Begriff	**Erklärung**	**Beispiel**
X	Menge der Feature Vektoren	Vektoren mit den Eigenschaften der Pilze
$\mathbf{X}$	Feature Vektor; Feature Space	
x	Dimension des korrelierenden Vektors	
$\mathbf{w}$	Weight-Vektor	
$\mathbf{w}^T\mathbf{x}$		
$c(x_n)$	Klasse von x_n	$c(x_1) =$ "$toxic$" $c(x_2) =$ "$edible$"
$\mathbf{R}$		

1 Begriffe

2 Definition Formel & Algorithmen

2.1 Specification of learning tasks

Realworld → Modelworld

1. **Reale Welt:**

- O - Menge an Objekten
- C - Menge an Klassen
- $\gamma : O \to C$ - idealer Classifier für O

Aufgabe-Klassifizierung:

- bestimme von $o \in O$ die Klasse $\gamma(o) \in C$

2. **Model Welt**

- X - Menge von feature Vektoren
- C - Menge von Klassen
- $c : X \to C$ - idealer Classifier für X *(c ist unbekannt)*
- $D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ \ldots \ , (\mathbf{x}_n, c(\mathbf{x}_n))\}$ - Menge von Beispielen *(bereits klassifiziert)*

Todo: Schätze $c(\mathbf{x})$, welche implizit durch D gegeben sind, durch Model-Funktion $y(\mathbf{x})$

Machine Learning:

1. Formuliere Model-Function: $y : X \to C, \mathbf{x} \mapsto y(\mathbf{x})$
2. Nutze Statistik, Theorie und Algorithmen aus ML um den fit zwischen $c(\mathbf{x})$ und $y(\mathbf{x})$ zu Maximieren

2.2 LMS: Least Mean Squared

Ziel: Fitting $y(x)$; Anpassung der weights, sodass Klassifizierungsfehler möglichst gering sind.

Input: D - Trainingsdaten $(\mathbf{x}, c(\mathbf{x}))$ mit $\mathbf{x} \in \mathbf{R}^p$ und Zielklasse c($\mathbf{x}$) η Learning rate, kleine positive Konstante

INKLUDIERE LMS TEIL HIER vermutlich aber irrelevant

2.3 Lineare Regression

Grundformel für lineare Gerade, mit:
$y(x)$ - abhängige Variabel x - unabhängige Variable w_1 - Anstieg der Geraden w_0 - Schnittpunkt der y-Achse

$$y(x) = w_o + w_1 \cdot x \tag{1}$$

Wobei das minimale w_0 und w_1 sich ergeben aus:

$$w_1 = \frac{\sum_{i=1}^{n} (x_i - \overline{x} \cdot (y_i - \overline{y})}{\sum_{i=1}^{n} (x_i - \overline{x}^2} \tag{2}$$

$$w_0 = \overline{y} - w_1 \cdot \overline{x} \tag{3}$$

Goodness of Modelfit, Regressionserror Residual sum of Squares(RSS).

Der Residue wird aus der Differenz zwischen (beobachteten) Realwelt-Wert y_i und geschätzten/modelierten Wert $y(\mathbf{x}_i)$

$$RSS(\mathbf{w}) = \sum_{i=1}^{n} (y_i - y(\mathbf{x}_i))^2 \tag{4}$$

Higher-Dimensional Feature Space ML:ll-16

2.4 Concept Learning

Setting:
X - Menge an Feature Vektoren
C - Ist eine Menge mit zwei Klassen *Beipiel:* $\{0,1\}$,$\{ja,\ nein\}$
$c : X \to C$ -idealer Klssifizierer für X
$D = \{(\mathbf{x}_1, c(\mathbf{x}_1)), \ldots, (\mathbf{x}_n, c(\mathbf{x}_n))\} \subseteq X \times C$ - Menge mit Beispielen
Todo:
Schätze $c(x)$, was implizit durch D mit feature-Value-Muster

3 Definitionen Text

3.1 Supervised learning

Eine Funktion mithilfe von input-output-Daten lernen;
automatisierte Klassifikation mit von Menschen bereits klassifizierten Daten als Grundlage
Beispiel: optical character recognizition

3.2 Unsupervised learning

identifiziert/findet selbstständig Muster und Strukturen in Daten;

- automatisierte Kategorisierung durch Cluster Analysis
- Parameter Optimierung durch Expectation Maximation
- Feature Extrahierung durch Factor Analysis

Beispiel: intrusion detection in a network data stream

3.3 Reinforcement learning

"Learn, adapt, or optimize a behavior strategy in order to maximize the ownbenefit by interpreting feedback that is provided by the environment."
Beispiel: program to play tetris

3.4 Feature Vektor

Ein Feature Vektor ist ein Vektor in dem jede Dimension eine Eigenschaft(Feature) des beschriebenen Objektes enthält.

