Machine Intelligence:: Deep Learning
Week 5

Beate Sick, Loran Avci, Oliver Dürr

Part I:

Winterthur, 23. March. 2021

Outline of the DL Module (tentative)

- Day 1: Jumpstart to DL
  - What is DL
  - Basic Building Blocks
  - Keras
- Day 2: CNN I
  - ImageData
- Day 3: CNN II and RNN
  - Tips and Tricks
  - Modern Architectures
  - 1-D Sequential Data
- Day 4: Looking at details
  - Linear Regression
  - Backpropagation
    - Resnet
  - Likelihood principle
- Day 5: Probabilistic Aspects
  - Likelihood principle (cont'd)
  - TensorFlow Probability (TFP)
  - Negative Loss Likelihood NLL
  - Count Data
- Day 6: Probabilistic models in the wild
  - Complex Distributions
  - Generative modes with normalizing flows
- Day 7: Uncertainty in DL
  - Bayesian Modeling
- Day 8: Uncertainty cont'd
  - Bayesian Neural Networks
  - Projects

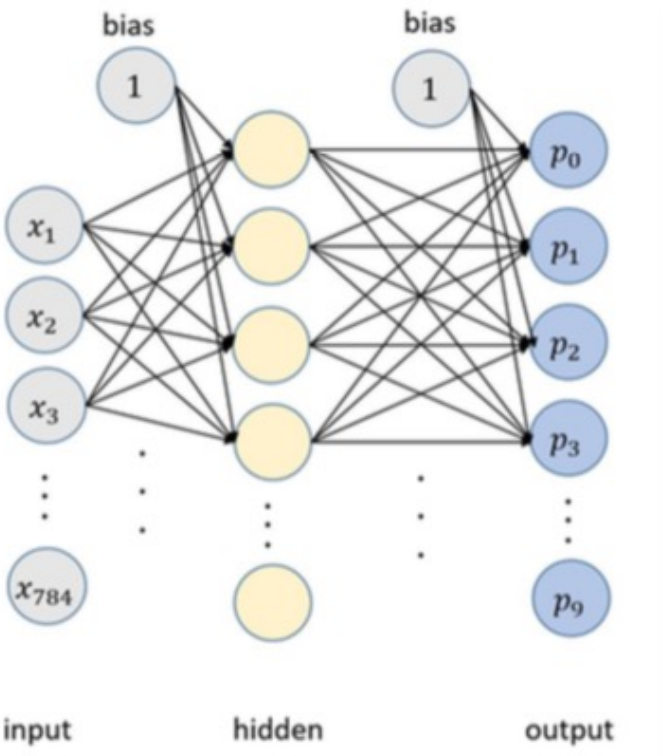
Projects please register (see website)
https://docs.google.com/spreadsheets/d/18VFrPbKq3YSOg8Ebc1q1wGgkfgaWl7IkcCClGEDGj6Q/edit#gid=0

Learning Objectives for today

- Get an understanding of
  - Likelihood Principle (cont'd)
  - Probabilistic Models
  - TensorFlow Probability
  - Exercise

  Beate ~ 11:00
  - NLL as quality measure of model (scoring rule)
  - Count Data
  - Exercise

More than 2 classes (recap)



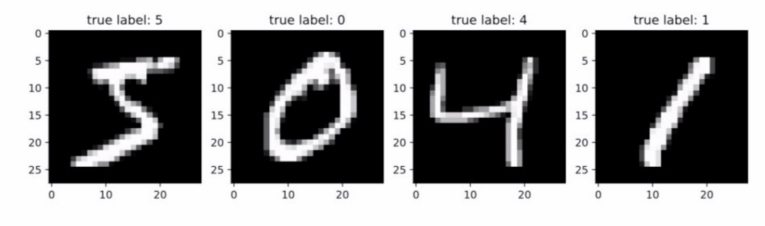


Figure 2.11 The first four digits of the MNIST data set—the standard data set used for benchmarking NN for images classification

$$crossentropy = -\frac{1}{n}\left(\sum_{j\ for\ y=0}\log(p_0(x_j)) + \sum_{j\ for\ y=1}\log(p_1(x_j)) + \ldots + \sum_{j\ for\ y=K-1}\log(p_{k-1}(x_j))\right)$$

$$crossentropy = -\frac{1}{n}\sum_{i=1}^{n} {}^{true}p_i \cdot log(p_i)$$

Loss function for regression

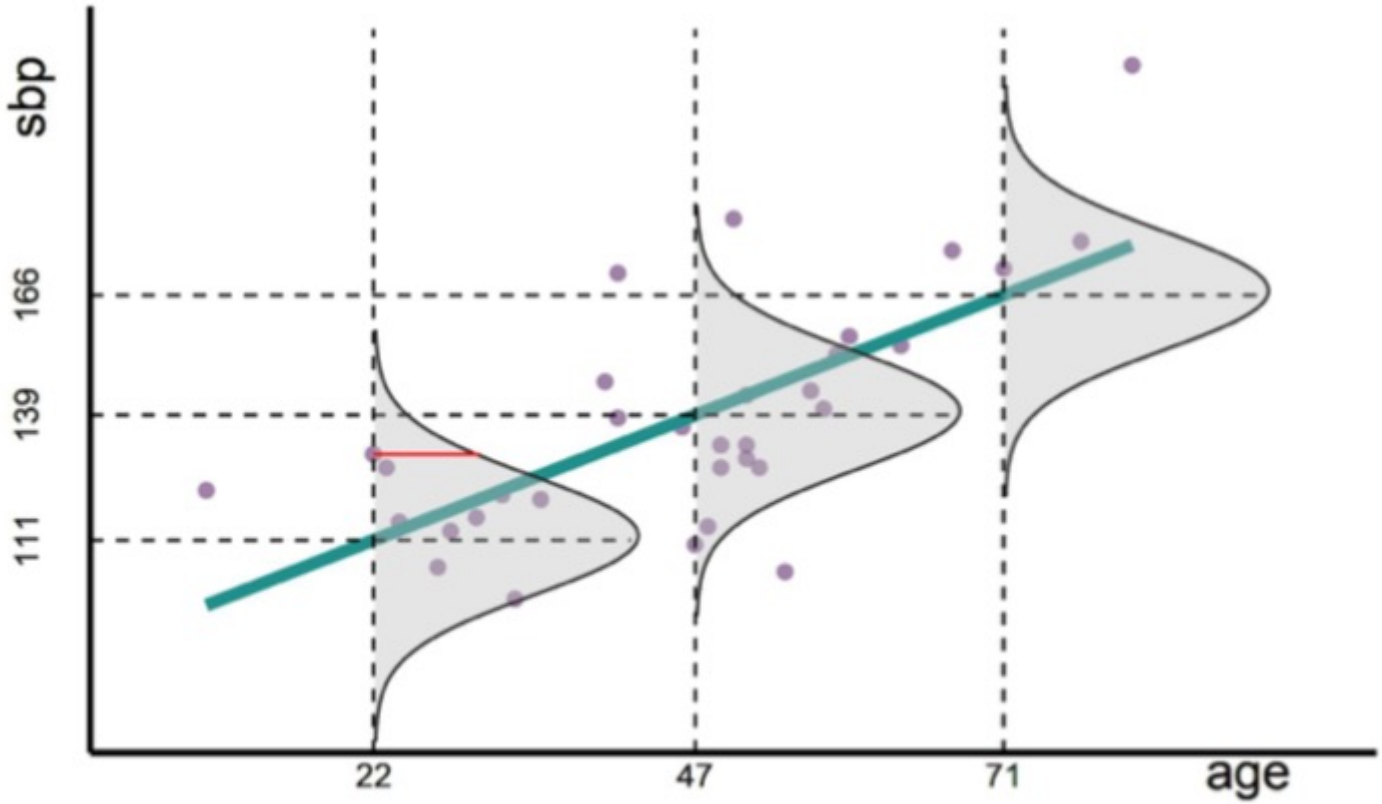
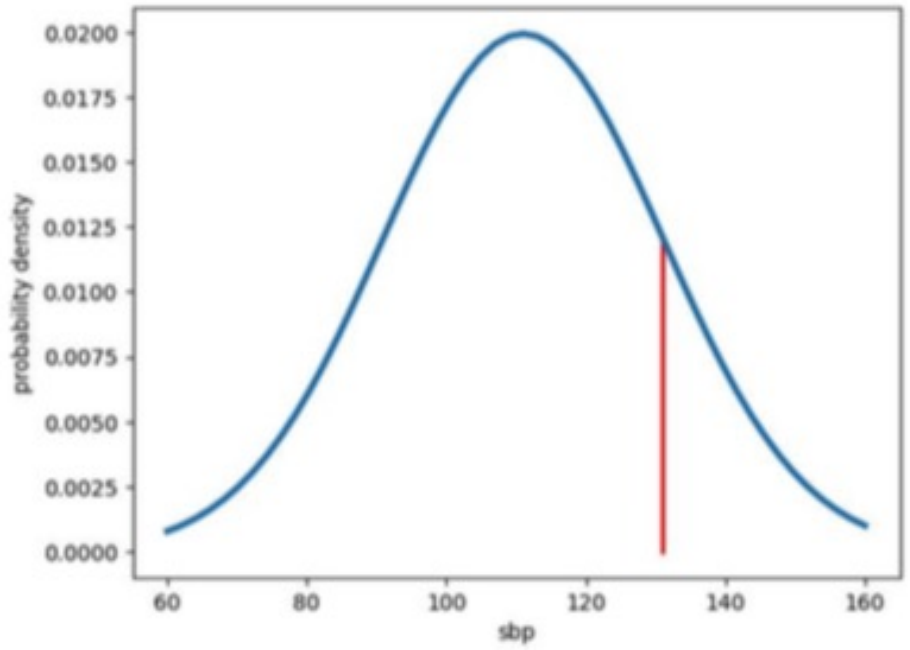


