

# COMP6248 Differentiable Programming

(and some Deep Learning)

Kate Farrahi and Jonathon Hare

Vision, Learning and Control  
University of Southampton

# Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data

$$\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N \quad \{\mathbf{x}_n\}_{n=1}^N$$

# Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data  $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N \quad \{\mathbf{x}_n\}_{n=1}^N$

Function Approximator  $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta}) + \nu$

# Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data  $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N \quad \{\mathbf{x}_n\}_{n=1}^N$

Function Approximator  $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta}) + \nu$

Parameter Estimation  $E_0 = \sum_{n=1}^N \{\|\mathbf{y}_n - f(\mathbf{x}_n; \boldsymbol{\theta})\|\}^2$

# Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data  $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N \quad \{\mathbf{x}_n\}_{n=1}^N$

Function Approximator  $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta}) + \nu$

Parameter Estimation  $E_0 = \sum_{n=1}^N \{\|\mathbf{y}_n - f(\mathbf{x}_n; \boldsymbol{\theta})\|\}^2$

Prediction  $\hat{\mathbf{y}}_{N+1} = f(\mathbf{x}_{N+1}, \hat{\boldsymbol{\theta}})$

# Machine Learning - A Recap

All credit for this slide goes to Niranjana

Data  $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N \quad \{\mathbf{x}_n\}_{n=1}^N$

Function Approximator  $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta}) + \nu$

Parameter Estimation  $E_0 = \sum_{n=1}^N \{\|\mathbf{y}_n - f(\mathbf{x}_n; \boldsymbol{\theta})\|\}^2$

Prediction  $\hat{\mathbf{y}}_{N+1} = f(\mathbf{x}_{N+1}, \hat{\boldsymbol{\theta}})$

Regularisation  $E_1 = \sum_{n=1}^N \{\|\mathbf{y}_n - f(\mathbf{x}_n; \boldsymbol{\theta})\|\}^2 + g(\|\boldsymbol{\theta}\|)$

# Machine Learning - A Recap

All credit for this slide goes to Niranjana

Data  $\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N \quad \{\mathbf{x}_n\}_{n=1}^N$

Function Approximator  $\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta}) + \nu$

Parameter Estimation  $E_0 = \sum_{n=1}^N \{\|\mathbf{y}_n - f(\mathbf{x}_n; \boldsymbol{\theta})\|\}^2$

Prediction  $\hat{\mathbf{y}}_{N+1} = f(\mathbf{x}_{N+1}, \hat{\boldsymbol{\theta}})$

Regularisation  $E_1 = \sum_{n=1}^N \{\|\mathbf{y}_n - f(\mathbf{x}_n; \boldsymbol{\theta})\|\}^2 + g(\|\boldsymbol{\theta}\|)$

Modelling Uncertainty  $p(\boldsymbol{\theta} | \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N)$

# Machine Learning - A Recap

All credit for this slide goes to Niranjana

Data	$\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N \quad \{\mathbf{x}_n\}_{n=1}^N$
Function Approximator	$\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta}) + \nu$
Parameter Estimation	$E_0 = \sum_{n=1}^N \{\ \mathbf{y}_n - f(\mathbf{x}_n; \boldsymbol{\theta})\ \}^2$
Prediction	$\hat{\mathbf{y}}_{N+1} = f(\mathbf{x}_{N+1}, \hat{\boldsymbol{\theta}})$
Regularisation	$E_1 = \sum_{n=1}^N \{\ \mathbf{y}_n - f(\mathbf{x}_n; \boldsymbol{\theta})\ \}^2 + g(\ \boldsymbol{\theta}\ )$
Modelling Uncertainty	$p(\boldsymbol{\theta}   \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N)$
Probabilistic Inference	$\mathbb{E}[g(\boldsymbol{\theta})] = \int g(\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} = \frac{1}{N_s} \sum_{n=1}^{N_s} g(\boldsymbol{\theta}^{(n)})$



# Machine Learning - A Recap

All credit for this slide goes to Niranjana

Data	$\{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N \quad \{\mathbf{x}_n\}_{n=1}^N$
Function Approximator	$\mathbf{y} = f(\mathbf{x}, \boldsymbol{\theta}) + \nu$
Parameter Estimation	$E_0 = \sum_{n=1}^N \{\ \mathbf{y}_n - f(\mathbf{x}_n; \boldsymbol{\theta})\ \}^2$
Prediction	$\hat{\mathbf{y}}_{N+1} = f(\mathbf{x}_{N+1}, \hat{\boldsymbol{\theta}})$
Regularisation	$E_1 = \sum_{n=1}^N \{\ \mathbf{y}_n - f(\mathbf{x}_n; \boldsymbol{\theta})\ \}^2 + g(\ \boldsymbol{\theta}\ )$
Modelling Uncertainty	$p(\boldsymbol{\theta}   \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N)$
Probabilistic Inference	$\mathbb{E}[g(\boldsymbol{\theta})] = \int g(\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} = \frac{1}{N_s} \sum_{n=1}^{N_s} g(\boldsymbol{\theta}^{(n)})$
Sequence Modelling	$\mathbf{x}_n = f(\mathbf{x}_{n-1}, \boldsymbol{\theta})$

# What is Deep Learning?

Deep learning is primarily characterised by function compositions:

- Feedforward networks:  $\mathbf{y} = f(g(\mathbf{x}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_f)$ 
  - Often with relatively simple functions (e.g.  $f(\mathbf{x}, \boldsymbol{\theta}_f) = \sigma(\mathbf{x}^\top \boldsymbol{\theta}_f)$ )

# What is Deep Learning?

Deep learning is primarily characterised by function compositions:

- Feedforward networks:  $\mathbf{y} = f(g(\mathbf{x}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_f)$ 
  - Often with relatively simple functions (e.g.  $f(\mathbf{x}, \boldsymbol{\theta}_f) = \sigma(\mathbf{x}^\top \boldsymbol{\theta}_f)$ )
- Recurrent networks:  
 $\mathbf{y}_t = f(\mathbf{y}_{t-1}, \mathbf{x}_t, \boldsymbol{\theta}) = f(f(\mathbf{y}_{t-2}, \mathbf{x}_{t-1}, \boldsymbol{\theta}), \boldsymbol{\theta}) = \dots$

# What is Deep Learning?

Deep learning is primarily characterised by function compositions:

- Feedforward networks:  $\mathbf{y} = f(g(\mathbf{x}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_f)$ 
  - Often with relatively simple functions (e.g.  $f(\mathbf{x}, \boldsymbol{\theta}_f) = \sigma(\mathbf{x}^\top \boldsymbol{\theta}_f)$ )
- Recurrent networks:  
 $\mathbf{y}_t = f(\mathbf{y}_{t-1}, \mathbf{x}_t, \boldsymbol{\theta}) = f(f(\mathbf{y}_{t-2}, \mathbf{x}_{t-1}, \boldsymbol{\theta}), \boldsymbol{\theta}) = \dots$

In the early days the focus of deep learning was on learning functions for classification. Nowadays the functions are much more general in their inputs and outputs.

# What is Differentiable Programming?

- Differentiable programming is a term coined by Yann Lecun<sup>1</sup> to describe a superset of Deep Learning.

---

<sup>1</sup><https://www.facebook.com/yann.lecun/posts/10155003011462143>

<sup>2</sup>See our ICLR 2019 paper: <https://arxiv.org/abs/1812.03928>

# What is Differentiable Programming?

- Differentiable programming is a term coined by Yann Lecun<sup>1</sup> to describe a superset of Deep Learning.
- Captures the idea that computer programs can be constructed of parameterised functional blocks in which the parameters are learned using some form of gradient-based optimisation.

---

<sup>1</sup><https://www.facebook.com/yann.lecun/posts/10155003011462143>

<sup>2</sup>See our ICLR 2019 paper: <https://arxiv.org/abs/1812.03928>

# What is Differentiable Programming?

- Differentiable programming is a term coined by Yann Lecun<sup>1</sup> to describe a superset of Deep Learning.
- Captures the idea that computer programs can be constructed of parameterised functional blocks in which the parameters are learned using some form of gradient-based optimisation.
  - The implication is that we need to be able to compute gradients with respect to the parameters of these functional blocks. We'll start explore this in detail next week...

---

<sup>1</sup><https://www.facebook.com/yann.lecun/posts/10155003011462143>

<sup>2</sup>See our ICLR 2019 paper: <https://arxiv.org/abs/1812.03928>

# What is Differentiable Programming?

- Differentiable programming is a term coined by Yann Lecun<sup>1</sup> to describe a superset of Deep Learning.
- Captures the idea that computer programs can be constructed of parameterised functional blocks in which the parameters are learned using some form of gradient-based optimisation.
  - The implication is that we need to be able to compute gradients with respect to the parameters of these functional blocks. We'll start explore this in detail next week...
  - The idea of Differentiable Programming also opens up interesting possibilities:
    - The functional blocks don't need to be direct functions in a mathematical sense; more generally they can be *algorithms*.
    - What if the functional block we're learning parameters for is itself an algorithm that optimises the parameters of an internal algorithm using a gradient based optimiser?!<sup>2</sup>

---

<sup>1</sup><https://www.facebook.com/yann.lecun/posts/10155003011462143>

<sup>2</sup>See our ICLR 2019 paper: <https://arxiv.org/abs/1812.03928>



# Is all Deep Learning Differentiable Programming?

- Not necessarily!
  - Most deep learning systems are trained using first order gradient-based optimisers, but there is an active body of research on gradient-free methods.

---

<sup>3</sup>including at least myself, my PhD students and Geoff Hinton!

# Is all Deep Learning Differentiable Programming?

- Not necessarily!
  - Most deep learning systems are trained using first order gradient-based optimisers, but there is an active body of research on gradient-free methods.
  - There is an increasing interest in methods that use different styles of learning, such as Hebbian learning, within deep networks. More broadly there are a number of us<sup>3</sup> who are interested in biologically motivated models and learning methods.

---

<sup>3</sup>including at least myself, my PhD students and Geoff Hinton!

# Why should we care about this?

# Where did it all start & what was the motivation?

# What is the objective of this module?

# What will we cover in the module?

# How is this module going to be delivered?

# Lecture & lab session plan



# What do we expect you already know?

# What might you already know?

# Assessment Structure

# Assessment Timetable

# The Main Assignment

The ICLR Reproducibility Challenge