Lecture-6: Uncertainty in Neural Networks
Motivation

“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*
Types of Uncertainty

Classical Neural Network (or any parametric model)

\[ y = f(x, \theta) ; \quad D = \{x, t\} \]

Output/Prediction Uncertainty

\[ y \sim p(y|x, \theta) \]

Model (Weight) Uncertainty

\[ \theta \sim p(\theta|D) \]

Conventions for this lecture
- \( \theta \) = neural network weights
- \( x \) = neural network input
- \( y \) = neural network output
- \( D \) = training data (input and targets)
Three Different Ways to represent Distributions in Practice

Point Estimate

Parametric Distribution

Empirical Distribution
Pros and Cons

**Point Estimate**
- Calculation efficient and stable
- Uncertainty not accounted for
- Cannot be sampled
- Assumption that point estimate is representative: e.g. Failure with multi-modality

**Parametric Distribution**
- Calculation relatively efficient
- Analytical Representation
- Can be sampled
- Breaks down if assumption on class of distribution is not correct
- Hard to represent multi-modality

**Empirical Distribution**
- Flexible: No assumptions
- Can be sampled (more or less)
- Breaks down for higher dimensional data
- High memory consumption
- Artifacts due to discretization
Output Uncertainty for Regression
Recap: Classification

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

We will focus on Regression henceforth …

Understates the true uncertainty!
Motivation for Output Uncertainty:
Mean Average Precision @ 7

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

Related Kaggle Challenge
Neural Network with Output Uncertainty

\[ y \sim p(y|x, \theta) \]

Let’s commit to a parametric distribution:

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

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

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

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

Your choice!
Also popular: The Laplace Distribution
Neural Network with Output Uncertainty

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

\[ D = \{x, t\} \]

We will optimize the Log-Likelihood with respect to the weights \( \theta \)

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

Use a variant of Stochastic Gradient Descent to train.
Network with Output Uncertainty: Architecture Variants

\[
\text{Loss: } -\log P(Y|X)
\]

Understates the true uncertainty!
Multimodality

Fit Gaussian

Does not make sense
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
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

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Mean</th>
<th>Variance</th>
<th>weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cluster 1</td>
<td>4.1</td>
<td>1.8</td>
<td>0.6</td>
</tr>
<tr>
<td>Cluster 2</td>
<td>-2.0</td>
<td>2.7</td>
<td>0.38</td>
</tr>
<tr>
<td>Cluster 3</td>
<td>-0.6</td>
<td>0.8</td>
<td>0.02</td>
</tr>
</tbody>
</table>

Further Reading


Further Practice with GMMs

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

If you want to know how many clusters to assume:
The Mixture Density Network: Architecture

- $\sigma$: standard deviations
- $\mu$: means
- $\pi$: cluster weights

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

Reference
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))
The Mixture Density Network: Result
Model Uncertainty
Model Uncertainty

\[ y = f(x, \theta) ; D = \{x, t\} \]

Model (Weight) Uncertainty

\[ \theta \sim p(\theta|D) \]

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.

<table>
<thead>
<tr>
<th>Head</th>
<th>Tails</th>
</tr>
</thead>
<tbody>
<tr>
<td>501 071</td>
<td>498 929</td>
</tr>
</tbody>
</table>

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

Data

<table>
<thead>
<tr>
<th>t</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>0</td>
<td>1</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

What is your prediction for y?

How sure are you? In other words: How sure are you about your model?
Epistemic and Aleatoric Uncertainty

**Epistemic**
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**
Model/Weight Uncertainty in Neural Networks

Bayes Theorem

\[
P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}
\]

Is this radically different than “normal” Neural Networks?
Preparation:
Neural Networks in the Bayesian Framework

Parametric Model with Noise Term

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

Inference on Weights

\[ P(\theta | D) = \frac{P(D | \theta)P(\theta)}{P(D)} \]
Preparation: Neural Networks in the Bayesian Framework

Likelihood

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

Prior

\[ P(\theta) = \prod_{\theta} N(\theta|0,1) \]

Log Posterior

\[ \mathcal{L}_\phi(\theta, D) = C_1 \sum_{(x,t) \in D} (f(x,\theta) - t)^2 + C_2 \sum_{\theta} \theta^2 \]

\[ P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)} \]

