Machine Intelligence:: Deep Learning
Week 5

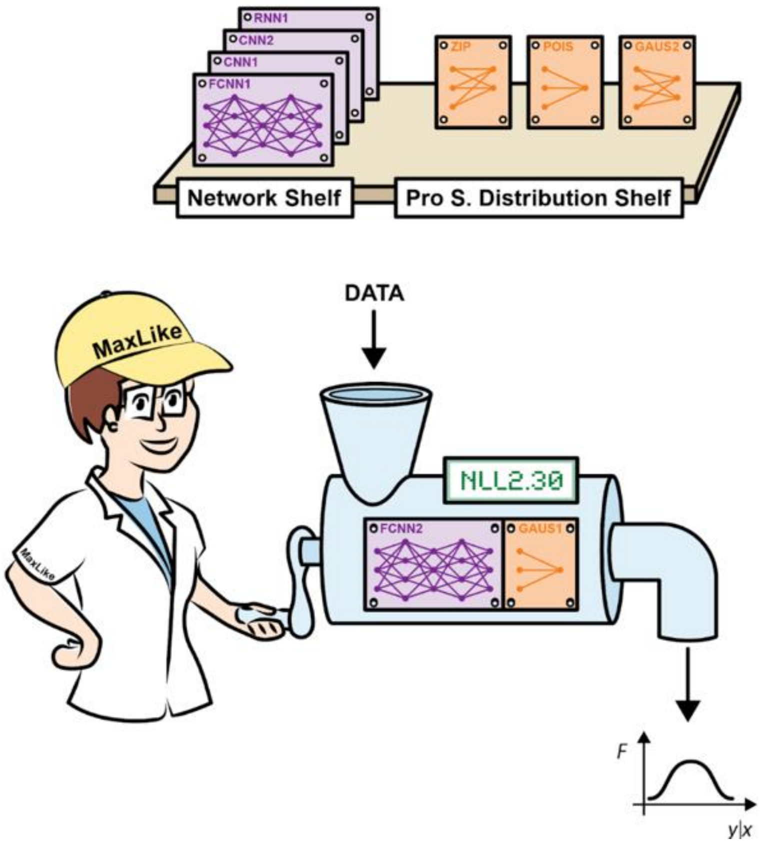
Beate Sick, Elvis Murina, Oliver Dürr

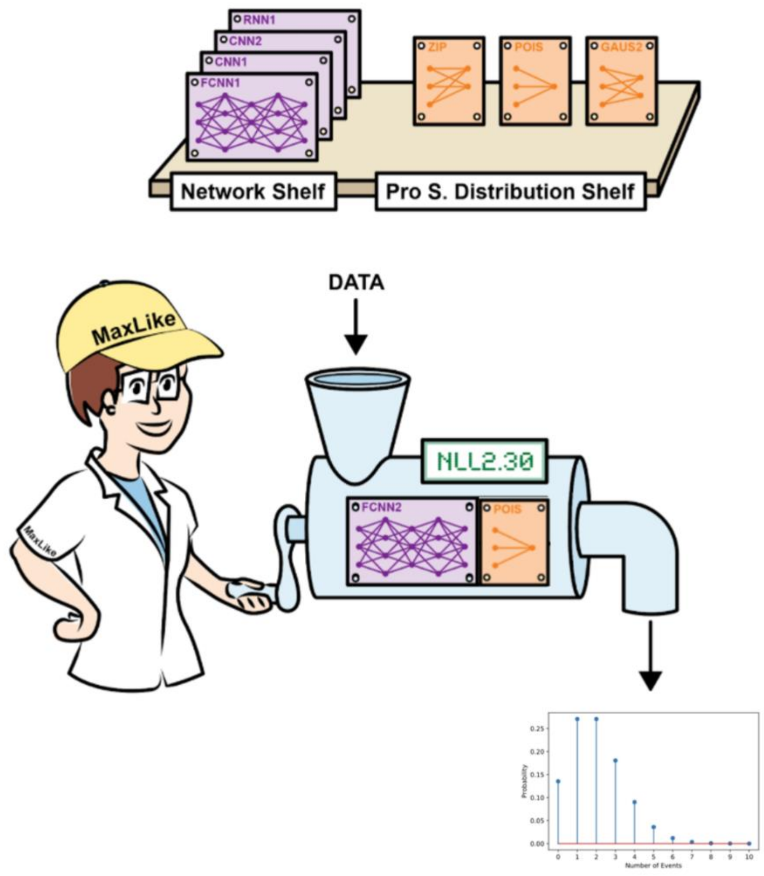
Institut für Datenanalyse und Prozessdesign
Zürcher Hochschule für Angewandte Wissenschaften

Part I: Probablistic models with flexible CPDs

Winterthur, 24. March. 2020

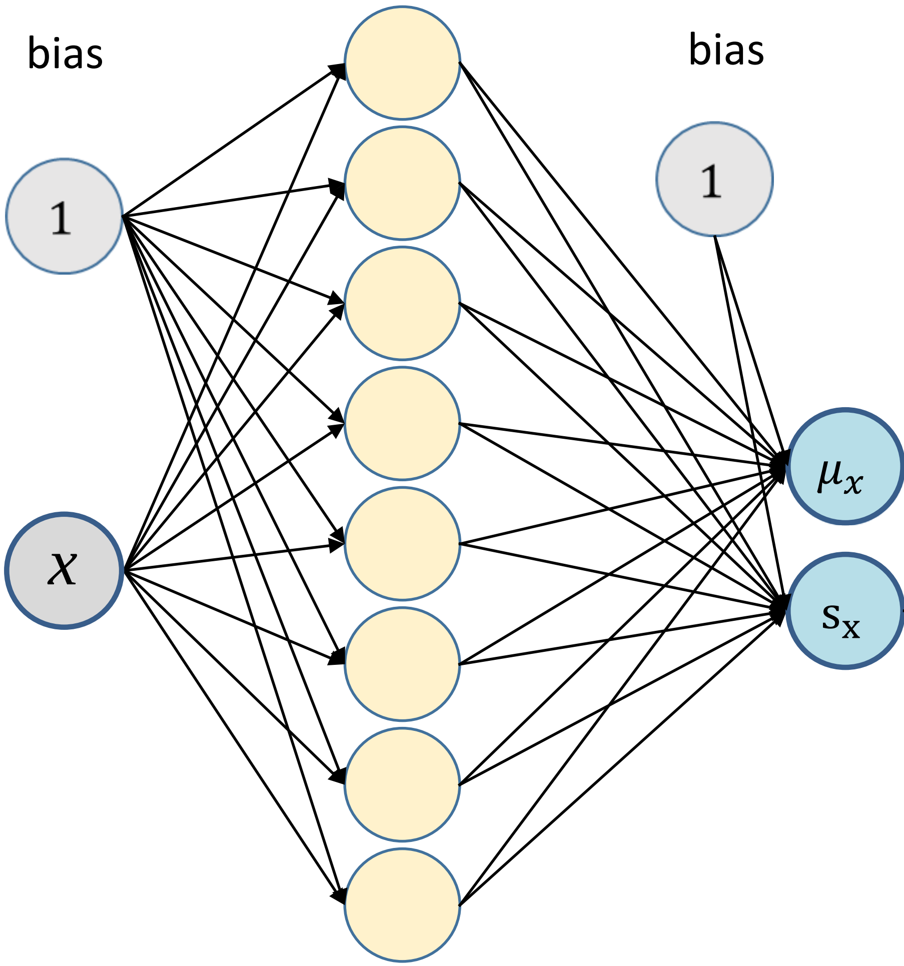
We have a flexible tool where the choice of the architecture and the choice of the outcome distribution is independent

