Probability and Uncertainty in Deep Learning

Andreas Damianou

andreasdamianou.com

Amazon Research, Cambridge

Computer Science Colloquium, Warwick
6 June 2019

amazon.com

Notebook

http://adamian.github.io/talks/Damianou_DL_tutorial_18.ipynb

Outline

- Deep neural networks
- Uncertainty: Bayesian deep neural networks
- Focus: Bayesian generative models
- Deep Gaussian processes

What is Machine Learning?

$$\text{data} + \text{model} \xrightarrow{\text{compute}} \text{prediction}$$

To combine model with data we need:

- A **prediction function** $f(\cdot)$ includes our beliefs about the regularities of the universe.
- An **loss function** $L(\cdot)$ defines the cost of misprediction.

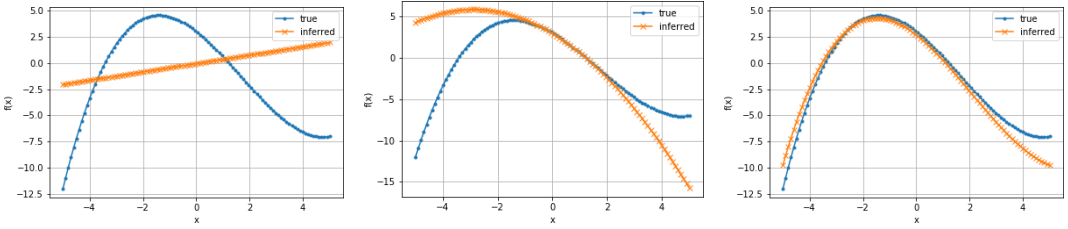
N. Lawrence's tutorial: inverseprobability.com/talks/notes/probabilistic-machine-learning.html

Regression example

Predict outputs $\mathbf{y}$ from inputs $\mathbf{x}$: $f(\mathbf{x}) : \mathbf{x} \mapsto \mathbf{y}$.

$$f(\mathbf{x}) = \phi(\mathbf{x}\mathbf{w} + \mathbf{b})$$

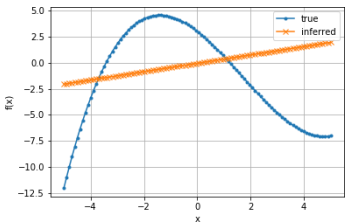
$$L(f(\mathbf{x}), \mathbf{y}) = (f(\mathbf{x}) - \mathbf{y})^2$$

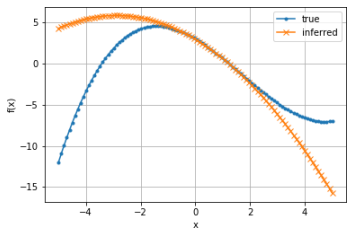


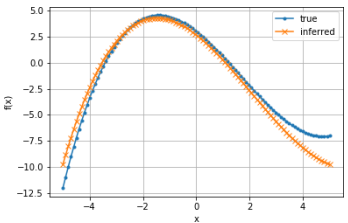
Regression example

Predict outputs $\mathbf{y}$ from inputs $\mathbf{x}$: $f(\mathbf{x}) : \mathbf{x} \mapsto \mathbf{y}$.

$$f(\mathbf{x}) = \phi(\mathbf{x} \quad \mathbf{w} + \mathbf{b})$$
$$f(\mathbf{x}) = \phi(\phi(\mathbf{x}\mathbf{w}' + \mathbf{b}')\mathbf{w} + \mathbf{b})$$
$$L(f(\mathbf{x}), \mathbf{y}) = (f(\mathbf{x}) - \mathbf{y})^2$$







Deep neural networks: hierarchical function definitions

A neural network is a composition of functions (layers), each parameterized with a *weight vector* $\mathbf{w}_l$. E.g. for 2 layers:

$$f_{\text{net}} = h_2(h_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2).$$

Generally $f_{\text{net}} : \mathbf{x} \mapsto \mathbf{y}$ with:

$$\begin{aligned}
\mathbf{h}_1 &= \varphi(\mathbf{x}\mathbf{w}_1 + b_1) \\
\mathbf{h}_2 &= \varphi(\mathbf{h}_1\mathbf{w}_2 + b_2) \\
&\dots \\
\hat{\mathbf{y}} &= \varphi(\mathbf{h}_{L-1}\mathbf{w}_L + b_L)
\end{aligned}$$

ϕ is the (non-linear) activation function.

