WBL Deep Learning:: Lecture 4

Beate Sick, Oliver Dürr

Deep Learning as probabilistic modeling

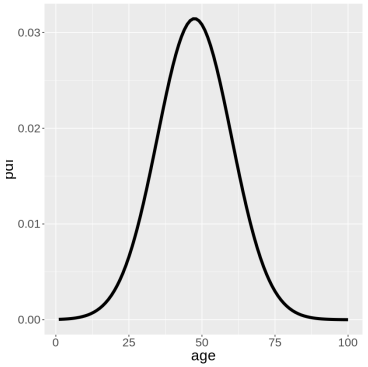
Zürich, 9/19/2022 (possible start) or 9/26/2022

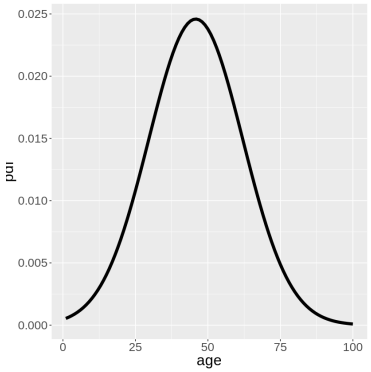
At the end of today [probabilistic age estimation]

x

NN

$p(y|x)$





Outline: Probabilistic Modeling (Likelihood based)

- 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?
  → use NLL as loss function
- Examples using TensorFlow Probability
  - Regression via fcNN (CPD: $N(\mu, \sigma)$)
  - Count data via fcNN (CPD: Poisson, NB, ZIP)
  - Regression for Age estimation via CNN (CPD: $N(\mu, \sigma)$)

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

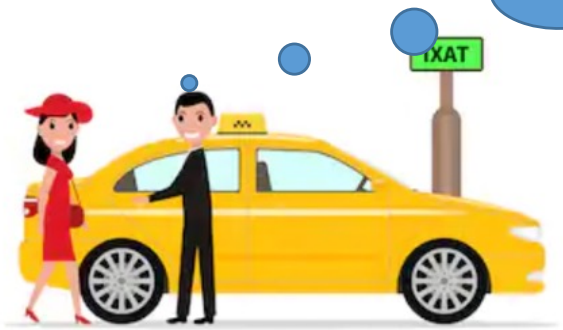
Practical reasons:

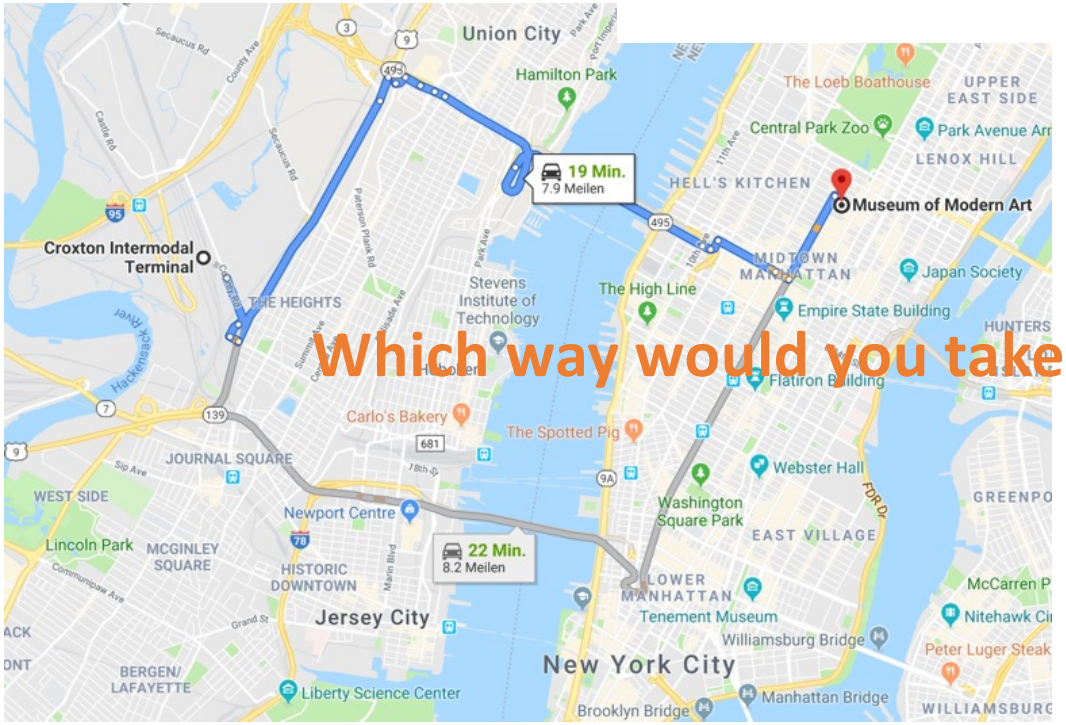
- We often want to optimize expected costs which requires CPD for computing.
- **They come for free in most DL problems**
- Choosing the right probabilistic model enhances existing DL architectures
  - Count data (using Poisson, NB, or Poisson)
  - Parallel Wavenet, PixelCNN++ (using mixtures of logistics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

Probabilistic travel time prediction (expected cost)

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

Let's use my probabilistic travel time gadget!



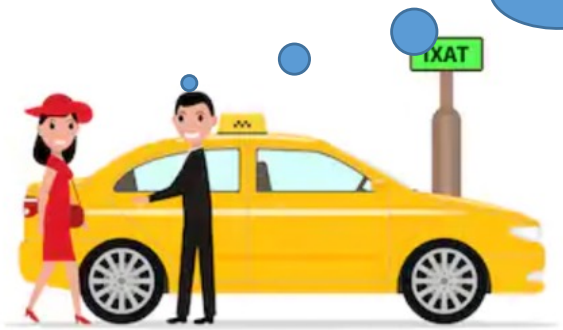


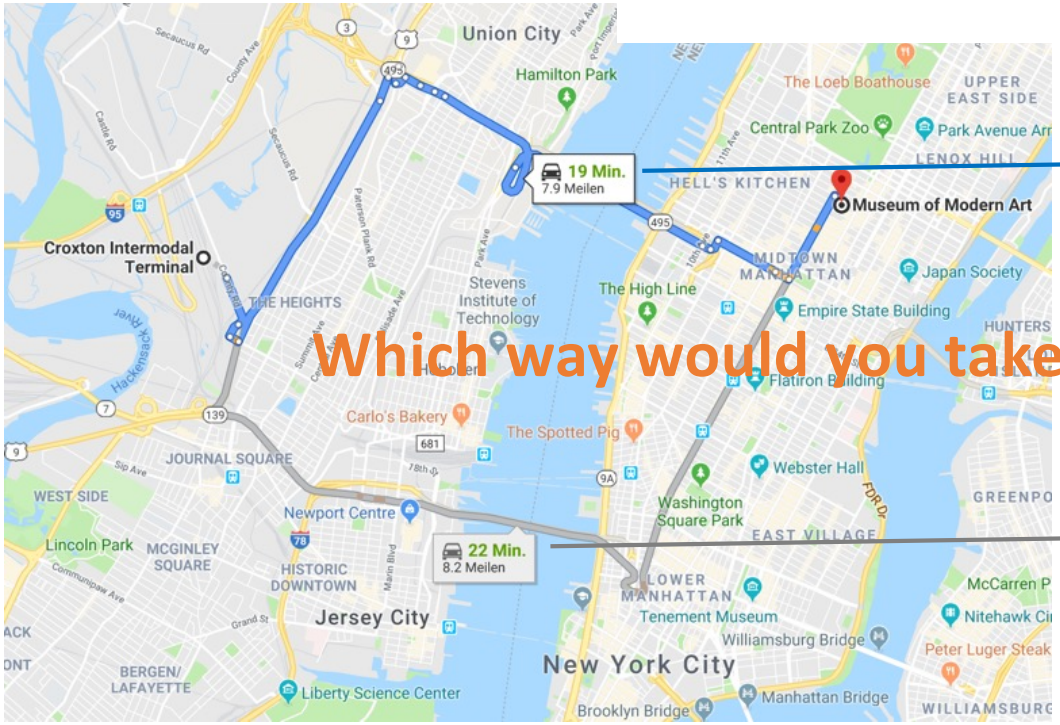
Which way would you take?

