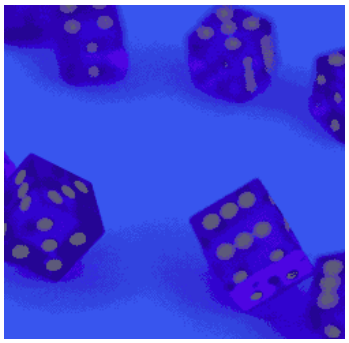
EE 046746 - Technion - Computer Vision

Elias Nehme

Tutorial 08 - Uncertainty Quantification in Deep Learning

Agenda

Why We Need Uncertainty?



- Image source

With great power..

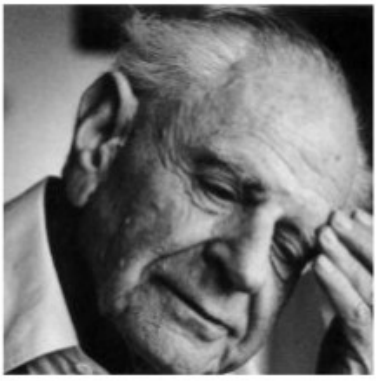
- Control handed-over to automated systems; with many dangerous scenarios
  - Medical diagnosis
  - Automotive and Self-driving Cars
  - High frequency Trading (affecting economic markets on global scale)
  - etc.
- Image source

Knowing what you don't know

- A deep learning model should be able to say: “sorry, I don’t know”
- If it can tell us how certain it is we can plan accordingly

It is correct, somebody might say, that (...) Socrates did not know anything; and it was indeed wisdom that they recognized their own lack of knowledge, (...).

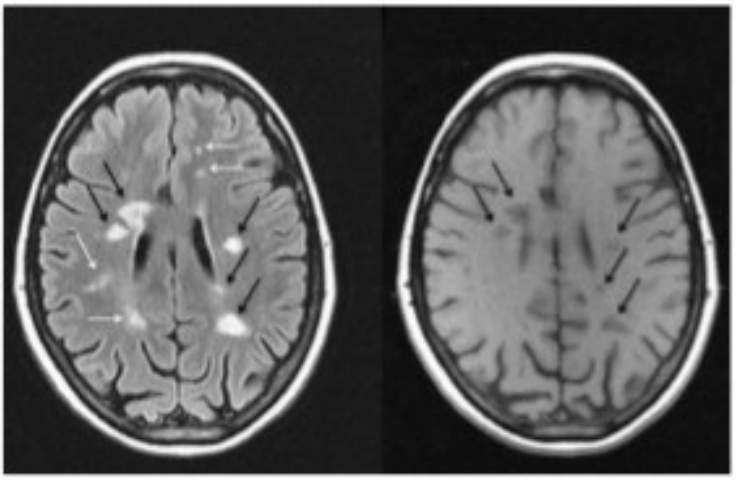
—Karl R. Popper, *The World of Parmenides*



- Image source

Diagnosis in medicine

- Suppose you are trying to classify lesions in MRI

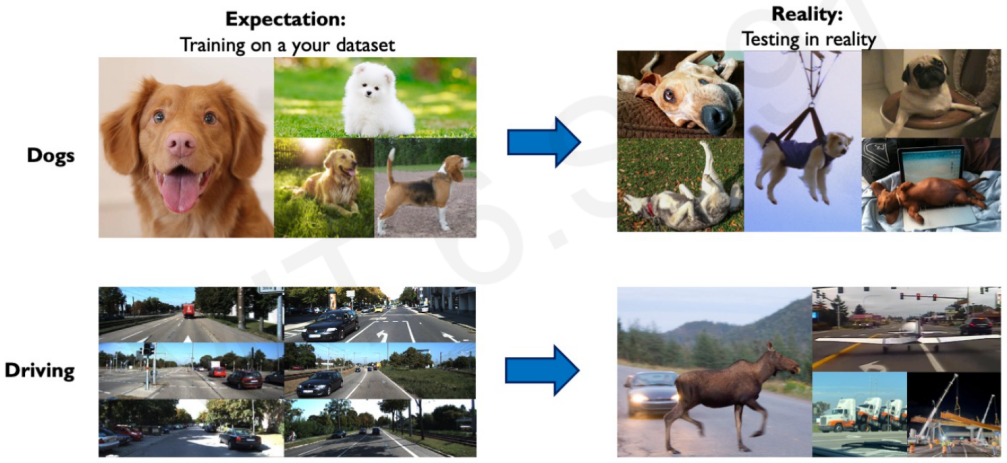


Different MRI contrasts (T2/T1). Source: http://www.msdiscovery.org. 2019.

- Later on: change in acquisition setup leads to differently looking samples
- Problem: we will still get predictions with **high probability** for an Out-of-distrbution (OOD) sample.
- Image source

Out-of-distribution detection

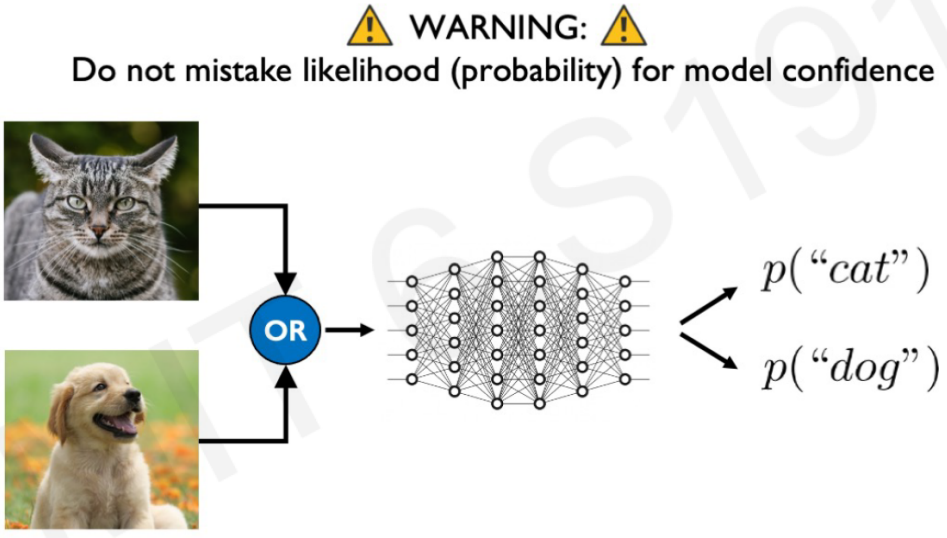
- Systems applied to toy data deployed in real-life settings



- Image source

Out-of-distribution detection

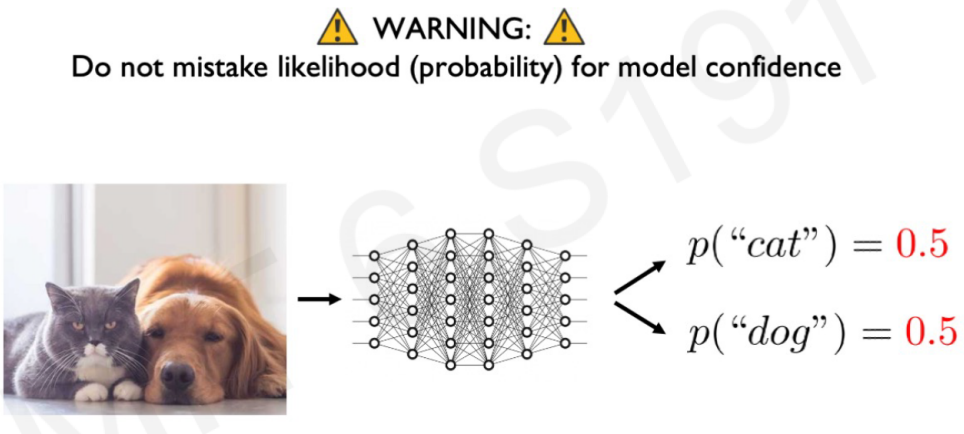
- Likelihood vs confidence



- Image source

Out-of-distribution detection

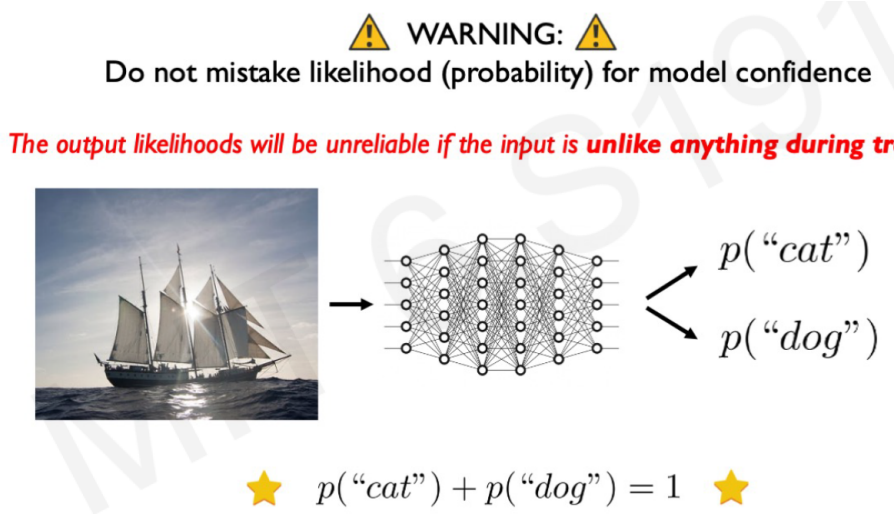
- Likelihood vs confidence



- Image source

Out-of-distribution detection

- Likelihood vs confidence



- Classifier forced to choose between 2 possible outcomes even if it have no clue
- July 2015, classification system erroneously identified two African American humans as gorillas, raising concerns of racial discrimination.