Figure 4.12 Scatterplot and regression model for the blood pressure example: The dots are the measured data points, the straight line is the linear model, the bell-shaped curves are the conditional probability distributions of the outcome Y conditioned on the observed value x. The length of the red bar indicates likelihood for the observed sbp of 131 for a 22 year old woman.

Zoom for one example



$$f(y = 131, \mu = 111, \sigma = 20) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu)^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi \cdot 400}} e^{-\frac{(131-111)^2}{2400}} \approx 0.01209$$

Figure 4.13 The conditional normal density function f. The height of the red bar indicates the likelihood of the specific value under this model.

For all training Examples:

$$L(a,b) = f(y_1, \mu_{x_2}, \sigma) \cdot f(y_2, \mu_{x_2}, \sigma) \ldots = \prod_{i=1}^{n} f(y_i, \mu_i, \sigma) = \prod_{i=1}^{n} f(y_i, a \cdot x_i + b)$$

Exercise from last time

- Now you don’t know how many dollar signs are on the die.
- You throw the die 10 times and get k=2 dollar signs.
- What is you best guess?

- Work Through Exercise:
- Work through the code until you reach the first exercise. In the exercise it is your task to determine the probability to observe two-times a dollar sign in ten dice throws, if you consider a die that has dollar signs on 0, 1, 2, 3, 4, 5, or all 6 faces.
- https://github.com/tensorchiefs/dl_book/blob/master/chapter_04/nb_ch04_01.ipynb
-

Exercise from last time

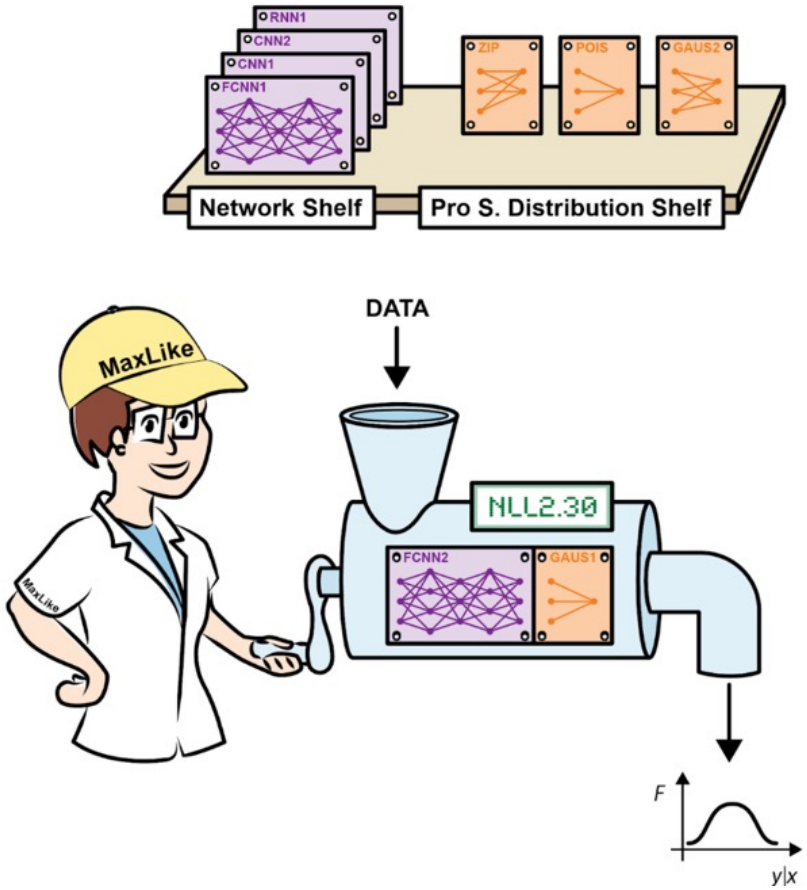
Task 1:
* Calculate the excepted cross-entropy for the MNIST example if you just guess, each class with p=1/10.

* Load MNIST and make a small CNN without training and calculate the loss

Task 2 (for reference)
Work through the code until you reach the exercise indicated by a pen icon, then do the exercise. Your task in the exercise is to use the digit predictions on MNIST images which are made by an untrained CNN and compute the value of the loss by 'hand'. You'll see that you get a value close to the value 2.3 which you calculated above.

https://github.com/tensorchiefs/dl_book/blob/master/chapter_04/nb_ch04_02.ipynb

Probabilistic models



Outline

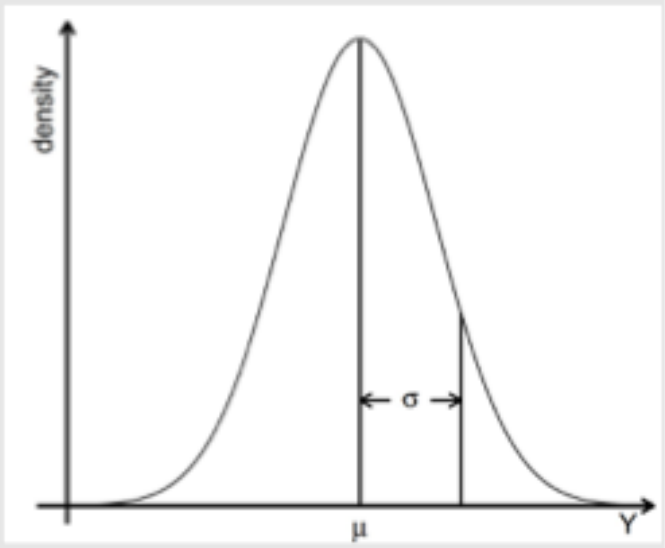
- Recap Probability Distribution
- What is a probabilistic model?

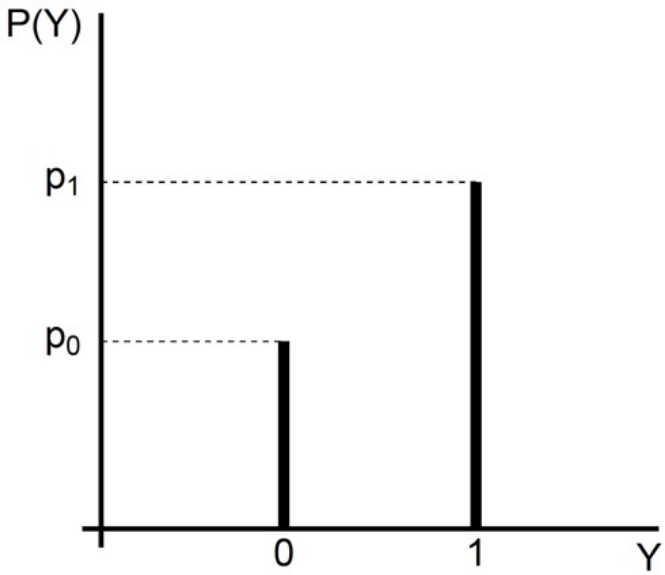
  → predicts whole **c**onditional **p**robability **d**istribution (CPD)
- How to fit a probabilistic DL model?

