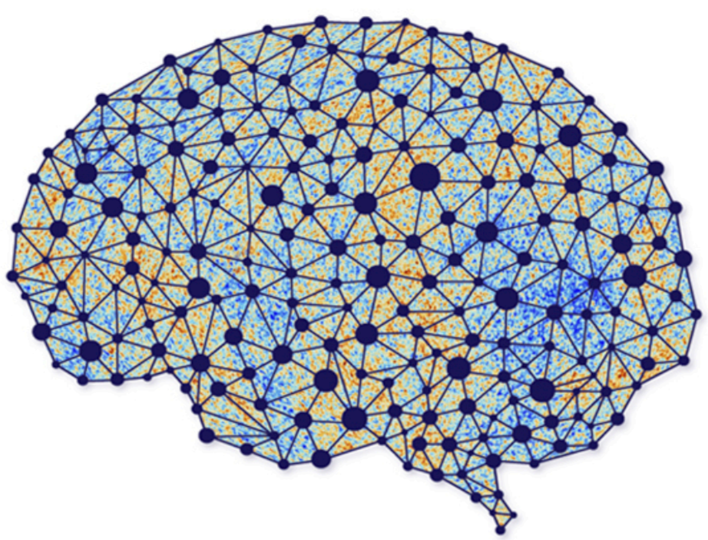
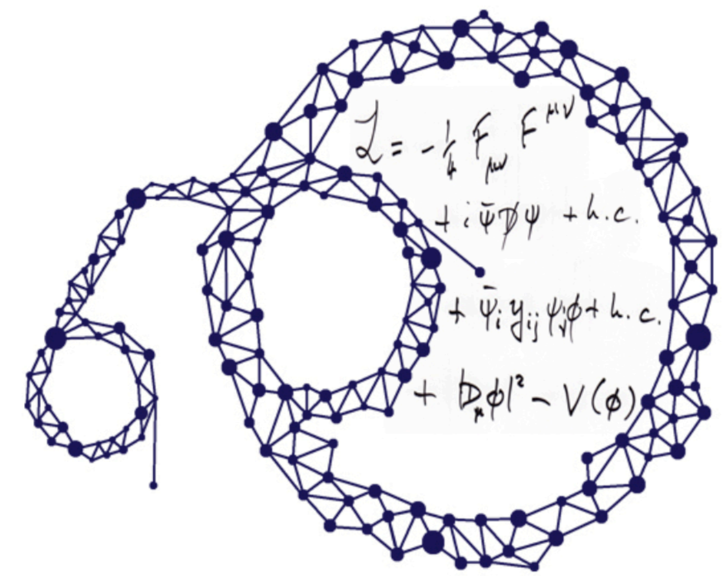
Physics 361 - Machine Learning in Physics

Lecture 4 – Basics of Machine Learning

Jan. 29th 2025



AI
∩
Universe



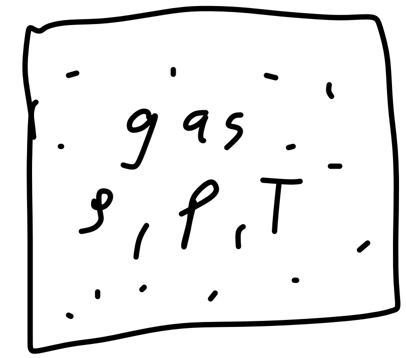
Moritz Münchmeyer

Unit 1: Background

1.4 Information theory background (cont.)

Entropy

- In statistical physics (thermodynamics) the entropy is given by



$$S = (k_B) \log \Omega$$

here we set it to 1

Ω: number of microstates (equally likely)

- We can re-write this as

$$S = -\log p \quad \text{where} \quad p = \frac{1}{\Omega}$$

is the probability of each m. state

- The general definition of entropy

Shannon entropy $$\boxed{S = -\sum_i p_i \log p_i}$$

- log here is base e $\rightarrow$ entropy is in „nats"

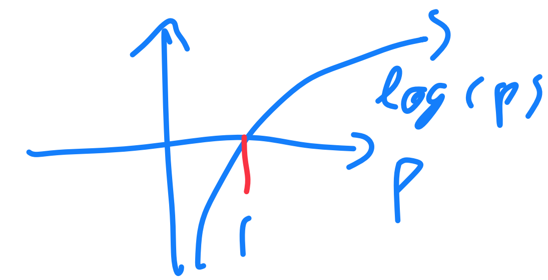
(base 2 $\rightarrow$ " in bits)

