UNIVERSITAT DE BARCELONA

Jordi Vitrià and Axel Brando

Deep Learning

Probabilistic Modeling and Uncertainty Estimation

Motivation

Motivation

Uncertainty is an ill-defined concept.

What do we mean by **uncertainty**?

Level 1: Complete Certainty
A purely deterministic world. Think classical Newtonian physics.
Level 2: Risk without Uncertainty
A known probability distribution over a known set of outcomes. "This is life in a hypothetical honest casino, where the rules are transparent and always followed."
Level 3: Fully Reducible Uncertainty
There's a probability distribution over a set of known outcomes, but the parameters of that distribution are unknown. As your sample size increases, tools of (classical) statistical inference can bring this arbitrarily close to Level 2 uncertainty.
Level 4: Partially Reducible Uncertainty
The "true model" generating the data changes over time too frequently, or is too complex, to be estimated, so that you cannot use statistics to reduce this to Level 2 uncertainty even with arbitrarily large amounts of data.
Level 5: Irreducible Uncertainty
Ignorance that resists quantification and cannot be reduced with data. The realm of philosophy?
Level ∞: Zen Uncertainty
I… don't know what to tell you here.

Will Wilson
@WAWilsonIV

Motivation

Uncertainty is an ill-defined concept.

What do we mean by **uncertainty**?

Deterministic World

Casino with known dices

Casino with unknown dices

Incomplete information

Level 1: Complete Certainty
A purely deterministic world. Think classical Newtonian physics.
Level 2: Risk without Uncertainty
A known probability distribution over a known set of outcomes. "This is life in a hypothetical honest casino, where the rules are transparent and always followed."
Level 3: Fully Reducible Uncertainty
There's a probability distribution over a set of known outcomes, but the parameters of that distribution are unknown. As your sample size increases, tools of (classical) statistical inference can bring this arbitrarily close to Level 2 uncertainty.
Level 4: Partially Reducible Uncertainty
The "true model" generating the data changes over time too frequently, or is too complex, to be estimated, so that you cannot use statistics to reduce this to Level 2 uncertainty even with arbitrarily large amounts of data.
Level 5: Irreducible Uncertainty
Ignorance that resists quantification and cannot be reduced with data. The realm of philosophy?
Level ∞: Zen Uncertainty
I… don't know what to tell you here.

Will Wilson
@WAWilsonIV

Motivation

What do we mean by **uncertainty**?

In machine learning, it refers to the fact that predictions and/or parameters are considered **random variables**.

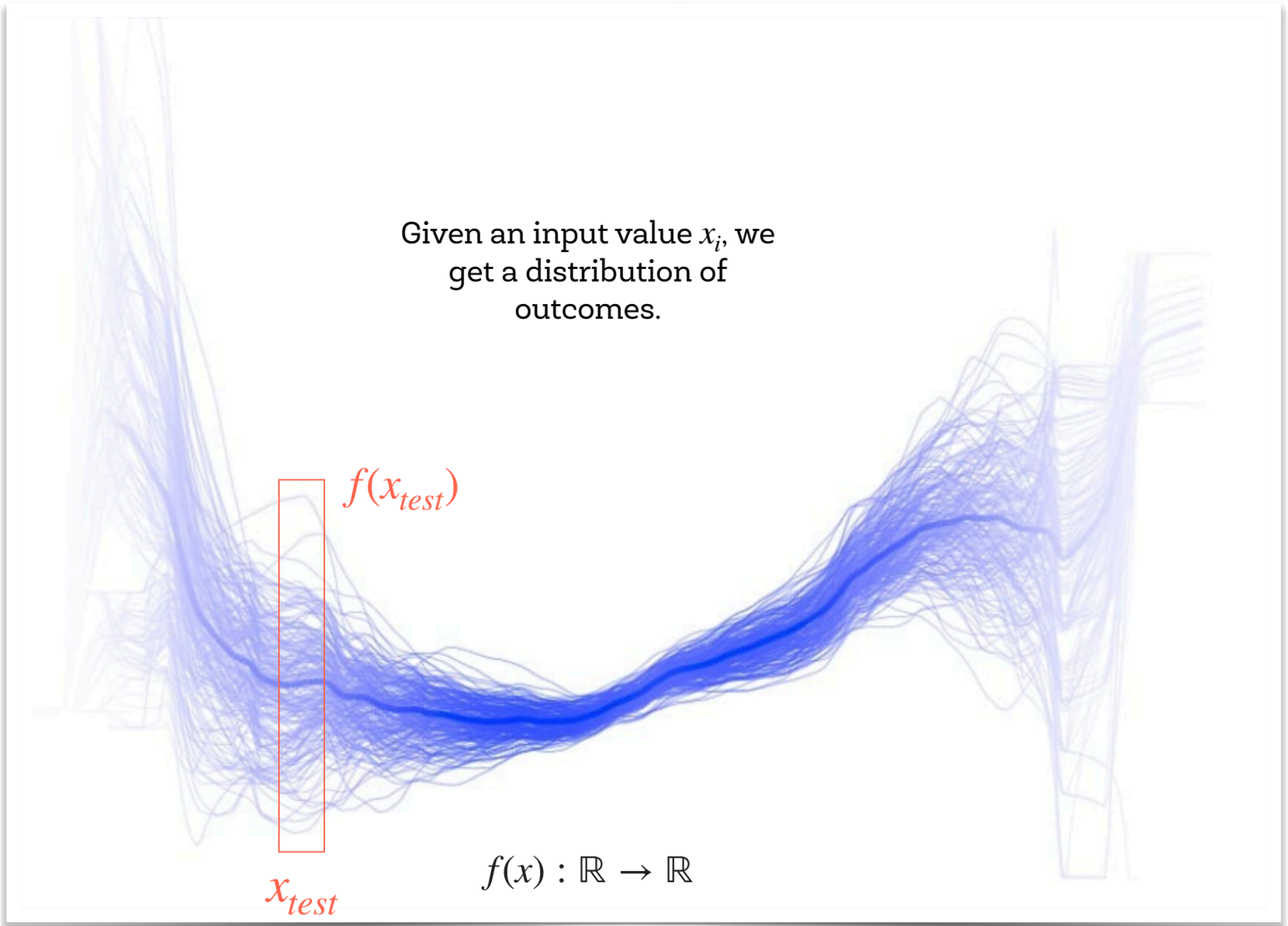
> The outcome of the model will be a **distribution** rather a single prediction (in both cases, regression and classification).

Its utility is to quantify our **trust** on model predictions.

> Uncertainty is a complex concept that needs to be **modeled**, **measured**, and **applied**.

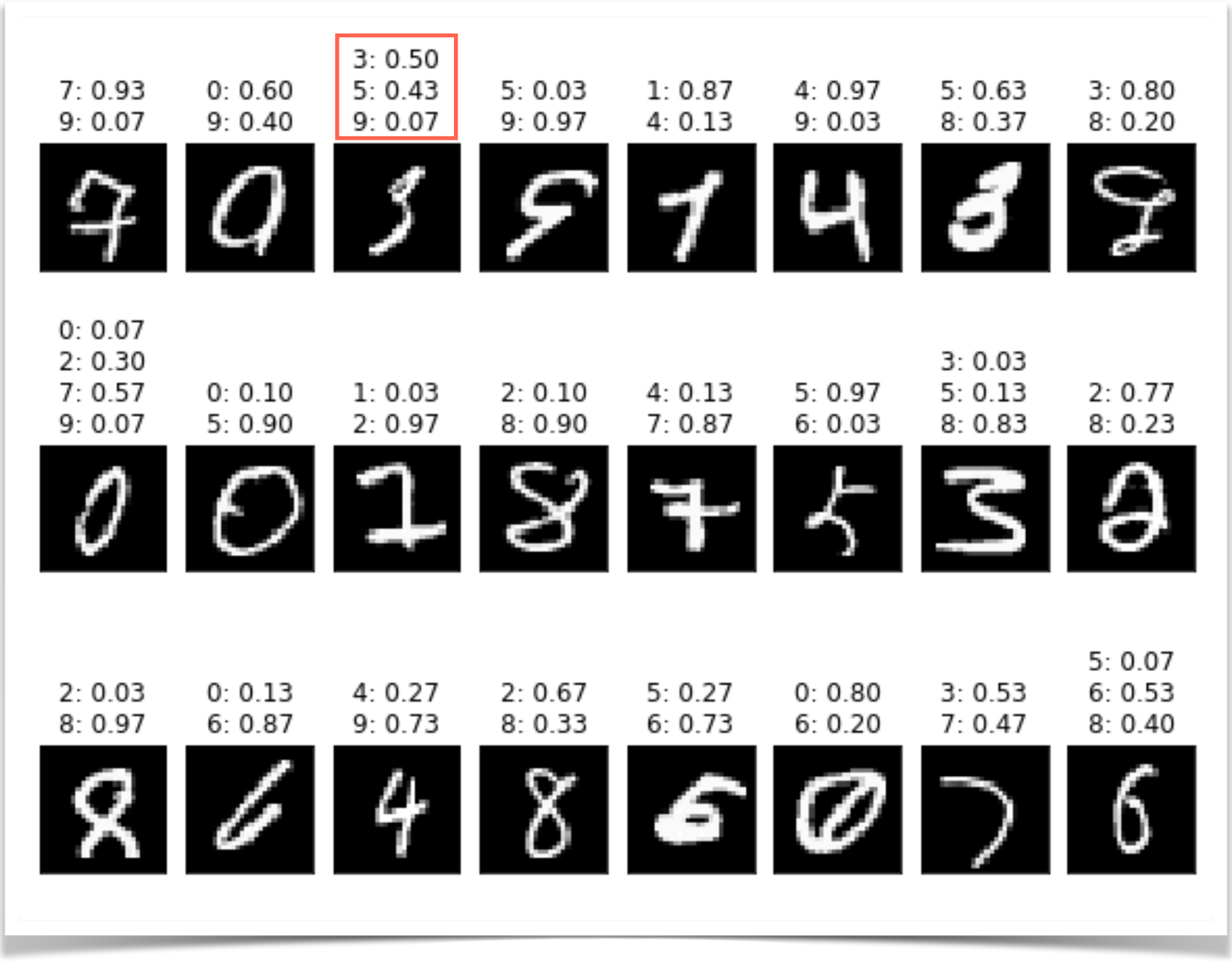
Motivation

Regression $f(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{R}^m$



Motivation

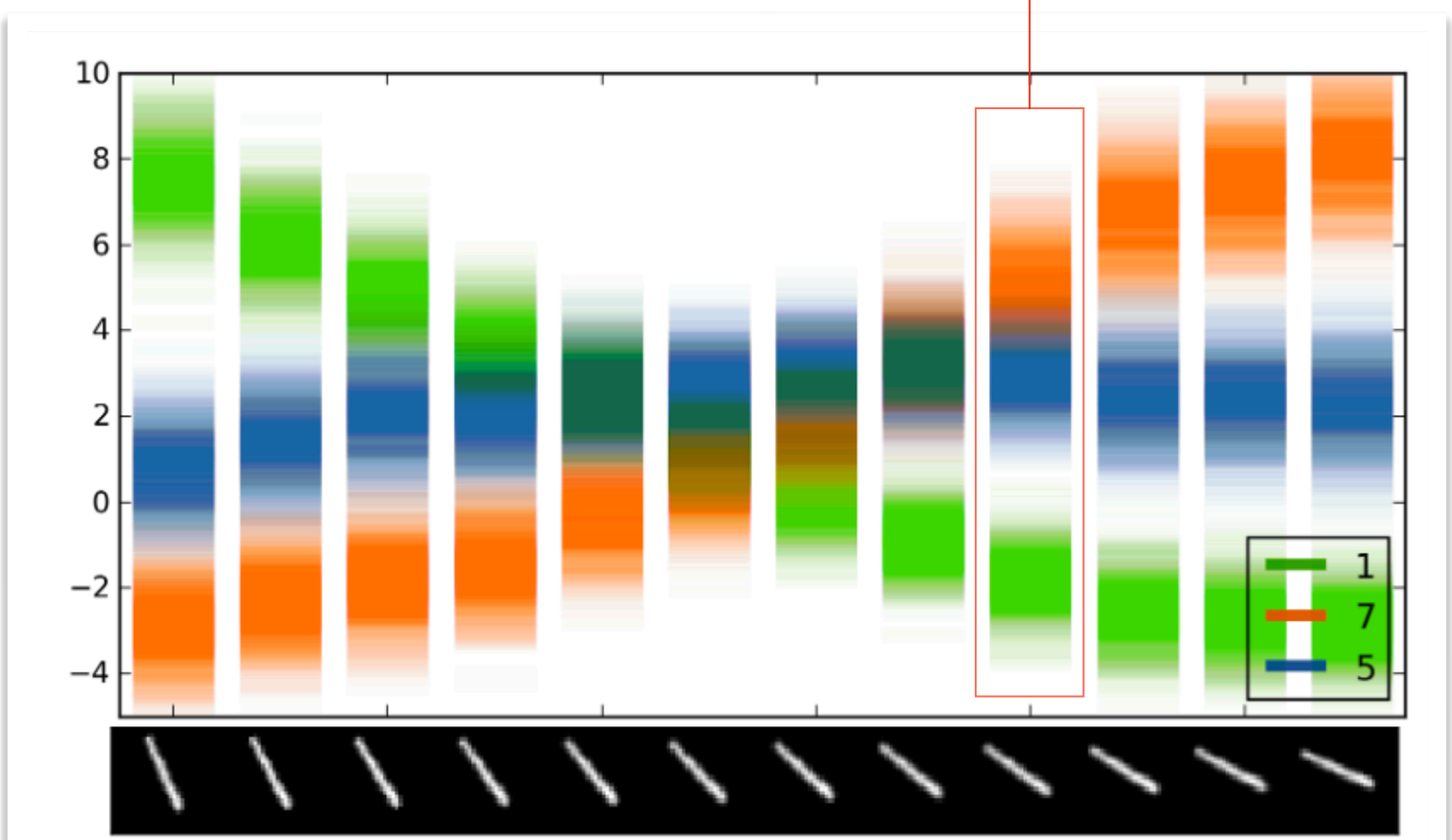
Classification $f(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{C}$



Motivation

Classification $f(\mathbf{x}) : \mathbb{R}^n \rightarrow \mathbb{C}$ **(out of distribution)**

Given an ensamble of deep learning models (trained with the MNIST dataset) this is the distribution of logit values corresponding to three different classes.

