Advanced Machine Learning
Lecture 12

Neural Networks

24.11.2016

Bastian Leibe
RWTH Aachen
http://www.vision.rwth-aachen.de/

leibe@vision.rwth-aachen.de

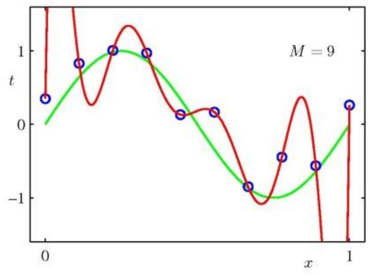
Talk Announcement

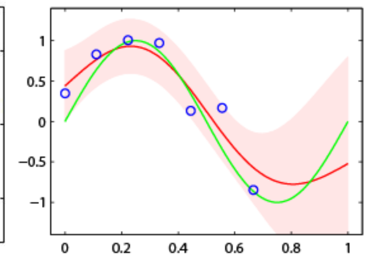
- **Yann LeCun (NYU & FaceBook AI)**
  28.11. 15:00-16:30h, SuperC 6th floor (Ford Saal)
  The rapid progress of AI in the last few years are largely the result of advances in deep learning and neural nets, combined with the availability of large datasets and fast GPUs. We now have systems that can recognize images with an accuracy that rivals that of humans. This will lead to revolutions in several domains such as autonomous transportation and medical image analysis. But all of these systems currently use supervised learning in which the machine is trained with inputs labeled by humans. The challenge of the next several years is to let machines learn from raw, unlabeled data, such as video or text. This is known as predictive (or unsupervised) learning. Intelligent systems today do not possess "common sense", which humans and animals acquire by observing the world, by acting in it, and by understanding the physical constraints of it. I will argue that the ability of machines to learn predictive models of the world is a key component of that will enable significant progress in AI. The main technical difficulty is that the world is only partially predictable. A general formulation of unsupervised learning that deals with partial predictability will be presented. The formulation connects many well-known approaches to unsupervised learning, as well as new and exciting ones such as adversarial training.
- **No lecture next Monday - go see the talk!**

This Lecture: *Advanced Machine Learning*

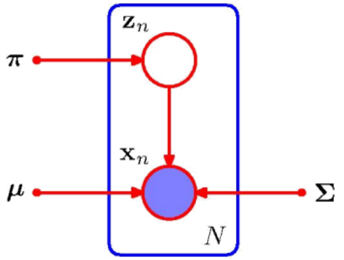
- Regression Approaches
  - Linear Regression
  - Regularization (Ridge, Lasso)
  - Kernels (Kernel Ridge Regression)
  - Gaussian Processes

$$f : \mathcal{X} \rightarrow \mathbb{R}$$

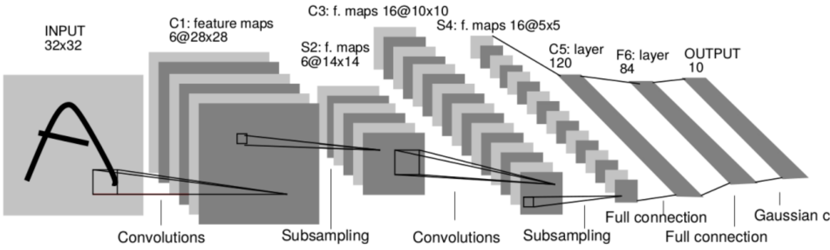




- Approximate Inference
  - Sampling Approaches
  - MCMC



- Deep Learning
  - Linear Discriminants
  - Neural Networks
  - Backpropagation
  - CNNs, RNNs, ResNets, etc.



Recap: Generalized Linear Discriminants

- **Extension with non-linear basis functions**
  - Transform vector $\mathbf{x}$ with M nonlinear basis functions $\phi_j(\mathbf{x})$:

$$y_k(\mathbf{x}) = g\left(\sum_{j=1}^{M} w_{kj}\phi_j(\mathbf{x}) + w_{k0}\right)$$

  - Basis functions $\phi_j(\mathbf{x})$ allow non-linear decision boundaries.
  - Activation function $g(\,\cdot\,)$ bounds the influence of outliers.
  - Disadvantage: minimization no longer in closed form.

- **Notation**

$$y_k(\mathbf{x}) = g\left(\sum_{j=0}^{M} w_{kj}\phi_j(\mathbf{x})\right) \qquad \text{with } \phi_0(\mathbf{x}) = 1$$

Recap: Gradient Descent

- Iterative minimization
  - Start with an initial guess for the parameter values $w_{kj}^{(0)}$.
  - Move towards a (local) minimum by following the gradient.

- Basic strategies
  - "Batch learning"

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

  - "Sequential updating"

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E_n(\mathbf{w})}{\partial w_{kj}} \right|_{\mathbf{w}^{(\tau)}}$$

where

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w})$$

Recap: Gradient Descent

- Example: Quadratic error function

$$E(\mathbf{w}) = \sum_{n=1}^{N} \left(y(\mathbf{x}_n; \mathbf{w}) - \mathbf{t}_n\right)^2$$

- Sequential updating leads to delta rule (=LMS rule)

$$\begin{aligned} w_{kj}^{(\tau+1)} &= w_{kj}^{(\tau)} - \eta \left(y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}\right) \phi_j(\mathbf{x}_n) \\ &= w_{kj}^{(\tau)} - \eta \delta_{kn} \phi_j(\mathbf{x}_n) \end{aligned}$$

  - where

$$\delta_{kn} = y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn}$$

$\Rightarrow$ Simply feed back the input data point, weighted by the classification error.

Recap: Probabilistic Discriminative Models

- Consider models of the form

$$p(\mathcal{C}_1|\phi) = y(\phi) = \sigma(\mathrm{w}^T\phi)$$

with

$$p(\mathcal{C}_2|\phi) = 1 - p(\mathcal{C}_1|\phi)$$

- This model is called logistic regression.

- Properties
  - Probabilistic interpretation
  - But discriminative method: only focus on decision hyperplane
  - Advantageous for high-dimensional spaces, requires less parameters than explicitly modeling $p(\phi|\mathcal{C}_k)$ and $p(\mathcal{C}_k)$.

