Deep Learning

Lecture 10: Uncertainty

Prof. Gilles Louppe
g.louppe@uliege.be

LIÈGE université

Today

How to model uncertainty in deep learning?

- Uncertainty
- Aleatoric uncertainty
- Epistemic uncertainty
- Adversarial attacks

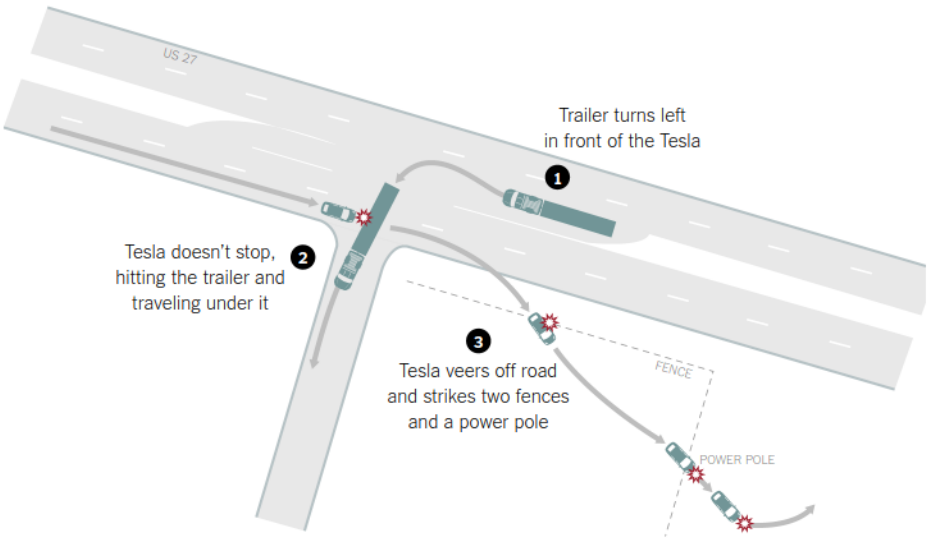
*"Every time a scientific paper presents a bit of data, it's accompanied by an **error bar** – a quiet but insistent reminder that no knowledge is complete or perfect. It's a **calibration of how much we trust what we think we know**."*

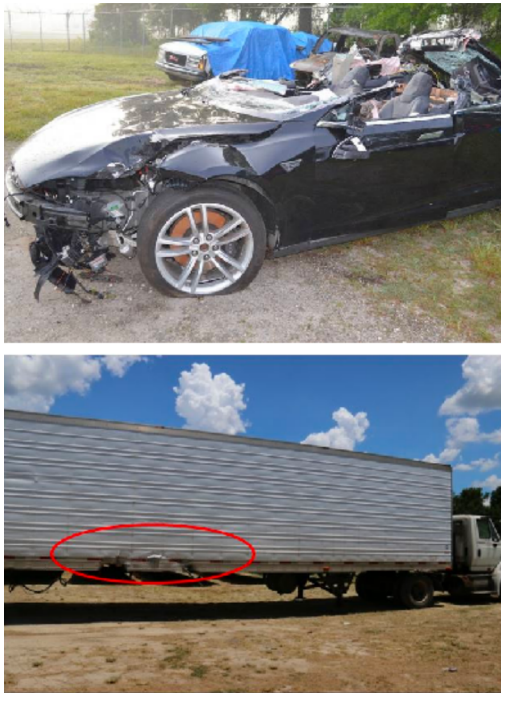
Carl Sagan

Uncertainty

Motivation

In May 2016, there was the first fatality from an assisted driving system, caused by the perception system confusing the white side of a trailer for bright sky.

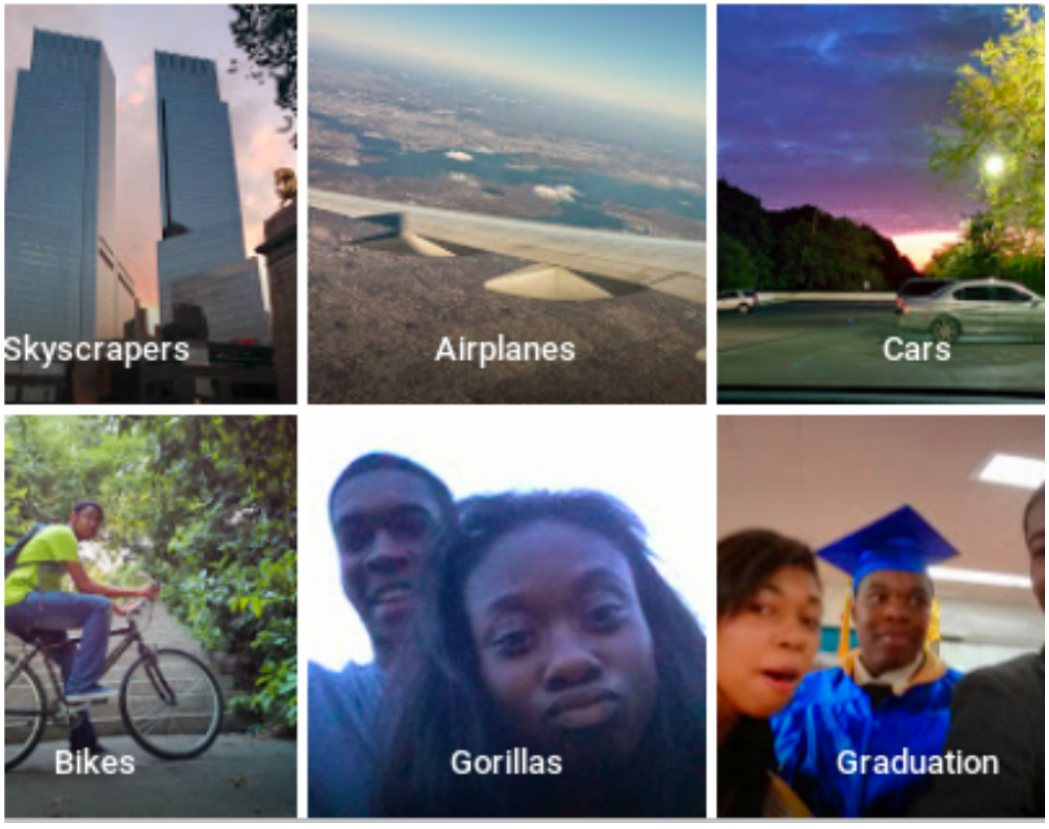




—

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



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 were able to assign a high level of uncertainty to their erroneous predictions, then they may have been able to make better decisions, and likely avoid disaster.

—

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

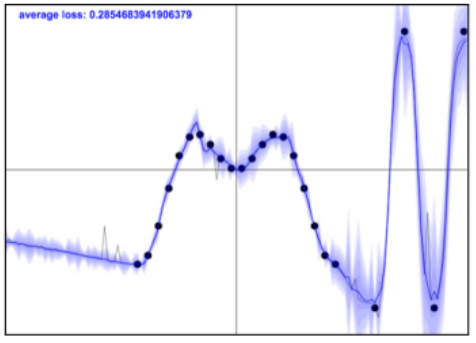
Aleatoric uncertainty

Aleatoric uncertainty captures noise inherent in the observations.

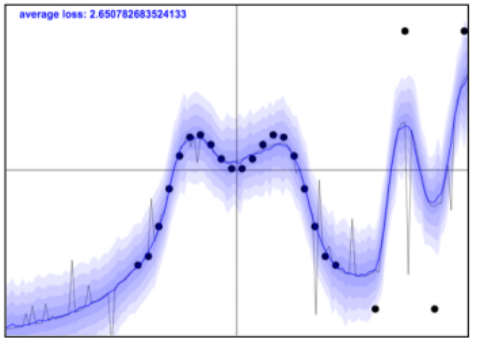
- For example, sensor noise or motion noise result in uncertainty.
- This uncertainty cannot be reduced with more data.
- However, aleatoric uncertainty could be reduced with better measurements.

Aleatoric uncertainty can further be categorized into homoscedastic and heteroscedastic uncertainties:

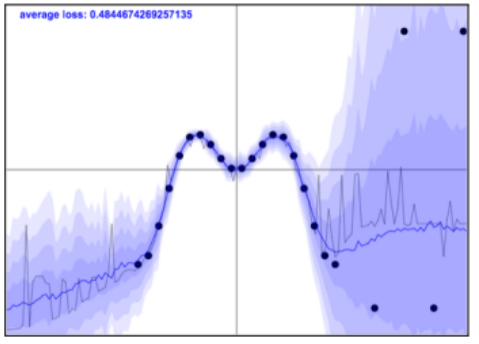
- Homoscedastic uncertainty relates to the uncertainty that a particular task might cause. It stays constant for different inputs.
- Heteroscedastic uncertainty depends on the inputs to the model, with some inputs potentially having more noisy outputs than others.



(a) Homoscedastic model with small observation noise.



(b) Homoscedastic model with large observation noise.



(c) Heteroscedastic model with data-dependent observation noise.

—
Credits: Yarin Gal, Uncertainty in Deep Learning, 2016.

Regression with uncertainty

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

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

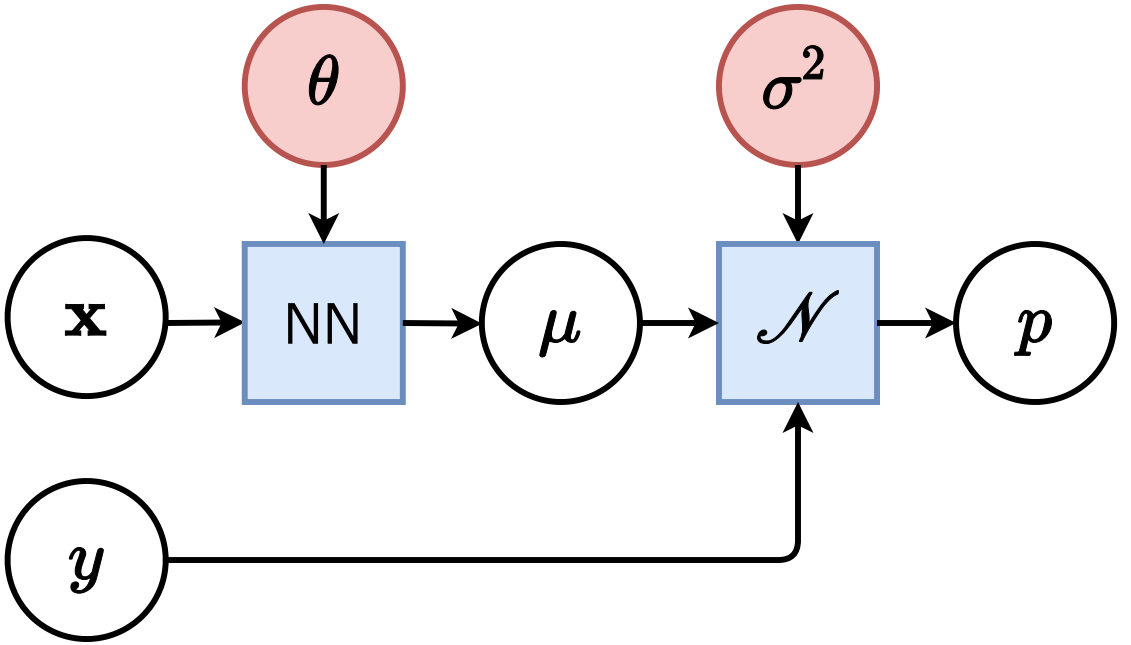
We model aleatoric uncertainty in the output by modelling the conditional distribution as a Normal distribution,

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

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

We do not wish to learn a function $\hat{y} = f(\mathbf{x})$ that would only produce point estimates.

