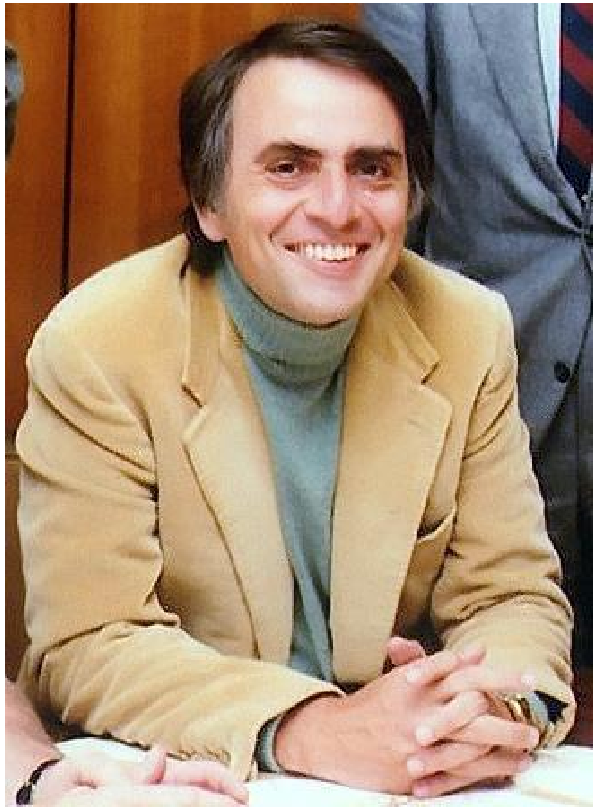
Lecture-6:
Uncertainty in Neural Networks

Motivation

SIEMENS
Ingenuity for life

“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**,
The Demon-Haunted World: Science as a Candle in the Dark



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

Further Reading
Sagan, Carl. The demon-haunted world: Science as a candle in the dark. Ballantine Books, 2011.

Types of Uncertainty

SIEMENS
Ingenuity for life

Classical Neural Network
(or any parametric model)

$$y = \mathrm{f}(x, \theta) \,;\, \mathrm{D} = \{\mathrm{x}, \mathrm{t}\}$$

Conventions for this lecture

- θ = neural network weights
- x = neural network input
- y = neural network output
- D = training data (input and targets)

Output/Prediciton Uncertainty

$$y \sim p(y|x, \theta)$$

Model (Weight) Uncertainty

$$\theta \sim p(\theta|\mathrm{D})$$

Three Different Ways to represent Distributions in Practice

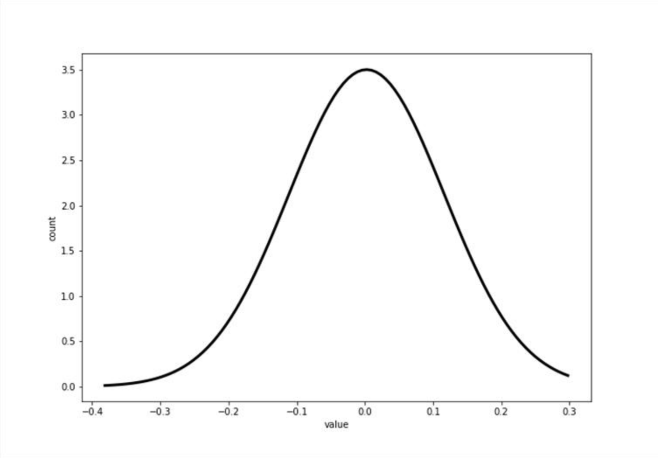
SIEMENS
Ingenuity for life

data

Point Estimate

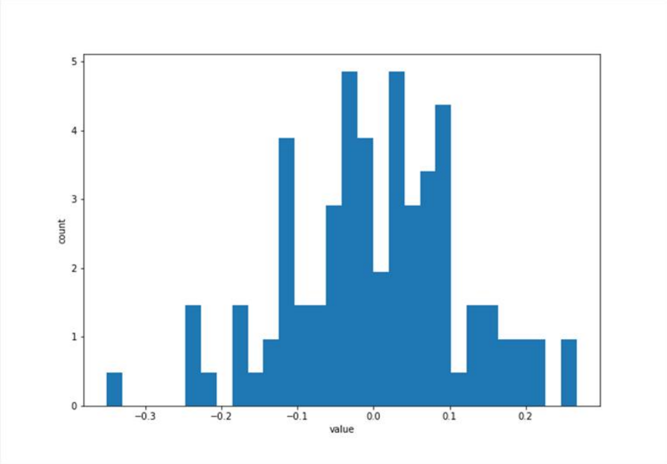
Mean

Parametric Distribution



Mean	Var.

Empirical Distribution



Bin1	Bin 2	Bin 3	…	Bin N

Pros and Cons

SIEMENS
Ingenuity for life

data

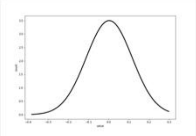
Point Estimate

Mean

þ Calculation efficient and stable

ý Uncertainty not accounted for
ý Cannot be sampled
ý Assumption that point estimate is representative: e.g. Failure with multi-modality

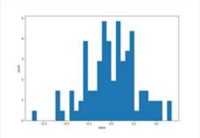
Parametric Distribution



þ Calculation relatively efficient
þ Analytical Representation
þ Can be sampled

ý Breaks down if assumption on class of distribution is not correct
ý Hard to represent multi-modality

Empirical Distribution



þ Flexible: No assumptions
þ Can be sampled (more or less)

ý Breaks down for higher dimensional data
ý High memory consumption
ý Artifacts due to discretization

Output Uncertainty for Regression

Recap: Classification

SIEMENS
Ingenuity for life

The standard setup:

- Classification predicts a categorical distribution of the classes
- Targets are one-hot-encoded
- The loss function is the cross-entropy

X

Neural Network

Dense Layer
Act:Softmax

Categorial

0	1	2	3	4	…	9
0	0	.98	0	0		.02

0	1	2	3	4	…	9
0	0	1	0	0		0

target

Understates the true uncertainty!

We will focus on Regression henceforth …

Motivation for Output Uncertainty: Mean Average Precision @ 7

SIEMENS
Ingenuity for life

Challenge:
Give at most 7 recommendations from a large product catalog. You are evaluated based on the precision on these at most 7 items.

Idea:
Select at most 7 recommendations where the model is certain about the recommendation.

In General:
Situations where a response/action can be rejected.

Further Reading
Brando, Axel, et al. "Uncertainty Modelling in Deep Networks: Forecasting Short and Noisy Series." arXiv preprint arXiv:1807.09011 (2018).

Related Kaggle Challange
https://www.kaggle.com/c/santander-product-recommendation

Neural Network with Output Uncertainty

SIEMENS
Ingenuity for life

$$y \sim p(y|x, \theta)$$

Let's commit to a parametric distribution:

$$y \sim \mathcal{N}(y|\mu, \sigma)$$

Your choice!
Also popular: The Laplace Distribution

We will model μ as a Neural Network: $\mu(x, \theta)$

We either model σ as a scalar parameter under the assumption of homoskestic uncertainty or as a Neural Network: $\sigma(x, \theta)$ for heteroskedastic uncertainty

$$\mathcal{N}(y|\mu, \sigma) = \frac{1}{\sqrt{2\pi\sigma(x, \theta)^2}} \exp\left(-\frac{(y - \mu(x, \theta))^2}{2\sigma(x, \theta)^2}\right)$$

Neural Network with Output Uncertainty

SIEMENS
Ingenuity for life

$$\mathcal{N}(y|\mu,\sigma) = \frac{1}{\sqrt{2\pi\sigma(x,\theta)^2}} \exp\left(-\frac{(y-\mu(x,\theta))^2}{2\sigma(x,\theta)^2}\right)$$

$$D = \{x, t\}$$

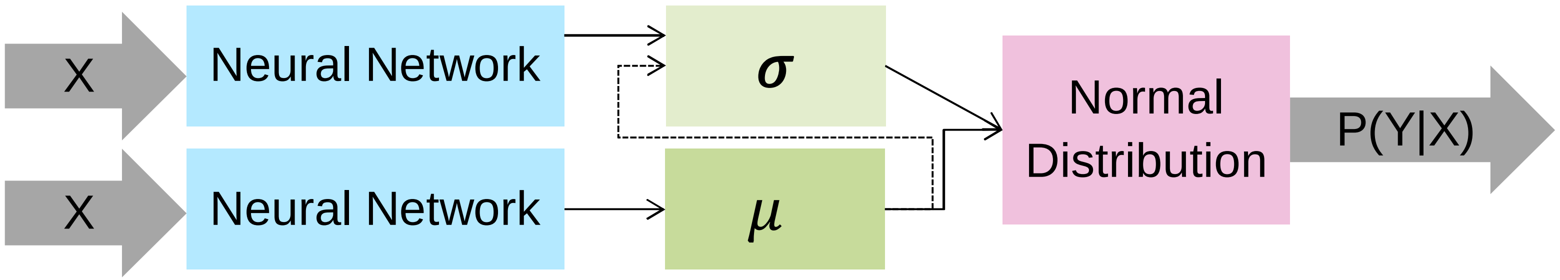
We will optimize the Log-Likelihood with respect to the weights θ

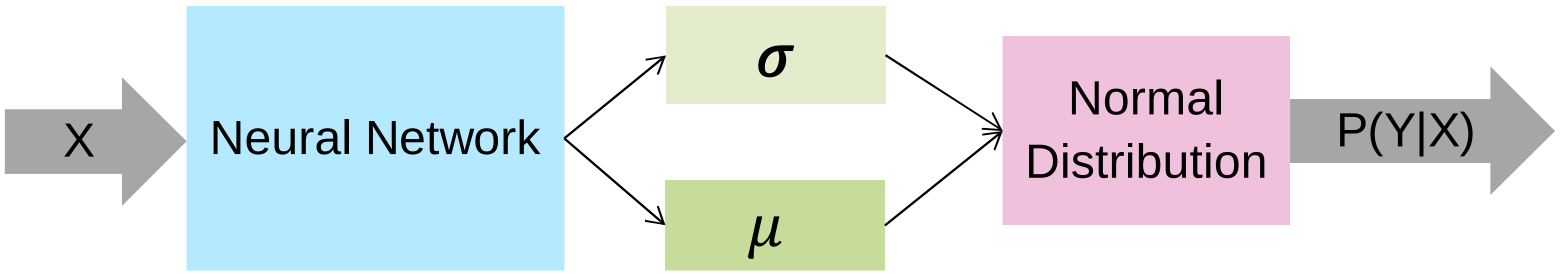
$$\mathcal{L}(\theta, D) = -\sum_{i=0}^{|D|}\left[-\log\big(\sigma(x_i,\theta)\big) - \frac{(t_i-\mu(x_i,\theta))^2}{2\sigma(x_i,\theta)^2} + C\right]$$

Use a variant of Stochastic Gradient Descent to train.