Recap: Logistic Regression

- Let's consider a data set $\{\phi_n, t_n\}$ with $n = 1,\ldots,N$, where $\phi_n = \phi(\mathbf{x}_n)$ and $t_n \in \{0,1\}$, $\mathbf{t} = (t_1,\ldots,t_N)^T$.

- With $y_n = p(\mathcal{C}_1|\phi_n)$, we can write the likelihood as

$$p(\mathbf{t}|\mathbf{w}) = \prod_{n=1}^{N} y_n^{t_n} \{1 - y_n\}^{1-t_n}$$

- Define the error function as the negative log-likelihood

$$\begin{aligned} E(\mathbf{w}) &= -\ln p(\mathbf{t}|\mathbf{w}) \\ &= -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\} \end{aligned}$$

  - This is the so-called cross-entropy error function.

Softmax Regression

- **Multi-class generalization of logistic regression**
  - **In logistic regression, we assumed binary labels $t_n \in \{0, 1\}$**
  - **Softmax generalizes this to K values in 1-of-K notation.**

$$\mathbf{y}(\mathbf{x};\mathbf{w}) = \begin{bmatrix} P(y=1|\mathbf{x};\mathbf{w}) \\ P(y=2|\mathbf{x};\mathbf{w}) \\ \vdots \\ P(y=K|\mathbf{x};\mathbf{w}) \end{bmatrix} = \frac{1}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x})} \begin{bmatrix} \exp(\mathbf{w}_1^\top \mathbf{x}) \\ \exp(\mathbf{w}_2^\top \mathbf{x}) \\ \vdots \\ \exp(\mathbf{w}_K^\top \mathbf{x}) \end{bmatrix}$$

  - **This uses the softmax function**

$$\frac{\exp(a_k)}{\sum_j \exp(a_j)}$$

  - **Note: the resulting distribution is normalized.**

Softmax Regression Cost Function

- **Logistic regression**
  - Alternative way of writing the cost function

$$
\begin{aligned}
E(\mathbf{w}) &= -\sum_{n=1}^{N} \{t_n \ln y_n + (1-t_n)\ln(1-y_n)\} \\
&= -\sum_{n=1}^{N} \sum_{k=0}^{1} \{\mathbb{I}\,(t_n = k) \ln P\,(y_n = k | \mathbf{x}_n; \mathbf{w})\}
\end{aligned}
$$

- **Softmax regression**
  - Generalization to K classes using indicator functions.

$$
E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} \left\{ \mathbb{I}\,(t_n = k) \ln \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{\sum_{j=1}^{K} \exp(\mathbf{w}_j^\top \mathbf{x})} \right\}
$$

$$
\nabla_{\mathbf{w}_k} E(\mathbf{w}) = -\sum_{n=1}^{N} \left[\mathbb{I}\,(t_n = k) \ln P\,(y_n = k | \mathbf{x}_n; \mathbf{w})\right]
$$

Optimization

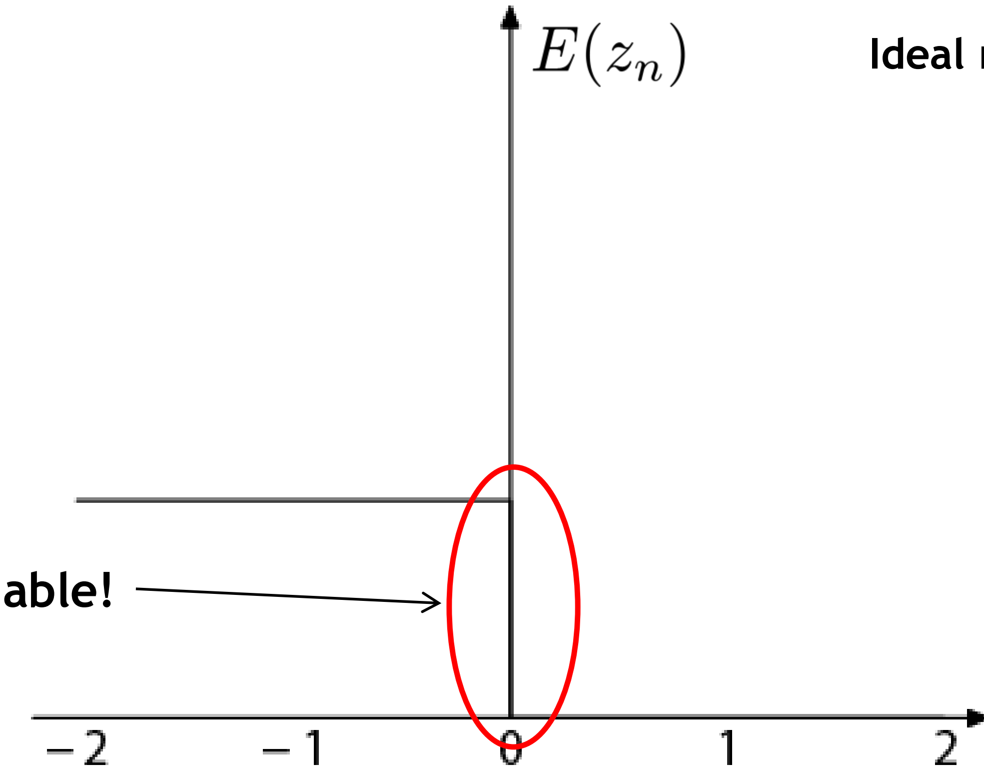
- **Again, no closed-form solution is available**
  - Resort again to Gradient Descent
  - Gradient

$$\nabla_{\mathbf{w}_k} E(\mathbf{w}) = -\sum_{n=1}^{N} \left[\mathbb{I}\left(t_n = k\right) \ln P\left(y_n = k | \mathbf{x}_n; \mathbf{w}\right)\right]$$

- **Note**
  - $\nabla_{\mathbf{w}k} E(\mathbf{w})$ is itself a vector of partial derivatives for the different components of $\mathbf{w}_k$.
  - We can now plug this into a standard optimization package.