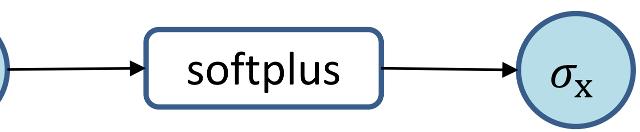


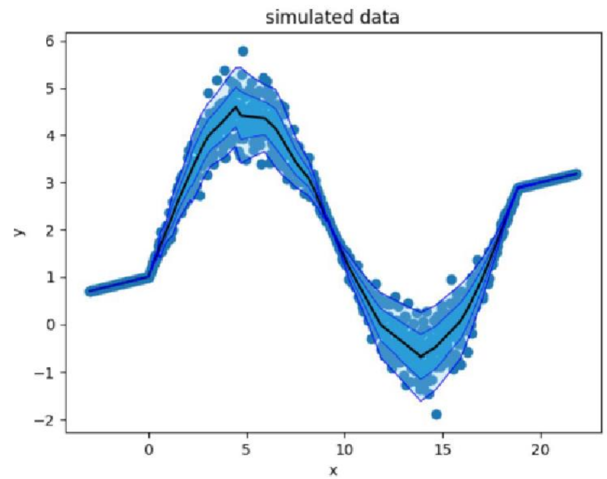


Modeling continuous data continued

Modeling a flexible variance



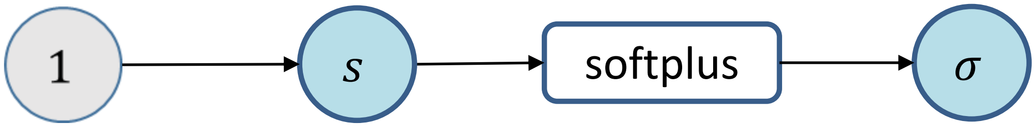


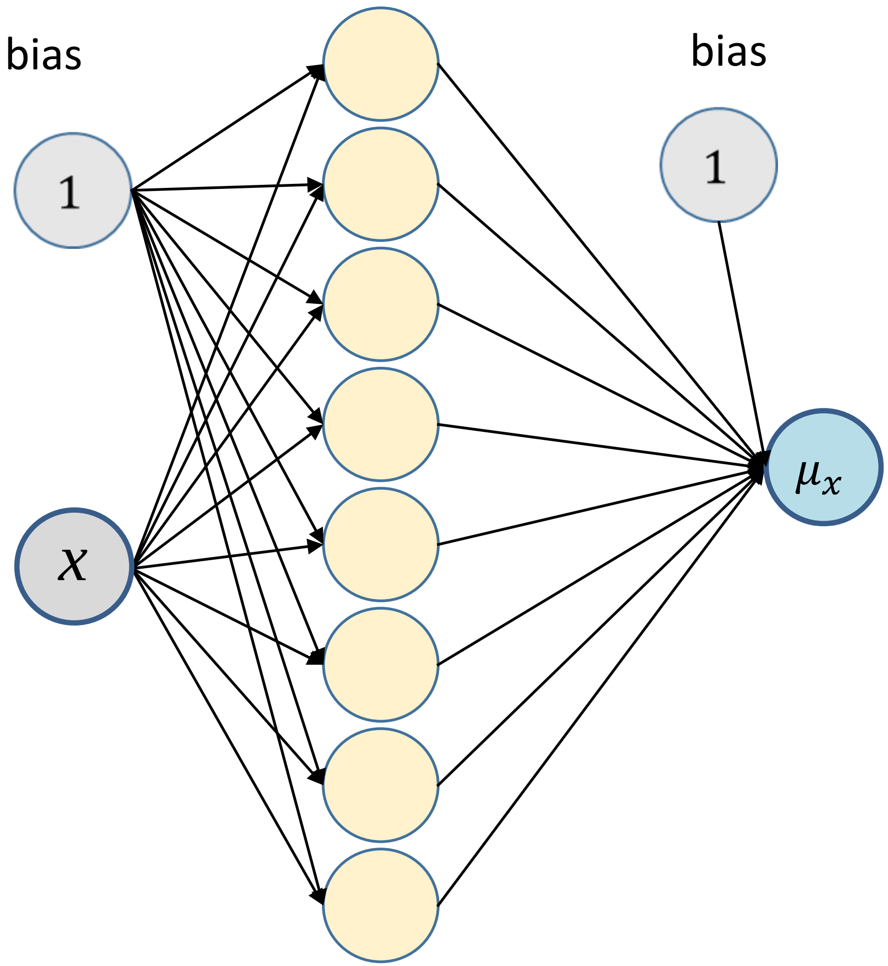


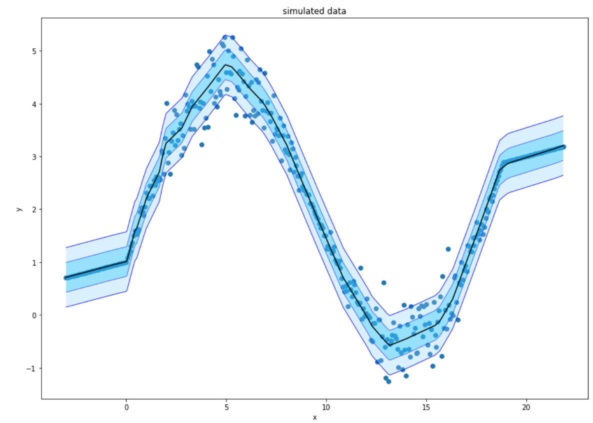
Minimize the negative log-likelihood (NLL):

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

Modeling a constant variance







Minimize the negative log-likelihood (NLL):

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

Modeling count data continued

The camper example – tabular data modeled via fcNN

N=250 groups visiting a national park

Y=count: number of fishes cought

X1=persons: number of persons in group

X2=child: number of children in the group

X3=bait: indicates of life bait was used

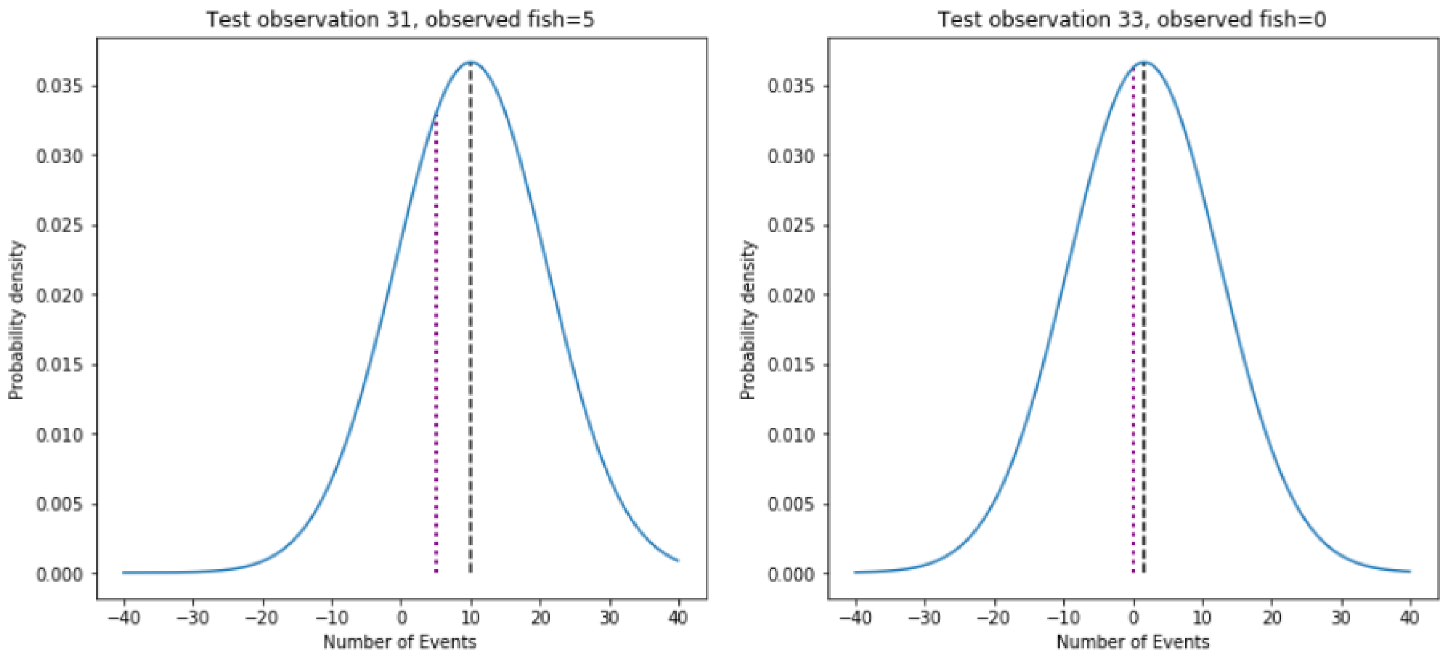
X4=camper: indicates if camper is brought

