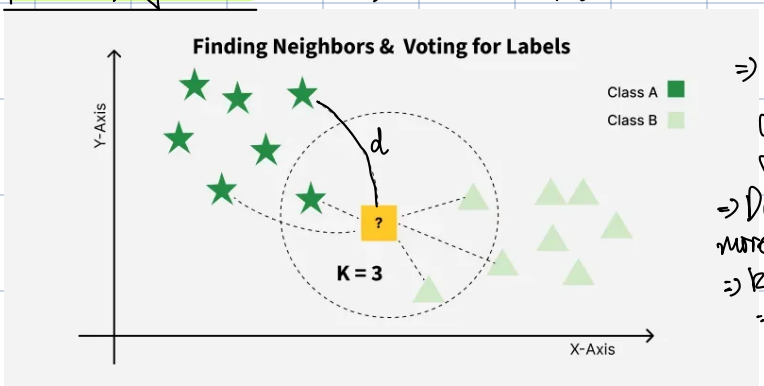
mlfp (Python 3.12.0)

Model is not actually doing general learning, but simply "memorizing" the dataset ⇒ Overfitting

⇒ Dataset too small (not enough for general learning) & Model has more learning parameters than it needs to correctly model the dataset.

① **k-NN Algorithm:** i) Euclidean Distance $d(x, x_i) = \|x - x\| = \sqrt{\sum_{j=1}^{d} (x_j - x_j)^2}$ (d-dim.) Minkowski: $d = \|x - x'\|_p = \left(\sum_{i=1}^{d} (x_i - x_i)^p\right)^{1/p}$



⇒ no assumptions about the underlying data distribution it makes a non-parametric and distance-based learning method.

⇒ Data has lots of noise or outliers, using a large k making predic more stable.

⇒ k too large ⇒ model become simple, loss of important patterns = "underfitting"

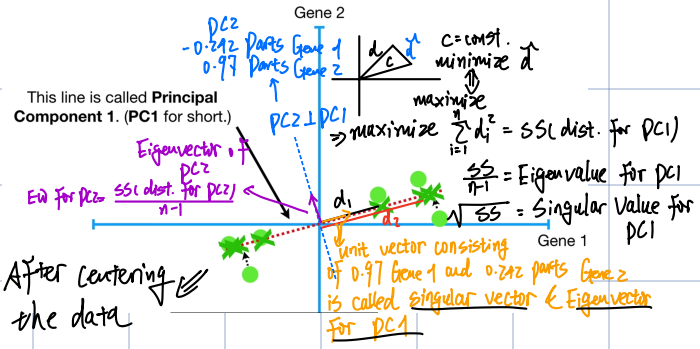
Manhattan: $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$

Selecting k: Cross-Validation, Elbow Method, $k \in 2\mathbb{N}_0 + 1$.

② **Dimensionality reduction**: reduce high-dim Data, lowering computational costs with minimal loss of Info.

I) PCA (Principal component analysis) (linear method)

⇐ EV of the data's covariance matrix



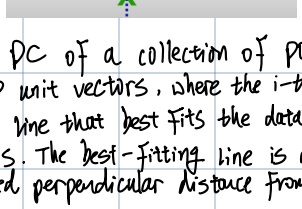
$\frac{SS(\text{dist. for PC1})}{n-1}$ = Variation for PC1 = Var PC1

$\frac{SS(\text{dist. for PC2})}{n-1}$ = Variation for PC2 = Var PC2

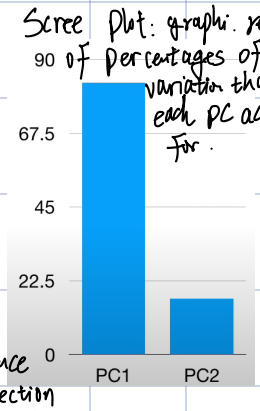
PC1 accounts $\frac{\text{Var PC1}}{\text{Var PC1} + \text{Var PC2}} = 83\%$ of Variation

