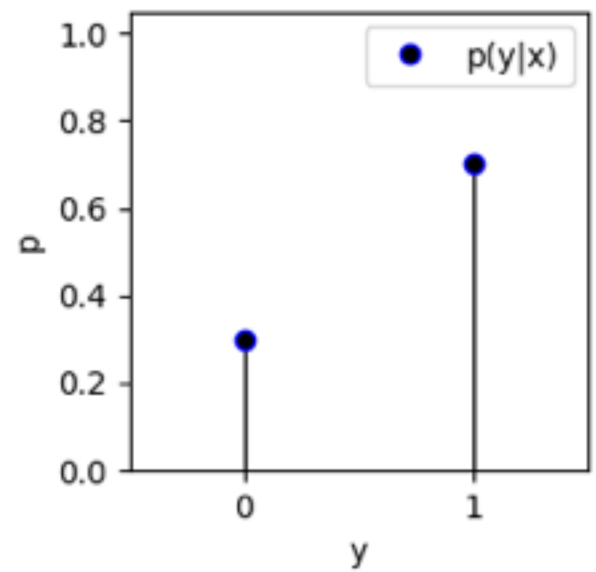
Linear Neuron



$$g(\mathbf{z}_i) = \mathbf{z}_i$$

$$p(y \mid z) = \mathcal{N}(y - z, I)$$

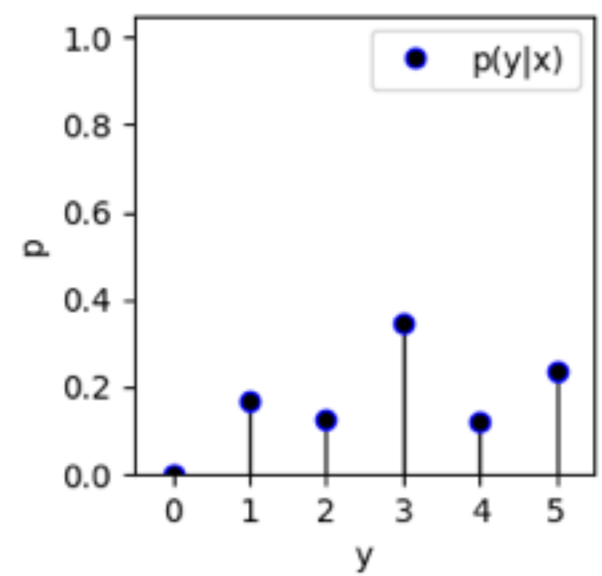
Sigmoid Neuron



$$g(z) = \sigma(z) = \frac{1}{1 + e^{-z}}$$

$$p(y \mid z) = \sigma((2y - 1)z)$$

Softmax Neuron



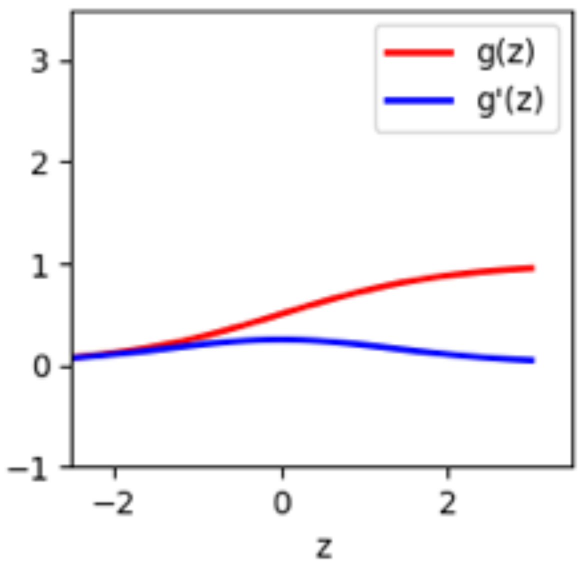
$$g(\mathbf{z}_i) = \frac{\exp \mathbf{z}_i}{\sum_j \exp z_j}$$