A Note on Error Functions

$t_n \in \{-1, 1\}$



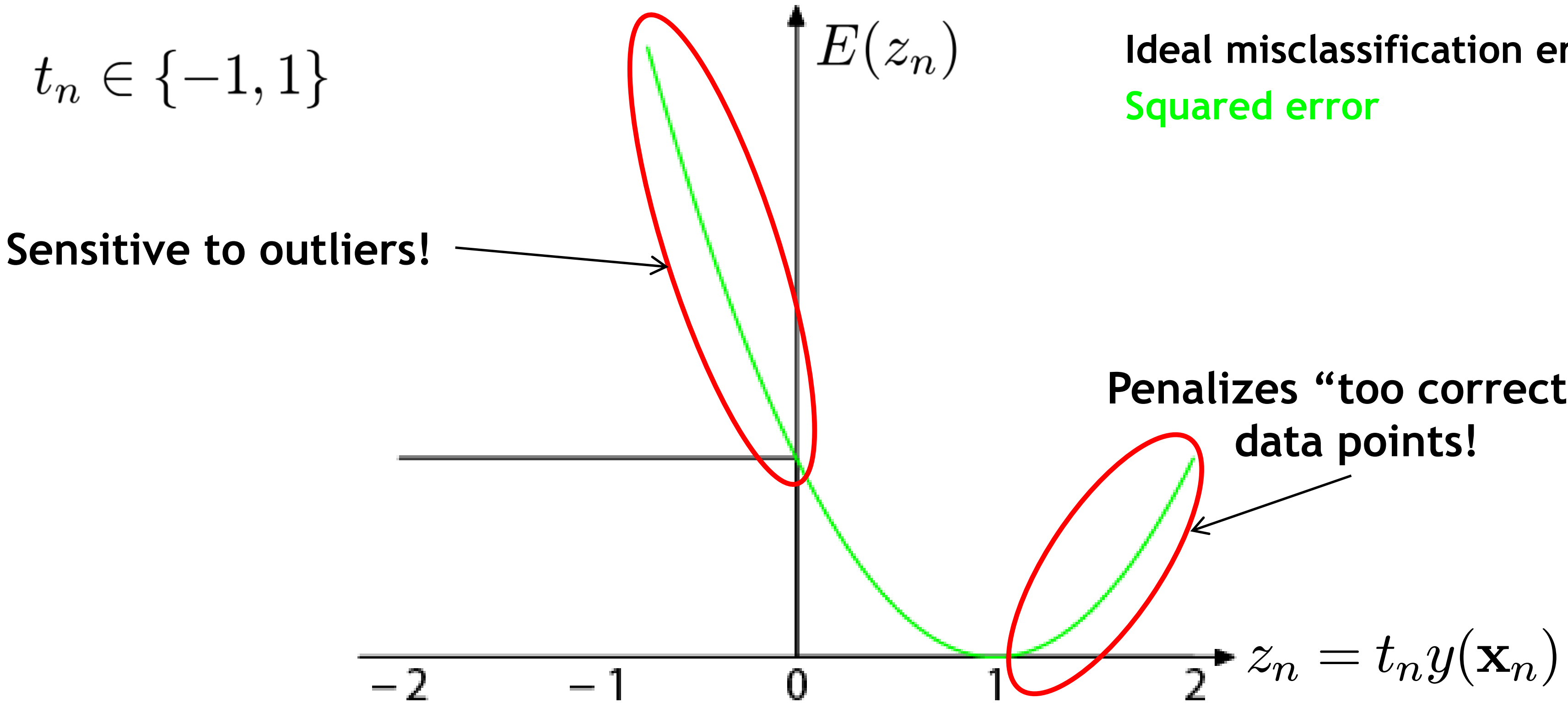
- **Ideal misclassification error function (black)**
  - This is what we want to approximate,
  - Unfortunately, it is not differentiable.
  - The gradient is zero for misclassified points.
  - $\Rightarrow$ We cannot minimize it by gradient descent.