  → use NLL as loss function
- How to evaluate a probabilistic DL model?
  - Definition of scoring functions
  - The logarithmic score and it uniqueness
- Examples using count data

Recap: probability distributions

- Describe a random variable here called *y*
- Characterized by probability density (pdf) for continuous or by the probability mass (pmf) for discrete distributions

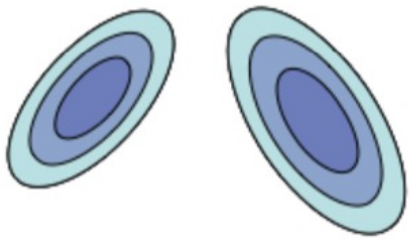




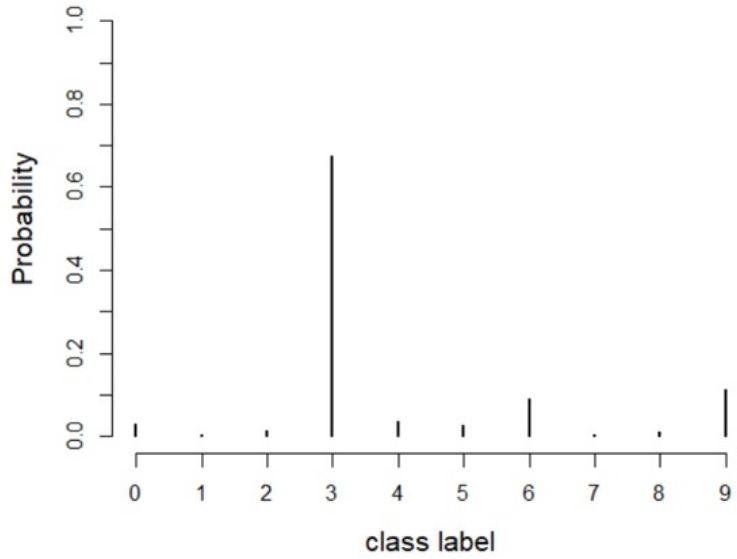
- Few parameters describe the distribution
  - E.g. μ and σ for a Gaussian $\mathrm{p}(\mathrm{y};\mu,\sigma)$ (sometimes $\mathrm{f}(\mathrm{y};\mu,\sigma)$ to stress density)
  - E.g. p_1 for Bernoulli

Recap: Multivariate probability distributions

- Random variable *y* is multidimensional
- Characterized by probability density (pdf) for continuous or probability mass for discrete distributions



$$\begin{pmatrix} \sigma_1^2 & \sigma_{12} \\ \sigma_{21} & \sigma_2^2 \end{pmatrix}$$



- Few parameters describe the distribution
  - E.g. $\vec{\mu}$ and $\mathbf{\Sigma}$ for a Gaussian $\mathrm{f}(\vec{y}; \vec{\mu}, \mathbf{\Sigma})$
  - E.g. $(p_0, p_1 \ldots, p_9)$ for categorical* with values $\vec{y}$="0","1",…,"9"

*There is some confusion how to name this distribution. Sometimes it is also know as multinomial.

TensorFlow Distributions (Part I)

TensorFlow Probability (TFP) is nice add-on to Keras
Nice abstraction, different distributions all have the same API.

Methods for Tensorflow distributions	Description	Numerical result when calling the method on `dist = tfd.Normal(loc=1.0, scale=0.1)`

`sample(n)`	Samples *n* numbers from the distribution.	`dist.sample(3).numpy()`
		
		`array([1.0985107, 1.0344477, 0.9714464], dtype=float32)`
		
		Note that these are random numbers

`prob(value)`	Returns the likelihood (probability density in the case of models for continuous outcomes) or probability (in the case of models for discrete outcomes) for the values (tensor)	`dist.prob((0,1,2)).numpy()`
		
		`array([7.694609e-22, 3.989423e+00, 7.694609e-22], dtype=float32)`

`log_prob(value)`	Returns the log-likelihood or log-probability for the values (tensor)	`dist.log_prob((0,1,2)).numpy()`
		
		`array([-48.616352 , 1.3836466, -48.616352 ], dtype=float32)`

`cdf(value)`	Returns the cumulative distribution function (CDF) that this is a sum or the integral, up to the given values (tensor)	`dist.cdf((0,1,2)).numpy()`
		
		`array([7.619854e-24, 5.000000e-01, 1.000000e+00], dtype=float32)`

`mean()`	Returns the mean of the distribution	`dist.mean().numpy()`
		
		`1.0`

`stddev()`	Returns the standard deviation of the distribution	`dist.stddev().numpy()`
		
		`0.1`

Demo time: TensorFlowDistributions:

https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/12_TensorFlowDistributions_Demo.ipynb

What is a probabilistic model?

