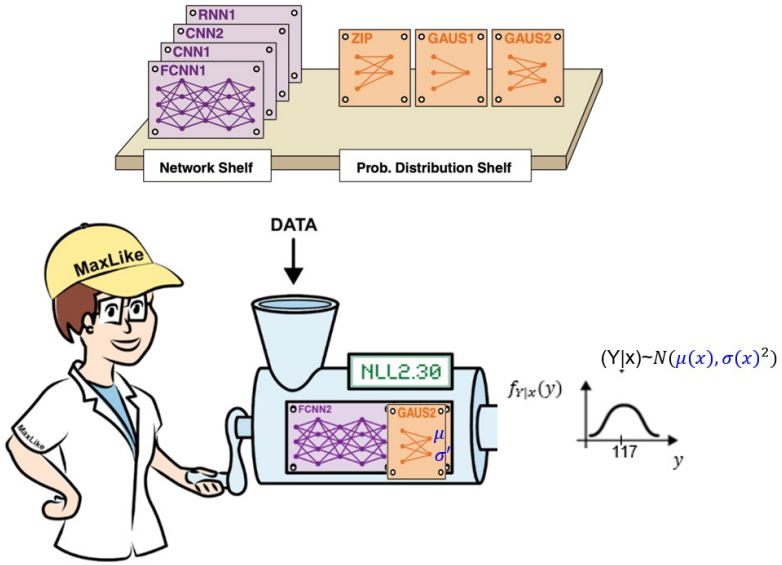
WBL Probabilistic Deep Learning:: Day 3

Beate Sick, Oliver Dürr

Day 3: Model evaluation and transformer NNs

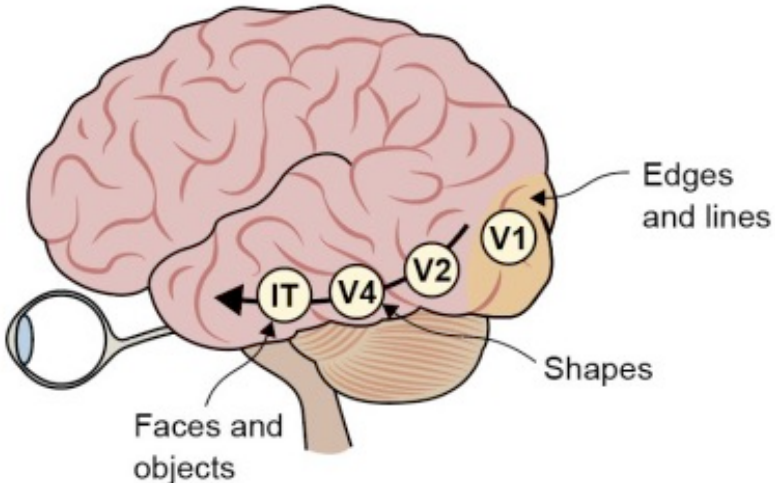


Topics (may change)

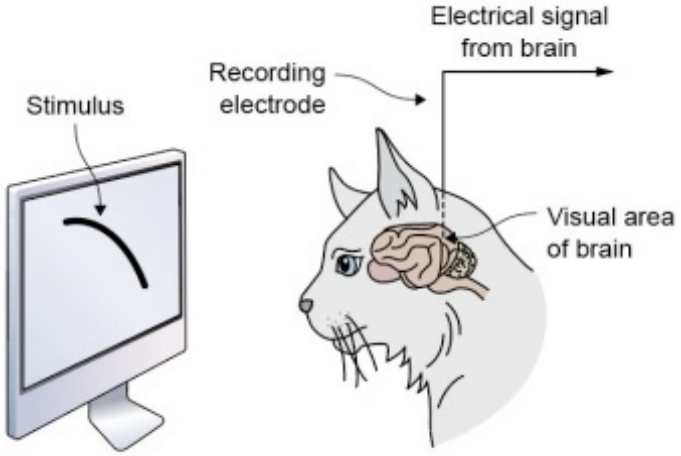
- Half day 1
  - Introduction to probabilistic DL
  - Introduction to Keras with pytorch backend
- Day 2
  - Loss function and optimization
  - Convolutional Neural Networks (CNN) for image data
  - Working with pretrained models
  - Transfer learning
  - Projects
- Day 3
  - Understand where the NN looks at
  - Model evaluation: Prediction Performance & Calibration
  - Transformer Architectures
  - Projects
- Half day 4
  - Project presentations
  - tba

Date
03.02.2025
10.02.2025
17.02.2025
24.02.2025

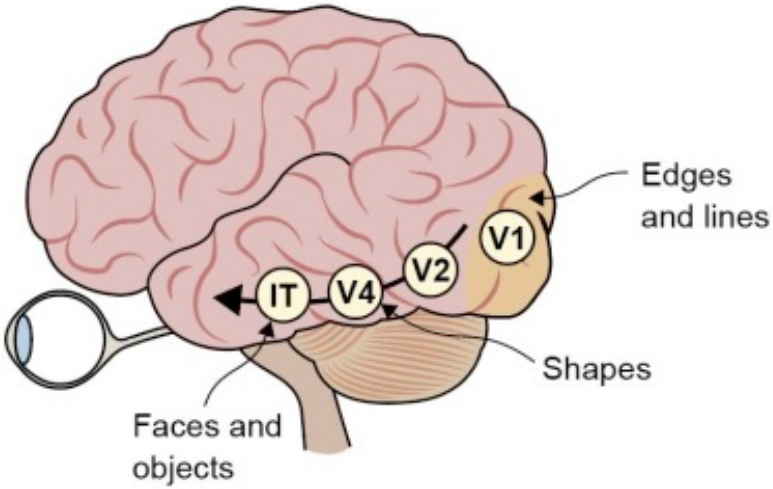
Biological and Artificial NNs

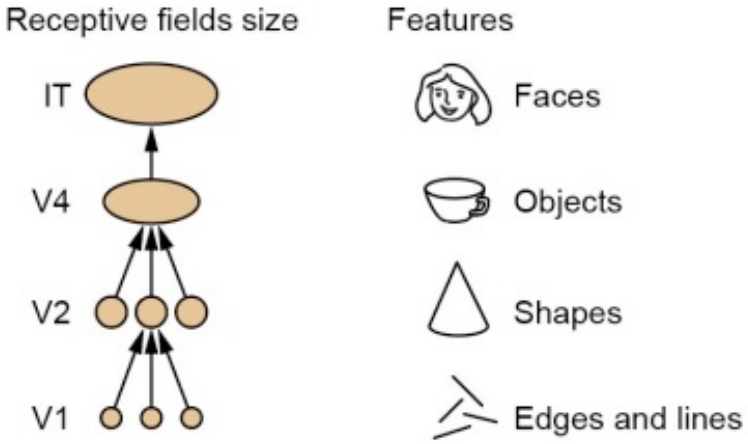


How does the brain respond to visually recieved stimuli?



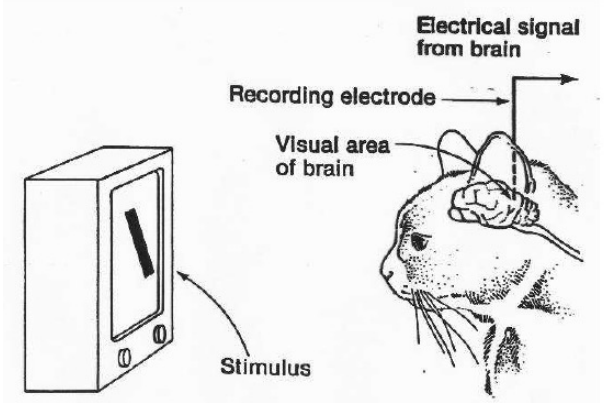
Setup of the experiment of Hubel and Wiesel in late 1950s in which they discovered neurons in the visual cortex that responded when moving **edges** were shown to the cat.

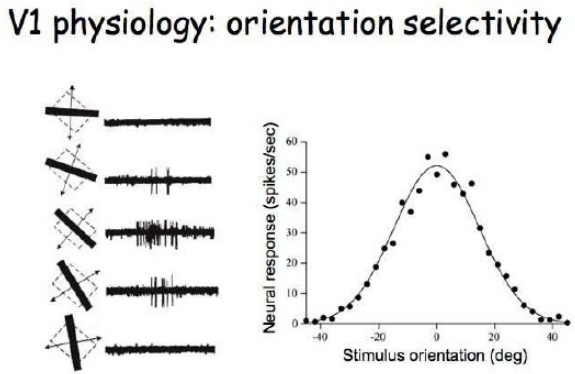


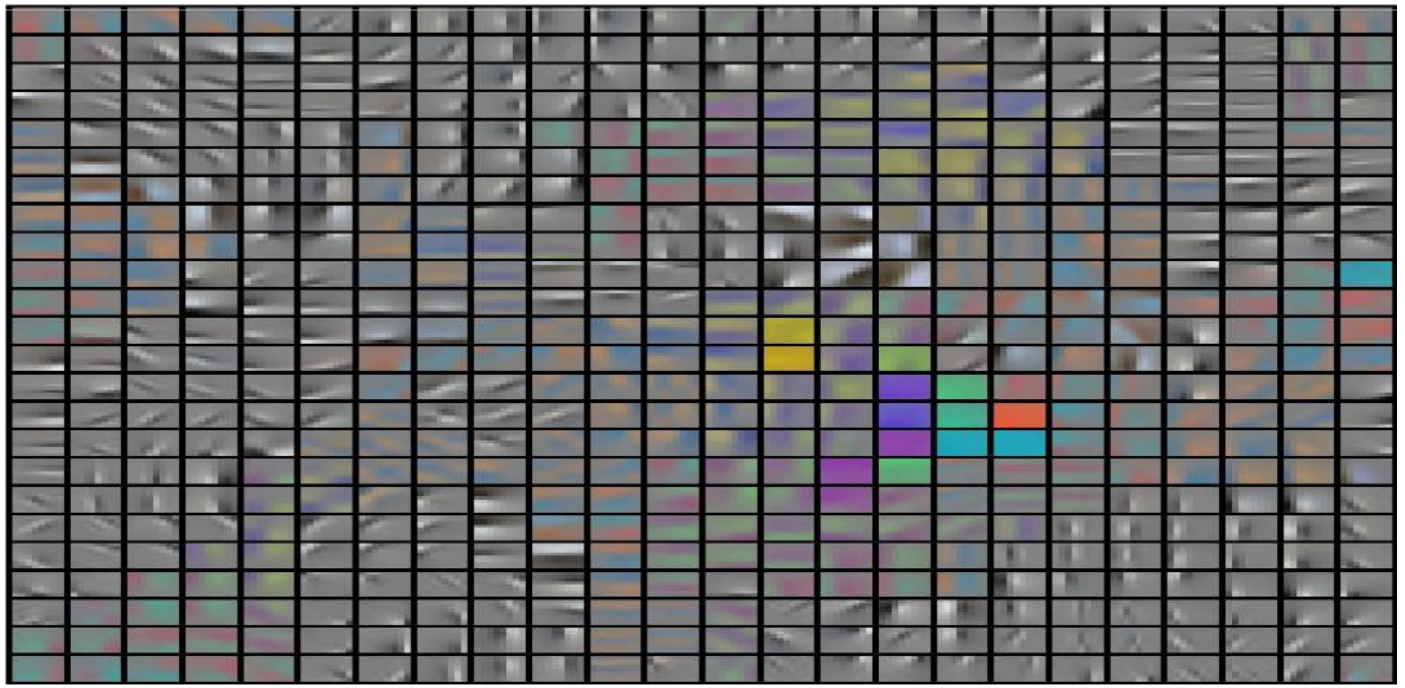


Organization of the visual cortex in a brain, where neurons in different regions respond to more and more complex stimuli

Compare neurons in brain region V1 in first layer of a CNN





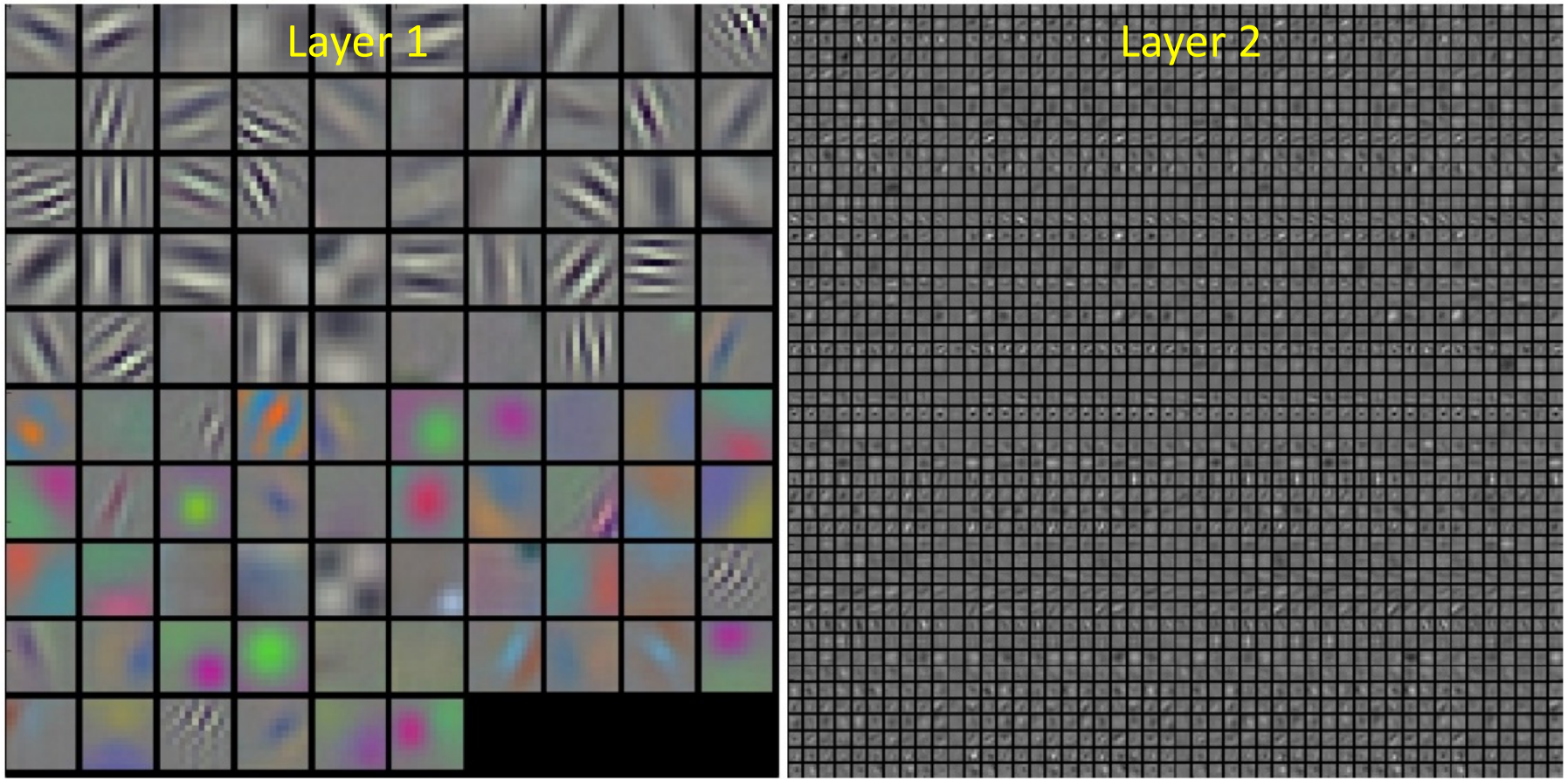


Neurons in brain region V1 and neurons in 1. layer of a CNN respond to similar patterns

Slide (modified) credits: Nando de Freitas

Visualize the weights used in filters

Filter weights from a trained Alex Net

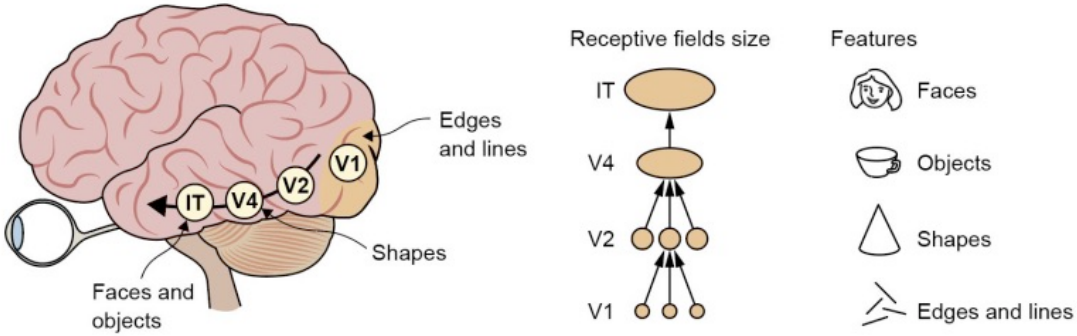


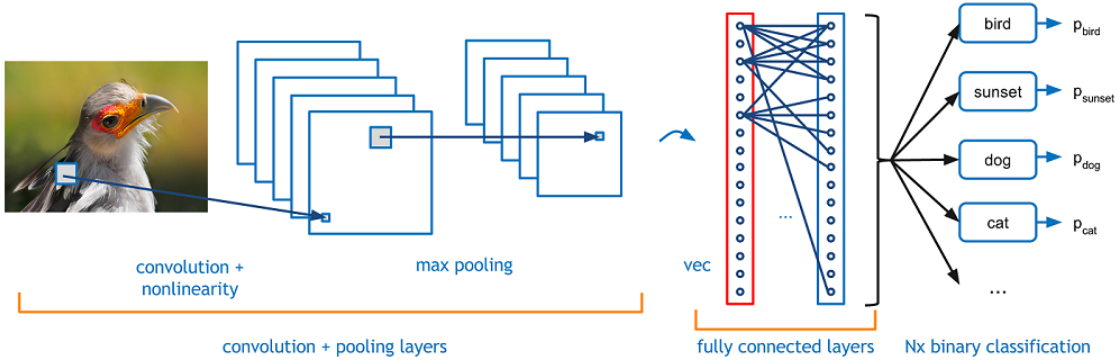
Only in layer 1 the filter pattern correspond to extracted patterns in the image.

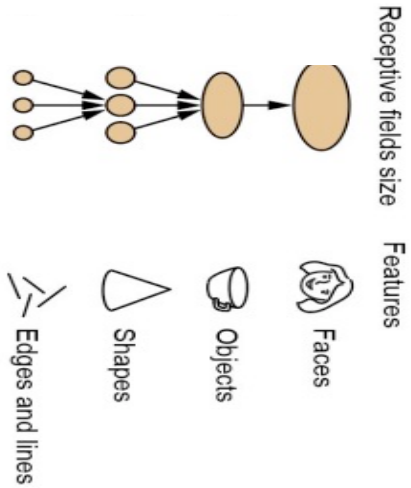
In higher layers we can only check if patterns look noisy, which would indicate that the network that hasn't been trained for long enough, or possibly with a too low regularization strength that may have led to overfitting.

Image credits: http://cs231n.github.io/understanding-cnn/

Weak analogies between brain and CNNs architecture





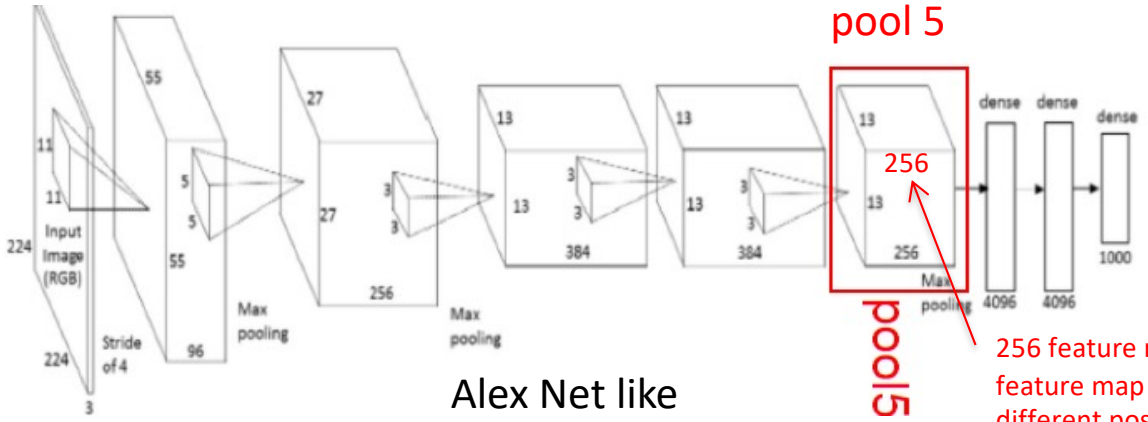


xAI = explainable AI
Where does a CNN look at?

Visualize patches yielding high values in activation maps

Figure 4: Top regions for six pool$_5$ units. Receptive fields and activation values are drawn in white. Some units are aligned to concepts, such as people (row 1) or text (4). Other units capture texture and material properties, such as dot arrays (2) and specular reflections (6).

http://cs231n.github.io/understanding-cnn/



Alex Net like

Here we show image patches that activate maps in layer 5 most.

256 feature maps generated by 256 different 3x3 filters. Each feature map consists of equivalent neurons looking at different positions of the input volume.

What kind of image (patches) excites a certain neuron corresponding to a large activation in a feature map?

10 images from data set leading to high signals 6 feature maps of **conv6**

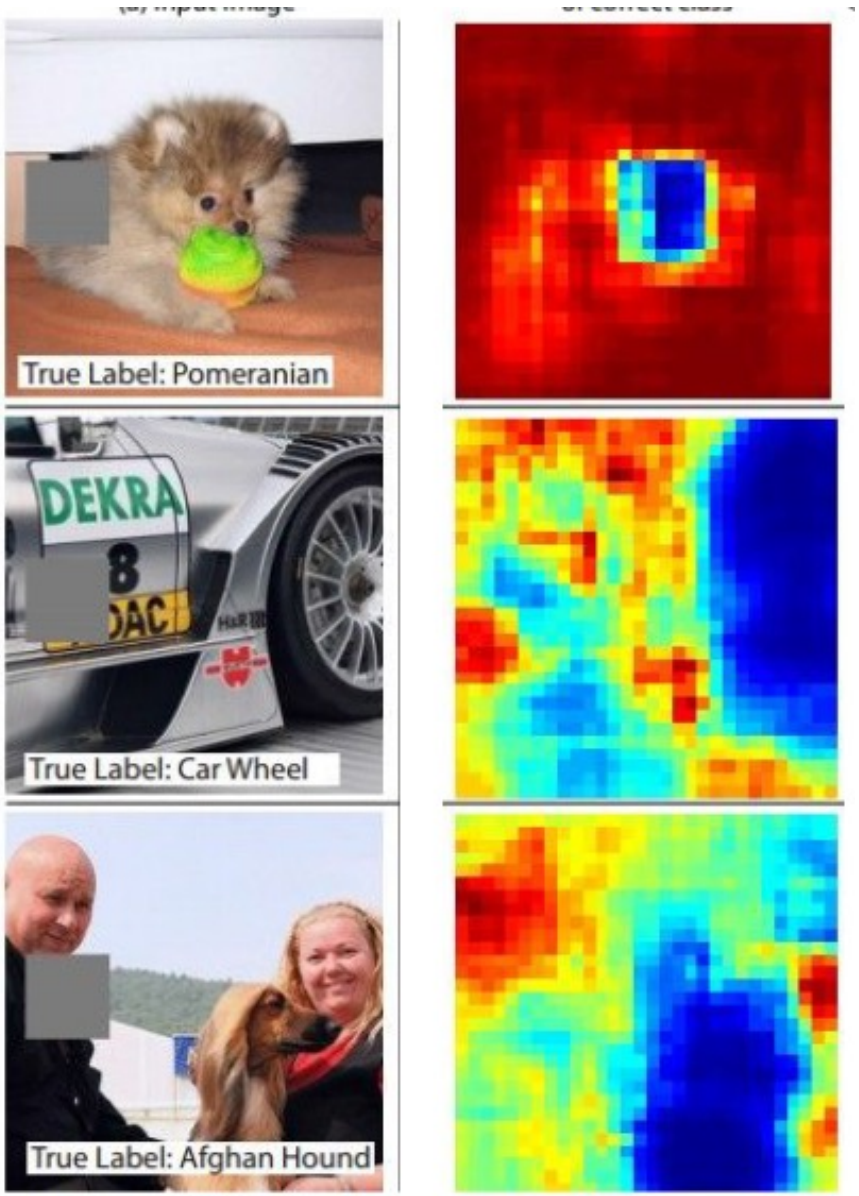
10 images from data set leading to high signals 6 feature maps of **conv9**

image credit: cs231n

Which pixels are important for the classification?
Occlusion experiments

Occlude part of the image with a mask and check for each position of the mask how strongly the score for the correct class is changing.

Warning:
Usefulness depends on application…



Occlusion experiments [Zeiler & Fergus 2013]

image credit: cs231n

Which pixels are important for the classification?
LIME: Local Interpretable Model-agnostic Explanations

Idea:

1) perturb interpretable features of the instance – e.g. randomly delete super-pixels in an image and track as perturbation vector such as (0,1,1,0,…,1)=x.
2) Classify perturbed instance by your model, here a CNN, and track the achieved classification-score
3) Identify for which features/super-pixels the presence in the perturbed input version are important to get a high classification score (use RF or lasso for y~x)

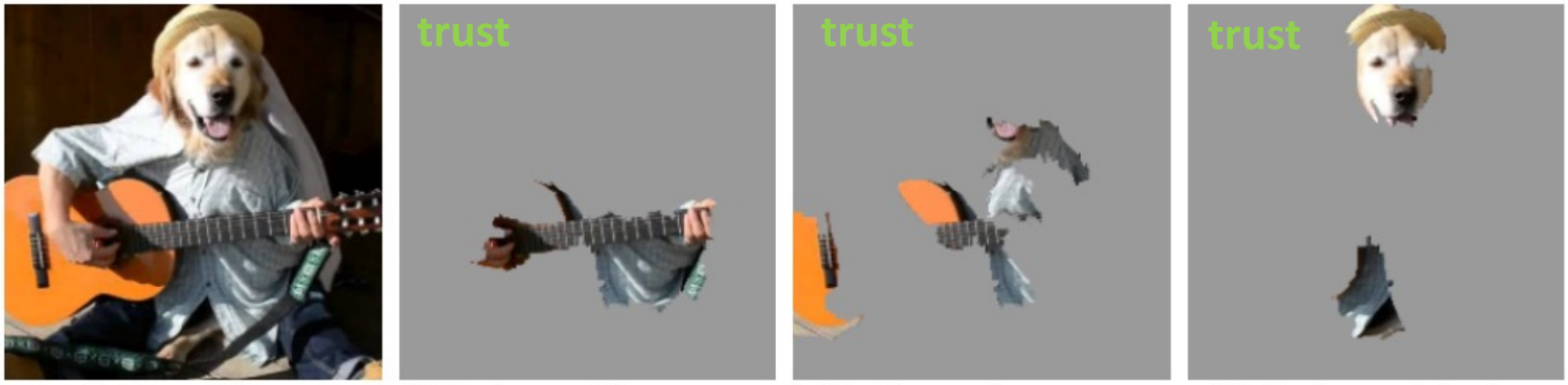
https://arxiv.org/abs/1602.04938

(a) Husky classified as wolf (b) Explanation

-> presence of snow was used to distinguish wolf and husky

-> Explain the CNN classification by showing instance-specific important features

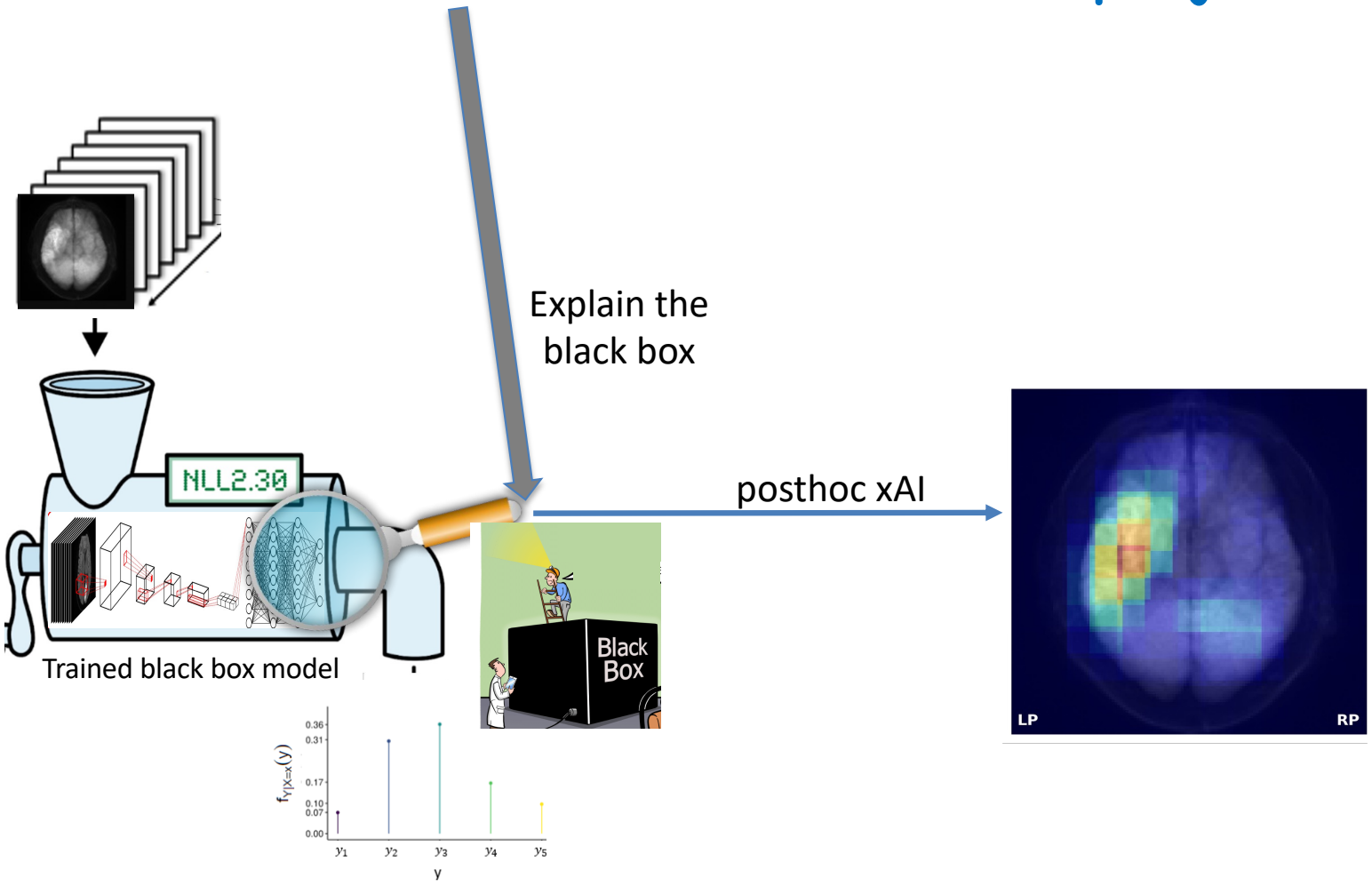
visualize important feature allows to judge the individual classification



(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

Tf, python code: https://github.com/marcotcr/lime -> image tutorial
https://github.com/marcotcr/lime/blob/master/doc/notebooks/Tutorial%20-%20Image%20Classification%20Keras.ipynb

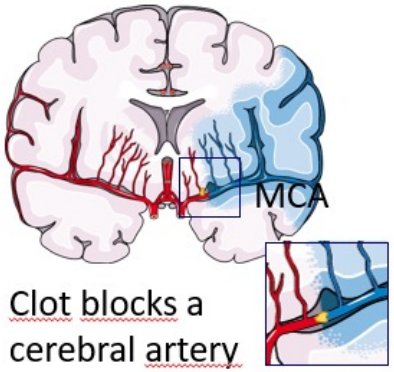
xAI in a medical research project



xAI delivers an «explanation» for the prediction by highlighting a region in the input image, that was identified (posthoc) to be responsible for the predichtion.