$$p(y = i \mid \mathbf{z}) = g(\mathbf{z}_i)$$

Slides by Michael Lutter

Hidden Neuron Types

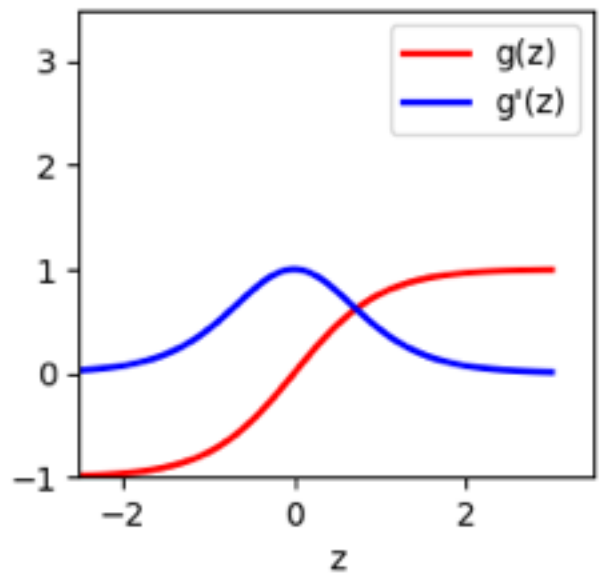
Sigmoid Neuron



$$g(\mathbf{z}_i) = \sigma(\mathbf{z}_i) = \frac{1}{1 + e^{-\mathbf{z}_i}}$$

$$g'(\mathbf{z}_i) = \sigma(\mathbf{z}_i)\left(1 - \sigma(\mathbf{z}_i)\right)$$

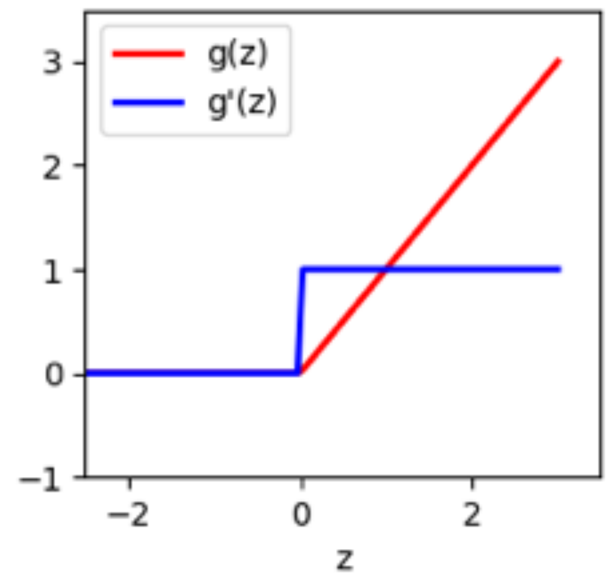
Tanh Neuron



$$g(\mathbf{z}_i) = \tanh(\mathbf{z}_i)$$

$$g'(\mathbf{z}_i) = 1 - \tanh(\mathbf{z}_i)^2$$

ReLu Neuron

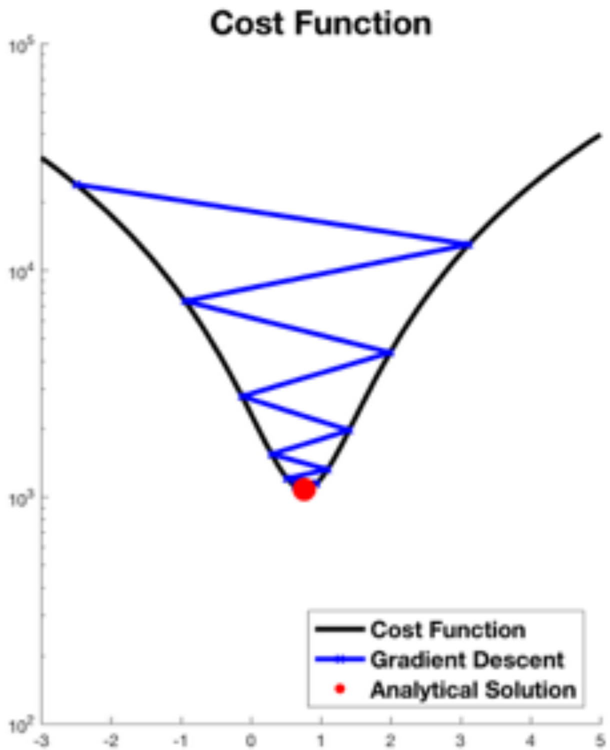


$$g(\mathbf{z}_i) = \max(\mathbf{0}, \mathbf{z}_i)$$

$$g'(\mathbf{z}_i) = \begin{cases} 1, & \mathbf{z}_i \geq 0 \\ 0, & \mathbf{z}_i < 0 \end{cases}$$