Network with Output Uncertainty: Architecture Variants

SIEMENS
Ingenuity for life



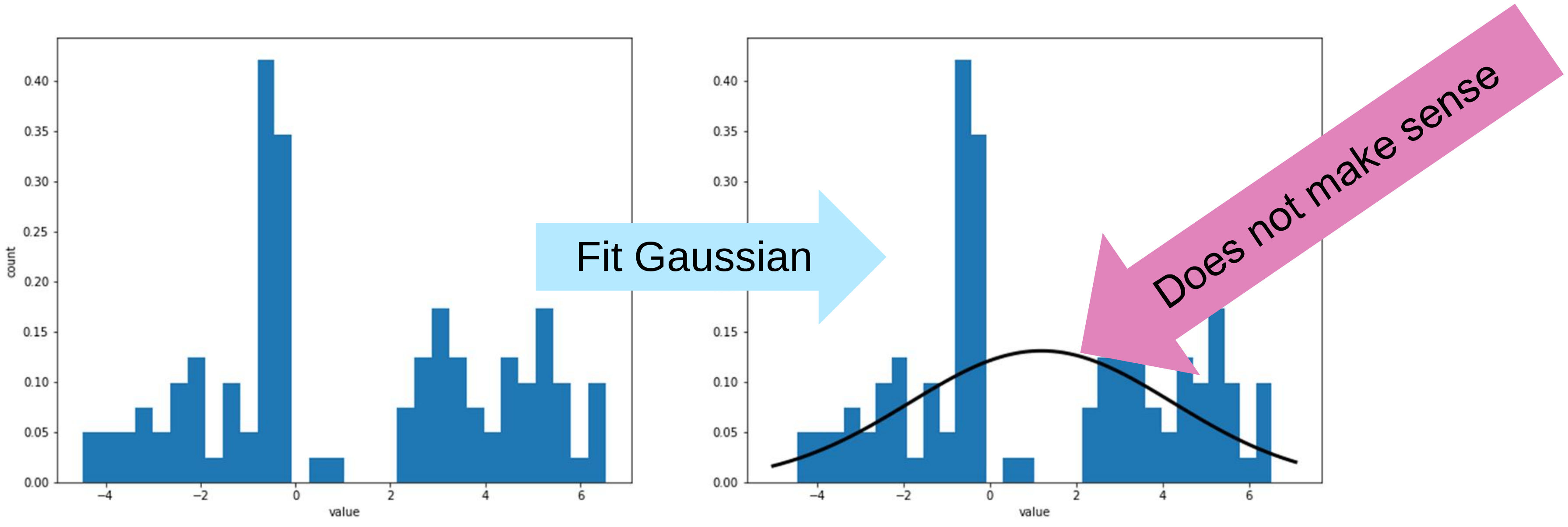


Understates the true uncertainty!

Loss: $-\log P(\mathbf{Y}|\mathbf{X})$

Multimodality

SIEMENS
Ingenuity for life

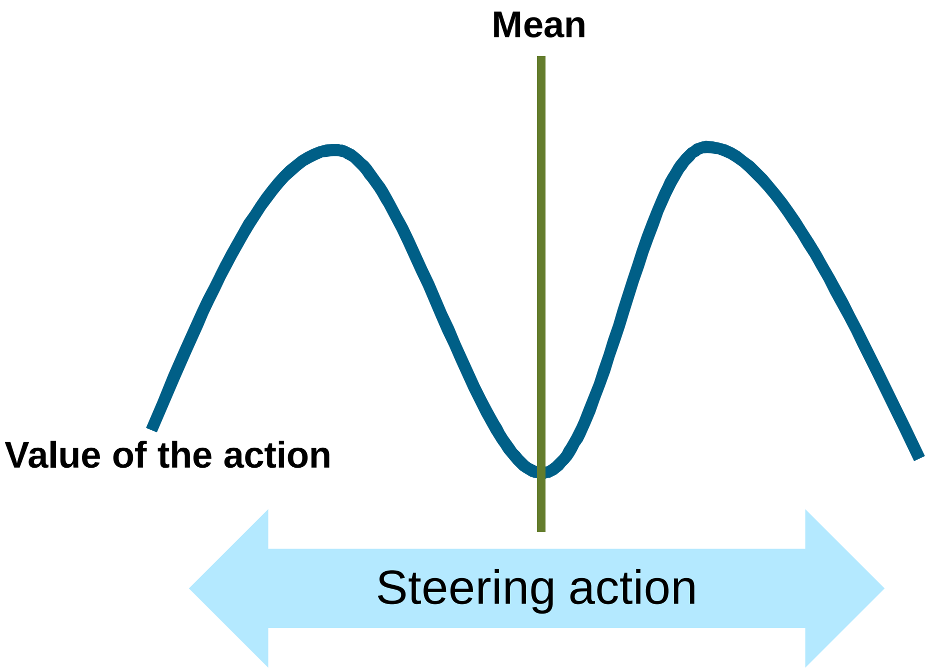


Example of Multimodality

Learning the value of an action
(see also Reinforcement Learning)

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

Further Reading
Depeweg, Stefan, et al. "Learning and policy search in stochastic dynamical systems with bayesian neural networks." arXiv preprint arXiv:1605.07127 (2016).



Result of Point Estimation

https://commons.wikimedia.org/wiki/File:Bus_in_hole.jpg

Inverse Problems

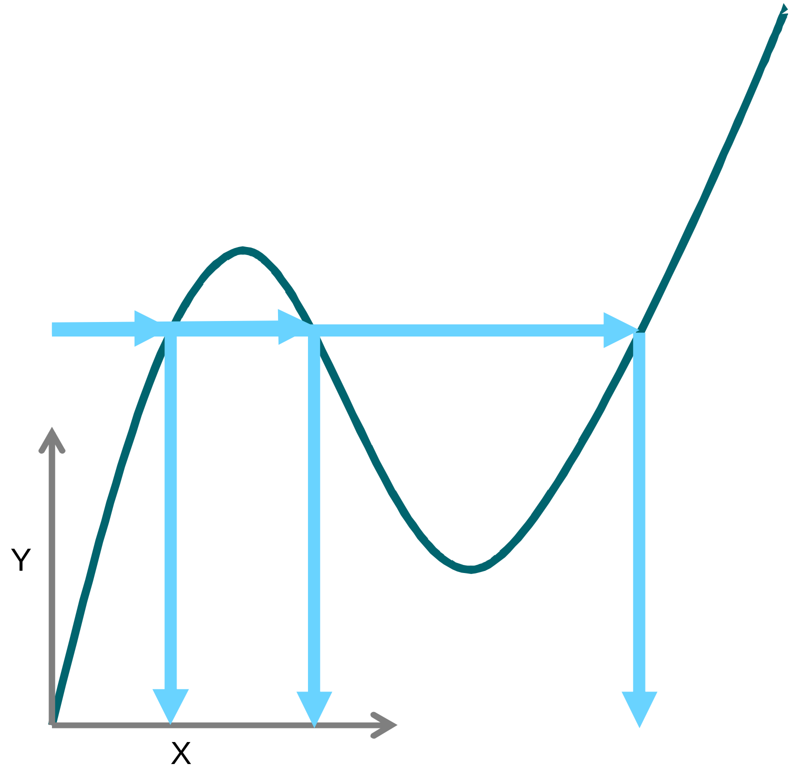
1. We are interested in X, but X can only be perceived indirectly via Y.
2. The relation between X and Y is given by a known, possibly stochastic function

$$y = g(x)$$

This means we would like to learn the inverse of g:

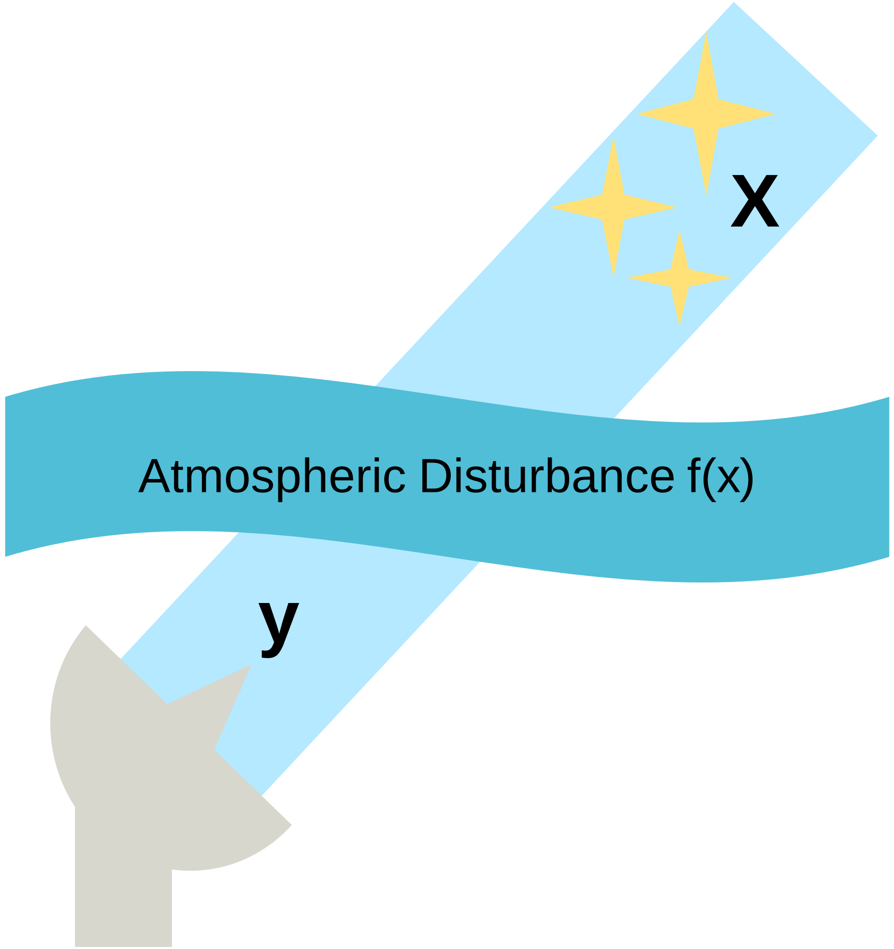
$$x = f(y)$$

The challenge is that g may not be invertible!
This means that f may be stochastic even if g is not.



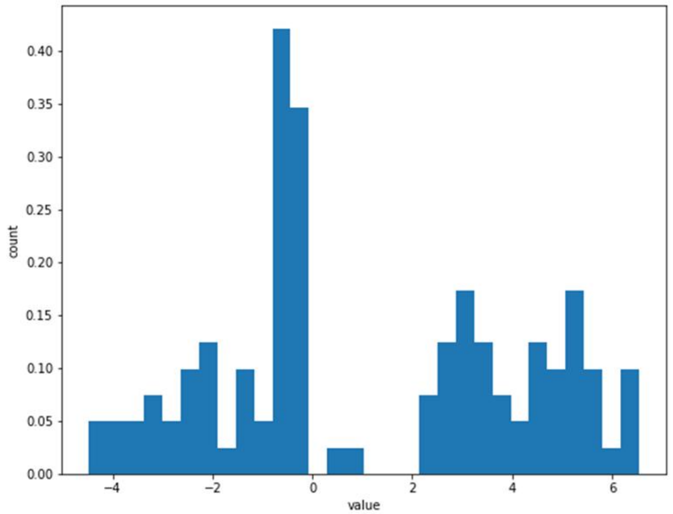
Inverse Problems: Examples

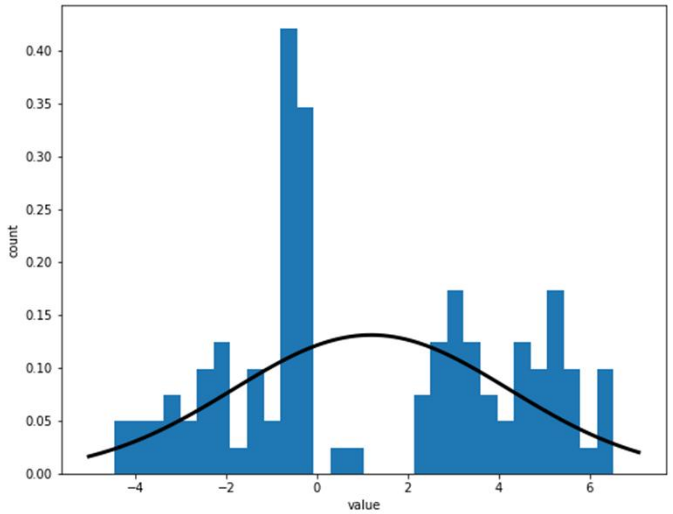
1. Image De-Noising
2. Astronomy Imaging
3. Deconvolution
4. Localization
5. …



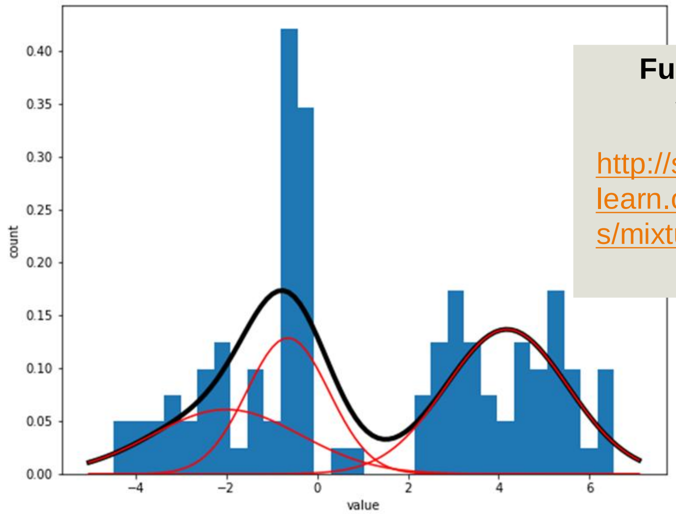
Gaussian Mixtures

SIEMENS
Ingenuity for life





Mean	Variance
1.2	3



	Mean	Var.	weight
Cluster 1	4.1	1.8	0.6
Cluster 2	-2.0	2.7	0.38
Cluster 3	-0.6	0.8	0.02

Further Practice with GMMs

http://scikit-learn.org/stable/modules/mixture.html#bgmm

If you want to know how many clusters to assume:

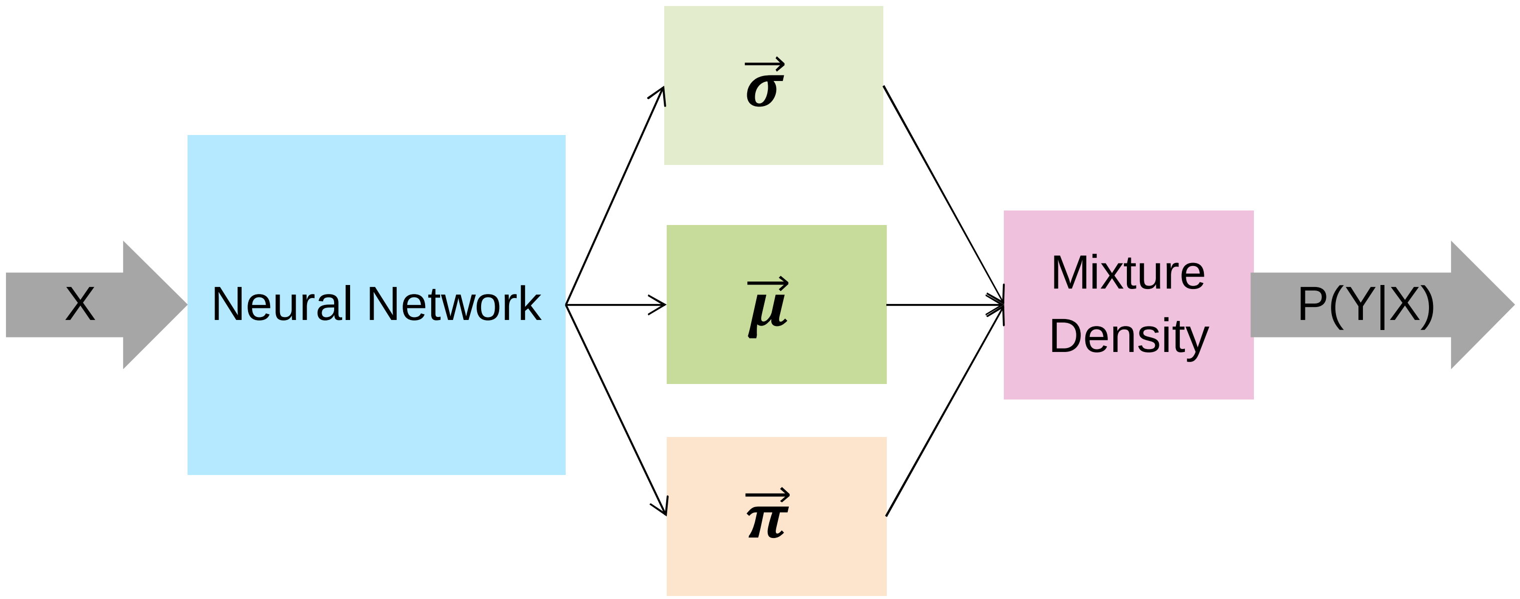
Further Reading

Rasmussen, Carl Edward. "The infinite Gaussian mixture model." Advances in neural information processing systems. 2000.

Further Reading

Blei, David M., and Michael I. Jordan. "Variational inference for Dirichlet process mixtures." Bayesian analysis 1.1 (2006): 121-143.

The Mixture Density Network: Architecture



- $\vec{\boldsymbol{\sigma}}$: standard deviations
- $\vec{\boldsymbol{\mu}}$: means
- $\vec{\boldsymbol{\pi}}$: cluster weights

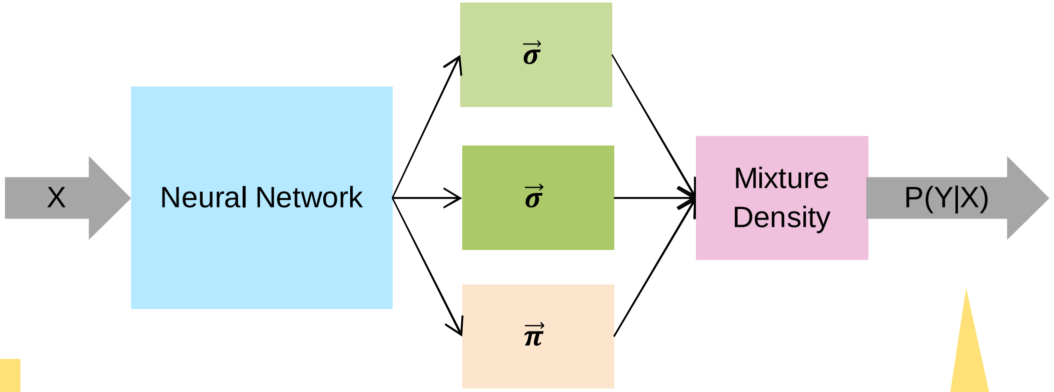
- **Loss**: $-\log P(\mathbf{Y}|\mathbf{X})$

Reference
Bishop, Christopher M. *Mixture density networks*. Technical Report NCRG/4288, Aston University, Birmingham, UK, 1994.

