Probabilistic Reasoning in Deep Learning

Dr Konstantina Palla, PhD
palla@stats.ox.ac.uk

September 2017

Deep Learning Indaba, Johannesburgh

OVERVIEW OF THE TALK

- Basics of Bayesian Inference

Overview of the Talk

- Basics of Bayesian Inference
- Bayesian reasoning inside DNNs.

Overview of the Talk

- Basics of Bayesian Inference
- Bayesian reasoning inside DNNs.
  - The Bayesian paradigm as an approach to account for uncetainty in the DNNs.

OVERVIEW OF THE TALK

- Basics of Bayesian Inference
- Bayesian reasoning inside DNNs.
  - > The Bayesian paradigm as an approach to account for uncetainty in the DNNs.
- Connection to other statistical models (nonparametric models).

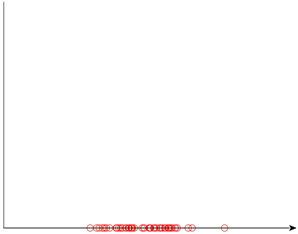
OVERVIEW OF THE TALK

- Basics of Bayesian Inference
- Bayesian reasoning inside DNNs.
  - > The Bayesian paradigm as an approach to account for uncetainty in the DNNs.
- Connection to other statistical models (nonparametric models).
  - > How kernels and Gaussian Processes fit in the picture of NNs.

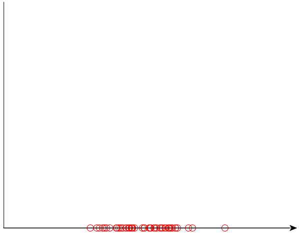
Bayesian Inference: Basics

Problem: How was it **generated**?

Data: Observations
$\mathbf{Y} = y^{(1)}, y^{(2)}, \ldots, y^{(N)}$



Data: Observations
$\mathbf{Y} = y^{(1)}, y^{(2)}, \ldots, y^{(N)}$



Problem: How was it **generated**?

Answer: Make assumptions $\rightarrow$ distributional

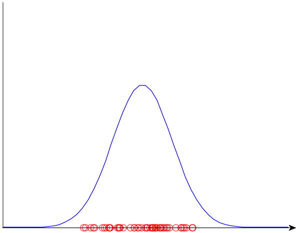
θ

$\downarrow \quad y^{(i)} \,|\theta \sim p_\theta(y|\theta)$

y

BAYESIAN INFERENCE: BASICS

Data: Observations
$\mathbf{Y} = y^{(1)}, y^{(2)}, \ldots, y^{(N)}$



$$y^{(i)} | \boldsymbol{\theta} \sim \mathcal{N}(y; \mu, \sigma^2)$$
$$\boldsymbol{\theta} = \{\mu, \sigma^2\}$$

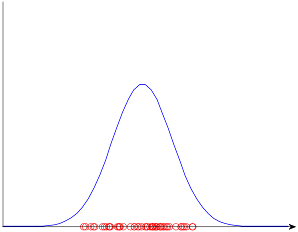
Problem: How was it **generated**?

Answer: Make assumptions $\rightarrow$ distributional

$$y^{(i)} \, |\theta \sim p_\theta(y|\theta)$$

BAYESIAN INFERENCE: BASICS

Data: Observations
$\mathbf{Y} = y^{(1)}, y^{(2)}, \ldots, y^{(N)}$



$$y^{(i)}|\boldsymbol{\theta} \sim \mathcal{N}(y; \mu, \sigma^2)$$
$$\boldsymbol{\theta} = \{\mu, \sigma^2\}$$

Problem: How was it **generated**?

Answer: Make assumptions $\rightarrow$ distributional

$$y^{(i)} \,|\theta \sim p_\theta(y|\theta)$$

likelihood: the probability of the data under the model parameters θ

$$p(\mathbf{Y}|\boldsymbol{\theta}) = \prod_{i=1}^{N} p(y^{(i)}|\boldsymbol{\theta})$$
$$= \prod_{i=1}^{N} \mathcal{N}(y^{(i)}|\mu, \sigma^2)$$

Bayesian Principle

All forms of uncertainty should be expressed by means of probability distributions.

Prior: $\theta \sim p(\theta)$
Our prior belief before seeing the data

Posterior: Our updated belief after having seen the data

Bayes' Rule

$$p(\theta|\mathbf{Y}) = \frac{p(\mathbf{Y}|\theta)p(\theta)}{p(\mathbf{Y})}$$

Bayesian Inference: Basics

Baye's rule

$$p(\theta \mid \mathbf{Y}) = \frac{p(\mathbf{Y} \mid \theta) p(\theta)}{p(\mathbf{Y})}$$

Baye's rule

posterior

$$p(\theta \mid \mathbf{Y}) = \frac{p(\mathbf{Y} \mid \theta)p(\theta)}{p(\mathbf{Y})}$$

The posterior is proportional to **likelihood** $\times$ **prior**

Bayesian Inference: Basics

Baye's rule

likelihood

$$p(\theta \mid \mathbf{Y}) = \frac{p(\mathbf{Y} \mid \theta)p(\theta)}{p(\mathbf{Y})}$$

posterior

The posterior is proportional to **likelihood** $\times$ **prior**

BAYESIAN INFERENCE: BASICS

likelihood

Baye's rule

prior

$$p(\theta \mid \mathbf{Y}) = \frac{p(\mathbf{Y} \mid \theta)p(\theta)}{p(\mathbf{Y})}$$

posterior

The posterior is proportional to **likelihood** $\times$ **prior**

Bayesian Inference: Basics

Baye's rule

likelihood

prior

$$p(\theta \mid \mathbf{Y}) = \frac{p(\mathbf{Y} \mid \theta)p(\theta)}{p(\mathbf{Y})}$$

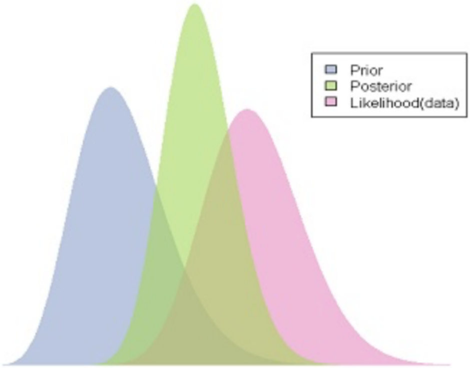
posterior

The posterior is proportional to **likelihood** $\times$ **prior**

Update my **prior beliefs** after I see the **data**

Bayesian Inference: Basics

$$P(\theta|\mathbf{Y}) \propto P(\mathbf{Y}|\theta)P(\theta)$$



Make predictions

For unseen y^* marginalise!

$$P(y^*|\mathbf{Y}) = \int P(y^*|\theta)P(\theta|\mathbf{Y})\mathrm{d}\theta$$

Make predictions

For unseen y^* marginalise!

$$P(y^*|\mathbf{Y}) = \int P(y^*|\theta)P(\theta|\mathbf{Y})\mathrm{d}\theta$$

The prediction does not depend on a point estimate of θ but it is expressed as a weighted average over all the possible values of θ

Bayesian Inference: Basics

Why is this useful?

