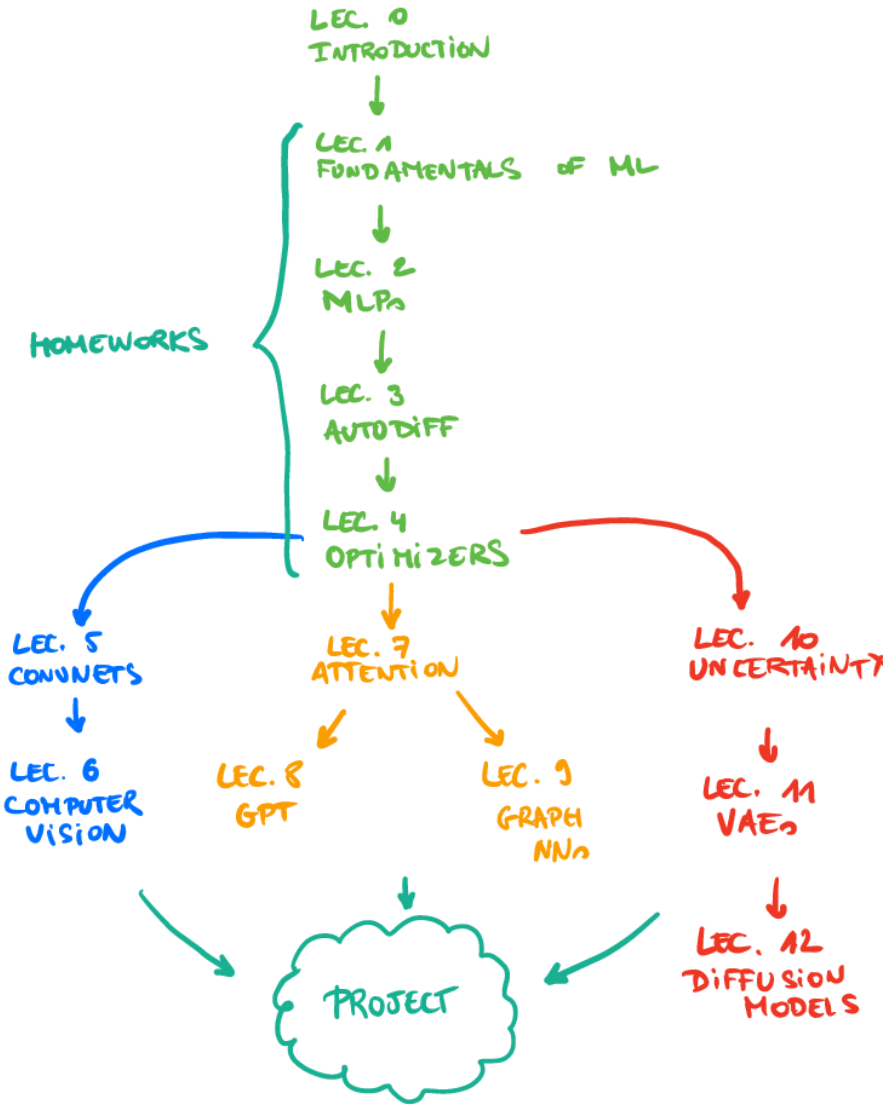
Deep Learning

Lecture 10: Uncertainty

Prof. Gilles Louppe
g.louppe@uliege.be

LIÈGE université


LEC. 0
INTRODUCTION
LEC. 1
FUNDAMENTALS OF ML
LEC. 2
MLPs
LEC. 3
AUTODIFF
LEC. 4
OPTIMIZERS
HOMEWORKS
LEC. 5
CONVNETS
LEC. 6
COMPUTER VISION
LEC. 7
ATTENTION
LEC. 8
GPT
LEC. 9
GRAPH NNs
LEC. 10
UNCERTAINTY
LEC. 11
VAEs
LEC. 12
DIFFUSION MODELS
PROJECT

Today

How to model uncertainty in deep learning?

- Uncertainty
- Aleatoric uncertainty
- Epistemic uncertainty

*"Every time a scientific paper presents a bit of data, it's accompanied by an **error bar** – a quiet but insistent reminder that no knowledge is complete or perfect. It's a **calibration of how much we trust what we think we know**."*

Carl Sagan

Uncertainty

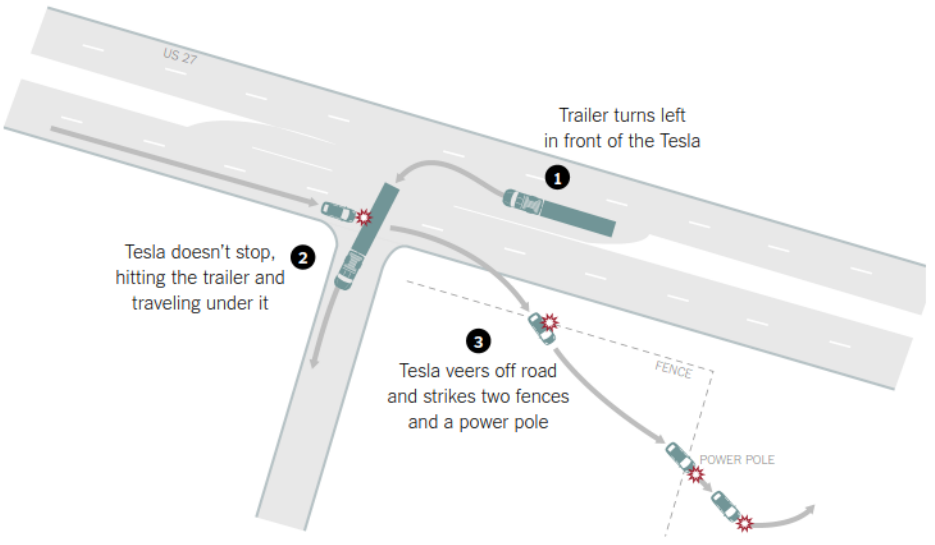
Uncertainty refers to epistemic situations involving imperfect or unknown information. It applies to predictions of future events, to physical measurements that are already made, or to the unknown.

Uncertainty arises in partially observable or stochastic environments, as well as due to ignorance, indolence, or both.meteorology, ecology and information science.

Why is uncertainty important?

—
Credits: Wikipedia, 2023.

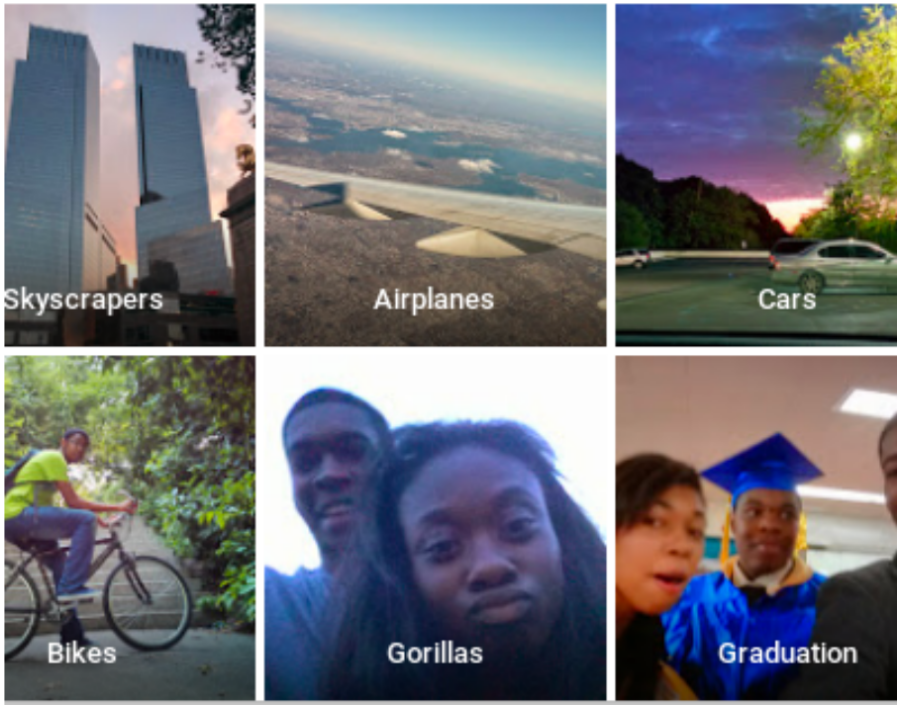
Case 1. First assisted driving fatality in May 2016: Perception system mistook trailer's white side for bright sky.