PC2 accounts 17%

rotate ⇒



⇒

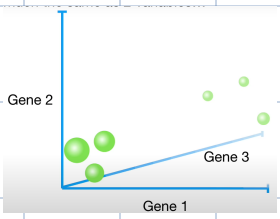


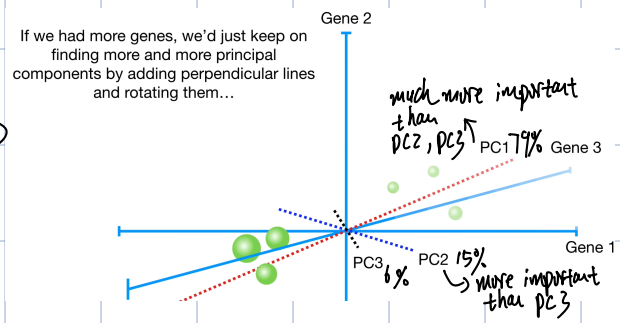
The PC of a collection of points are a sequence of p unit vectors, where the i-th vector is the direction of a line that best fits the data while being ⊥ to the first i-1 vectors. The best-fitting line is defined as one that minimizes the average squared perpendicular distance from the points to the line

More Variables: i) center data ii) best Fitting through origin: PC1 iii) draw PC2, PC3 ⊥ to each other iv) compute Variation of each PC. as scree plot

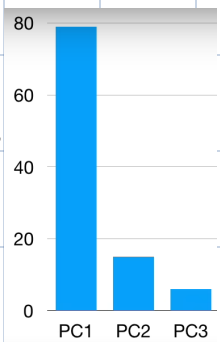
	Mouse 1	Mouse 2	Mouse 3	Mouse 4	Mouse 5	Mouse 6
Gene 1	10	11	8	3	2	1
Gene 2	6	4	5	3	2.8	1
Gene 3	12	9	10	2.5	1.3	2

⇒



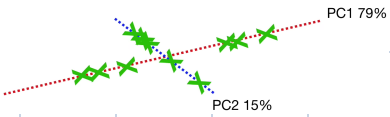


⇒

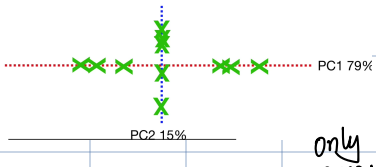


only with PC1 PC2 account for 94% of the variation of data. (Approx. with 2D, PC1 and PC2)

v) proj. into PC1-PC2-plane. =>



rotate. =>



High-Dim. Data reduced into 2-dim. PC1 x PC2 Diagram.

only works well if the first 2 PC account for most of the variation in data

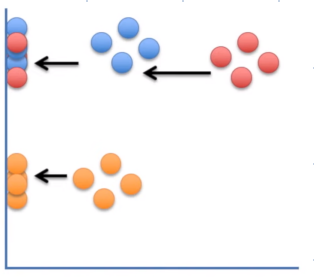
II t-SNE (t-distributed stochastic neighbor embedding) (better als PCA) capture local structure

PCA works better on linear data (not so good if scree plot uniform distributed)

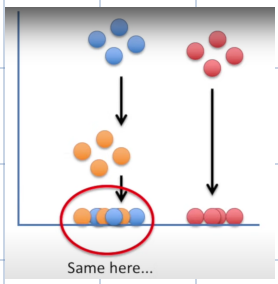
⇒ map high-Dim data points into a lower-dim. space, in a way that preserves the local relationships between points.

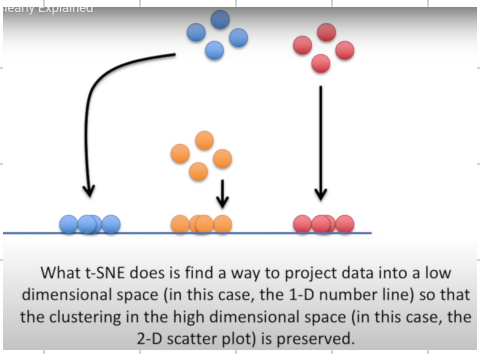
(A) ⇝ measure similarity between data points in the high-dim. space and representing this similarity as possibilities

Then (B) constructs a similar probability distribution in lower-dim. space and minimizes the difference between the two distributions using a technique called gradient descent