Slides by Michael Lutter

Gradient Descent



Optimization Objective:

$$\theta^* = \operatorname{argmin} J(\theta)$$
$$\theta_{i+1} = \theta_i^{\theta} + \Delta\theta_i = \theta_i - \alpha\, \nabla_{\theta_i} J(\theta)$$

Cost Functions:

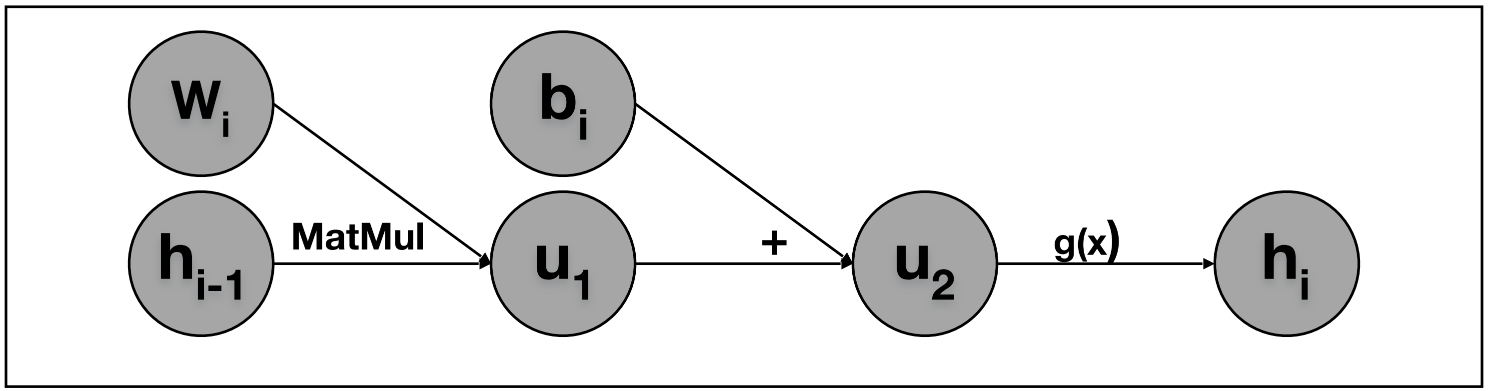
$$J(\theta) = \underset{p_d}{E}\{|y - f(x,\theta)|_1\} \rightarrow \text{Median of } p(y \mid z)$$

$$J(\theta) = \underset{p_d}{E}\{|y - f(x,\theta)|_2\} \rightarrow \text{Mean of } p(y \mid z)$$

$$J(\theta) = \underset{p_d}{E}\{-\log(p_m(y \mid x,\theta))\}$$

Slides by Michael Lutter

Backpropagation



$$\boldsymbol{u}_0 = \boldsymbol{h}_{i-1} \qquad \frac{d}{d\boldsymbol{u}_0}\boldsymbol{u}_1 = \boldsymbol{W}_i^T \qquad \frac{d}{dW_i}\boldsymbol{u}_1 = [\boldsymbol{u}_0 \quad \dots \quad \boldsymbol{u}_0]$$