Probabilistic travel time prediction (expected cost)

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

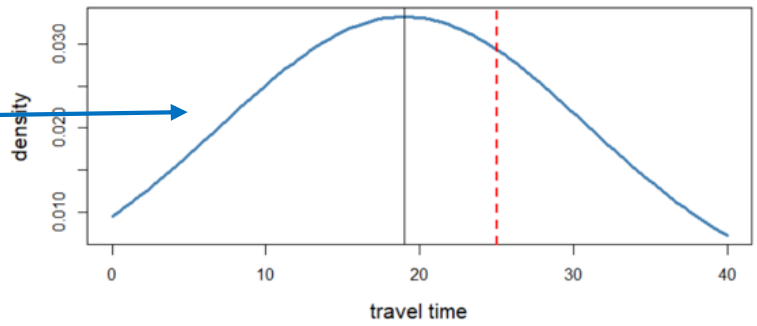
Let's use my probabilistic travel time gadget!



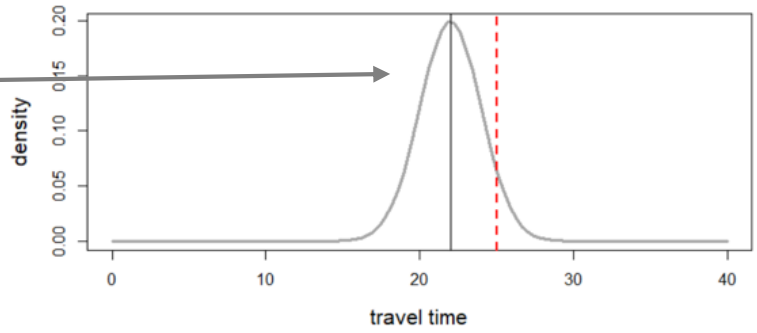


Which way would you take?

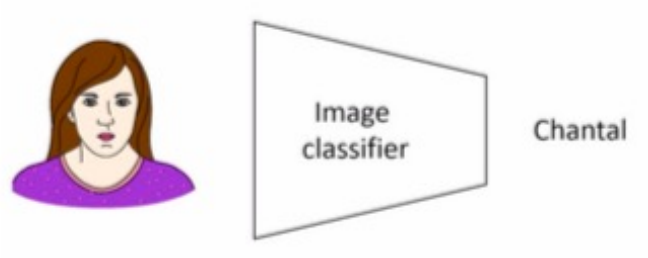
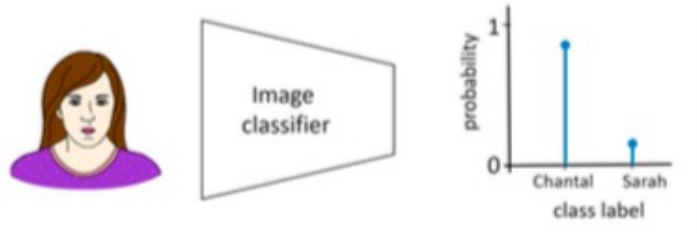
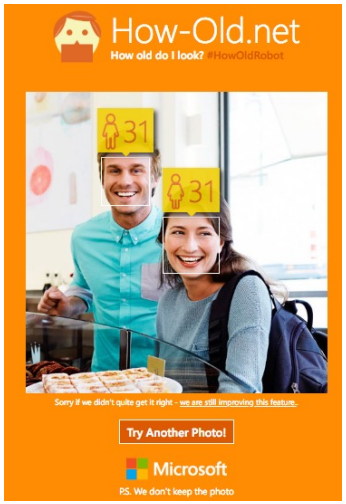
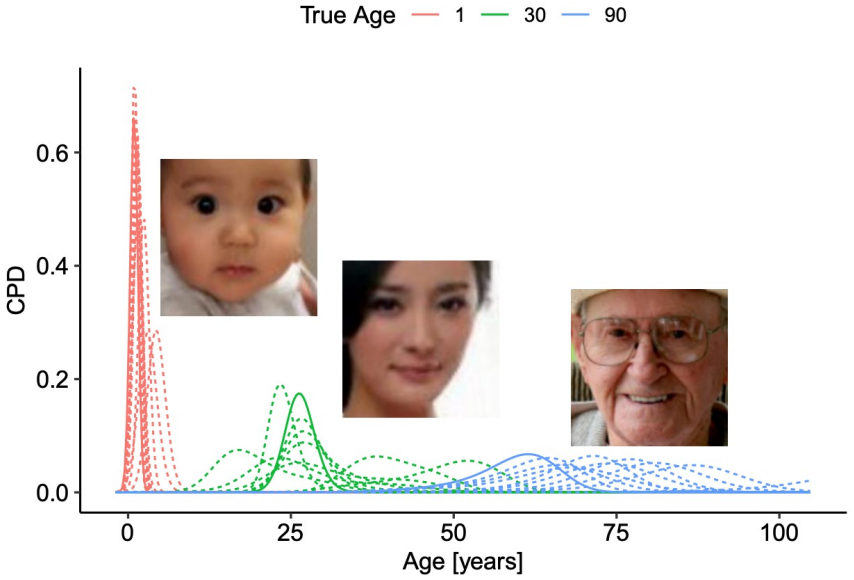
Chance to get tip: 69%



Chance to get tip: 93%



Recap Deterministic / Probabilistic

	Deterministic	Probabilistic
"Classification"		
"Regression"		

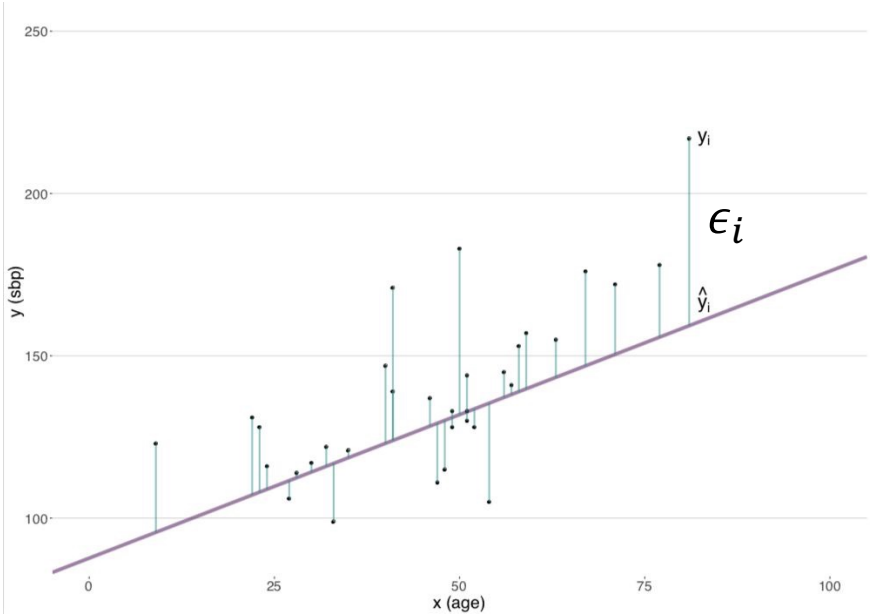
Conditional probability distribution (CPD)

$p(y|x)$

Probabilistic models

Traditional Formulation of Regression

- Suppose we observe y_i and $x_i = (\mathrm{x_{i1}}, \ldots x_{ip})$ for $i = 1, \ldots, n$
- We can model the relationship as
  - $Y_i = f(X_i) + \epsilon_i$
  - Where ϵ_i is a random error with mean zero.