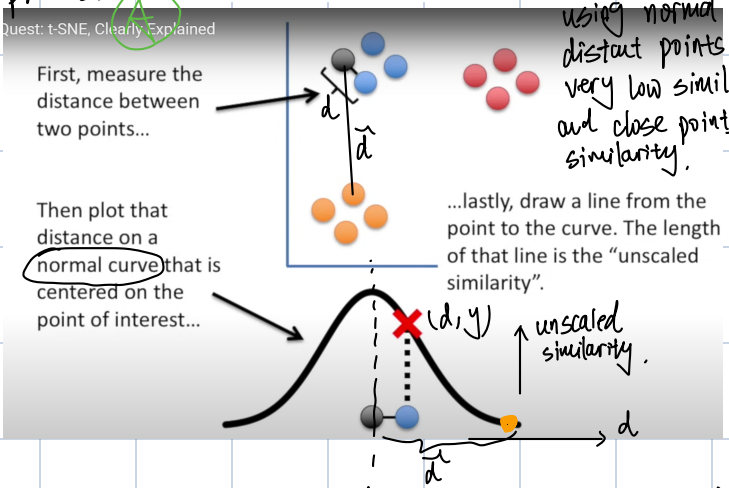
NOTE: If we just projected the data onto one of the axes, we'd just get a big mess that doesn't preserve the original clustering.





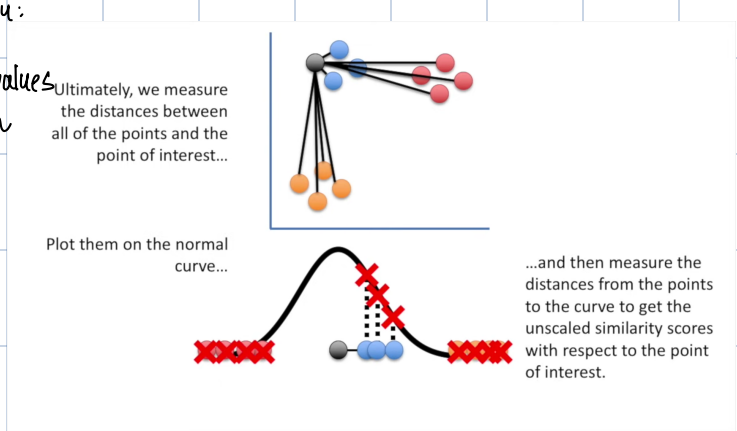
What t-SNE does is find a way to project data into a low dimensional space (in this case, the 1-D number line) so that the clustering in the high dimensional space (in this case, the 2-D scatter plot) is preserved.

Prozess: (A)

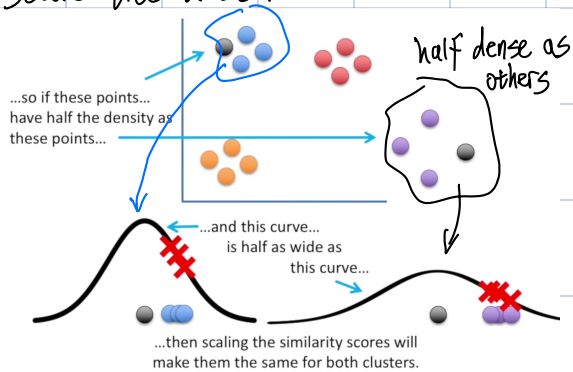


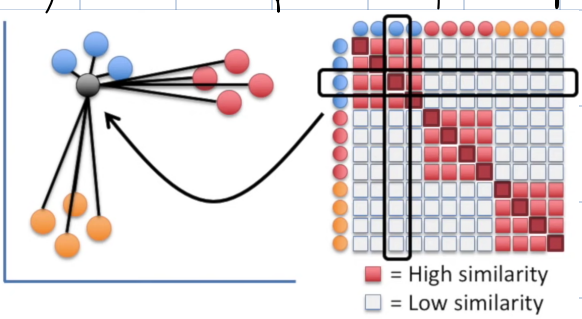
using normal distribu.: distant points have very low similarity values and close points high similarity.

