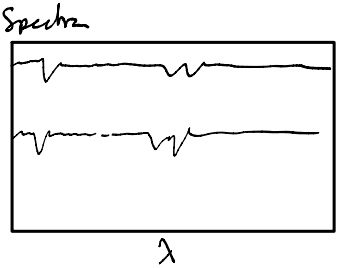
MLYP

Lecture 2

Types of ML:

- Nearest neighbor
- Linear regression



$\dim(y_i) = 7\text{e}3$ pixels per spectra

n = # spectra in training: (x_i)

Introduction of symbols

Supervised (Training):

Training set: { features: x_i  .jpg.
labels: y_i bvr 792

Train $f(x_i, \hat{W}) \sim y_i$,

Test data: x_i^*, y_i^* & x_i, y_i to find the weights, $\hat{W}$.

$$\underset{n\times p}{X} = [x_1, \ldots x_n]^T$$

Fitting, in closed form:
Note: Not lin. regression

$y_i = \vec{a} \cdot \vec{x}_i + \text{noise}$

$\vec{a}$: p-vector, $\hat{a} = (X^T C^{-1} X)^{-1} X^T C^{-1} Y$, $Y = [y_1, y_2, \ldots y_n]^T$

↳ Similar to W.

▷ Fitting: I want to know $\vec{a}$

Regression: We want to predict y^*.

- Seek $\hat{a}$ s.t. $E[|\hat{a} \cdot \vec{x}^* - y^*|]_{x^*, y^*}$ is small.
- features are independent
- test data are drawn from the same distribution as training

- is a linear relation
  - ↳ Between $\hat{a}$, $\vec{x}$?

Then: $\underset{p\times 1}{\hat{a}} \leftarrow \underbrace{(\underset{p\times n}{X^T}\underset{n\times p}{X})^{-1}}_{p\times p}\underset{p\times n}{X^T}\underset{n\times 1}{Y}$ delivers the MVUE

- ↳ symmetric, real, non-negative eigenvalues.

When: $p \to \infty$: $\hat{a} \leftarrow X^T(XX^T)^{-1}Y$

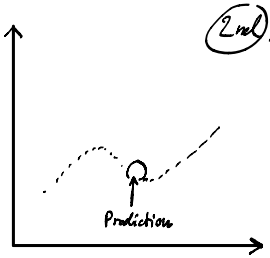
Useful python

`linalg()`, `@`, `solve()`

never use: `inv`.

Lecture 3

Linear Regression Revisited



(2nd) ML definition: We care about data, not parameters
↑ prediction of

In fact, we will choose a larger number of parameters than data.

Important Lin. Alg.
- Rank
- SVD
- Condition #
- Non-negative definite.
  ↳ $\lambda \geq 0$

Recall $\hat{a}$ = vector of parameters

$$= (X^T X)^{-1} X Y$$

$p \times n \quad n \times p$

rank $\leq \min(n, p)$

Since we only care about the prediction of data:

$$\begin{cases} \hat{y}^* = X^* \vec{a} \\ \hat{y}^* = X^* (X^T X)^{-1} X Y & n > p \\ \hat{y}^* = X^* X^T (X X^T)^{-1} Y & n < p \end{cases}$$

Pseudo inverse: $Y^* = X^* X^+ Y$

X^+ definition: $\forall Y = X^+ Y = \underset{\vec{a}}{\operatorname{argmin}} \underbrace{\| Y - X \cdot \vec{a} \|_2^2}_{\text{if } n>p \text{ this cannot be solved}} + \lambda \|a\|_2^2$

