

Machine Intelligence:: Deep Learning

Week 5

Beate Sick, Elvis Murina, Oliver Dürr

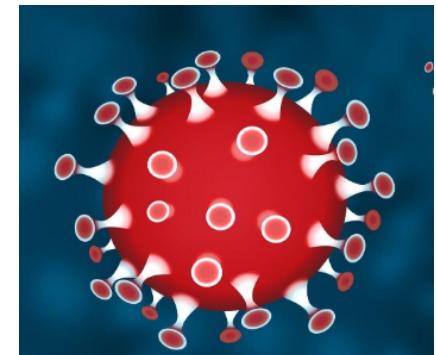
Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

Part I:

Winterthur, 24. March. 2020

Organizational Issues

- Projects
 - No Posters → 10-15 Minutes Talks about your projects
 - Please register:
 - Support for Projects
 - Please write E-Mail (all three of us) with topic and questions
 - beate.sick@zhaw.ch
 - oliver.duerr@gmail.com
 - elvismurina@hotmail.com
 - If necessary we will organize a call
- Lectures
 - Online using zoom (for the time being)
 - Try <https://zoom.us/test> before the lecture
 - Please mute your mic
 - Please do ask question
 - If something goes wrong, blame corona not us!

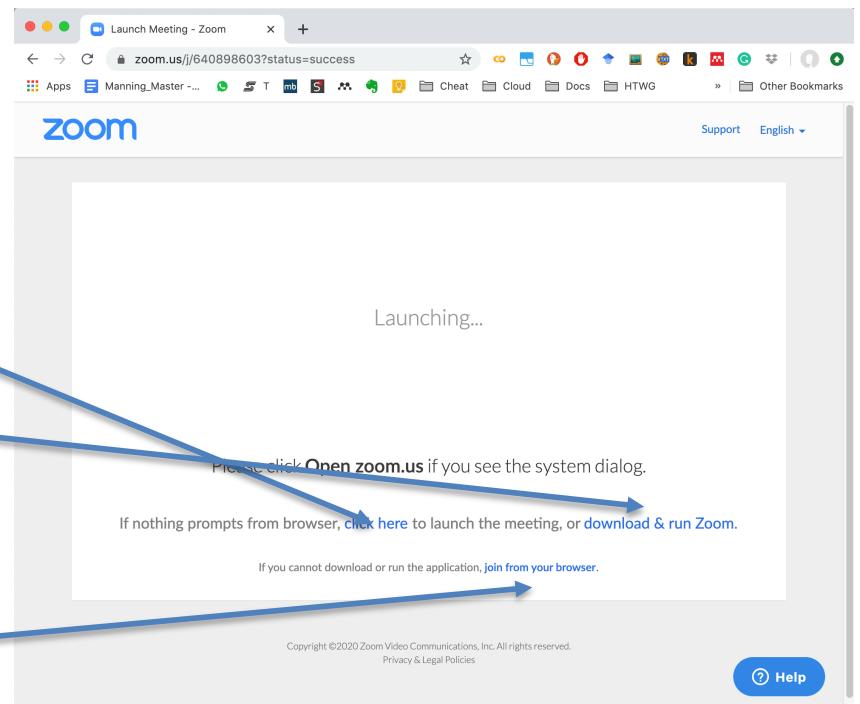


Zoom first steps after clicking on link

You have zoom installed

You want to install zoom

You don't want or cannot
install zoom, run it in your
browser



We recommend that you install the zoom application / program. Also on mobile / tablet.

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/18VFrPbKq3YSOg8Ebc1q1wGgkfgaWI7IkCCIGEDGj6Q/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)

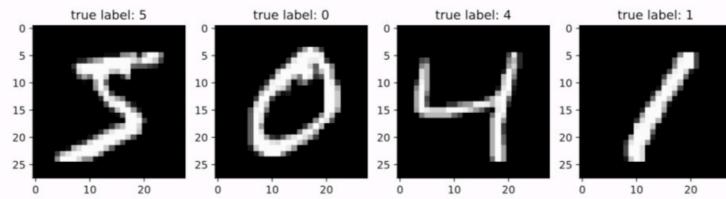
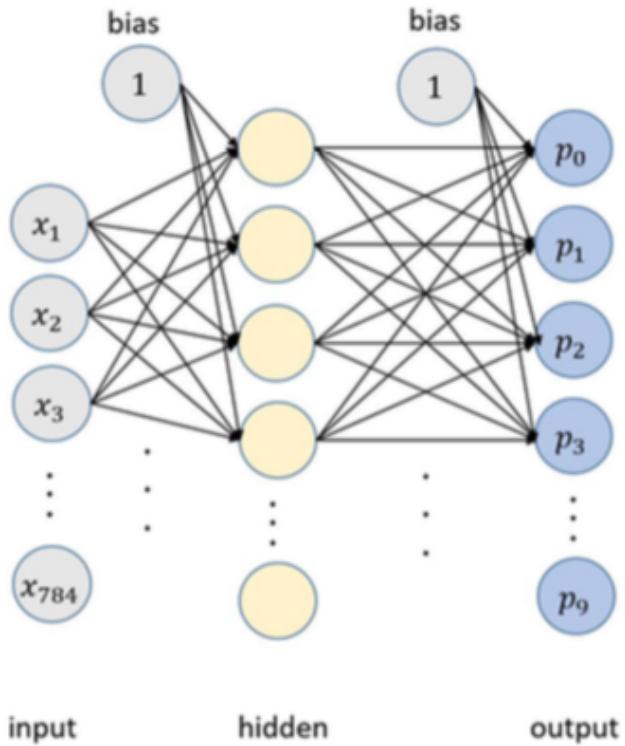


Figure 2.11 The first four digits of the MNIST data set—the standard data set used for benchmarking NN for images classification

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

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

Excercise



Task 1:

- * Calculate the expected 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

Loss function for regression

Regression

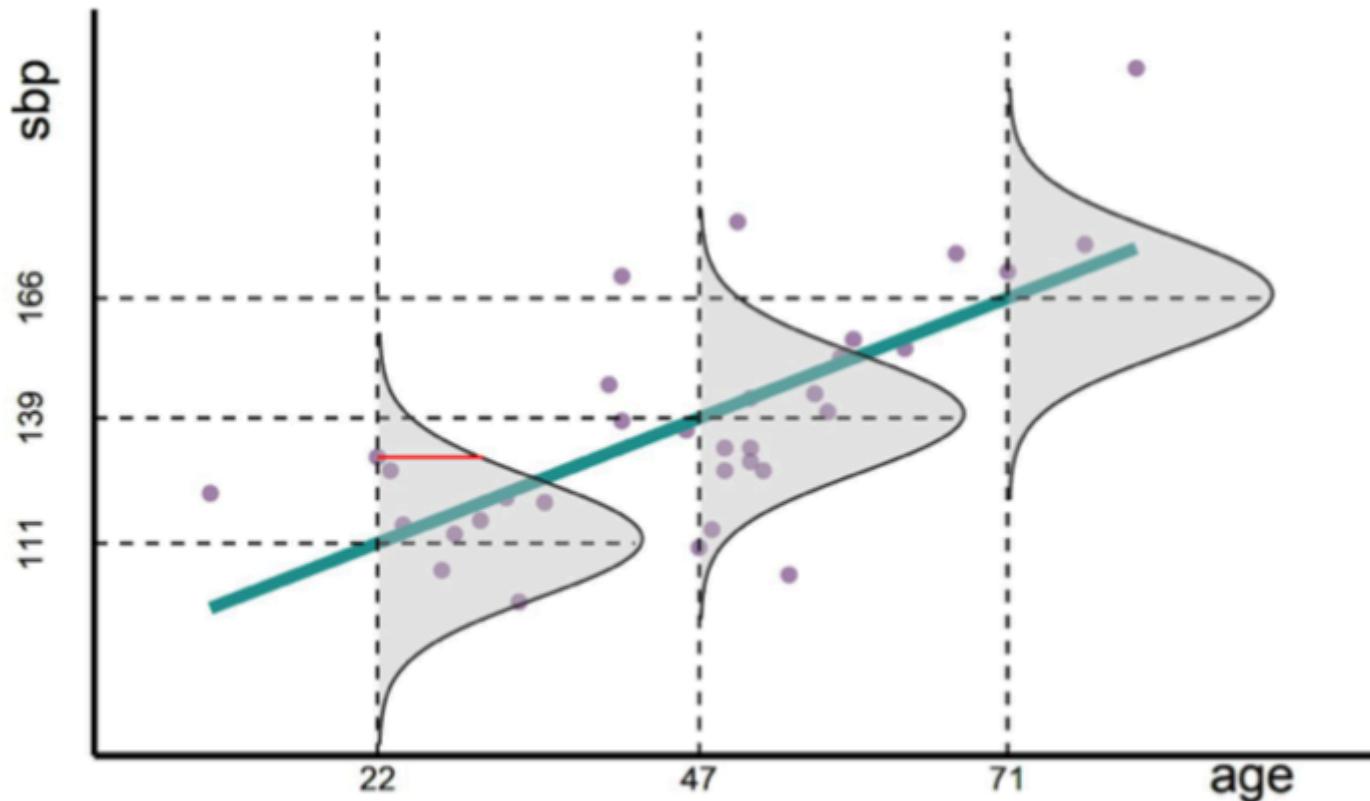
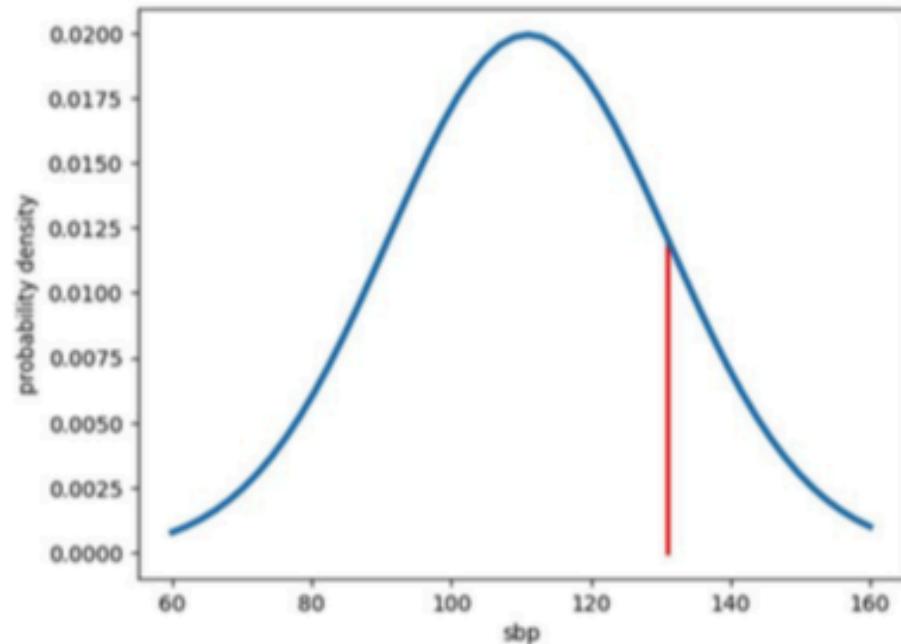


