Introduction to machine learning 2

- Artificial neural nets
- Deep learning
  - Convolutional neural networks
  - Recurrent neural networks
  - Generative models
- Physics example

Iftach Sadeh

August 2025

iftach.sadeh@desy.de

- Largely derived from:
  - University of Toronto CSC411 - Introduction to Machine Learning (Fall 2016). See: http://www.cs.toronto.edu/~urtasun/courses/CSC411_Fall16/CSC411_Fall16.html
  - MIT's introductory course on deep learning - MIT 6.S191 - http://introtodeeplearning.com/
  - Lecture playlist - https://www.youtube.com/playlist?list=PLtBw6njQRU-rwp5__7C0oIVt26ZgjG9NI

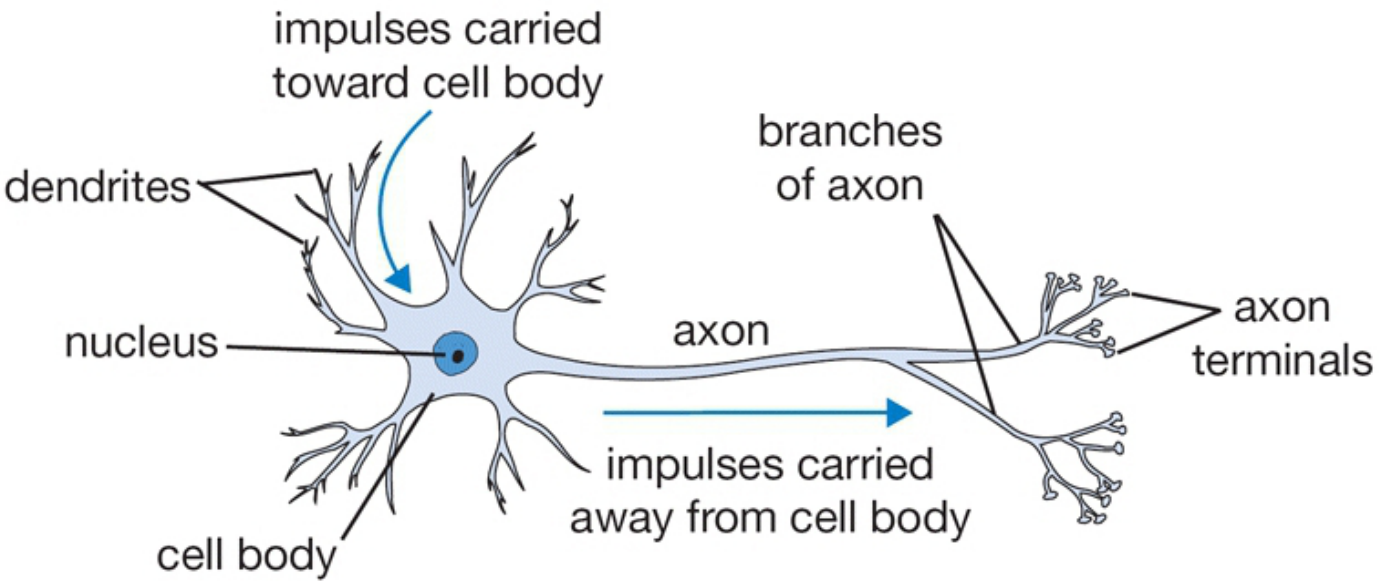
DESY

Artificial neural networks (ANN)

- We would like to construct non-linear discriminative classifiers that utilise functions of input variables
- Use a large number of simpler functions:
  - If these functions are fixed (Gaussian, sigmoid, polynomial basis functions), then optimisation still involves linear combinations of (fixed functions of) the inputs
  - Or we can make these functions depend on additional parameters ➜ need an efficient method of training extra parameters

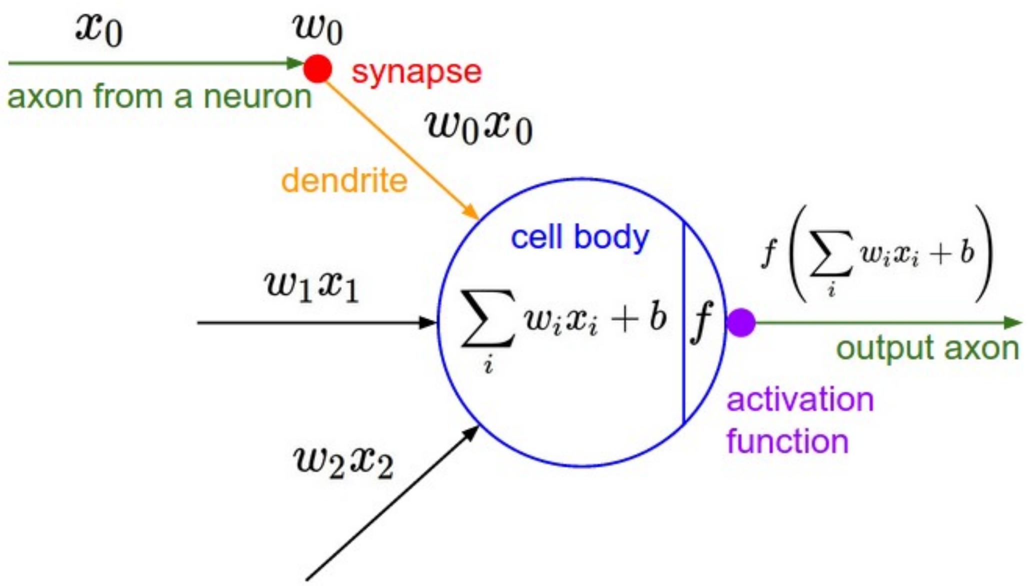
Artificial neural networks (ANN)

- We would like to construct non-linear discriminative classifiers that utilise functions of input variables
- Use a large number of simpler functions:
  - If these functions are fixed (Gaussian, sigmoid, polynomial basis functions), then optimisation still involves linear combinations of (fixed functions of) the inputs
  - Or we can make these functions depend on additional parameters ➜ need an efficient method of training extra parameters

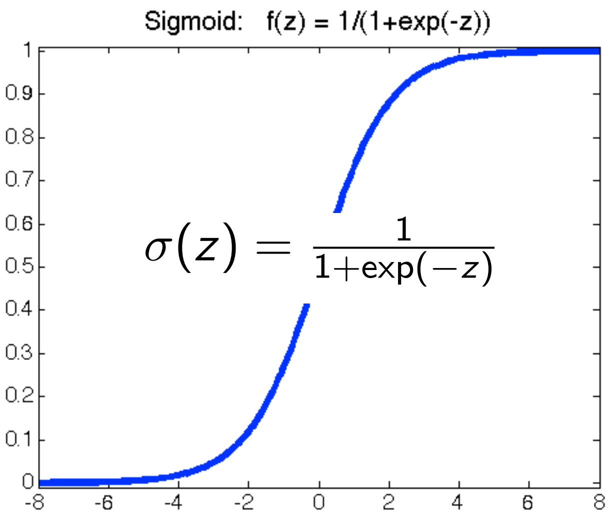


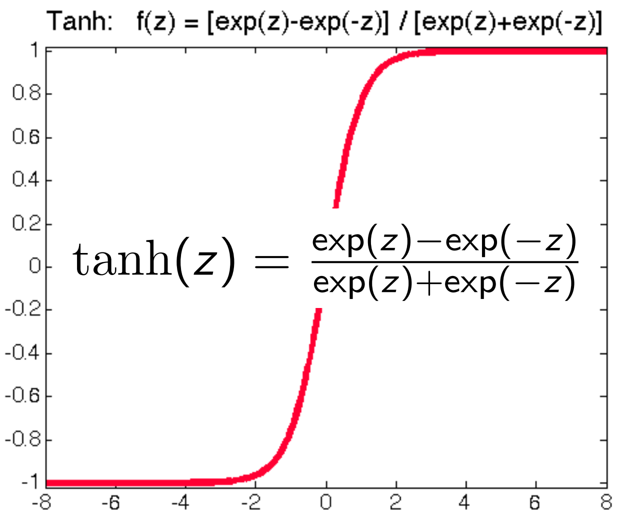
Artificial neural networks (ANN)

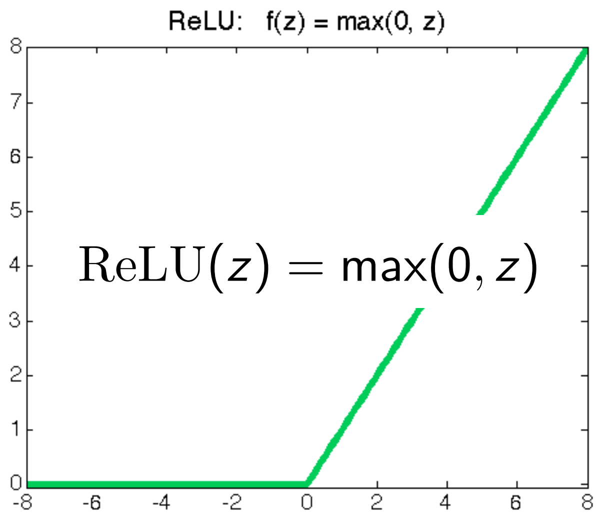
- We would like to construct non-linear discriminative classifiers that utilise functions of input variables
- Use a large number of simpler functions:
  - If these functions are fixed (Gaussian, sigmoid, polynomial basis functions), then optimisation still involves linear combinations of (fixed functions of) the inputs
  - Or we can make these functions depend on additional parameters ➜ need an efficient method of training extra parameters

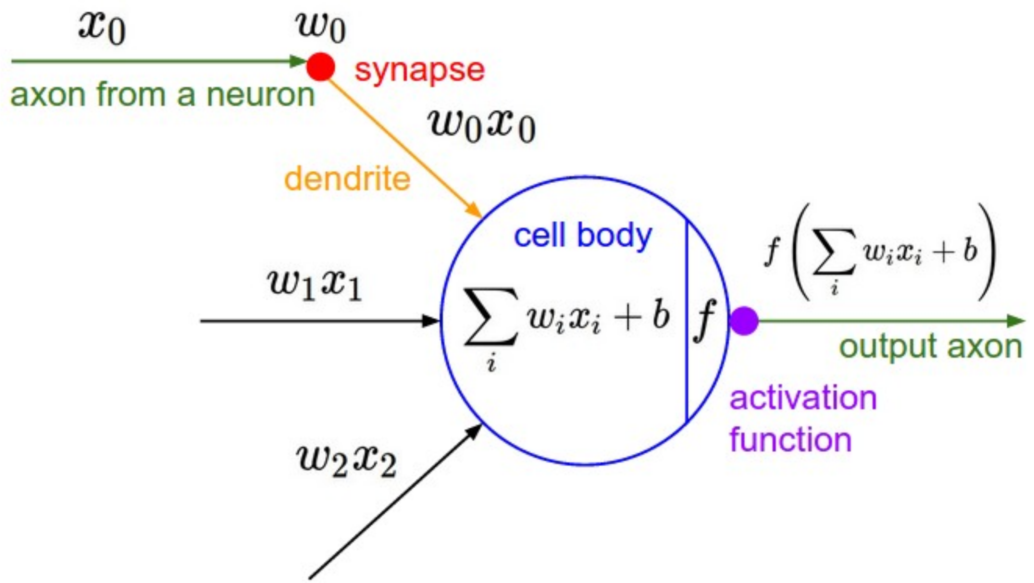


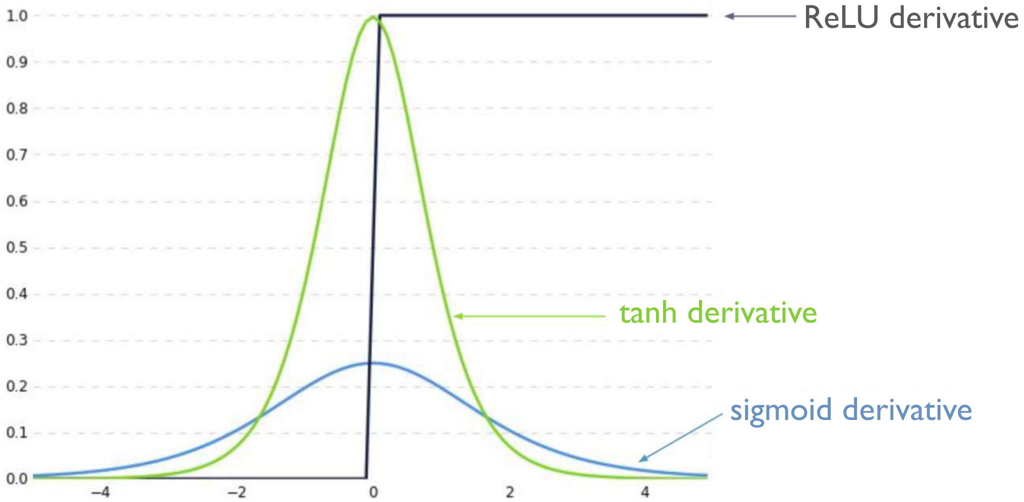
ANN activation functions



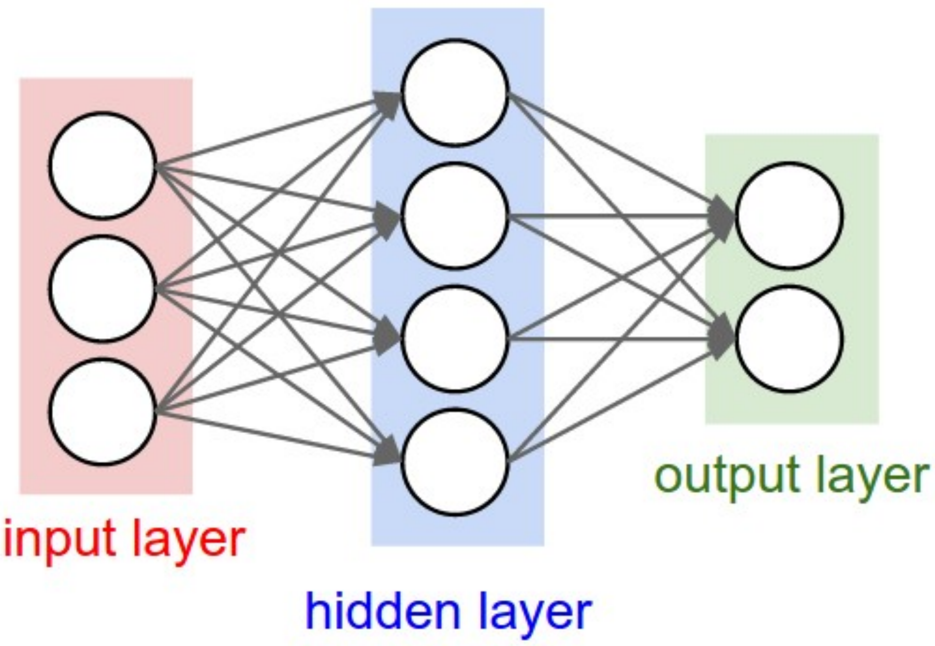


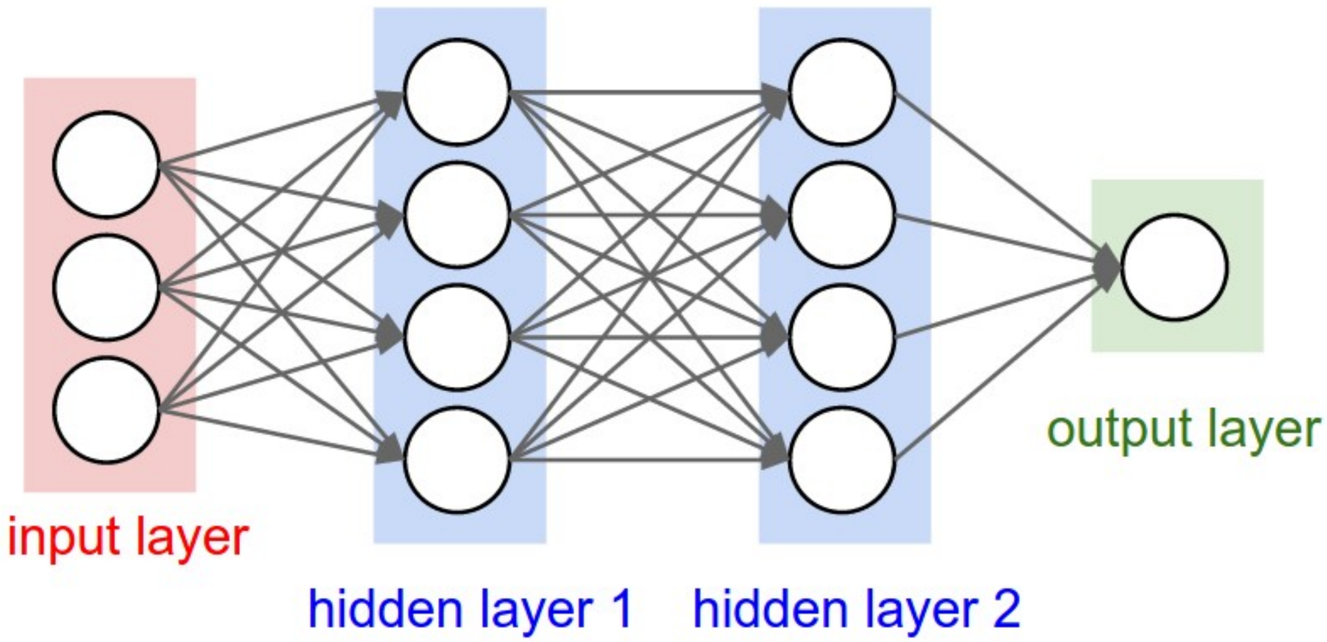






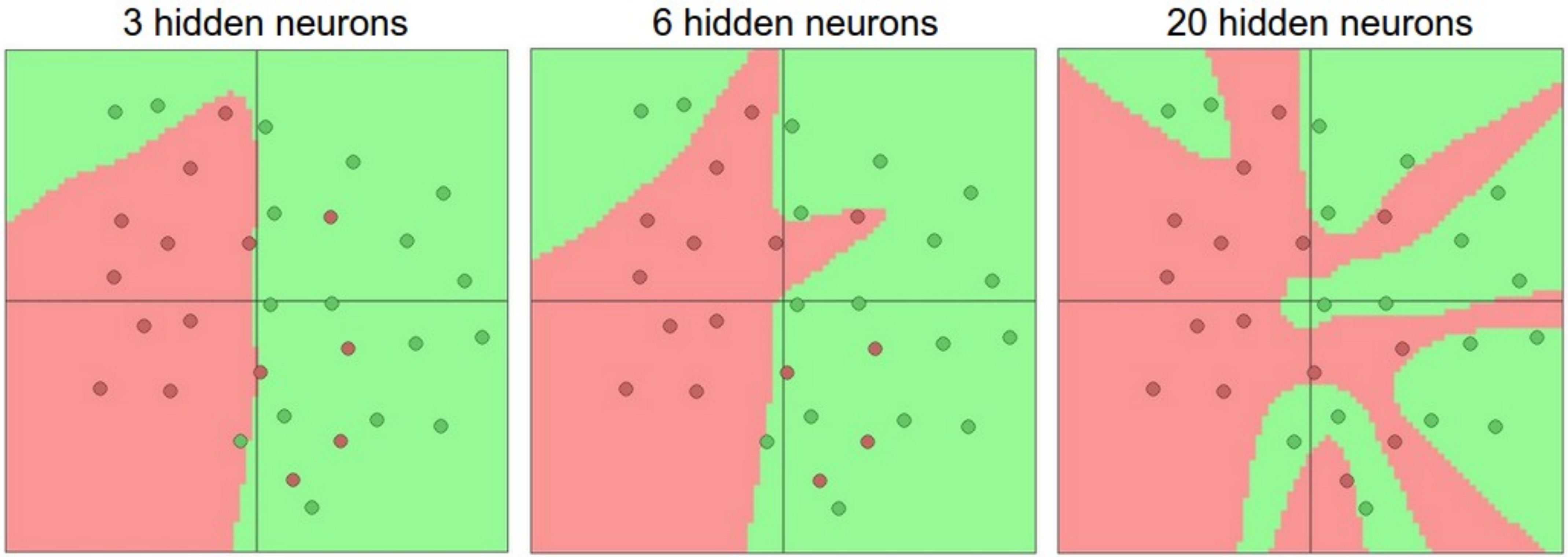
ANN architecture examples





ANN architecture examples

- An ANN with at least one hidden layer is a universal approximator (can represent any function)
- The capacity of the network increases with more hidden units and more hidden layers



ANN activation & loss functions

- **Regression ➜** sigmoid activation & **mean-square error (MSE)** loss function generally works

E: MSE ("error") **t**: True value

$$E = \frac{1}{2}\sum_k (o_k - t_k)^2$$

o: Output of neurone ("after" activation function)

$$o_k(\mathbf{x}) = \frac{1}{1+\exp(-z_k)}$$

$$z_k = w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj}$$

ANN activation & loss functions

- **Regression ➜** sigmoid activation & **mean-square error (MSE)** loss function generally works
- **Classification for a binary** (2-class) problem, **cross-entropy** loss:

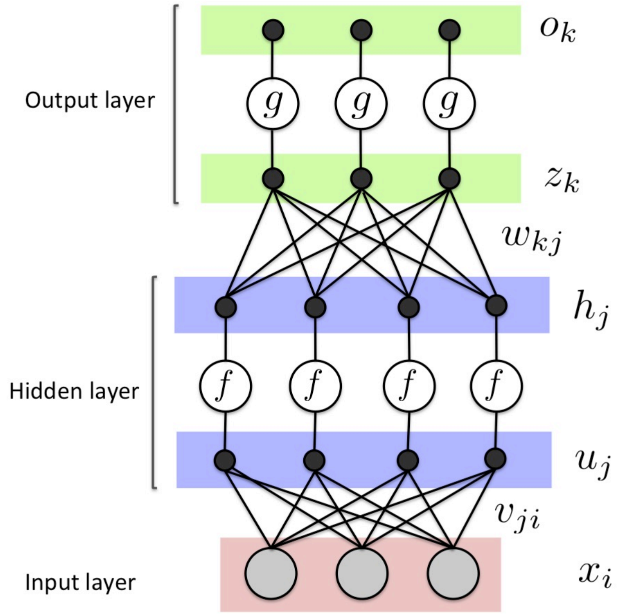
$$E = -\sum_{n=1}^{N} t^{(n)} \log o^{(n)} + (1 - t^{(n)}) \log(1 - o^{(n)})$$

$$o^{(n)} = (1 + \exp(-z^{(n)})^{-1}$$

- **Classification for multi-class** problems:

$$E = -\sum_{n}\sum_{k} t_k^{(n)} \log o_k^{(n)}$$

$$o_k^{(n)} = \frac{\exp(z_k^{(n)})}{\sum_j \exp(z_j^{(n)})}$$



E: MSE ("error") **t**: True value

$$E = \frac{1}{2}\sum_k (o_k - t_k)^2$$

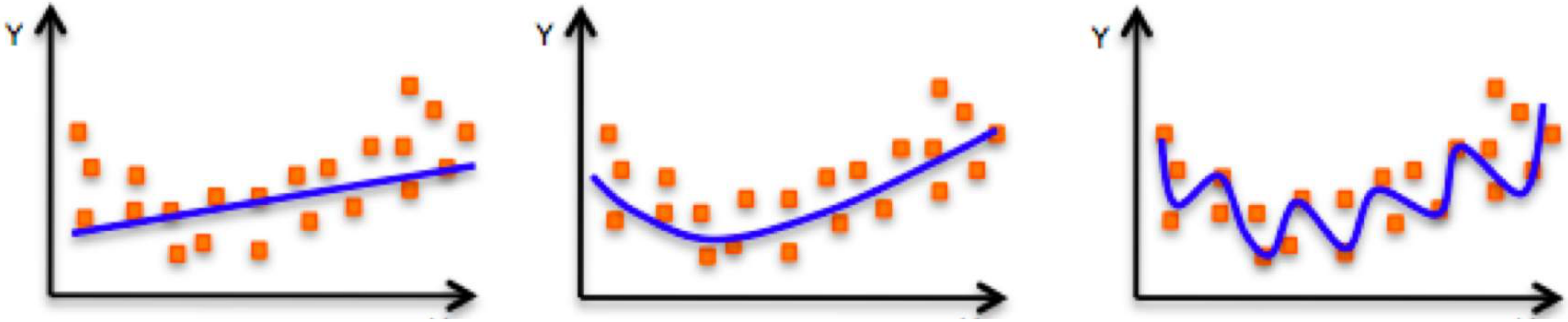
o: Output of neurone ("after" activation function)

$$o_k(\mathbf{x}) = \frac{1}{1 + \exp(-z_k)}$$

$$z_k = w_{k0} + \sum_{j=1}^{J} h_j(\mathbf{x}) w_{kj}$$

Overfitting

- **Problem:**
  - The training data contains information about the true patterns in the mapping from input to output. But it also contains statistical & systematic noise
    - The target values may be unreliable
    - There are statistical fluctuations ➜ there will be accidental patterns
  - ➜ When we fit the model, we end up predicting both true and spurious properties