- Image source

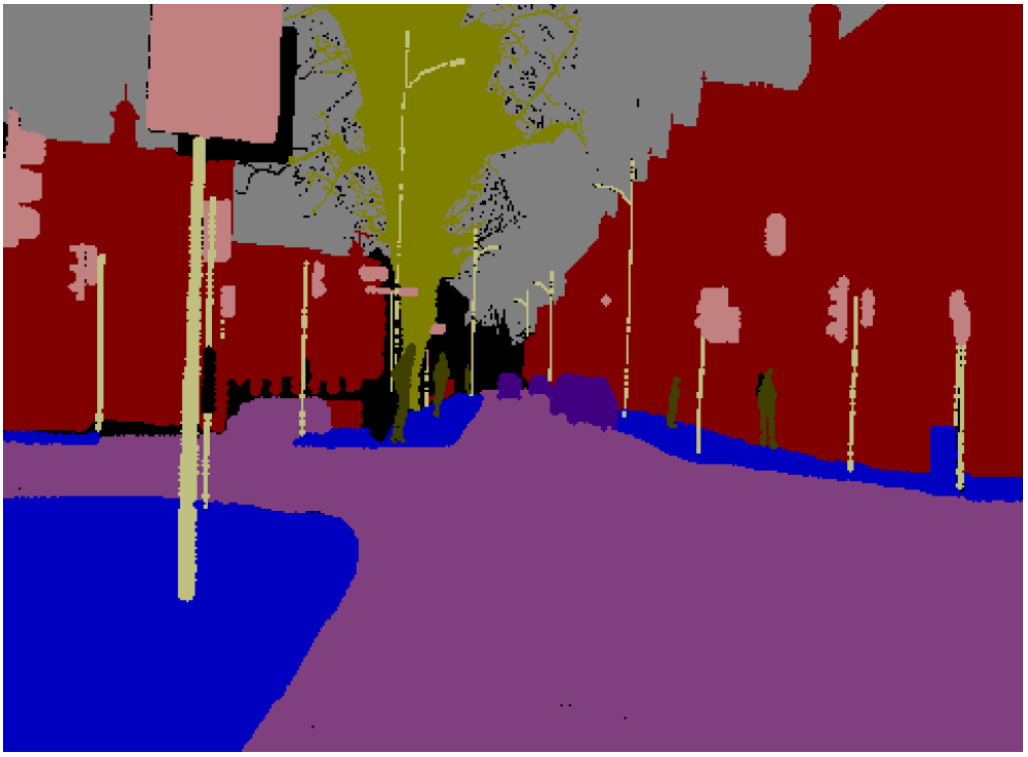
Autonomous Driving

- Semantic Segmentation used as a strong cue for motion planning

- Image source

Autonomous Driving

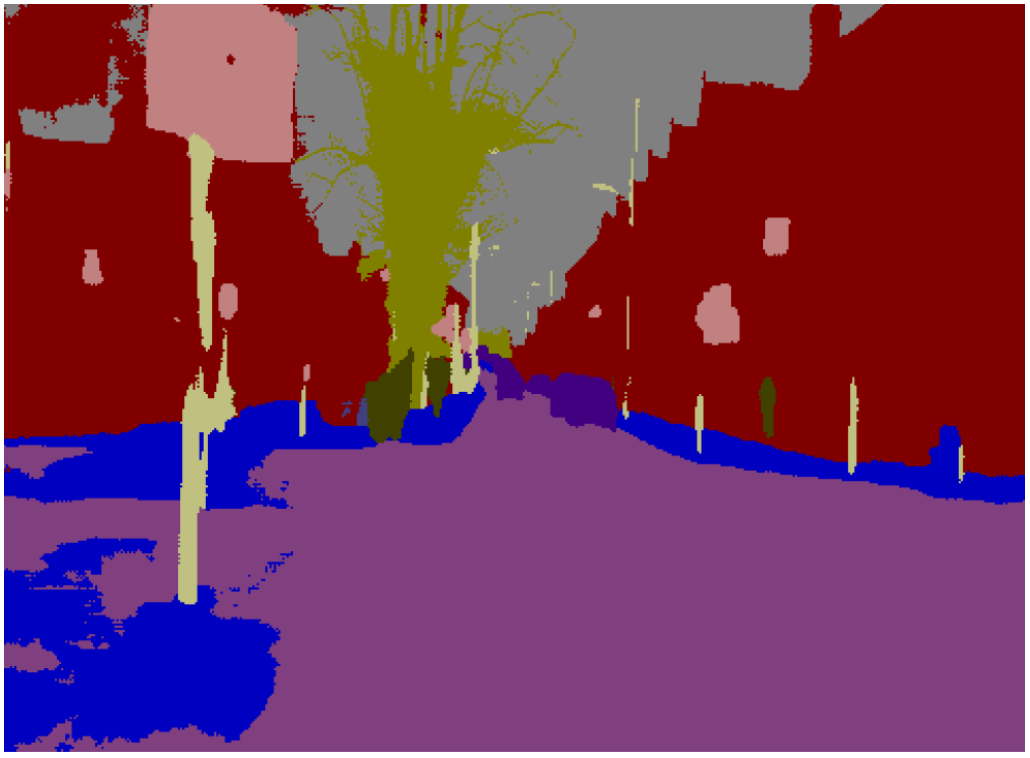
- Ground Truth Segmentation Labels



- Image source

Autonomous Driving

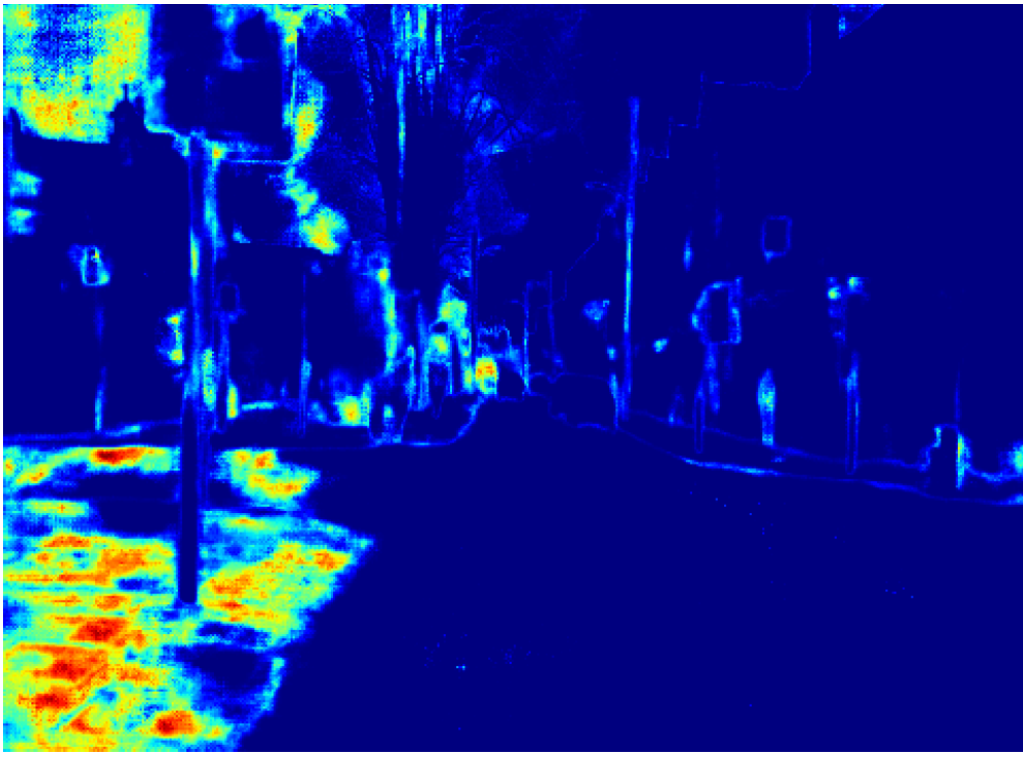
- Predicted Segmentation Labels



- Image source

Autonomous Driving

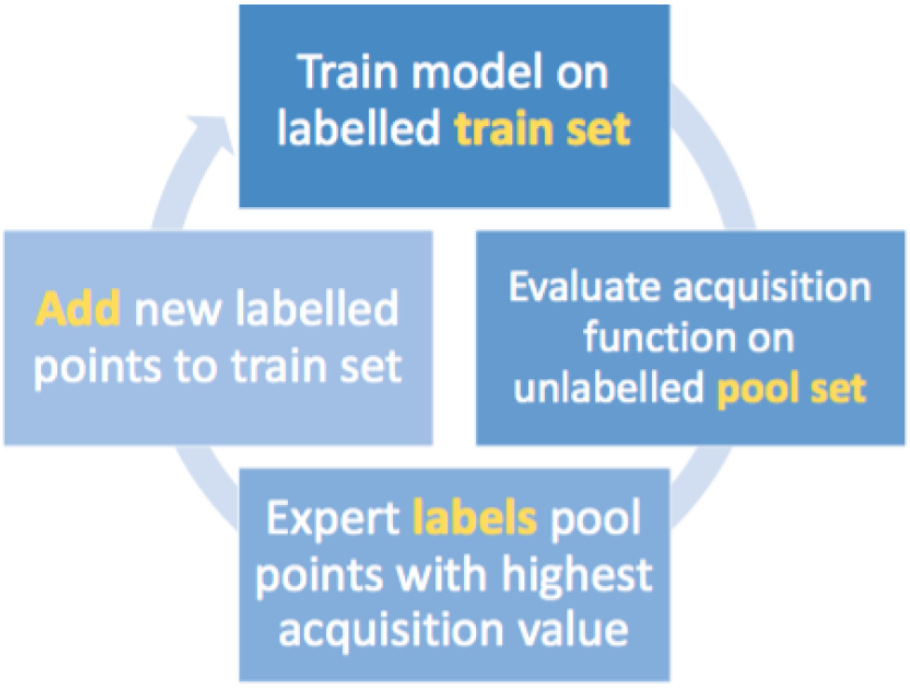
- Predicted Uncertainty per-pixel



- May 2016, tragically experienced the first fatality from an assisted driving system. White side of tractor trailer confused against a brightly lit sky.
- Image source

Active learning

- Agent chooses which unlabelled data is most informative
- Asks external \oracle" (eg human annotator) for a label only for that
- Acquisition function: ranks points based on their potential informativeness, e.g. model uncertainty



- Image source

- Many more applications including:
  - Explore/exploit dilemma in Reinforcement Learning

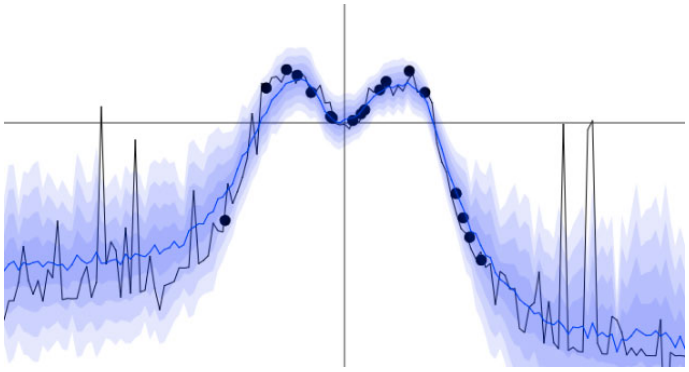
- Model understanding/Dataset bias understanding
- Sample-adaptive loss-function tuning
- etc

Types of Uncertainty

- Epistemic/Model Uncertainty
- Aleatoric/Data Uncertainty

Epistemic/Model Uncertainty

- Episteme in Latin is "Knowledge/understanding"
- Also called model uncertaintly or reducible uncertainty
- Describes the uncertainty in the model itself (network weights)
- High when we do not have good coverage of the sample space
- Can be reduced with more training data