Stroke application: Motivation

Ischemic stroke (88%)



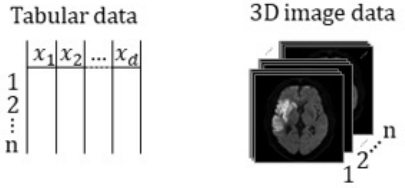
Clot blocks a cerebral artery

Large vessel occlusion stroke

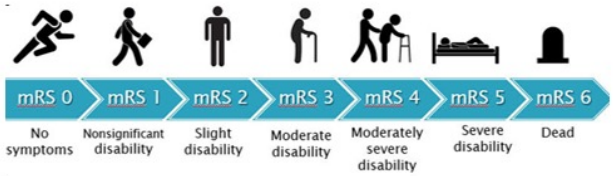
1.9 million neurons/min.

Time is brain!

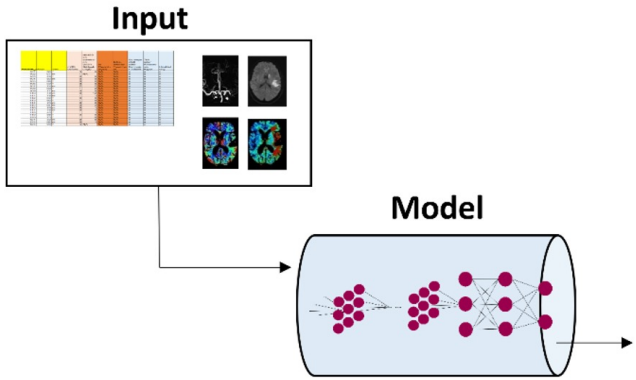
Semi-structured patient data ($n = 407$)



ordinal outcome



binary outcome

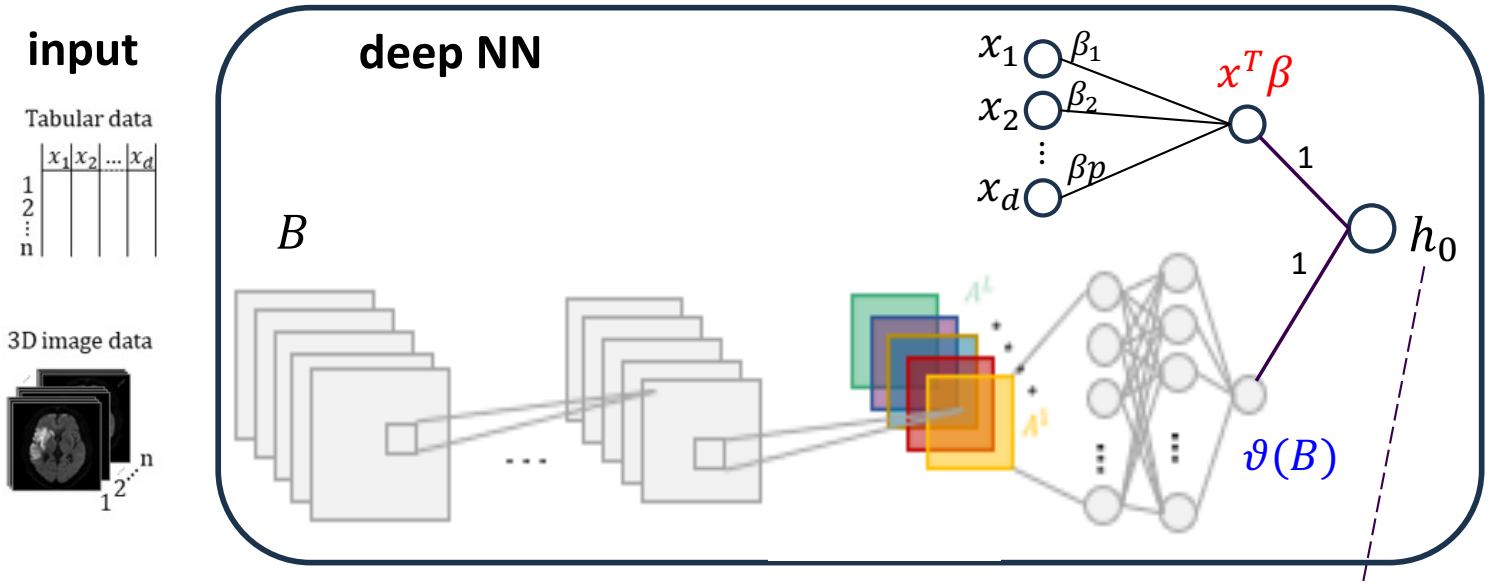


Treat or not?

Why?

Is it certain?

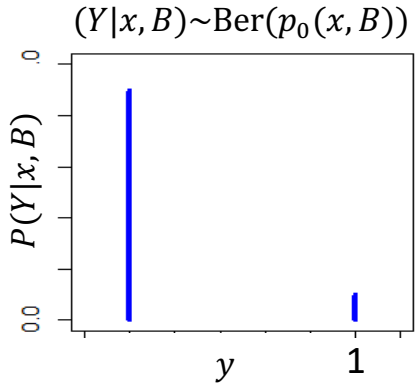
Architecture of a multi-modal NN for a binary Y

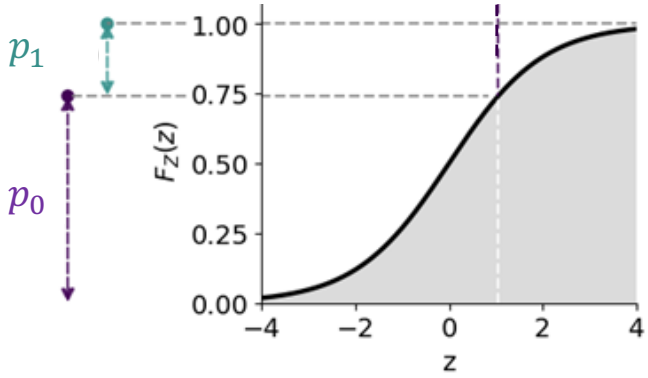


$p_0 = P(\text{favorable}|x, B) = \sigma(h_0) = 0.8$

$p_1 = 1 - p_0 = P(\text{unfavorable}|x, B) = 0.2$

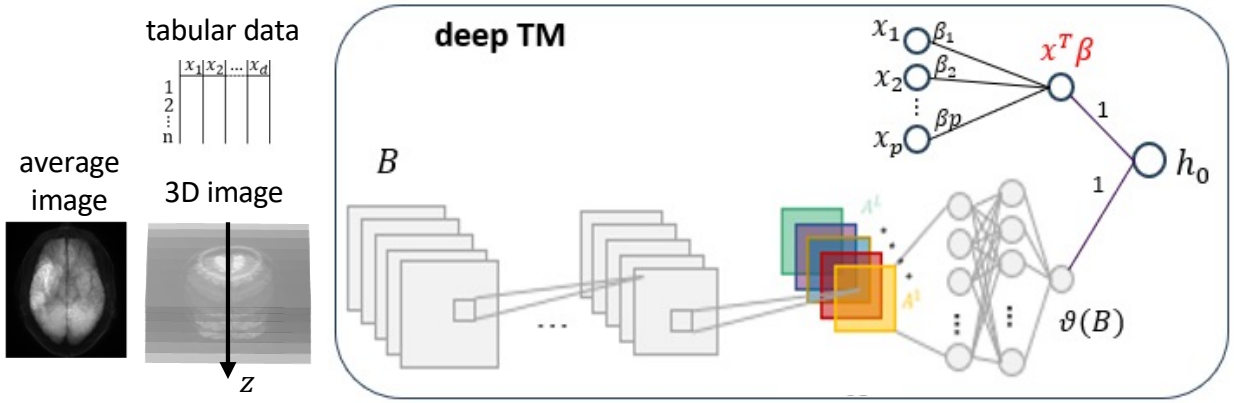
$z_0 = h_0 = h(Y = 0|B, x) = \vartheta(B) - x^T\beta$





Occlussion for multi-modal NN in stroke application

Trained deep TM

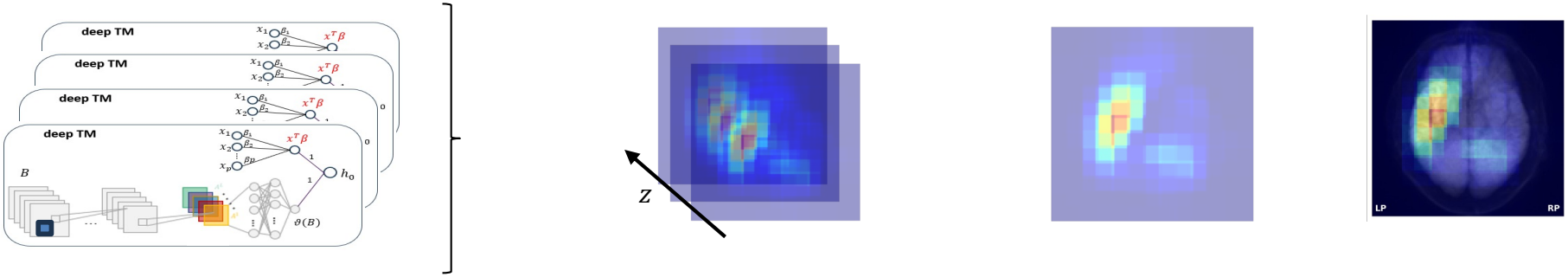


$p_0 = F_z(h_0) = 0.2$

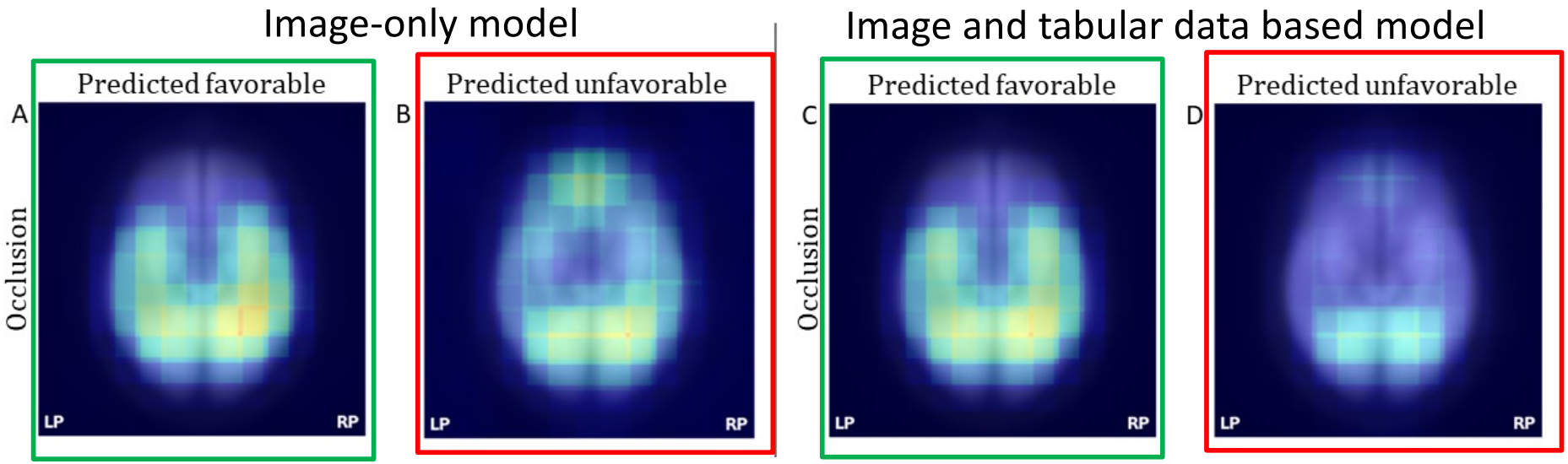
$\boldsymbol{p_1 = 1 - p_0 = 0.8}$

Occlusion

Predict probabilities with different occluding masks → Determine reduction of probability of predicted class due to occlusion per voxel → Averaging along z-direction → Overlay



Explanation maps for patients predicted as un/favorable

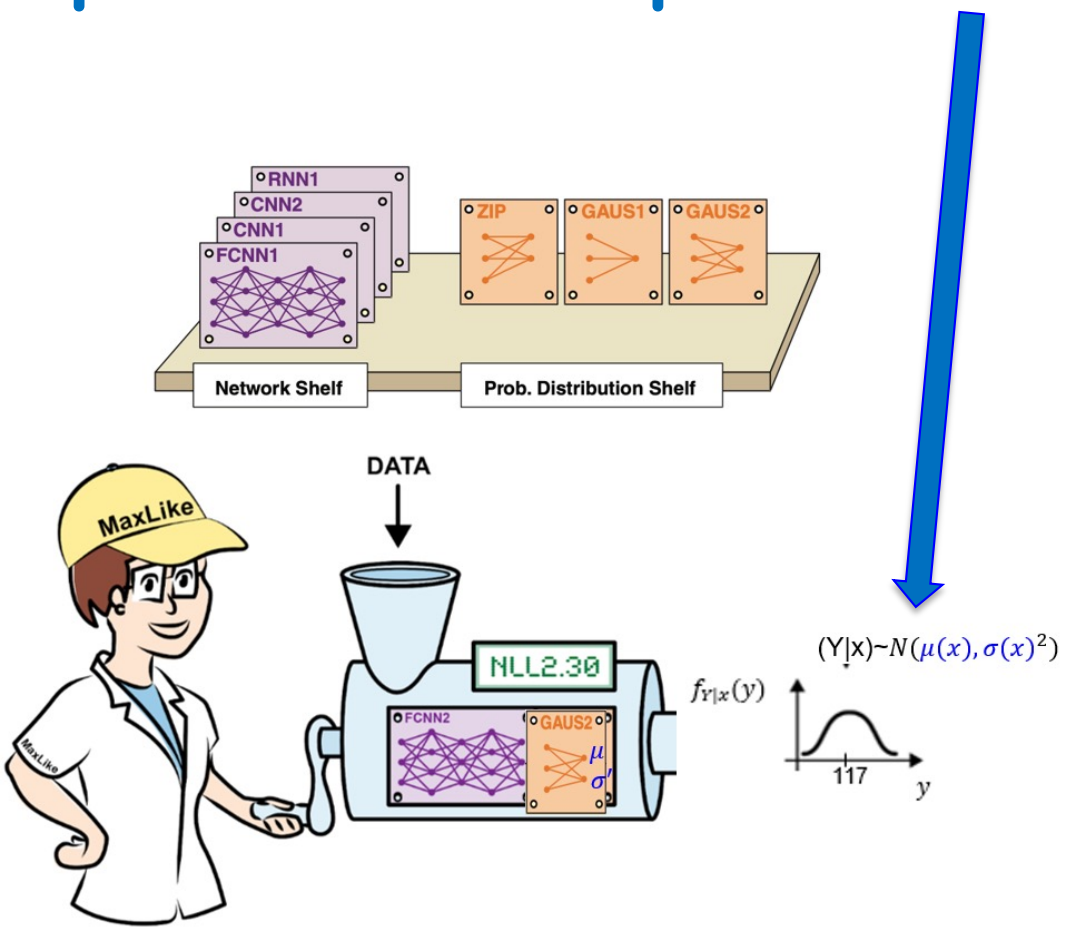


→ xAI for image only based model highlights frontal lobe if bad outcome is predicted

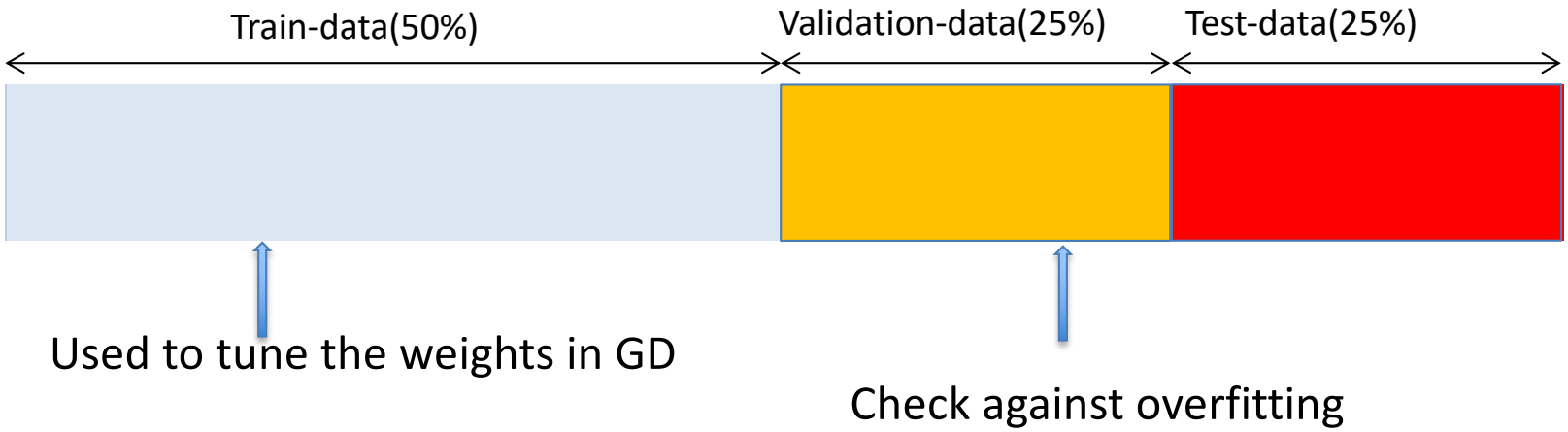
→ Neurologist know that the frontal lobe indicates the age of the patient

→ When providing age as tabular data in addition to the image to the data, the frontal lobe is not anymore hightlighted

Evaluating the quality of the probabilistic predictions



Model evaluation is done on Validation and Test Set



Best practice: Lock an extra test data set away, and use it only at the very end, to evaluate the chosen model, that performed best on your validation set.
Reason: **When trying many models, you probably overfit on the validation set**.

Determine performance metrics, such as NLL, Brier score, AUC or MSE, to evaluate the predictions **on new validation or test data**

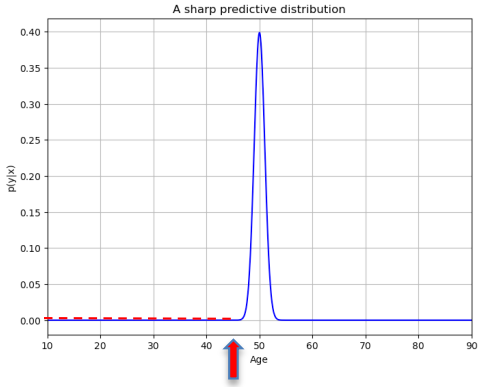
Evaluation of probabilistic models

Which model is better / evaluating probabilistic models

Predicted age distribution

Meryl Streep, 41y

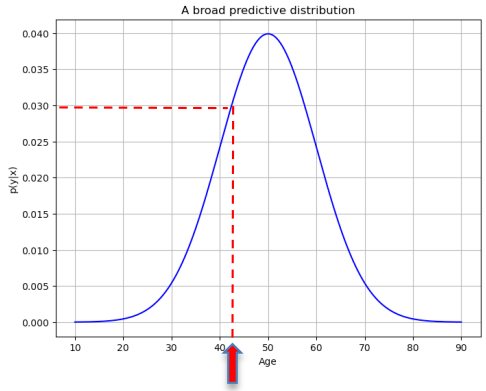
Model 1



$p(y|x)$

$L_{\text{streep}} = 0.0001$

Model 2



$p(y|x)$

$L_{\text{streep}} = 0.03$

Which model is better?

For the evaluation, we don't care if $p(y|x)$ is from a NN or from another model.
Evaluation of $p(y|x)$ has long tradition, e.g. in metrology (Good 1952).

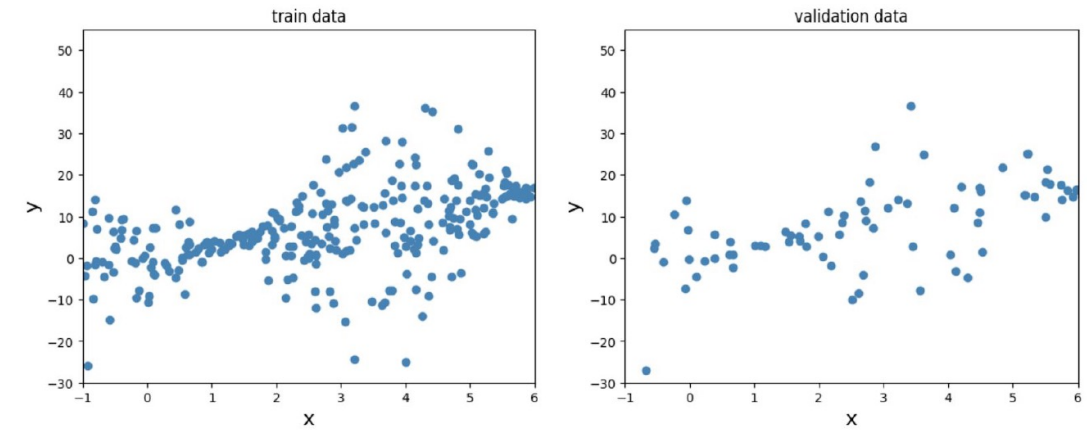
Is the test Likelihood or NLL suited for evaluation?

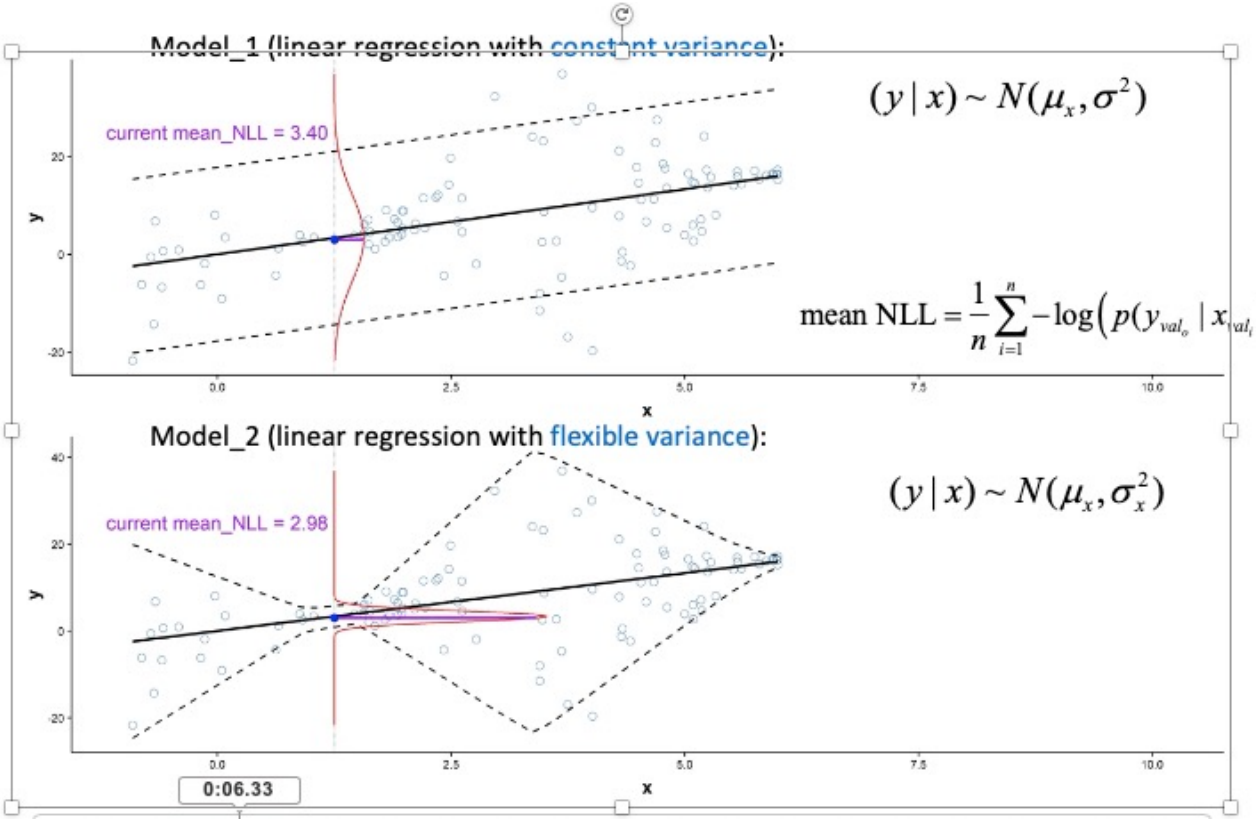
- The best measures for evaluation of a probabilistic model:
  - test **NLL** (negative-log-likelihood) for continuous and discrete outcomes
  - test log-likelihood (aka log-score) for continuous and discrete outcomes
  - … some few other "proper" scores you will see later

- Why?
  - **the objective in the training was to minimize the NLL**
    → the better the NLL the better the model!!
    Otherwise we should have optimized another measure ;)
  - The NLL (or log-score) is "strictly proper" and the only smooth and local score which gets only optimal if the predicted distribution matches the data generating distribution.

- Why not?
  - People like to see accuracy, AUC, MSE… → provide these measures additionally even though they are not "proper" and the model was not optimized for this purpose.

Which of the two probabilistic models is better?