> Uncertainty is taken into account

No point estimates BUT distribution over the unknown θ; $p(\theta|\mathbf{Y})$

Why is this useful?

> Uncertainty is taken into account

No point estimates BUT distribution over the unknown θ; $p(\theta|\mathbf{Y})$

> Natural handling of missing data

- a probability distribution is estimated for each missing value

$$\theta \quad ?$$

$$\theta \sim p(\theta|\mathbf{Y})$$

Bayesian Inference: Basics

Why is this useful?

> Uncertainty is taken into account

No point estimates BUT distribution over the unknown θ; $p(\theta|\mathbf{Y})$

> Natural handling of missing data

- a probability distribution is estimated for each missing value

θ ?

$$\theta \sim p(\theta|\mathbf{Y})$$

> Adresses issues like regularization, overfitting $\rightarrow$ useful in Deep Neural Networks.

Part II

Bayesian Reasoning in (Deep) Neural Networks

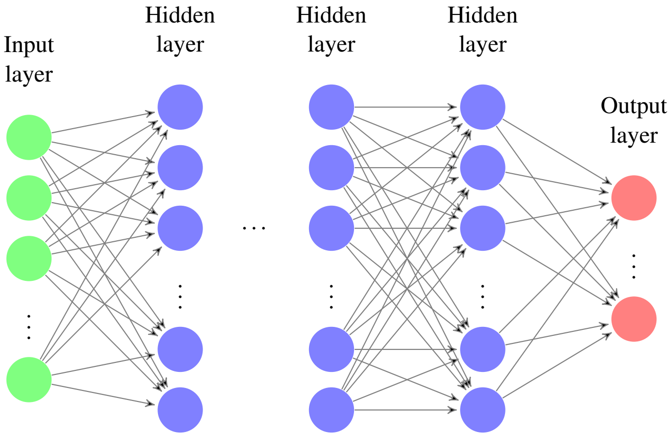
PART II

Bayesian Reasoning in (Deep) Neural Networks

Overview of (D)NNs

Bayesian Reasoning in (Deep) Neural Networks

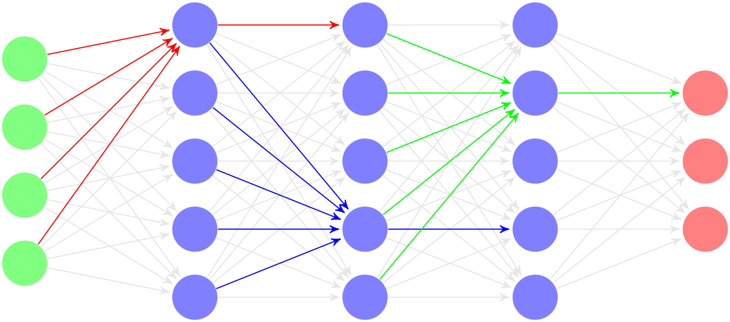
Overview of (Deep) Neural Networks



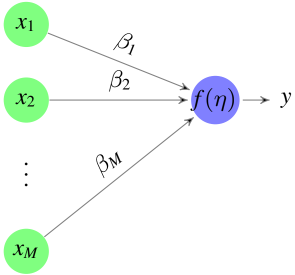
What is Deep Learning

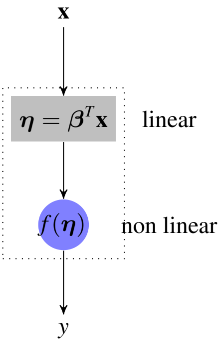
A framework for constructing *flexible* models → a way of constructing outputs using functions on inputs.

Repetition of building blocks



Building block



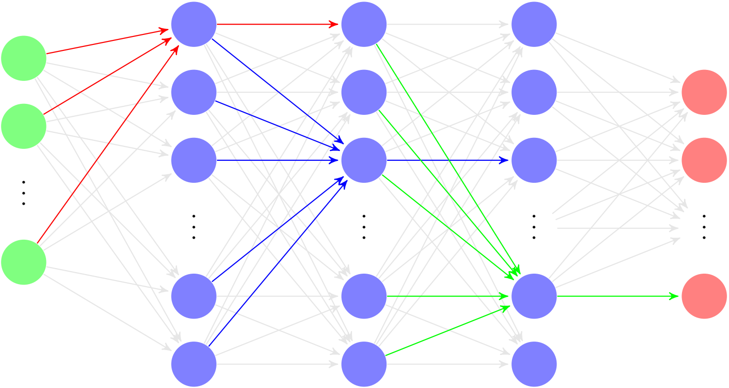


> Output: y

> Linear predictor : $\boldsymbol{\eta} = \sum_{m=1}^{M} \beta_m x_m = \boldsymbol{\beta}^T \mathbf{x}$, coefficients $\boldsymbol{\beta}$ weights

> Inverse Link function: $f(\boldsymbol{\eta})$ activation function

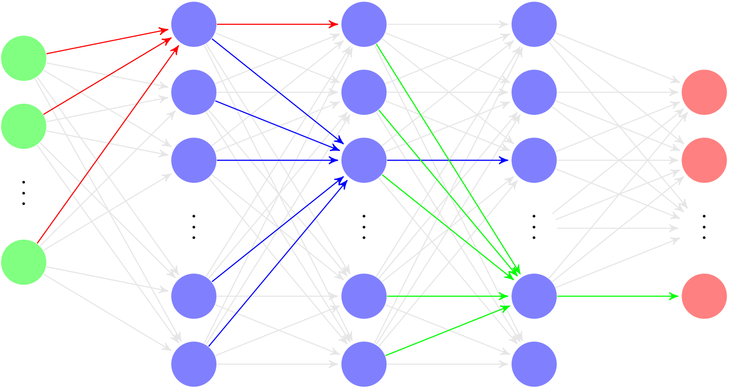
Overview of Deep NNs



In each layer l

the input is linearly composed $\rightarrow$ $\boldsymbol{\eta}_l = \boldsymbol{\beta}_l^T \mathbf{x}_l$

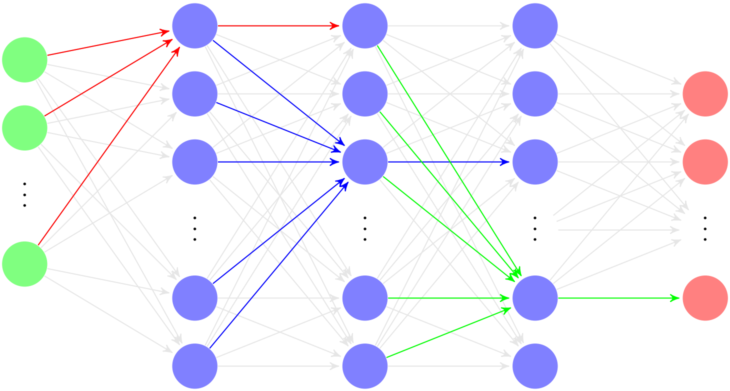
Overview of Deep NNs



In each layer l

the input is linearly composed$\rightarrow$ $\boldsymbol{\eta}_l = \boldsymbol{\beta}_l^T \mathbf{x}_l$

a non-linear function is applied $\rightarrow f_l(\boldsymbol{\eta}_l)$



In each layer l

the input is linearly composed $\rightarrow \boldsymbol{\eta}_l = \boldsymbol{\beta}_l^T \mathbf{x}_l$

a non-linear function is applied $\rightarrow f_l(\boldsymbol{\eta}_l)$

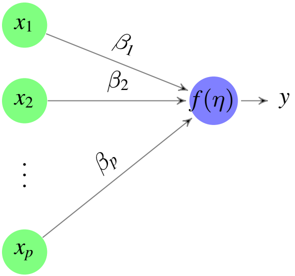
A Deep Neural Network $\rightarrow$ construction of recursive building blocks

