Machine Intelligence:: Deep Learning
Week 5

Beate Sick, Oliver Dürr, Jonas Brändli

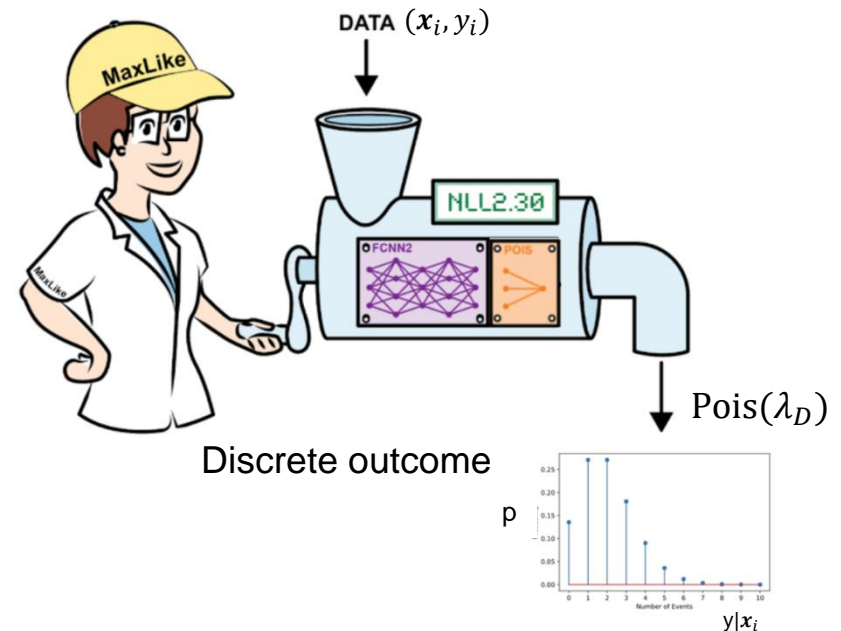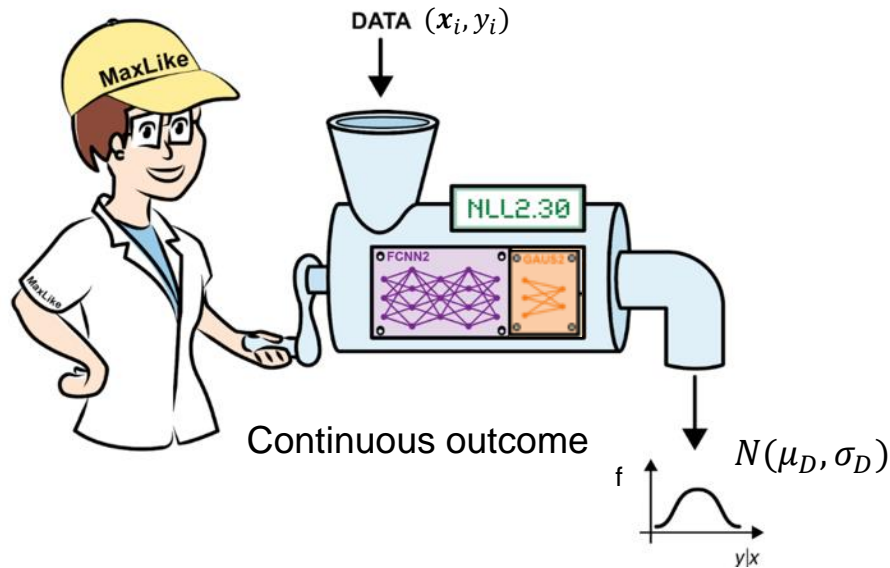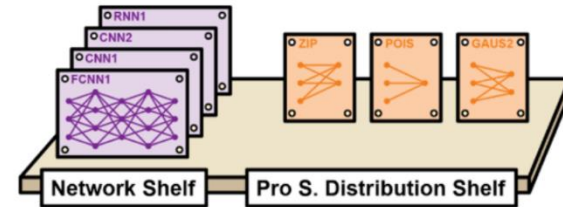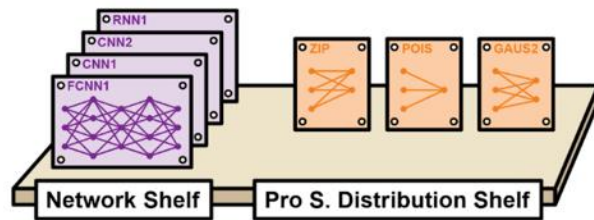Probabilistic Prediction Models

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

Probabilistic DL Models for continuous or discrete outcomes

Continuous outcome
$N(\mu_D, \sigma_D)$

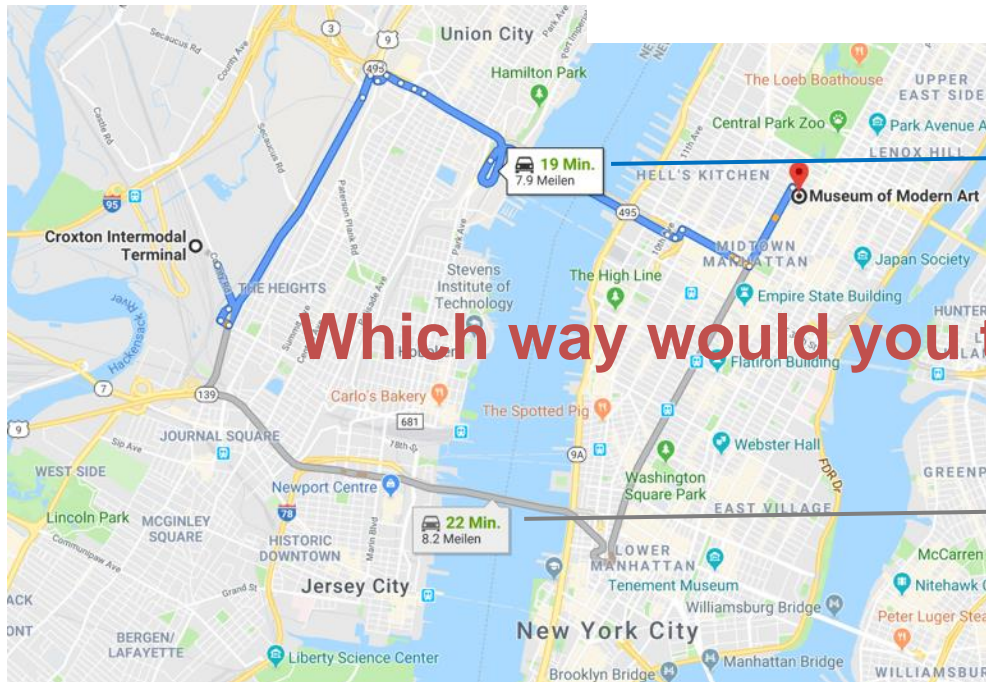Discrete outcome
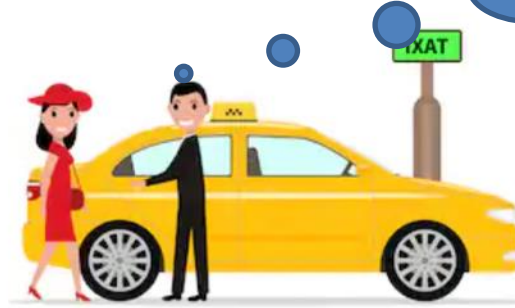$\text{Pois}(\lambda_D)$

Probabilistic prediction models predict conditional probability distributions (CPDs), rather than only a point estimate.
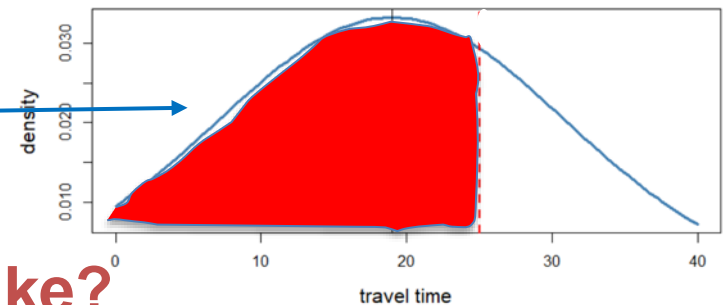
What is a probabilistic model good for?

Probabilistic travel time prediction (cost)

5
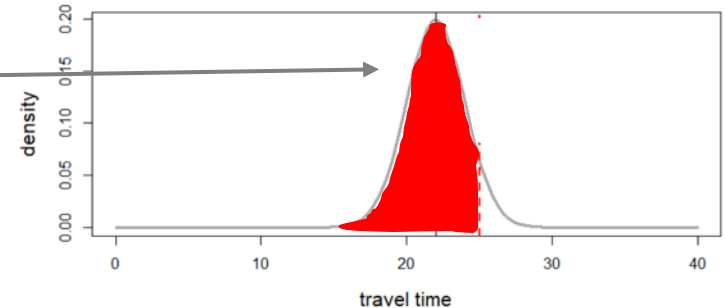
How is a probabilistic model trained via the Maximum Likelihood principle?
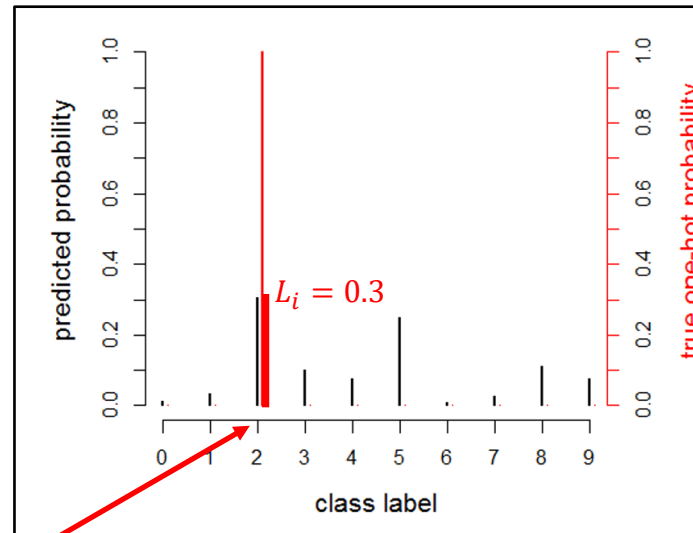
Likelihood L_i of the i-th observation (x_i, y_i)

- Based on the features x_i a CPD for the outcome is predicted
- The likelihood L_i of the observed y_i under the CPD is density of the CPD at the position y_i : $L_i = f_{\text{pred}}(y_i)$

CPD for a continuous outcome
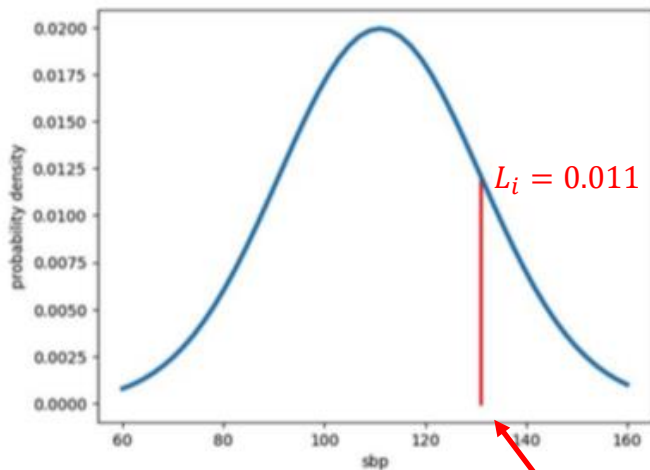
CPD for a discrete outcome