The ϵ_i are the difference to the "true values".

This yields

$$E[Y_i] = E[Y_i|X_i] = f(X_i)$$ (expression which is useful again)

Do not eat this poisonous fruit

Critique on the formulation

Xiao-Li Meng
@XiaoLiMeng1

The term "regression" reflects statisticians' modesty, and perhaps also our regrets? We should not have started statistical modeling with regression, for it confuses probabilistic model fitting with deterministic line/curve fitting, building wrong intuitions for generations.

7:48 am · 1 Oct 2019 · Twitter for iPhone

Source: twitter.com/XiaoLiMeng1

Modern Formulation of regression (distributional)

- Suppose we observe y_i and $x_i = (x_{i1}, \dots, x_{ip})$ for $i = 1, \dots, n$
- We believe that there is a relationship between Y and at least one of the X's.
- We can model the relationship by an outcome distribution (dist) of Y depending on X:

$$\mathrm{Y}_i \sim \mathrm{dist}(z(x_i))$$

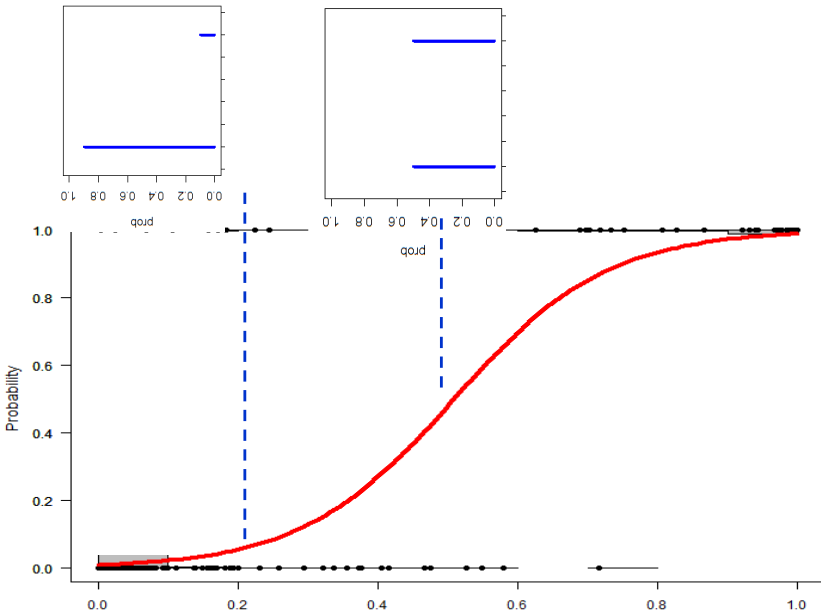
- Where z is an unknown function "controlling" the parameters of an outcome distribution dist.
- Examles
  - $dist = N\big(\mu = z_1(x), \sigma = z_2(x)\big)$
  - $dist = N(\mu = z\,(x), \sigma)$ σ fixed does not depend on x.
  - $dist = Exp\big(\lambda = z\ (x)\big)$

GLM examples

Logistic Regression

$$z(x) = a \cdot x + b$$
$$Y \sim Bern(p = \mathrm{sig}(z(x)))$$

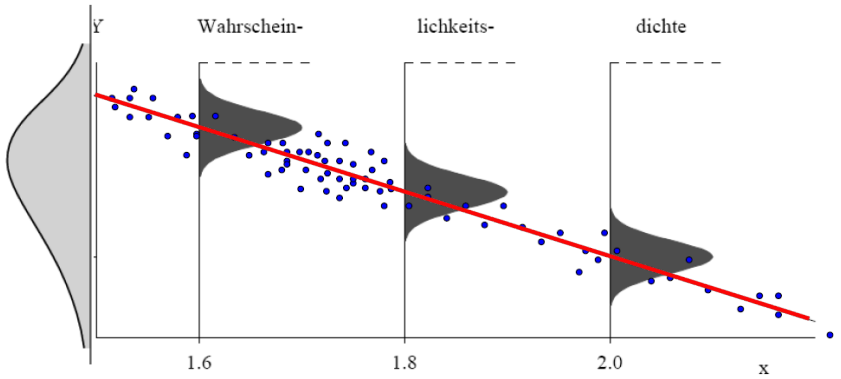
```
glm(y ~ ., binomial(logit))
```



Linear Regression

$$z(x) = a \cdot x + b$$
$$Y \sim Normal(\mu = z(x), \sigma = const)$$

```
glm(y ~ ., gaussian(identity))
```



Linear regression is confusingly special case (constant variance, identity link)

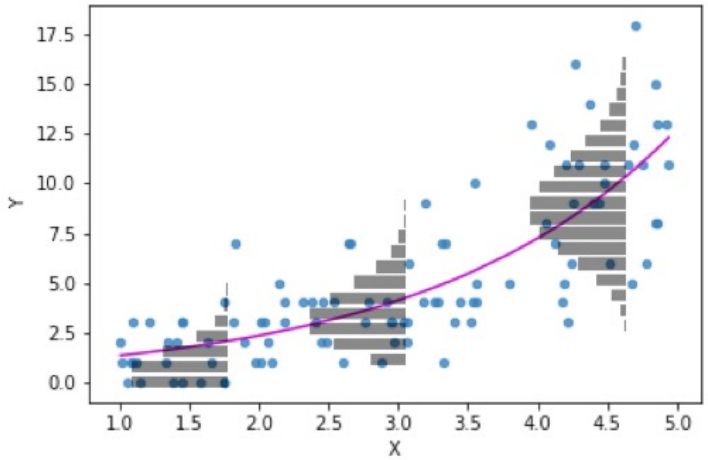
Right image adapted from: Stahel's book Statistische Datenanalyse

GLM examples

Poisson Regression

$$z(x) = a \cdot x + b$$
$$Y \sim Pois(rate = \mathrm{Exp}(z(x)))$$

```
glm(y ~ ., poisson(log))
```



GLM

1. Linear Predictor
   $z(x) = \eta(x) = \beta_0 + \sum \beta_i x_i = \beta x^T$
2. Inverse Link Function
   Some manipulation of $z(x)$ gives expectation of prob. distribution

Probabilistic Neural networks

- Replace linear predictor with (deep) NN
- Model not just expectation but all parameters of the CPD

Image credit: https://www.lexjansen.com/nesug/nesug10/sa/sa04.pdf

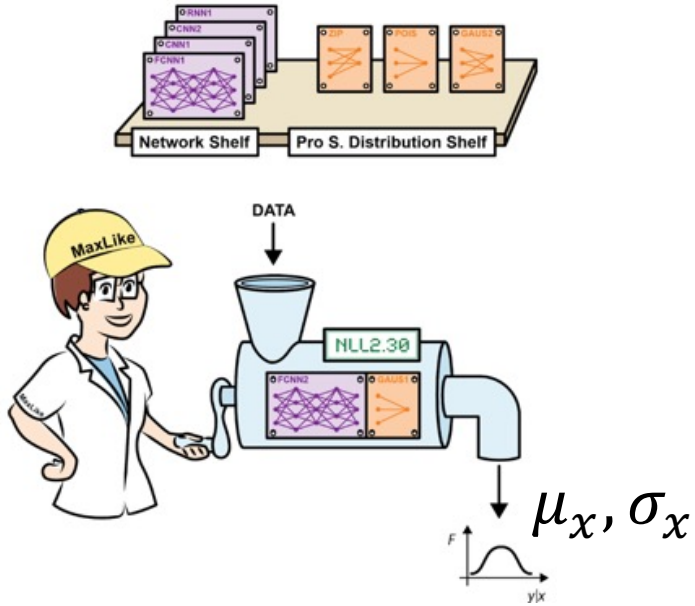
CPD in ML / deep learning setting

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

- After training all information is in the optimally chosen weights $\hat{w}$

$$p(y|x, \hat{w})$$

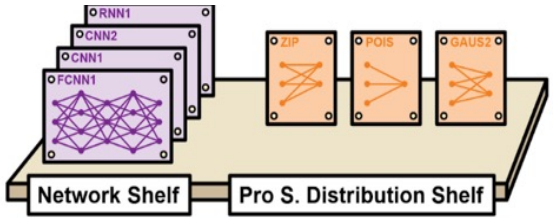
- Deep Learning recipe for probabilistic models