Model 1 (constant variance)

$$(y \mid x) \sim N(\mu_x, \sigma^2)$$

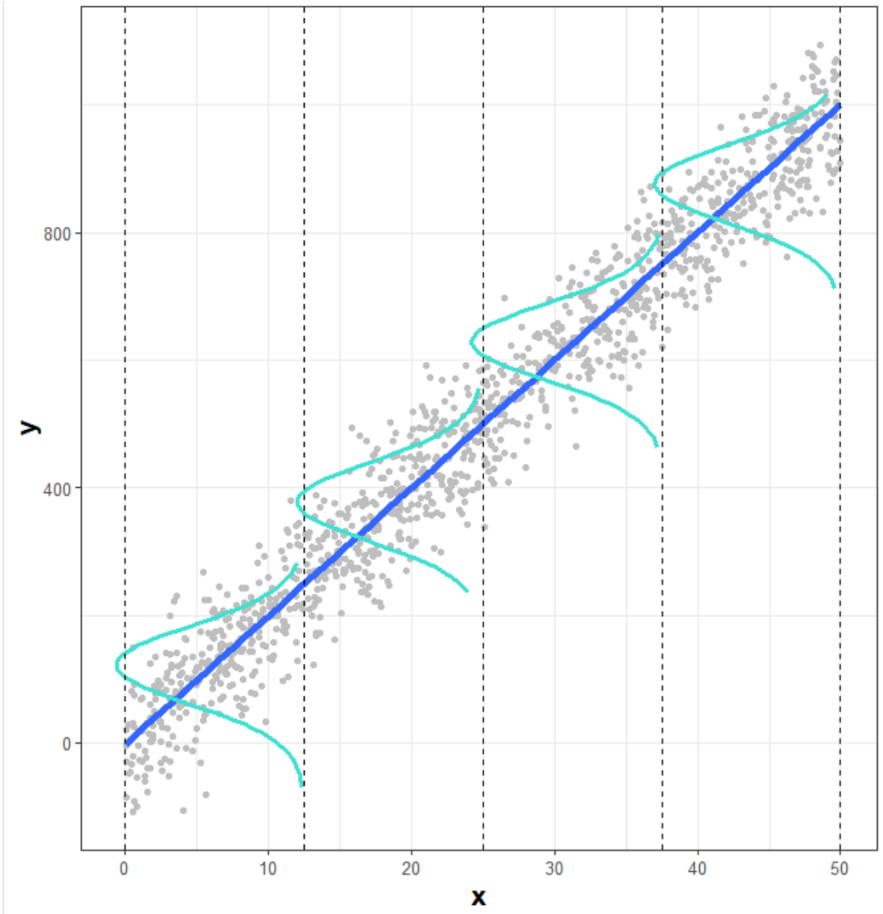
Model 2 (flexible variance)

$$(y \mid x) \sim N(\mu_x, \sigma_x^2)$$

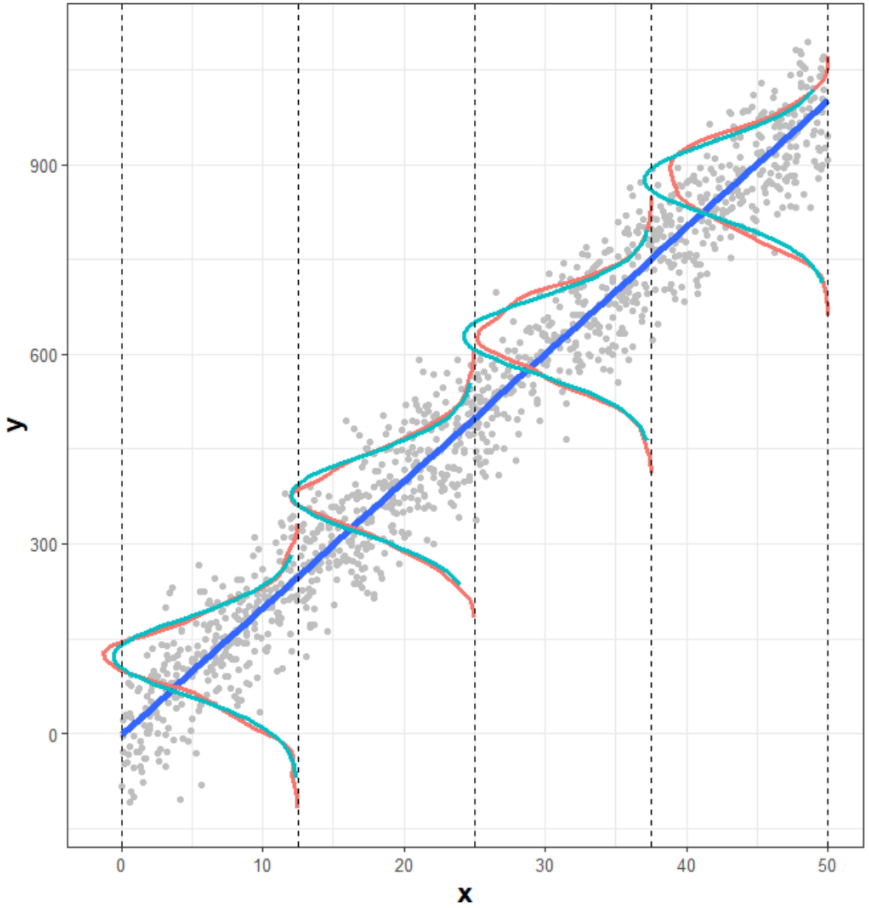
https://www.youtube.com/watch?v=QeWd0g_UzZY

Do predicted and observed outcome distribution match?

Validation data along with predicted outcome distribution (Gauss with const σ)



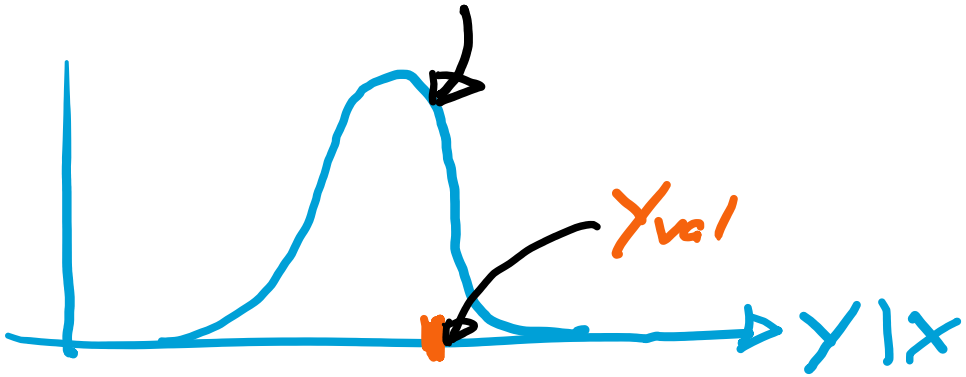
Validation data along with predicted and observed outcome distribution



A large validation data set is needed to ensure underlying assumption:
observed distribution = distribution of data generating distribution

What is a good "score" to evaluate a probabilistic model?

- For evaluation we have
  - Values $(x_{\text{val}}, y_{\text{val}})$ from DGP (or validation set)
  - Predicted distribution $p_{\text{pred}}(Y|x_{\text{val}})$



In the following:

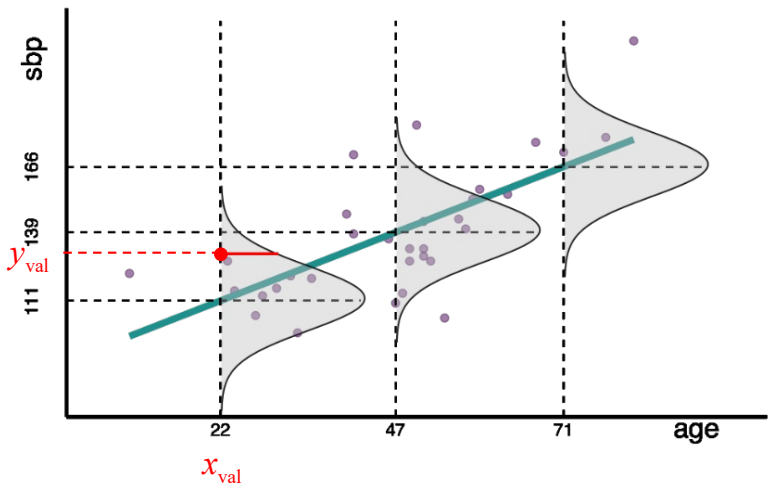
1. We create a "score" which take as input $p_{\text{pred}}(Y|x)$ for a single pair $(x_{\text{val}}, y_{\text{val}})$

2. Evaluate if the the score gets best if the predicted distribution p_{pred} matches the distribution of the data generating process (DGP) by using $(x_{\text{val}}, y_{\text{val}}) \sim DGP$

Scores to evaluate probabilistic prediction models

- We need validation data: $(x_{\text{val}}, y_{\text{val}})$
- We need predicted outcome distribution, given x: $p_{\text{pred}}(Y|x)$
- The score *S* takes *one instance* and yields a real number (smaller is better)

  $S(p_{\text{pred}}(Y|x_{\text{val}}), y_{\text{val}})$



Example 1: NLL (aka log-score):

$$S_{NLL}(p_{\text{pred}}(Y|x_{\text{val}}), y_{\text{val}}) = -\log(p_{\text{pred}}(y_{val}|x_{\text{val}}))$$

Example 2: weighted MSE:

$$S_{MSE}(p_{\text{pred}}(Y|x_{\text{val}}), y_{\text{val}}) = \int (y_{\text{val}} - y)^2 p_{\text{pred}}(Y|x_{\text{val}})\ \text{dy}$$

Empirical loss as average score

- If we use a validation set with n instances (x_{val_i}, y_{val_i}) to evaluate the model, the average score is used as empirical loss:

$$\text{empirical loss} = \frac{1}{n}\sum_{i=1}^{n} S\left(p_{\text{pred}}(y \mid x_{val_i}), y_{val_i}\right)$$

- The empirical loss approximates the expected loss:

$$\text{expected loss} = \int_{y} S\left(p_{\text{pred}}(y \mid x'), y'\right) \cdot p_{true}(y', x') \; dx'dy'$$

p_{pred} : predicted distribution
p_{true} : data generating distribution

Local scores

A score is local if the predicted distribution is evaluated only at the actual observed outcome of the validation data

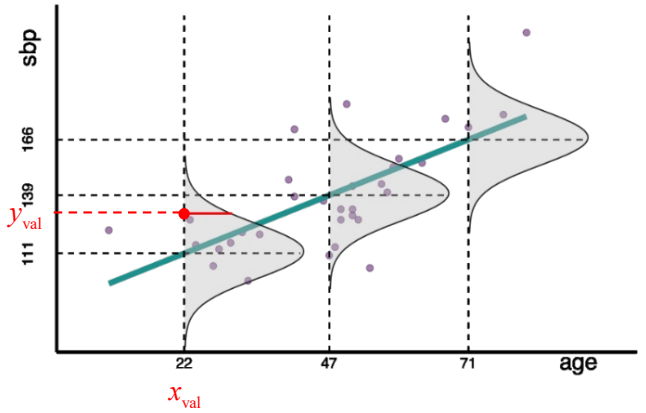
$$S(p_{\text{pred}}(Y|x_{\text{val}}), y_{\text{val}}) = S(p_{\text{pred}}(y_{\text{val}}|x_{\text{val}}), y_{\text{val}})$$

Example 1: NLL (aka log-score, ignorance):

$$S_{NLL}(p_{\text{pred}}(Y|x_{\text{val}}), y_{\text{val}}) = -\log(p_{\text{pred}}(y_{\text{val}}|x_{\text{val}}))$$

Example 2: linear score

$$S_{NL}(p_{\text{pred}}(Y|x_{\text{val}}), y_{\text{val}}) = -p_{\text{pred}}(y_{\text{val}}|x_{\text{val}})$$



Proper Scores

For a proper score holds:

The expected value of a *proper score* takes its minimal (optimal) value, if predicted distribution p_{pred}=p_{true} data generating distribution

The expected value of a *strictly* proper score takes its minimal value, *only* if predicted distribution p_{pred}=p_{true} data generating distribution

$$\int\int S(p_{true}(y|x'),y')p_{true}(y',x')dy'dx' < \int\int S\left(p_{pred}(y|x'),y'\right)p_{true}(y',x')dy'dx' \text{ if } p_{pred} \neq p_{true}$$

The score with true conditional distribution

The score with predicted cond. distribution

The log-score is strictly proper – proof for nerds

To show: $\int\int S(p_{pred}(y|x'),y')p_{true}(y',x')dy'dx' > \int\int S(p_{true}(y|x'),y')p_{true}(y',x')dy'dx'$

$$\int_{x,y} S\left(p_{pred}(y\,|\,x'),y'\right)\cdot p_{true}(y',x')\;\;dx'dy' \;=\; \int_{x,y} S\left(p_{true}(y\,|\,x'),y'\right)\cdot p_{true}(y',x')\;\;dx'dy' \;+$$

$$\underbrace{\left\{\int_{x,y} S\left(p_{pred}(y\,|\,x'),y'\right)\cdot p_{true}(y',x')\;\;dx'dy' \;-\; \int_{x,y} S\left(p_{true}(y\,|\,x'),y'\right)\cdot p_{true}(y',x')\;\;dx'dy'\right\}}_{>0 \quad \text{for strictly proper scores } S}$$

Proof that NLL is strictly proper $S_{NLL}\left(p(y\,|\,x_{val}),y_{val}\right) = -\log\left(p(y_{val}\,|\,x_{val})\right)$

$$\int S_{\text{NLL}}\left(p_{\text{pred}}(y\,|\,x'),y'\right)\cdot p_{\text{true}}(y',x')\;\;dx'dy' \;-\; \int S_{\text{NLL}}\left(p_{\text{true}}(y\,|\,x'),y'\right)\cdot p_{\text{true}}(y',x')\;\;dx'dy'$$

$$= \int -\log\left(p_{\text{pred}}(y'\,|\,x')\right)\cdot \underbrace{p_{\text{true}}(y',x')}_{=p_{true}(y'|x')\cdot p(x')}\;\;dx'dy' \;-\; \int -\log\left(p_{\text{true}}(y'\,|\,x')\right)\cdot \underbrace{p_{\text{true}}(y',x')}_{=p_{true}(y'|x')\cdot p(x')}\;\;dx'dy'$$

$$= \int \log\left(\frac{p_{\text{true}}(y'\,|\,x')}{p_{\text{pred}}(y'\,|\,x')}\right)\cdot p_{\text{true}}(y'\,|\,x')dy'\,p_{\text{true}}(x')\;\;dx' = \text{KL}\left(p_{\text{true}}(\cdot\,|\,x);p_{\text{pred}}(\cdot\,|\,x)\right) > 0 \qquad \forall\; p_{pred} \;\neq\; p_{true}$$

The linear score is not proper – proof for nerds

The linear score is not proper, meaning p_{true} does not yield the best expected score.

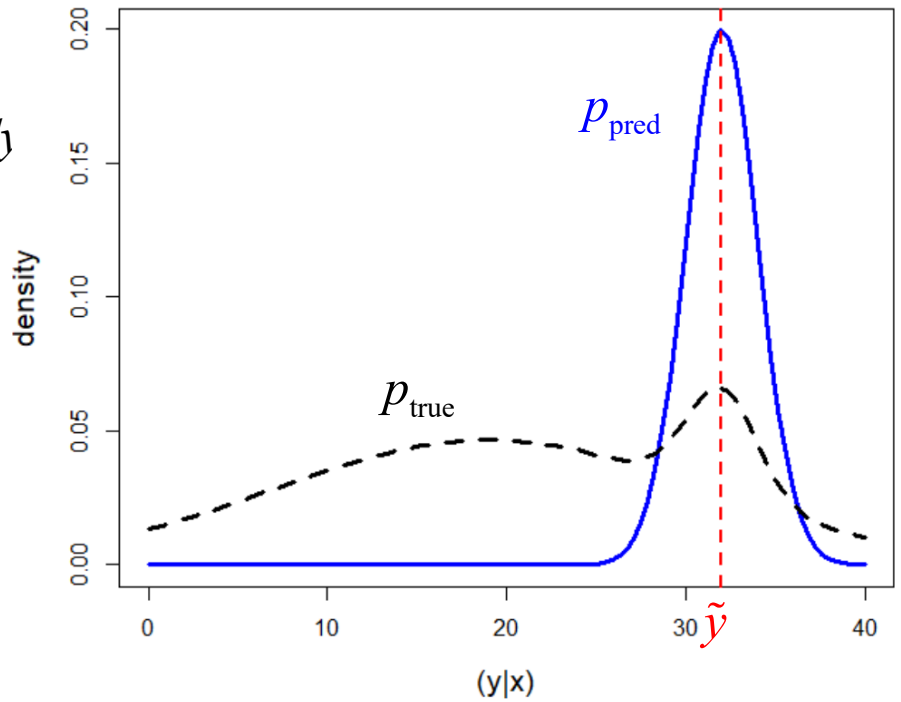
$$S_{\text{lin}}\left(p(y \mid x_{val}), y_{val}\right) = -p(y_{val} \mid x_{val})$$

$$E_{p_{\text{true}}}(S_{p_{\text{pred}}}) = \int_{x,y} S\left(p_{true}(y \mid x'), y'\right) \cdot p_{true}(y', x') \; dx'dy$$

$$= \int_{x,y} -p_{true}(y' \mid x') \cdot p_{true}(y', x') \; dx'dy'$$

If p_{true} is not constant, then there is a $\tilde{y}$ higher than mean probability:

$$-p_{true}(\tilde{y} \mid x') < E_{p_{\text{true}}}(S_{p_{\text{true}}})$$



Proof:

Construct p_{pred} that scores better than p_{true}:

$$p_{\text{pred}}(\tilde{y} \mid x') = \frac{1}{\sigma} \cdot \text{kernel}\left(\frac{y' - \tilde{y}}{\sigma}\right)$$

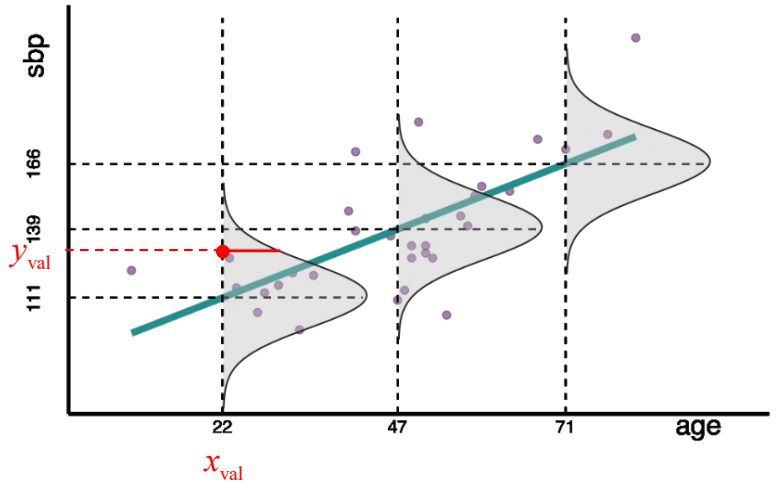
$$E_{p_{\text{true}}}(S_{p_{\text{pred}}}) = \int_{x,y} -p_{\text{pred}}(y' \mid x') \cdot p_{true}(y', x') \; dx'dy' \;\rightarrow\; -p_{\text{true}}(\tilde{y} \mid x') \;<\; E_{p_{\text{true}}}(S_{p_{\text{true}}})$$

The uniqueness of the log-score

It is provable that the **log-score** is the **only smooth, proper and local** score for continuous variables

(Bernardo, J. M., 1979: Expected information as expected utility. *Ann. Stat.,* **7,** 686–690)

$$S_{\mathrm{NLL}}\left(p(y \mid x_{val}), y_{val}\right) = -\log\left(p(y_{val} \mid x_{val})\right)$$



Scoring Probabilistic Forecasts: The Importance of Being Proper

JOCHEN BRÖCKER

Centre for the Analysis of Time Series, London School of Economics, London, United Kingdom

LEONARD A. SMITH

Centre for the Analysis of Time Series, London School of Economics, London, and Pembroke College, Oxford University, Oxford, United Kingdom

(Manuscript received 4 November 2005, in final form 23 May 2006)

ABSTRACT

Questions remain regarding how the skill of operational probabilistic forecasts is most usefully evaluated or compared, even though probability forecasts have been a long-standing aim in meteorological forecasting. This paper explains the importance of employing proper scores when selecting between the various measures of forecast skill. It is demonstrated that only proper scores provide internally consistent evaluations of probability forecasts, justifying the focus on proper scores independent of any attempt to influence the behavior of a forecaster. Another property of scores (i.e., locality) is discussed. Several scores are examined in this light. There is, effectively, only one proper, local score for probability forecasts of a continuous variable. It is also noted that operational needs of weather forecasts suggest that the current concept of a score may be too narrow; a possible generalization is motivated and discussed in the context of propriety and locality.

https://journals.ametsoc.org/doi/full/10.1175/WAF966.1

Prominent Scores for binary classifiers

Definition 9.9 (Scoring rules for binary predictions) Let $Y \sim \mathrm{B}(\pi)$ be the predictive distribution for a binary event, i.e.

$$f(y) = \begin{cases} \pi & \text{for } y = 1, \\ 1 - \pi & \text{for } y = 0. \end{cases}$$

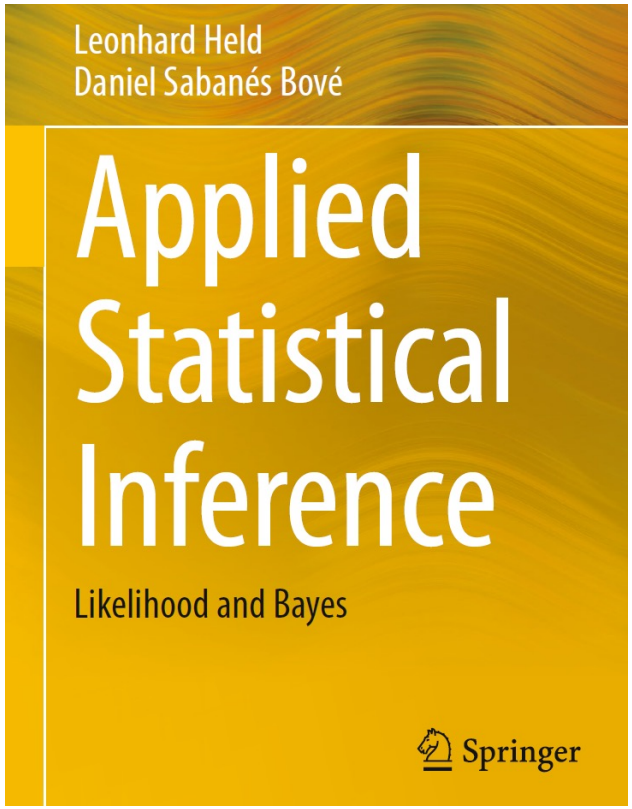
The *Brier score* BS, the *absolute score* AS and the *logarithmic score* LS are defined as

Strictly proper: $\mathrm{BS}\big(f(y), y_o\big) = (y_o - \pi)^2,$

Not proper: $\mathrm{AS}\big(f(y), y_o\big) = |y_o - \pi|$ and

Strictly proper: $\mathrm{LS}\big(f(y), y_o\big) = -\log f(y_o),$

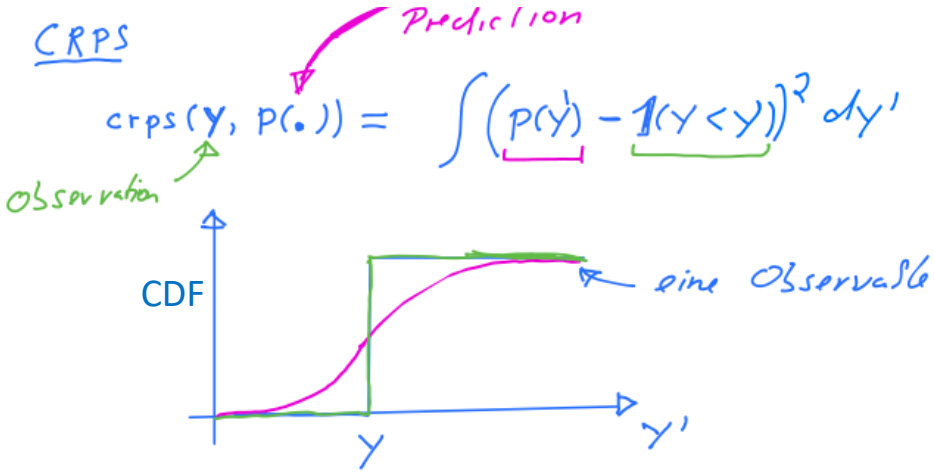
respectively.



Remark: For binary classification, the log score is not the only strictly proper score.

