Machine Intelligence: Deep Learning
Week 7

RNN continued

&

unsupervised learning

Beate Sick, Oliver Dürr, Elvis Murina

Institut für Datenanalyse und Prozessdesign

Zürcher Hochschule für Angewandte Wissenschaften

beate.sick@zhaw.ch

Topics

- **Special RNN architectures: LSTM, GRU…**
- **2D visualization of high dimensional data**
  - PCA (recap)
  - t-SNE
- **Unsupervised feature construction**
  - principle components
  - features from pre-trained CNNs
  - autoencoder (AE)
- **Train an image classifier with only few labeled data**
  - quality of features as important success factor

Stateful RNN model

Training a stateful RNNs

- RNN are often trained on sequence data with inherent order

- Sequences are often very long and need to be cut between mini-batches

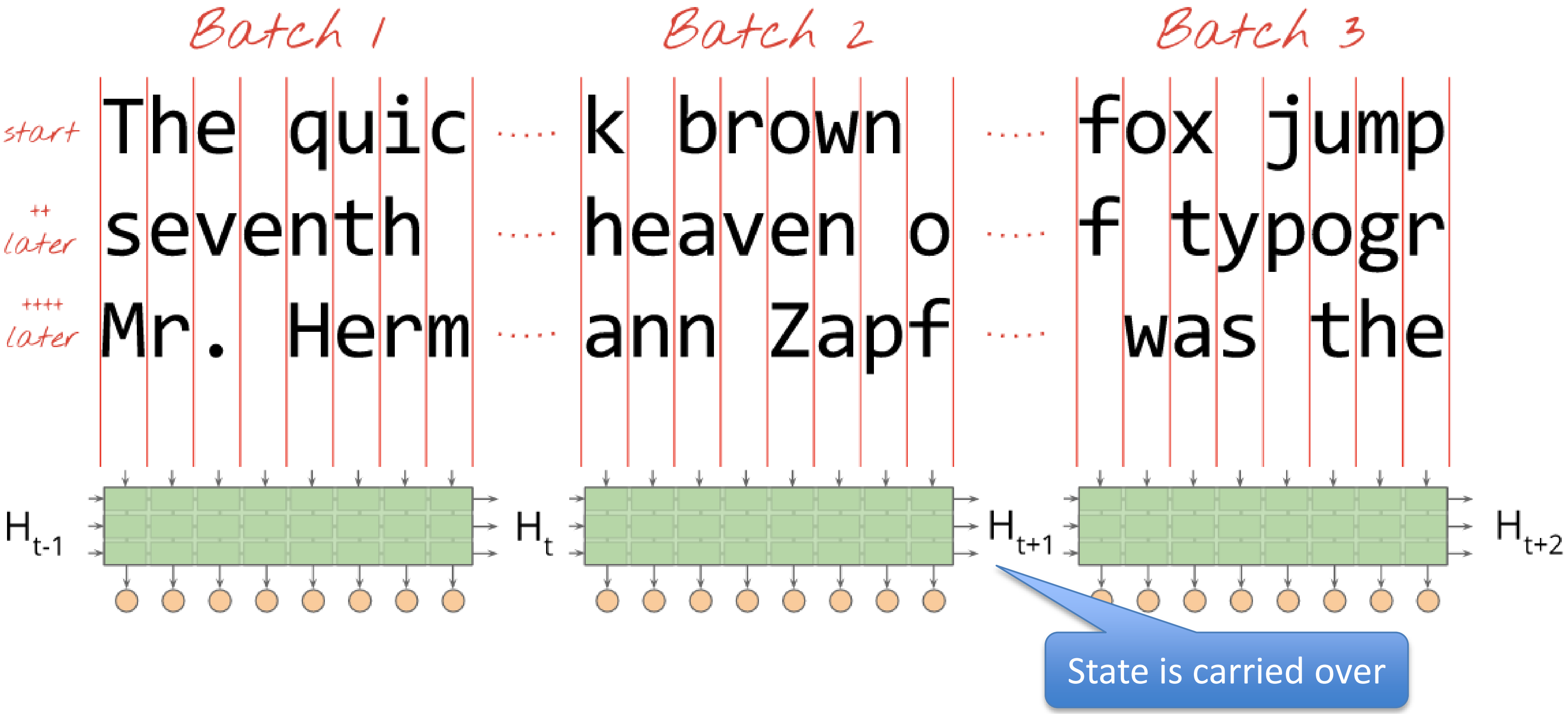
- By default the hidden state is initialized with zeros in each mini-batch

- In stateful RNN we connect sequences in the right order between mini-batches allowing to make use of the hidden state learned so far

- This requires a careful construction of the mini-batches and an appropriate transfer of the hidden state between mini-batches

Mini-batches in statefull RNN

The gradient is propagated back a fixed amount of steps defined by the size of a mini-batch. In stateful RNNs the hidden state is carried over between mini-batches and hence between connecting sequences given appropriate batches.


Batch 1
Batch 2
Batch 3
start
++
later
++++
later
The quic ..... k brown ..... fox jump
seventh ..... heaven o ..... f typogr
Mr. Herm ..... ann Zapf ..... was the
H_{t-1}
H_t
H_{t+1}
H_{t+2}
State is carried over

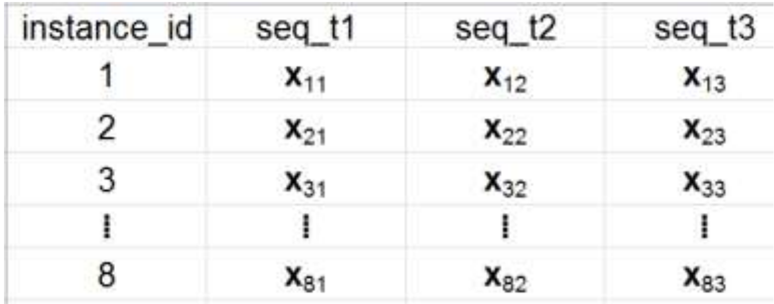
Slide credit Martin Gorner (TensorFlow without PhD)

Vanishing/Exploding Gradient problem during training a RNN

Recall: Loss of a mini-batch is used to determine update

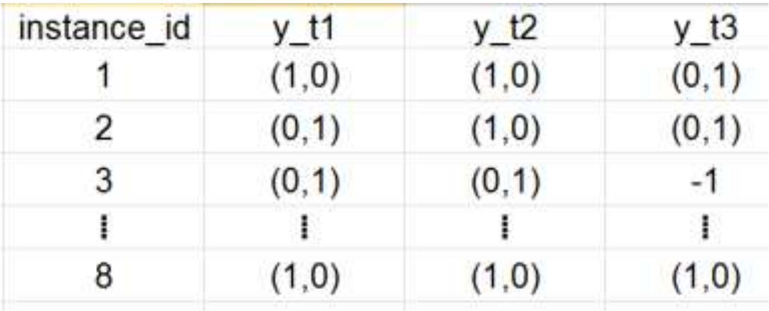
mini-batch of size M=8

train data input (S=len(seq)=3):



instance_id	seq_t1	seq_t2	seq_t3
1	x_{11}	x_{12}	x_{13}
2	x_{21}	x_{22}	x_{23}
3	x_{31}	x_{32}	x_{33}
⋮	⋮	⋮	⋮
8	x_{81}	x_{82}	x_{83}

train data target (2 classes, K=2):



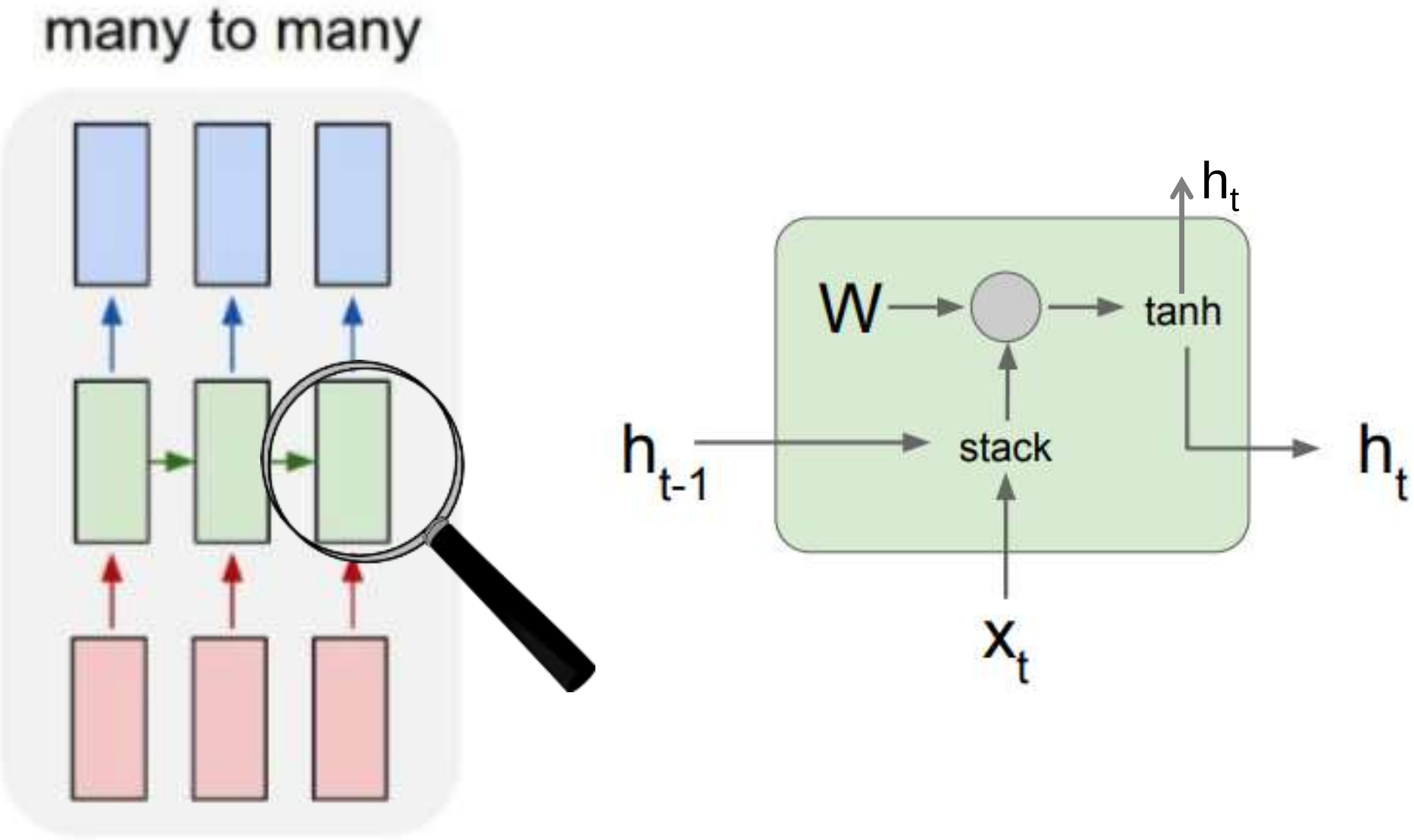
instance_id	y_t1	y_t2	y_t3
1	(1,0)	(1,0)	(0,1)
2	(0,1)	(1,0)	(0,1)
3	(0,1)	(0,1)	-1
⋮	⋮	⋮	⋮
8	(1,0)	(1,0)	(1,0)

Cost C or Loss is given by the cross-entropy averaged over all instances in mini-batch:

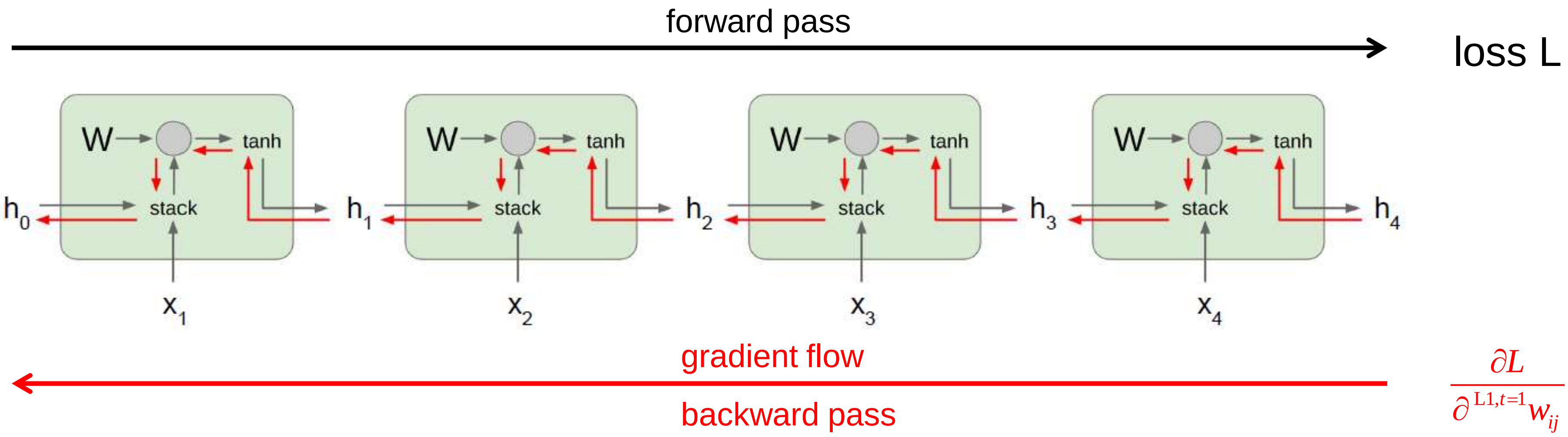
$$\text{Loss} = \frac{1}{8}\sum_{m=1}^{8}\left[\sum_{s=1}^{3}\left(-\sum_{k=1}^{2} y_{msk}\cdot\log\left(p_{msk}\right)\right)\right]$$

Based on the mini-batch loss the weights in the tow weight matrices of layer 1 and layer 2 are updated.

Recall: Design of a RNN "cell"



Backpropagation in RNNs: Gradient is multiplied at each time step with same factor: Gradient explosion/vanishing



Propagating the gradient of the cost function via chain rule to the first time point involves multiplying at each time step with $\mathbf{W}^T$ (and the derivation of tanh).

⟹ Vanishing gradient if we multiply at each time step with a number <1
(more precisely we have only a number if W is a scalar, otherwise we need to look on the first singular value of $\mathbf{W}^T$)

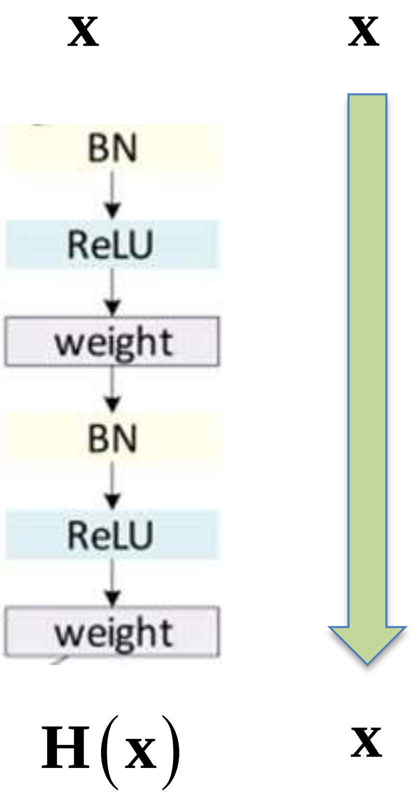
⟹ Exploding gradient if we multiply at each time step with a number >1
(more precisely we have only a number if W is a scalar, otherwise we need to look on the first singular value of $\mathbf{W}^T$)

Solution: gradient clipping (hack), or use better architecture like LSTM or GRU!

slide (adapted) from cs231 lecture 10

GRU and LSTM cells to avoid vanishing/exploding gradients

Recall: Highway Networks



Idea: Use nonlinear transform T to determine how much of the output **y** is produced by H or the identity mapping. Technically we do that by:

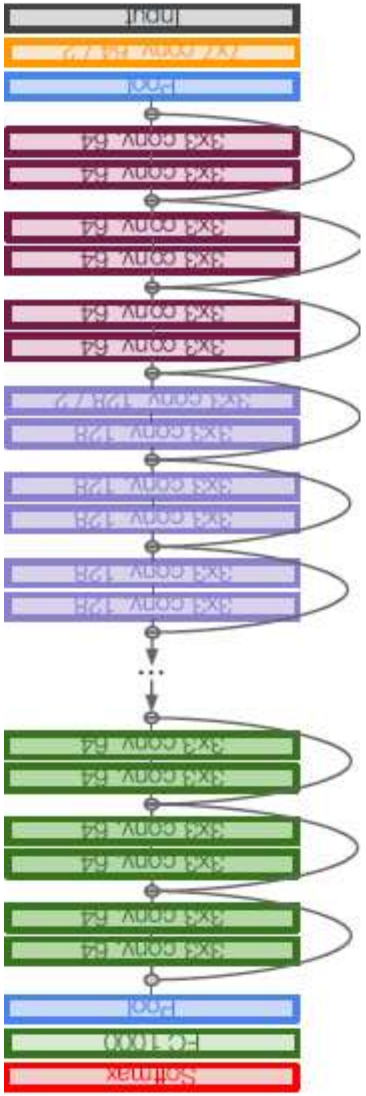
$$\mathbf{y} = H(\mathbf{x}, \mathbf{W_H}) \cdot T(\mathbf{x}, \mathbf{W_T}) + \mathbf{x} \cdot (1 - T(\mathbf{x}, \mathbf{W_T})).$$

Special case:

$$\mathbf{y} = \begin{cases} \mathbf{x}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0} \\ H(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1} \end{cases}$$

This opens a highway for the gradient:

$$\frac{d\mathbf{y}}{d\mathbf{x}} = \begin{cases} \mathbf{I}, & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{0}, \\ H'(\mathbf{x}, \mathbf{W_H}), & \text{if } T(\mathbf{x}, \mathbf{W_T}) = \mathbf{1}. \end{cases}$$

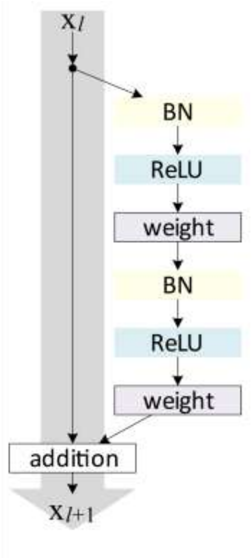


Srivastava, Greff, Schmidhuber 2014, “Highway Networks” https://arxiv.org/pdf/1505.00387.pdf

Recall: ResNet

- use ResNet like architectures allowing for a gradient highway

(in CNN also batch-normalization and ReLU helped to train deep NN, but cannot naively transfered to recurrent NN)



$x_{l+1} = x_l + F(x_l)$

ResNet basic design (VGG-style)

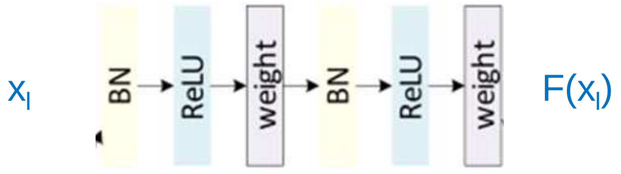
- add shortcut connections every two
- all 3x3 conv (almost)

152 layers:
Why does this train at all?