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

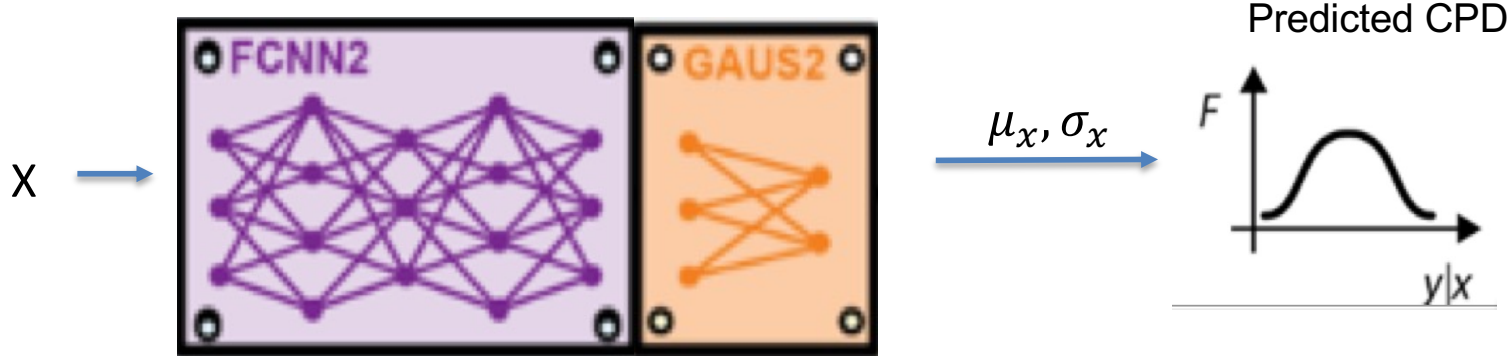
Regression output as CPD

- A trained network outputs for each x the parameters of a distribution.
- For example, network controls μ_x, σ_x of $N(\mu_x, \sigma_x)$ for a given x.



Input: Images, numbers,…

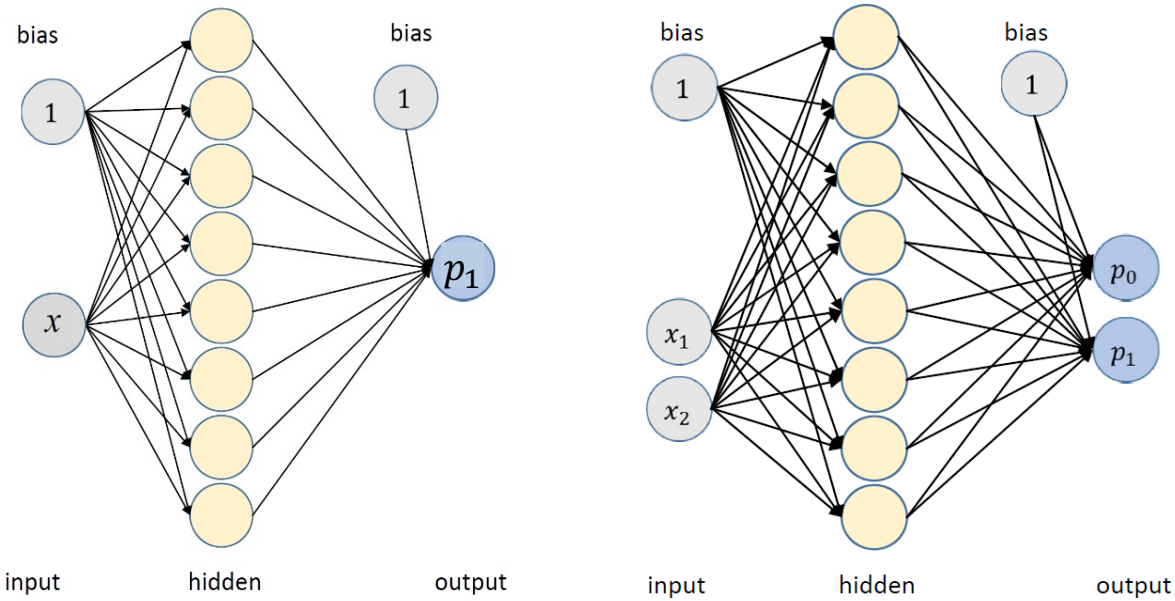
Output controls parameters of cpd



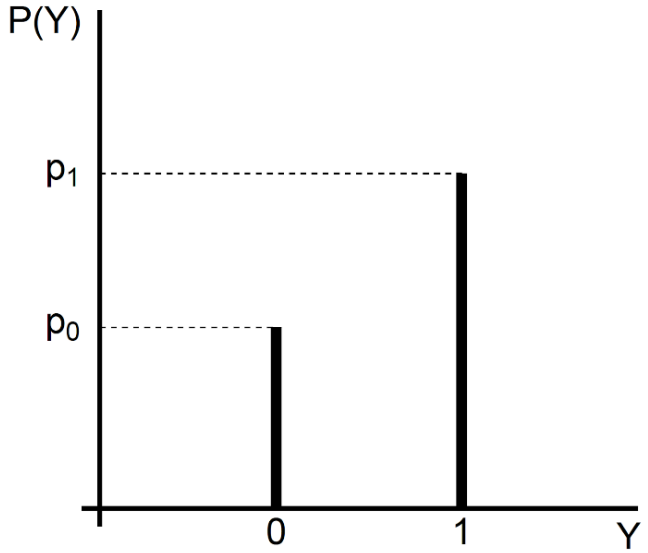
Binary Classification Output as CPD

- Binary Classification single output is sufficient (prob. p_1 for class 1)
- This can be seen as parameter p_1 for Bernoulli distribution
  - $y \sim Bern(p_1)$