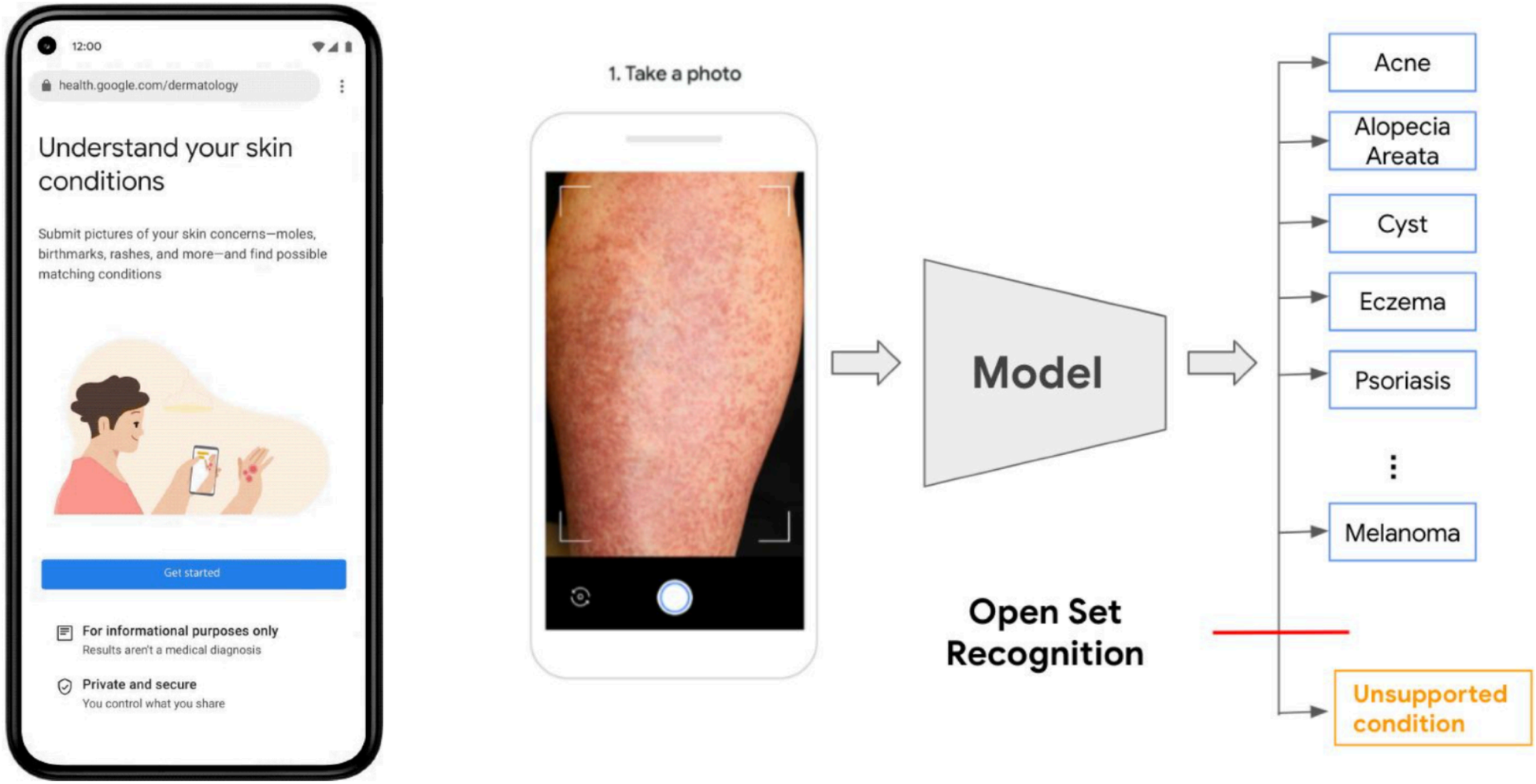


(a) Softmax *input* scatter

Out of distribution input samples

Applications

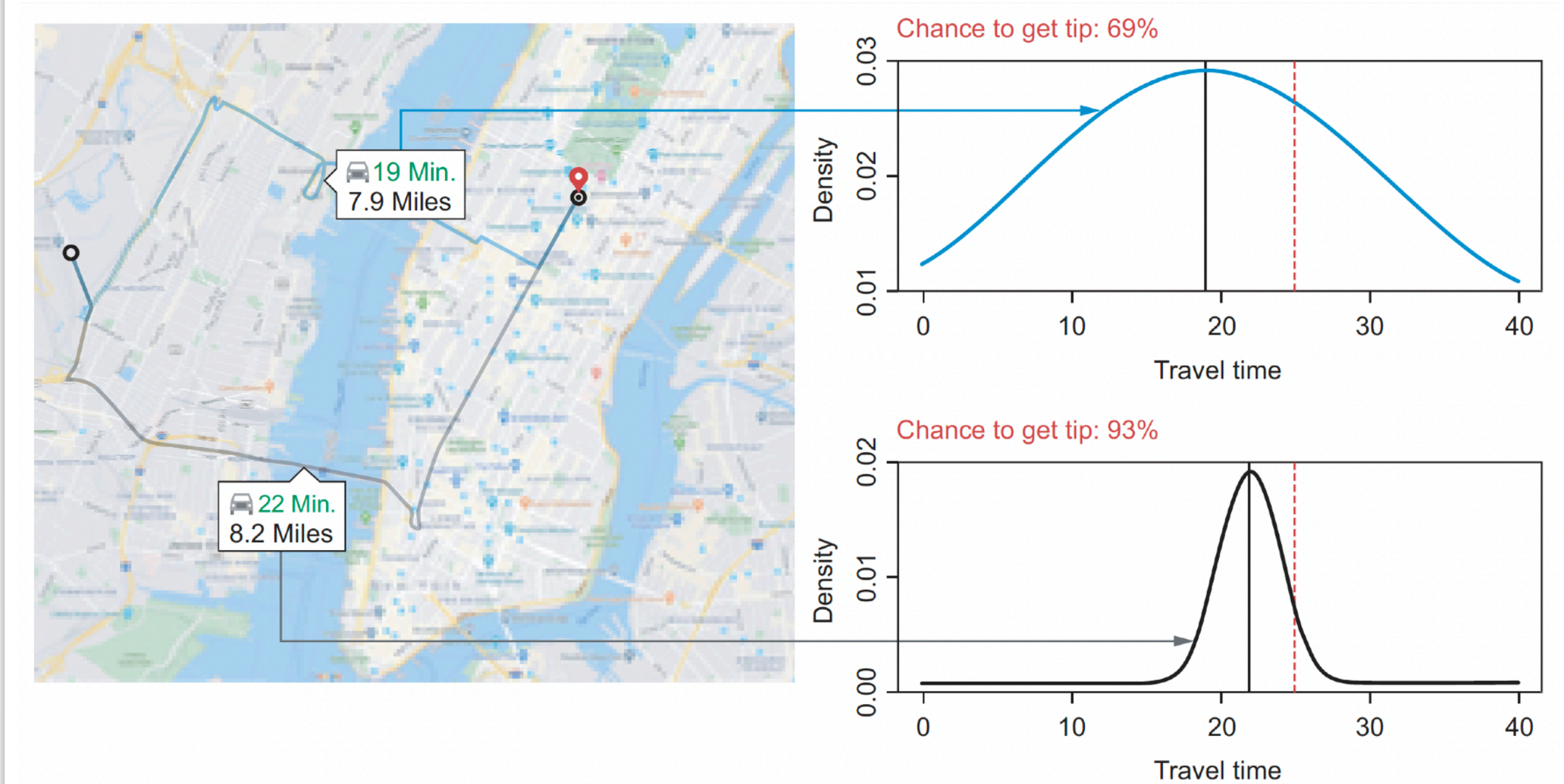
We can use model uncertainty to decide when to trust the model or to defer to a human.



Applications

Decision making and risk minimization

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

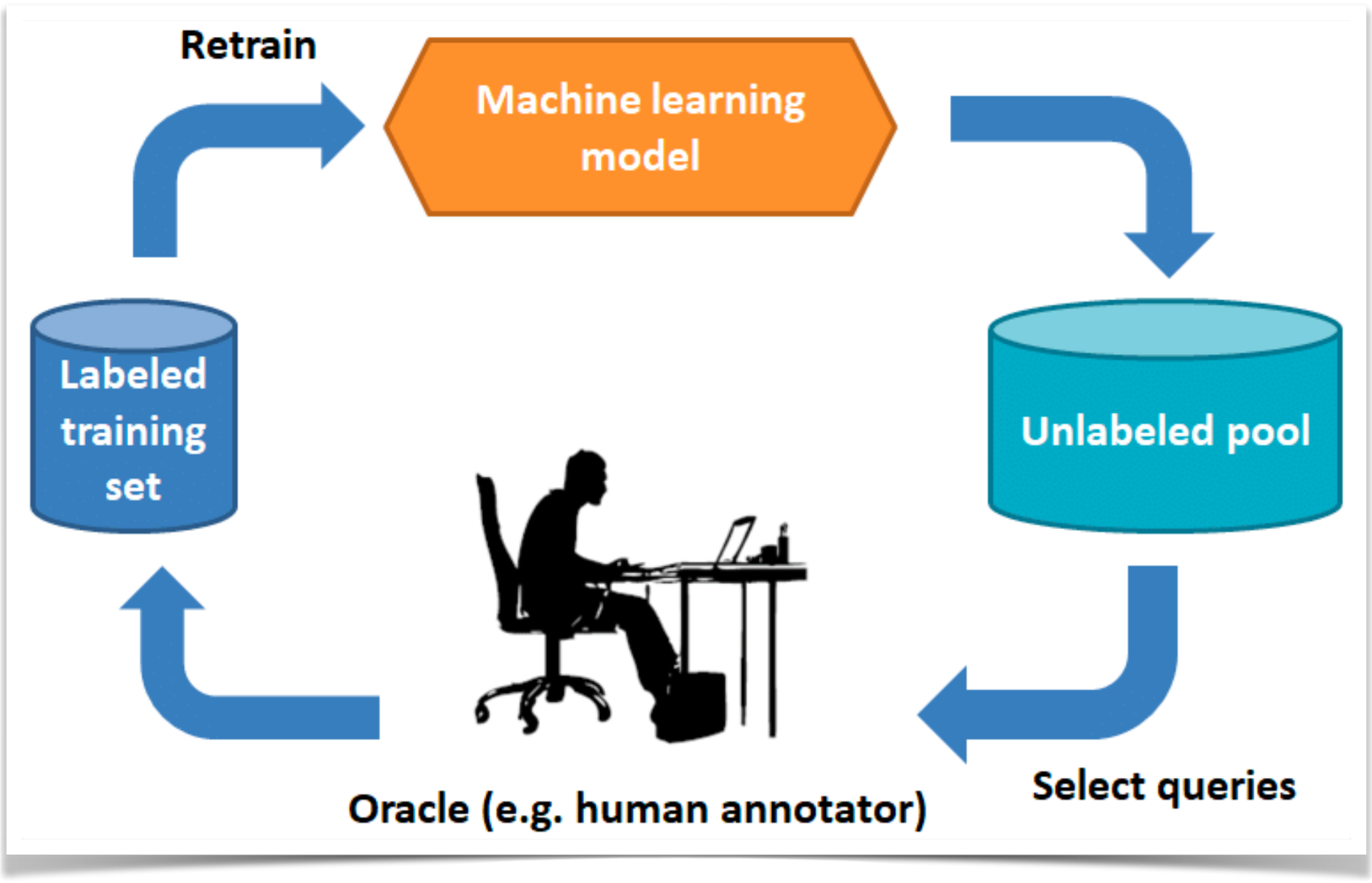


From "Probabilistic Deep Learning With Python, Keras and TensorFlow Probability" by Oliver Dürr, Beate Sick, Elvis Murina. ISBN 9781617296079