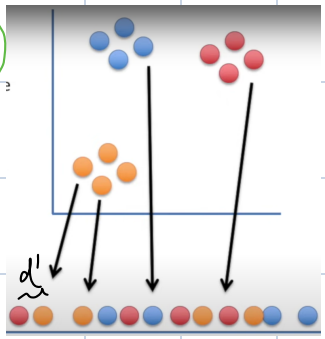
⇒



⇒ scale the unscaled similarities so that they add to 1 (density cluster, same scaled similarity scores) ⇒ do this procedure for all points of interest:

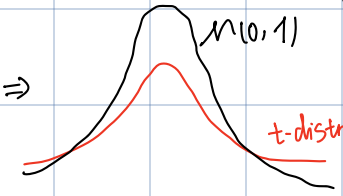




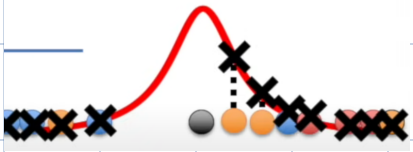


randomly project the data onto the number line, and calculate similarity scores for the points on the number line

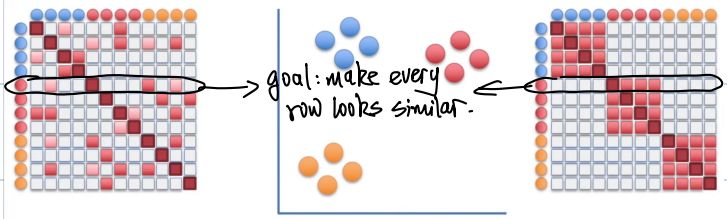
⇒ drawing a line from the point to a curve, but using t-distribution



⇒ So, using a t-distribution, we calculate "unscaled" similarity scores for all the points and then scale them like before.



⇒ scaling ⇒ do for all points, end up with matrix of similarity scores on number line.



t-SNE ⇒

t-SNE moves the points a little bit at a time, and each step it chooses a direction that makes the matrix on the left more like the matrix on the right.

uniform Manifold Approxi. and Projection

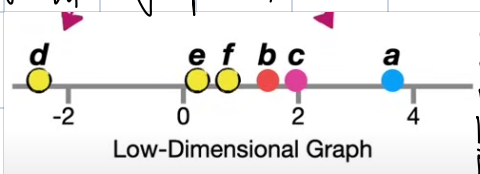
III UMAP (t-SNE's performance suffers with large datasets and using it correctly can be challenging.

increased speed. + better preservation of data's global structure

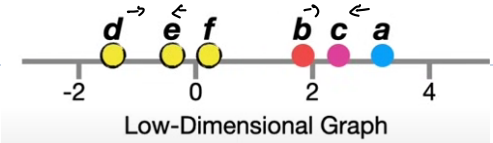
but both algorithms performs strong local structure and group similar categories together; UMAP separates much more clearly the groups of similar categories from each other.

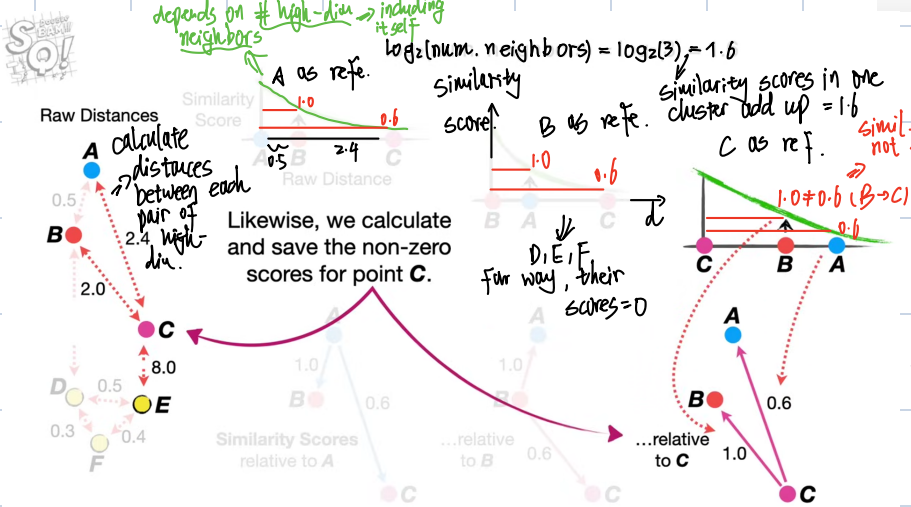
Process: A, B, C seem to form a cluster.; same as D, E, F; Goal of UMAP is to create a low-dim. graph of this data that preserves these high-dim clusters and their relationship to each other

⇒



initialize low-dim points, and move these points around until they form clusters that ⇒ have the same clusters as in the high-dim. data.



