Mixture Density Networks

One way to model the probability distribution and the uncertainty estimation

Axel Brando Guillaumes
Advisor: Jordi Vitrià Marca
Co-advisor: Santi Seguí Mesquida

January 31st, 2017

Universitat de Barcelona

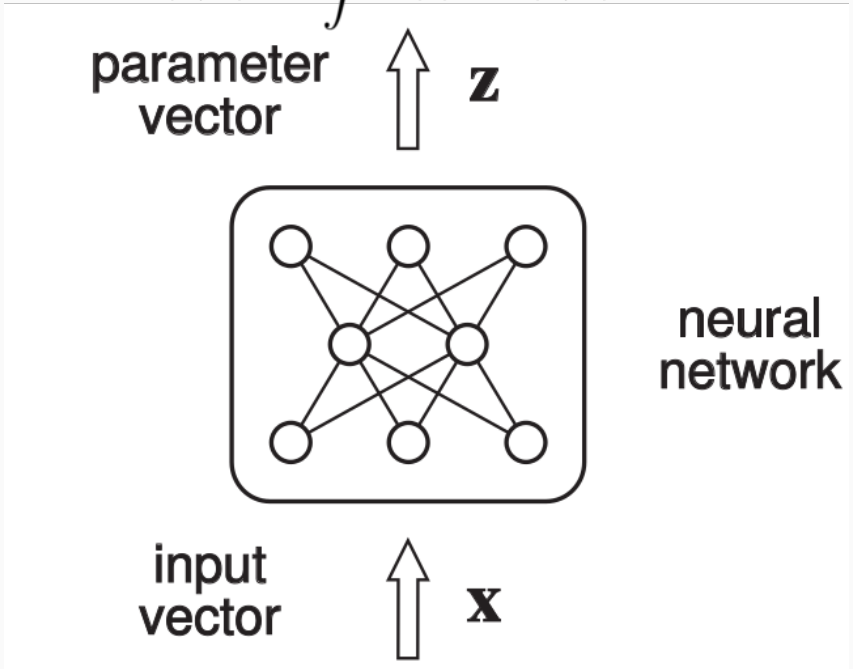
Table of contents

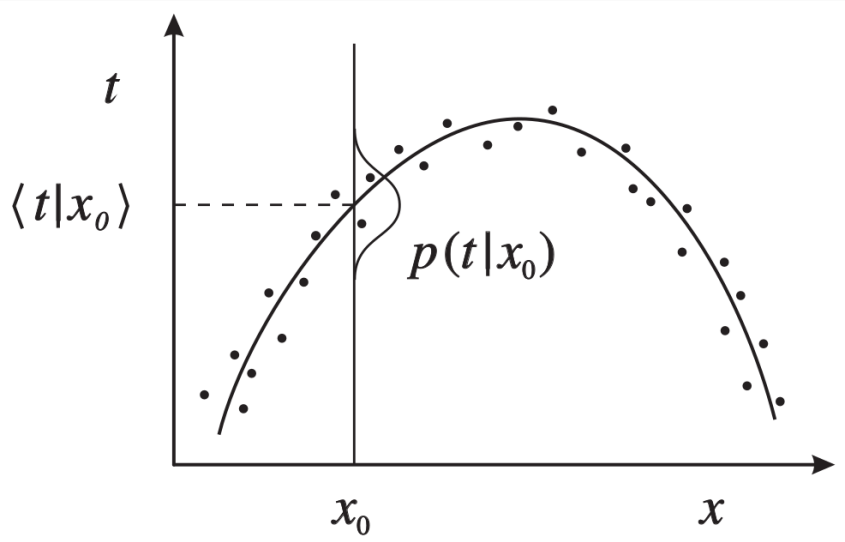
MOTIVATION

Standard Neural Network limitation (a priori)

According to Bishop in his article Mixture Density Networks, a typical Neural Network (NN) approximates the conditional average of the target data, conditioned on the input vector.

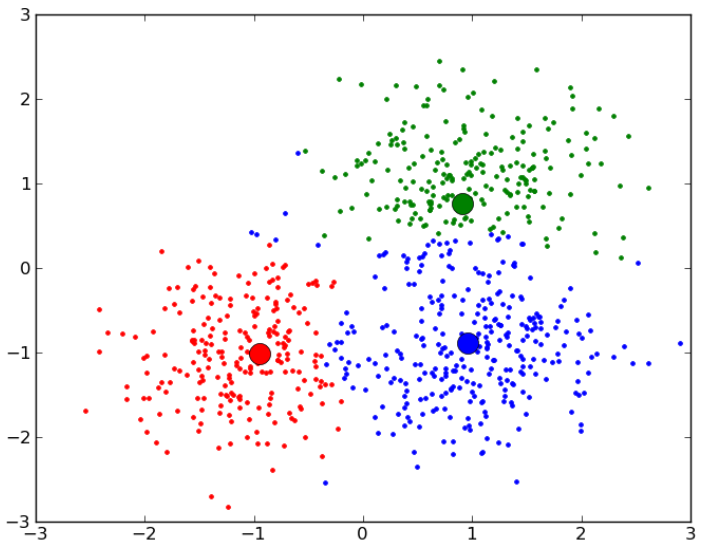
$$\langle t|x\rangle \equiv \int p(t) \cdot p(t|x)dt$$





Standard Neural Network limitation (a priori)