Image source: Bishop, 2006

RWTH AACHEN UNIVERSITY

A Note on Error Functions

$$t_n \in \{-1, 1\}$$

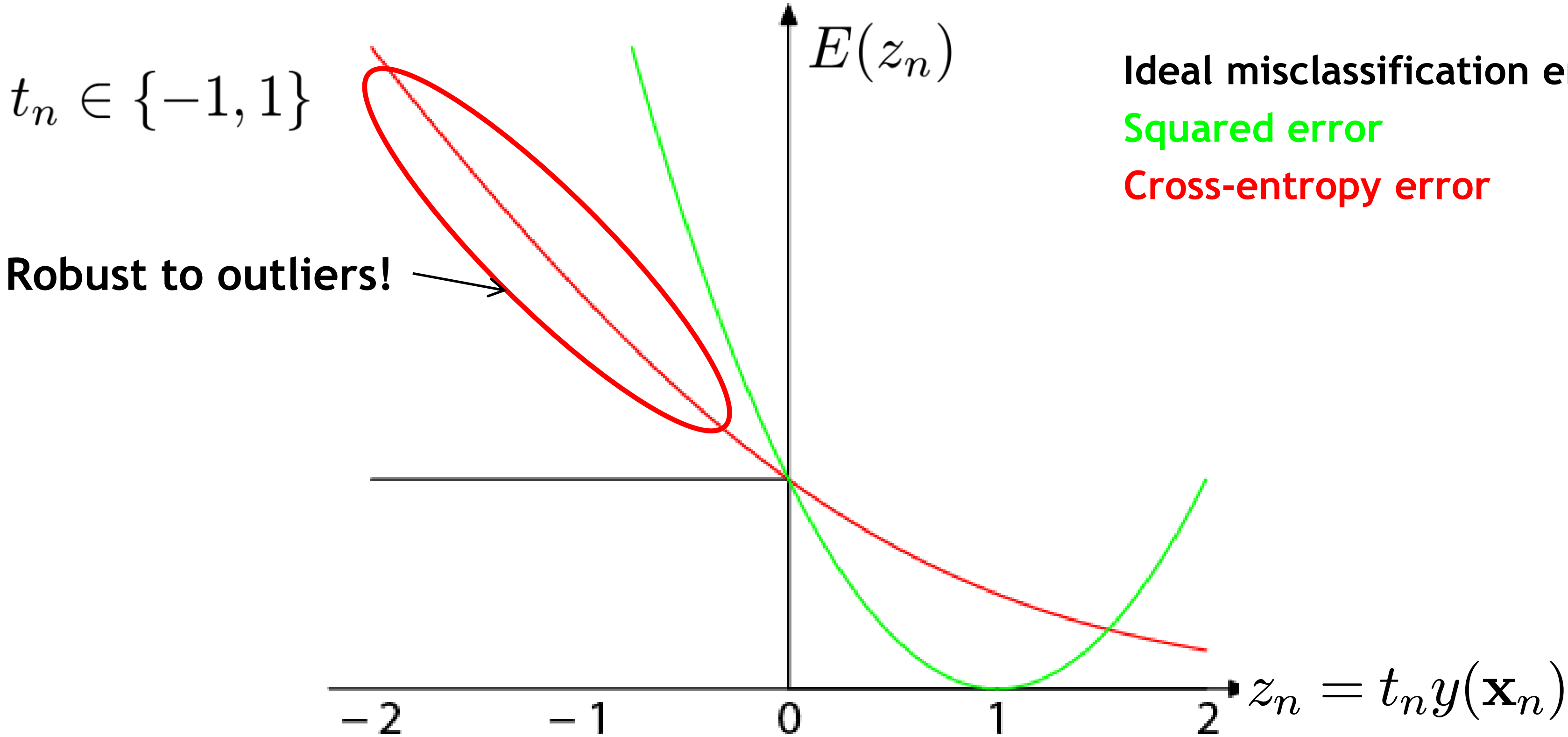


- Squared error used in Least-Squares Classification
  - Very popular, leads to closed-form solutions.
  - However, sensitive to outliers due to squared penalty.
  - Penalizes "too correct" data points

$\Rightarrow$ Generally does not lead to good classifiers.

Image source: Bishop, 2006

A Note on Error Functions



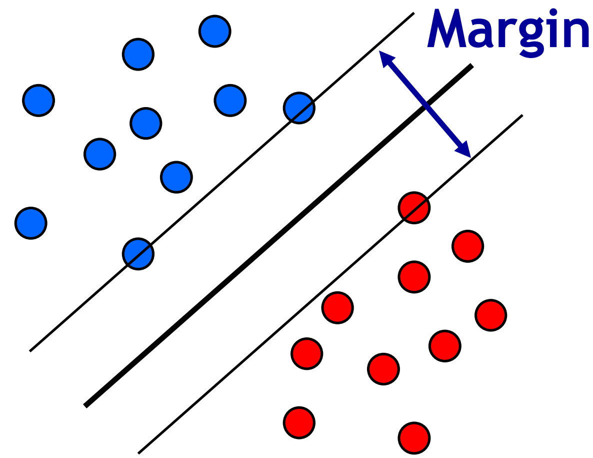
- **Cross-Entropy Error**
  - Minimizer of this error is given by posterior class probabilities.
  - Concave error function, unique minimum exists.
  - Robust to outliers, error increases only roughly linearly
  - But no closed-form solution, requires iterative estimation.

Image source: Bishop, 2006

Side Note: Support Vector Machine (SVM)

- **Basic idea**
  - The SVM tries to find a classifier which maximizes the margin between pos. and neg. data points.
  - Up to now: consider linear classifiers

$$\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0$$



- **Formulation as a convex optimization problem**
  - Find the hyperplane satisfying

$$\underset{\mathbf{w},b}{\arg\min}\ \frac{1}{2}\|\mathbf{w}\|^2$$

under the constraints

$$t_n(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n + b) \geq 1 \quad \forall n$$

based on training data points $\mathbf{x}_n$ and target values $t_n \in \{-1, 1\}$.

SVM – Analysis

- **Traditional soft-margin formulation**

$$\min_{\mathbf{w}\in\mathbb{R}^D,\, \xi_n\in\mathbb{R}^+} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n$$

"Maximize the margin"

subject to the constraints

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n$$

"Most points should be on the correct side of the margin"

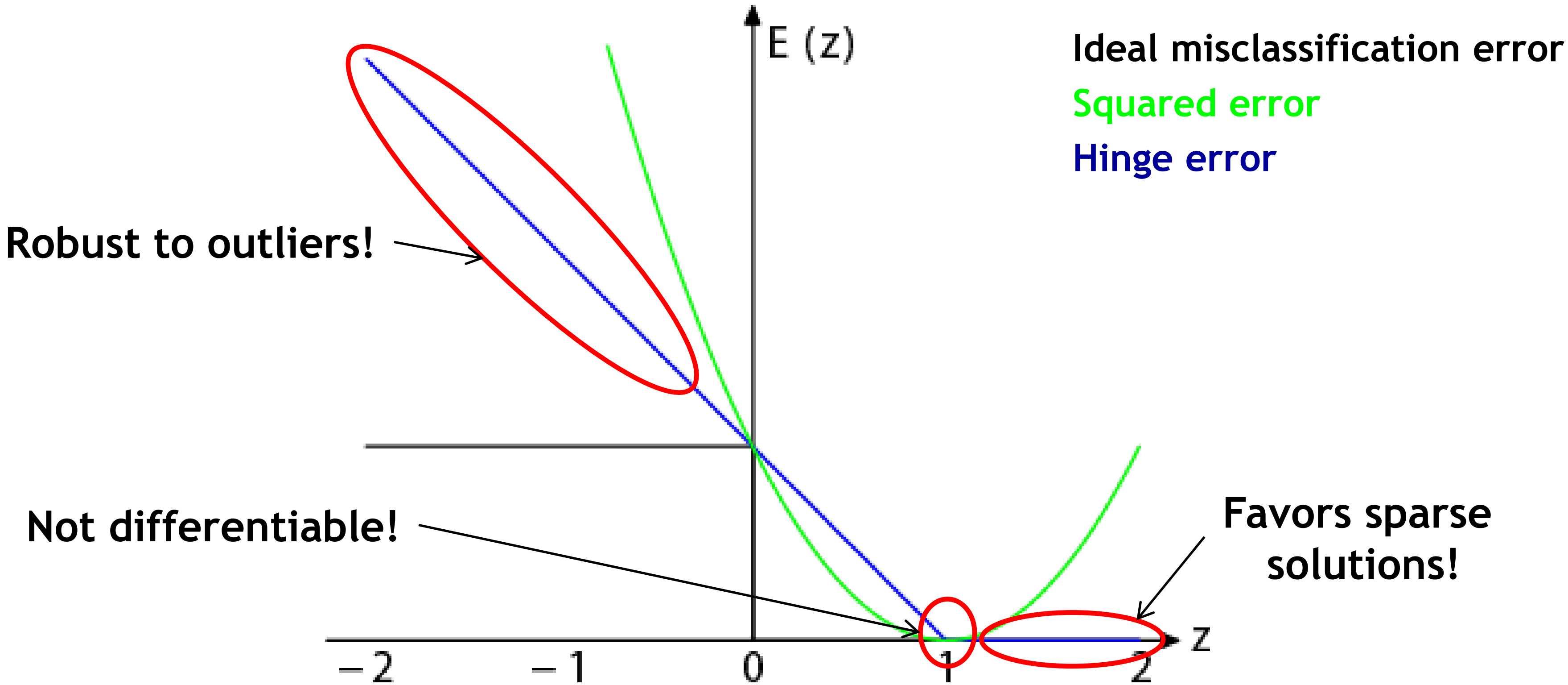
- **Different way of looking at it**
  - We can reformulate the constraints into the objective function.