Deep neural networks: hierarchical function definitions

A neural network is a composition of functions (layers), each parameterized with a *weight vector* $\mathbf{w}_l$. E.g. for 2 layers:

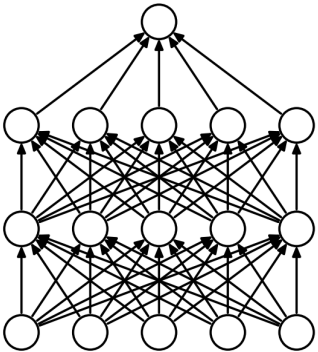
$$f_{\text{net}} = h_2(h_1(\mathbf{x}; \mathbf{w}_1); \mathbf{w}_2).$$

Generally $f_{\text{net}} : \mathbf{x} \mapsto \mathbf{y}$ with:

$$\begin{aligned}
\mathbf{h}_1 &= \varphi(\mathbf{x}\mathbf{w}_1 + b_1) \\
\mathbf{h}_2 &= \varphi(\mathbf{h}_1\mathbf{w}_2 + b_2) \\
&\dots \\
\hat{\mathbf{y}} &= \varphi(\mathbf{h}_{L-1}\mathbf{w}_L + b_L)
\end{aligned}$$

ϕ is the (non-linear) activation function.

Graphical depiction



Optimization

One layer:

$$Loss = \frac{1}{2}(\mathbf{h} - \mathbf{y})^2$$

$$\mathbf{h} = \phi(\mathbf{x}\mathbf{w})$$

$$\frac{\vartheta Loss}{\vartheta \mathbf{w}} = \underbrace{(\mathbf{y} - \mathbf{h})}_{\epsilon} \frac{\vartheta \phi(\mathbf{x}\mathbf{w})}{\vartheta \mathbf{w}}$$

Two layers:

$$Loss = \frac{1}{2}(\mathbf{h}_2 - \mathbf{y})^2$$

$$\mathbf{h}_2 = \phi\left[\underbrace{\phi(\mathbf{x}\mathbf{w}_0)}_{\mathbf{h}_1}\mathbf{w}_1\right]$$

$$\frac{\vartheta Loss}{\vartheta \mathbf{w}_0} = \cdots$$

$$\frac{\vartheta Loss}{\vartheta \mathbf{w}_1} = \cdots$$

Derivative w.r.t $\mathbf{w}_1$

$$\begin{aligned}
\frac{\vartheta(\mathbf{h}_2 - \mathbf{y})^2}{\vartheta \mathbf{w}_1} &= -2\frac{1}{2}(\mathbf{h}_2 - \mathbf{y})\frac{\vartheta \mathbf{h}_2}{\vartheta \mathbf{w}1} = \\
&= (\mathbf{y} - \mathbf{h}_2)\frac{\vartheta \phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta \mathbf{w}_1} = \\
&= (\mathbf{y} - \mathbf{h}_2)\frac{\vartheta \phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta \mathbf{h}_1\mathbf{w}_1}\frac{\vartheta \mathbf{h}_1\mathbf{w}_1}{\vartheta \mathbf{w}_1} = \\
&= \underbrace{(\mathbf{y} - \mathbf{h}_2)}_{\epsilon_2}\underbrace{\frac{\vartheta \phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta \mathbf{h}_1\mathbf{w}_1}}_{g_1}\mathbf{h}_1^T
\end{aligned}$$

$\mathbf{h}_1$ is computed during the *forward pass*.

Derivative w.r.t $\mathbf{w}_0$

$$\frac{\vartheta(\mathbf{h}_2 - \mathbf{y})^2}{\vartheta \mathbf{w}_0} = -2\frac{1}{2}(\mathbf{h}_2 - \mathbf{y})\frac{\vartheta \mathbf{h}_2}{\vartheta \mathbf{w}_0} =$$

$$= (\mathbf{y} - \mathbf{h}_2)\frac{\vartheta \phi(\mathbf{h}_1\mathbf{w}_1)}{\vartheta \mathbf{h}_1\mathbf{w}_1}\frac{\vartheta \mathbf{h}_1\mathbf{w}_1}{\vartheta \mathbf{h}_1}\frac{\vartheta \mathbf{h}_1}{\vartheta \mathbf{w}_0} =$$

$$= \epsilon_2\ g_1\ \mathbf{w}_1^T\ \frac{\vartheta \phi(\mathbf{x}\mathbf{w}_0)}{\mathbf{x}\mathbf{w}_0}\frac{\vartheta \mathbf{x}\mathbf{w}_0}{\vartheta \mathbf{w}_0} =$$

$$= \epsilon_2\ g_1\ \mathbf{w}_1^T\ \underbrace{\frac{\vartheta \phi(\mathbf{x}\mathbf{w}_0)}{\vartheta \mathbf{x}\mathbf{w}_0}}_{g_0}\ \mathbf{x}^T$$