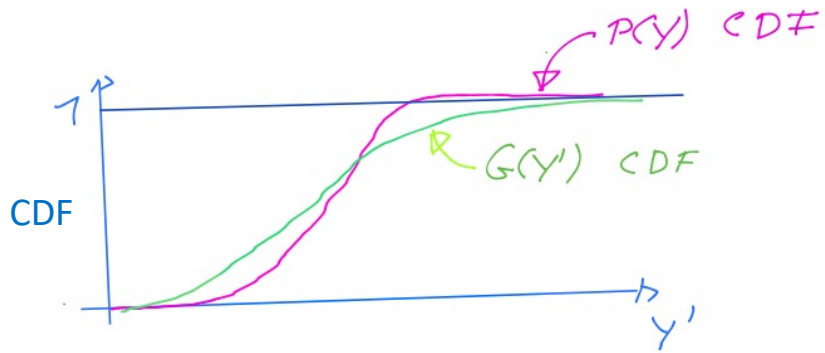
Other strictly (but non-local) proper scoring rules

In forecasting communities (power consumption / weather / …)
Continuous Ranked Probability Score is often used CPRS. $P(y')$ is CDF



Hard to compute (integral) in general

- Associated Score divergence (**expectations / many variables**)*
  - $D_{CPRS}(P,G) = \int \left(P(y) - G(y)\right)^2 dy$



*See e.g. https://www.stat.berkeley.edu/~ryantibs/statlearn-s23/lectures/calibration.pdf for the calculation

Non-proper scoring rules

The following performance measures are very popular but not proper:

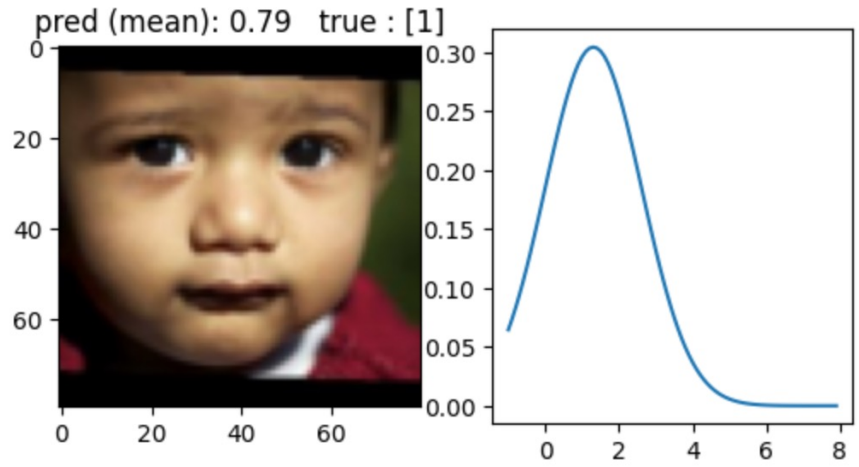
- Accuracy – for categorical outcome
- Kappa – for categorical outcome
- Area under the curve (AUC) - for binary outcome
- Mean Square Error (MSE) – for continuous outcome
- Mean Absolut Error (MAE) – for continuous outcome
- …

Exercise Probabilitic Age Prediction

Notebook: 03

https://github.com/tensorchiefs/dlwbl_eth25/blob/master/notebooks/03_uk_faces.ipynb

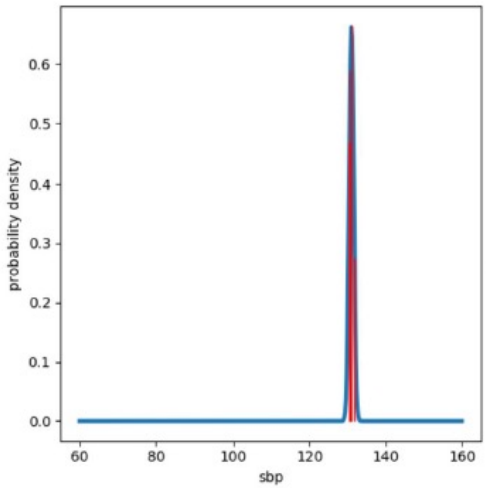
Work through NB until here:
(don't start with the PIT exercise)



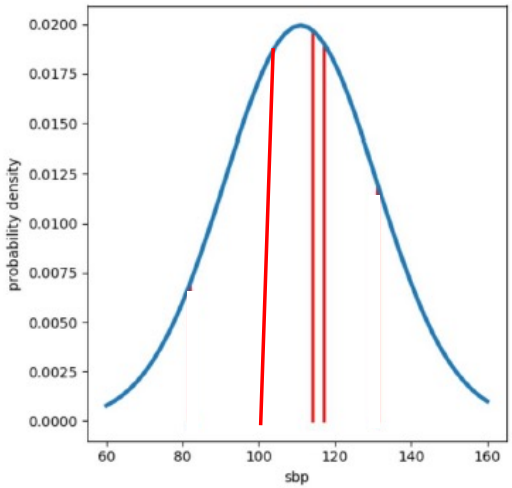
NLL and MAE on the test set

```
# Since the NLL is also the loss function, we can also use the evaluate function of keras.
```

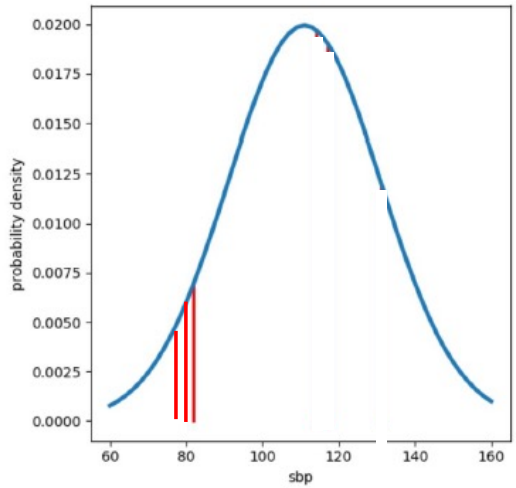
Calibration & Sharpness



calibrated
&
sharp



calibrated
&
not sharp

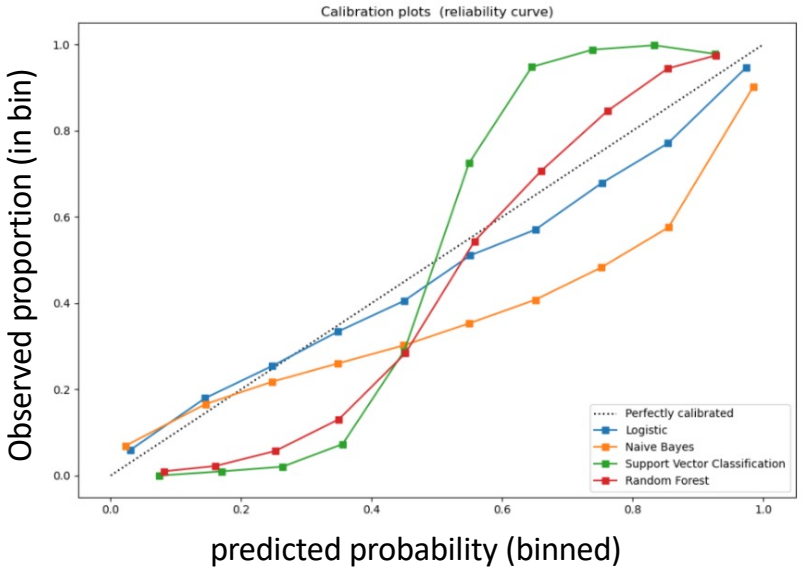


not calibrated
&
Not sharp

Calibration

- Consider the classification of tomorrows weather 
  - Y: Weather ∈ {Sunny, Mixed, Rainy}
- Example
  - Classifier $p_{\text{pred}}(Y = \text{sunny}) = 0.7$
  - In exactly 70% of those cases, it should be sunny
  - Check: Take all days with $p_{\text{pred}}(Y = \text{sunny}) = 0.7$ and check if on 70% of these days the weather was sunny. Do this which each probability bin.
- Calibration plot

$$\frac{\#(y_{\text{obs}} = \text{"rain"})}{\#y_{\text{obs}}}$$

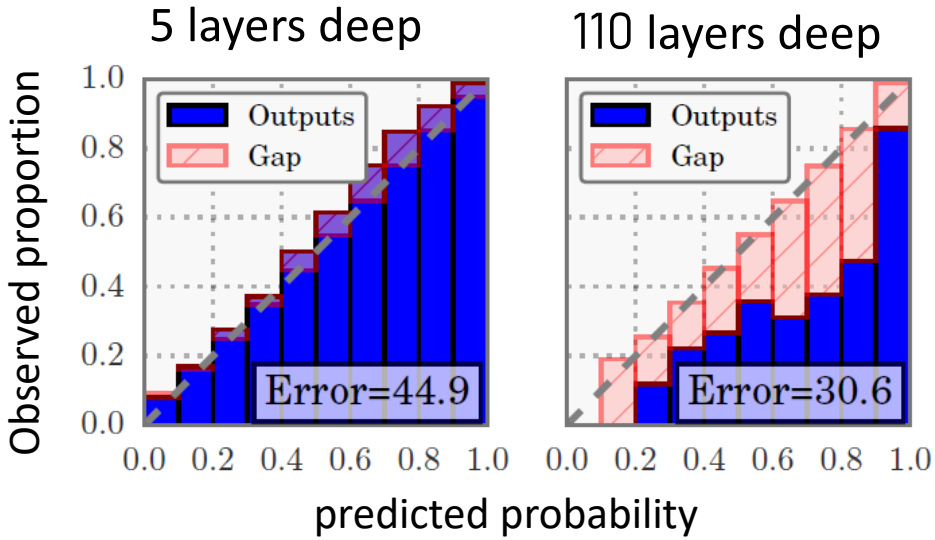


$p_{\text{pred}}(Y = \text{"rain"})$

Do NN produce calibrated probabilities?

Guo et al. (2017)
[On Calibration of Modern NN]

The deeper CNNs get
- the fewer miss-classifications
- the less well calibrated they get



Good news:
deep NN can be “recalibrated” and then we get calibrated probabilities.

Guo et all https://arxiv.org/pdf/1706.04599.pdf

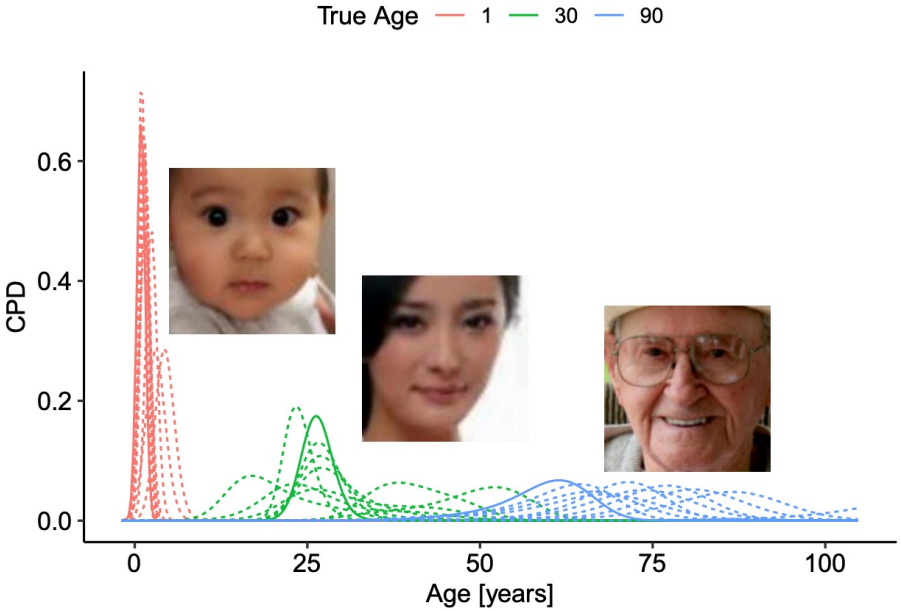
Calibration vs Sharpness

- E.g. in Konstanz, the rain probability for the next day is 20% (over a year)
- A Classifier that predicts $p_{rain} = 0.2$ for each day is well calibrated
- A good classifier should make sharp classifications (prob of rain close to 0% or 100%) but still be calibrated

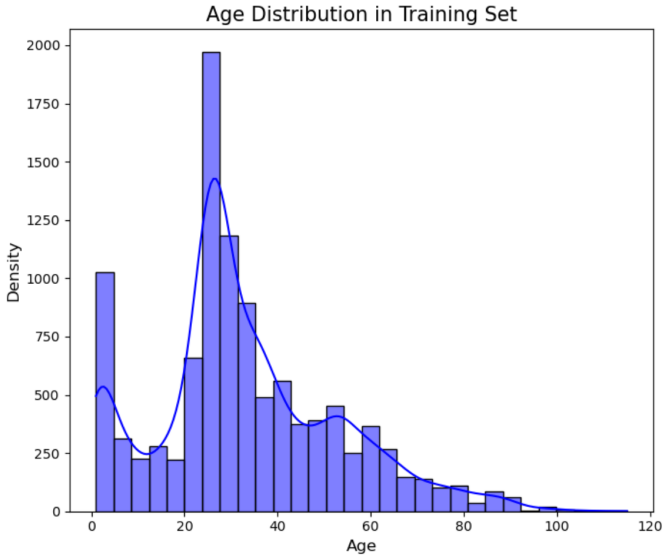
Example

	Good Classifier Prediction p_{rain}	Const Classifier Prediction p_{rain}	Actual Rain
Day 1	0.9	0.2	Yes
Day 2	0.05	0.2	No
Day 3	0.04	0.2	No
Day 4	0.03	0.2	No
Day 5	0.95	0.2	Yes

Calibration vs Sharpness $p(Y|x)$



Sharper predictor depends on x.
But is it calibrated?



A very uninformative but calibrated forecaster would just predict the population distribution (for every) x.

Is there a score that takes into account calibration and sharpness?

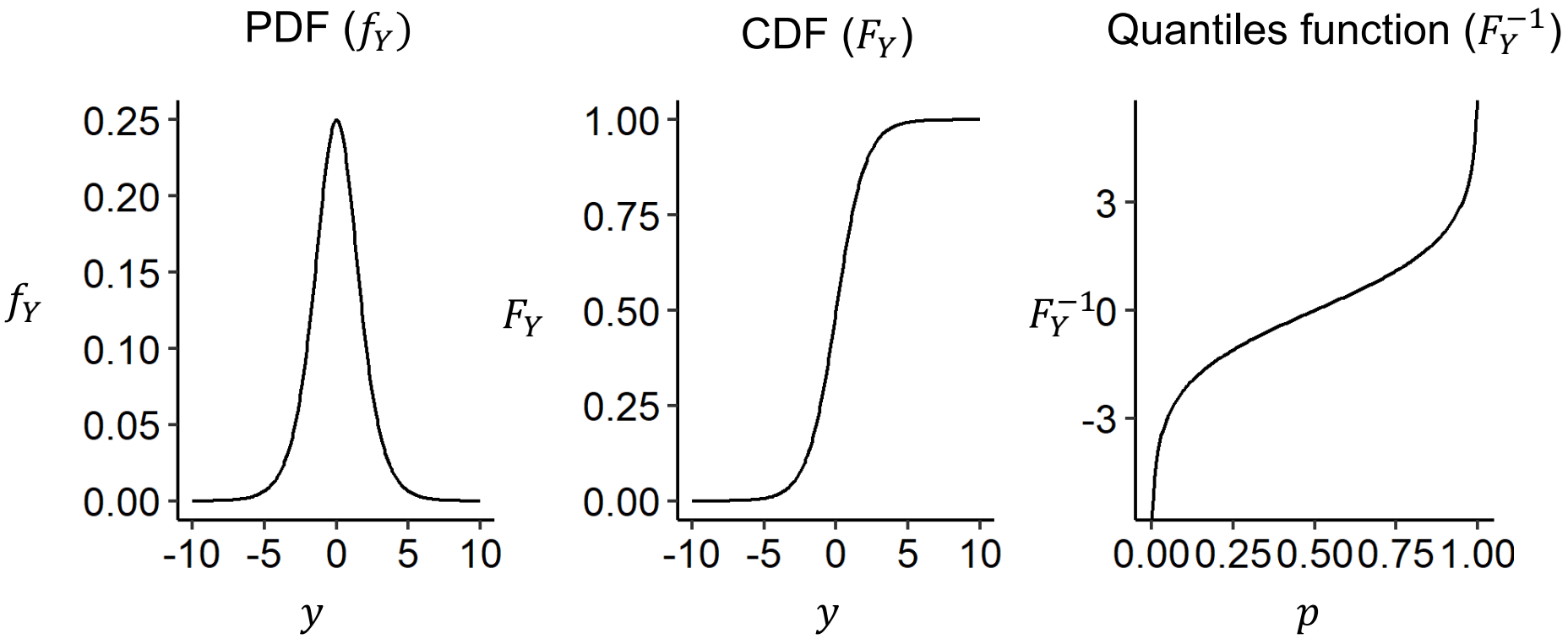
Yes, there is: the **NLL** ☺

NLL as general cure-all in probabilistic modeling

- Maximize likelihood ↔ minimize negative log-likelihood (NLL)
- The log-score (NLL) is strictly proper score for regression.
- The log-score (NLL) is also strictly proper for classification models.
- To train a probabilistic model: minimize NLL!
- To evaluate or compare probabilistic models: use the validation NLL!

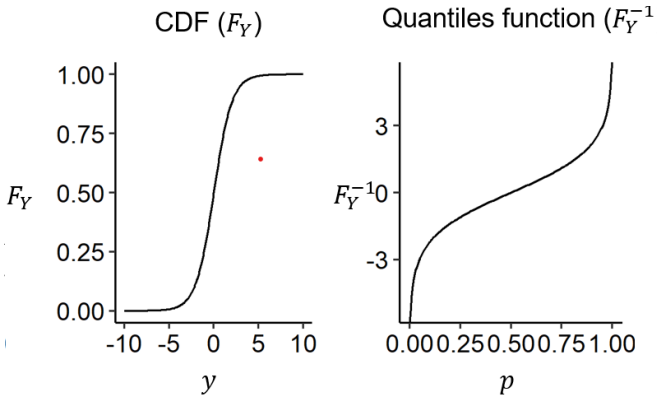
PIT
(Probability Integral Transform)
Histograms to visually assess calibration

Recall the quantile function

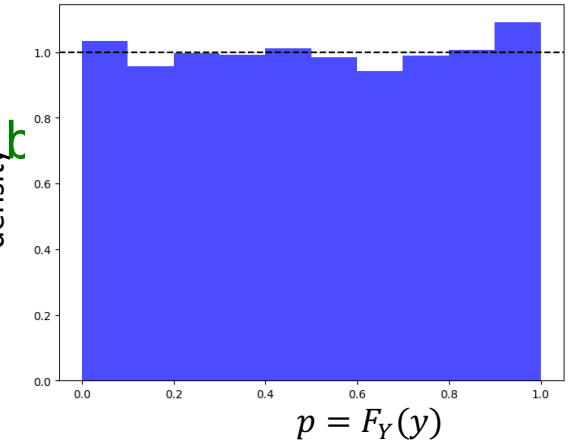


Preliminaries: Quantiles are uniformly distributed

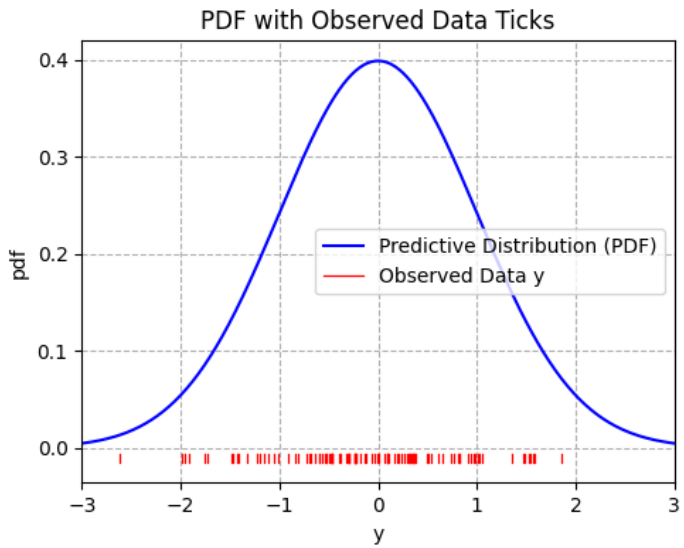
- The p −quantiles q_p with correspond to a value $y \in \text{range}(Y)$ for which a proportion $\text{p} \in [0,1]$ of all observations drawn from $Y \sim P_Y$ are smaller than q_p.
  - Between the 0.10 quantile $q_{0.1}$ and the 0.11 quantile $q_{0.11}$ there is 1% of all data
  - Between the 0.52 quantile $q_{0.52}$ and the 0.53 quantile $q_{0.53}$ there is 1% of all data
  - …
- For the quantile q_p holds: $F_Y(q_p) = p$
  $\rightarrow q_p = F_Y^{-1}(p)$, where F_Y is the CDF.
- The CDF-values of y-values thar are randomly drawn from a valid distribution P_Y yield PIT-values p^p which follow a uniform distribution

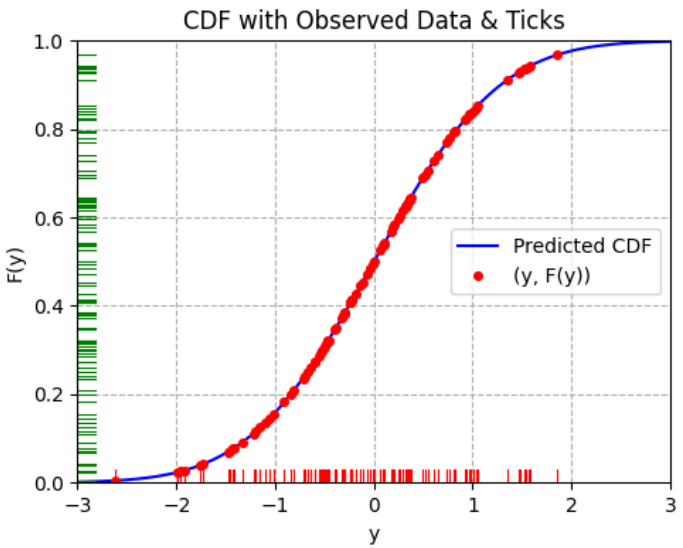



```
import scipy.stats as stats
dist = stats.norm(1.2, 1) # Normal distrib
q = dist.rvs(10**4) # draw random samples
p = dist.cdf(q) # Calculate p: the CDF of
q-values
plt.hist(p, bins=10, density=True, color='blue',
```



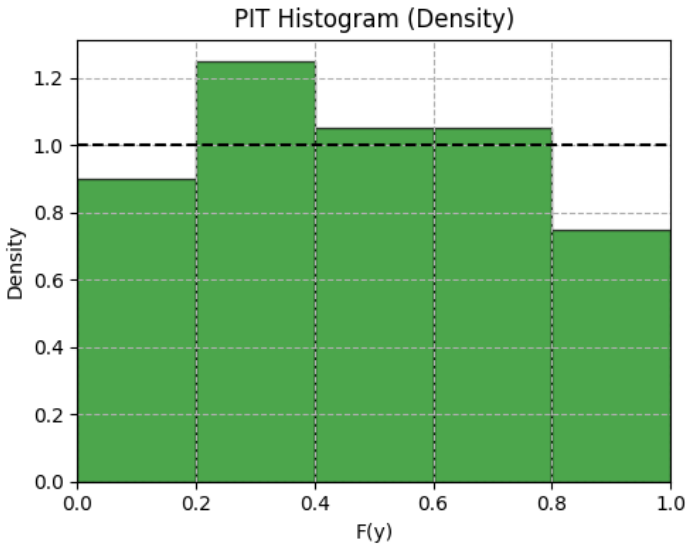
Calibration Plot / PIT Histogram (ideal case)





If the predicted outcome distribution matches the true outcome distribution from the Data Generating Process (= observed outcome distribution)
→ the PIT shows a uniform distribution.

Data ~ N(0,1)
Pred. Distribution ~ N(0, 0.5)

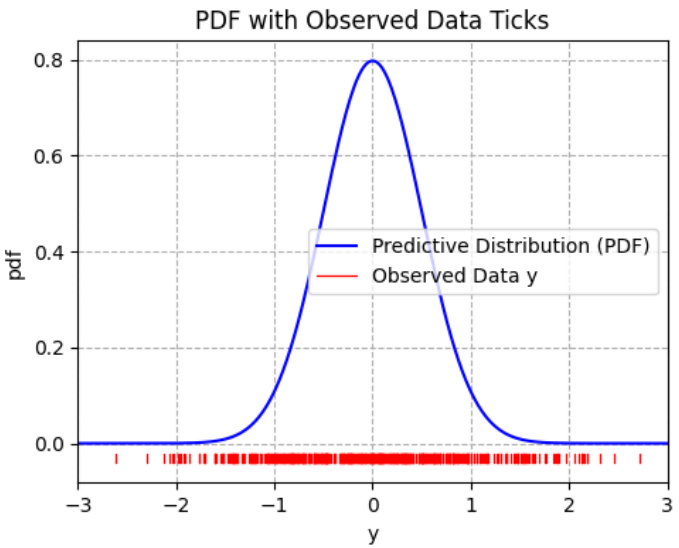


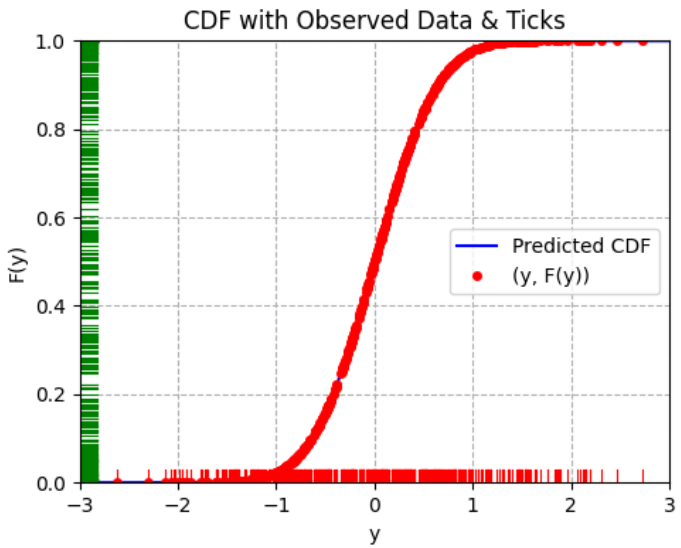
No systematic derivation from uniformity

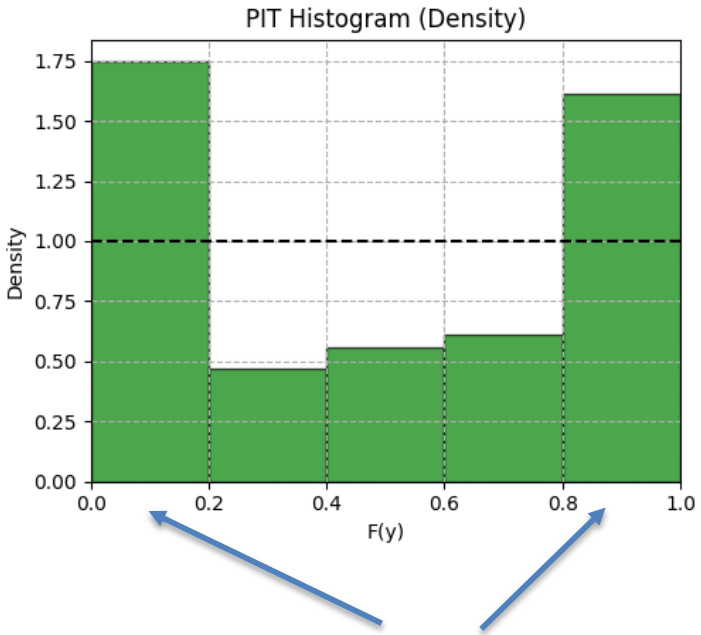
Note that the predicted outcome distribution might change depending on x.
As long as it matches the DGP, the PIT histograms are always uniform

https://github.com/ioskn/mldl_htwg/blob/master/dl_prob_1/code_for_slides.ipynb

Calibration Plot / PIT Histogram (over optimistic)





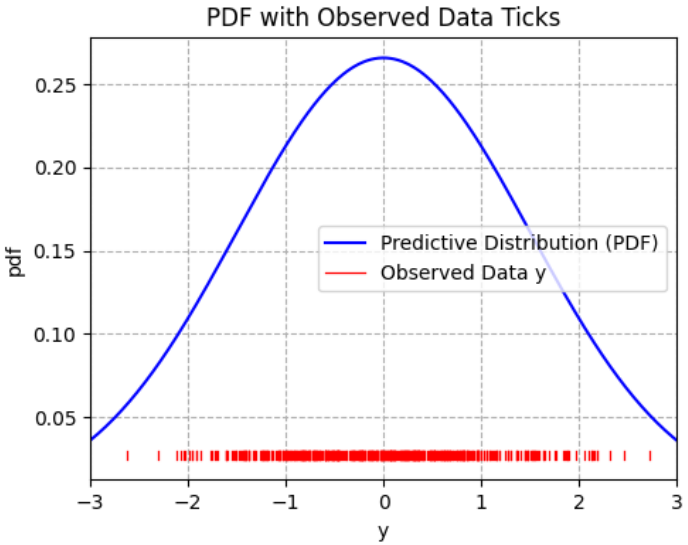


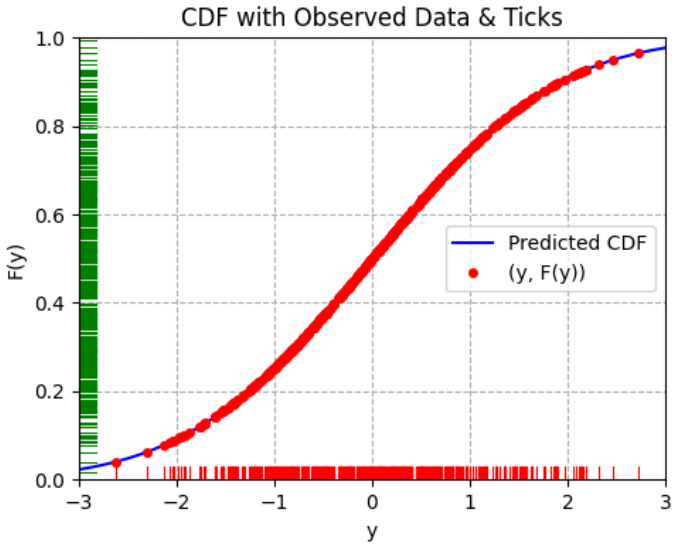
The predicted outcome distribution underestimates the true uncertainty of the DGP distribution (to confident)
→ PIT shows U-form

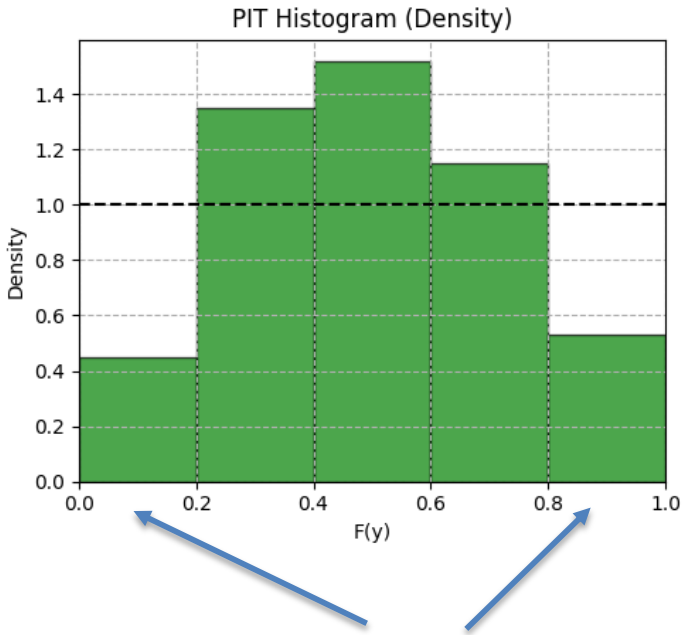
Data ~ N(0,1)
Pred. Distribution ~ N(0, 0.5)

More observations in low/high quantiles, than forecasted.