Regularisation Parameter

if $p > n$ we need λ

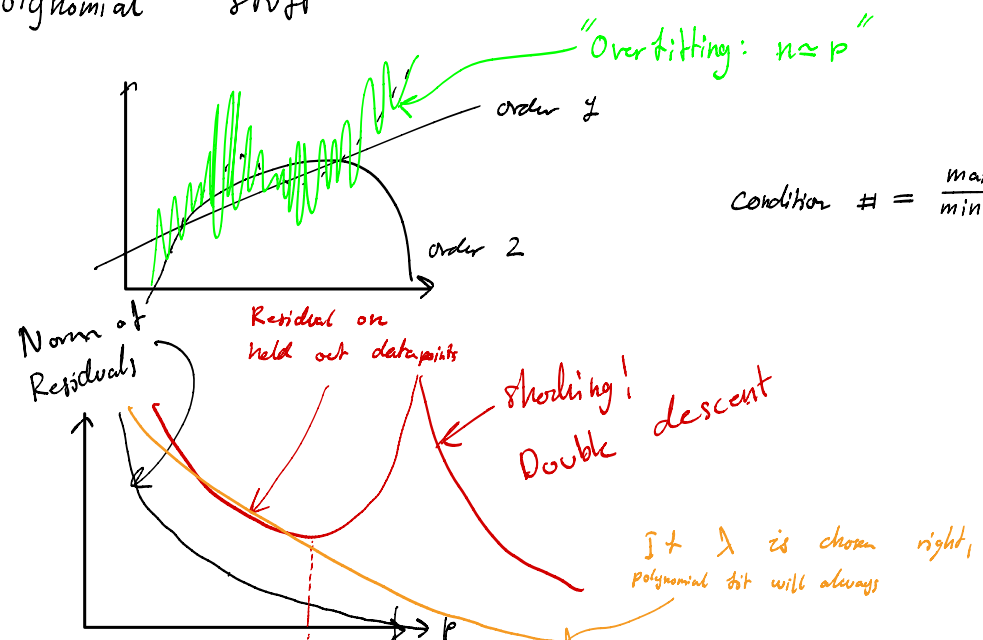
$$\|q\|_a^b \equiv \left[\sum_i |q_i|^a\right]^{b/a}$$

↑ Norm since $\vec{a}$ can in general be a matrix/tensor.

$$\Lambda \equiv \lambda I_{p\times p}, \qquad \hat{y}^* = \lim_{\lambda\to 0} X^*\overbrace{(X^TX + \Lambda)}^{p\times p}{}^{-1}X^TY$$

$$\hat{y}^* = \lim_{\lambda\to 0} X^*\Lambda^{-1}X^T\underbrace{(X\Lambda^{-1}X^T + I)}_{n\times n}{}^{-1}{}_{n>n}Y$$

Polynomial Stuff



Condition # = $\frac{\max \lambda_i}{\min \lambda_j}$

↑ ?

Lecture 4

So far only Lin. reg., how do we evaluate the quality of a model?

- Train, validation, test, cross validation

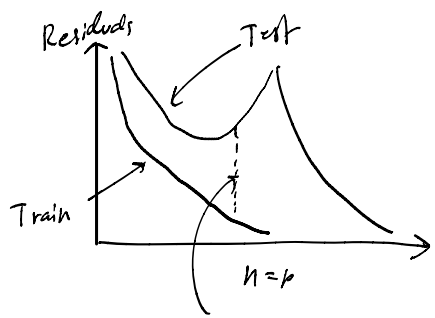
- Feature importance

Supervised: ML is about prediction.

A good (ML) model predicts held out data well.

Last time (No distribution): $\hat{a} = \underset{a}{\operatorname{argmin}} \underbrace{\left[\| Y - X \cdot a \| + \lambda \|a\| \right]}_{\text{Loss function (or cost)}}$

Overfitting:



Difference is large? Probably "overfitting"

No splitting of data in train, validation, test.

Already for our simple, p parameter n data points Lin. Reg. we have choices on the order/number of parameters. If our choices is inspired by the validation or test data, the ML model might not perform on new data.

Training data is used to set the weights:

$$\hat{W} = \underset{W}{\operatorname{argmin}} \sum_{i \in \text{training}} w_i \, \text{loss}\left(\left[y_i - f(x_i; W) \right] \right) + \text{reg}(W)$$

- Choices:

hyper-parameters

- High level methods (in this course: lin. reg., k-nearest neighbors, MLP Multi layer percept...)
- Integers, polynomials, cos/sin, choices within each method.
  - Not $\mathbb{Z}$, but in general Yes/no questions.
    - especially neural network architecture
- Loss functions
  - Robustness
  - Regularization
  - Symmetries
  - Sparseness
- Continuous parameters

↳ Hyper parameters, θ, are different than weights (continuous parameters), because they cannot be optimized. We have to make a choice.