Data: https://stats.idre.ucla.edu/r/dae/zip

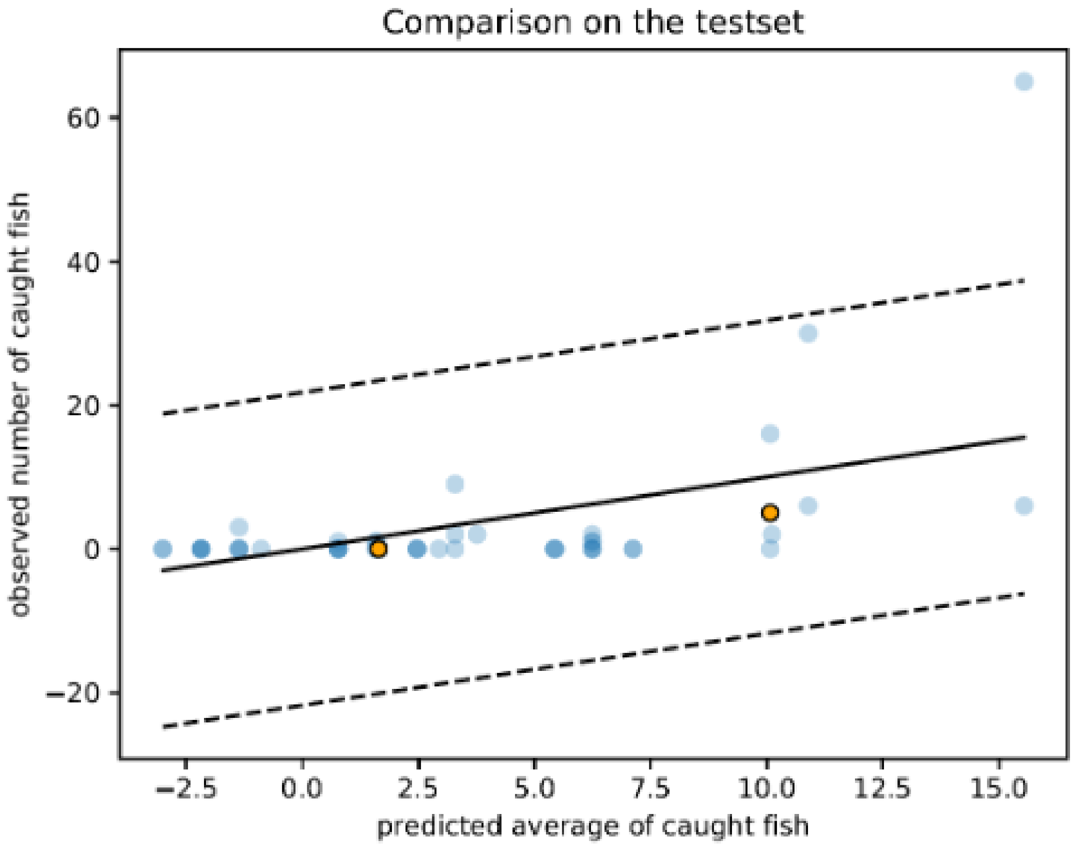
Model 1: linear regression, predicted CPDs for test observations

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.



Model 1: linear regression, visualize the CPDs by quantiles

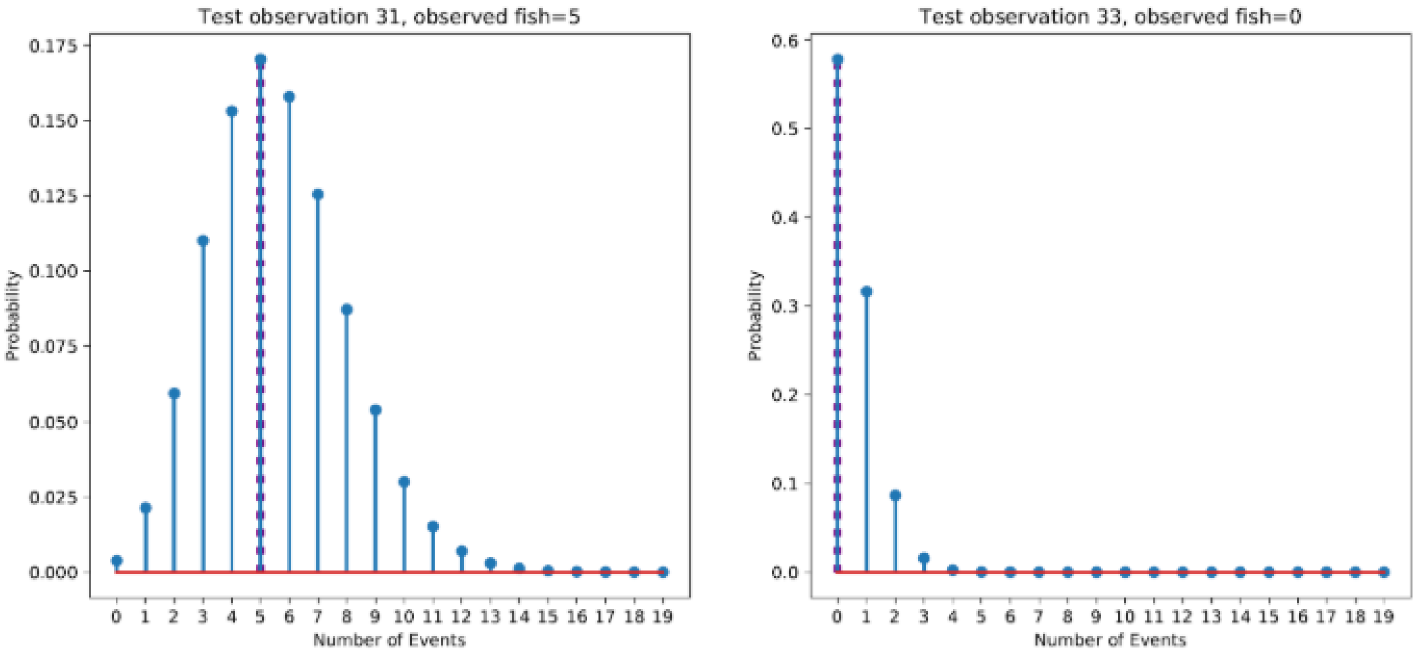


The mean of the CPD is depicted by the solid lines.
The dashed lines represent the 0.025 and 0.975 quantiles, yielding the borders of a 95% prediction interval.
Note that different combinations of predictor values can yield the same parameters of the CPD.

Model 2: Poisson regression, predicted CPDs for test observations

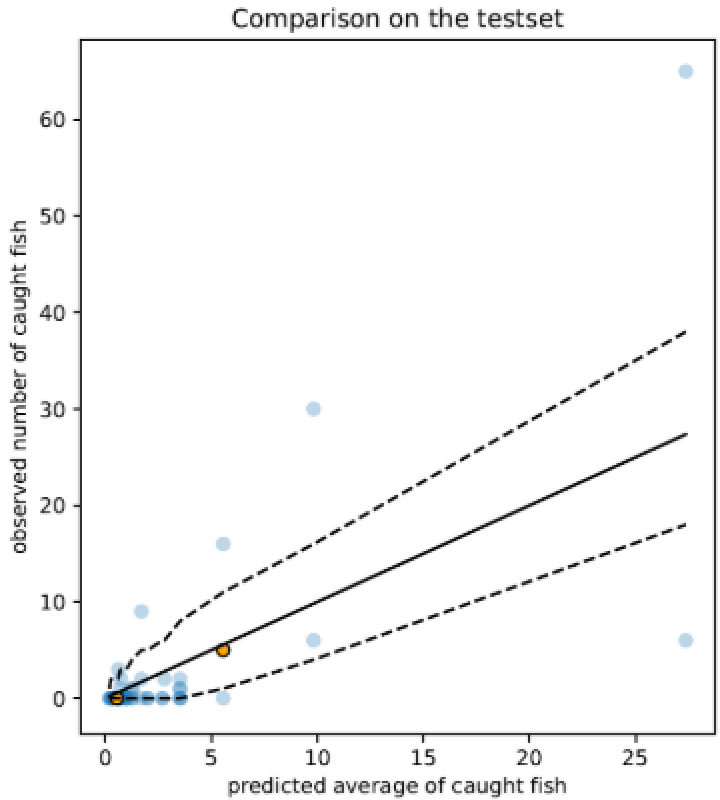
Predict CPD for outcome in test data:

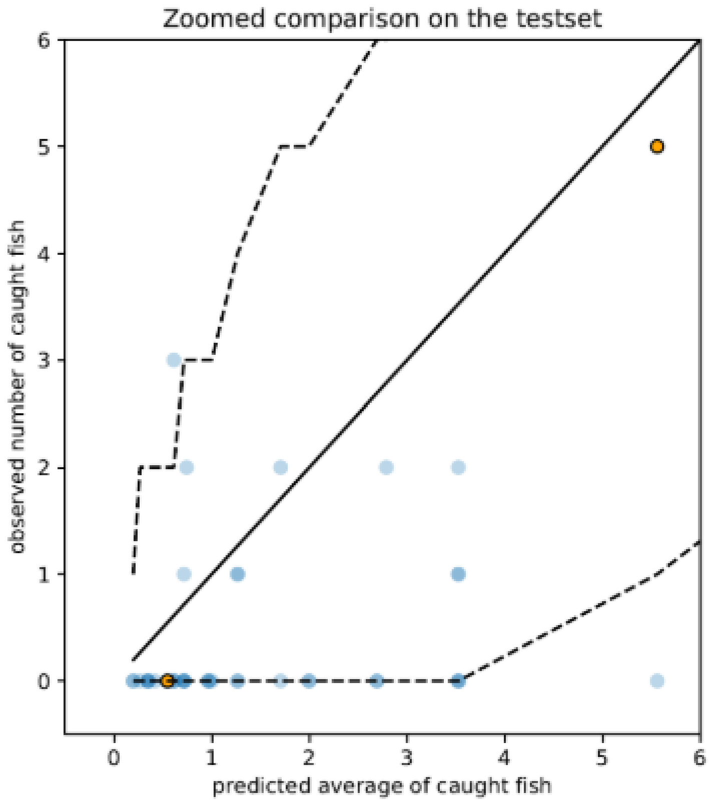
Group 31 used livebait, had a camper and were 4 persons with one child. Y=5 fish.
Group 33 used livebait, didn't have a camper and were 4 persons with two childern. Y=0 fish.