Overview of Deep NNs

$$\mathbf{x} \longrightarrow \boldsymbol{\eta} = \boldsymbol{\beta}^T \mathbf{x} \longrightarrow f(\boldsymbol{\eta}) \longrightarrow \cdots \longrightarrow \boldsymbol{\eta}_l = \boldsymbol{\beta}_l^T \mathbf{x}_l \longrightarrow f_l(\boldsymbol{\eta}_l) \longrightarrow y$$

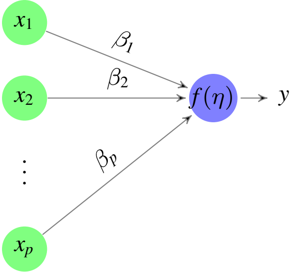
A Deep Neural Network can represent functions of increasing complexity.

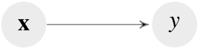
DEEP NNs AS A PROBABILISTIC MODEL



$$\eta = \boldsymbol{\beta}^T \mathbf{x}$$

Deep NNs as a probabilistic model



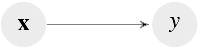


$$\boxed{p(y|\mathbf{x}; \boldsymbol{\beta})}$$

Parametric!

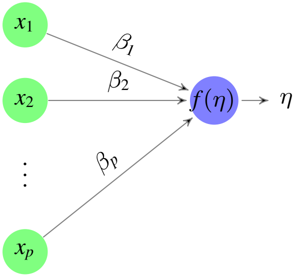
$$\eta = \boldsymbol{\beta}^T \mathbf{x}$$

Deep NNs as a probabilistic model



$p(y|\mathbf{x}; \boldsymbol{\beta})$

Parametric!



$$\eta = \boldsymbol{\beta}^T \mathbf{x}$$

Example: Regression

$$f(\eta) = \eta = \boldsymbol{\beta}^T \mathbf{x}$$

$$y = \boldsymbol{\beta}^T \mathbf{x} + \epsilon, \ \epsilon \sim \mathcal{N}(0, \sigma^2)$$

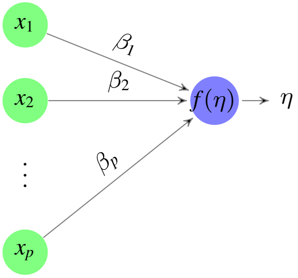
$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^T \mathbf{x}, \sigma^2)$$

Deep NNs as a probabilistic model

$\mathbf{x} \longrightarrow y$

$$\boxed{p(y|\mathbf{x}; \boldsymbol{\beta})}$$

Parametric!



$$\eta = \boldsymbol{\beta}^T \mathbf{x}$$

Example: Regression

$$f(\eta) = \eta = \boldsymbol{\beta}^T \mathbf{x}$$

$$y = \boldsymbol{\beta}^T \mathbf{x} + \epsilon, \ \epsilon \sim \mathcal{N}(0, \sigma^2)$$

$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^T \mathbf{x}, \sigma^2)$$

Train the network: learn $\boldsymbol{\beta}$?

Learn $\boldsymbol{\beta}$

Maximum Likelihood Principal

Minimize the negative log-likelihood $\mathcal{J}(\boldsymbol{\beta}) = -\log(p(\mathbf{Y}|\mathbf{x}, \boldsymbol{\beta}))$

- β_{MLE} : $\quad \arg\min_{\beta} J(\beta)$

Learn β

Maximum Likelihood Principal

Minimize the negative log-likelihood $\mathcal{J}(\boldsymbol{\beta}) = -\log(p(\mathbf{Y}|\mathbf{x}, \boldsymbol{\beta}))$

- β_{MLE} : $\arg\min_{\beta} J(\beta)$

Regression

$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^T \mathbf{x}, \sigma^2)$$

DEEP NNS AS A PROBABILISTIC MODEL

Learn β

Maximum Likelihood Principal

Minimize the negative log-likelihood $\mathcal{J}(\boldsymbol{\beta}) = -\log(p(\mathbf{Y}|\mathbf{x}, \boldsymbol{\beta}))$

- β_{MLE} : $\quad \underset{\beta}{\arg\min}\, J(\beta)$

Regression

Squared Loss function

$$J(\boldsymbol{\beta}) = \frac{1}{2} \sum_{i=1}^{N} \{y^{(i)} - \boldsymbol{\beta}^T \mathbf{x}^{(i)}\}^2 + const.$$

$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^T \mathbf{x}, \sigma^2)$$

Learn β

Maximum Likelihood Principal

Minimize the negative log-likelihood $\mathcal{J}(\boldsymbol{\beta}) = -\log(p(\mathbf{Y}|\mathbf{x}, \boldsymbol{\beta}))$

- β_{MLE} : $\arg\min_{\beta} J(\beta)$

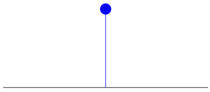
Regression

$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^T\mathbf{x}, \sigma^2)$$

Squared Loss function

$$J(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{N}\{y^{(i)} - \boldsymbol{\beta}^T\mathbf{x}^{(i)}\}^2 + const.$$

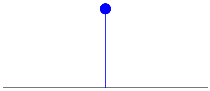
> Solve using Stachastic Gradient, back-propagation etc.



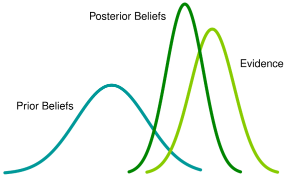
Maximum Likelihood Estimation

$\boldsymbol{\beta}$ is fixed but unknown

- Point estimates
- Overfitting



$\delta_{\boldsymbol{\beta}}$



$p(\boldsymbol{\beta})$

Maximum Likelihood Estimation

$\boldsymbol{\beta}$ is fixed but unknown

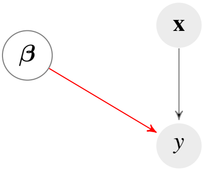
- Point estimates
- Overfitting

Bayesian Approach

$\boldsymbol{\beta}$ is unknown/uncertain

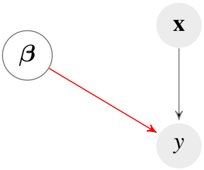
+ Account for uncertainty $p(\boldsymbol{\beta}), p(\boldsymbol{\beta}|\mathbf{y})$
+ Regularisation