Homoscedastic aleatoric uncertainty

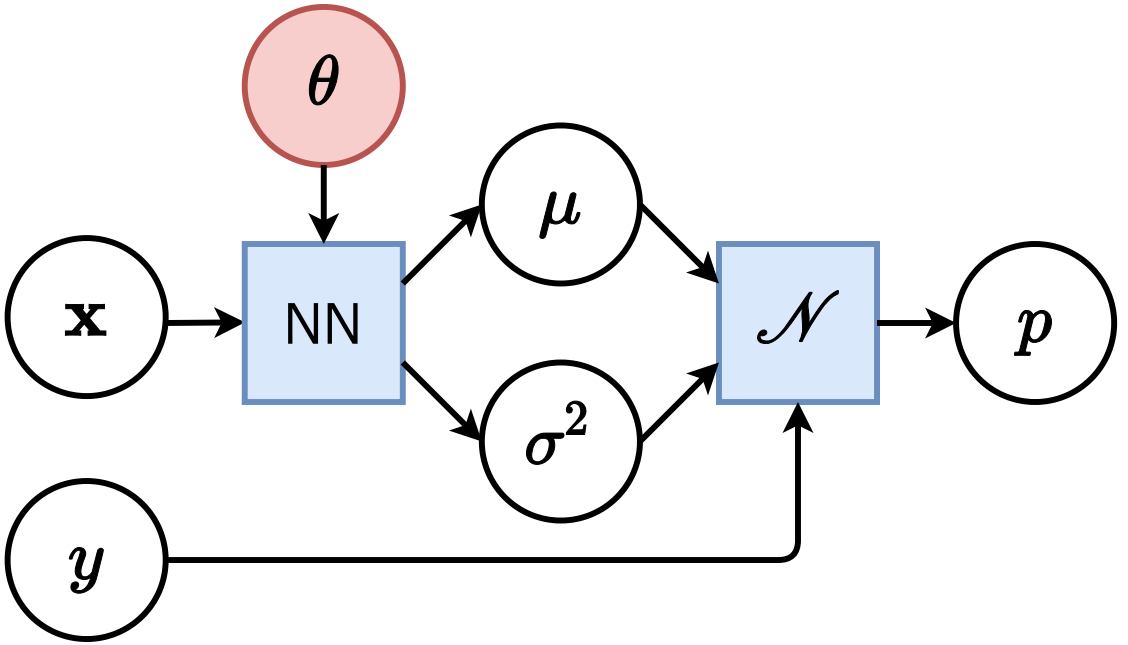


We have,

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

Exercise
What if σ^2 was fixed?

Heteroscedastic aleatoric uncertainty



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

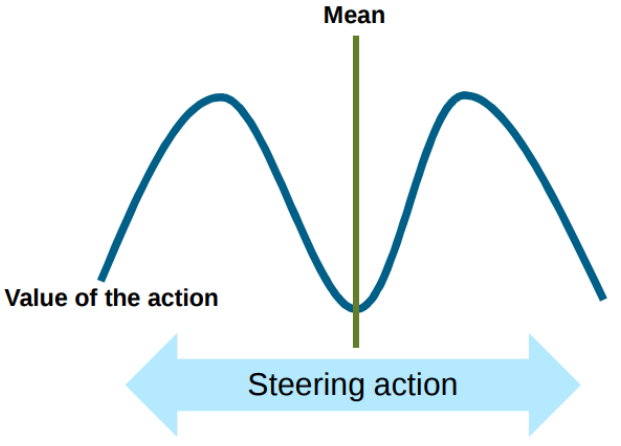
$$
\begin{aligned}
&\arg\max_{\theta} p(\mathbf{d}|\theta) \\
&= \arg\max_{\theta} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} p(y_i|\mathbf{x}_i, \theta) \\
&= \arg\max_{\theta} \prod_{\mathbf{x}_i, y_i \in \mathbf{d}} \frac{1}{\sqrt{2\pi}\sigma(\mathbf{x}_i)} \exp\left(-\frac{(y_i - \mu(\mathbf{x}_i))^2}{2\sigma^2(\mathbf{x}_i)}\right)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

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

Multimodality

Modelling $p(y|\mathbf{x})$ as a unimodal Gaussian is not always a good idea since the conditional distribution may be multimodal.

https://commons.wikimedia.org/wiki/File:Newport_Whitepit_Lane_pot_hole.JPG



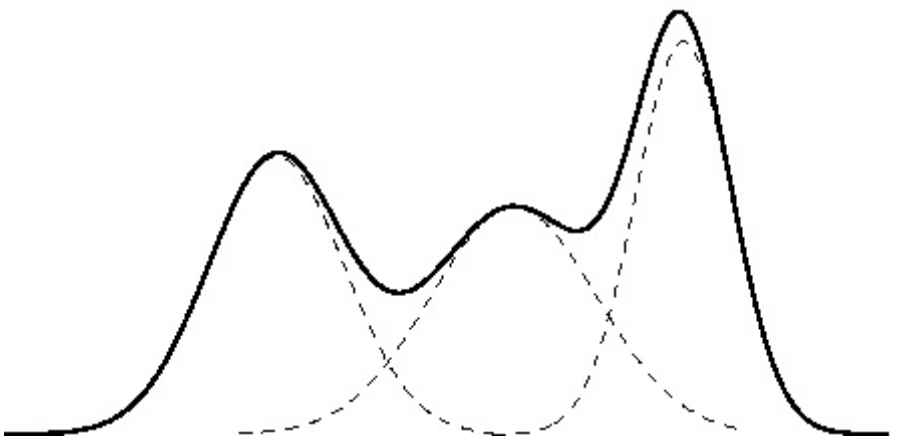
(and it would be even worse to have only point estimates for y!)

Gaussian mixture model

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

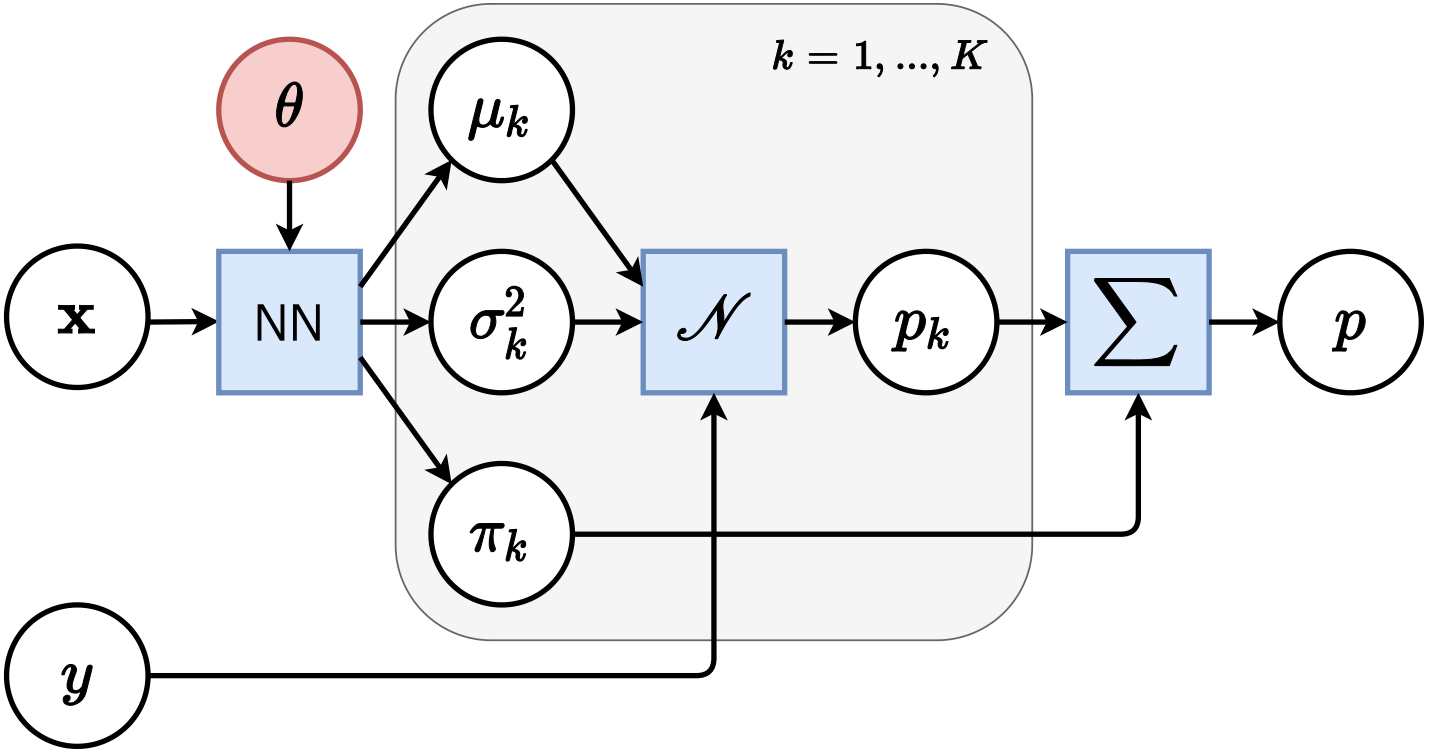
$$p(y|\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(y; \mu_k, \sigma_k^2),$$