Properties of the entropy

- <u>discrete PDF</u> $$S = -\sum_i p_i \underbrace{\log(p_i)}$$

$-\log(p_i)$ is the „self-information" of event i

$$= -\mathbb{E}_{x \sim p(x)} \left[\log(p(x)) \right]$$



A unlikely event has a large self info.

self information $I(x) = -\log(p(x))$

A certain event has self inform. 0

maximum possible entropy is $S = \log(N)$

- <u>continuous PDF</u> $$S = -\int dx\, p(x) \log(p(x))$$

$$= -\mathbb{E}_{x \sim p(x)} \left[\log(p(x)) \right]$$

More uniform distributions have a higher entropy.
(spread out)

Kullback - Leibler divergence

- The relative entropy = KL divergence provides a measure of the similarity of two probab. distr. $P(x)$ and $Q(x)$:

difference of self information of sample x

$$D_{\mathrm{KL}}(P\|Q) = \mathbb{E}_{\mathrm{x}\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = \mathbb{E}_{\mathrm{x}\sim P}\left[\log P(x) - \log Q(x)\right]$$

- If P and Q are the same then $D_{KL} = 0$.

  e.g in generative ML

  $P(x)$: true (unknown) PDF of the training images

  $Q(x)$: PDF of generated images

- D_{KL} is easy to evaluate by sampling (if P and Q are known).
- $D_{KL} \geq 0$

- Interpretation: If our data is described by P and we have a theory/model Q that is meant to describe the data, at what rate will collection of data inform me that theory Q differs from P.

  = KL divergence

- Minimizing the KL divergence is the same as maximizing the likelihood of the data given the model Q. We define the cross-entropy as the part of the KL divergence that depends on the parameters of Q.

  Cross entropy $H(P,Q) = -\mathbb{E}_{x \sim P(x)} \log [Q(x)]$

- The KL divergence is not a true distance metric. E.g. it is not symmetric in P, Q.

  Other alternatives: - Wasserstein distance
  - F-divergence

- Another interesting concept: mutual information

$$I(X,Y) = D_{KL}[\,p(x,y) \,||\, p(x)\,p(y)\,]$$

vanishes if X, Y are independent.

All of these information theory concepts are frequently used in machine learning.

Unit 2: Basics of Machine Learning

Sources: e.g. deeplearningbook.org

Overview

Most machine Learning algorithms have the following elements:

- dataset
  - training data
  - test data
  - validation data
- cost function / loss function / training objective
- model / architecture
- optimization procedure

We first discuss these at the example of curve fitting.
Then we train a simple neural network.

Unit 2: Machine Learning Basics

2.1 Machine Learning concepts using the example of Linear and Polynomial Regression

Linear regression

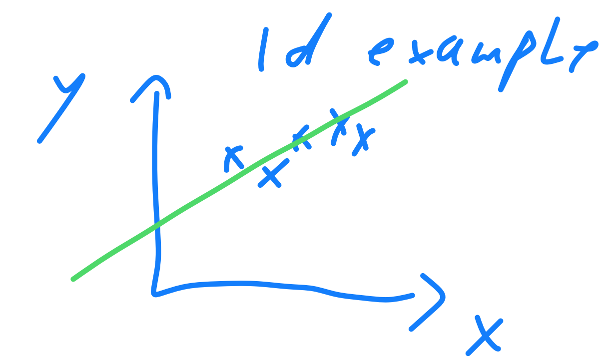
- The simplest machine learning algorithm.
- Predict 1 number from N features:

$$\hat{y} = \vec{w}^T \vec{x} + b$$

This can be re-written as

$$\hat{y} = \vec{w}^T \vec{x}$$

where we include b by adding an element 1 to x $\quad x = \begin{pmatrix} \vdots \\ 1 \end{pmatrix}$



- notation: We define the design matrix as

$$\mathbf{X} = \begin{array}{c} \\ \text{example} \end{array} \begin{array}{c} \text{features} \\ \begin{pmatrix} | & | & | & | \\ | & | & | & | \end{pmatrix} \end{array}$$

$\mathbf{X}^{\text{train}}$: training data

$\vec{y}$: training labels

vector over examples

- training set $\{\mathbf{X}^{\text{train}}, \vec{y}^{\,\text{train}}\}$

  test set $\{\mathbf{X}^{\text{test}}, \vec{y}^{\,\text{test}}\}$

- The typical cost function for such regression problems is the Mean Squared Error (MSE).

$$MSE = \frac{1}{m} \sum_i (\hat{y}_i - y_i)^2$$

prediction truth

for linear regression: $\hat{y}_i = \vec{w}^T x_i$.