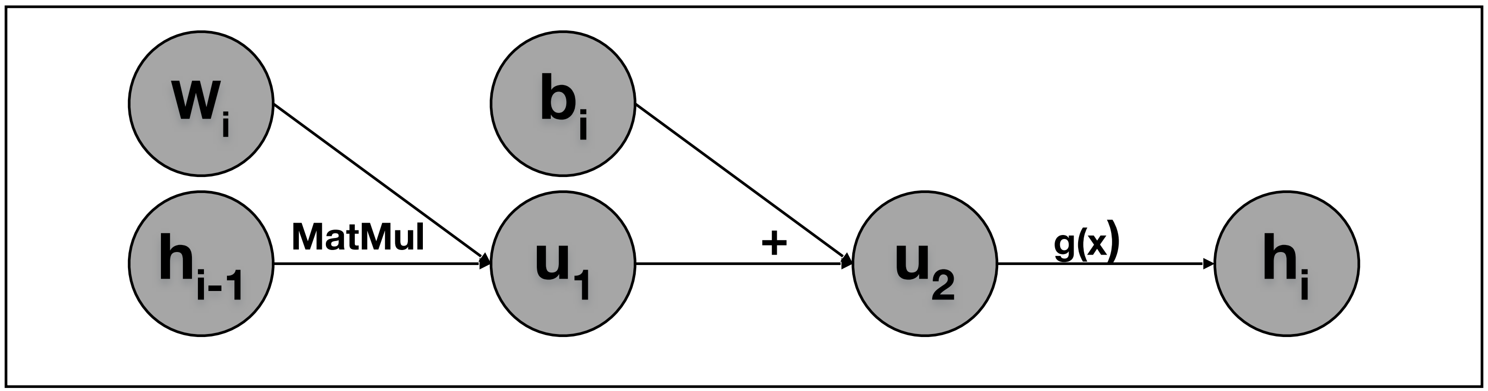
$$\boldsymbol{u}_1 = \boldsymbol{W}_i^T \boldsymbol{u}_0 \qquad \frac{d}{d\boldsymbol{u}_1}\boldsymbol{u}_2 = \boldsymbol{I} \qquad \frac{d}{d\boldsymbol{b}_i}\boldsymbol{u}_2 = \boldsymbol{I}$$

$$\boldsymbol{u}_2 = \boldsymbol{u}_1 + \boldsymbol{b}_i \qquad \frac{d}{d\boldsymbol{u}_2}\boldsymbol{u}_3 = \boldsymbol{g}'(\boldsymbol{u}_2)$$

$$\boldsymbol{u}_3 = g(\boldsymbol{u}_2) = \boldsymbol{h}_i$$

Slides by Michael Lutter

Backpropagation



$$\nabla_{\boldsymbol{b}_i} J(\theta) = \frac{d\boldsymbol{u}_2}{d\boldsymbol{b}_i}\frac{d\boldsymbol{u}_3}{d\boldsymbol{u}_2} \odot \nabla J_{\boldsymbol{u}_3} = \boldsymbol{I}\ \boldsymbol{g}'(\boldsymbol{u}_2) \odot \nabla J_{\boldsymbol{u}_3}$$

$$\nabla_{\boldsymbol{W}_i} J(\theta) = \frac{d\boldsymbol{u}_1}{d\boldsymbol{W}_1}\frac{d\boldsymbol{u}_2}{d\boldsymbol{u}_1}\frac{d\boldsymbol{u}_3}{d\boldsymbol{W}_i} \odot \nabla_{\boldsymbol{u}_3} J = (\boldsymbol{g}'(\boldsymbol{u}_2) \odot \nabla J_{\boldsymbol{u}_3})\ \boldsymbol{u}_0^T$$

$$\nabla_{\boldsymbol{u}_0} J(\theta) = \frac{d\boldsymbol{u}_1}{d\boldsymbol{u}_0}\frac{d\boldsymbol{u}_2}{d\boldsymbol{u}_1}\frac{d\boldsymbol{u}_3}{d\boldsymbol{u}_2} \odot \nabla J_{\boldsymbol{u}_3} = \boldsymbol{W}_i^T\ \boldsymbol{g}'(\boldsymbol{u}_2) \odot \nabla J_{\boldsymbol{u}_3}$$

Slides by Michael Lutter

Backpropagation Algorithm

- Multi-layer perceptrons are usually trained using back-propagation:
  - Non-convex, many local optima.
  - Can get stuck in poor local optima.
  - The design of a working backprop algorithm is somewhat of a “black art”.
  - Because of that, their use has diminished somewhat.