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \quad \text{Besipiel: } \mathbf{Gitarre} = \begin{bmatrix} Farbe : blau \\ Baujahr : 1997 \\ Material : Holz \\ Elektrisch : ja \end{bmatrix}$$

3.5 Ground Truth

Überprüfung der Klassifizierung eines Lernprozesses auf Richtigkeit für gewolltes Model.
Beispiel: Überprüfung eines Spamfilter nach falsch kategorisierten Mails

4 Linear Models

4.1 Overfitting

- fitting: der Prozess die Parameter einer Modelfunktion y so anzupassen das sie der der Beispieldaten D am besten passen
- overfiiting: "Fitting the data more than is warranted"
- alias besser passen als berechtigt?
- Gründe:
  - zu komplizierte Modefunktion (zu viele Features)
  - zu wenig Daten in D
  - zu viel Datenrauschen
  - D ist zu biased alias nicht repräsentativ
- Folgen:
  - kleiner Error auf D_{tr} anber großer Error auf D_{test} und IRL
  - loss of inductive Bias
  - increase of variance as a result of sensitivity to noise
- Overfitting finden:
  - Visuell untersuchen für Fälle mit Dimensionen < 3 sonst embedding oder projizieren in kleinere Dimensionen
  - Validieren: wenn $Err_{fit} = Err_{val}(y) - Err_{tr}(y)$ zu groß ist
- Overfitting vermeiden:
  - Early stopping through model selection: nach m schritten überprüfen ob sich Err_{fit} noch verkleinert und stoppen wenn er sich vergrößert
  - Qualität (schlechte Beispiele raus) und / oder Quantität (mehr Daten gleichen Rauschen aus) von D verbessern
  - Manually enforcing a higher bias by using a less complex hypothesis space alias Removing Features: In this approach, irrelevant features are removed from the dataset. This enhances the algorithm's ability to generalize
  - Regularization (WUHU!)

4.1.1 Well- and Ill-posed problems

A mathematical problem is called well-posed if

1. a solution exists,
2. the solution is unique,
3. the solution's behavior changes continuously with the initial conditions.

Otherwise, the problem is called ill-posed.

4.2 Regularization

Automatic adjustment of the loss function to penalize model complexity. Let $L(\mathbf{w}$ denote a loss function used to optimize the parameters $\mathbf{w}$ of a model function $y(\mathbf{x})$. Regularization introduces a trade-off between model complexity and inductive bias:

$$\mathcal{L}(\mathbf{w}) = L(\mathbf{w}) + \lambda * R(\mathbf{w})$$

where $\lambda \geq 0$ controls the impact of the regularization term $R(\mathbf{w}) \geq 0$. $\mathcal{L}$ is called "objective function".

4.2.1 Regularized Linear Regression

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^{n} (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \lambda \cdot \overrightarrow{w}^T \overrightarrow{w}$$

Estimate $\mathbf{w}$ by minimizing the residual sum of squares:

$$\hat{w} = \underset{\mathbf{w} \in \mathbf{R^{P+1}}}{\operatorname{argmin}} \mathcal{L}(\mathbf{w})$$

$$\rightsquigarrow RSS(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w}^T \mathbf{w}$$

Ableitung bilden um RSS($\mathbf{w}$) zu minimieren und man kommt auf:

$$\mathbf{w} = (X^T X + \operatorname{diag}(0, \lambda, ..., \lambda))^{-1} X^T \mathbf{y}$$

$$\operatorname{diag}(0, \lambda, ..., \lambda) = \begin{bmatrix} 0 & 0 & 0 & ... & 0 \\ 0 & \lambda & 0 & ... & 0 \\ 0 & 0 & \lambda & ... & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & ... & \lambda \end{bmatrix}$$

$$\hat{y}(\mathbf{x}_i) = \hat{\mathbf{w}}^T \mathbf{x}_i$$

Um so höher λ um so einfacher ist die Funktion - Regularization archived!