Calibration Plot / PIT Histogram (too conservative)







The predicted outcome distribution overestimates the uncertainty of the true distribution (to conservativ)
→ PIT shows peak

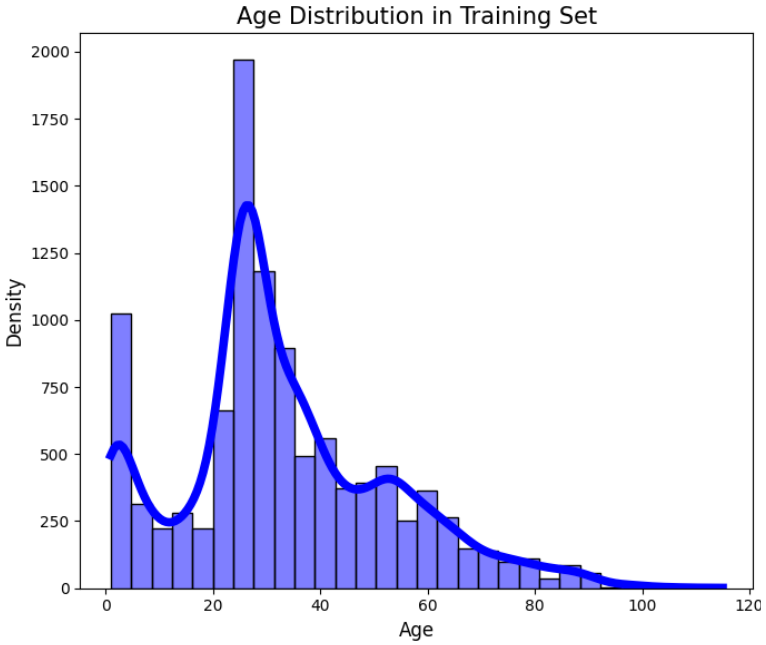
Data ~ N(0,1)
Pred. Distribution ~ N(0, 1.5)

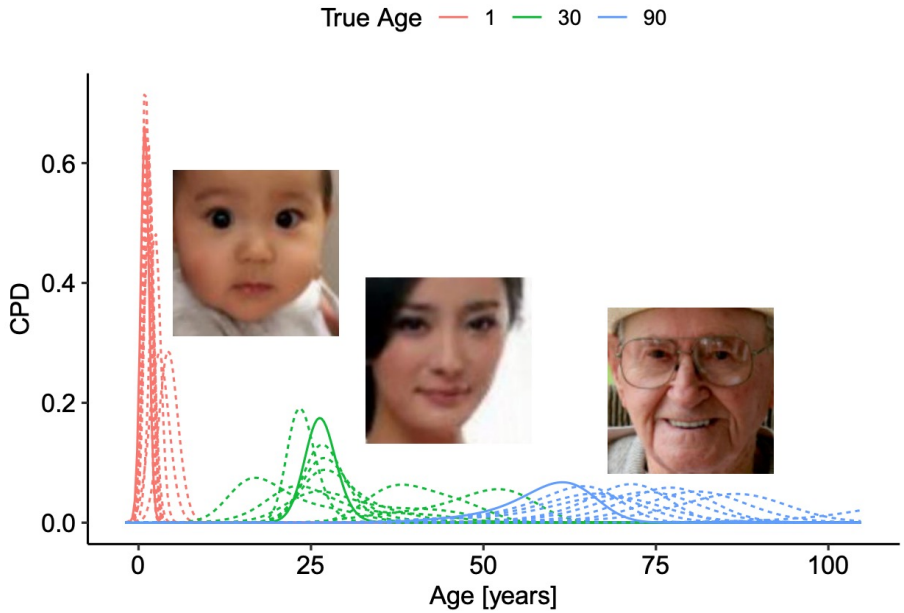
Less observations in low/high quantils, than forcasted.

Some note on PIT (Probability Integral Transform)

- Probability Integral Transform (PIT)
- Algorithm
  - Step 1:
    - Compute the CDF $F_{Y|x_i}(y|x_i)$ of your predicted outcome distribution $p_{\text{pred}}(y|x_i)$ on the test set.
      The CDF is integrating $p_{\text{pred}}(y|x_i)$ hence the name “Integral” in PIT.
  - Step 2:
    - Calculate $F_{Y|x_i}(y_i|x_i)$ at the observed values (”transformation”)
    - Also called PIT-Values, should be uniformly distributed
  - Step 3:
    - Do a PIT histogram of the computed $F_{Y|x_i}(y_i|x_i)$ → if the PIT shows a uniform distribution, the prediction model is well calibrated.
  - Step 3 (alternative version)
    - Do a QQ-Plot of PIT-values against uniform distribution, better for small sample size.

Calibration vs Sharpness $p(y|x)$





A very uninformative but calibrated forecaster would just predict the population distribution fat blue line (for every) x.

PIT would be uniform!

Sharper predictor depends on x. Still calibrated?

Benchmark your model

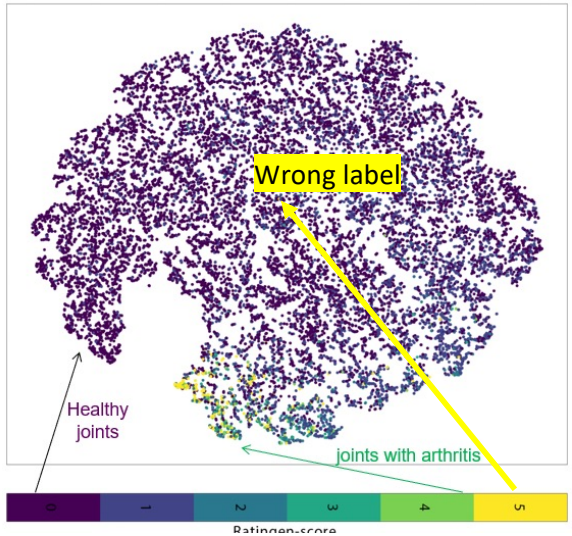
- Apply your model to benchmark data sets for which the performance of other SoA models are known from literature

- Apply beside your model also baseline models to your data and compare the performances
  - Null model that predicts always the train marginal outcome distribution
  - Random Forest (or other simple shallow model)
    - On raw input features (e.g. image pixel values)
    - On constructed features (e.g. extracted from pretrained model)

Perform an error analysis

- Visualize/investigate wrongly classified instance
- Check grouping of wrongly classified instance
- Check for labeling errors
- Perform xAI, check interpretability
- Check intermediate results in debugging mode

Not cute?
Why?
Cage?



outcome

Exercise Probabilitic Age Prediction

Notebook: 03

https://github.com/tensorchiefs/dlwbl_eth25/blob/master/notebooks/03_uk_faces.ipynb

Continue with NB and do the PIT exercise:

🔧 Exercise PIT-Histograms

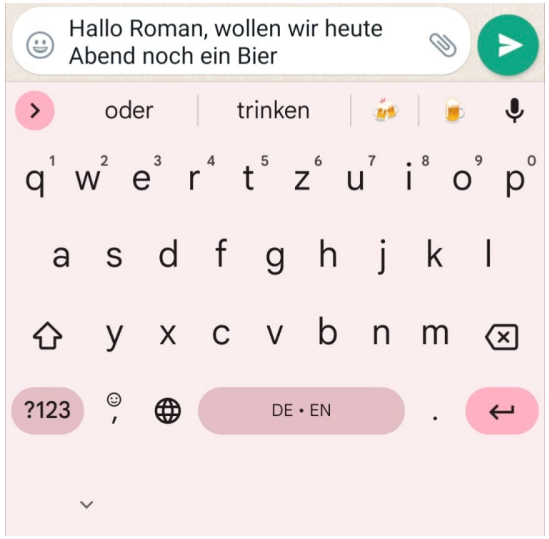
Create a so-called PIT histgram (Probability Integral Transform) to check if the model is calibrated. For that take calculate the values of the cumulative distribution function (CDF) of the predicted distribution evaluated at the observed ages. Then plot a histogram of these values. The histogram should be uniform if the model is calibrated. Describe your findings.

Hint: You can use the function `scipy.stats.norm.cdf` to calculate the CDF of the normal distribution.

]:
```
from scipy import stats
import torch
d = model_flex(X_test)
mu = d[:, 0].reshape(-1, 1).detach().cpu().numpy()
sigma = torch.exp(d[:, 1]).reshape(-1, 1).detach().cpu().numpy()
pit_values = stats.norm.cdf(y_test, loc=mu, scale=sigma)
```

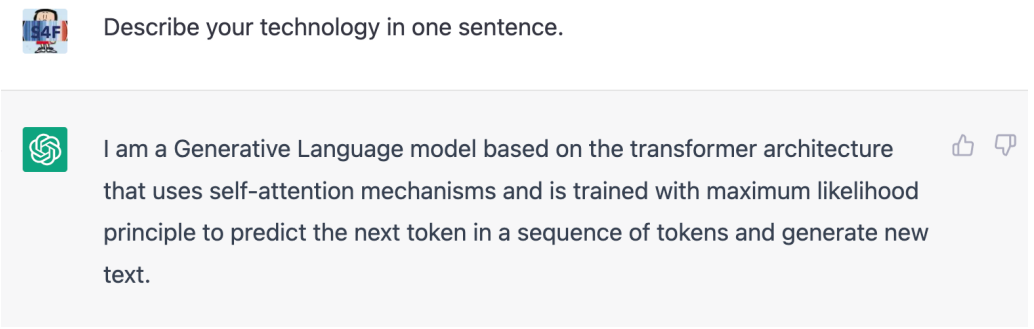
Small Excurse on Large Language Models

Generative Language Models: "predict next token"



Generative Language models:

Like you pressing (most) probable suggestion



Steps 1: Describe your technology in one sentence. → I
Steps 2: Describe your technology in one sentence. I → am
Steps 3: Describe your technology in one sentence. I am → a
Steps 4: Describe your technology in one sentence. I am a → generative

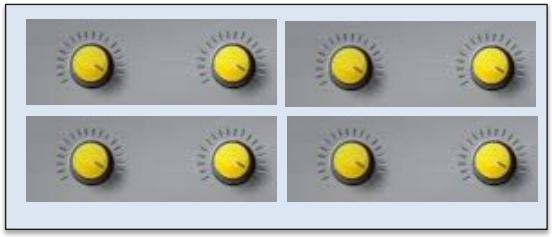
Step 36 Describe your technology in one sentence. I am a … new text. → END

LLM are probabilistic models

"The weather is really nice today. I'm thinking about going for a"

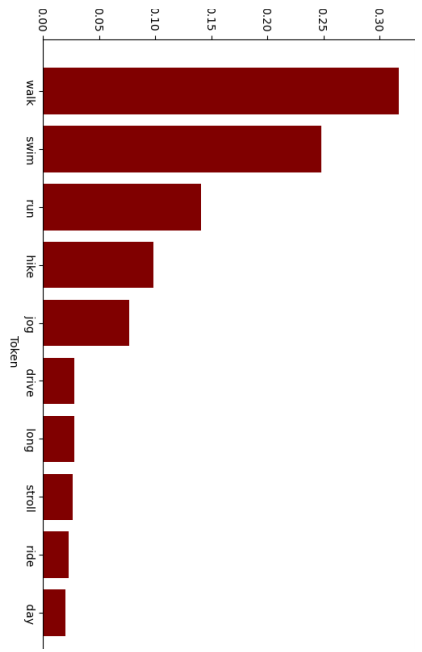
$\boldsymbol{x}$

LLM



A model with parameters $\boldsymbol{\theta}$

Probability for y



$\boldsymbol{y}$

Quiz: Number of parameters in GTP-3.5?

$$p_{\boldsymbol{\theta}}(y|\boldsymbol{x})$$

GTP (Generative Pretrained Transformer) and Llama are causal/autoregressive

Ingredient to LLM

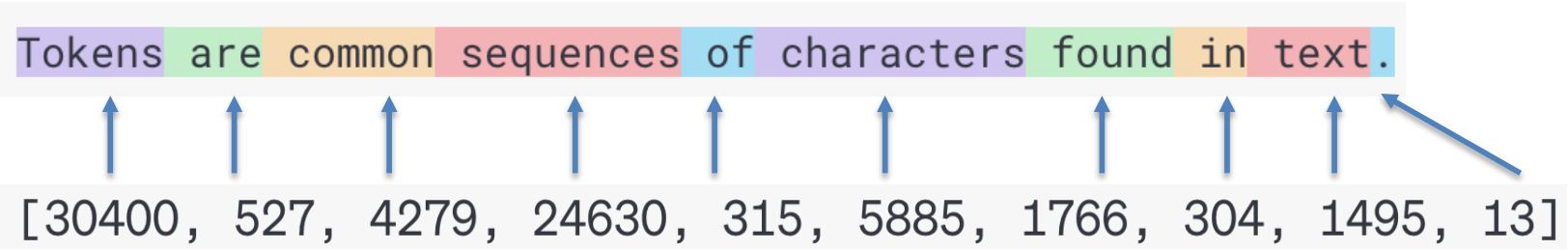
- Coding Strings to number
  - Tokenization
  - Embedding

- Transformer Architecture
  - Here we give a short introduction

Tokenization: Byte Pair Encoding

ChatGPT works with tokens not with words. It has 50257 different tokens.

https://platform.openai.com/tokenizer



Mein Luftkissenfahrzeug ist voller Aale

Mein Luftkissenfahrzeug ist voller Aale

Emergence or stochastical parrot 🦜?

Emergence or stochastical parrot 🦜?

差不多

差不多

[17597, 106, 16937, 43240]

English is coded shorter (since most of text is in English for BPE Algorithm)
Fun Fact: “spell lollipop backwards” used to trick ChatGPT(before 25 Sep Version 2024)

Tokenization: One-Hot-Encoded

- Tokens are categorical data cat ≰ sat
- One Hot Encoding needed
- N size of the relevant vocabulary, levels of categories

Example: Look at vocabulary with N=9 to tokenize the 2 text-samples below

Vocabulary: `{'the': 1, 'cat': 2, 'sat': 3, 'on': 4, 'mat': 5, 'dog': 6, 'ate': 7, 'my': 8, 'homework': 9}`

Text-sample: "The cat sat on the hat" "The dog ate my homework"

Tokenized: `[1, 2, 3, 4, 1, 5]` `[1, 6, 7, 8, 9]`

1-hot:

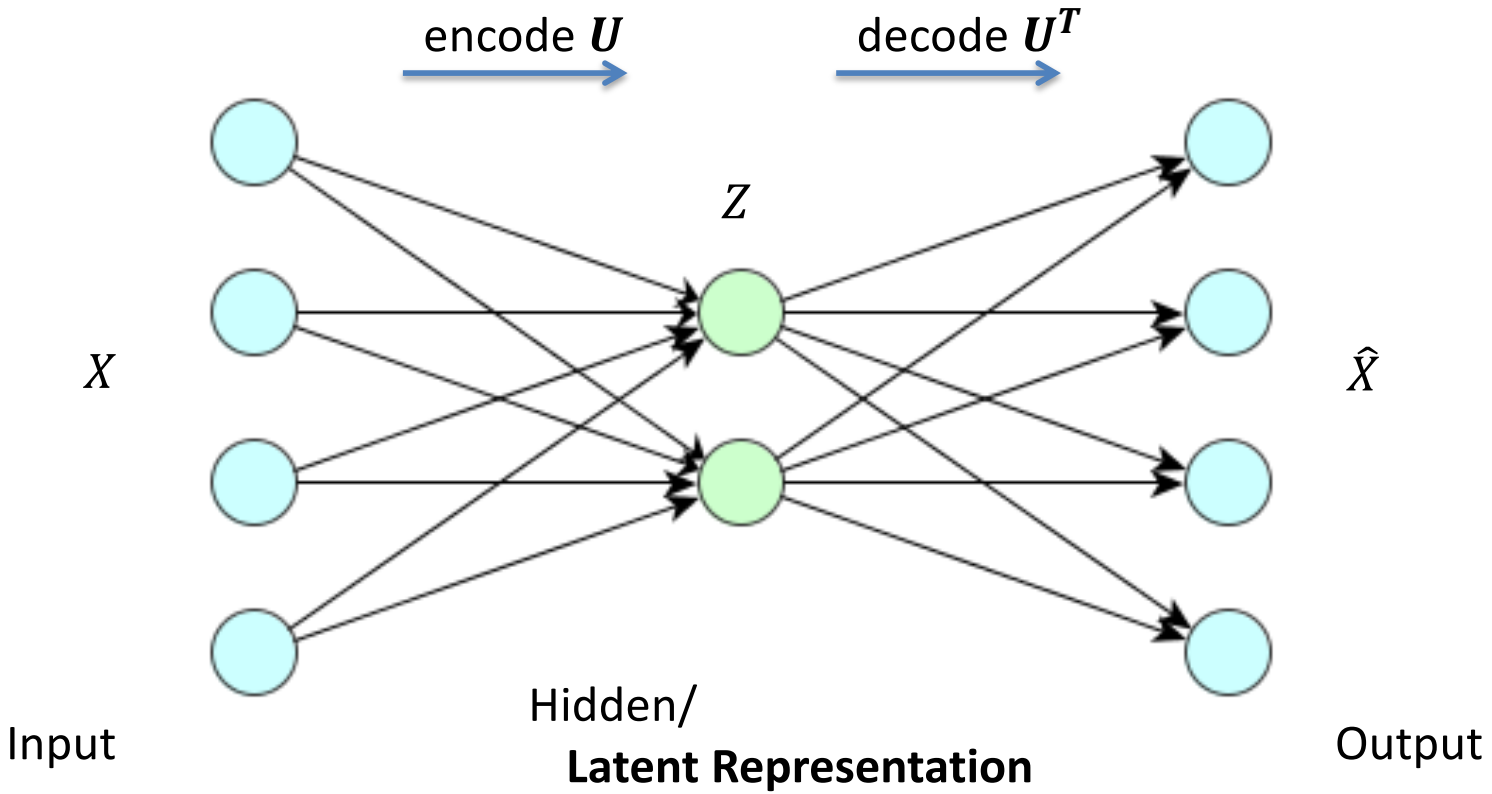
1	0	0	0	0	0	0	0	0	the
0	1	0	0	0	0	0	0	0	cat
0	0	1	0	0	0	0	0	0	sat
0	0	0	1	0	0	0	0	0	on
1	0	0	0	0	0	0	0	0	the
0	0	0	0	1	0	0	0	0	hat

N elements (=vocabulary size)

1	0	0	0	0	0	0	0	0	the
0	0	0	0	0	1	0	0	0	dog
0	0	0	0	0	0	1	0	0	ate
0	0	0	0	0	0	0	1	0	my
0	0	0	0	0	0	0	0	1	homework

N elements (=vocabulary size)

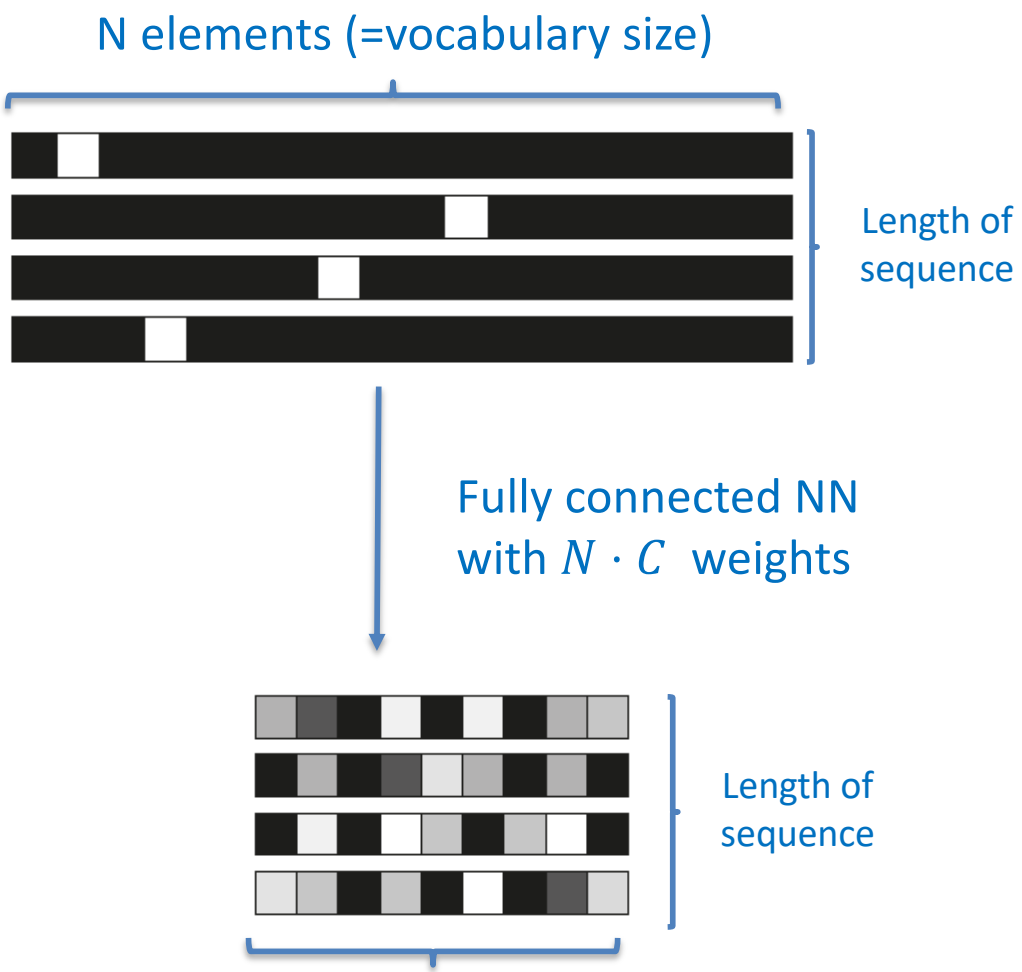
Recap: PCA Going from High in Low Dimensional Space



- In PCA $\boldsymbol{U}$ determined in training so that reconstruction error $\left\|X - \hat{X}\right\|^2$ is minimized

Embeddings

1-hot encodings or tokens
- Based on vocabulary of size N
- sparse: 1 one and N-1 zeros
- High-dim: vector-length = N

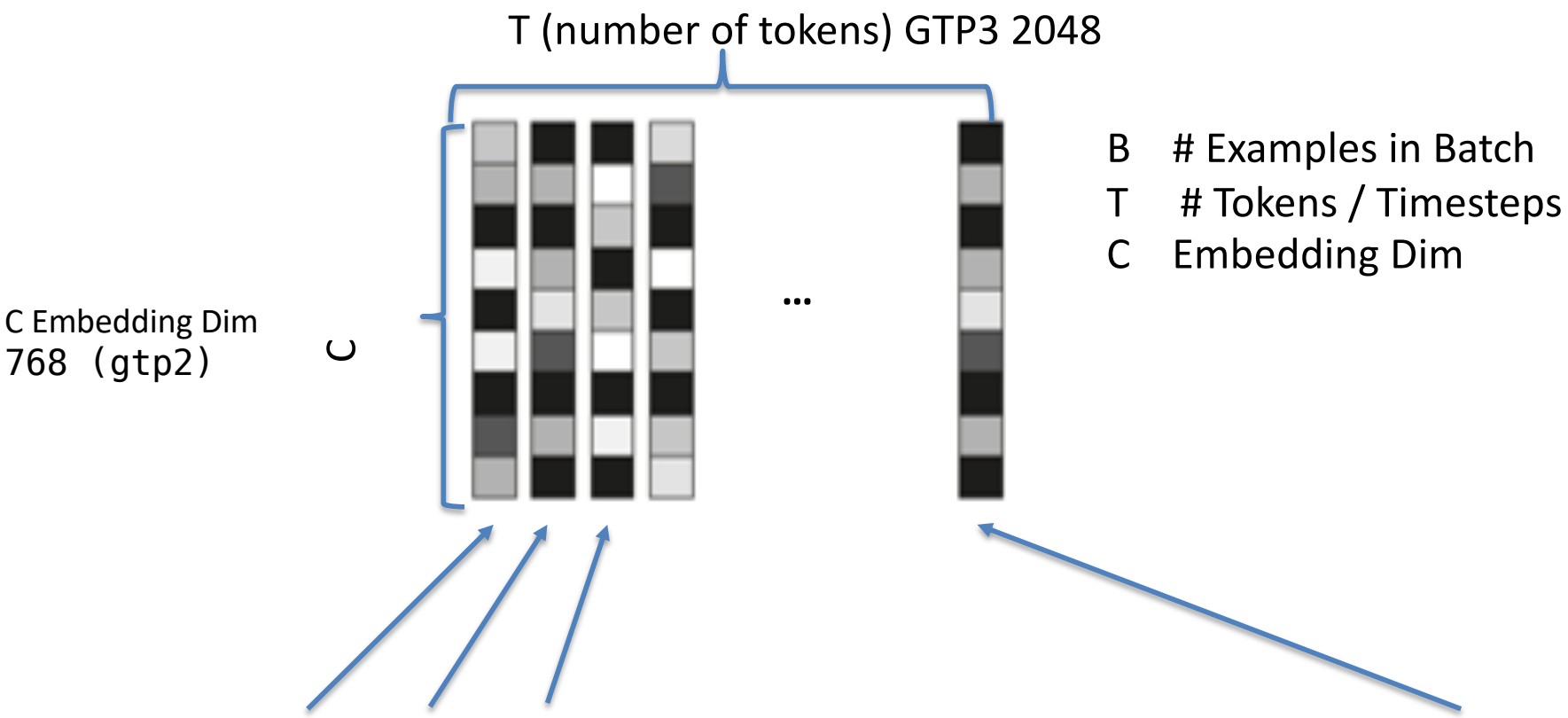


Word embedding's are
- Dense
- Low-dimensional: vector-length = p
- Learned from data via fcNN N→p

There are pretrained word embeddings like word2vec.