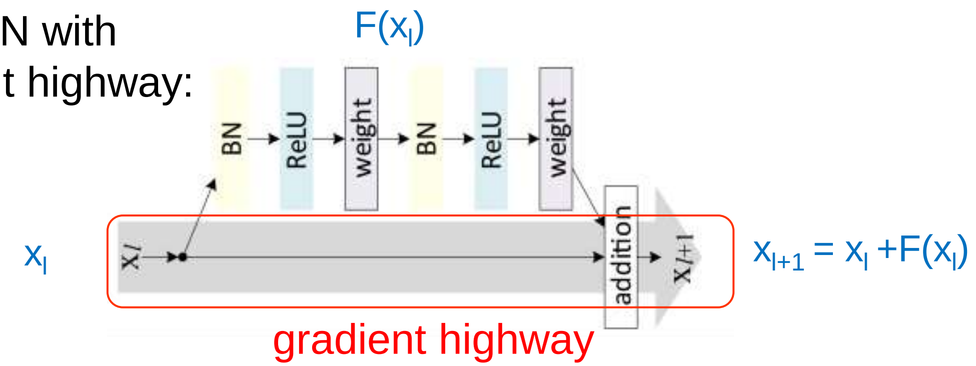
This deep architecture could still be trained, since the gradients can skip layers which diminish the gradient!

Provide gradient highway also in recurrent NN: GRU, LSTM

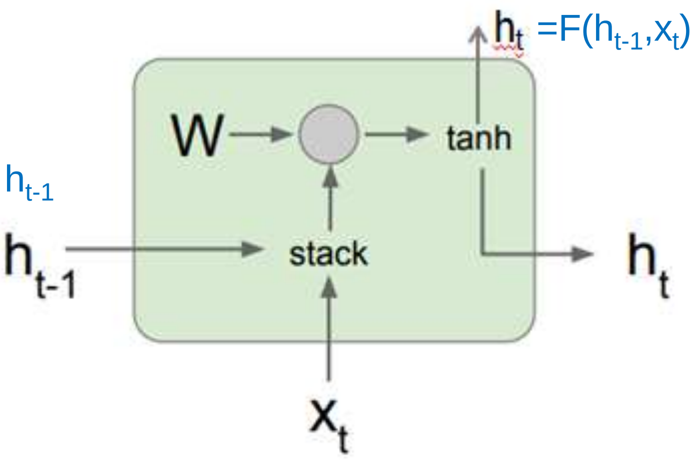
CNN classic:



CNN with gradient highway:

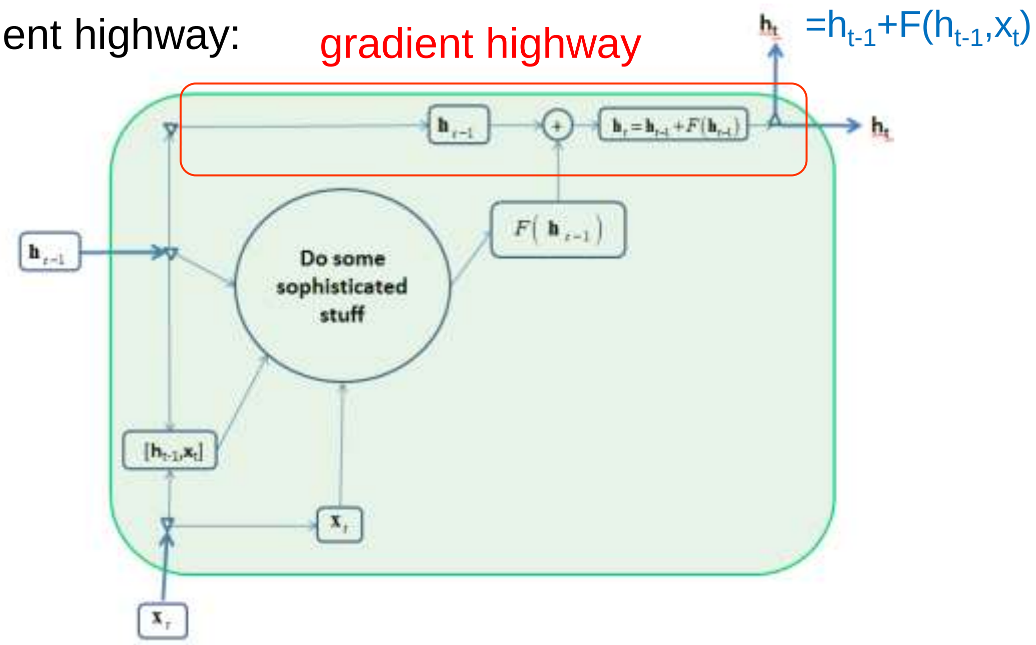


RNN classic:

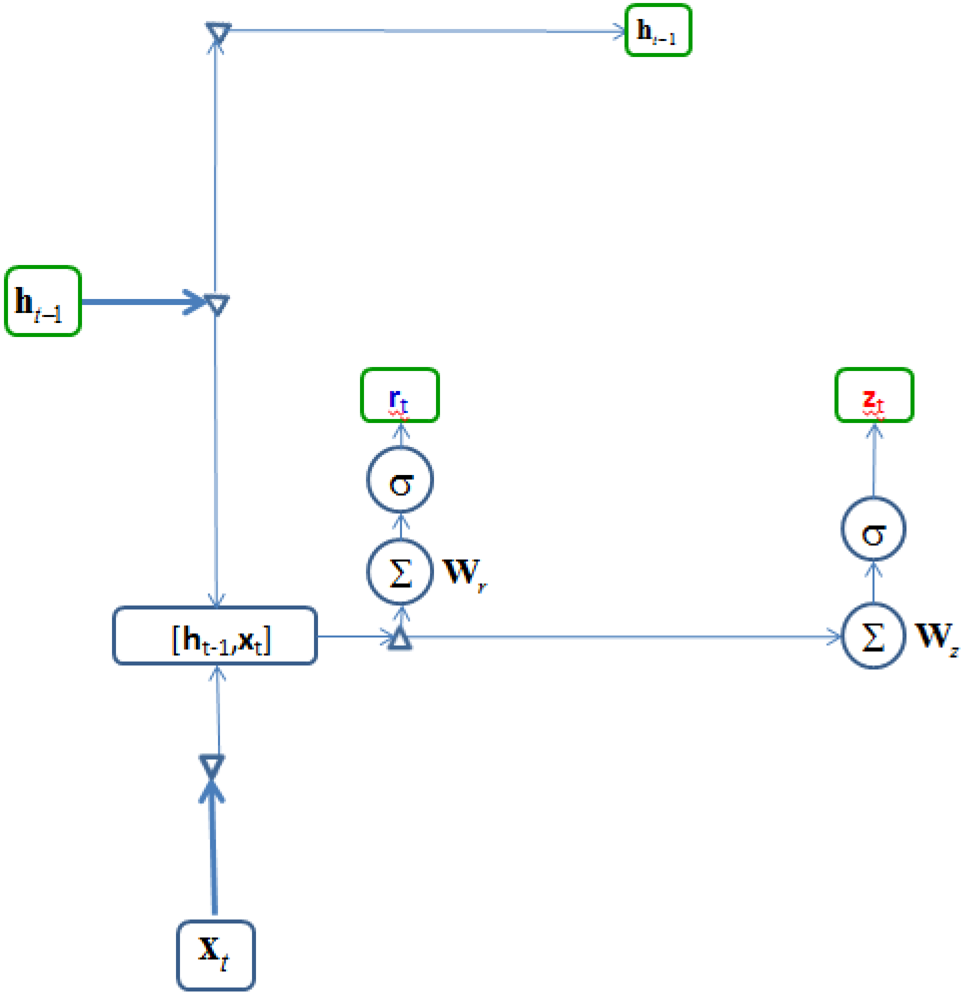


$$\mathbf{h}_t = \tanh\left([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W} + \mathbf{b}\right)$$

RNN with gradient highway:


gradient highway
$h_t = h_{t-1} + F(h_{t-1}, x_t)$
$\mathbf{h}_{t-1}$
+
$\mathbf{h}_t = \mathbf{h}_{t-1} + F(\mathbf{h}_{t-1})$
$\mathbf{h}_t$
$\mathbf{h}_{t-1}$
Do some sophisticated stuff
$F(\mathbf{h}_{t-1})$
$[\mathbf{h}_{t-1}, \mathbf{x}_t]$
$\mathbf{x}_t$
$\mathbf{x}_t$

Towards Gated Recurrent Units (GRU)



$$\mathbf{r}_t = \mathrm{gate}_{r,t} = \mathrm{sigmoid}\left([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W}_r + \mathbf{b}_r\right)$$

$$\mathbf{z}_t = \mathrm{gate}_{\mathrm{update}} = \mathrm{sigmoid}\left([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W}_z + \mathbf{b}_z\right)$$

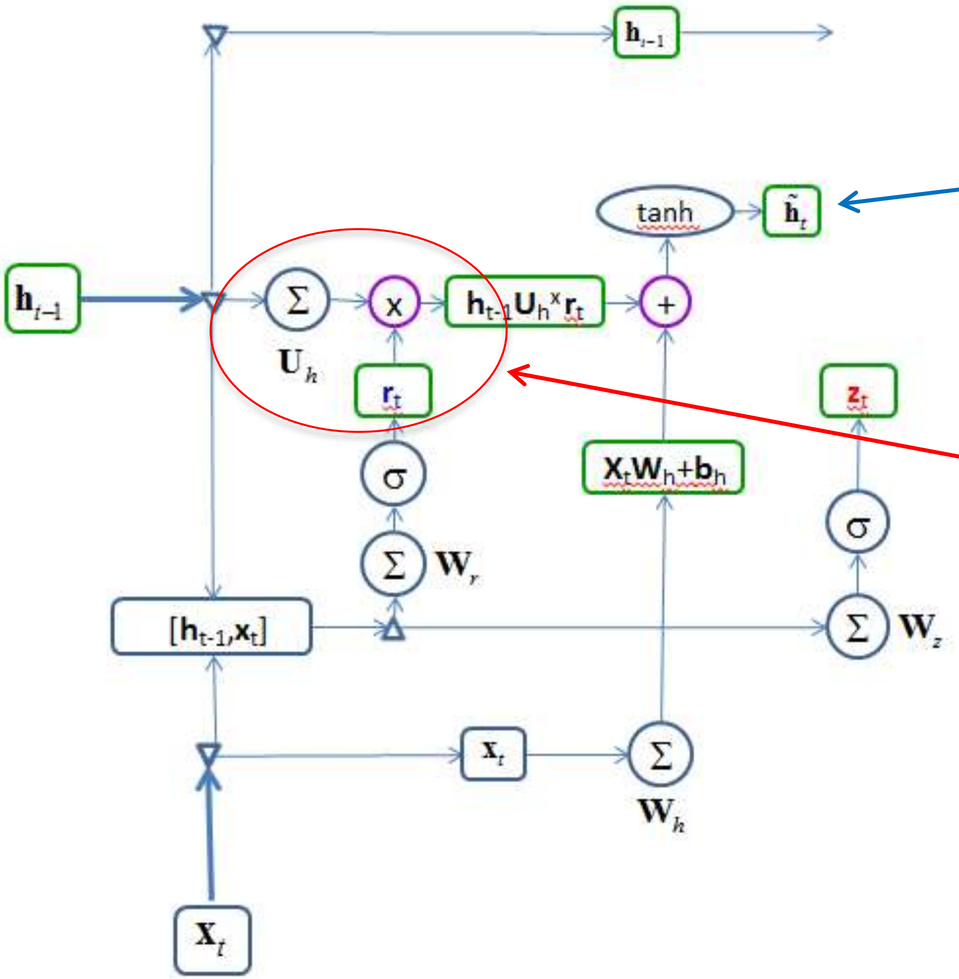
Idea:

The **relevant gate r** controls which part of the previous hidden state is relevant for making a prediction or should be dropped

The **updated gate z** controls how much information from the previous hidden layer h_{i-1} and the new input should be propagated to the current hidden layer h_i.

Remark: all internal vectors/tensors with green frame have same length/shape.

Towards Gated Recurrent Units (GRU)



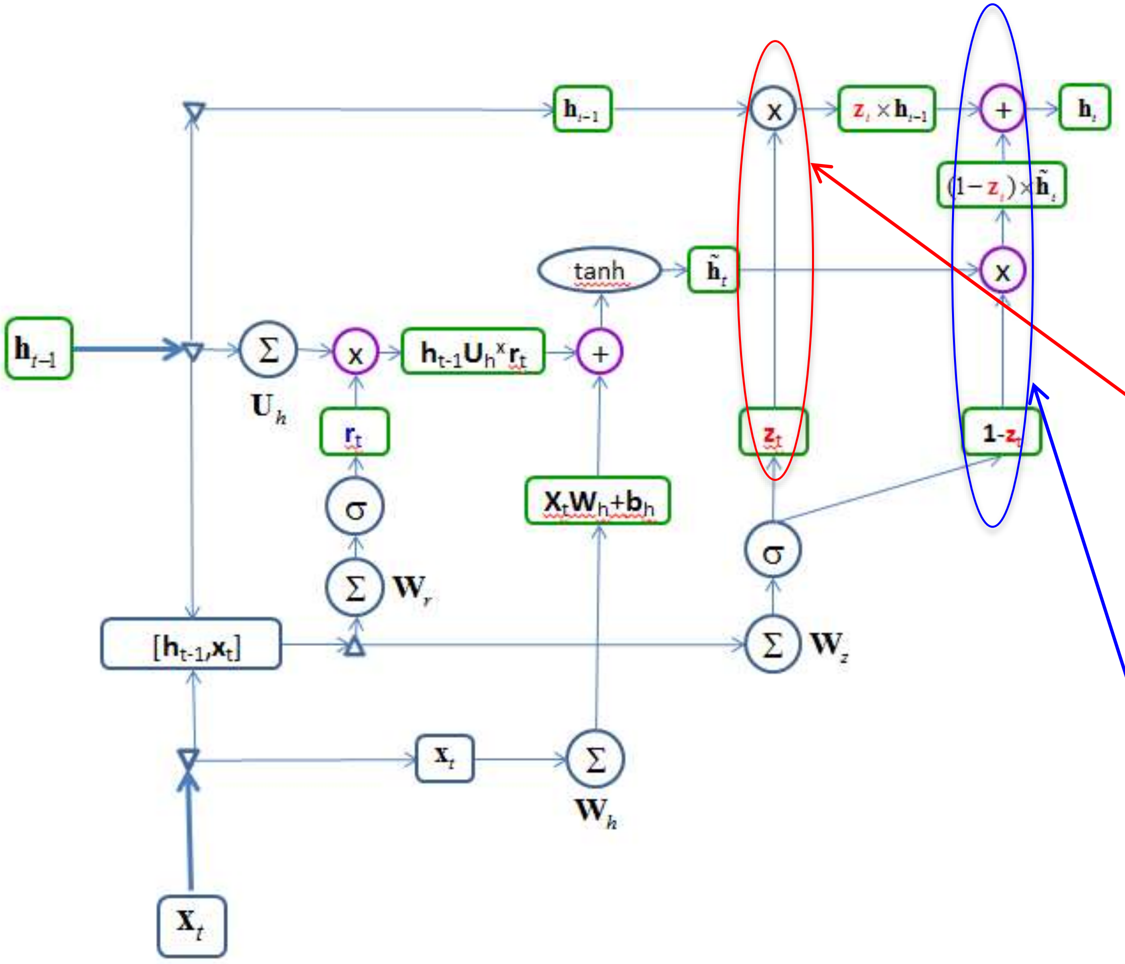
The new proposed state $\tilde{h}$ is:

$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{x}_t \cdot \mathbf{W}_h + \mathbf{b}_h + \underbrace{\mathbf{h}_{t-1}\mathbf{U}_h \otimes \mathbf{r}_t}\right)$$

Here, we use the relevant gate r to control what part of $\boldsymbol{h_{t-1}}$ we need to compute a new proposal.

Remark: all internal vectors with green frame have same length.
All ops within a purple circle are performed per element-wise on the ingoing vectors

The Gated Recurrent Unit (GRU)



The new hidden state is:

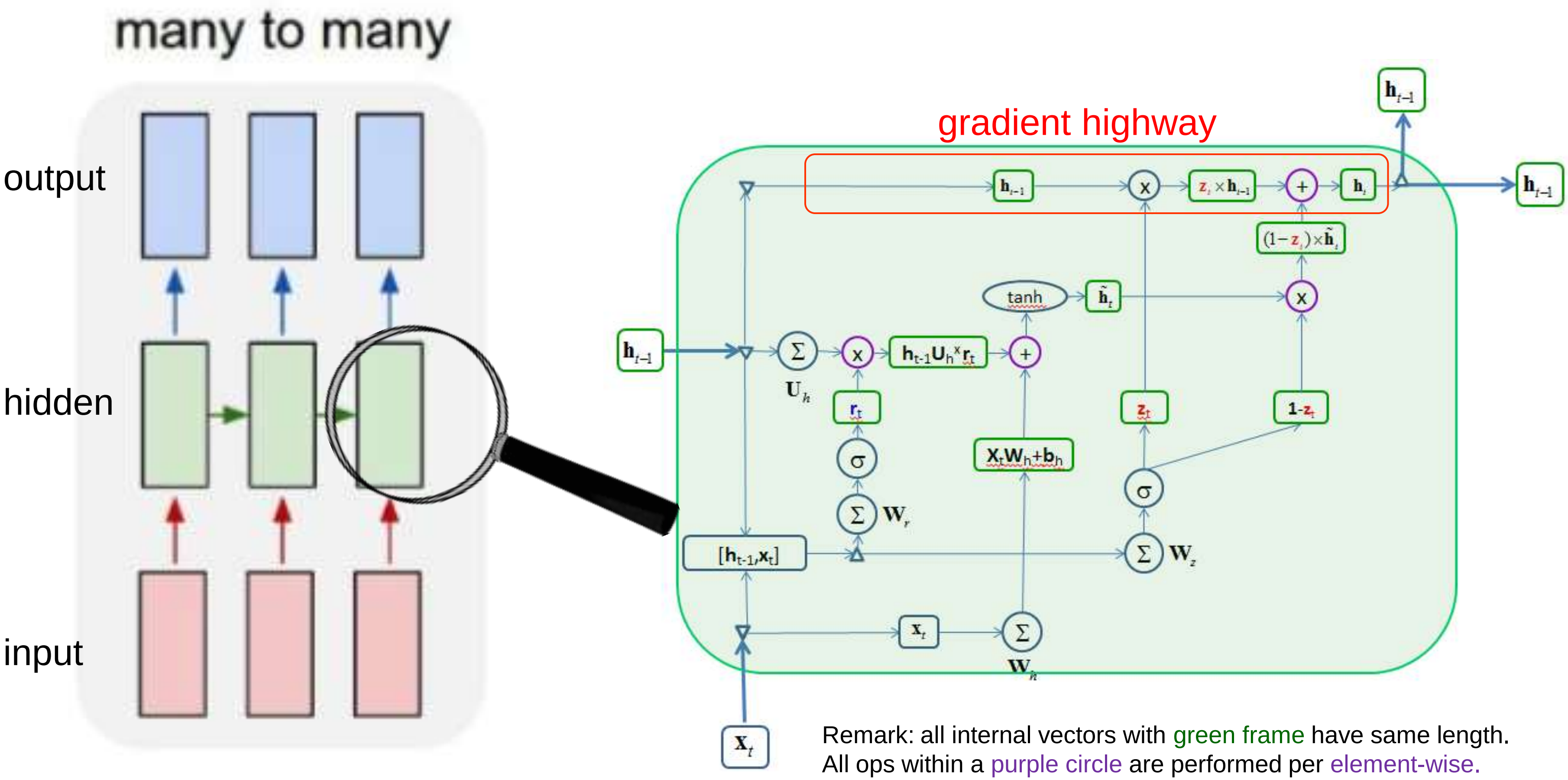
$$h_t = (1\text{-}z_t) \otimes \tilde{h}_t \oplus z_t \otimes h_{t-1}$$

If all elements of $\mathbf{z}_t$ are 1 then the hidden state stays unchanged.