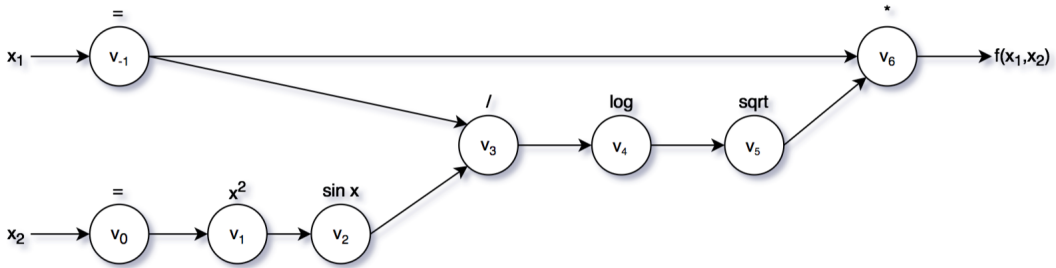
DNN optimization observations

- Propagation of error is just the chain rule (calculus).
- Factors accumulate, can lead to problems.
- Difficult to do all this by hand.

Go to notebook!

Automatic differentiation

Example: $f(x_1, x_2) = x_1 \sqrt{\log \frac{x_1}{sin(x_2^2)}}$ has *symbolic graph*:



(image: sanyamkapoor.com)

Lottery ticket hypothesis

There's no automatic regularization (hence tricks like early stopping, dropout etc).

- Optimization landscape: multiple optima and difficult to navigate
- Over-parameterized networks contain multiple sub-networks ("lottery tickets")
- "Winning ticket": a lucky sub-network found a good solution
- Over-parameterization: more tickets, higher winning probability
- Of course this means we have to prune or at least regularize.

(Frankle and Carbin (2018))

(Bayesian) Occam's Razor

"A plurality is not to be posited without necessity" (W. of Ockham).

"Everything should be made as simple as possible, but not simpler" (A. Einstein).

- How to represent complexity if we only deal with one of many possible sets of parameters?
- Is a $10-$layer network with all weights $= 0$ more complex than an shallow one?
- Bayesian deep learning marginalizes out the parameters.

(Bayesian) Occam's Razor

"A plurality is not to be posited without necessity" (W. of Ockham).

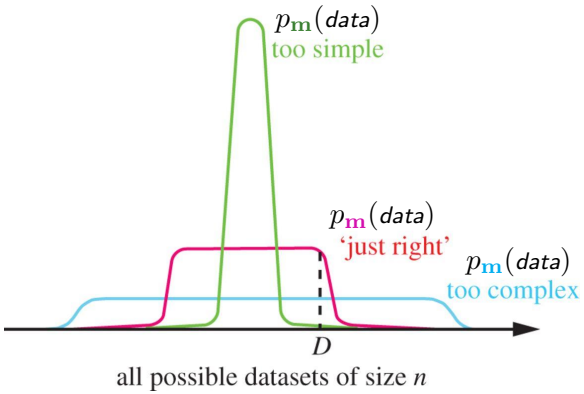
"Everything should be made as simple as possible, but not simpler" (A. Einstein).

- How to represent complexity if we only deal with one of many possible sets of parameters?
- Is a $10-$layer network with all weights $= 0$ more complex than an shallow one?
- Bayesian deep learning marginalizes out the parameters.

(Bayesian) Occam's Razor

$D \coloneqq (\mathbf{x}, \mathbf{y})$

$m = \{\#\text{layers, activation\_type, ...}\}$



Higher $p_m(\textit{data} = D)$ when m explains D well without being unnecessarily complex.