Maximum a Posteriori Estimation



Maximum a Posteriori Estimation

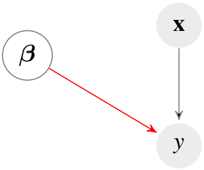
Distribution over the network weights $\boldsymbol{\beta}$



$$\boldsymbol{\beta} \sim p(\boldsymbol{\beta})$$
$$y|\mathbf{x}, \boldsymbol{\beta} \sim p(y|\mathbf{x}, \boldsymbol{\beta})$$

Maximum a Posteriori Estimation

Distribution over the network weights $\boldsymbol{\beta}$



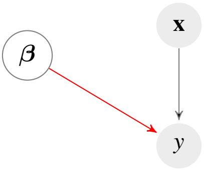
$$\boldsymbol{\beta} \sim p(\boldsymbol{\beta})$$
$$y|\mathbf{x}, \boldsymbol{\beta} \sim p(y|\mathbf{x}, \boldsymbol{\beta})$$

Update the belief over $\boldsymbol{\beta}$ $\rightarrow$ Bayes' rule

$$p(\boldsymbol{\beta}|\mathbf{Y}) \propto \overbrace{p(\mathbf{Y}|\boldsymbol{\beta}, \mathbf{X})}^{\text{Likelihood}} \; \overbrace{p(\boldsymbol{\beta})}^{\text{prior}}$$

Maximum a Posteriori Estimation

Distribution over the network weights $\boldsymbol{\beta}$



$$\boldsymbol{\beta} \sim p(\boldsymbol{\beta})$$
$$y|\mathbf{x}, \boldsymbol{\beta} \sim p(y|\mathbf{x}, \boldsymbol{\beta})$$

Update the belief over $\boldsymbol{\beta} \rightarrow$ Bayes' rule

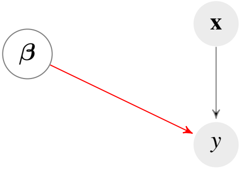
$$p(\boldsymbol{\beta}|\mathbf{Y}) \propto \overbrace{p(\mathbf{Y}|\boldsymbol{\beta}, \mathbf{X})}^{\text{Likelihood}} \; \overbrace{p(\boldsymbol{\beta})}^{\text{prior}}$$

Maximum a Posteriori

Find $\boldsymbol{\beta}_{MAP}$ that maximizes $\log p(\boldsymbol{\beta}|\mathbf{Y})$
Find $\boldsymbol{\beta}_{MAP}$ that minimizes $\mathcal{J}(\boldsymbol{\beta}) = -\log p(\boldsymbol{\beta}|\mathbf{Y})$

- β_{MAP} : $\quad \arg\min_{\beta} J(\beta)$

Regression: Maximum a Posteriori

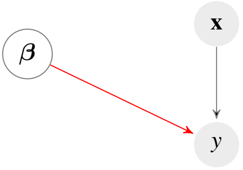


$$y = \boldsymbol{\beta}^T \mathbf{x} + \epsilon$$

$$\overbrace{p(\boldsymbol{\beta})}^{\text{prior over the weights}} = \overbrace{\mathcal{N}(\mathbf{m}_\beta, \lambda^{-1}\mathbb{I})}^{\text{multivariate Gaussian}}$$

$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^T \mathbf{x}, \sigma^2)$$

Regression: Maximum a Posteriori



Loss function: $\mathcal{J}(\boldsymbol{\beta}) = -\log p(\boldsymbol{\beta}|\mathbf{Y})$

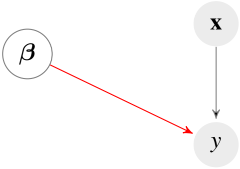
$$y = \boldsymbol{\beta}^T \mathbf{x} + \epsilon$$

$$\mathcal{J}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{N}\{y^{(i)} - \boldsymbol{\beta}^T \mathbf{x}^{(i)}\}^2 + \overbrace{\boxed{\frac{\lambda}{2}\boldsymbol{\beta}^T\boldsymbol{\beta}}}^{\text{weight decay - regularization}} + const.$$

$$\overbrace{\boxed{p(\boldsymbol{\beta})}}^{\text{prior over the weights}} = \overbrace{\boxed{\mathcal{N}(\mathbf{m}_\beta, \lambda^{-1}\mathbb{I})}}^{\text{multivariate Gaussian}}$$

$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^T \mathbf{x}, \sigma^2)$$

Regression: Maximum a Posteriori



$$y = \boldsymbol{\beta}^T \mathbf{x} + \epsilon$$

prior over the weights

multivariate Gaussian

$$\boxed{p(\boldsymbol{\beta})} = \boxed{\mathcal{N}(\mathbf{m}_\beta, \lambda^{-1}\mathbb{I})}$$

$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^T \mathbf{x}, \sigma^2)$$

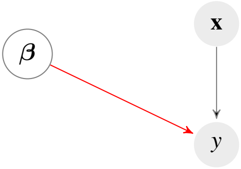
Loss function: $\mathcal{J}(\boldsymbol{\beta}) = -\log p(\boldsymbol{\beta}|\mathbf{Y})$

weight decay - regularization

$$\mathcal{J}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{N}\{y^{(i)} - \boldsymbol{\beta}^T \mathbf{x}^{(i)}\}^2 + \boxed{\frac{\lambda}{2}\boldsymbol{\beta}^T \boldsymbol{\beta}} + const.$$

The posterior distribution over the weights allows for a natural regularizer to arise! *Bayesian side effect*

Regression: Maximum a Posteriori



$$y = \boldsymbol{\beta}^T \mathbf{x} + \epsilon$$

$$\overbrace{\boxed{p(\boldsymbol{\beta})}}^{\text{prior over the weights}} = \overbrace{\boxed{\mathcal{N}(\mathbf{m}_\beta, \lambda^{-1}\mathbb{I})}}^{\text{multivariate Gaussian}}$$

$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^T \mathbf{x}, \sigma^2)$$

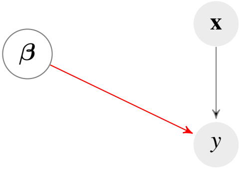
Loss function: $\mathcal{J}(\boldsymbol{\beta}) = -\log p(\boldsymbol{\beta}|\mathbf{Y})$

$$\mathcal{J}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{N}\{y^{(i)} - \boldsymbol{\beta}^T \mathbf{x}^{(i)}\}^2 + \overbrace{\boxed{\frac{\lambda}{2}\boldsymbol{\beta}^T\boldsymbol{\beta}}}^{\text{weight decay - regularization}} + const.$$

The posterior distribution over the weights allows for a natural regularizer to arise! *Bayesian side effect*

$\boldsymbol{\beta}_{MAP}$: $\quad \arg\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta})$

Regression: Maximum a Posteriori



$$y = \boldsymbol{\beta}^T \mathbf{x} + \epsilon$$

$$\underbrace{\boxed{p(\boldsymbol{\beta})}}_{\text{prior over the weights}} = \underbrace{\boxed{\mathcal{N}(\mathbf{m}_\beta, \lambda^{-1}\mathbb{I})}}_{\text{multivariate Gaussian}}$$

$$p(y|\mathbf{x}, \boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^T \mathbf{x}, \sigma^2)$$

Loss function: $\mathcal{J}(\boldsymbol{\beta}) = -\log p(\boldsymbol{\beta}|\mathbf{Y})$

$$\mathcal{J}(\boldsymbol{\beta}) = \frac{1}{2}\sum_{i=1}^{N}\{y^{(i)} - \boldsymbol{\beta}^T \mathbf{x}^{(i)}\}^2 + \overbrace{\boxed{\frac{\lambda}{2}\boldsymbol{\beta}^T\boldsymbol{\beta}}}^{\text{weight decay - regularization}} + const.$$

The posterior distribution over the weights allows for a natural regularizer to arise! *Bayesian side effect*

$\boldsymbol{\beta}_{MAP}$: $\arg\min_{\boldsymbol{\beta}} J(\boldsymbol{\beta})$

Predicitons:

$y^*|\mathbf{Y}, \mathbf{x}^* \sim \mathcal{N}(\boldsymbol{\beta}_{MAP}^T \mathbf{x},^* \sigma^2)$

What we have seen so far...