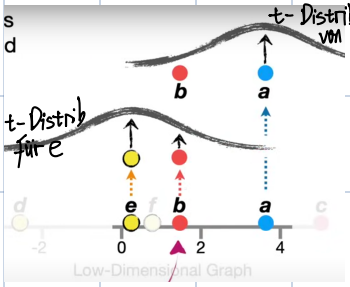


randomly select a pair of points in a cluster proporti. to their high-dim score. (sym.) (Ang. selected a, b)

flips a coin and decides to move b to a or a to b. (Ang. b moves to a)

b should also moves from one of other points from other clusters

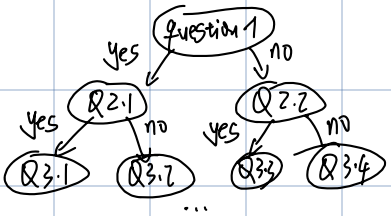
⇒ how far should b move? wants move away from to minimize this low-dimensional score; closer to a in order to maximize the low-dim score von a-t-Distri. from b.



⇒ ... until a,b,c cluster and d,e,f cluster

(Avg. b moves away from e) d,e,f have same possibility to be moved away from (indep. of scores.)

Differences t-SNE & UMAP: A) t-SNE starts with random initialization of low-dim graph; UMAP uses Spectral Embedding and starts always with the exact same low-dim graph. ② t-SNE moves every single point a little bit by each iteration; UMAP can move just one point, or a small subset of points, each time and this helps it scale well with super big datasets



③ Advanced Machine Learning.

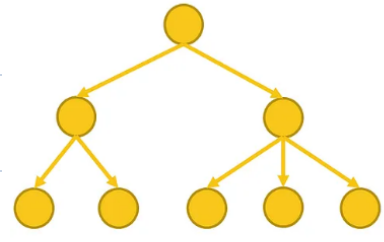
- Decision Trees: make decisions (yes or no) by asking a series of questions

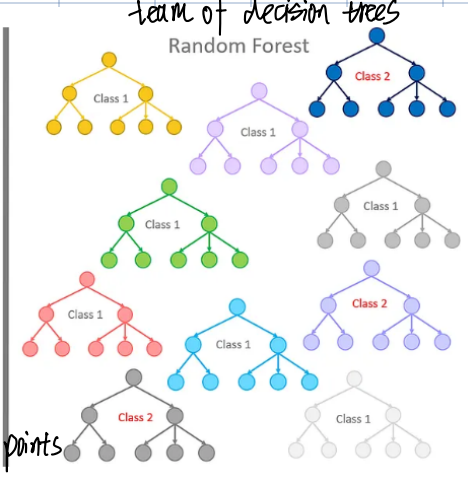
~> simple to understand, versality (can handle both categorical and numerical data), no need for data preprocessing, not affected by outliers or missing values; Feature selection (using first the most informative features, can be particularly beneficial when dealing large number datasets.

~> Overfitting (can create a tree perfectly classif. training data but performs poorly on test data); Bias towards features with more levels.; Instability (sensitive to small changes in the data, slight variation leads to drastically different trees); Difficulty in capturing complex relationships (only yes, no).

- Random Forest.

In a Random Forest, instead of just one decision tree making all the decisions, we create an entire "forest" of decision trees. Each tree gives its "opinion" or prediction based on the data it has seen. The final output is then determined by considering the output of all the trees in the forest.