Integrating out weights

- Define: $D \coloneqq (\mathbf{x}, \mathbf{y})$
- Remember Bayes' rule:

$$p(\mathbf{w}|D) = \frac{p(D|\mathbf{w})p(\mathbf{w})}{p(D) = \int p(D|\mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}}$$

- From calculus (gradients for $\mathbf{w}^*$) to Bayesian inference (posterior $p(\mathbf{w}|D)$).

Integrating out weights

- Define: $D \coloneqq (\mathbf{x}, \mathbf{y})$
- Remember Bayes' rule:

$$p(\mathbf{w}|D) = \frac{p(D|\mathbf{w})p(\mathbf{w})}{p(D) = \int p(D|\mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}}$$

- From calculus (gradients for $\mathbf{w}^*$) to Bayesian inference (posterior $p(\mathbf{w}|D)$).

Probabilistic re-formulation

- Training minimizing loss:

$$\arg\min_{\mathbf{w}} \underbrace{\frac{1}{2}\sum_{i=1}^{N}(f_{\text{net}}(\mathbf{w}, x_i) - y_i)^2}_{\text{fit}} + \underbrace{\lambda \sum_i \parallel \mathbf{w}_i \parallel}_{\text{regularizer}}$$

- Equivalent probabilistic view for regression, maximizing posterior probability:

$$\arg\max_{\mathbf{w}} \underbrace{\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})}_{\text{fit}} + \underbrace{\log p(\mathbf{w})}_{\text{regularizer}}$$

  where $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \sim \mathcal{N}$ and $p(\mathbf{w}) \sim$ Laplace

- For full Bayesian inference we also need $\int p(\mathbf{y}|\mathbf{x}, \mathbf{w}) p(\mathbf{w}) \mathrm{d}\mathbf{w}$

Probabilistic re-formulation

- Training minimizing loss:

$$\arg\min_{\mathbf{w}} \underbrace{\frac{1}{2}\sum_{i=1}^{N}(f_{\text{net}}(\mathbf{w}, x_i) - y_i)^2}_{\text{fit}} + \underbrace{\lambda \sum_i \parallel \mathbf{w}_i \parallel}_{\text{regularizer}}$$

- Equivalent probabilistic view for regression, maximizing posterior probability:

$$\arg\max_{\mathbf{w}} \underbrace{\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})}_{\text{fit}} + \underbrace{\log p(\mathbf{w})}_{\text{regularizer}}$$

  where $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \sim \mathcal{N}$ and $p(\mathbf{w}) \sim$ Laplace

- For full Bayesian inference we also need $\int p(\mathbf{y}|\mathbf{x}, \mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}$

Probabilistic re-formulation

- Training minimizing loss:

$$\arg\min_{\mathbf{w}} \underbrace{\frac{1}{2}\sum_{i=1}^{N}(f_{\text{net}}(\mathbf{w}, x_i) - y_i)^2}_{\text{fit}} + \underbrace{\lambda \sum_i \parallel \mathbf{w}_i \parallel}_{\text{regularizer}}$$

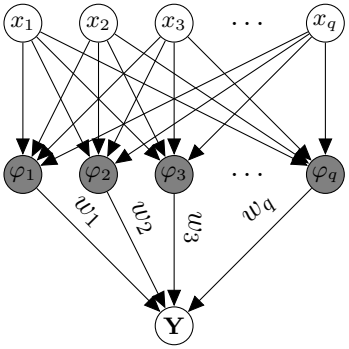
- Equivalent probabilistic view for regression, maximizing posterior probability:

$$\arg\max_{\mathbf{w}} \underbrace{\log p(\mathbf{y}|\mathbf{x}, \mathbf{w})}_{\text{fit}} + \underbrace{\log p(\mathbf{w})}_{\text{regularizer}}$$

  where $p(\mathbf{y}|\mathbf{x}, \mathbf{w}) \sim \mathcal{N}$ and $p(\mathbf{w}) \sim$ Laplace