The updated gate $\mathbf{z}_t$ controls how much of the previous hidden state $\mathbf{h}_{t-1}$ and the new input $\mathbf{x}_t$ should be propagated to the current hidden state $\mathbf{h}_t$

The updated gate $\mathbf{z}_t$ controls also how much information from proposed new state $\tilde{h}$ is entering the new state

Solution via “highway allowing” architecture: GRU

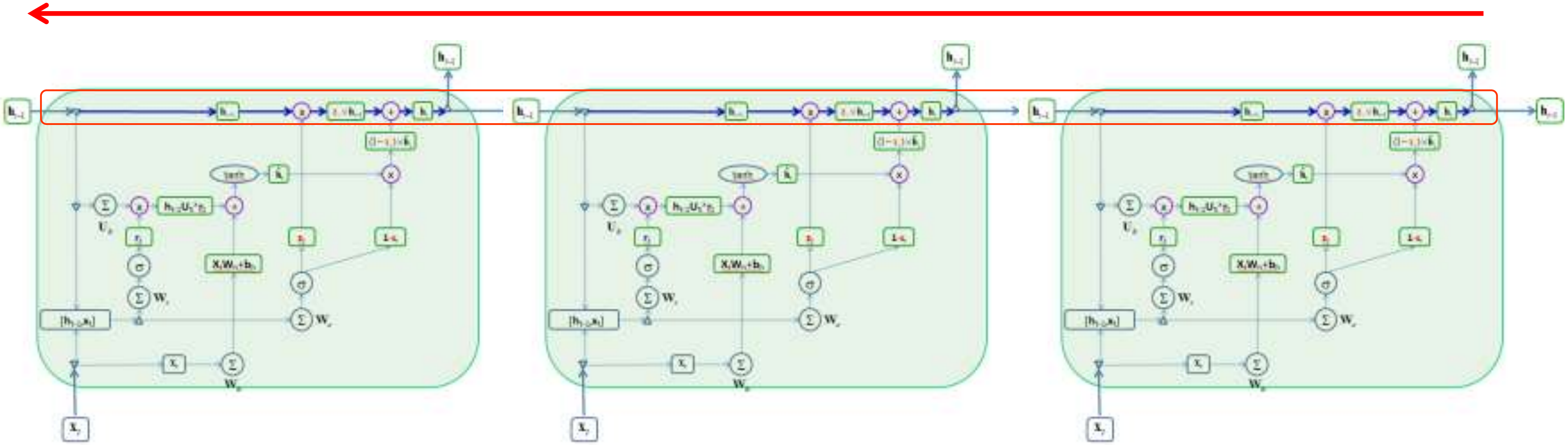


Remark: all internal vectors with green frame have same length.
All ops within a purple circle are performed per element-wise.

The gradient high-way avoids gradient vanishing. The GRU also avoids gradient explosion since the element-wise operations on vector-elements that change over the time steps, avoids multiplying the gradients with the same number in each step.

The Gated Recurrent Unit (GRU): Gradient Flow

Uninterrupted gradient flow!



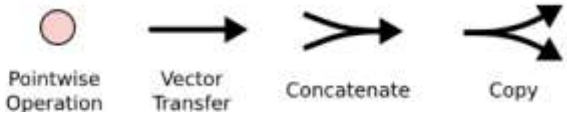
Relevant gate: $\mathbf{r}_t = \sigma\left([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W}_r + \mathbf{b}_r\right)$

Update gate: $\mathbf{z}_t = \sigma\left([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W}_z + \mathbf{b}_z\right)$

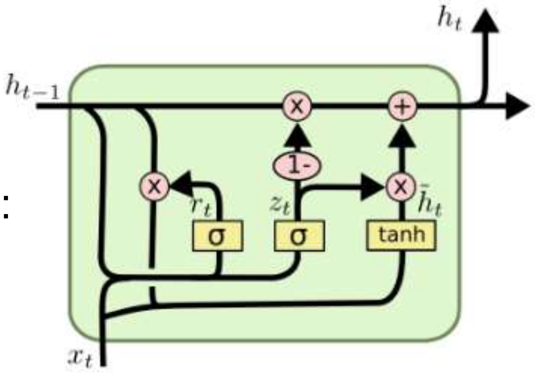
Proposed hidden state: $\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{x}_t \cdot \mathbf{W}_h + \mathbf{b}_h + \mathbf{h}_{t-1}\mathbf{U}_h \otimes \mathbf{r}_t\right)$

New hidden state is: $\mathbf{h}_t = (\mathbf{1}\text{-}\mathbf{z}_{\mathrm{t}}) \otimes \tilde{\mathbf{h}}_t \oplus \mathbf{z}_{\mathrm{t}} \otimes \mathbf{h}_{t-1}$

Long Short Term Memory cell (LSTM) as GRU-extension

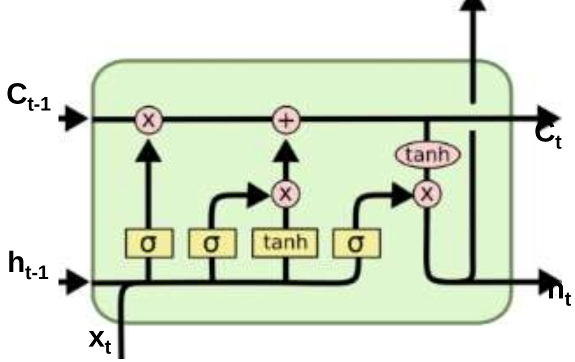


GRU-Cell:



$\mathbf{h}_t$

LSTM-Cell:



2 gates, 1 cell states (h)

Relevant gate: $\mathbf{r}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W}_r + \mathbf{b}_r)$

Update gate: $\mathbf{z}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W}_z + \mathbf{b}_z)$

Proposed hidden state: $\tilde{\mathbf{h}}_t = \tanh(\mathbf{x}_t \cdot \mathbf{W}_h + \mathbf{b}_h + \mathbf{h}_{t-1} \mathbf{U}_h \otimes \mathbf{r}_t)$

New hidden state is: $\mathbf{h}_t = (\mathbf{1\text{-}z}_t) \otimes \tilde{\mathbf{h}}_t \oplus \mathbf{z}_t \otimes \mathbf{h}_{t-1}$

3 gates, 2 cell states (S:h, L:C)

Forget gate: $\mathbf{f}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W}_f + \mathbf{b}_f)$

Input gate: $\mathbf{i}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W}_i + \mathbf{b}_i)$

Output gate: $\mathbf{o}_t = \sigma([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W}_o + \mathbf{b}_o)$

Proposed cell state: $\tilde{\mathbf{C}}_t = \tanh([\mathbf{h}_{t-1}, \mathbf{x}_t] \cdot \mathbf{W}_C + \mathbf{b}_C)$

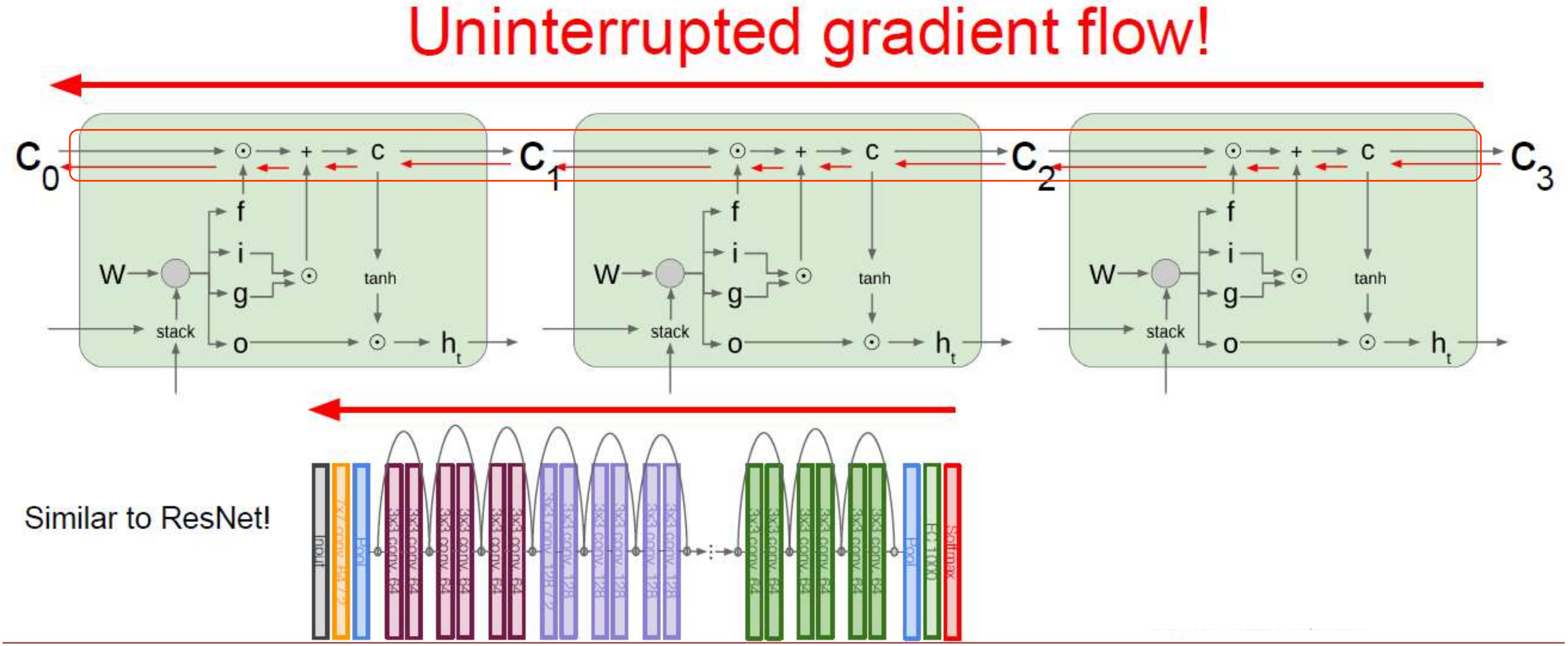
New L cell state: $\mathbf{C}_t = \mathbf{f}_t \otimes \mathbf{C}_{t-1} \oplus \mathbf{i}_t \otimes \tilde{\mathbf{C}}_t$

New S hidden state: $\mathbf{h}_t = \mathbf{o}_t \otimes \tanh(\mathbf{C}_t)$

Credits: Colah blog http://colah.github.io/posts/2015-08-Understanding-LSTMs
Long Short Term Memory networks (LSTM) were introduced by Hochreiter & Schmidhuber (1997).
A simplified variation, the Gated Recurrent Unit, or GRU, introduced by Cho, et al. (2014).

Long Short Term Memory (LSTM): Gradient Flow

LSTM has an additional cell state C for a “long term memory”.

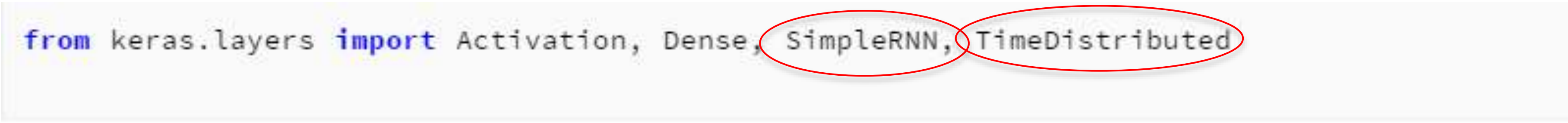

Uninterrupted gradient flow!

Similar to ResNet!

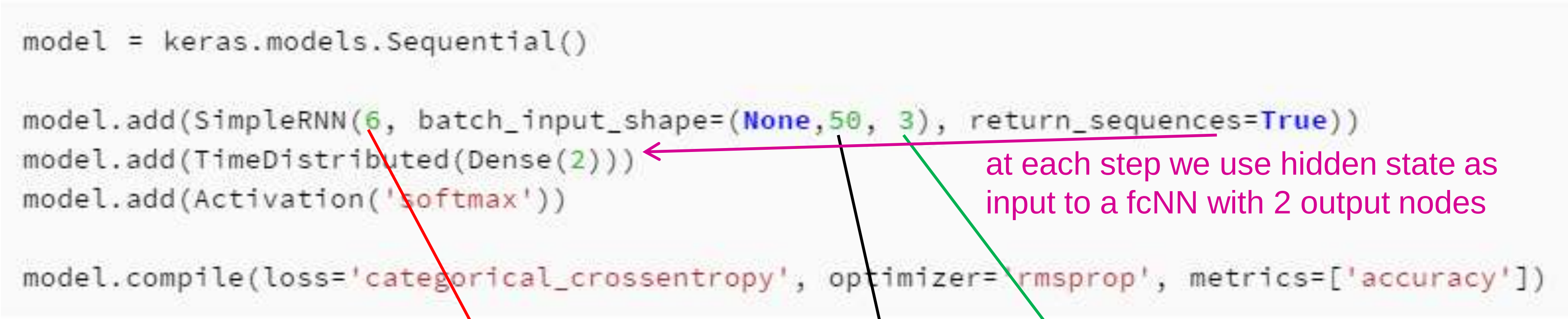
LSTM: Hochreiter et al., 1997

Slide credit: cs231 2017 stanford

RNN in Keras



```python
from keras.layers import Activation, Dense, SimpleRNN, TimeDistributed
```



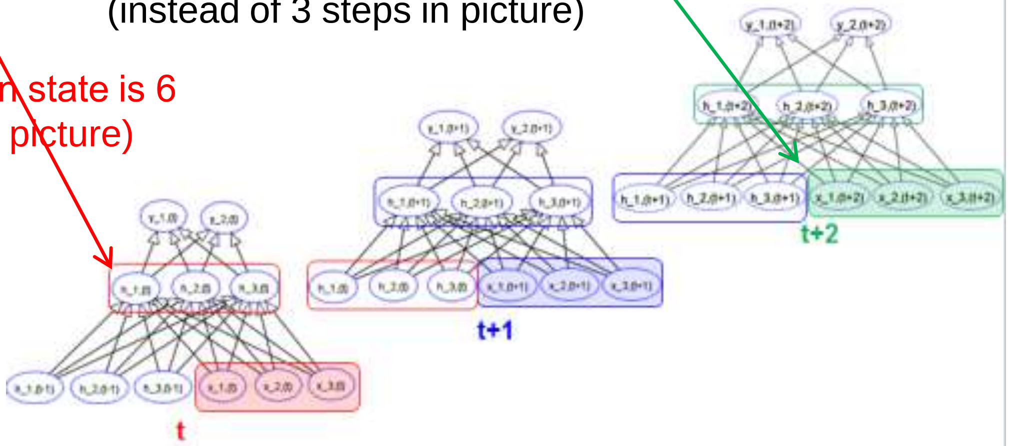
```python
model = keras.models.Sequential()

model.add(SimpleRNN(6, batch_input_shape=(None,50, 3), return_sequences=True))
model.add(TimeDistributed(Dense(2)))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
```

at each step we use hidden state as input to a fcNN with 2 output nodes

length of input vector at each time step is here 3

each training sequence consists out of 50 elements (instead of 3 steps in picture)

“capacity” of hidden state is 6 (instead of 3 in the picture)



Example: The ice cream store

Toy Example: Ice Cream Store

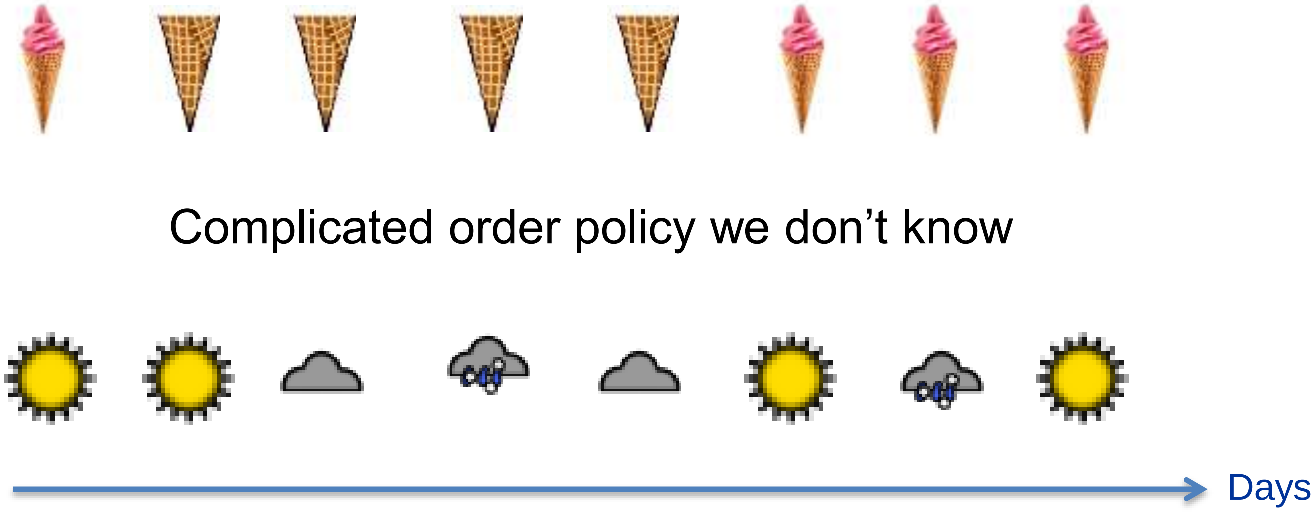
y store can sell icecream

(1,0) ice cream available

(0,1) store has run out of ice cream

x weather (1,0,0) (0,1,0) (0,0,1)

Sample



Complicated order policy we don’t know

Train a RNN and predict if ice is available based on weather in last couple of days

Resources

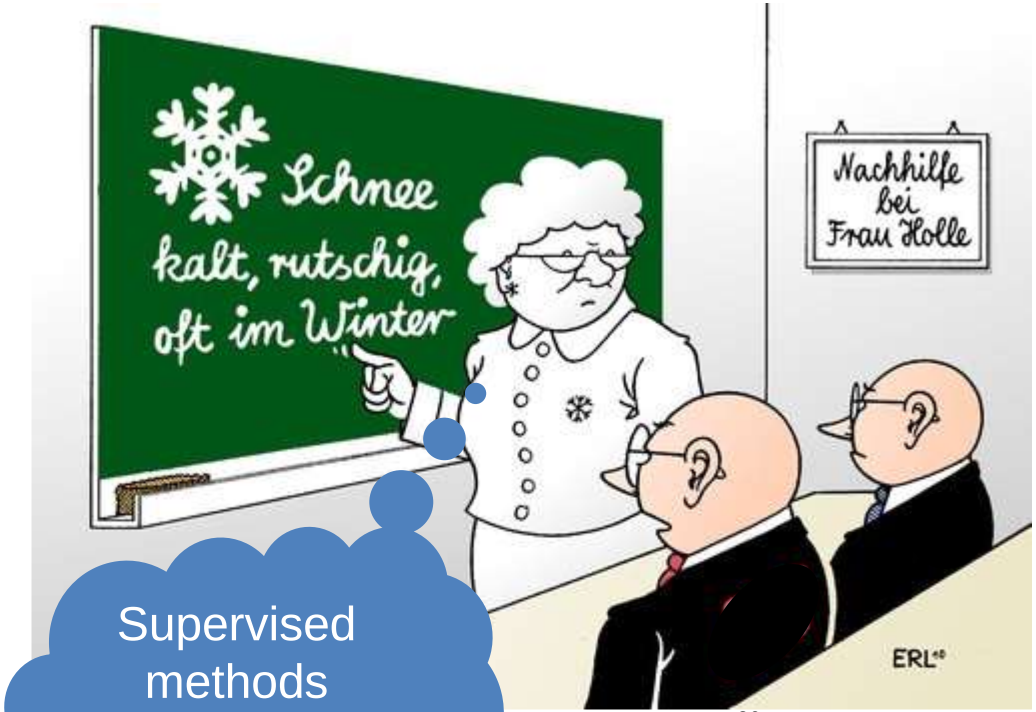
- Many figures are taken from the following resources:
  - **Deep Learning Book** chap10
    - http://www.deeplearningbook.org/contents/rnn.html
  - Other online DL courses
    - Lecture on RNN: http://cs231n.stanford.edu/slides/winter1516_lecture10.pdf
    - Video to CS231n https://www.youtube.com/watch?v=iX5V1WpxxkY
    - CS 598 LAZ Lecture 2 and 3 on RNN
  - Blog Posts
    - Karpathy, May 2015: The unreasonable effectiveness of Recurrent Neural Networks http://karpathy.github.io/2015/05/21/rnn-effectiveness/
    - Colah, August 2015: Understanding LSTM Networks http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Unsupervised Learning