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



Mixture density network

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

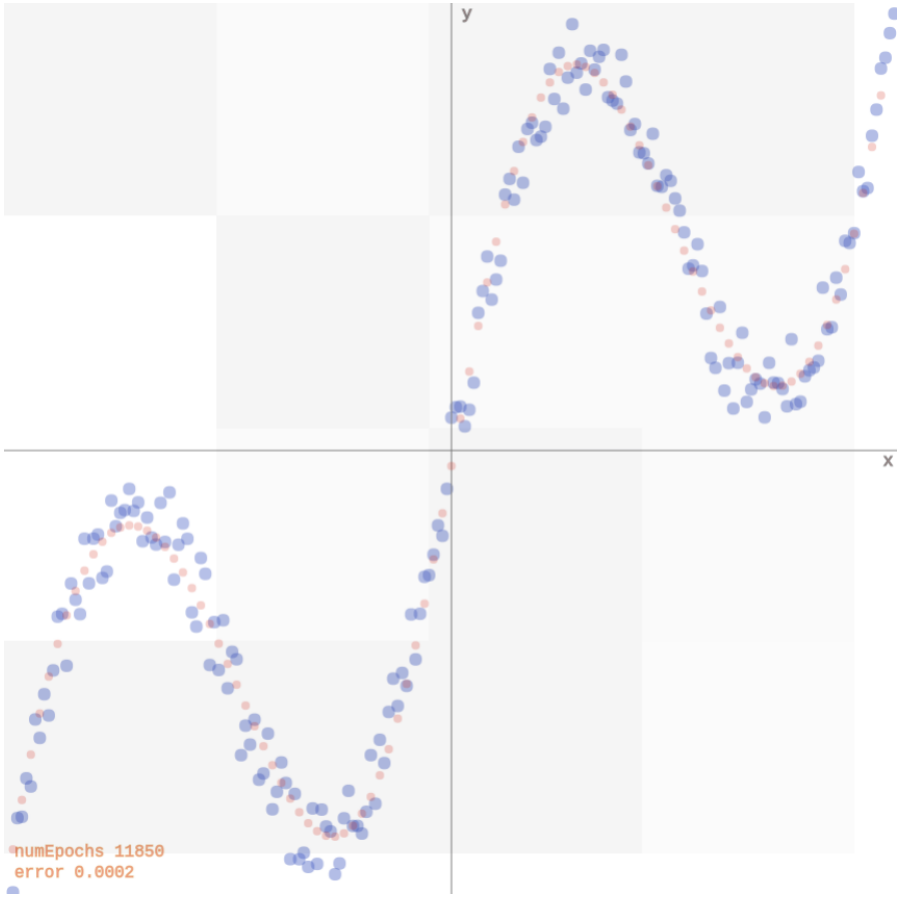


Illustration

Let us consider training data generated randomly as

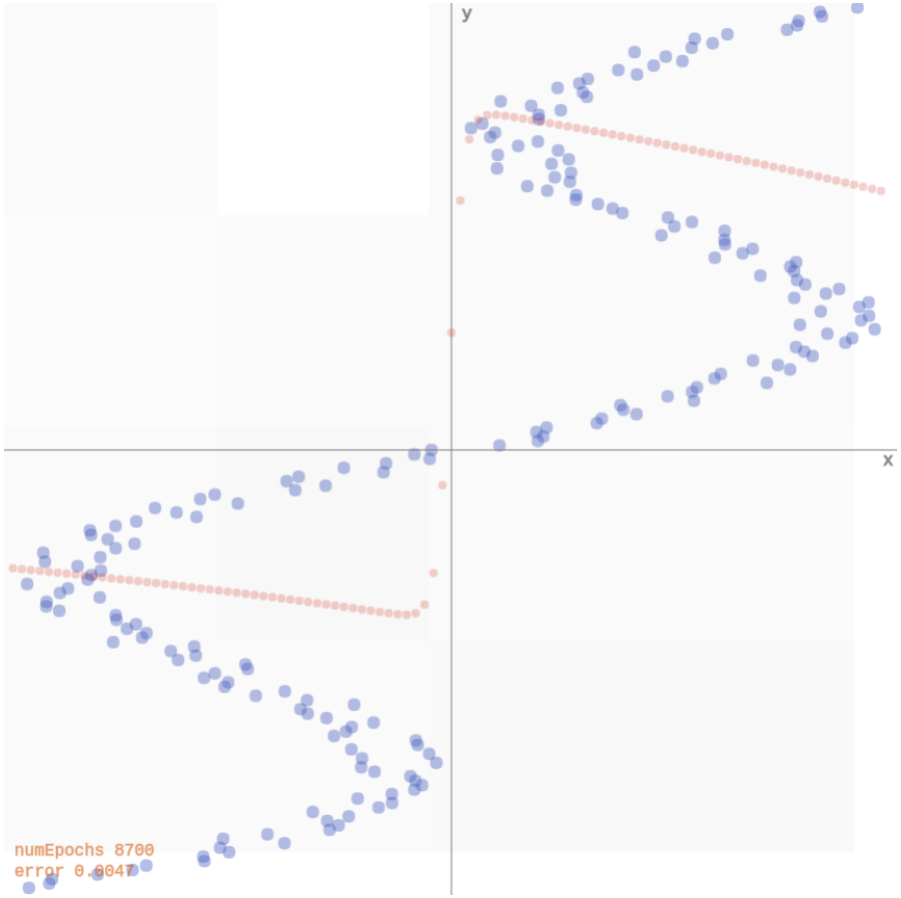
$$y_i = \mathbf{x}_i + 0.3 \sin(4\pi \mathbf{x}_i) + \epsilon_i$$

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



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

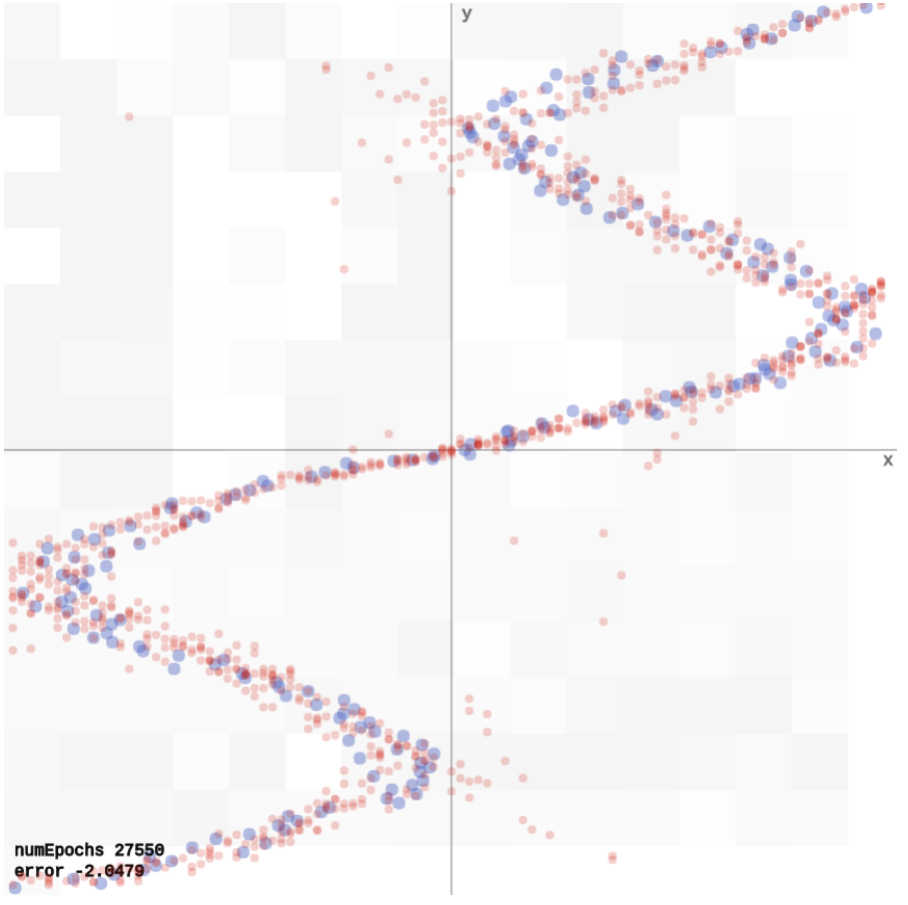
—
Credits: David Ha, Mixture Density Networks, 2015.



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 some sort of average of the correct values. [demo]

—

Credits: David Ha, Mixture Density Networks, 2015.

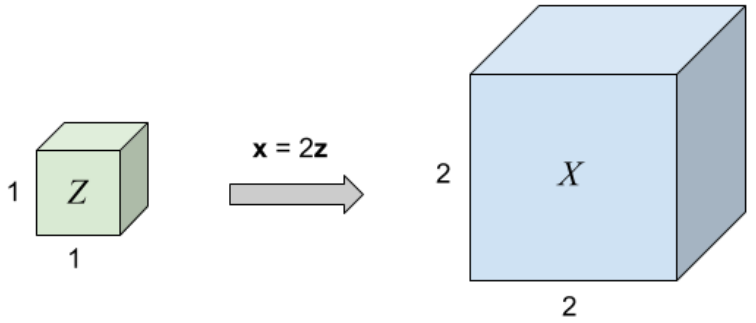


A mixture density network models the data correctly, as it predicts for each input a distribution for the output, rather than a point estimate. [demo]

—

Credits: David Ha, Mixture Density Networks, 2015.

Normalizing flows

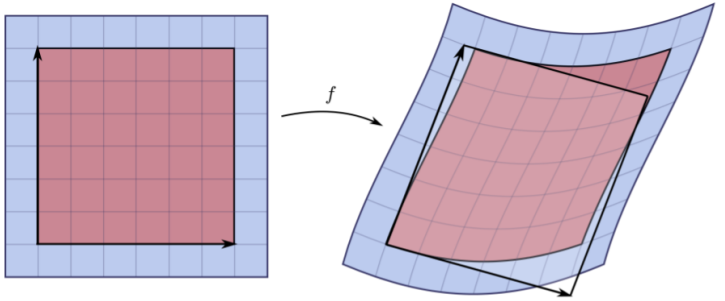


Assume $p(\mathbf{z})$ is a uniformly distributed unit cube in $\mathbb{R}^3$ and $\mathbf{x} = f(\mathbf{z}) = 2\mathbf{z}$.

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?