Simple regression via a NN: no probabilistic model in mind

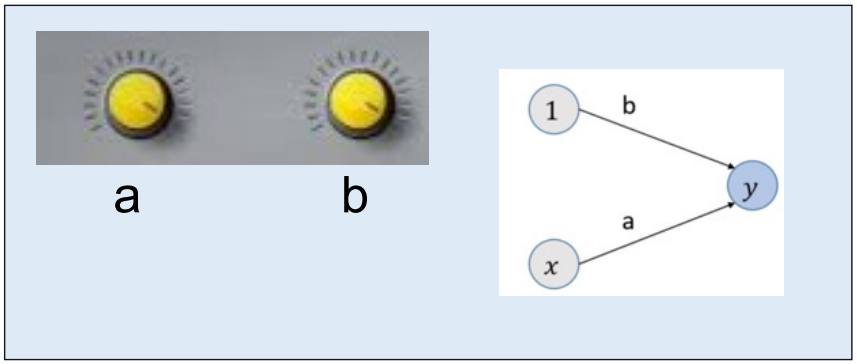
age

Input x

Systolic blood pressure

Output y

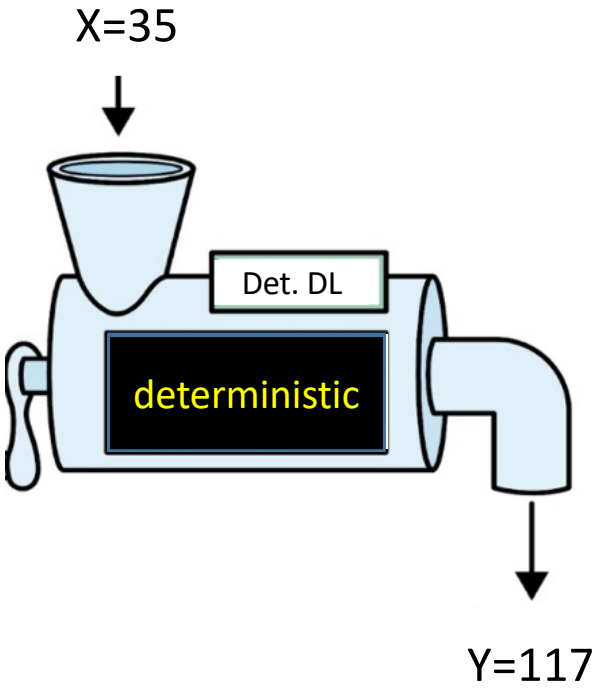
$x = 35$ →

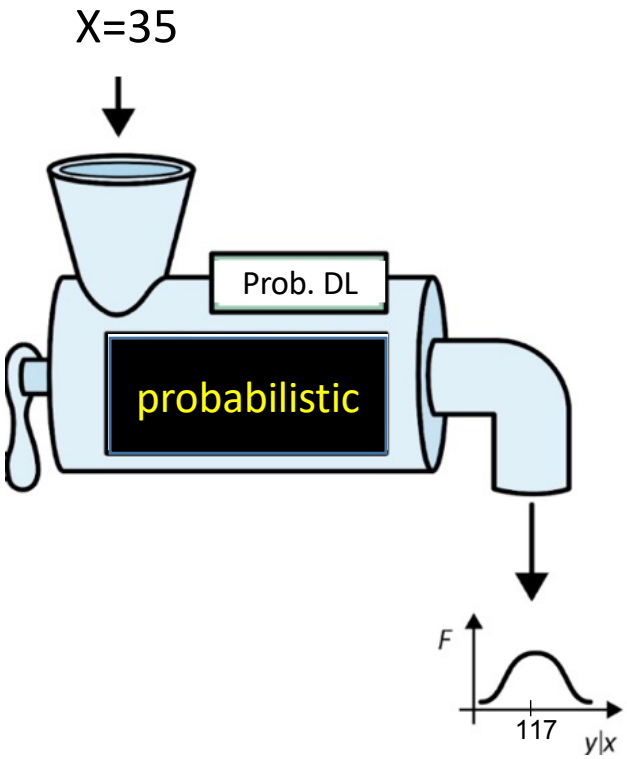


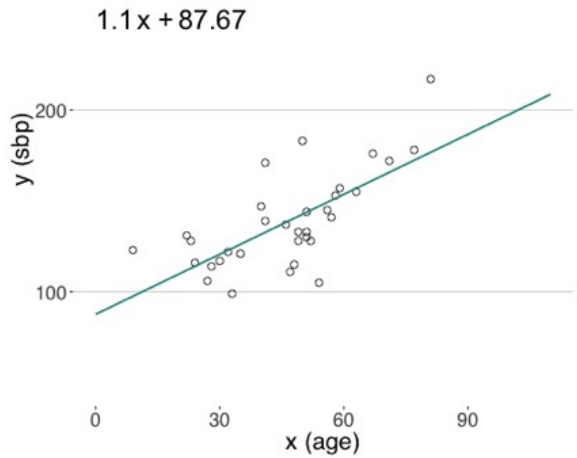
→ $\hat{sbp} = 117$

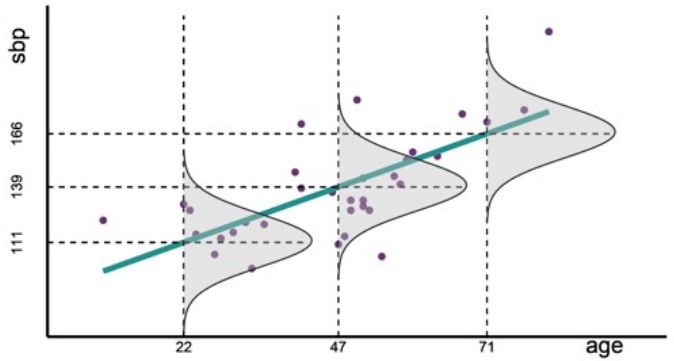
One input x (age) → one predicted outcome (sbp)

Traditional versus probabilistic regression DL models









Describes the spread of the data

Deterministic versus probabilistic classification DL models

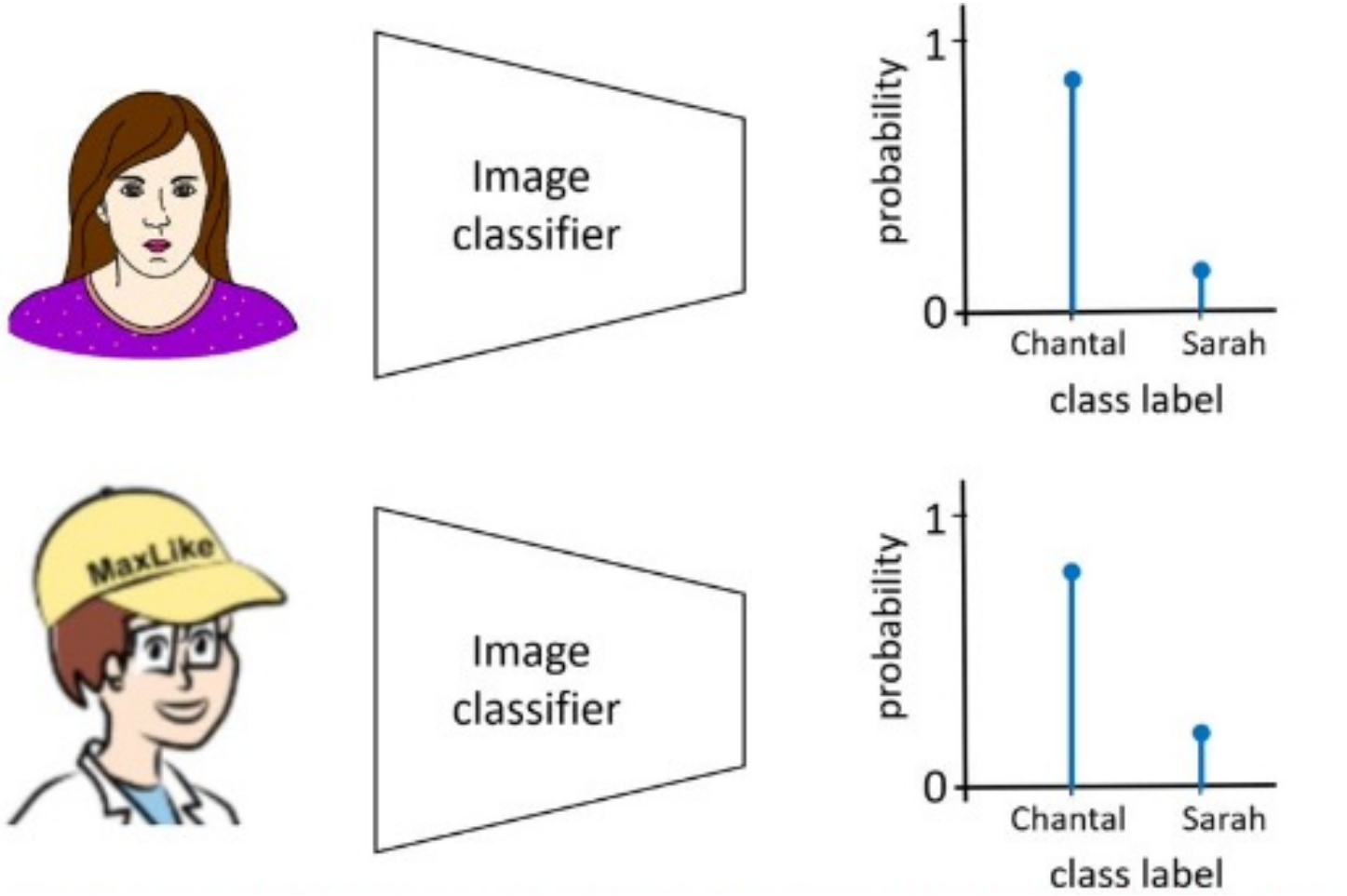
Image classifier

Chantal

Image classifier

Chantal

Classification in DL is already probabilistic



Why is it important to know about probabilities?