$$\min_{\mathbf{w}\in\mathbb{R}^D} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{\text{L}_2\text{ regularizer}} + \underbrace{C\sum_{n=1}^{N}[1 - t_n y(\mathbf{x}_n)]_+}_{\text{"Hinge loss"}}$$

L_2 regularizer — "Hinge loss"

where $[x]_+ := \max\{0, x\}$.

SVM Error Function (Loss Function)



- “Hinge error” used in SVMs
  - Zero error for points outside the margin ($z_n > 1$).
  - Linearly increasing error for misclassified points ($z_n < 1$).

  $\Rightarrow$ Leads to sparse solutions, not sensitive to outliers.
  - Not differentiable around $z_n = 1$ $\Rightarrow$ Cannot be optimized directly.

Image source: Bishop, 2006

SVM – Discussion

- SVM optimization function

$$\min_{\mathbf{w} \in \mathbb{R}^D} \underbrace{\frac{1}{2}\|\mathbf{w}\|^2}_{\text{L}_2 \text{ regularizer}} + \underbrace{C \sum_{n=1}^{N} [1 - t_n y(\mathbf{x}_n)]_+}_{\text{Hinge loss}}$$

- Hinge loss enforces sparsity
  - Only a subset of training data points actually influences the decision boundary.
  - This is different from sparsity obtained through the regularizer! There, only a subset of input dimensions are used.
  - Unconstrained optimization, but non-differentiable function.
  - Solve, e.g. by *subgradient descent*
  - Currently most efficient: *stochastic gradient descent*

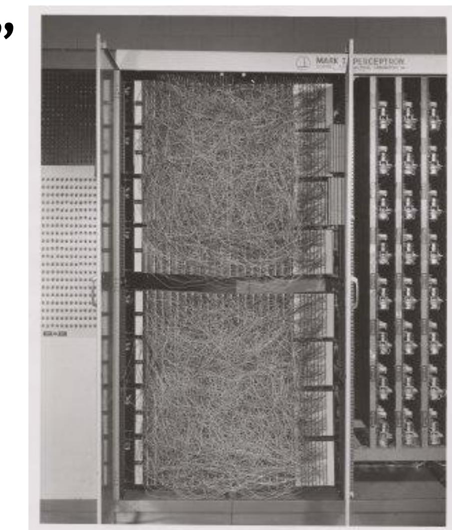
Topics of This Lecture

- A Brief History of Neural Networks
- Perceptrons
  - Definition
  - Loss functions
  - Regularization
  - Limits
- Multi-Layer Perceptrons
  - Definition
  - Learning

A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron

- And a cool learning algorithm: "Perceptron Learning"
- Hardware implementation "Mark I Perceptron" for 20×20 pixel image analysis



The New York Times

"The embryo of an electronic computer that [...] will be able to walk, talk, see, write, reproduce itself and be conscious of its existence."

Image source: Wikipedia, clipartpanda.com

A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron

1969 Minsky & Papert

- They showed that (single-layer) Perceptrons cannot solve all problems.
- This was misunderstood by many that they were worthless.

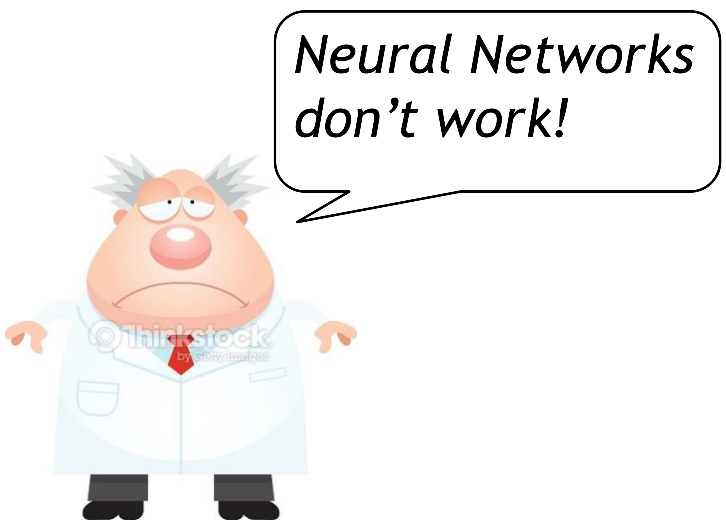


Image source: colourbox.de, thinkstock

A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron

1969 Minsky & Papert

1980s Resurgence of Neural Networks

- Some notable successes with multi-layer perceptrons.
- Backpropagation learning algorithm

OMG! They work like the human brain!

Oh no! Killer robots will achieve world domination!

Image sources: clipartpanda.com, cliparts.co

A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron

1969 Minsky & Papert

1980s Resurgence of Neural Networks

- Some notable successes with multi-layer perceptrons.
- Backpropagation learning algorithm
- But they are hard to train, tend to overfit, and have unintuitive parameters.
- So, the excitement fades again.

Image source: clipartof.com, colourbox.de

A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron

1969 Minsky & Papert

1980s Resurgence of Neural Networks

1995+ Interest shifts to other learning methods

- Notably Support Vector Machines
- Machine Learning becomes a discipline of its own.