Figure 4.12 Scatterplot and regression model for the blood pressure example: The dots are the measured data points, the straight line is the linear model, the bell-shaped curves are the conditional probability distributions of the outcome Y conditioned on the observed value x. The length of the red bar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

Zoom for one example



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

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:

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

MSE-Loss

$$w = \operatorname{argmax}_w \left\{ \prod_{i=1}^n f(y_i | x_i, w) \right\} \Rightarrow$$

$$w = \operatorname{argmin}_w \left\{ \sum_{i=1}^n -\log f(y_i | x_i, w) \right\} \Rightarrow$$

$$w = \operatorname{argmin}_w \left\{ \sum_{i=1}^n -\log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(y_i - \mu_{x_i})^2}{2\sigma^2}} \right) \right\} \Rightarrow$$

$$w = \operatorname{argmin}_w \left\{ \sum_{i=1}^n -\log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) + \frac{(\mu_{x_i} - y_i)^2}{2\sigma^2} \right\} \quad (5) \Rightarrow$$

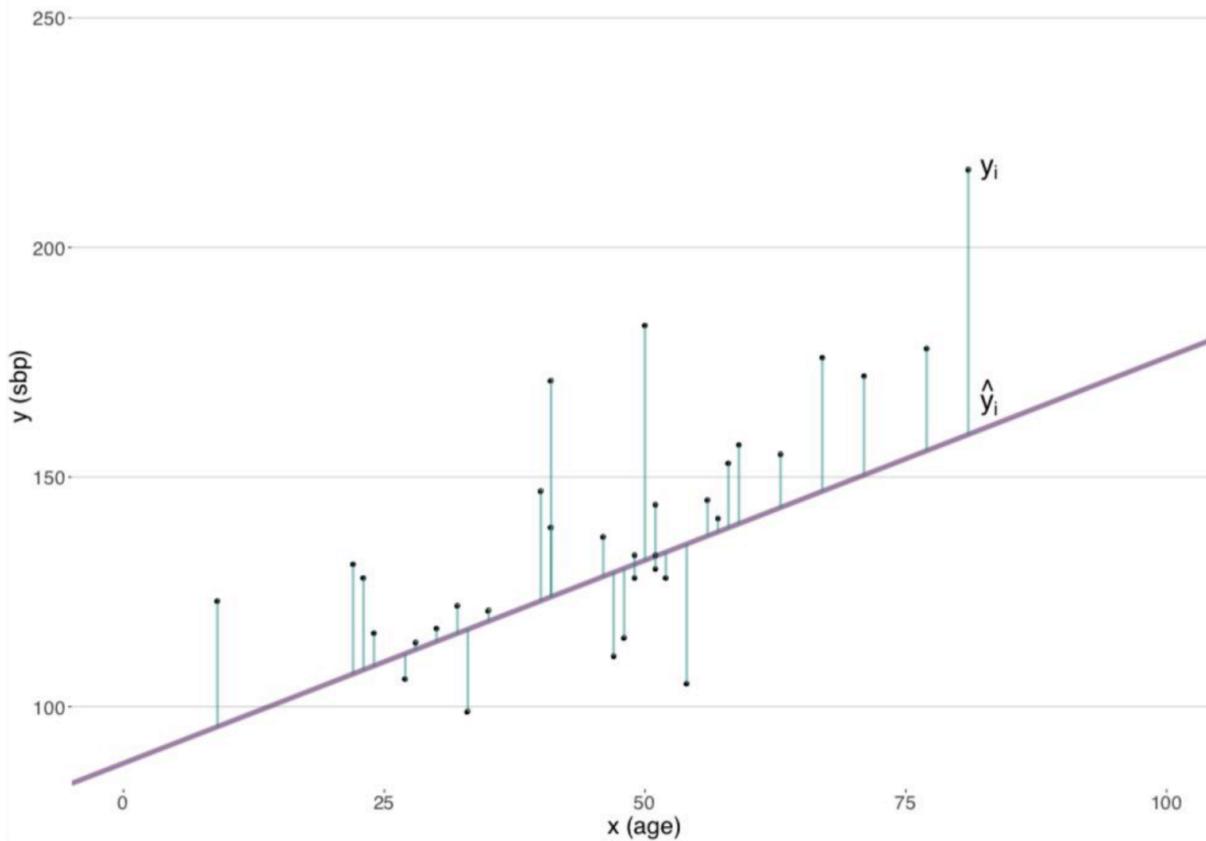
$$w = \operatorname{argmin}_w \left\{ \sum_{i=1}^n + \frac{(\mu_{x_i} - y_i)^2}{2\sigma^2} \right\} \Rightarrow$$

$$w = \operatorname{argmin}_w \left\{ \frac{1}{2\sigma^2} \sum_{i=1}^n (\mu_{x_i} - y_i)^2 \right\} \Rightarrow$$

$$(a, b) = \operatorname{argmin}_w \left\{ \frac{1}{n} \sum_{i=1}^n (\mu_{x_i} - y_i)^2 \right\}$$