= opinion gathering" more exciting by given each tree a slightly differ. set of data points and features to learn from. (bagging and feature randomness)

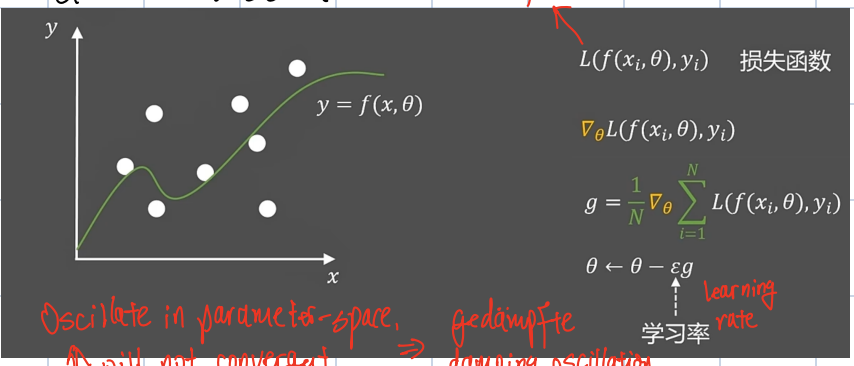
~> less possible overfitting than single decision Trees.; handles large Data Set and Feature Spaces

Training of individual trees can be done in parallel; can provide insights into which features are most import.

in making predictions.

⇝ Complexity ↑ Less Interpretable than single decisi. Tree ↑ longer prediction time

- Gradient Descent.



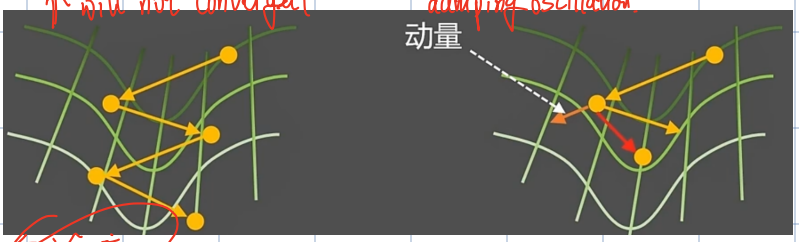
loss function minimieren

⇒ random gradient descent: every $m \leq N$ different data points

$g = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$

(Speicheraufwand ↓) (if std. derivation small, although noise but proper convergent Wahrscheinlichkeit).

Oscillate in parameter-space, with not convergent ⇒ gedämpfte damping oscillation



$g = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$

$v \leftarrow \alpha v - \varepsilon g$ α控制动量

$\theta \leftarrow \theta + v$

control of history descent. "decay rate"

= Stochastic Gradient Descent

AdaGrad (2011)

$g = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$

$r \leftarrow r + g^2$ → value

$\theta \leftarrow \theta - \frac{\varepsilon}{\sqrt{r} + \delta} g$

δ是一个小量，稳定数值计算

careful adjust learning rate to avoid loss function oscillate.

RMSProp (2012)

$g = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$

quickly find convergent direction

$r \leftarrow \rho r + (1 - \rho) g^2$

to avoid learning rate decreases too fast. (manuell control)

$\theta \leftarrow \theta - \frac{\varepsilon}{\sqrt{r} + \delta} g$

Wenn Gradient g Fluctuates, learning rate decreases fast (at beginning); after a while need to g small

δ是一个小量，稳定数值计算

Adam (2014) (impuls + self-adjusting)

$g = \frac{1}{m} \nabla_\theta \sum_{i=1}^{m} L(f(x_i, \theta), y_i)$

$s \leftarrow \rho_1 s + (1 - \rho_1) g$ control impuls with ρ_1 ------ 自适应动量

$r \leftarrow \rho_2 r + (1 - \rho_2) g^2$ control selbst-adj. with ρ_2.

$\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$

$\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$

$\theta \leftarrow \theta - \frac{\varepsilon \hat{s}}{\sqrt{\hat{r}} + \delta} g$

- Gradient Boosting Trees

Learning Rate

Boosted Ensemble = First Tree + η * Second Tree Loss(Boosted Ensemble) < Loss(First Tree)

⇒ $F(2) = F(1) + \eta * \text{second-tree}$, $\text{second tree} = -\frac{\partial L}{\partial F(1)} = -\frac{\partial \text{Loss Function}}{\partial \text{Previous Model's Output}}$.

⇒ $F(m) = F(m-1) - \eta * \frac{\partial L}{\partial F(m-1)}$ (Overfitting) ⇒ Decision Trees are not independ. anymore!