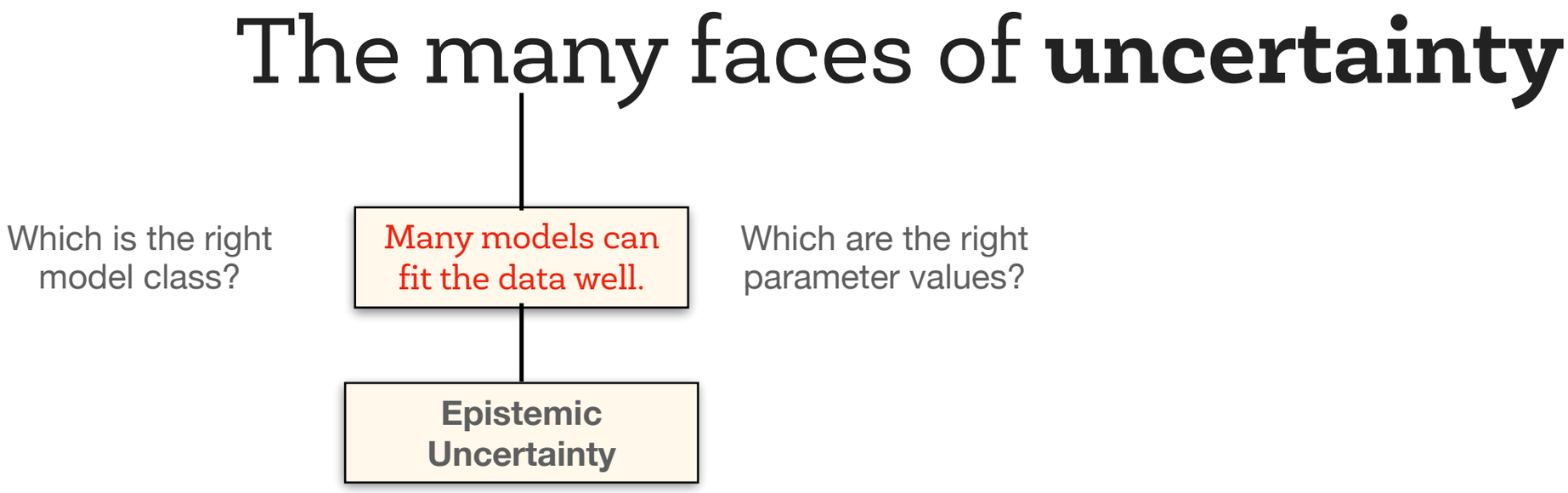
Applications

Active Learning

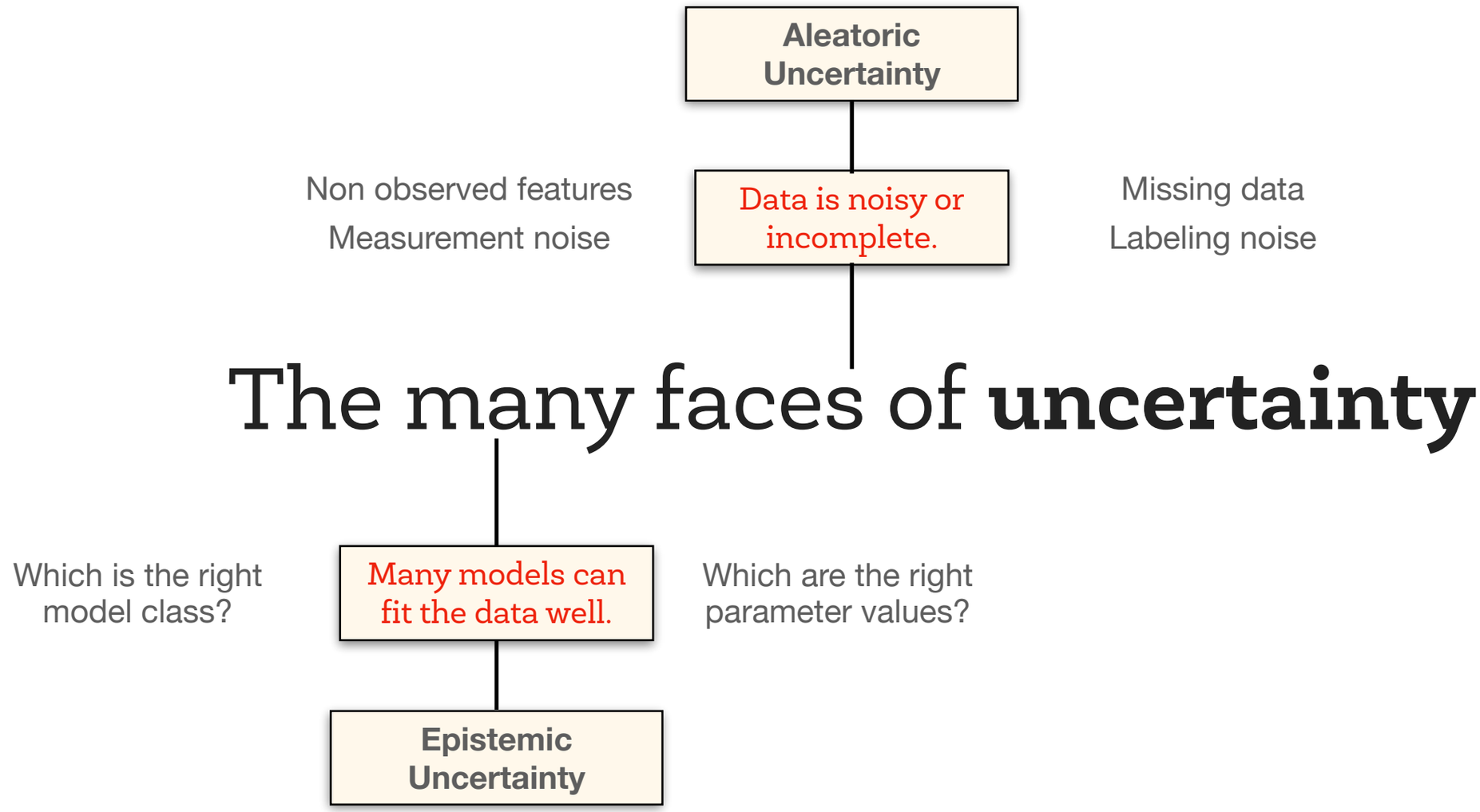
Active learning is a special case of machine learning in which a learning algorithm can interactively query a user (or some other information source) to label new data points with the desired outputs.



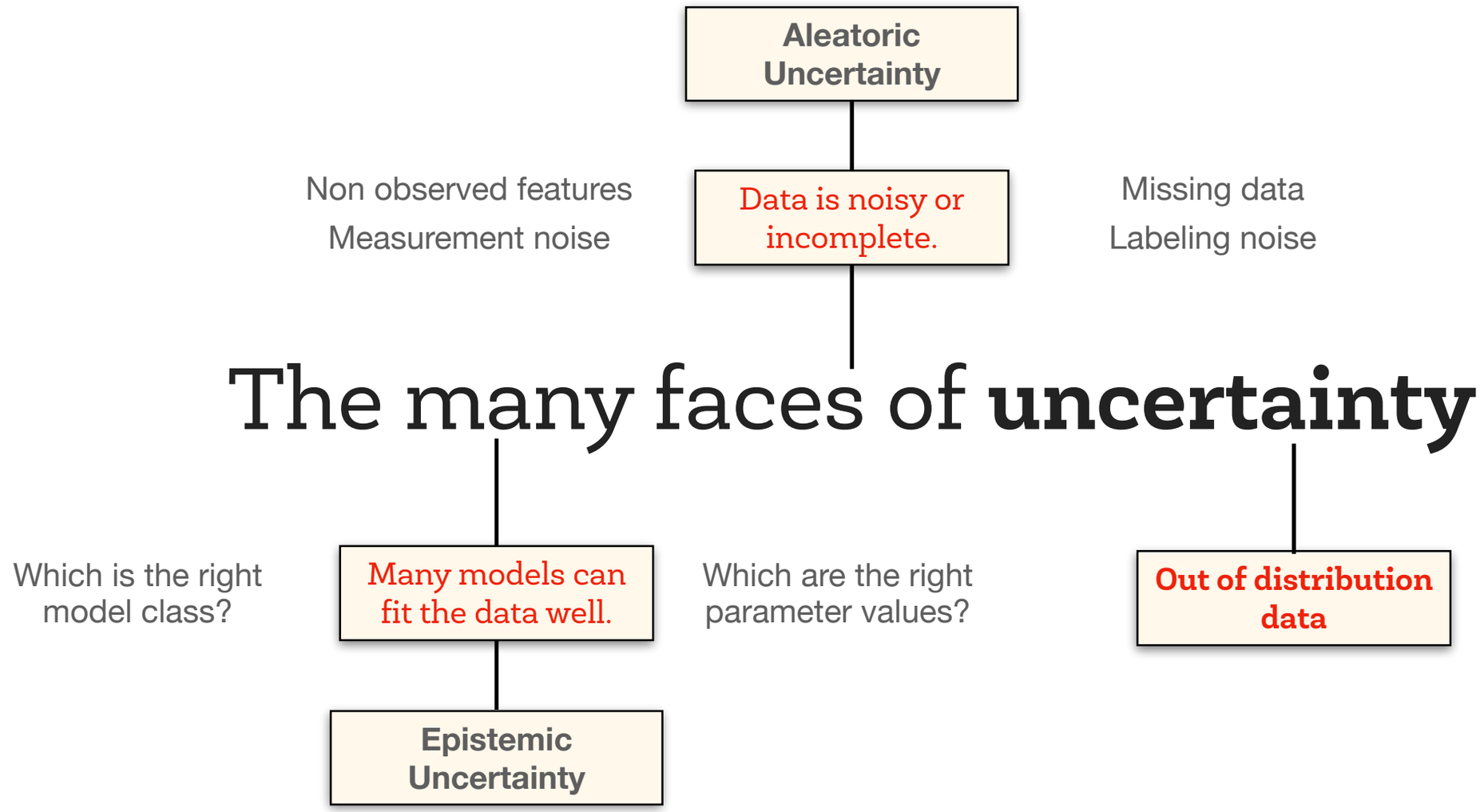
Sources of Uncertainty



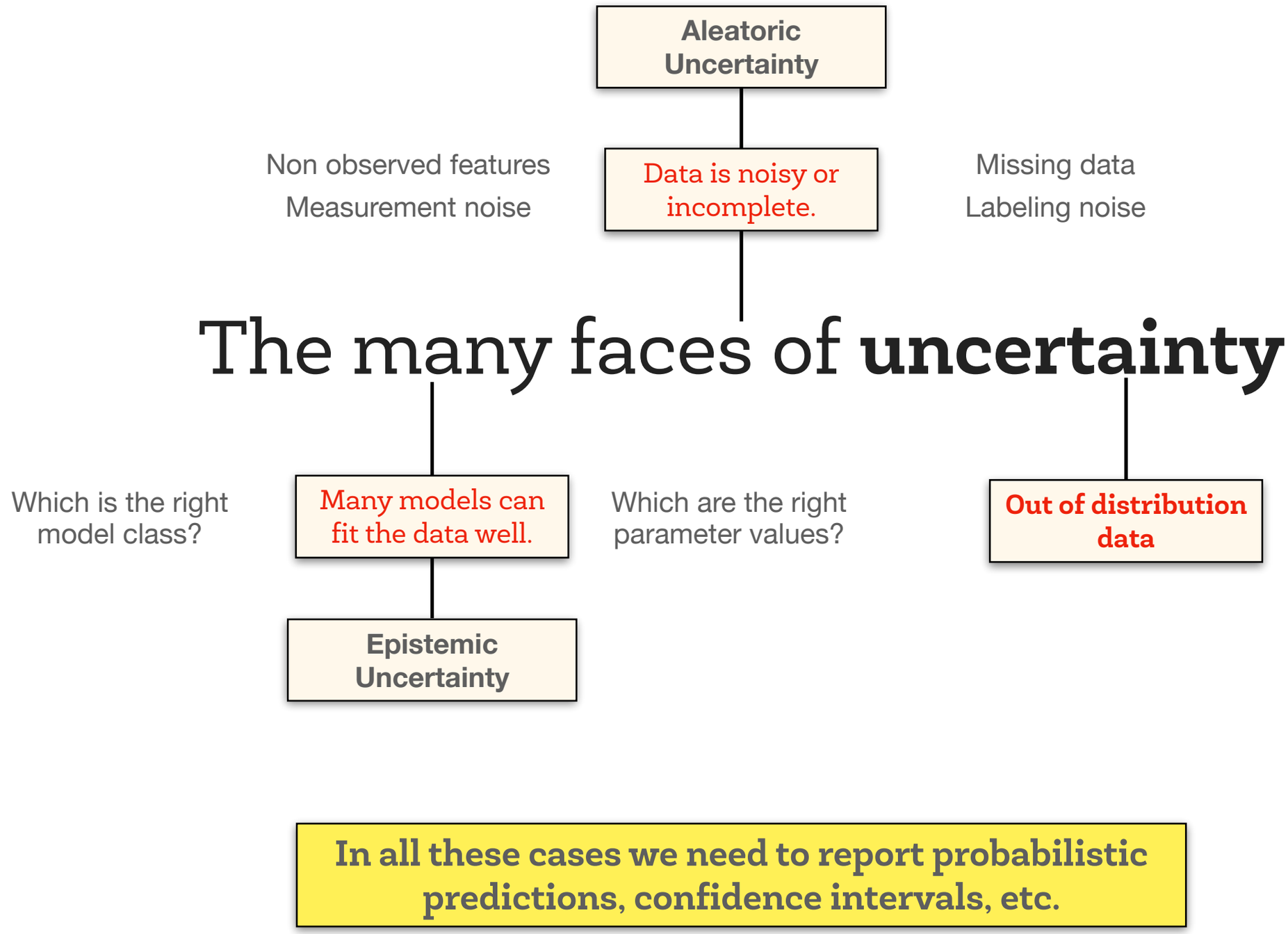
Sources of Uncertainty



Sources of Uncertainty



Sources of Uncertainty



Neural networks and the probabilistic point of view. The case of regression.