Validation

assess $Q_\theta = \sum_{i \in \text{validation}} w_i \, \text{loss}\left(y_i - f(x_i; \hat{W}_\theta) \right)$

↑ optimized on training

Multi - Layer - Perceptron

Perceptron: Linear action, non-linear response

: The small element that intelligent systems are made of.

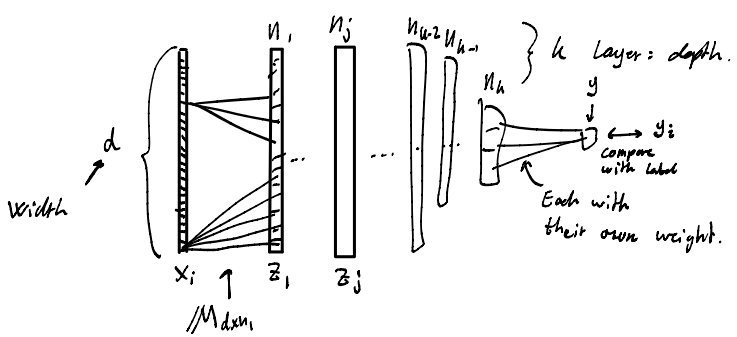
$$y = f(x; W), \quad \hat{W} = \arg\min_{W} \left[\sum_{i \in \text{train}} \text{loss}\left(y_i - f(x_i; W)\right) + \lambda \|W\| \right]$$

MLP ↑ Last time ↑ $\hat{a}$ is the specific case for Lin. reg.

The simplest case, where x (input data) has a fixed shape:

$x_i \in \mathbb{R}^d$ (Just a list of numbers)

$X_{\text{train}} = [x_1, x_2, \ldots, x_n]$ ← # of data samples



If every node in every layer is connected to every node in the next layer, the graph is said to be fully connected. If not, the network is said to be sparse.

↑ Weights for each d entry connecting n_1 nodes in the next layer. The weights depend on all other entries in a non-linear way?

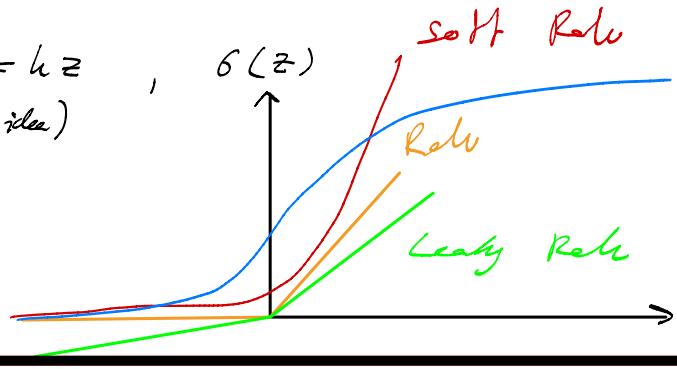
$$z_j = \sigma\left(W_j \cdot z_{j-1} - b_j\right)$$

$n_j \times n_{j-1}$ n_{j-1} n_j

Non-lin. included here, applied to every entry of the vector in parenthesis. They're called Activation Functions

Examples of non-linearities:

$\sigma(z) = kz$ (ideal idea)



Lecture 5

Optimization in Neural Networks

Call that in the Training we wanted:

$$\underset{W}{\operatorname{argmin}} \sum_{i \in \text{train}} \text{loss}\left(y_i - \overbrace{f(x_i; W)}^{\text{could be MLP}}\right)$$

- Autodifferentiation: $\left.\frac{df}{dw}\right|_{x_i}$ exists because the non-linearity is differentiable and its argument is simply linear.
- Back propagation

Convex ← loss — labels (contain cat?)

total loss: $Q = \sum_{i \in train} loss\left(y_i - f(x_i, W)\right)$

data: (youtube videos)

$$\frac{d\,loss}{dW} = loss'\left(y_i - f(x_i, W)\right)\left[-\frac{\partial f}{\partial w}\Big|_{x_i}\right]$$

$$\Rightarrow \frac{dQ}{dW} = \sum_{i \in train} \frac{d\,loss_i}{dW}$$

"sub"

A small (say 10%) subset of x_i can be used to become an estimator for $\frac{dQ}{dW}$.