- Cool online demo for epistemic uncertainty: https://mlg.eng.cam.ac.uk/yarin/blog_3d801aa532c1ce.html

Aleatoric/Data Uncertainty

- Aleator in Latin means "dice player"

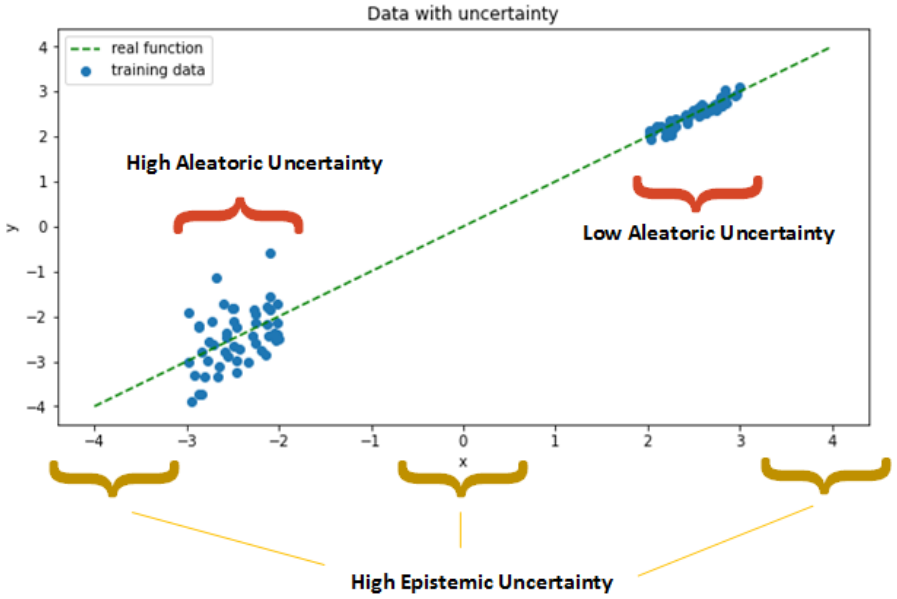
- Rolling a pair of dice again and again will not reduce the uncertainty
- Image source

Aleatoric/Data Uncertainty

- Also called data uncertainty or irreducible uncertainty
- Describes the confidence in the input data

- High when the input data is noisy
- Can not be reduced by adding more training data
- To decrease it, we need to acquire data with a better sensor (less noise)
- Further classified into:
  - homoscedastic - fixed for all samples
  - heteroscedastic - sample-dependent noise

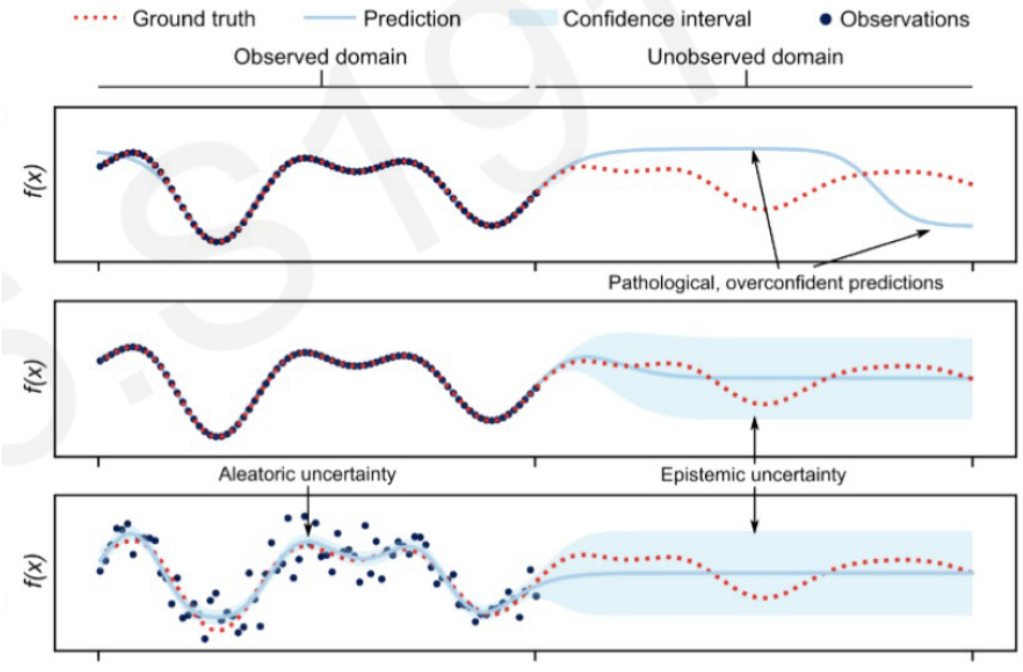
Visual Example 1



- Data generation model is given by $y_i = \omega_0 + \omega_1 x_i + \epsilon$
- Here we have heteroscedastic data uncertainty since $\epsilon = \epsilon(x_i)$

- Image source

Visual Example 2

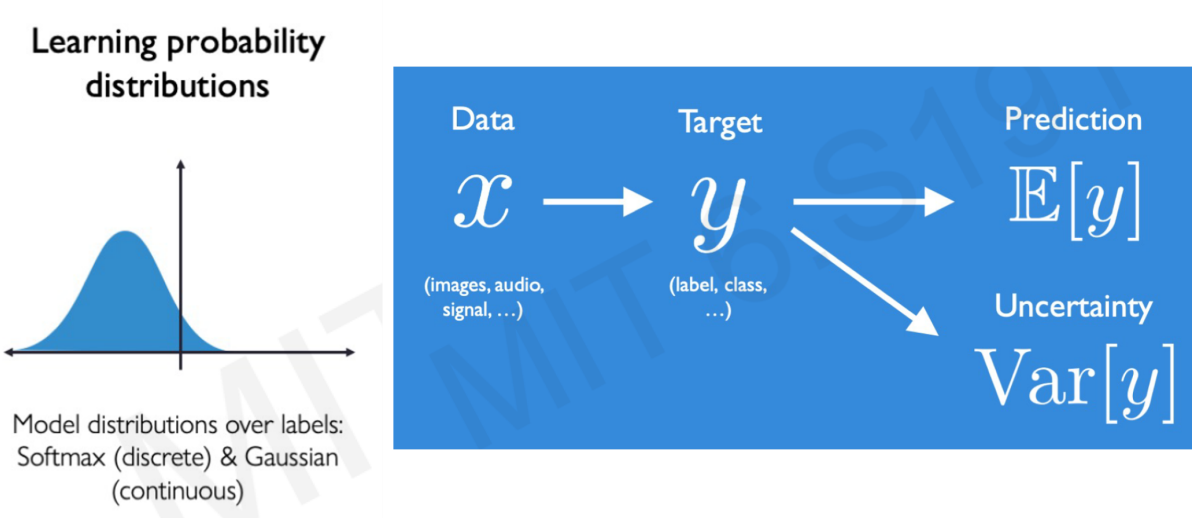


- Extrapolation is an obvious case of high epistemic uncertainty
- Image source

Computing Uncertainty

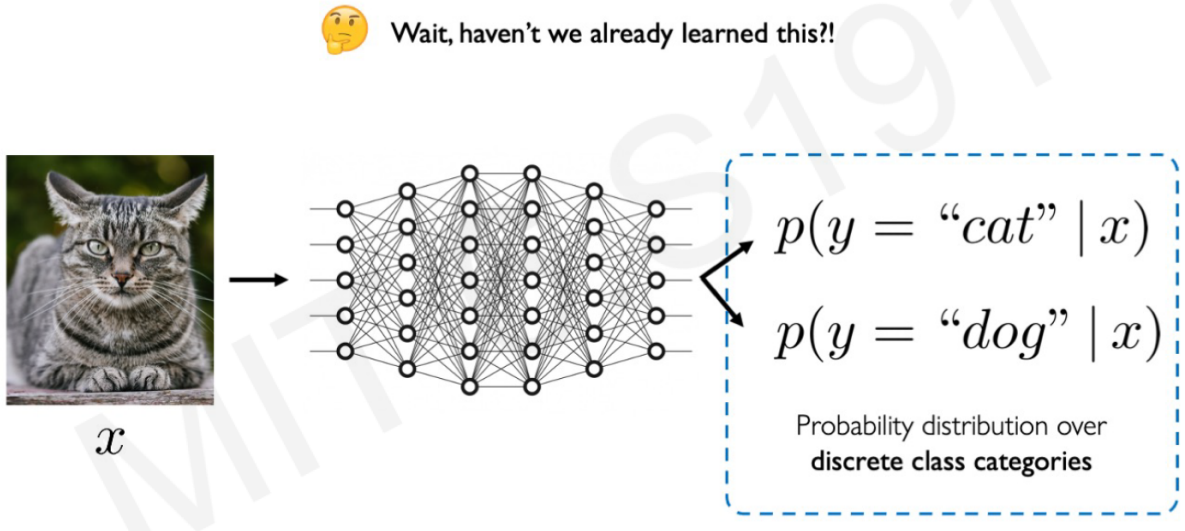
- Likelihood Modelling
- Bayesian Neural Networks
- Dropout at Test time

Likelihood Modelling



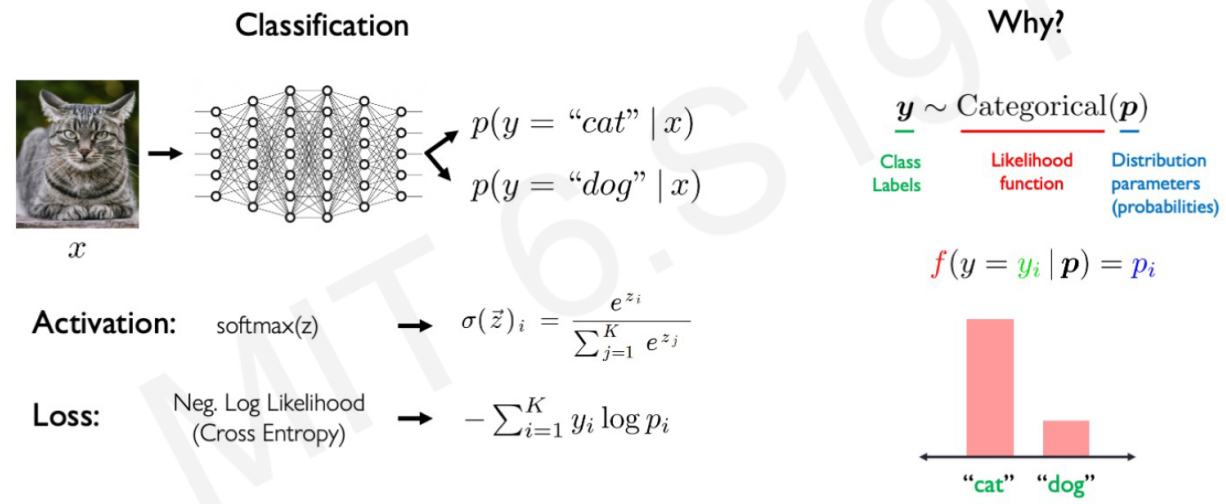
- Main idea is simple: swap point estimate with parameterized likelihood
- Image source