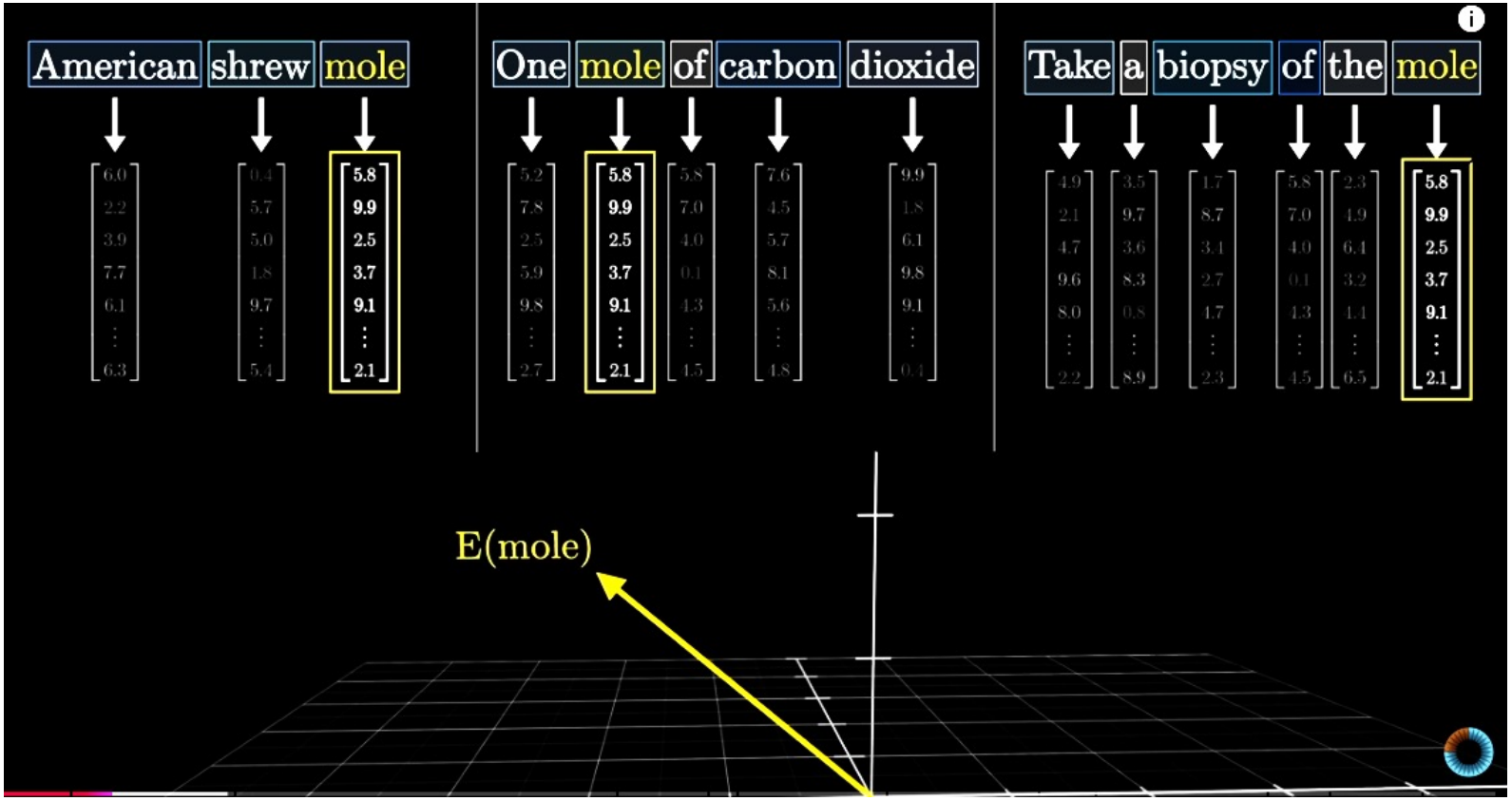
Recap Understand Text as a Sequence of Vectors

- After Tokenization and Embedding Text looks like



In DL we usually take batches the primary object of interest
a tensors of size 3 with (B, T, C) dimensions

Note, embeddings have no context

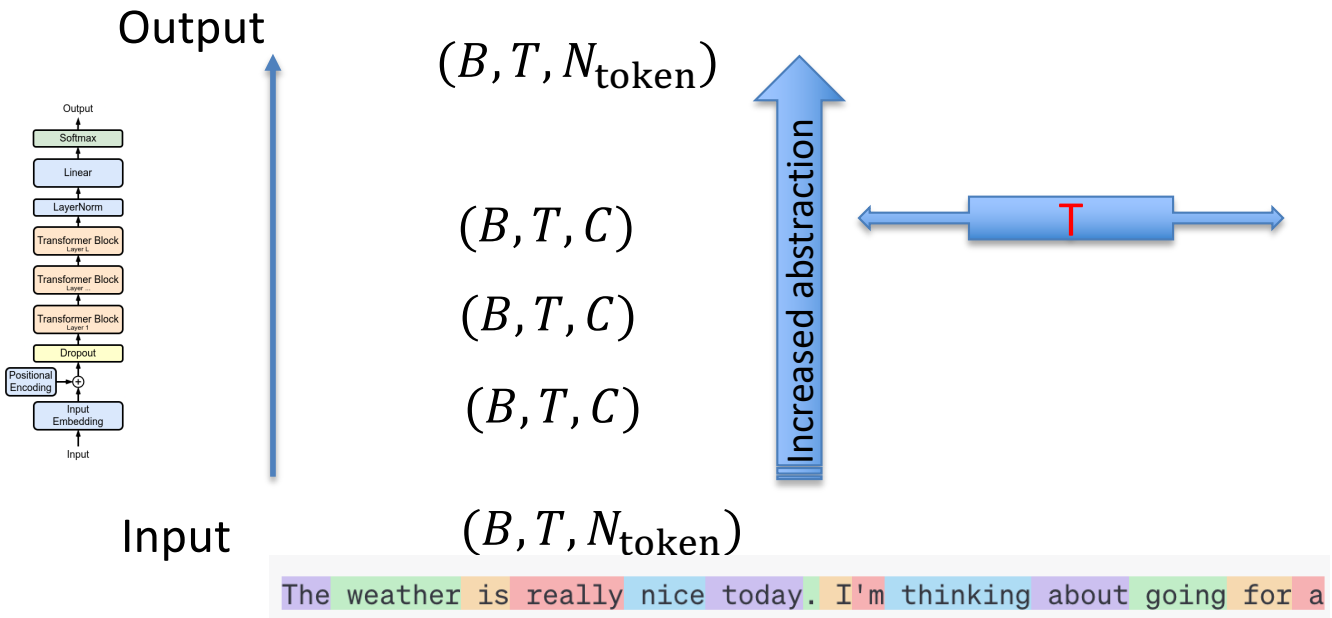


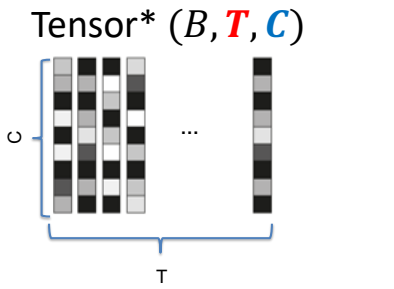
The context should change the embedding.
Mole:
Maulwurf (Tier), Leberfleck (Med), Maulwurf (Spionage), Mol (Chemie), Mole (Landbrücke)

https://www.youtube.com/watch?v=eMlx5fFNoYc&t=559s&ab_channel=3Blue1Brown

In Language: need for context

- Example
  - Server, can I have the check?
  - Looks like I just crashed the server
- We need the context, to understand the meaning of server.
  - Information needs to be passed in T dimension 

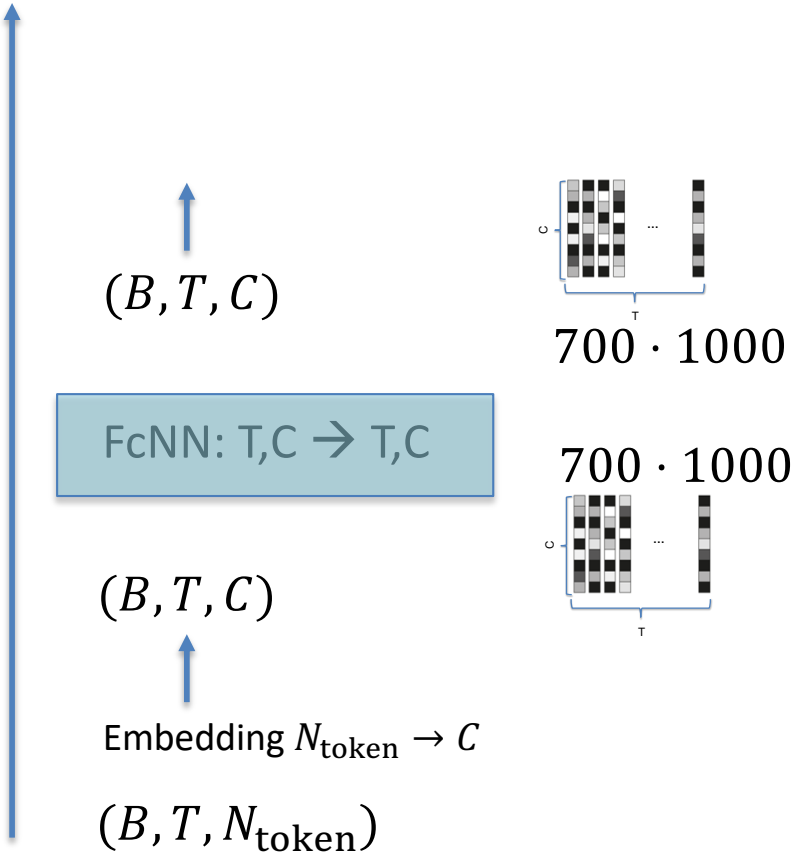




B # Examples in Batch
T # Tokens
C Embedding Dim

*Strictly speaking this is transposed

A naïve approach for context



$700 \cdot 1000$

$700 \cdot 1000$

One Block would have
$(700 \cdot 1000)^2 \approx 490\text{B}$ parameter

Too much for a single layer

Other architectures have been developed in the past.
RNNs and LSTMs

Context: Transformer

Attention Is All You Need

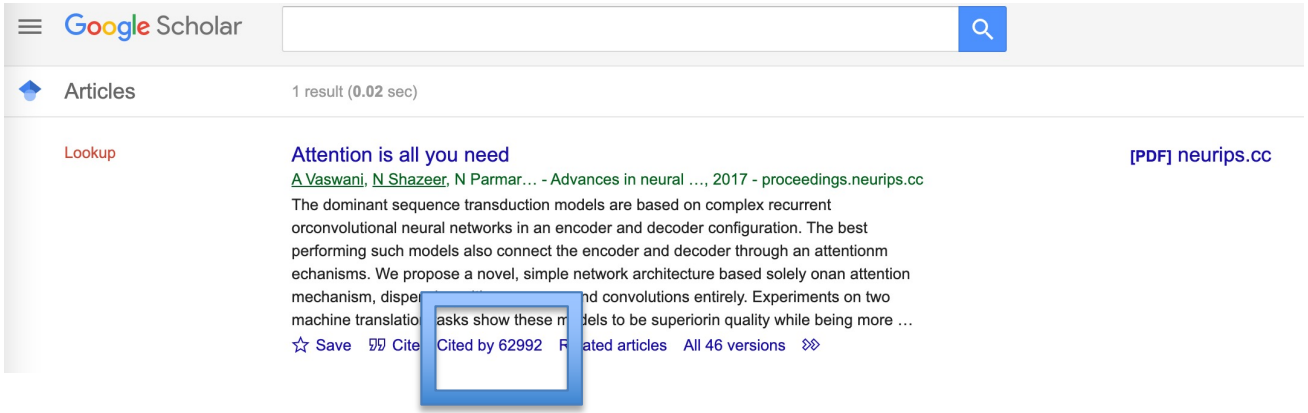
Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including

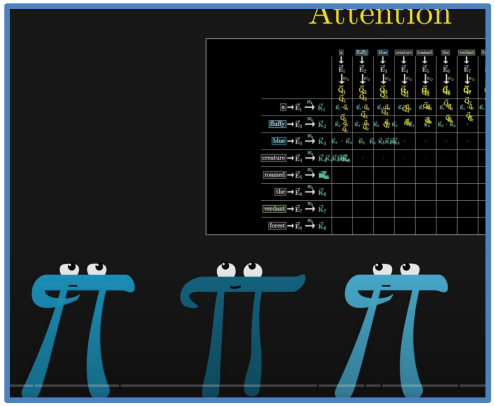
https://arxiv.org/abs/1706.03762

- Introduced for translation
- Basis for Language models (GPT-2, GPT-3, Chat-GPT, Bert)



Sources for Transformer

- 3Blue1Brown:
  - https://www.youtube.com/watch?v=eMlx5fFNoYc&t=559s&ab_channel=3Blue1Brown



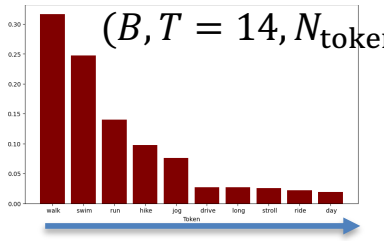
- Live Codeing (and explanation) Andrej Karparthy nano-GPT https://youtu.be/kCc8FmEb1nY

Further Resources

- CS25 Andrej Karparty Introduction to Transformers
  - Nice intuition with message parsing https://www.youtube.com/watch?v=XfpMkf4rD6E
- Lennart Svensson: https://youtu.be/0SmNEp4zTpc
  - Good intuition (weighted average, mathematical sound)
- Ava Soleimany: https://www.youtube.com/watch?v=QvkQ1B3FBqA
  - Good intuition (Search query idea)
- https://jalammar.github.io/illustrated-transformer/
  - Nice illustrations
- CS25 Andrej Karparty Introduction to Transformers
  - Nice intuition with message parsing https://www.youtube.com/watch?v=XfpMkf4rD6E

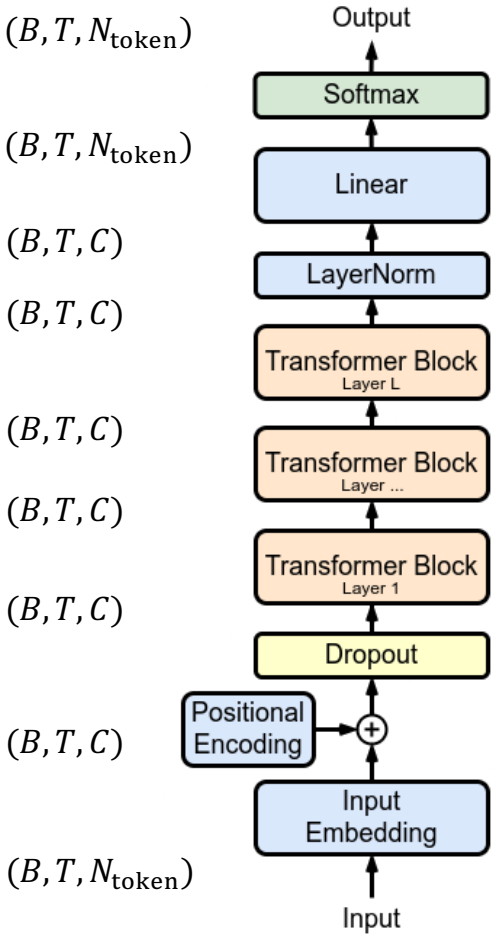
Follow the forward pass

B # Examples in Batch
T # Tokens
C Embedding Dim



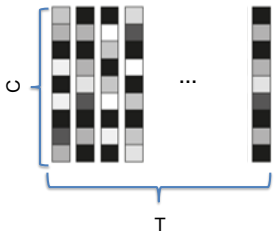
$(B, T = 14, N_{\text{token}})$

N_{token}



(B, T, N_{token})
(B, T, N_{token})
(B, T, C)
(B, T, C)
(B, T, C)
(B, T, C)
(B, T, C)
(B, T, N_{token})

(B, T, C) Tensor



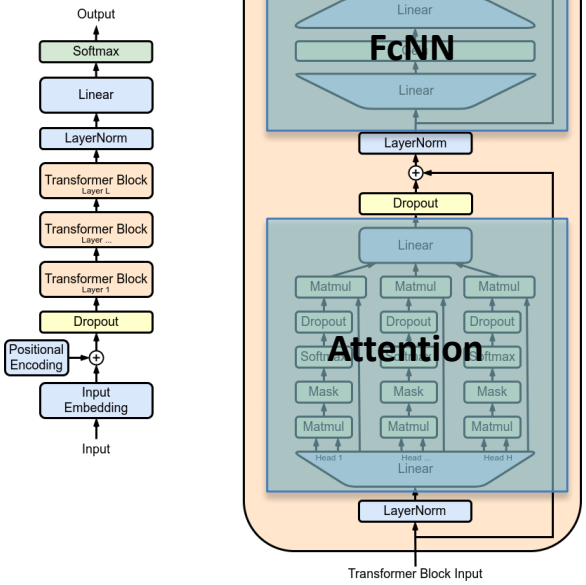
The dimension of the tensor (B, T, C) stays, it is just *transformed.*
Mostly along C dimension.

[791, 9282, 374, 2216, 6555, 3432, 13, 358, 2846, 7422, 922, 2133, 369, 264, 198]

The weather is really nice today. I'm thinking about going for a

Transformer Block

B # Examples in Batch
T # Tokens
C Embedding Dim



(B, T, C)

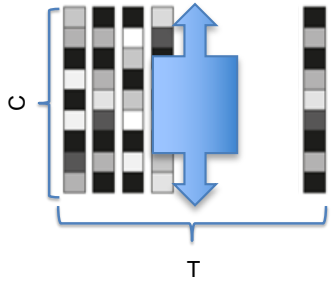
FcNN Block
FcNN with C in and out.
Updates representation

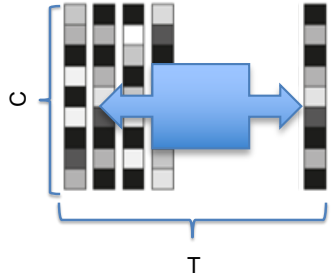
(B, T, C)

Attention Block:
Updates with information form (left) neighbors

(B, T, C)

Information Processing





Information is processed in two steps first

- Along T
- Along C

Total Memory $O(T) + O(C)$

Most operations along C-Direction

Image credit Wikipedia

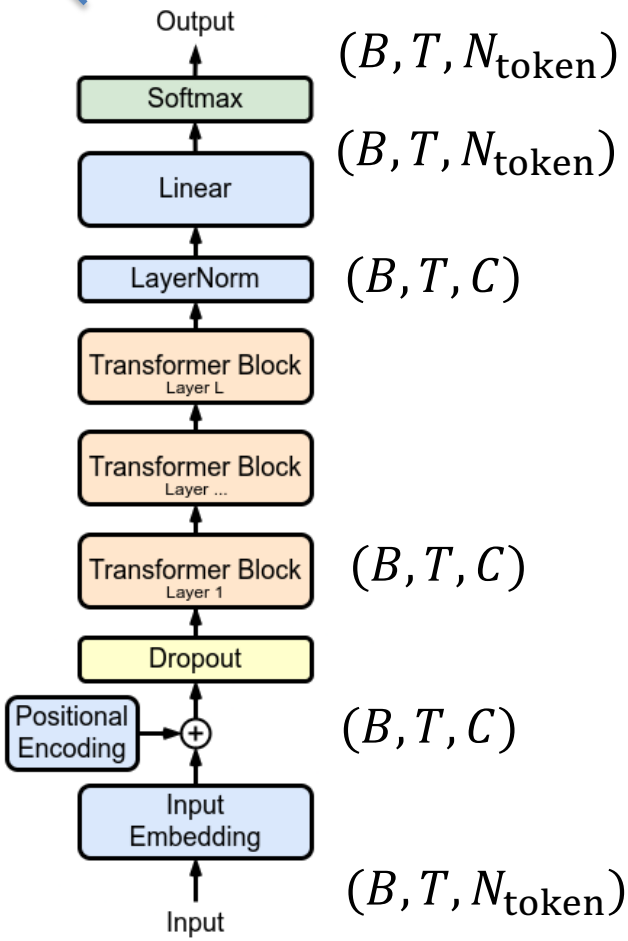
Training parallelism / Causal LLM

B # Examples in Batch
T # Tokens
C Embedding Dim

Shift input one token to the left

The weather is really nice today. I'm thinking about going for a walk

[128000, 791, 9282, 374, 2216, 6555, 3432, 13, 358, 2846, 7422, 922, 2133, 369, 264]



(B, T, N_{token})

(B, T, N_{token})

(B, T, C)

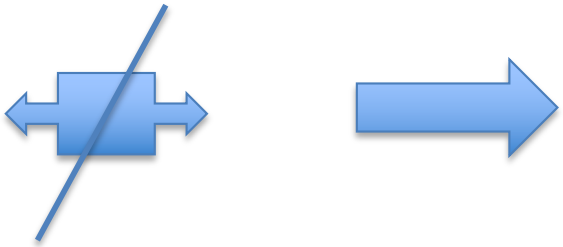
(B, T, C)

(B, T, C)

(B, T, N_{token})

Attention ("Causality")

When predicting next word only information from pervious token must be used.



[128000, 791, 9282, 374, 2216, 6555, 3432, 13, 358, 2846, 7422, 922, 2133, 369, 264]

The weather is really nice today. I'm thinking about going for a

Using Transformers

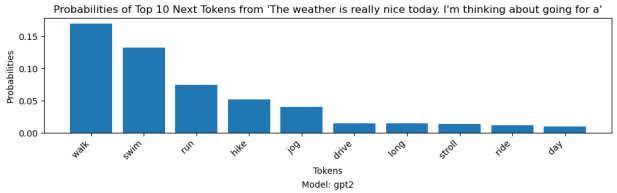
B # Examples in Batch
T # Tokens
C Embedding Dim

```
### Getting the model
model_name = "gpt2-medium" #"deepseek-ai/DeepSeek-R1-Distill-Llama-8B", …
model = AutoModelForCausalLM.from_pretrained(model_name) #← Downloading model
model.num_parameters() / (1E9) #0.35 Billion Weight ChatGPT 3.5 has 175B

### Tokenizing
tokenizer = AutoTokenizer.from_pretrained(model_name)
input_context = "The weather is really nice today. I'm thinking about going for a"
input_ids = torch.tensor(tokenizer.encode(input_context)).unsqueeze(0) #128000, 791,…,264
input_ids.shape #1, 14 this is short for (1,14,50257)

### Predicting the next Tokens
outputs = model(input_ids) # We ask the model for the next token
probs = torch.nn.functional.softmax(outputs.logits, dim=-1)
```

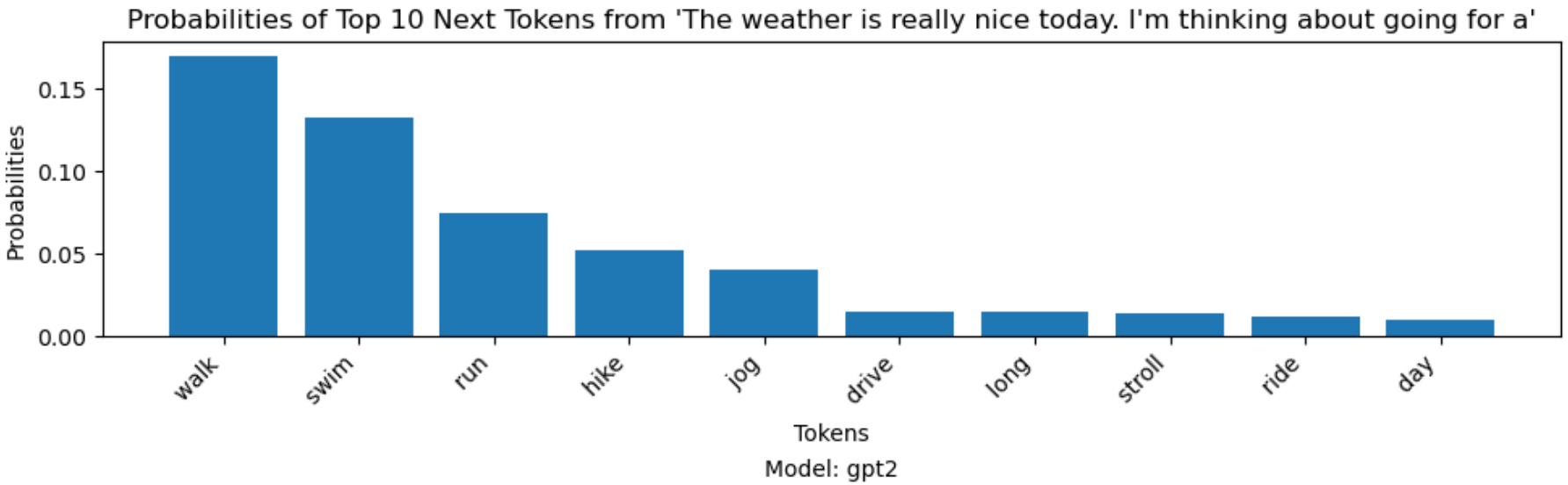
Input:
The weather is really nice today. I'm thinking about going for a

probs[,2,]
is the probability distribution of the word after “The weather is”

More interesting is
probs[,-1,]the probability for the next word / token after
The weather is really nice today. I'm thinking about going for a



Probability for the next word



Now sample from that distribution, and add this token to generate new text. Say, we sampled swim

new_input = "`The weather is really nice today. I'm thinking about going for a`"

Sampling next token from distribution

Model: gpt2-medium

Prompt: The weather is really nice today. I'm thinking about going for a

jog around the neighborhood." Beth dropped her rain boots, pulled out a notebook from her bag and scanned the documents that filled up the notable shelf above her desk. A few days ago, Mr. Richard Fair asked to borrow a laptop to help him look

swim today," he said, gesturing to a hastily painted house in a garden at the expense of some gardening students who had come to try and solve matching English numbers.

crash tomorrow." Beetle gold: Ranzbash Talib was part of Russia's 2008 Olympic hockey gold medal team We are really looking forward to lift and have great club setting over Christmas than for beer and burro rounds today

Only take the most probable token

walk today. I'm thinking about going for a walk today. I'm thinking about going for a walk today. I'm thinking about going for a walk today. I'm thinking about going for a walk today.

Findings:

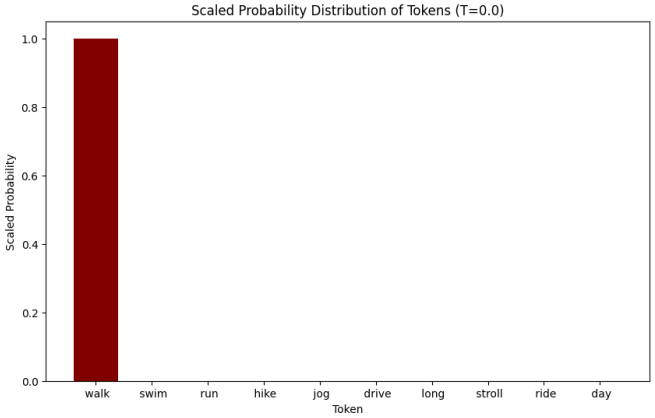
Sampling from distribution, yields incoherent text. Sampling using maximum can yields repetive text

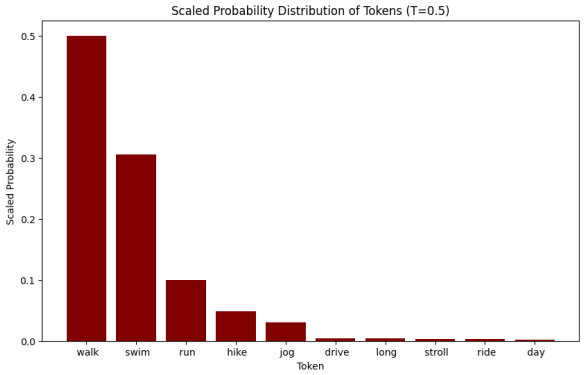
Tricks:

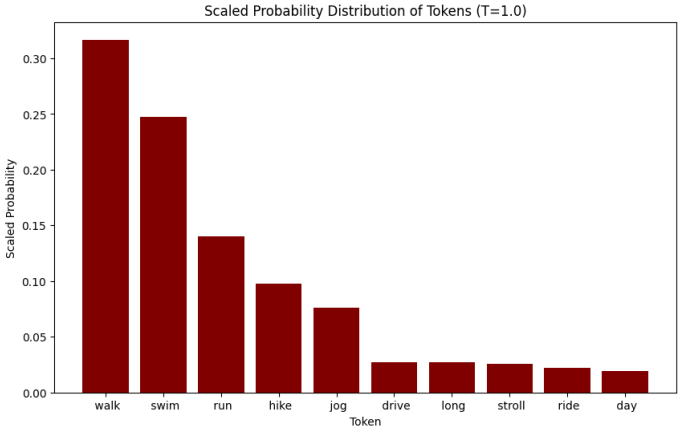
Apply a temperature, top-k, top-p and p_min

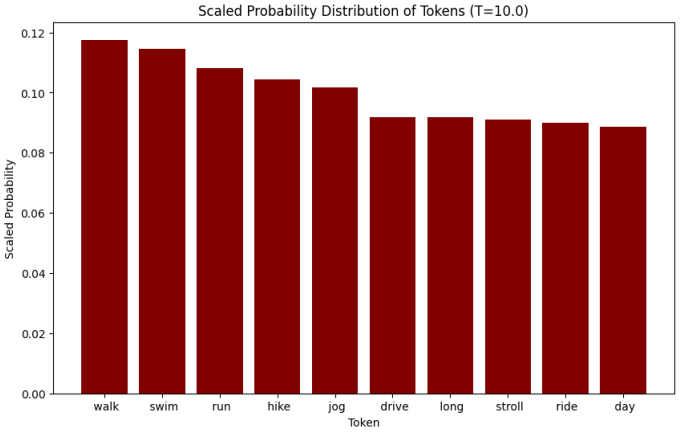
Temperature scaling: scaled probability

$$p(y|\text{"prompt"}) = \frac{\exp(\text{logits}/\text{temperature})}{\sum_{y'} \exp(\text{logits}_{y'}/\text{temperature})}$$









https://github.com/oduerr/llm_playground/tree/main/hugging_face/sampling_from_LLM

Additional Sampling Tricks

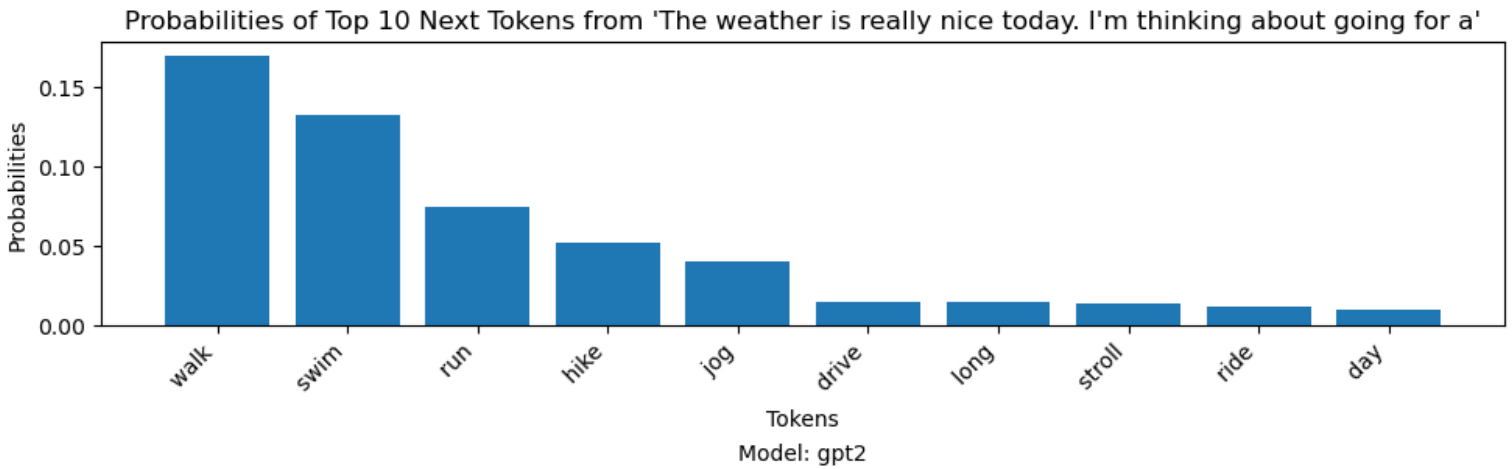
- Top-p (Nucleus Sampling):
  - Dynamically selects tokens until **cumulative probability exceeds p** (e.g., 90%).
  - **Adaptive**: Number of tokens changes based on model confidence.
  - Focuses sampling on the **most likely tokens**, maintaining flexibility
- Min-p Filtering:
  - Removes tokens with probabilities below a minimum threshold (e.g., 5%).
  - Acts as a final confidence filter to exclude low-probability outliers.
  - Ensures all selected tokens have a baseline likelihood.
- Guidelines
  - Lower Temperature less creative
  - Instructions based models, have low temperature
    - For `SmolLM2-1.7B-Instruct`
      - max_new_tokens=50, temperature=0.2, top_p=0.9