$L_i = 0.011$

$L_i = 0.3$

position of the observed outcome value

We use the NLL as loss to train probabilistic models

DATA $(\boldsymbol{x}_i, y_i)$

NLL2.30

FCNN2   GAUS2

$N(\mu(\boldsymbol{x}_i), \sigma(\boldsymbol{x}_i))$

f

y|x

DATA $(\boldsymbol{x}_i, y_i)$

NLL2.30

FCNN2   POIS

Pois$(\lambda(\boldsymbol{x}_i)$

p

Number of Events

y|$\boldsymbol{x}_i$

position of the observed outcome value y_i
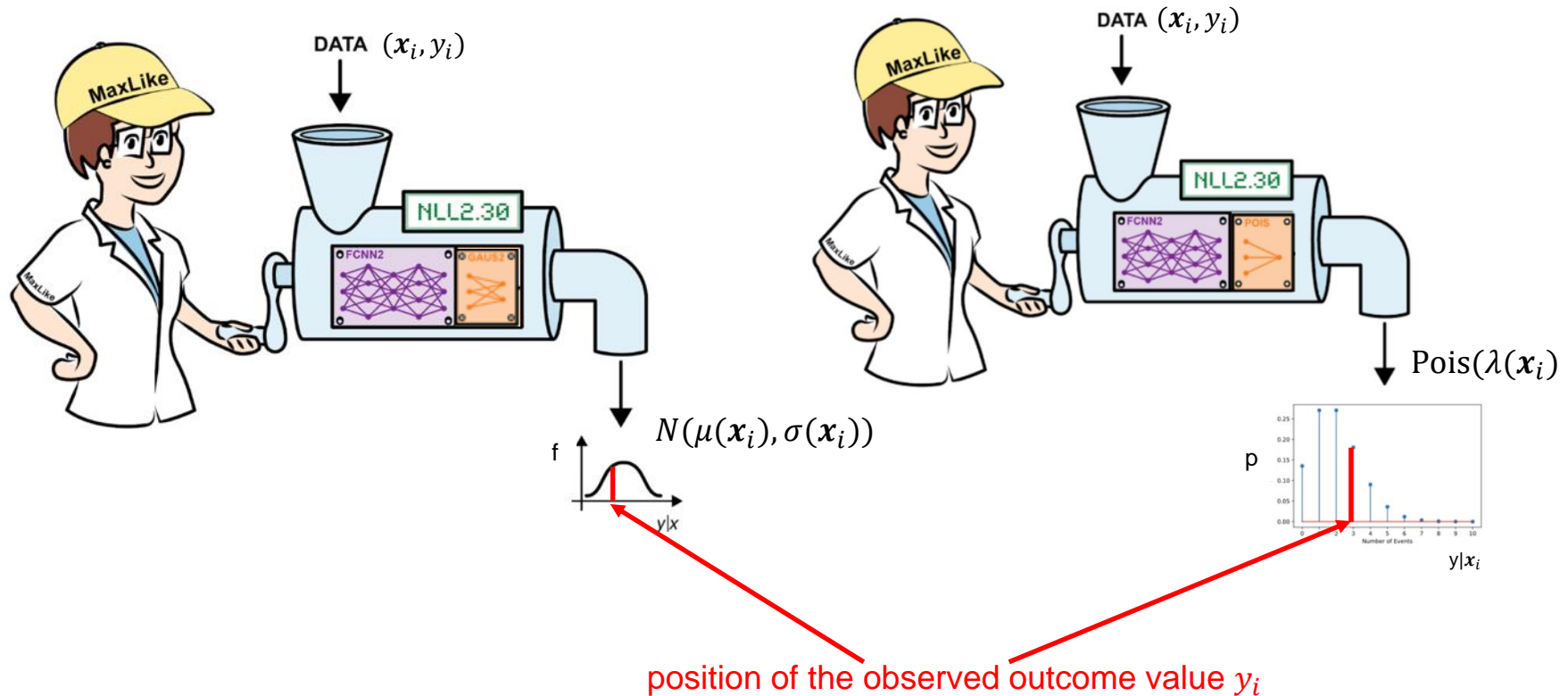
$$\text{NLL} = -\sum_{i=1}^{n} \log(L_i) = -\sum_i \log f(y_i|\boldsymbol{x}_i) = -\sum_i \log p(y_i|\boldsymbol{x}_i)$$

Side track:

Why the name X-entropy?

Side track: Entropy

$$H(P) = -\sum_i p_i \cdot \log(p_i)$$

Entropy is a measure for "untidyness".
If a distribution has only one peak, it is tidy and H=0
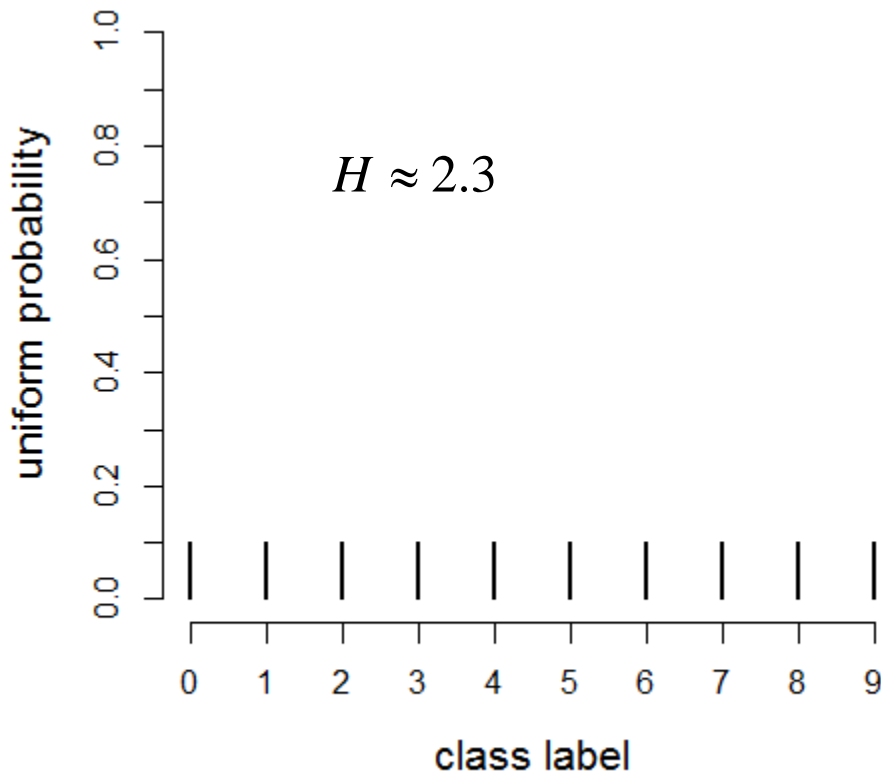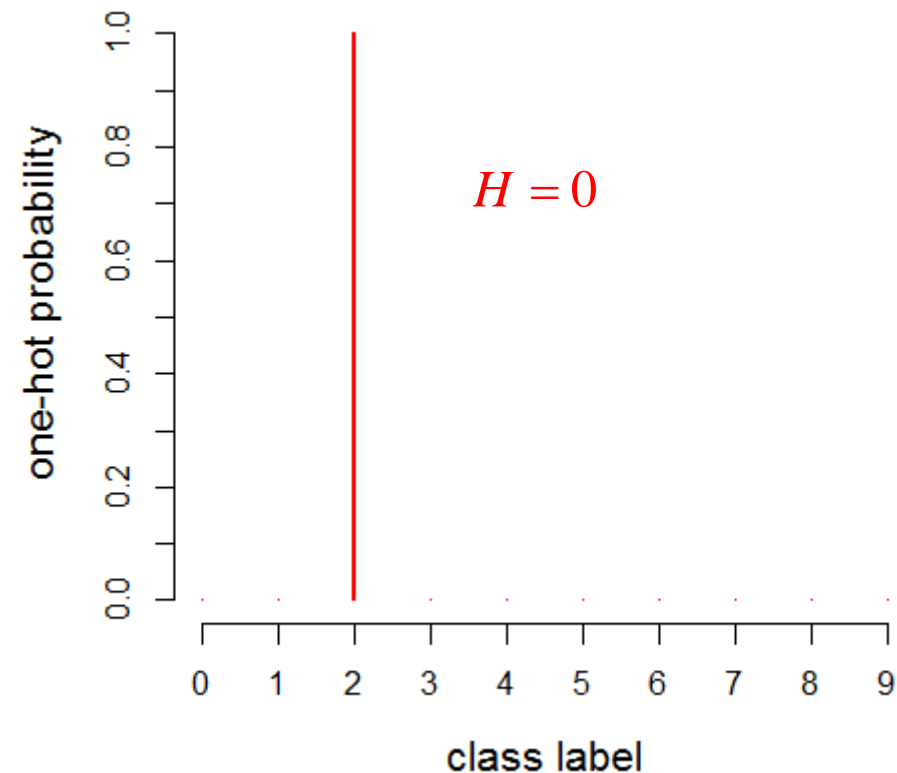If all outcomes are equally probable it is maximal untidy

$H = 0$

$H \approx 2.3$

X-entropy

$$\text{X-entropy} = \sum_i \text{H}(^{true}\text{p}_i, {}^{pred}\text{q}_i) = -\sum_i {}^{true}\text{p}_i \cdot \log(^{pred}\text{q}_i) = \text{NLL}$$
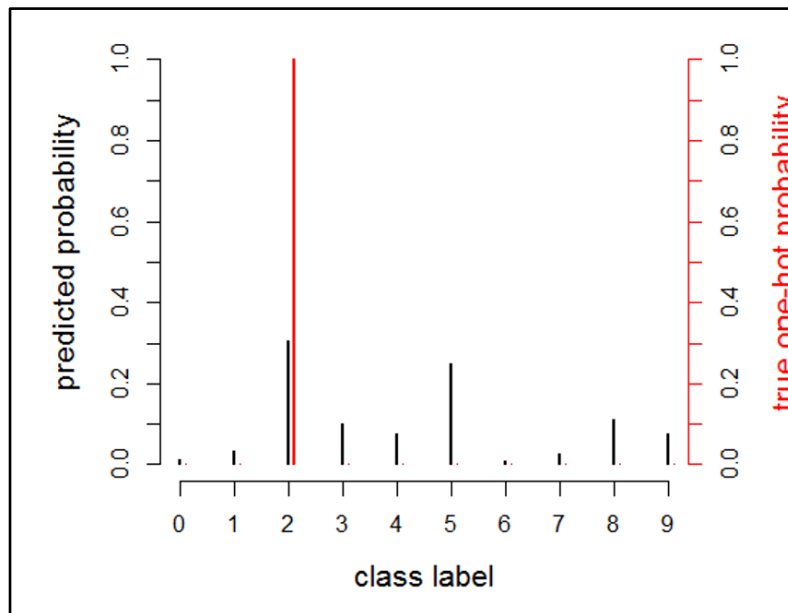
The cross-entropy is the same as the NLL, if the true probability distribution puts the whole weight on one class (which is in statistics often not the case, when it is possible, that different observations have the same features, but do not belong to the same class).

Looking back at homework

Task:

- Calculate (with calculator or numpy) 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

- See NB 12b_mnist_loglike

Machine Learning based on the Maximum Likelihood principle
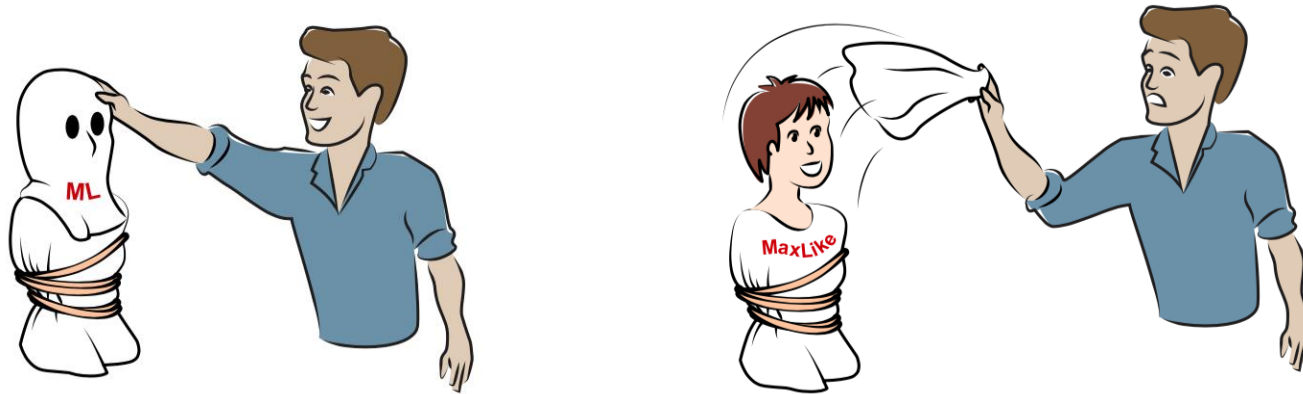
Unmasking the secrets of almost all loss functions in machine learning (ML in figure) and deep learning. After: https://www.instagram.com/neuralnetmemes/