—

Credits: Kendall and Gal, What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?, 2017.


NOW
THIS
Newsflare via Reuters
This Tesla's 'smart summon' feature
caused it to crash into this $3.5M private jet



Case 2. An image classification system erroneously identifies two African Americans as gorillas, raising concerns of racial discrimination.

—

Credits: Kendall and Gal, What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?, 2017.

If these systems had assigned a high level of uncertainty to their erroneous predictions, then they might have been capable of making better decisions, and likely averting disaster.

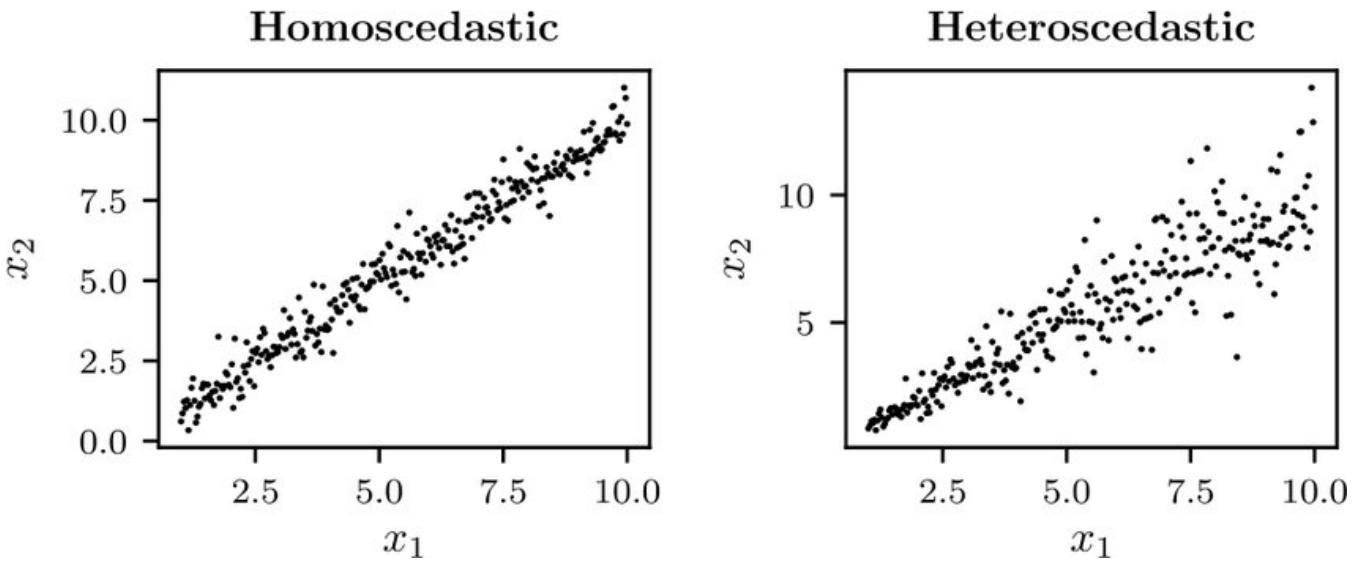
Aleatoric uncertainty

Aleatoric uncertainty captures noise inherent in the observations. For example, sensor noise or motion noise result in uncertainty.

This uncertainty cannot be reduced with more data. However, aleatoric uncertainty could be reduced with better measurements.

Aleatoric uncertainty can further be categorized into:

- Homoscedastic uncertainty, which relates to the uncertainty that a particular task might cause. It stays constant for different inputs.
- Heteroscedastic uncertainty, which depends on the inputs to the model, with some inputs potentially having more noisy outputs than others.



Neural density estimation

Consider training data $(\mathbf{x}, y) \sim P(X, Y)$, with

- $\mathbf{x} \in \mathbb{R}^p$,
- $y \in \mathbb{R}$.

We do not wish to learn a function $\hat{y} = f(\mathbf{x})$, which would only produce point estimates. Instead we want to learn the full conditional density

$$p(y|\mathbf{x}).$$

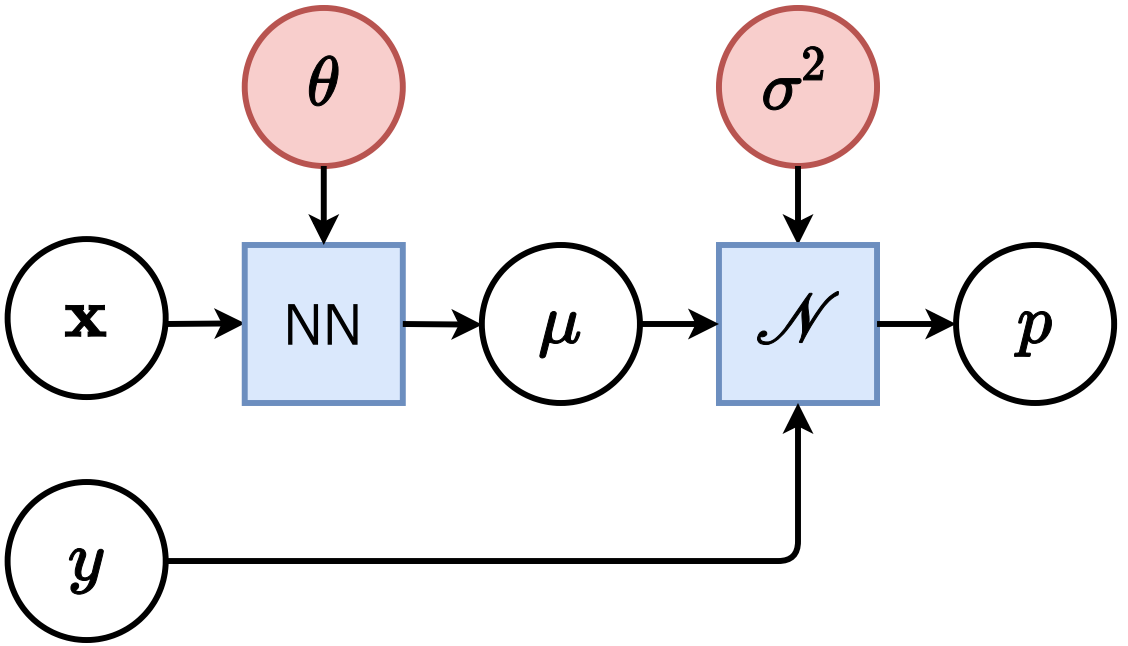
NN with Gaussian output layer

We can model aleatoric uncertainty in the output by modelling the conditional distribution as a Gaussian distribution,

$$p(y|\mathbf{x}) = \mathcal{N}(y; \mu(\mathbf{x}), \sigma^2(\mathbf{x})),$$

where $\mu(x)$ and $\sigma^2(\mathbf{x})$ are parametric functions to be learned, such as neural networks.

Note: The Gaussian distribution is a modelling choice. Other parametric distributions can be used.