From supervised learning to unsupervised feature construction

unsupervised learning

Supervised learning



Supervised methods work best at the moment

26

Citation (Yann LeCun, 2018)

"THE REVOLUTION WILL NOT BE SUPERVISED"

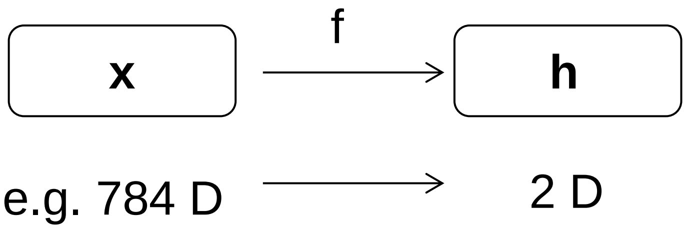
http://engineering.nyu.edu/news/2018/03/06/revolution-will-not-be-supervised-promises-facebooks-yann-lecun-kickoff-ai-seminar

2D visualization can help to discover group structure

- Learn mapping to new (low-dimensional) data representation / features

  $f: \boldsymbol{x} \rightarrow \boldsymbol{h}$

Each observation (e.g. subject, image, document) is described by many features (e.g. purchases, pixels values, words)



x

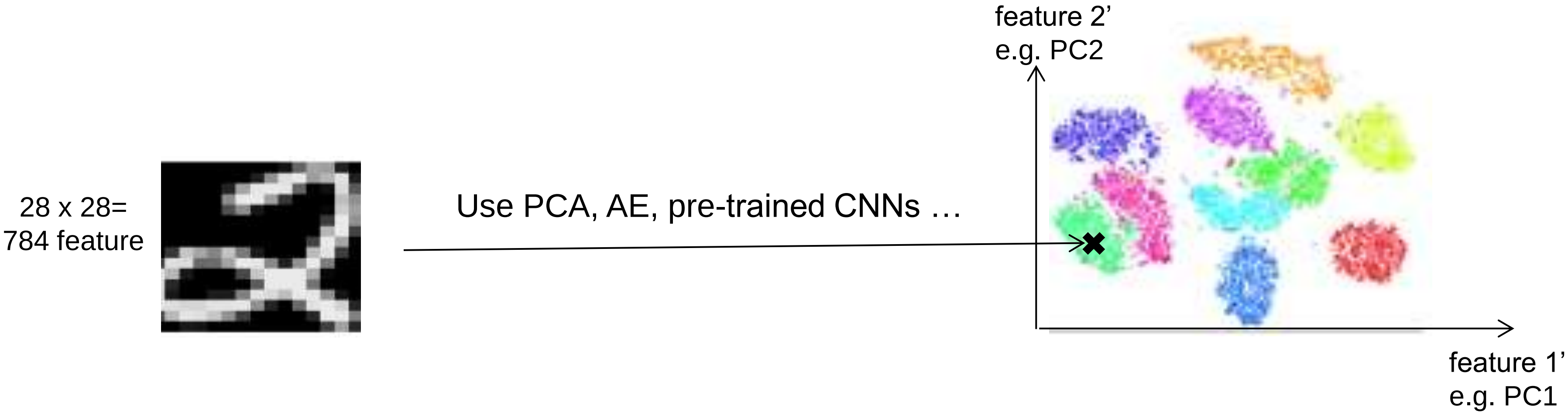
f

h

e.g. 784 D

2 D

Same observations in new data representation e.g. given by the first 2 principle components.



28 x 28=
784 feature

Use PCA, AE, pre-trained CNNs …

Principle Idea of low-dimensional data visualization: it's all about compromise

- Have data in high dimensional space with distance, (e.g. 99 features)

or (➔)

- Have distances / dissimilarities d_{ij} between many objects (e.g. 100 Objects)

- Draw this in low dimensional space (2, 3)

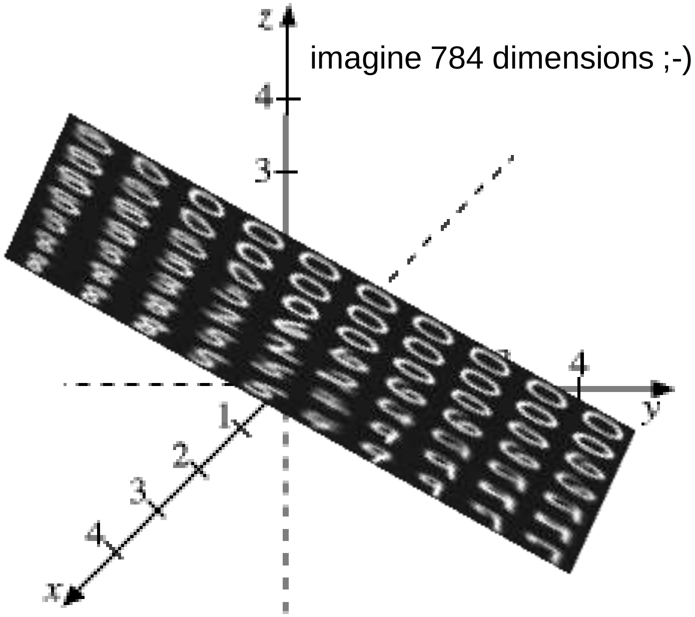
$$d_{ij} \rightarrow d^{*}_{ij} = \left\| \vec{y}_i - \vec{y}_j \right\|^2$$

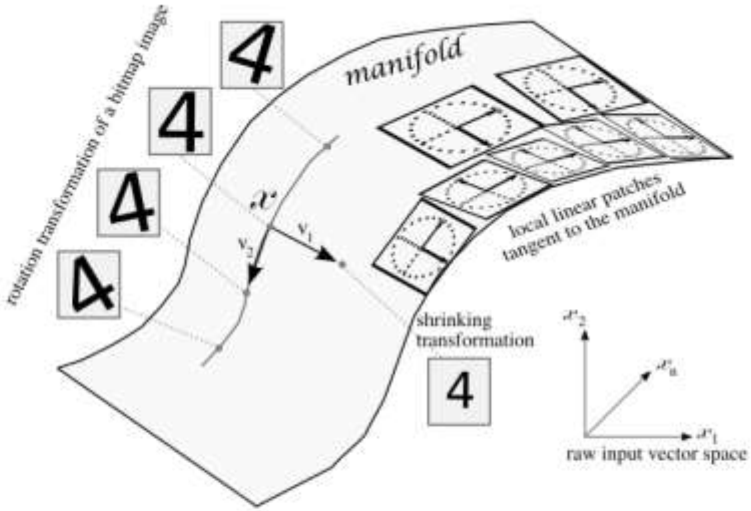
distance between i, j in high dimensional space

distance between i, j in low 2,3 dimensional space (Euclidean)

- The distances in (low-D) d^*_{ij} should match the original ones d_{ij} (high-D) as "**good as possible"**

The Manifold Hypothesis:
Why dimension reduction should work at all

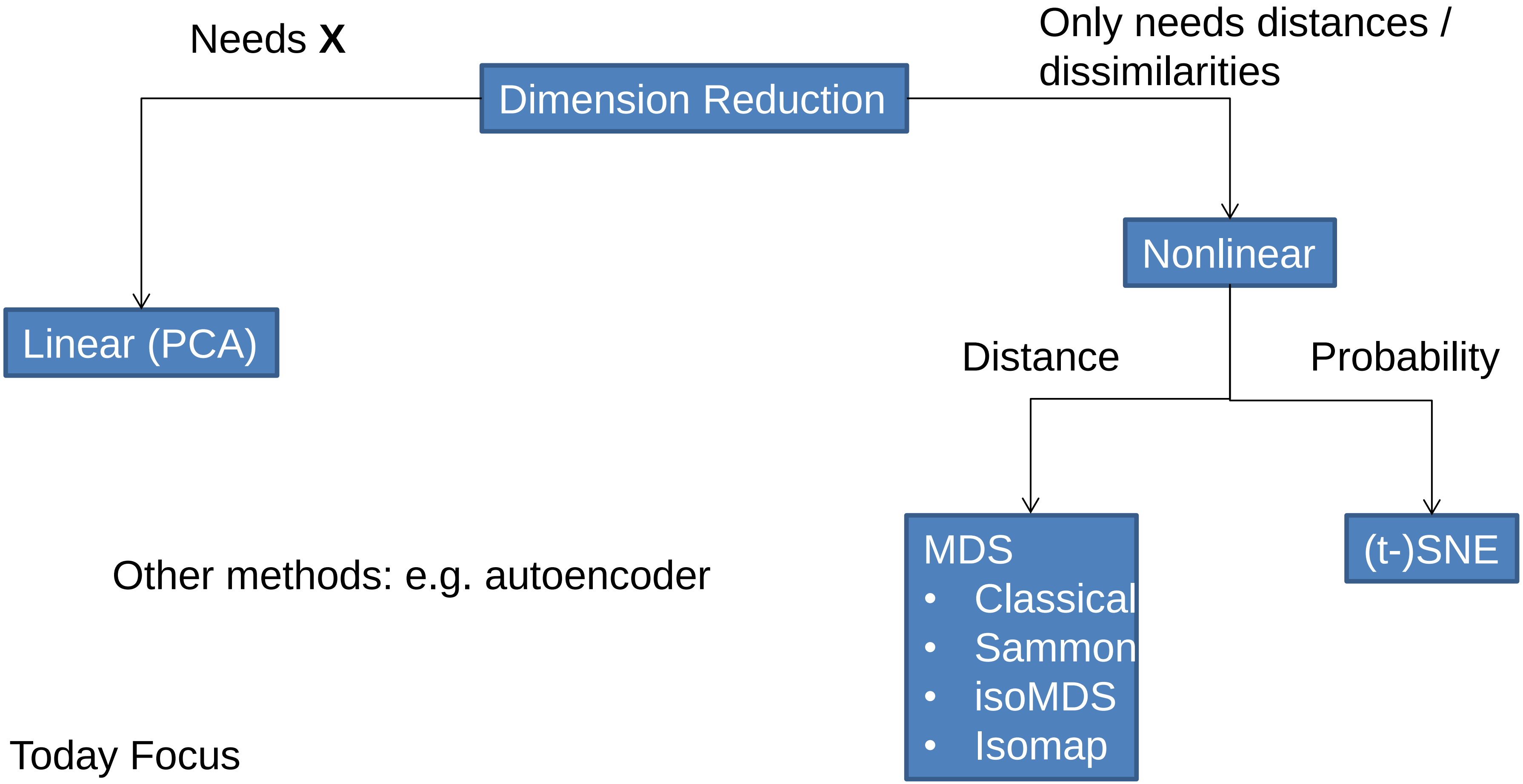

imagine 784 dimensions ;-)


rotation transformation of a bitmap image
4
4
4
4
manifold
x
v1
v2
local linear patches tangent to the manifold
shrinking transformation
4
x'2
x'n
x'1
raw input vector space

When using PCA for dimension reduction we hope that data live on a 2D hyperplane which is spanned by the first 2 principle components

The real data „lives“ on a „low“ dimensional manifold of all possible data.

Visualizing Distances: A Taxonomy of techniques



Needs **X**

Only needs distances / dissimilarities

Dimension Reduction

Linear (PCA)

Nonlinear

Distance

Probability

MDS
- Classical
- Sammon
- isoMDS
- Isomap

(t-)SNE

Other methods: e.g. autoencoder

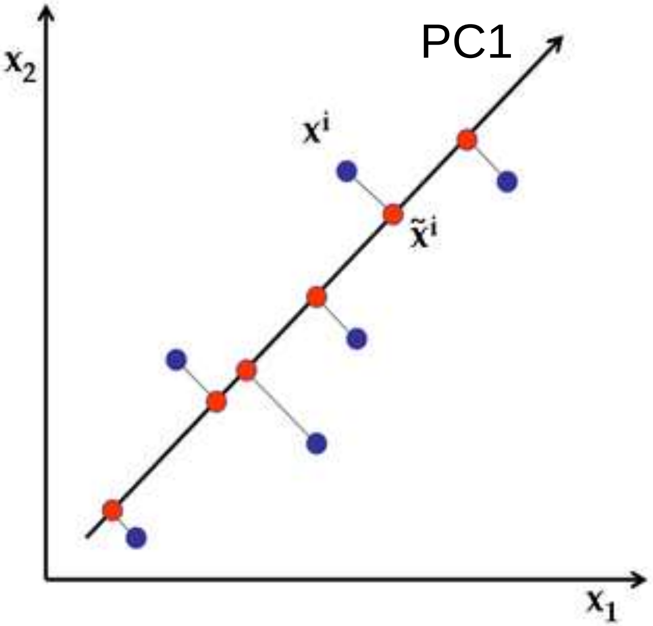
Today Focus
- PCA 1901 (Karl Pearson)
- t-SNE 2008 (Maaten, Hinton)

For a fine grained map see (Maarten, http://www.iai.uni-bonn.de/~jz/dimensionality_reduction_a_comparative_review.pdf)

Two interpretations of the first principle component

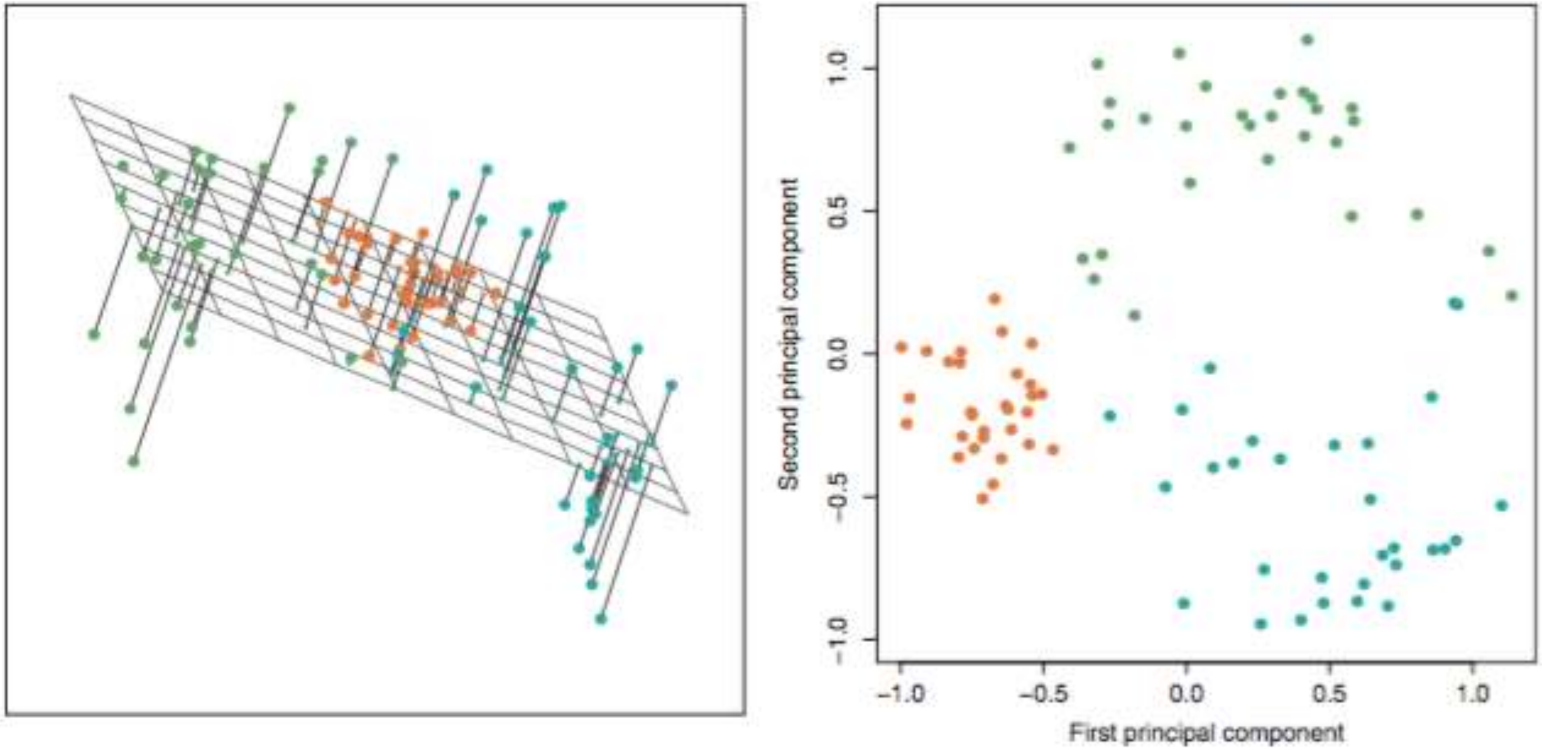
PC1 points in the direction with maximum variance. $Var(x) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$

PC1 is the projection line with minimal sum of squared orthogonal distances



In low dimensions “large” distances are especially good preserved
when using PCA for dimensions reduction!

PC1 & PC2 are the 2D hyperplane capturing most variance and minimizing sum of squared orthogonal distances

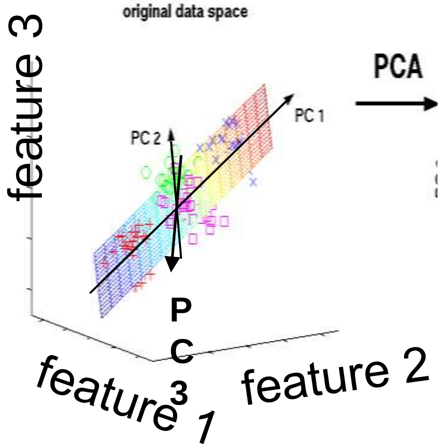


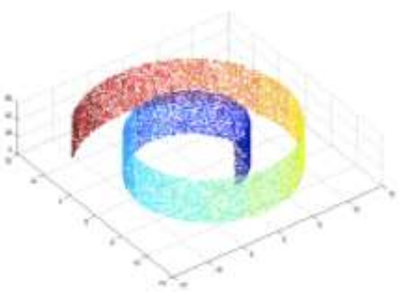
For 2D PCA visualization we need also the second principle component that is orthogonal to the first pc and points in the direction of second largest variance.

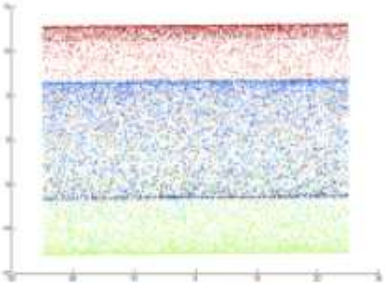
In this 2D plane "large" distances are especially good preserved.