Likelihood Modelling: Classification



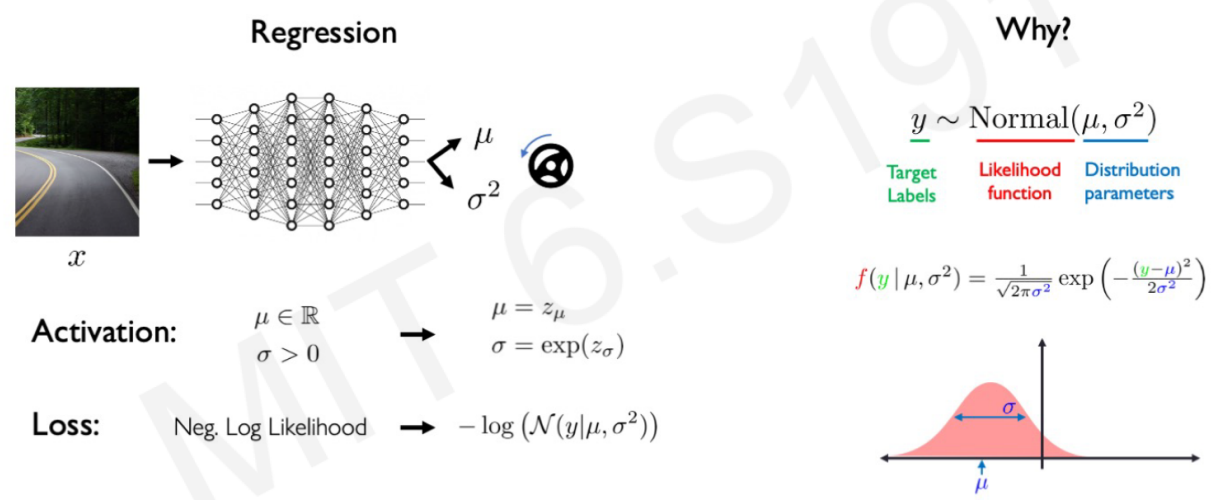
- Classification: Output is a probability distribution with Softmax and Cross-entropy
- Image source

Likelihood Modelling: Classification



- Underlying assumption is that the output is Categorially distributed
- Image source

Likelihood Modelling: Regression



- Same logic only assume the output is Gaussian
- Image source

Likelihood Modelling: Regression

- Where are the uncertainty "labels" coming from?
  - We do not need labels for uncertainty!
- The negative log-likelihood for a Gaussian distribution is composed of 2 terms ($\hat{y} \equiv \mu$):

$$-\log\left(\mathcal{N}\left(y|\mu,\sigma^2\right)\right) = \frac{1}{2}\frac{\left(y-\hat{y}\left(x\right)\right)^2}{\hat{\sigma}^2\left(x\right)} + \frac{1}{2}\log\left(\hat{\sigma}^2\left(x\right)\right) + C$$

- The first term favors very large uncertainty
- The second term favors very small uncertainty
- The compromise is reached such that for small errors the uncertainty is small
- Note that this requires another head for the net (to output sigma)

Likelihood Modelling: Summary

	Classification (discrete)	Regression (continuous)
Targets	$y \in \{1, \cdots, K\}$	$y \in \mathbb{R}$
Likelihood	$y \sim \text{Categorical}(\boldsymbol{p})$ `tfp.distributions.Categorical(probs=p)`	$y \sim \text{Normal}(\mu, \sigma^2)$ `tfp.distributions.Normal(mu, sigma)`
Parameters	$\boldsymbol{p} = \{p_1, \cdots, p_K\}$	(μ, σ^2)
Constraints	$\sum_i p_i = 1; \quad p_i > 0$	$\mu \in \mathbb{R}; \quad \sigma > 0$
Loss function	Cross Entropy $-\sum_{i=1}^{K} y_i \log p_i$ `dist.cross_entropy(y)`	Negative Log-Likelihood $-\log\left(\mathcal{N}(y \mid \mu, \sigma^2)\right)$ `-1 * dist.log_prob(y)`

- Image source

Bayesian Neural Networks

Estimating epistemic uncertainty

✓ Aleatoric uncertainty can be learned directly using neural networks

💪 Epistemic uncertainty is **much more challenging** to estimate

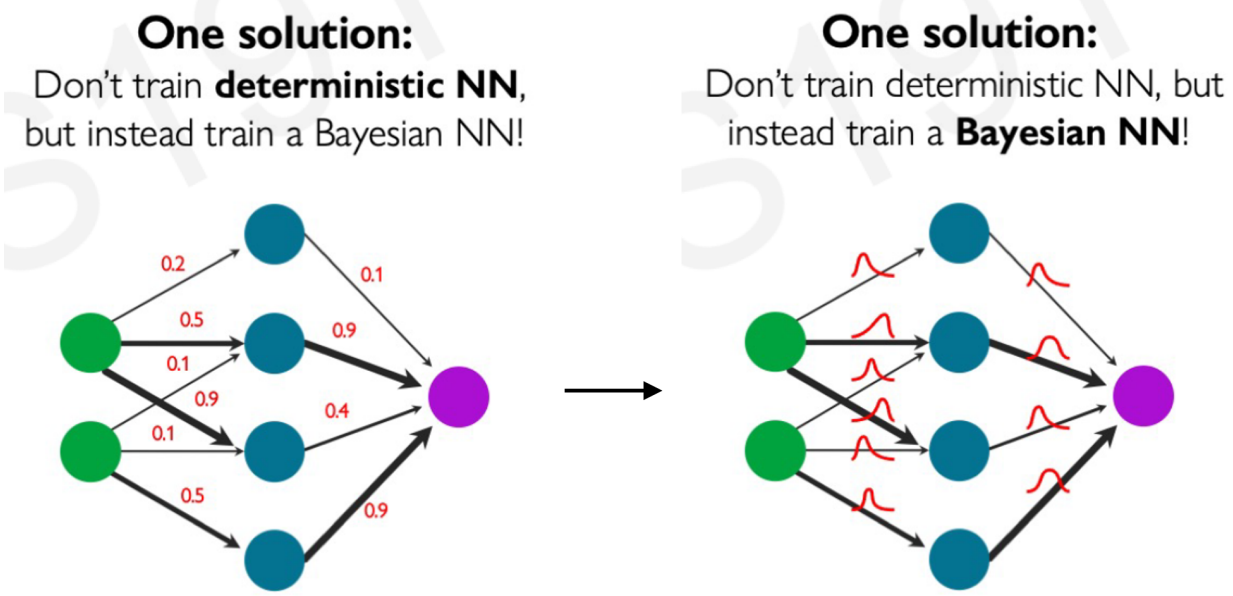
🤔 How can a model understand when it does not know the answer?

Knowing when we don't know is hard

…even for humans!

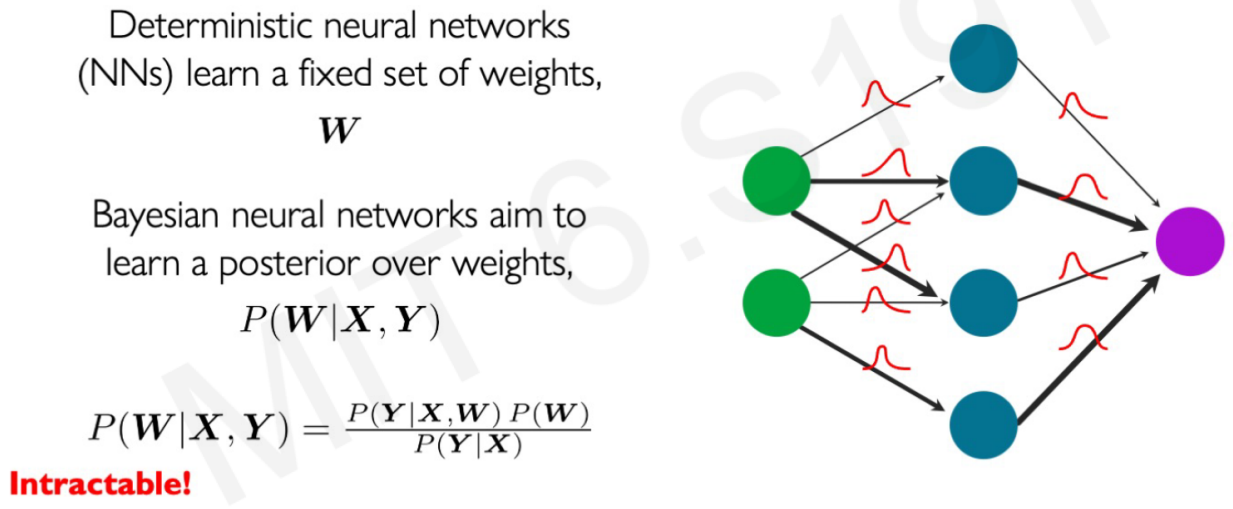
- Epistemic uncertainty is significantly harder to estimate
- Image source

Bayesian Neural Networks: key idea



- Main idea: put probability distrbutions over weights
- Image source

Bayesian Neural Networks: intractable directly



- Straighforward Bayesian inference is intractable
- Image source

Bayesian Neural Networks: stochastic variational inference

- The underlying approximations rely on "Stochastic Variational Inference".
- The posterior is approximated using a parametric distrbution (eg Gaussian)
- Optimization is performed using the "evidence lower bound" (ELBO)
- "Monte Carlo" (MC) Integration is involved to approximate the gradients.
- Still becomes extremely heavy for very large models.