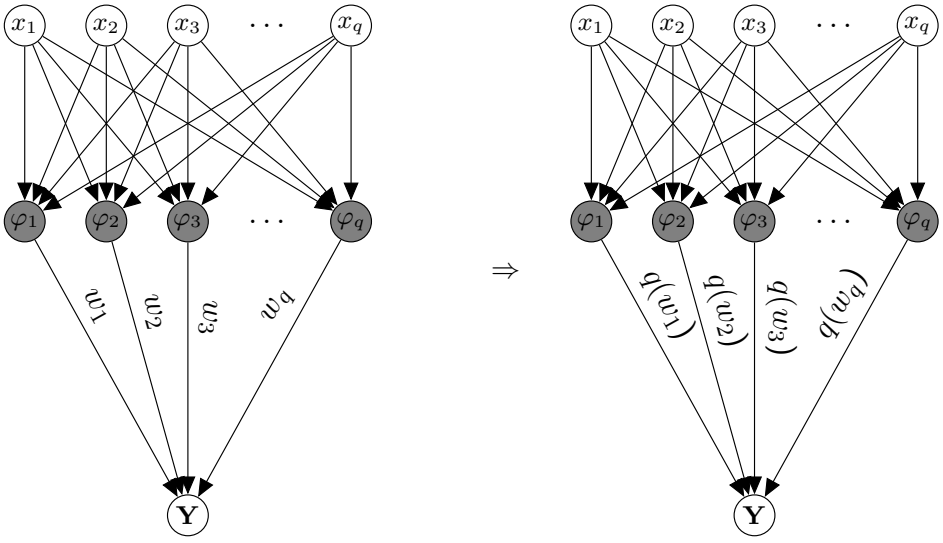
- For full Bayesian inference we also need $\int p(\mathbf{y}|\mathbf{x}, \mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}$

A standard neural network

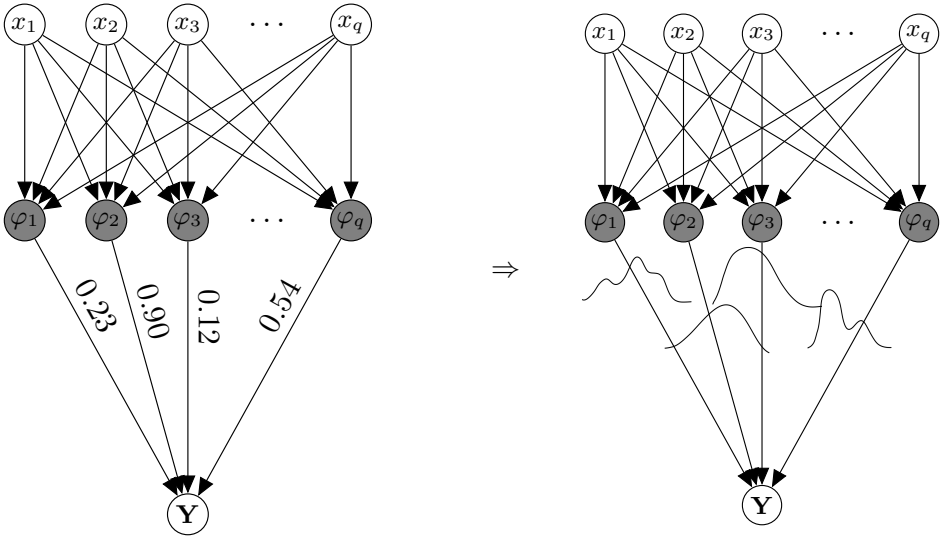


Once we've defined all w's with back-prop, then f (and the whole network) becomes deterministic.

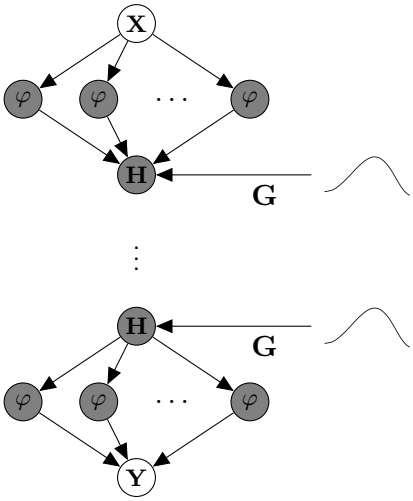
BNN with priors on its weights



BNN with priors on its weights



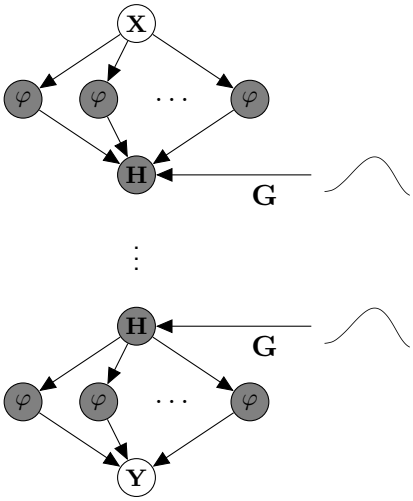
Alternative view: hierarchy of random variables