- Notation: L_p norm of a vector is

$$\|\vec{x}\|_p = \left(\sum_i |x_i|^p \right)^{1/p}$$

Thus we can rewrite

$$MSE = \frac{1}{m} \| \hat{\vec{y}} - \vec{y} \|_2^2$$

- Goal is to minimize the MSE.

train to minimize MSE_{train} wrt. $\vec{w}$.

→ hope MSE_{test} will also be small.

• For linear regression we can solve for $\vec{w}$ analytically:

$$\nabla_{\boldsymbol{w}} \mathrm{MSE}_{\text{train}} = 0$$

$$\Rightarrow \nabla_{\boldsymbol{w}} \frac{1}{m} ||\hat{\boldsymbol{y}}^{(\text{train})} - \boldsymbol{y}^{(\text{train})}||_2^2 = 0$$

$$\Rightarrow \frac{1}{m} \nabla_{\boldsymbol{w}} ||\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}||_2^2 = 0$$

$$\Rightarrow \nabla_{\boldsymbol{w}} \left(\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right)^\top \left(\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - \boldsymbol{y}^{(\text{train})}\right) = 0$$

$$\nabla_{\boldsymbol{w}} \left(\boldsymbol{w}^\top \boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - 2\boldsymbol{w}^\top \boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})} + \boldsymbol{y}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})}\right) = 0$$

$$\Rightarrow 2\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\boldsymbol{w} - 2\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})} = 0$$

$$\Rightarrow \boxed{\boldsymbol{w} = \left(\boldsymbol{X}^{(\text{train})\top}\boldsymbol{X}^{(\text{train})}\right)^{-1}\boldsymbol{X}^{(\text{train})\top}\boldsymbol{y}^{(\text{train})}}$$

we need matrix multiplication and inversion.

deeplearningbook.org

Background on matrix calculus

(required to understand the last slide, but not often used in this course)

derivative of quadratic forms:

1. For a symmetric matrix A:

$$\nabla_w (w^T A w) = 2Aw.$$

2. For a general (not necessarily symmetric) matrix A:

$$\nabla_w (w^T A w) = (A + A^T)w.$$

3. Gradient with respect to w^T:

$$\nabla_{w^T} (w^T A w) = w^T (A + A^T).$$

derivatives of inner products

- If $f(w) = b^T w$ for a vector b, then:

$$\nabla_w (b^T w) = b$$

- If $f(w) = w^T b$, then

$$\nabla_w (w^T b) = b$$

Properties of Transposition

- $(AB)^T = B^T A^T$
- $(A^T)^T = A$
- If A is square and invertible, then $(A^{-1})^T = (A^T)^{-1}$.

since bilinear forms are scalars we have:

$$w^T A b = (w^T A b)^T$$

much more about this:

Reference: "Matrix Cookbook" https://www.math.uwaterloo.ca/~hwolkowi/matrixcookbook.pdf

Linear regression (cont'd)

Very simple ML model:

$$\hat{y} = \vec{w}^T \vec{x}$$

$\vec{x}$: input

$\vec{y}$: output

training data:

$$X^{train} = \overbrace{\left(\qquad\qquad\right)}^{\text{features}} \left.\vphantom{\Bigg(}\right\} \text{examples}$$

$$\vec{y}^{\,train} = \text{desired output} = \text{label}$$

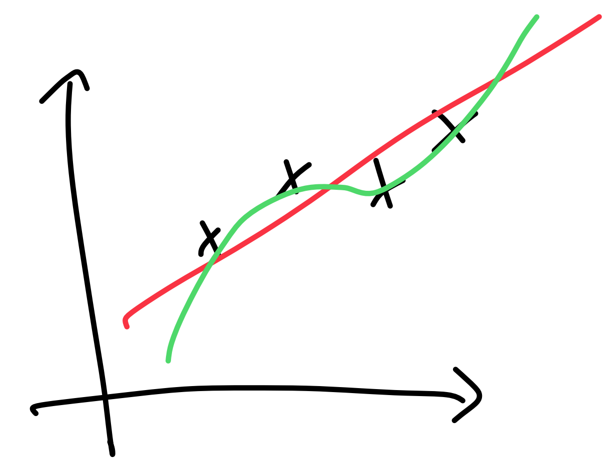
Loss function:

$$MSE = \frac{1}{m} \| \underbrace{\hat{\vec{y}}}_{\text{estimate}} - \underbrace{\vec{y}}_{\text{training "label"}} \|_2^2$$