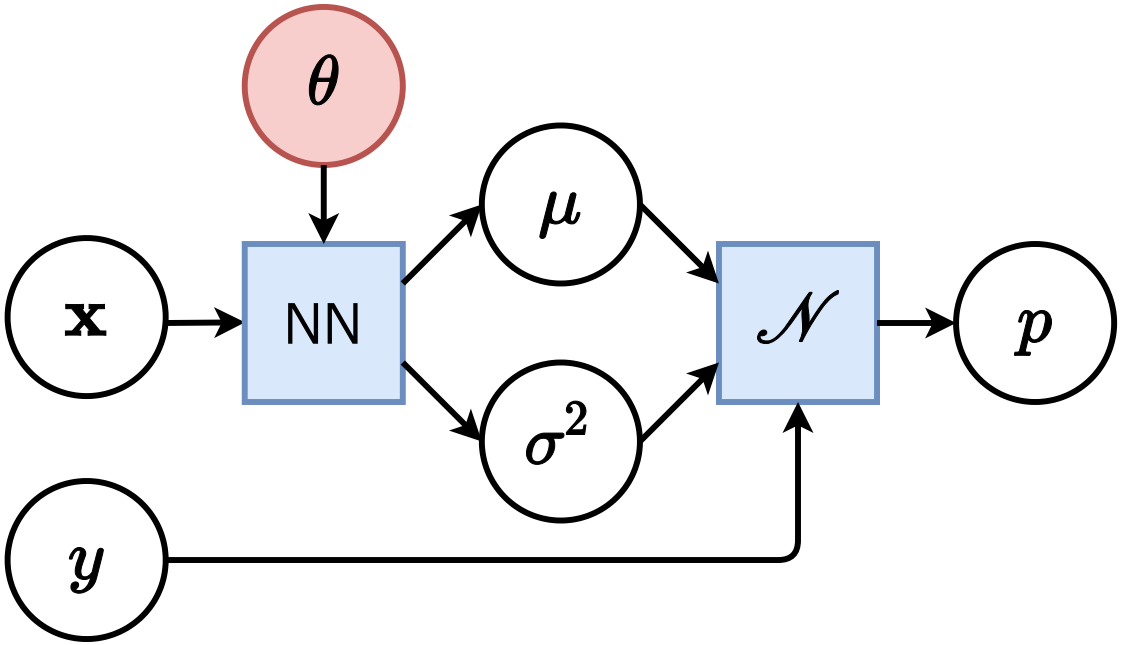


Case 1: Homoscedastic aleatoric uncertainty

We have,

$$
\begin{aligned}
&\arg\max_{\theta,\sigma^2} p(\mathbf{d}|\theta,\sigma^2) \\
&= \arg\max_{\theta,\sigma^2} \prod_{\mathbf{x}_i,y_i\in\mathbf{d}} p(y_i|\mathbf{x}_i,\theta,\sigma^2) \\
&= \arg\max_{\theta,\sigma^2} \prod_{\mathbf{x}_i,y_i\in\mathbf{d}} \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i-\mu(\mathbf{x}_i))^2}{2\sigma^2}\right) \\
&= \arg\min_{\theta,\sigma^2} \sum_{\mathbf{x}_i,y_i\in\mathbf{d}} \frac{(y_i-\mu(\mathbf{x}_i))^2}{2\sigma^2} + \log(\sigma) + C
\end{aligned}
$$

What if σ^2 was fixed?



Case 2: Heteroscedastic aleatoric uncertainty

Same as for the homoscedastic case, except that that σ^2 is now a function of $\mathbf{x}_i$:

$$
\begin{aligned}
&\arg\max_{\theta} p(\mathbf{d}|\theta) \\
&= \arg\max_{\theta} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} p(y_i|\mathbf{x}_i, \theta) \\
&= \arg\max_{\theta} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} \frac{1}{\sqrt{2\pi}\sigma(\mathbf{x}_i)} \exp\left(-\frac{(y_i - \mu(\mathbf{x}_i))^2}{2\sigma^2(\mathbf{x}_i)}\right) \\
&= \arg\min_{\theta} \sum_{\mathbf{x}_i, y_i \in \mathbf{d}} \frac{(y_i - \mu(\mathbf{x}_i))^2}{2\sigma^2(\mathbf{x}_i)} + \log(\sigma(\mathbf{x}_i)) + C
\end{aligned}
$$

What is the purpose of $2\sigma^2(\mathbf{x}_i)$? What about $\log(\sigma(\mathbf{x}_i))$?

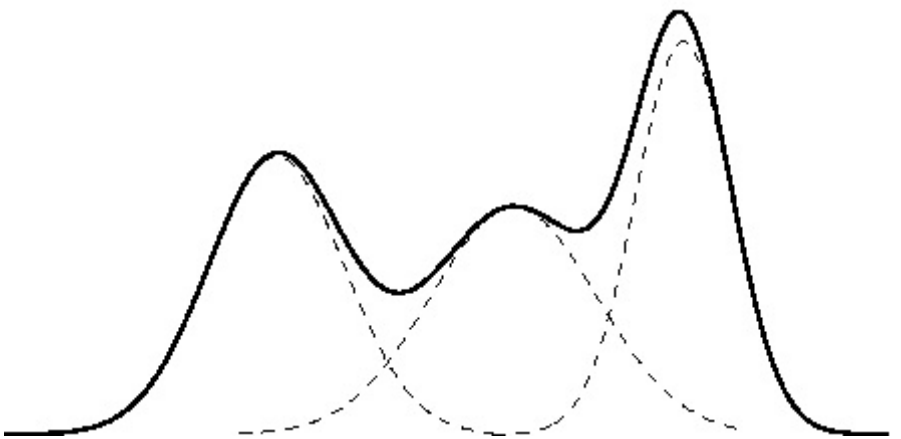
Modelling $p(y|\mathbf{x})$ as a unimodal (Gaussian) distribution can be inadequate since the conditional distribution may be **multimodal**.

Gaussian mixture model

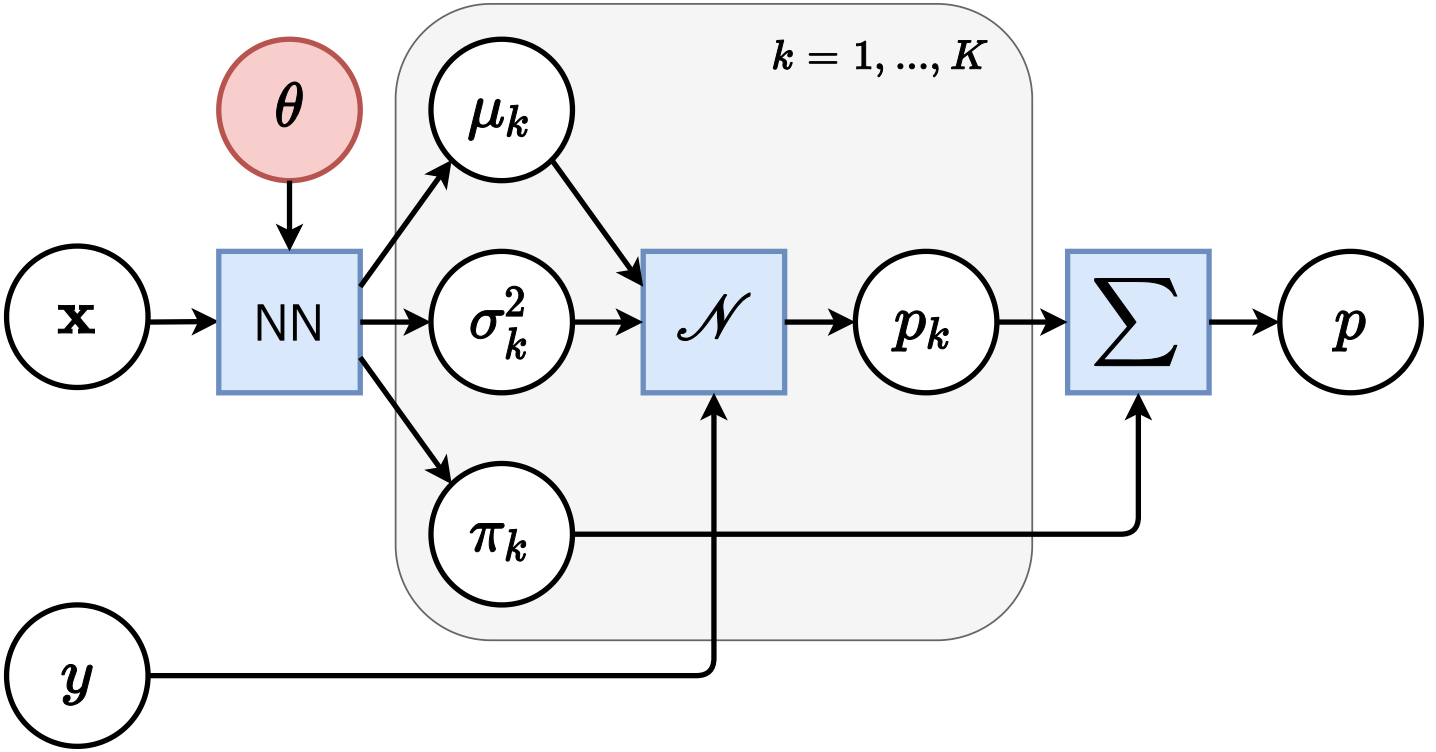
A Gaussian mixture model (GMM) defines instead $p(y|\mathbf{x})$ as a mixture of K Gaussian components,

$$p(y|\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(y; \mu_k, \sigma_k^2),$$

where $0 \leq \pi_k \leq 1$ for all k and $\sum_{k=1}^{K} \pi_k = 1$.