Underfitting
Model does not have capacity to fully learn the data

Ideal fit

Overfitting
Too complex, extra parameters, does not generalize well

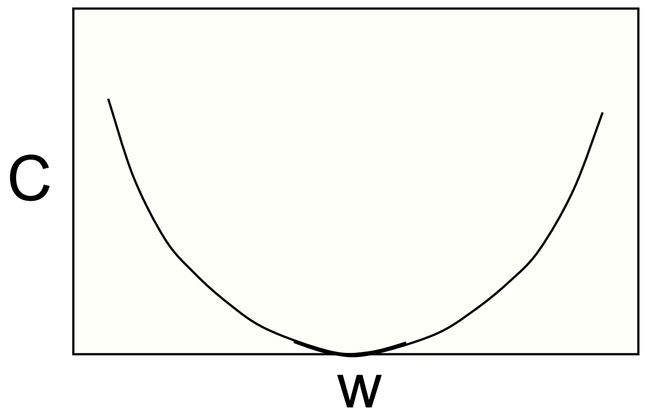
Overfitting

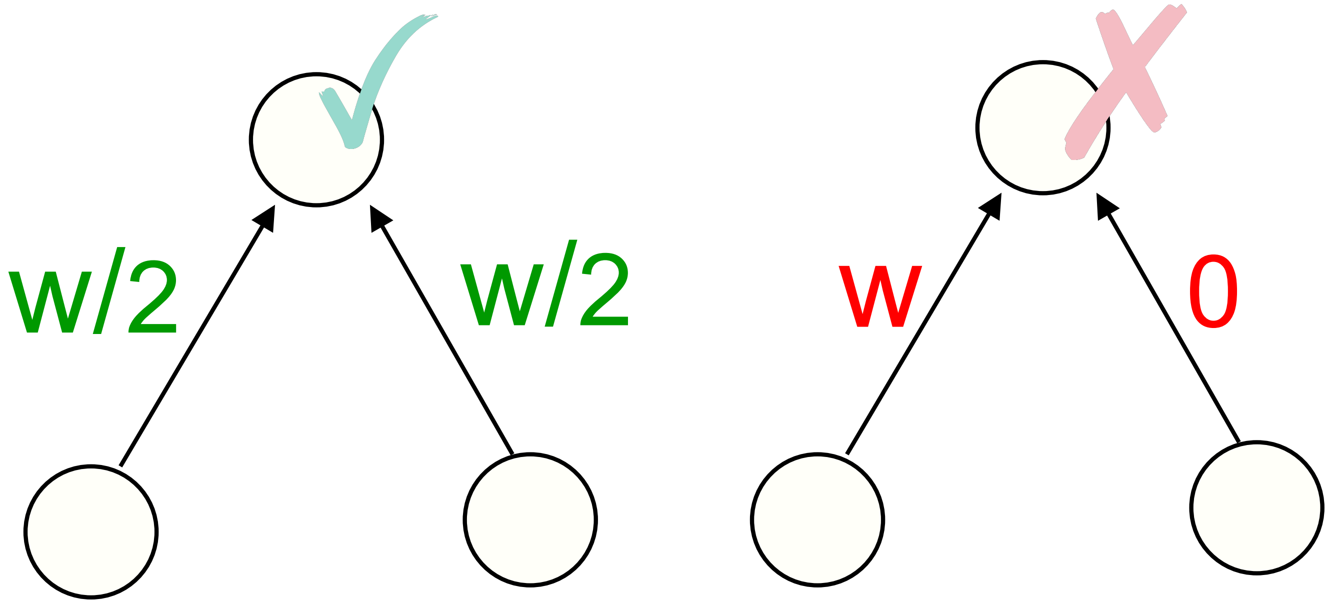
- **Problem:**
  - The training data contains information about the true patterns in the mapping from input to output. But it also contains statistical & systematic noise
    - The target values may be unreliable
    - There are statistical fluctuations ➜ there will be accidental patterns
  - ➜ When we fit the model, we end up predicting both true and spurious properties
- **Solution:**
  - Use a model that has appropriate complexity
    - Enough to model the true regularities
    - Not enough to also model the spurious regularities (assuming they are weaker)
  - Standard ways to limit the capacity of ANNs
    - Limit the number of hidden units
    - Limit the size of the weights
    - Stop the learning before it begins to overfit

Limit the size of the weights - weight decay

- Add an extra term (C) to the cost function that penalises (squared) weights
- ➜ Keeps weights small unless they have big error derivatives
  - Improves **generalisation**.
  - **Prevent fitting fluctuations**.
  - **Smoother** model ➜ the output changes more slowly as the input changes.

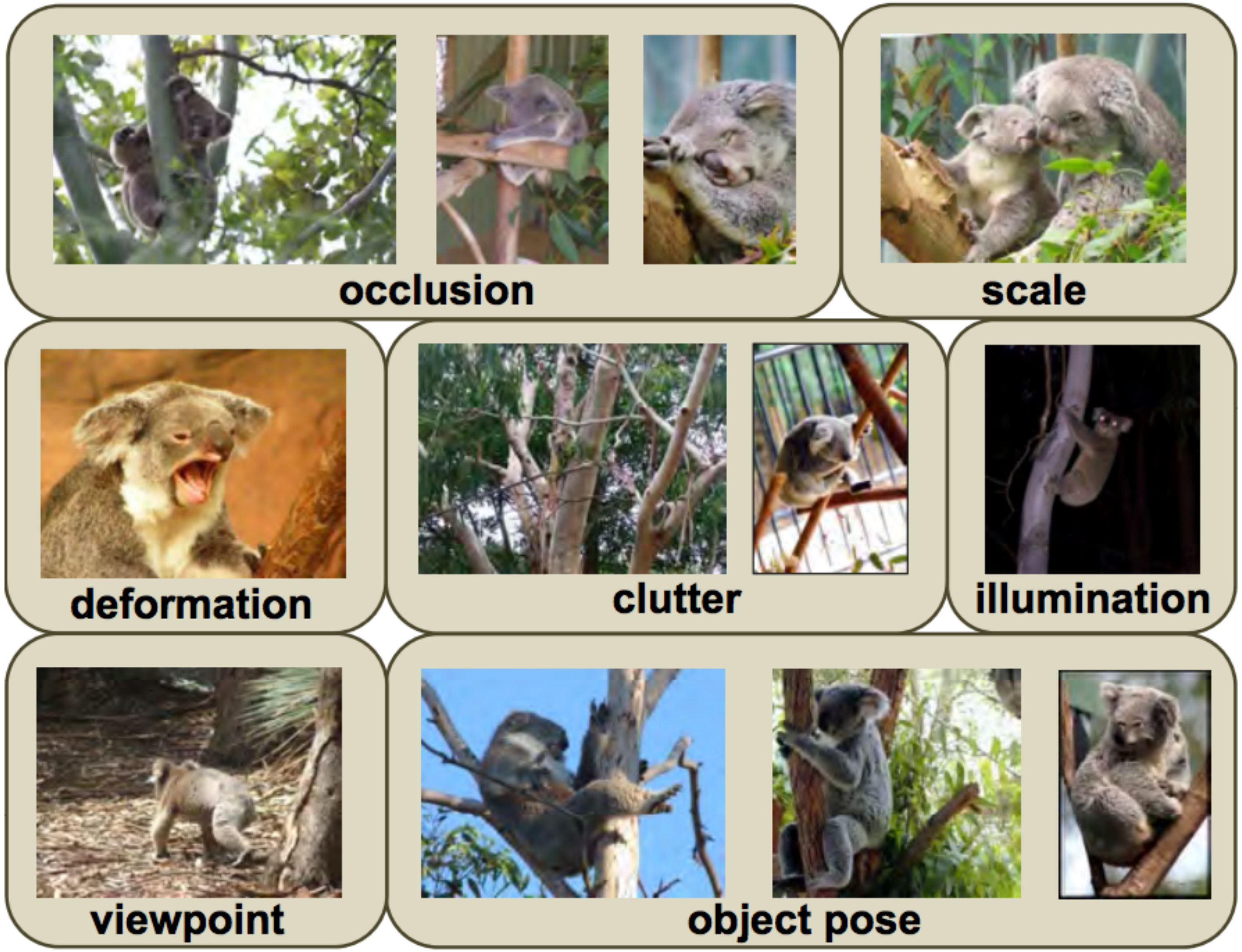
$$C = \ell + \frac{\lambda}{2} \sum_i w_i^2$$





Deep learning

- Difficult scene conditions



Deep learning

- Difficult scene conditions
- Lots of variation within a given class






Deep learning

- Difficult scene conditions
- Lots of variation within a given class
- Huge number of classes

Deep learning

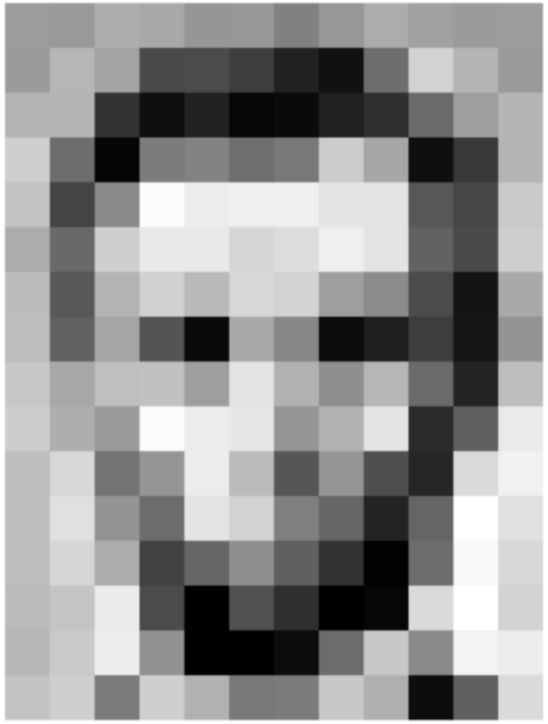
- **Difficulties**:
  - Segmentation: real scenes are cluttered
  - Invariances: many variations do not affect nominal shape
  - Deformations: natural shape classes allow variations (faces, letters, chairs)
  - A huge amount of computation

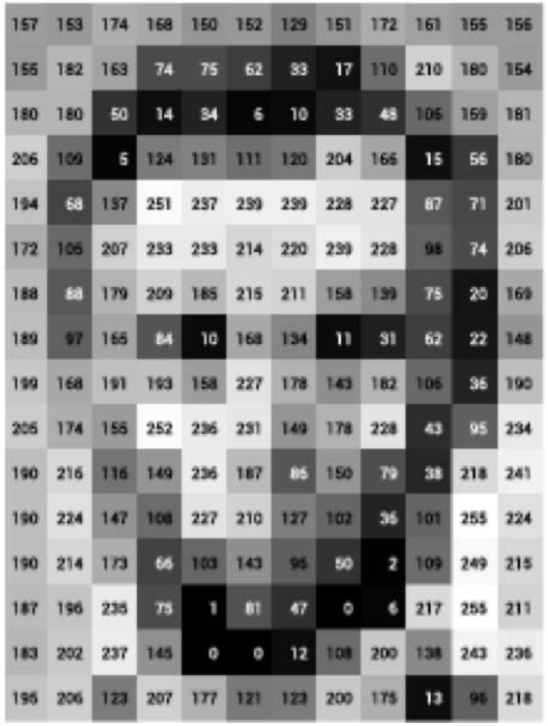
UofT - CSC411 (2016) https://www.youtube.com/watch?v=uxUTc1N75Kg https://www.youtube.com/watch?v=PoiHAiDHgDs https://www.youtube.com/watch?v=XbuaUYMwOP8 https://www.youtube.com/watch?v=zZnTy3O7tJs

Computer vision

Deepfake Superman moustache disaster of 2018

Computer vision





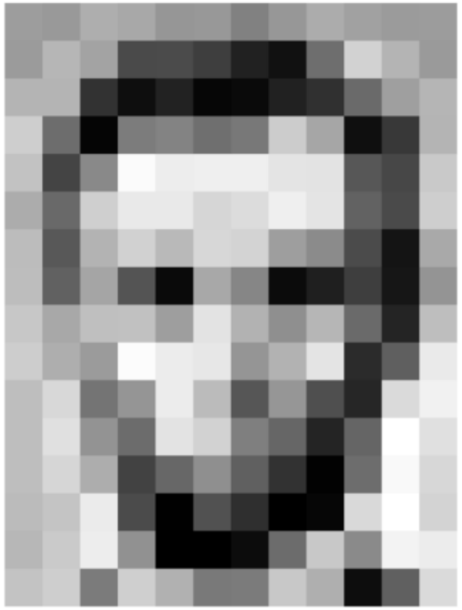
What the computer sees

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

MIT - 6.S191 (2019)

Computer vision

- **Regression**: output variable takes continuous value
- **Classification**: output variable takes class label ➜ can produce probability of belonging to a particular class



Input Image

157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

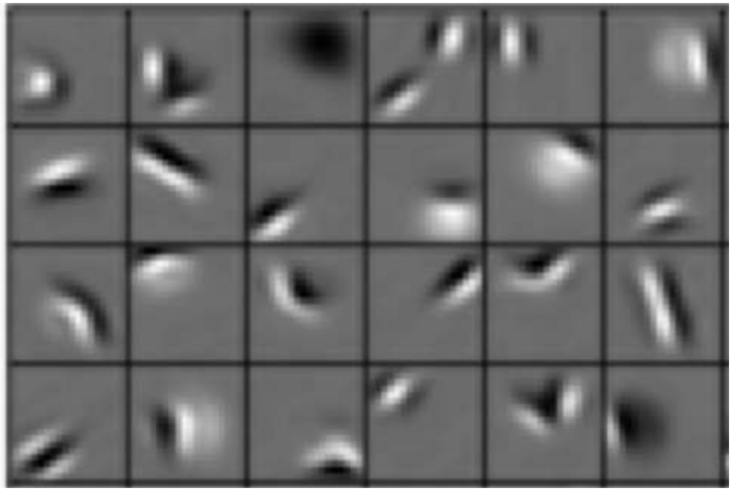
Pixel Representation

classification

$$\begin{array}{l} \text{Lincoln} \\ \text{Washington} \\ \text{Jefferson} \\ \text{Obama} \end{array} \begin{bmatrix} 0.8 \\ 0.1 \\ 0.05 \\ 0.05 \end{bmatrix}$$

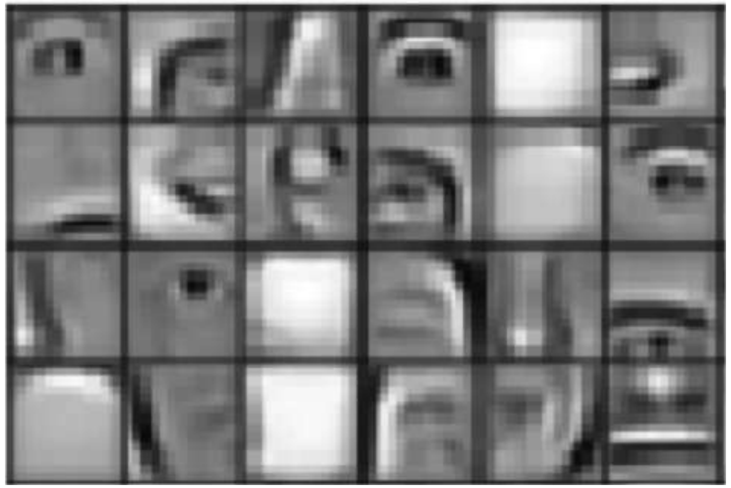
Hierarchy of features

Low level features



Edges, dark spots

Mid level features



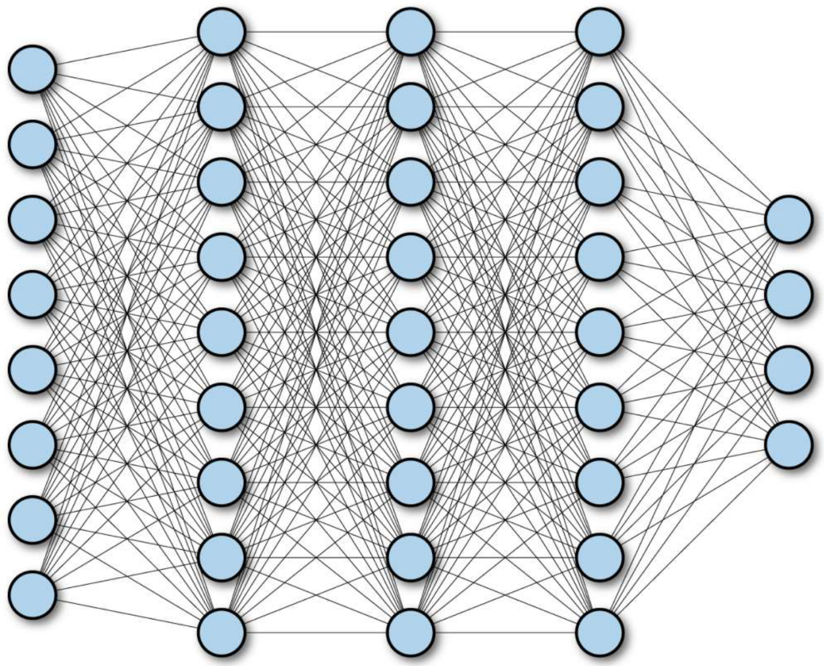
Eyes, ears, nose

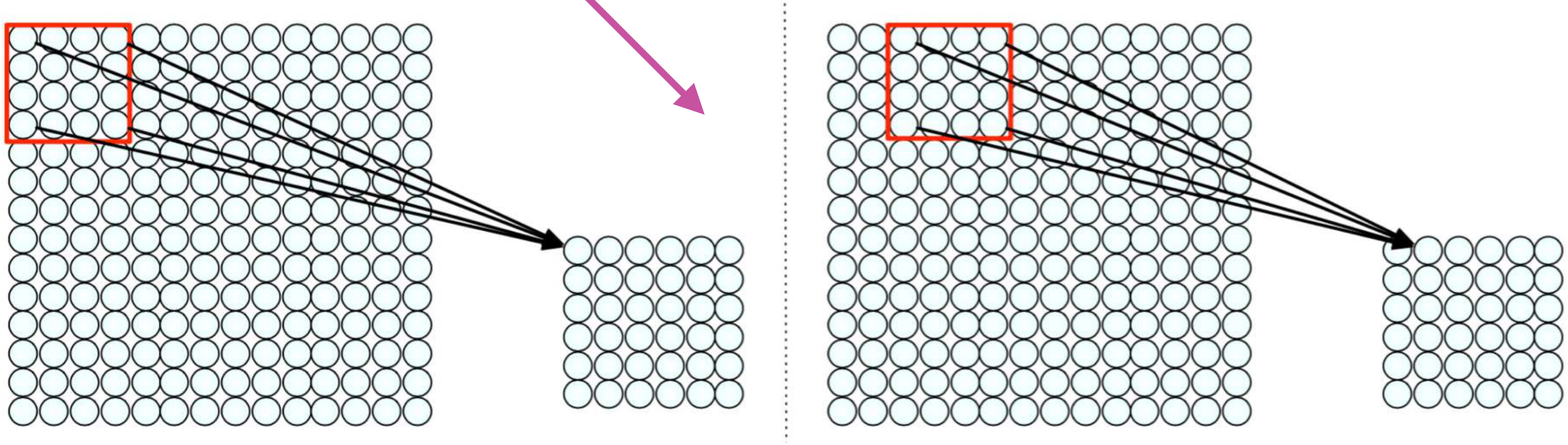
High level features

Facial structure

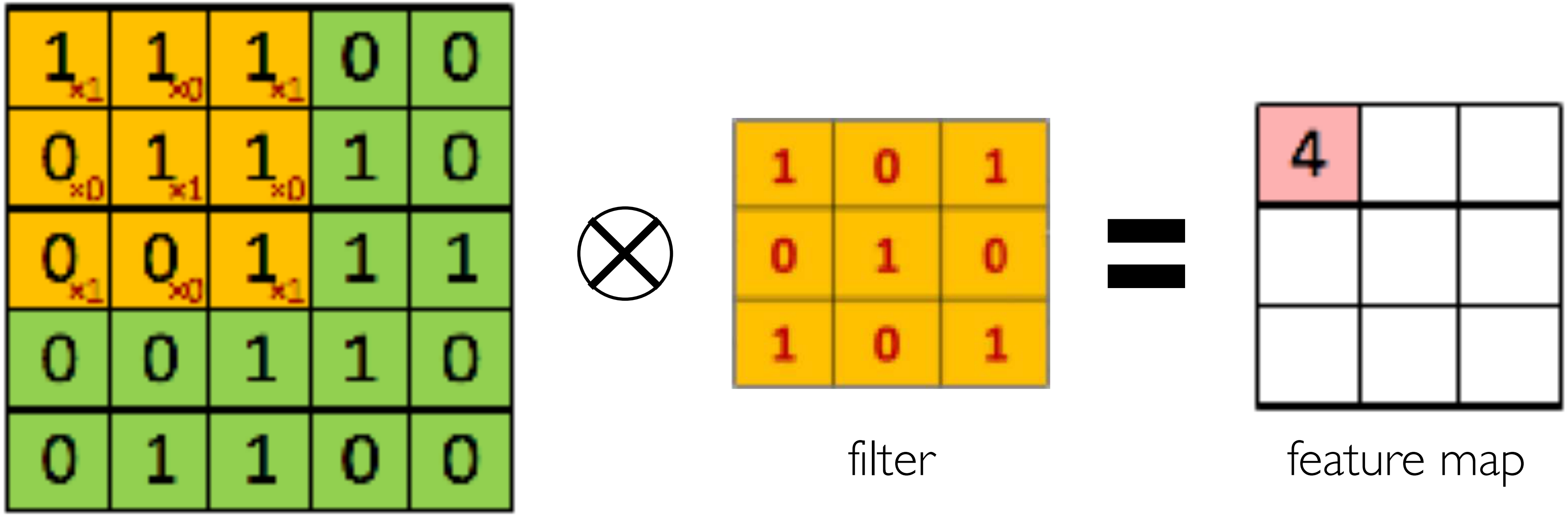
Fully connected layer

- Apply a set of weights (a filter) to extract **local features**
- Use **multiple filters** to extract different features
- **Spatially share** parameters of each filter
- Example:
  - Filter of size 4x4 : 16 different weights
  - Apply same filter to 4x4 patches (convolution) in input
  - Shift by 2 pixels for next patch