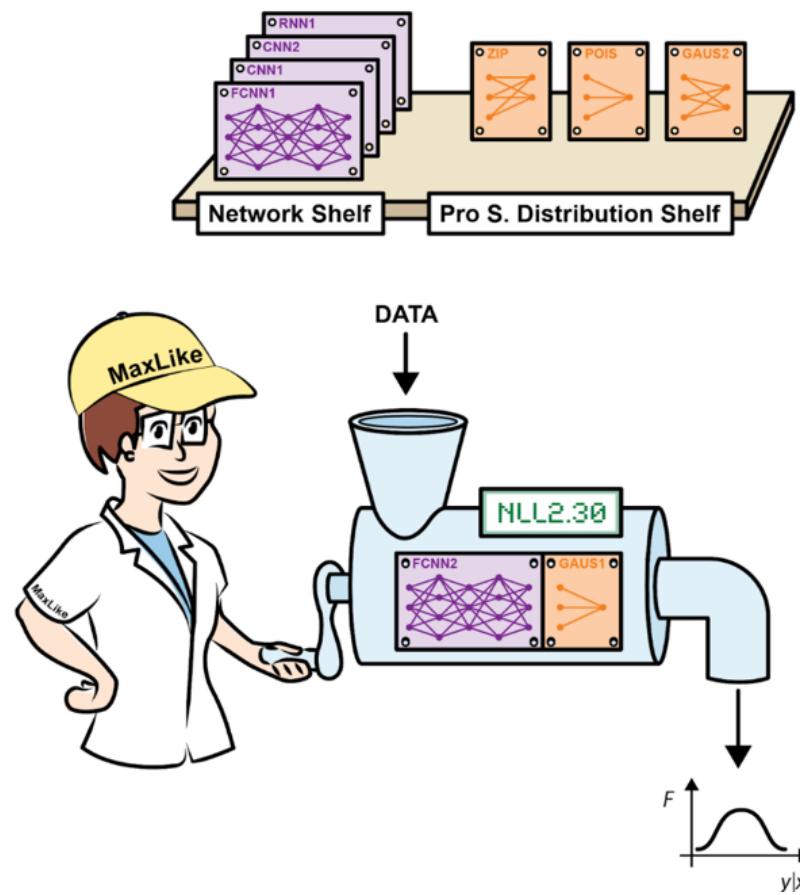
Here we assume constant spread

Side note: how to get optimal σ



$$\hat{\sigma} = \sqrt{\frac{1}{n-2} \sum_i (y_i - \hat{y})^2}$$

Probabilistic models

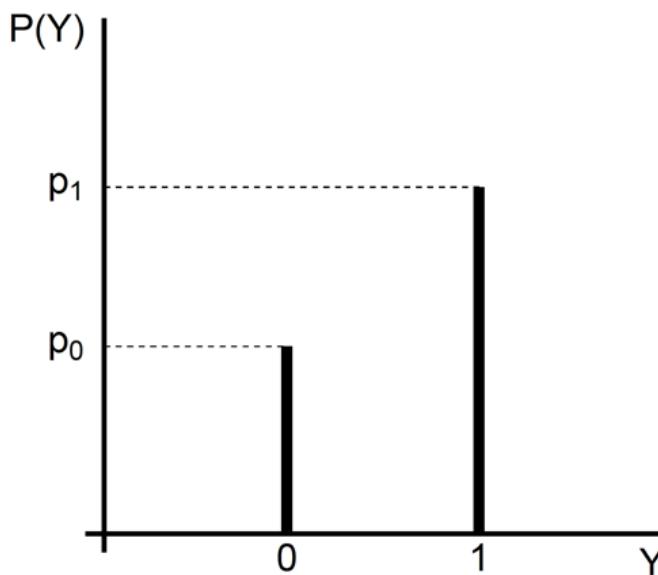
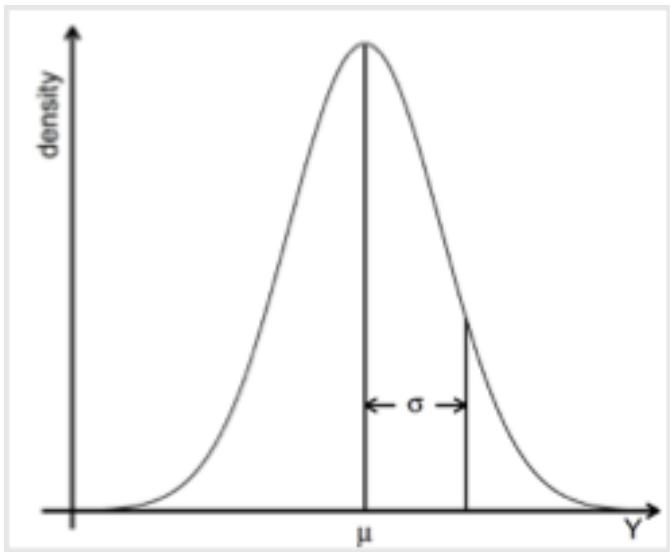


Outline

- What is a probabilistic model?
→ predicts whole **conditional probability distribution** (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 its uniqueness
- Examples using count data

Recap: probability distributions

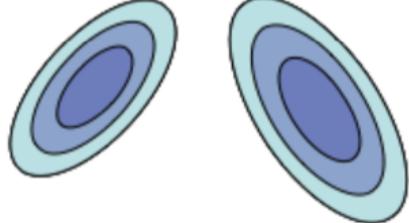
- Describe a random variable here called y
- Characterized by probability density (pdf) for continuous or by the probability mass discrete distributions



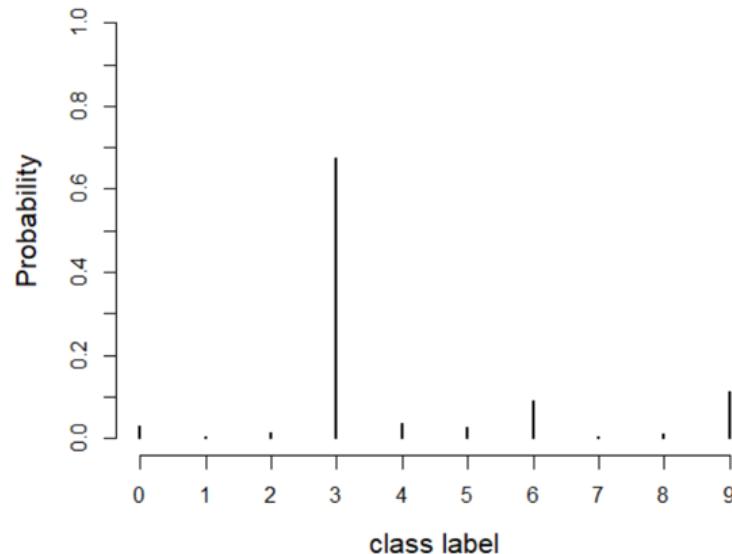
- Few parameters describe the distribution
 - E.g. μ and σ for a Gaussian $p(y; \mu, \sigma)$ (sometimes $f(y; \mu, \sigma)$ to stress density)
 - E.g. p_1 for Bernoulli

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 Σ for a Gaussian $f(\vec{y}; \vec{\mu}, \Sigma)$
 - E.g. $(p_0, p_1 \dots, p_9)$ for categorical* with values $\vec{y} = "0", "1", \dots, "9"$

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

TensorFlow Distributions (Part I)

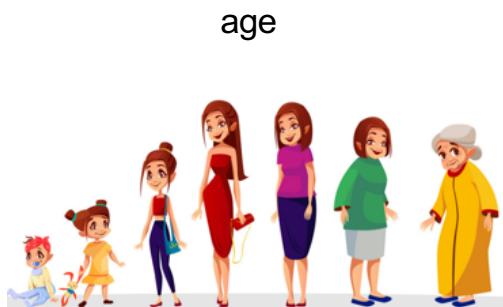
TensorFlow Probability (TFP) is nice add-on to Keras
Nice abstraction, different distributions all have the same API.

Methods for Tensorflow distributions	Description	Numerical result when calling the method on <code>dist = tfd.Normal(loc=1.0, scale=0.1)</code>
<code>sample(n)</code>	Samples n numbers from the distribution.	<code>dist.sample(3).numpy()</code> <code>array([1.0985107, 1.0344477, 0.9714464], dtype=float32)</code> Note that these are random numbers
<code>prob(value)</code>	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)	<code>dist.prob((0,1,2)).numpy()</code> <code>array([7.694609e-22, 3.989423e+00, 7.694609e-22], dtype=float32)</code>
<code>log_prob(value)</code>	Returns the log-likelihood or log-probability for the values (tensor)	<code>dist.log_prob((0,1,2)).numpy()</code> <code>array([-48.616352, 1.3836466, -48.616352], dtype=float32)</code>
<code>cdf(value)</code>	Returns the cumulative distribution function (CDF) that this is a sum or the integral, up to the given values (tensor)	<code>dist.cdf((0,1,2)).numpy()</code> <code>array([7.619854e-24, 5.000000e-01, 1.000000e+00], dtype=float32)</code>
<code>mean()</code>	Returns the mean of the distribution	<code>dist.mean().numpy()</code> 1.0
<code>stddev()</code>	Returns the standard deviation of the distribution	<code>dist.stddev().numpy()</code> 0.1

Demo time: TensorFlowDistributions: https://github.com/ioskn/ml1_htwg/blob/master/dl_prob_1/TensorFlowDistributions.ipynb

What is a probabilistic model?

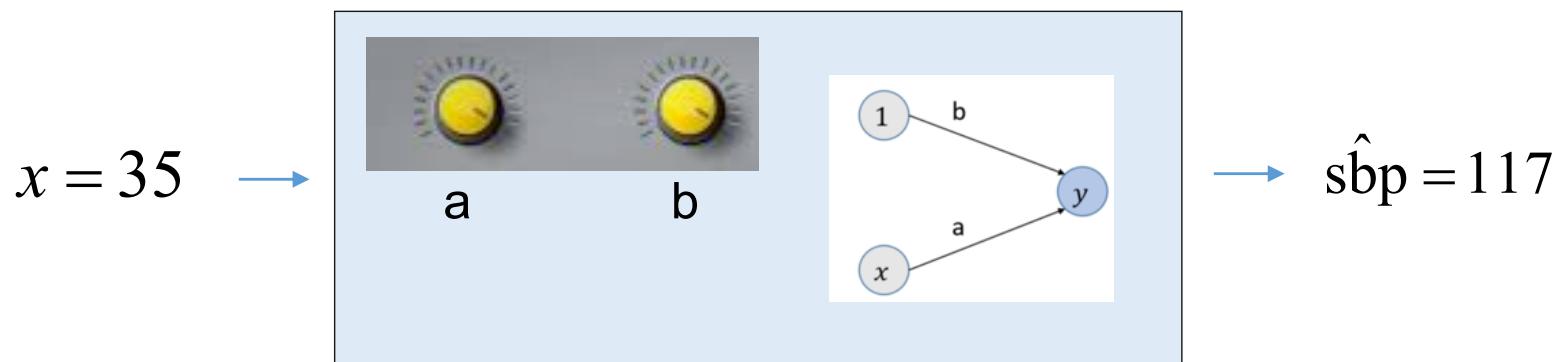
Simple regression via a NN: no probabilistic model in mind