5 Neural Networks

5.1 Perception Learning

Idee: Lass mal ein Gehirn programmieren!
Typisches Beispiel: Schrifterkennung

$$y(\mathbf{x}) = 1 \Leftrightarrow (\sum_{j=0}^{p} w_j x_j) \geq 0$$

sonst ist $\mathrm{y}(\mathbf{x}) = 0$

- wenn $w_0 = -\theta$ und $x_0 = 1$(canonical form)
- sonst $y(\mathbf{x}) = 1 \Leftrightarrow (\sum_{j=1}^{p} w_j x_j - \theta) \geq 0$
- $0 = w_0 + w_1 * x_1 + w_2 * x_2$ für 2D Fälle

5.1.1 PT Algorithm

PT(D, η)

1. *initialize_random_weights*$(\mathbf{w})$, $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. $(\mathbf{x}, c(\mathbf{x})) =$ *random_select*(D)
5. $\delta = c(\mathbf{x}) - y(\mathbf{x})$ // $y(\mathbf{x}) \overset{(*)}{=}$ *heaviside*$(\mathbf{w}^T\mathbf{x}) \in \{0,1\}$, $c(\mathbf{x}) \in \{0,1\} \rightsquigarrow \delta \in \{0,1,-1\}$
6. $\Delta\mathbf{w} \overset{(*)}{=} \eta \cdot \delta \cdot \mathbf{x}$
7. $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$
8. **UNTIL**(*convergence*$(c(D), y(D))$ **OR** $t > t_{\max}$)
9. *return*$(\mathbf{w})$

- If a separating hyperplane between X_0 and X_1 exists, the PT algorithm will converge. If no such hyperplane exists, convergence cannot be guaranteed.
- A separating hyperplane can be found in polynomial time with linear programming. The PT Algorithm, however, may require an exponential number of iterations.
- Classification problems with noise are problematic

5.2 Gradient Descent

- Finde den kürzesten Weg in ein Min/Max über partielle Ableitungen
- The gradient of a function is the direction of steepest ascent or descent.
- in der VL ist ein Beweis den ich nicht abtippe weil irrelevant

5.2.1 Linear Regression + Squared Loss

$$L_2(\mathbf{w}) = \frac{1}{2} \cdot \sum_{(\mathbf{x},c(\mathbf{x}))\in D} (c(\mathbf{x}) - y(\mathbf{x}))^2$$

Jetzt müssen wir für jedes w_i aus $\mathbf{w}$ eine partielle Ableitung machen um den weight vector zu updaten ($\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$)

$$\Delta\mathbf{w} = \frac{\delta}{\delta w_i} L_2(\mathbf{w}) = \eta \cdot \sum_D (c(\mathbf{x}) - \mathbf{w}^T\mathbf{x}) \cdot \mathbf{x}$$

η = learning rate - a small positiv constant - legen wir selbst fest

5.2.2 The Batch Gradient Descent (BGD) Algorithm

BGD(D, η)

1. *initialize_random_weights*$(\mathbf{w})$, $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. $\Delta\mathbf{w} = 0$
5. FOREACH $(\mathbf{x}, c(\mathbf{x})) \in D$ DO
6. $\delta = c(\mathbf{x}) - y(\mathbf{x})$ // $y(\mathbf{x}) \overset{(*)}{=} \mathbf{w}^T\mathbf{x}$, $\delta \in \mathbb{R}$.
7. $\Delta\mathbf{w} \overset{(*)}{=} \Delta\mathbf{w} + \eta \cdot \delta \cdot \mathbf{x}$ // $\delta \cdot \mathbf{x}$ is the derivative of $\ell_2(c(\mathbf{x}), y(\mathbf{x}))$.
8. ENDDO
9. $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$ // $\Delta\mathbf{w}$ is $-\eta \cdot \nabla L_2(\mathbf{w})$ here.
10. **UNTIL**(*convergence*$(c(D), y(D))$ **OR** $t > t_{\text{max}}$)
11. *return*$(\mathbf{w})$