- Neural Networks.

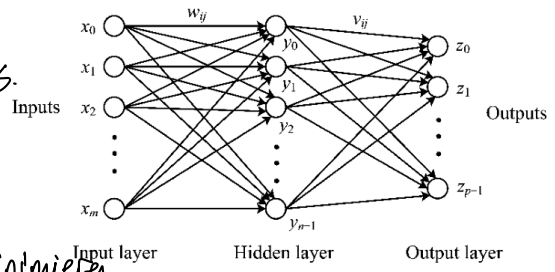
① MLP (Multi-Layer-Perceptron)

composition of a linear mapping followed by a non-linear function

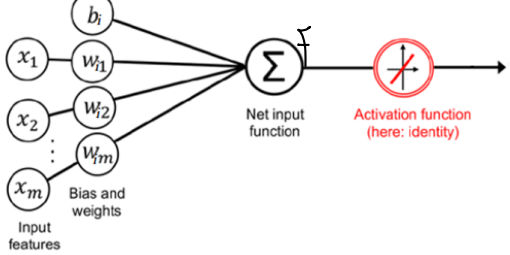
$f(x) = \sigma(Wx + b) = \sigma\left(\sum_{i=1}^{n_{out}} \langle w_i, x \rangle + b_i\right)$ $x \in \mathbb{R}^n = \mathbb{R}^{n_{in}}$ $W \in \mathbb{R}^{n_{out} \times n_{in}}$ $b \in \mathbb{R}^{n_{out}}$ ⇒ single Block of MLP

W: weight, b: bias, σ: activition function

⇒ MLP = $F_L \circ \dots \circ F_1$ each F_i repres. a layer of MLP



(a) Multilayer perceptron (MLP)



(b) Neuron of an MLP

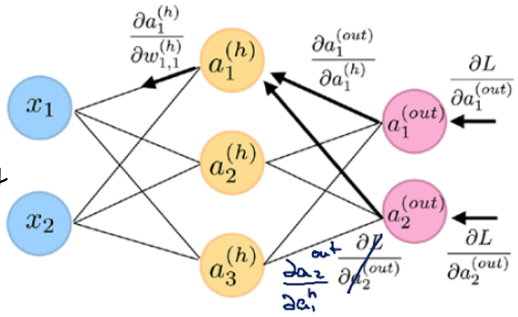
⇒ determine W, b: Loss Function minimieren

⇒ Gradient Descent. $W \leftarrow W - \eta \frac{\partial L(W,b)}{\partial W}$

$b \leftarrow b - \eta \frac{\partial L(W,b)}{\partial b}$

⇒ determine $\frac{\partial L}{\partial W}, \frac{\partial L}{\partial b}$? → backpropagating using chain-rules of derivatives

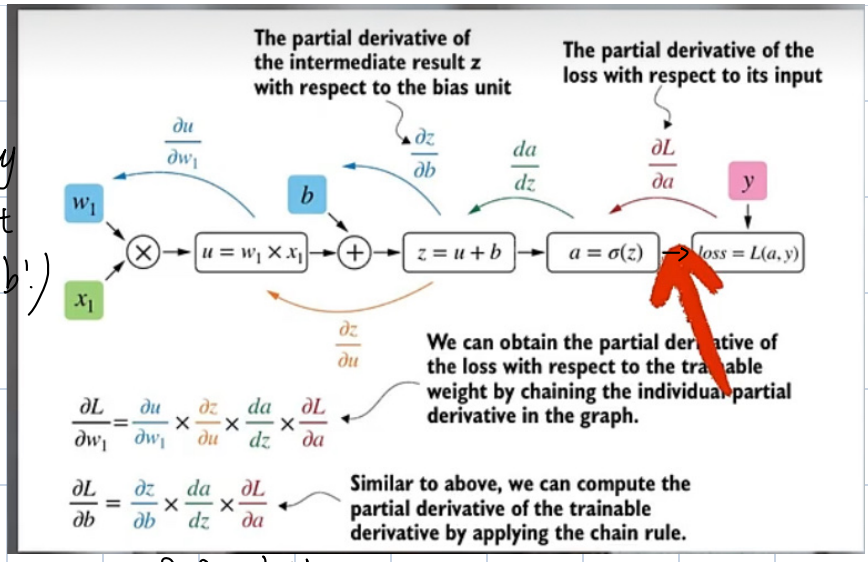
⇒ save the time-space - complexity. (avoid repeated calcul.)