- 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

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

A **normalizing flow** is a change of variable f parameterized by an invertible neural network that transforms a base distribution $p(\mathbf{z})$ into $p(\mathbf{x})$. 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 $\mathbf{z}_K = \mathbf{x}$
- $p(\mathbf{z}_k) = p(\mathbf{z}_{k-1} = g_k(\mathbf{z}_k)) \left|\det J_{g_k}(\mathbf{z}_k)\right|$

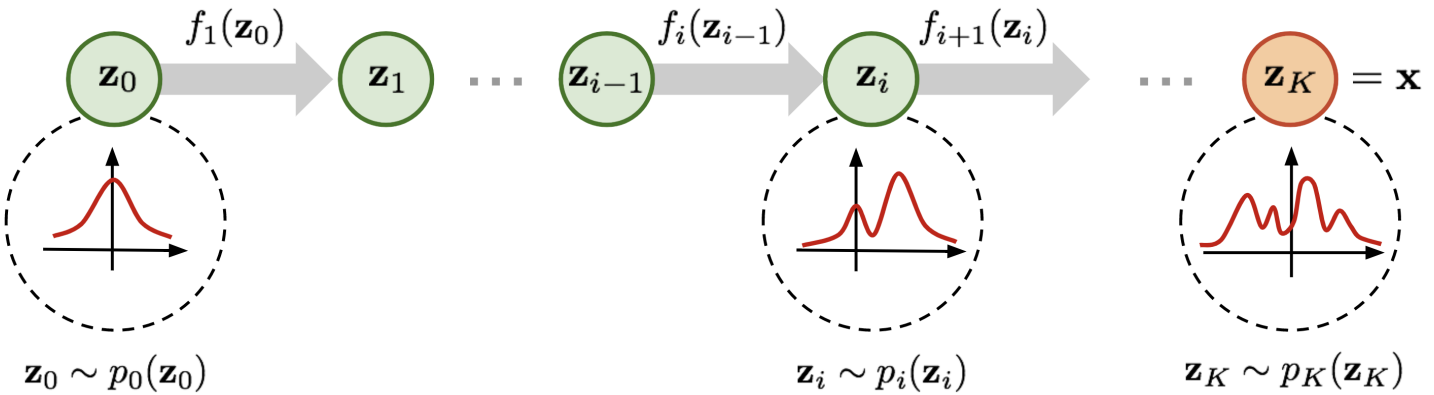
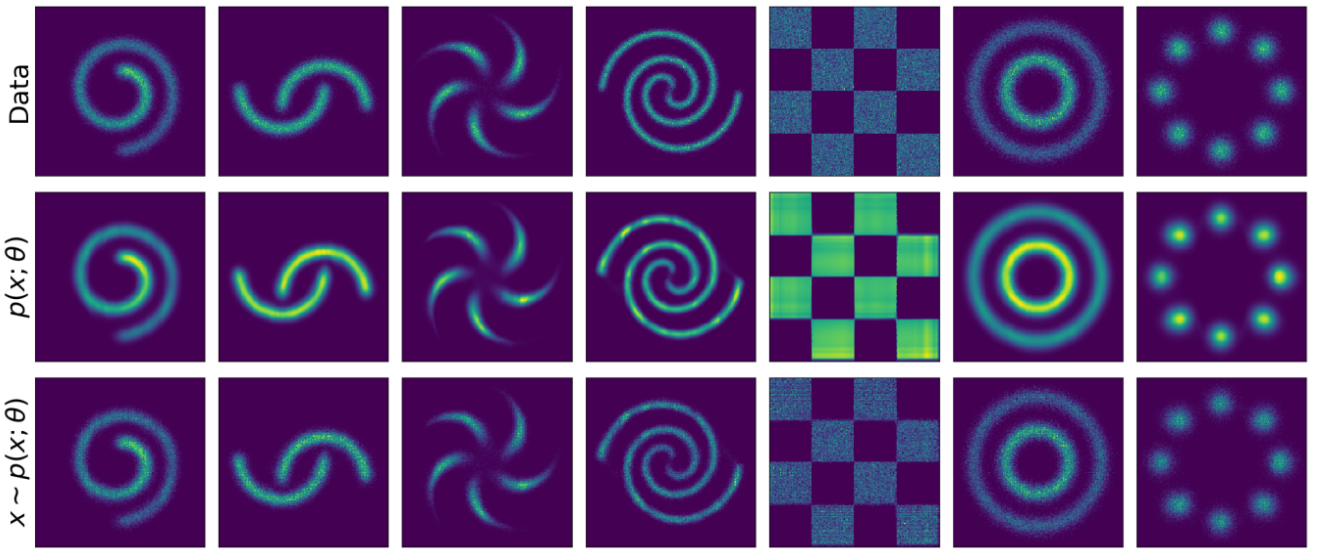


Image credits: Lilian Weng, 2018.

Example



Normalizing flows shine in applications for **density estimation**, where access to the density function is required (Wehenkel and Louppe, 2019).

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 (why?).
- It is also often referred to as model uncertainty.

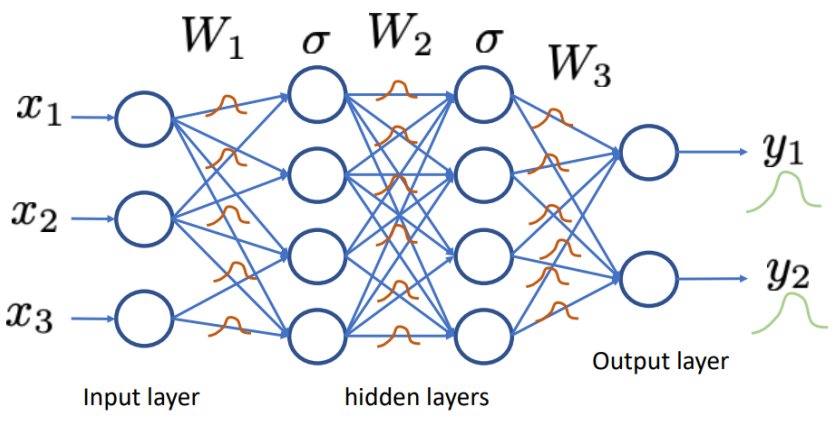
—

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.

- This stems mainly from the fact that the marginal $p(\mathbf{d})$ 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 $p(\omega|\mathbf{d})$.

As before (see Lecture 10), 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

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

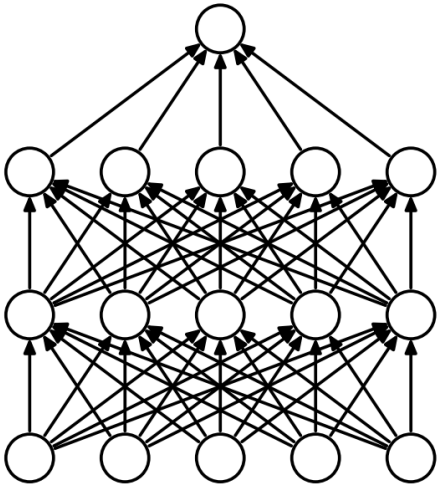
In the context of Bayesian neural networks, this procedure is also known as Bayes by backprop (Blundell et al, 2015).

Dropout

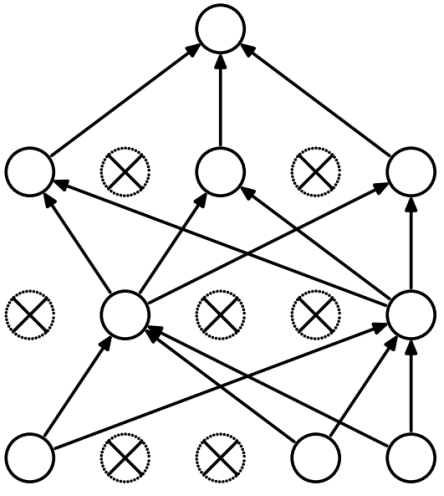
Dropout is an empirical technique that was first proposed to avoid overfitting in neural networks.

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.



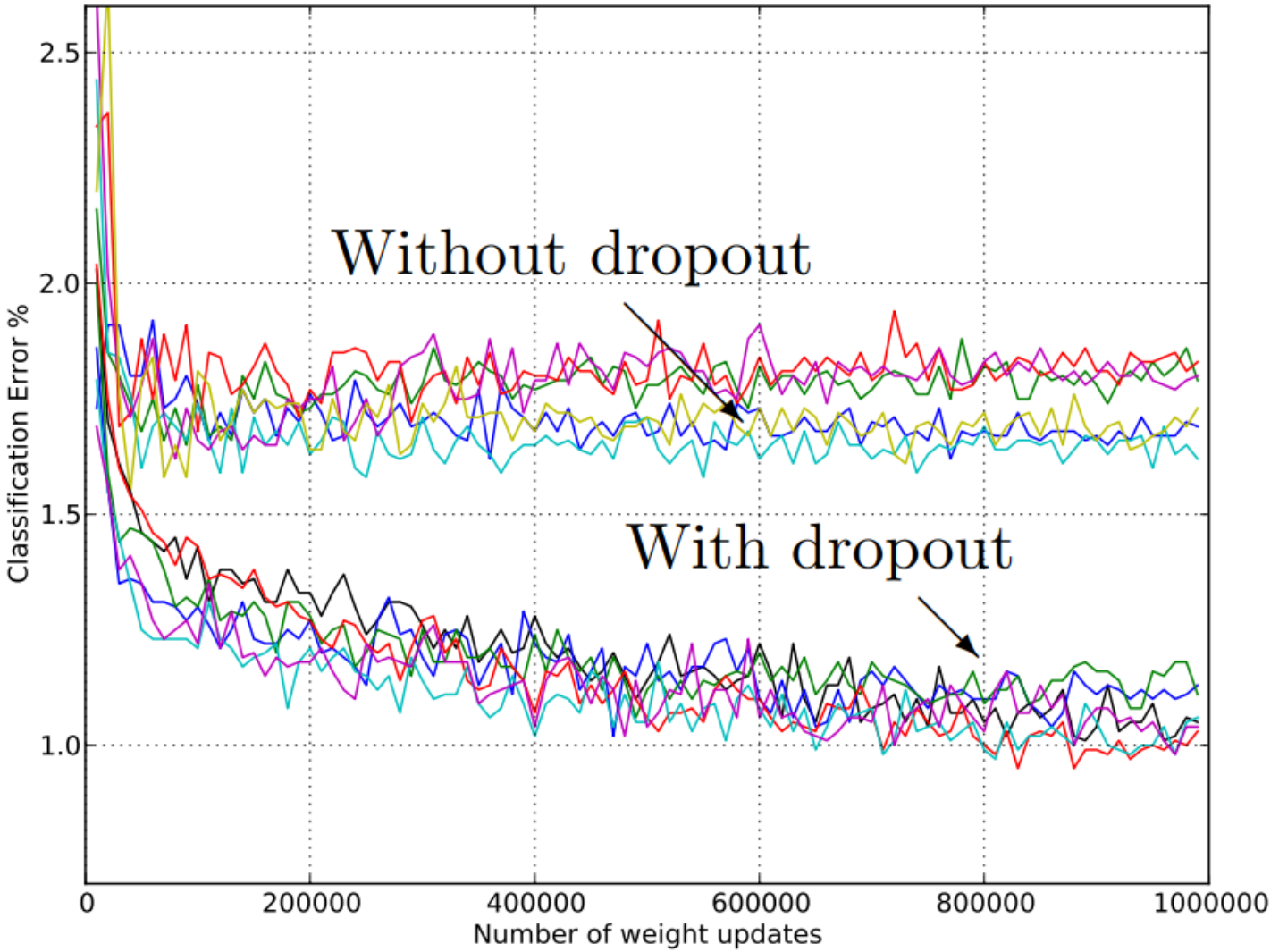
(a) Standard Neural Net



(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 %
Number of weight updates
2.5
2.0
1.5
1.0
0
200000
400000
600000
800000
1000000