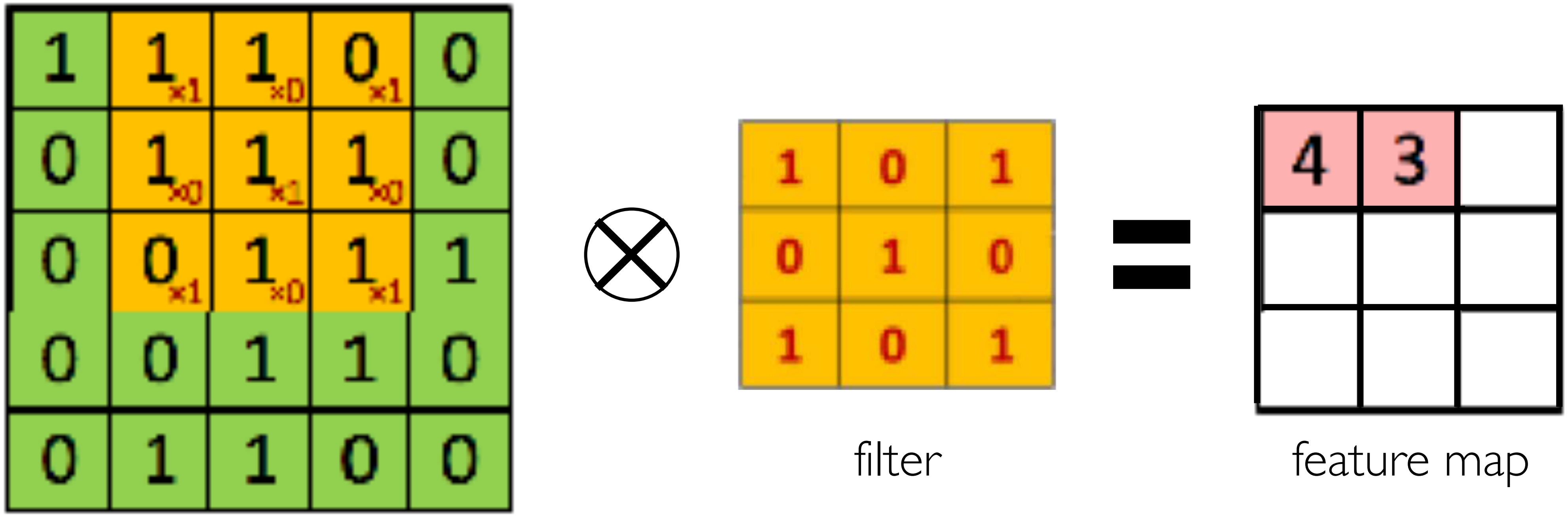




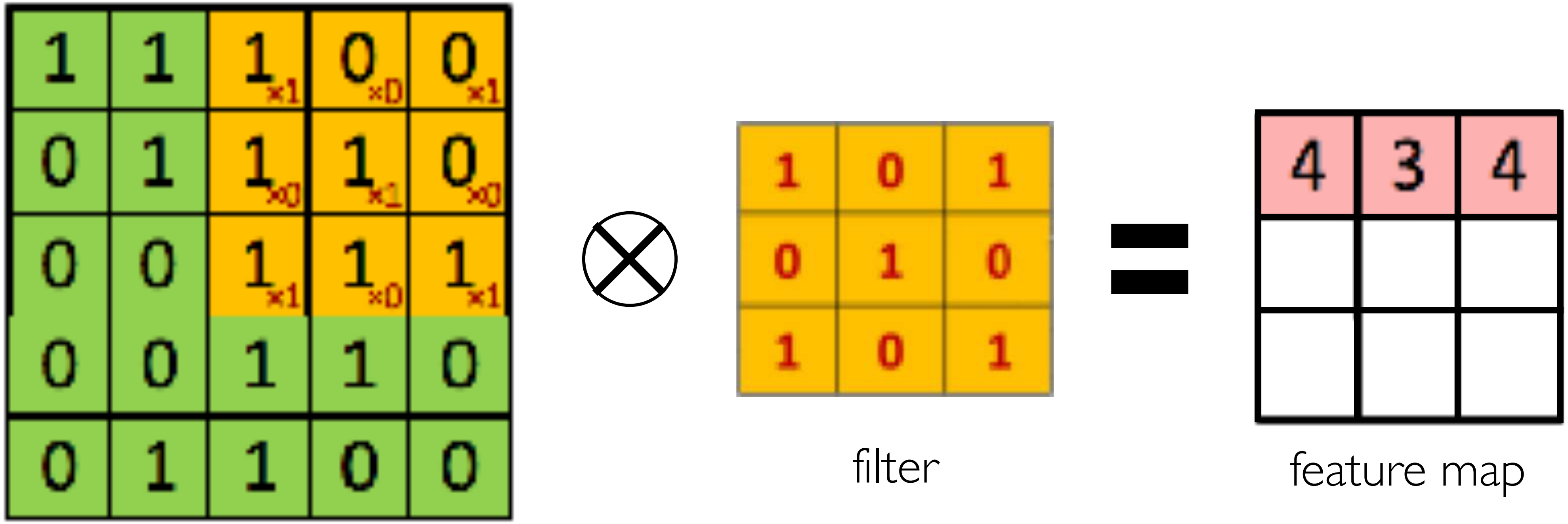
Convolution



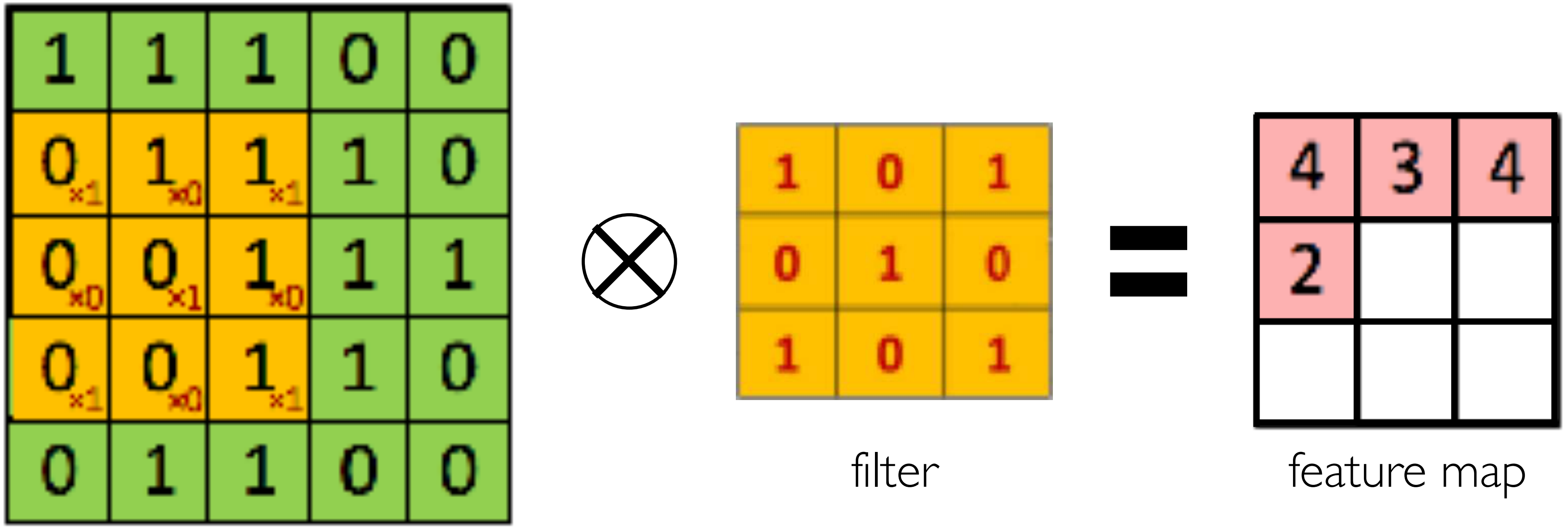
Convolution



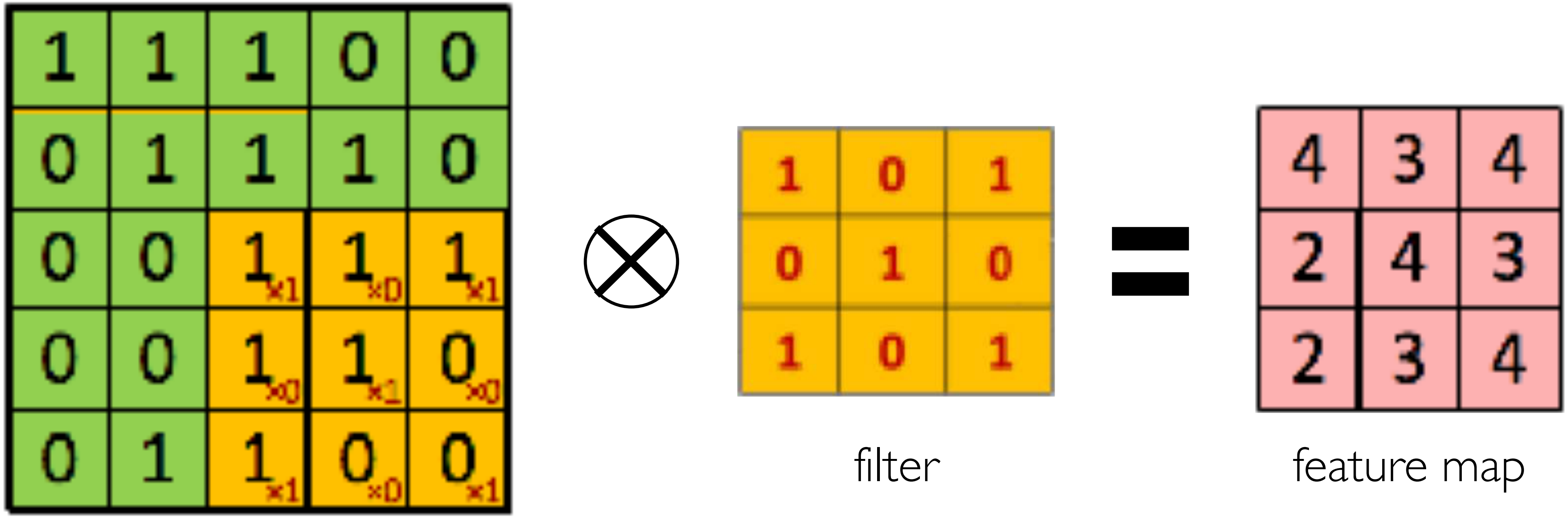
Convolution



Convolution

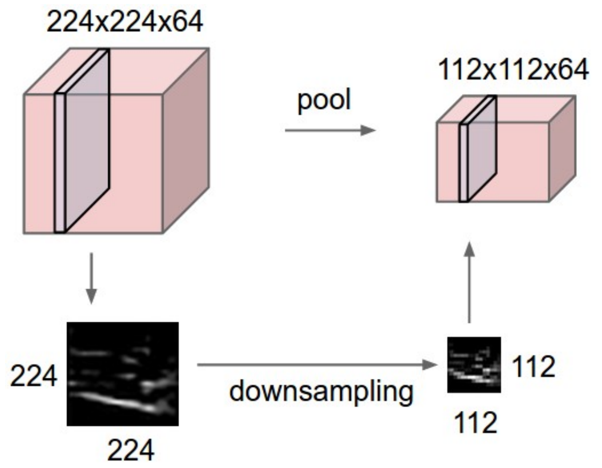


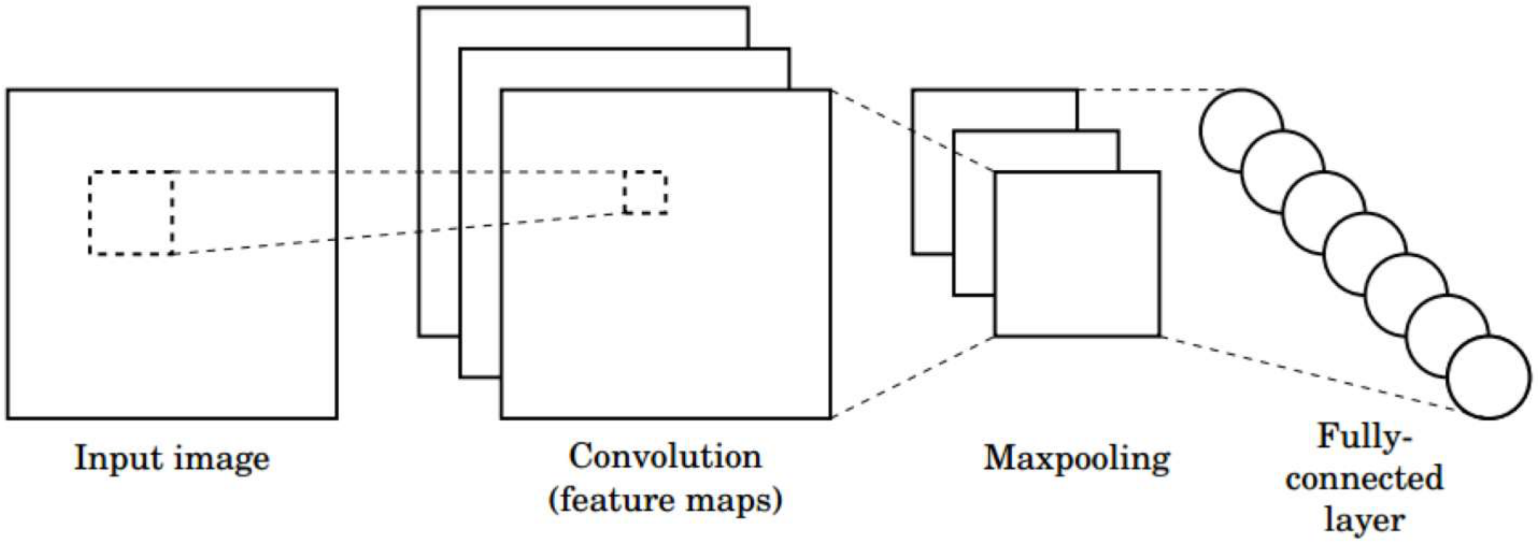
Convolution



Convolution

- **Convolution**: Apply filters with learned weights to generate feature maps
- **Non-linearity**: Often ReLU.
- **Pooling**: Downsampling operation on each feature map





x

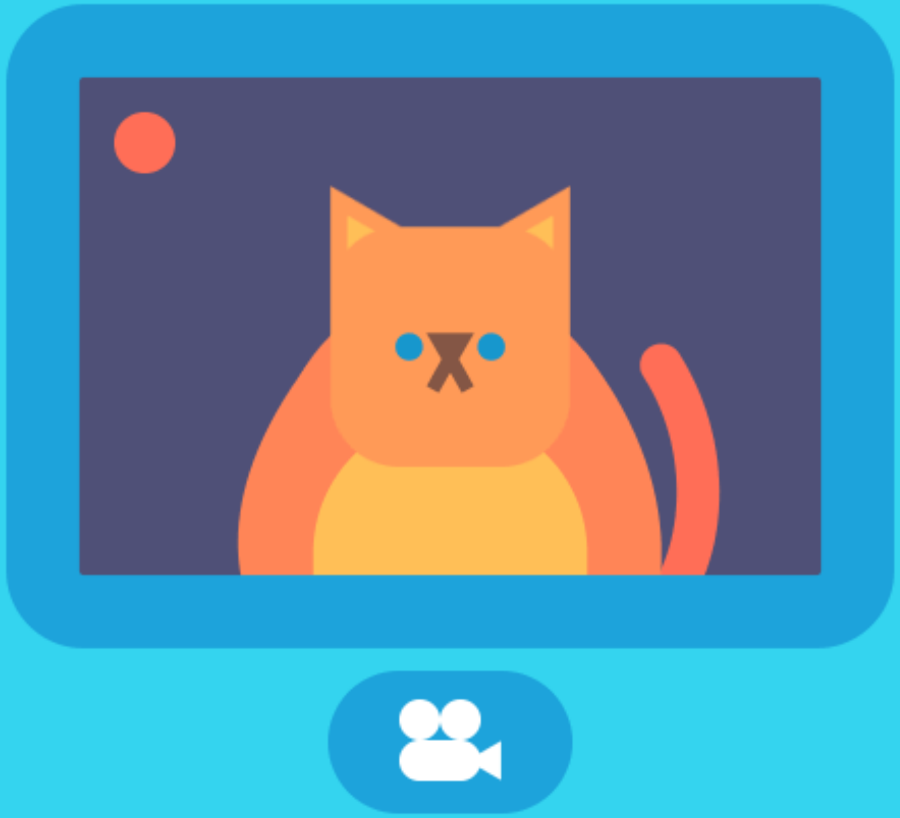
1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

y

Max-pooling
➜ reduce dim ;
invariance to small-
scale translations

6	8
3	4

Sequences / time-series analysis



Sequences / time-series analysis

"This morning I took my cat for a walk."

given these two words — predict the next word

- **One-hot encoding** maps words to eigenvalues:

$$[\,1\;0\;0\;0\;0\;0\;1\;0\;0\;0\,]$$

for — a

↓

prediction

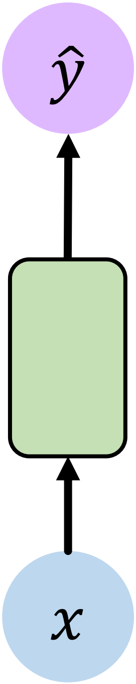
Sequences / time-series analysis

- Information from the **distant past** is needed in order to make predictions…

"**France** is where I grew up, but I now live in Boston. I speak fluent ____."

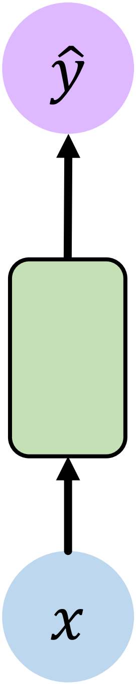
- In general, need to:
  - Handle **variable-length** sequences
  - Track **long-term** trends / dependencies
  - Maintain information about the **order**
  - **Share parameters** across the sequence