Potential drawback of PCA:
The swiss roll example

There is (almost) no reason, why the data should lie on a 2D hyper plane.







In 2D PCA plot large distances within a plane are good preserved.

Better goal: Preserve local structure. Keep local distances intact.

Image credit: van der Maaten *et al.* 2007

Side track:
Kulback-Leibler divergence

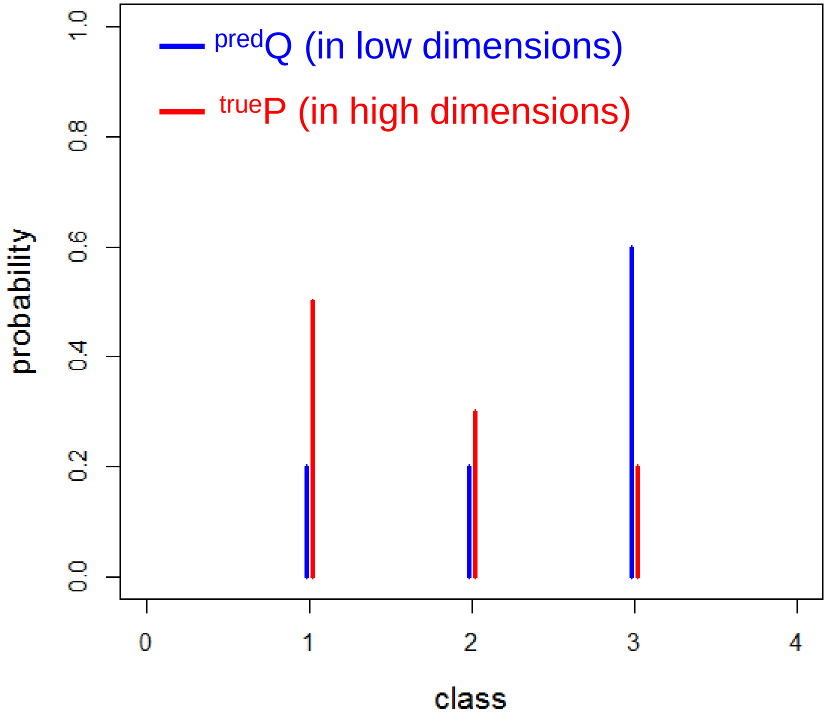
Side track: Kulback-Leibler divergence

$$D_{KL}\left({}^{true}P \,||\, {}^{pred}Q\right) = \sum_i {}^{true}p_i \cdot \log\left(\frac{{}^{true}p_i}{{}^{pred}q_i}\right)$$

$$= \sum_i {}^{true}p_i \cdot \left(\log({}^{true}p_i) - \log({}^{pred}q_i)\right)$$

$$= E_{{}^{true}p}\left(\log({}^{true}p_i) - \log({}^{pred}q_i)\right)$$

$$= E_{{}^{true}p}\left(\log\left(\frac{{}^{true}p_i}{{}^{pred}q_i}\right)\right)$$

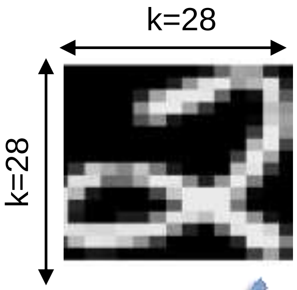


$$D_{KL}\left(Q \,||\, P\right) \neq D_{KL}\left(P \,||\, Q\right)$$

The Kulback-Leibler divergence measures the distance between two distributions, giving the two distributions not symmetric roles.

The Kullback–Leibler divergence $D_{KL}({}^{true}P||{}^{pred}Q)$ is defined only if ${}^{pred}q_i=0$ implies ${}^{true}p_i=0$, for all *i* . Whenever q_i is zero the contribution of the *i*-th term is interpreted as zero because lim q->0 (q*log(q)) = 0

Side track: Cross-entropy is used as cost function (recap)



Flatten to vector with k² elements

input $\mathbf{x}^t$

$$\begin{bmatrix} 24.0 \\ -5.1 \\ 12.2 \\ 89.9 \\ \vdots \\ 3.2 \end{bmatrix}$$

$$\mathbf{x}_{(1,k^2)} \cdot \mathbf{W}_{(k^2,10)} + \mathbf{b}_{(1,10)} = \mathbf{z}_{(1,10)}$$

Score or logit $\mathbf{z}^t$

$$\begin{bmatrix} -0.9 \\ 0.1 \\ 2.3 \\ 1.2 \\ 0.9 \\ 2.1 \\ -1.2 \\ -0.2 \\ 1.3 \\ 0.9 \end{bmatrix}$$

$$\hat{p}_k = \frac{e^{z_k}}{\sum_j e^{zj}}$$

Softmax $\mathbf{p}$=S($\mathbf{z}$)

$$\begin{bmatrix} 0.01 \\ 0.03 \\ 0.31 \\ 0.10 \\ 0.08 \\ 0.25 \\ 0.01 \\ 0.03 \\ 0.11 \\ 0.08 \end{bmatrix}$$

1-hot labels $\mathbf{y}$

0	0
0	1
1	2
0	3
0	4
0	5
0	6
0	7
0	8
0	9

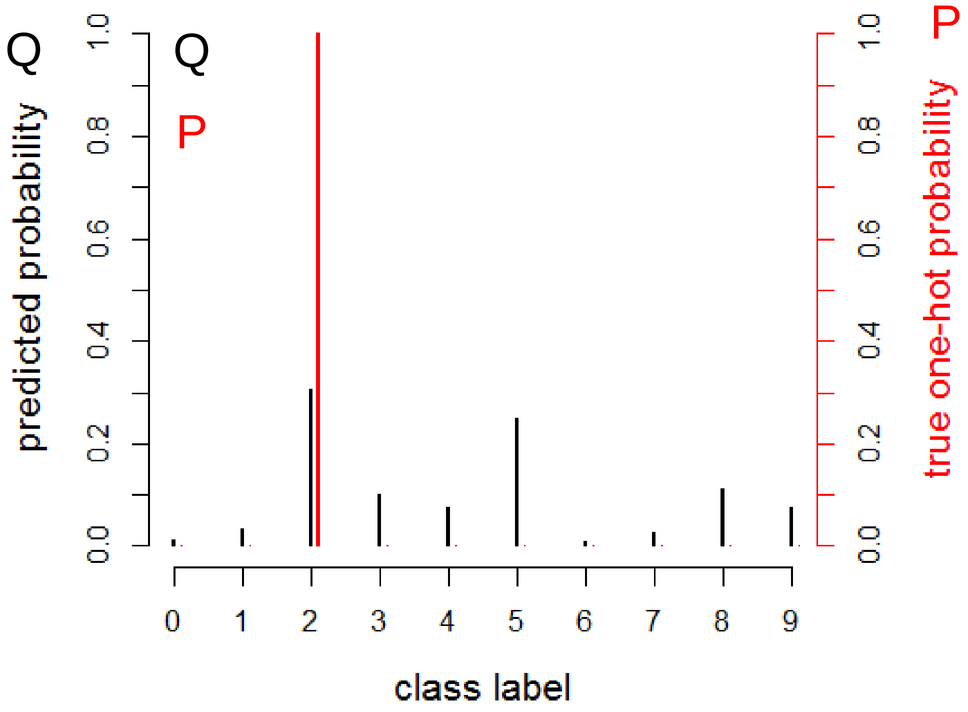
$D(\mathbf{p},\mathbf{y})$ cross-entropy

$$= -\sum_{k=1}^{2} y_k \cdot \log\left(p_k\right)$$

Cost C or Loss averaged over all images in a mini-batch:

$$\text{C}=\frac{1}{N}\sum_i D(\mathbf{p}_i,\mathbf{y}_i)$$

Side track: X-entropy and KL-divergence



x-entropy=KL

in case of 1-hot encoded true P

$$\text{x-entropy} = -\sum_i {}^{true}\text{p}_i \cdot \log\left({}^{pred}\text{q}_i\right)$$

$$\text{KL}({}^{true}\text{p}|{}^{pred}\text{q}) = \sum_i {}^{true}\text{p}_i \cdot \log\left(\frac{{}^{true}\text{p}_i}{{}^{pred}\text{q}_i}\right)$$

$$= \sum_i {}^{true}\text{p}_i \cdot \log\left({}^{true}\text{p}_i\right) - \sum_i {}^{true}\text{p}_i \cdot \log\left({}^{pred}\text{q}_i\right)$$

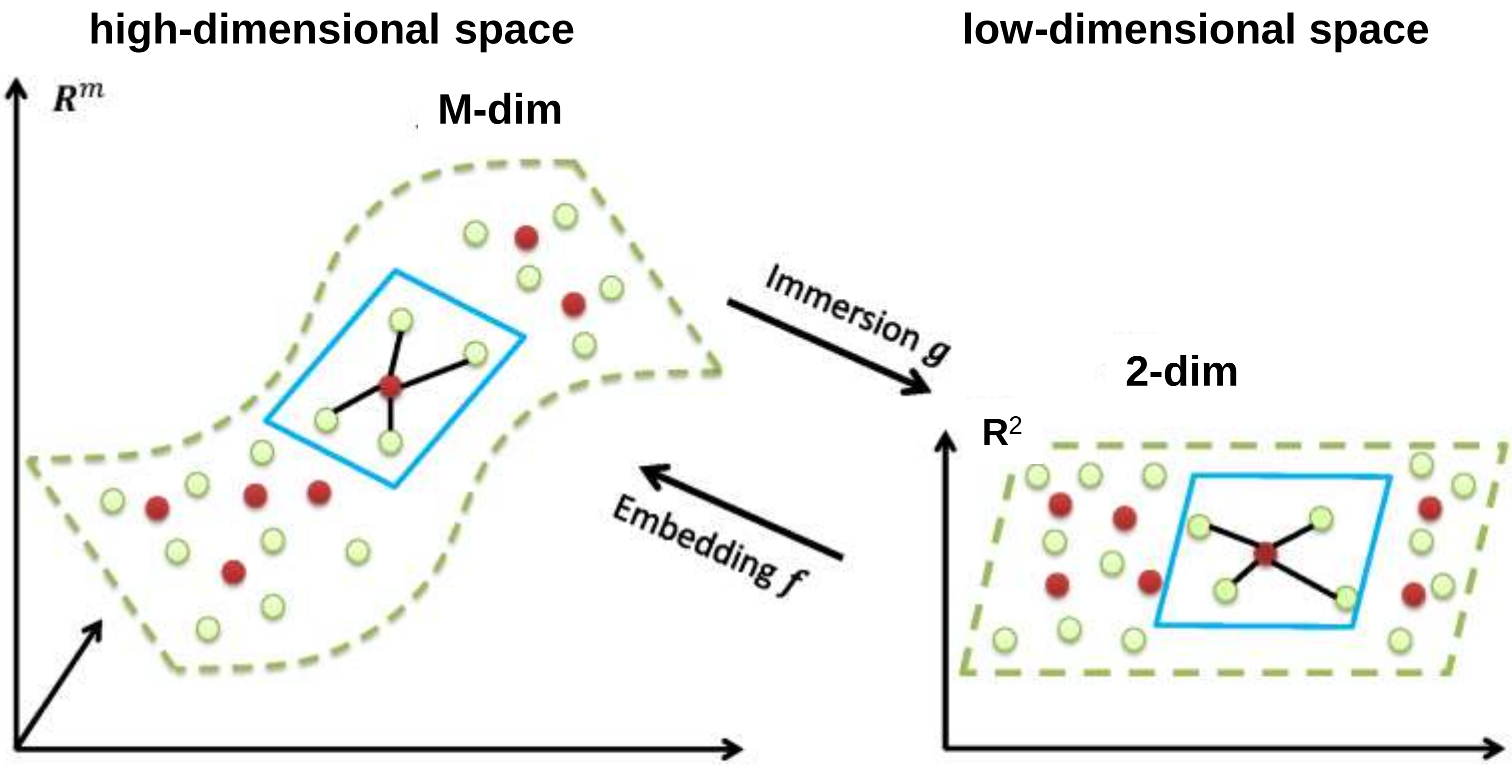
$$= 0 - \sum_i {}^{true}\text{p}_i \cdot \log\left({}^{pred}\text{q}_i\right)$$

Stochastic Neighbour Embedding (SNE)

The new kid on the block

Motivation of Stochastic Neighbour Embedding

t-SNE aims to preserve the close neighborhood of each data point opposed to dimension reduction with PCA that preserves large distances.



high-dimensional space

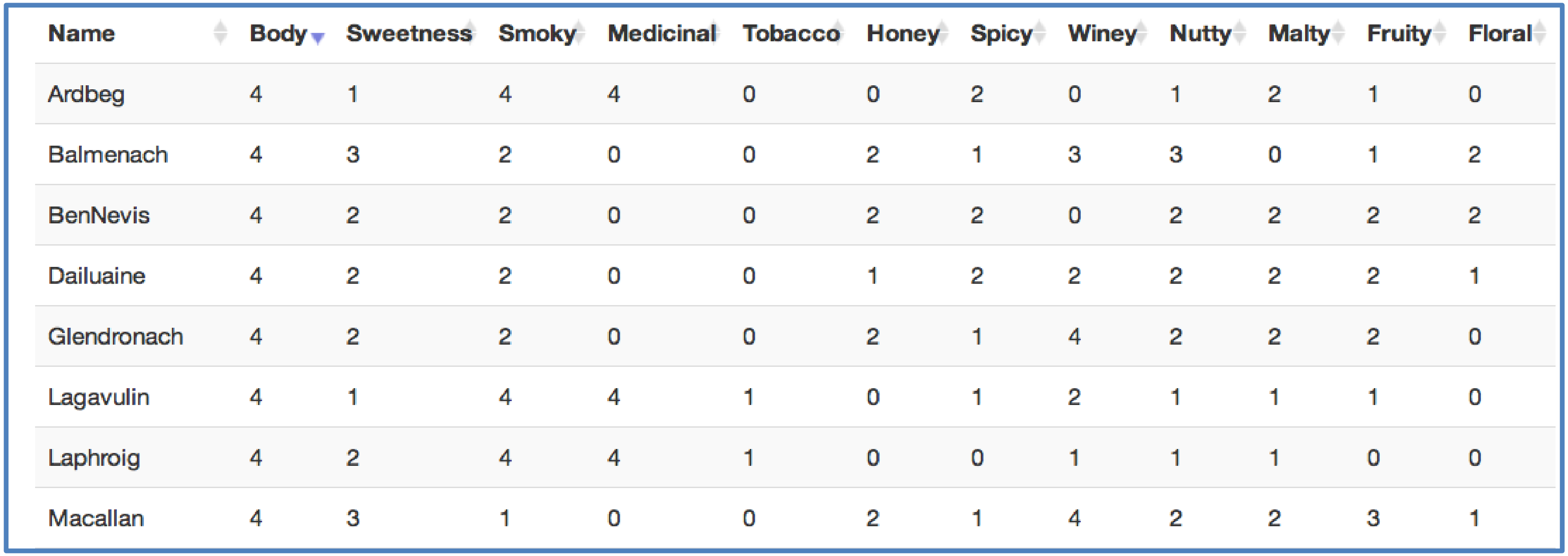
low-dimensional space

M-dim

2-dim

Image credits: https://www.eecis.udel.edu/~zhou/Research.html

Example high dimensional data: Whiskey dataset



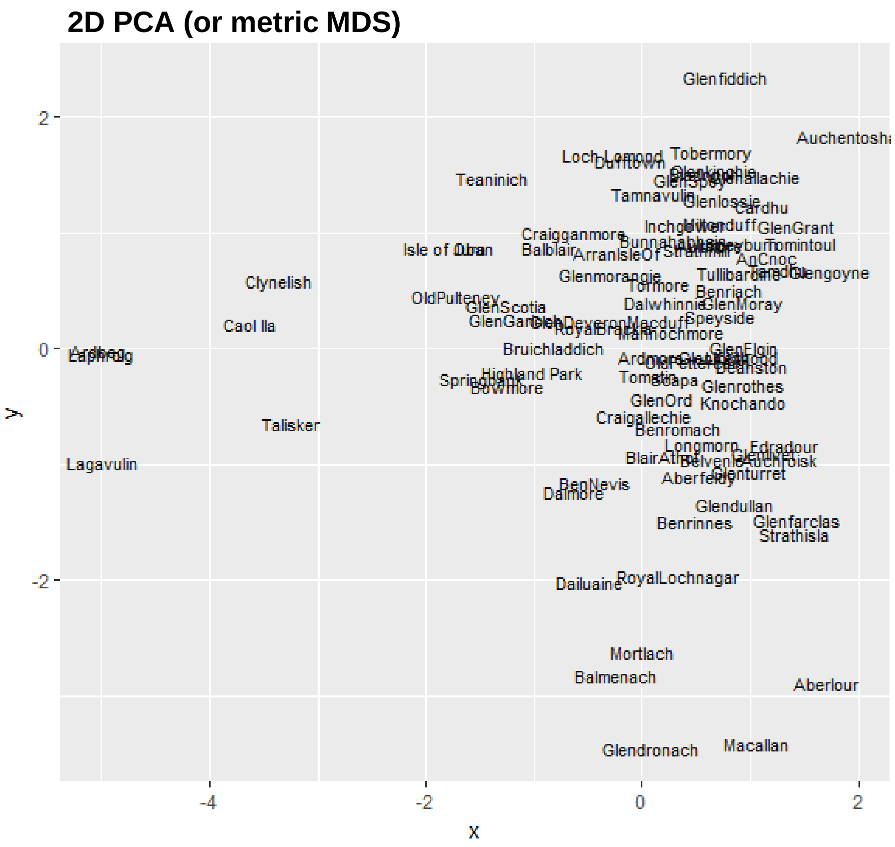
Name	Body	Sweetness	Smoky	Medicinal	Tobacco	Honey	Spicy	Winey	Nutty	Malty	Fruity	Floral
Ardbeg	4	1	4	4	0	0	2	0	1	2	1	0
Balmenach	4	3	2	0	0	2	1	3	3	0	1	2
BenNevis	4	2	2	0	0	2	2	0	2	2	2	2
Dailuaine	4	2	2	0	0	1	2	2	2	2	2	1
Glendronach	4	2	2	0	0	2	1	4	2	2	2	0
Lagavulin	4	1	4	4	1	0	1	2	1	1	1	0
Laphroig	4	2	4	4	1	0	0	1	1	1	0	0
Macallan	4	3	1	0	0	2	1	4	2	2	3	1