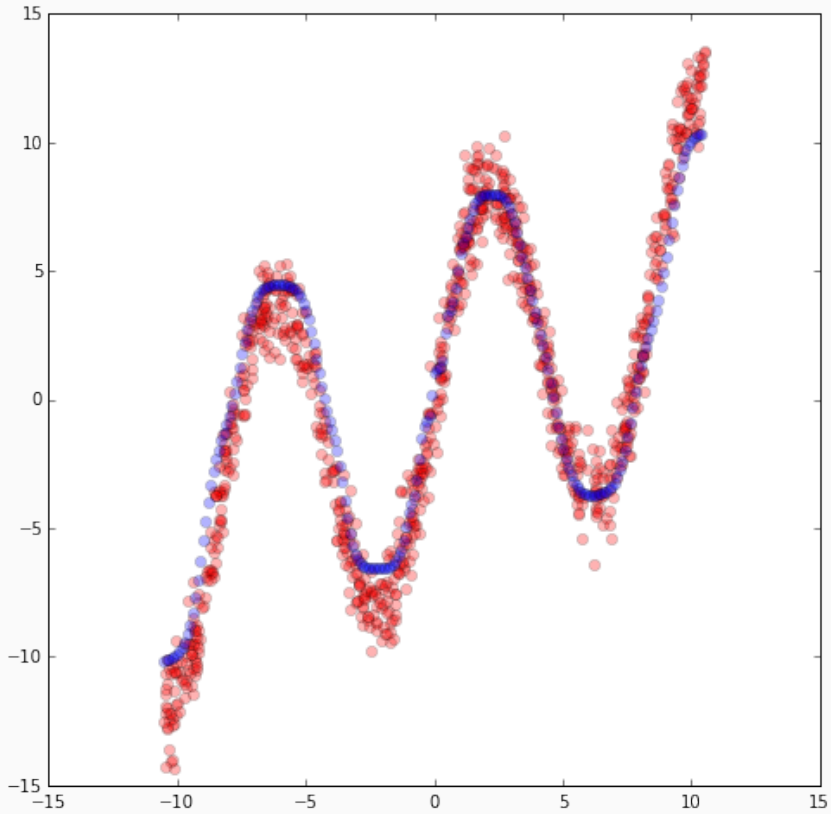
- For **classification problems**: these averages represent the posterior probabilities of class membership (it can be regarded as optimal with a suitable chosen target coding scheme).



For more information, see the appendix part of the slides.

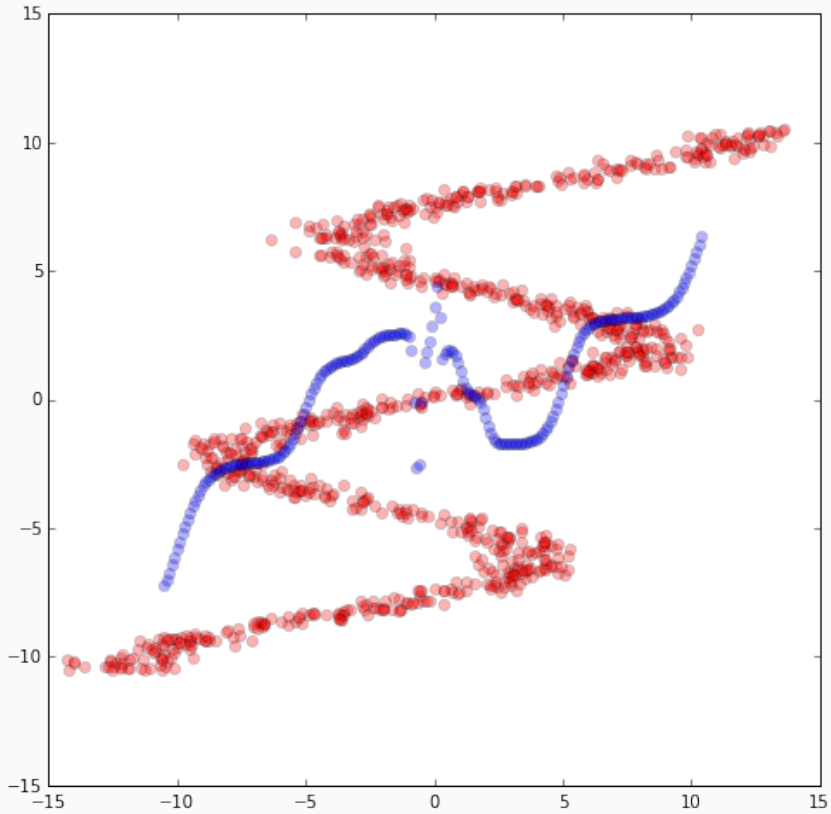
Standard Neural Network limitation (a priori)

- For **regression** problems: The conditional averages provide a very limited description of the properties of the target variable.



Standard Neural Network limitation (a priori)

- For **approximating distributions** problems: The average of several correct target values is not necessary itself a correct value.



A possible solution...

To solve these problems, we must model the conditional probability distribution of the target data conditioned on the input vector.

$$p(\boldsymbol{y}|\boldsymbol{x}) = \prod_{k=1}^{c} p(y_k|\boldsymbol{x})$$

Once we have modelled the distribution of probabilities, could we model the **uncertainty** of a certain prediction?

MIXTURE DENSITY NETWORKS

Mixture Density Networks

The probability density of the target data is then represented as a linear combination of kernel functions in the form

$$p(\boldsymbol{y}|\boldsymbol{x}) = \sum_{i=1}^{m} \alpha_i(\boldsymbol{x})\phi_i(\boldsymbol{y}|\boldsymbol{x})$$

where m is the number of components in the mixture and $\alpha_i(\boldsymbol{x})$ are called mixing coefficients.