Reminder

From a statistical standpoint, a given set of **observations** in a dataset is a **random sample** from an unknown **population**.

The goal of **maximum likelihood estimation** is to make **inferences about the probability distribution** that is most likely to have generated the sample.

If we follow the parametric framework, associated with each probability distribution there is a unique vector $\theta = [\theta_1, \ldots, \theta_n]$ of parameters that index the probability distribution within a parametric family $\{f(\cdot\,; \theta | \theta \in \Theta)\}$.

Reminder

Evaluating the joint density at the observed data sample $\mathbf{y} = (y_1, y_2, \ldots, y_n)$ gives a real-valued function, $L_n(\theta) = f_n(\mathbf{y}; \theta)$ which is called the **likelihood function**.

For **independent and identically distributed** random variables, $f_n(\mathbf{y}; \theta)$ will be the product of density functions:

$$L_n(\theta) = \prod_{i=1}^{n} f(y_i; \theta)$$

$$f(x) = N(x; (\mu, \sigma)) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma}$$

$$f(x) = \sum_{j=1}^{K} \omega_j N(x; (\mu_j, \sigma_j))$$

$(\ldots)$

In probability theory and statistics, a collection of random variables is independent and identically distributed **if each random variable has the same probability distribution as the others and all are mutually independent**.

Probabilistic Neural Networks

A non-probabilistic model outputs only **one best guess** (f.e. the mean, $\mathbb{E}(y \mid x)$) for the outcome, whereas a **probabilistic model** predicts **a whole probability distribution** over all possible outcomes.

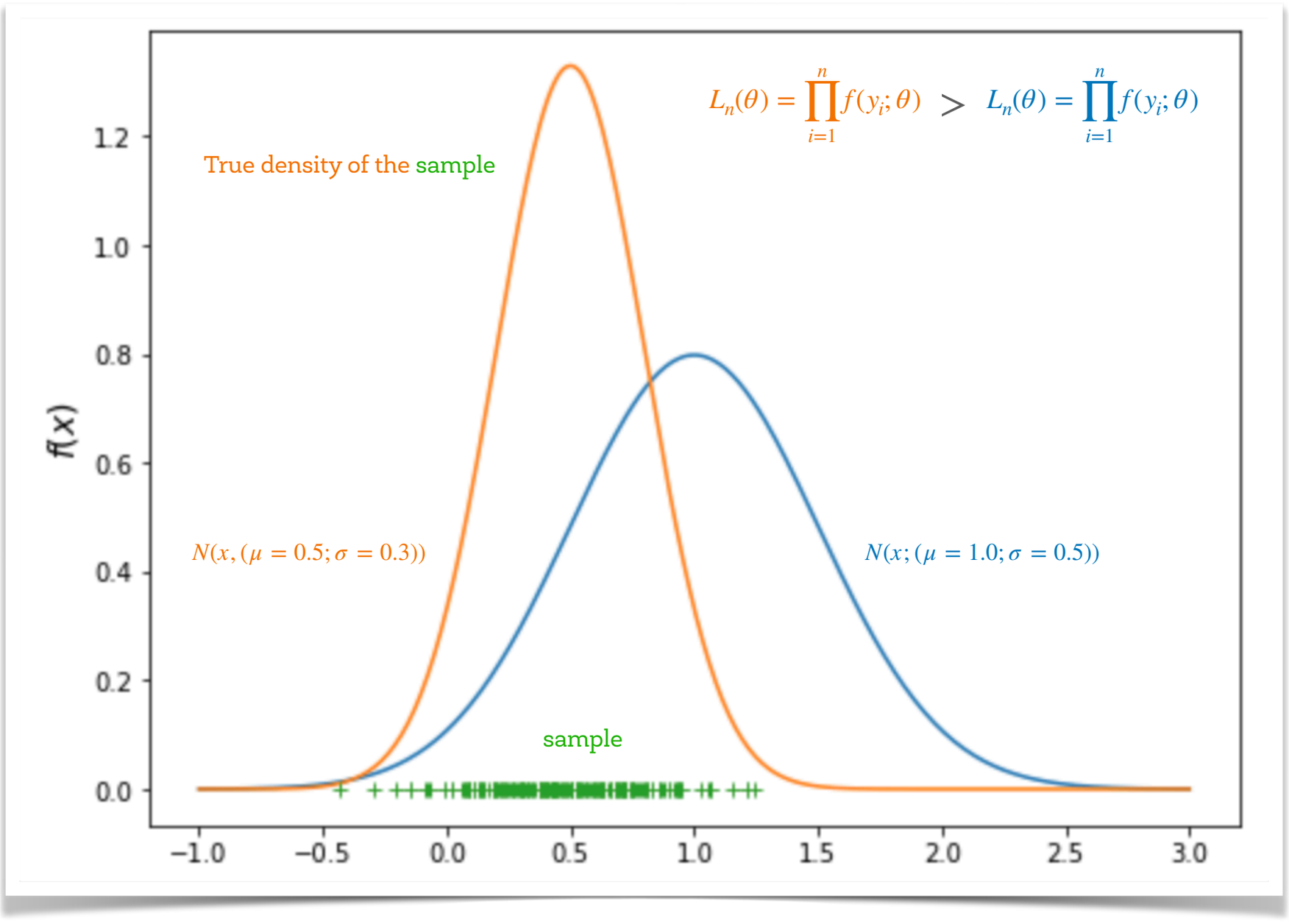
There is a generally valid approach for **deriving the loss function** when working with parametric probabilistic models.

This approach is called the **maximum likelihood approach**.

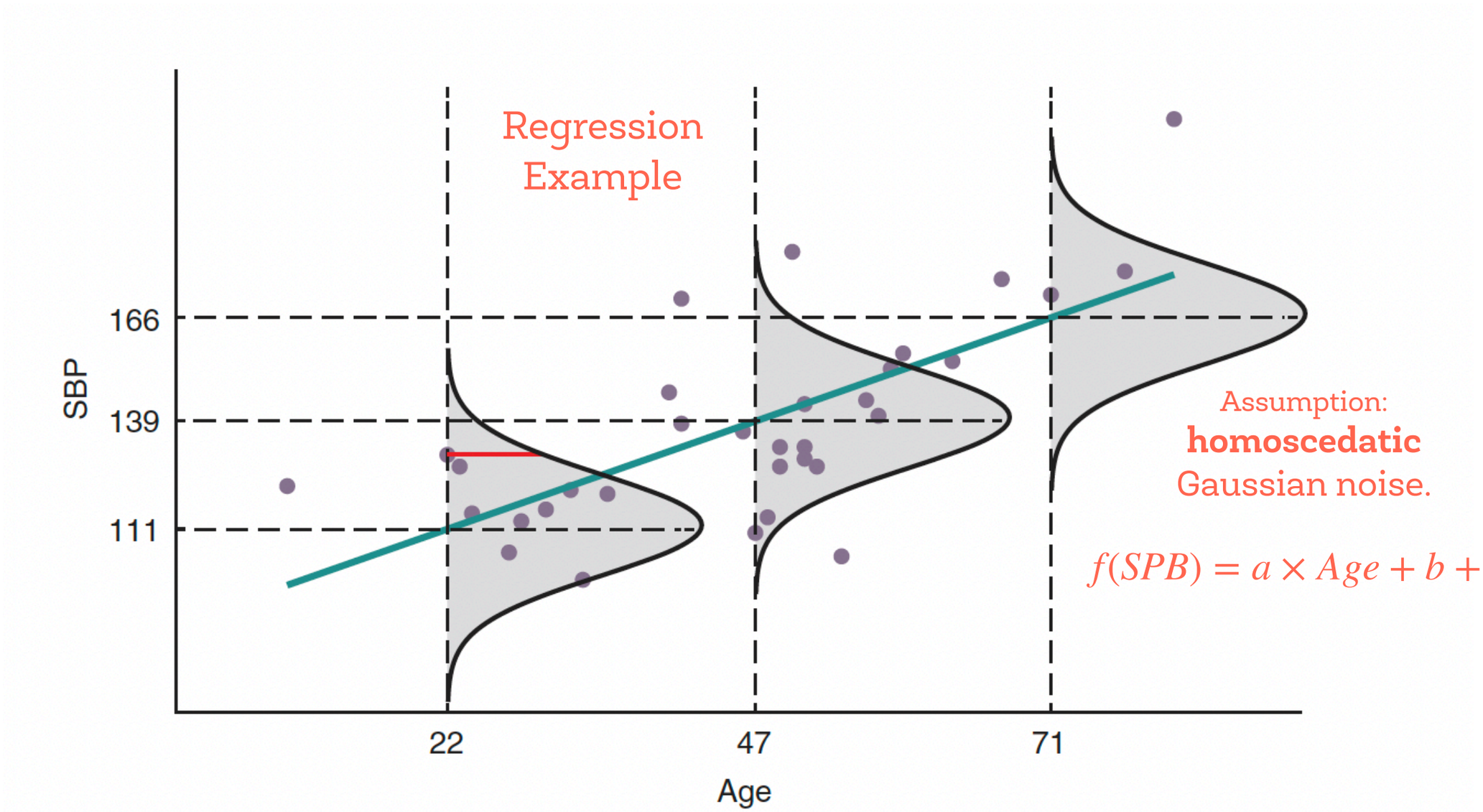
> The task of the NN is to choose the model's parameter(s) so that the observed data has the highest **likelihood**.

https://arxiv.org/pdf/1906.02530.pdf

Reminder


$L_n(\theta) = \prod_{i=1}^{n} f(y_i;\theta)$ > $L_n(\theta) = \prod_{i=1}^{n} f(y_i;\theta)$
True density of the sample
$N(x, (\mu = 0.5; \sigma = 0.3))$
$N(x; (\mu = 1.0; \sigma = 0.5))$
sample
f(x)
1.2
1.0
0.8
0.6
0.4
0.2
0.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
3.0

Probabilistic Neural Networks



$f(\text{SPB} \mid \text{Age}; \theta)$

SPB: Systolic Blood Pressure

Probabilistic Neural Networks

General regression template

```
def my_loss(y_true,y_pred):
  loss = -tf.reduce_sum(tf.math.log(f(y_true,y_pred)))
  return loss

model = Sequential()
model.add(Dense(1, activation='linear',
                batch_input_shape=(None, 1)))
model.compile(loss=my_loss,optimizer="adam")
```

1 Defines a custom loss function

3 Calculates the sum of all losses (see equation 4.4)

2 Sets up an NN equivalent to linear regression; includes one linear activation and a bias term

The Maximum Likelihood approach tells you that you need to find those values for the weights in the NN, here $w = (a, b)$, which maximize the **likelihood for the observed data**.

The following product shows the **likelihood of the data**:

$$w = \arg\max_{w} \prod_{i=1}^{n} f(y_i; x_i, w) = \arg\min_{w} \sum_{i=1}^{n} -\log f(y_i; x_i, w)$$

Probabilistic Neural Networks

General regression template

Defines a custom loss function

```
def my_loss(y_true,y_pred):
  loss = -tf.reduce_sum(tf.math.log(f(y_true,y_pred)))
  return loss

model = Sequential()
model.add(Dense(1, activation='linear',
                batch_input_shape=(None, 1)))
model.compile(loss=my_loss,optimizer="adam")
```

Calculates the sum of all losses (see equation 4.4)

Sets up an NN equivalent to linear regression; includes one linear activation and a bias term

Now we plug in the expression for the Normal density function:

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

$$w = \arg\min_w \sum_{i=1}^{n} \frac{(y_i-\mu_{x_i})^2}{2\sigma^2} = \arg\min_w \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i-\mu_{x_i})^2 = \arg\min_w \sum_{i=1}^{n} (y_i-\mu_{x_i})^2$$

So that we finally obtain the formula for the MSE loss!

The output of the network $\hat{y}$ is the expected mean of the distribution μ_x

$$\mathbb{E}(y \mid x)$$

Probabilistic Neural Networks

One assumption in classical linear regression is **homoscedasticity**, meaning that the variance of the outcome does not depend on the input value x. Therefore, you need only one output node to compute the first parameter μ_x of the conditional Normal distribution.