Recurrent neural networks

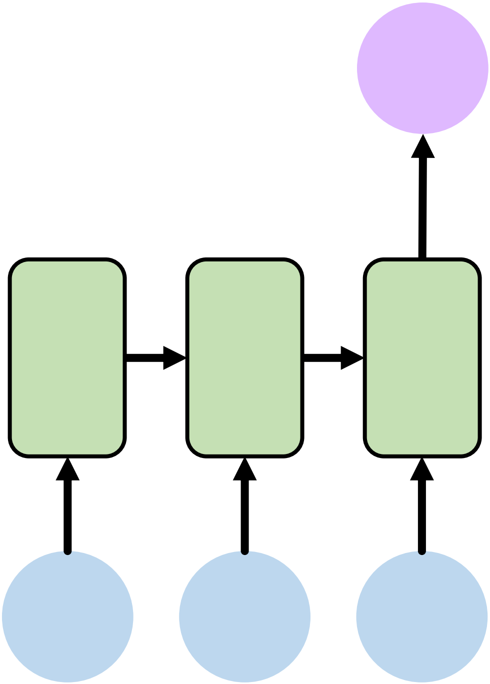


One to One
"Vanilla" neural network

Recurrent neural networks

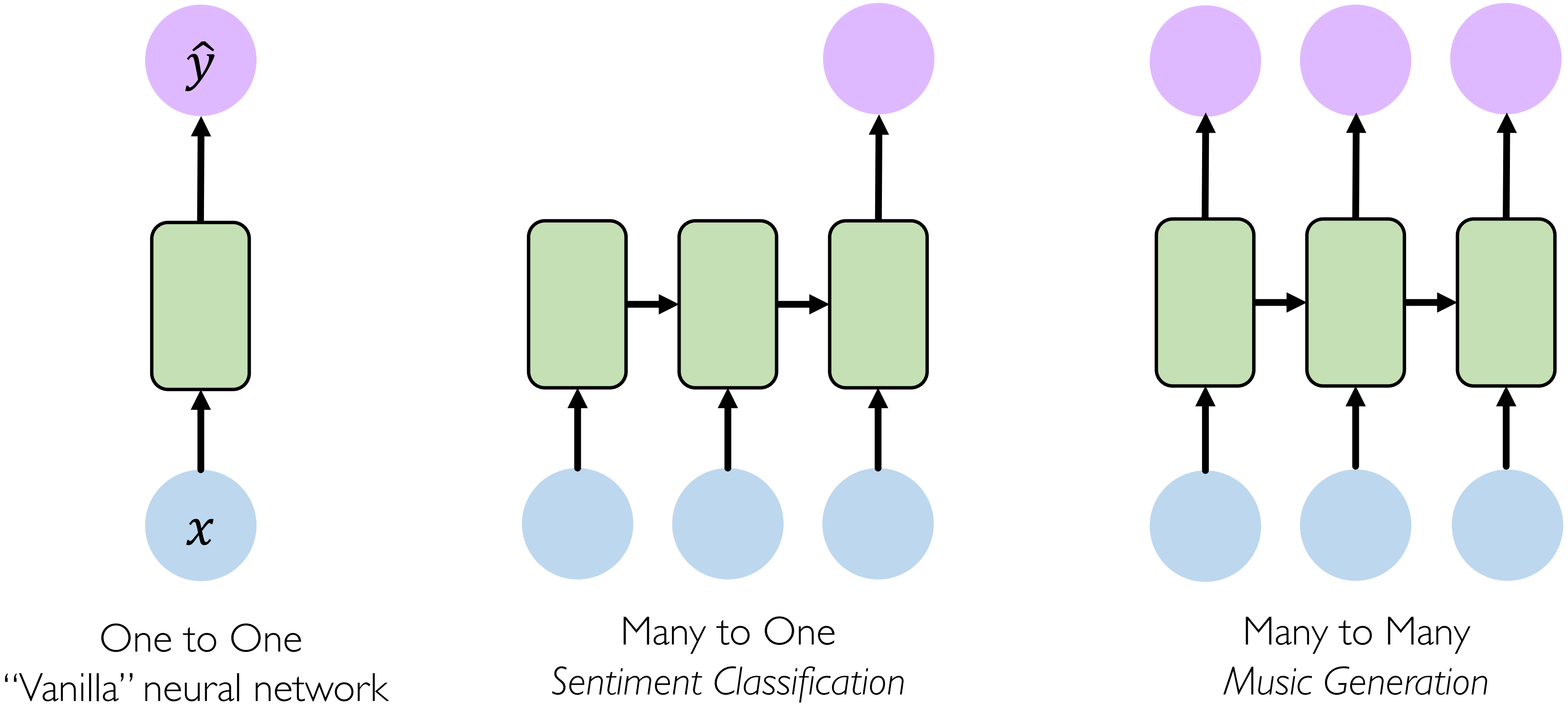


One to One
"Vanilla" neural network

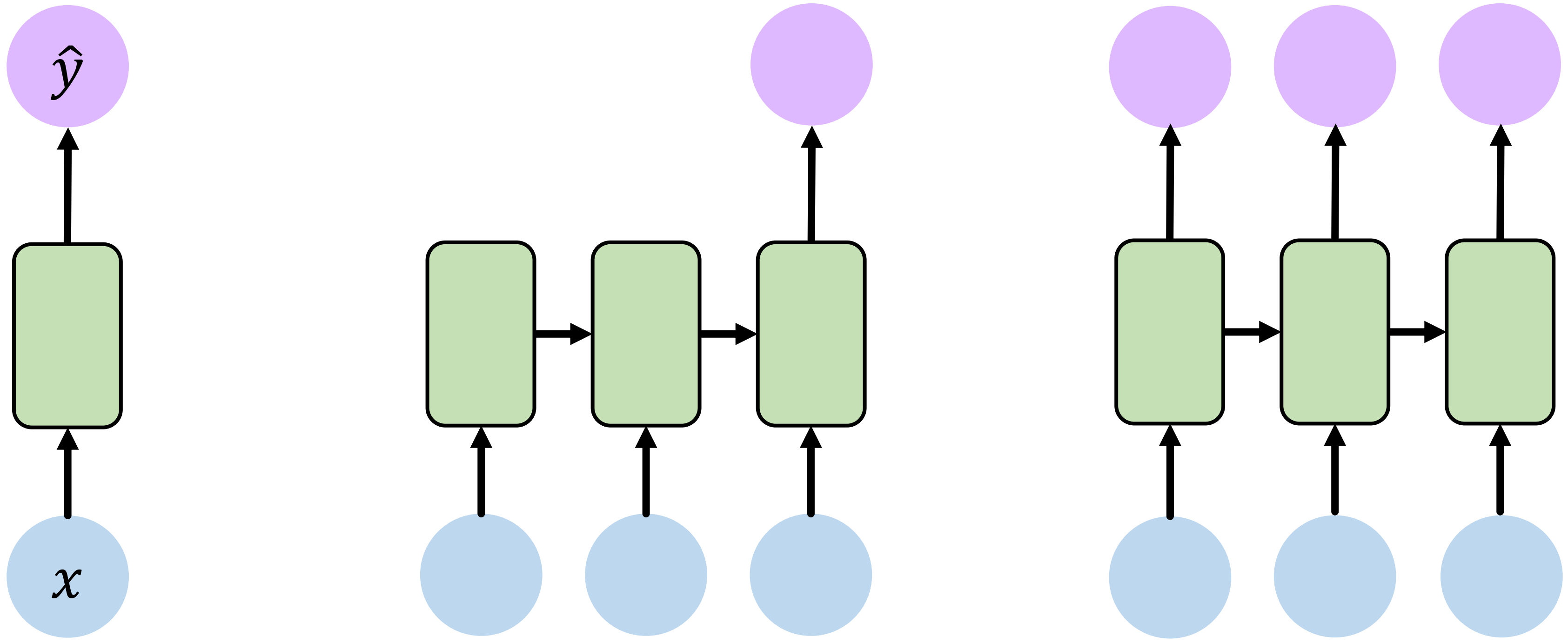


Many to One
Sentiment Classification

Recurrent neural networks



Recurrent neural networks



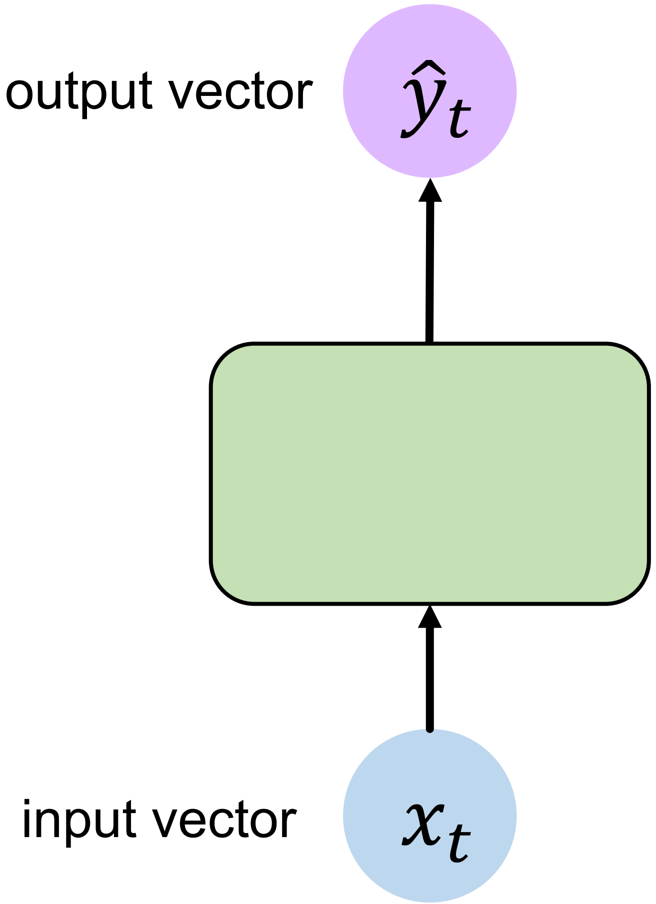
One to One
"Vanilla" neural network

Many to One
Sentiment Classification

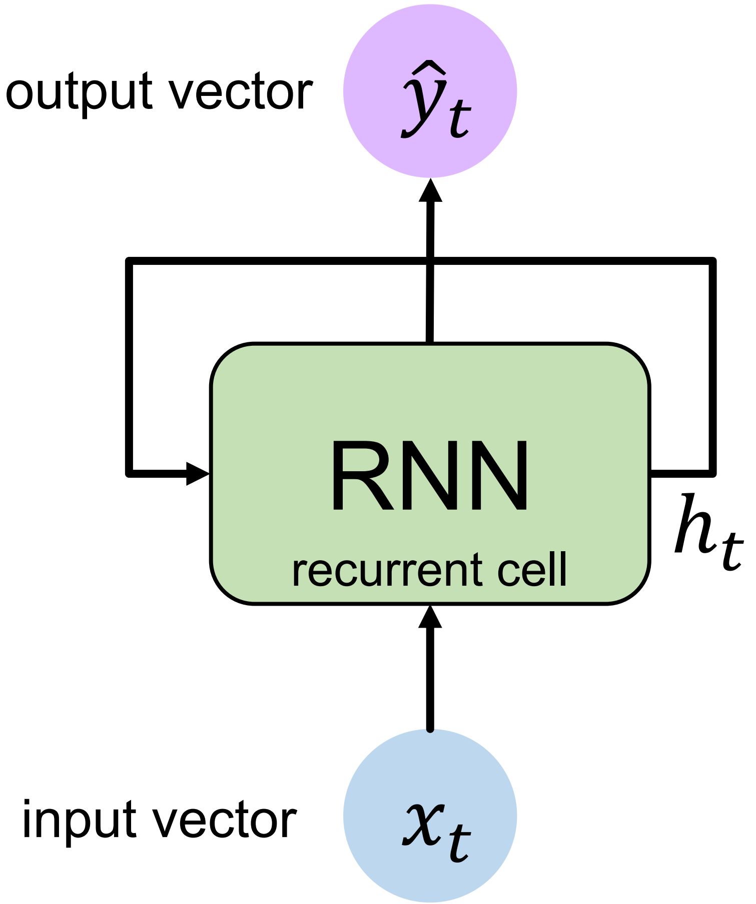
Many to Many
Music Generation

… and many other architectures and applications

Recurrent neural networks



Recurrent neural networks



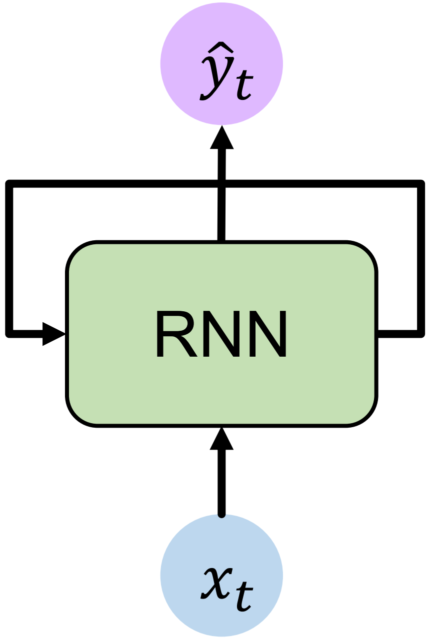
Apply a **recurrence relation** at every time step to process a sequence:

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

new state — function parameterized by W — old state — input vector at time step *t*

Note: the same function and set of parameters are used at every time step

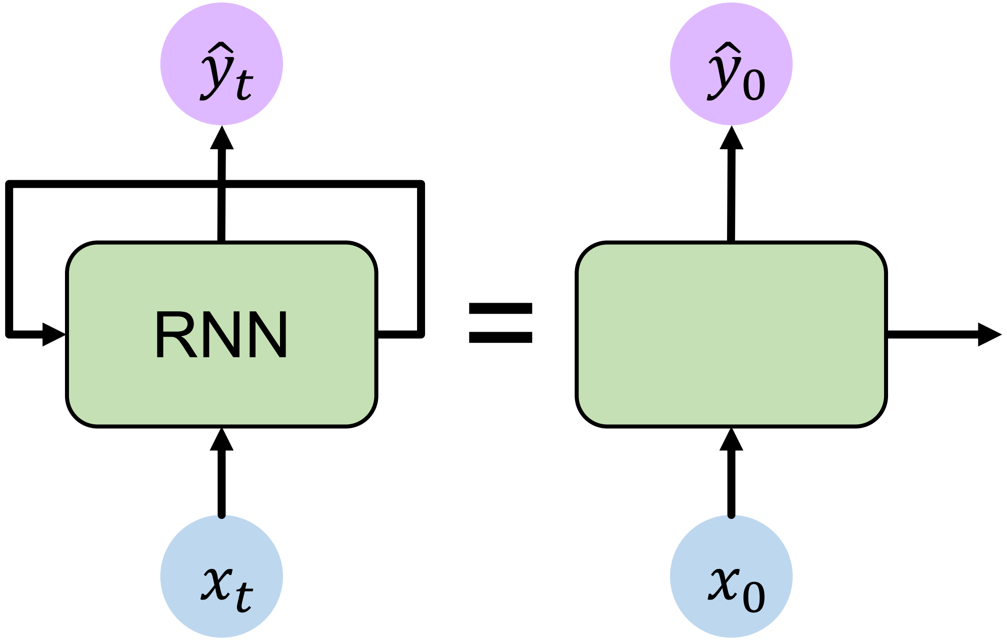
Recurrent neural networks



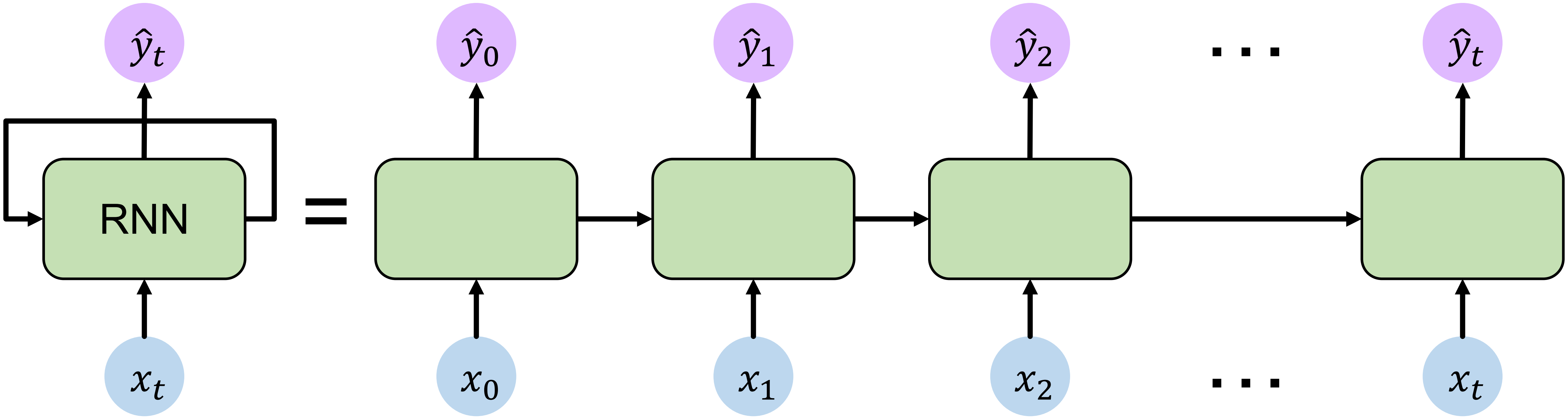
=

Represent as computational graph unrolled across time

Recurrent neural networks

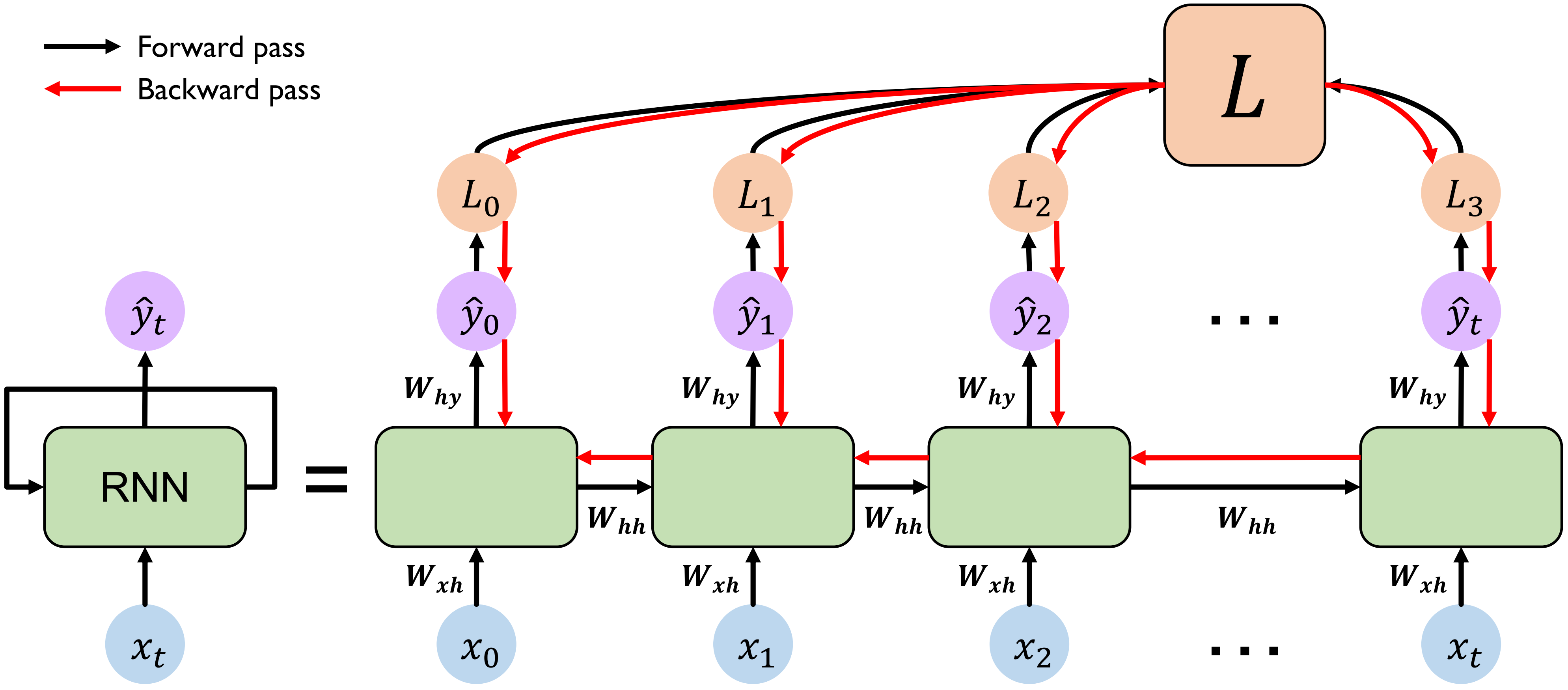


Recurrent neural networks



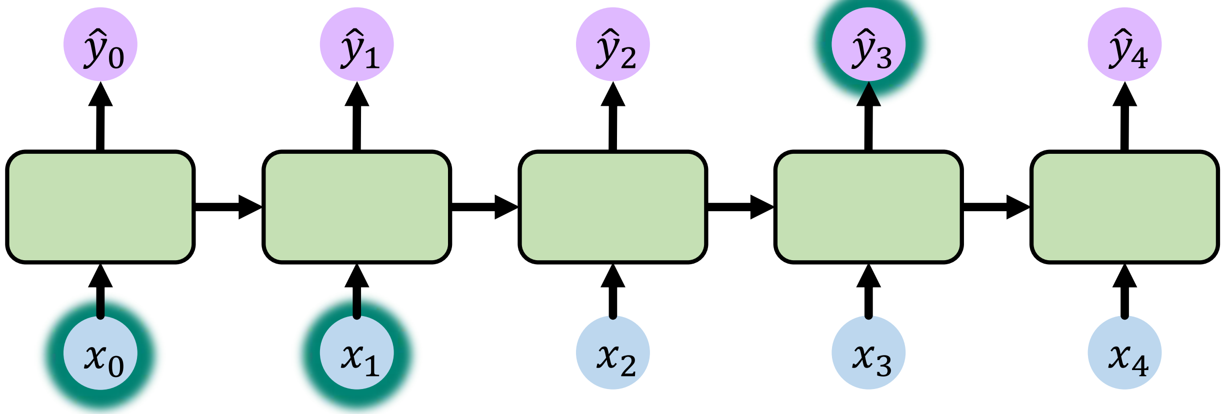
Back-propagation through time

- Forward: take derivative of loss for each parameter
- Backward: shift params to minimise loss

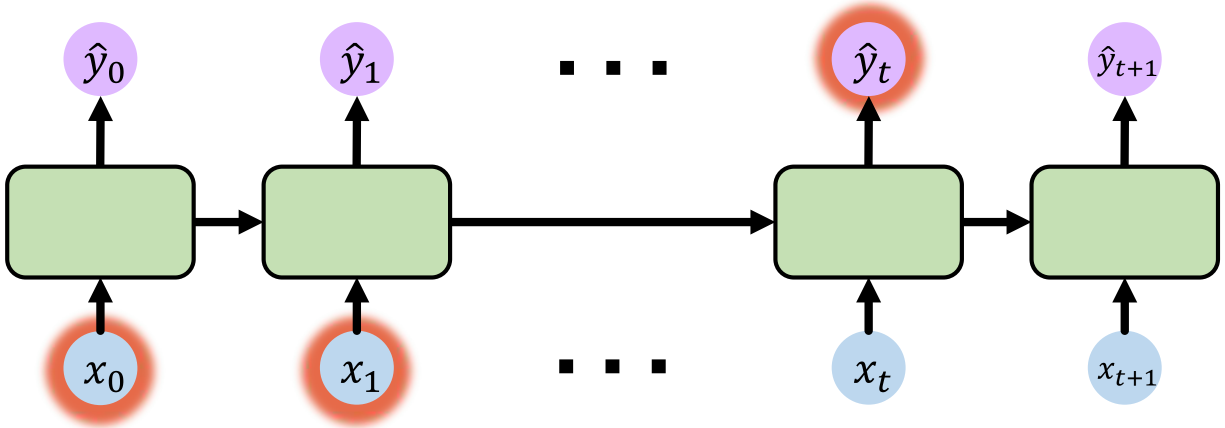


Vanishing gradient problem

"The clouds are in the ___"

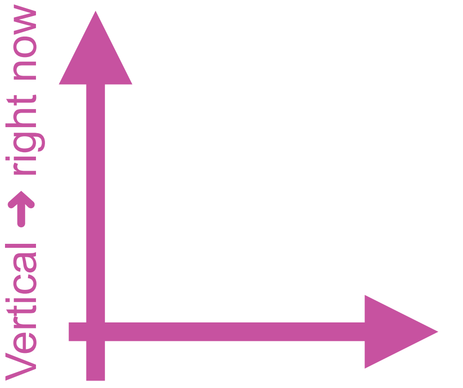


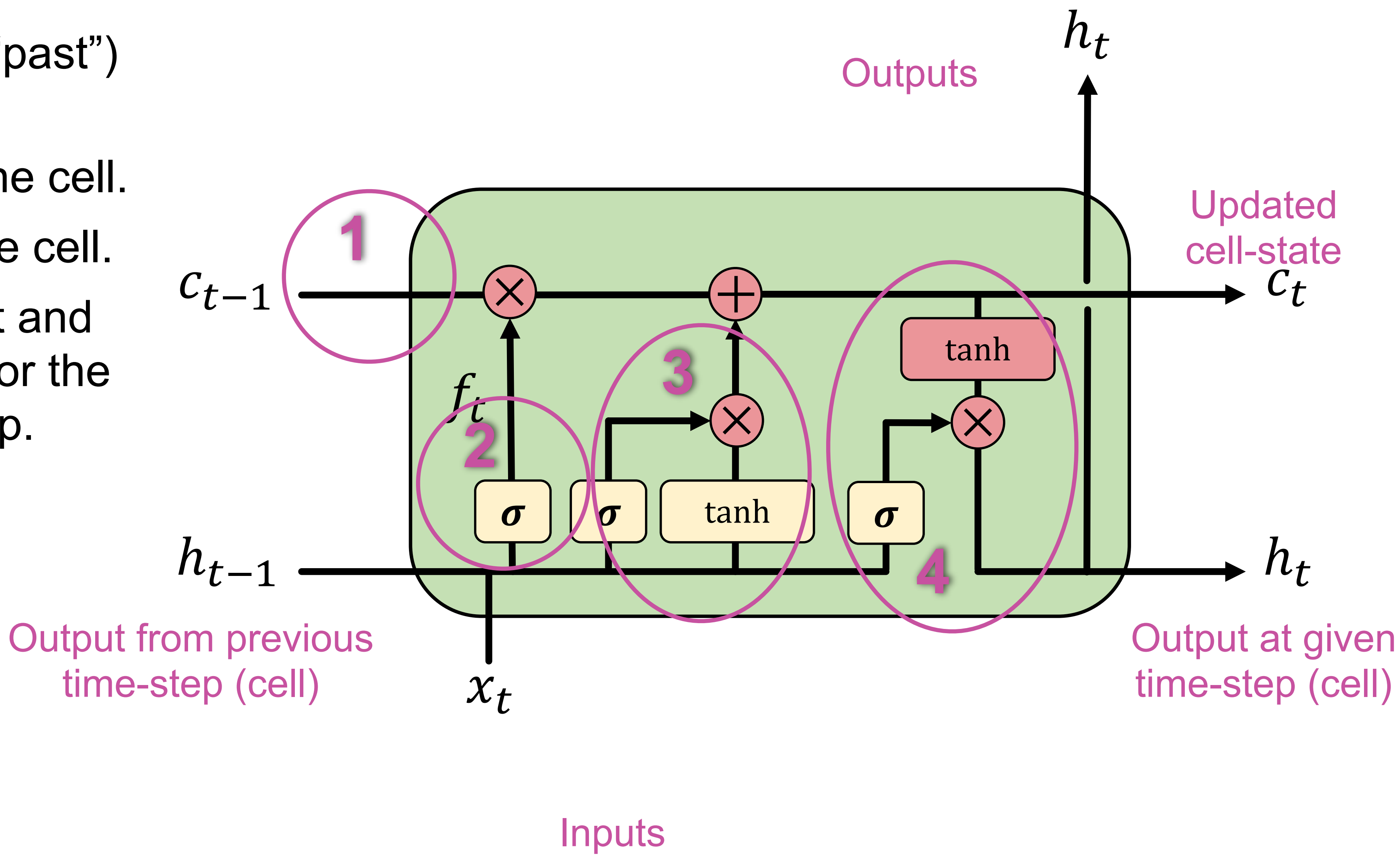
"I grew up in France, … and I I speak fluent___ "



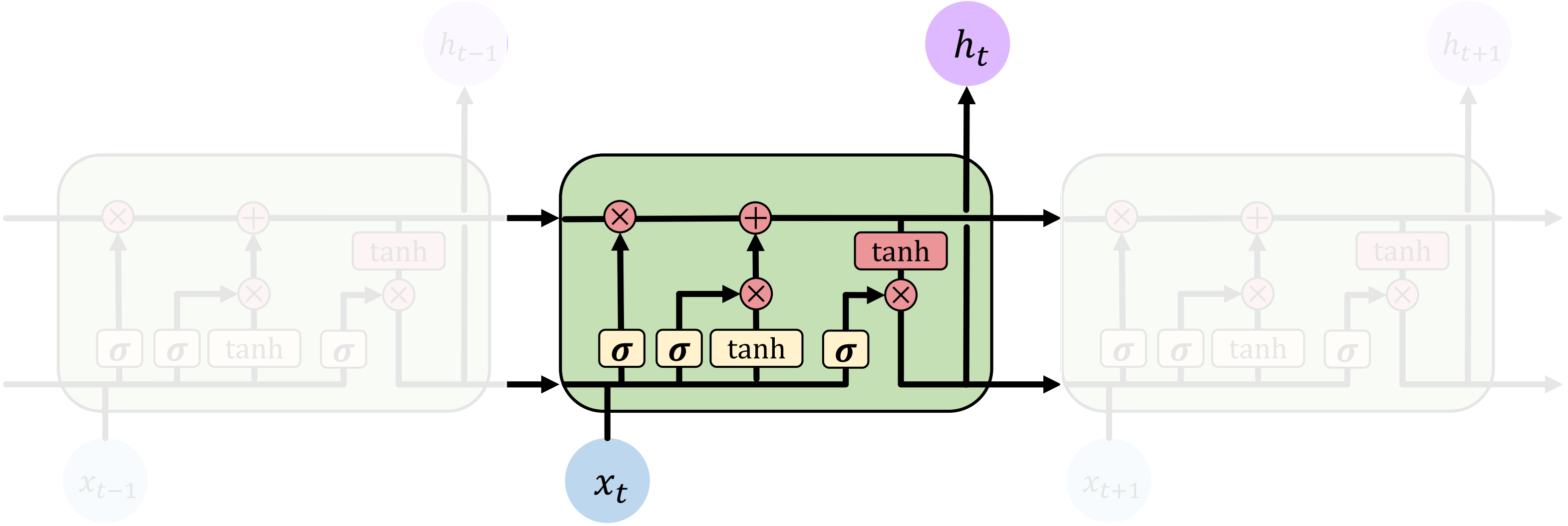
Vanishing gradient problem ➜ Long-short memory units (LSTM)

1. Pass-in the previous ("past") state for modification.
2. "Forget" a sub-set of the cell.
3. Update a sub-set of the cell.
4. Derive a filtered output and an updated cell-state for the next ("future") time-step.