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:

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

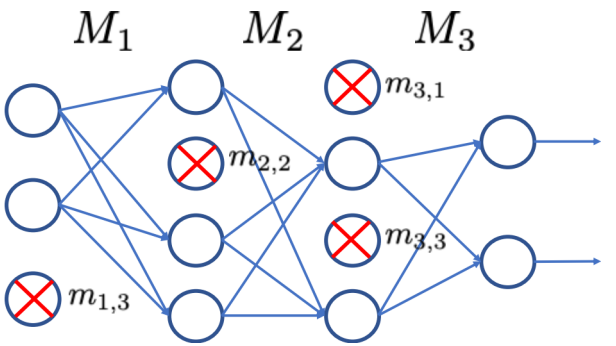
where $\delta_a(x)$ denotes a (multivariate) Dirac distribution centered at a.

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.

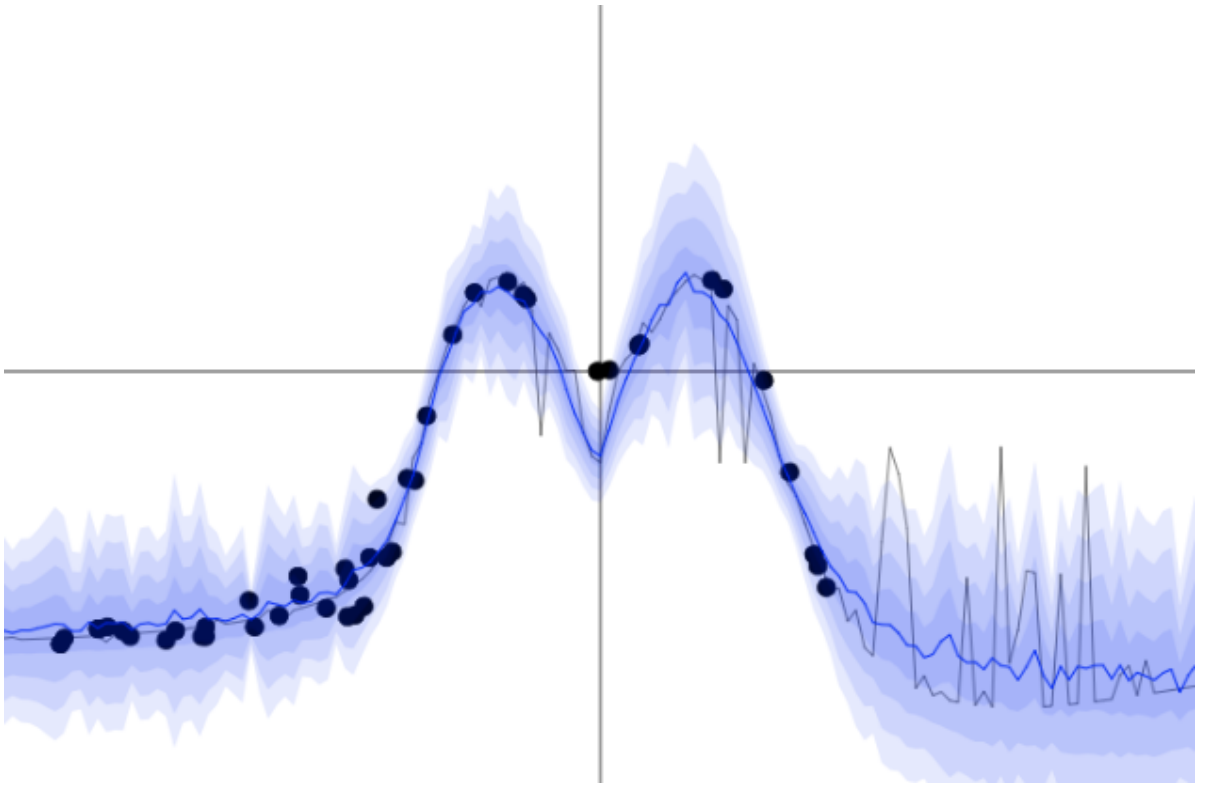
Uncertainty estimates from dropout

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:

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

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



Yarin Gal's demo.

Pixel-wise depth regression

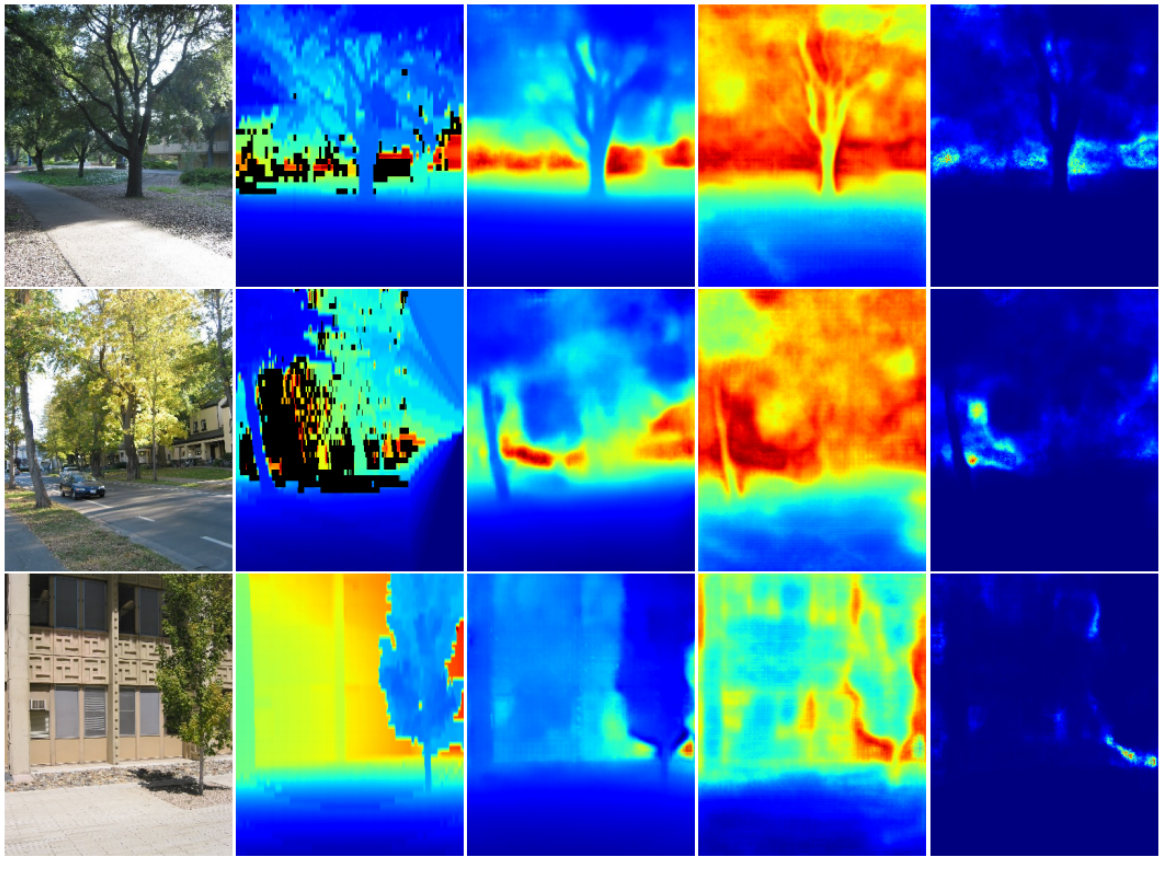


Figure 6: Qualitative results on the Make3D depth regression dataset. Left to right: input image, ground truth, depth prediction, aleatoric uncertainty, epistemic uncertainty. Make3D does not provide labels for depth greater than 70m, therefore these distances dominate the epistemic uncertainty signal.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

Bayesian Infinite Networks

Consider the 1-layer MLP with a hidden layer of size q and a bounded activation function σ:

$$f(x) = b + \sum_{j=1}^{q} v_j h_j(x)$$

$$h_j(x) = \sigma\left(a_j + \sum_{i=1}^{p} u_{i,j} x_i\right)$$

Assume Gaussian priors $v_j \sim \mathcal{N}(0, \sigma_v^2), b \sim \mathcal{N}(0, \sigma_b^2), u_{i,j} \sim \mathcal{N}(0, \sigma_u^2)$ and $a_j \sim \mathcal{N}(0, \sigma_a^2)$.

For a fixed value $x^{(1)}$, let us consider the prior distribution of $f(x^{(1)})$ implied by the prior distributions for the weights and biases.

We have

$$\mathbb{E}[v_j h_j(x^{(1)})] = \mathbb{E}[v_j]\mathbb{E}[h_j(x^{(1)})] = 0,$$

since v_j and $h_j(x^{(1)})$ are statistically independent and v_j has zero mean by hypothesis.

The variance of the contribution of each hidden unit h_j is

$$\begin{aligned}\mathbb{V}[v_j h_j(x^{(1)})] &= \mathbb{E}[(v_j h_j(x^{(1)}))^2] - \mathbb{E}[v_j h_j(x^{(1)})]^2 \\ &= \mathbb{E}[v_j^2]\mathbb{E}[h_j(x^{(1)})^2] \\ &= \sigma_v^2 \mathbb{E}[h_j(x^{(1)})^2],\end{aligned}$$

which must be finite since h_j is bounded by its activation function.