> Being Bayesian over the weights $\boldsymbol{\beta}$, i.e. putting a prior distribution, allows for a better loss function in learning. natural regularization

Bayesian Reasoning in Deep Learning

What we have seen so far...

> Being Bayesian over the weights $\boldsymbol{\beta}$, i.e. putting a prior distribution, allows for a better loss function in learning. natural regularization

> (D)NNs **don't** handle missing data

$$\mathbf{x} \xrightarrow{p(\mathbf{y}|\mathbf{x};\boldsymbol{\beta})} \mathbf{y}$$

$\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, i = 1, \dots, N$ train data

$\mathbf{y}^* : p(\mathbf{y}^*|\mathbf{x}^*; \boldsymbol{\beta})$

Bayesian Reasoning in Deep Learning

What we have seen so far...

> Being Bayesian over the weights $\boldsymbol{\beta}$, i.e. putting a prior distribution, allows for a better loss function in learning. natural regularization

> (D)NNs **don't** handle missing data

> Need for $p(\mathbf{x}); \mathbf{x} \sim p(\mathbf{x})$

$$\mathbf{x} \xrightarrow{p(\mathbf{y}|\mathbf{x};\boldsymbol{\beta})} \mathbf{y}$$

$\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, i = 1, \ldots, N$ train data

$\mathbf{y}^* : p(\mathbf{y}^*|\mathbf{x}^*; \boldsymbol{\beta})$

$\boldsymbol{x}^{(i)}\,{}^{?}, \mathbf{y}^{(i)}$

Bayesian Reasoning in Deep Learning

What we have seen so far...

> Being Bayesian over the weights $\boldsymbol{\beta}$, i.e. putting a prior distribution, allows for a better loss function in learning. natural regularization

> (D)NNs **don't** handle missing data

> Need for $p(\mathbf{x})$; $\mathbf{x} \sim p(\mathbf{x})$

> (D)NNs **don't** generate new examples

$$\mathbf{x} \xrightarrow{p(\mathbf{y}|\mathbf{x};\boldsymbol{\beta})} \mathbf{y}$$

$\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, i = 1, \ldots, N$ train data

$\mathbf{y}^* : p(\mathbf{y}^*|\mathbf{x}^*;\boldsymbol{\beta})$

$\boldsymbol{x}^{(i)}\,?, \mathbf{y}^{(i)}$

Bayesian Reasoning in Deep Learning

What we have seen so far...

> Being Bayesian over the weights $\boldsymbol{\beta}$, i.e. putting a prior distribution, allows for a better loss function in learning. natural regularization

> (D)NNs **don't** handle missing data

- > Need for $p(\mathbf{x})$; $\mathbf{x} \sim p(\mathbf{x})$

> (D)NNs **don't** generate new examples

- > $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$
- draw $\mathbf{x}^{(i)} \sim p(\mathbf{x}^{(i)})$
- draw $\mathbf{y}^{(i)} \sim p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)})$

$$\mathbf{x} \xrightarrow{p(\mathbf{y}|\mathbf{x};\boldsymbol{\beta})} \mathbf{y}$$

$\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, i = 1, \ldots, N$ train data

$\mathbf{y}^* : p(\mathbf{y}^*|\mathbf{x}^*; \boldsymbol{\beta})$

$\boldsymbol{x}^{(i)}$?, $\mathbf{y}^{(i)}$?

What we have seen so far...

> Being Bayesian over the weights $\boldsymbol{\beta}$, i.e. putting a prior distribution, allows for a better loss function in learning. natural regularization

$$\mathbf{x} \xrightarrow{p(\mathbf{y}|\mathbf{x};\boldsymbol{\beta})} \mathbf{y}$$

> (D)NNs **don't** handle missing data

- > Need for $p(\mathbf{x})$; $\mathbf{x} \sim p(\mathbf{x})$

$\mathbf{x}^{(i)}, \mathbf{y}^{(i)}, i = 1, \ldots, N$ train data

$\mathbf{y}^* : p(\mathbf{y}^*|\mathbf{x}^*; \boldsymbol{\beta})$

> (D)NNs **don't** generate new examples

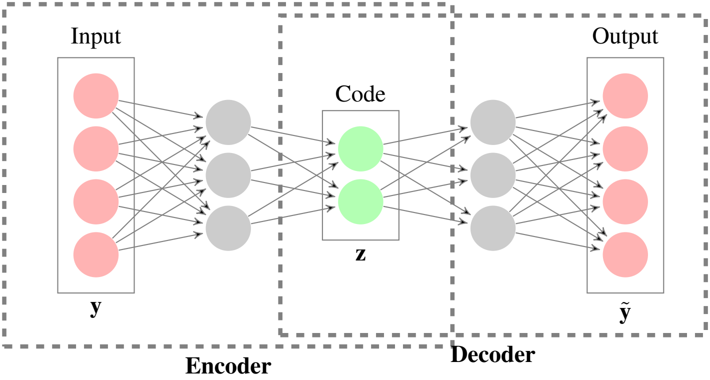
- > $p(\mathbf{x}, \mathbf{y}) = p(\mathbf{y}|\mathbf{x})p(\mathbf{x})$
- draw $\mathbf{x}^{(i)} \sim p(\mathbf{x}^{(i)})$
- draw $\mathbf{y}^{(i)} \sim p(\mathbf{y}^{(i)}|\mathbf{x}^{(i)})$

$\boldsymbol{x}^{(i)}$?, $\mathbf{y}^{(i)}$?