Input x

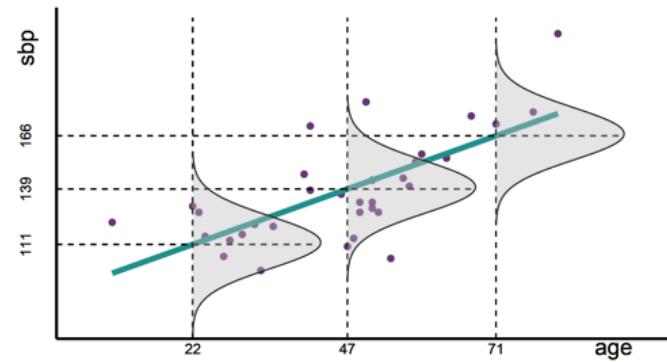
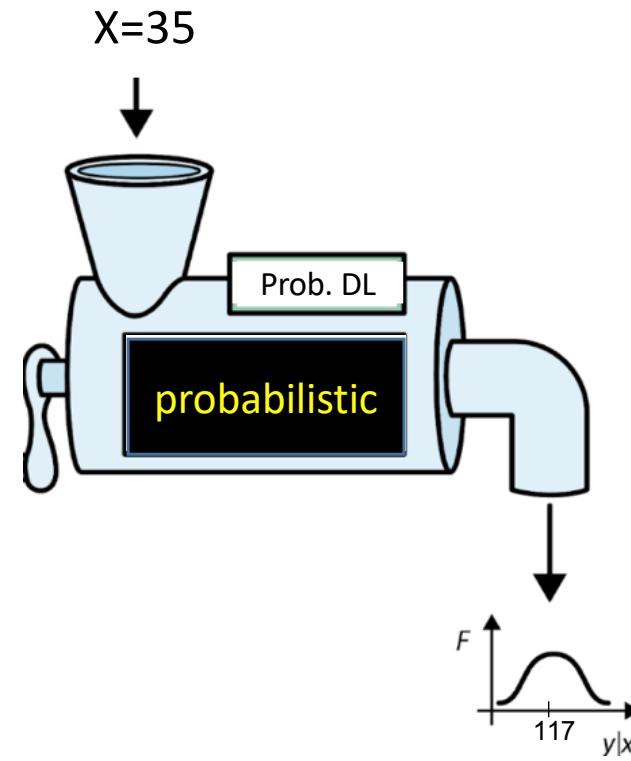
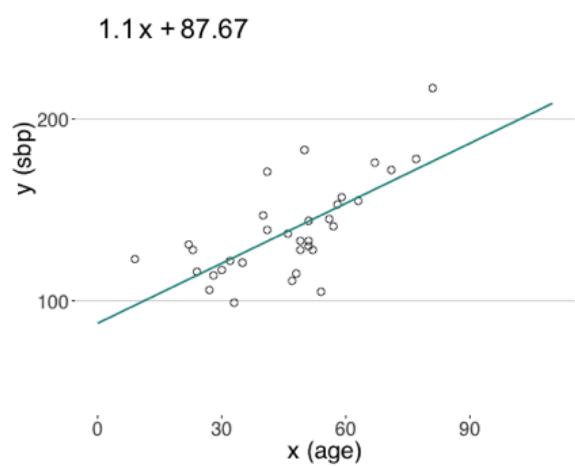
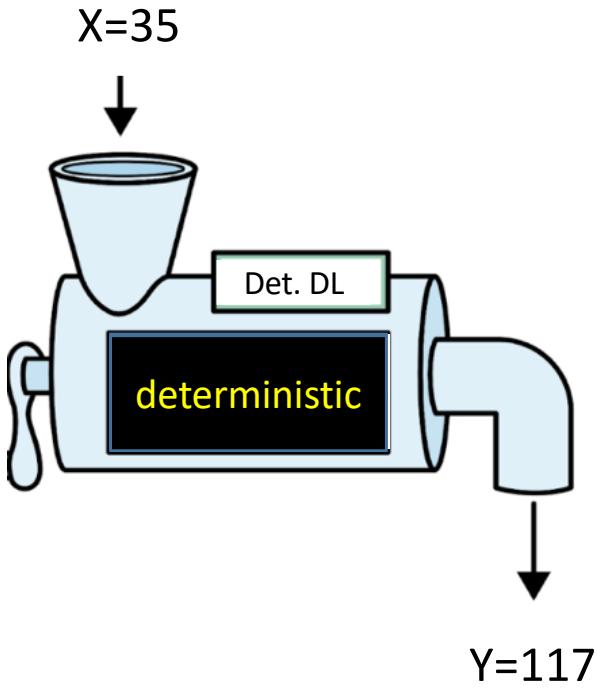


Output y



One input x (age) \rightarrow one predicted outcome (sbp)

Traditional versus probabilistic regression DL models



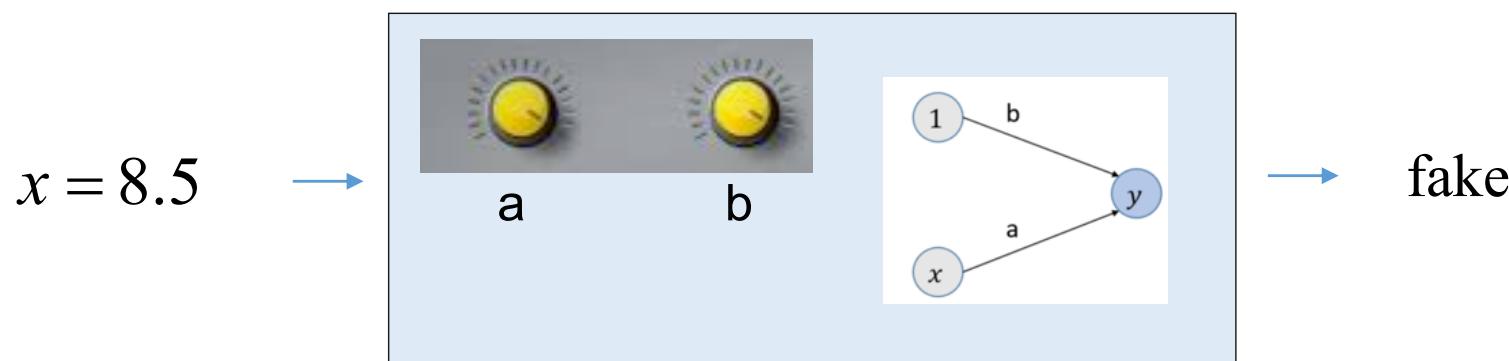
Describes the spread of the data

Binary classification: no probabilistic model in mind



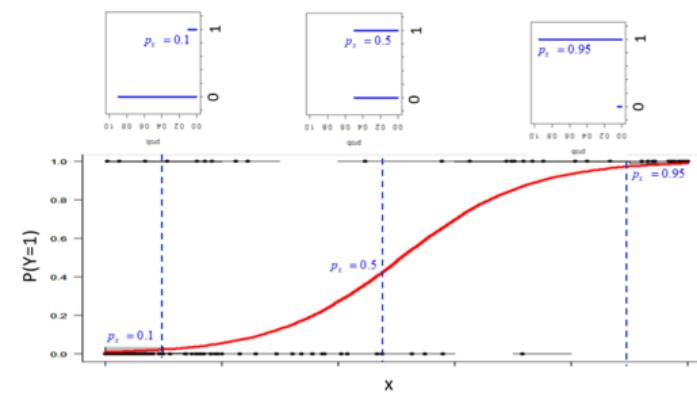
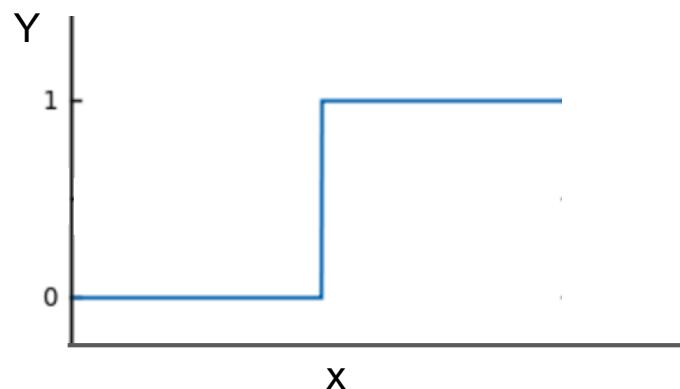
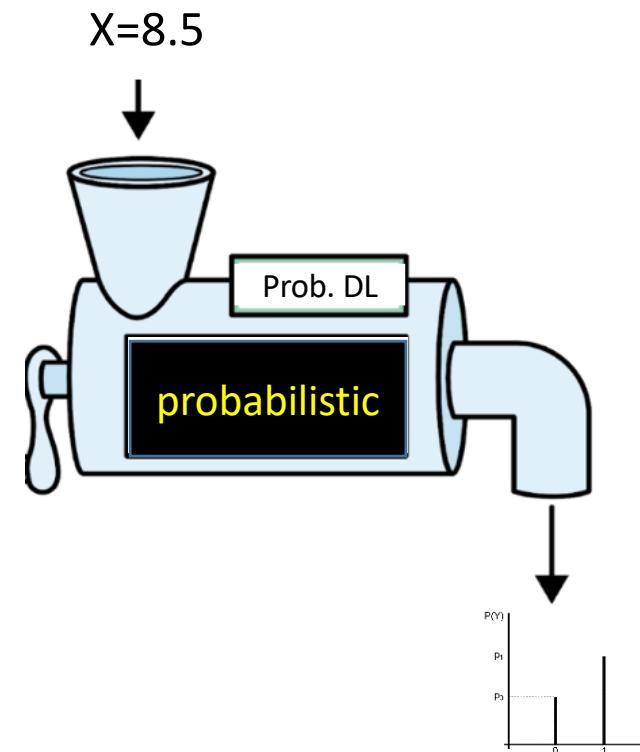
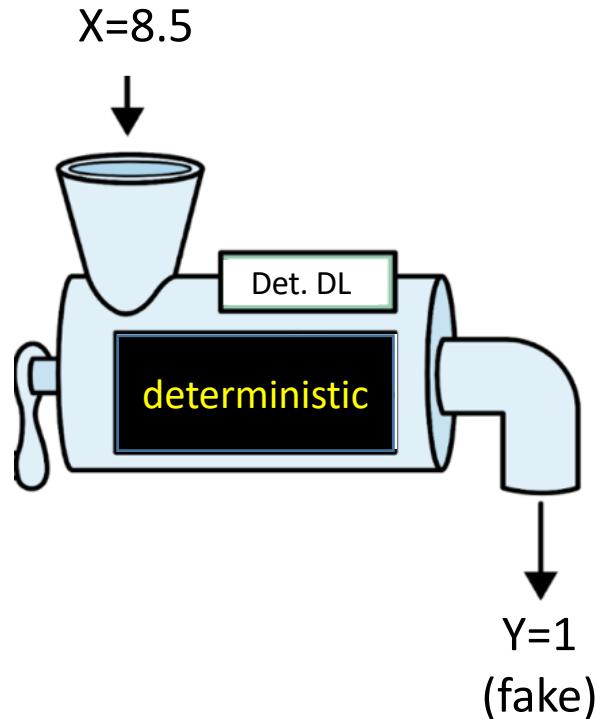
Fake or real?