And we selected the kernel functions which are Gaussian of the form:

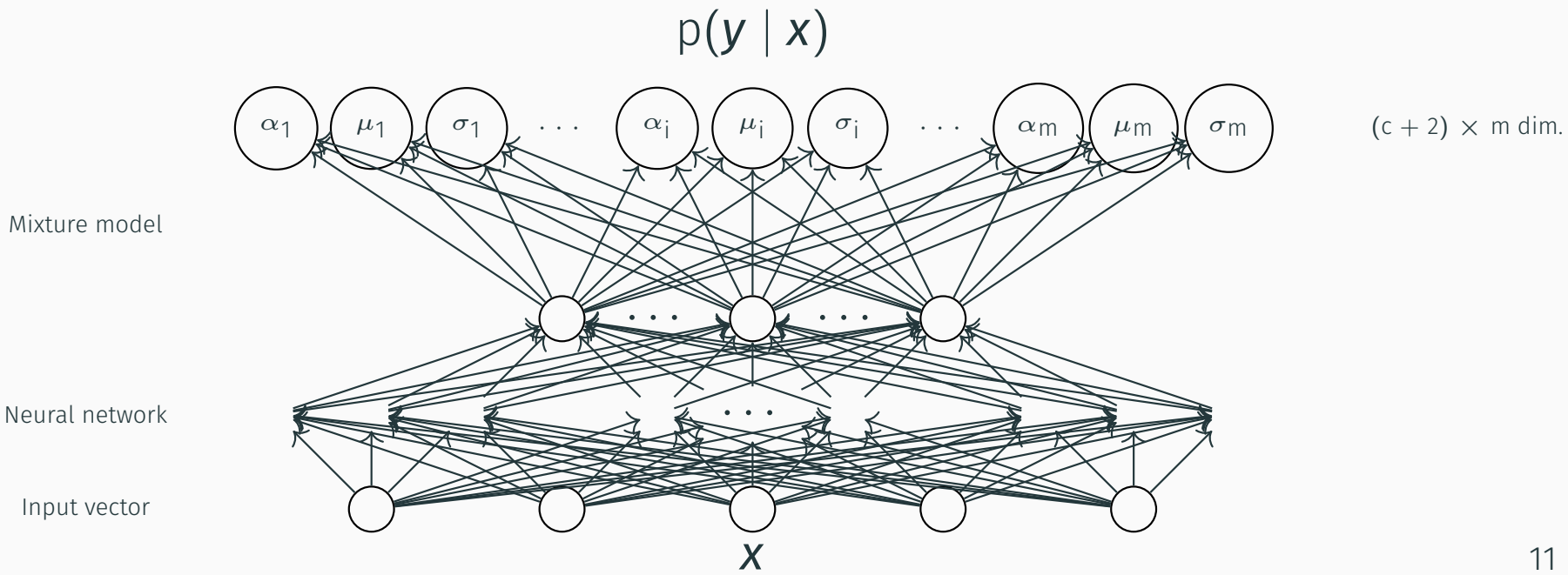
$$\phi_i(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{(2\pi)^{c/2}\sigma_i(\boldsymbol{x})^c} \exp\left\{-\frac{\| \boldsymbol{y} - \boldsymbol{\mu}_i(\boldsymbol{x}) \|^2}{2\sigma_i(\boldsymbol{x})^2}\right\}$$

where $\boldsymbol{\mu}_i$ represents the center of the i^{th} kernel. We assumed that the components of the output vector were statically independent within each component of the distribution, and it can be described by a common variance $\sigma_i(\boldsymbol{x})$.

The network outputs: Forward step

The total number of network outputs is given by $(c + 2) \times m$, where c is the outputs for a network used in the conventional manner and m the number of mixtures used.

$$\sum_{i=1}^{m} \alpha_i(\boldsymbol{x}) = 1, \qquad \alpha_i = \frac{\exp(z_i^{\alpha})}{\sum_{j=1}^{m} \exp(z_j^{\alpha})}, \qquad \sigma_i = \exp(z_i^{\sigma}), \qquad \mu_{i,k} = z_{i,k}^{\mu}$$



The Backward step: How to train this model

To estimate the output parameters, following to the **Maximum Likelihood Estimation** method (MLE), we can define a loss function which requires the minimisation of the negative log-likelihood function:

$$\mathcal{L}(y \mid x) = -\log\left(p(y \mid x)\right) = -\log\left(\sum_{i=0}^{m} \alpha_i(x)\phi_i(y|x)\right)$$

$$= -\log\left(\sum_{i=0}^{m} \frac{\alpha_i(x)}{(2\pi)^{c/2}\sigma_i(x)^c} \exp\left\{-\frac{\| y - \mu_i(x) \|^2}{2\sigma_i(x)^2}\right\}\right)$$

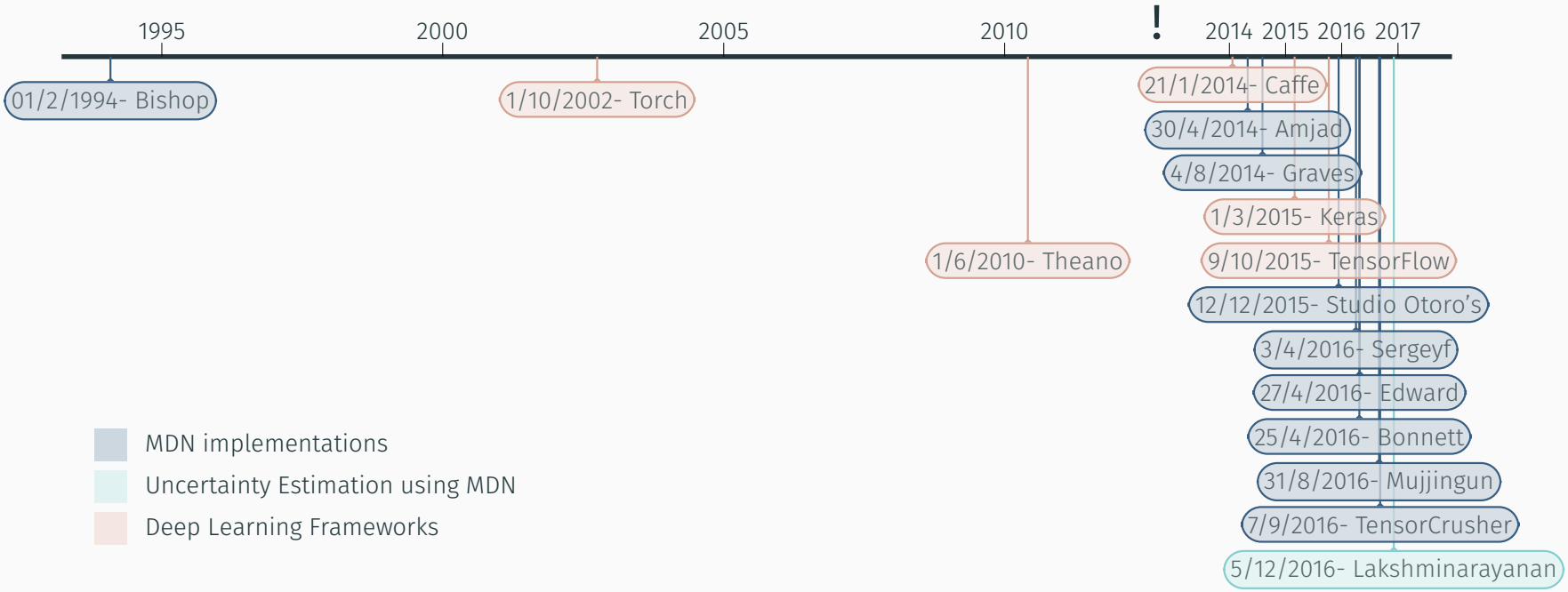
How to make predictions using MDNs?