---

convex function:

$y_i \quad f(x_i; W)$

distance is ≥ 0

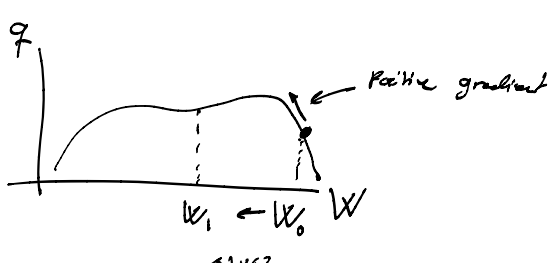
$$\frac{d\hat{Q}}{dW} = \frac{N_{train}}{N_{sub}} \underbrace{\sum_{i \in sub}}_{\text{Batch}} \frac{d\,loss}{dW}$$

Stochastic gradient

↑ Subset is randomly chosen.

Leon Bottou: Stochastic gradient descent

Optimization:



Optimizers:

Newton's method: The next step, $\Delta w = \frac{\frac{dq}{dw}}{\frac{d^2q}{dw^2}}$

Great for 1D w. For higher Dim. w:

- Conjugate gradient method (great)
- BFGS (Best, but complicated)

Why not Newton? Let's look at cost. For $\dim(w) = 10^9$

$$\frac{\frac{dq}{dw}}{\underbrace{\frac{d^2q}{dw^2}}_{10^9 \times 10^9 \text{ matrix. not cheap.}}}$$

For astronomical dim of w, we cannot use regular optimizers.

Stochastic Gradient Descent

You are at w_t at iteration t.

You know: $\frac{\hat{dQ}}{dw}\Big|_{w_t}$ subset

$$w_{t+1} = w_t + \alpha_t \frac{\hat{dQ}}{dw}\Big|_{w_t \text{ subset}}, \quad \alpha_t = \frac{\alpha_0}{(t+a)^\gamma}$$

Training rate.

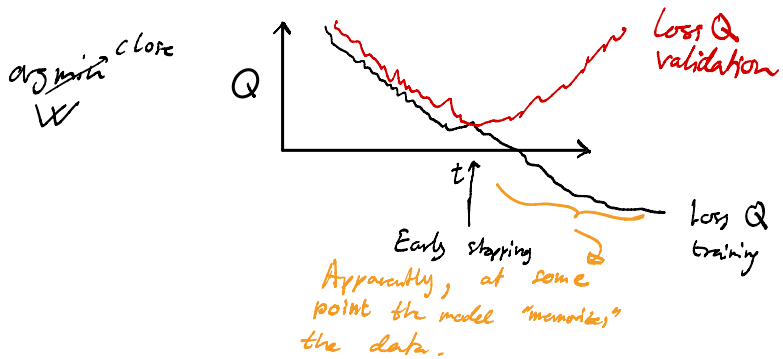
For some choice of a, γ You will be sure find the minimum.

Conceptual importance in ML. Might seem like a poor optimizer, but we cannot afford to take $\frac{dQ}{dw}$ more than once.

① A better method: ADAM.

② The Landscape is terrible: There is only local minima.
Due to degeneracies, There will never be a global minimum

③ You don't want to be at the local minimum.



④ Performance on the validation is the same as test.

Lecture 6

Unsupervised ML

Data without labels, Y.

Note:
There is a small difference between self-supervised and unsupervised.

① Clustering ⟷ ② Low-Dim. structure.