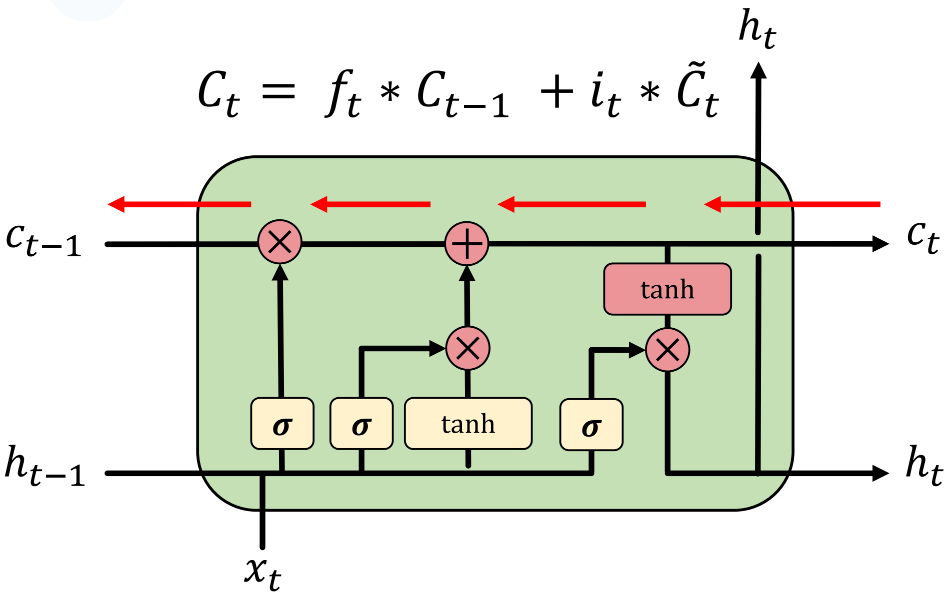


Long-short memory units (LSTM)

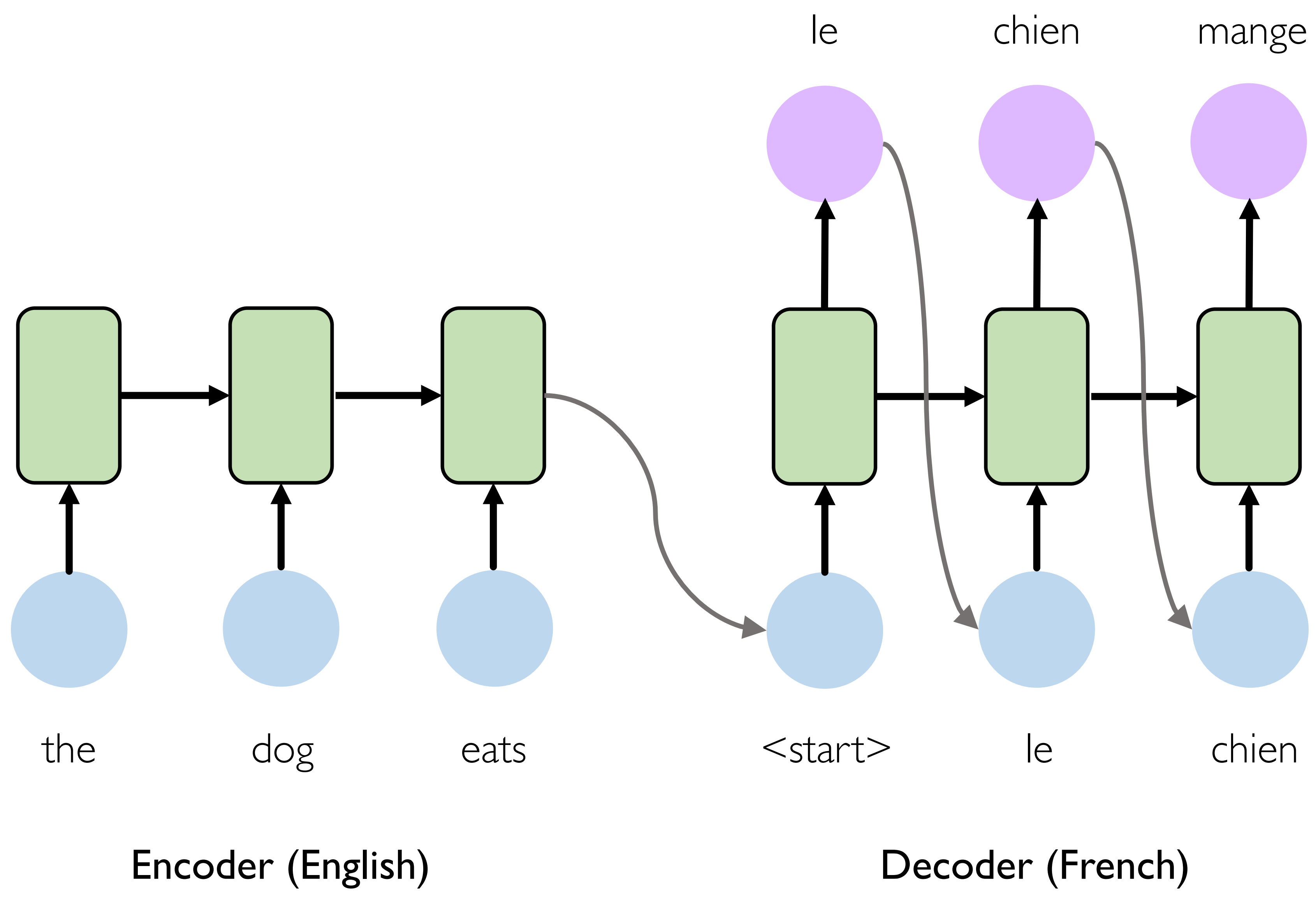


- Use gates to control the flow of information:
  - **Forget gate** gets rid of irrelevant information
  - **Selectively update** cell state
  - Output gate returns a **filtered version of the cell state**
- Back-propagation from C_t to C_{t-1} requires only element-wise multiplication - No matrix multiplication ➜ avoid vanishing gradient problem.

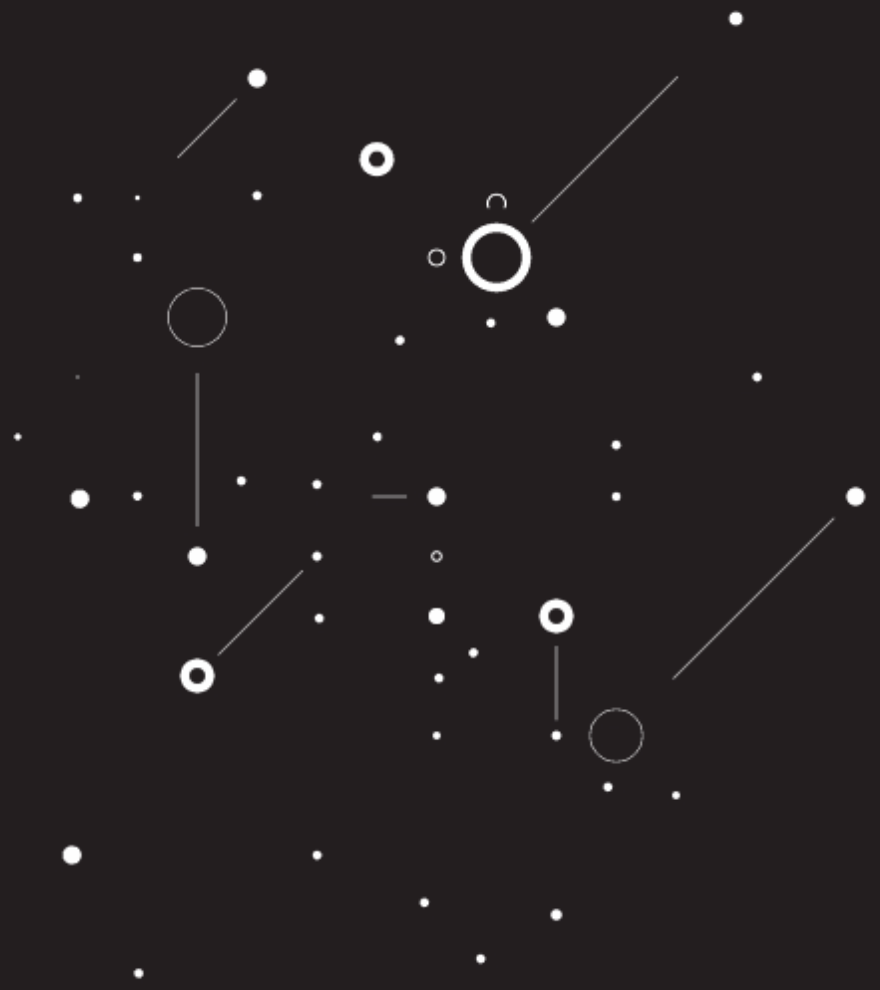
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



LSTMs for machine translation

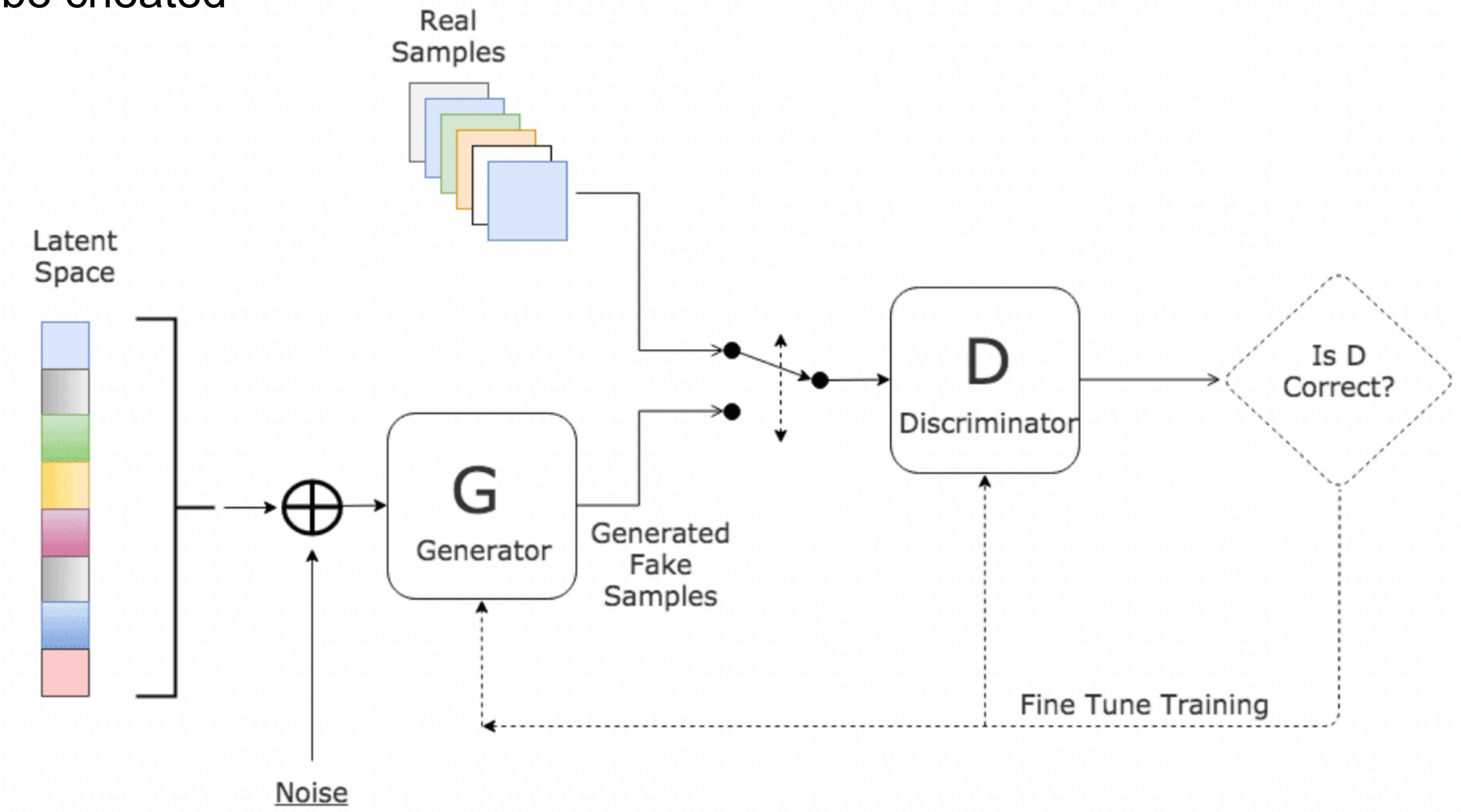


Generative models



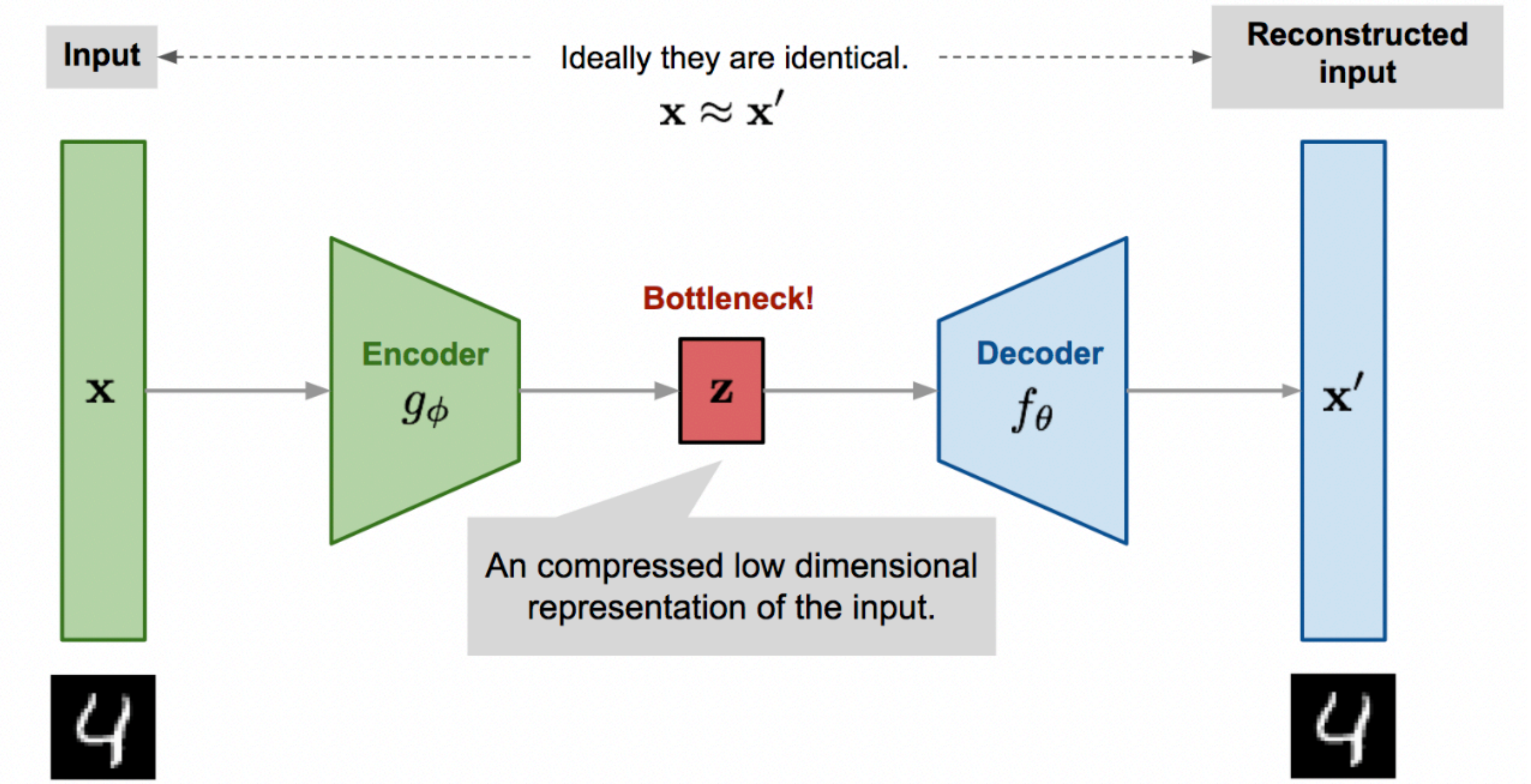
GAN ➜ generative adversarial network

- The generator G is trying hard to trick the discriminator, while the critic model D is trying hard not to be cheated



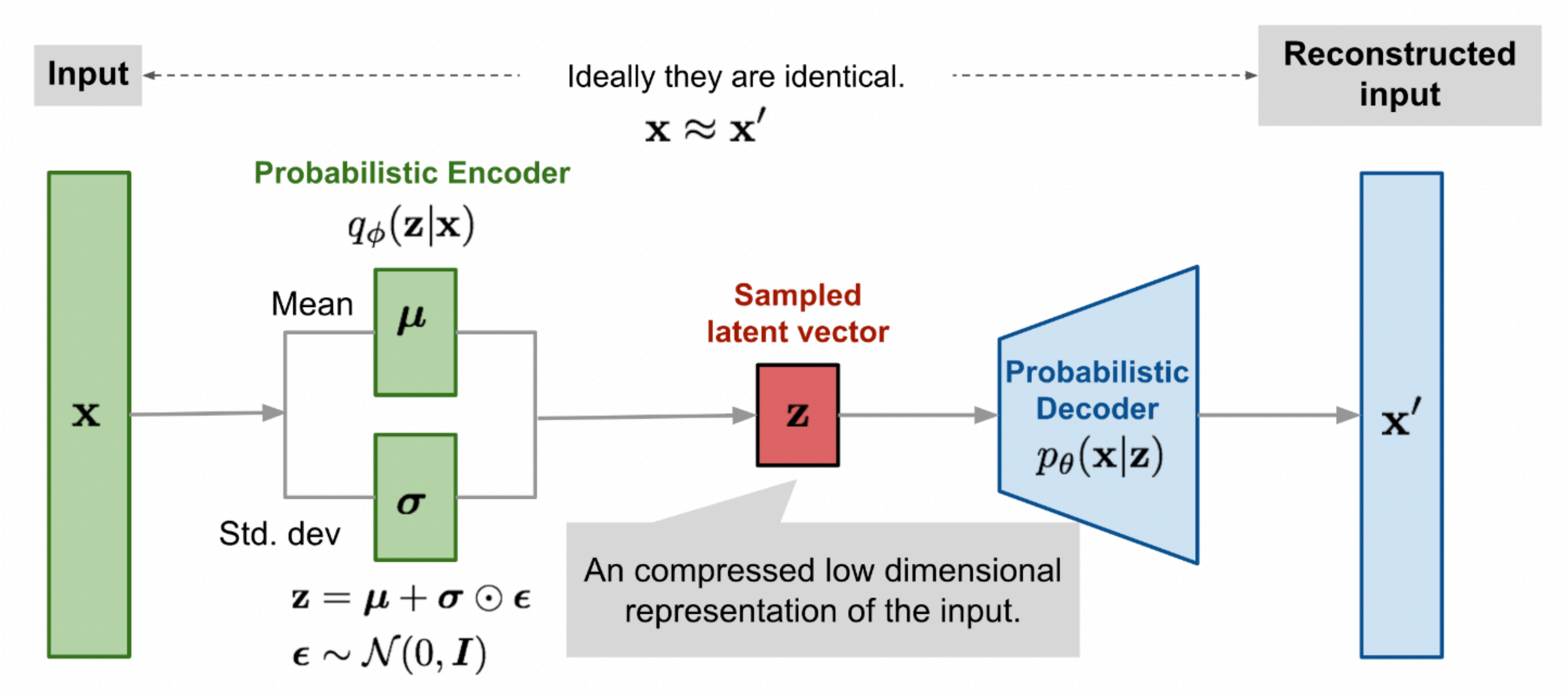
VAE ➜ variational autoencoder

- Learn an identity function in an unsupervised way to reconstruct the original input while compressing the data in the process



VAE ➜ variational autoencoder

- Instead of mapping the input into a fixed vector, we want to map it into a distribution



https://lilianweng.github.io/posts/2018-08-12-vae/

Large language models

1. Definition

An LLM is:

- A **neural network** with **hundreds of millions to trillions of parameters**.
- Trained on **massive amounts of text data** from books, websites, articles, and other sources.
- Capable of performing tasks like text generation, translation, summarization, question answering, and more—without being explicitly programmed for each.

---

2. Core Architecture: Transformer

The foundational architecture behind LLMs is the **transformer**, which:

- Uses **self-attention mechanisms** to weigh the importance of each word in a sequence relative to others.
- Enables parallel processing, making it more efficient and scalable than older models like RNNs or LSTMs.

---

3. Training Process

- LLMs are trained through **unsupervised or self-supervised learning**, primarily using a technique called **masked language modeling** (e.g., BERT) or **causal language modeling** (e.g., GPT).
- The model learns to **predict the next word** in a sentence or fill in missing words based on context.