Image source: clipartof.com

A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron

1969 Minsky & Papert

1980s Resurgence of Neural Networks

1995+ Interest shifts to other learning methods

- Notably Support Vector Machines
- Machine Learning becomes a discipline of its own.
- The general public and the press still love Neural Networks.

I'm doing Machine Learning.

So, you're using Neural Networks?

Actually...

A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron

1969 Minsky & Papert

1980s Resurgence of Neural Networks

1995+ Interest shifts to other learning methods

2005+ Gradual progress

- Better understanding how to successfully train deep networks
- Availability of large datasets and powerful GPUs
- Still largely under the radar for many disciplines applying ML

Are you using Neural Networks?

Come on. Get real!

A Brief History of Neural Networks

1957 Rosenblatt invents the Perceptron

1969 Minsky & Papert

1980s Resurgence of Neural Networks

1995+ Interest shifts to other learning methods

2005+ Gradual progress

2012 Breakthrough results

- ImageNet Large Scale Visual Recognition Challenge
- A ConvNet halves the error rate of dedicated vision approaches.
- Deep Learning is widely adopted.

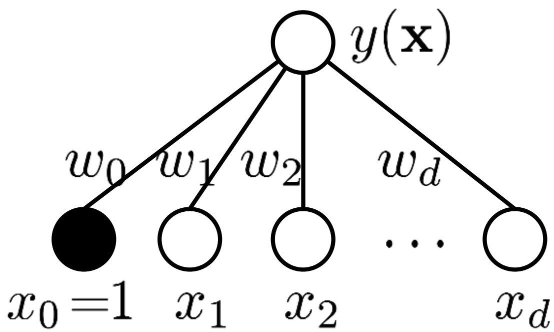
Image source: clipartpanda.com, clipartof.com

Topics of This Lecture

- A Short History of Neural Networks
- Perceptrons
  - Definition
  - Loss functions
  - Regularization
  - Limits
- Multi-Layer Perceptrons
  - Definition
  - Learning

Perceptrons (Rosenblatt 1957)

- **Standard Perceptron**



Output layer

Weights

Input layer

- **Input Layer**
  - Hand-designed features based on common sense

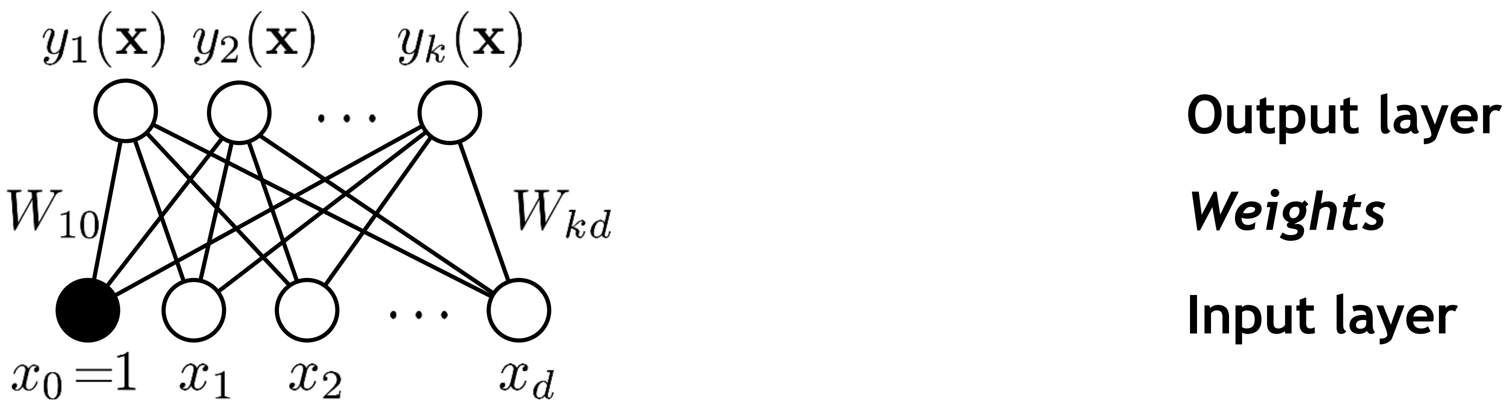
- **Outputs**
  - Linear outputs
    $$y(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + w_0$$
  - Logistic outputs
    $$y(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x} + w_0)$$

- **Learning = Determining the weights $\mathbf{w}$**

RWTH AACHEN UNIVERSITY

Extension: Multi-Class Networks

- One output node per class



- Outputs
  - Linear outputs

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

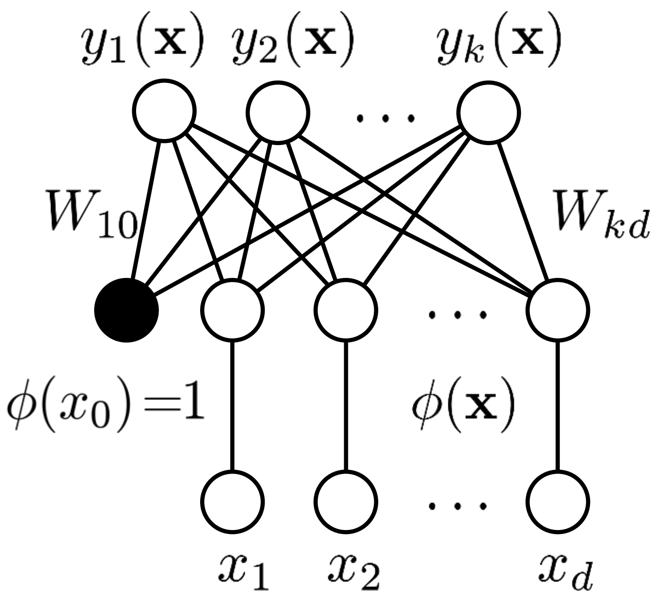
Logistic outputs

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

$\Rightarrow$ Can be used to do multidimensional linear regression or multiclass classification.