Gradient for hidden layer weight:

$$\frac{\partial L}{\partial w_{1,1}^{(h)}} = \frac{\partial L}{\partial a_1^{(out)}} \cdot \frac{\partial a_1^{(out)}}{\partial a_1^{(h)}} \cdot \frac{\partial a_1^{(h)}}{\partial w_{1,1}^{(h)}} + \frac{\partial L}{\partial a_2^{(out)}} \cdot \frac{\partial a_2^{(out)}}{\partial a_1^{(h)}} \cdot \frac{\partial a_1^{(h)}}{\partial w_{1,1}^{(h)}}$$

⇒ activition function don't use $\mathbb{X}$ (discontinuity by 0, otherwise derivative = 0, by backpropagating at some point $(\,)' = 0$ ⇒ no update for parameter W, b.)



≤ MLPs

② CNN (Convolutional Neural Network)

2D-convolution $[W * X]_{ij} = \sum_{u=0}^{H-1} \sum_{v=0}^{W-1} W_{uv} X_{i+u,j+v}$ $\in \mathbb{R}$.

Input	Kernel	Output (feature map)
0 1 2 3 4 5 6 7 8	0 1 2 3	19 25 37 43

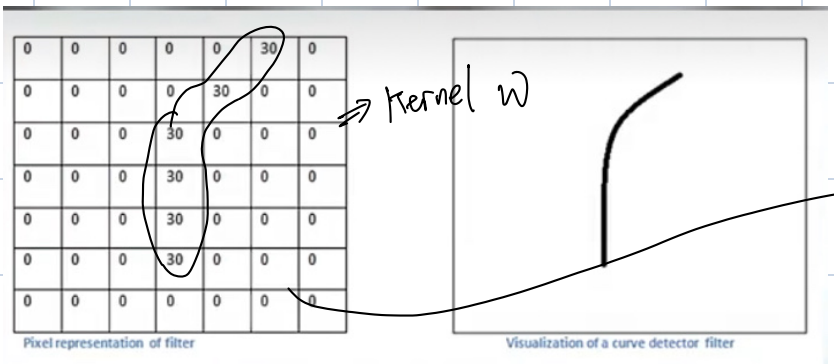
1D Convolution

Input	Kernel	Output
0 1 2 3 4 5 6	1 2	2 5 8 11 14 17

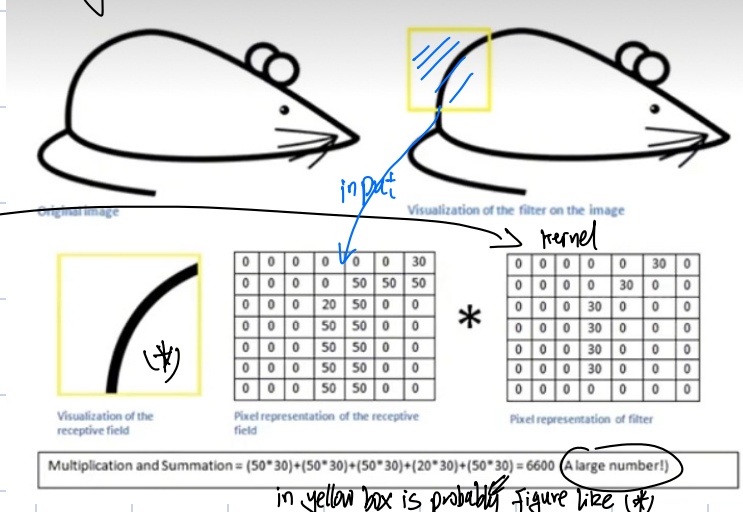
convolutional operation measures similarity between each patch of input data x and the filter W.

value high: similar figure.

Each kernel represents a characteristic feature in image.:



⇒ Kernel W



in yellow box is probably figure like (*)

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0 ⇒ this local structure is not same as (*)