- wichtig: immer wenn irgendwo $\mathbf{w}^T\mathbf{x}$ steht haben wir $x_0 = 1$ zu $\mathbf{x}$ hinzugefügt
- funktionsweise BGD. wir berechnen über die Ableitung in welche Richtung wir müssen und gehen dann einen Schritt der große η
- die "convergence" schaut ob der Squared Loss noch größer als ein ε ist (das wir auch festlegen)
- BGD ist nicht der schnellste (bestenfalls linear) aber sehr einfach (Newton-Raphson algorithm, BFGS algorithm sind z.B. schneller)
- BGD nimmt den global loss: loss of all examples in D ("batch gradient descent")(Schritt in Richtung die für alle Punkte am besten ist)
- man kann auch den (squared) loss in Bezug auf einzelne Beispiele nehmen (pointwise loss) (dann gehts halt im Zickzack runter) berechnet sich dann $\ell_2(c(\mathbf{x}), y(\mathbf{x})) = \frac{1}{2}(c(\mathbf{x}) - \mathbf{w}^T\mathbf{x})^2$

- bzw. die weight adaptation: $\Delta \mathbf{w} = \eta \cdot (c(\mathbf{x}) - \mathbf{w}^T \mathbf{x}) \cdot \mathbf{x}$
- für BGD_σ wird Zeile 9 zu
  $\mathbf{w} = \mathbf{w} + \Delta \mathbf{w} + \eta \cdot 2\lambda \cdot \begin{pmatrix} 0 \\ \overrightarrow{\mathbf{w}} \end{pmatrix}$

5.2.3 The Incremental Gradient Descent IGD Algorithm

IGD(D, η)

1. *initialize_random_weights*$(\mathbf{w})$, $t = 0$
2. **REPEAT**
3. $t = t + 1$
4. FOREACH $(\mathbf{x}, c(\mathbf{x})) \in D$ DO
5. $\delta = c(\mathbf{x}) - y(\mathbf{x})$ // $y(\mathbf{x}) \overset{(*)}{=} \mathbf{w}^T \mathbf{x}$, $\delta \in \mathbf{R}$.
6. $\Delta \mathbf{w} \overset{(*)}{=} \eta \cdot \delta \cdot \mathbf{x}$ // $\delta \cdot \mathbf{x}$ is the derivative of $\ell_2(c(\mathbf{x}), y(\mathbf{x}))$.
7. $\mathbf{w} = \mathbf{w} + \Delta \mathbf{w}$
8. ENDDO
9. **UNTIL**(*convergence*$(c(D), y(D))$ **OR** $t > t_{\max}$)
10. *return*$(\mathbf{w})$

- kleinere Schritte als BGD
- can better avoid getting stuck in a local minimum of the loss function then BGD

5.2.4 Linear Regression + Squared Loss

$$L_{0/1}(\mathbf{w}) = \sum_D \frac{1}{2} \cdot (c(\mathbf{x}) - \text{sign}(\mathbf{w}^T \mathbf{x}))$$

$L_{0/1}8\mathbf{w})$ cannot be expressed as a diffrentiable function alias es kann nicht abgeleitet werden damit ist gradient descent nicht möglich

5.2.5 Logistic Regression + Logistic Loss + Regularization

Wie oben nur mit neuer Formel für $\Delta \mathbf{w}$:

$$\Delta \mathbf{w} = -\eta \cdot \nabla \mathcal{L}_\sigma(\mathbf{w}) = \eta \cdot \sum_D (c(\mathbf{x}) - \sigma(\mathbf{w}^T \mathbf{x})) \cdot \mathbf{x} - \eta \cdot 2\lambda \cdot \begin{pmatrix} 0 \\ \overrightarrow{\mathbf{w}} \end{pmatrix}$$

logistic loss Formel:

$$\mathcal{L}_\sigma(\mathbf{w}) = \sum_D -c(\mathbf{x}) \cdot \log(y(\mathbf{x})) - (1 - c(\mathbf{x})) \cdot \log(1 - y(\mathbf{x})) + \lambda \cdot \overrightarrow{\mathbf{w}}^T \overrightarrow{\mathbf{w}}$$

5.3 Multilayer Perceptron

5.3.1 Linear Separability

2 Klassen sind teilbar wenn ich da eine gerade Linie / Ebene / Hyperplane dazwischen packen kann.... oder:
Two sets of feature vectors, X_0, X_1, sampled from a p-dimensional feature space $\mathbf{X}$, are called linearly separable if p+1 real numbers, $w_0, w_1, ..., w_p$, exist such that the following conditions holds:

1. $\forall \mathbf{x} \in X_0 : \sum_{j=0}^{p} w_j x_j < 0$
2. $\forall \mathbf{x} \in X_1 : \sum_{j=0}^{p} w_j x_j \geq 0$

Problem: viele Probleme sind nicht linear separierbar Lösung: wir zeichnen mehrere Linien! (nehmen multilayer perceptron)

- The first, second, and third layer of the shown multilayer perceptron are called input, hidden,and output layer respectively
- input units perform no computation but only distribute the values to the next layer
- Compared to a single perceptron, the multilayer perceptron poses a significantly more challenging training (= learning) problem, which requires continuous (and non-linear) threshold functions along with sophisticated learning strategies.
- a continuous and non-linear threshold function:

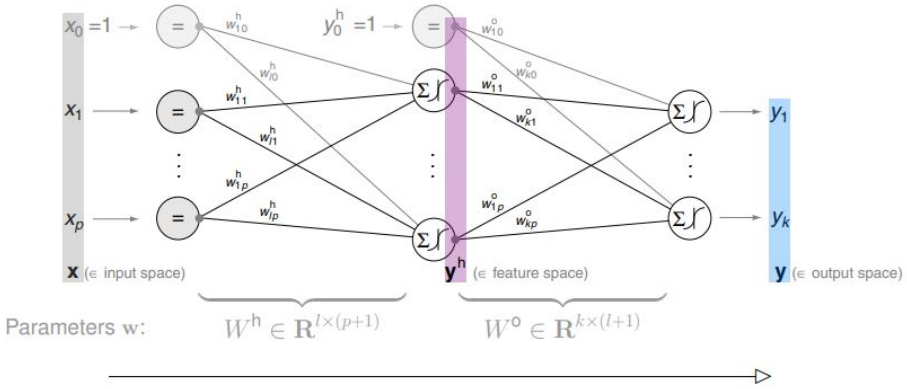
$$\sigma(z) = \frac{1}{1+e^{-z}} \Rightarrow \frac{\delta\sigma(z)}{\delta z} = \sigma(z) \cdot (1 - \sigma(z))$$

- und damit: $y(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^T\mathbf{x}}}$
- eine Alternative zu σ ist:

$$\tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{e^{2z} - 1}{e^{2z} + 1}$$

- A "multilayer" perceptron with linear threshold functions can be expressed as a single linear function and hence is equivalent to the power of a single perceptron only ⇒ Employing a nonlinear is necessary
- Multilayer perceptrons are also called multilayer networks or (artificial) neural network
-

A multilayer perceptron $\mathbf{y}(\mathbf{x})$ with one hidden layer and k-dimensional output layer:



Forward pass computation (aka. forward propagation) :

$$\mathbf{y}(\mathbf{x}) = \sigma\left(W^o\, \mathbf{y}^h(\mathbf{x})\right) = \boldsymbol{\sigma}\left(W^o \begin{pmatrix} 1 \\ \boldsymbol{\sigma}\left(W^h\, \mathbf{x}\right) \end{pmatrix}\right)$$

5.3.2 Forward propagation

The input data is fed in the forward direction through the network. Each hidden layer accepts the input data, processes it as per the activation function and passes to the successive layer

(a) Propagate $\mathbf{x}$ from input to hidden layer: (IGD$_{\text{MLP}}$ algorithm, Line 5)

$$W^h \in \mathbf{R}^{l\times(p+1)} \qquad \mathbf{x} \in \mathbf{R}^{p+1}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w_{10}^h & \dots & w_{1p}^h \\ & \vdots & \\ w_{l0}^h & \dots & w_{lp}^h \end{bmatrix} \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_p \end{bmatrix}\right) = \begin{bmatrix} y_1^h \\ \vdots \\ y_l^h \end{bmatrix}$$

(b) Propagate $\mathbf{y}^h$ from hidden to output layer: (IGD$_{\text{MLP}}$ algorithm, Line 6)

$$W^o \in \mathbf{R}^{k\times(l+1)} \qquad \mathbf{y}^h \in \mathbf{R}^{l+1} \qquad \mathbf{y} \in \mathbf{R}^k$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w_{10}^o & \dots & w_{1l}^o \\ & \vdots & \\ w_{k0}^o & \dots & w_{kl}^o \end{bmatrix} \begin{bmatrix} 1 \\ y_1^h \\ \vdots \\ y_p^h \end{bmatrix}\right) = \begin{bmatrix} y_1 \\ \vdots \\ y_k \end{bmatrix}$$