Extension: Non-Linear Basis Functions

- **Straightforward generalization**



Output layer

Weights

Feature layer

Mapping (fixed)

Input layer

- **Outputs**
  - Linear outputs

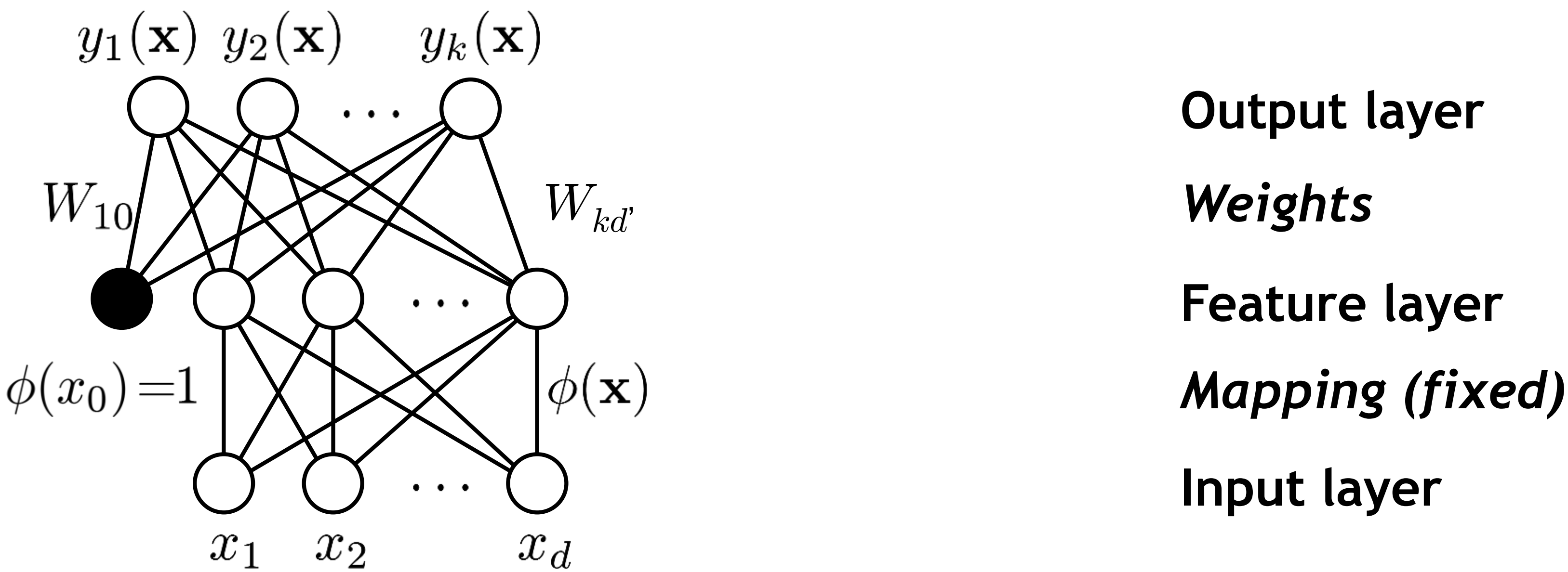
$$y_k(\mathbf{x}) = \sum_{i=0}^{d} W_{ki}\phi(x_i)$$

Logistic outputs

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

Extension: Non-Linear Basis Functions

- Straightforward generalization



- Remarks
  - Perceptrons are generalized linear discriminants!
  - Everything we know about the latter can also be applied here.
  - Note: feature functions $\phi(\mathbf{x})$ are kept fixed, not learned!

Perceptron Learning

- Very simple algorithm
- Process the training cases in some permutation
  - If the output unit is correct, leave the weights alone.
  - If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
  - If the output unit incorrectly outputs a one, subtract the input vector from the weight vector.
- This is guaranteed to converge to a correct solution if such a solution exists.

Perceptron Learning

- Let's analyze this algorithm...
- Process the training cases in some permutation
  - If the output unit is correct, leave the weights alone.
  - If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
  - If the output unit incorrectly outputs a one, subtract the input vector from the weight vector.
- Translation

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)}$$

Perceptron Learning

- Let's analyze this algorithm...
- Process the training cases in some permutation
  - If the output unit is correct, leave the weights alone.
  - If the output unit incorrectly outputs a zero, add the input vector to the weight vector.
  - If the output unit incorrectly outputs a one, subtract the input vector from the weight vector.
- Translation

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left(y_k(\mathbf{x}_n; \mathbf{w}) - t_{kn} \right) \phi_j(\mathbf{x}_n)$$

  - This is the Delta rule a.k.a. LMS rule!

⇒ Perceptron Learning corresponds to 1st-order (stochastic) Gradient Descent of a quadratic error function!

Loss Functions

- We can now also apply other loss functions
  - L2 loss $\Rightarrow$ **Least-squares regression**

    $$L(t, y(\mathbf{x})) = \sum_n (y(\mathbf{x}_n) - t_n)^2$$

  - L1 loss: $\Rightarrow$ **Median regression**

    $$L(t, y(\mathbf{x})) = \sum_n |y(\mathbf{x}_n) - t_n|$$

  - Cross-entropy loss $\Rightarrow$ **Logistic regression**

    $$L(t, y(\mathbf{x})) = -\sum_n \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\}$$

  - Hinge loss $\Rightarrow$ **SVM classification**

    $$L(t, y(\mathbf{x})) = \sum_n [1 - t_n y(\mathbf{x}_n)]_+$$

  - Softmax loss $\Rightarrow$ **Multi-class probabilistic classification**

    $$L(t, y(\mathbf{x})) = -\sum_n \sum_k \left\{ \mathbb{I}(t_n = k) \ln \frac{\exp(y_k(\mathbf{x}))}{\sum_j \exp(y_j(\mathbf{x}))} \right\}$$

Regularization

- In addition, we can apply regularizers
  - E.g., an L2 regularizer

$$E(\mathbf{w}) = \sum_n L(t_n, y(\mathbf{x}_n; \mathbf{w})) + \lambda ||\mathbf{w}||^2$$

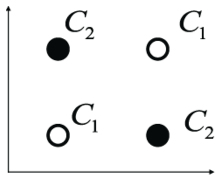
  - This is known as *weight decay* in Neural Networks.
  - We can also apply other regularizers, e.g. L1 ⇒ sparsity
  - Since Neural Networks often have many parameters, regularization becomes very important in practice.
  - We will see more complex regularization techniques later on...