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

Model 2: Poisson regression, visualize the CPDs by quantiles





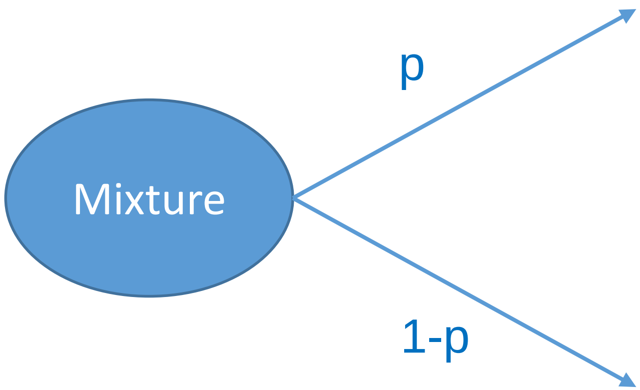
The mean of the CPD is depicted by the solid lines.
The dashed lines represent the 0.025 and 0.975 quantiles, yielding the borders of a 95% prediction interval.
Note that different combinations of predictor values can yield the same parameters of the CPD.

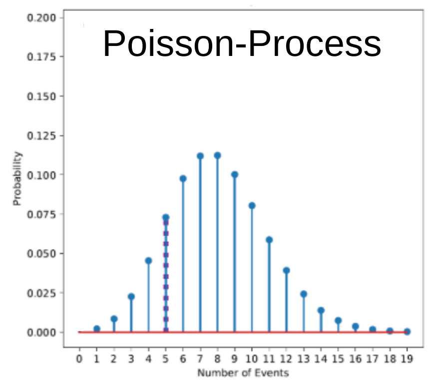
Modeling count data:

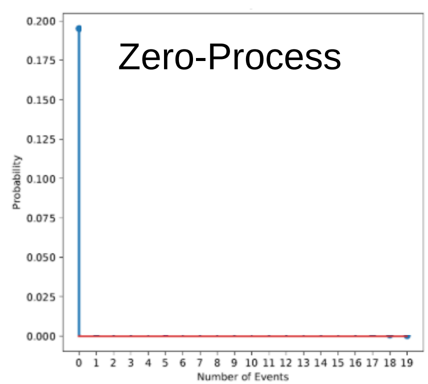
M3: ZIP regression

Zero-Inflated Poisson (ZIP) as Mixture Process

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



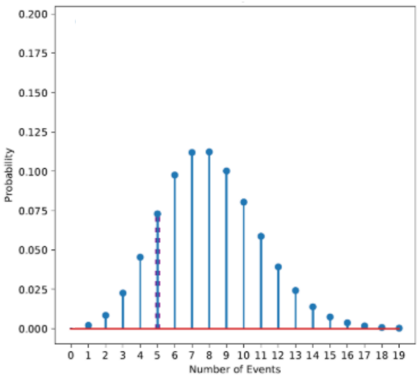




Zero-Inflated Poisson (ZIP) can be seen as Mixture Distribution

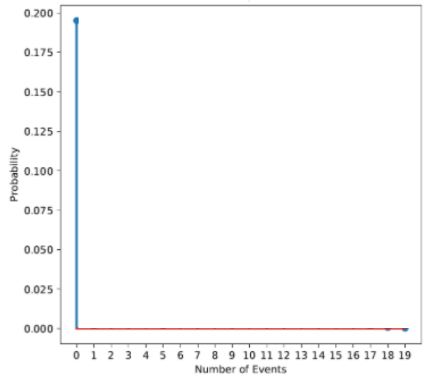
Poisson-Process

$p\ \cdot$



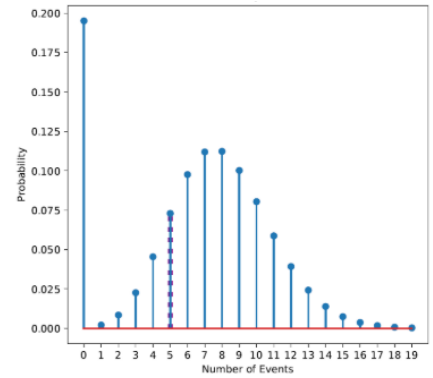
Zero-Process

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



Zero-inflated Poisson

$=$



Custom distribution for a ZIP distribution

```
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
        ])
```