Philosophical reasons:

"The true logic of this world is in the calculus of probabilities"
James Clerk Maxwell

"It is scientific to say what is more likely and what is less likely..."
Richard Feynman

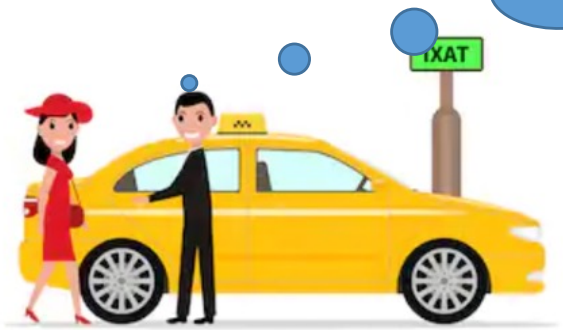
Practical reasons:

- We often want to optimize expected costs which requires CPD for computing.
- They come for free in most DL problems
- Choosing the right probabilistic model enhances existing DL architectures
  - Parallel Wavenet (using mixtures of logistics distributions as output)
  - PixelCNN++ (using mixtures of logistics distributions as output)

Probabilistic travel time prediction (cost)

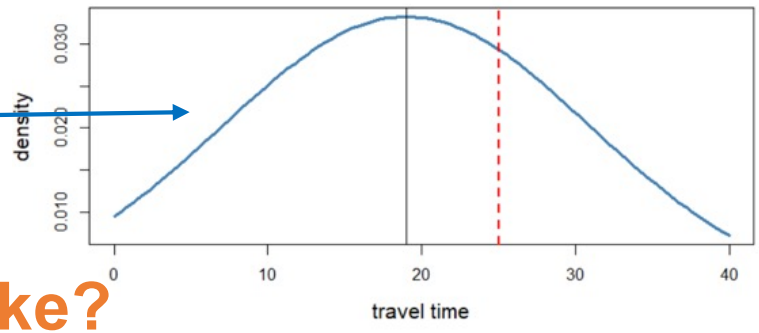
You'll get 500$ tip if I arrive at MOMA within 25 minutes!

Let's use my probabilistic travel time gadget!

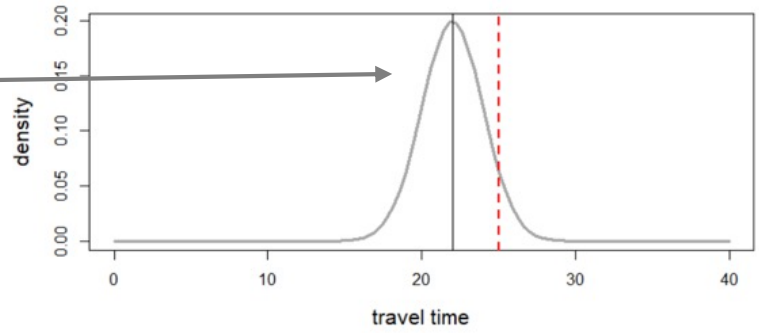


Which way would you take?

Chance to get tip: 69%



Chance to get tip: 93%

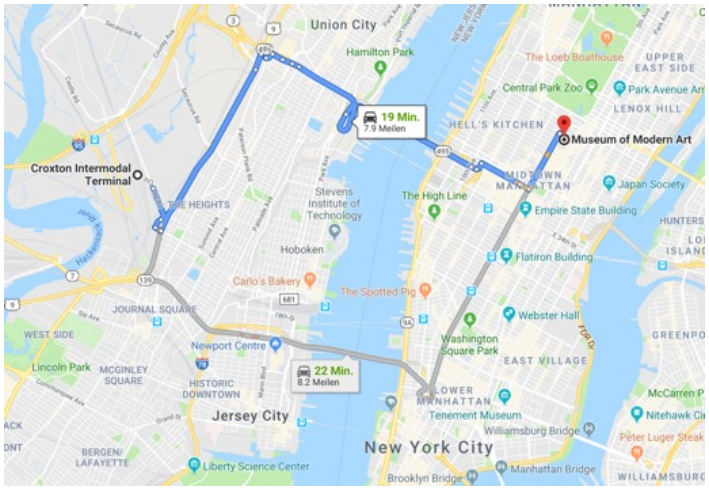


Conditional probability distribution

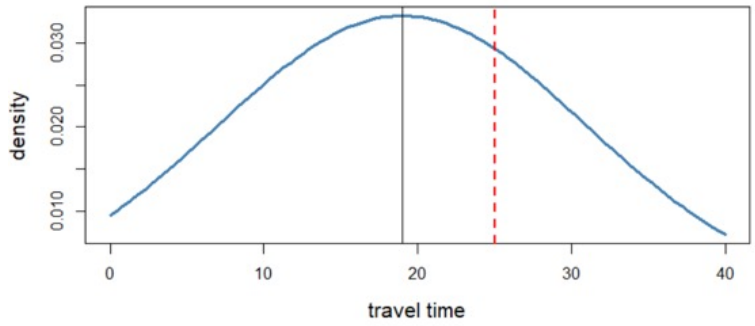
- Probability distribution now depends (“*is conditioned on*”) on an external value x

Example

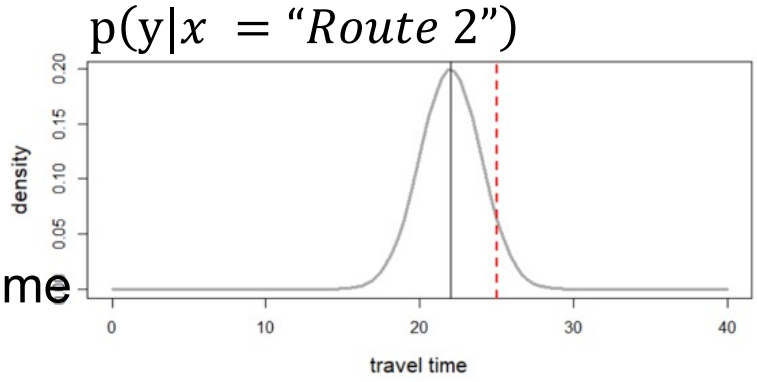
$$p(y) \rightarrow p(y|x)$$



$p(y|x = \text{“}Route\ 1\text{”})$



$p(y|x = \text{“}Route\ 2\text{”})$

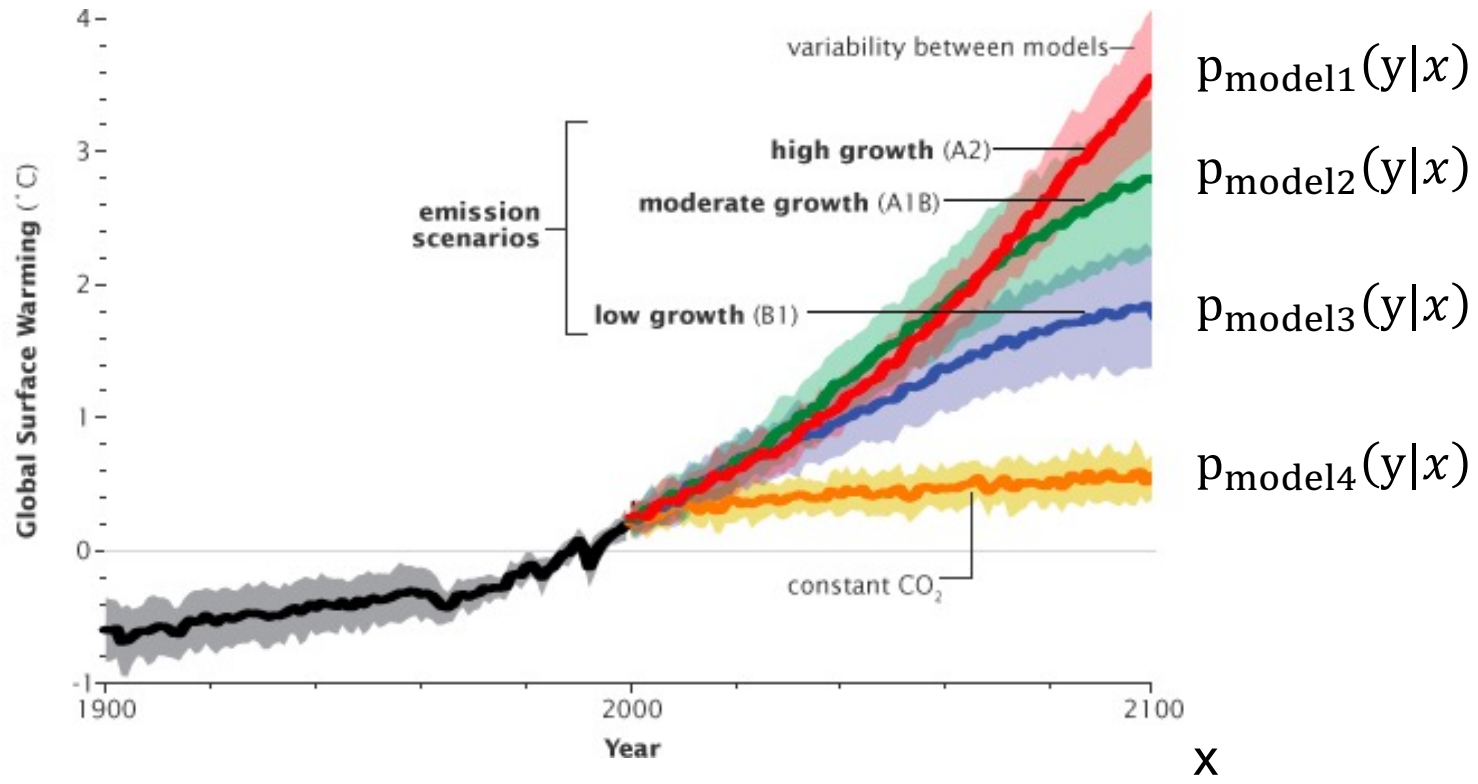


$p(y|x)$

- y is a 1D continuous variable travel time
- x codes the route

Examples of probabilistic models: global warming forecast

For each time-point x a probability distributions is reported (for 4 models)