- The simplest prediction is to predict the mean: $\langle \boldsymbol{y}|\boldsymbol{x}\rangle = \sum_i \alpha_i(\boldsymbol{x})\boldsymbol{\mu}_i(\boldsymbol{x})$. This is equivalent to the function computed by a standard network trained by least-squares.
- The mean $\boldsymbol{\mu}_i(\boldsymbol{x})$ of the highest component $\max_i(\frac{\alpha_i(\boldsymbol{x})}{\sigma_i(\boldsymbol{x})^c})$
- Generate a sample considering the total "probability mass" associated with each mixture components.

GOALS OF THE PROJECT

GOALS OF THE PROJECT

If we look at the Mixture Density Network literature...



✓To develop a reference implementation (numerical stability problems).

✓To test the current methods to measure uncertainty through MDN for a real problem.

✓To propose a reliable method regarding the predicted uncertainty for a real problem.

WHY WE USED KERAS(Tensorflow)

The software library used

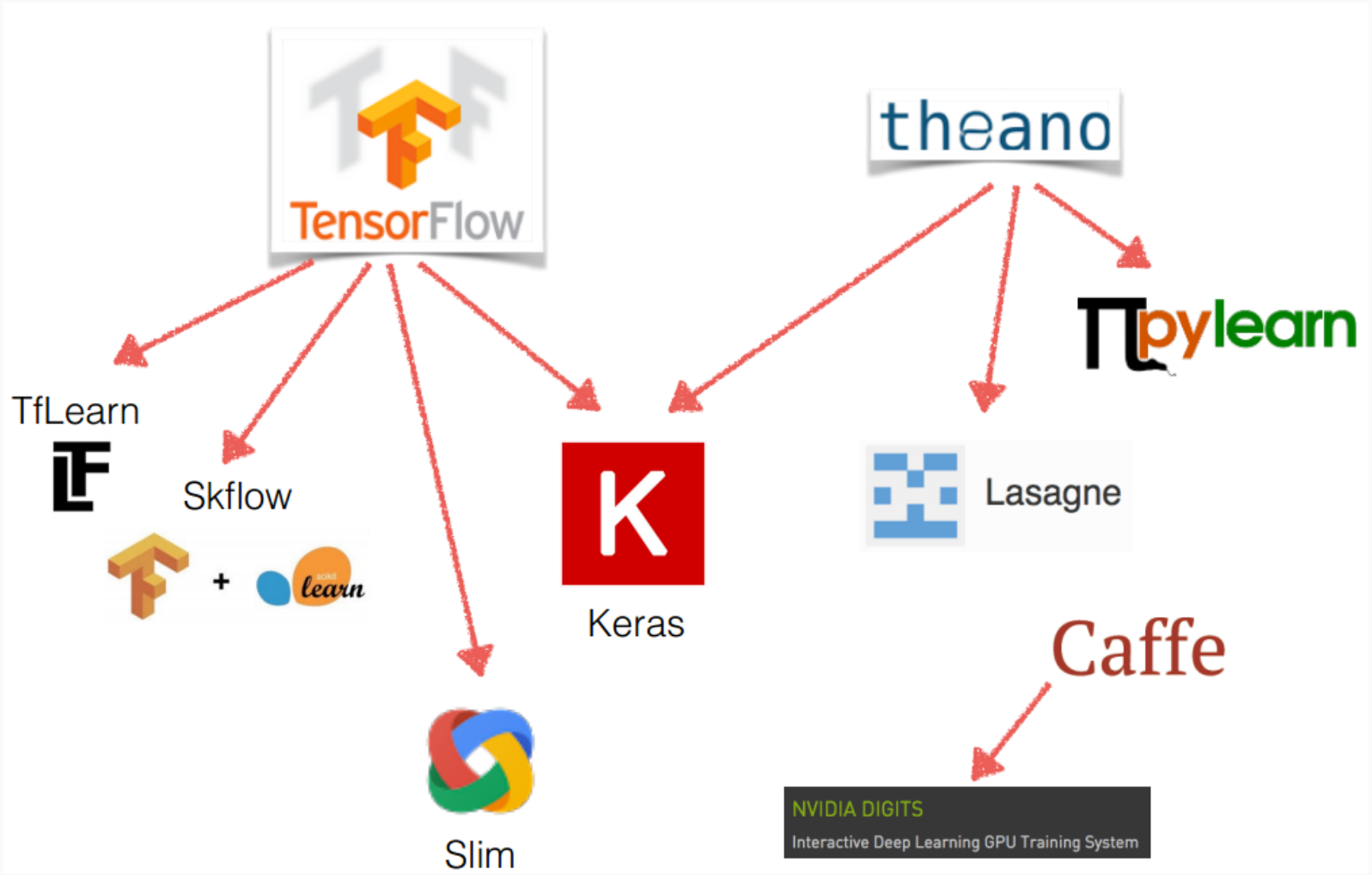


Figure 1: Slide extracted from the introduction presentation of the Deep Learning from Scratch course.