Forward propagation: Batch Mode

(a) Propagate $\mathbf{x}$ from input to hidden layer: (IGD$_{\text{MLP}}$ algorithm, Line 5)

$$W^{\mathsf{h}} \in \mathbf{R}^{l\times(p+1)} \qquad D \subset \mathbf{R}^{p+1}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w^{\mathsf{h}}_{10} & \dots & w^{\mathsf{h}}_{1p} \\ & \vdots & \\ w^{\mathsf{h}}_{l0} & \dots & w^{\mathsf{h}}_{lp} \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \\ x_{11} & \dots & x_{1n} \\ & \vdots & \\ x_{p1} & \dots & x_{pn} \end{bmatrix}\right) = \begin{bmatrix} y^{\mathsf{h}}_{11} & \dots & y^{\mathsf{h}}_{1n} \\ & \vdots & \\ y^{\mathsf{h}}_{l1} & \dots & y^{\mathsf{h}}_{ln} \end{bmatrix}$$

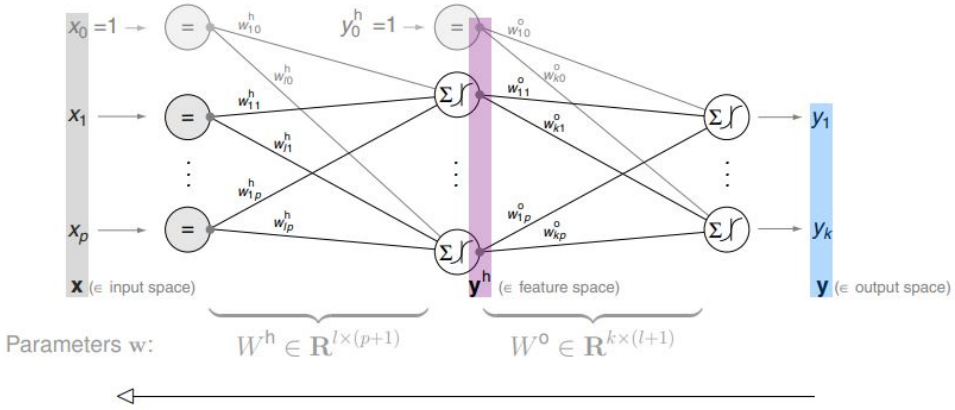
(b) Propagate $\mathbf{y}^{\mathsf{h}}$ from hidden to output layer: (IGD$_{\text{MLP}}$ algorithm, Line 6)

$$W^{\mathsf{o}} \in \mathbf{R}^{k\times(l+1)}$$

$$\boldsymbol{\sigma}\left(\begin{bmatrix} w^{\mathsf{o}}_{10} & \dots & w^{\mathsf{o}}_{1l} \\ & \vdots & \\ w^{\mathsf{o}}_{k0} & \dots & w^{\mathsf{o}}_{kl} \end{bmatrix} \begin{bmatrix} 1 & \dots & 1 \\ y^{\mathsf{h}}_{11} & \dots & y^{\mathsf{h}}_{1n} \\ & \vdots & \\ y^{\mathsf{h}}_{l1} & \dots & y^{\mathsf{h}}_{ln} \end{bmatrix}\right) = \begin{bmatrix} y_{11} & \dots & y_{1n} \\ & \vdots & \\ y_{k1} & \dots & y_{kn} \end{bmatrix}$$

5.3.3 Backwards Propagation

The considered multilayer perceptron $\mathbf{y}(\mathbf{x})$:



Weight update (aka. backward propagation) wrt. the global squared loss:

$$L_2(\mathbf{w}) = \frac{1}{2}\cdot \mathrm{RSS}(\mathbf{w}) = \frac{1}{2}\cdot \sum_{(\mathbf{x},\mathbf{c}(\mathbf{x}))\in D} \sum_{o=1}^{k} (c_o(\mathbf{x}) - y_o)^2$$

- $L_2(\mathbf{w})$ usually contains various local minima