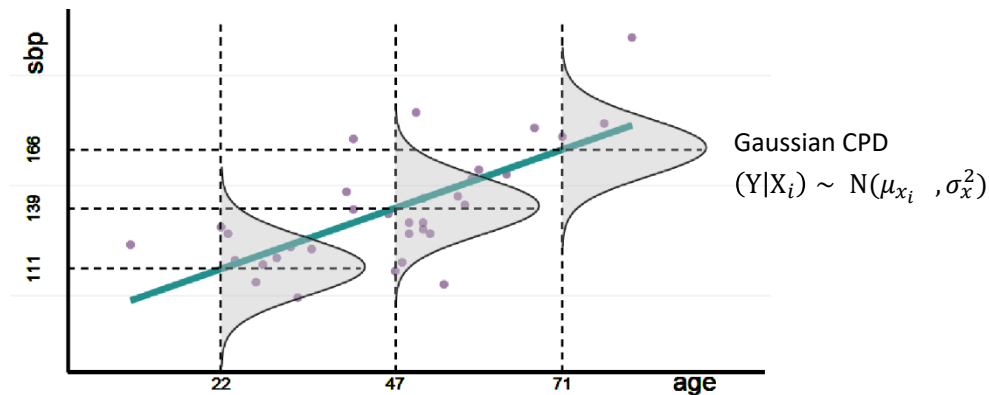Tune the parameters weights of the network, so that observed data (training data) is most likely under the predicted conditional probability distribution (CPD).
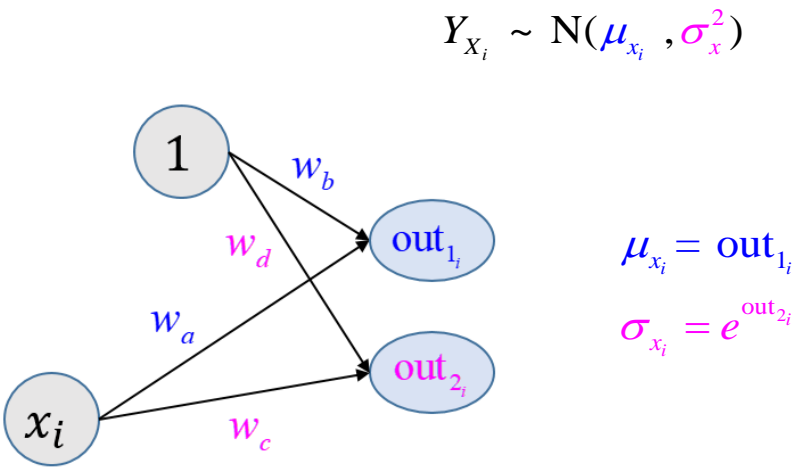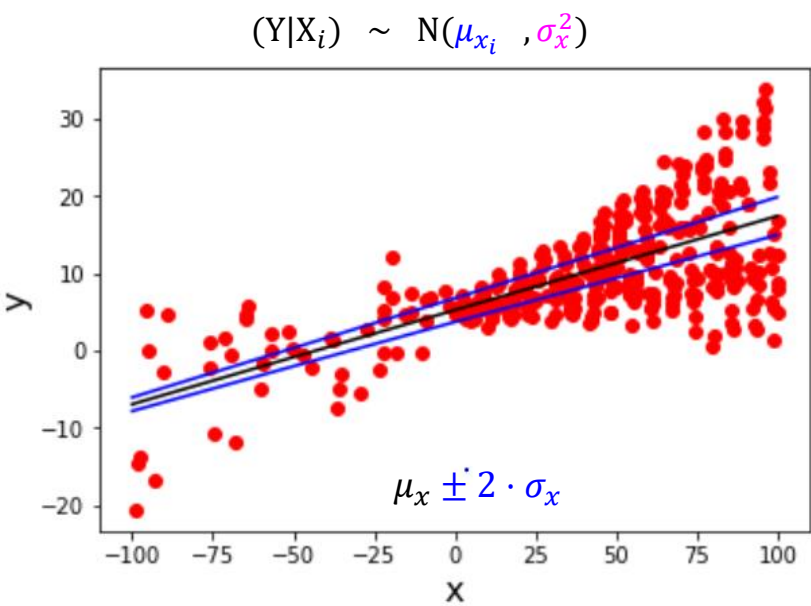
Practically: Minimize Negative Log-Likelihood:

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

Modeling continuous outcomes:
M1: Linear regression

Gaussian CPD

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

Fit a probabilistic regression with non-constant variance

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

$$\mu_x \pm 2 \cdot \sigma_x$$

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

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

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

Gaussian CPD

$$f(x) = \frac{1}{\sqrt{2\pi\sigma_{x_i}^2}} \cdot e^{-\frac{(y_i - \mu_{x_i})^2}{2\sigma_{x_i}^2}}$$
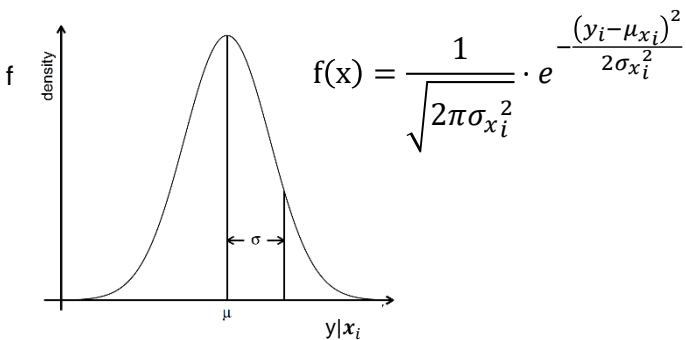
$$\widehat{w} = \operatorname*{argmin}_{w} \sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma_{x_i}^2}}\right) + \frac{(y_i - \mu_{x_i})^2}{2\sigma_{x_i}^2}$$

$$= \operatorname*{argmin}_{w} \sum_{i=1}^{n} (y_i - \mu_{x_i})^2$$

For lin regregession with constant variance, minimizing the NLL is equivalent to minimizing the MSE

SGD

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

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

Modelling the standard deviation (positive values)

- The variance or standard deviation are both non-negative $\sigma_{x_i} \geq 0$.
- 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

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

$\mu_x \pm 2 \cdot \sigma_x$

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

$$\sigma_{x_i} = e^{\text{out}_{2_i}}$$
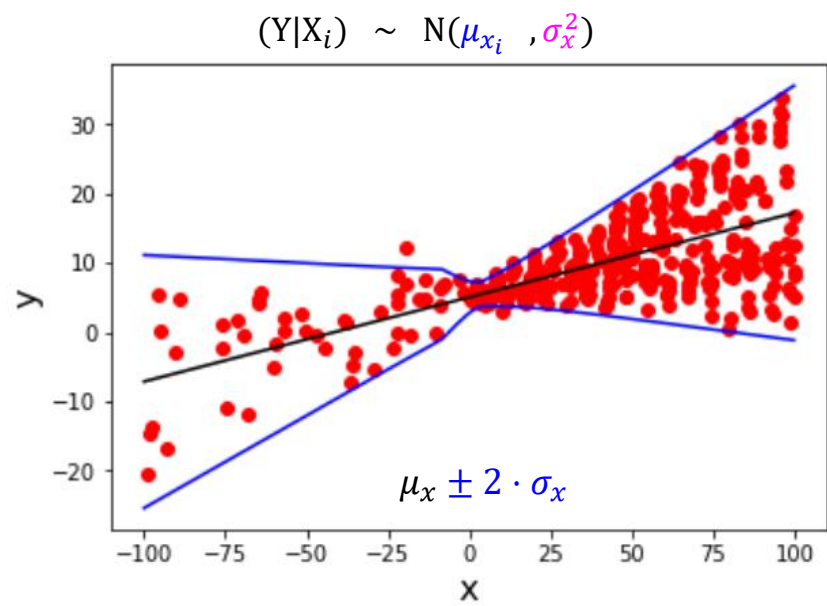
Minimize the negative log-likelihood (NLL):

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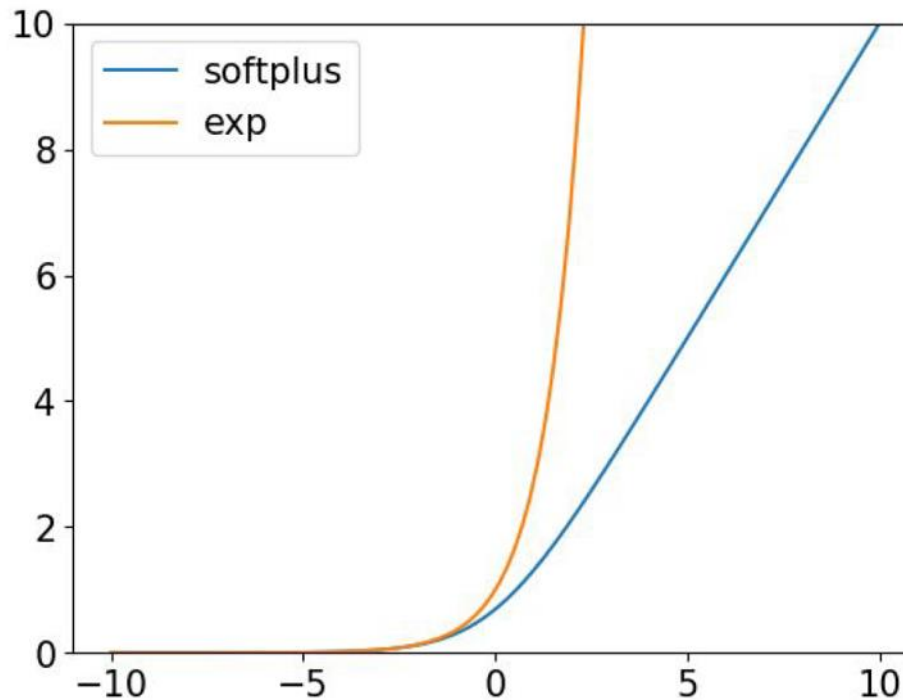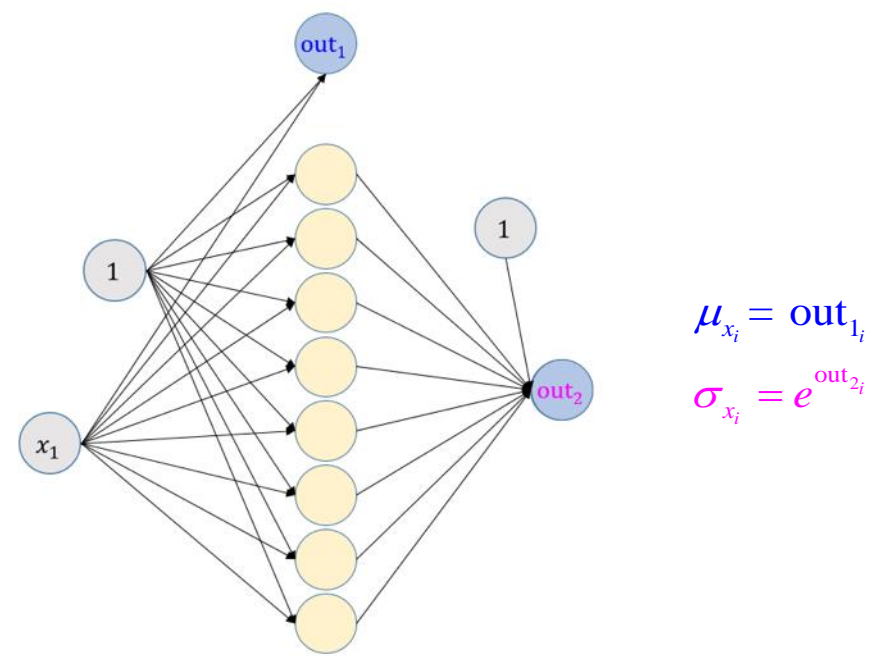
SGD

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

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

How to evaluate
a probabilistic prediction model?