A mixture density network (MDN) is a neural network implementation of the Gaussian mixture model.

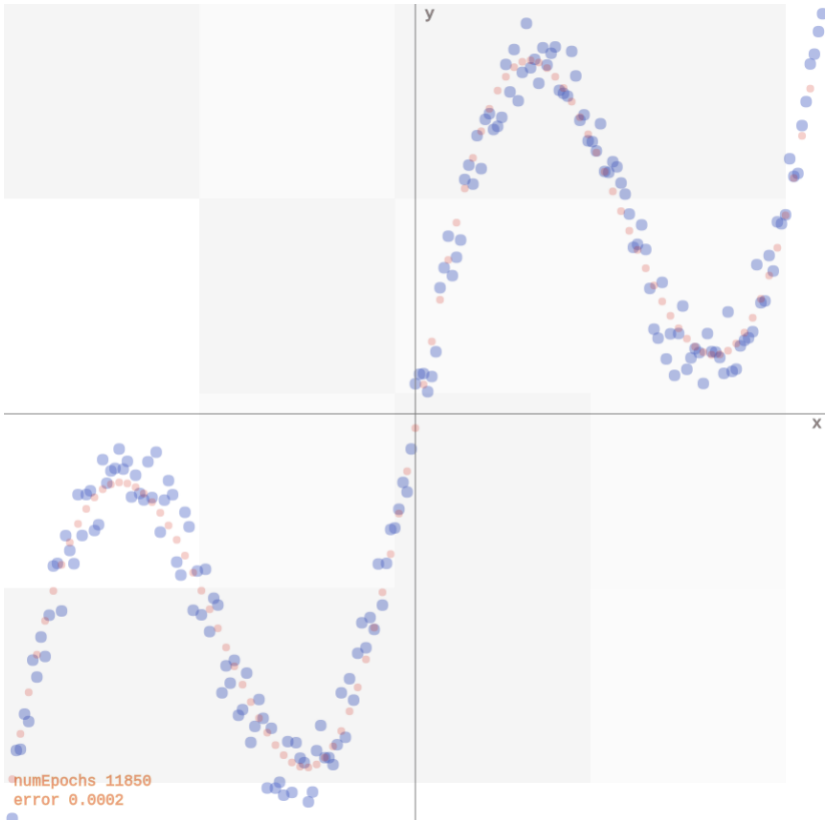


Illustration

Let us consider training data generated randomly as

$$y_i = \mathbf{x}_i + 0.3 \sin(4\pi \mathbf{x}_i) + \epsilon_i$$

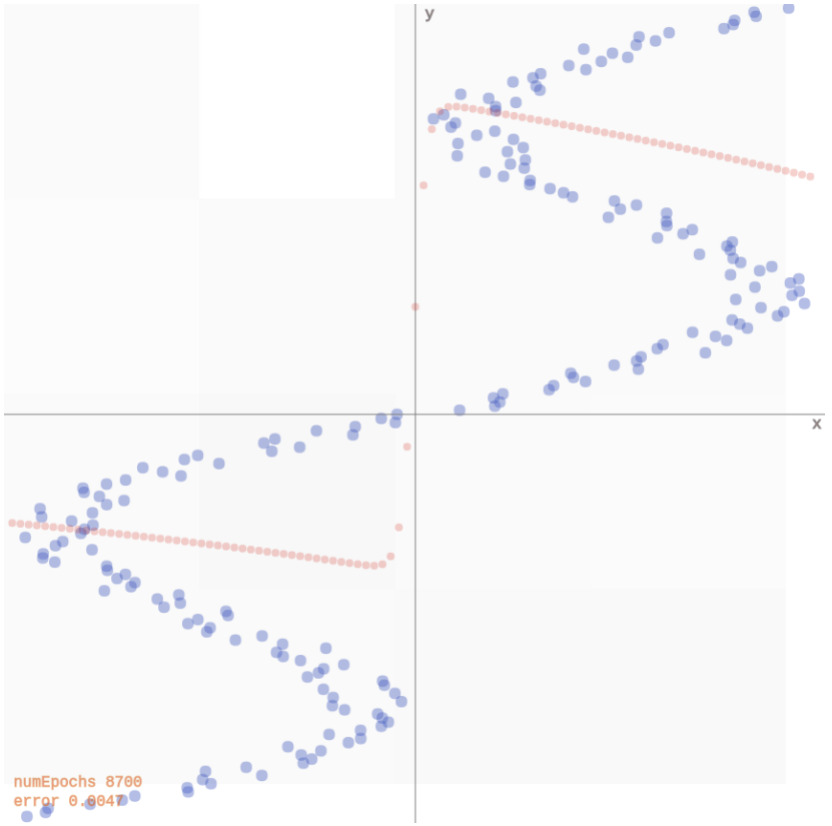
with $\epsilon_i \sim \mathcal{N}$.



The data can be fit with a 2-layer network producing point estimates for y (demo).

—

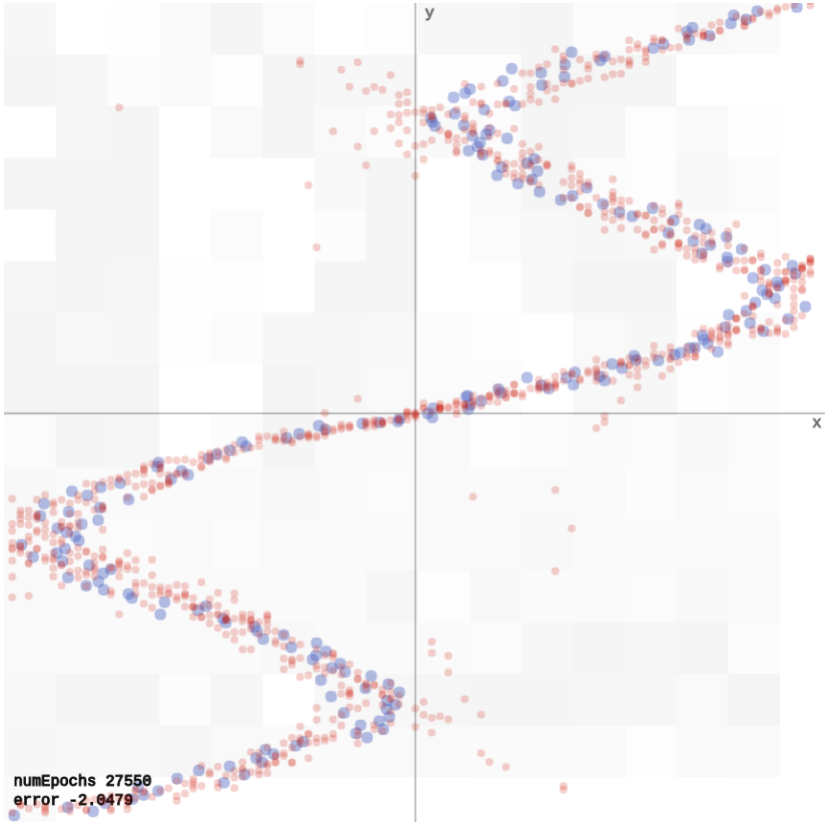
Credits: David Ha, Mixture Density Networks, 2015.



If we flip $\mathbf{x}_i$ and y_i, the network faces issues since for each input, there are multiple outputs that can work. It produces an average of the correct values (demo).

—

Credits: David Ha, Mixture Density Networks, 2015.