Need for a fully probabilstic model

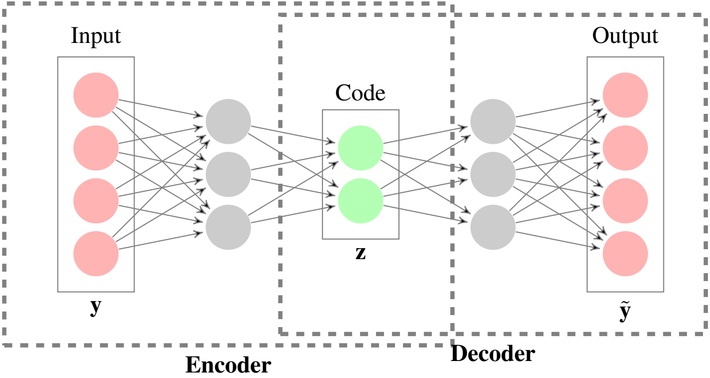
Autoencoder as a fully probabilistic Neural Network

Autoencoder



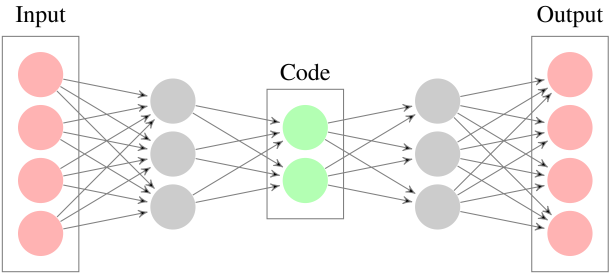
- Two parts:
  - Encoder: "encodes" the input to hidden code (representation)
  - Decoder: "decodes" the input back
- Minimize reconstruction error $\mathcal{L} = \|y - \tilde{y}\|_2^2$

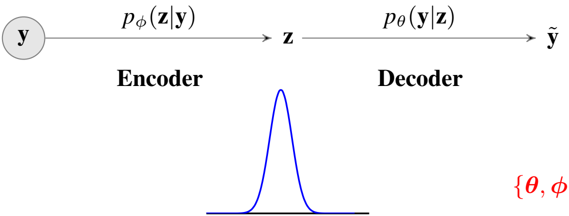
Autoencoder



Unsupervised method: Experiences data $\mathbf{y}$ and attempts to learn the distribution over $\mathbf{y}$, i.e. $p(\mathbf{y}|\mathbf{z})$

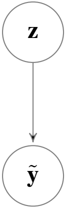
Bayesian Reasoning in Deep Learning





$\{\boldsymbol{\theta}, \boldsymbol{\phi}\}$ NN's weights

Decoder $\rightarrow$ Generative Model



Latent variable model

$$\mathbf{z} \sim p(\mathbf{z})$$
$$\tilde{\mathbf{y}} \sim p_\theta(\mathbf{y}|\mathbf{z})$$

Bayesian Reasoning in Deep Learning

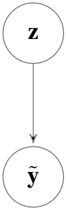
Encoder $\rightarrow$ Inference Mechanism

Learn $p_\phi(\mathbf{z}|\mathbf{y}) \rightarrow \phi$

Learn θ

Decoder $\rightarrow$ Generative Model

Latent variable model



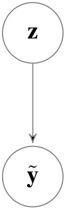
$\mathbf{z} \sim p(\mathbf{z})$

$\tilde{\mathbf{y}} \sim p_\theta(\mathbf{y}|\mathbf{z})$

BAYESIAN REASONING IN DEEP LEARNING

Encoder $\rightarrow$ Inference Mechanism

Decoder $\rightarrow$ Generative Model



Latent variable model

Learn $p_\phi(\mathbf{z}|\mathbf{y}) \rightarrow \phi$

Learn θ

$\mathbf{z} \sim p(\mathbf{z})$

$\tilde{\mathbf{y}} \sim p_\theta(\mathbf{y}|\mathbf{z})$

Inference: Learn ϕ, θ weights!

Variational Inference

$$p_\phi(\mathbf{z}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{y})^{?}}$$

Cannot sample from $p_\phi(\mathbf{z}|\mathbf{y}) \rightarrow$ cannot construct it using the NNs structure (weights ϕ)

Variational Inference

Main Principle

approximate distribution
$q_\phi(\mathbf{z}) \approx p(\mathbf{z}|\mathbf{y})$

$$p_\phi(\mathbf{z}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{y})^{?}}$$

Cannot sample from $p_\phi(\mathbf{z}|\mathbf{y}) \rightarrow$ cannot construct it using the NNs structure (weights ϕ)

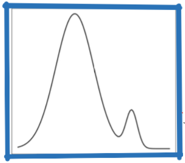
Variational Inference

Main Principle

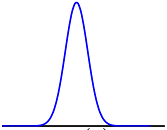
approximate distribution
$q_\phi(\mathbf{z}) \approx p(\mathbf{z}|\mathbf{y})$

$$p_\phi(\mathbf{z}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{y})}?$$

Cannot sample from $p_\phi(\mathbf{z}|\mathbf{y}) \rightarrow$ cannot construct it using the NNs structure (weights ϕ)

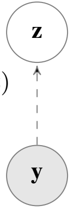


True posterior



$q_\phi(\mathbf{z})$

$\mathbf{z} \sim q_\phi(\mathbf{z})$



Variational Inference

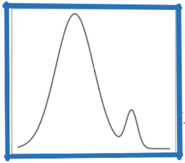
Main Principle

approximate distribution
$q_\phi(\mathbf{z}) \approx p(\mathbf{z}|\mathbf{y})$

$$p_\phi(\mathbf{z}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{z})p(\mathbf{z})}{p(\mathbf{y})}?$$

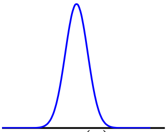
Cannot sample from $p_\phi(\mathbf{z}|\mathbf{y}) \rightarrow$ cannot construct it using the NNs structure (weights ϕ)

KL measures the information lost when using q to approximate p



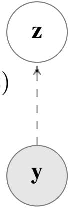
True posterior

$KL[q_\phi(\mathbf{z})||p(\mathbf{z}|\mathbf{y})]$



$q_\phi(\mathbf{z})$

$\mathbf{z} \sim q_\phi(\mathbf{z})$



Variational Inference

Cost function

A metric for how well $\{\phi, \theta\}$ capture the data generating $p_\theta(\mathbf{y}|\mathbf{z})$ & latent distributions $p_\phi(\mathbf{z}|\mathbf{y})$

Minimize
$\mathcal{L} = -\log p(\mathbf{y}|\theta, \phi)$

Variational Inference

Cost function

A metric for how well $\{\phi, \theta\}$ capture the data generating $p_\theta(\mathbf{y}|\mathbf{z})$ & latent distributions $p_\phi(\mathbf{z}|\mathbf{y})$

Minimize
$$\mathcal{L} = -\log p(\mathbf{y}|\theta, \phi)$$

Reconstruction cost:
Expected log-likelihood measures how well samples from $q_\phi(\mathbf{z})$ are able to explain the data $\mathbf{y}$.

Penalty:
This divergence measures how much information is lost (in units of nats) when using $q_\phi(\mathbf{z})$ to represent $p(\mathbf{z}|\mathbf{y})$

Minimize $\mathcal{F}$

$$\mathcal{F}(\phi, \theta; \mathbf{y}) = \underbrace{-\mathbb{E}_{q_\phi(\mathbf{z}|y)}[\log p_\theta(\mathbf{y}|\mathbf{z})]}_{\text{Reconstruction}} + \underbrace{KL[q_\phi(\mathbf{z})||p(\mathbf{z}|\mathbf{y})]}_{\text{Penalty}}$$

Variational Inference

Cost function

A metric for how well $\{\phi, \theta\}$ capture the data generating $p_\theta(\mathbf{y}|\mathbf{z})$ & latent distributions $p_\phi(\mathbf{z}|\mathbf{y})$

Reconstruction cost:
Expected log-likelihood measures how well samples from $q_\phi(\mathbf{z})$ are able to explain the data $\mathbf{y}$.

