1 Linear Models

1.1 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
  - Regularization (WUHU!)

1.1.1 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.

1.2 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".

1.2.1 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!

2 Neural Networks

2.1 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 $y(\mathbf{x}) = 0$

- wenn $w_0 = -\theta$ und $x_0 = 1$(canonical form)
- sonst $y(\mathbf{x}) = 1 \Leftrightarrow (\sum_{j=1}^{p} w_j x_j - \theta) \geq 0$

2.1.1 PT Algorithm

PT(D, η)

$\quad$ *initialize_random_weights*$(\mathbf{w})$, $t = 0$
$\quad$ **REPEAT**
$\qquad$ $t = t + 1$
$\qquad$ $(\mathbf{x}, c(\mathbf{x})) =$ *random_select*(D)
$\qquad$ $\delta = c(\mathbf{x}) - y(\mathbf{x})$ $\quad$ // $y(\mathbf{x}) \stackrel{(\star)}{=}$ *heaviside*$(\mathbf{w}^T\mathbf{x}) \in \{0, 1\}$, $c(\mathbf{x}) \in \{0, 1\}$ $\leadsto$
$\qquad$ $\Delta\mathbf{w} \stackrel{(\star)}{=} \eta \cdot \delta \cdot \mathbf{x}$
$\qquad$ $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$
$\quad$ **UNTIL**(*convergence*$(c(D), y(D))$ **OR** $t > t_{\text{max}}$)
$\quad$ *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

2.2 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

2.2.1 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

2.2.2 The Batch Gradient Descent (BGD) Algorithm

BGD(D, η)

initialize_random_weights$(\mathbf{w})$, $t = 0$
REPEAT
  $t = t + 1$
  $\Delta\mathbf{w} = 0$
  FOREACH $(\mathbf{x}, c(\mathbf{x})) \in D$ DO
    $\delta = c(\mathbf{x}) - y(\mathbf{x})$ // $y(\mathbf{x}) \overset{(\star)}{=} \mathbf{w}^T\mathbf{x}$, $\delta \in \mathbf{R}$.
    $\Delta\mathbf{w} \overset{(\star)}{=} \Delta\mathbf{w} + \eta \cdot \delta \cdot \mathbf{x}$ // $\delta \cdot \mathbf{x}$ is the derivative of ℓ
  ENDDO
  $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w}$ // $\Delta\mathbf{w}$ is $-\eta \cdot \nabla L_2(\mathbf{w})$ here.
UNTIL(*convergence*$(c(D), y(D))$ **OR** $t > t_{\max}$)
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
  $\mathbf{w} = \mathbf{w} + \Delta\mathbf{w} + \eta \cdot 2\lambda \cdot \binom{0}{\vec{\mathbf{w}}}$

2.2.3 The Incremental Gradient Descent IGD Algorithm

IGD(D, η)

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

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

2.2.4 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

2.2.5 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:

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

2.3 Multilayer Perceptron

2.3.1 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. the solution is unique,