Large language models

- Predict the next word in a sentence - generate one token at a time

Building a basic language model

Calculating the probabilities of all possible next words

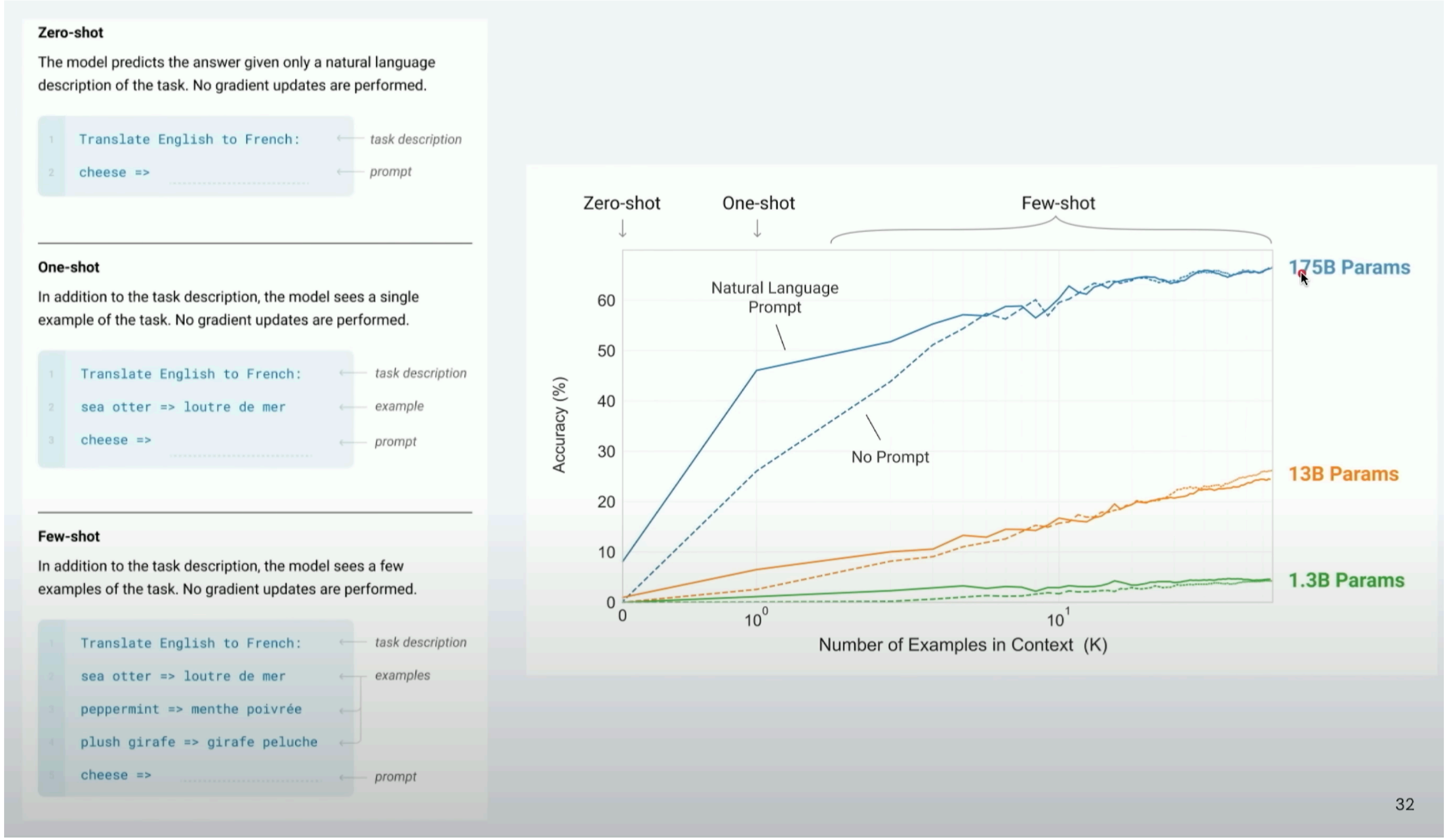
```python
def get_prob_distribution(stem_counts, stem):
  denominator = stem_counts[stem]
  probs = {}

  for k, v in stem_counts.items():
    if k[:-1] == stem:
      probs[k[-1]] = stem_counts[k] / denominator

  return probs
```

```python
probs = get_prob_distribution(stem_counts, ('it', 'was', 'the'))
probs
```

```
{'age': 0.3333333333333333,
 'best': 0.16666666666666666,
 'epoch': 0.3333333333333333,
 'worst': 0.16666666666666666}
```

Large language models

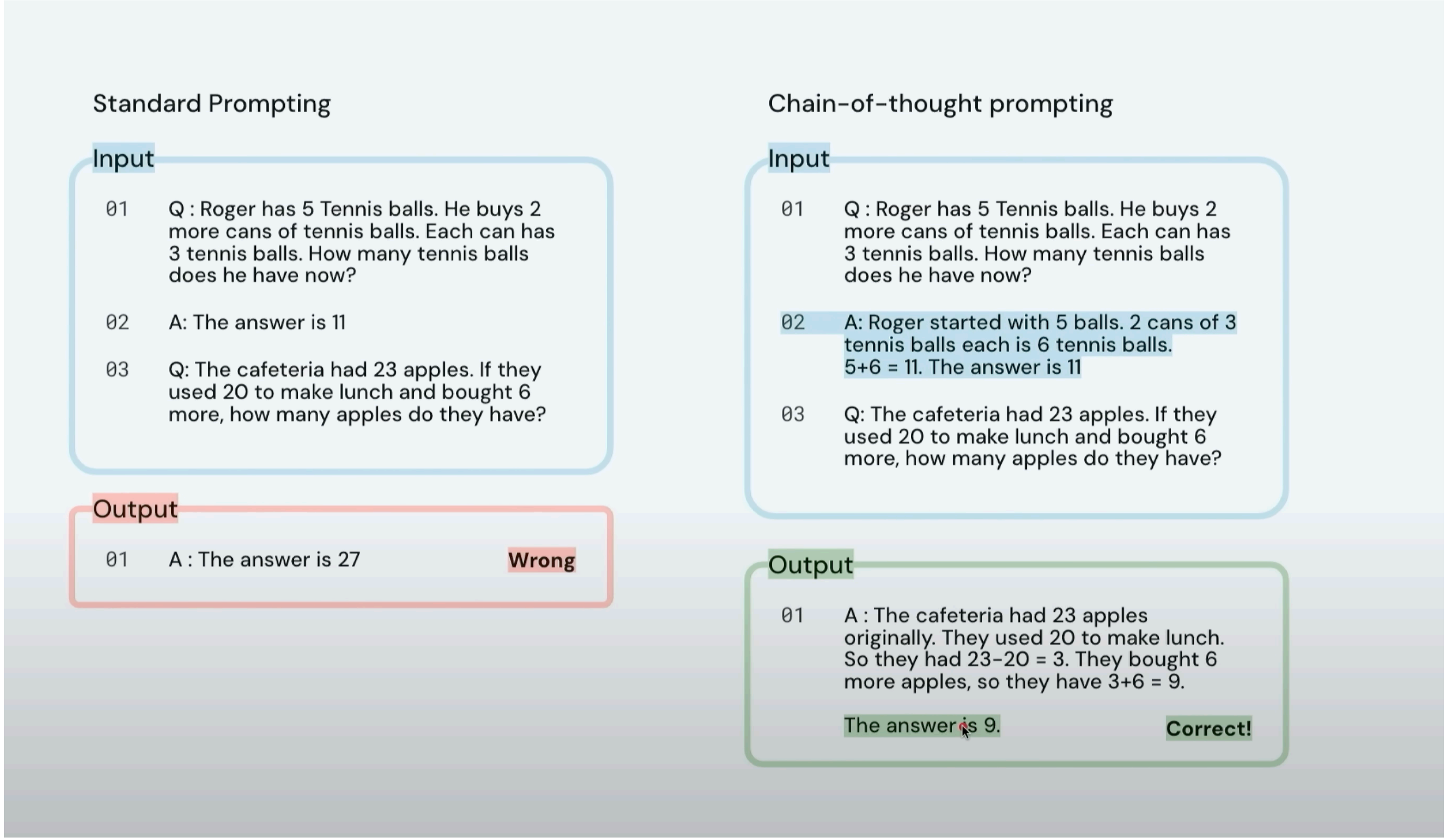
- Dramatic increase in zero-shot accuracy with increasing # parameters / context windows



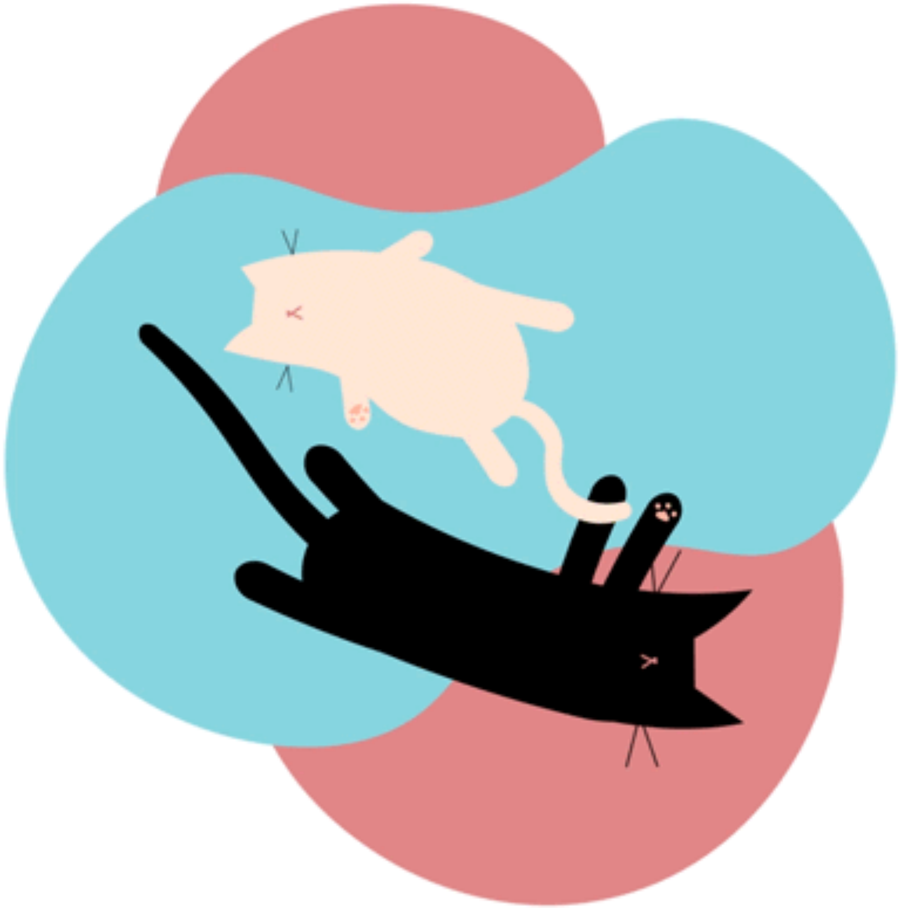
MIT 6.S191 (Google): Large Language Models - https://www.youtube.com/watch?v=ZNodOsz94cc

Large language models

- Context matters ➜ prompt engineering



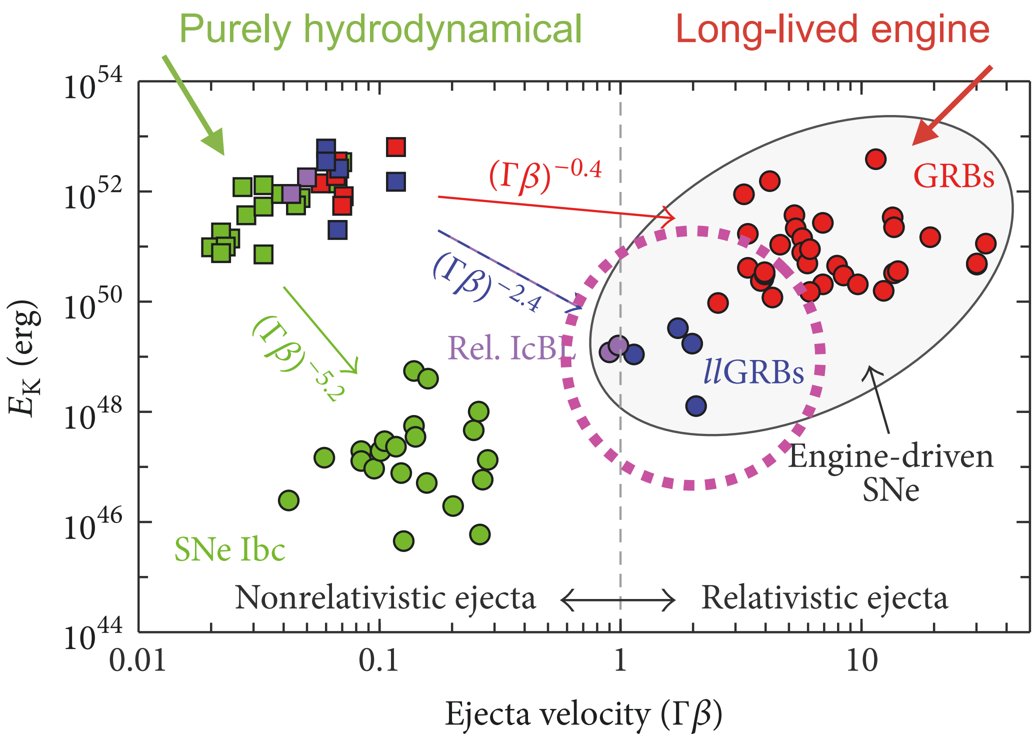
Coffee break



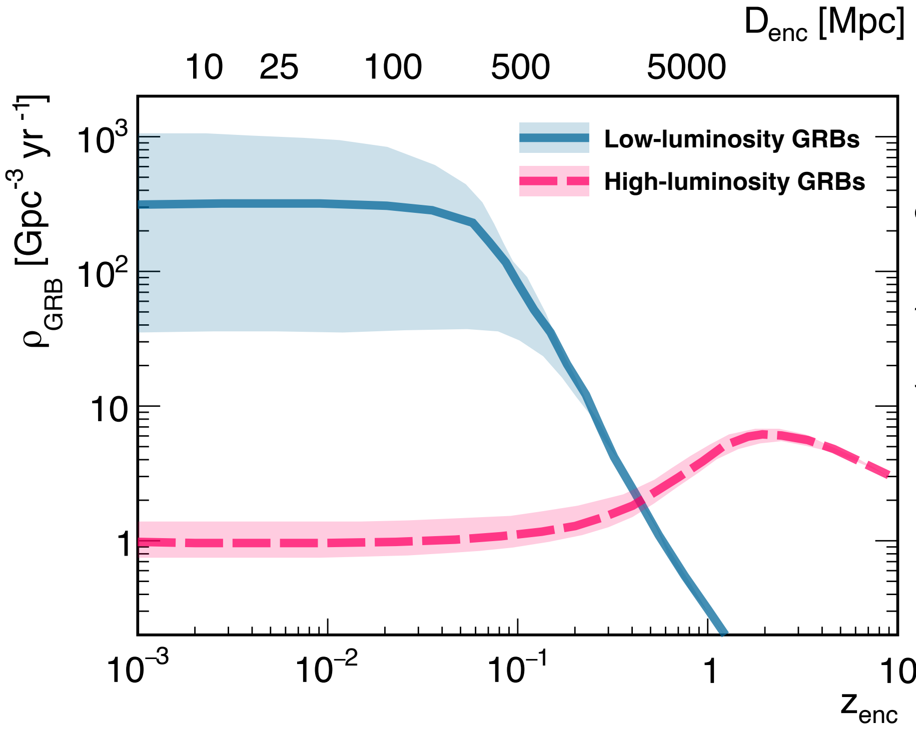
Astrophysics applications

Low luminosity GRBs as a benchmark pop. of short transients

- Expected high event rates ➜ possibly detected by self-triggering γ-rays / optical.
- Probe GRB physics.
- Possible association with ultra-high energy cosmic rays & neutrinos.
- …



Cano et al (2016) arxiv:1604.03549



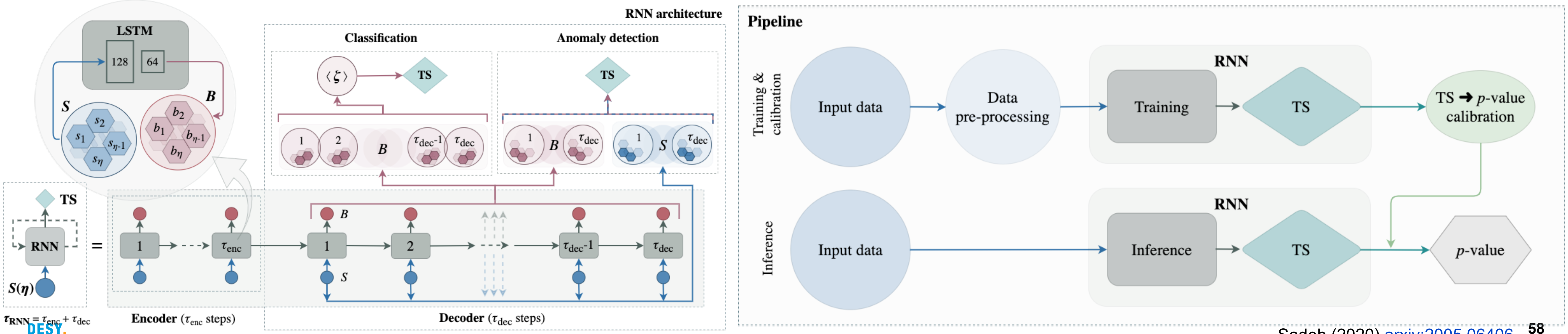
Liang et al (2007) arxiv:0605200

MMS transient detection

- **MMS observations**
  - Strategies
    - Real-time detection of signals in multiple channels
    - Near- and late-time follow-up for direct association of events
    - Archival stacking/population studies
    - Correlation of multiple low-significance observables, which combined may result in meaningful detections
  - Challenges
    - Uncertainties on instrument simulations (e.g., detector efficiency)
    - Uncertainties on physical backgrounds (e.g., galactic foregrounds)
    - Precise modelling of observing conditions (e.g., clouds, night-sky background)
    - Subtraction of artefacts (e.g., stars, satellites)
    - Extremely quick follow-up with multiple MMS/MWL facilities is necessary
- **Machine learning anomaly detection approach**
  - Training exclusively with real data ➜ mitigates systematics (no imperfect simulations used)
  - Does not require explicit physical modelling of perspective sources (generally not well constrained)
  - Facilitates data-fusion of inputs from different experiments
  - Extremely fast for evaluation, enabling quick response and coordination between facilities

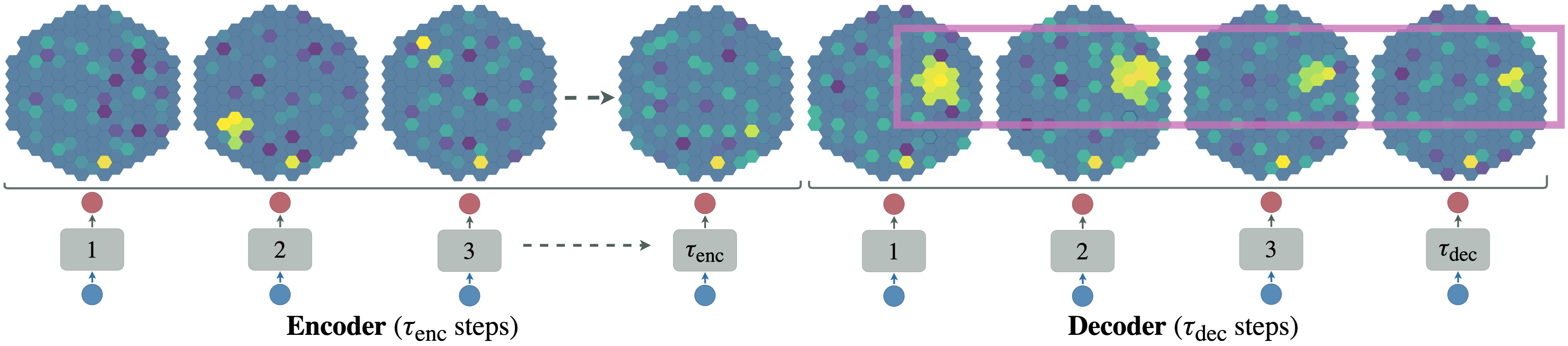
Recurrent neural networks for transient detection

- **Two methodologies for source detection**
  - Anomaly detection
    - Train an RNN to predict a time-series of the expected background
    - Compare the predictions to the true time series ➜ identify a transient event as an anomalous flare
  - Classification
    - Train an RNN to classify a time series as background or signal, using labels
    - Training requires both background data and signal data (➜ introduces some model dependence)
- **Calibration pipeline**
  - The results are calibrated statistically ➜ significance / p-value estimates for discovery

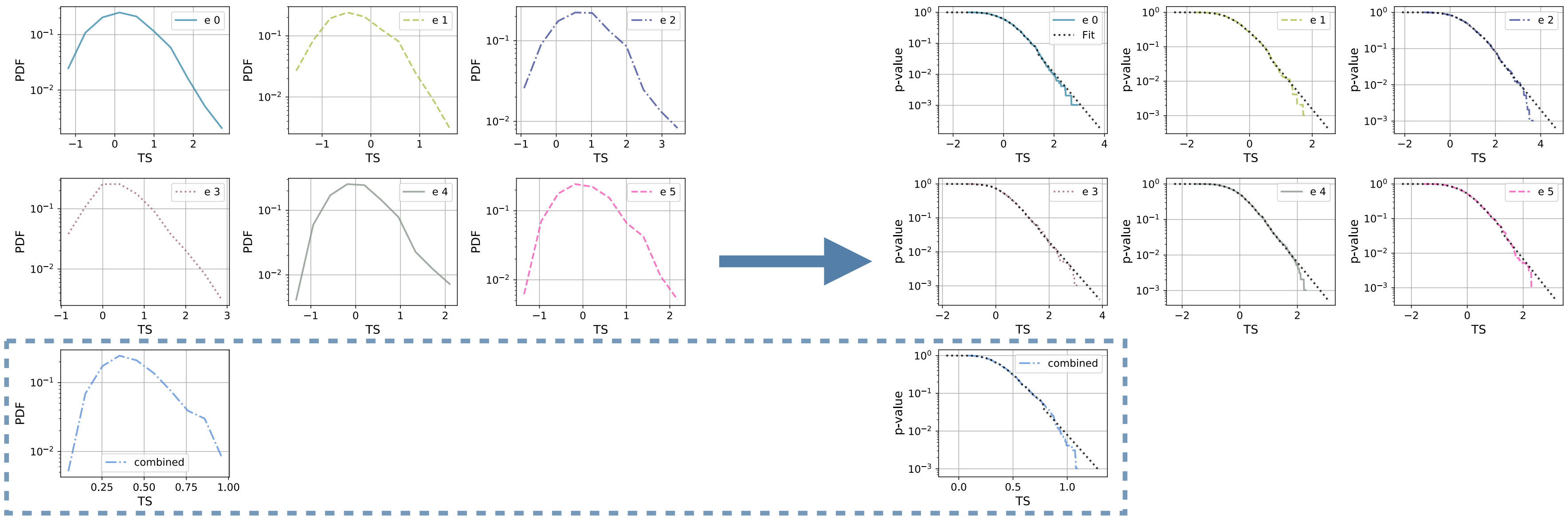


γ-ray transients

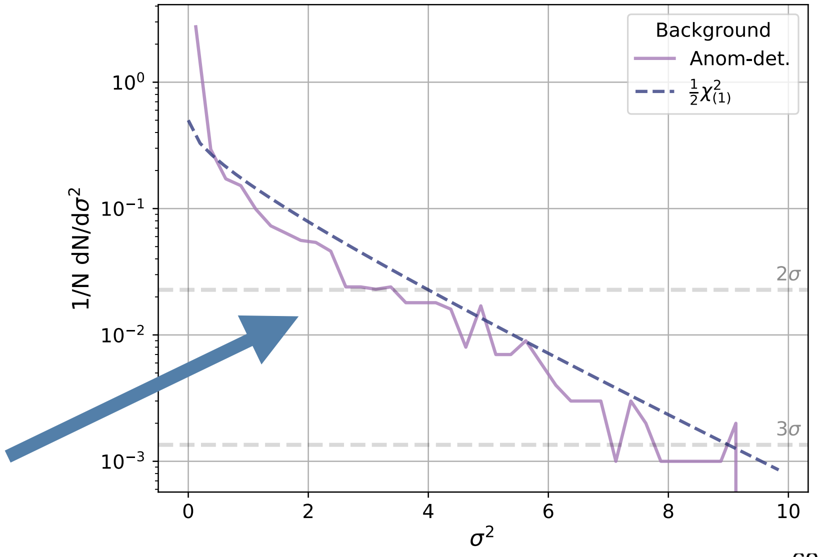
- **Example for the Cherenkov telescope array (CTA)**
  - Methodology
    - Train an RNN to predict a time-series of γ-ray event counts (binned in time & energy bins)
    - Add “auxiliary” input data, which affect the γ-ray rates (e.g., zenith of observation)
    - Compare the predictions to the true γ-ray rates, and identify a transient event as an anomalous flare
  - Training strategy
    - Anomaly detection: training exclusively on background data ➜ no-source in the region of interest; data potentially scrambled in time
    - Classification: also use simulations of GRBs ➜ simple spectral and temporal templates