The Mixture Density Network: Implementation in TensorFlow Probability

```
import tensorflow as tf
import tensorflow_probability as tfp
tfd = tfp.distributions

X_ph = tf.placeholder(tf.float32, [None, D])
y_ph = tf.placeholder(tf.float32, [None])

K = 10
net = tf.layers.dense(X_ph, 15, activation=tf.nn.relu)
net = tf.layers.dense(net, 15, activation=tf.nn.relu)
locs = tf.layers.dense(net, K, activation=None)
scales = tf.layers.dense(net, K, activation=tf.exp)
logits = tf.layers.dense(net, K, activation=None)

components = [ tfd.Normal(loc=loc, scale=scale) for loc, scale
                in zip(tf.unstack(tf.transpose(locs)), tf.unstack(tf.transpose(scales)))]

cat = tfd.Categorical(logits=logits)

mdn = tfd.Mixture(cat=cat, components=components)

loss = -tf.reduce_mean(mdn.log_prob(y_ph))
```

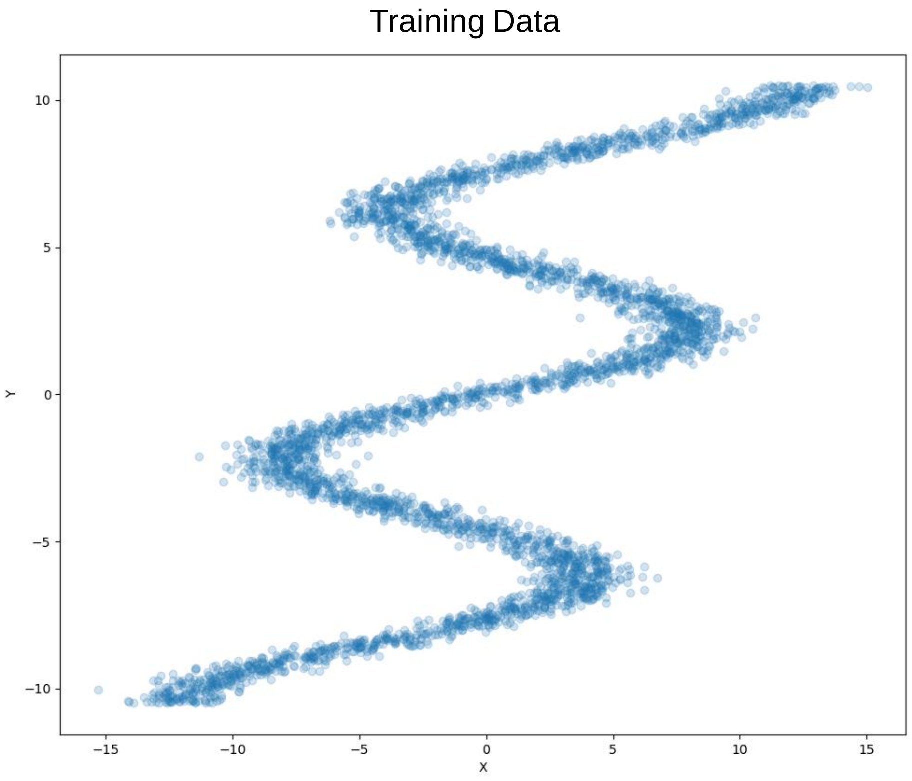
Note the activation! We want $\vec{\sigma} \epsilon (0, \infty)$



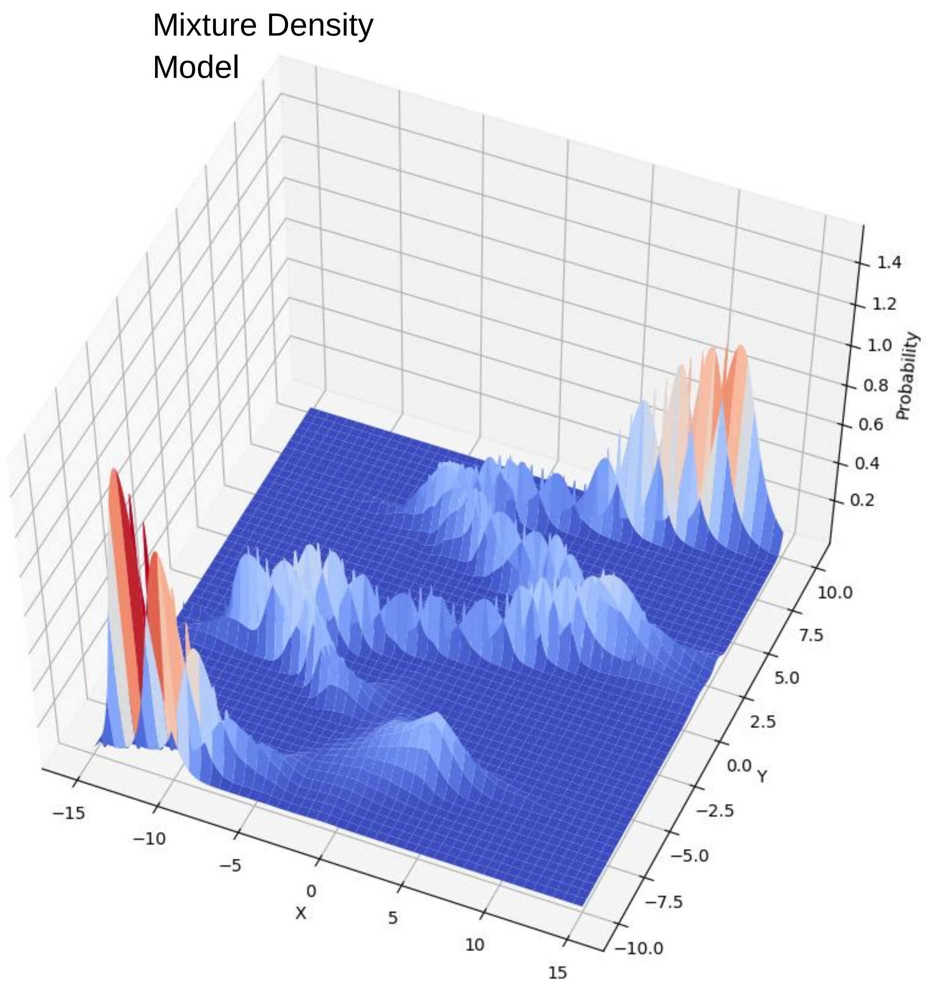
Understates the true uncertainty!

The Mixture Density Network: Result

SIEMENS
Ingenuity for life



MDN Training



Model Uncertainty

Model Uncertainty

$$y = \mathrm{f}(x, \theta)\,; \mathrm{D} = \{\mathrm{x}, \mathrm{t}\}$$

Model (Weight) Uncertainty

$$\theta \sim p(\theta|\mathrm{D})$$

Implies an uncertainty in the output!

Is it qualitatively different than simple output uncertainty?

Example: Tossing a Coin

You have tossed a coin 1 000 000 times.

Head	Tails
501 071	498 929

What is your prediction for the outcome of the next flip?

How sure are you? In other words: How sure are you about your model?

Example: Predicting a Time Series

SIEMENS
Ingenuity for life

Data

t	0	1	2	3	4	5	6
y	0	1	?	?	?	?	?

What is your prediction for y?

How sure are you? In other words: How sure are you about your model?

Epistemic and Aleatoric Uncertainty

Epistimic

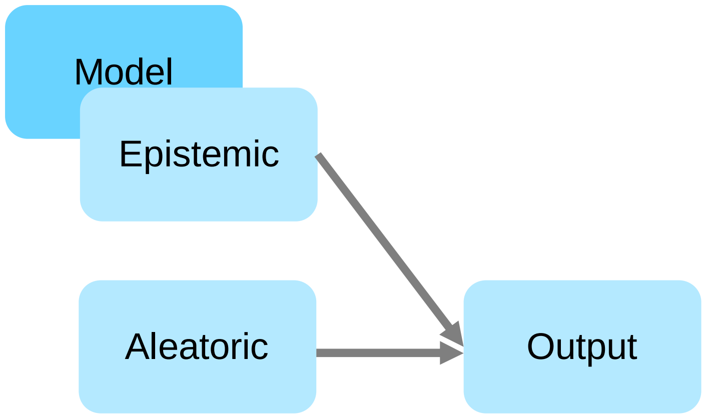
Caused by insufficient number of observations. In other words: The model is underdetermined.

More observations will reduce this type of uncertainty

Aleatoric

Caused by stochasticity or un-observability of an aspect of a system.

More observations will not reduce this type of uncertainty.
Different types of observations might.



Further Reading
Kendall, Alex, and Yarin Gal. "What uncertainties do we need in bayesian deep learning for computer vision?." Advances in neural information processing systems. 2017.

Model/Weight Uncertainty in Neural Networks

SIEMENS
Ingenuity for life

Bayes Theorem

Likelihood of Data | Prior

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Normalization

Is this radically different than “normal” Neural Networks?