Inference:

- Consider intermediate *random* variables $\mathbf{H}$
- Called *latent variables*
- All uncertainty summarized in $\mathbf{H}_l$ and propagated across layers

Alternative view: hierarchy of random variables



Inference:

- Consider intermediate *random* variables $\mathbf{H}$
- Called *latent variables*
- All uncertainty summarized in $\mathbf{H}_l$ and propagated across layers

Re-cap so far

- Bayesian deep neural network marginalize out weights and find $p(w|D)$.
- Latent variable view considers a stack of random variables with parametric (neural network-style) mappings.

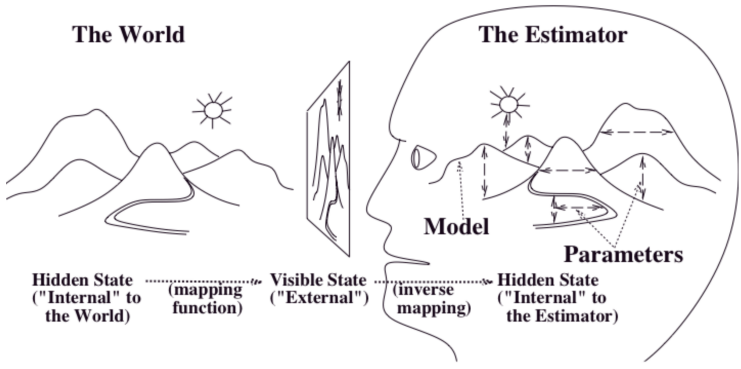
BNN properties

- Generalization: A BNN understands (and avoids) complexity (and therefore overfitting).
- Uncertainty: A BNN "knows" what it doesn't know by propagating uncertainty.
- Data generation: A model which generalizes well, should also understand -or even be able to create "imagine")- variations.

Need for uncertainty

- Reinforcement learning
- Critical predictive systems
- Active learning
- Semi-automatic systems
- Scarce data scenarios
- ...

Generative models: learn by synthesis



Observations: D Interpretation: $\mathbf{H}$

Inference: $p(\mathbf{H}|D) = \frac{p(D|\mathbf{H})p(\mathbf{H})}{p(D)}$

Credit for slide: Iasonas Kokkinos, George Papandreou

Generation

- Learn latent space from actor
- Generate by condition on new observation

Show video..

int8.io/variational-autoencoder-in-tensorflow/

What's the catch?

- Remember Bayes rule:

$$p(\mathbf{w}|D) = \frac{p(D|\mathbf{w})p(\mathbf{w})}{p(D) = \int p(D|\mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}}$$

- $\int p(D|\mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}$ is typically very difficult to compute. Hence we resort to (expensive) approximations.

What's the catch?

- Remember Bayes rule:

$$p(\mathbf{w}|D) = \frac{p(D|\mathbf{w})p(\mathbf{w})}{p(D) = \int p(D|\mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}}$$

- $\int p(D|\mathbf{w})p(\mathbf{w})\mathrm{d}\mathbf{w}$ is typically very difficult to compute. Hence we resort to (expensive) approximations.

Black-box VI (github.com/blei-lab/edward)