Normalization
Preparation: 
Neural Networks in the Bayesian Framework

\[ \mathcal{L}_\theta(\theta, D) = C_1 \sum_{(x,t) \in D} (f(x, \theta) - t)^2 + C_2 \sum_{\theta} \theta^2 \]

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

**Further Reading**
Why Bayes is Difficult in Practice

Two Main Challenges:

1. How to represent Distributions?
2. How to calculate $P(D) = \int P(D|\theta)P(\theta)d\theta$?
### Some Recipes to deal with Bayes Theorem

| Reduce it to a point estimate | Assume conjugate distribution of the exponential family for $P(D|\theta)$ and $P(\theta)$ |
|--------------------------------|----------------------------------------------------------------------------------|
| → use numerical optimization to find it | → solve analytically |
| "That's what we just did!" | "Not working for Neural Networks!" |

| Empirically estimate $P(\theta|D)$ | Make simplifying assumptions such as factorization of $P(\theta|D)$ |
|----------------------------------|---------------------------------------------------------------|
| → use Markov Chain Monte Carlo Methods to sample $P(\theta|D)$. (MCMC can sample from un-normalized distributions) | → use Variational Methods to approximate $P(\theta|D)$ |
| "Let’s take a closer look!" | "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

Problem Statement:
1. We want to sample from the distribution of Neural Network weights given the Data: \( P(\theta|D) \).
2. For a given \( \theta \) 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 \( \tilde{P}(\theta|D) \)
How to Jump: The Metropolis-Hastings Algorithm

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 $Q(\theta^* | \theta^t)$
   • We call the proposed weight vector $\theta^*$
   • We call the current weight vector $\theta^t$

2. Define an Acceptance Probability for the proposed weight vector:
   $\text{accept} \sim A(\theta^*, \theta^t)$
   • If accept: $\theta^{t+1} = \theta^*$
   • else: $\theta^{t+1} = \theta^t$
How to Jump: The Metropolis-Hastings Algorithm

If the Acceptance Probability is chosen as

\[ A(\theta^*, \theta^t) = \min \left( 1, \frac{\bar{P}(\theta^* | D)Q(\theta^t | \theta^*)}{\bar{P}(\theta^t | D)Q(\theta^* | \theta^t)} \right) \]

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

**Further Reading**
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 make 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

$\rightarrow O(S \times W)$

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

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

$$P(\theta|D) \approx Q_\omega(\theta)$$

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

$$\forall i: \int d\theta_i \cdot Q_{\omega_i}(\theta_i) = 1$$

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

$$D_{KL}(Q_\omega(\theta) \mid \mid P(\theta|D)) = \int d\theta \cdot Q_\omega(\theta) \log \frac{Q_\omega(\theta)}{P(\theta|D)}$$

If we can measure it, maybe we can minimize it with respect to $\omega$…

Further Reading

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

Further Reading
Parametric Estimate via the Variational Method

\[ D_{KL}(Q_\theta(\theta) \mid \mid P(\theta|D)) = \int d\theta \cdot Q_\theta(\theta) \log \frac{Q_\theta(\theta)}{P(\theta|D)} \]

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

\[ \int d\theta \cdot Q_\theta(\theta) \log \frac{Q_\theta(\theta)}{P(\theta|D)} = \int d\theta \cdot Q_\theta(\theta) \log \frac{Q_\theta(\theta)}{P(\theta,D)} + \log P(D) \]

Rearrange:

\[ \log P(D) = D_{KL}(Q_\theta(\theta) \mid \mid P(\theta|D)) + \int d\theta \cdot Q_\theta(\theta) \log \frac{P(\theta,D)}{Q_\theta(\theta)} \geq 0 \]

Lower bound on \( \log P(D) \)!
The Evidence Lower Bound

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

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

**Option A:**
Use factorization and assume exponential family for $Q$

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

**Coordinate Ascent Variational Inference (CAVI) / Mean Field Approximation**

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

(for deep neural networks)

**Further Reading**
The Evidence Lower Bound in Detail

\[ \int d\theta \cdot Q_\omega(\theta) \log \frac{P(\theta, D)}{Q_\omega(\theta)} := \text{ELBO}(D) \]