Can be minimized w.r.t. to $\vec{w}$:

$$\vec{w} = \left(X^{train\,T} X^{train} \right)^{-1} X^{train\,T} \vec{y}^{\,train}$$

Polynomial regression



- In polynomial regression we fit a higher order polynomial.

  E.g. 2nd order pol. $\hat{y} = b + w_1 x + w_2 x^2$

  General: $\hat{y} = b + \sum_{i=1}^{N} w_i x^i$

- This model can be solved for $\vec{w}$ with the same equations as linear regression because it is still linear in the parameters w_i.

Capacity, Overfitting, Underfitting

- How well does a model trained/fitted to the training data work on novel data.
  $\Rightarrow$ Generalization error

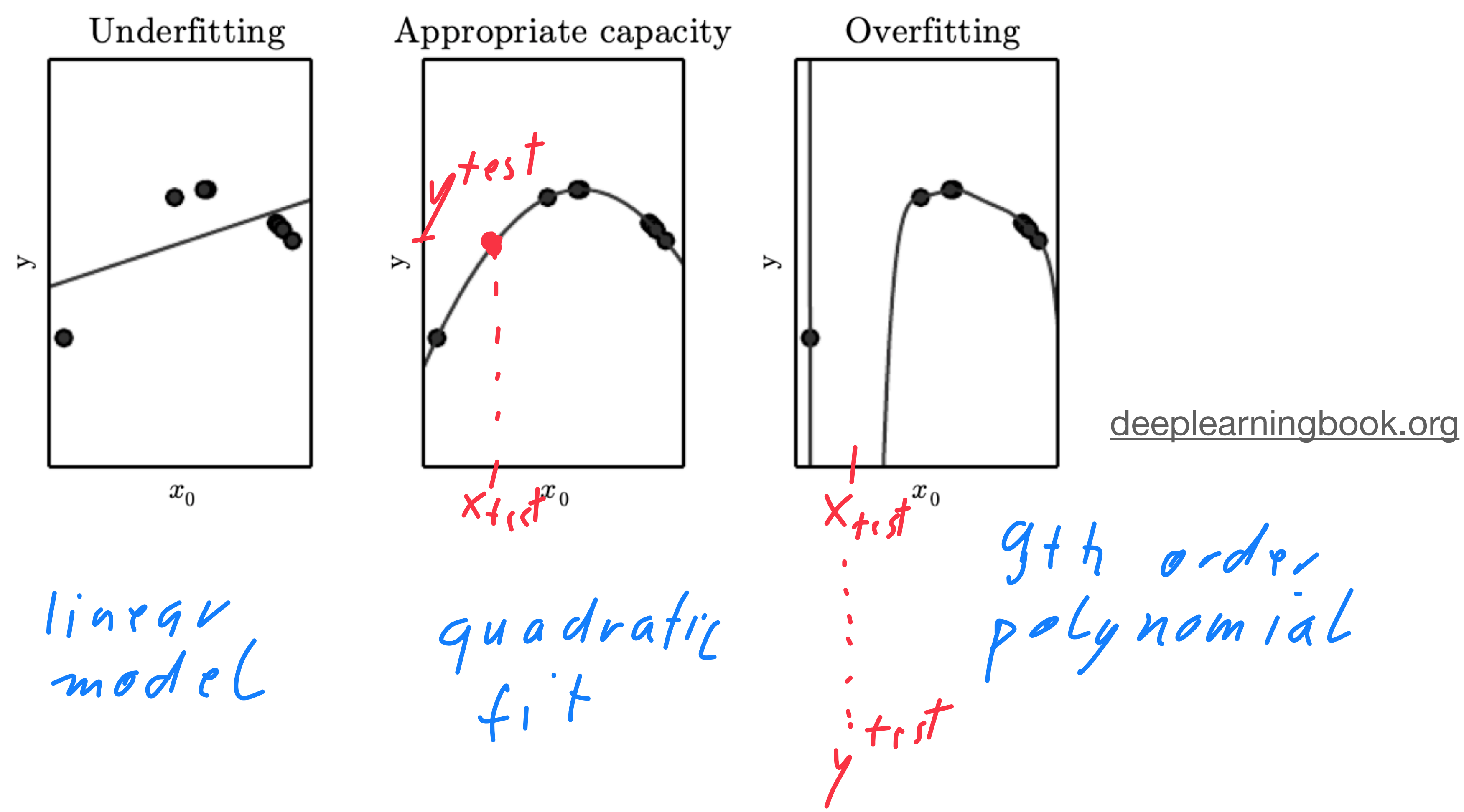
- We usually assume that training data and test data are drawn from the same distribution. „i.i.d." data: independent identically distributed