Check prediction quality on NEW data

It's difficult to make predictions, especially about the future

Nils Bohr, physics Nobel price 1922

Common data split:

Train-data(50%)   Validation-data(25%)   Test-data(25%)
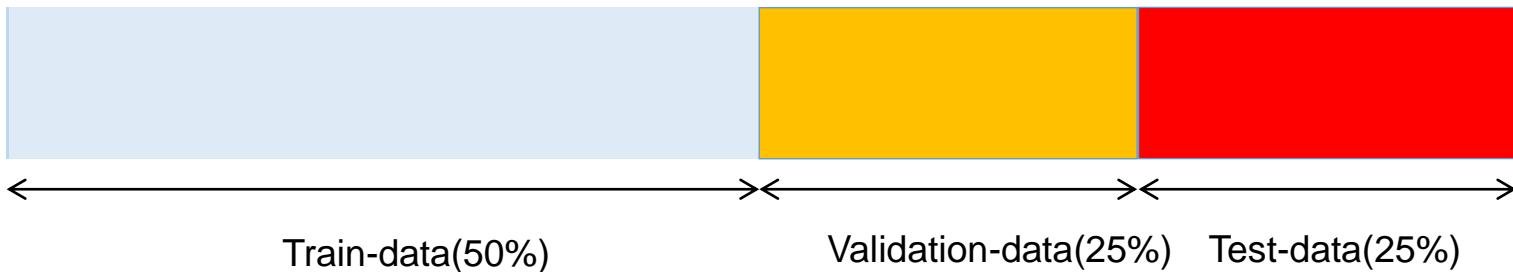
Visually: Do predicted and observed outcome distribution match?

Validation data along with predicted outcome distribution (Gauss with const σ)
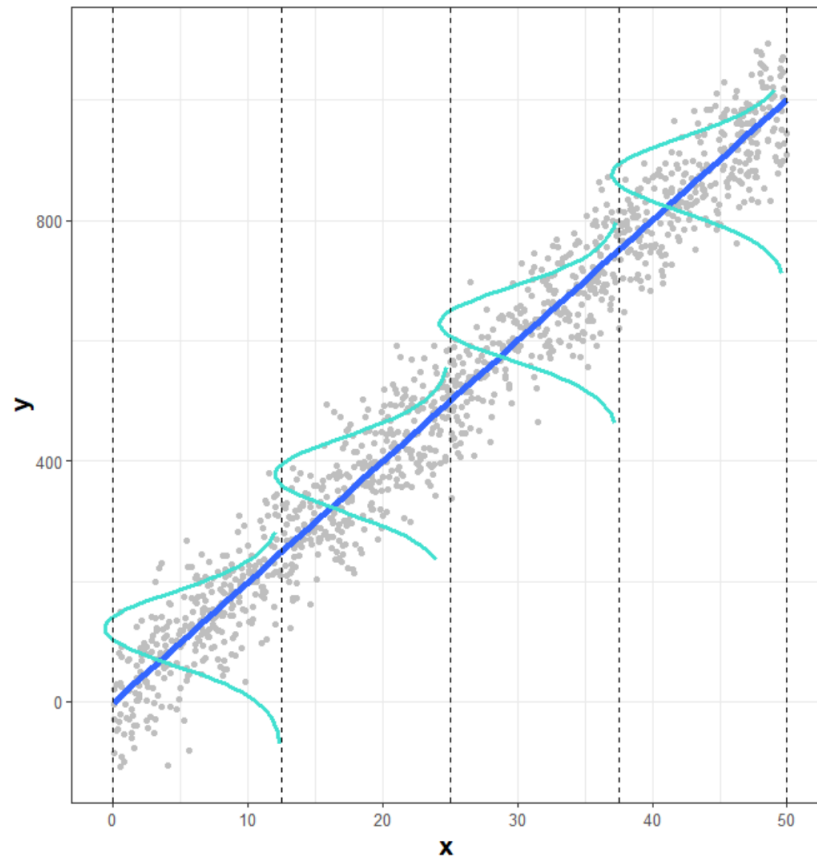
Validation data along with predicted and observed outcome distribution
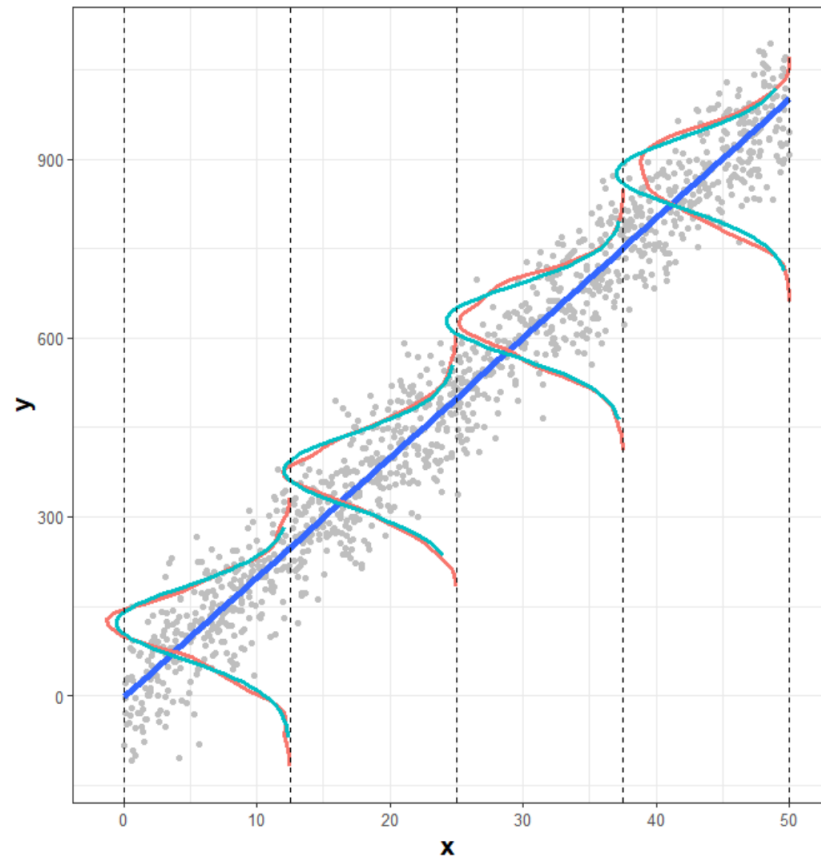
A large validation data set is needed to ensure underlying assumption: observed distribution = data generating distribution

Example: Simulated data for linear regression models
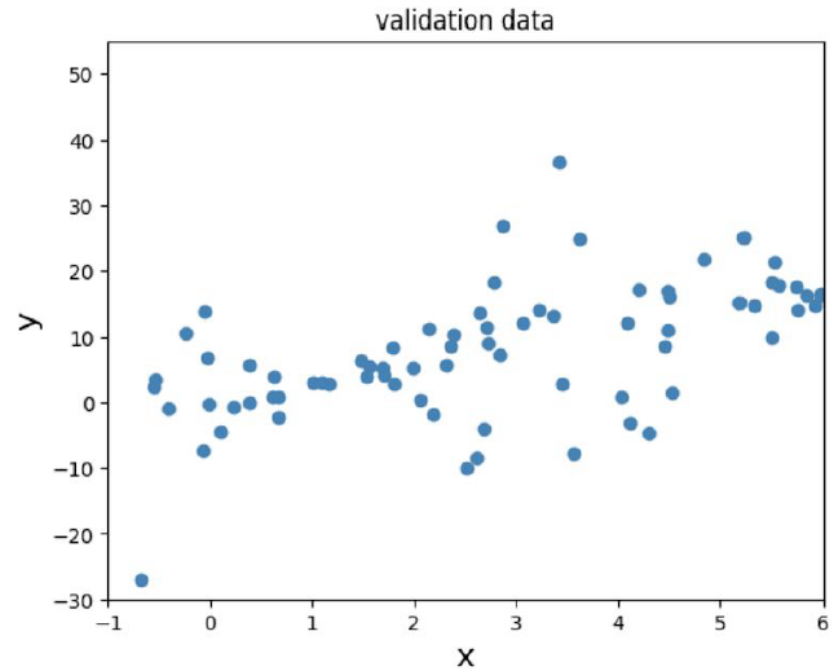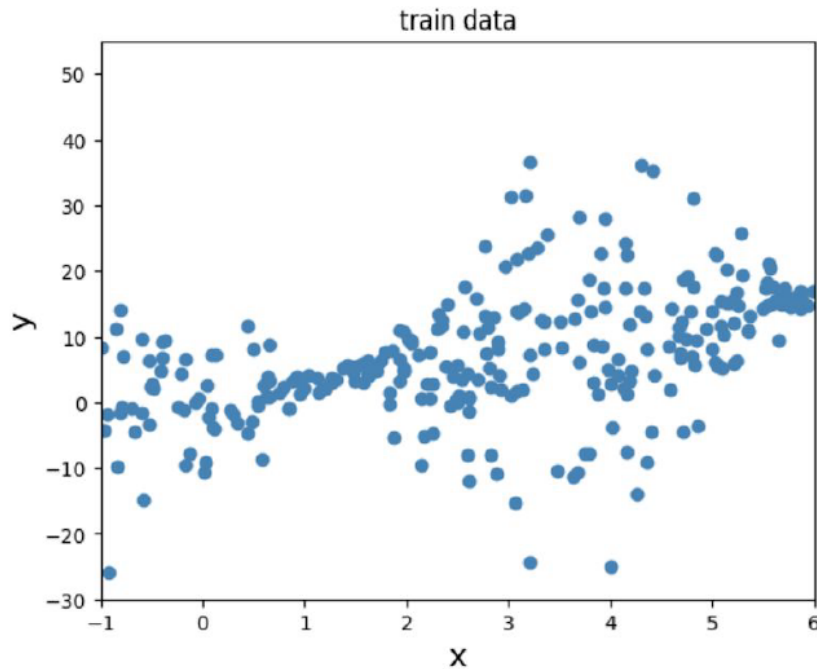
Model_1 (linear regression with constant variance): $(y \mid x) \sim N(\mu_x, \sigma^2)$

Model_2 (linear regression with flexible variance): $(y \mid x) \sim N(\mu_x, \sigma_x^2)$

Predicted outcome distribution from model_1 (constant σ)
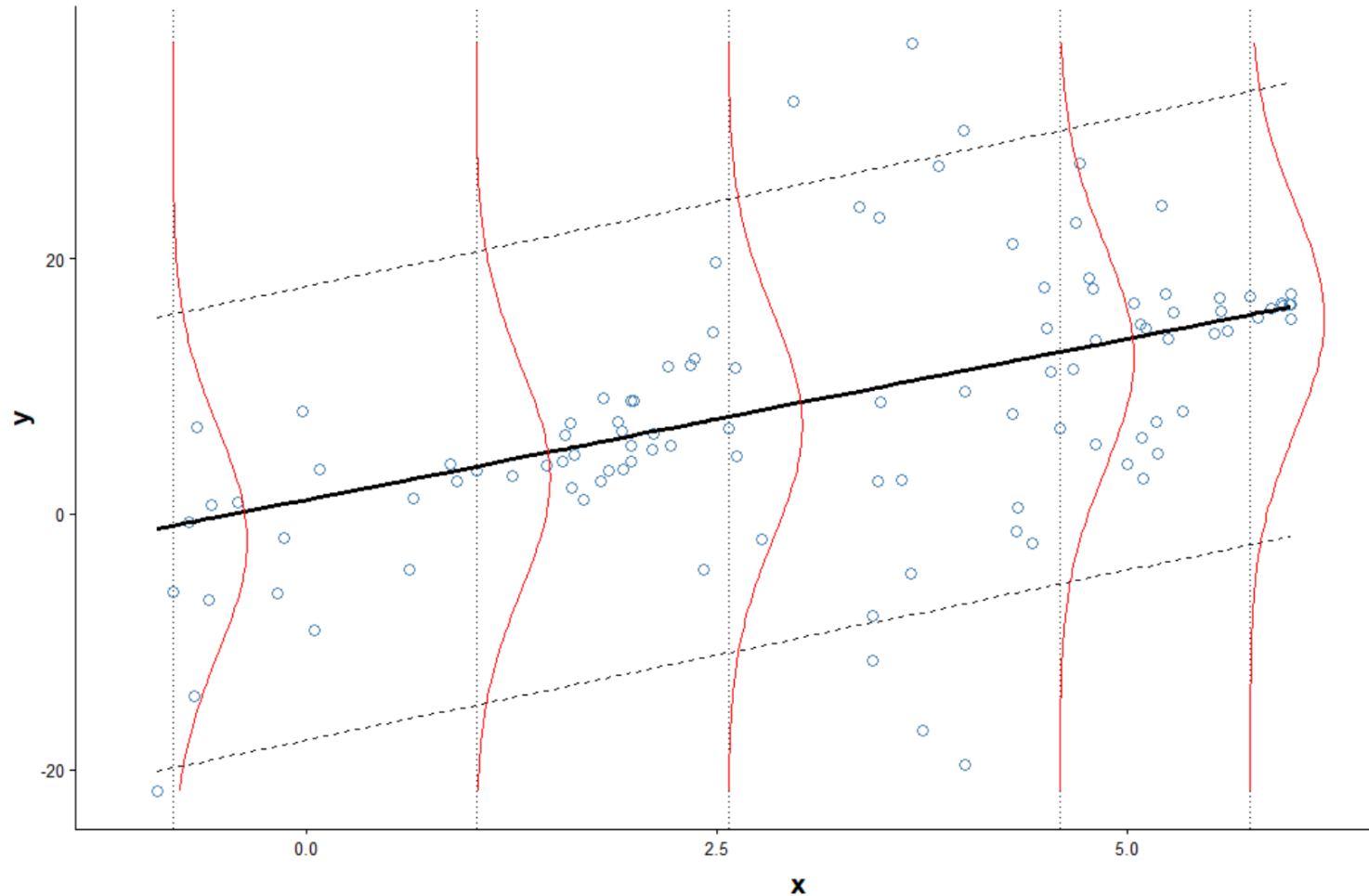
Validation data