Preparation:
Neural Networks in the Bayesian Framework

Parametric Model with Noise Term

$$t = \mathrm{f}(x, \theta) + \varepsilon$$
$$\varepsilon \sim N(0, \sigma)$$
$$\mathrm{D} = \{\mathrm{x}, \mathrm{t}\}$$

Inference on Weights

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Preparation:
Neural Networks in the Bayesian Framework

SIEMENS
Ingenuity for life

Likelihood

$$P(y|\theta) = N(y|\,\mathrm{f}(x,\theta) - t, \sigma)$$

$$P(D|\theta) = \prod_{(x,t)\in D} N(y|\,\mathrm{f}(x,\theta) - t, \sigma)$$

Prior

$$P(\boldsymbol{\theta}) = \prod_{\theta} N(\theta|0,1)$$

Log Posterior

$$\mathcal{L}\wp(\theta, D) = \underbrace{C_1 \sum_{(x,t)\in D} (\mathrm{f}(x,\theta) - t)^2}_{\text{MSE}} + \underbrace{C_2 \sum_{\theta} \theta^2}_{\text{L2 Weight Regularization}}$$

$$P(\theta|D) = \frac{\overbrace{P(D|\theta)}^{\text{Likelihood of Data}}\overbrace{P(\theta)}^{\text{Prior}}}{\underbrace{P(D)}_{\text{Normalization}}}$$

Preparation:
Neural Networks in the Bayesian Framework

SIEMENS
Ingenuity for life

$$\mathcal{L}\wp(\theta, D) = \underbrace{C_1 \sum_{(x,t)\in D} (\mathrm{f}(x,\theta) - t)^2}_{\text{MSE}} + \underbrace{C_2 \sum_{\theta} \theta^2}_{\text{L2 Weight Regularization}}$$

Optimization gives the **Maximum A Posterior** (MAP) Estimate

Further Reading
Barber, David, and Christopher M. Bishop. "Ensemble learning in Bayesian neural networks." NATO ASI SERIES F COMPUTER AND SYSTEMS SCIENCES 168 (1998): 215-238.

Why Bayes is Difficult in Practice

Two Main Challenges:

1. How to represent Distributions?
2. How to calculate $P(D) = \int P(D|\theta)P(\theta)d\theta$?

Likelihood of Data

Prior

$$P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}$$

Normalization

Some Recipes to deal with Bayes Theorem

Reduce it to a point estimate

→ use numerical optimization to find it

þ *That's what we just did!*

Assume conjugate distribution of the exponential family for $P(D|\theta)$ and $P(\theta)$

→ solve analytically

ý *Not working for Neural Networks!*

Empirically estimate $P(\theta|D)$

→ use Markov Chain Monte Carlo Methods to sample $P(\theta|D)$. (MCMC can sample from un-normalized distributions)

Ü *Let's take a closer look!*

Make simplifying assumptions such as factorization of $P(\theta|D)$

→ use Variational Methods to approximate $P(\theta|D)$

Ü *Let's take a closer look!*

***Note**: There are more recipes for special types of models. E.g. Message passing for graphical models.*

Empirical Estimate of Weights Distribution via Markov Chain Monte Carlo

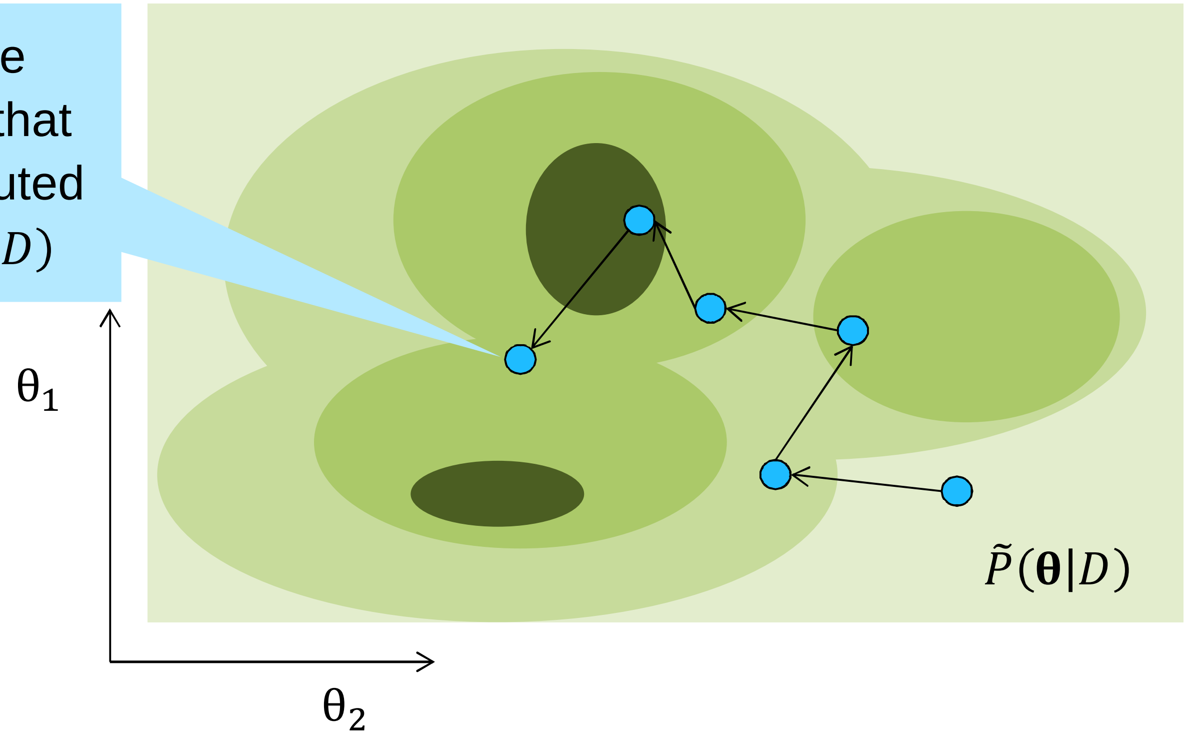
SIEMENS
Ingenuity for life

Problem Statement:

1. We want to sample from the distribution of Neural Network weights given the Data: $P(\theta|D)$.
2. For a given θ we can evaluate the un-normalized $\tilde{P}(\theta|D) = P(D|\theta)P(\theta)$.

Idea:

Jump around in the weight space such that the points are distributed proportional to $\tilde{P}(\boldsymbol{\theta}|D)$



How to Jump: The Metropolis-Hastings Algorithm

SIEMENS
Ingenuity for life

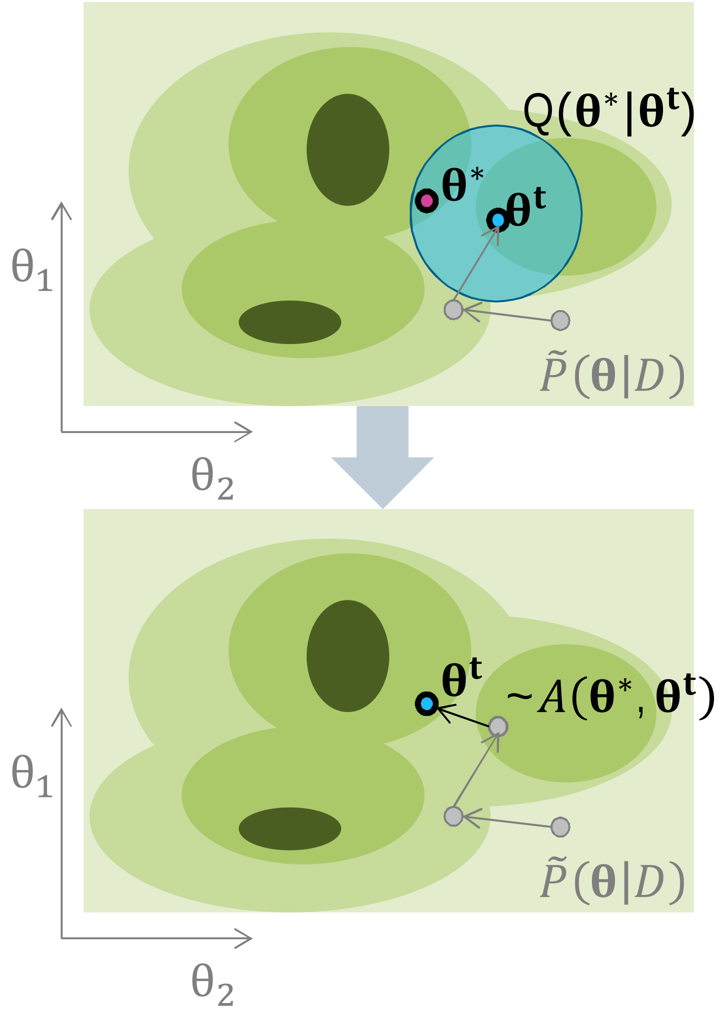
Split the Jump into two Phases

1. **Propose a new point in weight space**
   by sampling from a arbitrary but known proposal distribution
   - We call the proposal distribution $Q(\boldsymbol{\theta}^* | \boldsymbol{\theta}^{\mathbf{t}})$
   - We call the proposed weight vector $\boldsymbol{\theta}^*$
   - We call the current weight vector $\boldsymbol{\theta}^{\mathbf{t}}$

2. **Define an Acceptance Probability for the proposed weight vector**:

$$\text{accept} \sim A(\boldsymbol{\theta}^*, \boldsymbol{\theta}^{\mathbf{t}})$$

   - If accept: $\boldsymbol{\theta}^{\mathbf{t+1}} = \boldsymbol{\theta}^*$
   - else: $\boldsymbol{\theta}^{\mathbf{t+1}} = \boldsymbol{\theta}^{\mathbf{t}}$