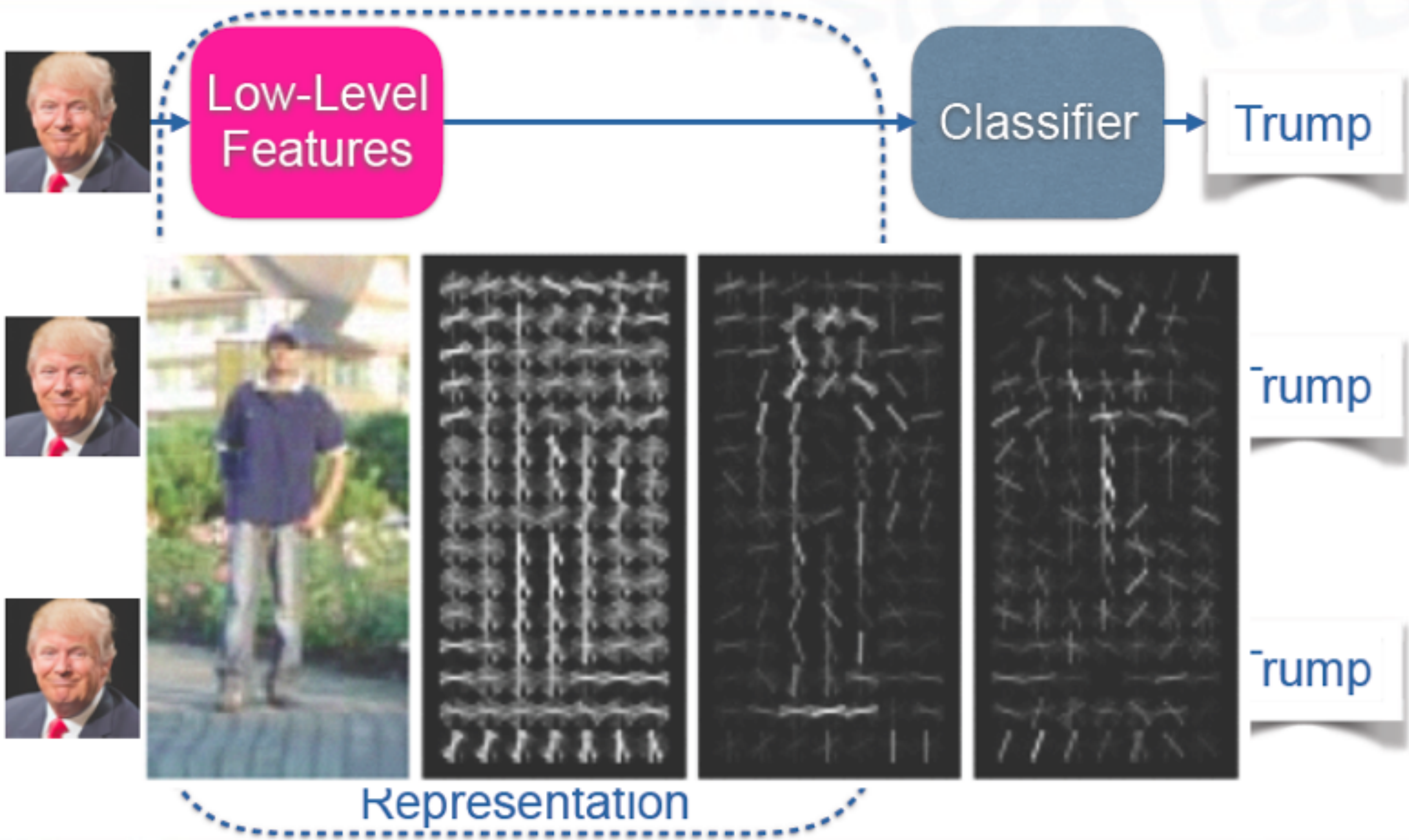
- Nonetheless:
  - When these models work, they can work very well!