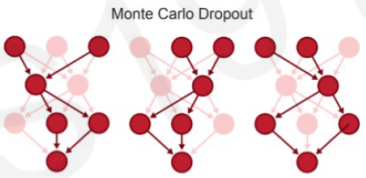
Bayesian Neural Networks: Efficient approximations

Approximations through sampling

Evaluate T stochastic forward passes using different samples of weights $\{\boldsymbol{W}_t\}_{t=1}^{T}$

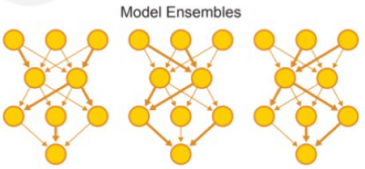
Dropout as a form of stochastic sampling

$z_{w,t} \sim Bernoulli(p) \quad \forall\, w \in \boldsymbol{W}$



Ensemble of T independently trained models, each learning a unique $\boldsymbol{W}_t$

$\boldsymbol{W}_t = train(f; \boldsymbol{X}, \boldsymbol{Y})$

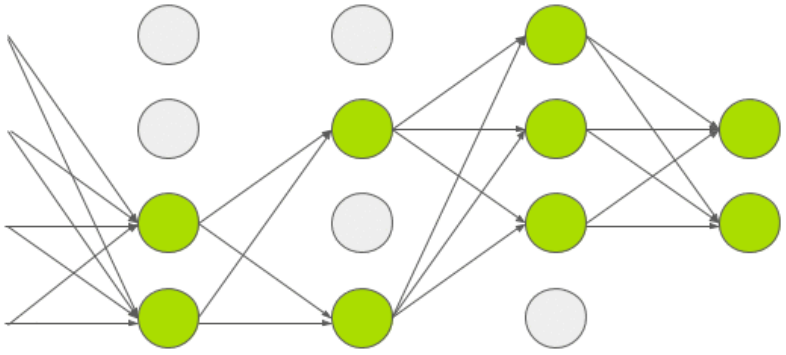


$$\mathbb{E}(\hat{\boldsymbol{Y}}|\boldsymbol{X}) = \frac{1}{T}\sum_{t=1}^{T} f(\boldsymbol{X}|\boldsymbol{W}_t) \qquad Var(\hat{\boldsymbol{Y}}|\boldsymbol{X}) = \frac{1}{T}\sum_{t=1}^{T} f(\boldsymbol{X})^2 - \mathbb{E}(\hat{\boldsymbol{Y}}|\boldsymbol{X})^2$$

- What? Dropout at test time?
- Image source

Dropout at Inference

- It can be proven that Dropout with L2 weight decay = approximate Bayesian inference of a Gaussian process.
- Main idea: Dropout **both** at **training** and at **test** time.
- Intuition: Training 2^m different models simultaneously, where m is the number of nodes in the network.
- For each batch, a randomly sampled set of these models is trained.



- Image source
- Icon source

Evidental Deep Learning

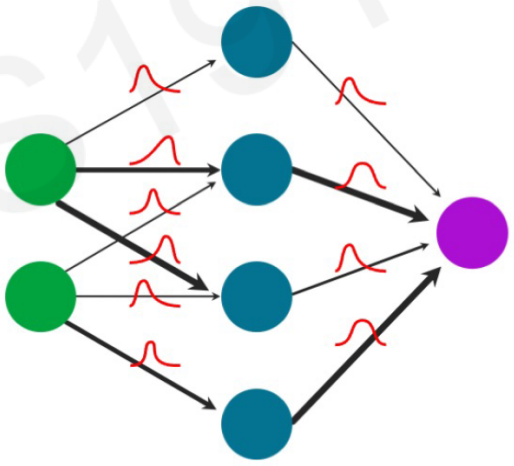
Downsides of Bayesian deep learning

Slow: Requires running the network T times for every input

Memory: Store T copies of the network in parallel

Efficiency: Sampling hinders real-time ability on edge devices (robotics)

Calibration: Sensitive to choice of prior and is often over-confident

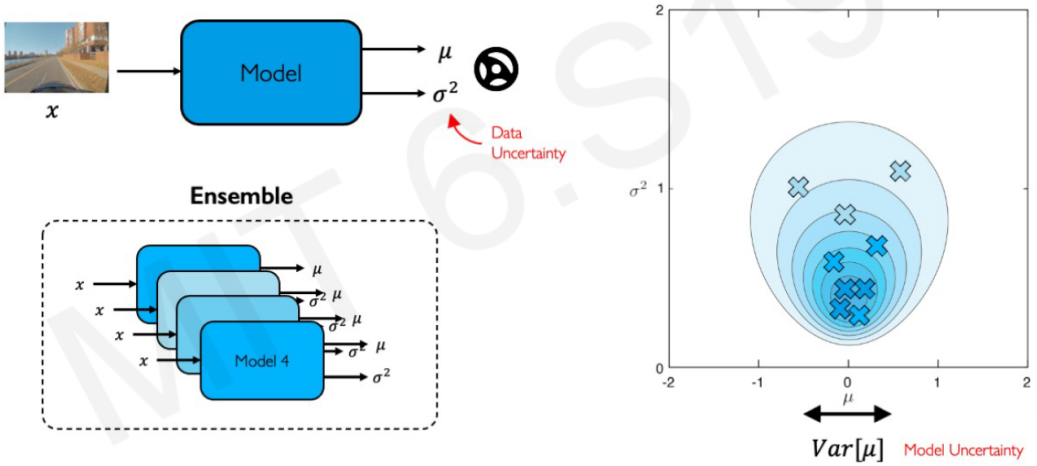


- There is a need for a single-pass model that is able to provide both uncertainties
- Image source

Evidental Deep Learning

Beyond sampling for approximating uncertainty

Sampling an ensemble of models to approximate the uncertainty



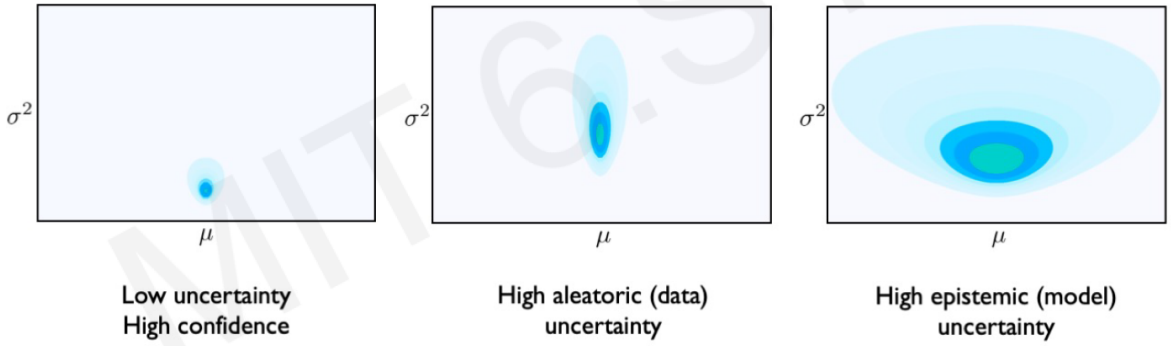
- Each model from the ensemble leads to a single point (μ, σ^2)
- The model uncertainty is then given by the variance of μ - $Var\,[\mu]$
- Image source

Evidental Deep Learning

Evidential deep learning

Treat learning as an **evidence acquisition** process

More evidence → increased predictive confidence



- Ultimately we are interested in the marginal distributions over μ and σ
- Image source

Evidental Deep Learning

Evidential learning for regression

Sampling from an evidential distribution yields individual new distributions over the data

$$y \sim \text{Normal}(\mu, \sigma^2)$$

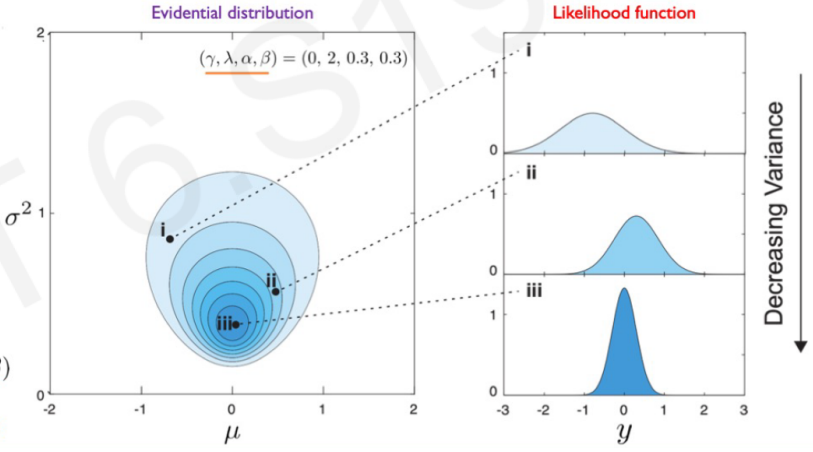
Target Labels | Likelihood function | Distribution parameters

Assume the distribution parameters are not known, place priors over each and probabilistically estimate!

$$\mu \sim \text{Normal}(\gamma, \sigma^2 v^{-1})$$
$$\sigma^2 \sim \Gamma^{-1}(\alpha, \beta)$$

$$\mu, \sigma^2 \sim \text{NormalInvGamma}(\gamma, v, \alpha, \beta)$$

Distribution parameters | Evidential Prior | Model parameters



- Key idea: Instead of training ensemble of models to output scalars μ and σ, output their distributions directly
- Image source

Evidental Deep Learning

Evidential learning for classification

Sampling from an evidential distribution yields individual new distributions over the data

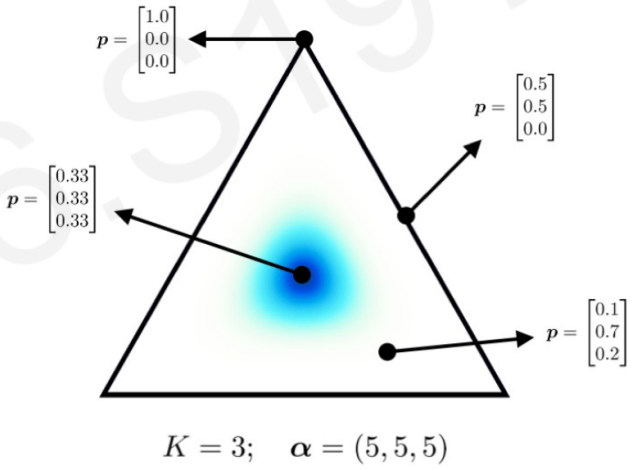
$$y \in \{1, \cdots, K\}$$

$$y \sim \text{Categorical}(\boldsymbol{p})$$

Class Labels | Likelihood function | Distribution parameters (probabilities)

$$\boldsymbol{p} \sim \text{Dirichlet}(\boldsymbol{\alpha})$$

Distribution parameters | Evidential Prior | Model parameters



$$K = 3; \quad \boldsymbol{\alpha} = (5, 5, 5)$$

- For Classification: The prior will be on the individual probabilities
- Image source

Evidental Deep Learning

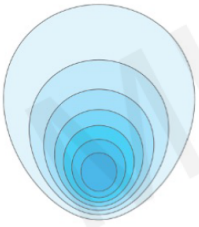
Evidential distributions for regression and classification