How to Jump: The Metropolis-Hastings Algorithm

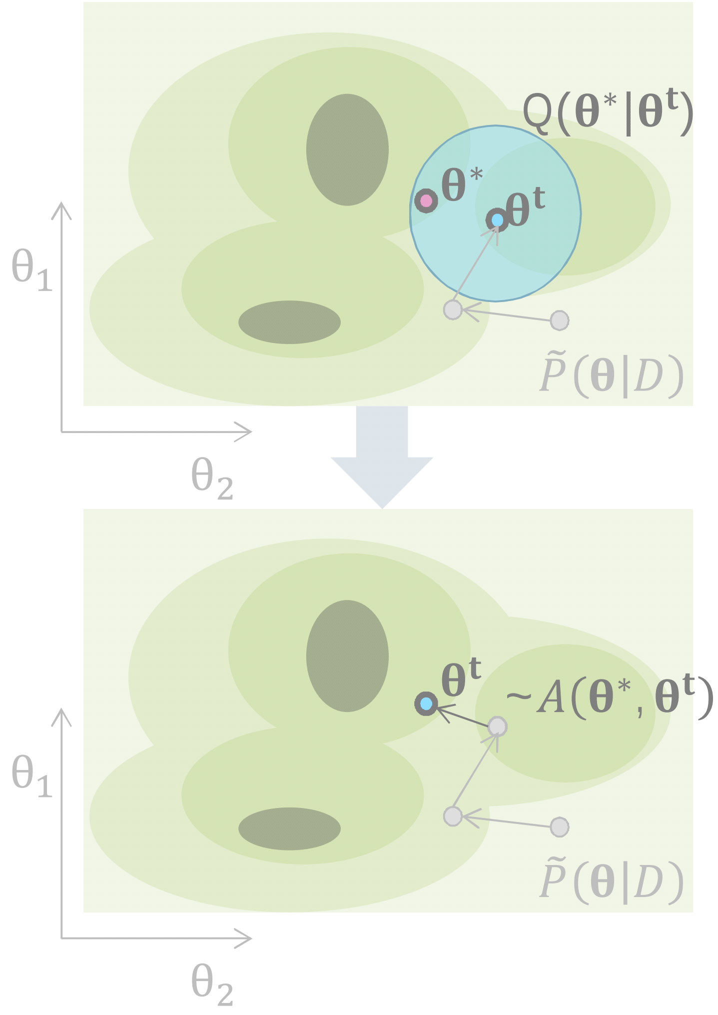
SIEMENS
Ingenuity for life

If the Acceptance Probability is chosen as

$$A(\boldsymbol{\theta}^*, \boldsymbol{\theta}^{\mathbf{t}}) = \min\left(1, \frac{\tilde{P}(\boldsymbol{\theta}^*|D)\mathrm{Q}(\boldsymbol{\theta}^{\mathbf{t}}|\boldsymbol{\theta}^*)}{\tilde{P}(\boldsymbol{\theta}^{\mathbf{t}}|D)\mathrm{Q}(\boldsymbol{\theta}^*|\boldsymbol{\theta}^{\mathbf{t}})}\right)$$

It can be proven that distribution of the sampled $\boldsymbol{\theta}^{\mathbf{t}}$, $\boldsymbol{\theta}^{\mathbf{t+1}}$, $\boldsymbol{\theta}^{\mathbf{t+2}}$… converges to $P(\theta|D)$

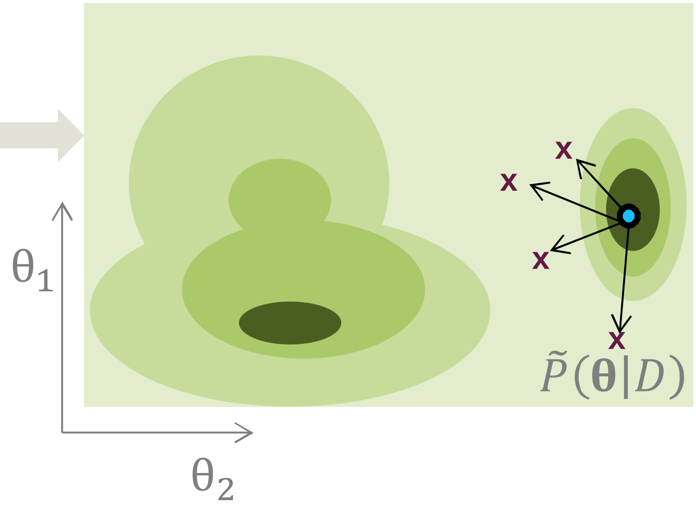
Further Reading
Chapter 11, Sampling Methods of Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin.



Caveats of Markov Chain Monte Carlo

1. Computational Complexity: One sampling step requires computation of $P(D|\theta)P(\theta)$ which includes D.

2. Convergence to $P(\theta|D)$ can be slow:



3. Trade-off between high acceptance and efficient traversal of weight-space
   - Long jumps make acceptance low
   - Short jumps makes movement in space slow

There is a large body of research addressing these points: Best use an existing framework or library!

Python Libraries
- PyMC3: https://docs.pymc.io/
- Edward: http://edwardlib.org/
- TensorFlow Probability: https://www.tensorflow.org/probability/

Big Caveat of Markov Chain Monte Carlo for Deep Learning

What's the memory requirement?

- S samples needed for each weight to build histogram
- W weights in a Deep Neural Network

→ O(S*W)

For VGG19: 26578886 and 1000 samples for each weight: ~26 Billion!
One weight is float32, so ~100 GB for the sample traces.

Parametric Estimate via the Variational Method

SIEMENS
Ingenuity for life

So, $P(\theta|D)$ is intractable? Let‘s simplify and assume:

$$P(\boldsymbol{\theta}|D) \approx Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$$

$$Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) = \prod_i Q_{\omega_i}(\theta_i)$$

$$\forall_i: \int d\theta_i . Q_{\omega_i}(\theta_i) \;\; = 1$$

Let’s measure the discrepancy between $P(\boldsymbol{\theta}|D)$ and $Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$ with the Kullback–Leibler divergence:

$$\mathrm{D}_{KL}(Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \,||P(\boldsymbol{\theta}|D)) = \int d\boldsymbol{\theta} . Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \log \frac{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{P(\boldsymbol{\theta}|D)}$$

Using D_{KL} is a design choice. There are good reasons to choose differently.

Further Reading
Hernández-Lobato, J. M., et al. "Black-Box α-divergence minimization." 33rd International Conference on Machine Learning, ICML 2016. Vol. 4. 2016

Further Reading
Ranganath, Rajesh, et al. "Operator variational inference." Advances in Neural Information Processing Systems. 2016.

If we can measure it, maybe we can minimize it with respect to ω…

Parametric Estimate via the Variational Method

SIEMENS
Ingenuity for life

$$\mathrm{D}_{KL}(Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})\,||P(\boldsymbol{\theta}|D)) = \int d\boldsymbol{\theta}.\,Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})\,\log\frac{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{P(\boldsymbol{\theta}|D)}$$

Still contains the intractable term $P(\boldsymbol{\theta}|D)$. Let's dissect it!

$$\int d\boldsymbol{\theta}.\,Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})\,\log\frac{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{P(\boldsymbol{\theta}|D)} = \int d\boldsymbol{\theta}.\,Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})\,\log\frac{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{P(\boldsymbol{\theta},D)} + \log P(D)$$

Rearrange:

$$\log P(D) = \mathrm{D}_{KL}(Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})\,||P(\boldsymbol{\theta}|D)) + \int d\boldsymbol{\theta}.\,Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})\,\log\frac{P(\boldsymbol{\theta},D)}{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}$$

Const for given D

≥ 0

Lower bound on log P(D)!

The Evidence Lower Bound

SIEMENS
Ingenuity for life

Let's give it a name: **Evidence Lower Bound**

$$\text{ELBO}(D) := \int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})\ \log \frac{P(\boldsymbol{\theta}, D)}{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}$$

Option A:
Use factorization and assume exponential family for Q

Coordinate Ascent Variational Inference (CAVI) / Mean Field Aproximation

ý

Option B:
Use sampling and stay in the gradient descent framework of neural networks.

Automatic Differentiation Variational Inference (ADVI) / Black Box Variational Inference

þ

(for deep neural networks)

Further Reading
Chapter 10, Approximate Inference of Christopher M. Bishop. 2006. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer-Verlag, Berlin.

The Evidence Lower Bound in Detail

SIEMENS
Ingenuity for life

$$\int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \, \log \frac{P(\boldsymbol{\theta}, D)}{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})} := \mathrm{ELBO}(D)$$

Further deconstruction yields:

$P(\boldsymbol{\theta}, D) = P(D|\boldsymbol{\theta})P(\boldsymbol{\theta})$

$$\int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \, \log \frac{P(\boldsymbol{\theta}, D)}{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})} = \int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \, \log P(D|\boldsymbol{\theta}) - \int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \, \log \frac{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{P(\boldsymbol{\theta})}$$

Average Log Likelihood

$\mathrm{D}_{KL}(Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \,||P(\boldsymbol{\theta}))$
Regularization

Alternative Decomposition

SIEMENS
Ingenuity for life

$$\int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \,\log \frac{P(\boldsymbol{\theta}, D)}{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})} = \int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \,\log P(D|\boldsymbol{\theta}) - \int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \,\log \frac{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{P(\boldsymbol{\theta})}$$

Average Log Likelihood

$\mathrm{D}_{KL}(Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})\,||P(\boldsymbol{\theta}))$

Regularization

$$\int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \,\log \frac{P(\boldsymbol{\theta}, D)}{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})} = \int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \,\log P(\boldsymbol{\theta}, \boldsymbol{D}) - \int d\boldsymbol{\theta}.\, Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \,\log Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})$$

Energy

Entropy

Can be solved analytically

Stochastic Gradient of the ELBO

SIEMENS
Ingenuity for life

$$\nabla_{\boldsymbol{\omega}} \mathrm{ELBO}(D) = \nabla_{\boldsymbol{\omega}} \int Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \, \log P(D|\boldsymbol{\theta}) - Q_{\boldsymbol{\omega}}(\boldsymbol{\theta}) \log \frac{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{P(\boldsymbol{\theta})} \, d\boldsymbol{\theta}$$

$$= \nabla_{\boldsymbol{\omega}} \mathbb{E}_{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})} \left[\log P(d|\boldsymbol{\theta}) - \log \frac{Q_{\boldsymbol{\omega}}(\boldsymbol{\theta})}{P(\boldsymbol{\theta})} \right]$$

If we could move the derivative into the Expectation, we could approximate it by sampling!