Penalty:
This divergence measures how much information is lost (in units of nats) when using $q_\phi(\mathbf{z})$ to represent $p(\mathbf{z}|\mathbf{y})$

Minimize $\mathcal{F}$

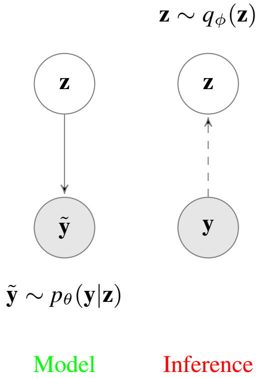
$$\mathcal{F}(\phi, \theta; \mathbf{y}) = \underbrace{-\mathbb{E}_{q_\phi(\mathbf{z}|y)}[\log p_\theta(\mathbf{y}|\mathbf{z})]}_{\text{Reconstruction}} + \underbrace{KL[q_\phi(\mathbf{z})||p(\mathbf{z}|\mathbf{y})]}_{\text{Penalty}}$$

Penalty is derived from your model and does not need to be designed

Bayesian & DL marriage - What have we gained?

- ++ Principled inference approach - Bayesian

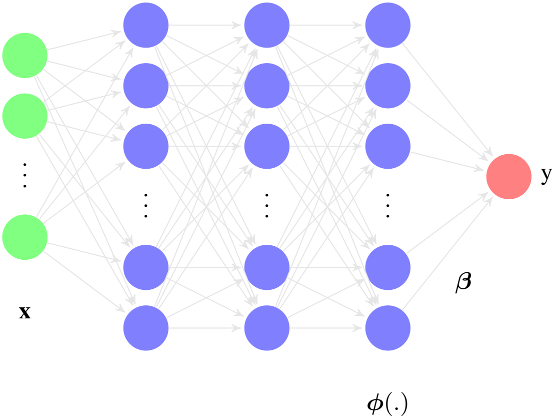
  Penalty as intrinsic part of the model!
- ++ Encode uncertainty
- ++ Impute missing data



Deep Learning as a kernel method

- So far... Predictions made using point estimates $\boldsymbol{\beta}$ (weights) parametric
- Now... Store the entire training set in order to make predictions $\rightarrow$ *kernel approach* non-parametric

DEEP LEARNING AS A KERNEL METHOD



Regression example

$$y = \boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x}) = \sum_{m=1}^{M} \beta_m \phi_m(\mathbf{x})$$

Deep Learning as a Kernel Method

Parametric approach

$$\boxed{J(\boldsymbol{\beta}) = \tfrac{1}{2}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}^2 + \tfrac{\lambda}{2}\boldsymbol{\beta}^T\boldsymbol{\beta}}$$

> MLE training $\rightarrow$ point estimates of $\boldsymbol{\beta}$

$$\nabla J(\boldsymbol{\beta}) = 0 \rightarrow \boldsymbol{\beta}_{\mathrm{MLE}} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}\phi(\mathbf{x})_n$$

> Predictions:

$$y^* = \boldsymbol{\beta}_{\mathrm{MLE}}^T\boldsymbol{\phi}(\mathbf{x}^*)$$

Predictions purely based on the learned parameter- all the information from the data is transferred into a set of parameters

Deep Learning as a Kernel Method

Parametric approach

$$\boxed{J(\boldsymbol{\beta}) = \frac{1}{2}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}^2 + \frac{\lambda}{2}\boldsymbol{\beta}^T\boldsymbol{\beta}}$$

> MLE training $\rightarrow$ point estimates of $\boldsymbol{\beta}$

$$\nabla J(\boldsymbol{\beta}) = 0 \rightarrow \boldsymbol{\beta}_{\mathrm{MLE}} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}\phi(\mathbf{x})_n$$

> Predictions:

$$y^* = \boldsymbol{\beta}_{\mathrm{MLE}}^T\boldsymbol{\phi}(\mathbf{x}^*)$$

Predictions purely based on the learned parameter- all the information from the data is transferred into a set of parameters

parametric!

Change of scenery...let the data speak

$$\boxed{J(\boldsymbol{\beta}) = \tfrac{1}{2}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}^2 + \tfrac{\lambda}{2}\boldsymbol{\beta}^T\boldsymbol{\beta}}$$

$$\nabla J(\boldsymbol{\beta}) = 0 \rightarrow$$

$$\boldsymbol{\beta}_{\text{MLE}} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}\phi(\mathbf{x}_n)$$

$$= \sum_{n=1}^{N}\alpha_n\boldsymbol{\phi}(\mathbf{x}_n) = \underbrace{\boldsymbol{\Phi}^T}_{\substack{\text{design matrix}\\ M\times N}} \underbrace{\boldsymbol{\alpha}}_{N\times 1}$$

$$\alpha_n = -\frac{1}{\lambda}(\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n)$$

Change of scenery...let the data speak

$$\boxed{J(\boldsymbol{\beta}) = \frac{1}{2}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}^2 + \frac{\lambda}{2}\boldsymbol{\beta}^T\boldsymbol{\beta}}$$

dual representation

$$\nabla J(\boldsymbol{\beta}) = 0 \rightarrow$$

$$\boldsymbol{\beta}_{\text{MLE}} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}\phi(\mathbf{x}_n)$$

$$= \sum_{n=1}^{N}\alpha_n\boldsymbol{\phi}(\mathbf{x}_n) = \underbrace{\boldsymbol{\Phi}^T}_{\substack{\text{design matrix} \\ M \times N}} \underbrace{\boldsymbol{\alpha}}_{N \times 1}$$

$$\alpha_n = -\frac{1}{\lambda}(\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n)$$

$$\nabla J(\boldsymbol{\alpha}) = 0 \rightarrow$$

$$\boldsymbol{\alpha} = (\underbrace{\mathbf{K}}_{\substack{\text{Gram matrix} \\ N \times N}} + \lambda\mathbf{I}_N)^{-1}\mathbf{y}$$

$$\kappa_{lm} = \langle\boldsymbol{\phi}(\mathbf{x}_l), \boldsymbol{\phi}(\mathbf{x}_m)\rangle = \underbrace{\boxed{k(\mathbf{x}_l, \mathbf{x}_m)}}_{\text{kernel function}}$$

Deep Learning as a Kernel Method

Why?

$$\boldsymbol{\beta}_{\text{MLE}} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}\phi(\mathbf{x}_n)$$

DEEP LEARNING AS A KERNEL METHOD

Why?