Regression (continuous)

$$y \in \mathbb{R}$$

$$y \sim \text{Normal}(\mu, \sigma^2)$$

$$\mu, \sigma^2 \sim \text{NormalInvGamma}(\gamma, \upsilon, \alpha, \beta)$$

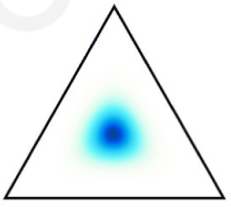


Classification (discrete)

$$y \in \{1, \cdots, K\}$$

$$y \sim \text{Categorical}(\boldsymbol{p})$$

$$\boldsymbol{p} \sim \text{Dirichlet}(\boldsymbol{\alpha})$$



Side note:

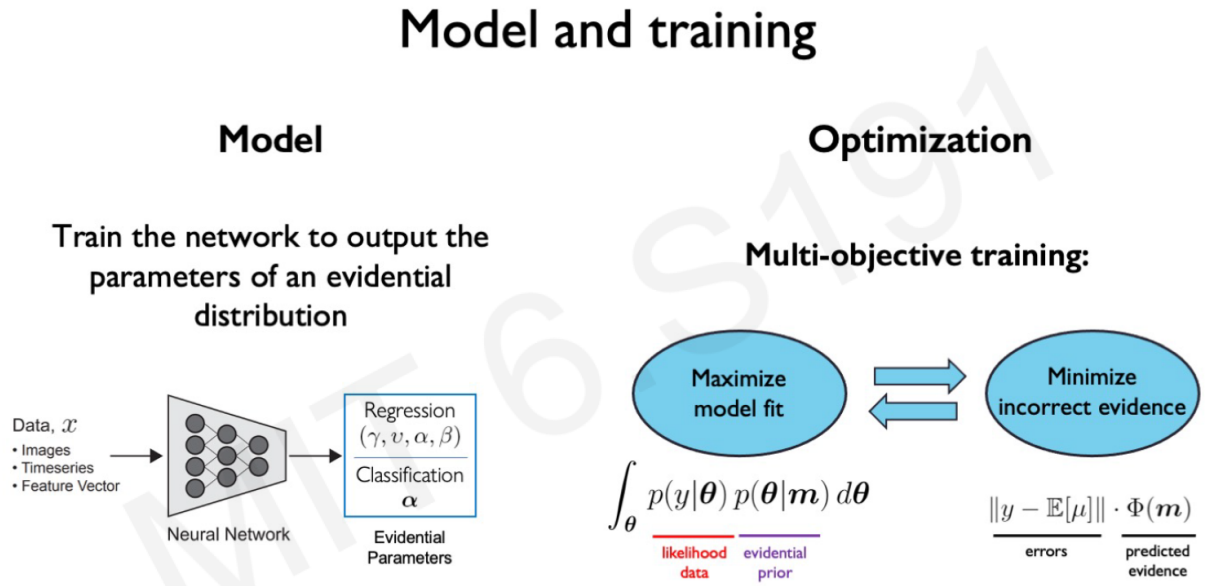
Choice of evidential distribution is closely related to conjugate priors in Bayesian inference.

It is often easiest to pick your evidential distribution to be a conjugate prior of your likelihood

$$p(\theta|y) = \frac{p(y|\theta)\, p(\theta)}{\int_{\theta'} p(y|\theta')\, p(\theta')\, d\theta'}$$

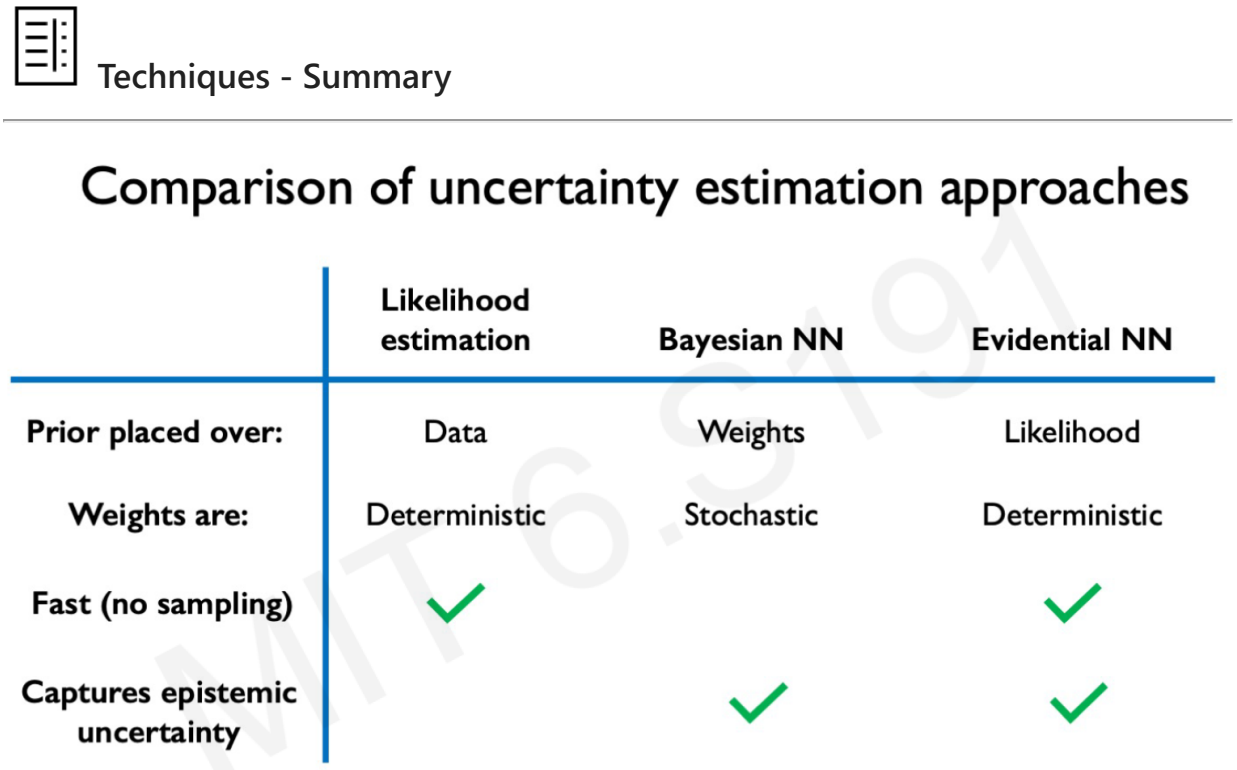
- Key idea: Instead of training ensemble of models to output scalars, output nested distributions directly
- The previous choices are not random, stem from "conjugate priors" in Bayesian analysis:
  - If posterior and prior are in the same "probability distribution family" (eg exponential) then the prior is said to be conjugate to the likelihood.
  - Examples include: Poisson Likelihood-Gamma Prior, Categorial Likelihood-Dirichlet Prior,...
- Image source

Evidental Deep Learning



- Pytorch implementation: https://github.com/dougbrion/pytorch-classification-uncertainty
- Image source

Techniques - Summary

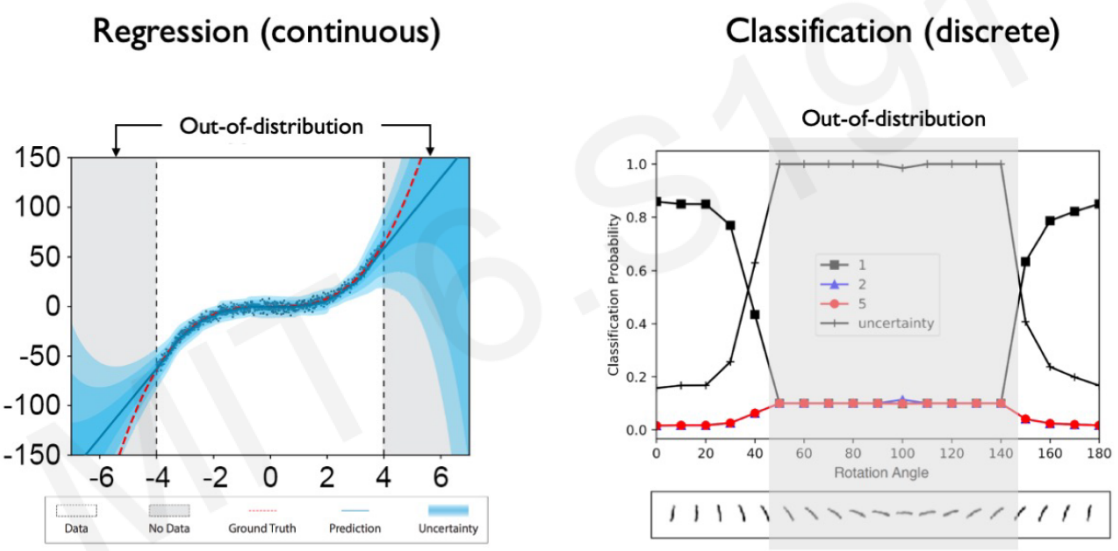


- Image source

Computer Vision Applications

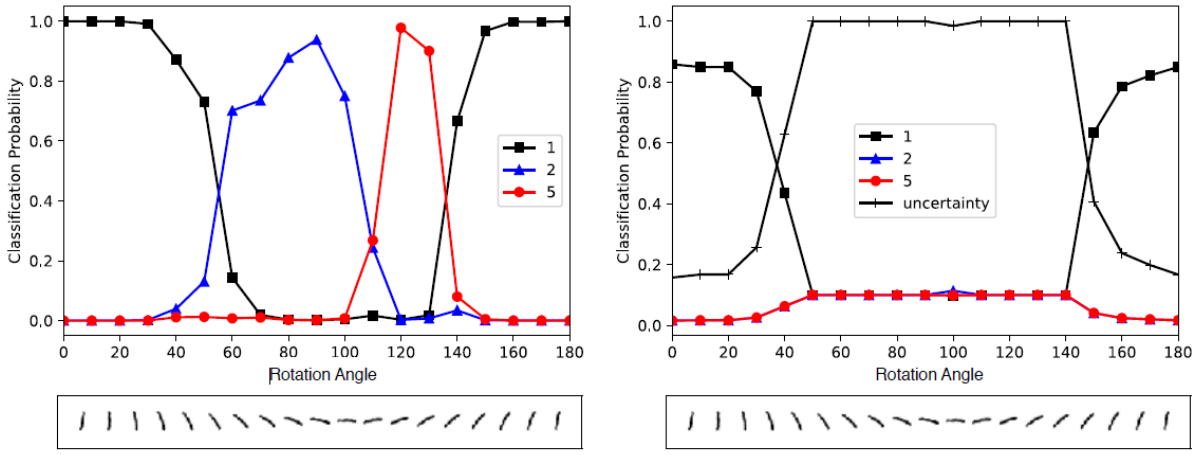
Application 1: Out-of-distribution detection