OUR IMPLEMENTATION OF THE MIXTURE DENSITY NETWORK

Compilation of techniques to avoid NaNs

Other authors (Bonnett, Mujjingun, Sergeyf, TensorCrusher, Amjad) found numerical problems when performing a MDN implementation. Therefore, we decided to make a compilation of some proposed techniques to avoid these problems:

The underflow problem:

$$\log \mathcal{L}(y \mid x) = -\log\left(p(y \mid x)\right) = -\log\left(\sum_{i=0}^{m} \frac{\alpha_i(x)}{(2\pi)^{c/2}\sigma_i(x)^c} \exp\left\{-\frac{\| y - \mu_i(x) \|^2}{2\sigma_i(x)^2}\right\}\right)$$

$$= -\log\left(\sum_{i=0}^{m} \exp\left\{\log\left(\alpha_i(x)\right) - \frac{c}{2}\log\left(2\pi\sigma_i(x)\right) - \frac{\| y - \mu_i(x) \|^2}{2\sigma_i(x)^2}\right\}\right)$$

Using the log-sum-exp trick: $\log \sum_{i=1}^{n} e^{x_i} = \max_i x_i + \log \sum_{i=1}^{n} e^{x_i - \max_i x_i}$

Compilation of techniques to avoid NaNs

It was not enough. A NaN can appear due to three possible scenarios:

1. $\log(\epsilon), \epsilon \approx 0$
2. $\frac{1}{\epsilon}, \epsilon \approx 0$
3. $e^x, x > 1e^4$

After testing many options, we considered that there are certain tricks that can help this last problem not to happen so often:

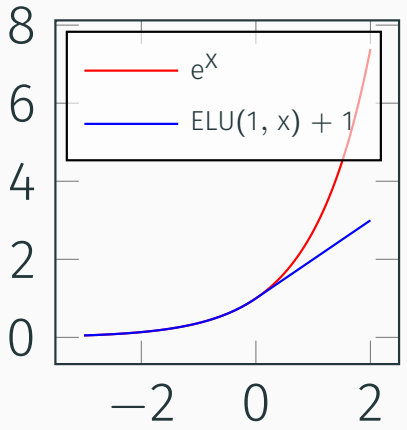
- Applying a Gradient Clipping technique when we are training the neural network.
- Clipping of the $\alpha(x)$ and $\sigma(x)$ parameters values.
- Clipping of the real y or estimated value z could solve the problem of an extreme point but clipping too many times can lead to a less generic solution.
- Applying weight regularisation technique to the respective weights of the $\sigma(x)$ part.
- Applying Batch normalisation technique to the output of the layers.

Our proposals

First, 2 proposals to solve the stability problem:

- To clip the $\alpha(\boldsymbol{x})$.
- Replacement of the $\sigma(\boldsymbol{x})$ exponential function to $\mathrm{ELU}(1, \mathrm{x}) + 1$:

$$\mathrm{ELU}(\alpha, \mathrm{x}) = \begin{cases} \alpha(\mathrm{e}^{\mathrm{x}} - 1) & \text{for } \mathrm{x} < 0 \\ \mathrm{x} & \text{for } \mathrm{x} \geq 0 \end{cases}$$

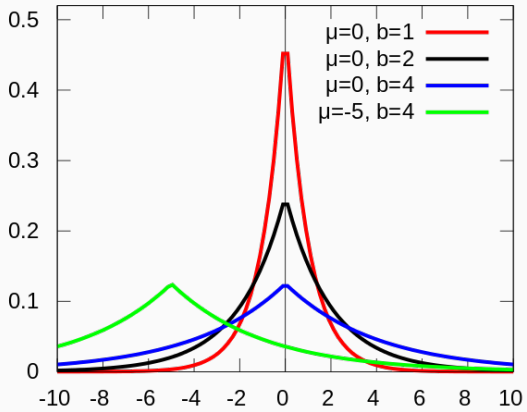


Our proposals

Second, to generalise reference model to Laplacian distribution:

- Mixture of Laplace distribution functions:

$$f(x \mid \mu, b) = \frac{1}{2b} \exp\left(-\frac{|x - \mu|}{b}\right)$$



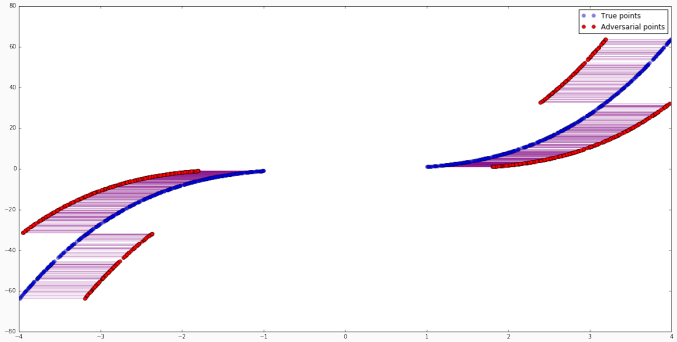
$$\log \mathcal{L}(\boldsymbol{y} \mid \boldsymbol{x}) = -\log\left(\sum_{i=0}^{m} \exp\left\{\log\left(\alpha_i(\boldsymbol{x})\right) - c\log\left(2b_i(\boldsymbol{x})\right) - \frac{|\,\boldsymbol{y} - \boldsymbol{\mu}_i(\boldsymbol{x})\,|}{b_i(\boldsymbol{x})}\right\}\right)$$

Our proposals: The addition of adversarial gradient

Third, to add to our reference model the idea of Lakshminarayanan et al. presented in NIPS 2016 (2 months ago) where they stated that an ensemble of MDNs trained with adversarial training allows to estimate predictive uncertainty.

$$\mathcal{L}_{\text{adv}}(y \mid x) = \lambda \mathcal{L}(y \mid x) + (1 - \lambda)\mathcal{L}(y \mid x + \omega)$$

Where $\mathcal{L}_{\text{adv}}$ - new adversarial loss, $\mathcal{L}$ - original loss, $\lambda \in [0, 1]$ and $\omega = \epsilon \operatorname{sign}(-\nabla_x \mathcal{L}(y \mid x))$ i.e. is the ϵ-weighted sign of the gradient of the loss function with respect to the input.