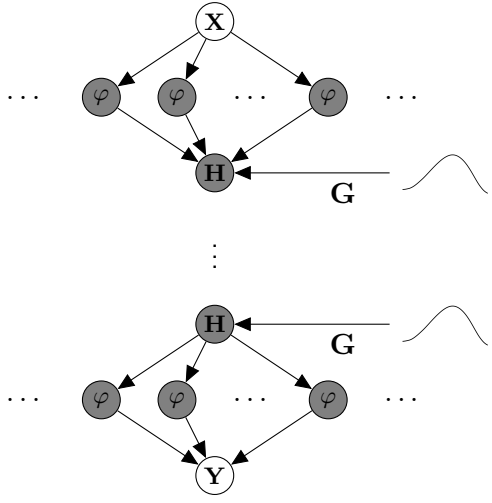
```
47  # MODEL
48  W_0 = Normal(loc=tf.zeros([D, 10]), scale=tf.ones([D, 10]))
49  W_1 = Normal(loc=tf.zeros([10, 10]), scale=tf.ones([10, 10]))
50  W_2 = Normal(loc=tf.zeros([10, 1]), scale=tf.ones([10, 1]))
51  b_0 = Normal(loc=tf.zeros(10), scale=tf.ones(10))
52  b_1 = Normal(loc=tf.zeros(10), scale=tf.ones(10))
53  b_2 = Normal(loc=tf.zeros(1), scale=tf.ones(1))
54
55  X = tf.placeholder(tf.float32, [N, D])
56  y = Normal(loc=neural_network(X), scale=0.1 * tf.ones(N))
57
58  # INFERENCE
59  qW_0 = Normal(loc=tf.Variable(tf.random_normal([D, 10])),
60                scale=tf.nn.softplus(tf.Variable(tf.random_normal([D, 10]))))
```

```
69  qb_2 = Normal(loc=tf.Variable(tf.random_normal([1])),
70                scale=tf.nn.softplus(tf.Variable(tf.random_normal([1]))))
71
72  inference = ed.KLqp({W_0: qW_0, b_0: qb_0,
73                       W_1: qW_1, b_1: qb_1,
74                       W_2: qW_2, b_2: qb_2}, data={X: X_train, y: y_train})
75  inference.run()
```

Re-cap so far

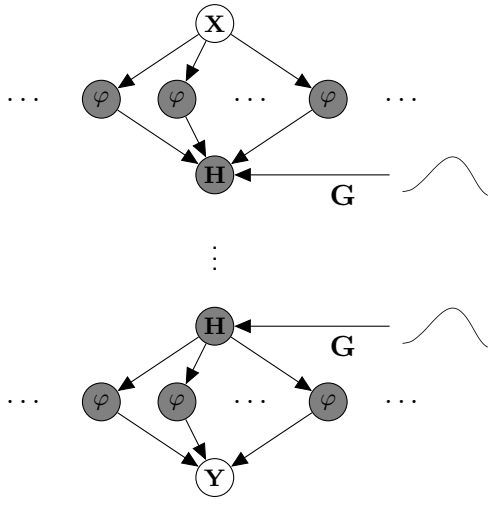
- Bayesian deep neural network marginalize out weights and find $p(w|D)$.
- Latent variable view considers a stack of random variables with parametric (neural network-style) mappings.
- Generative models possible in the Bayesian setting.
- Bayesian inference is challenging, but tools are being developed.

From NN to GP



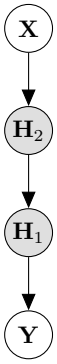
- NN: $\mathbf{H}_2 = \mathbf{W}_2\phi(\mathbf{H}_1)$
- GP: ϕ is $\infty-$dimensional so:
  $\mathbf{H}_2 = f_2(\mathbf{H}_1; \theta_2) + \epsilon$
- NN: $p(\mathbf{W})$
- GP: $p(f(\cdot))$

From NN to GP



- NN: $\mathbf{H}_2 = \mathbf{W}_2\phi(\mathbf{H}_1)$
- GP: ϕ is $\infty-$dimensional so:
  $\mathbf{H}_2 = f_2(\mathbf{H}_1; \theta_2) + \epsilon$
- NN: $p(\mathbf{W})$
- GP: $p(f(\cdot))$

A simpler representation



- $p(\mathbf{Y}|\mathbf{X}) = \int_{\mathbf{H}_1,\mathbf{H}_2} p(\mathbf{Y}|\mathbf{H}_1)p(\mathbf{H}_1|\mathbf{H}_2)p(\mathbf{H}_2|\mathbf{X})$
- There are no weights; just functions, data and *hyper*-parameters.

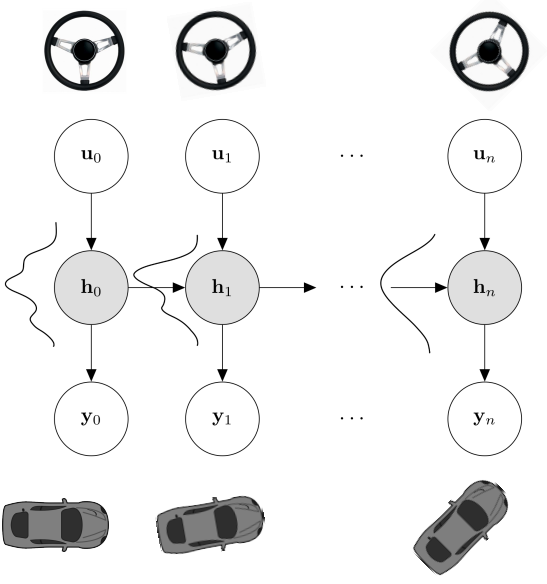
A. Damianou, PhD thesis, 2015

Re-cap so far

- Bayesian deep neural network marginalize out weights and find $p(w|D)$.
- Latent variable view considers a stack of random variables with parametric (neural network-style) mappings.
- Generative models possible in the Bayesian setting.
- Bayesian inference is challenging, but tools are being developed.
- Deep GPs consider a stack of random variables with functional non-parametric mappings.