Robot Navigation

- Neural network controlling the steering angle of a 4-wheeled robot:

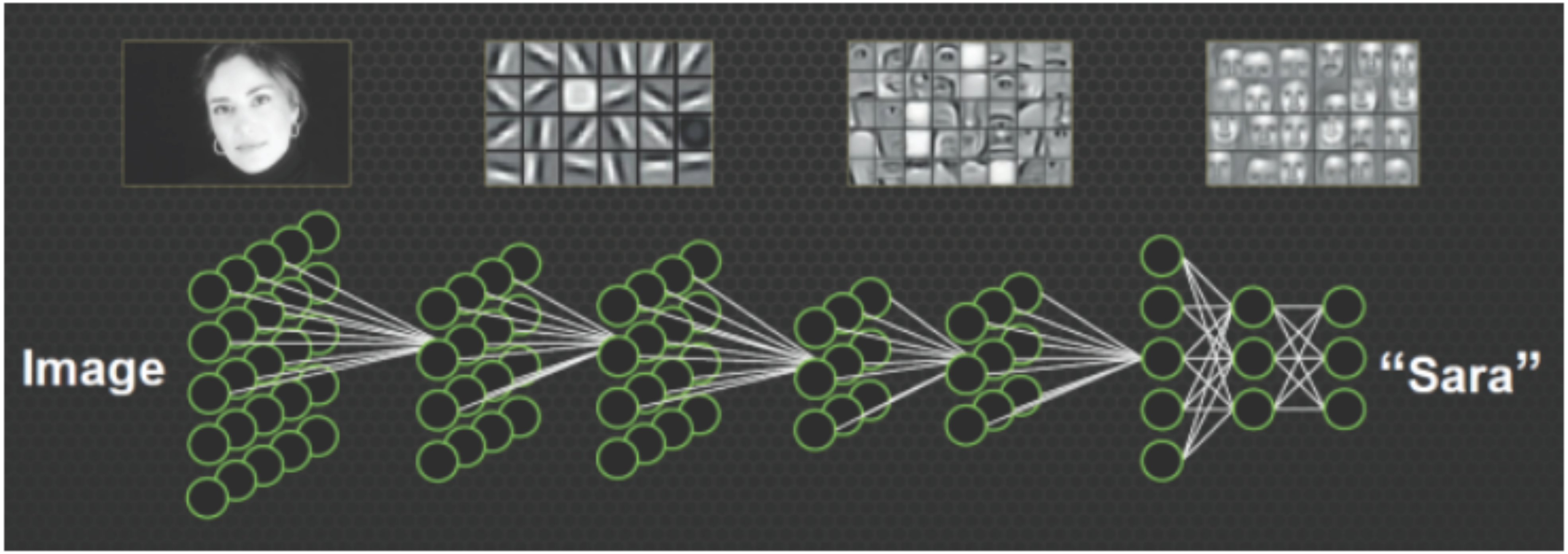
[LeCun]

From Data to Representations to Interpretations



Learning Deep Image Feature Hierarchies

- Deep learning gives ~ 10% improvement on ImageNet
  - 1.2M images
  - 1000 categories
  - 60 million parameters



Impact of Deep Learning in Computer Vision

- 2012-2014 classification results in ImageNet

CNN
non-CNN

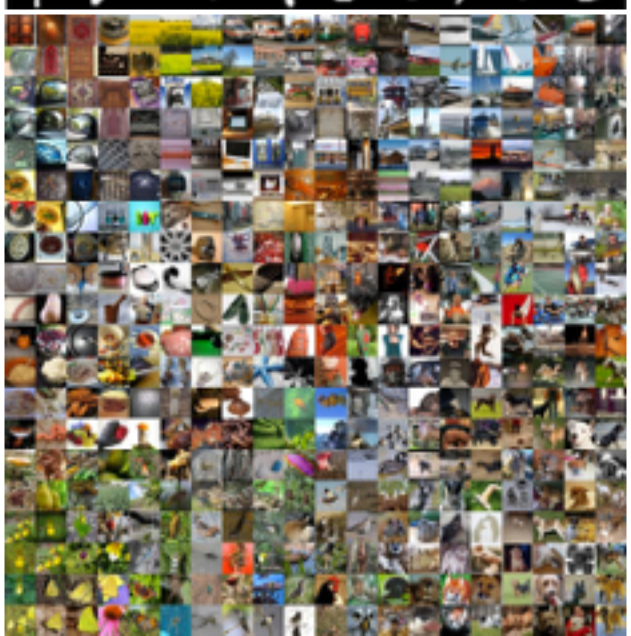
2012 Teams	%error
Supervision (Toronto)	15.3
ISI (Tokyo)	26.1
VGG (Oxford)	26.9
XRCE/INRIA	27.0
UvA (Amsterdam)	29.6
INRIA/LEAR	33.4