If you also allow the second parameter, σ_x, to depend on x, then you need an NN with a second output node. If the variance of the output is not constant but dependent on x, we talk about **heteroscedasticity**.

In this case, it can be shown that:

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

Probabilistic Neural Networks

Heteroscedastic regression

```
import math
def my_loss(y_true,y_pred):
  mu=tf.slice(y_pred,[0,0],[-1,1])
  sigma=tf.math.exp(tf.slice(y_pred,[0,1],[-1,1]))

  a=1/(tf.sqrt(2.*math.pi)*sigma)
  b1=tf.square(mu-y_true)
  b2=2*tf.square(sigma)
  b=b1/b2
  loss = tf.reduce_sum(-tf.math.log(a)+b,axis=0)
  return loss

model =  Sequential()
model.add(Dense(20, activation='relu',batch_input_shape=(None, 1)))
model.add(Dense(50, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(2, activation='linear'))
model.compile(loss=my_loss,\
optimizer="adam",metrics=[my_loss])
```

Defines a custom loss

2 Extracts the first column for μ

3 Extracts the second column for σ

4

1 Defines an NN with 3 hidden layers as in listing 4.4 but now with 2 outcome nodes

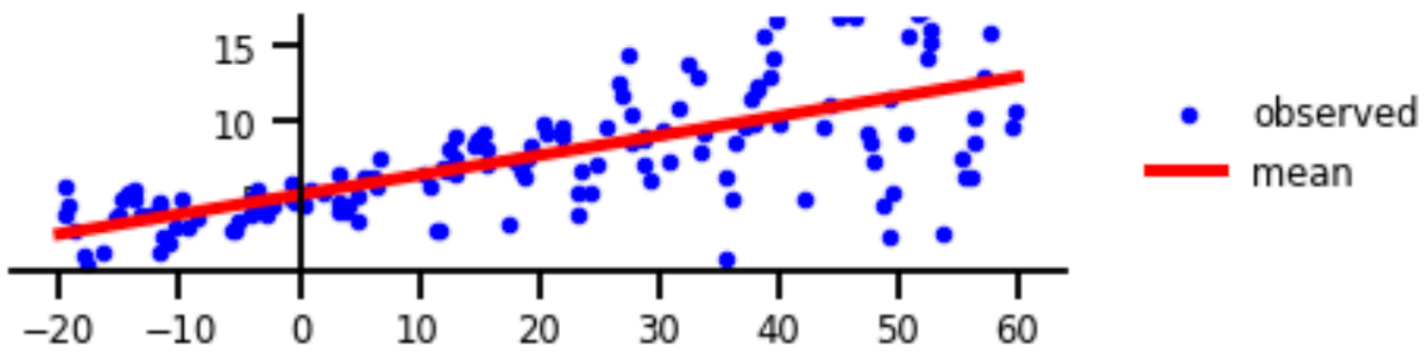
Uses the custom loss for the fitting

Probabilistic Neural Networks

Homoscedastic Probabilistic TF model

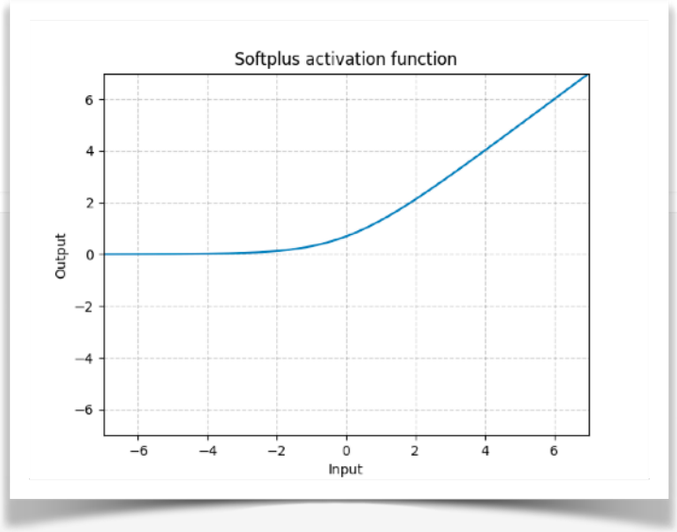
```
# Build model.
model = tf.keras.Sequential([
  tf.keras.layers.Dense(1),
  tfp.layers.DistributionLambda(lambda t: tfd.Normal(loc=t, scale=1)),
])

# Do inference.
model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.01), loss=negloglik)
model.fit(x, y, epochs=1000, verbose=False);

# Profit.
[print(np.squeeze(w.numpy())) for w in model.weights];
yhat = model(x_tst)
assert isinstance(yhat, tfd.Distribution)
```

1 2 3 4



Probabilistic Neural Networks

Heteroscedastic Probabilistic TF model



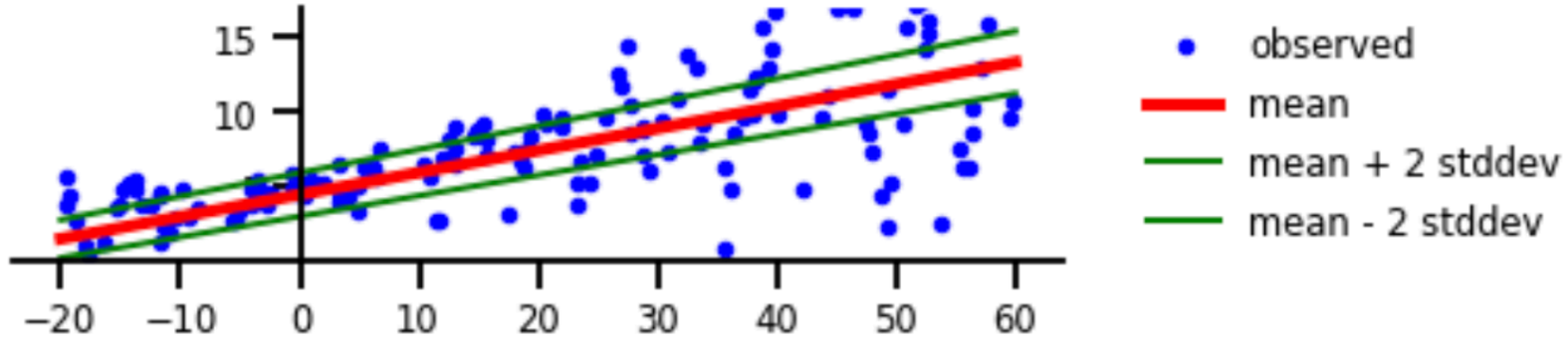

```python
# Build model.
model = tf.keras.Sequential([
  tf.keras.layers.Dense(1 + 1),
  tfp.layers.DistributionLambda(
      lambda t: tfd.Normal(loc=t[..., :1],
                           scale=1e-3 + tf.math.softplus(0.05 * t[...,1:]))),
])

# Do inference.
model.compile(optimizer=tf.optimizers.Adam(learning_rate=0.01), loss=negloglik)
model.fit(x, y, epochs=1000, verbose=False);

# Profit.
[print(np.squeeze(w.numpy())) for w in model.weights];
yhat = model(x_tst)
assert isinstance(yhat, tfd.Distribution)
```

1

2

3

4



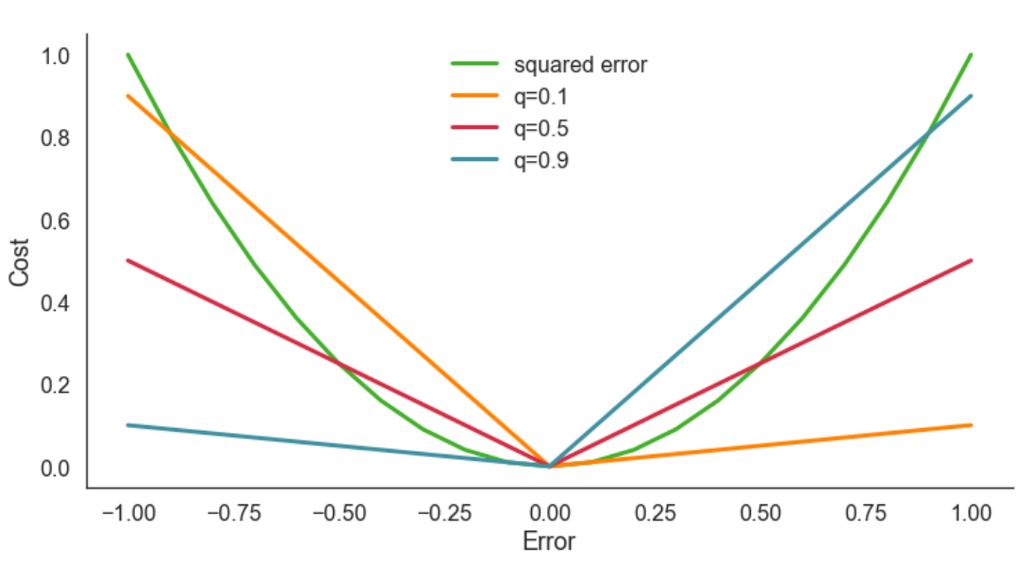
Quantile Regression: A Distribution-Free approximation

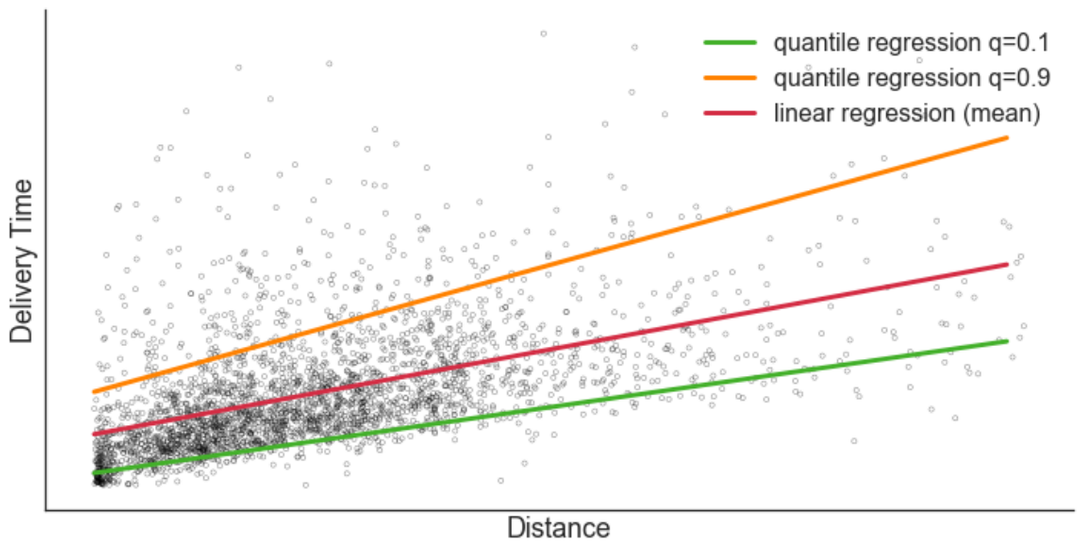
In statistics and econometrics, an extension to classic regression has been proposed: **Quantile Regression (QR)**.

Given $\tau \in (0,1)$, the original τ-th QR loss function is defined as

$$\mathscr{L}_{\tau}(x, y; w) = \left(y - w \cdot x\right) \cdot (\tau - I[y < w \cdot x])$$

where $I\left[p\right]$ is the indicator function that verifies the condition p.





Quantile Regression: A Distribution-Free approximation

Aleatoric uncertainty estimation

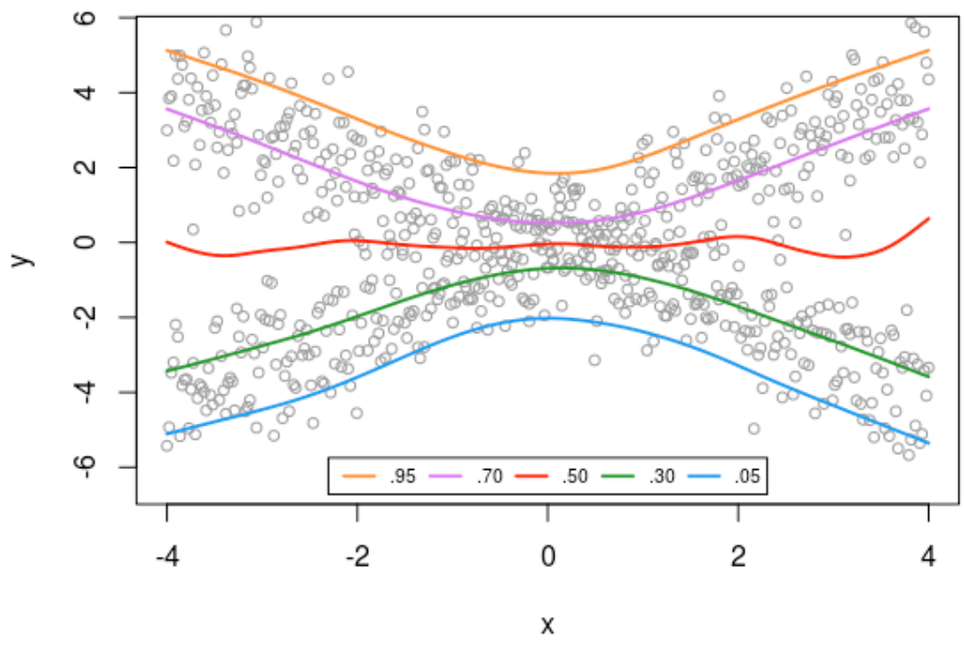
Given $\tau \in (0,1)$, the τ-th QR loss function with a NN, $\phi(x)$, is defined as

$$\mathscr{L}_{\tau}(x, y; w) = \Big(y - \phi_{w}(x)\Big) \cdot (\tau - I[y < \phi_{w}(x)])$$

where $I\big[p\big]$ is the indicator function that verifies the condition p.