Quantify transparency



One input $x \rightarrow$ one predicted outcome

Deterministic versus probabilistic classification DL models



spread measure for uncertainty

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:

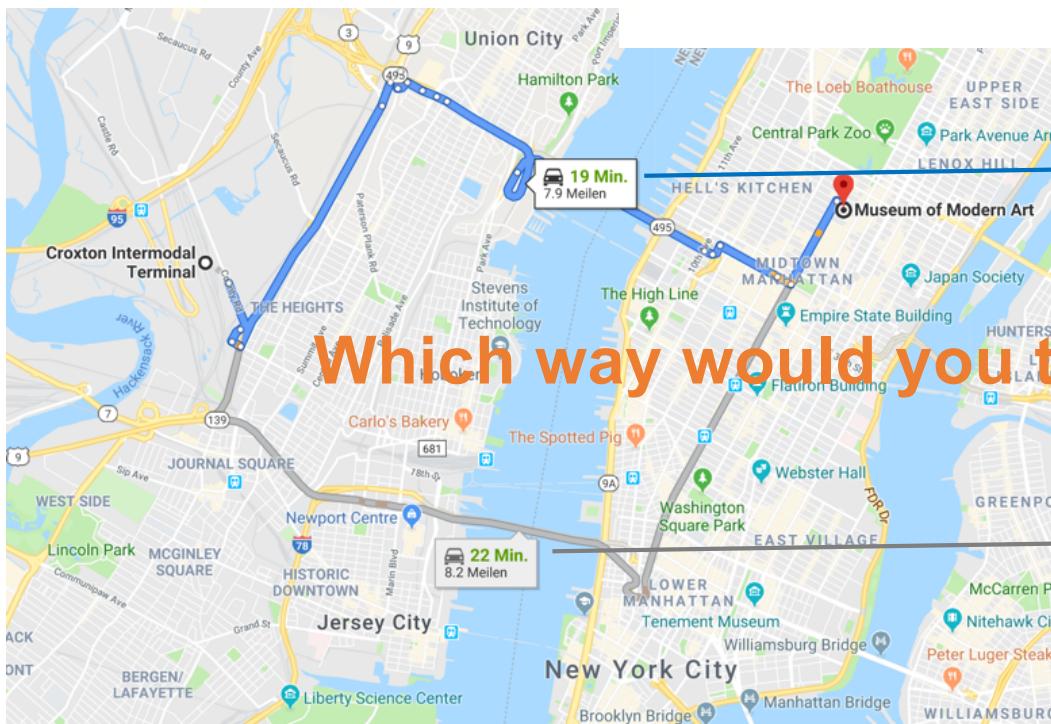
- We often want to optimize expected costs which requires CPD for computing.
- They come for free in most DL problems
- Choosing the right probabilistic model enhances existing DL architectures
 - Parallel Wavenet (using mixtures of logistics distributions as output)
 - PixelCNN++ (using mixtures of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

Probabilistic travel time prediction (cost)

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

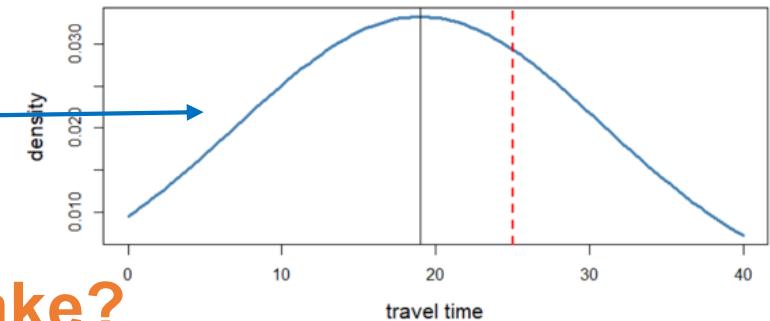


Let's use my probabilistic travel time gadget!

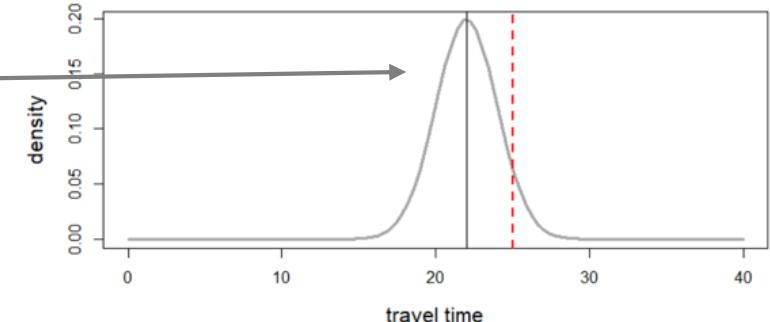


Which way would you take?

Chance to get tip: 69%



Chance to get tip: 93%

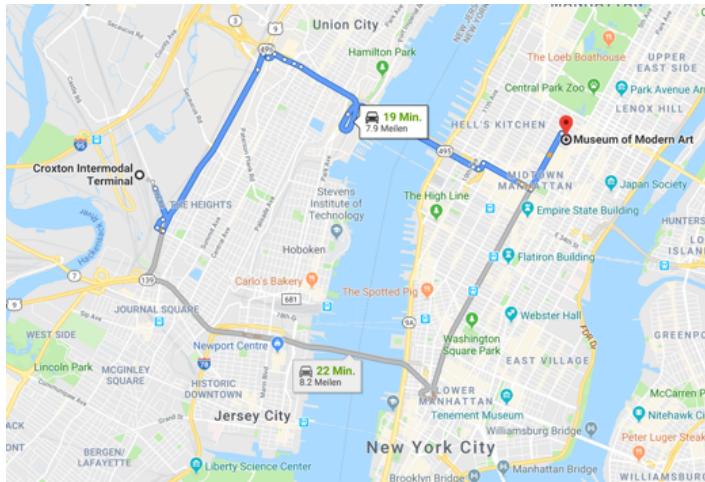


Conditional probability distribution

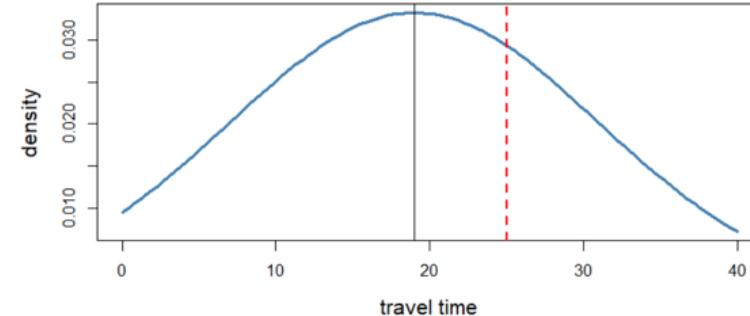
- Probability distribution now depends (“*is conditioned on*”) on an external value x