Toy learning problems



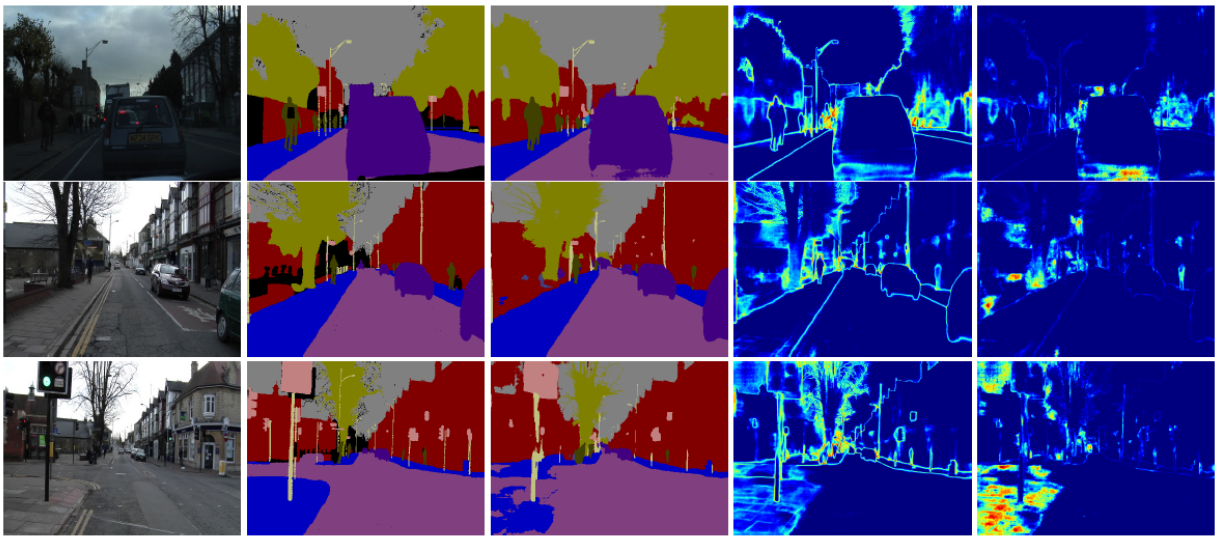
- Image source

Application 1: Out-of-distribution detection



- Standard Softmax on the left, Evidental deep learning on the right
- Rotation angle effectively induces out-of-distribution samples
- Erroneously high confidence with standard method
- Image source

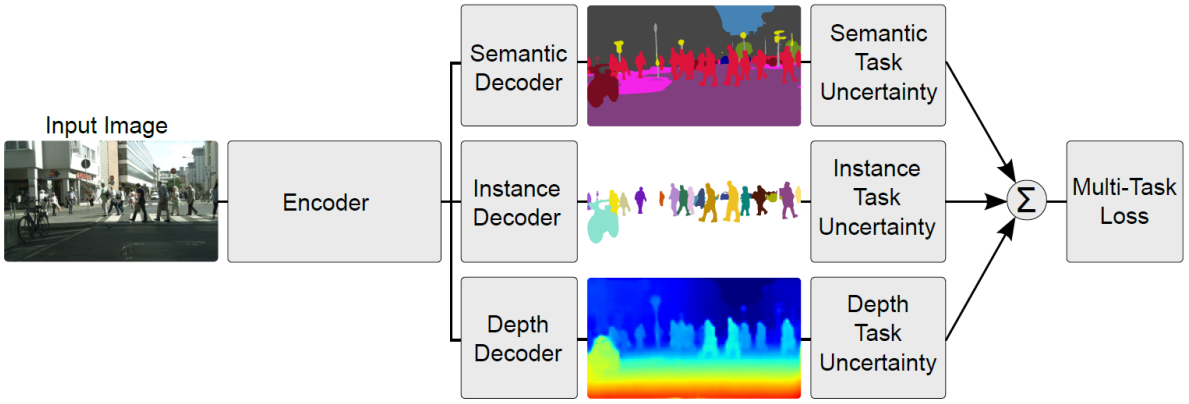
Application 2: Autonomous Driving



(a) Input Image (b) Ground Truth (c) Semantic Segmentation (d) Aleatoric Uncertainty (e) Epistemic Uncertainty

- Aleatoric uncertainty captures object boundaries where labels are noisy
- Bottom row - failure case shows increase in epistemic uncertainty
- Image source

Application 2: Autonomous Driving



- Multi-task Learning: Improves over learning each task individually
- Improve generalization by sharing info between tasks: "what is learned from one task can help learn another task"
- Optimal weighting between the importance of the different tasks given by the homoscedastic aleatoric uncertainty

$$\mathcal{L}_{total} = \sum_i \alpha_i \mathcal{L}_i \approx \frac{1}{2\sigma_1^2}\mathcal{L}_1 + \frac{1}{2\sigma_2^2}\mathcal{L}_2 + \log(\sigma_1) + \log(\sigma_2)$$

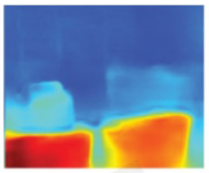
- Image source

Application 2: Autonomous Driving

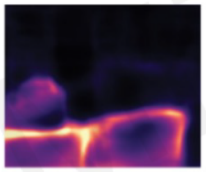
Applications of evidential learning

Monocular Depth Estimation

RGB input

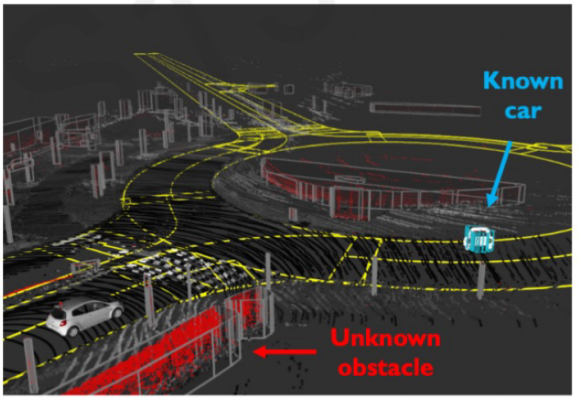


Predicted depth



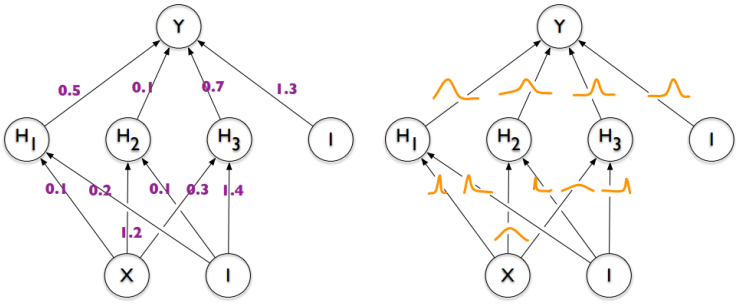
Predicted uncertainty

LiDAR Object Classification



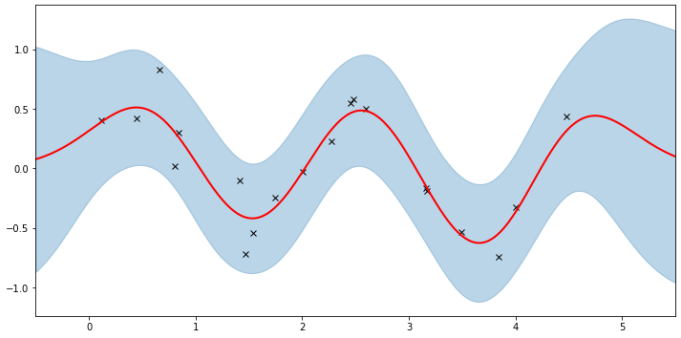
- Monocular depth estimation
- 3D Object recognition in point clouds
- Pose Estimation in SLAM
- etc.
- Image source

Blitz - Bayesian Neural Networks in PyTorch



- Blitz is a bayesian neural netowrk library for PyTorch
- Acompanies the ~first paper that revived this field from ICML 2015.
- Relatively easy to use, however, quite heavy for large models.

Pyro - Deep Universal Probabilistic Programming



- Pyro is a deep probabilistic programming library for PyTorch
- It has extensive tools, and can deal with a variety of applications including generative models etc.

- Other availabel resources online:
  - https://github.com/Harry24k/bayesian-neural-network-pytorch
  - https://github.com/dougbrion/pytorch-classification-uncertainty
  - https://github.com/cpark321/uncertainty-deep-learning
  - https://github.com/ivannz/mlss2019-bayesian-deep-learning
  - etc.

Simple code example: BNN with dropout

- Code based on example by Chanwoo Park

In [1]:
```
# imports
import torch
import torch.nn as nn
import torch.nn.functional as F
import pandas as pd
import numpy as np
import math
import torch.optim as optim
from torch.autograd import Variable
import matplotlib.pyplot as plt
from torchvision import datasets, transforms
%matplotlib inline

# GPU ID
device = torch.device("cuda:0")
```

In [2]:
```
# simple network model for 1D regression
class SimpleModel(torch.nn.Module):
    def __init__(self, dropout_rate, decay):
        super(SimpleModel, self).__init__()
        self.dropout_rate = dropout_rate
        self.decay = decay
        self.f = torch.nn.Sequential(
            torch.nn.Linear(1,20),
            torch.nn.ReLU(),
            torch.nn.Dropout(p=self.dropout_rate),
            torch.nn.Linear(20, 20),
            torch.nn.ReLU(),
            torch.nn.Dropout(p=self.dropout_rate),
            torch.nn.Linear(20,1)
        )
    def forward(self, X):
        return self.f(X)
```