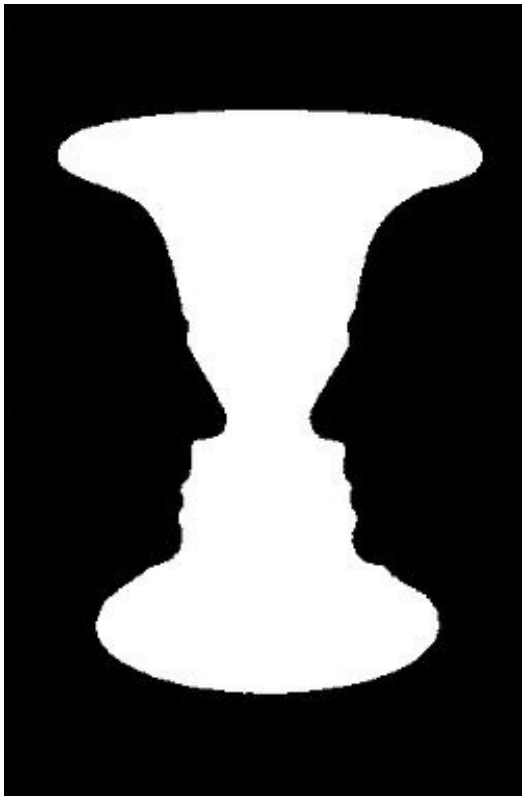


$p(y|x)$

- y is a 1D continuous variable (of the increase of the global surface temperature)
- x is 1D continuous time

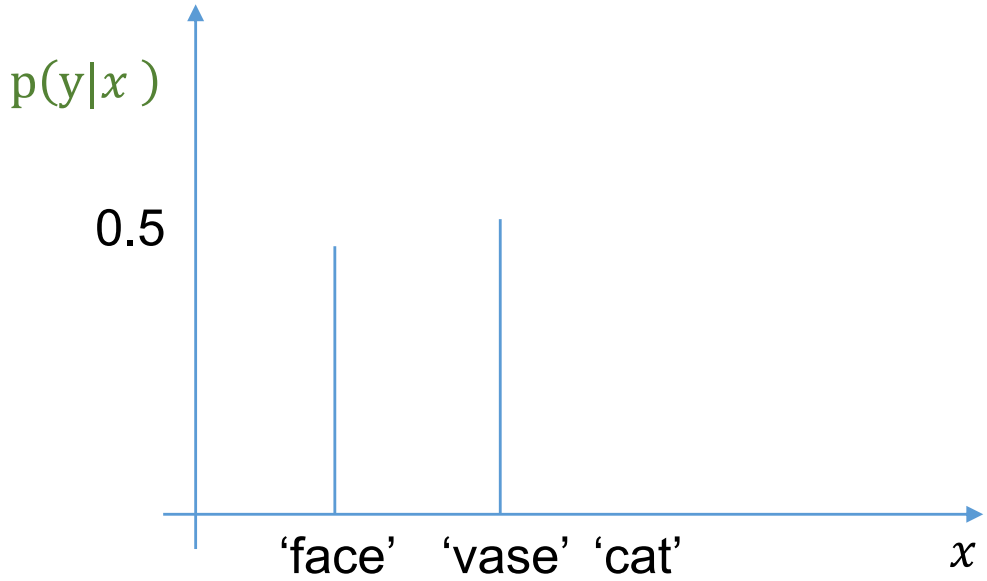
Figure from: https://earthobservatory.nasa.gov/features/GlobalWarming

Examples of probabilistic models: image recognition



$p(y|x)$

- y is a categorical discrete variable y='face','vase','ca
- x the image (e.g. 256x256)



Remark: This is a quite uncommon situation in image classification. Usually there is only one possibility.

Image credit: https://en.wikipedia.org/wiki/Rubin_vase

CPD in ML / deep learning setting

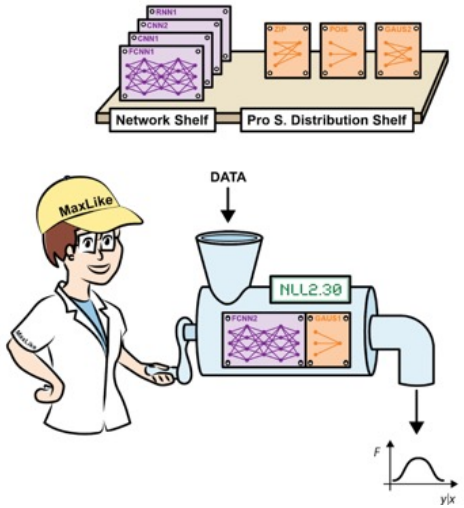
- In machine learning, we learn from *N* examples $D = \{x^{(i)}, y^{(i)}\}$ with $\mathrm{i} = 1, \ldots N$. CPD:

$$p(y|x, D)$$

- After training all information is in the weights w

$$p(y|x, w)$$

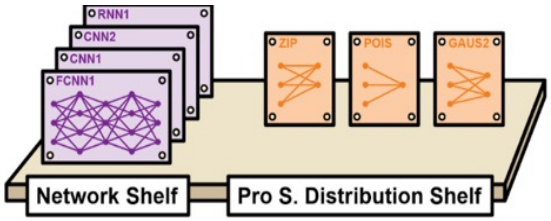
- Deep Learning recipe for probabilistic models

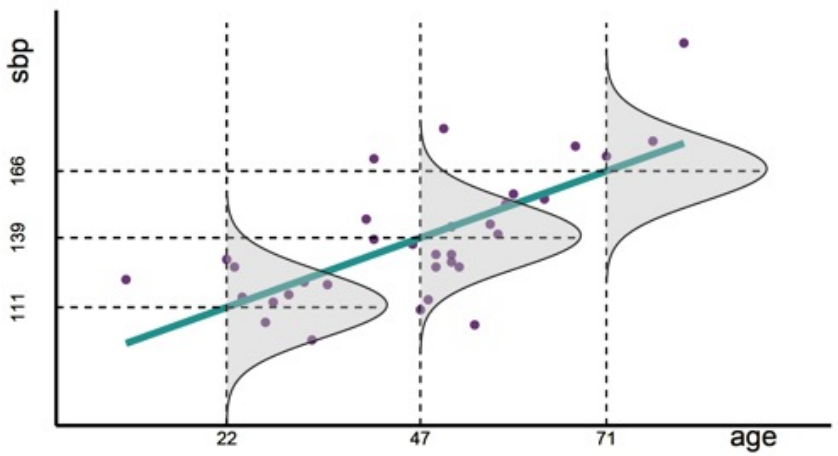


The network determines the parameters of a CPD given an *x*.