- Generalization is closely related to overfitting and underfitting.

  underfit: training error is large

  overfit: training error is small but test error is large.

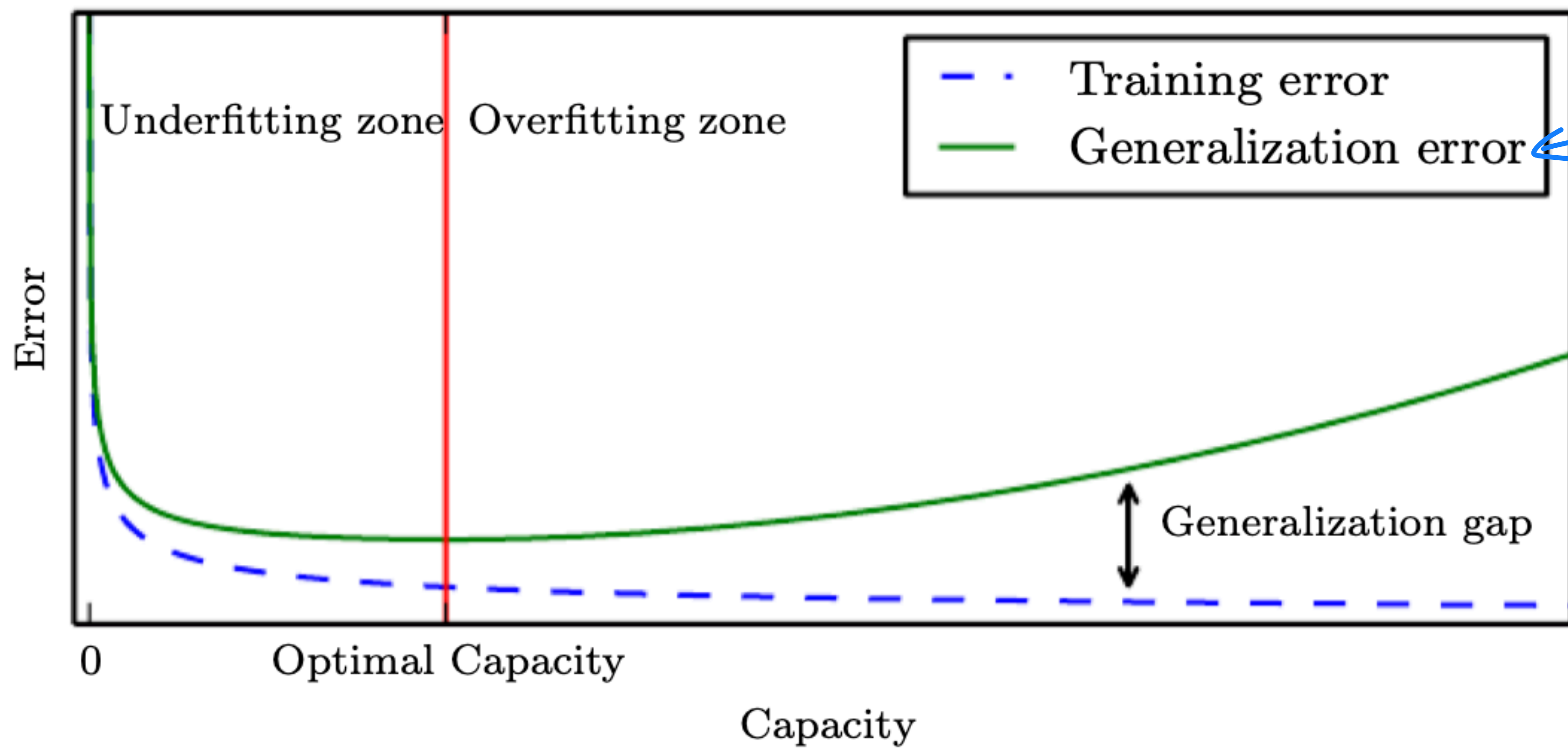
Example of polynomial regression



- The space of functions that the model can represent is called the capacity.
- Occam's razor: chose the simplest „that fits"
- Lower capacity → tends to generalize better
  Higher capacity → reduces training error.