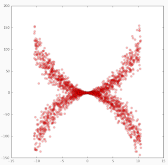
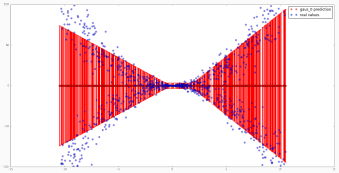
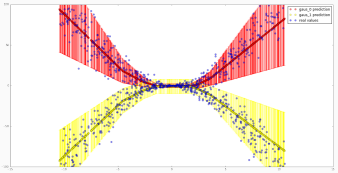
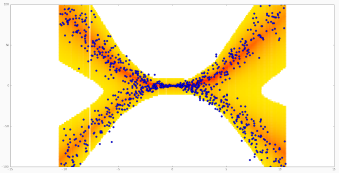


RESULTS

SIMPLE SYNTHETIC PROBLEMS TO TEST THE REFERENCE MODEL

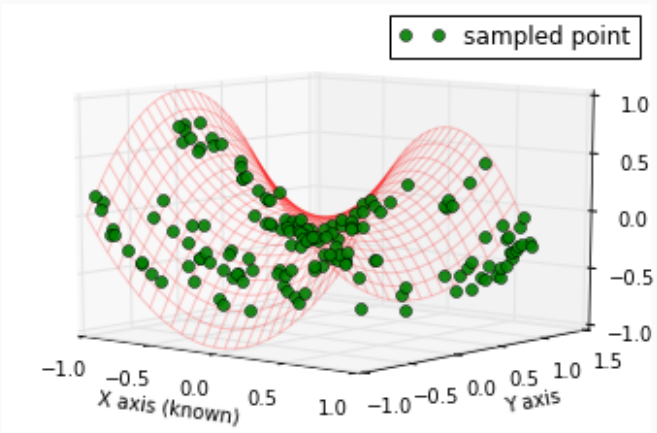
2 dimensional example

$y = \pm x^2 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, x)$

3 dimensional example

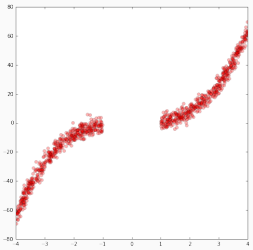
$z = x^2 - y^2$ with x and (y, z) as input and output information resp.

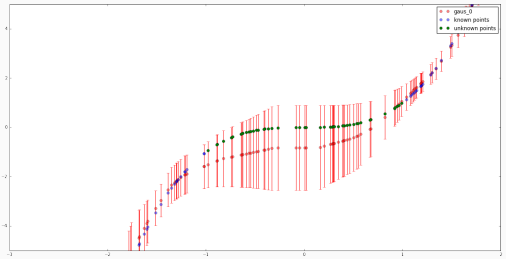


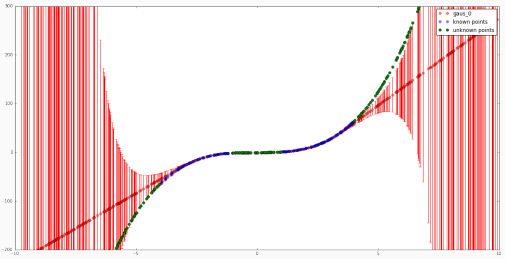
TEST OF AN ENSEMBLE OF MDNs WITH ADVERSARIAL TRAINING

A simple example to verify Deep Ensemble model proposed by Lakshminarayanan et al. in Simple and Scalable Predictive Uncertainty Estimation using Deep Ensemble.

$y = x^3 + \epsilon$ where $\epsilon \sim \mathcal{N}(0, 3)$ and $x \sim \mathcal{U}((-4, -1) \cup (1, 4))$







For more information, see the appendix part of the slides.

APPLICATION TO A REAL SPECIAL TIME SERIES PROBLEM

The results of Always Zero prediction by using different metrics were: MAE: 160.14, MSE: 9, 104, 905.15 and RMSE: 3, 017.43

Results of the LSTM proposed solution

1. **LSTM** - MAE: 109.93, MSE: 9, 053, 751.38 and RMSE: 3, 008, 94.
2. **LSTM+Gaus. MDN** - MAE: 174.06, MSE: 9, 102, 559.05 and RMSE: 3, 017.04
3. **LSTM+Laplac. MDN** - MAE: 114.28, MSE: 9, 056, 219.58 and RMSE: 3, 009.35

Results of the DNN proposed solution

DNN - MAE: 112.69, MSE: 8, 650, 128.79 and RMSE: 2, 941.11

The training time of the DNN case was totally manageable compared to the LSTMs ones.