Significance calibration for anomaly detection

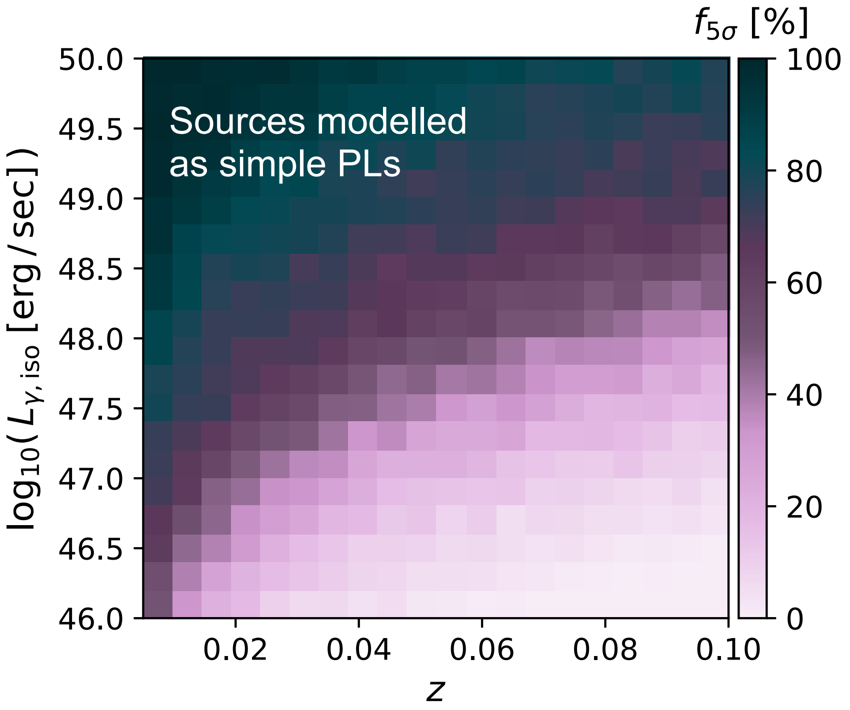


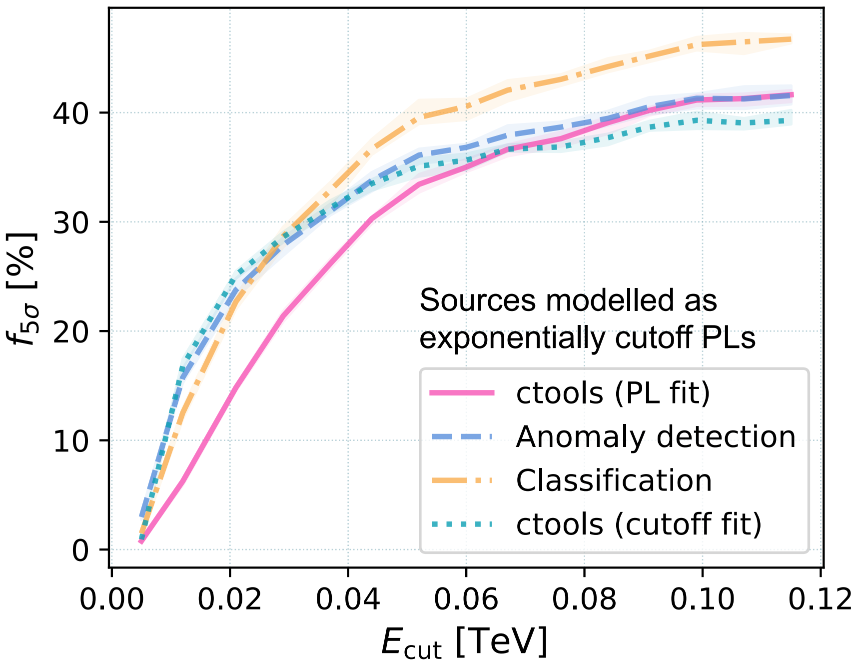
- In this example, the outputs of the RNN are γ-ray event counts in 6 energy bins
- Calibration procedure
    - Calculate a test statistic (TS) for each metric (based on the normalised difference between the RNN predictions and the ground truth)
    - Map TS ➜ p-values from TS distribution
    - Derive combined TS from the logarithms of individual p-values
    - Map combined TS ➜ combined p-value from distribution
- The combined TS distribution is compared to the expected background hypothesis



Serendipitous γ-ray transient detection

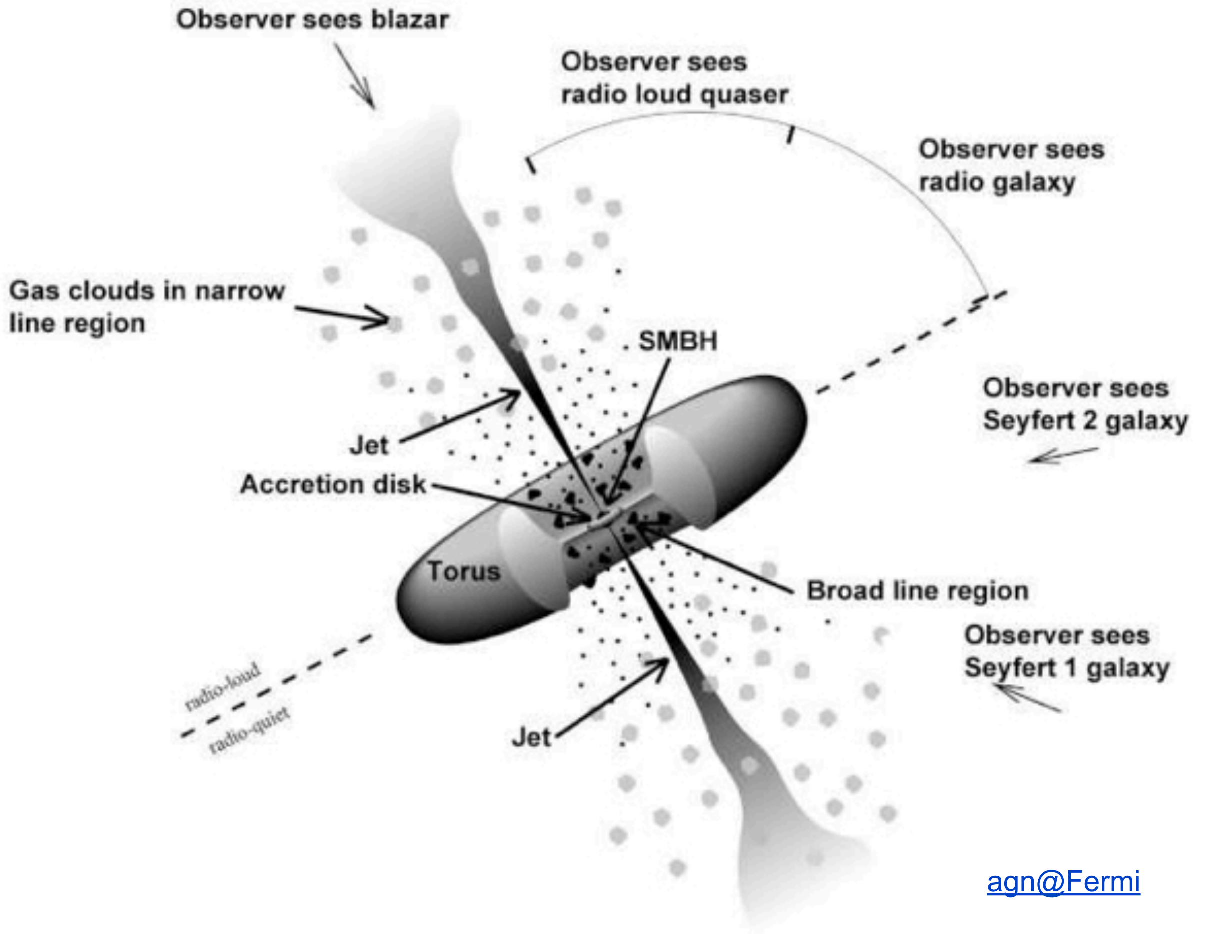
- Methodology
  - Shown here for a sample with expected properties for LL-GRBs, assuming either simple power-law (PL) or exponentially cutoff spectral PL models.
  - The reference detection rate (ctools) indicates a likelihood-based method, implemented as part of the ctools software package for CTA simulations
- Main takeaways
  - When simple PL models are fit the the data, both RNN methods perform better than the likelihood approach





Predicting MWL blazar flares

- Blazars
  - Active galactic nuclei (AGN) with a relativistic jet pointing towards the observer
  - Unresolved radio core, with flat or inverted spectrum
  - Extreme (temporal / amplitude) variability
  - High degree of optical & radio polarisation
  - Most common sources of EGAL GeV-TeV γ-rays



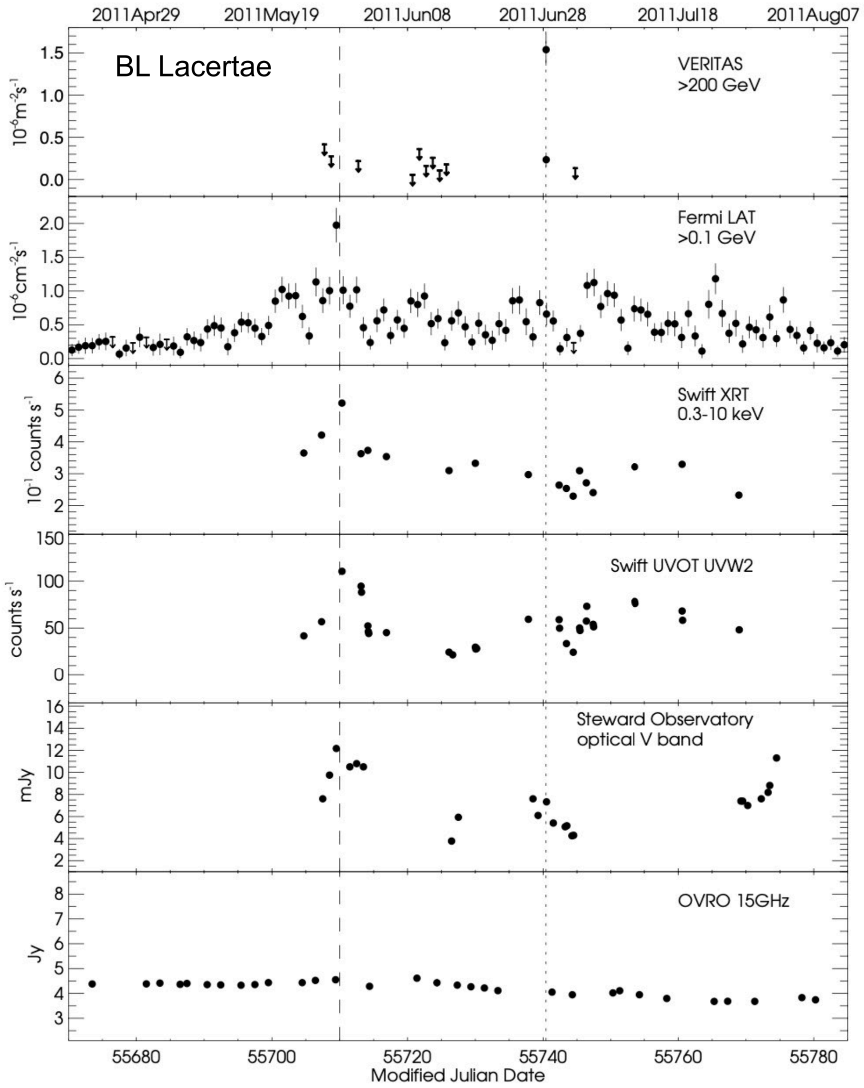
agn@Fermi

Predicting MWL blazar flares

- Blazar variability
    - Occurs on different time scales (minutes ➜ months)
    - Short-duration on top of slower variability trends
    - Flaring mechanisms still unclear

- Open questions
    - Origin of the HE emission ➜ leptonic (IC) and/or hadronic (proton synchrotron ; photo-meson) ➜ neutrinos & UHECRs?
    - Role of magnetic fields
    - Origin of ultra-short (~minute) variability ➜ turbulence ; magnetic reconnection ; shocks?
    - Extreme BL Lacs ➜ origin of very hard TeV spectra

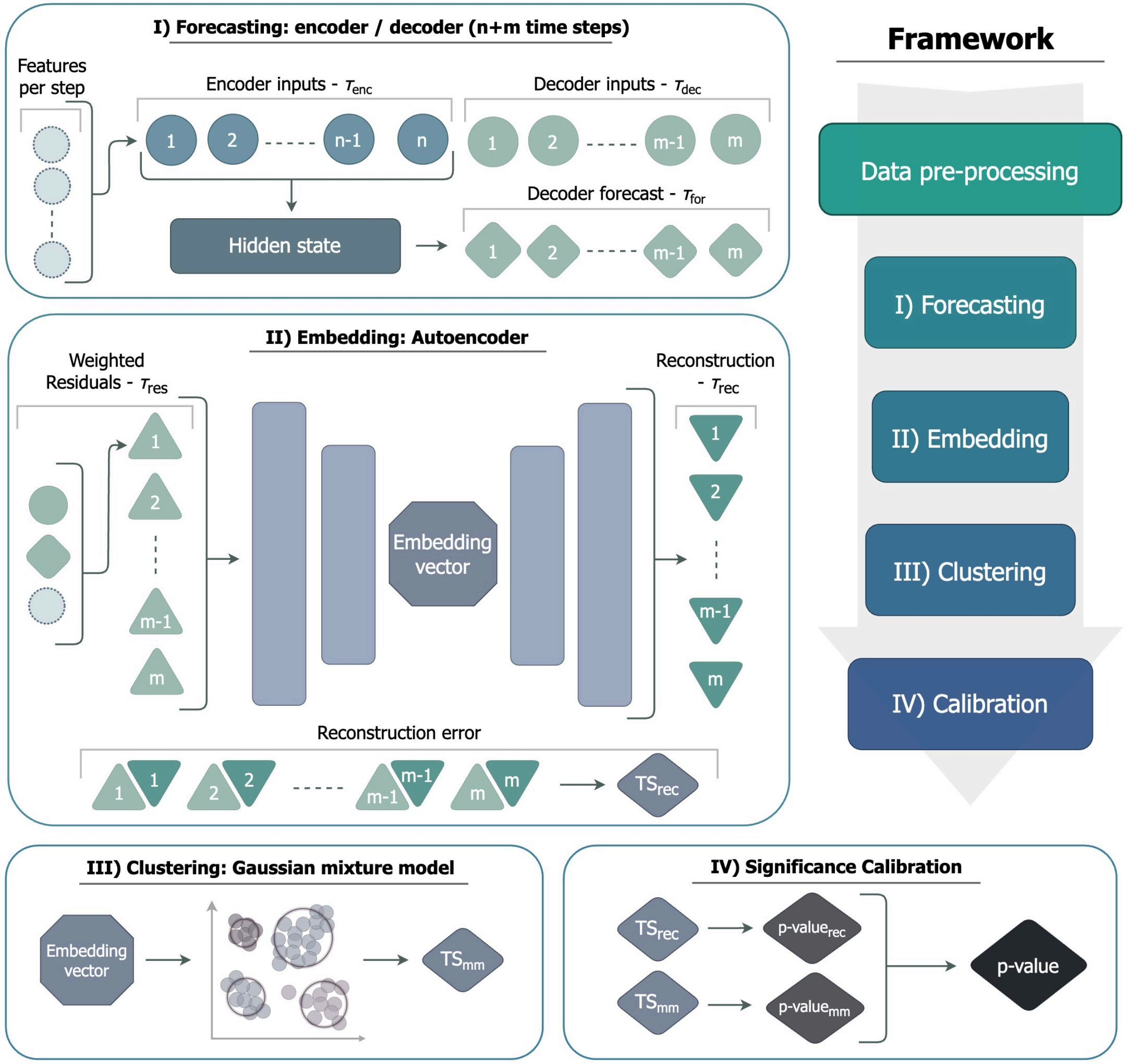
- Moving forward
    - ➜ (Simultaneous) MWL observations (+ polarisation)
    - ➜ Characterising variability on different time-scales



Arlen et al (2012) arxiv:1211.3073

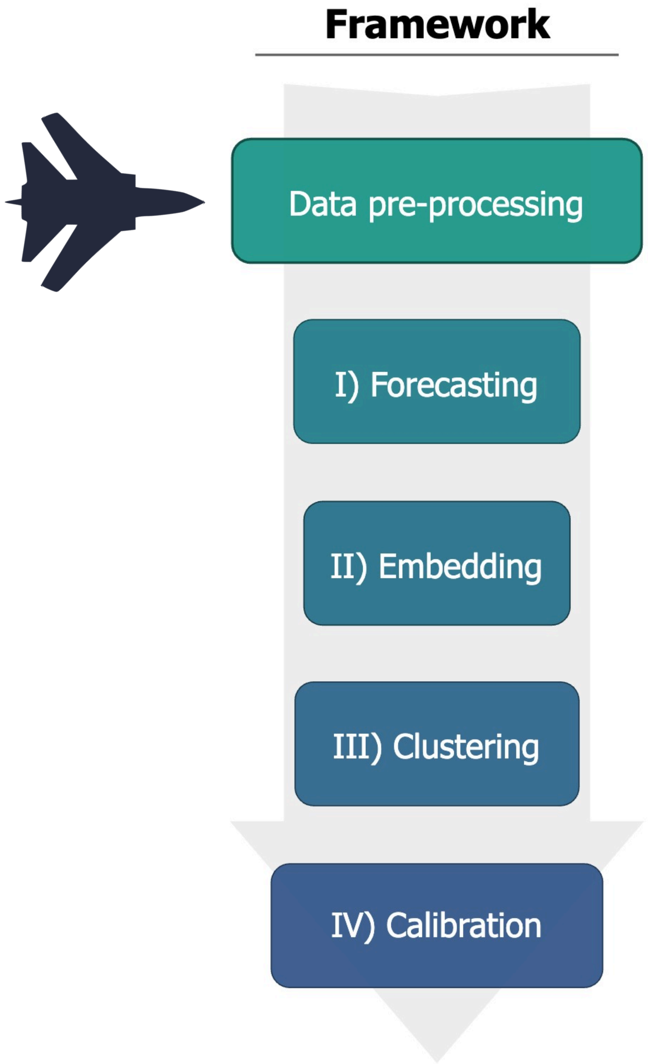
Predicting MWL blazar flares

H. Stolte, J. Sinapius,
I. Sadeh, E. Pueschel,
D. Berge, M. Weidlich

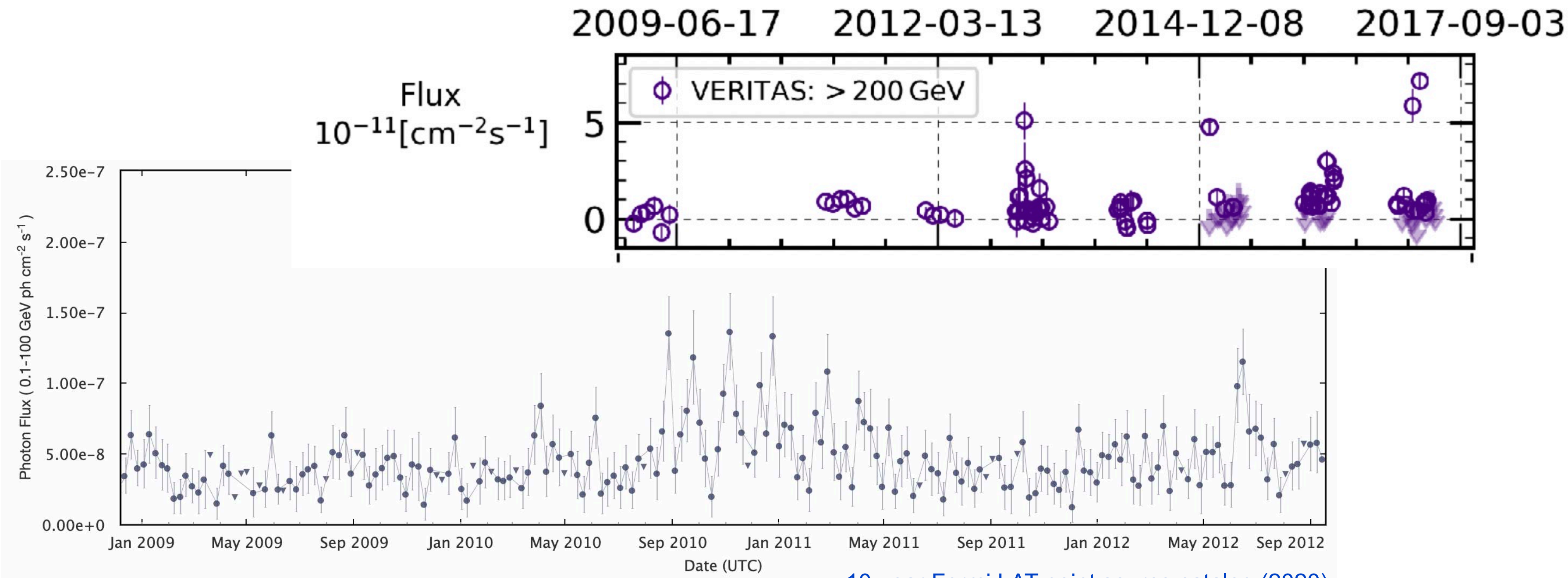


Predicting MWL blazar flares

H. Stolte, J. Sinapius,
I. Sadeh, E. Pueschel,
D. Berge, M. Weidlich



Valverde et al (2020) arxiv:2002.04119

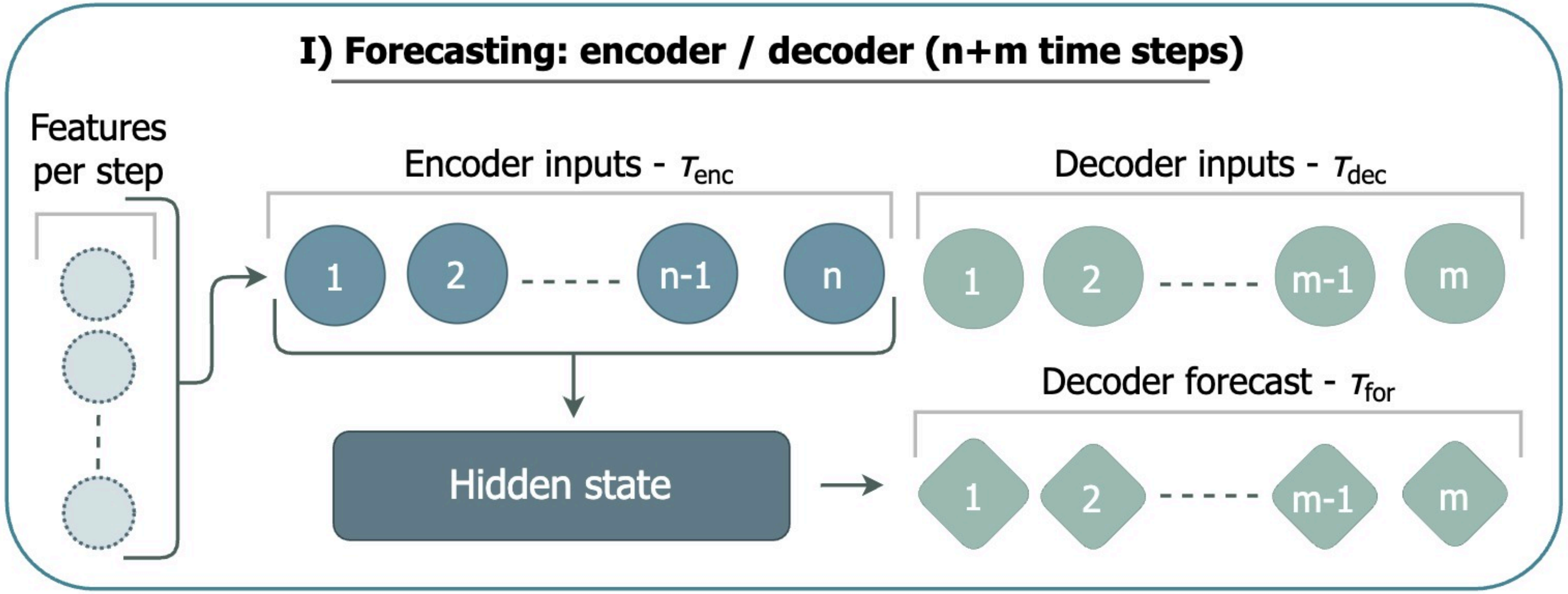


10-year Fermi LAT point source catalog (2020)

- Simulation dataset
  - Fermi-LAT ⊕ CTA ➜ modelled after 1ES 1215+303
  - Bayesian blocks ➜ general flux scale
  - Historically inspired sparsity in the VHE channel
  - Add stochastic noise ⊕ long-term (small scale) plateaus