We define $V(x^{(1)}) = \mathbb{E}[h_j(x^{(1)})^2]$, and is the same for all j.

What if $q \to \infty$?

By the Central Limit Theorem, as $q \to \infty$, the total contribution of the hidden units, $\sum_{j=1}^{q} v_j h_j(x)$, to the value of $f(x^{(1)})$ becomes a Gaussian with variance $q\sigma_v^2 V(x^{(1)})$.

The bias b is also Gaussian, of variance σ_b^2, so for large q, the prior distribution $f(x^{(1)})$ is a Gaussian of variance $\sigma_b^2 + q\sigma_v^2 V(x^{(1)})$.

Accordingly, for $\sigma_v = \omega_v q^{-\frac{1}{2}}$, for some fixed ω_v, the prior $f(x^{(1)})$ converges to a Gaussian of mean zero and variance $\sigma_b^2 + \omega_v^2 \sigma_v^2 V(x^{(1)})$ as $q \to \infty$.

For two or more fixed values $x^{(1)}, x^{(2)}, \ldots$, a similar argument shows that, as $q \to \infty$, the joint distribution of the outputs converges to a multivariate Gaussian with means of zero and covariances of

$$\begin{aligned}\mathbb{E}[f(x^{(1)})f(x^{(2)})] &= \sigma_b^2 + \sum_{j=1}^{q} \sigma_v^2 \mathbb{E}[h_j(x^{(1)})h_j(x^{(2)})] \\ &= \sigma_b^2 + \omega_v^2 C(x^{(1)}, x^{(2)})\end{aligned}$$

where $C(x^{(1)}, x^{(2)}) = \mathbb{E}[h_j(x^{(1)})h_j(x^{(2)})]$ and is the same for all j.

This result states that for any set of fixed points $x^{(1)}, x^{(2)}, \ldots$, the joint distribution of $f(x^{(1)}), f(x^{(2)}), \ldots$ is a multivariate Gaussian.

In other words, the infinitely wide 1-layer MLP converges towards a Gaussian process.

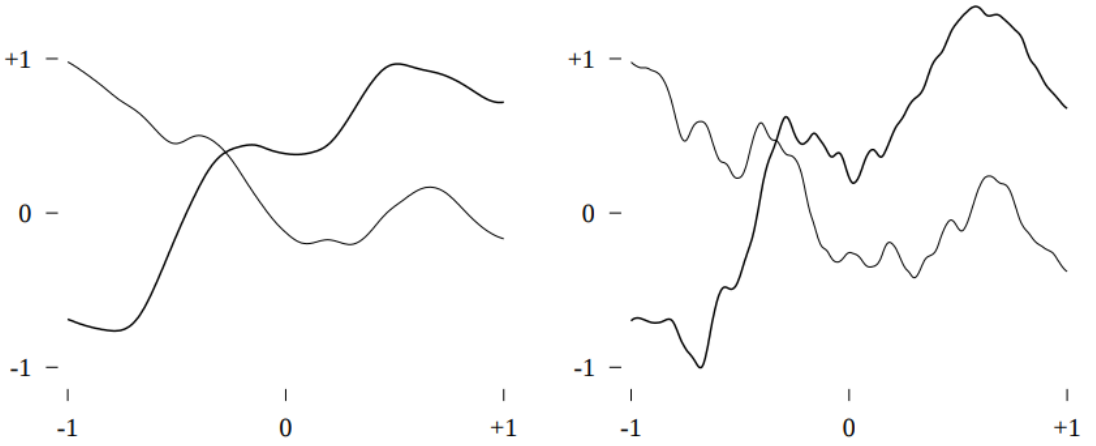


Figure 2.2: Functions drawn from Gaussian priors for a network with 10 000 tanh hidden units. Two functions drawn from a prior with $\sigma_u = 5$ are shown on the left, two from a prior with $\sigma_u = 20$ on the right. In both cases, $\sigma_a/\sigma_u = 1$ and $\sigma_b = \omega_v = 1$. The functions with different σ_u were generated using the same random number seed, the same as that used to generate the functions in the lower-right of Figure 2.1. This allows a direct evaluation of the effect of changing σ_u. (Use of a step function is equivalent to letting σ_u go to infinity, while keeping σ_a/σ_u fixed.)

(Neal, 1995)

Adversarial attacks

Intriguing properties of neural networks

"We can cause the network to misclassify an image by applying a certain hardly perceptible perturbation, which is found by maximizing the network's prediction error. In addition, the specific nature of these perturbations is not a random artifact of learning: the same perturbation can cause a different network, that was trained on a different subset of the dataset, to misclassify the same input."

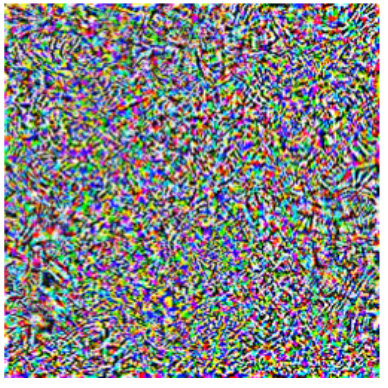
The existence of the adversarial negatives appears to be in contradiction with the network's ability to achieve high generalization performance. Indeed, if the network can generalize well, how can it be confused by these adversarial negatives, which are indistinguishable from the regular examples?"

(Szegedy et al, 2013)

Attacks

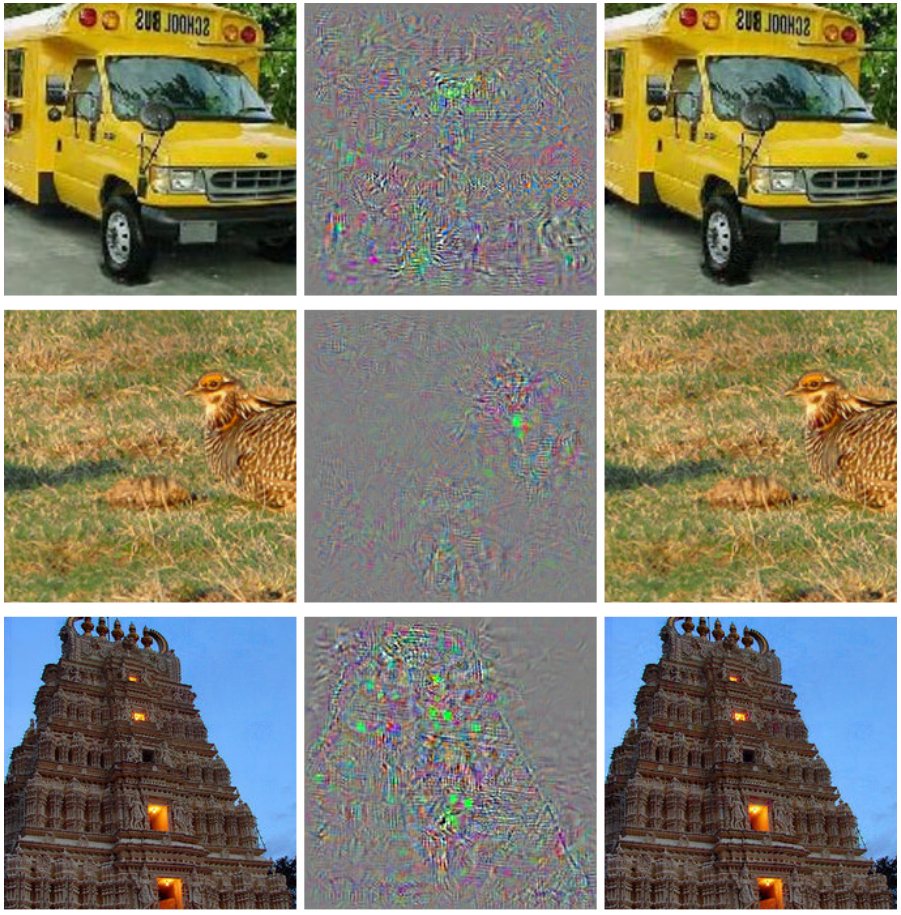
“pig”

+ 0.005 x



=

“airliner”



(Left) Original images. (Middle) Adversarial noise. (Right) Modified images. All are classified as 'Ostrich'.

—

Credits: Szegedy et al, Intriguing properties of neural networks, 2013.

Fooling deep structured prediction models

Figure 1: We cause the network to generate a *minion* as segmentation for the adversarially perturbed version of the original image. Note that the original and the perturbed image are indistinguishable.