Example

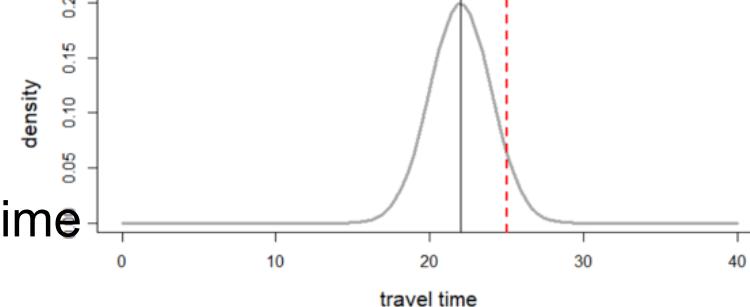
$$p(y) \rightarrow p(y|x)$$



$$p(y|x = \text{"Route 1"})$$



$$p(y|x = \text{"Route 2"})$$

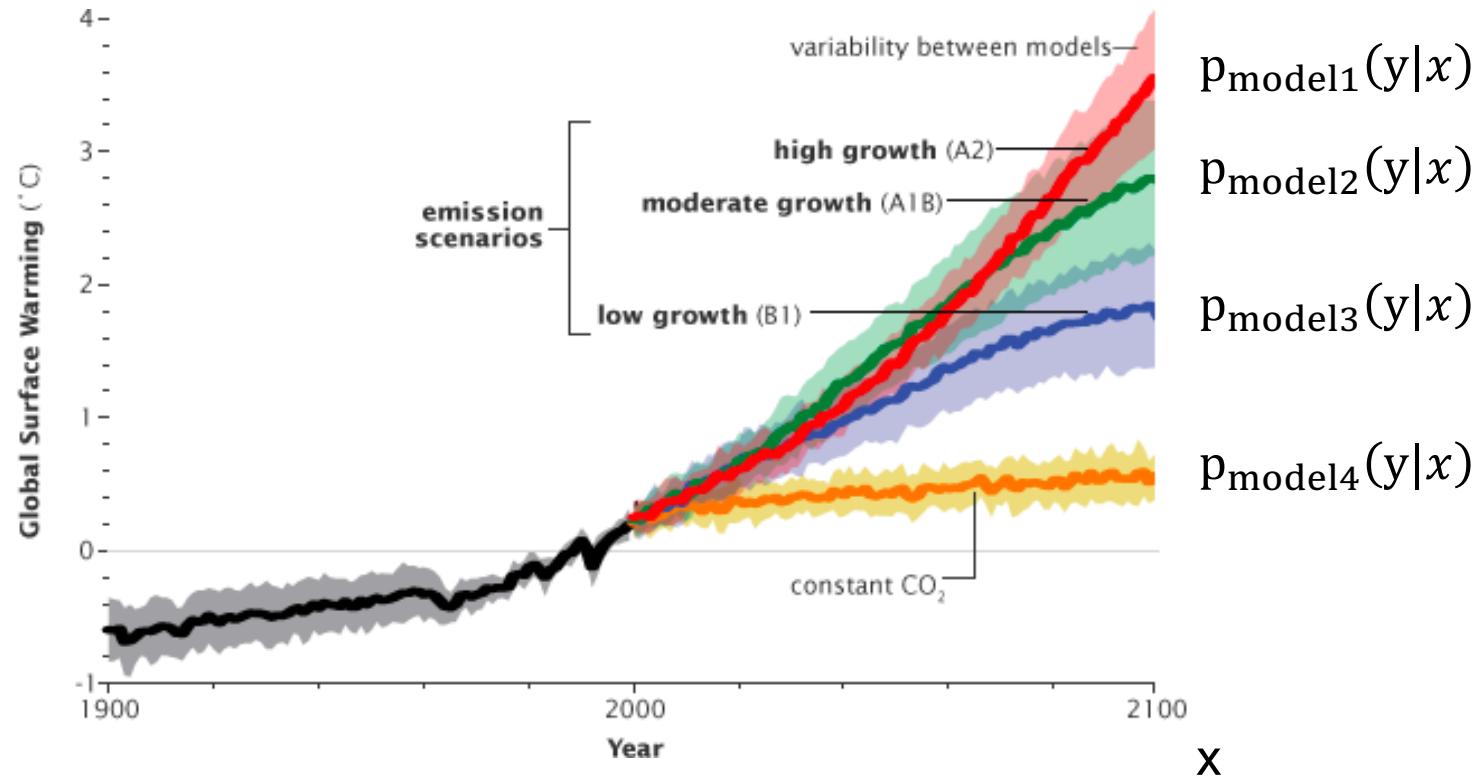


$$p(y|x)$$

- 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)



$$p(y|x)$$

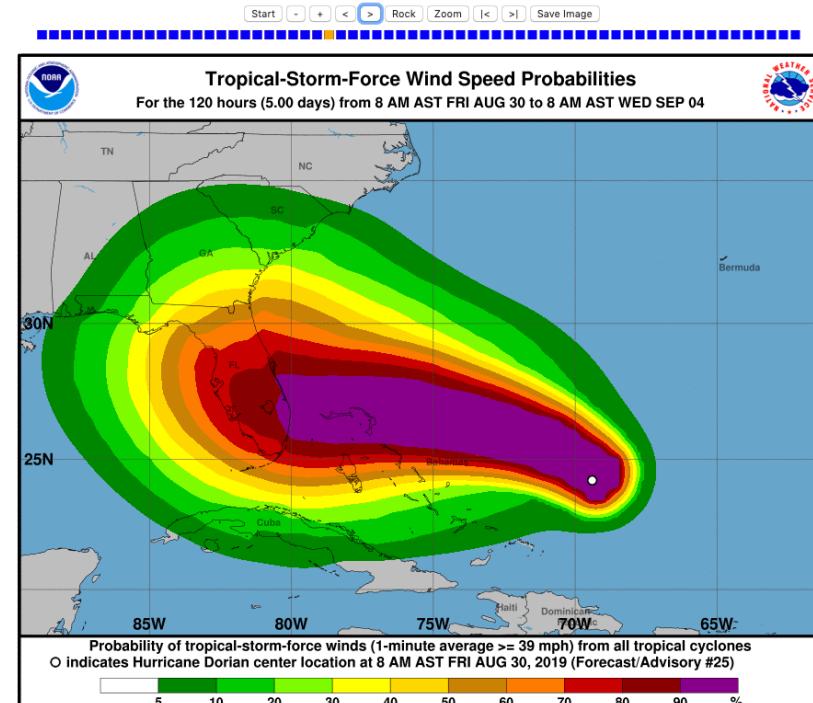
- y is a 1D continuous variable (of the increase of the global surface temperature)
- x is 1D continuous time

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

Examples of probabilistic models: path of hurricane Dorian (2019)



Storm at least $> 74\text{mph}$



Storm at least $> 39\text{mph}$

$p(y|x)$ probability of being hit by storm (in next 5 days)

- y is a 2D continuous variable (space)
- x category of storm

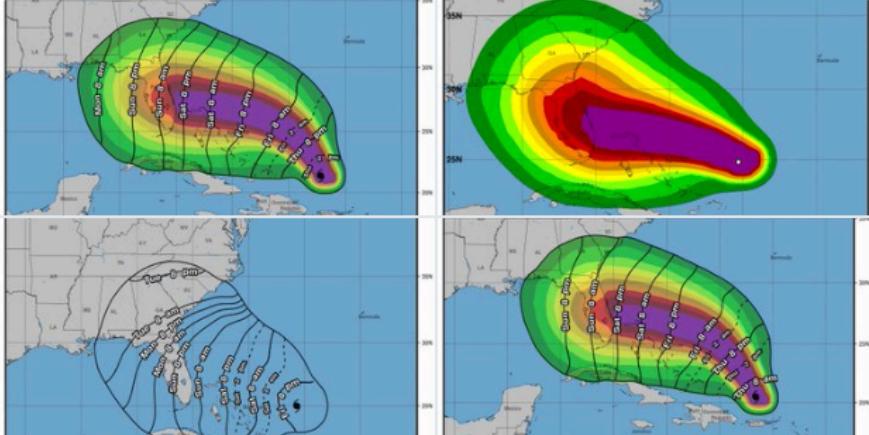
For models created at different time points and additional models

https://www.nhc.noaa.gov/archive/2019/DORIAN_graphics.php?product=wind_probs_64

Fun fact: Not everybody understands them...

 Donald J. Trump ✅
@realDonaldTrump

Just as I said, Alabama was originally projected to be hit.
The Fake News denies it!



47.3K 9:14 PM - Sep 5, 2019

43.5K people are talking about this

<https://www.r-bloggers.com/was-trump-confused-by-rainbow-color-map/>

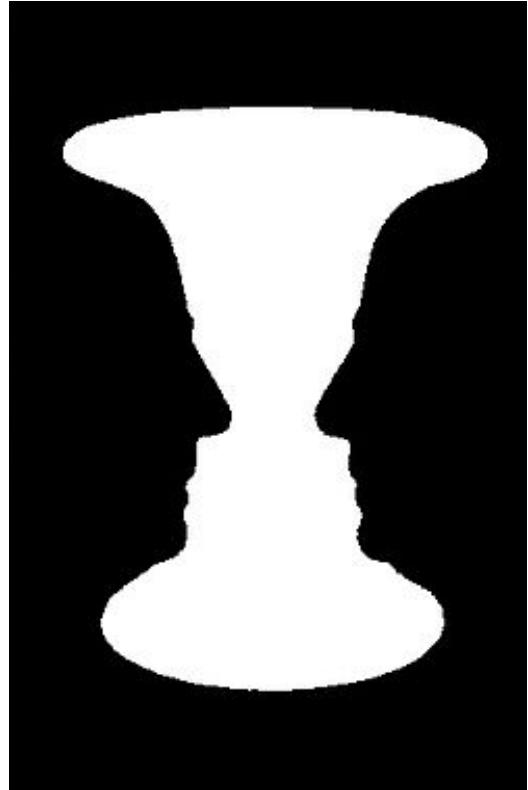


<https://twitter.com/hashtag/sharpiegate>
https://www.reddit.com/r/misc/comments/d0fxlc/donald_trump_hurricane_dorian_sharpiegate/