Root mean square error (RMSE) or mean absolute error (MAE)

Validation data
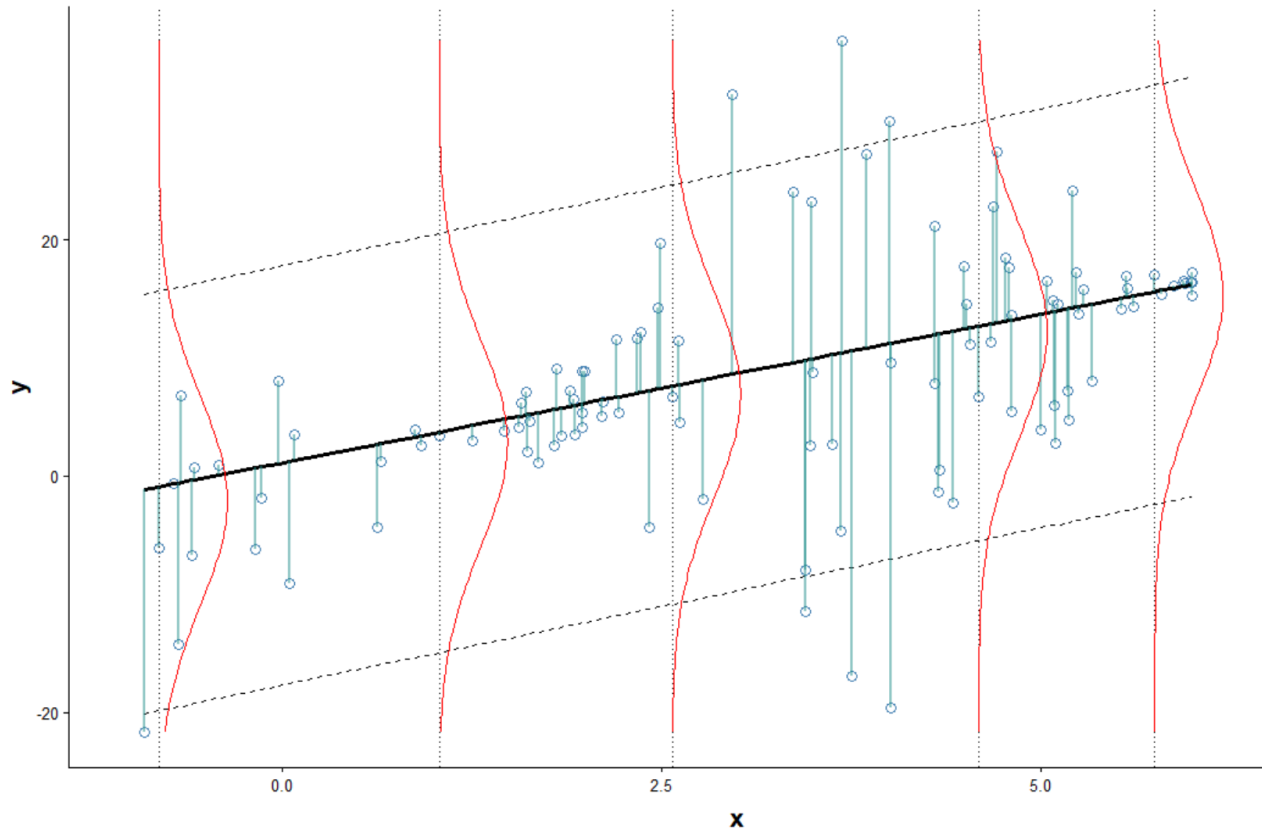
$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{\mu}_{x_i})^2}$$

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\left|y_i - \hat{\mu}_{x_i}\right|$$

RMSE and MAE alone do not capture performance for probabilistic models!

Both only depend on the mean (μ) of the CPD, but not on it's shape or spread (σ) and are not appropriate to evaluate the quality of the predicted distribution of a probabilistic model.

Use the NLL to score a probabilistic prediction model

A score *S* takes the CPD and *one test instance* and yields a real number
(the smaller the score the better is the predicted CPD)

$$S_{\text{NLL}}\left(p(y\,|\,x_{val}),y_{val}\right)=-\log\left(p(y_{val}\,|\,x_{val})\right)$$
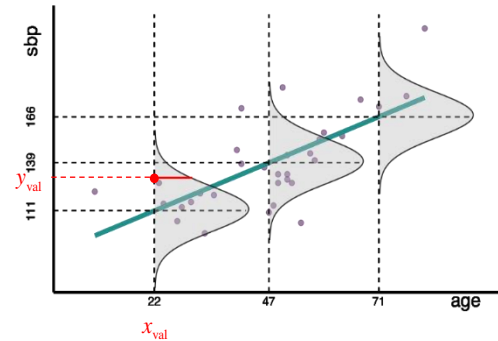
Definition of the term "strictly proper": A strictly proper score is then and only then minimal, when the predicted CPD matches the true CPD.

It is provable that the negative log-likelihood (NLL, aka 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)

Use validation NLL to compare probabilistic models

Model_1 (linear regression with constant variance):

current mean_NLL = 5.48

$$(y \mid x) \sim N(\mu_x, \sigma^2)$$
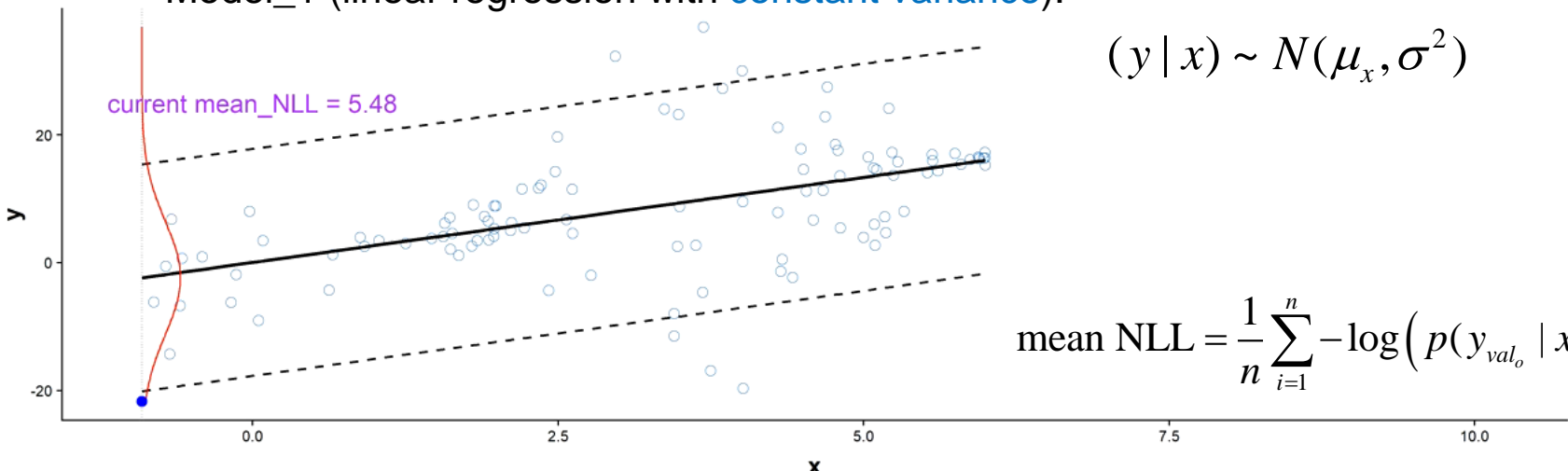
$$\text{mean NLL} = \frac{1}{n} \sum_{i=1}^{n} -\log\left(p(y_{val_o} \mid x_{val_i}) \right)$$

Model_2 (linear regression with flexible variance):

current mean_NLL = 4.84

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

Tensorchief's youtu.be channel

26
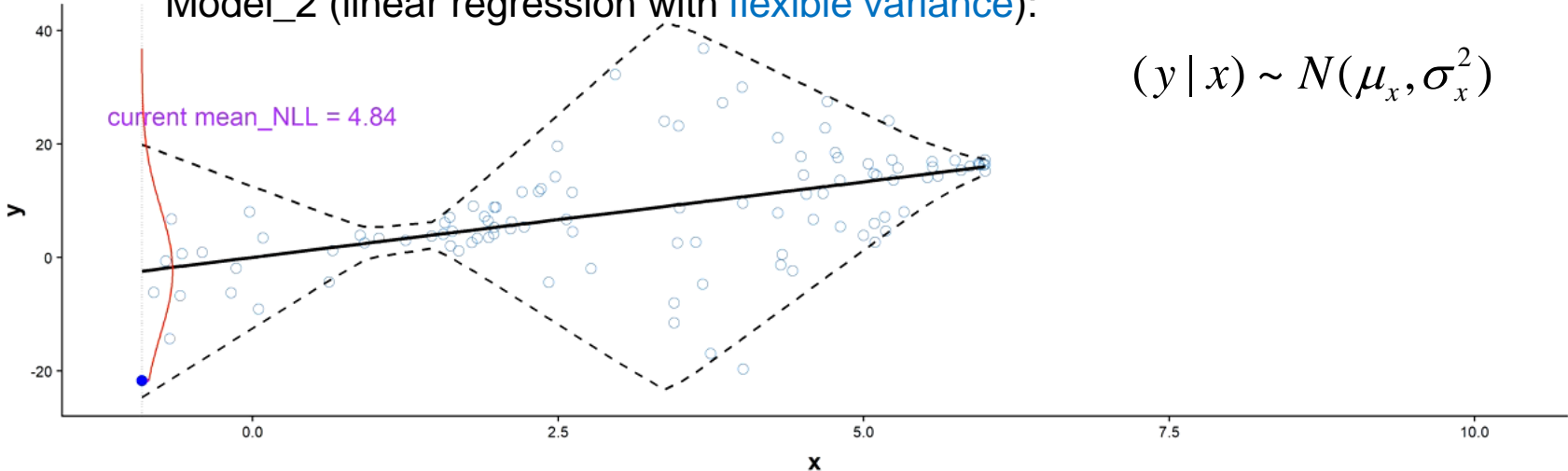
Two practical ways of modeling and fitting CPD

Approach 1 (using standard keras):

- Use the nodes in the last layer of a NN to control the parameters of a CPD (e.g. μ and σ in case of a Gaussian)

- Use the CPD the determine the likelihood for each observation and compute the NLL over all training observations, which serves as loss.