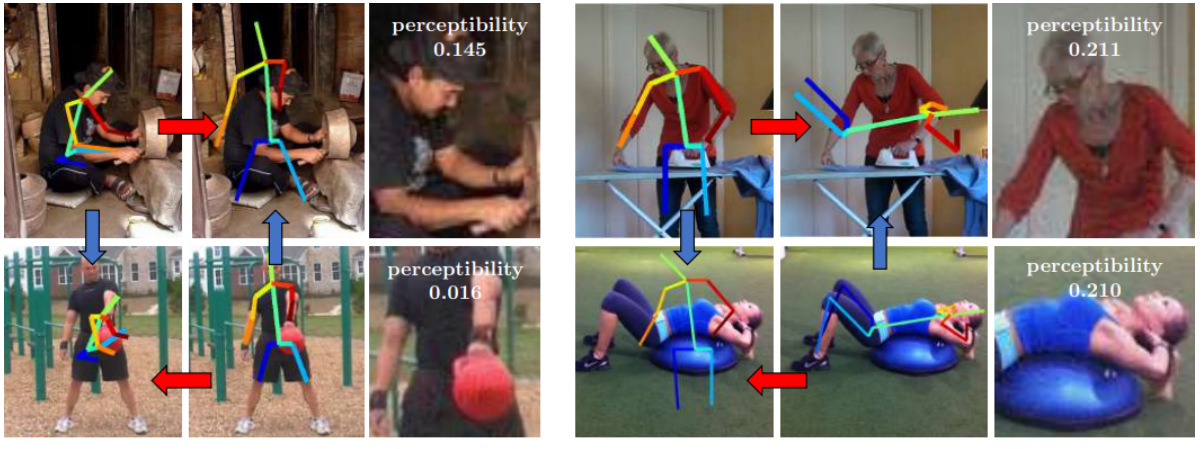
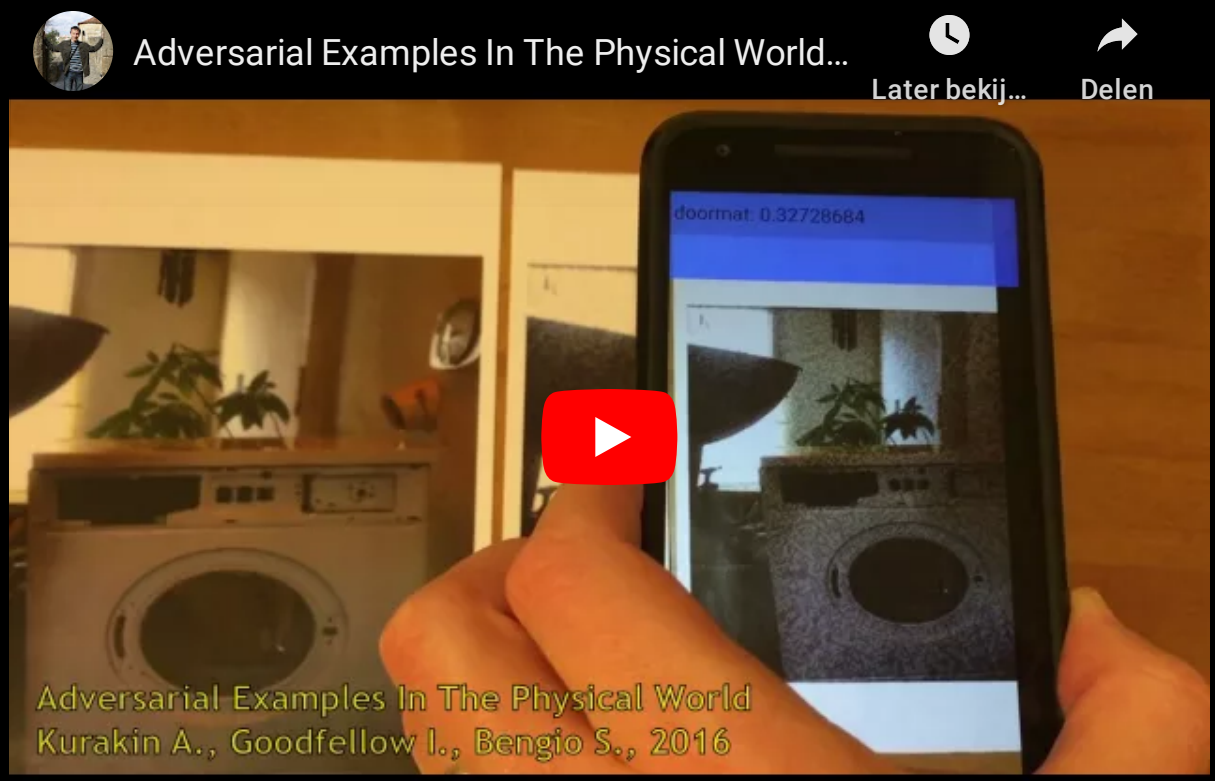
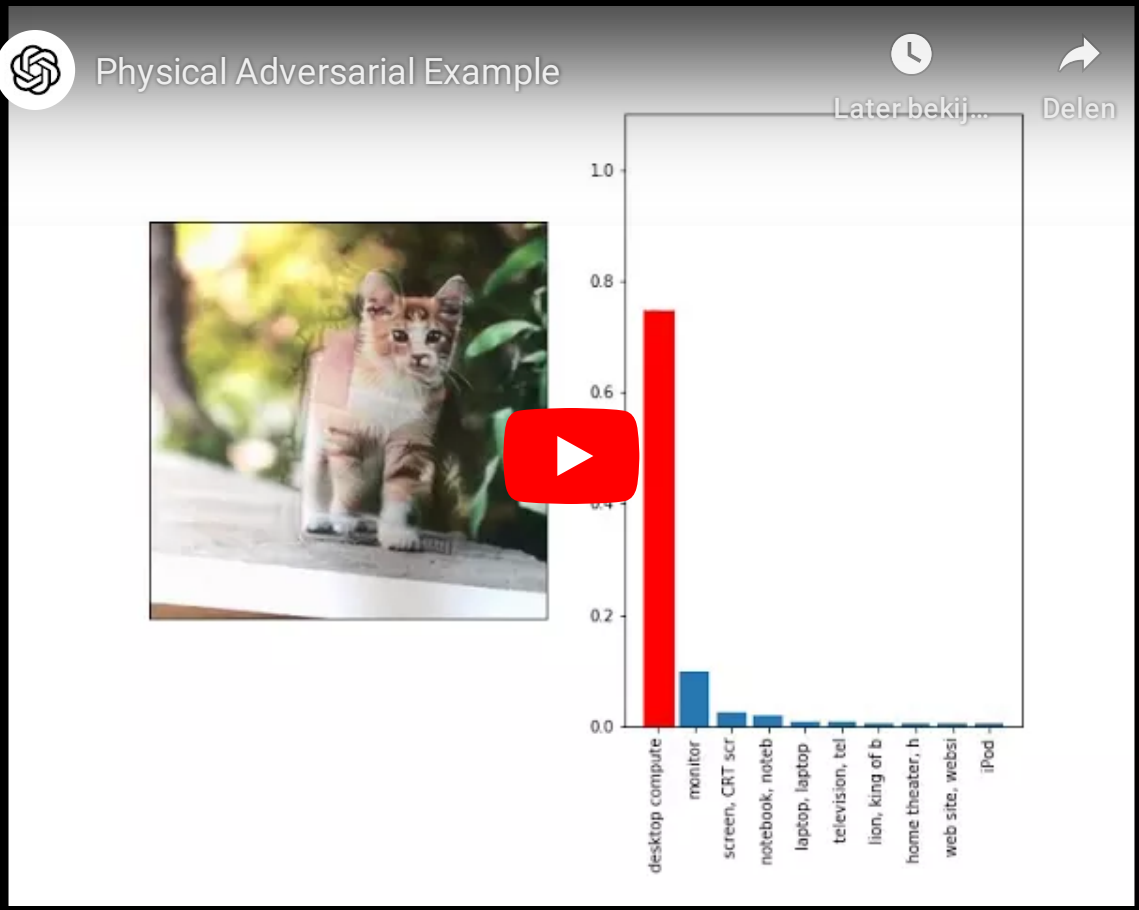


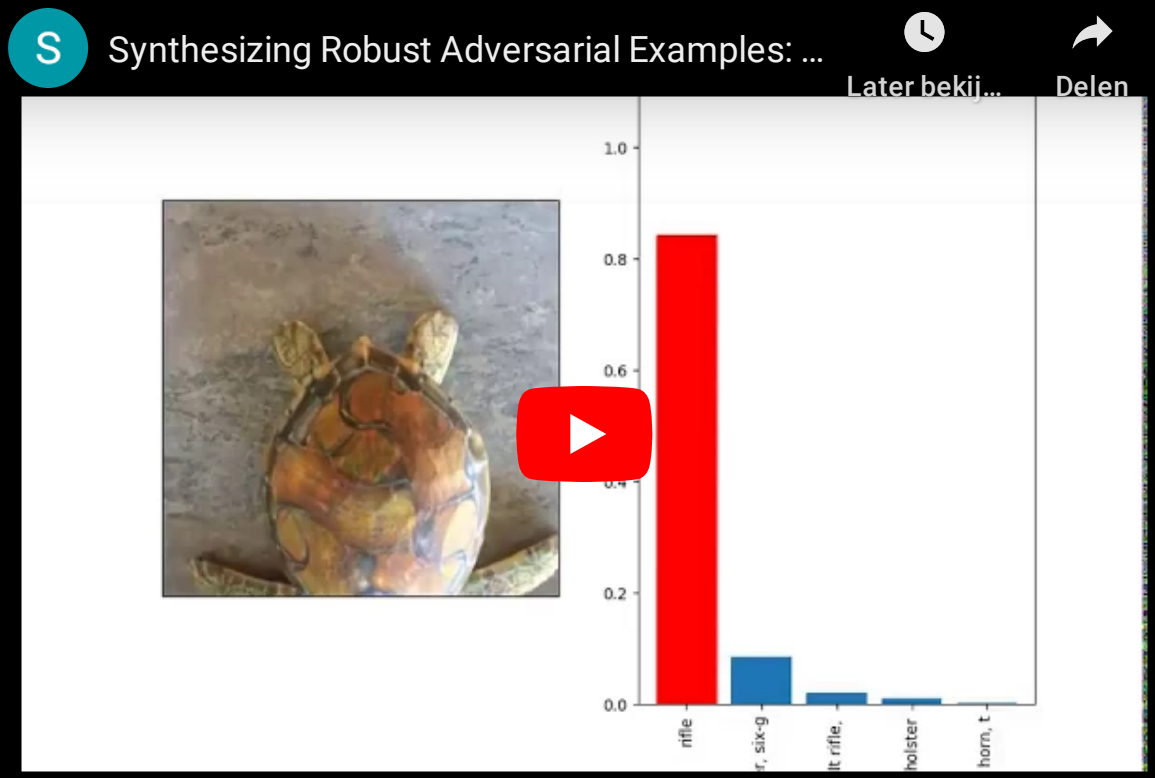
Figure 4: Examples of successful targetted attacks on a pose estimation system. Despite the important difference between the images selected, it is possible to make the network predict the wrong pose by adding an imperceptible perturbation.

(Cisse et al, 2017)

Adversarial examples in the physical world




Physical Adversarial Example
Later bekij...
Delen
1.0
0.8
0.6
0.2
0.0
desktop compute
monitor
screen, CRT scr
notebook, noteb
laptop, laptop
television, tel
lion, king of b
home theater, h
web site, websi
iPod


S
Synthesizing Robust Adversarial Examples: ...
Later bekij...
Delen
1.0
0.8
0.6
0.2
0.0
rifle
holster
horn, t

Creating adversarial examples

"The deep stack of non-linear layers are a way for the model to encode a non-local generalization prior over the input space. In other words, it is assumed that is possible for the output unit to assign probabilities to regions of the input space that contain no training examples in their vicinity.

It is implicit in such arguments that local generalization—in the very proximity of the training examples—works as expected. And that in particular, for a small enough radius $\epsilon > 0$ in the vicinity of a given training input $\mathbf{x}$, an $\mathbf{x} + \mathbf{r}$ satisfying $||\mathbf{r}|| < \epsilon$ will get assigned a high probability of the correct class by the model."