Blog post:
Colormap (green for 0.1 quite saturated)
and wrong magnitude

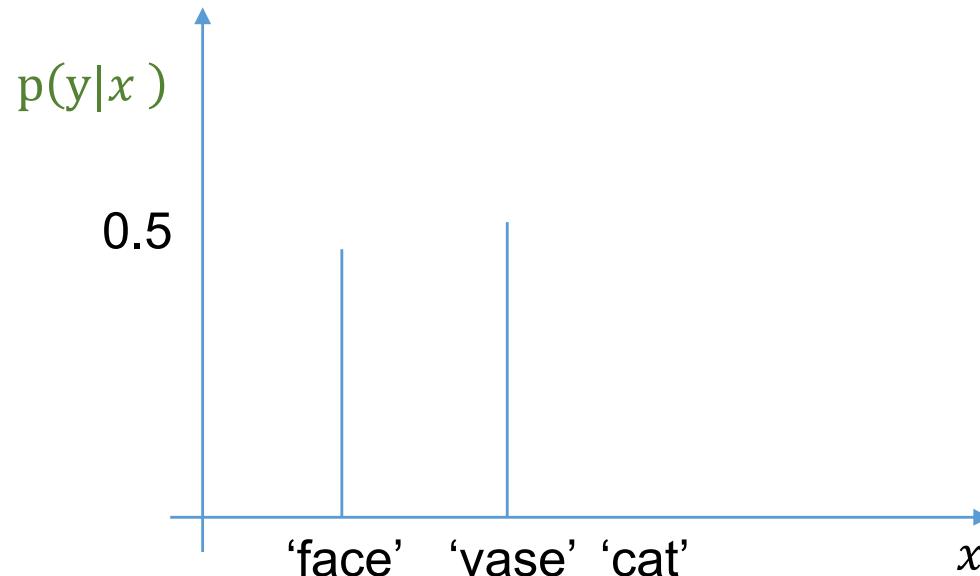
... some of them are even geniuses 'In my great and unmatched wisdom'

Examples of probabilistic models: image recognition



$p(y|x)$

- y is a categorical discrete variable $y = \text{'face', 'vase', 'cat'}$
- 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

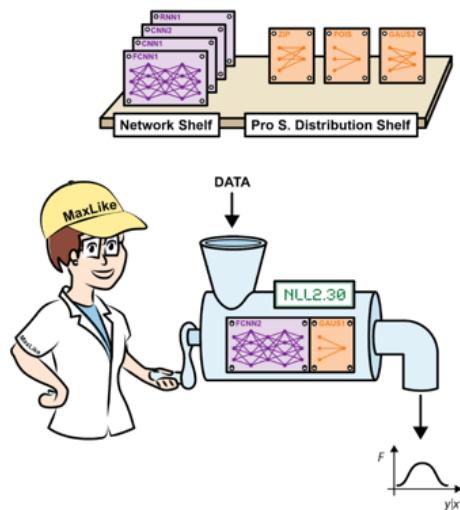
- In machine learning, we learn from N examples $D = \{x^{(i)}, y^{(i)}\}$ with $i = 1, \dots, N$. CPD:

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

- After training all information is in the weights w

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

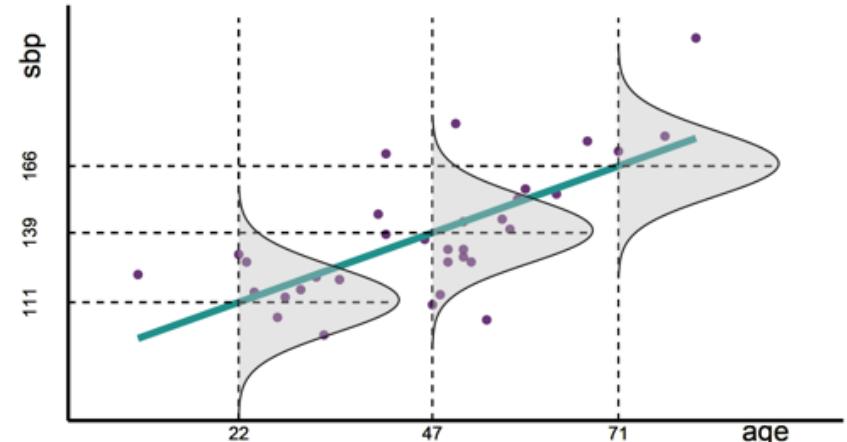
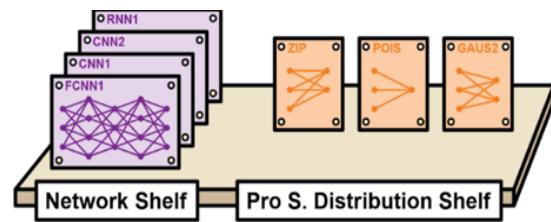
- Deep Learning recipe for probabilistic models



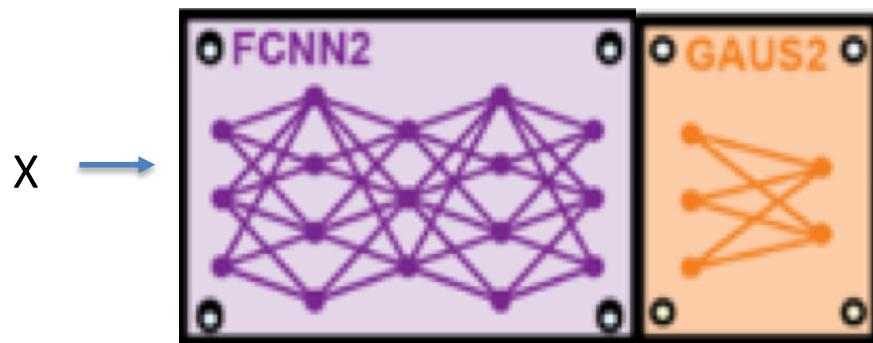
The network determines the parameters of a CPD given an x .

Regression output as CPD

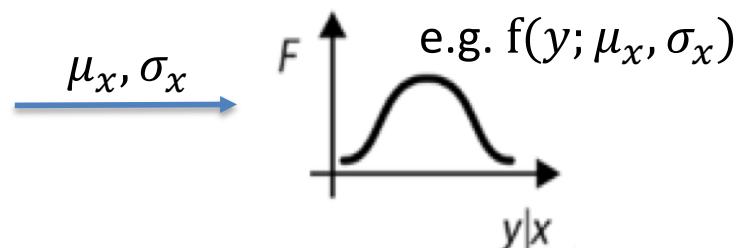
- A trained network outputs for each x the parameters of a distribution.
- E.g. network output μ_x, σ_x for a given x .



Input: Images, numbers,...

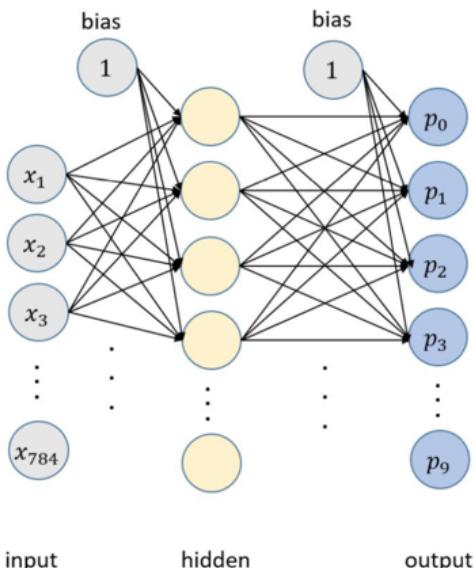


Output: Parameters of output distribution



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
 - $y \sim \text{Bern}(p_1)$
- 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, \dots, p_k **for a categorical CDF** with k categories.



The output values are parameters for a categorical probability distribution.