Two possible NN architectures



Predicted CPD

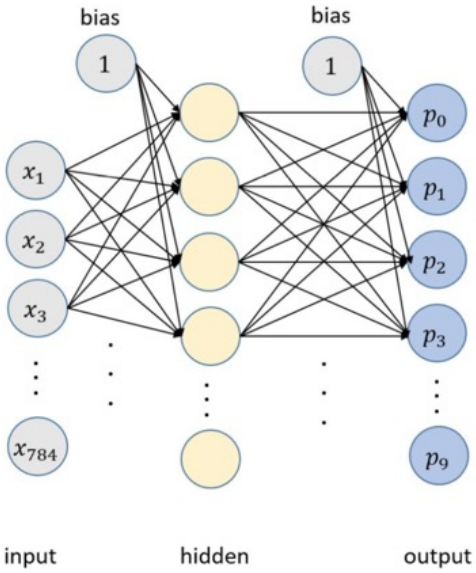


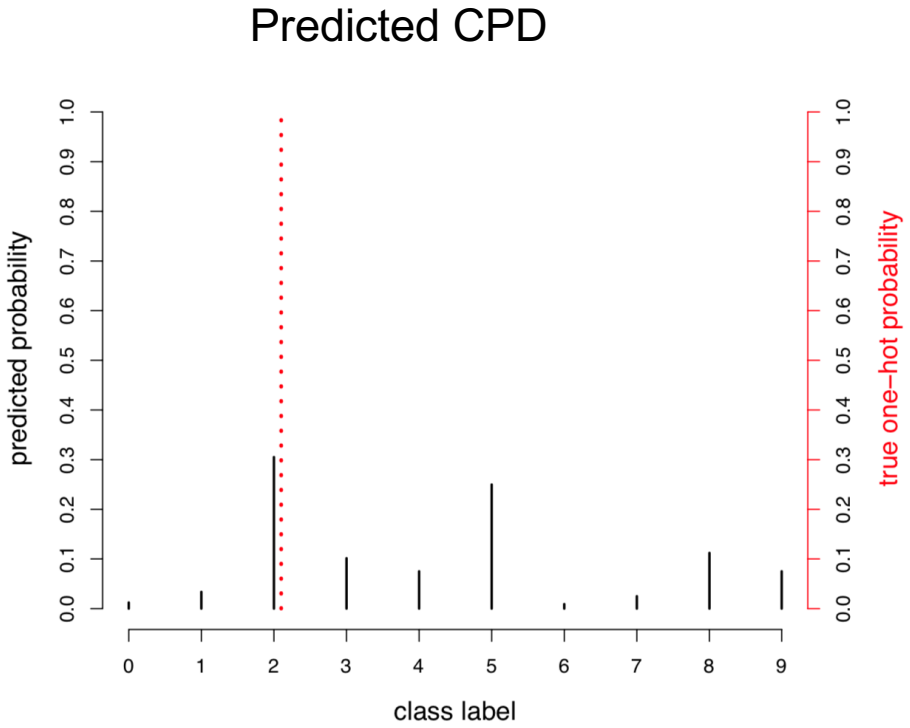
Which activation function is used in last layer?

$$P(Y=k)=\begin{cases} p_0 & \text{for } k=0 \\ p_1 & \text{for } k=1 \end{cases} \quad \text{with } \sum p_i = 1$$

Multi-class Classification Output as CPD

- In multi-class 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.





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

Conditional**P**robability**D**istribution:

Unifying framework for classification and regression.

Network controls the parameters of a CPD for given value x.

How to fit a probabilistic model?

Recap: Maximum Likelihood
(one of the most beautiful ideas in statistics)

Ronald Fisher in 1913
Also used before by
Gauss, Laplace

Likelihood / “probability“
(often known)

$M(\theta) \longrightarrow$ Data

Tune the parameter(s) θ of the model M so that (observed) data is most likely

Machine Learning and ML…

Maximum Likelihood rules Machine Learning

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

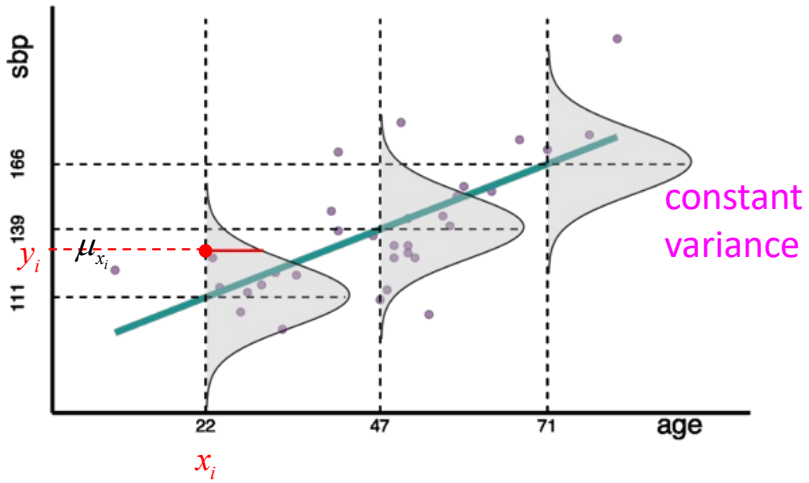
$$\hat{w} = argmax \prod_{i=1}^{N} p(y_i|x_i, w))$$

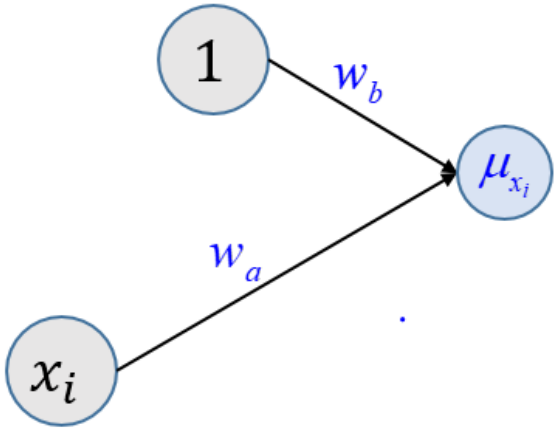
Practically: Minimize Negative Log-Likelihood NLL of the CPD (take log; minimize after multiplication with -1)

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

Example linear regression

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





Maximum likelihood:

$$\mathbf{w}_{\mathrm{ML}} = \underset{w}{\operatorname{argmax}} \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \, \mathrm{e}^{-\frac{(y_i-\mu_{x_i})^2}{2\sigma^2}}$$

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

Negative Log-Likelihood (NLL)

$$(\hat{w}_a, \hat{w}_b)_{\mathrm{ML}} = \underset{w_a, w_b}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \left(y_i - (w_a \cdot x_i + w_b)\right)^2$$

gradient descent with MSE loss

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

Minimizing NLL loss $\overset{\sigma \; const}{\Longrightarrow}$ Minimizing MSE loss

Proof:
MaxLik of Gaussian with constant variance is MSE-Loss

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

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

"NLL" Inserting the Gaussian

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

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

Here we assume constant spread

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

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

Here we assume constant spread

$$\hat{w} = argmin_w \{ \frac{1}{n} \sum_{i=1}^{n} (\mu_{x_i} - y_i)^2 \}$$

The MSE Formula

How to get the variance which is assumed to be constant?

The constant variance drops out in the MSE optimization. There are two ways to get it **after the fitting.** Then μ_{x_i} are the predicted means.

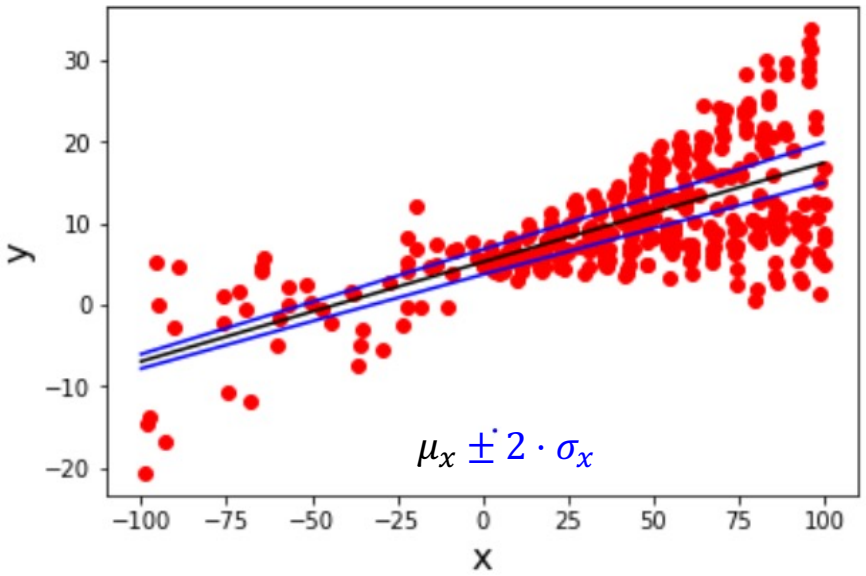
- From residuals after fitting, it's.