How can we learn <u>several quantiles simultaneously</u>…?

$$\mathscr{L}_{\tau}(x, y; \boldsymbol{w}) = \sum_{\tau \in \{0.05, 0.5, 0.95\}} \Big(y - \phi_{\tau}(x)\Big) \cdot \Big(\tau - I\big[y < \phi_{\tau}(x)\big]\Big)$$



Quantile Regression: A Distribution-Free approximation

Aleatoric uncertainty estimation

The main advantage of QR is that we are directly estimating a confidence interval is a **way that does not depend on any parametric assumtion**!

Delving into the details

The probabilistic sources of uncertainty

Formally, in regression problems we have two random variables; the **dependent** variable, Y, with samples $y_i \in \mathbb{R}$ and the **independent** one, X, with samples $\boldsymbol{x}_i \in \mathbb{R}^D, D \geq 1$.

In this probabilistic context, we assume that the data set $\mathscr{D} = \left\{(\boldsymbol{x}_i, y_i) \mid \boldsymbol{x}_i \in \mathbb{R}^D, y_i \in \mathbb{R}\right\}_{i=1}^{n}$ is sampled from an unknown joint distribution, that can be written in this way:

$$p\left(\boldsymbol{X}, Y\right) = p\left(Y \mid \boldsymbol{X}\right) p(\boldsymbol{X}) \approx \sum_{i=1}^{N} \underbrace{p\left(Y \mid \boldsymbol{X}, M_i\right)}_{Aleatoric} \underbrace{p\left(M_i \mid \boldsymbol{X}\right)}_{Epistemic} p(\boldsymbol{X}),$$

Sources of uncertainty

where $\{M_i\}_{i=1}^{N}$ represents the set of all possible models (all possible parameter sets) from a given family. This is correct if $\{M_i\}_{i=1}^{N}$ is a family of highly expressive models (such as NN).

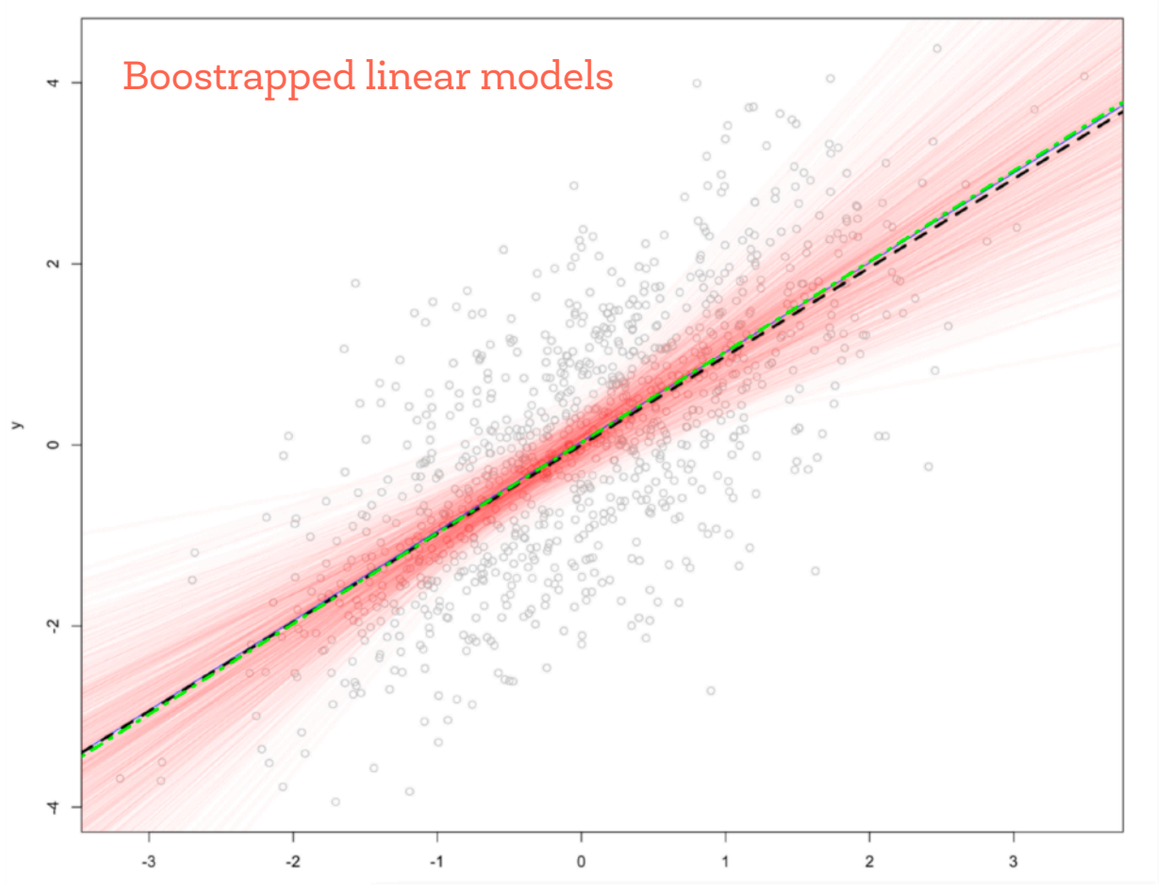
The probabilistic sources of uncertainty

$$p\left(Y \mid \boldsymbol{X}\right) \approx \sum_{i=1}^{N} p\left(Y \mid \boldsymbol{X}, M_i\right) \underbrace{p\left(M_i \mid \boldsymbol{X}\right)}_{Epistemic},$$

The probabilistic NN we have seen modeled aleatoric uncertainty, but they didn't consider **epistemic uncertainty**!

What is the meaning of epistemic uncertainty in this context?

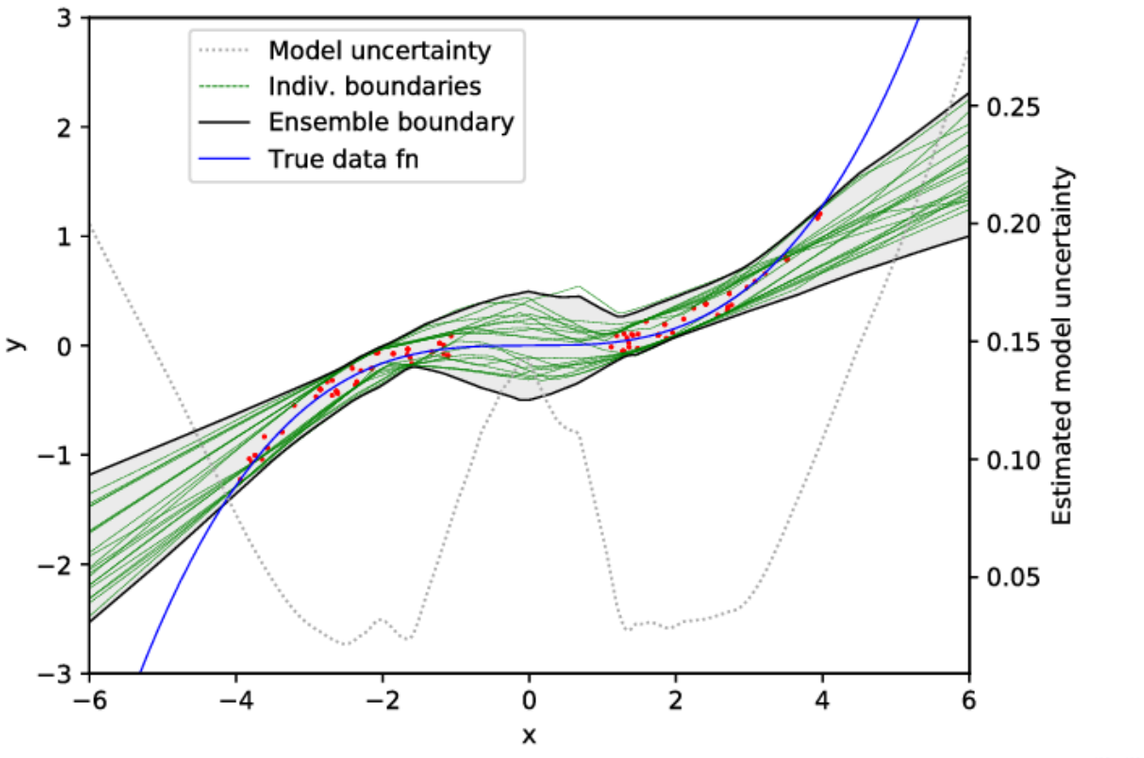
Uncertainty about the parameters!



Deep Ensembles: A naive but powerful baseline for Epistemic Uncertainty Estimation

One of the simplest but most effective ways to measure epistemic uncertainty is to consider ensembles of models instead of a monolitic model.

The core idea behind ensembling is that by having a committee of models, different strengths will complement one another, and many weaknesses will cancel each other out.



685 x 482

We can use different strategies to build the ensemble:

- Different snapshots of the same model (i.e., model trained after 1, 10, 100 epochs).
- Different solutions of the same model (i.e., trained with different random initializations each time).

Integrating all: A simple recipe

$$p\left(Y \mid \boldsymbol{X}\right) \approx \sum_{i=1}^{N} \underbrace{p\left(Y \mid \boldsymbol{X}, M_i\right)}_{Aleatoric} \underbrace{p\left(M_i \mid \boldsymbol{X}\right)}_{Epistemic}$$

1. Select a neural network architecture for learning the conditional distribution.

2. Re-train several (diverse) instances of the model, $\{M_i\}_{i=1}^{N}$.

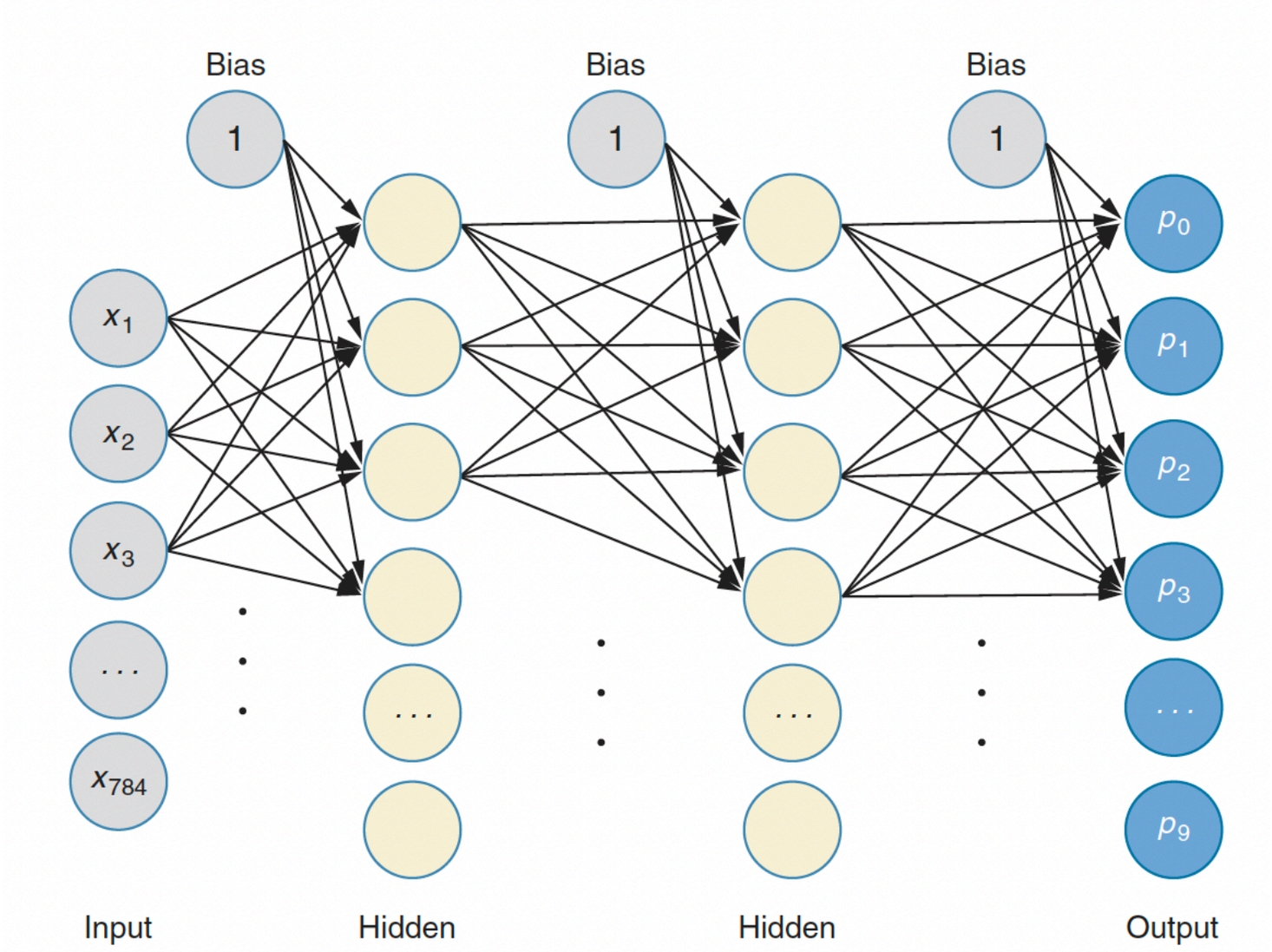
3. Evaluate each parametrized distribution of the ensemble and calculate the expected value of all these evaluations.