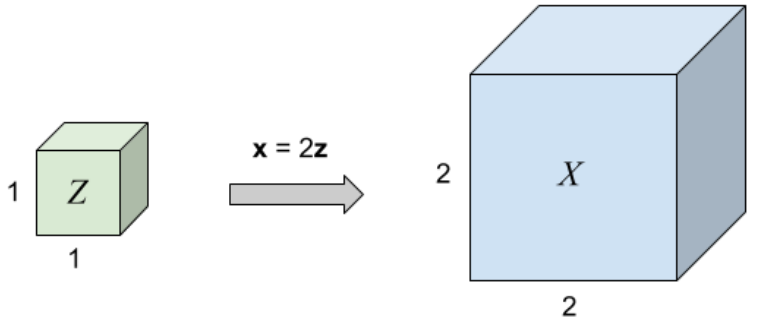


A mixture density network models the data correctly, as it predicts for each input a distribution for the output, rather than a point estimate (demo).

—

Credits: David Ha, Mixture Density Networks, 2015.

Normalizing flows



Assume $p(\mathbf{z})$ is a uniformly distributed unit cube in $\mathbb{R}^3$ and $\mathbf{x} = f(\mathbf{z}) = 2\mathbf{z}$.

Since the total probability mass must be conserved,

$$p(\mathbf{x} = f(\mathbf{z})) = p(\mathbf{z}) \frac{V_{\mathbf{z}}}{V_{\mathbf{x}}} = p(\mathbf{z}) \frac{1}{8},$$

where $\frac{1}{8} = \left| \det \begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{pmatrix} \right|^{-1}$ represents the determinant of the linear transformation f.

What if f is non-linear?

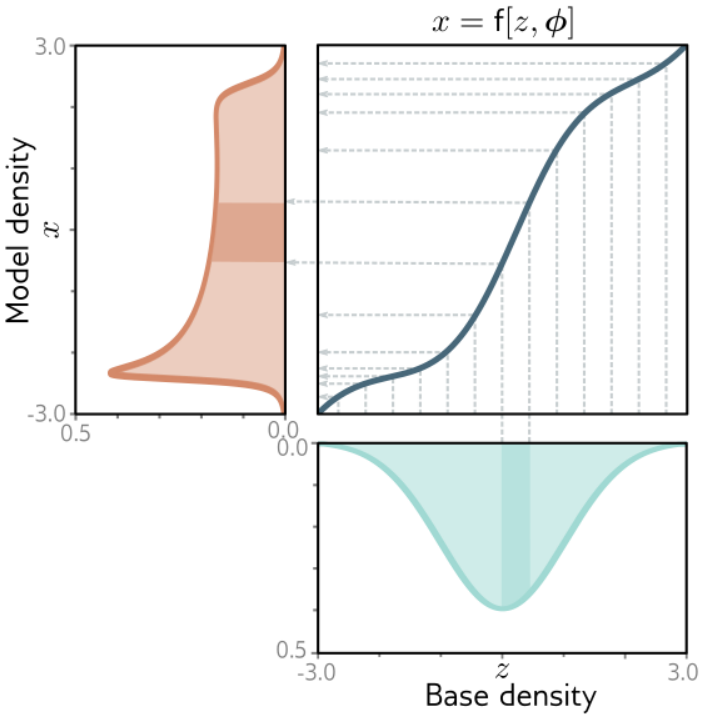


Figure 16.2 Transforming distributions. The base density (cyan, bottom) passes through a function (blue curve, top right) to create the model density (orange, left). Consider dividing the base density into equal intervals (gray vertical lines). The probability mass between adjacent lines must remain the same after transformation. The cyan-shaded region passes through a part of the function where the gradient is larger than one, so this region is stretched. Consequently, the height of the orange-shaded region must be lower so that it retains the same area as the cyan-shaded region. In other places (e.g., $z = -2$), the gradient is less than one, and the model density increases relative to the base density.

—

Image credits: Simon J.D. Prince, Understanding Deep Learning, 2023.

Change of variable theorem

If f is non-linear,

- the Jacobian $J_f(\mathbf{z})$ of $\mathbf{x} = f(\mathbf{z})$ represents the infinitesimal linear transformation in the neighborhood of $\mathbf{z}$;
- if the function is a bijective map, then the mass must be conserved locally.

Therefore, the local change of density yields

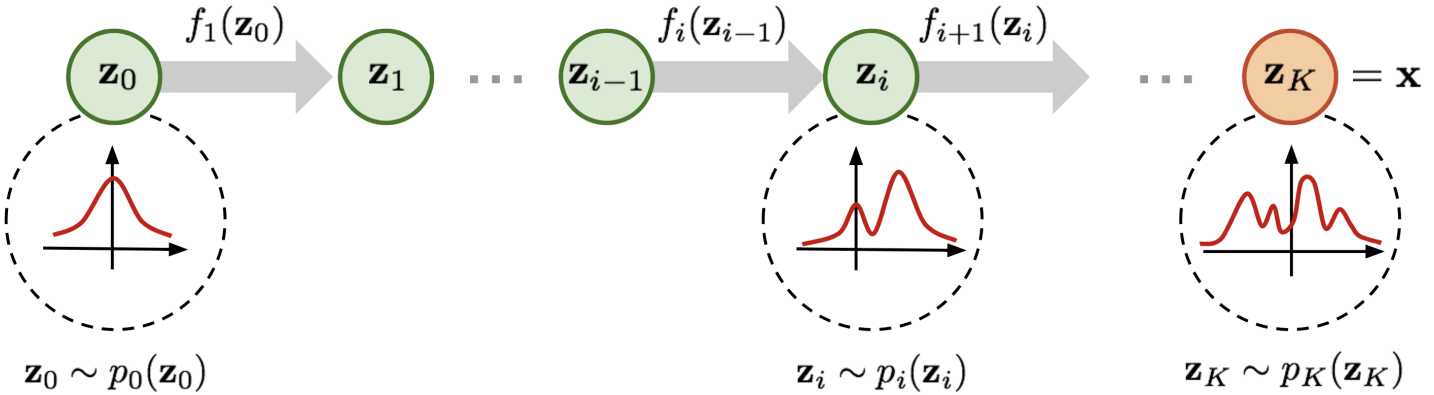
$$p(\mathbf{x} = f(\mathbf{z})) = p(\mathbf{z}) \left|\det J_f(\mathbf{z})\right|^{-1} .$$

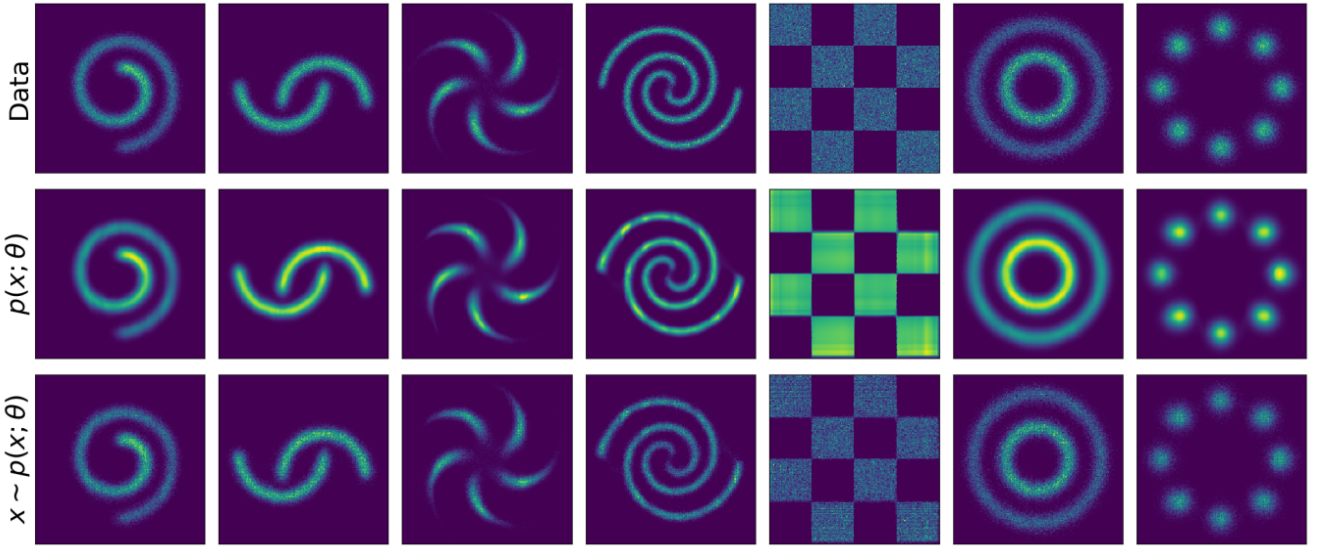
Similarly, for $g = f^{-1}$, we have

$$p(\mathbf{x}) = p(\mathbf{z} = g(\mathbf{x})) \left|\det J_g(\mathbf{x})\right| .$$

Normalizing flows

A normalizing flow is a change of variable f that transforms a base distribution $p(\mathbf{z})$ into $p(\mathbf{x})$ by a series of invertible transformations.