⇒ Need to optimize the capacity or „regularize."

Typical behavior of error vs capacity (after training out the model)



error on the test data set

deeplearningbook.org

We could e.g. train models with various orders of polynomials (or neural network parameter numbers).

Usually this is not the main approach. Instead we use „regularization" to avoid overfitting.

Regularization

- Idea: Keep the model capacity fix, but encourage simpler solutions.

- Common method: weight decay

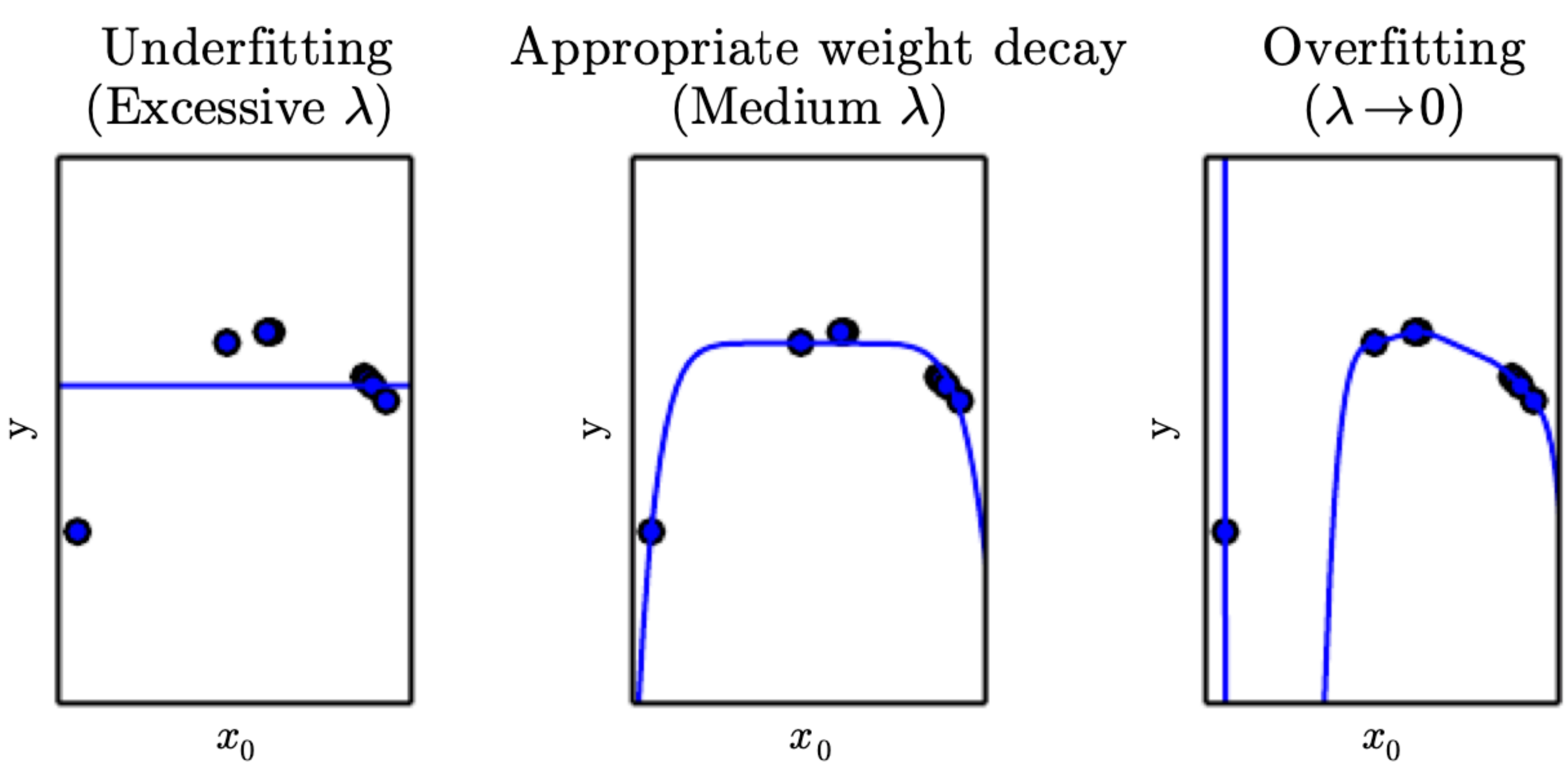
New loss to minimize $J(\vec{w}) = MSE_{train} + \lambda \underbrace{\vec{w}^T \vec{w}}$