↕

③ De-noise data ↗

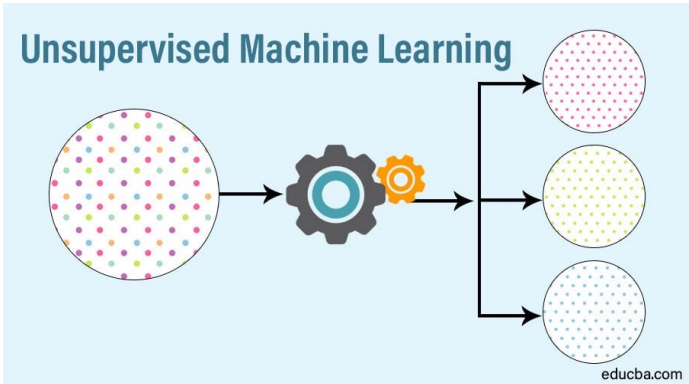
Maybe 7k pixels pr data, but maybe it is only a subset that matters?

All these are related!

④ Predict or generate new data

Clustering / Labelling:

⑤ Assess the consistency of new data.



Methods of unsupervised learning:

(A) Principal Component Analysis

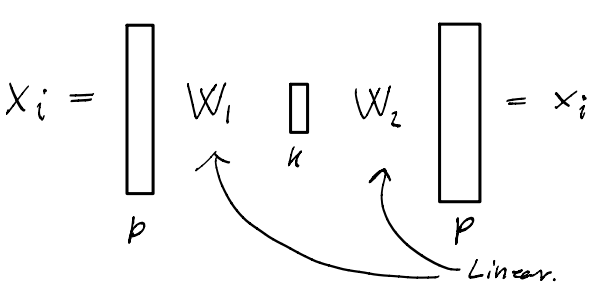
Features, x. Each n are a p-vector. All features in one:

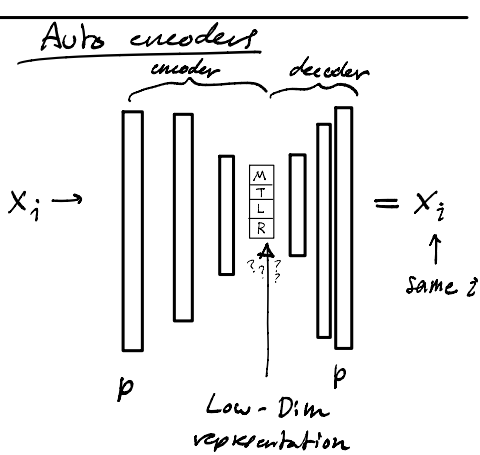
$$\underset{n\times p}{X} = [x_1, x_2, \ldots, x_n]^T = \underset{n\times k}{A} \cdot \underset{k\times p}{G} + \text{residuals}$$

(Matrix factorization)

rank k, $k < \min(n, p)$

$$Q = \underset{A,G}{\operatorname{argmin}} \|X - A\cdot G\|_2^2 \quad @ \text{ rank } k.$$

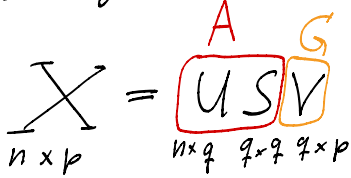




Methods for small Data:

Single Value Decomposition: SVD

"Finding eigenvalues" for rectangular matrices.



$q = \min(n, p)$

Recall: Square X:

$$X = U^T X U$$

Recall pseudo inverse

$$(X^T X)^{-1} X^T = X^T \quad n > p$$

$$X^T (X X^T)^{-1} = X^T \quad n < p$$

Pseudo inverse for PCA:

$$X^T = V^T \text{ "}\tfrac{1}{S}\text{" } U^T$$

↳ For zeros, return zero.

Big Data: Low-Rank SVD

One liner in "PsyKitLearn".

Comments on PCA:

1. Workhorse
2. Outliers are bad: $X = A \cdot G + S + \text{residual}$ (↑ handles outliers.)
3. $p \to \infty$ & kernel PCA