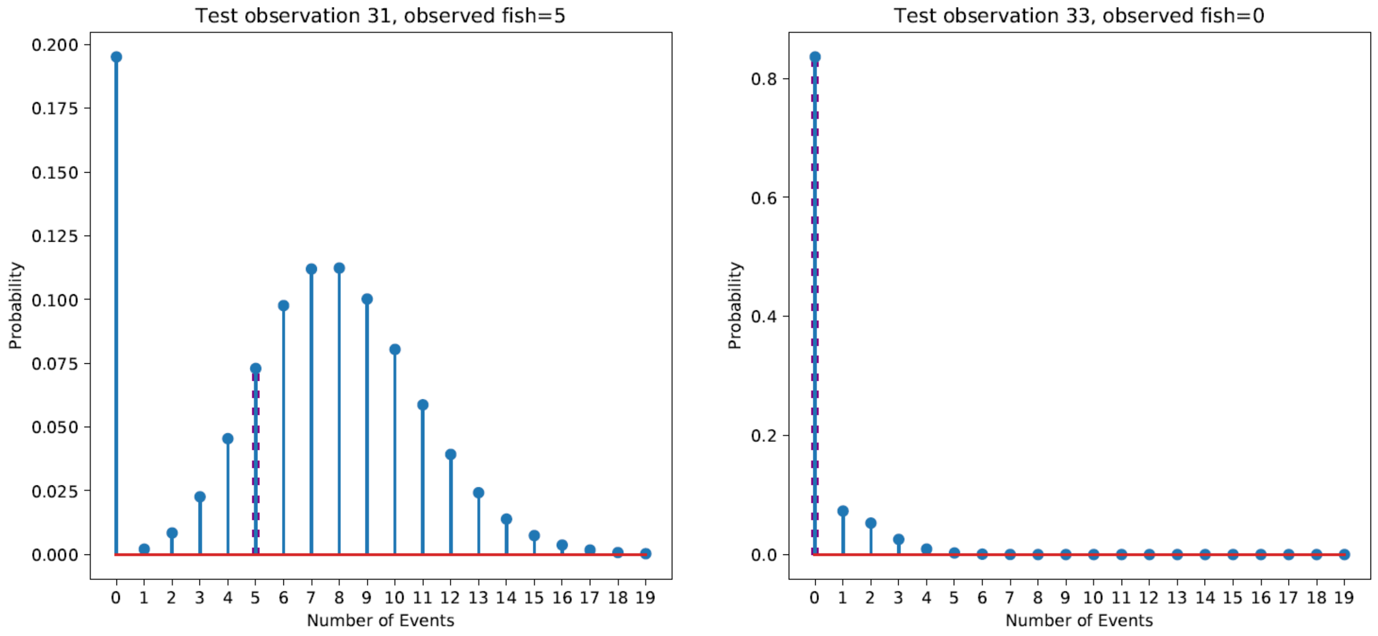
Model 3: Zero-Inflated Poisson regression via NNs in keras

```
## 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

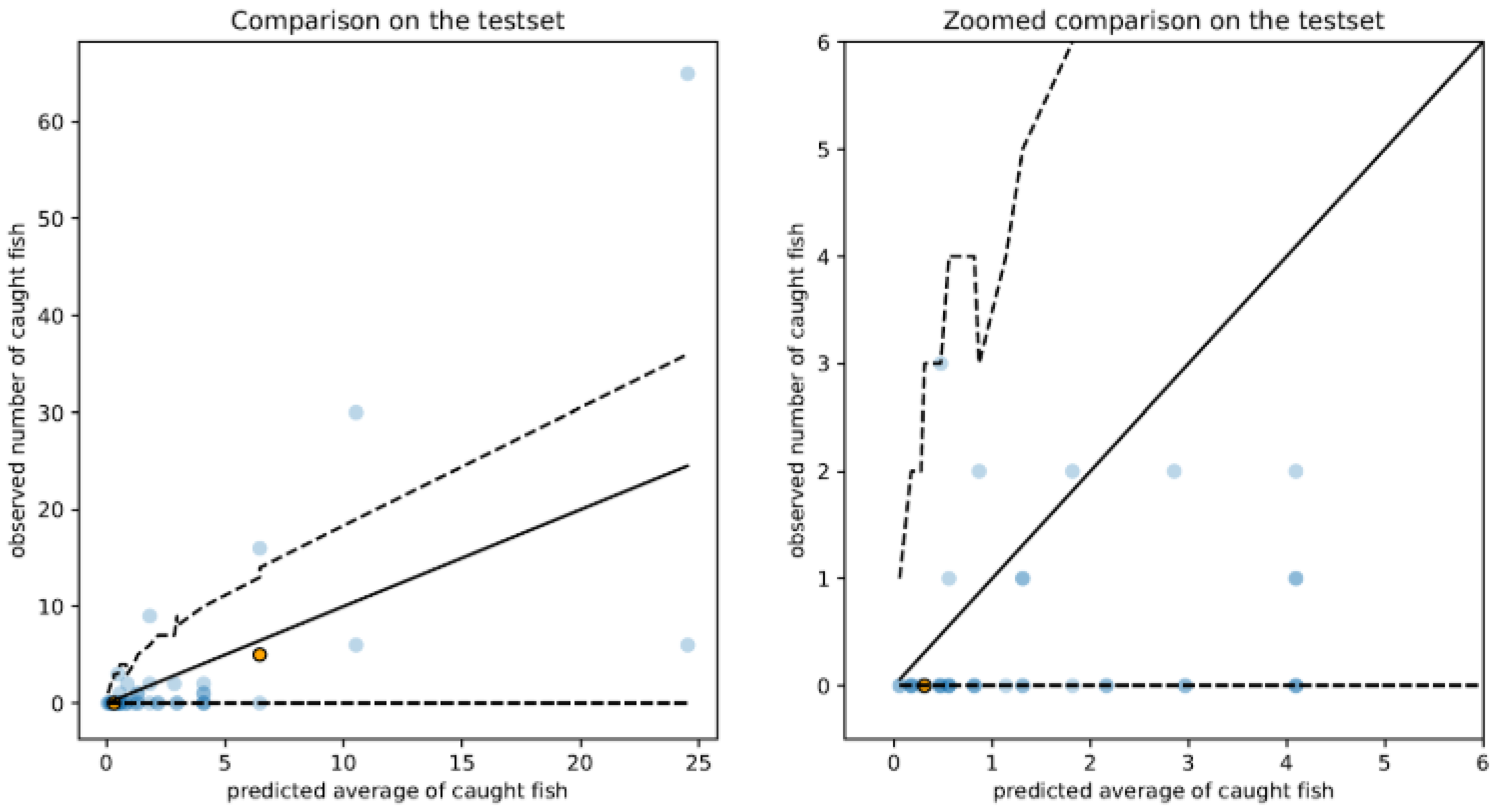
Predict CPD for outcome in test data:

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.



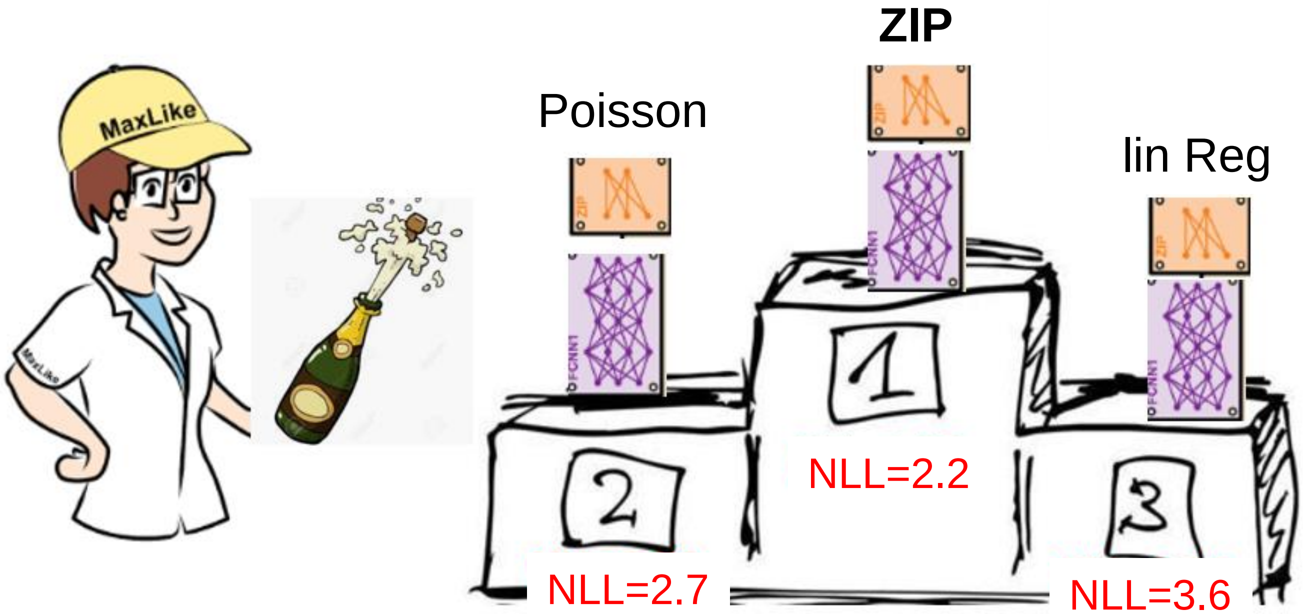
What is the likelihood of the observed outcome in test obs 31 and 33?

Model 3: ZIP regression, visualize the CPDs by quantiles



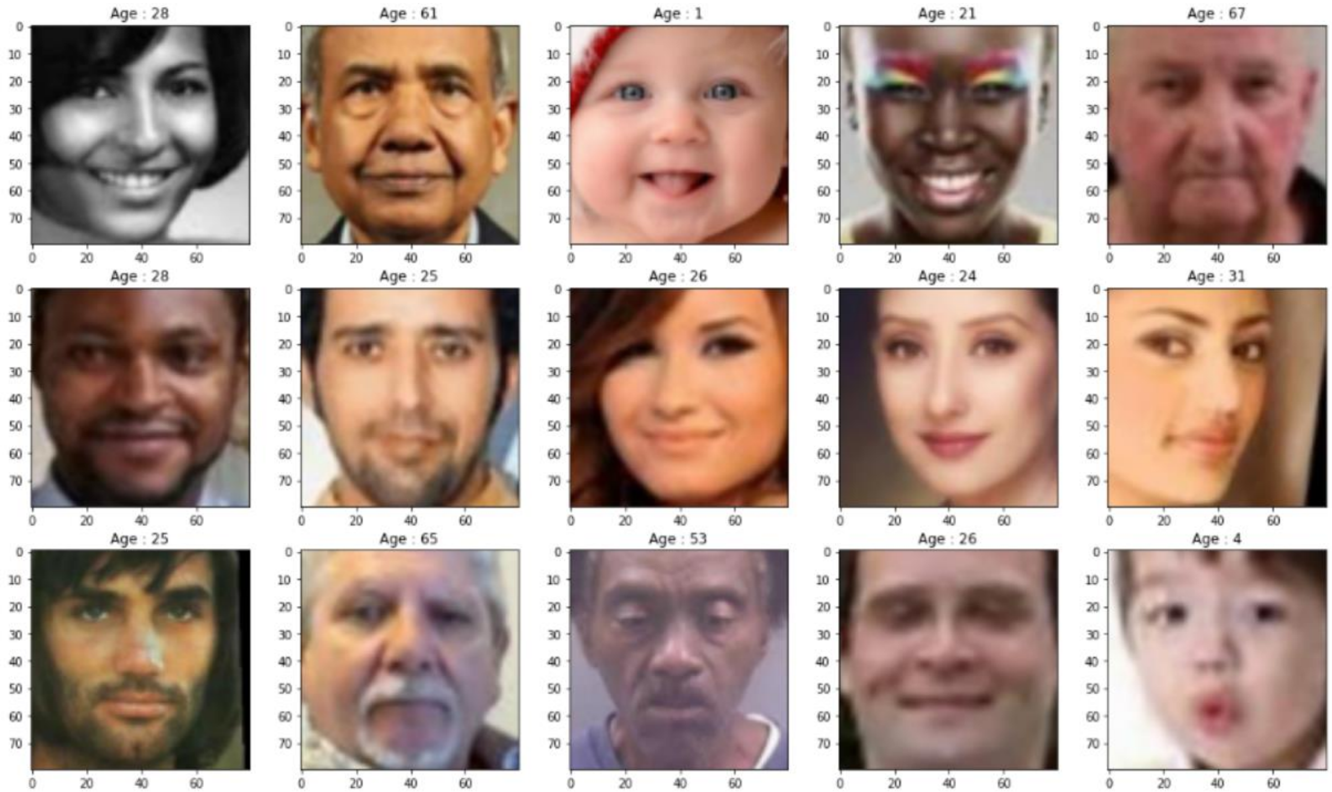
The mean of the CPD is depicted by the solid lines.
The dashed lines represent the 0.025 and 0.975 quantiles, yielding the borders of a 95% prediction interval.