APPLICATION TO A REAL SPECIAL TIME SERIES PROBLEM

Results of the DNN + Adversarial training proposed solution

DNN+Adv - MAE: 110.62, MSE: 8, 648, 857.65 and RMSE: 2, 940.89

DNN+Adv+Laplac. MDN - MAE: 113.28, MSE: 9, 055, 498.91 and RMSE: 3009.23

APPLICATION TO A REAL SPECIAL TIME SERIES PROBLEM

Application of the results and the adversarial data set definition concept

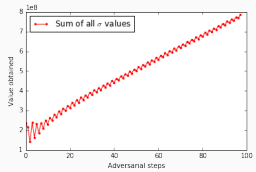
$$\mu_*(x) = \frac{1}{m}\sum_{i=1}^{m}\mu_i(x) \qquad \sigma_*^2(x) = \frac{1}{m}\sum_{i=1}^{m}\left(\sigma_m^2(x) + \mu_i^2(x)\right) - \mu_*^2(x)$$

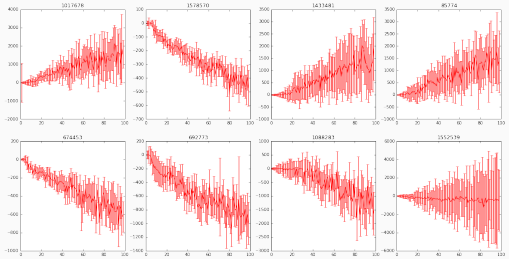
Ensemble of 5 **DNN+Adv. training + 1 Laplac. MDN**:

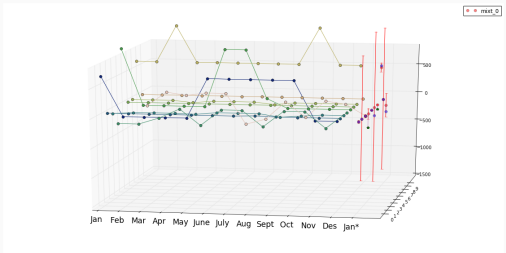
- MAE: 113.22, MSE: 9, 055, 350.57 and RMSE: 3, 009.21.

How to verify if σ_* gives us information concerning the confidence of the μ_*

To generate **adversarial data sets**, i.e. if points were more "unknown" then we obtain a greater σ. The idea was to create a series of data sets more and more adversarial in time.





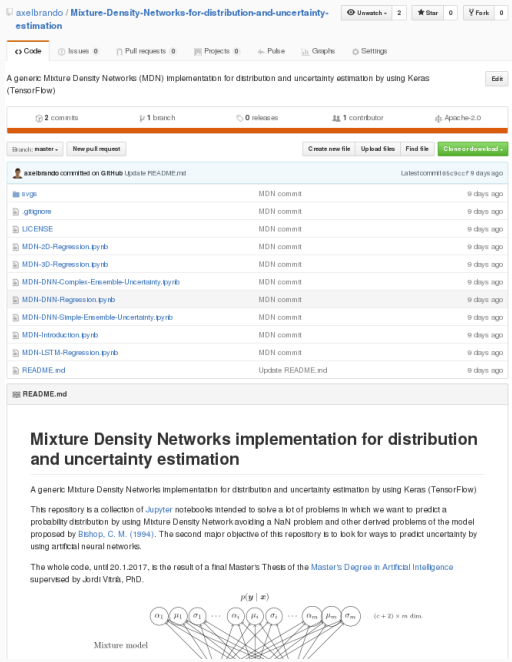


CONCLUSIONS

Conclusions

Three important aims satisfied:

✓A **Development of a reference implementation** (numerical stability problems) (log-sum-exp trick, clipping of parameters, weight regularisation, batch normalisation) + **proposal of new techniques** (clipping of α, replacement of σ function, replacement of distribution function) has been done:
https://github.com/axelbrando/Mixture-Density-Networks-for-distribution-and-uncertainty-estimation



✓An understanding and **testing of the current methods** to measure **uncertainty** through MDN has been done (Adversarial training and Deep Ensembles).

✓A proposal of a **reliable method regarding** the predicted **uncertainty** has been done (Generation of adversarial data sets).

Future work

Lines to follow:

⋆ The application of Adversarial training for recurrent architectures.

⋆ Findings regarding a more suitable number of mixtures to solve our problem.

⋆ The continuation of the idea regarding the creation of another set of data that we can later ensure that it is an ”unknown” set for the trained neural network and verify that the σ is high.

⋆ The calculation of the statistics for Ensemble prediction using other relationships between the different mixtures.

⋆ Findings concerning the point of connection between this method and other Bayesian methods that currently give good estimates of uncertainty.

Thank you for your attention.

Appendix: Cross-entropy error function

Minimisation of a sum-of-squares (cross-entropy) error function leads to network ouputs which approximate the conditional averages of the target data, conditioned on the input vector.

- For **classification problems**: these averages represent the posterior probabilities of class membership (can be regarded as optimal).
- For **continuous variable prediction problems**? The conditional averages provide a very limited description of the properties of the target variable.
- For **multi-valued mapping problems**? The average of several correct target values is not necessary itself a correct value.

To solve these problems, we must model the conditional probability distribution of the target data, again conditioned on the input vector.

Appendix: When standard neural network could be optimal for classification problems?

Extracted from Subramanya, A et al. paper called 'Semi-Supervised Learning with Measure Propagation'.

- Page 226: "In fact, the sum-of-squares error function is not the most appropriate for classification problems. It was derived from maximum likelihood on the assumption of Gaussian distributed target data. However, the target values for a ***l-of-c coding scheme are binary***, and hence far from having a Gaussian distribution."

While squared-error has worked well in the case of regression problems (Bishop, 1995),1 for classification, it is often argued that squared-loss is not the optimal criterion and alternative loss functions such as the cross-entropy (Bishop, 1995), logistic (Ng and Jordan, 2002), hinge-loss (Vladimir, 1998) have been proposed.