Normalizing flows shine in applications for **density estimation**, where access to the density function is required.

—

Image credits: Wehenkel and Louppe, 2019.

Formally,

- f is a composition $f = f_K \circ \ldots \circ f_1$, where each f_k is an invertible neural transformation;
- $g_k = f_k^{-1}$;
- $\mathbf{z}_k = f_k(\mathbf{z}_{k-1})$, with $\mathbf{z}_0 = \mathbf{z}$ and $\mathbf{z}_K = \mathbf{x}$;
- $p(\mathbf{z}_k) = p(\mathbf{z}_{k-1} = g_k(\mathbf{z}_k)) \left|\det J_{g_k}(\mathbf{z}_k)\right|$.

—
Image credits: Lilian Weng, 2018.

Example: coupling layers

Assume $\mathbf{z} = (\mathbf{z}_a, \mathbf{z}_b)$ and $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$. Then,

- Forward mapping $\mathbf{x} = f(\mathbf{z})$:

$$\mathbf{x}_a = \mathbf{z}_a, \quad \mathbf{x}_b = \mathbf{z}_b \odot \exp(s(\mathbf{z}_a)) + t(\mathbf{z}_a),$$

- Inverse mapping $\mathbf{z} = g(\mathbf{x})$:

$$\mathbf{z}_a = \mathbf{x}_a, \quad \mathbf{z}_b = (\mathbf{x}_b - t(\mathbf{x}_a)) \odot \exp(-s(\mathbf{x}_a)),$$

where s and t are arbitrary neural networks.

For $\mathbf{x} = (\mathbf{x}_a, \mathbf{x}_b)$, the log-likelihood is

$$\log p(\mathbf{x}) = \log p(\mathbf{z} = g(\mathbf{x})) \left|\det J_g(\mathbf{x})\right|$$

where the Jacobian $J_g(\mathbf{x}) = \frac{\partial \mathbf{z}}{\partial \mathbf{x}}$ is a lower triangular matrix

$$\begin{pmatrix} \mathbf{I} & 0 \\ \frac{\partial \mathbf{z}_b}{\partial \mathbf{x}_a} & \operatorname{diag}(\exp(-s(\mathbf{x}_a))) \end{pmatrix},$$

such that $\left|\det J_g(\mathbf{x})\right| = \prod_i \exp(-s(\mathbf{x}_a))_i = \exp(-\sum_i s(\mathbf{x}_a)_i)$.

Therefore, the log-likelihood is

$$\log p(\mathbf{x}) = \log p(\mathbf{z} = g(\mathbf{x})) - \sum_i s(\mathbf{x}_a)_i.$$

Epistemic uncertainty

Epistemic uncertainty accounts for uncertainty in the model or in its parameters. It captures our ignorance about which model generated the collected data.

It can be explained away given enough data.

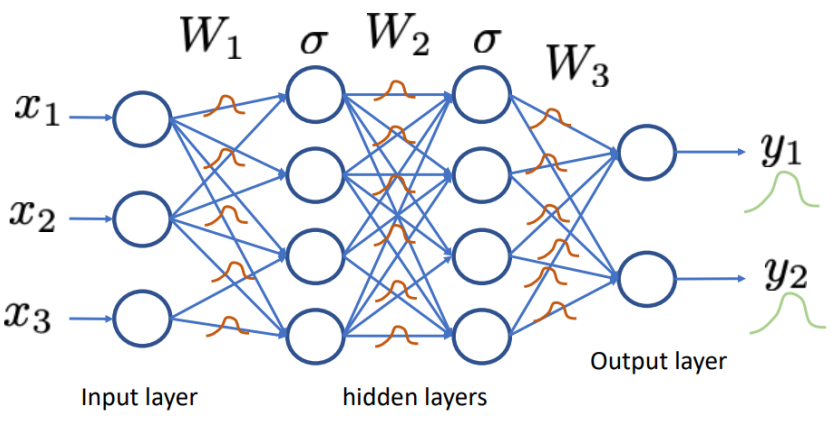
—

Credits: Kendall and Gal, What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?, 2017.

Bayesian neural networks

To capture epistemic uncertainty in a neural network, we model our ignorance with a prior distribution $p(\omega)$ over its weights.

Then we invoke Bayes for making predictions.



The prior predictive distribution at $\mathbf{x}$ is given by integrating over all possible weight configurations,

$$p(y|\mathbf{x}) = \int p(y|\mathbf{x}, \omega) p(\omega) d\omega.$$

Given training data $\mathbf{d} = \{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$ a Bayesian update results in the posterior

$$p(\omega|\mathbf{d}) = \frac{p(\mathbf{d}|\omega) p(\omega)}{p(\mathbf{d})}$$

where the likelihood $p(\mathbf{d}|\omega) = \prod_i p(y_i|\mathbf{x}_i, \omega)$.

The posterior predictive distribution is then given by

$$p(y|\mathbf{x}, \mathbf{d}) = \int p(y|\mathbf{x}, \omega) p(\omega|\mathbf{d}) d\omega.$$

Bayesian neural networks are easy to formulate, but notoriously difficult to perform inference in.

$p(\mathbf{d})$ is intractable to evaluate, which results in the posterior $p(\omega|\mathbf{d})$ not being tractable either.

Therefore, we must rely on approximations.

Variational inference

Variational inference can be used for building an approximation $q(\omega; \nu)$ of the posterior $p(\omega|\mathbf{d})$.

We can show that minimizing

$$\mathrm{KL}(q(\omega; \nu)||p(\omega|\mathbf{d}))$$

with respect to the variational parameters ν, is identical to maximizing the evidence lower bound objective (ELBO)

$$\mathrm{ELBO}(\nu) = \mathbb{E}_{q(\omega;\nu)}\left[\log p(\mathbf{d}|\omega)\right] - \mathrm{KL}(q(\omega; \nu)||p(\omega)).$$

The integral in the ELBO is not tractable for almost all q, but it can be minimized with stochastic gradient descent:

1. Sample $\hat{\omega} \sim q(\omega; \nu)$.
2. Do one step of maximization with respect to ν on

$$\hat{L}(\nu) = \log p(\mathbf{d}|\hat{\omega}) - \mathrm{KL}(q(\omega; \nu)||p(\omega))$$

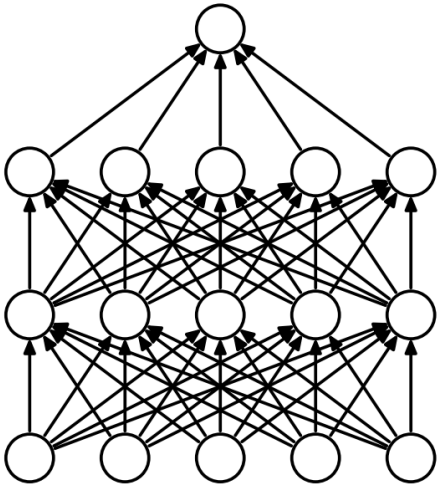
In the context of Bayesian neural networks, this procedure is also known as Bayes by backprop (Blundell et al, 2015).

Dropout

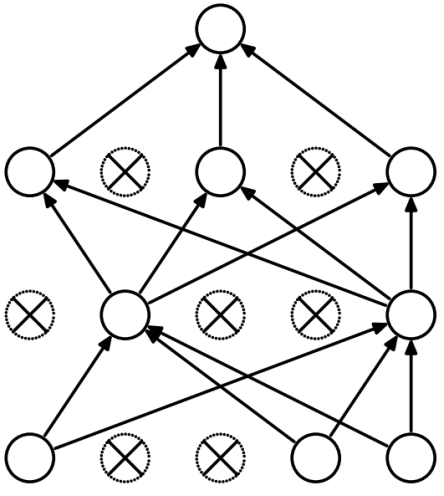
Dropout is an empirical technique that was first proposed to avoid overfitting in neural networks.