$$\widehat{\sigma^2} = \frac{1}{n-2}\sum\left(y_i - \mu_{x_i}\right)^2$$

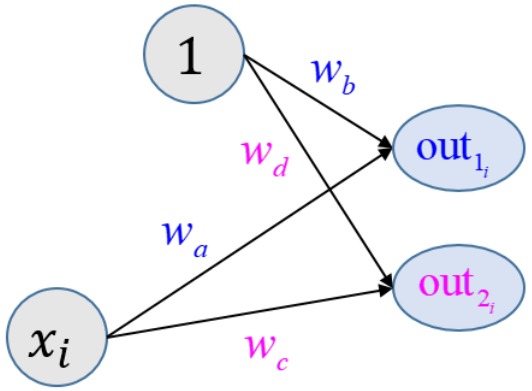
- By optimizing the variance σ in the NLL

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

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 mean negative log-likelihood (NLL) on train data:

$$NLL(w) = \frac{1}{n} \sum_{train-data} -log(p(y_i|x_i, w))$$

gradient descent with NLL loss

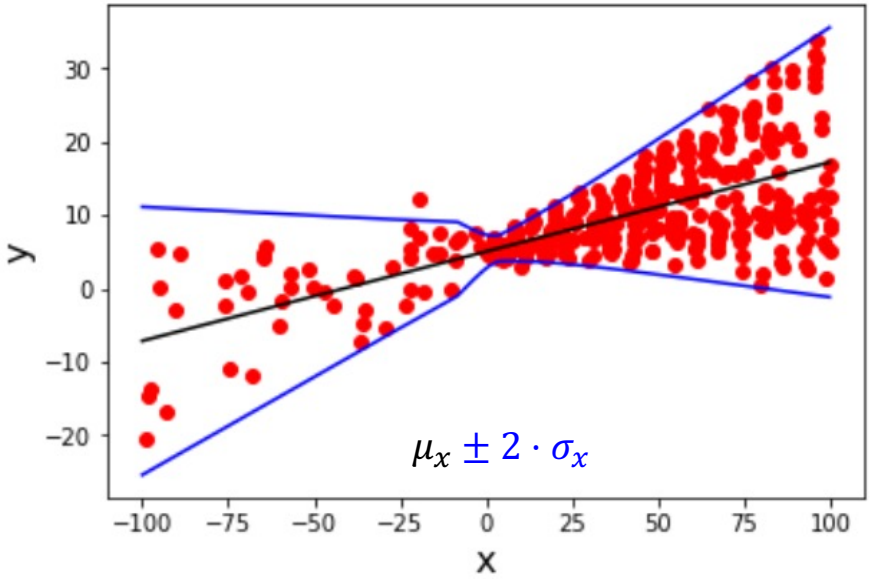
$$\hat{\mathbf{w}}_{\mathrm{ML}} = \underset{w}{\mathrm{argmin}} \sum_{i=1}^{n} -\log\left(\frac{1}{\sqrt{2\pi\sigma_{x_i}^2}}\right) + \frac{\left(y_i - \mu_{x_i}\right)^2}{2\sigma_{x_i}^2}$$

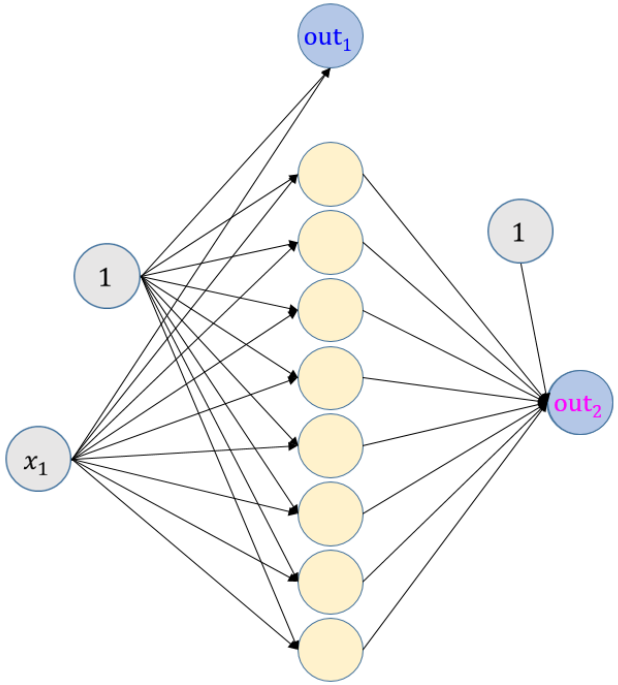
$\hat{w}_a$ $\hat{w}_b$

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

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 mean negative log-likelihood (NLL) on train data:

$$NLL(w) = \frac{1}{n} \sum_{train-data} -log(p(y_i|x_i, w)$$

gradient descent with NLL loss

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

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

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)