- Keras has build in loss functions (https://keras.io/losses/) for the following CPDs:
    - Discrete outcomes
        - Bernoulli CPD: loss='binary_crossentropy '
        - Categorical CPD: loss='categorical_crossentropy '
        - Poisson CPD: loss='poisson'
    - Continuous outcomes
        - Gaussians CPD (with fixed σ): loss='mean_squared_error'

Approach 2 (using tensorFlow robability):
- use CPDs and corresponding loss functions from TFP

TensorFlow Probability

TFP: Working with probability distributions

```python
import tensorflow_probability as tfp
tfd = tfp.distributions
d = tfd.Normal(loc=[3], scale=1.5)
x = d.sample(2)
px = d.prob(x)
print(x)
print(px)
```

Creates a 1D Normal distribution with a mean of 3 and a standard deviation of 1.5

Samples two realizations from the Normal distribution

Computes the likelihood for each of the two sampled values in the defined Normal distribution

```
tf.Tensor( [[2.3158536][5.8714476]],
shape=(2, 1), dtype=float32)

tf.Tensor( [[0.23968826][0.04256715]],
shape=(2, 1), dtype=float32)
```

```python
x = np.linspace(-10,10,100)
plt.plot(x, d.prob(x))
plt.show()
```


https://towardsdatascience.com/introduction-to-tensorflow-probability-6d5871586c0e
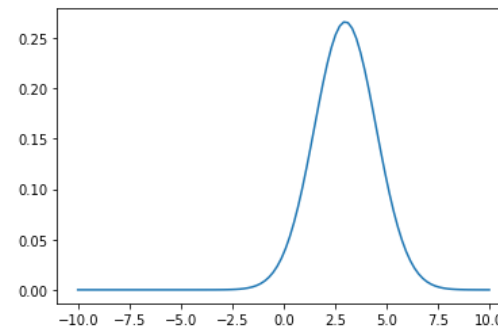
TensorFlow Distributions

TensorFlow Probability (TFP) is nice add-on to Keras:

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`

https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/12_TensorFlowDistributions_Demo.ipynb

TFP Lambda Layer for outputting a TFP distribution

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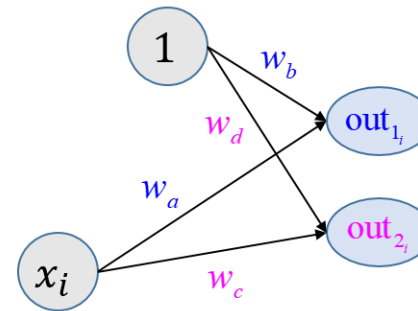
The Lambda Layer

```
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()
```

Get the loss and predicted CPD from the TFP distribution

```python
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
```

```python
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
```

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

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

```python
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_2022/blob/master/notebooks/13_linreg_with_tfp.ipynb

Modeling count data:

M2: Poisson regression
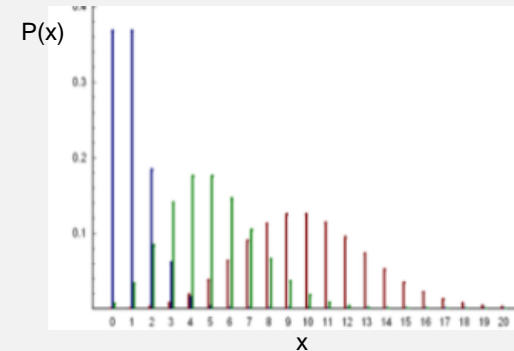
Recall the classical Poisson model for count data

X: number of incidences **per time unit**

The Poisson model is appropriate to model **counts X per unit**, assuming that
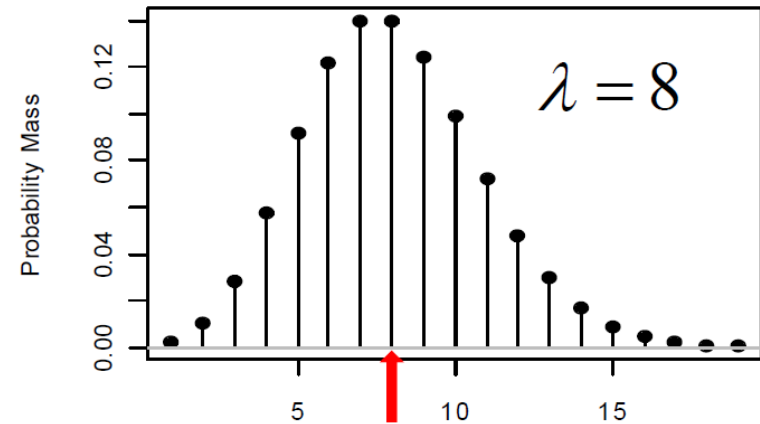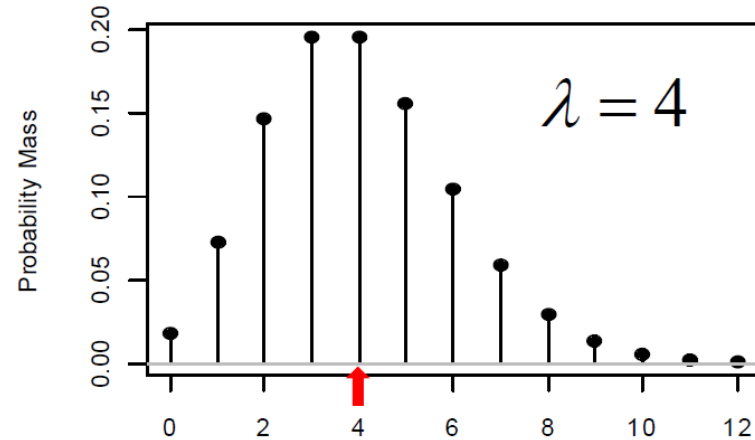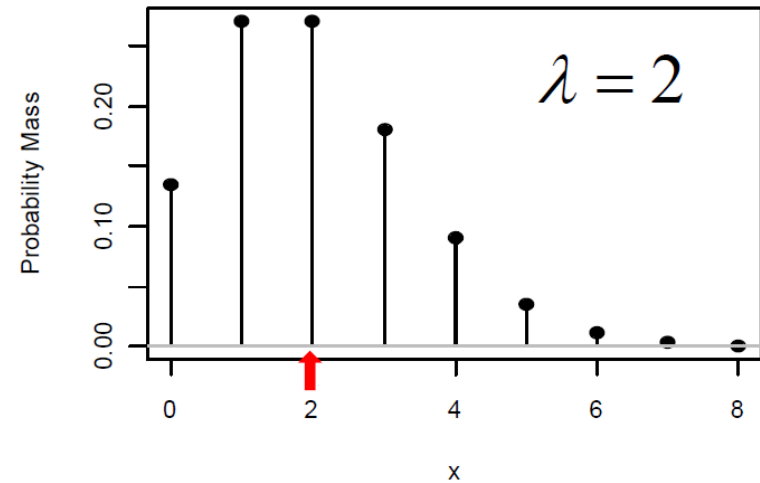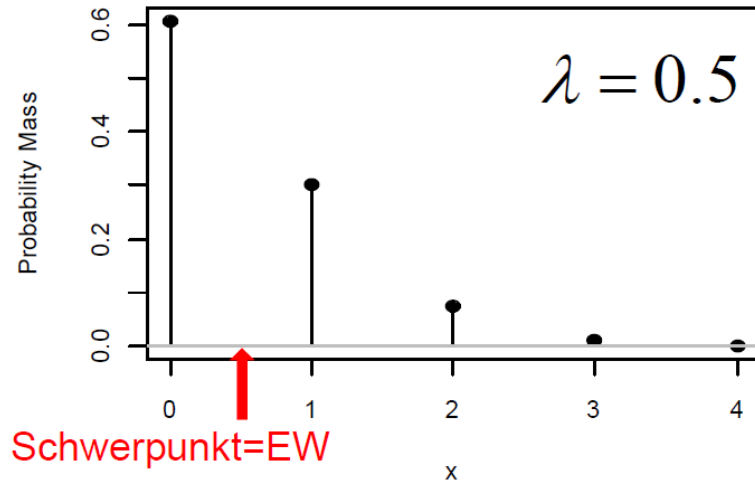1) the unknown incidence rate λ (per time unit) is constant and
2) the incidences occur independently

$$P(X = x) = \frac{\lambda^x}{x!} e^{-\lambda} \ , \quad x = 0, 1, 2, \ldots$$

$$E(X) = Var(X) = \lambda \quad \text{: expected number of counts per time unit}$$

The shape and mean of the Poisson distribution depends on λ

Schwerpunkt=EW

The Poisson distribution in tfp

```
dist = tfd.poisson.Poisson(rate = 2) #A

vals = np.linspace(0,10,11) #B

p = dist.prob(vals) #C

print(dist.mean().numpy())   #D

print(dist.stddev().numpy())    #E

#A Poisson distribution with parameter rate = 2
#B Integer values from 0 to 10 for the x-axis in figure 5.13
#C Computes the probability for the values
#D The mean value, yielding 2.0
#E The standard deviation, yielding sqrt(2.0) = 1.41...
```
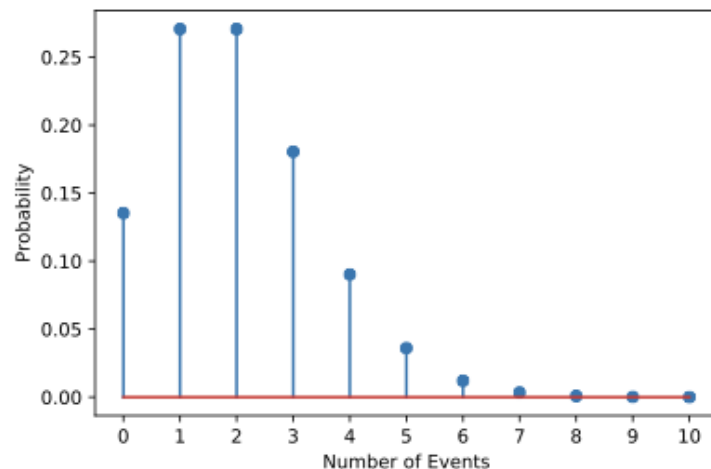

Figure 5.13 Poisson distribution for the case that there are, on average, two events per unit

Poisson regression for count data

Goal: Predict a Poissonian CPD for (Y|X=x) which depends on predictor values

CPD: $Y_{X_i} = (Y|X_i) \sim \text{Pois}(\lambda_{x_i})$

We only need to model λ_x to fix the Poissonian CPD!

Model:

$$\log(\lambda_i) = \beta_0 + \beta_1 x_{i1} + ... + \beta_p x_{ip}$$

Linear predictor η_i

link-function: ensures positive λ after back-transformation

CPD encoded by Poisson regression

CPD: $Y_{X_i} = (Y|X_i) \sim \text{Pois}(\lambda_{x_i})$

$Y_x \in \mathbb{N}_0$, $\lambda_x \in \mathbb{R}$

$$\log\left(E(Y_{x_i})\right) = \log(\lambda_{x_i}) = \beta_0 + \beta_1 x_{i1}$$

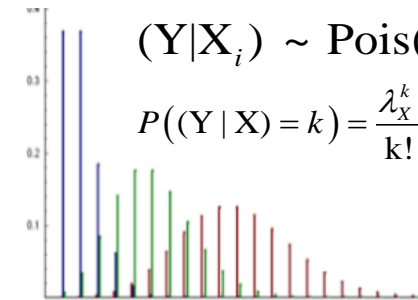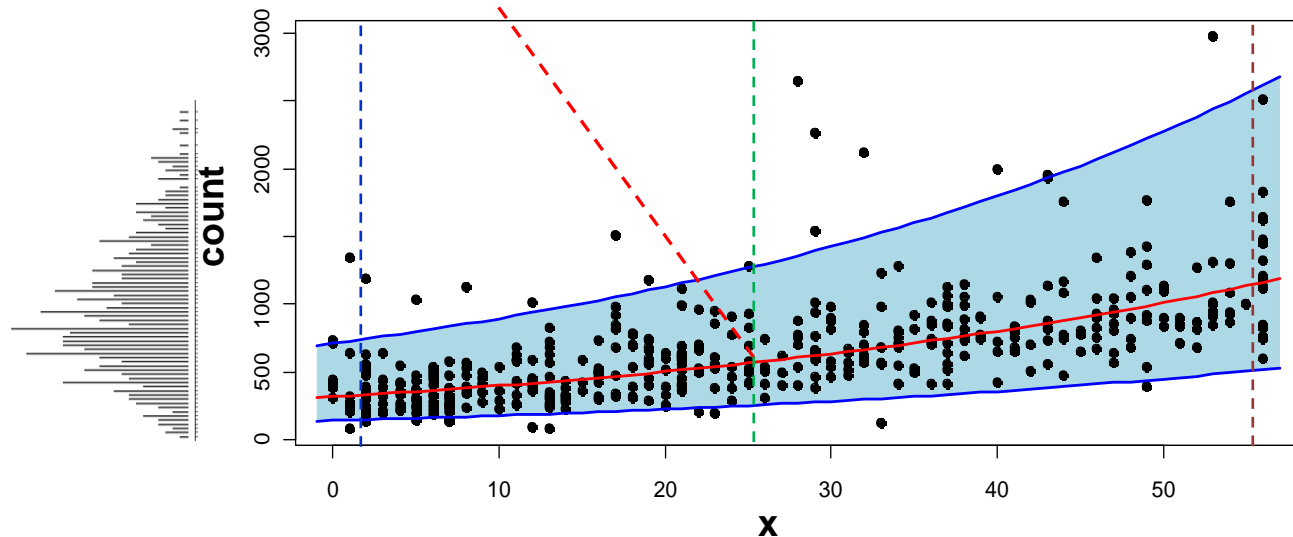$$E(Y_{x_i}) = \text{Var}(Y_{x_i}) = \lambda_{x_i} = e^{\beta_0 + \beta_1 x_{i1}}$$

The predicted value of the Poisson regression gives the only one parameter of the CPD: λ_x that depends on the predictor values. We have no error term in the regression formula since the uncertainty of the outcome (counts) is given by the probabilistic Poisson model $\text{Pois}(\lambda_x)$

$(Y|X_i) \sim \text{Pois}(\lambda_{x_i})$

$$P\big((Y|X) = k\big) = \frac{\lambda_X^k}{k!} e^{-\lambda_X} , \quad k = 0,1,2,...$$

$Y \sim V_{arbirary}^{discrete}$

$(Y|X_i) \sim \text{Pois}(\lambda_{x_i})$

Model 2: Poisson regression via NNs in keras

We use a NN without hidden layer to control the rate λ.