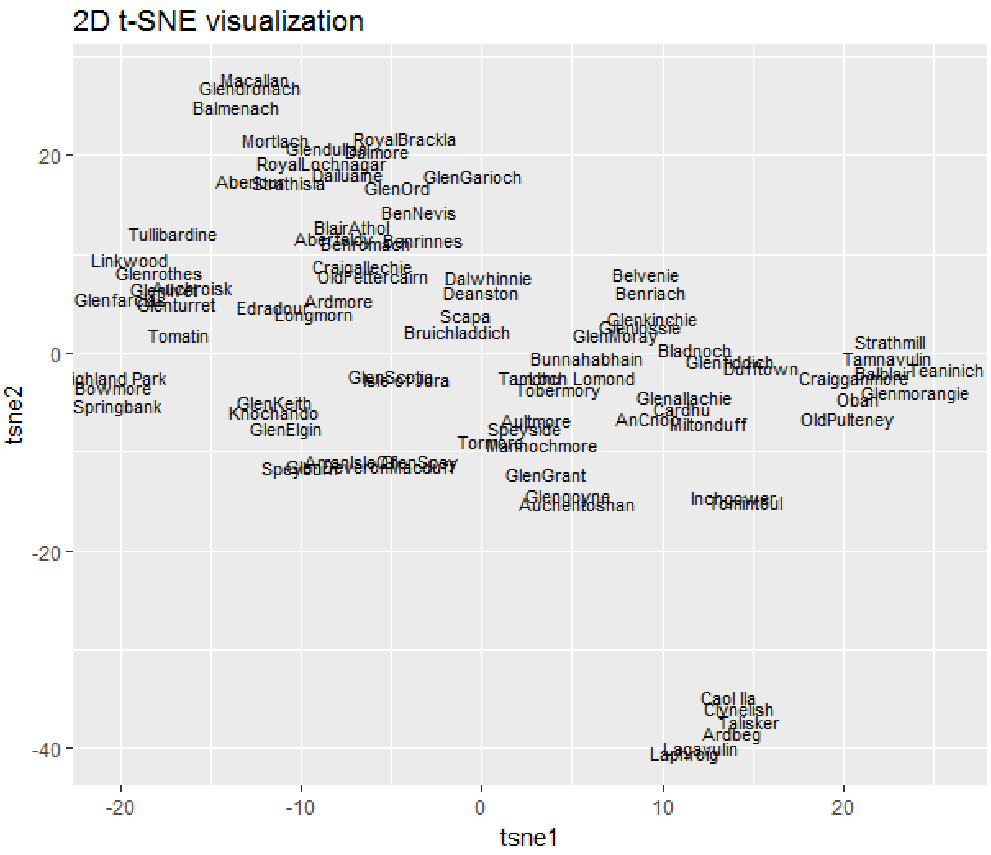
84 Whiskeys, 12 Features

⋮

Data Source: Whiskyclassified.com, https://www.mathstat.strath.ac.uk/outreach/nessie/nessie_whisky.html

Whiskies 2D representation by MDS/PCA or t-SNE


2D PCA (or metric MDS)


2D t-SNE visualization

PCA

- less nice clustering of similar whiskies

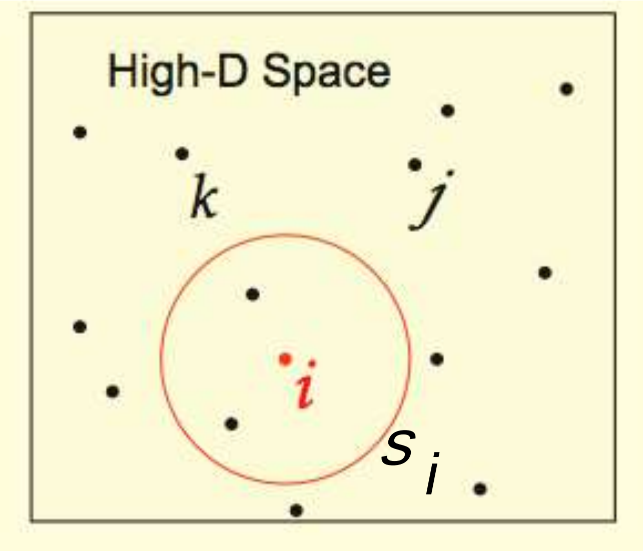
+ allows to construct PCA scores components from new high-D data

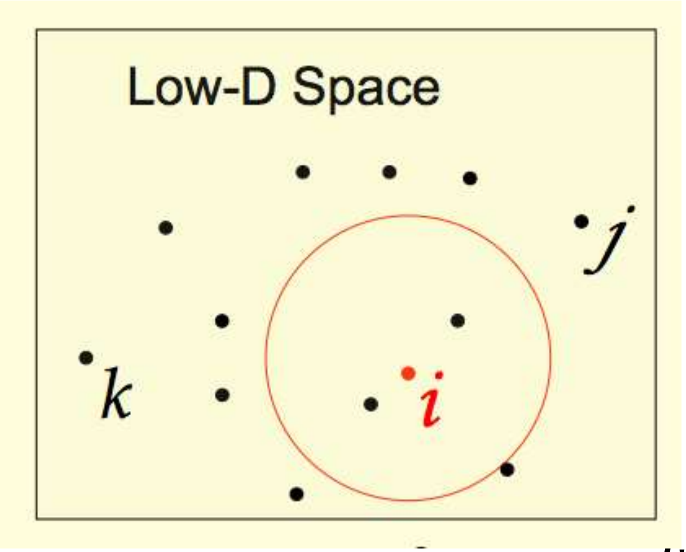
t-SNE

+ nice clustering of similar whiskies

- does not allow to construct t-SNE components from new high-D data

SNE Details: Similarity of j as high probability to be picked when bell-shaped distribution is centred at i





$$p_{j|i} = \frac{e^{-d_{ij}^2 / 2\sigma_i^2}}{\sum_k e^{-d_{ik}^2 / 2\sigma_i^2}}$$

$$q_{j|i} = \frac{e^{- d^{*2}_{ij}}}{\mathring{a}_k \, e^{- d^{*2}_{ik}}}$$

probability of picking j
given that you start at I
(based on a t-distribution)

probability of picking j
given that you start at i
(based on a Gauss)

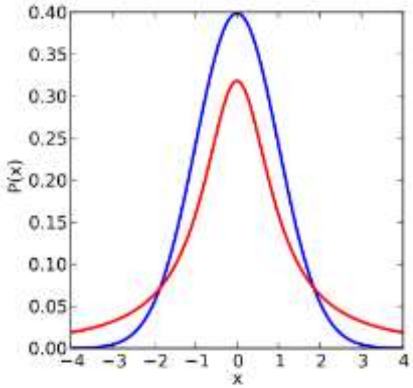
σ$_i$ is selected so that the number of neighbors is about constant (parameter **perplexity**)

Cost function

$$Cost = \sum_i KL(P_i \,||\, Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}$$

- KL is standard, when comparing two distributions
- Solution: data point placement in low-D space so that Cost is minimal
- For points where p_{ij} is large and $q_{ij} \not\approx p_{ij}$ we lose a lot.
  - Nearby points in high-D really want to be nearby in low-D
- For small p_{ij} we lose a little for all q and even less for small q.
  - Widely separated points in high-D have a mild preference for being widely separated in low-D

The t-SNE variant



The t-SNE introduced by van Maarten & Hinton 2008

Change

- t-distribution (df=1) instead of Gaussian in low-d
  - Allows point-pairs, that are far apart in high-dim, to be even more far apart in low-dim (heavy tails of p-dist still give comparable high picking probability)

$$q_{ij}\ \mu\ \frac{1}{1+d_{ij}^2}$$

- Due to σ_i the probability is not symmetrical. Symmetrisation

$$p_{i,j} = (p_{j|i} + p_{i|j})/2n \qquad Cost = KL(P\|Q) = \mathring{a}_{i<j} p_{ij} \log\frac{p_{ij}}{q_{ij}}$$

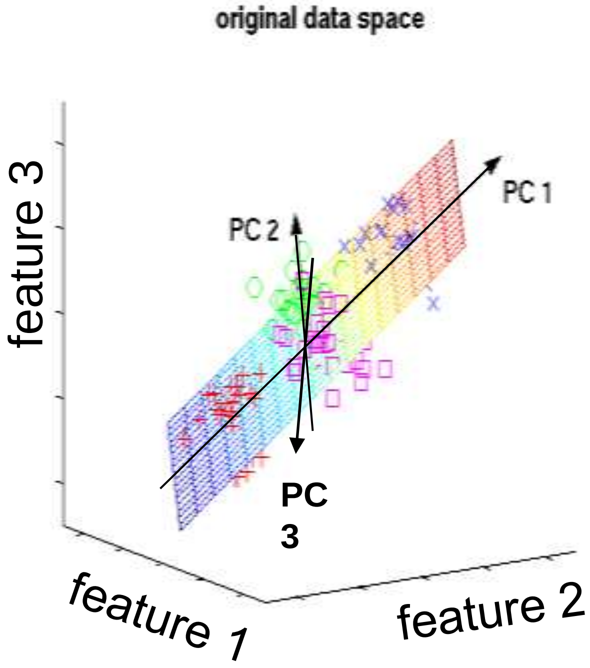
- Cost (single sum), optimized with gradient descent

Things to keep in mind when doing t-SNE

- Neighborhoods are modeled - distances between clusters are not meaningful
- Arguments `perplexity` and `#iterations` are most important
- We should allow for enough iteration to get a stable configuration in 2D
- Argument `perplexity` corresponds roughly to #neighbors expected
- `perplexity` has often default 10, but try also other values between 5 and 50
- `perplexity` should be smaller than #observations!
- Often t-SNE is applied on the first 50 PCs of the data
- approximate Barnes Hut implementation,much faster for large data sets $O(N^2) \rightarrow Nlog(N)$
- t-SNE is most appropriate in case of many data (>50)

Unsupervised feature construction

Unsupervised feature construction via PCA



The first principal component (PC_1) is constructed to capture most of the variance, the second the second most,….

Each principle component can be represented as linear combination of the original features

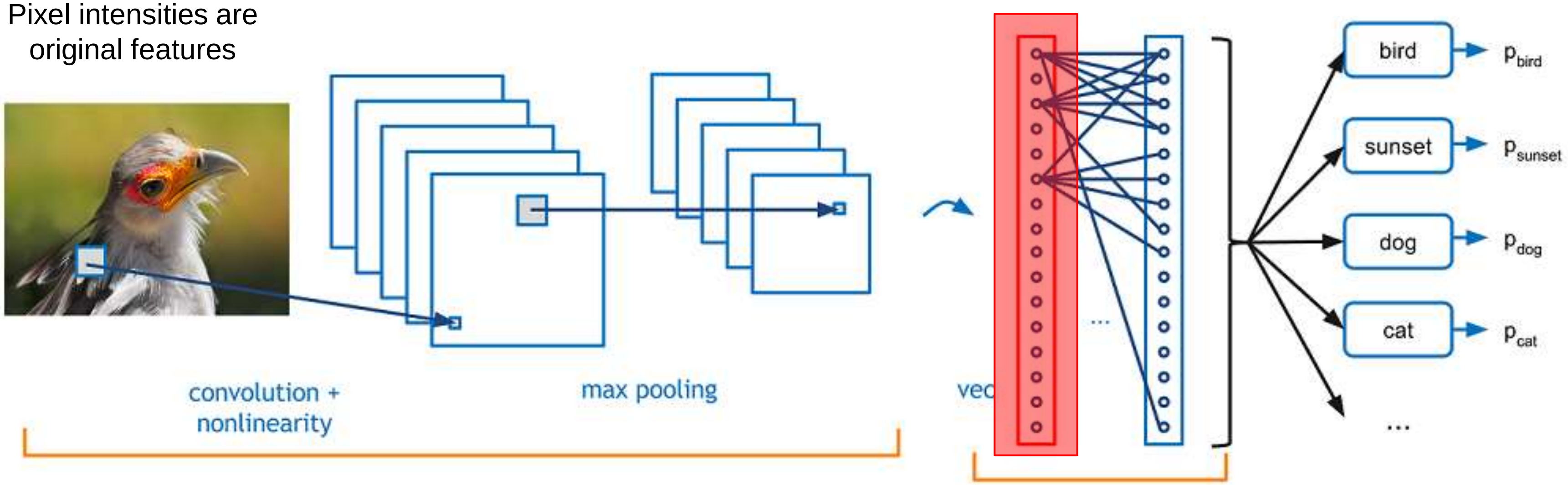
linear combination:

$$y_k = a_{1k}x_1 + a_{2k}x_2 + ... + a_{pk}x_p$$

Unsupervised feature construction via a trained CNN

Original raw data:
Pixel intensities are
original features

CNN features =
new data representation

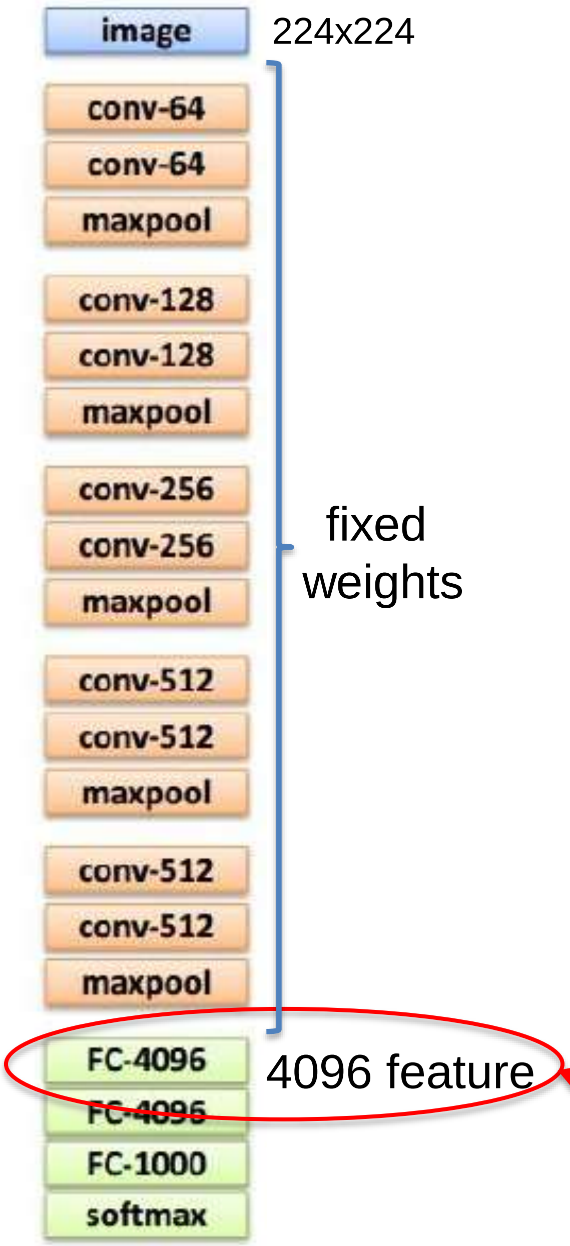


This part of the NN combines original features to new features

This part of the NN uses extracted feature for prediction

Image credits:
https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/

Unsupervised feature construction via a trained CNN



- Load a pre-trained CNN – e.g VGG16
- Rescale image to required size (224x224 for VGG16)
- Do a forward pass and fetch 4096 features that are used as CNN representations, dump these features into a file on disk

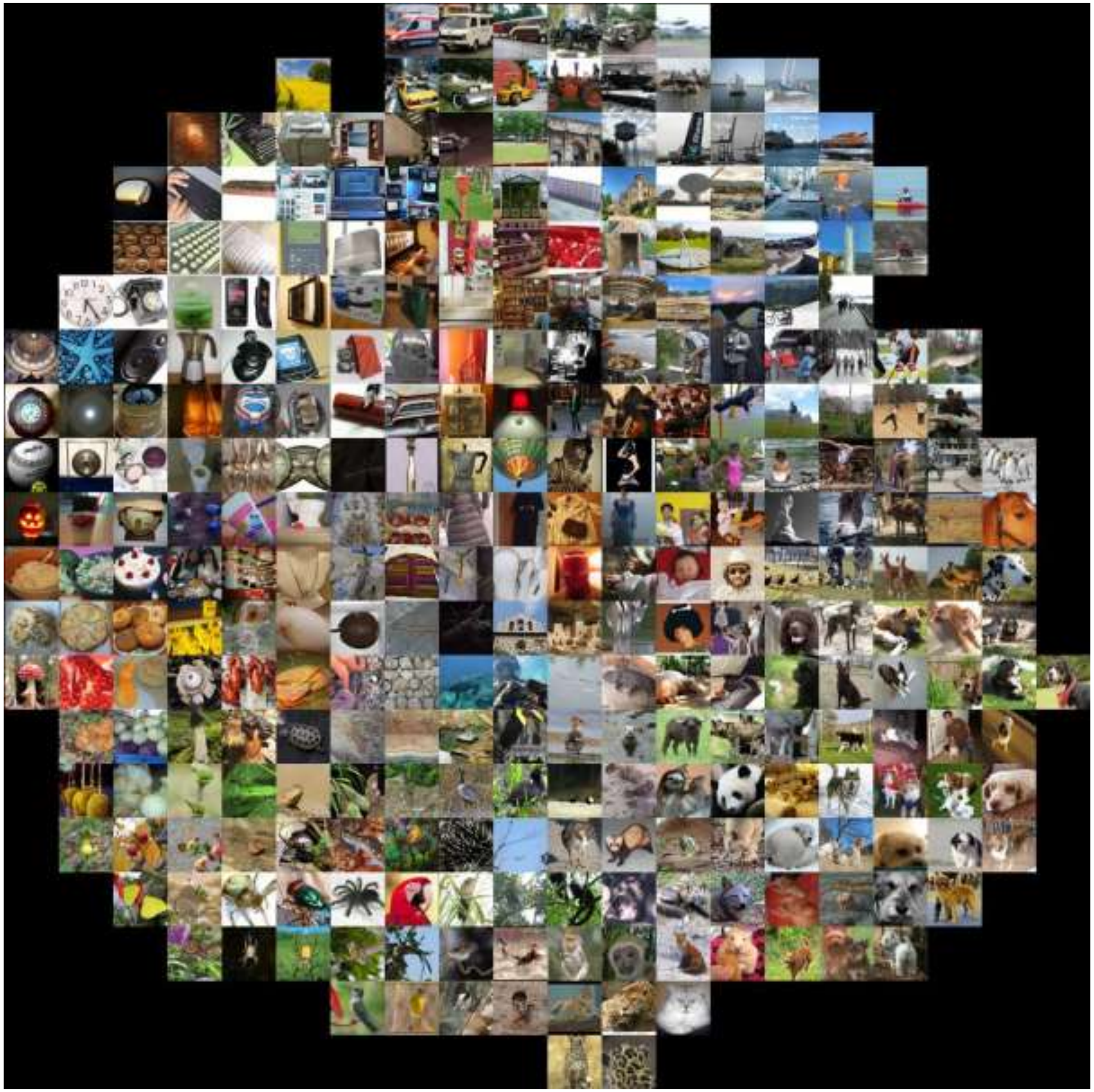
Fetch this CNN feature vector for each image

Similarity Maps in practice

Application determines used similarity measure

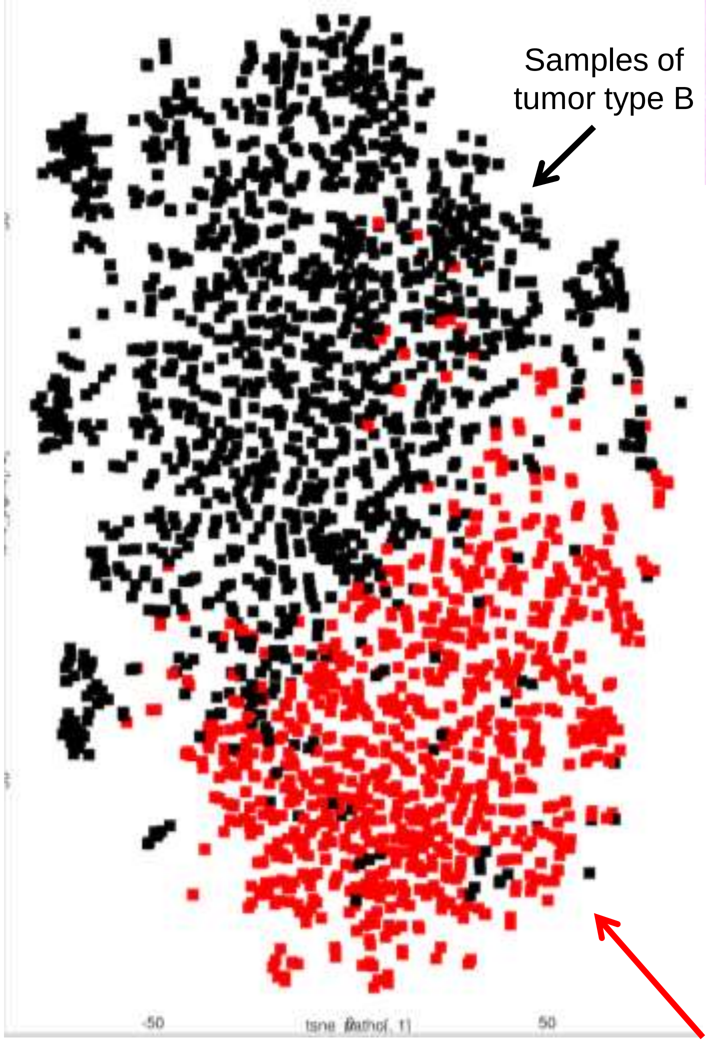
Applications (Visual Map of Images)

Semantic similarities not pixel wise!