Regression output as CPD

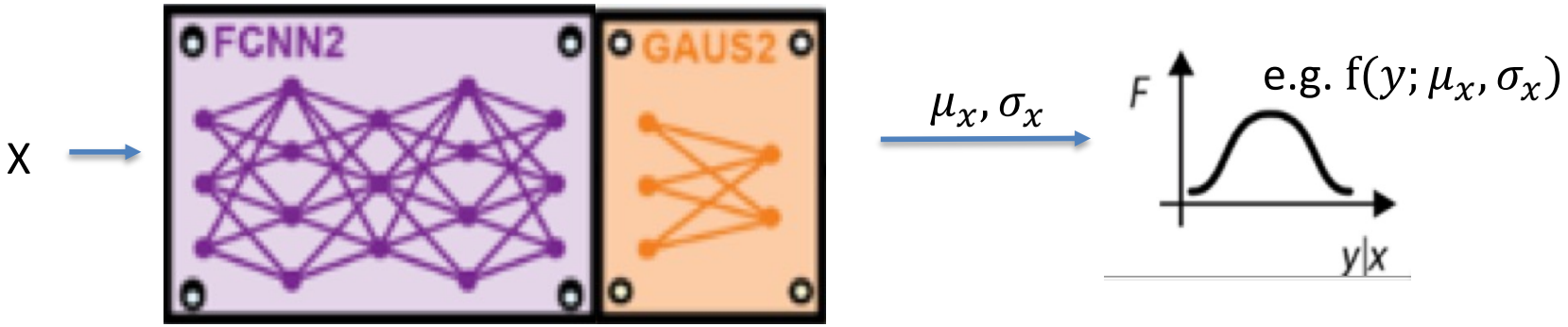
- A trained network outputs for each x the parameters of a distribution.
- E.g. network output μ_x, σ_x for a given x.





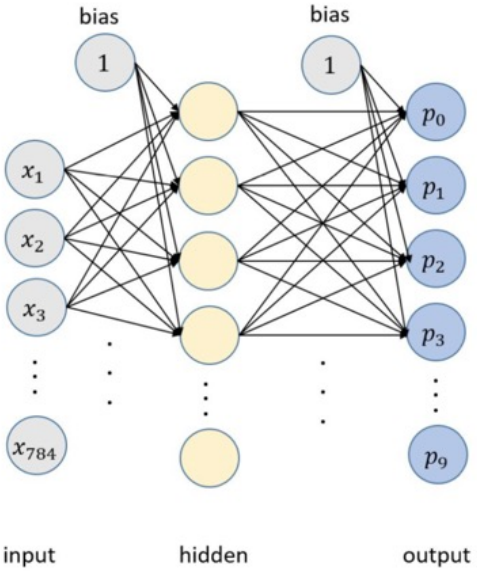
Input: Images, numbers,…

Output: Parameters of output distribution



Classification Output as CPD

- Binary Classification single output is sufficient (prob. p_1 for class 1)
- This can be seen as parameter p_1 for Bernoulli distribution
  - $y \sim Bern(p_1)$
- In general classification settings the output at node *k* can be interpreted as probability for that class **or alternatively as parameters** $p_1, p_2, \ldots, p_k$ **for a categorical CDF** with k categories.



The output values are parameters for a categorical probability distribution.

$$p(Y = k \mid x, w) = \begin{cases} p_0(x, w) & \text{for k=0} \\ p_1(x, w) & \text{for k=1} \\ \vdots & \vdots \\ p_9(x, w) & \text{for k=9} \end{cases} \quad \text{with} \sum_{i=0}^{9} p_i(x, w) = 1$$

CPD: Unifying framework for classification and regression. Network controls the parameters of a CPD for given value x.

How to fit a probabilistic model?

Maximum Likelihood

Tune the parameters weights of the network, so that observed data (training data) is most likely.

Practically: Minimize Negative Log-Likelihood of the CPD

$$\hat{w} = argmin \sum_{i=i}^{N} -\log(p(y_i|x_i, w))$$

Two practical ways of modeling and fitting CPD

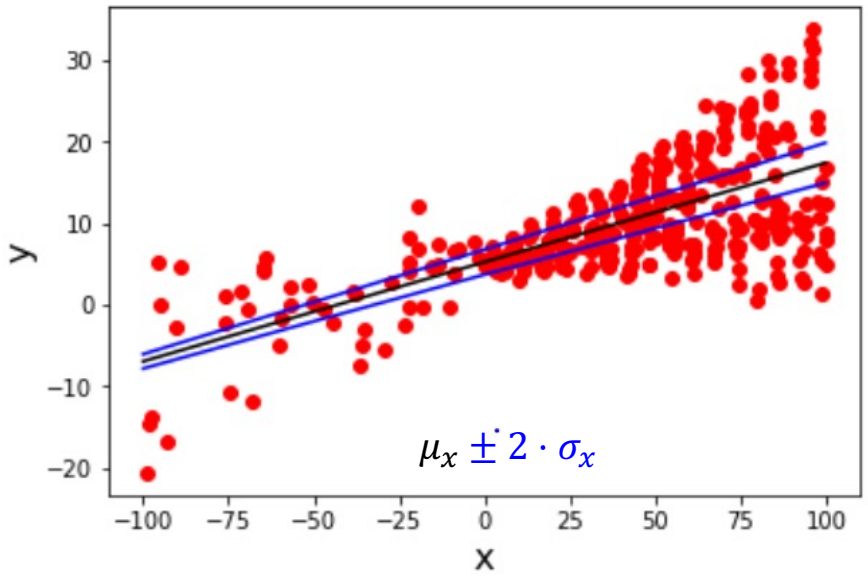
Approach 1 (explicit loss function)

- Use a NN and interpret the output(s) as the parameters of a CPD (e.g. μ and σ in case of a Gaussian)
- Use the correct loss function or provide a custom loss function for non-standard CPDs.
- Keras has build in loss functions (https://keras.io/losses/) for the following CPDs:
  - Classification
    - Bernoulli: loss='binary_crossentropy '
    - Categorical: loss='categorical_crossentropy '
  - Regression (see also extra slide)
    - Gaussians (with fixed σ): loss='mean_squared_error'
    - Poisson: loss='poisson'

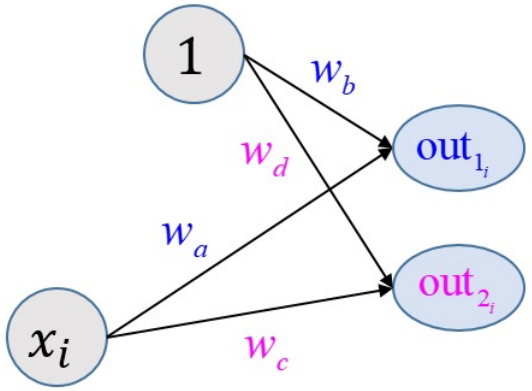
Approach 2 (Model the CPD)

- Use a NN with as many outputs as parameters of the CPD
- Use the output to model the parameters of the CPD
- This approach can be done with the TensorFlow Probability (TFP) Framework

Fit a probabilistic regression with non-constant variance



$$Y_{X_i} \sim \mathrm{N}(\mu_{x_i}, \sigma_x^2)$$



$$\mu_{x_i} = \mathrm{out}_{1_i}$$

$$\sigma_{x_i} = e^{\mathrm{out}_{2_i}}$$

Minimize the negative log-likelihood (NLL):

$$\hat{w} = argmin \sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma_{x_i}^2}}\right) + \frac{(y_i - \mu_{x_i})^2}{2\sigma_{x_i}^2}$$

gradient descent with NLL loss

$\hat{w}_a$ $\hat{w}_b$

$\hat{w}_c$ $\hat{w}_d$

Modelling the standard deviation (positive values)

- The variance or standard deviation are both positive
- Neural networks output is not constrained.
- Two common approaches to fix this (exp or softplus)