2013 Teams	%error
Clarifai (NYU spinoff)	11.7
NUS (singapore)	12.9
Zeiler-Fergus (NYU)	13.5
A. Howard	13.5
OverFeat (NYU)	14.1
UvA (Amsterdam)	14.2
Adobe	15.2
VGG (Oxford)	15.2
VGG (Oxford)	23.0

2014 Teams	%error
GoogLeNet	6.6
VGG (Oxford)	7.3
MSRA	8.0
A. Howard	8.1
DeeperVision	9.5
NUS-BST	9.7
TTIC-ECP	10.2
XYZ	11.2
UvA	12.1

- 2015 results: MSR under 3.5% error using 150 layers!

Status Quo – Image Classification



MNIST

10 classes
70k Images
0.20 % Human Performance
0.21 % Best Performance

CIFAR 10

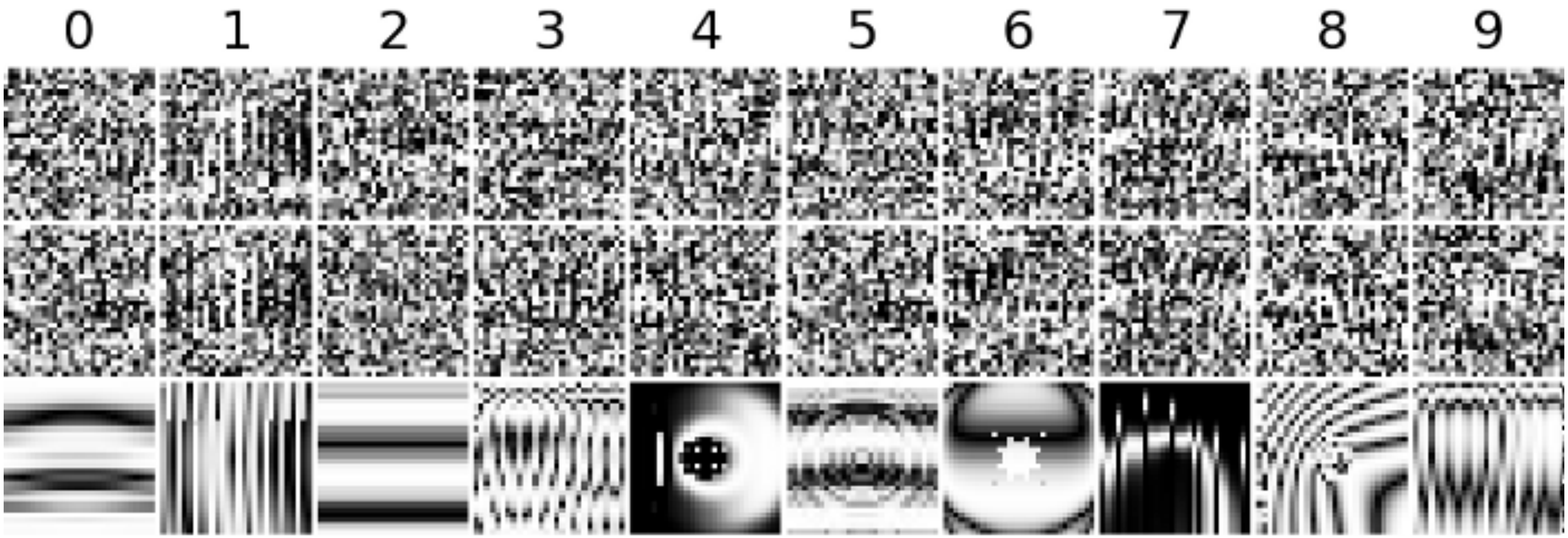
10 classes
60k Images
6.00 % Human Performance
4.41 % Best Performance