Note that different combinations of predictor values can yield the same parameters of the CPD.

Validation NLL allows to rank different probabilistic models



Probabilistic models with complex input data

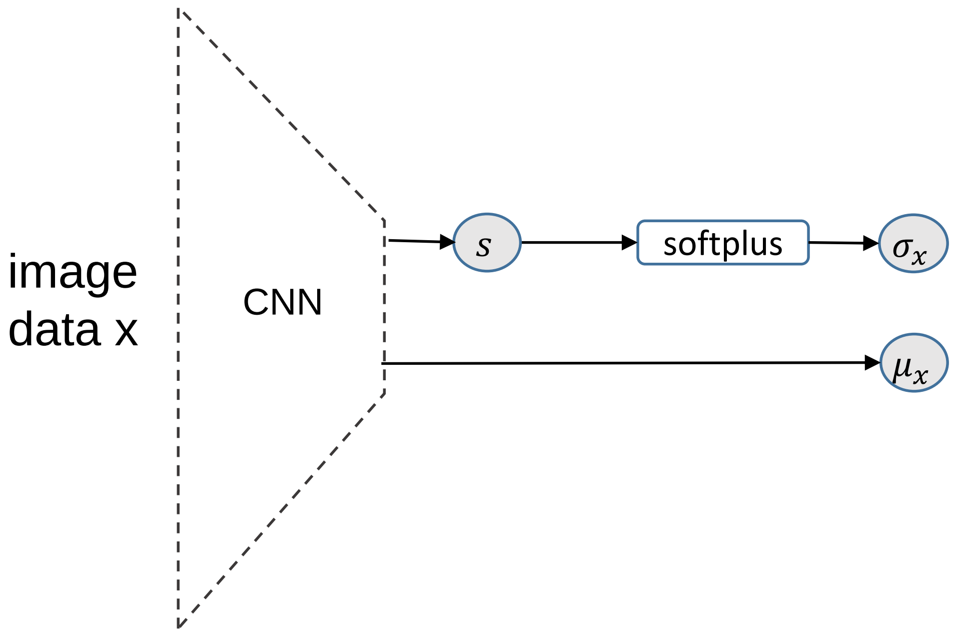
The UTK face data – face image data with known age



Data: https://stats.idre.ucla.edu/r/dae/zip

UTKFace data set containing N= 23′708 images of cropped faces of humans with known age ranging from 1 to 116 years.

Modeling a flexible variance



Minimize the negative log-likelihood (NLL):

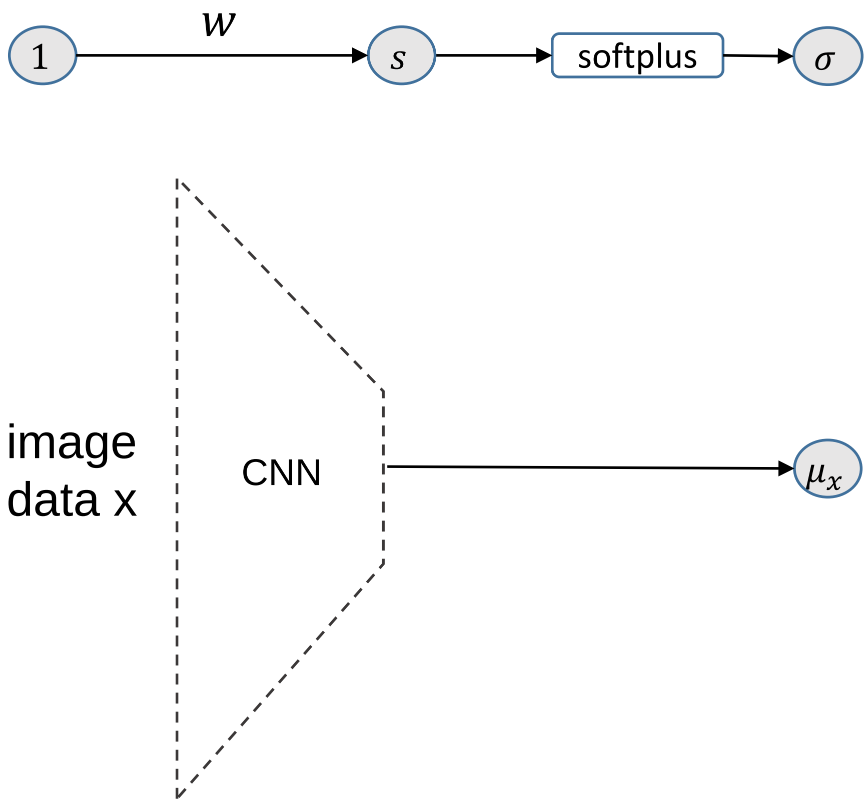
$$\hat{\mathbf{w}}_{\mathrm{ML}} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) + \frac{\left(y_i - \mu_{x_i}\right)^2}{2\sigma^2}$$

CNNs for modeling Gaussian CPDs