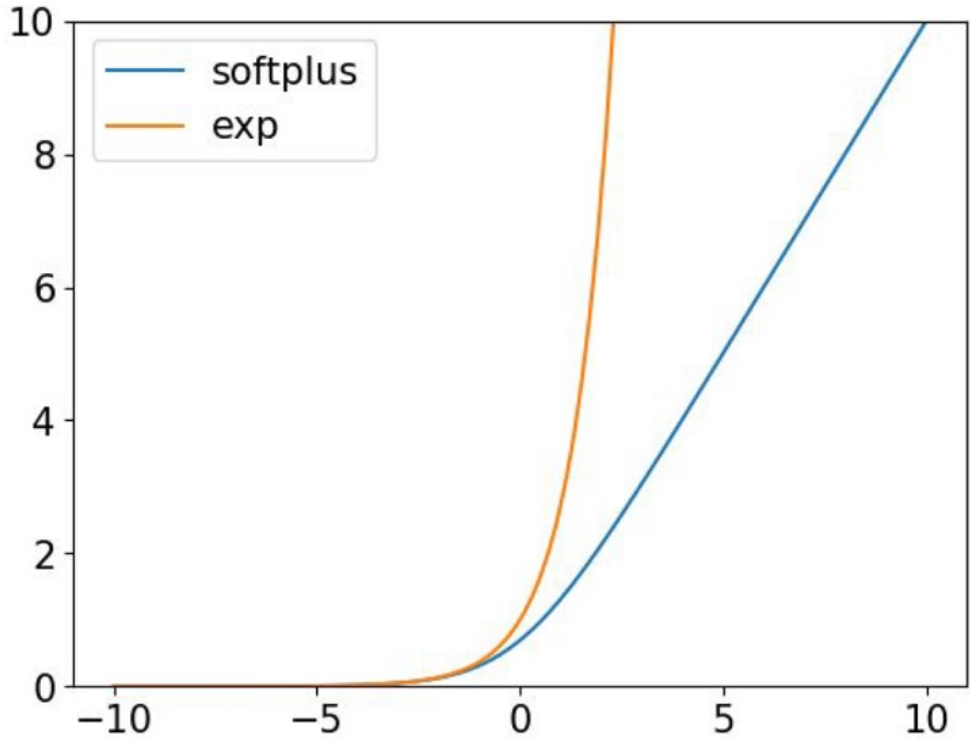
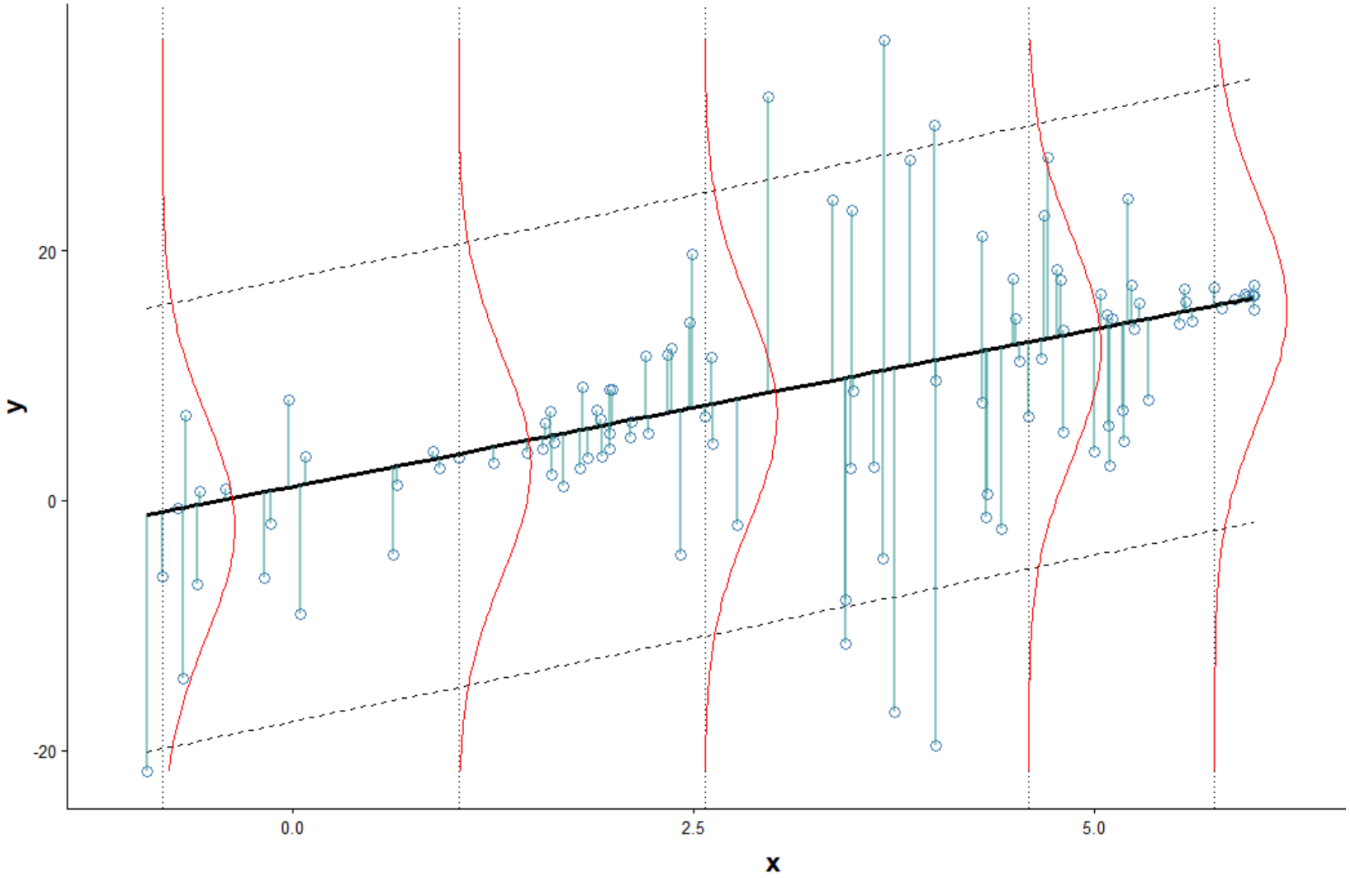


Figure 5.sp: The softplus function compared with the exponential function. Both functions map arbitrary values to positive values.

How to evaluate a probabilistic prediction model?

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

Validation data



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

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

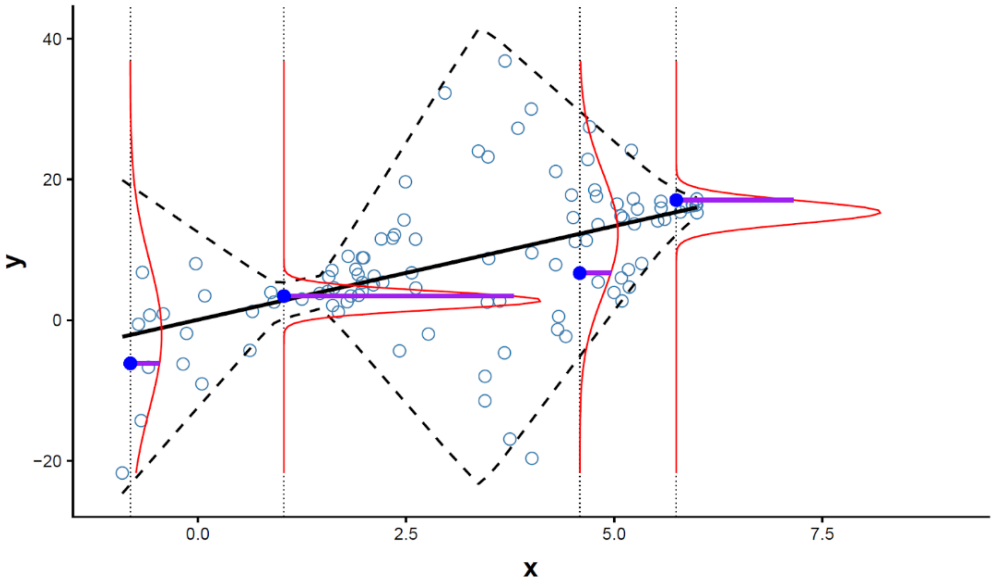
RMSE and MAE alone do not capture performance for probabilistic models!

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 on test data to assess the prediction performance