Imagenet

1000 classes
1200k Images
5.10 % Human Performance
4.80 % Best Performance

Slides by Michael Lutter

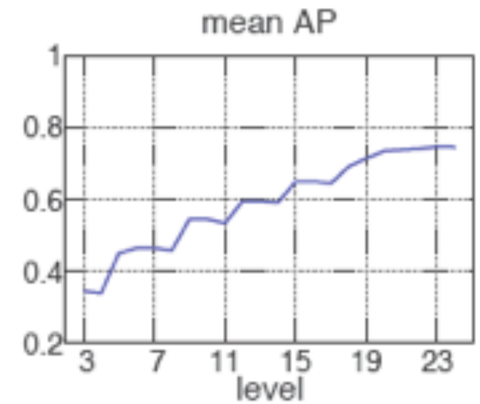
Status Quo – Image Classification



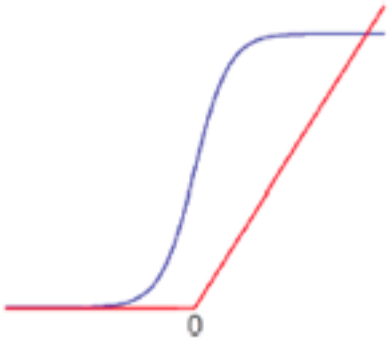
Anh Nguyen et.al., “Deep Neural Networks are Easily Fooled: High Confidence Predictions for Unrecognizable Images”, 2015

Why These Improvements in Performance?

- Features are learned rather than hand-crafted



- More layers capture more invariances [1]
- More data to train deeper networks
- More computing (GPUs)
- Better regularization: Dropout
- New nonlinearities
  - Max pooling, Rectified linear units (ReLU)



- Theoretical understanding of deep networks remains shallow

[1] Razavian, Azizpour, Sullivan, Carlsson, CNN Features off-the-shelf: an Astounding Baseline for Recognition. CVPRW'14.

Theoretical Results on Deep Learning

- **Approximation, depth, width, and invariance theory**
  - Perceptrons and multilayer feedforward networks are universal approximators: Cybenko '89, Hornik '89, Hornik '91, Barron '93
  - Scattering networks are deformation stable for Lipschitz non-linearities: Bruna-Mallat '13, Wiatowski '15, Mallat '16

- **Generalization and regularization theory**
  - # training examples grows exponentially with network size: Barlett '03
  - Distance and margin-preserving embeddings: Giryes '15, Sokolik '16
  - Geometry, generalization bounds and depth efficiency: Montufar '15, Neyshabur '15, Shashua '14 '15 '16

[1] Cybenko. Approximations by superpositions of sigmoidal functions, Mathematics of Control, Signals, and Systems, 2 (4), 303-314, 1989.
[2] Hornik, Stinchcombe and White. Multilayer feedforward networks are universal approximators, Neural Networks, 2(3), 359-366, 1989.
[3] Hornik. Approximation Capabilities of Multilayer Feedforward Networks, Neural Networks, 4(2), 251–257, 1991.
[4] Barron. Universal approximation bounds for superpositions of a sigmoidal function. IEEE Transactions on Information Theory, 39(3):930–945, 1993.
[5] Bruna and Mallat. Invariant scattering convolution networks. Trans. PAMI, 35(8):1872–1886, 2013.
[6] Wiatowski, Bölcskei. A mathematical theory of deep convolutional neural networks for feature extraction. arXiv 2015.
[7] Mallat. Understanding deep convolutional networks. Phil. Trans. R. Soc. A, 374(2065), 2016
[8] Bartlett and Maass. Vapnik-Chervonenkis dimension of neural nets. The handbook of brain theory and neural networks, pages 1188– 1192, 2003.
[9] Giryes, Sapiro, A Bronstein. Deep Neural Networks with Random Gaussian Weights: A Universal Classification Strategy? arXiv:1504.08291.
[10] Sokolic. Margin Preservation of Deep Neural Networks, 2015
[11] Montufar. Geometric and Combinatorial Perspectives on Deep Neural Networks, 2015.
[12] Neyshabur. The Geometry of Optimization and Generalization in Neural Networks: A Path-based Approach, 2015.

Questions which you need to be able to answer...

- How does logistic regression relate to neural networks?
- How do neural networks relate to the brain?
- What kind of functions can single layer neural networks learn?
- Why do two layers help?
- How many layers do you need to represent arbitrary functions?
- Why did they make such splash in the late 1980s?
- Why were Neural Networks abandoned in the 1970s? Why did that somewhat happen again in the mid-1990s?
- Why did they re-awaken in the 2010s?
- What is the biggest problem of neural networks?