parameter to chose "lambda"

L-2 norm

L-2 regularization

This discourages large weights



deeplearningbook.org

L-2 regularization is one of several popular methods of regularization

$\boxed{\text{Regularization: any method meant to prevent overfitting without changing the model capacity.}}$

Two other popular methods which we discuss later:

- drop-out
- early stopping.

Hyperparameters

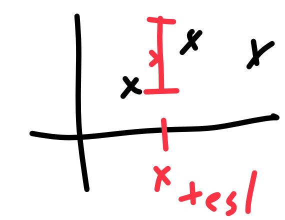
- Hyperparameters are parameters that are NOT part of the fitting/optimization procedure.

- Examples:
  - model architecture
    e.g. - maximum order of polynomials
    - depth of a neural network
  - Learning parameters
    e.g. learning rate
  - Regularization parameters e.g. λ

- To chose the best hyperparameters one often splits off a validation data set from the training data (~20% thereof).

Relation of MSE and maximum Likelihood

- In Linear regression we were Learning y from x.
  Instead now we want to Learn $p(y|x)$.
  $\Rightarrow$ we get an error bar.



- If we assume that the error is Gaussian we want to Learn:

$$p(y|x) = \mathcal{N}(y;\ \underbrace{\hat{y}(\vec{x},\vec{w})}_{\text{mean of prediction}},\ \underbrace{\sigma^2}_{\text{width of prediction error.}})$$

- The Loss is now the Likelihood of the training data:

plug in

$$\mathcal{L}(\theta) = \sum_{i=1}^{m} \log p(y^{(i)} \mid x^{(i)}; \theta)$$

model parameters, here $\vec{\theta} = \vec{w}$ in Linear regression.

$$= -m \log \sigma - \frac{m}{2} \log(2\pi) - \boxed{\sum_{i=1}^{m} \frac{|\hat{y}^{(i)}_{(\theta)} - y^{(i)}|^2}{\sigma^2}}$$

MSE Loss

- We want to maximize this with respect to θ.
  Maximize $\mathcal{L}$ is the same as minimizing MSE.

Unit 2: Machine Learning Basics

2.2 Neural Network Basics (Supervised Learning)

Supervised machine Learning

We want to Learn a complicated non-linear function to map input $\vec{x}$ to output $\vec{y}$.

$$\vec{x} \longrightarrow \boxed{f_\theta(\vec{x})} \longrightarrow \vec{y}$$

e.g. neural network

θ: model parameters = weights

We want to find the model parameters by minimizing some Loss function on our training data.

E.g.

$$L^{MSE}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(y_i - f_\theta(\vec{x}_i) \right)^2$$

A very simple neural network

- We need some way to specify the function $f_\theta(\vec{x})$. It turns out that a very simple „architecture" works very well in practice:

  Linear transformations combined with element-wise non-linearities

- A linear transformation / linear layer* is given by $\vec{y} = \vec{f}(\vec{x}) = W\vec{x} + \vec{b}$