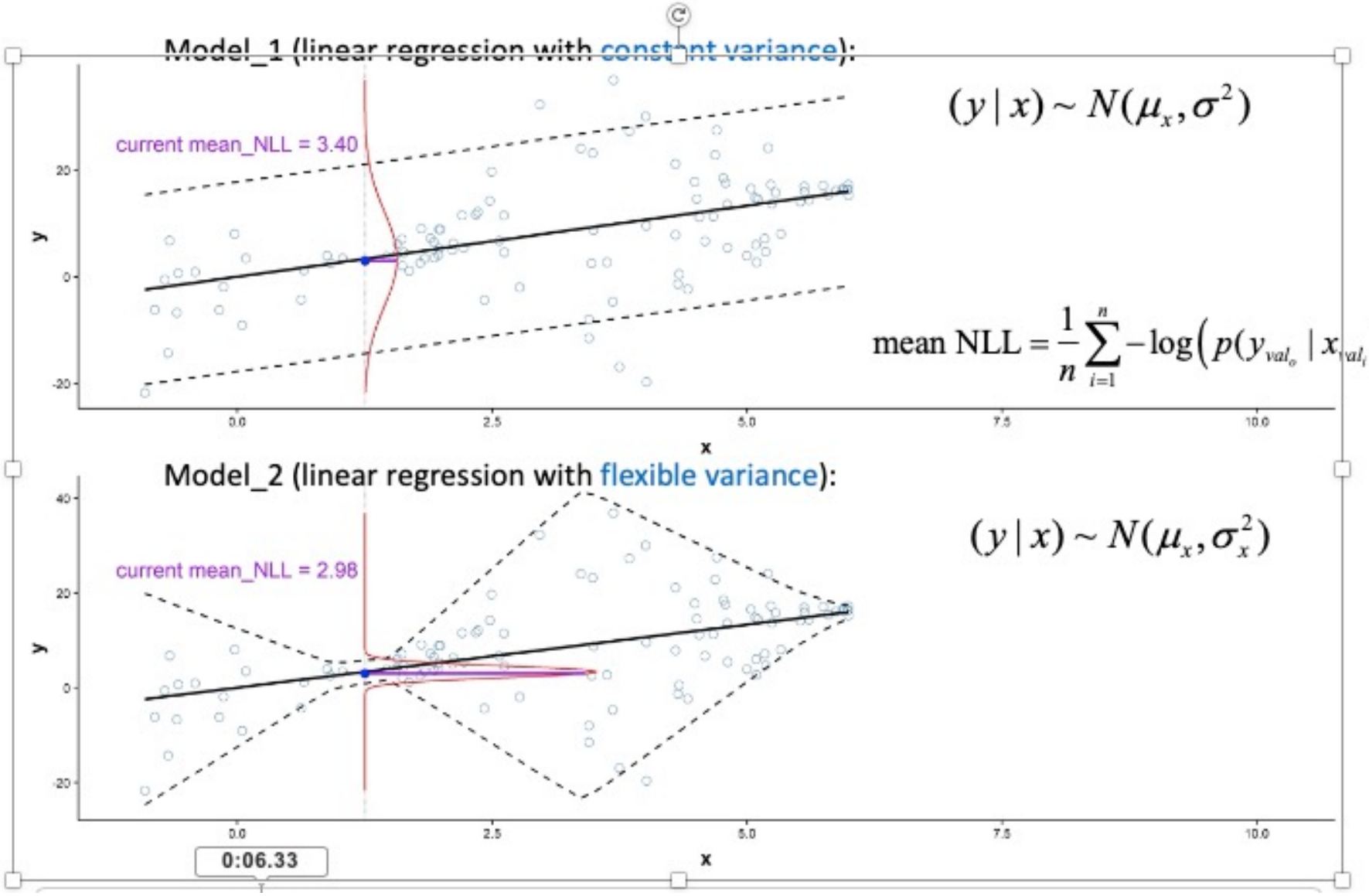
- Use the model on test data to predict for each input x_i a CPD p_{pred}
- Evaluate the predicted CPD at the position of the observed outcome y_i:

$$NLL = \frac{1}{n} \sum_{test-data} -log(p(y_i|x_i))$$



Tensorchief's youtu.be channel

Live Demo



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

Notes

- Log-Score a.k.a. NLL was first mentioned in connection to weather forecasting by Good (1952), who went so far as to suggest that the funding of the Met Office should depend on it.

- “If you see an analysis using something else, either it is a special case of the log scoring rule or it is possible much worse”.
  - Richard McElreath in Statistical Rethinking

- Widely used in DL-community as NLL on validation set to compare different models (sometimes different names) cross-entropy, ”bits per pixel”, perplexity exp(NLL)

Some hint to detailed treatment

- Statistical View: NLL is a so-called *proper scoring rule*
  - It's minimal (in expectation) if predicted distribution is data-generating distribution
    - Under certain assumptions (locality, smoothness, continuous outcome) NLL is the only one (strictly proper)*
  - For NLL: expected score $= E_{(x',y')\sim p_{true}}[-\log(p_{pred}(y'|x')\,]$
  - Gneiting, Raftery "Strictly Proper Scoring Rules, Prediction, and Estimation"
- Alternative View: Information Theory
  - The expectation of NLL is call *cross entropy*
  - For $p = p_{true} = p_{pred}$ it reaches its minimal value
    - $E_{(x',y')\sim p}\,[-\log(p\,(y'|x')\,] = E_p\ [-\log(p)\,] =: H(p)$ The Entropy
  - Information theory
    - Entropy coding length under true distribution
    - Cross Entropy coding length under wrong distribution p_{pred}
    - Difference is the KL-Divergence
  - See e.g. McElreath "Statistical Rethinking"

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

NLL as general cure-all in probabilistic modeling

Training of probabilistic models

- Maximize likelihood ↔ minimize negative log-likelihood (NLL)

Evaluation / model comparison:

- The log-score (NLL) is strictly proper score for regression.
- The log-score (NLL) is also strictly proper for classification models.

=> To evaluate or compare probabilistic models: use the validation NLL!

The NLL can be used for scoring and training

Probabilistic NN (technical details)

TensorFlow Probability

TensorFlow Probability (TFP) is an add-on to TF/Keras
TFP can model Distributions with tfd_...

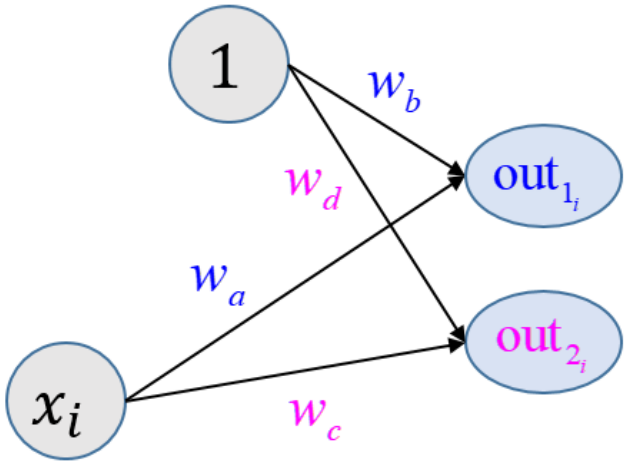
- Different distributions all have the same API.

```
d  = tfd_normal(loc = 1.0, scale = 0.1) # Create a normal distribution with mu=1 and sigma=0.1
(s = tfd_sample(d,2)) # Draw two random points, note we have tensor
as.numeric(s) # Now, we have numeric values
s$numpy()     # Alternative use when as.numeric() makes troubles
tfd_prob(d, 3) #Compute density/mass.
tfd_cdf(d, 3) #Compute CDF
tfd_mean(d) #Compute mean (expectation)
tfd_stddev(d) #Compute std
```

```
tf.Tensor([1.0305088  0.98513895], shape=(2,), dtype=float32)
[1] 1.030509 0.985139
[1] 1.030509 0.985139
tf.Tensor(0.0, shape=(), dtype=float32)
tf.Tensor(1.0, shape=(), dtype=float32)
tf.Tensor(1.0, shape=(), dtype=float32)
tf.Tensor(0.1, shape=(), dtype=float32)
```

Model the CPD

$y \sim N$(out_1, 1e-3+softplus(0.05*out_2))




```
#Connecting the output of the network (out) with the
#parameters of the CPD
my_dist <- function(out) {
  tfd_normal(
      loc=out[, 1, drop=F], #out_1 defines loc aka mean
      scale = 1e-3 + tf$math$softplus(0.05 * out[, 2, drop=F]) #out_2 defines scale aka std via softplus
  )
}

inputs = layer_input(shape=c(1))
out1 = inputs %>% layer_dense(units = 1)
out2 = inputs %>% layer_dense(units = 1)
out = layer_concatenate(c(out1,out2)) #only one layer
dist = layer_distribution_lambda(out,my_dist)
model_monotoic_sd <- keras_model(inputs = inputs, outputs = dist)
```

Connecting the out-nodes of network with distribution.

The output of the Keras model is now a distribution

```
NLL <- function(y, distr) -tfd_log_prob(distr,y)

model_monotoic_sd %>% compile(
  optimizer = optimizer_adam(lr=0.01),
  loss = NLL #pass-in the NLL
)
```

Calculates the NLL from the distribution distr (output of model) and the observed value y.
API: (y_observed, output of Keras model)

NLL as loss function to be optimized .

Exercise 1

- Getting Started with TFP
- Linear Regression with Keras and TFP
  - 3 Models for complex 1-D Data
- Notebook: 06-1d-regression