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}$$

$$\boldsymbol{\beta}_{\text{MLE}} = -\frac{1}{\lambda} \sum_{n=1}^{N} \{\boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x}_n) - y_n\} \phi(\mathbf{x}_n)$$

DEEP LEARNING AS A KERNEL METHOD

Why?

$$\boldsymbol{\beta}_{\text{MLE}} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}\phi(\mathbf{x}_n)$$

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{y}$$

Predictions:

point estimate- no memory of data

$$y^* = \boxed{\boldsymbol{\beta}_{\text{MLE}}^T}\ \boldsymbol{\phi}(\mathbf{x}^*)$$

Why?

$$\boldsymbol{\beta}_{\text{MLE}} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}\phi(\mathbf{x}_n)$$

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{y}$$

Predictions:

point estimate- no memory of data

$$y^* = \boxed{\boldsymbol{\beta}^T_{\text{MLE}}}\ \boldsymbol{\phi}(\mathbf{x}^*)$$

all data

$$y^* = k(\mathbf{X}, \mathbf{x}^*)^T(\mathbf{K} + \lambda\mathbf{I})^{-1}\ \boxed{\mathbf{y}}$$

Why?

$$\boldsymbol{\beta}_{\text{MLE}} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}\phi(\mathbf{x}_n)$$

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{y}$$

Predictions:

point estimate- no memory of data

$$y^* = \boxed{\boldsymbol{\beta}_{\text{MLE}}^T}\ \boldsymbol{\phi}(\mathbf{x}^*)$$

all data

$$y^* = k(\mathbf{X}, \mathbf{x}^*)^T(\mathbf{K} + \lambda\mathbf{I})^{-1}\ \boxed{\mathbf{y}}$$

parametric $\rightarrow$ non parametric

Why?

$$\boldsymbol{\beta}_{\text{MLE}} = -\frac{1}{\lambda}\sum_{n=1}^{N}\{\boldsymbol{\beta}^T\boldsymbol{\phi}(\mathbf{x}_n) - y_n\}\phi(\mathbf{x}_n)$$

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I}_N)^{-1}\mathbf{y}$$

Predictions:

point estimate- no memory of data

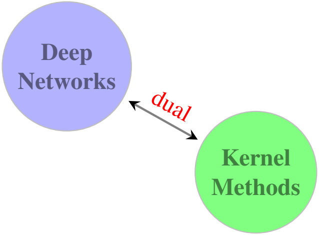
$$y^* = \boxed{\boldsymbol{\beta}^T_{\text{MLE}}}\ \boldsymbol{\phi}(\mathbf{x}^*)$$

all data

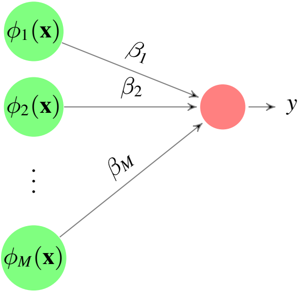
$$y^* = k(\mathbf{X}, \mathbf{x}^*)^T(\mathbf{K} + \lambda\mathbf{I})^{-1}\ \boxed{\mathbf{y}}$$

parametric $\rightarrow$ non parametric
Memory in β_{MLE} $\rightarrow$ Memory by actual storing all the data

Deep Learning as a Kernel Method



Step back...

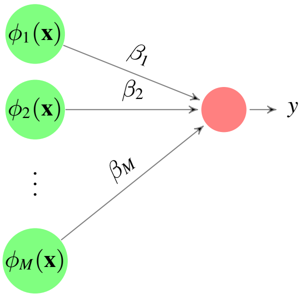


$$y = \boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x}) = \sum_m^M \beta_m \phi_m(\mathbf{x})$$

$$y = f(\mathbf{x})$$

TASK: learn the best regression function $f()$

Step back...



$$y = \boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x}) = \sum_m^M \beta_m \phi_m(\mathbf{x})$$

$$y = f(\mathbf{x})$$

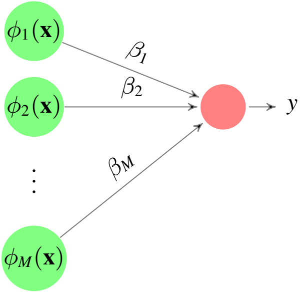
TASK: learn the best regression function $f()$

What about a bit more of Bayesian reasoning?

$\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)] \sim p(\mathbf{f})$

Consider $\mathbf{f}$ a multivariate variable- Apply distribution!

DEEP LEARNING AS A KERNEL METHOD



$$y = \boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x}) = \sum_m^M \beta_m \phi_m(\mathbf{x})$$

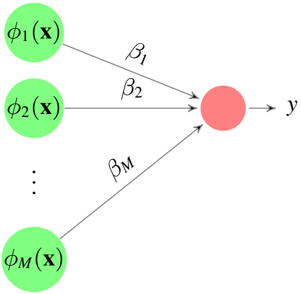
$$y = f(\mathbf{x})$$

$$\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)]$$

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}), \quad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

- Number of hidden units $\phi_m(\mathbf{x})$ to infinity, $M \to \infty$
- $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{m}, \Sigma)$

Deep Learning as a kernel Method



- Number of hidden units $\phi_m(\mathbf{x})$ to infinity, $M \to \infty$
- $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{m}, \Sigma)$

$$y = \boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x}) = \sum_m^M \beta_m \phi_m(\mathbf{x})$$

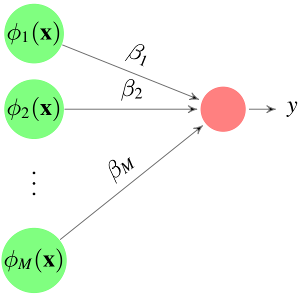
$$y = f(\mathbf{x})$$

$$\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)]$$

$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}), \quad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

When $N \to \infty$, sample from a Gaussian process

$$\mathbf{f} \sim \mathrm{GP}(\mathbf{0}, \mathbf{K})$$



$$y = \boldsymbol{\beta}^T \boldsymbol{\phi}(\mathbf{x}) = \sum_m^M \beta_m \phi_m(\mathbf{x})$$

$$y = f(\mathbf{x})$$

$$\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \ldots, f(\mathbf{x}_n)]$$

- Number of hidden units $\phi_m(\mathbf{x})$ to infinity, $M \to \infty$
- $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{m}, \Sigma)$

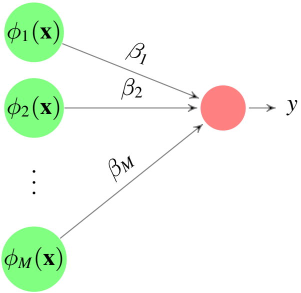
$$\mathbf{f} \sim \mathcal{N}(\mathbf{0}, \mathbf{K}), \quad K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$$

When $N \to \infty$, sample from a Gaussian process

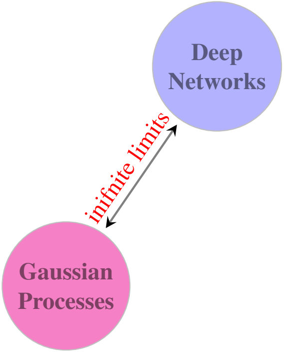
$$\mathbf{f} \sim \mathrm{GP}(\mathbf{0}, \mathbf{K})$$

Predict $y* = f(\mathbf{x}^*)$ $\quad p(f^*|\mathbf{X}, \mathbf{y}, \mathbf{x}^*) = \int p(y^*|\mathbf{x}^*, \mathbf{f}, \mathbf{X}, \mathbf{y}) p(\mathbf{f}|\mathbf{X}, \mathbf{y}) d\mathbf{f}$

Deep Learning as a kernel Method



Number of hidden units $\phi_m(\mathbf{x})$ to infinity, $M \rightarrow \infty$



Thank you!