```python
inputs = Input(shape=(X_train.shape[1],))
rate = Dense(1,
        activation=tf.exp)(inputs) #A
p_y = tfp.layers.DistributionLambda(tfd.Poisson)(rate)

model_p = Model(inputs=inputs, outputs=p_y) #C



def NLL(y_true, y_hat): #D
    return -y_hat.log_prob(y_true)

model_p.compile(Adam(learning_rate=0.01), loss=NLL)
model_p.summary()
```

Using the exp-activation ensures that the rate λ is a positive number

Glueing the NN and the output layer together. Note that output p_y is a tf.distribution

The second argument is the output of the model and thus a TFP distribution. It's as simple as calling log_prob to calculate the log probability of the observation that's needed to calculate the NLL
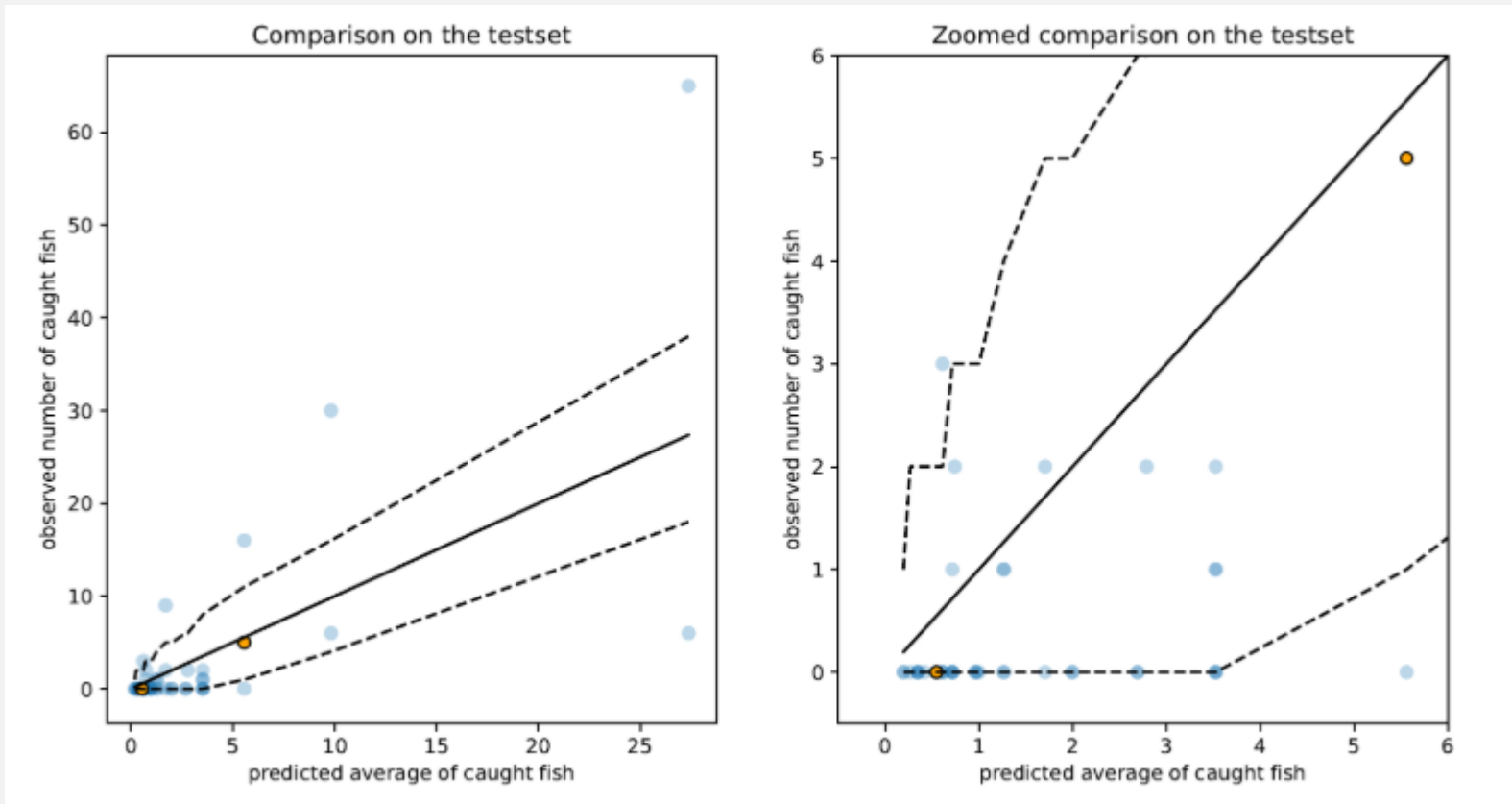
Predict CPD for outcome in test data:

Group 31 used livebait, had a camper and were 4 persons with one child. Y=5 fish.
Group 33 used livebait, didn't have a camper and were 4 persons with two childern. Y=0 fish.

What is the likelihood of the observed outcome in test obs 31 and 33?

The mean of the CPD is depicted by the solid lines.
The dashed lines represent the 0.025 and 0.975 quantiles,
yielding the borders of a 95% prediction interval.
Note that different combinations of predictor values can yield the same parameters of the CPD.

Approach 2 Model the CPD

Machen Sie Aufgabe 14_poisreg_with_tfp.ipynb

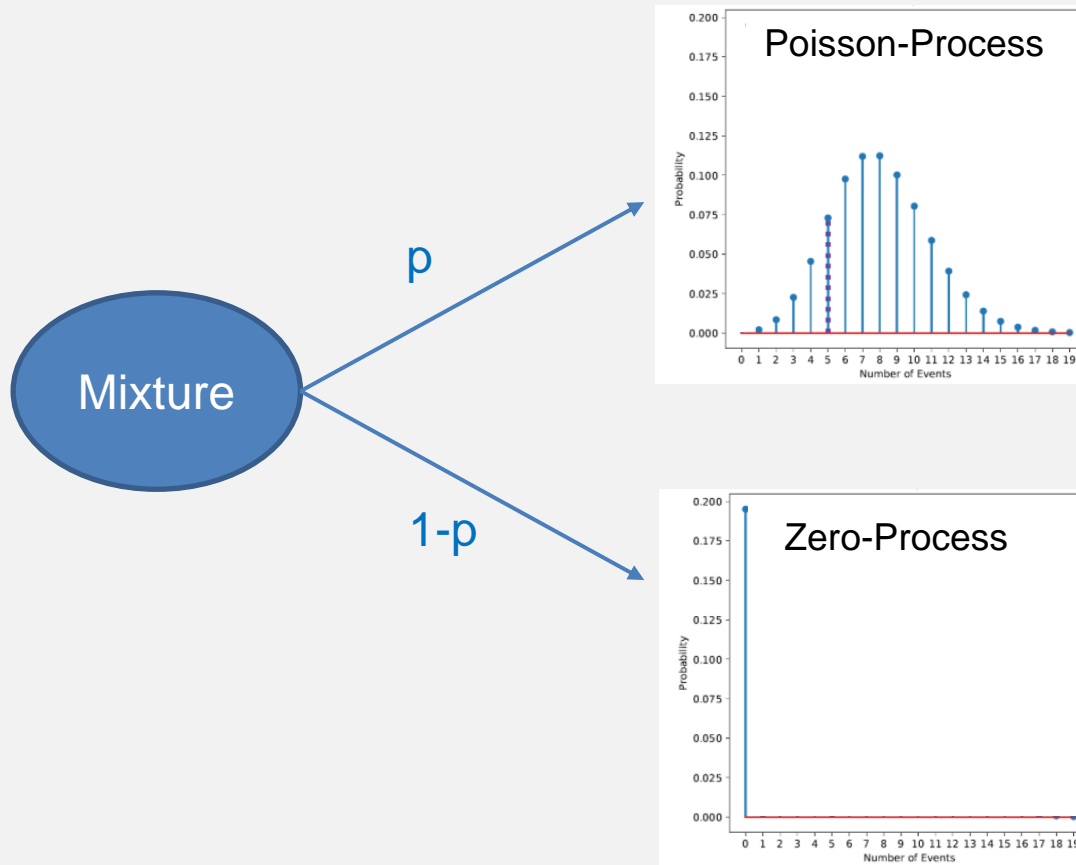https://github.com/tensorchiefs/dl_course_2022/blob/master/notebooks/14_poisreg_with_tfp.ipynb

Modeling count data:

M3: ZIP regression

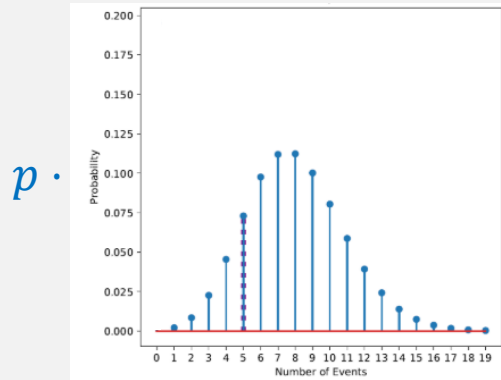# Zero-Inflated Poisson (ZIP) as Mixture Process

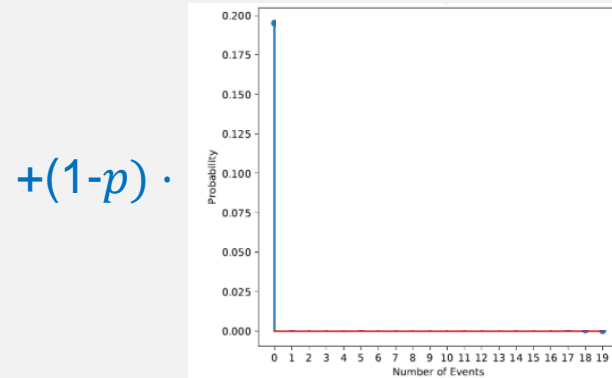How many fish a group catches does not only depend on luck ;-)

Zero-Inflated Poisson (ZIP) can be seen as Mixture Distribution
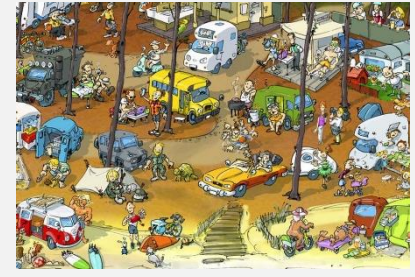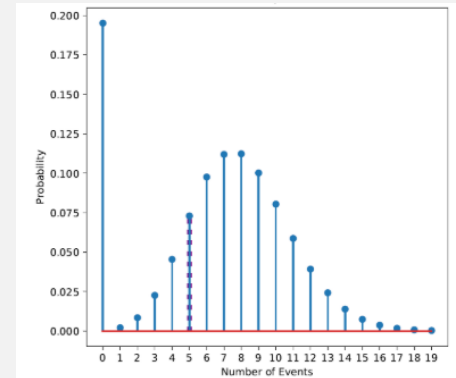
Poisson-Process   Zero-Process   Zero-inflated Poisson

$p \cdot$ $+(1\text{-}p) \cdot$ $=$

Custom distribution for a ZIP distribution