```
def NLL(y, distr):
  return -distr.log_prob(y)

def my_dist(params):
  return tfd.Normal(loc=params[:,0:1], scale=1e-3 + tf.math.softplus(0.05 * params[:,1:2]))# both paramete
rs are learnable

inputs = Input(shape=(80,80,3))
x = Convolution2D(16,kernel_size,padding='same',activation="relu")(inputs)
x = Convolution2D(16,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Flatten()(x)
x = Dense(500,activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(50,activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(2)(x)
dist = tfp.layers.DistributionLambda(my_dist)(x)

model_flex = Model(inputs=inputs, outputs=dist)
model_flex.compile(tf.keras.optimizers.Adam(), loss=NLL)
```

We model both parameters (μ_x, σ_x) of a Gaussian CPD $N(\mu_x, \sigma_x)$
→ More flexible than in classical regression where $\sigma = constant$

Modeling a constant variance



Minimize the negative log-likelihood (NLL):

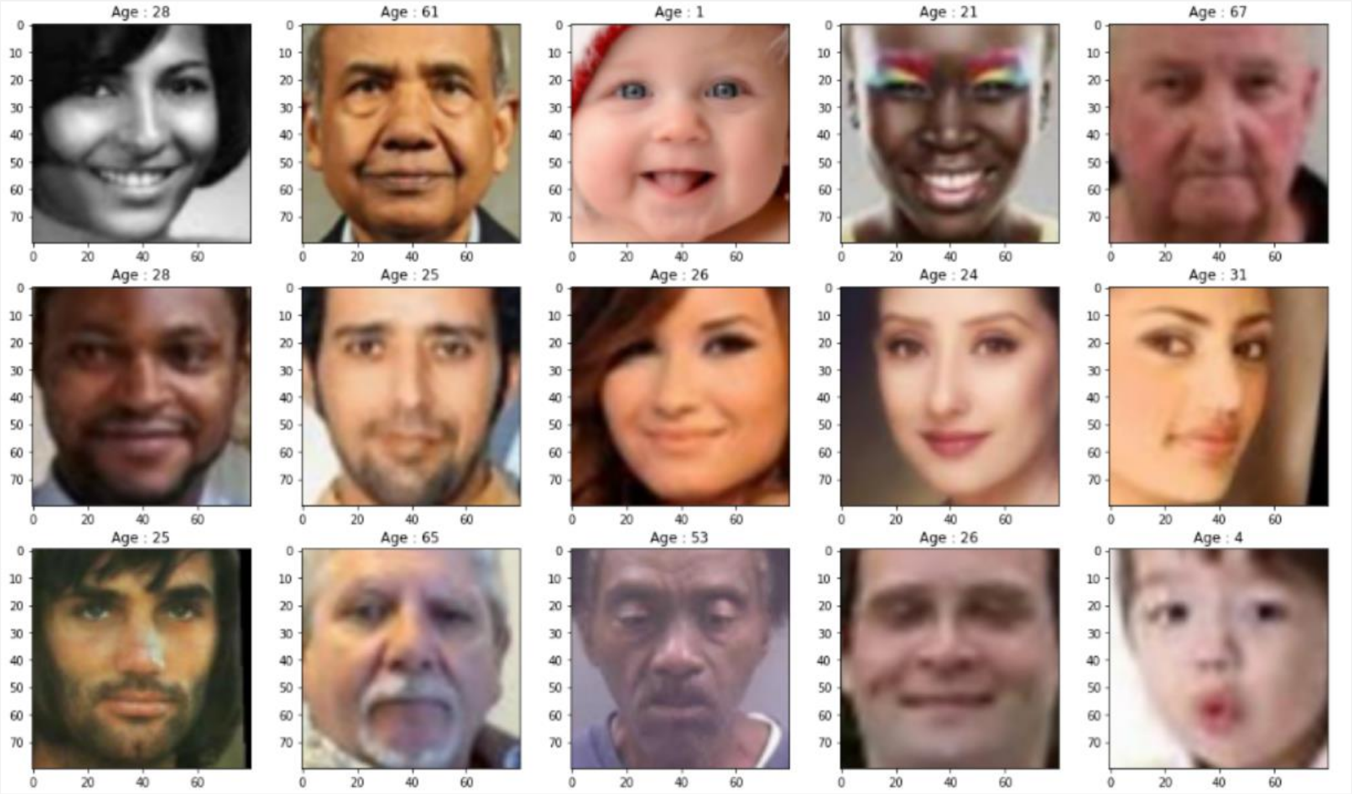
$$\hat{\mathbf{w}}_{\mathrm{ML}} = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma^{2}}}\right) + \frac{\left(y_i - \mu_{x_i}\right)^2}{2\sigma^2}$$

CNNs for modeling Gaussian CPDs