For instance,

$$p\left(Y \mid \boldsymbol{X}\right) \approx \frac{1}{M} \sum_{i=1}^{M} \sum_{\{x_j, y_j\} \in \mathscr{D}} \log \mathscr{N}(\mu_i(x_j), \sigma_i(x_j); y_j)$$

Neural networks and the probabilistic point of view. The case of classification.

Probabilistic Neural Networks



$$p_j = \frac{e^{a_j}}{\sum_{k=1}^{C} e^{a_k}}$$

$$H(y, p) = -\sum_{j=1}^{C} y_j \log(p_j)$$

Cross-Entropy Loss with SoftMax function.

Probabilistic Neural Networks

According to maximum likelihood estimation, we maximize $p(Y \mid X)$, which is equivalent to minimizing the **negative log likelihood**:

$$-\log p(Y \mid X) = \sum_{i=1}^{n} -\log p(y^i \mid x^i) \overset{?}{=} \sum_{i=1}^{n} \sum_{j=1}^{C} y_j^i \log p_j^i$$

Samples — Labels

This is called the cross-entropy loss.

Is there a connection between this loss and a probabilistic model?

Probabilistic Neural Networks

Cross-entropy can be seen as a measure of the **difference between two probability distributions** Y **and** P, for a given random variable C.

Cross-entropy can be calculated using the probabilities of the events c from Y and P, as follows:

$$H(Y,P) = -\sum_{c=1}^{C} y_c \log(p_c)$$

What we do is to compute the diference between our estimated distribution and a categorical distribution:

$$\underset{c_1\ c_2\ c_3}{\underset{P}{[0.2,0.7,0.1]}} \text{ vs } \underset{c_1\ c_2\ c_3}{\underset{Y}{[0,1,0]}}$$

Probabilistic Neural Networks

In classification problems (assuming a **multinomial model**) we want to estimate the probability of different outcomes.

If we make a single observation, and we observe outcome $j \in C$, then the likelihood is simply p_j:

$$\text{if } j = c_2 \text{ and } p = [\underset{c_1}{0.6}, \underset{c_2}{0.4}, \underset{c_3}{0.0}], \text{ then } p(c_2 \mid x) = p_2 = 0.4.$$

If we represent the actual observation as a vector y with one-hot encoding, then the **likelihood** **of the same single observation** can be represented as:

$$p = [0.6, 0.4, 0.0]$$

$$y = [0.0, 1.0, 0.0]$$

$$\prod_{j=1}^{C} p_j^{y_j}$$

$$p(c_2 \mid x) = p_0^0 \times p_2^1 \times p_3^0 = 0.4$$

since each term in the product except that corresponding to the observed value will be equal to 1.

Probabilistic Neural Networks

The negative log likelihood of the observation is then:

$$-\log \prod_{j=1}^{C} p_j^{y_j} = -\sum_{j=1}^{C} y_j \log p_j$$

Under this interpretation, the expression for the negative log likelihood above is also equal to the cross entropy!

That's why we minimize cross-entropy.

Uncertainty quality: Calibration

Calibration

What does it mean for one model to have better representation of its uncertainty than another?

The meteorological community has carefully considered this question and developed a set of proper scoring rules that a comparison function for probabilistic weather forecasts should satisfy in order to be well-calibrated, while still rewarding accuracy.

https://arxiv.org/pdf/1906.02530.pdf

Calibration

What does it mean for one model to have better representation of its uncertainty than another?

The meteorological community has carefully considered this question and developed a set of proper scoring rules that a comparison function for probabilistic weather forecasts should satisfy in order to be well-calibrated, while still rewarding accuracy.

Does predicted probability of correctness (confidence) match the observed frequency of correctness (accuracy)?

https://arxiv.org/pdf/1906.02530.pdf

Calibration

What does it mean for one model to have better representation of its uncertainty than another?

Of all the days where the model **predicted rain with 80% probability**, what fraction did we observe rain?

- 80% implies **perfect calibration**
- Less than 80% implies model is overconfident
- Greater than 80% implies model is under-confident

https://arxiv.org/pdf/1906.02530.pdf

Calibration

We can compute an estimate of the calibration error with a test dataset.

Expected Calibration Error

using a test dataset

Bin the probabilities in B bins

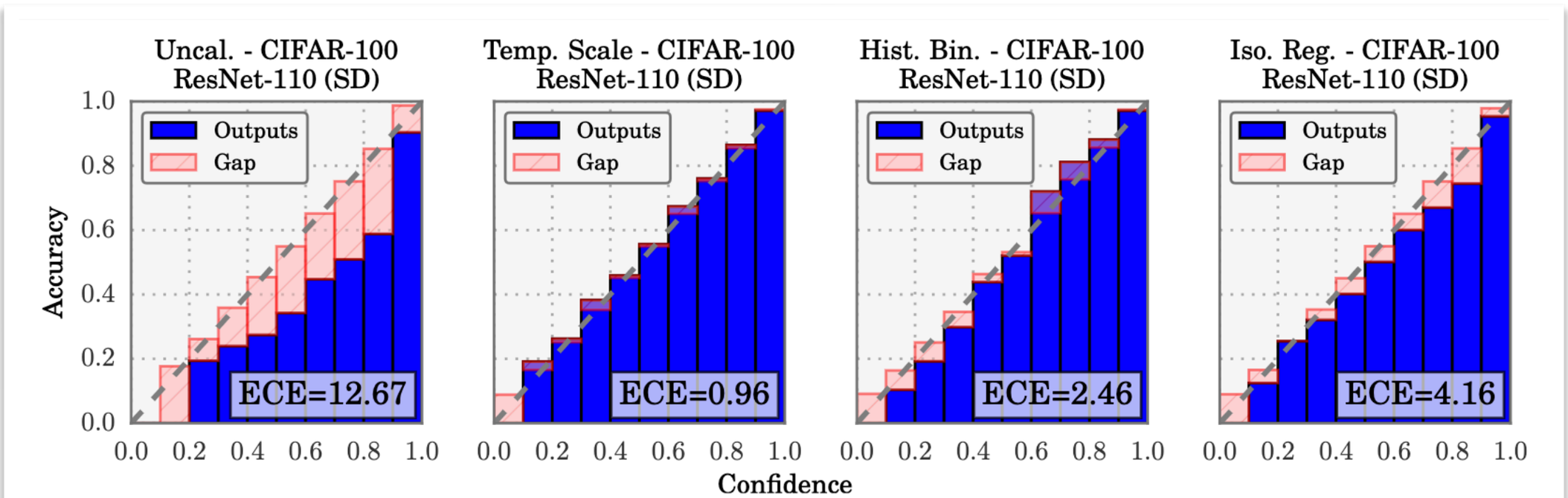
Compute the **within-bin accuracy** and **within-bin predicted confidence** with your dataset

Average the (weighted) calibration error between bins

$$\text{ECE} = \sum_{b=1}^{B} \frac{n_b}{N} \left| \text{accuracy}(b) - \text{confidence}(b) \right|$$

https://arxiv.org/pdf/1906.02530.pdf

Calibration

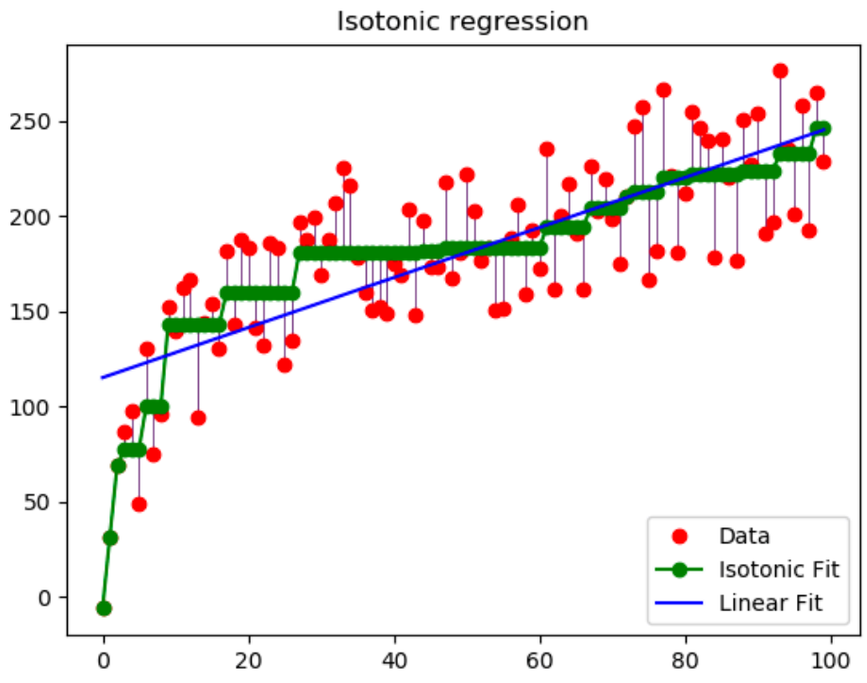


https://arxiv.org/abs/1706.04599

Calibration

There are two popular approaches to calibrating probabilities; they are the Platt Scaling and **Isotonic Regression.**

In statistics, isotonic regression or monotonic regression is the technique of fitting a free-form line to a sequence of observations under the following constraints: the fitted free-form line has to be non-decreasing (or non-increasing) everywhere, and it has to lie as close to the observations as possible.



https://docs.w3cub.com/scikit_learn/auto_examples/plot_isotonic_regression

Calibration

Calibrating a classifier consists of fitting a regressor (called a calibrator) that maps the output of the classifier to a calibrated probability in [0, 1].

The 'isotonic' method fits a non-parametric isotonic regressor, which outputs a step-wise non-decreasing function. It minimizes:

$$\sum_{i=1}^{n} (y_i - \hat{f}_i)^2$$

subject to $\hat{f}_i >= \hat{f}_j$ whenever $f_i >= f_j$.

y_i is the true label of sample i and $\hat{f}_i$ is the output of the calibrated classifier.

Classification uncertainty

How uncertainty is represented in DL classification systems.

In most cases, DL classification systems produce a result that comes in the form of a probability distribution over a set of classes.

One of the most basic ways of representing **aletatoric uncertainty** in a system's predictions is to **rely on these probabilities** to decide whether or not to trust the outcome of the system.

( **These probabilities can lead to errors**, since they may not be well calibrated, their interpretation may not be intuitive, or, worse still, they may be falsely considered to be safe predictions.)

José Mena, Oriol Pujol, and Jordi Vitrià. 2021. A Survey on Uncertainty Estimation in Deep Learning Classification Systems from a Bayesian Perspective. ACM Comput. Surv. 54, 9, Article 193 (December 2022)

How to obtain an uncertainty measure from the output of probabilistic classifiers.

Output of **classification systems** has the shape of a probability distribution for the predicted classes, resulting from the application of a softmax operation at the output logits.

Thus, a first approach to obtaining a measure of the uncertainty is by observing the output distribution itself.

José Mena, Oriol Pujol, and Jordi Vitrià. 2021. A Survey on Uncertainty Estimation in Deep Learning Classification Systems from a Bayesian Perspective. ACM Comput. Surv. 54, 9, Article 193 (December 2022)

How to obtain an uncertainty measure from the output of probabilistic classifiers.

Assuming a calibrated and correct prediction we can derive the the following metrics/heuristics (**uncertainty scores**) from observing the predictions of the original model:

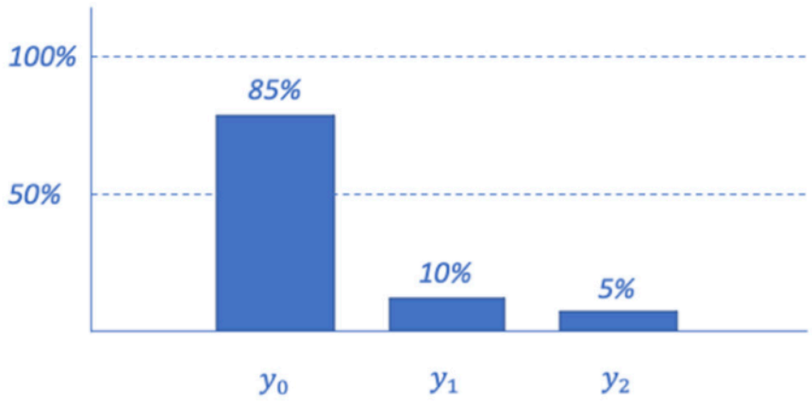
— **Softmax response**: in probabilistic classifiers, it corresponds to the probability of the predicted class.
— **Predictive entropy**: the entropy of the `softmax` probability distribution, as defined by information theory.
— **Least confidence**: the difference between the most confident prediction and 100% confidence.
— **Margin of confidence**: difference between the top two most confident predictions.
— **Ratio of confidence**: ratio between the top two most confident predictions.