(Szegedy et al, 2013)

$$\min_{\mathbf{r}} \ell(y_{\text{target}}, f(\mathbf{x} + \mathbf{r}; \theta))$$
$$\text{subject to } ||\mathbf{r}|| \leq L$$

A security threat

Adversarial attacks pose a security threat to machine learning systems deployed in the real world.

Examples include:

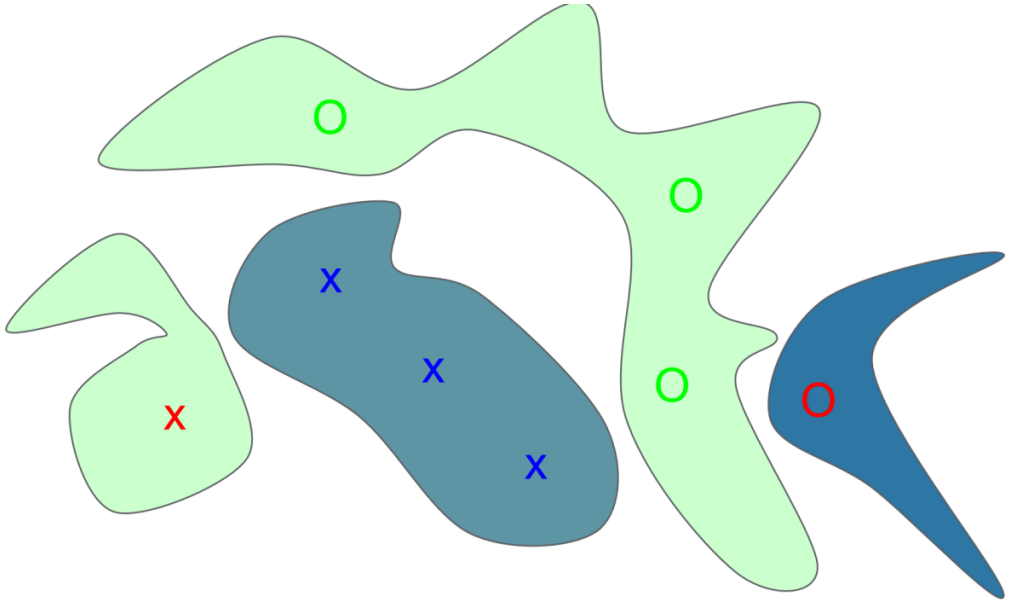
- fooling real classifiers trained by remotely hosted API (e.g., Google),
- fooling malware detector networks,
- obfuscating speech data,
- displaying adversarial examples in the physical world and fool systems that perceive them through a camera.

What if adversarial patches are put on road signs?
Say, for a self-driving car?

Origins of the vulnerability

Conjecture 1: Overfitting

Natural images are within the correct regions, but are also sufficiently close to the decision boundary.



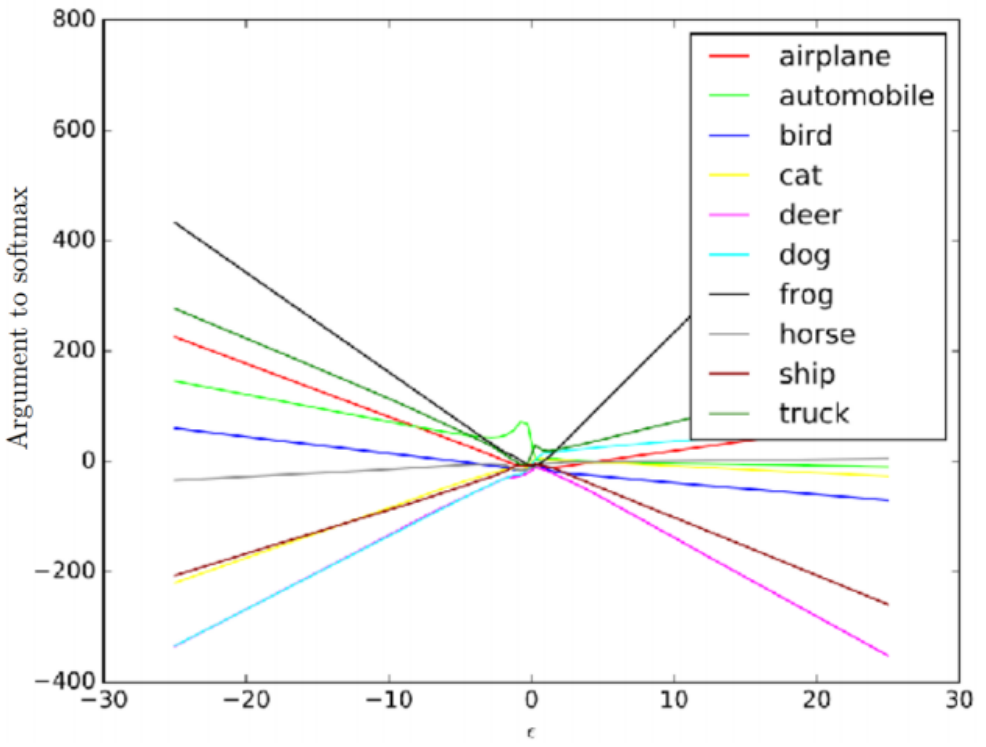
Conjecture 2: Excessive linearity

The decision boundary for most ML models, including neural networks, are near piecewise linear.

Then, for an adversarial sample $\hat{\mathbf{x}}$, its dot product with a weight vector $\mathbf{w}$ is such that

$$\mathbf{w}^T \hat{\mathbf{x}} = \mathbf{w}^T \mathbf{x} + \mathbf{w}^T \mathbf{r}.$$

- The adversarial perturbation causes the activation to grow by $\mathbf{w}^T \mathbf{r}$.
- For $\mathbf{r} = \epsilon \text{sign}(\mathbf{w})$, if $\mathbf{w}$ has n dimensions and the average magnitude of an element is m, then the activation will grow by $\epsilon m n$.
- Therefore, for high dimensional problems, we can make many infinitesimal changes to the input that add up to one large change to the output.



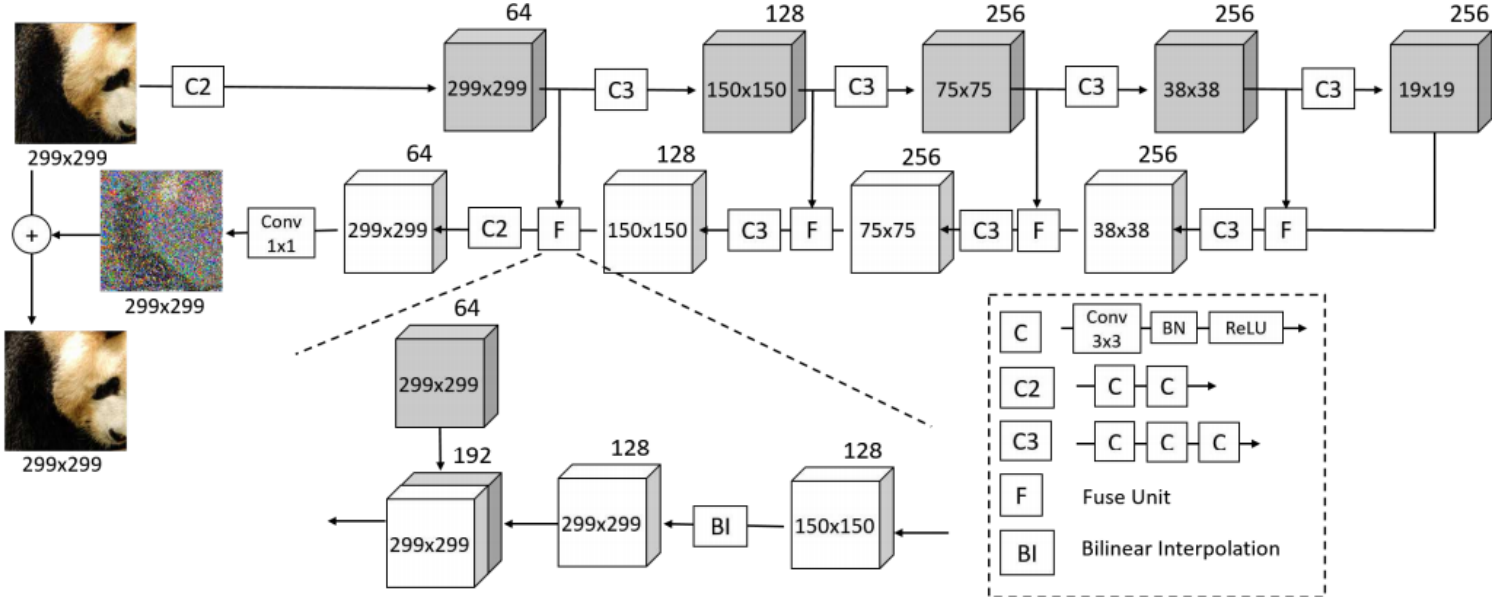
Empirical observation: neural networks produce nearly linear responses over ϵ.

Defenses

- Data augmentation
- Adversarial training
- Denoising / smoothing

Denoising

- Train the network to remove adversarial perturbations before using the input.
- The winning team of the defense track of the NIPS 2017 competition trained a denoising U-Net to remove adversarial noise.



Credits: Liao et al, Defense against Adversarial Attacks Using High-Level Representation Guided Denoiser, 2017.

Failed defenses

"In this paper we evaluate ten proposed defenses and demonstrate that none of them are able to withstand a white-box attack*. We do this by constructing defense-specific loss functions that we minimize with a strong iterative attack algorithm. With these attacks, on CIFAR an adversary can create imperceptible adversarial examples for each defense.*

By studying these ten defenses, we have drawn two lessons: existing defenses lack thorough security evaluations, and adversarial examples are much more difficult to detect than previously recognized."

(Carlini and Wagner, 2017)

"No method of defending against adversarial examples is yet completely satisfactory. This remains a rapidly evolving research area."

(Kurakin, Goodfellow and Bengio, 2018)

The end.

References

- Bishop, C. M. (1994). Mixture density networks (p. 7). Technical Report NCRG/4288, Aston University, Birmingham, UK.
- Kendall, A., & Gal, Y. (2017). What uncertainties do we need in bayesian deep learning for computer vision?. In Advances in neural information processing systems (pp. 5574-5584).
- Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), 1929-1958.
- Pierre Geurts, INFO8004 Advanced Machine Learning - Lecture 1, 2019.