There are at least two options:

1. **Log Derivative Trick**
   ý suffers from high variance
   þ flexible with respect to the distribution

Viable and flexible option in combination with variance control techniques.
Also used in Reinforcement Learning

Further Reading
Williams, Ronald J. "Simple statistical gradient-following algorithms for connectionist reinforcement learning." Machine learning 8.3-4 (1992): 229-256.

2. **Reparametrization** (a.k.a. Perturbation Analysis, Pathwise Derivatives)
   þ easy to implement, þ low gradient variance
   ý does not work for every distribution

Further Reading
Ranganath, Rajesh, Sean Gerrish, and David Blei. "Black box variational inference." Artificial Intelligence and Statistics. 2014.

Reparametrization Trick

SIEMENS
Ingenuity for life

$$\nabla_\omega \mathbb{E}_{p(z|\omega)}[f(z)] = \nabla_\omega \int p(z|\omega) f(z)\, dz$$

Find a way to reparametrize $p(z|\omega)$ such that $z \sim g(\varepsilon, \omega)$ and we just sample ε

$$\nabla_\omega \int p(z|\omega) f(z)\, dz = \nabla_\omega \int p(\varepsilon) f(g(\varepsilon, \omega))\, d\varepsilon$$

We move the derivative inside the integral:

warranted by the dominated convergence theorem

$$= \int p(\varepsilon) \nabla_\omega f(g(\varepsilon, \omega))\, d\varepsilon = \mathbb{E}_{p(\varepsilon)}[\nabla_\omega f(g(\varepsilon, \omega))]$$

Example:

$$z \sim \mathcal{N}(\mu, \sigma) \rightarrow z \sim \mu + \sigma\varepsilon; \varepsilon \sim \mathcal{N}(0,1)$$

þ ε is now independent of μ, σ and backpropagation works!

Further Reading
Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013).

Flipout

Challenge:
“because a network typically has many more weights than units, it is very expensive to compute and store separate weight perturbations for every example in a mini-batch. Therefore, stochastic weight methods are typically done with a single sample per mini-batch.“

Further Reading
Wen, Yeming, et al. "Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches." arXiv preprint arXiv:1803.04386 (2018).

For VGG19: 26578886 weights and a batch size of 100 : ~2.6 Billion!

→ **Same sample for the entire batch**
→ **high variance in the gradient**
→ **small learning rate needed**
→ **slow convergence**

Flipout

SIEMENS
Ingenuity for life

Idea:
Add independence to the weight samples in one batch with low computational cost.

Conditions:
1. weight distribution is symmetric
2. each weight is sampled independently

These conditions are met for our stochastic variable (perturbation) ε!

- Let ΔW be a matrix of independently sampled $\varepsilon \sim \mathcal{N}(0,1)$
- Let E be a matrix of independently sampled signs ± 1

$$\widehat{\Delta W} = \Delta W \circ E$$

$\widehat{\Delta W}$ and ΔW are equally distributed, but $\widehat{\Delta W}$ is less correlated!

Flipout

SIEMENS
Ingenuity for life

$$\widehat{\Delta W} = \Delta W \circ E$$

E is as big as a fully sampled weights matrix for the batch. Nothing gained yet!

Can we generate E cheaply?

Idea:
Use a rank one E induced by a product of two random sign vectors r and s :

$$E = rs^T$$

For a full analysis of the gradient variance reduction and computational cost see:

Further Reading
Wen, Yeming, et al. "Flipout: Efficient Pseudo-Independent Weight Perturbations on Mini-Batches." arXiv preprint arXiv:1803.04386 (2018).

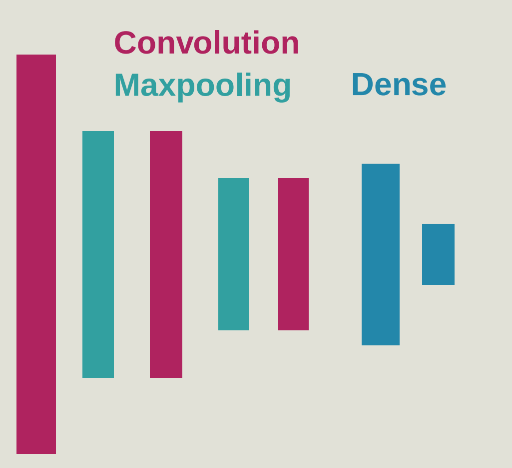
Implementation in TensorFlow Probability

SIEMENS
Ingenuity for life

The Bayes Version of the LeNet CNN Architecture:
Taken from the TensorFlow Probability Examples

```
neural_net = tf.keras.Sequential([
    tfp.layers.Convolution2DFlipout(6, kernel_size=5, padding="SAME", activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D(pool_size=[2, 2], strides=[2, 2], padding="SAME"),
    tfp.layers.Convolution2DFlipout(16, kernel_size=5, padding="SAME", activation=tf.nn.relu),
    tf.keras.layers.MaxPooling2D(pool_size=[2, 2], strides=[2, 2], padding="SAME"),
    tfp.layers.Convolution2DFlipout(120, kernel_size=5, padding="SAME", activation=tf.nn.relu),
    tf.keras.layers.Flatten(),
    tfp.layers.DenseFlipout(84, activation=tf.nn.relu),
    tfp.layers.DenseFlipout(10)
    ])

labels_distribution = tfd.Categorical( logits = neural_net(images) )

neg_log_likelihood = -tf.reduce_mean(labels_distribution.log_prob(labels))
kl = sum(neural_net.losses) / mnist_data.train.num_examples
elbo_loss = neg_log_likelihood + kl
```

$$\approx -\int d\boldsymbol{\theta}.\, Q_\omega(\boldsymbol{\theta})\ \log P(D|\boldsymbol{\theta}) + \int d\boldsymbol{\theta}.\, Q_\omega(\boldsymbol{\theta})\ \log \frac{Q_\omega(\boldsymbol{\theta})}{P(\boldsymbol{\theta})}$$

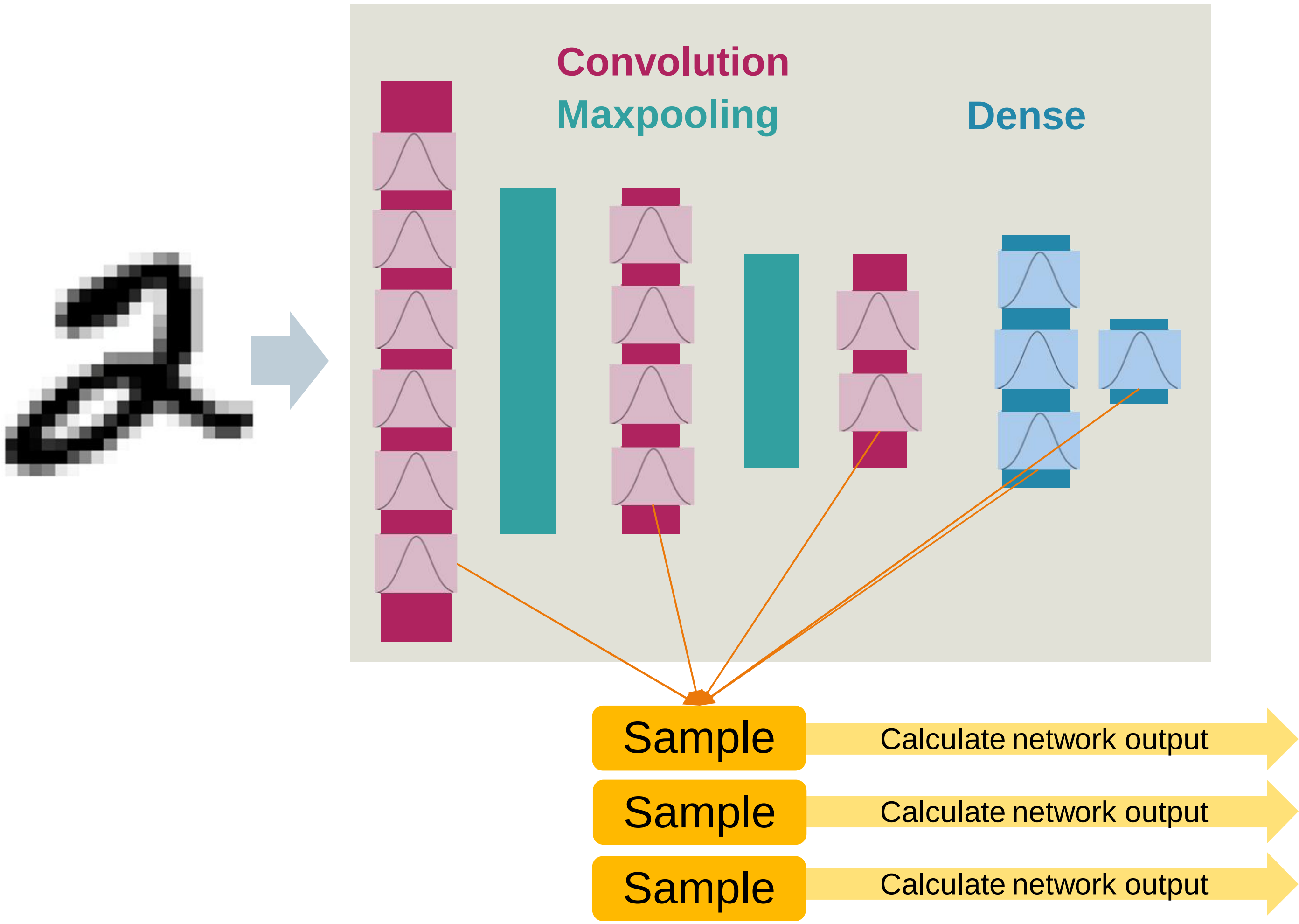


Further Reading on LeNet
LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.