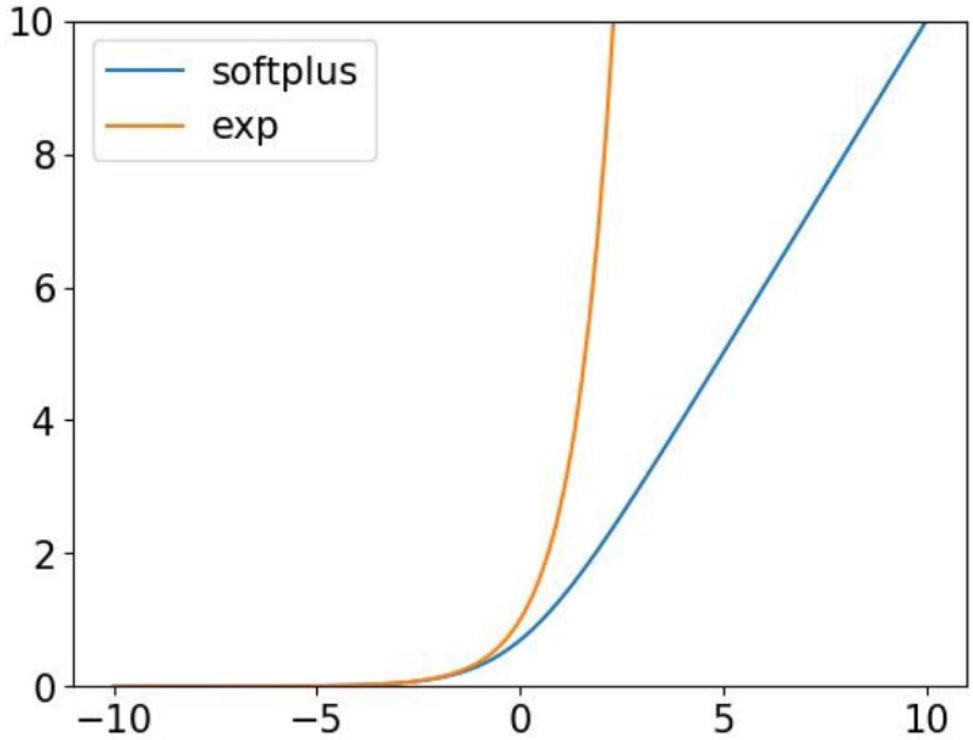
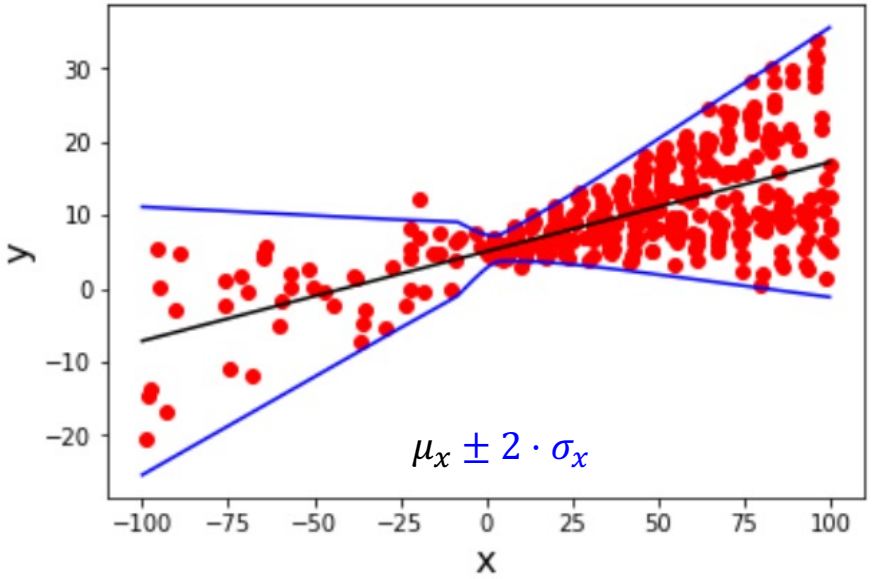
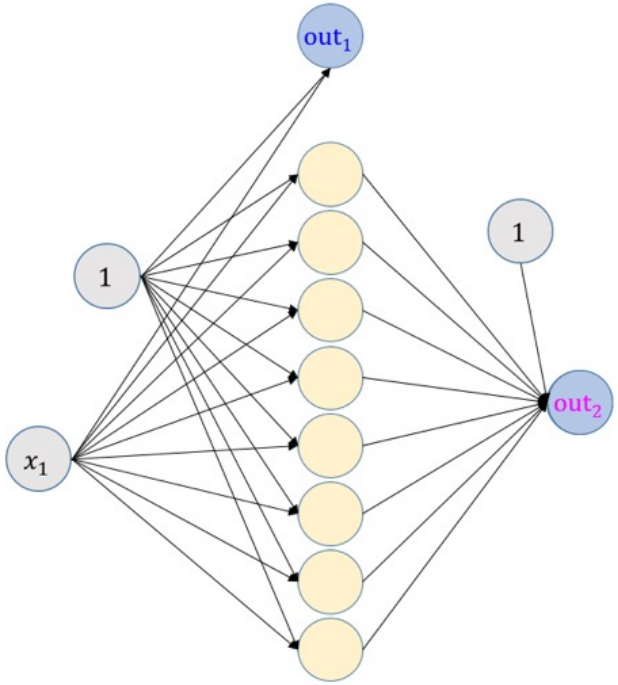


Figure 5.sp: The softplus function compared with the exponential function. Both functions map arbitrary values to positive values.

Fit a probabilistic regression with flexible non-constant variance

$$Y_{X_i} = (\mathrm{Y}|\mathrm{X}_i) \sim \mathrm{N}(\mu_{x_i}, \sigma_x^2)$$





$$\mu_{x_i} = \mathrm{out}_{1_i}$$

$$\sigma_{x_i} = e^{\mathrm{out}_{2_i}}$$

Minimize the negative log-likelihood (NLL):

$$\hat{\mathbf{w}}_{\mathrm{ML}} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}} - \frac{\left(y_i - \mu_{x_i}\right)^2}{2\sigma_{x_i}^2}\right)$$

gradient descent with NLL loss

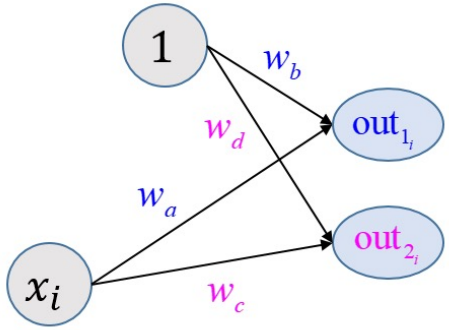
$$\hat{w}_1, \ \hat{w}_{.2}, \ \ldots, \ \hat{w}_{.27}$$

Note: we do not need to know the "ground truth for σ_x" – the likelihood does the job!

TFP Layers

TFP Lambda Layer

- Lambda Layer connects the output of the network with the distribution



The Lambda Layer

In [74]:
```
from tensorflow.keras.layers import Input
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam

def my_dist(params):
  return tfd.Normal(
      loc=params[:,0:1], #First for location
      scale=1e-3 + tf.math.softplus(0.05 * params[:,1:2])
  )

inputs = Input(shape=(1,))
h1  = Dense(10)(inputs)
# Here could be large network
params = Dense(2)(h1) #We have to parameters
dist = tfp.layers.DistributionLambda(my_dist)(params) #Connecting Network
model = Model(inputs=inputs, outputs=dist)
model.summary()
```

https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/12_TensorFlowDistributions_Demo.ipynb

Loss and predicted CPD

```
[ ]  1 def NLL(y, distr):
     2   return -distr.log_prob(y) #Just NLL, works for all distributions
     3
     4 model.compile(Adam(), loss=NLL) #F
```

```
  1 model = Model(inputs=inputs, outputs=dist)
  2 x = np.asarray([[1],[2.3]], dtype='float32')
  3 x.shape
  4 cpd = model(x) # Returns a CPD for the x
```

```
[ ]  1 cpd.mean() #The expectations at the positions
```

```
<tf.Tensor: shape=(2, 1), dtype=float32, numpy=
array([[-0.24419746],
       [-0.5616542 ]], dtype=float32)>
```

```
[ ]  1 cpd.stddev() #The spread at the positions
```

```
<tf.Tensor: shape=(2, 1), dtype=float32, numpy=
array([[0.71965677],
       [0.75376725]], dtype=float32)>
```

Model the CPD

Machen Sie Aufgabe 13_linreg_with_tfp

https://github.com/tensorchiefs/dl_course_2021/blob/master/notebooks/13_linreg_with_tfp.ipynb