```
def NLL(y, distr):
  return -distr.log_prob(y)

def my_dist(params):
  return tfd.Normal(loc=params[:,0:1], scale=1e-3 + tf.math.softplus(0.05 * params[:,1:2]))# both paramete
rs are learnable

input1 = Input(shape=(80,80,3))
input2 = Input(shape=(1,))
x = Convolution2D(16,kernel_size,padding='same',activation="relu")(input1)
x = Convolution2D(16,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = Convolution2D(32,kernel_size,padding='same',activation="relu")(x)
x = MaxPooling2D(pool_size=pool_size)(x)

x = Flatten()(x)
x = Dense(500,activation="relu")(x)
x = Dropout(0.3)(x)
x = Dense(50,activation="relu")(x)
x = Dropout(0.3)(x)
out1 = Dense(1)(x)
out2 = Dense(1)(input2)
params = Concatenate()([out1,out2])
dist = tfp.layers.DistributionLambda(my_dist)(params) #

model_const_sd = Model(inputs=[input1,input2], outputs=dist) ## use a trick with two inputs, input2 is jus
t ones
model_const_sd.compile(tf.keras.optimizers.Adam(), loss=NLL)
```

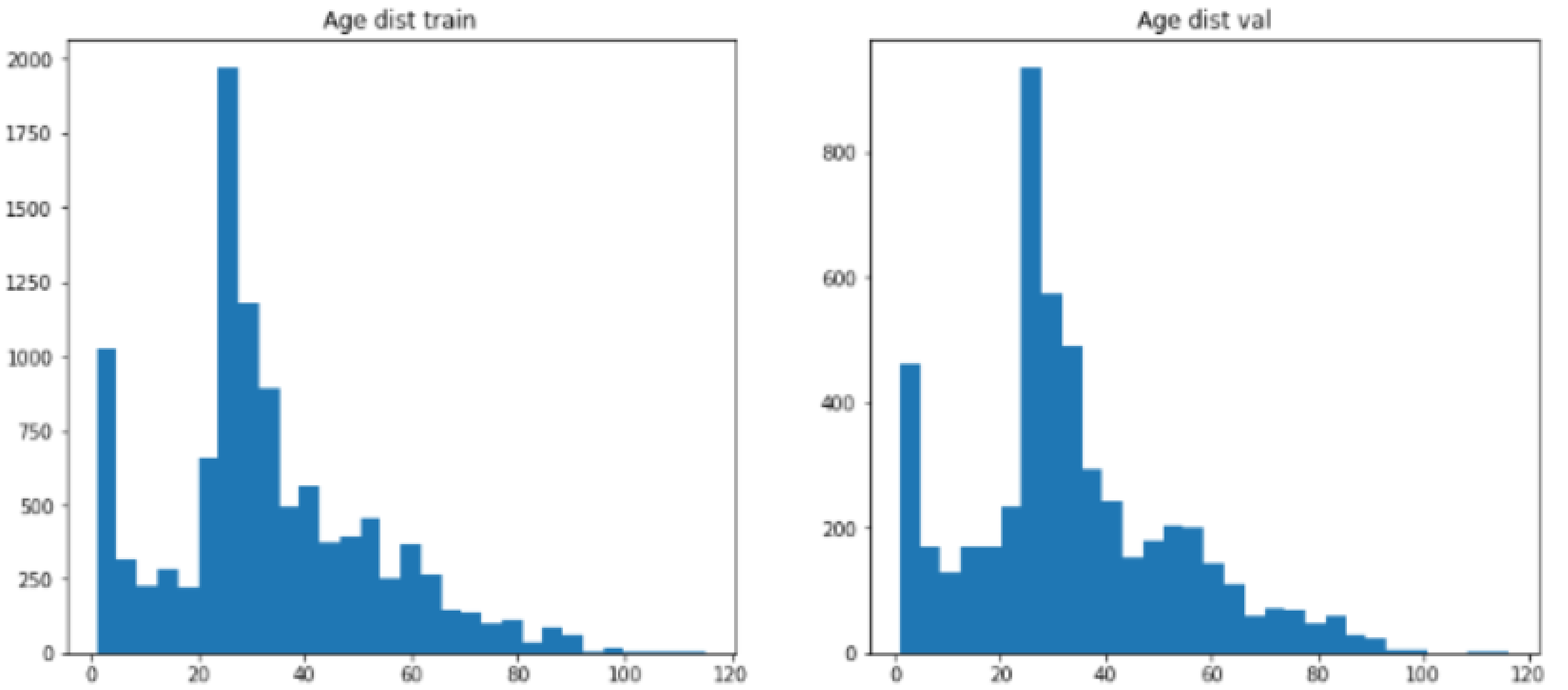
We model both parameters (μ, σ) of a Gaussian CPD $\mathrm{N}(\mu_x, \sigma)$ → But assume a constant variance

Excercise



Check out the models for age prediction with flexible and constant variance and try to understand the code and to answer the questions.

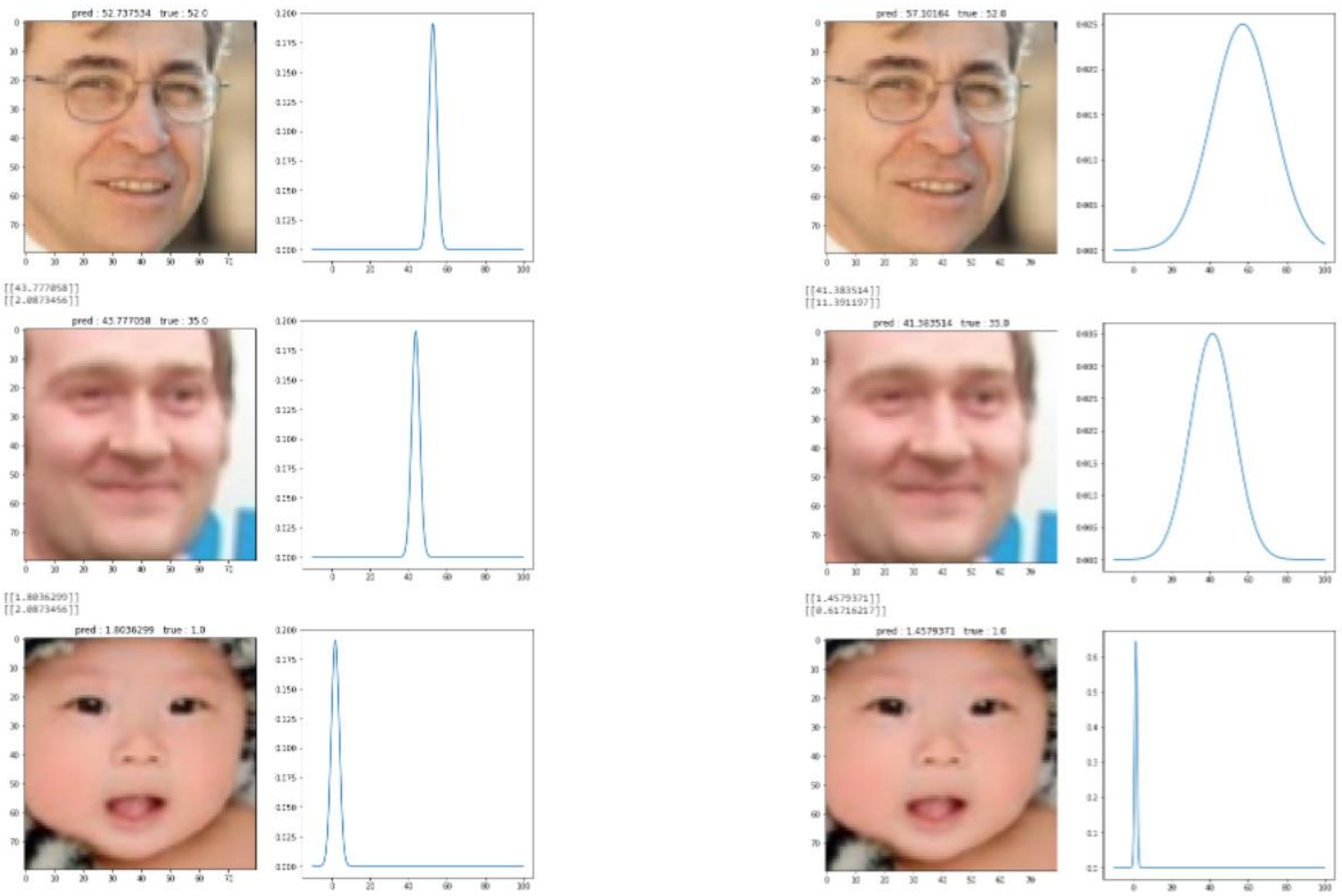
Age distribution



We have a lot of small children in the data set, for whom the age estimation is probably not so difficult.

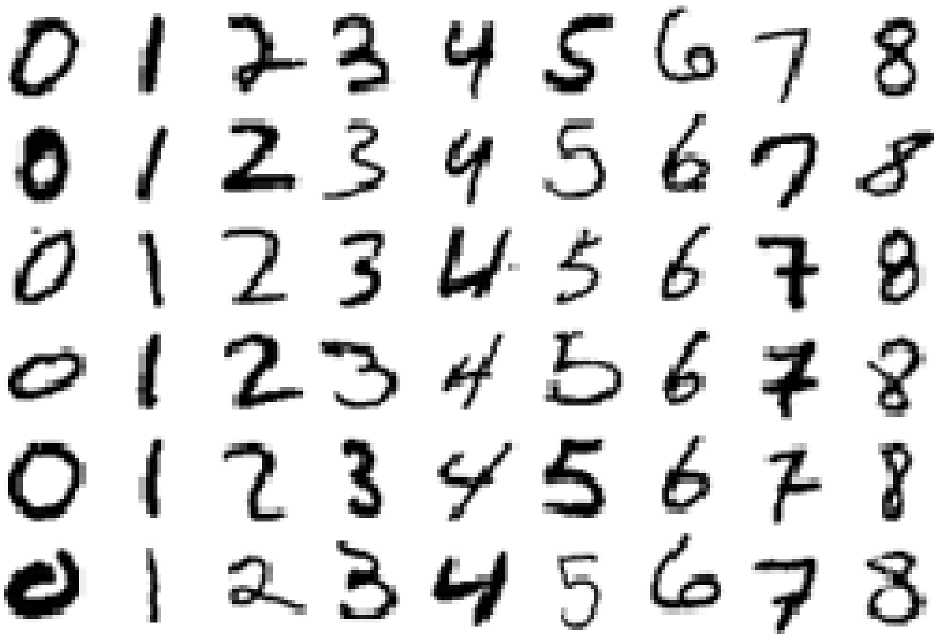
Resulting age CPDs

Constant variance flexible variance



In case of a flexible variance, a broad predicted Gaussian CPD does indicate high uncertainty about the age.

The MNIST data set – images of handwritten digits



The MNIST data set containing N= 60’000 images of handwritten digits of the ten classes 0, 1, 2, 3, 4, 5, 6,7, 8, 9,

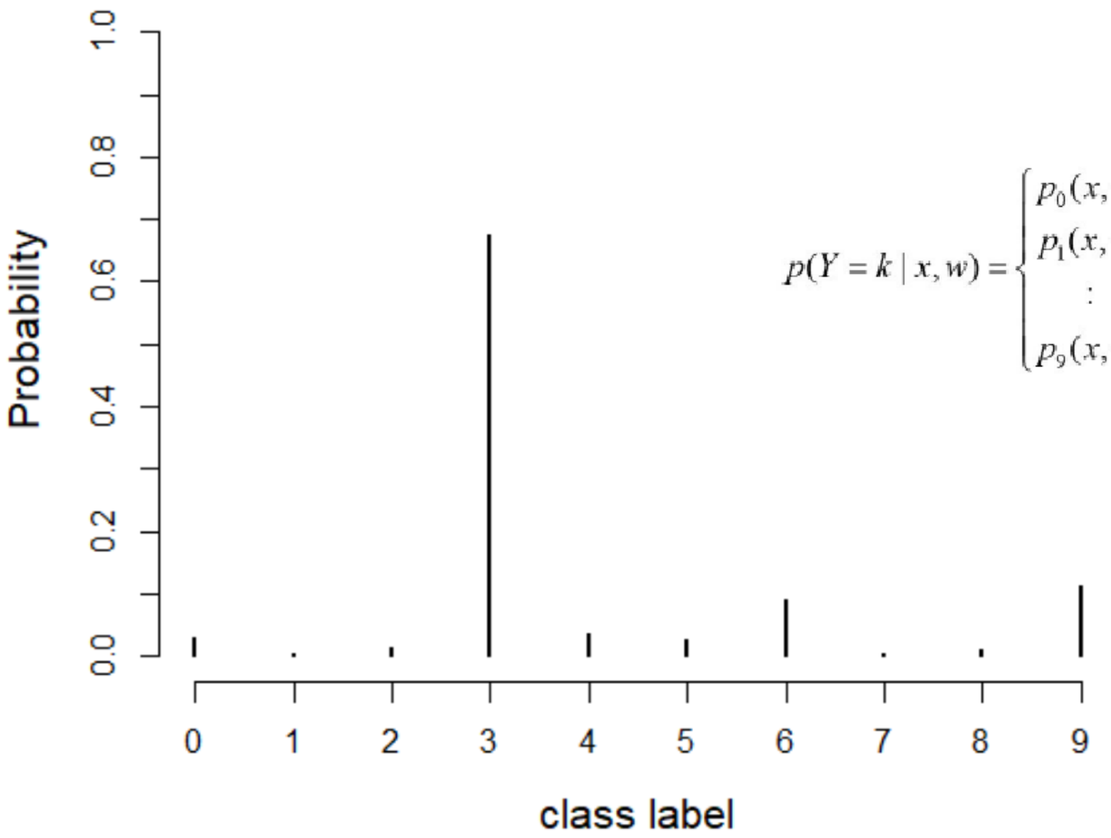
CNNs for modeling Gaussian CPDs

```
# here we define hyperparameter of the CNN
batch_size = 128
nb_classes = 10
img_rows, img_cols = 28, 28
kernel_size = (3, 3)
input_shape = (img_rows, img_cols, 1)
pool_size = (2, 2)
```

```
# define CNN with 2 convolution blocks and 2 fully connected layers
model = Sequential()

model.add(Convolution2D(8,kernel_size,padding='same',input_shape=input_shape))
model.add(Activation('relu'))
model.add(Convolution2D(8, kernel_size,padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))

model.add(Convolution2D(16, kernel_size,padding='same'))
model.add(Activation('relu'))
model.add(Convolution2D(16,kernel_size,padding='same'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=pool_size))

model.add(Flatten())
model.add(Dense(40))
model.add(Activation('relu'))
model.add(Dense(nb_classes))
model.add(Activation('softmax'))

# compile model and intitialize weights
model.compile(loss='categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

We model 10 parameters $(p_0, p_1, \ldots p_9)$ of a Multinomial CPD (it could also be done with 9 parameters)
→ Most flexible CPD for an outcome with 10 possible values

We predict for each image a multinomial CPD



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

The multinomial distribution is especially flexible because it has as many parameters as possible values (or actually one parameter less, because probabilities need to sum up to one).