Predicting MWL blazar flares

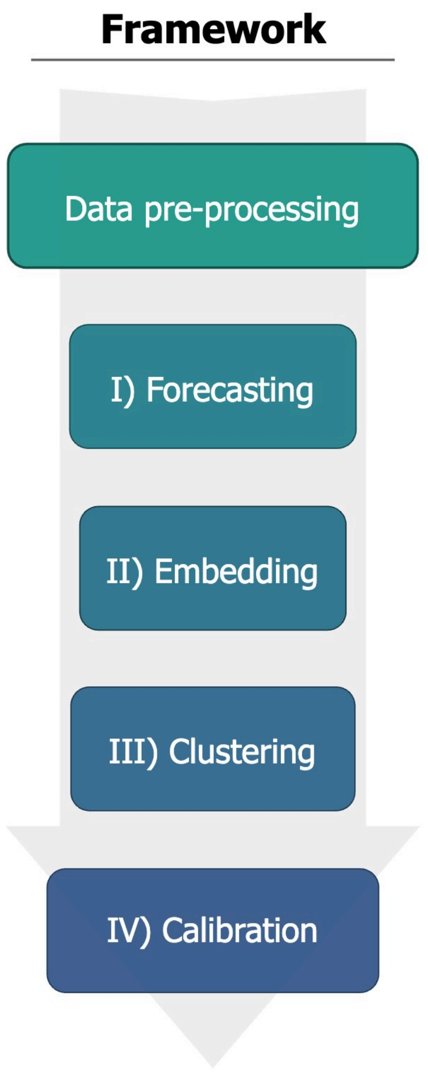
H. Stolte, J. Sinapius,
I. Sadeh, E. Pueschel,
D. Berge, M. Weidlich

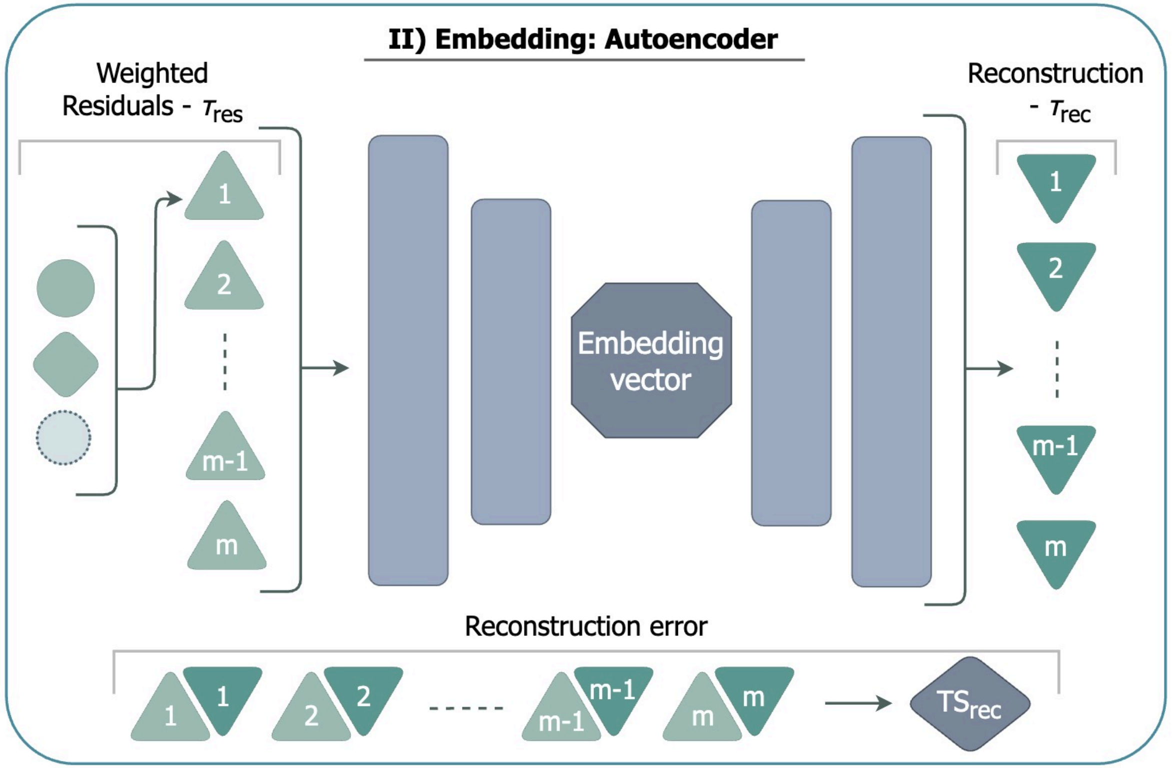


- Forecasting
  - MWL time-series as inputs ➜ encoder ⊕ decoder steps
  - An encoder-encoder trained exclusively on background data (shuffled time-series)
  - Project the encoder time-series onto the decoder span ➜ background-only hypothesis of "future" data

Predicting MWL blazar flares

H. Stolte, J. Sinapius,
I. Sadeh, E. Pueschel,
D. Berge, M. Weidlich

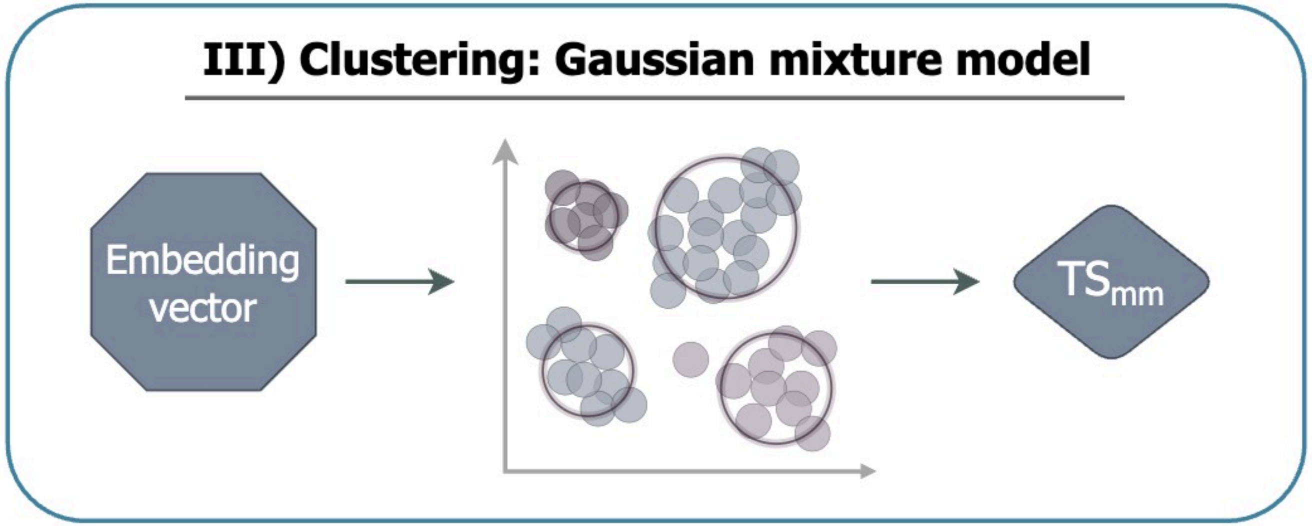




- Reconstruction
  - MWL time-series as inputs ➜ decoder ⊕ forecasting steps
  - An auto-encoder trained on multiple data classes (pure background + random fluctuations of different types)
  - ➜ Condense the data into a low-dimensional representation (latent dimension)

Predicting MWL blazar flares

H. Stolte, J. Sinapius,
I. Sadeh, E. Pueschel,
D. Berge, M. Weidlich

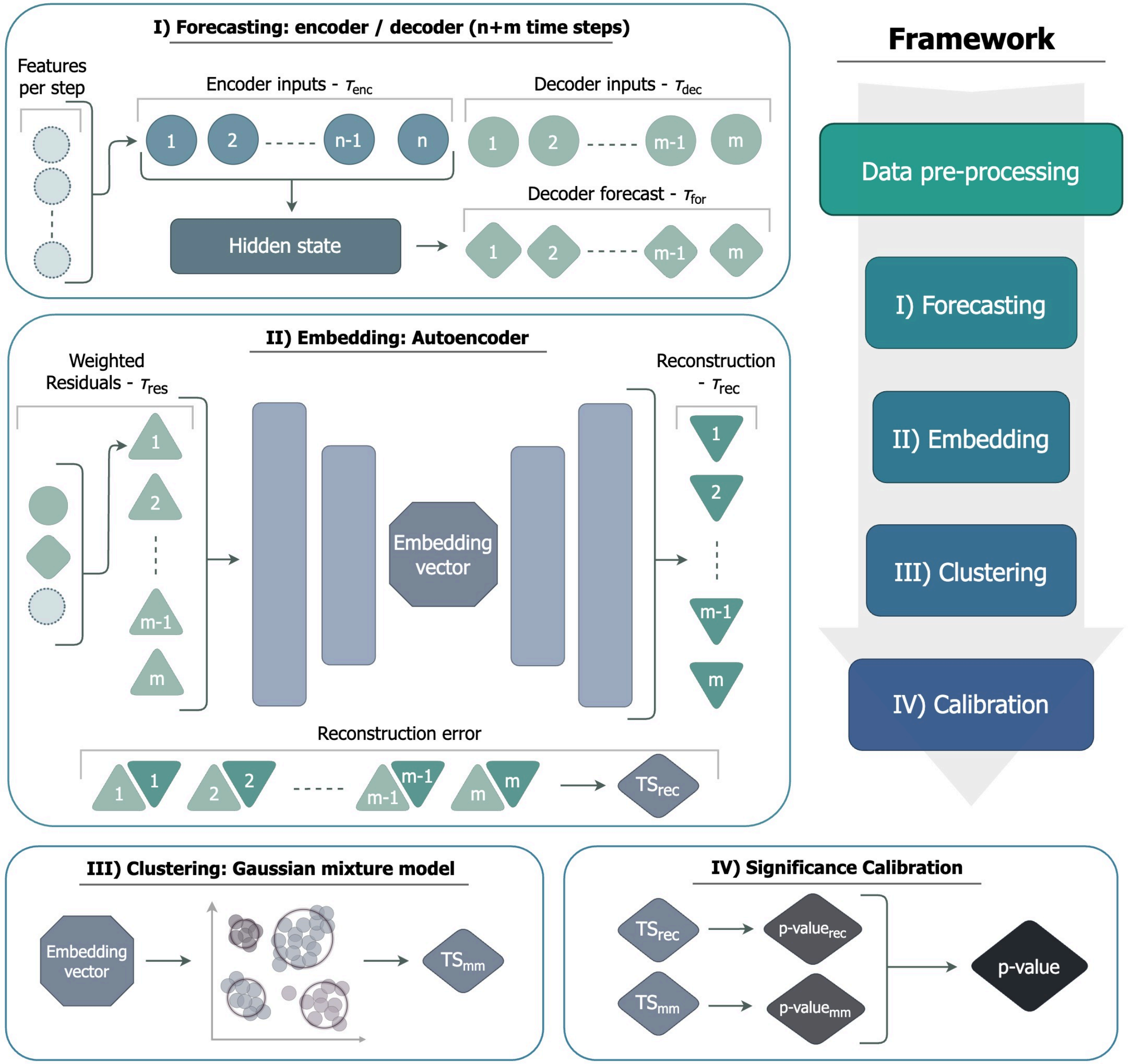


- Bayesian clustering
  - MCMC ➜ fit a Gaussian mixture model (GMM) to the (shuffled) background
    - Add reconstruction error as part of f_{aux}
  - Derive TS for anomalies ➜ probability for new data to belong to the GMM
  - Calibrate TS into p-values

Predicting MWL blazar flares

H. Stolte, J. Sinapius,
I. Sadeh, E. Pueschel,
D. Berge, M. Weidlich

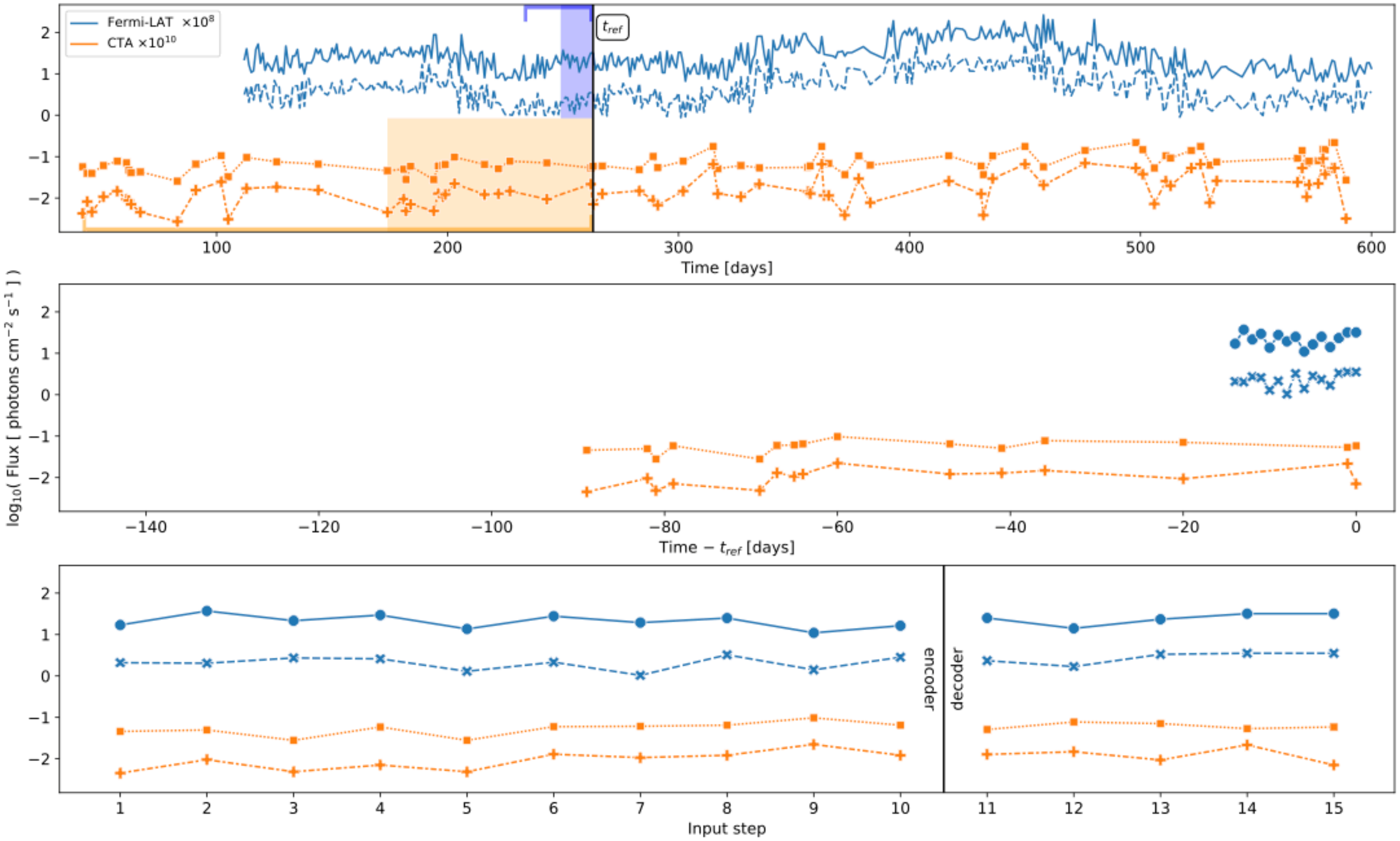
- Analysis strategy
  - Shuffle recent data in order to factor out known high states & other correlations
  - Construct sliding-window time-series
  - Enhance anomalies via predictions of a background-only hypothesis ➜ contrast with real data
  - Supplement auto-encoder background data with randomised examples of fluctuations to the inputs ➜ "open up" the cluster phase space
  - Fit the GMM on the background sample in cluster-space ➜ TS for anomalies



Predicting MWL blazar flares

H. Stolte, J. Sinapius,
I. Sadeh, E. Pueschel,
D. Berge, M. Weidlich

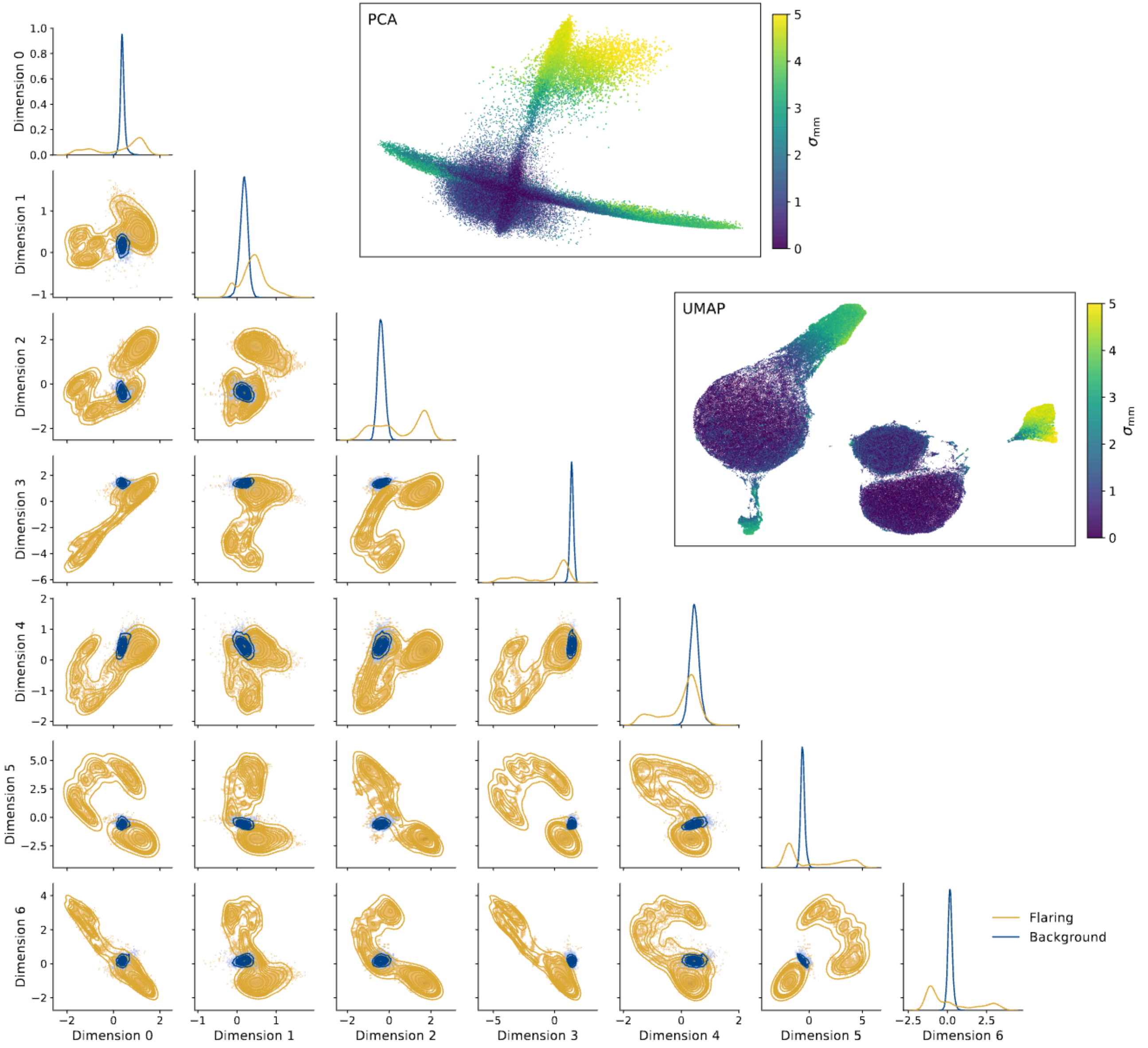
- Illustration of model inputs



Predicting MWL blazar flares

H. Stolte, J. Sinapius,
I. Sadeh, E. Pueschel,
D. Berge, M. Weidlich

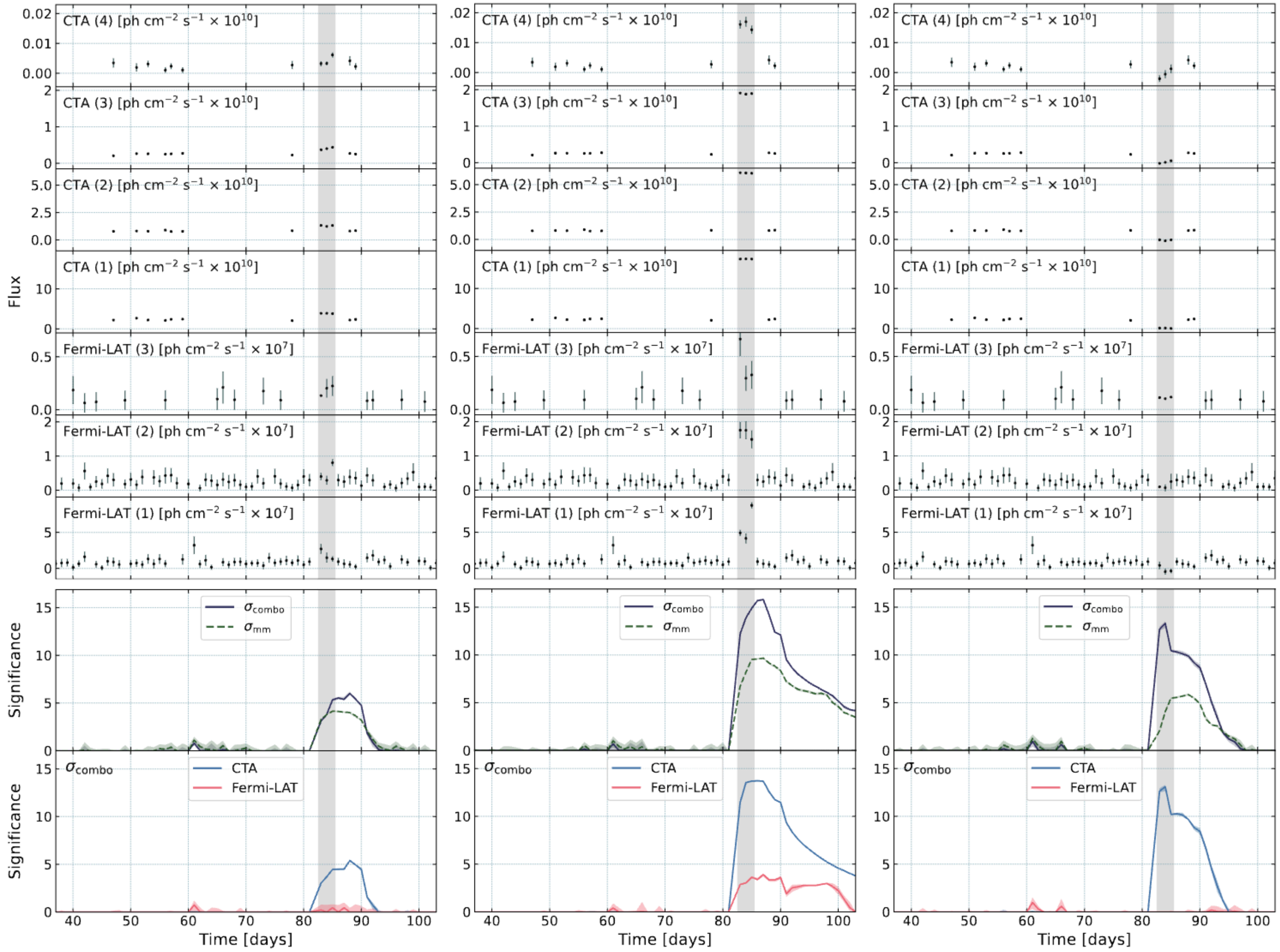
- Embedded "cluster-space" projections



Predicting MWL blazar flares

H. Stolte, J. Sinapius,
I. Sadeh, E. Pueschel,
D. Berge, M. Weidlich

- Performance on simulated "flares" for two input sources (Fermi-LAT & CTA)



Questions… ?