An example where uncertainty propagation matters: Recurrent learning.

Dynamics/memory: Deep Recurrent Gaussian Process



Avatar control

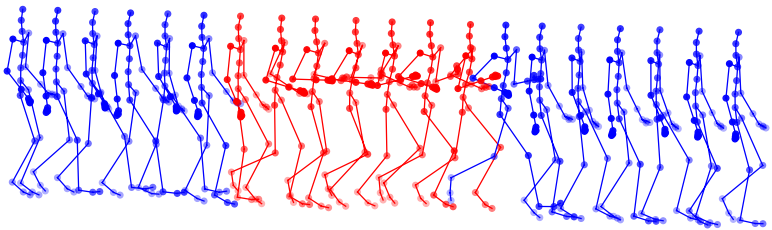


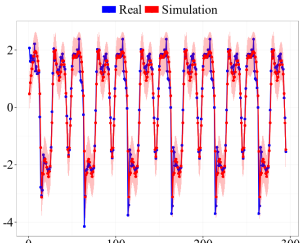
Figure: The generated motion with a step function signal, starting with walking (blue), switching to running (red) and switching back to walking (blue).

Videos:

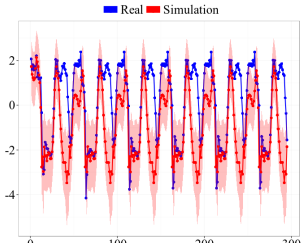
https://youtu.be/FR-oeGxV6yY *Switching between learned speeds*

https://youtu.be/AT0HMtoPgjc *Interpolating (un)seen speed*

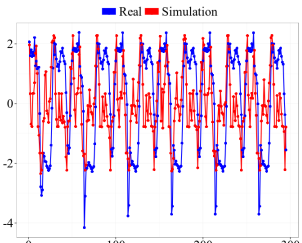
https://youtu.be/FuF-uZ83VMw Constant unseen speed



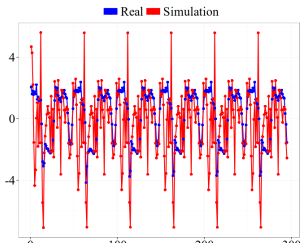
RGP



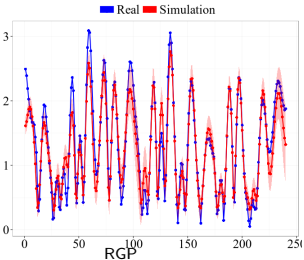
GPNARX

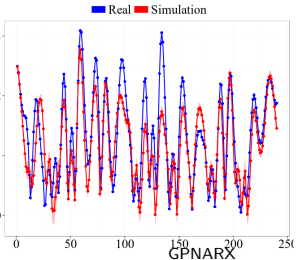


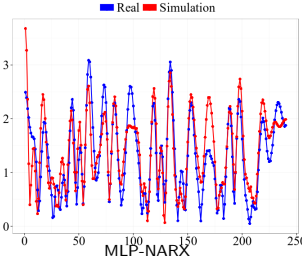
MLP-NARX

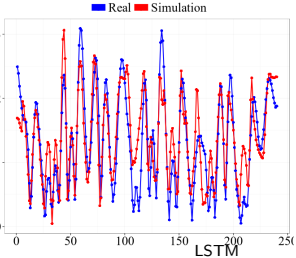


LSTM


Real
Simulation
RGP


Real
Simulation
GPNARX


Real
Simulation
MLP-NARX


Real
Simulation
LSTM

Summary

- DNNs are powerful but estimating their many parameters causes problems
- Bayesian NNs marginalize out the parameters
- Deep GPs work directly in the function space
- Calculus for gradients vs Bayes rule for posteriors
- No method is universally the best. Multiple considerations - ask me :-)