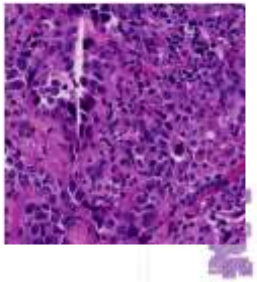


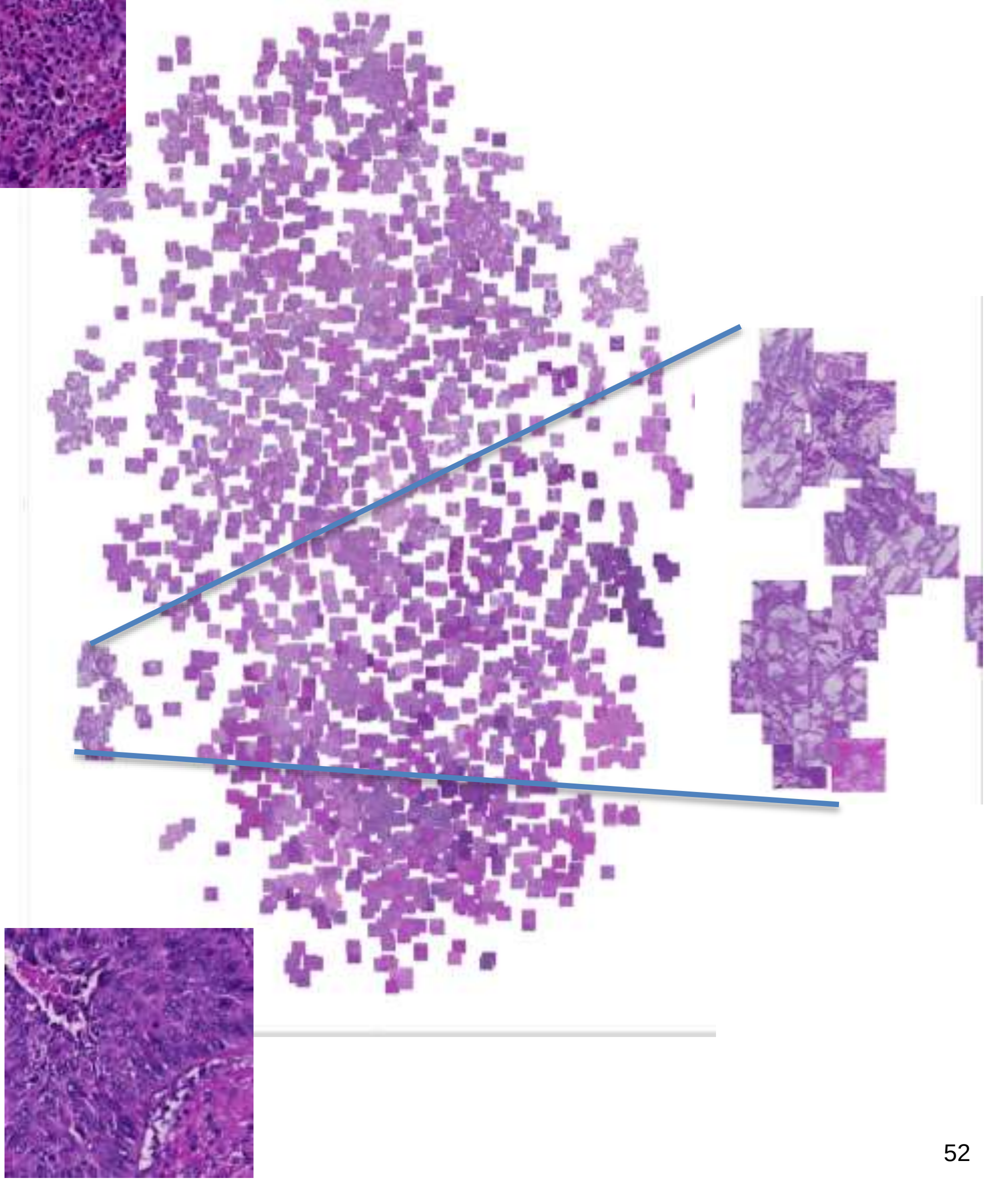
See: http://cs.stanford.edu/people/karpathy/cnnembed/ for more images

Example Project with Uni-Spital ZH: t-SNE plot of histological images based on unsupervised CNN features



Samples of tumor type B



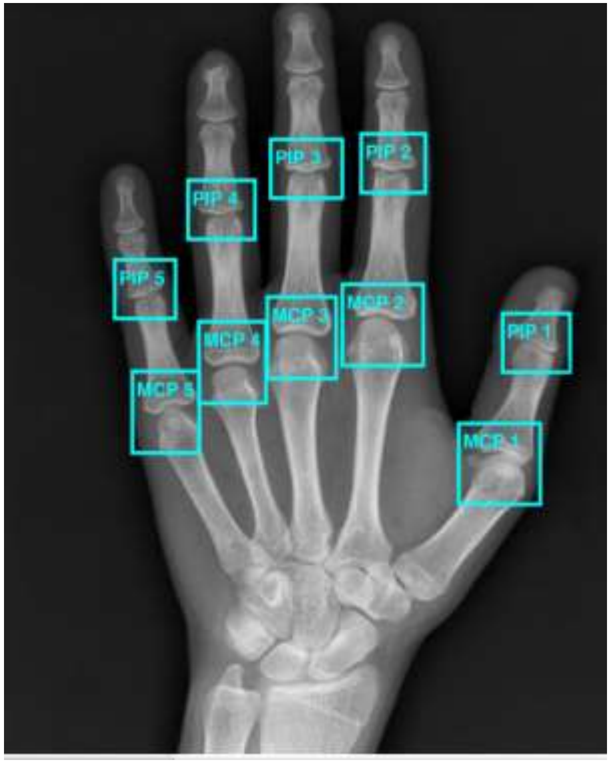


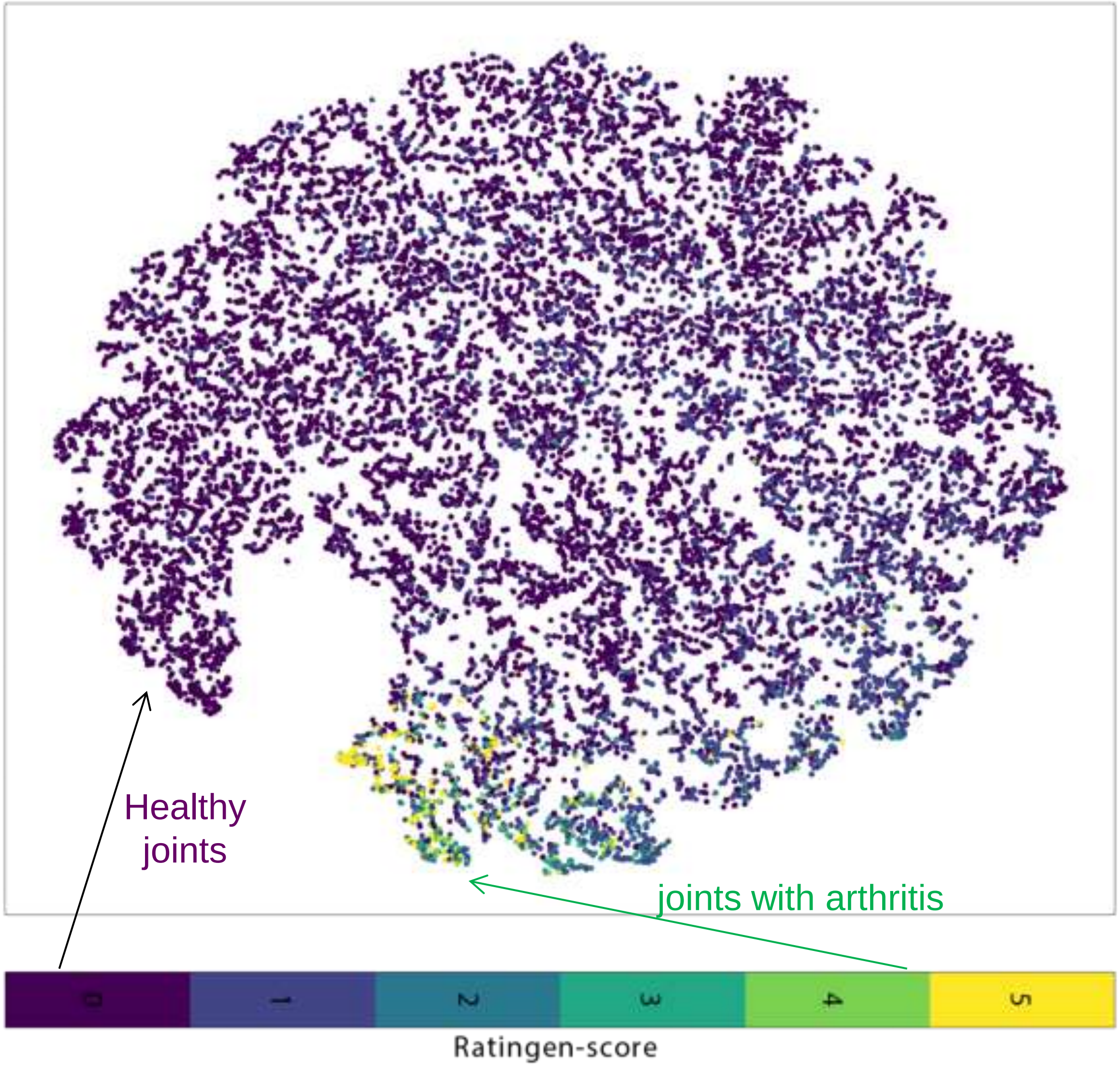
Samples of tumor type A

image credit: Elvis Murina

Example Project with Seantis and SCQM on Arthritis:

t-SNE plot of x-ray images with supervised CNN features





Healthy joints

joints with arthritis

image credit: Janick Rohrbach

2D or 3D maps as cell phenotype similarity maps, interpolation

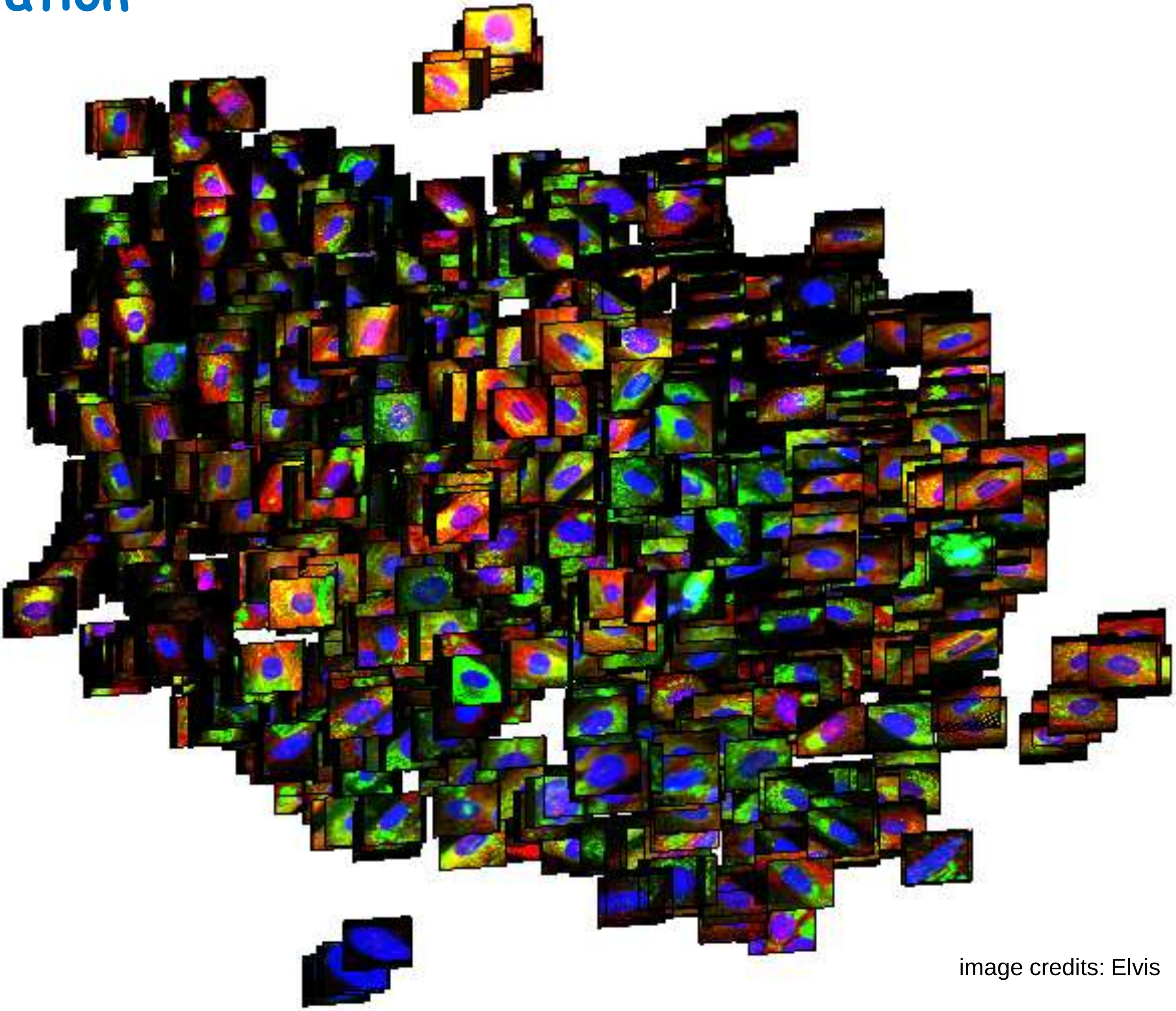
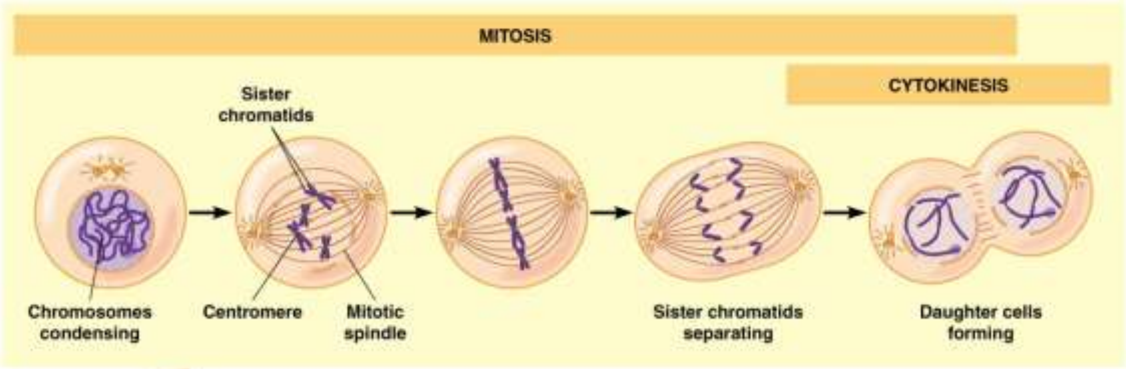
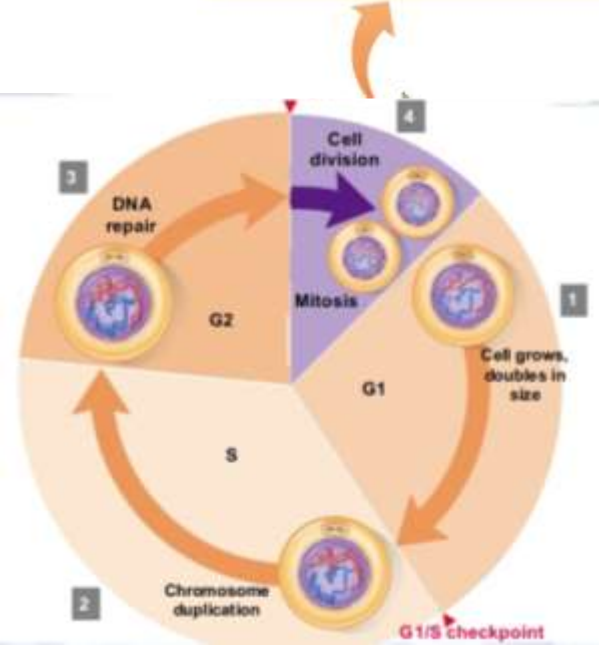