At each training step (i.e., for each sample within a mini-batch):

- Remove each node in the network with a probability p.
- Update the weights of the remaining nodes with backpropagation.



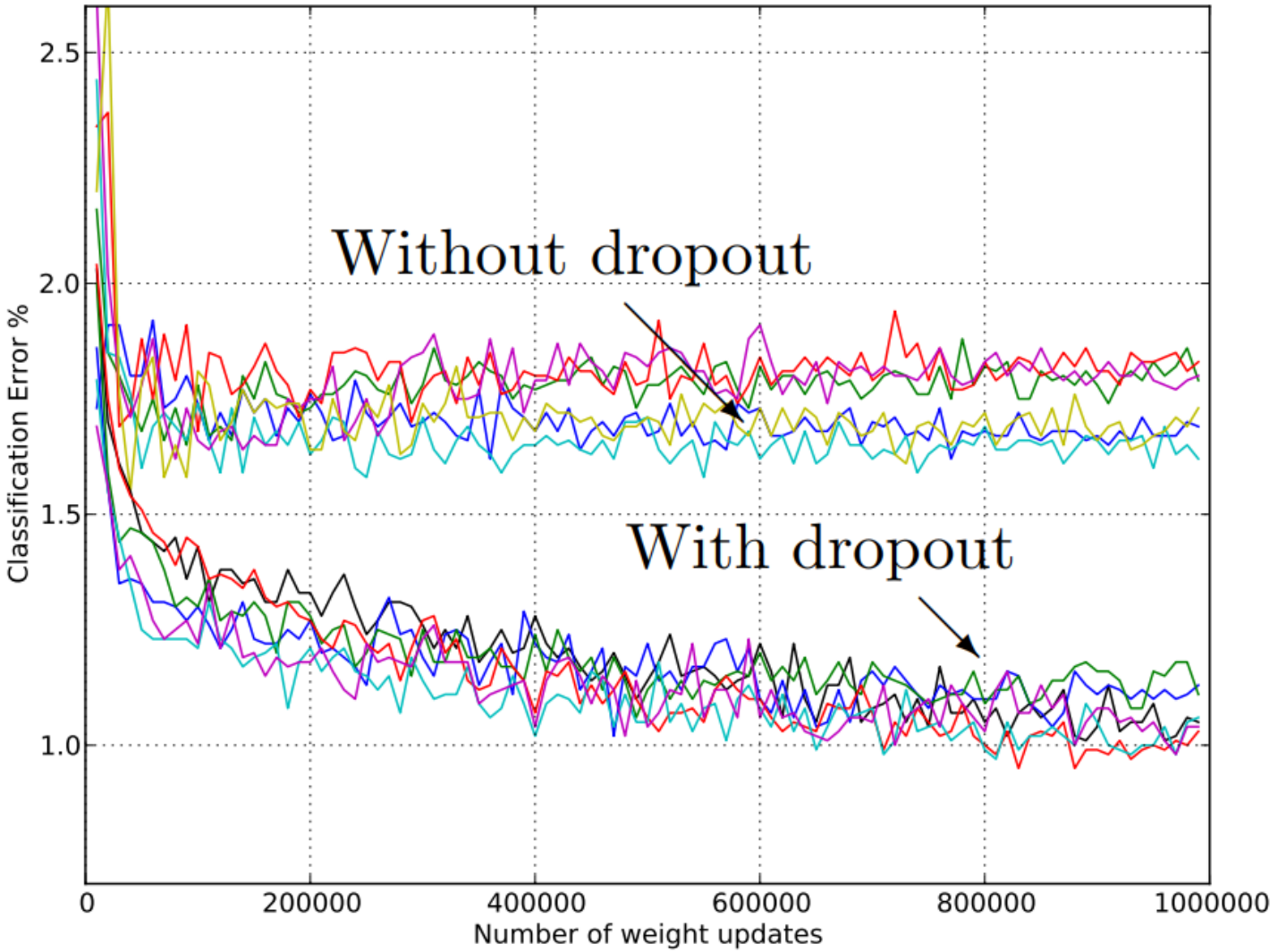
(a) Standard Neural Net



(b) After applying dropout.

At test time, either:

- Make predictions using the trained network without dropout but rescaling the weights by the dropout probability p (fast and standard).
- Sample T neural networks using dropout and average their predictions (slower but better principled).


Without dropout
With dropout
Classification Error %
2.5
2.0
1.5
1.0
0
200000
400000
600000
800000
1000000
Number of weight updates

Why does dropout work?

- It makes the learned weights of a node less sensitive to the weights of the other nodes.
- This forces the network to learn several independent representations of the patterns and thus decreases overfitting.
- It approximates Bayesian model averaging.

Dropout does variational inference

What variational family q would correspond to dropout?

- Let us split the weights ω per layer, $\omega = \{\mathbf{W}_1, ..., \mathbf{W}_L\}$, where $\mathbf{W}_i$ is further split per unit $\mathbf{W}_i = \{\mathbf{w}_{i,1}, ..., \mathbf{w}_{i,q_i}\}$.
- Variational parameters ν are split similarly into $\nu = \{\mathbf{M}_1, ..., \mathbf{M}_L\}$, with $\mathbf{M}_i = \{\mathbf{m}_{i,1}, ..., \mathbf{m}_{i,q_i}\}$.
- Then, the proposed $q(\omega; \nu)$ is defined as follows:

$$q(\omega; \nu) = \prod_{i=1}^{L} q(\mathbf{W}_i; \mathbf{M}_i)$$
$$q(\mathbf{W}_i; \mathbf{M}_i) = \prod_{k=1}^{q_i} q(\mathbf{w}_{i,k}; \mathbf{m}_{i,k})$$
$$q(\mathbf{w}_{i,k}; \mathbf{m}_{i,k}) = p\delta_0(\mathbf{w}_{i,k}) + (1-p)\delta_{\mathbf{m}_{i,k}}(\mathbf{w}_{i,k})$$

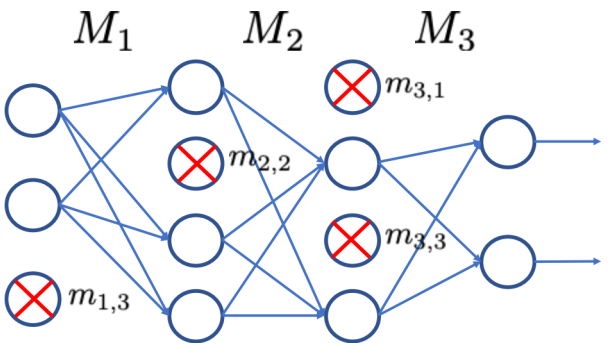
where $\delta_a(x)$ denotes a (multivariate) Dirac distribution centered at a.

Given the previous definition for q, sampling parameters $\hat{\omega} = \{\hat{\mathbf{W}}_1, ..., \hat{\mathbf{W}}_L\}$ is done as follows:

- Draw binary $z_{i,k} \sim \text{Bernoulli}(1-p)$ for each layer i and unit k.
- Compute $\hat{\mathbf{W}}_i = \mathbf{M}_i \text{diag}([z_{i,k}]_{k=1}^{q_i})$, where $\mathbf{M}_i$ denotes a matrix composed of the columns $\mathbf{m}_{i,k}$.

That is, $\hat{\mathbf{W}}_i$ are obtained by setting columns of $\mathbf{M}_i$ to zero with probability p.

This is strictly equivalent to dropout, i.e. removing units from the network with probability p.



Therefore, one step of stochastic gradient descent on the ELBO becomes:

1. Sample $\hat{\omega} \sim q(\omega; \nu) \Leftrightarrow$ Randomly set units of the network to zero $\Leftrightarrow$ Dropout.
2. Do one step of maximization with respect to $\nu = \{\mathbf{M}_i\}$ on

$$\hat{L}(\nu) = \log p(\mathbf{d}|\hat{\omega}) - \mathrm{KL}(q(\omega; \nu)||p(\omega)).$$

Maximizing $\hat{L}(\nu)$ is equivalent to minimizing

$$-\hat{L}(\nu) = -\log p(\mathbf{d}|\hat{\omega}) + \mathrm{KL}(q(\omega;\nu)||p(\omega))$$

This is also equivalent to one minimization step of a standard classification or regression objective:

- The first term is the typical objective (such as the cross-entropy).
- The second term forces q to remain close to the prior $p(\omega)$.
  - If $p(\omega)$ is Gaussian, minimizing the KL is equivalent to ℓ_2 regularization.
  - If $p(\omega)$ is Laplacian, minimizing the KL is equivalent to ℓ_1 regularization.