  $\vec{x}$: N_{dim}

  $\vec{y}$: M_{dim}

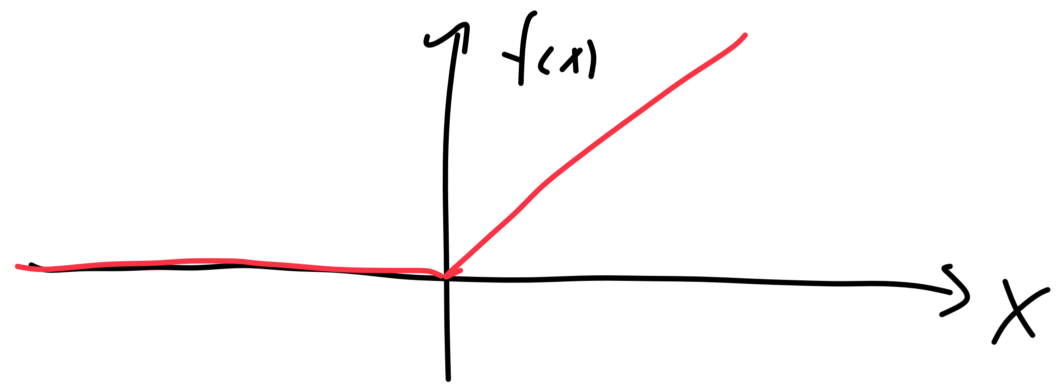
  W: matrix of weights

  $\vec{b}$: bias

- The most common non-linearity / activation function is the ReLU „rectified linear unit"

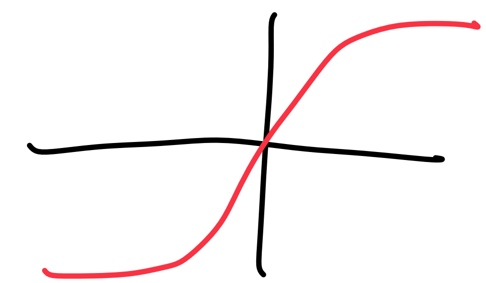
* fully connected linear layer

- ReLU is a 1-dimensional function



Piecewise linear function.

Alternatives: tanh, sigmoid



- A multilayer perceptron (MLP) is a „deep neural network" made out of a stack of the 2 building blocks.

$\vec{x} \longrightarrow \boxed{\text{Linear}} \longrightarrow \boxed{\text{reLU}} \longrightarrow \boxed{\text{Linear}} \longrightarrow \boxed{\text{reLU}} \text{ — , } \dots \longrightarrow \vec{y}$

The combination of ReLU and Linear Layer can be written as

$$f^i(x) = \underbrace{\max}_{\text{ReLU}}(0, W^i_j x^j + b^i)$$

component notation

next Layer uses $f(x)$ as x input.

If we stack these layers:

$$\max\left(0, W^i_{n,j} \max\left(0, W^j_{n-1,k} \max\left(0, \cdots\right) + b^j_{n-1}\right) + b^i_n\right)$$

Now we have a function that we can „fit" to the training data.

Loss for classification

For classification we want the NN output to be the probabilities of the various classes.

$$\vec{x} \longrightarrow \boxed{\text{MLP}} \longrightarrow y_j \xrightarrow{\text{softmax}} P_j(\vec{x})$$

e.g. image

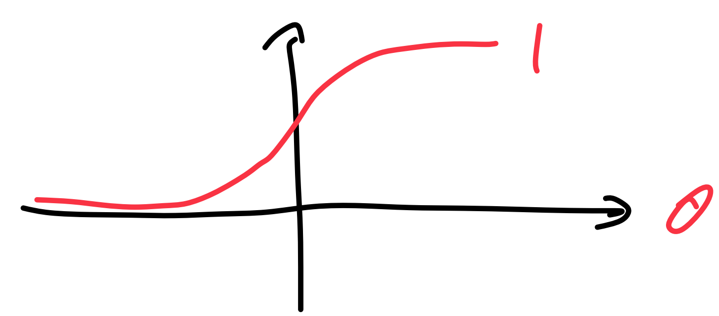
„logits" (y_j)

probability for $\vec{x}$ to be in class j. ($P_j(\vec{x})$)

A probability distribution need to have only positive probabilities and they must sum to 1.

To achive this we need the softmax function

$$P_j(y_j) = \frac{e^{y_j}}{\sum_i e^{y_i}}$$



The typical loss function for classification is the negative log likelihood = cross-entropy:

predicted by NN

$$L(\theta) = -\sum_{i=1}^{N_{examples}} \sum_{j=1}^{N_{classes}} y_j^i \log P_j(\vec{x}_i)$$

parameters of NN: W, b

$$y^i = \begin{pmatrix} 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{pmatrix}$$

one-hot vector which is 1 for the right class.

Course logistics

- **Reading for this lecture:**
  - **For example:** Deeplearningbook.org chapter 5.
- **Problem set**: First problem due Sunday night.