Pause bis 14:20

Playaround with LMMs

Have a look at the following NB:

https://github.com/tensorchiefs/dlwbl_eth25/blob/master/notebooks/03_sampling_lmm.ipynb



Training

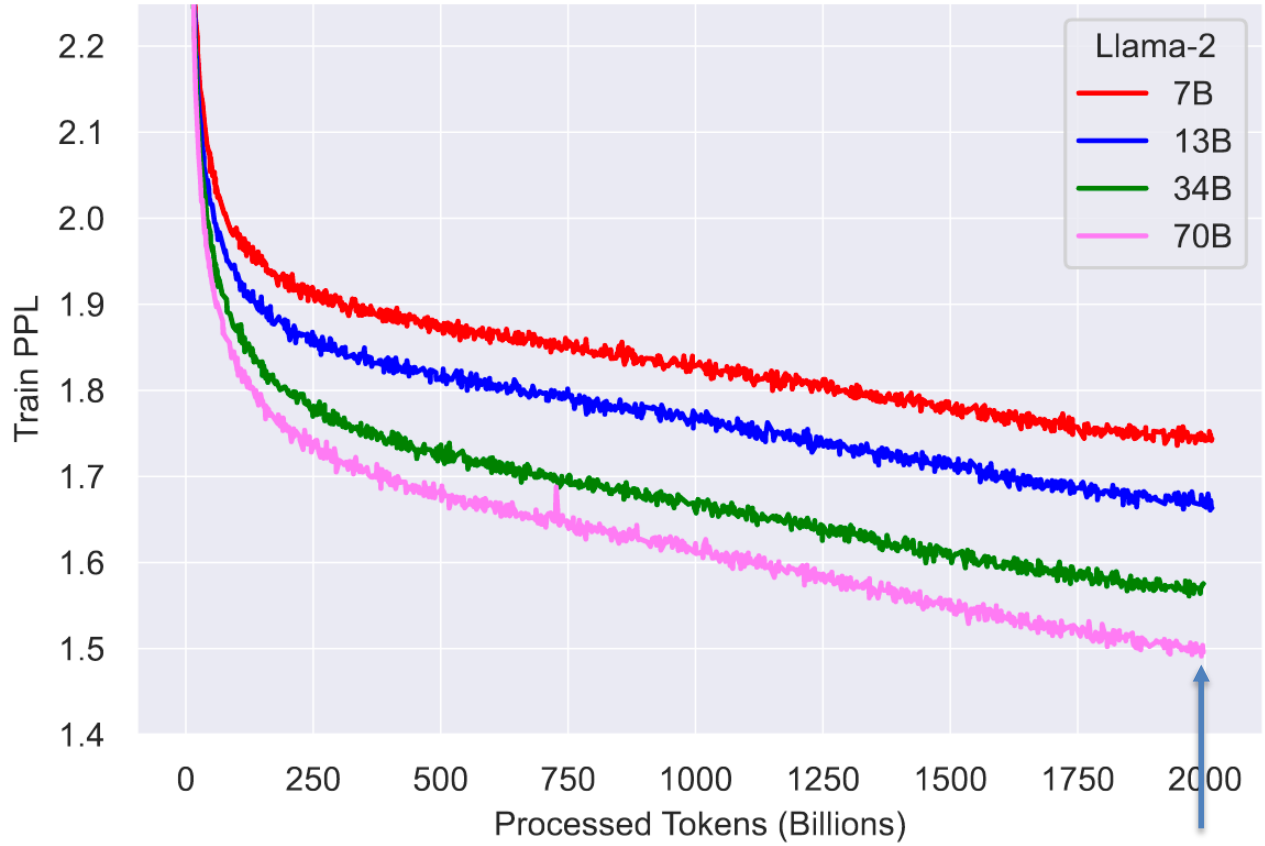
- Training (GPT-3)
  - Trained on “complete Internet” (common crawl 400 10^9 tokens) + Wikipedia …
  - Trained to predict next token (maximum likelihood principle)
  - Trained for few months (CO2 equivalent 120 cars for one year)
    - Prediction 100 pages 0.4 kwh
  - Data before **Q4 2021**

- In Language Modelling, instead of NLL the average of NLL over a sequence of length N is calculated, as follows.
  - PPL = exp(NLL/N)

Training LLama2

Number of parameters 7B-70B
Number of Tokens $N_{\text{token}} = 32k$

Context Length $T = 2k - 4k$
Number of of Layers 32-80
Context Size C=4096-8192

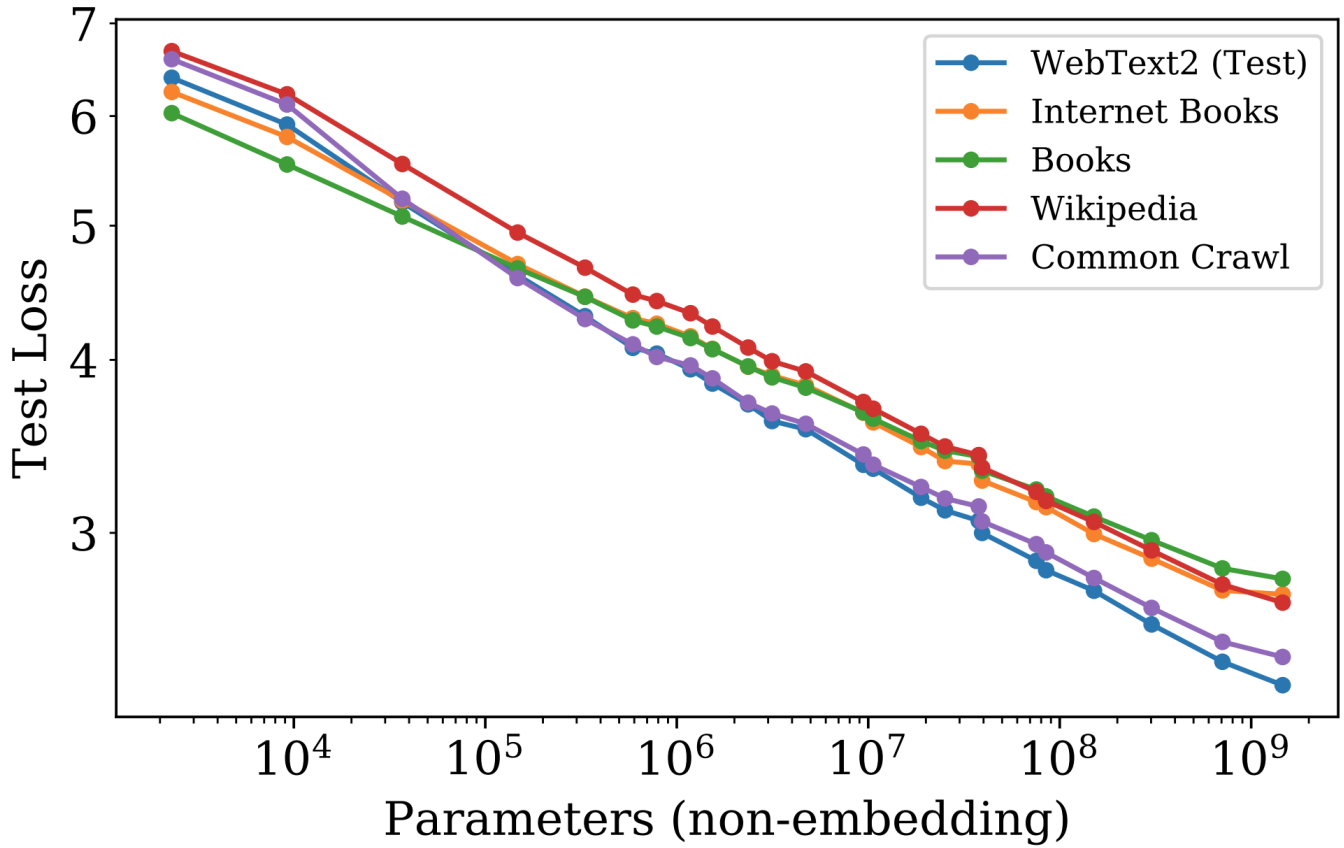


For 70B
1.7 Million GPU Hours
291.42 tCO2eq
(70 Household for a year)

http://arxiv.org/abs/2307.09288

Scaling Laws of LMM

- Performance L in NLL (hold-out-set) scales like
  - $L(N, D) = A + \frac{B}{N^{0.34}} + \frac{C}{D^{0.28}}$
    - N Number of parameters of model
    - D Data set size (A, B, C) constants



Finetuning / Alinment

Plain vanilla LLM

prompt **What is the capital of France?**

What is France's largest city?

What is France's population?

What is the currency of France?

Finetuned versions (Chat-GPT and Chat LLama)

prompt **What is the capital of France?**

Paris

Further aspects of fine tuning (Helpful, Honest, Harmless)

Emergent properties of LLM

Yann LeCun
Jan 2022

"I take an object, I put it on the table, and I push the table […] GPT-5000 is never gona learn what happens, this information is not present in any text.

ChatGPT-3.5 **(Dec 2022)**

I put an object on the table and I push the table. What happens to the object? Give your best guess in max 3 sentences.

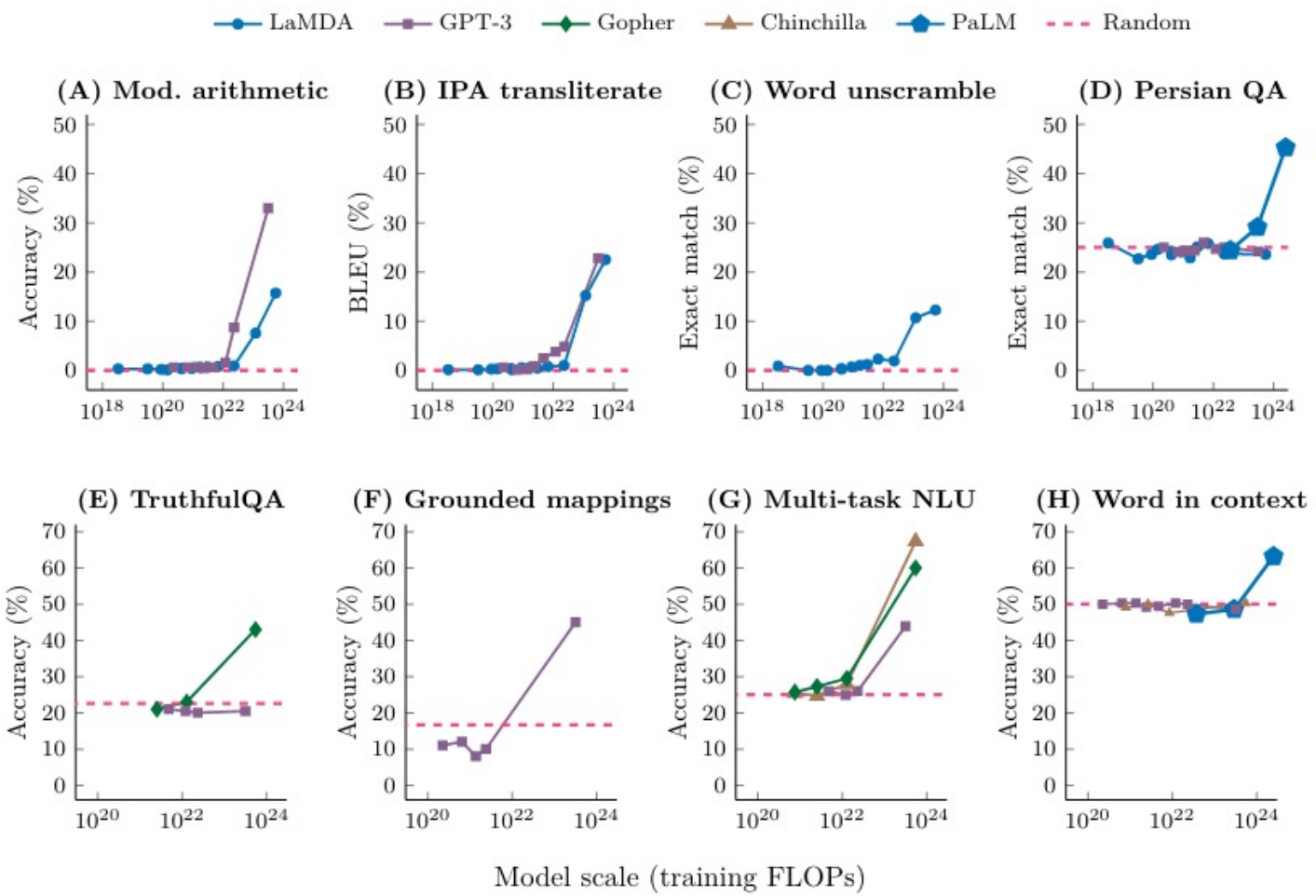
When you push the table, the object on it is likely to move along with the table, assuming the push isn't too violent or the object isn't overly fragile or unstable. The object's movement will be subject to the principles of inertia and friction; it will try to remain in its current state of motion until an external force acts upon it. However, if the push is too forceful, the object might slide or topple over.

https://twitter.com/liron/status/1659618568282185728

Why, scientific challenges

Stochastical parrots 🦜?

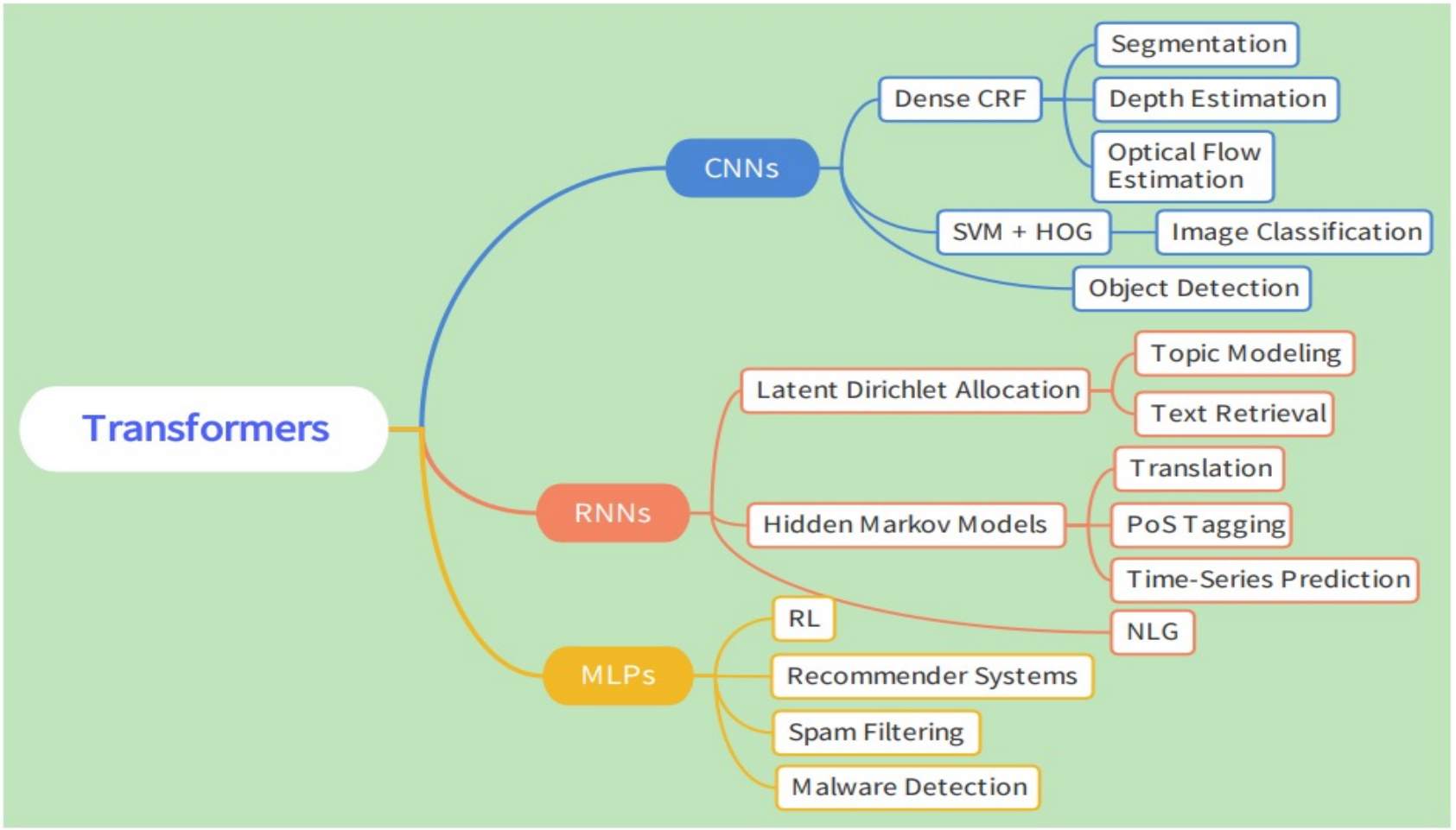
- LLM where trained to predict the next token in a sequence of tokens, but
- … suddenly, if the model is large enough, they can solve other tasks



https://arxiv.org/pdf/2206.07682.pdf

Pause bis 15:10

A general trend in DL Architectures



Credit: https://arxiv.org/pdf/2304.05366

Transformers for Vision

Overview

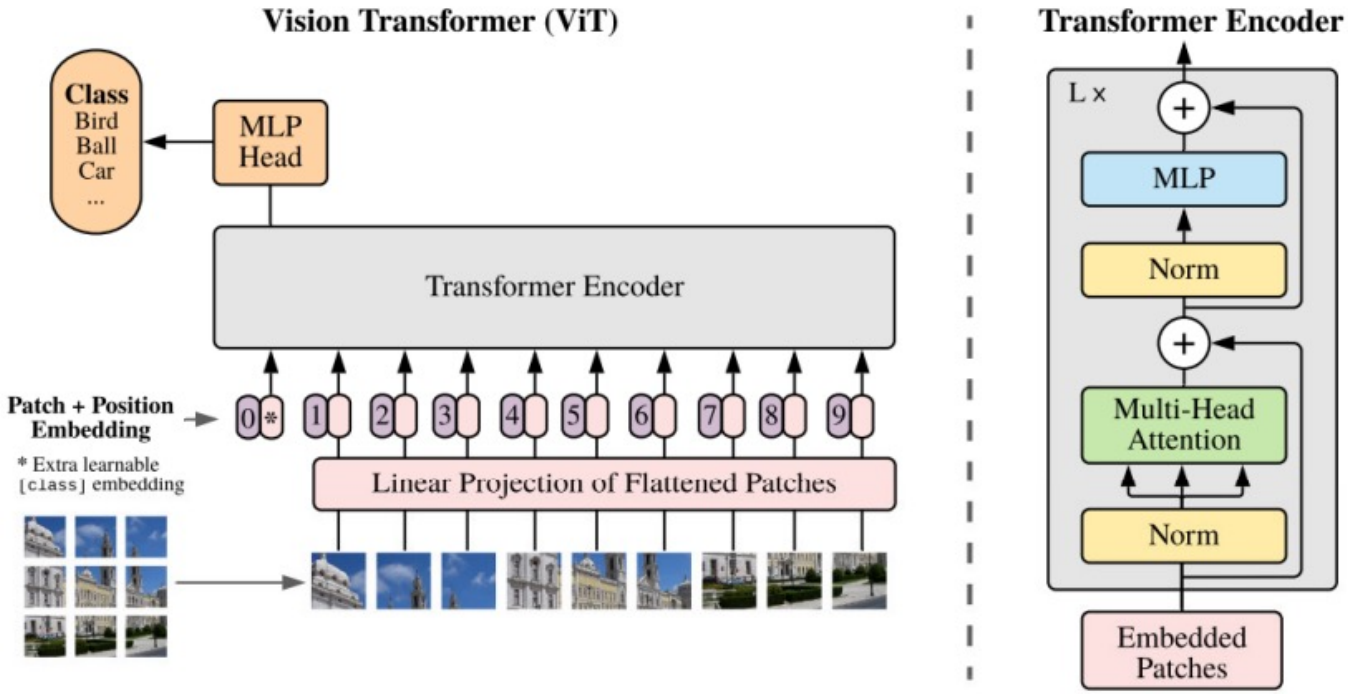


Figure 1: Model overview. We split an image into fixed-size patches, linearly embed each of them, add position embeddings, and feed the resulting sequence of vectors to a standard Transformer encoder. In order to perform classification, we use the standard approach of adding an extra learnable "classification token" to the sequence. The illustration of the Transformer encoder was inspired by Vaswani et al. (2017).

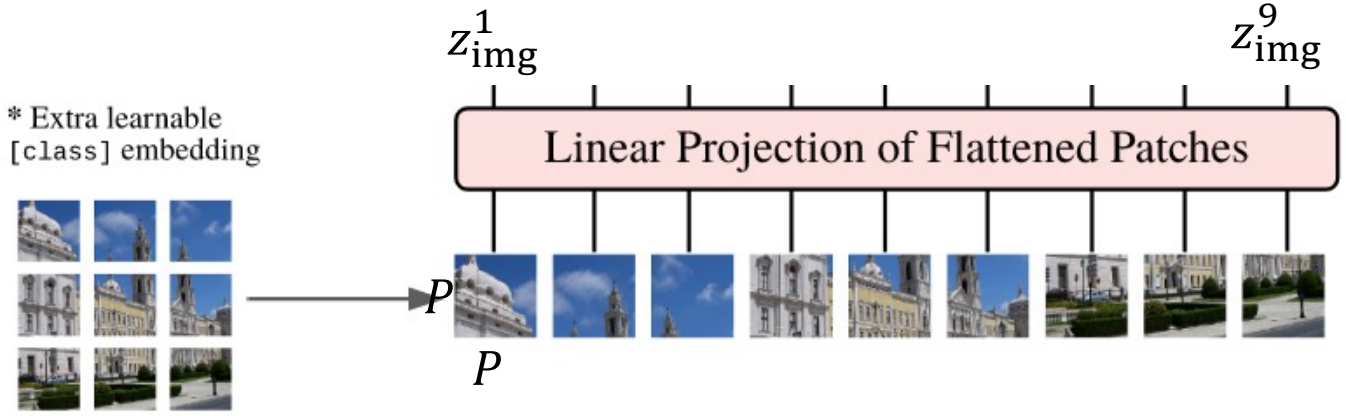
Question: How to bring Images (W,H,Col) to (T,C) Structure*?

https://arxiv.org/abs/2010.11929

* We ignore the Batch B first for not notational convenience.

Step 1: Transform image to patches

- We use Patches created from original images of dimension P.
- One Patch of Dimension (P, P, Col) corresponds to one token in time Dimension

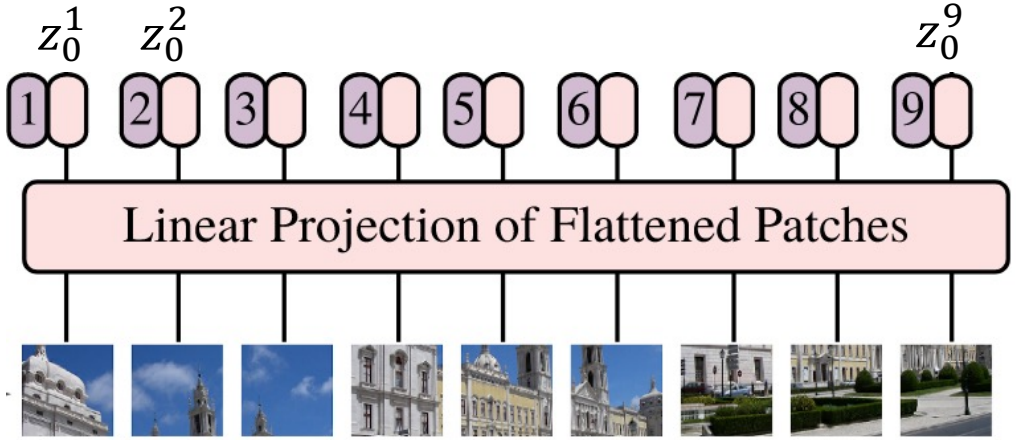


- We use Patches creates from original images dimension.
- One Patch of Dimension (P, P, Col) corresponds to one token in time Dimension.
- Embedding of the images
  - Flatten the patch in a vector x_p^t of size $P^2 \cdot \text{Col}$ and project it to a vector of Dimension C.
  - A image embedding matrix $\boldsymbol{E}$ of shape $(P^2 \cdot \text{Col}, C)$
  - $z_{\text{img}} = (x_p^1\boldsymbol{E}, x_p^2\boldsymbol{E}, \ldots x_p^T\boldsymbol{E})$

Step 2: Encoding the position

- To encode the position, we use numbers T (number of patches)
- This is categorical data, hence, we use one hot encoding (like tokens in LLM)
- We use a position embedding matrix E_{pos}
  - We learn the matrix E_{pos}
- $z_{pos} = E_{pos} \cdot (x_1, x_2, \ldots, x_T)$ with one-encoded vectors
- Linear Algebra
  - "Die Spalten einer Matrix sind die Bilder der Einheitsvektoren"
- $z_{pos} = E_{pos}$
- Adding positions embedding z_{pos} and image embedding z_{img}

$$z_0 = z_{\text{img}} + z_{pos}$$



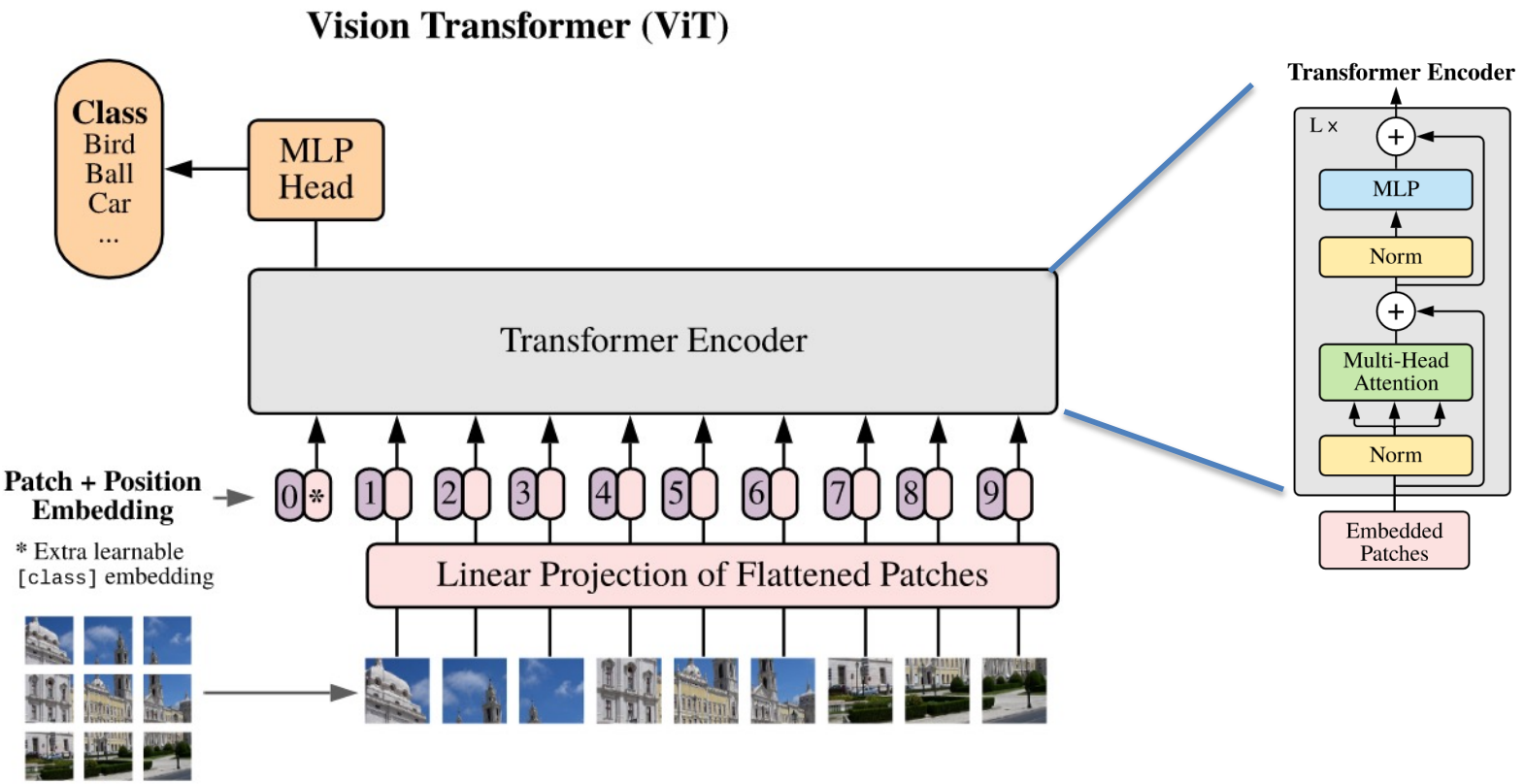
Training / Class Token

- z_0 is a (T, C) tensor and suitable for transformers
- Causal transformers in LLM predict next token in a sequence of tokens
- Here
  - No ordering, no causality
    - We use bidirectional attention influence from “future” is OK
  - Not senseful to predict next patch in a sequence of patches
    - We train using classification
    - Therefore, we use a classification token at position 0 (all other would be also OK) and do MLP with softmax at the and.