```python
def zero_inf(out):

    rate = tf.squeeze(tf.math.exp(out[:,0:1])) #  First NN output controls lambda. Exp guarantee value >0

    s = tf.math.sigmoid(out[:,1:2]) #  Second NN output controls p; sigmoid guarantees value in [0,1]

    probs = tf.concat([1-s, s], axis=1) #  The two probabilities for 0's or Poissonian distribution

    return tfd.Mixture(

        cat=tfd.Categorical(probs=probs),#  tfd.Categorical allows creating a mixture of two components

        components=[

        tfd.Deterministic(loc=tf.zeros_like(rate)), #  Zero as a deterministic value

        tfd.Poisson(rate=rate), #  Value drawn from a Poissonian distribution

        ])
```

Model 3: Zero-Inflated Poisson regression via NNs in keras

```python
## Definition of the custom parameterized distribution

inputs = tf.keras.layers.Input(shape=(X_train.shape[1],))

out = Dense(2)(inputs) #A

p_y_zi = tfp.layers.DistributionLambda(zero_inf)(out)

model_zi = Model(inputs=inputs, outputs=p_y_zi)
```
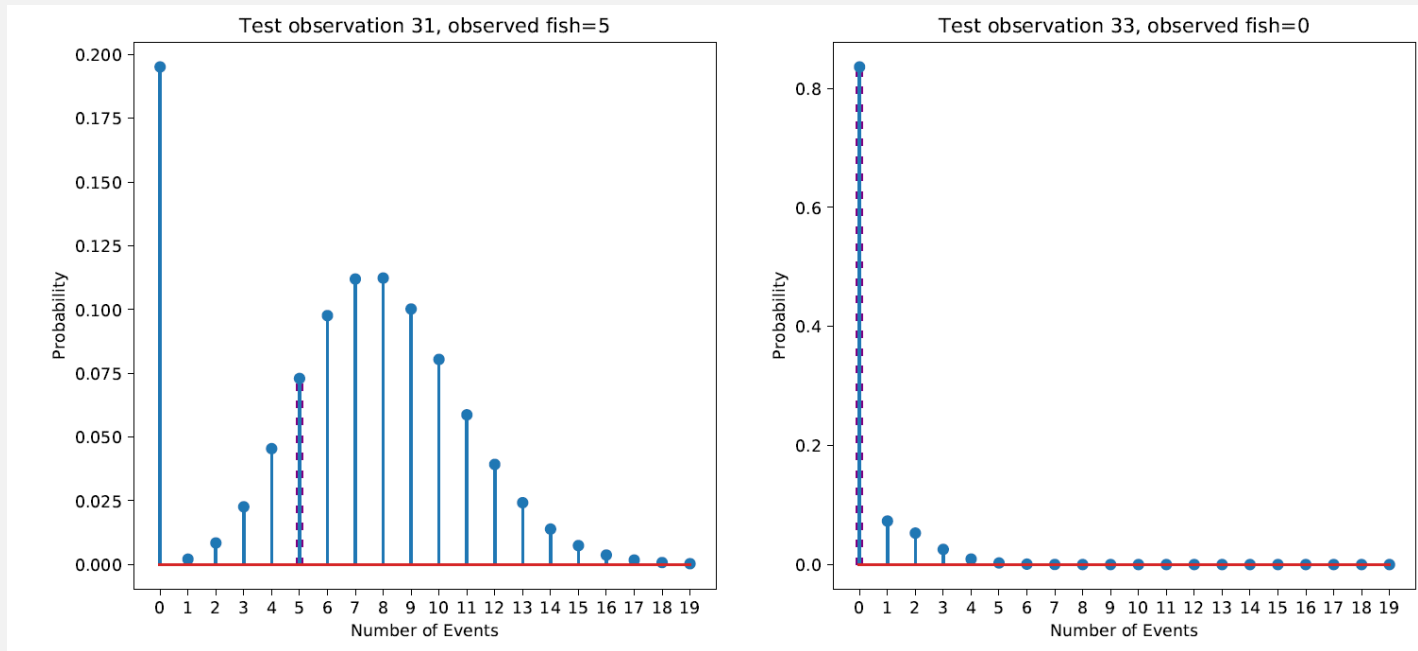
Model 3: ZIP regression, get test NLL from Gaussian CPD
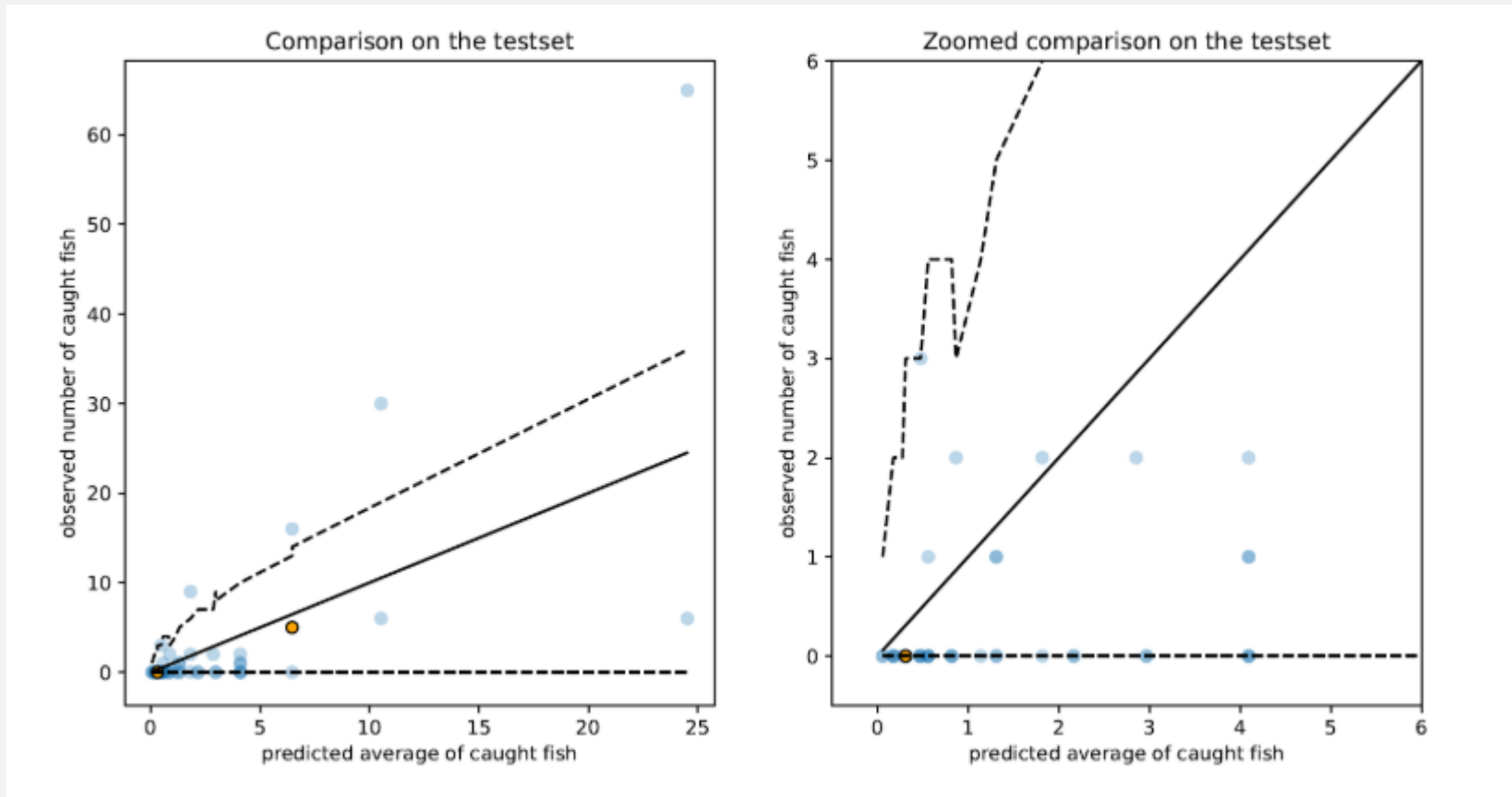
Predict CPD for outcome in test data:

Group 31 used livebait, had a camper and were 4 persons with one child. Y=5 fish.
Group 33 used livebait, didn't have a camper and were 4 persons with two childern. Y=0 fish.

What is the likelihood of the observed outcome in test obs 31 and 33?

# Model 3: ZIP regression, visualize the CPDs by quantiles

The mean of the CPD is depicted by the solid lines.
The dashed lines represent the 0.025 and 0.975 quantiles,
yielding the borders of a 95% prediction interval.
Note that different combinations of predictor values can yield the same parameters of the CPD.

Validation NLL allows to rank different probabilistic models
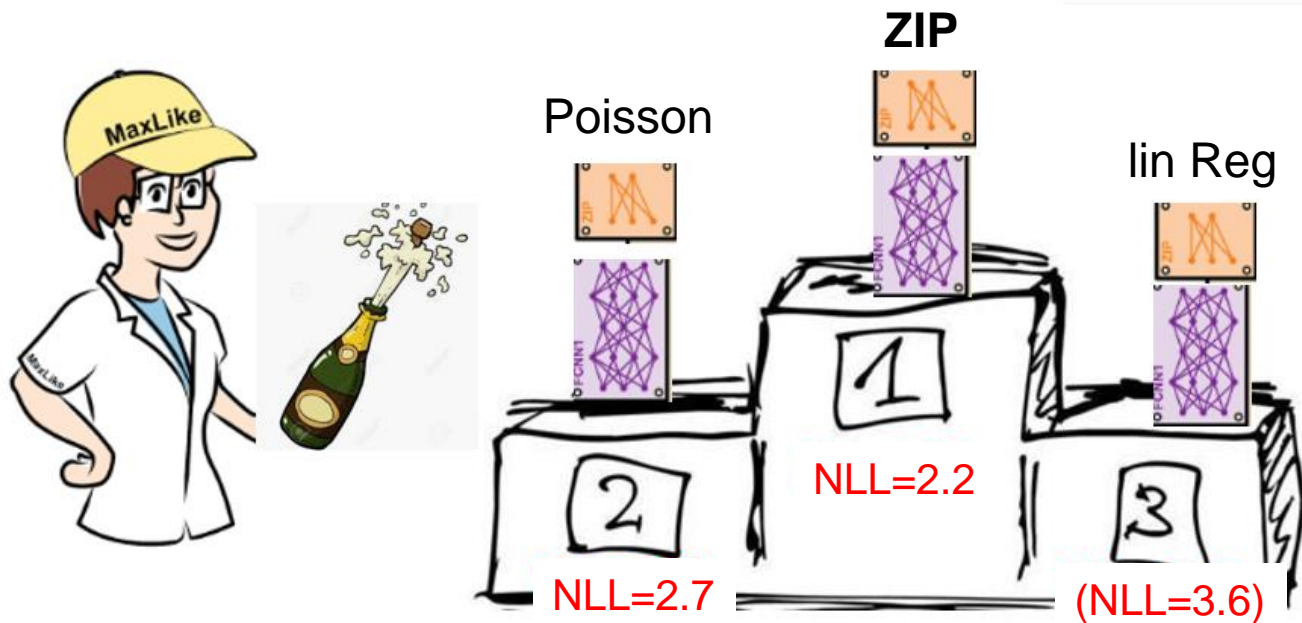
For prediction performance ranking, only compare NLL from models that were evaluated on the same data and that have the outcome scale.

Summary

- A probabilistic model predicts for each input a whole conditional probability distribution (CPD).

- The predicted CPD assigns for each possible outcome y, a probability with which it's expected.

- The negative log-likelihood (NLL) measures how well the CPD matches the actual distribution of the outcomes (lower is better).

- The NLL is used as a loss function when training a probabilistic model.

- The NLL on new data is used to measure, and to compare, the prediction performance of different probabilistic models.

- Using a proper choice for the CPD enhances the performance of your models.

- For continuous data, a common first choice is a Normal distribution

- For count data, common choices for distribution are Poisson, Negative-Binomial, or Zero-inflated Poisson (ZIP).