Further Practice
https://github.com/tensorflow/probability/tree/master/tensorflow_probability/examples

Sampling from a Bayesian Neural Network

SIEMENS
Ingenuity for life



0	1	2	3	4	...	9
0	0	.98	0	0		.02
0	.01	.97	0	0		.01
0	0	.99	0	0		0

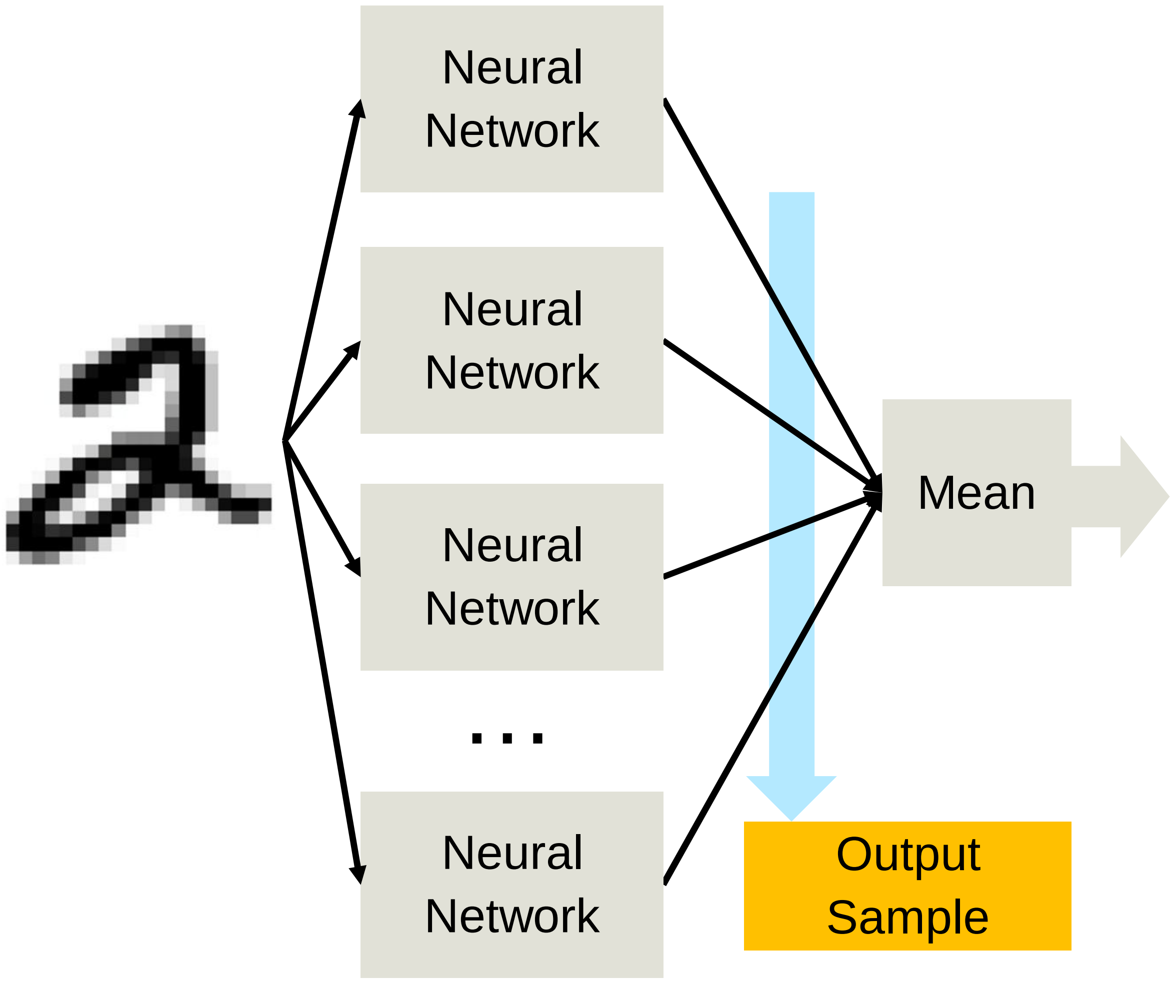
Pros

- **Estimate of Model Uncertainty**
  (If your use-case benefits from it!)
- Implies Output Uncertainty
- Acts as Regularization

Con

- Computational / Memory Overhead

Approximating Bayesian Networks with Ensembles



Pros

- Simple
- Parallelizable
- Reduction of expected error

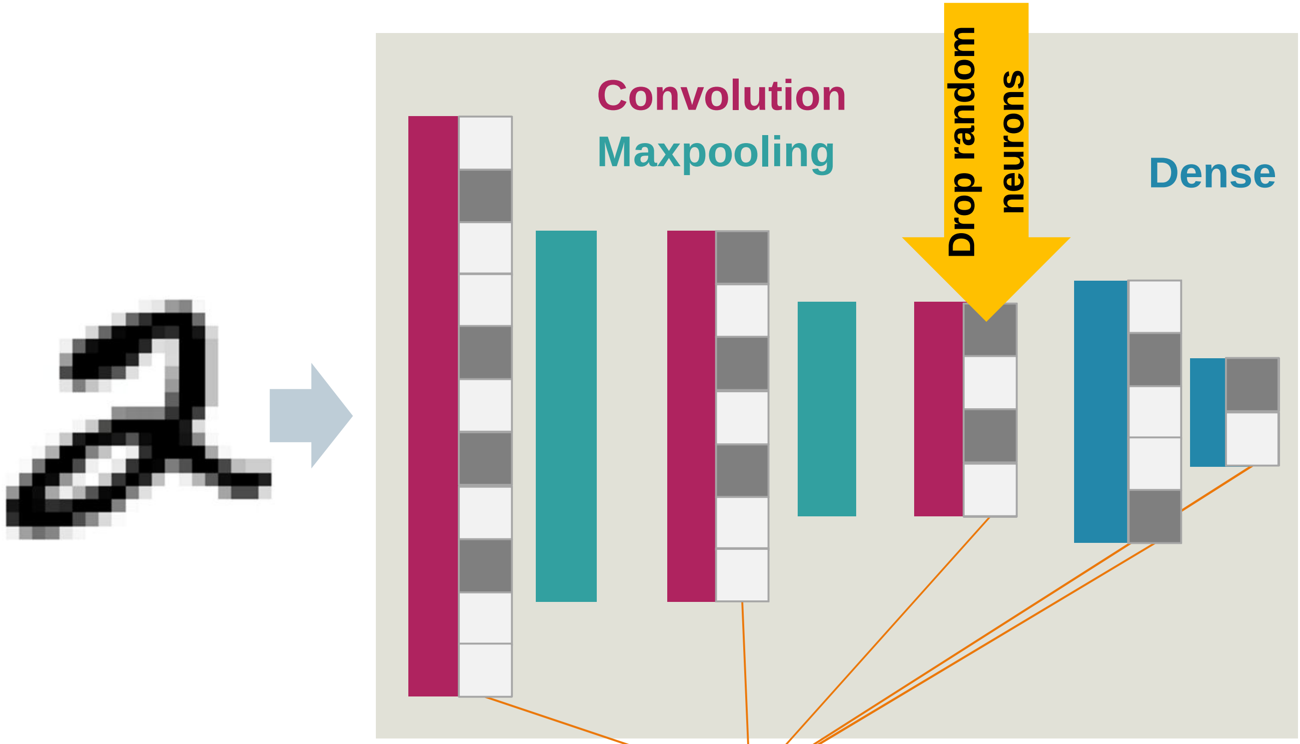
Con

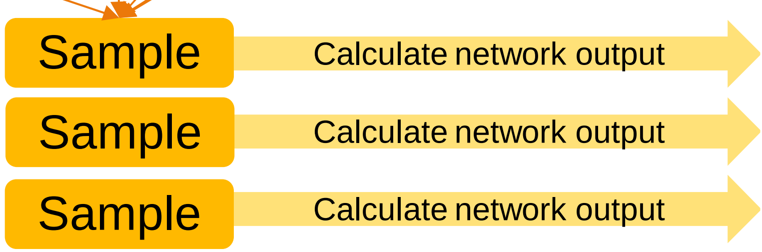
- Excessive training time for modern Deep Neural Networks

Further Reading
Perrone, Michael P., and Leon N. Cooper. When networks disagree: Ensemble methods for hybrid neural networks. No. TR-61. BROWN UNIV PROVIDENCE RI INST FOR BRAIN AND NEURAL SYSTEMS, 1992.

Approximating Ensembles with Dropout

SIEMENS
Ingenuity for life





0	1	2	3	4	...	9
0	0	.98	0	0		.02
0	.01	.97	0	0		.01
0	0	.99	0	0		0

Further Reading
Gal, Yarin, and Zoubin Ghahramani. "Dropout as a Bayesian approximation: Insights and applications." Deep Learning Workshop, ICML. Vol. 1. 2015.

Pros

- Implies Output Uncertainty
- Acts as Regularization
- Computationally highly efficient

Con

- Only indirect estimate of Model Uncertainty

In case you want to delve deeper into Bayesian Methods

SIEMENS
Ingenuity for life

More Theory

Further Reading
Jaynes, Edwin T. Probability theory: The logic of science. Cambridge university press, 2003

Further Reading
Gelman, Andrew, et al. Bayesian data analysis. Chapman and Hall/CRC, 2013.

Further Reading
Sivia, Devinderjit, and John Skilling. Data analysis: a Bayesian tutorial. OUP Oxford, 2006.

Further Practice
https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers

More Practice