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

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

$$\hat{w} = \operatorname{argmin} \sum_{i=1}^N -\log(p(y_i | x_i, w))$$

Two practical ways of modeling and fitting CPD

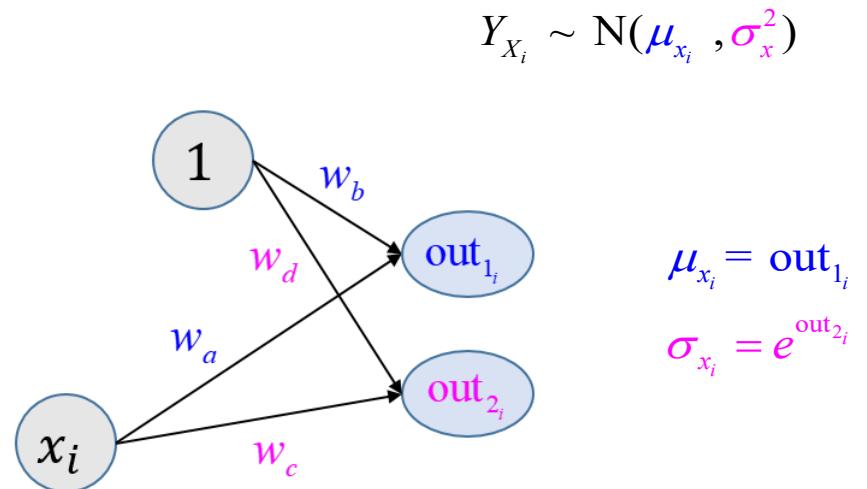
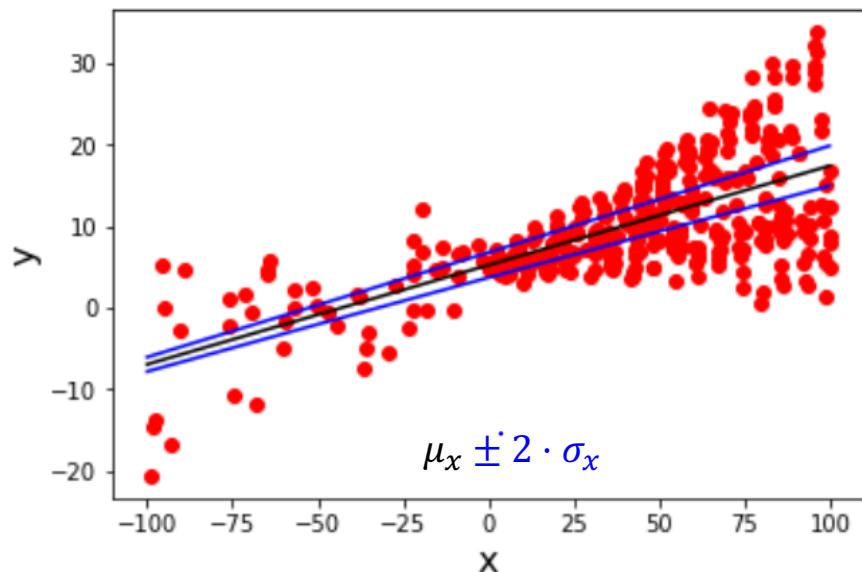
Approach 1 (explicit loss function)

- 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 CPD
- This approach can be done with the TensorFlow Probability (TFP) Framework

Fit a probabilistic regression with non-constant variance



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

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

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

Minimize the negative log-likelihood (NLL):

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

gradient descent with NLL loss

$$\begin{array}{cc} \hat{w}_a & \hat{w}_b \\ \hat{w}_c & \hat{w}_d \end{array}$$

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)

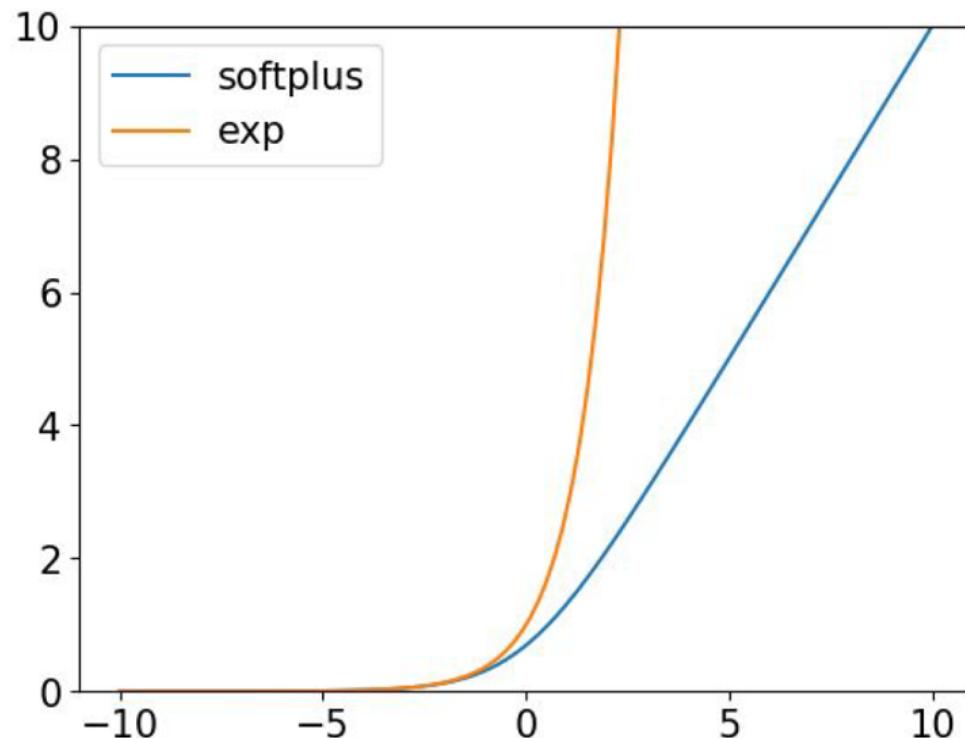
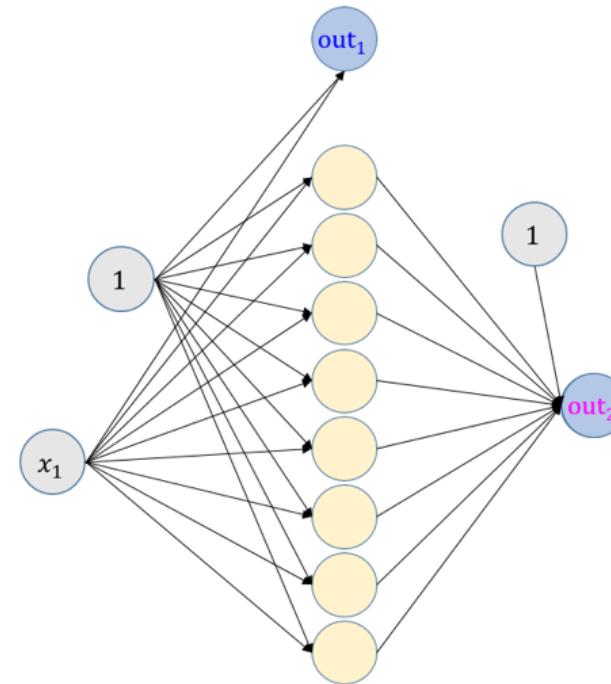
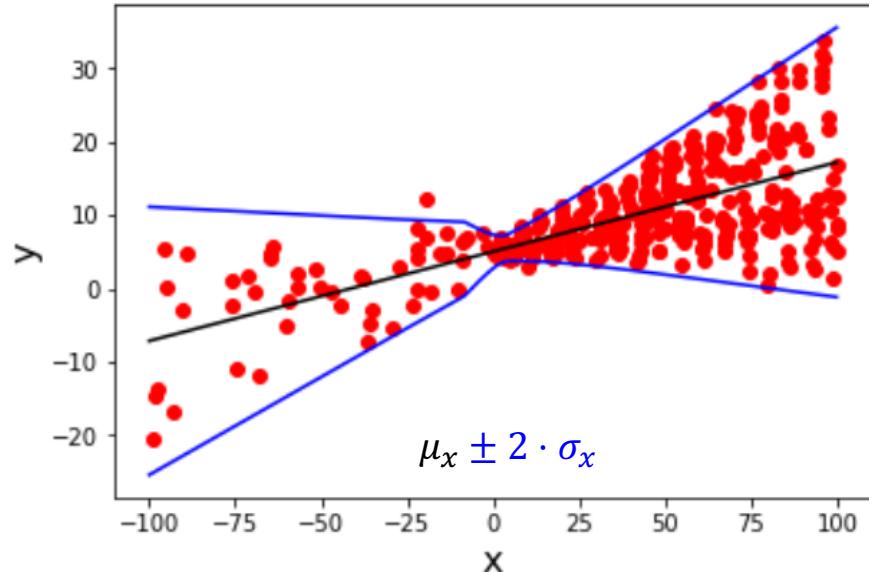


Figure 5.sp: The softplus function compared with the exponential function. Both functions map arbitrary values to positive values.

Fit a probabilistic regression with flexible non-constant variance

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



$$\mu_{x_i} = out_{1i}$$

$$\sigma_{x_i} = e^{out_{2i}}$$

Minimize the negative log-likelihood (NLL):

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

gradient descent with NLL loss

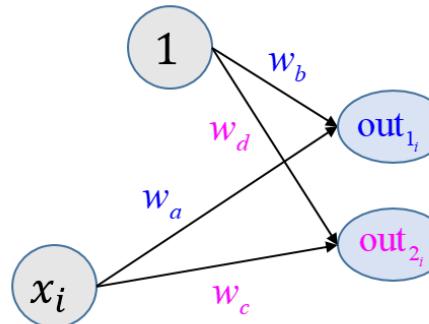
$$\hat{w}_1, \hat{w}_{.2}, \dots, \hat{w}_{.27}$$

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()
```

Loss

```
In [75]: def NLL(y, distr):
    return -distr.log_prob(y) #Just NLL, works for all distributions
```

```
model.compile(Adam(), loss=NLL) #F
```

```
In [77]: x = np.asarray([[1],[2.3]], dtype='float32')
x.shape
model.predict(x)
```

```
Out[77]: array([[0.5341634],
                 [0.9440286]], dtype=float32)
```

Training can be done the usual way by calling model.train

Approach 2 Model the CPD



Machen Sie Aufgabe 13_linreg_with_tfp

https://github.com/tensorchiefs/dl_course_2020/blob/master/notebooks/13_linreg_with_tfp.ipynb