Conversely, this shows that when training a network with dropout with a standard classification or regression objective, one is actually implicitly doing variational inference to match the posterior distribution of the weights.

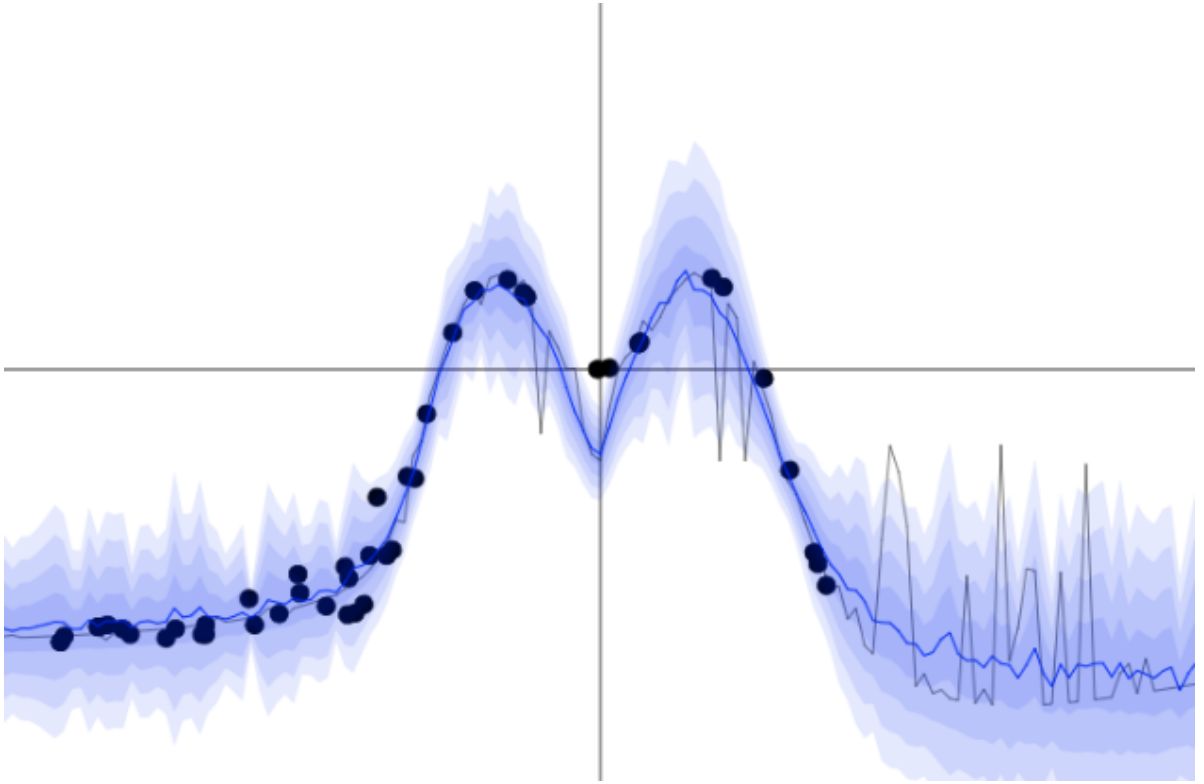
Uncertainty estimates from dropout

Proper epistemic uncertainty estimates at $\mathbf{x}$ can be obtained in a principled way using Monte-Carlo integration:

- Draw T sets of network parameters $\hat{\omega}_t$ from $q(\omega; \nu)$.
- Compute the predictions for the T networks, $\{f(\mathbf{x}; \hat{\omega}_t)\}_{t=1}^{T}$.
- Approximate the predictive mean and variance as follows:

$$\mathbb{E}_{p(y|\mathbf{x},\mathbf{d})}[y] \approx \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{x}; \hat{\omega}_t)$$

$$\mathbb{V}_{p(y|\mathbf{x},\mathbf{d})}[y] \approx \sigma^2 + \frac{1}{T}\sum_{t=1}^{T} f(\mathbf{x}; \hat{\omega}_t)^2 - \hat{\mathbb{E}}[y]^2$$



(demo)

Pixel-wise depth regression

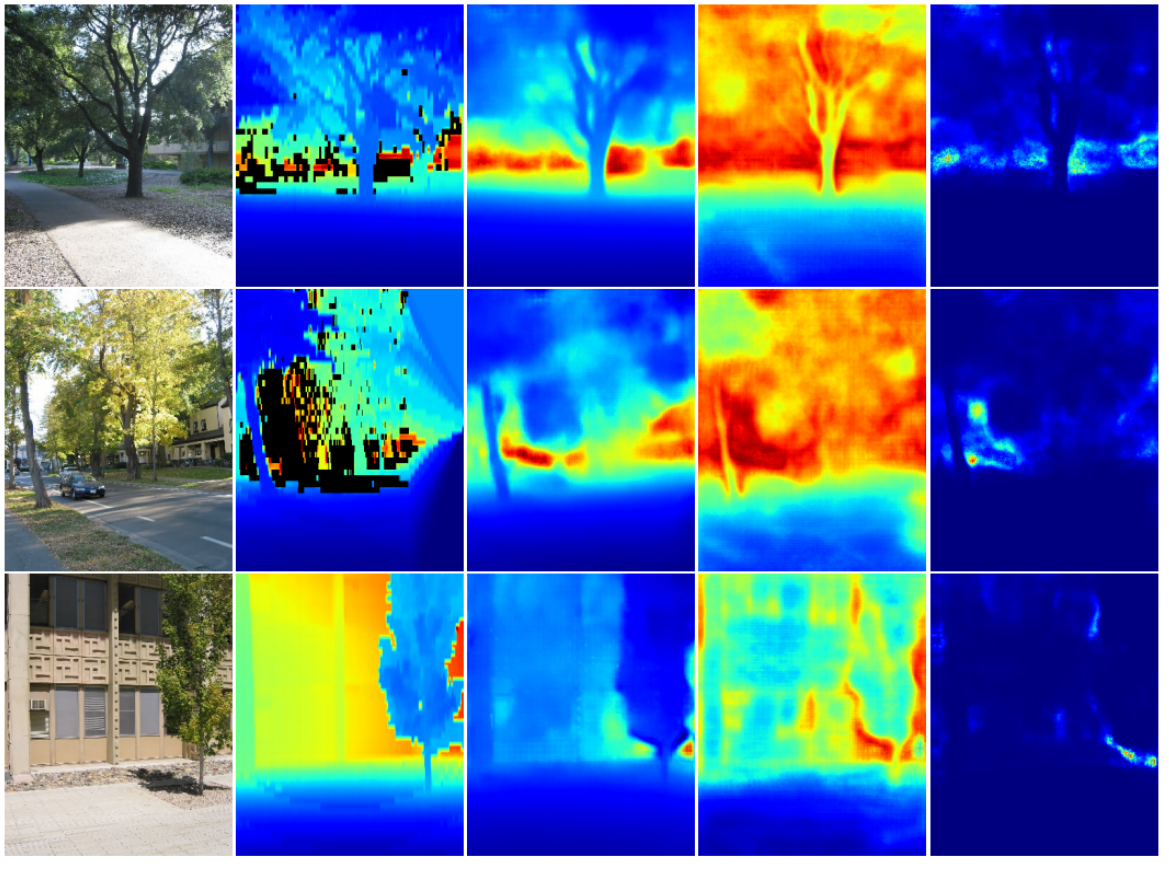


Figure 6: Qualitative results on the Make3D depth regression dataset. Left to right: input image, ground truth, depth prediction, aleatoric uncertainty, epistemic uncertainty. Make3D does not provide labels for depth greater than 70m, therefore these distances dominate the epistemic uncertainty signal. Aleatoric uncertainty is prevalent around depth edges or distant points.

—

Credits: Kendall and Gal, What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision?, 2017.

The end.