Appendix: Multivariate normal distribution

$X \sim \mathcal{N}(\mu, \Sigma)$,

$\mu = [\mu_1, \ldots, \mu_n]^T$ mean

Σ Covariance matrix (positive def. matrix of dimensions $k \times k$

PDF:

$$f_{\mathbf{x}}(x_1, \ldots, x_k) = \frac{1}{\sqrt{(2\pi)^k |\boldsymbol{\Sigma}|}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

$$f(x_1, \ldots, x_k \mid \mu, \Sigma) = \prod_{i=1}^{k} f(x_i \mid \mu, \sigma^2) = \left(\frac{1}{2\pi\sigma^2}\right)^{k/2} \exp\left(-\frac{\sum_{i=1}^{k}(x_i - \mu)^2}{2\sigma^2}\right)$$

If the mean and variance matrix are unknown, the standard approach to this problem is the **maximum likelihood method**, which requires maximisation of the log-likelihood function:

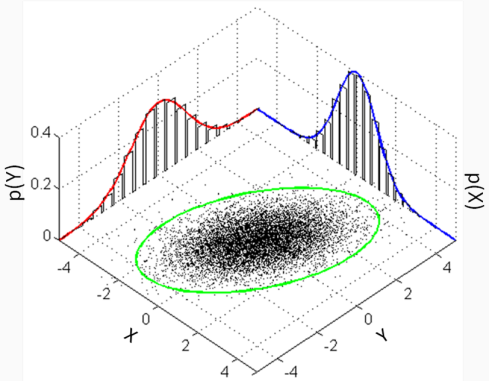
$$\boxed{\ln \mathcal{L}(\mu, \Sigma) = \sum_{i=1}^{k} \ln f(x_i \mid \mu, \Sigma) = -\tfrac{1}{2} \ln(|\boldsymbol{\Sigma}|) - \tfrac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1} (\mathbf{x} - \boldsymbol{\mu}) - \tfrac{k}{2} \ln(2\pi)}$$

where $\mathbf{x}$ is a vector of real numbers.

Appendix: Multivariate normal distribution

Taking derivatives with respect to μ and σ^2 and solving the resulting system of first order conditions yields the maximum likelihood estimates:

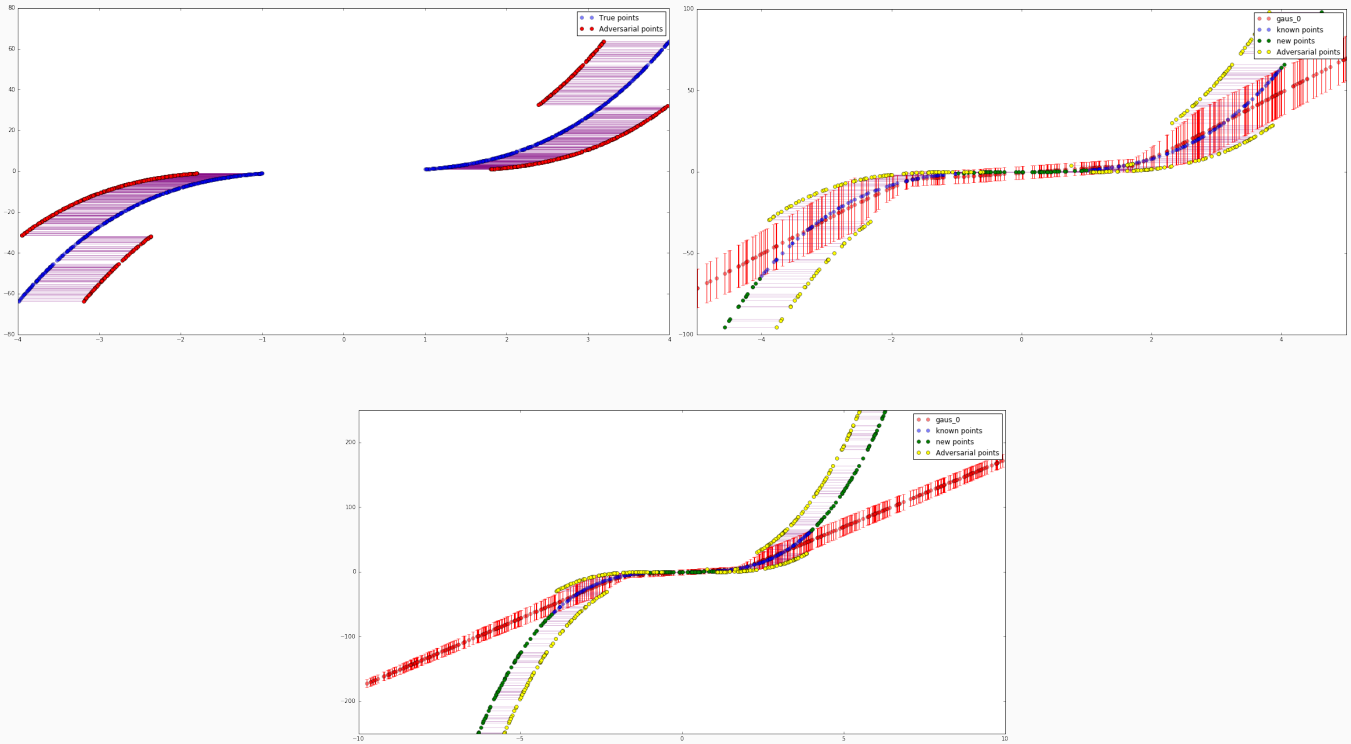
$$\hat{\mu} = \bar{x} \equiv \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2.$$



Appendix: What is an adversarial point?

$$\mathcal{L}_{\text{adv}}(y \mid x) = \lambda \mathcal{L}(y \mid x) + (1 - \lambda)\mathcal{L}(y \mid x + \omega)$$

$\omega = \epsilon \operatorname{sign}(-\nabla_x \mathcal{L}(y \mid x))$ i.e. is the ϵ-weighted sign of the gradient of the loss function with respect to the input.



Appendix: Quality of predictive Uncertainty

Two measures of the quality of predictive uncertainty from a neural network:

Calibration

It is the discrepancy between subjective forecasts and (empirical) long run frequencies. It is an orthogonal concept to accuracy: a network's predictions may be accurate and yet mis-calibrated.

Misspecifications

The generalisation of the predictive uncertainty to unknown unknowns.

Appendix: 3D Surface mean plot

3 dimensional example

$z = x^2 - y^2$ with x and (y, z) as input and output information resp.