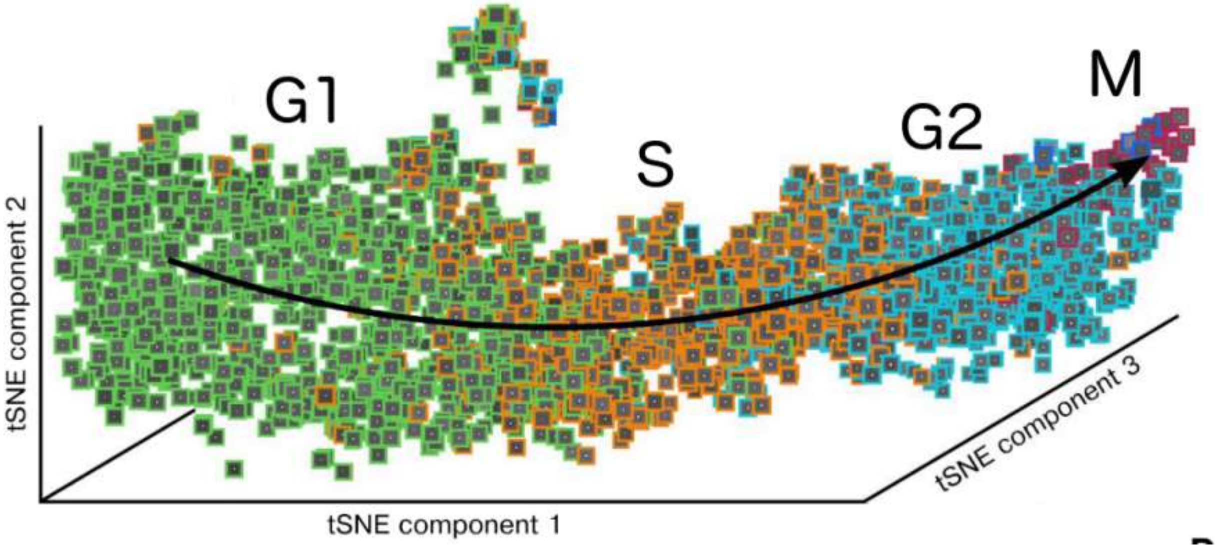
image credits: Elvis

t-sne revealed correct order of cell cycle



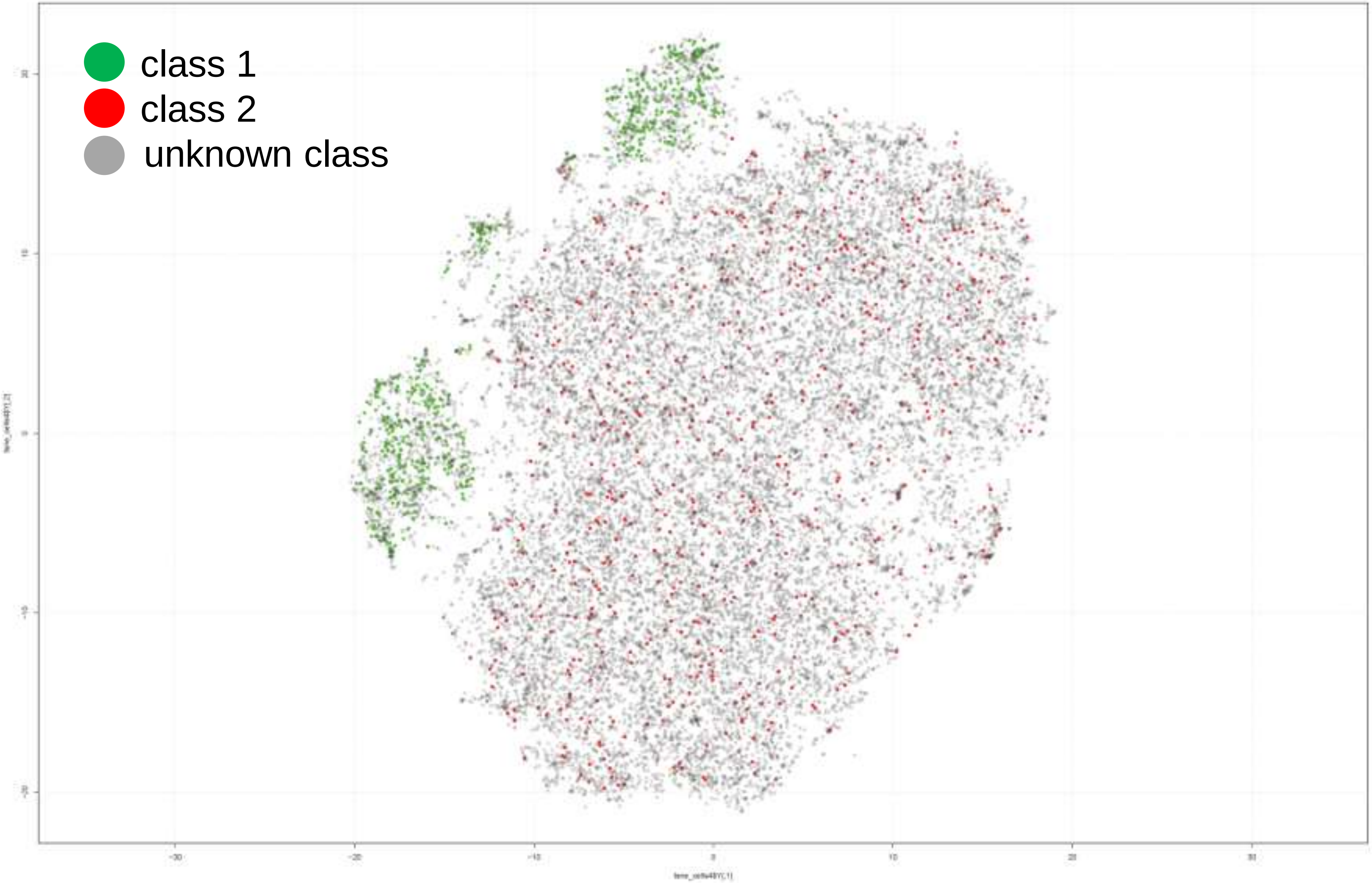
(a) The M (mitotic) phase





Source: eulenberg2016-cell-cycle.pdf
http://dx.doi.org/10.1101/081364

Enrich train set by knn-classification to next labeled data-point in t-sne containing labeled and unlabeled data



Exercise on using unsupervised constructed features for pattern recognition

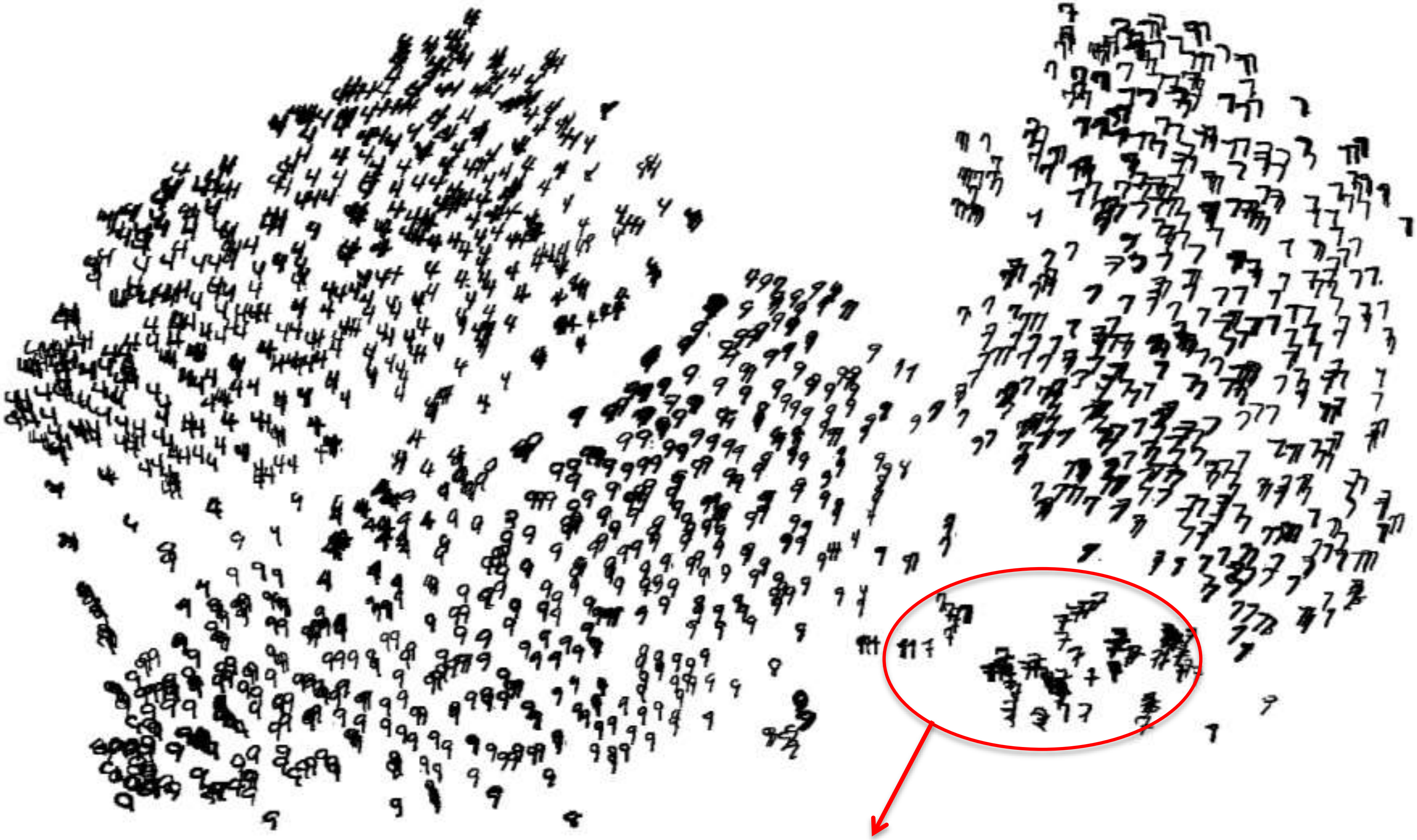
Use raw pixel features and unsupervised constructed CNN features for MNIST and CIFAR10 data to

1) do some 2D visualization

2) do some pattern recognition

3) use them for transfer learning

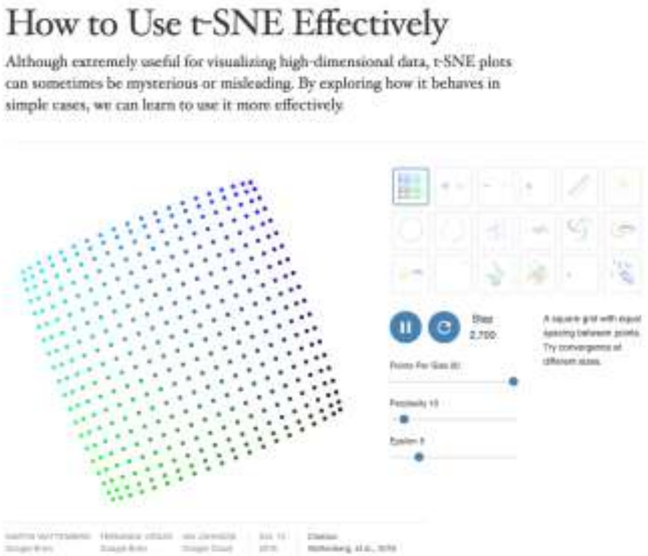
t-SNE can reveal sub-groups



European seven

Some more resources on t-SNE

- Very nice blog on t-SNE: http://distill.pub/2016/misread-tsne/



How to Use t-SNE Effectively

Although extremely useful for visualizing high-dimensional data, t-SNE plots can sometimes be mysterious or misleading. By exploring how it behaves in simple cases, we can learn to use it more effectively.

Step
2,700

A square grid with equal spacing between points. Try convergence at different sizes.

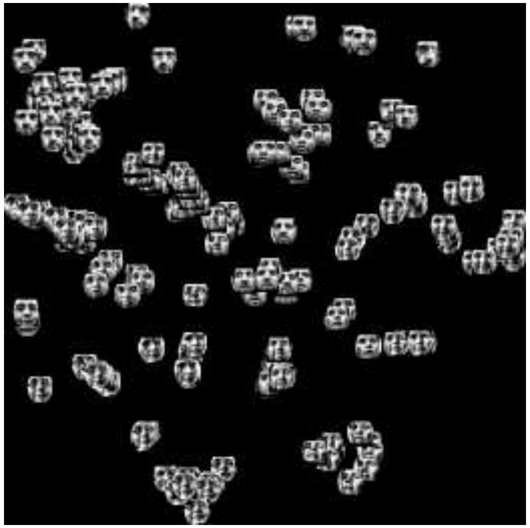
Points Per Side 20

Perplexity 10

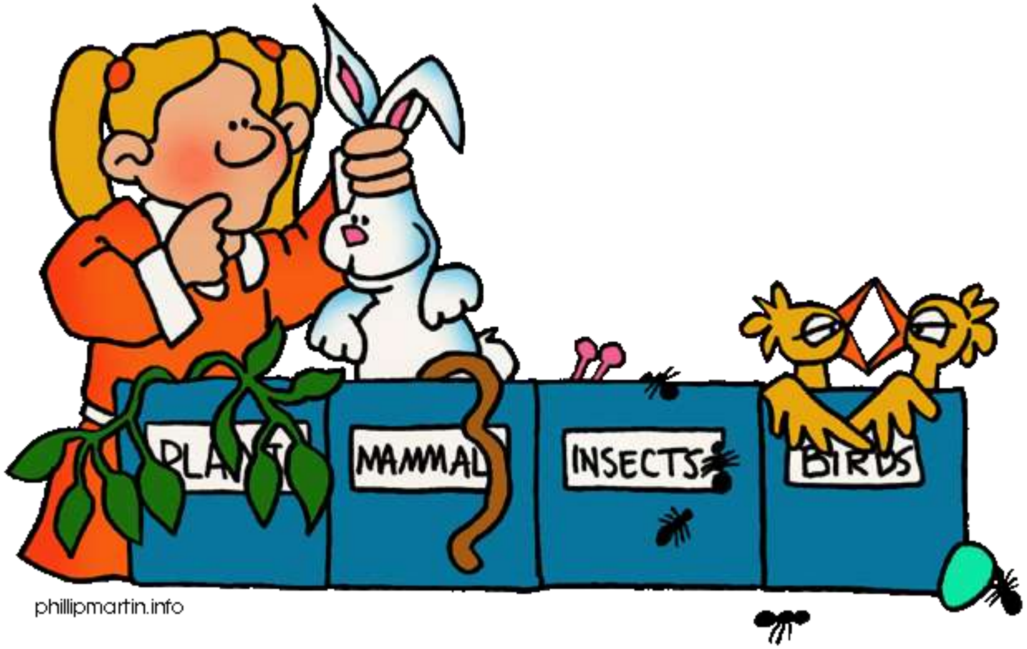
Epsilon 5

Martin Wattenberg Google Brain | Fernanda Viégas Google Brain | Ian Johnson Google Cloud | Oct. 13 2016 | Citation Wattenberg, et al., 2016

Code to plot the images in t-SNE, see e.g. cell 12 to cell 15 in: https://github.com/oduerr/dl_tutorial/blob/master/tensorflow/inception_cifar10/Analyse_CIFAR-10_TSNE.ipynb



Classification

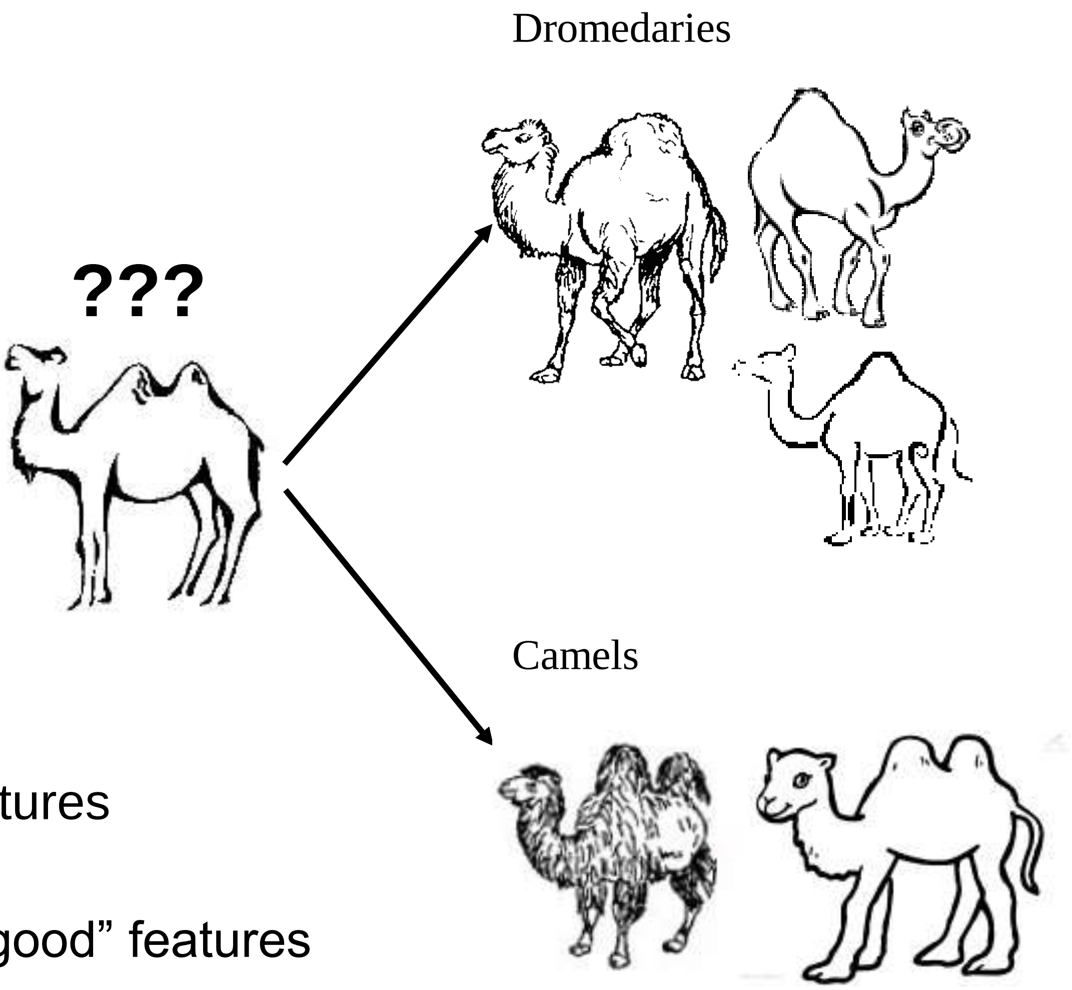


How to get a good classifier to discriminate between images showing Camels or Dromedaries?

DL 1-step-apporach:

Train a CNN

Requirement:
Lots of labeled raw image data



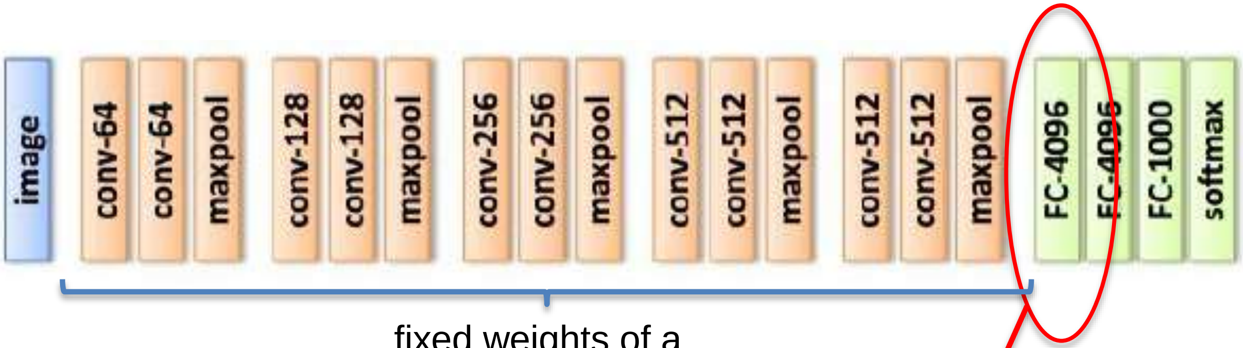
Traditional 2-step-apporach:

1) extract appropriate features
2) train a classifier using these features

Requirement:
quite few labeled data in case of “good” features

remark: here, the number of bumps would be an appropriate feature

Usage of unsupervised CNN features



fixed weights of a
pre-trained CNN

4096 feature

raw data

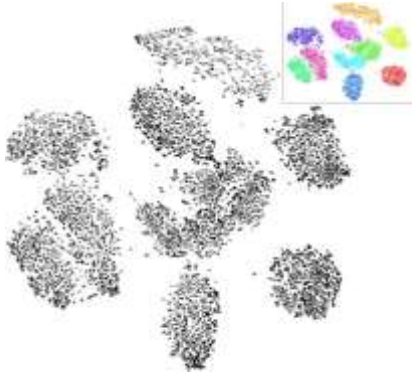
28 x 28=
784 feature

Feature extraction by
CNN, DAE, ladder …

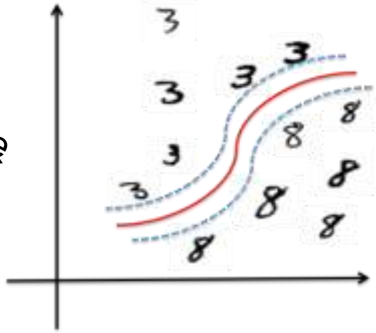
extracted
features

(0.2, 1.5)

visualization
Embedding
interpolation



Input for simple
Classifier trainable
with few instances



Exercise on using unsupervised constructed features for pattern recognition

Use raw pixel features and unsupervised constructed CNN features for MNIST and CIFAR10 data to

1) do some 2D visualization

2) do some pattern recognition

3) train a classifier with only few labeled data
   - aka transfer learning when using unsupervised CNN feature