- Our start is then

$$z_0 = \left[x_{\text{class}}, x_p^1\mathbf{E}, x_p^2\mathbf{E}, \ldots, x_p^T\mathbf{E}\right] + \mathbf{E}_{pos},$$

The complete Architecture



$$z_0 = \left[x_{\text{class}}, x_p^1\mathbf{E}, x_p^2\mathbf{E}, \ldots, x_p^T\mathbf{E}\right] + \mathbf{E}_{pos},$$
$$z'_\ell = \text{MSA}(\text{LN}(z_{\ell-1})) + z_{\ell-1}, \quad \ell = 1 \ldots L,$$
$$z_\ell = \text{MLP}(\text{LN}(z'_\ell)) + z'_\ell, \quad \ell = 1 \ldots L,$$
$$y = \text{LN}(z_0^L).$$

$$\mathbf{E} \in \mathbb{R}^{(P^2 \cdot \text{Col}) \times C},$$
$$\mathbf{E}_{pos} \in \mathbb{R}^{(T+1) \times C}.$$

Formula 1 in VIT adopted to our notation https://arxiv.org/abs/2010.11929

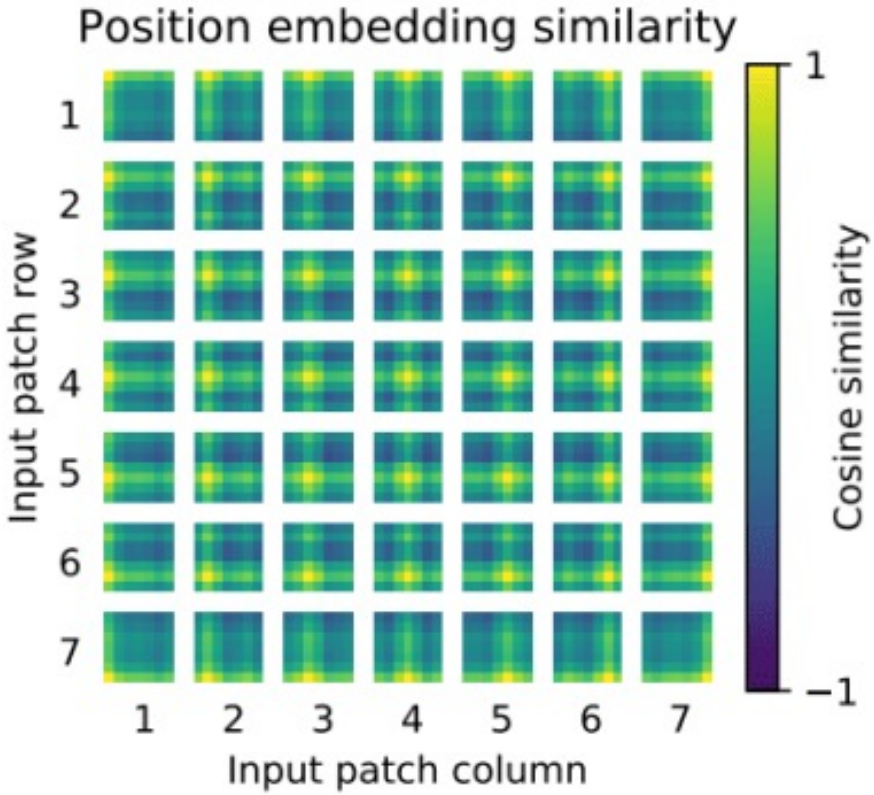
Results

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21k (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Table 2: Comparison with state of the art on popular image classification benchmarks. We report mean and standard deviation of the accuracies, averaged over three fine-tuning runs. Vision Transformer models pre-trained on the JFT-300M dataset outperform ResNet-based baselines on all datasets, while taking substantially less computational resources to pre-train. ViT pre-trained on the smaller public ImageNet-21k dataset performs well too. *Slightly improved 88.5% result reported in Touvron et al. (2020).

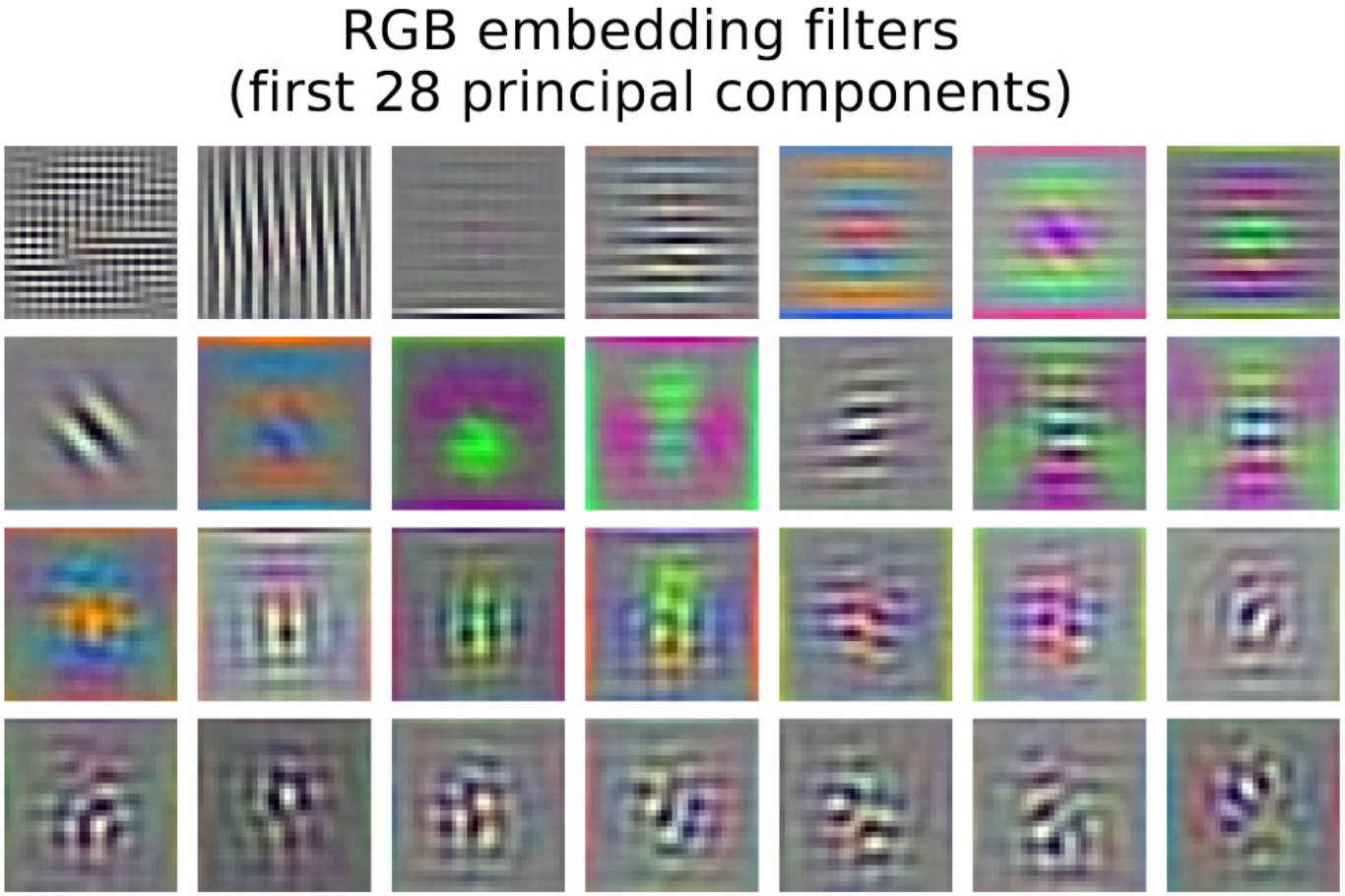
- ViT is trained on datasets with more than 14M images it can approach or beat state-of-the-art CNNs (ResNets or EfficientNets).

Results: Learned Position Embeddings



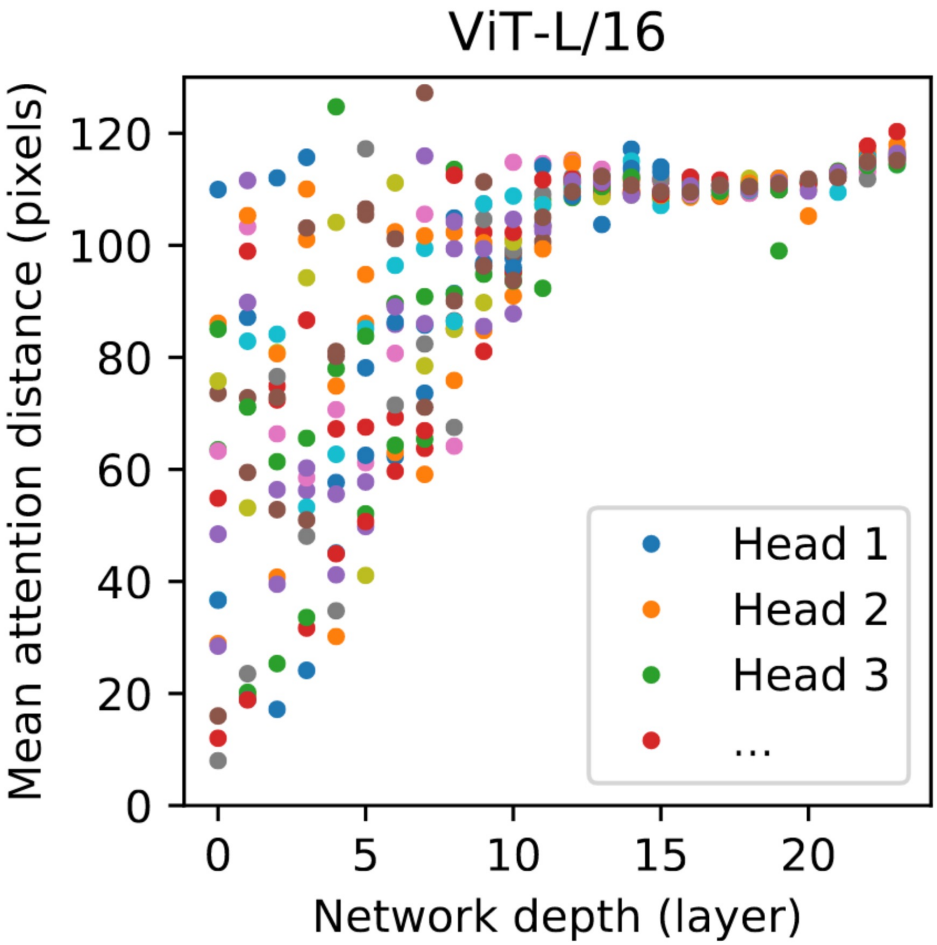
- Similarity of position embeddings of ViT-L/32.
- Tiles show the cosine similarity between the position embedding of the patch with the indicated row and column and the position embeddings of all other patches.
- The first patch (1,1) has high correlation with others which are on same row (1,.) and col(.1)

Result: Learned Weight Embeddings

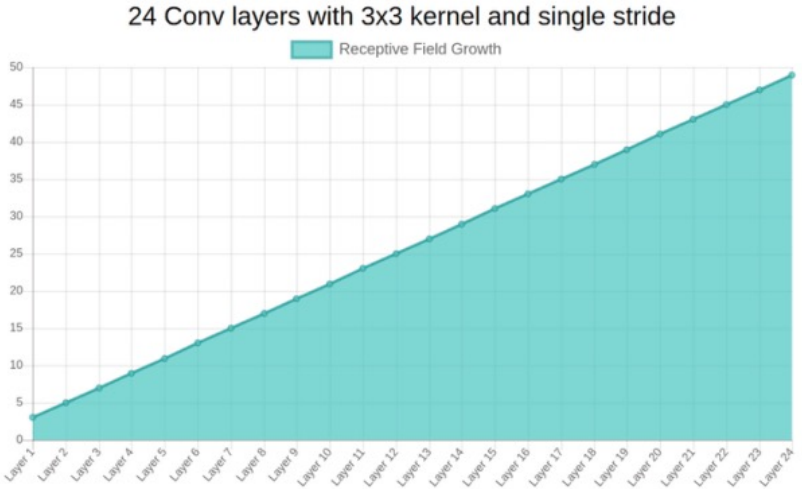


Similar to Alex Net, typical patterns are most important for embedding

Result: Attention



In the last layers, the network mostly focuses on the image as a whole.



Code for VIT Feature Extraction

```
#
model_name = "google/vit-large-patch32-224-in21k"
model = ViTModel.from_pretrained(model_name)


# Some image
url = 'http://images.cocodataset.org/val2017/000000039769.jpg'
image = Image.open(requests.get(url, stream=True).raw)

# Correct image processor
processor = ViTImageProcessor.from_pretrained(model_name)
inputs = processor(images=image, return_tensors="pt")

outputs = model(**inputs)
last_hidden_states = outputs.last_hidden_state
class_token = last_hidden_states[:, 0]
print(last_hidden_states.shape, class_token.shape)
```

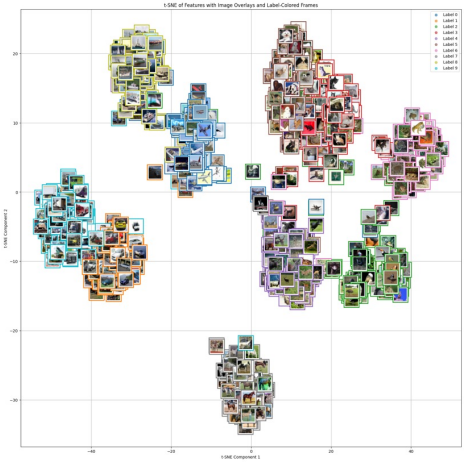
Possible Excercise

- Small Demo: 03_VIT_Demo.ipynb

- Possible Use a VIT for feature extraction 02_transfer_learning.ipynb

Possible Exercise: Results

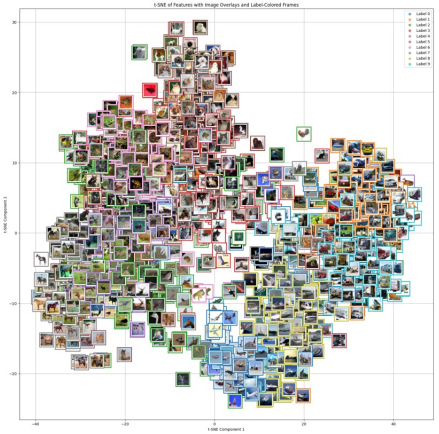
- Use a VIT for feature extraction in NB



```
Data Set: CIFAR10
Extractor: vit
Training Data: 800
Validation Data: 200
Test Data: 2000
```

Data	Model	Accuracy
RAW	Random Forest	0.3195
RAW	CNN	0.1280
Features	Random Forest	0.9415
Features	Neural Network	0.9500

- Use a VGG16 for feature extraction (last week)



```
Data Set: CIFAR10
Extractor: VGG16
Training Data: 800
Validation Data: 200
Test Data: 2000
```

Data	Model	Accuracy
RAW	Random Forest	0.3195
RAW	CNN	0.3020
Features	Random Forest	0.7495
Features	Neural Network	0.7645

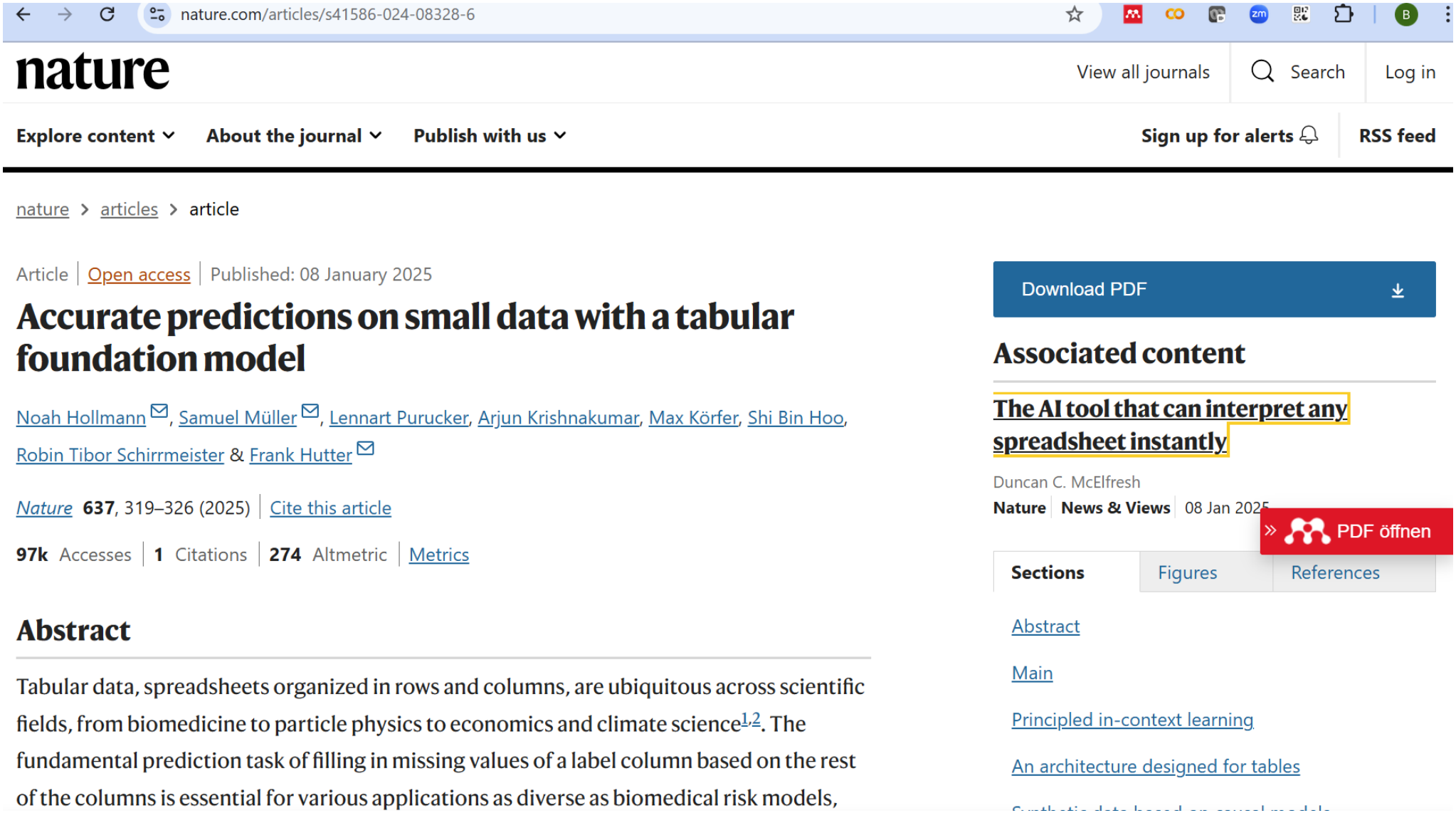
Summary on VIT

- Less inductive bias compared to CNN
  - Mainly creation of patches
  - No weight sharing etc.
- Need much data to learn features
  - Only trained on ImageNET21k (14 Million Images / 21000 Classes)

- Variants
  - Models working with unsupervised pretraining as LLM
    - DINO
  - Images of different scales
    - **Swin Transformer** (shifted window Transformer) is a **hierarchical** Vision Transformer (ViT)

TabPFN
a tabular foundation model

TabPFN: Tabular data based Prior-data Fitted Networks
a transformer based foundation model for tabular data



nature.com/articles/s41586-024-08328-6

nature

View all journals | Search | Log in

Explore content | About the journal | Publish with us | Sign up for alerts | RSS feed

nature > articles > article

Article | Open access | Published: 08 January 2025

Accurate predictions on small data with a tabular foundation model

Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister & Frank Hutter

Nature **637**, 319–326 (2025) | Cite this article

97k Accesses | **1** Citations | **274** Altmetric | Metrics

Abstract

Tabular data, spreadsheets organized in rows and columns, are ubiquitous across scientific fields, from biomedicine to particle physics to economics and climate science[1,2]. The fundamental prediction task of filling in missing values of a label column based on the rest of the columns is essential for various applications as diverse as biomedical risk models,

Download PDF

Associated content

The AI tool that can interpret any spreadsheet instantly

Duncan C. McElfresh

Nature | **News & Views** | 08 Jan 2025

PDF öffnen

Sections | Figures | References

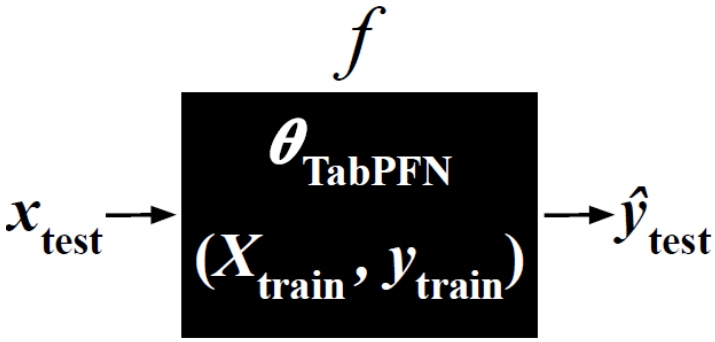
Abstract

Main

Principled in-context learning

An architecture designed for tables

Background on TabPFN

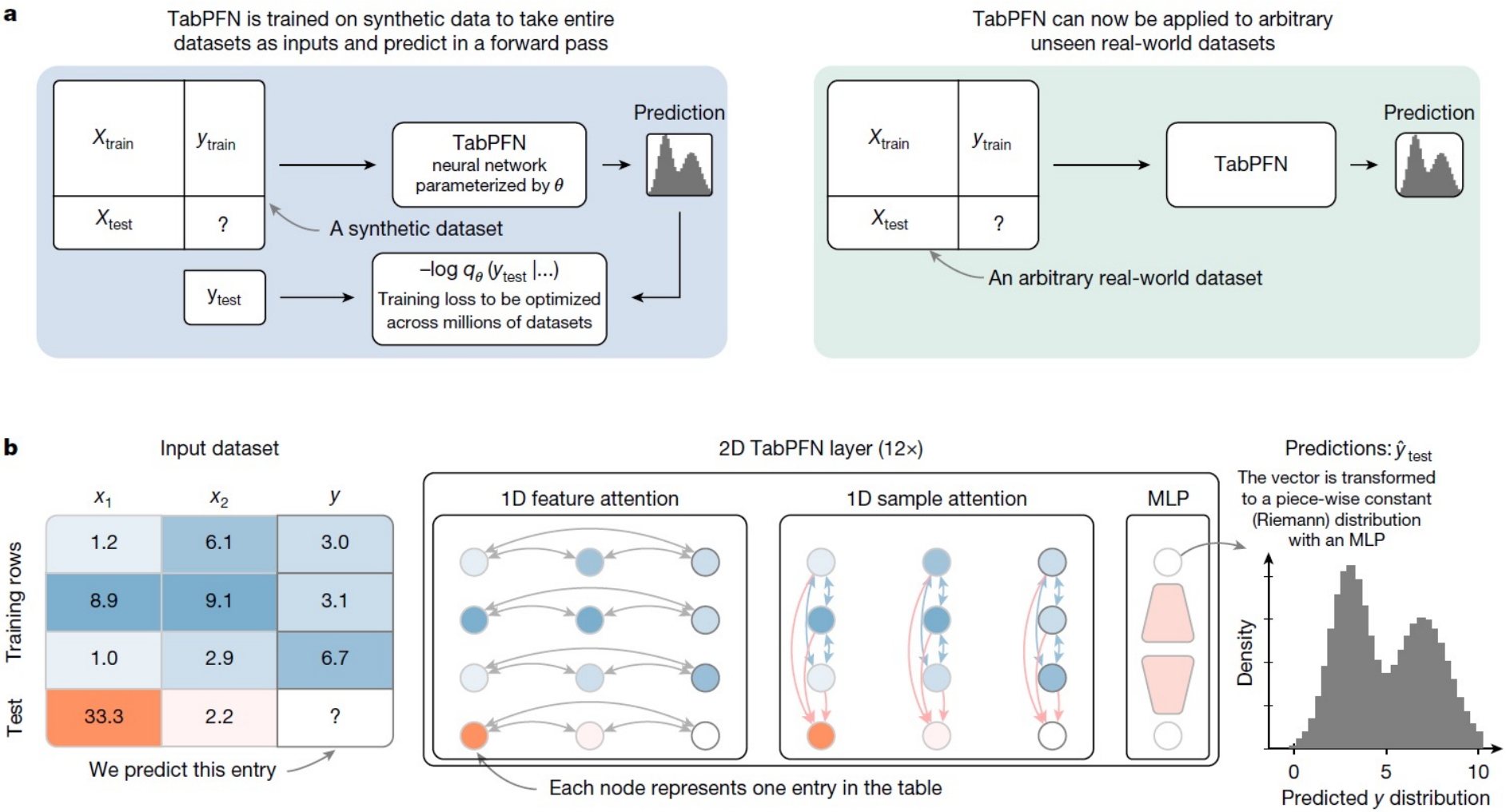


TabPFN[1] is a prior-data fitted network (PFN) that was pretrained to perform in-context learning on fresh tabular classification problems.

- Meta-learned Transformer model (26M parameters)
- Trained using synthetically-generating data: structural causal models are randomly generated, then training datasets are sampled from each causal model.
- On fresh classification tasks, no training (i.e. weight updating) is needed; instead, training data is given as context.

[1] TabPFN: A Transformer That Solves Small Tabular Classification Problems in a Second. N. Hollmann, S. Muller, K. Eggensperger, F. Hutter. ICLR 2023.

TabPFN is a transformer model with 2-way attention



"To better use the tabular structure, we propose an architecture that assigns a separate representation to each cell in the table, inspired by refs. 22,28. Our architecture, visualized in Fig. 1b, uses a two-way attention mechanism, with each cell attending to the other features in its row (that is, its sample) and then attending to the same feature across its column (that is, all other samples)."

Note: at the moment TabPFN seems to have problems with periodic structures such as seen in time series.

TabPFN is "prompted" with train and test data

- Instead of using prompts as context inputs, TabPFN uses the real data $(y_{\text{train}}, x_{\text{train}}), (x_{\text{test}})$.
- Hence, prompting in TabPFN can also be seen as data distillation.

```
from tabpfn import TabPFNClassifier, TabPFNRegressor

reg = TabPFNRegressor()

reg.fit(x_train, y_train)

preds = reg.predict(x_test, output_type="full")

preds.keys() #dict_keys([… 'median', 'quantiles'])

median = preds['median']
```

https://github.com/tensorchiefs/dlwbl_eth25/blob/master/notebooks/03_tabPFN_sinus.ipynb

Possible Exercise: TabPFN for non-linear regression

- https://github.com/tensorchiefs/dlwbl_eth25/blob/master/notebooks/03_tabPFN_sinus.ipynb