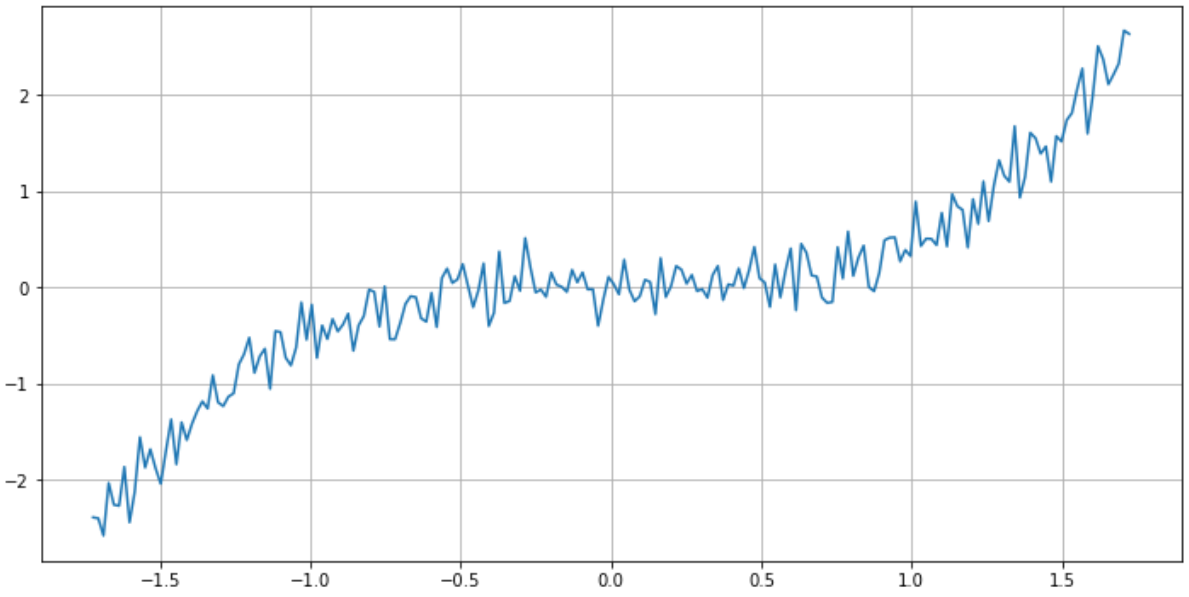
In [3]:
```
# estimate uncertainty by MC dropout sampling
def uncertainity_estimate(x, model, num_samples, l2):
    outputs = np.hstack([model(x).cpu().detach().numpy() for i in range(num_samples)]) # n번 inference
    y_mean = outputs.mean(axis=1)
    y_variance = outputs.var(axis=1)
    tau = l2 * (1. - model.dropout_rate) / (2. * N * model.decay)
    y_variance += (1. / tau)
    y_std = np.sqrt(y_variance)
    return y_mean, y_std
```

In [4]:
```
# generate data
N = 200 ##  number of points
min_value = -10
max_value = 10


x_obs = np.linspace(min_value, max_value, N)
```

```
noise = np.random.normal(loc = 10, scale = 80, size = N)
y_obs =  x_obs**3 + noise

x_test = np.linspace(min_value - 10, max_value + 10, N)
y_test = x_test**3 + noise

# Normalise data:
x_mean, x_std = x_obs.mean(), x_obs.std()
y_mean, y_std = y_obs.mean(), y_obs.std()
x_obs = (x_obs - x_mean) / x_std
y_obs = (y_obs - y_mean) / y_std
x_test = (x_test - x_mean) / x_std
y_test = (y_test - y_mean) / y_std
```

In [5]:

```
plt.figure(figsize=(12,6))
plt.plot(x_obs, y_obs)
plt.grid()
```



In [6]:

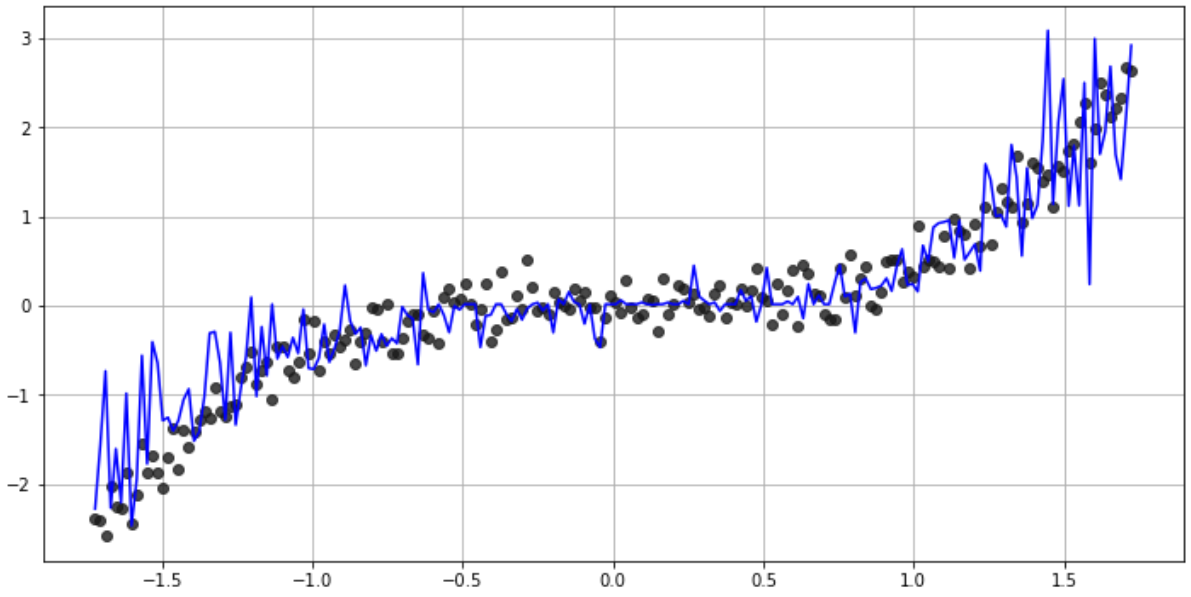
```
# define setting and optimizer
model = SimpleModel(dropout_rate=0.5, decay=1e-6).to(device)
criterion  = torch.nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.01, weight_decay=model.decay)

# train the model
for iter in range(20000):
    y_pred = model(torch.Tensor(x_obs).view(-1,1).to(device))
    y_true = Variable(torch.Tensor(y_obs).view(-1,1).to(device))
    optimizer.zero_grad()
    loss = criterion(y_pred, y_true)
    loss.backward()
    optimizer.step()

    if iter % 2000 == 0:
        print("Iter: {}, Loss: {:.4f}".format(iter, loss.item()))
```

```
Iter: 0, Loss: 1.2344
Iter: 2000, Loss: 0.2813
Iter: 4000, Loss: 0.2759
Iter: 6000, Loss: 0.2099
Iter: 8000, Loss: 0.2307
Iter: 10000, Loss: 0.2246
Iter: 12000, Loss: 0.2753
Iter: 14000, Loss: 0.2847
Iter: 16000, Loss: 0.2229
Iter: 18000, Loss: 0.2886
```

In [7]:

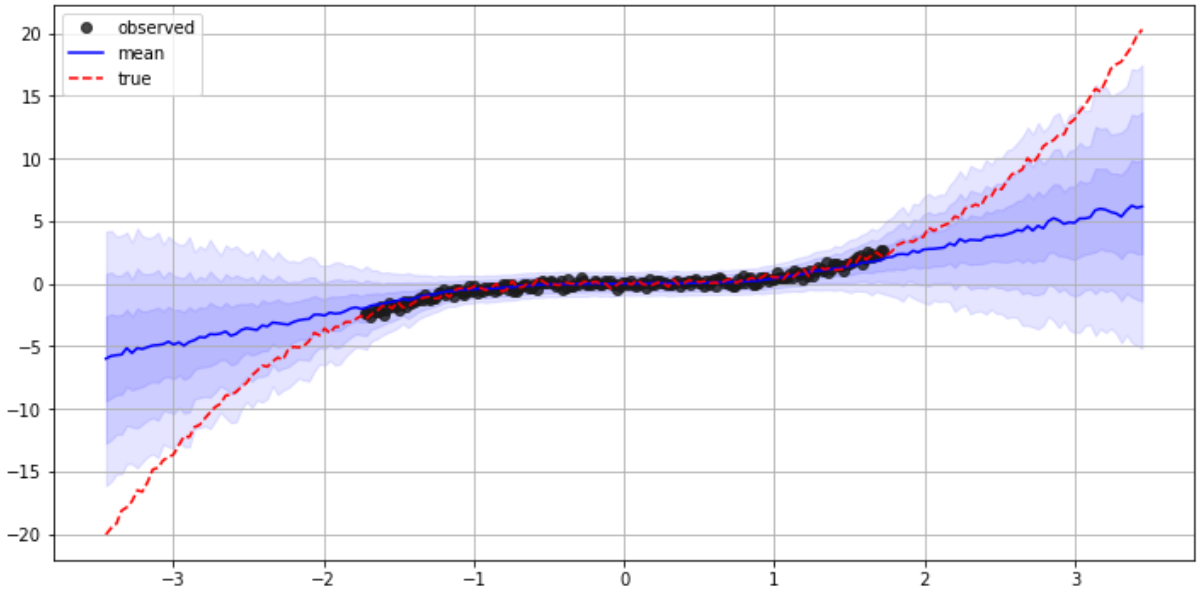
```
# plot the fit to the training data
plt.figure(figsize=(12,6))
y_pred = model(torch.Tensor(x_obs).view(-1,1).to(device))
plt.plot(x_obs, y_obs, ls="none", marker="o", color="0.1", alpha=0.8, label="observed")
```

```
plt.plot(x_obs, y_pred.cpu().detach().numpy(), ls="-", color="b", label="mean")
plt.grid()
```



In [9]:

```
# estimate uncertainty
iters_uncertainty = 200
lengthscale = 0.01
n_std = 3 # number of standard deviations to plot
y_mean, y_std = uncertainity_estimate(torch.Tensor(x_test).view(-1,1).to(device), model, iters_uncerta

# plot the confidence intervals
plt.figure(figsize=(12,6))
plt.plot(x_obs, y_obs, ls="none", marker="o", color="0.1", alpha=0.8, label="observed")
plt.plot(x_test, y_mean, ls="-", color="b", label="mean")
plt.plot(x_test, y_test, ls='--', color='r', label='true')
for i in range(n_std):
    plt.fill_between( x_test,
        y_mean - y_std * ((i+1.)),
        y_mean + y_std * ((i+1.)),
        color="b",
        alpha=0.1)
plt.legend()
plt.grid()
```



Recommended Videos

Warning!

- These videos do not replace the lectures and tutorials.
- Please use these to get a better understanding of the material, and not as an alternative to the written material.

Video By Subject

- Bayesian Neural Networks (Theory)
  - Variational Inference and ELBO - Yarin Gal - Part I
  - Dropout as efficient Bayesian Inference - Yarin Gal - Part II
- Bayesian Neural Networks vs Dropout (Practice)
  - MNIST demonstration - Elise Jennings, Argonne National Laboratory
- Evidental Deep Learning - MIT 6.S191 - Lecture 7

Credits

- What My Deep Model Doesn't Know... - Yarin Gal
- Deep Learning Is Not Good Enough, We Need Bayesian Deep Learning for Safe AI - Alex Kendall
- Bayesian Deep Learning 101 @ MLSS19 - Yarin Gal
- Slides Lecture 7 MIT 6.S191 (Introduction to Deep Learning) - Alexander Amini
- Uncertainty in Deep Learning. How to Measure? - Michel Kana
- Uncertainty Estimation in Deep Learning (Slideshare) - Christian Perone
- Research papers:
  - Weight Uncertainty in Neural Networks
  - Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning
  - What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?
  - Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics
  - Evidential Deep Learning to Quantify Classification Uncertainty
  - A General and Adaptive Robust Loss Function
  - Deep Evidential Regression
- Icons from Icon8.com - https://icons8.com