Further deconstruction yields:

\[ P(\theta, D) = P(D|\theta)P(\theta) \]

\[ \int d\theta \cdot Q_\omega(\theta) \log \frac{P(\theta, D)}{Q_\omega(\theta)} = \int d\theta \cdot Q_\omega(\theta) \log P(D|\theta) - \int d\theta \cdot Q_\omega(\theta) \log \frac{Q_\omega(\theta)}{P(\theta)} - D_{KL}(Q_\omega(\theta) \| P(\theta)) \]

- **Average Log Likelihood**
- **Regularization**
Alternative Decomposition

\[
\int d\theta \cdot Q_\omega(\theta) \log \frac{P(\theta, D)}{Q_\omega(\theta)} = \int d\theta \cdot Q_\omega(\theta) \log P(D|\theta) - \int d\theta \cdot Q_\omega(\theta) \log \frac{Q_\omega(\theta)}{P(\theta)}
\]

Average Log Likelihood

\[
\int d\theta \cdot Q_\omega(\theta) \log \frac{P(\theta, D)}{Q_\omega(\theta)} = \int d\theta \cdot Q_\omega(\theta) \log P(\theta, D) - \int d\theta \cdot Q_\omega(\theta) \log Q_\omega(\theta)
\]

Energy

\[
D_{KL}(Q_\omega(\theta) || P(\theta))
\]

Regularization

Can be solved analytically

Entropy
Stochastic Gradient of the ELBO

\[ \nabla_\omega \text{ELBO}(D) = \nabla_\omega \int Q_\omega(\theta) \log P(D|\theta) - Q_\omega(\theta) \log \frac{Q_\omega(\theta)}{P(\theta)} d\theta \\
= \nabla_\omega \mathbb{E}_{Q_\omega(\theta)} \left[ \log P(d|\theta) - \log \frac{Q_\omega(\theta)}{P(\theta)} \right] \]

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**
   - suffers from high variance
   - flexible with respect to the distribution

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

Further Reading


Reparametrization Trick

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

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

\[ \nabla_\omega \int p(z|\omega)f(z) \, dz = \nabla_\omega \int p(\epsilon)f(g(\epsilon, \omega)) \, d\epsilon \]

We move the derivative inside the integral:

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

Example:

\[ z \sim \mathcal{N}(\mu, \sigma) \rightarrow z \sim \mu + \sigma \epsilon; \epsilon \sim \mathcal{N}(0,1) \]

\( \epsilon \) is now independent of \( \mu, \sigma \) and backpropagation works!

Further Reading

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.“

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

Further Reading
Flipout

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) $\varepsilon$!

• Let $\Delta W$ be a matrix of independently sampled $\varepsilon \sim N(0,1)$
• Let $E$ be a matrix of independently sampled signs $\pm 1$

$$\Delta W = \Delta W \circ E$$

$\Delta W$ and $\Delta W$ are equally distributed, but $\Delta W$ is less correlated!
Flipout

\[ \Delta \hat{W} = \Delta W \times E \]

\( E \) 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
The Bayes Version of the LeNet CNN Architecture:

*Taken from the TensorFlow Probability Examples*

```python
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
```

Further Reading on LeNet


Further Practice

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

\[ -\int d\theta \cdot Q_\omega(\theta) \log P(D | \theta) + \int d\theta \cdot Q_\omega(\theta) \log \frac{Q_\omega(\theta)}{P(\theta)} \]
Sampling from a Bayesian Neural Network

Pros

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

Con

- Computational / Memory Overhead

### Example Table

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>.98</td>
<td>0</td>
<td>0</td>
<td></td>
<td>.02</td>
</tr>
<tr>
<td>0</td>
<td>.01</td>
<td>.97</td>
<td>0</td>
<td>0</td>
<td></td>
<td>.01</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>.99</td>
<td>0</td>
<td>0</td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>
Approximating Bayesian Networks with Ensembles

Pros
- Simple
- Parallelizable
- Reduction of expected error

Con
- Excessive training time for modern Deep Neural Networks

Further Reading
Approximating Ensembles with Dropout

Further Reading

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

Con
- Only indirect estimate of Model Uncertainty

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

Con
- Only indirect estimate of Model Uncertainty

### Further Reading

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>...</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>.98</td>
<td>0</td>
<td>0</td>
<td>.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>.01</td>
<td>.97</td>
<td>0</td>
<td>0</td>
<td>.01</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>.99</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>
In case you want to delve deeper into Bayesian Methods

Further Reading

Further Reading

Further Reading

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