José Mena, Oriol Pujol, and Jordi Vitrià. 2021. A Survey on Uncertainty Estimation in Deep Learning Classification Systems from a Bayesian Perspective. ACM Comput. Surv. 54, 9, Article 193 (December 2022)

How to obtain an uncertainty measure from the output of probabilistic classifiers.

SUMMARIZING UNCERTAINTY FOR PROBABILISTIC CLASSIFIERS

Example of a three classes prediction

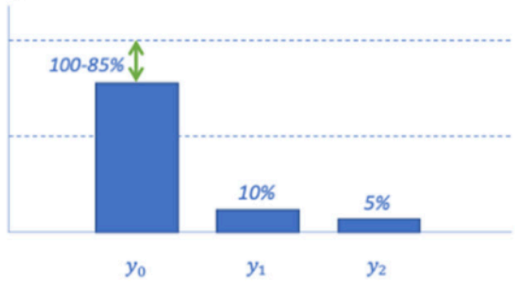


Softmax response: probability of the predicted class, 85%

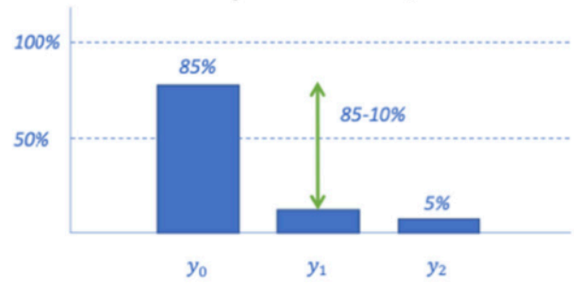
Predictive entropy: entropy of the softmax probability distribution

$$-\frac{(0.85\,log2(0.85))+(0.1\,log2(0.1))+(0.05\,log2(0.05))}{log2\,(3)} = 0.47$$

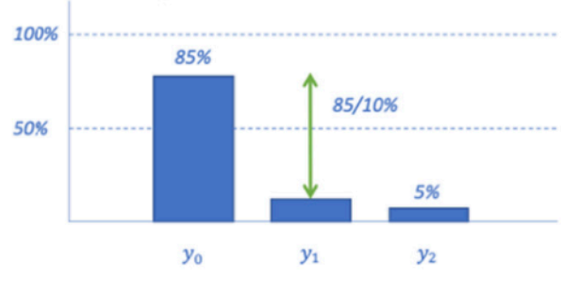
Least confidence: difference between the most confident prediction and 100% confidence, 15%



Margin of confidence: difference between the top two most confident predictions, 75%



Ratio of confidence: ratio between the top two most confident predictions, 8.5



José Mena, Oriol Pujol, and Jordi Vitrià. 2021. A Survey on Uncertainty Estimation in Deep Learning Classification Systems from a Bayesian Perspective. ACM Comput. Surv. 54, 9, Article 193 (December 2022)

How to obtain an uncertainty measure from the output of probabilistic classifiers.

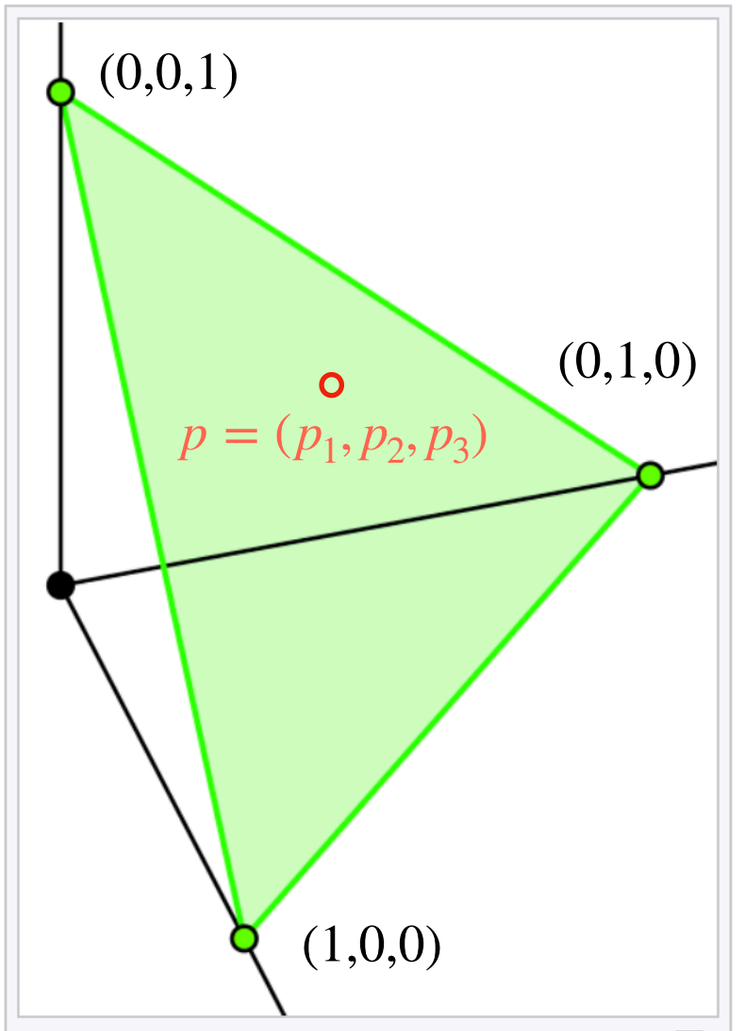
All of the metrics introduced so far obtain an aleatoric uncertainty score.

What about the epistemic uncertainty?

We can take into consideration a probability distribution of the multinomial parameters. The best option is to use a Dirichlet distribution.

José Mena, Oriol Pujol, and Jordi Vitrià. 2021. A Survey on Uncertainty Estimation in Deep Learning Classification Systems from a Bayesian Perspective. ACM Comput. Surv. 54, 9, Article 193 (December 2022)

How to obtain an uncertainty measure from the output of probabilistic classifiers.



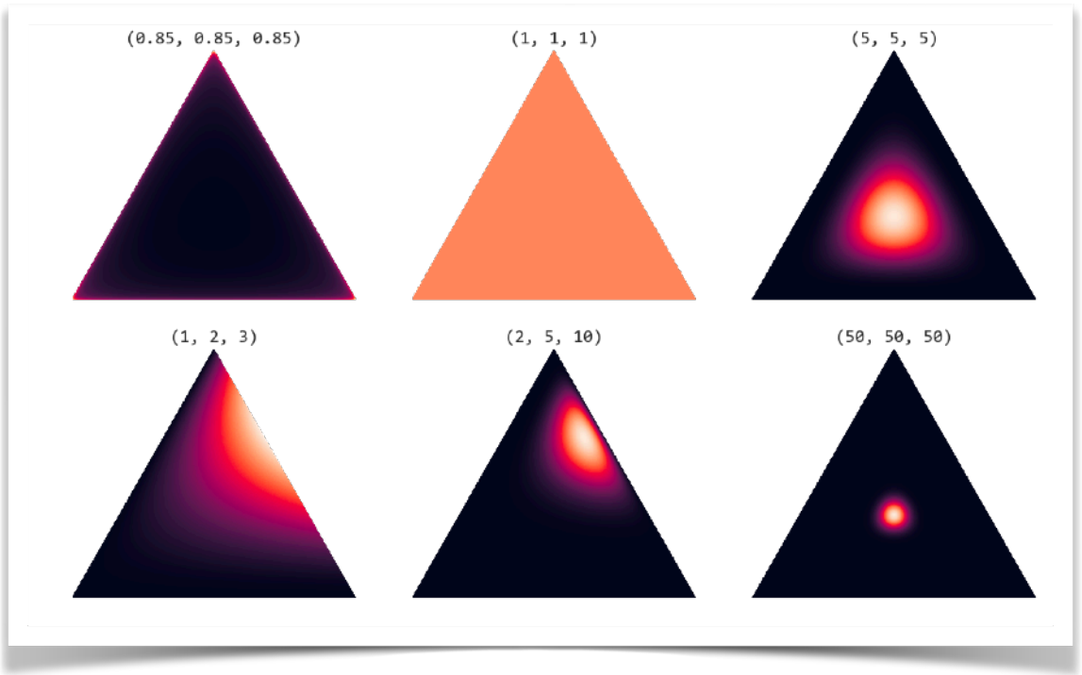
The possible probabilities for the categorical distribution with $k = 3$ are the 2-simplex $p_1 + p_2 + p_3 = 1$, embedded in 3-space.

https://en.wikipedia.org/wiki/Categorical_distribution

How to obtain an uncertainty measure from the output of probabilistic classifiers.

Here we consider the output distribution coming from a **Dirichlet probability density function**, thus $p(y \mid X, w^*) \sim Dir(\alpha)$.

α is called the concentration parameter of the Dirichlet distribution.

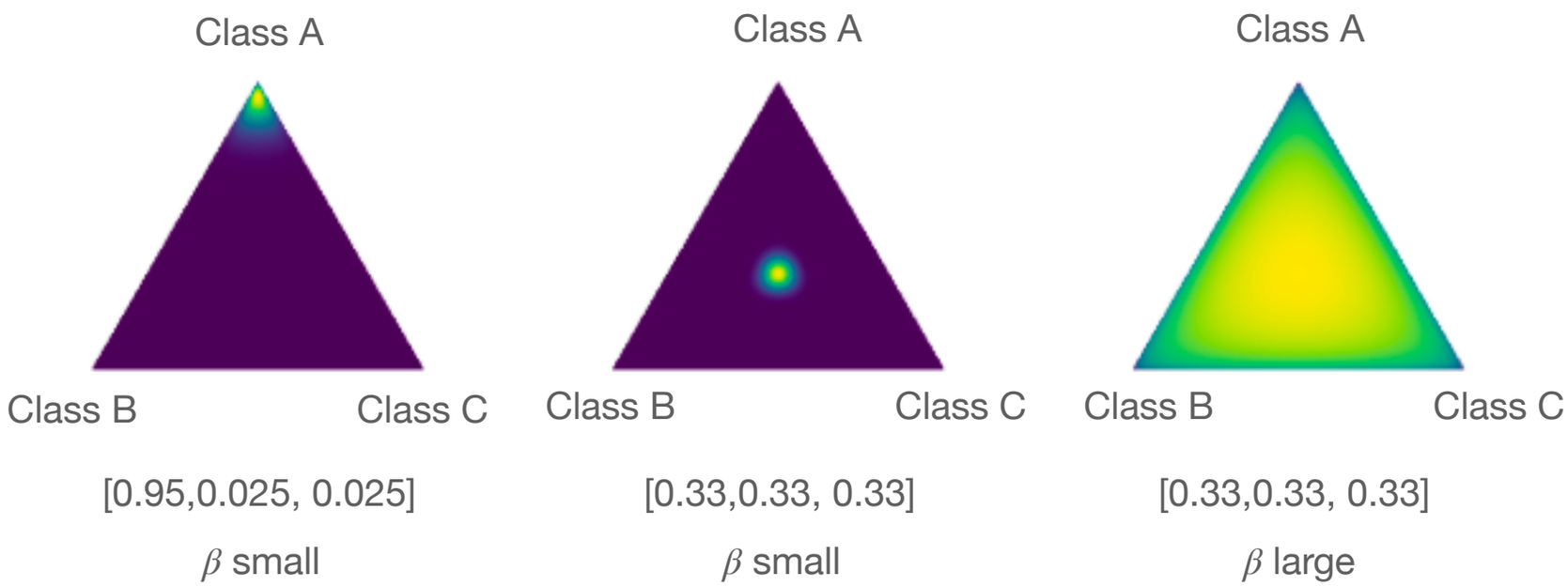


José Mena, Oriol Pujol, and Jordi Vitrià. 2021. A Survey on Uncertainty Estimation in Deep Learning Classification Systems from a Bayesian Perspective. ACM Comput. Surv. 54, 9, Article 193 (December 2022)

How to obtain an uncertainty measure from the output of probabilistic classifiers.

To relate the output of the classifier with the concentration parameter in the Dirichlet distribution, we can consider a decomposition of the concentration parameter in two terms: $\alpha = \beta \mathbf{y}$

The role of this scalar β is to control the spread of the distribution around the expected value, i.e. the original prediction $\mathbf{y}$.



José Mena, Oriol Pujol, and Jordi Vitrià. 2021. A Survey on Uncertainty Estimation in Deep Learning Classification Systems from a Bayesian Perspective. ACM Comput. Surv. 54, 9, Article 193 (December 2022)