Limitations of Perceptrons

- What makes the task difficult?
  - Perceptrons with fixed, hand-coded input features can model any separable function perfectly...
  - ...given the right input features.
  - For some tasks this requires an exponential number of input features.
    - E.g., by enumerating all possible binary input vectors as separate feature units (similar to a look-up table).
    - But this approach won't generalize to unseen test cases!

  ⇒ It is the feature design that solves the task!

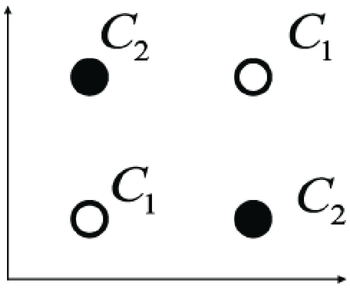
  - Once the hand-coded features have been determined, there are very strong limitations on what a perceptron can learn.
    - Classic example: XOR function.



Wait...

- Didn't we just say that...
  - Perceptrons correspond to generalized linear discriminants
  - And Perceptrons are very limited...
  - *Doesn't this mean that what we have been doing so far in this lecture has the same problems???*

- Yes, this is the case.
  - A linear classifier cannot solve certain problems (e.g., XOR).
  - However, with a non-linear classifier based on the right kind of features, the problem becomes solvable.



$\Rightarrow$ So far, we have solved such problems by hand-designing good features ϕ and kernels $\phi^\top\phi$.

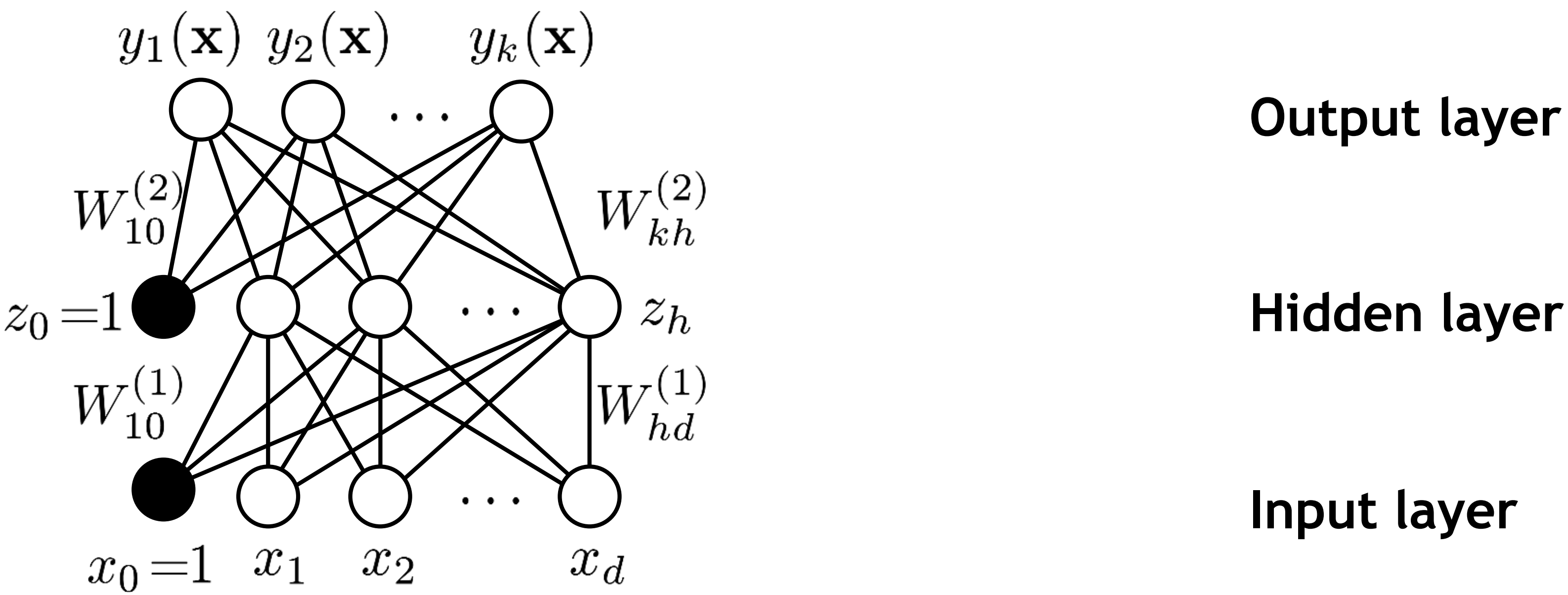
$\Rightarrow$ *Can we also learn such feature representations?*

Topics of This Lecture

- A Short History of Neural Networks
- Perceptrons
  - Definition
  - Loss functions
  - Regularization
  - Limits
- **Multi-Layer Perceptrons**
  - Definition
  - Learning

Multi-Layer Perceptrons

- **Adding more layers**



- **Output**

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

Multi-Layer Perceptrons

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

- **Activation functions $g^{(k)}$:**
  - For example: $g^{(2)}(a) = \sigma(a)$, $g^{(1)}(a) = a$
- **The hidden layer can have an arbitrary number of nodes**
  - 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)

Learning with Hidden Units

- Networks without hidden units are very limited in what they can learn
  - More layers of linear units do not help ⇒ still linear
  - Fixed output non-linearities are not enough.

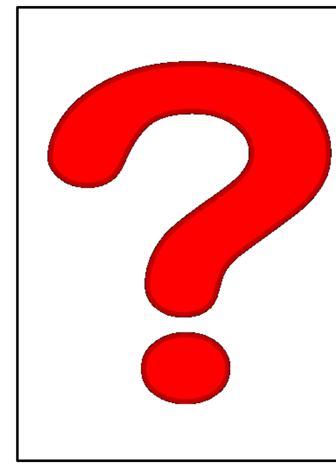
- We need multiple layers of adaptive non-linear hidden units. But how can we train such nets?
  - Need an efficient way of adapting all weights, not just the last layer.
  - Learning the weights to the hidden units = learning features
  - This is difficult, because nobody tells us what the hidden units should do.

  ⇒ Next lecture

References and Further Reading

- More information on Neural Networks can be found in Chapters 6 and 7 of the Goodfellow & Bengio book

Ian Goodfellow, Aaron Courville, Yoshua Bengio
Deep Learning
MIT Press, in preparation



https://goodfeli.github.io/dlbook/