COMP3242/6242: Deep Learning

Week 1 — Introduction and Propaedeutics

Stephen Gould
stephen.gould@anu.edu.au

Australian National University

Semester 1, 2026

Course Logistics

- **Course Staff:**

Steve

Carina

Enze

Evan

Isaac

Jingyang

Mehmet

Yuanchu

Zora

- **Lectures:** 8–9am Mondays and 3–4pm Tuesdays [**recorded**]
- **Labs:** 8am–10am Wed/Thu/Fri, 3pm–5pm Wed/Thu, 10am–12pm Fri
- **Communication:**
  - **In-person:** immediately following lectures or in labs
  - **Email:** `comp3242@anu.edu.au`
  - **Canvas:** handouts, announcements and student forum, lab assignments and submissions, project submissions

Course Philosophy

- deep learning is an incredibly fast moving field
  - faster than any other scientific/engineering advance in history
- the lectures in this course give you a solid understanding of the foundations and a few advanced topics
  - the field is massive and we don't cover everything
- you are expected to do a lot of self-study
  - read textbooks, papers, blogs beyond that covered in the course
- the labs and project allow you to learn by doing
  - hands-on experience to solidify your understanding
- we will use PyTorch but ideas translate to other software frameworks, e.g., TensorFlow and JAX

Assumed Knowledge

- **Assumed knowledge:**
  - linear algebra, multi-variate calculus, probability
  - computer programming (Python)

Course Materials

- **Textbooks:**

  - Goodfellow et al., *Deep Learning*, 2016 [http://www.deeplearningbook.org/]
  - Zhang et al., *Dive into Deep Learning*, 2023 [http://https://d2l.ai/]
  - Prince, *Understanding Deep Learning*, 2023 [https://udlbook.github.io/udlbook/]
  - Scardapane, *Alice's Adventures in a Differentiable Wonderland*, 2024 [https://www.sscardapane.it/alice-book]
- **Course notes and slides:** via Canvas
- **Papers and reading:** provided as reference for each lecture
- **Similar courses elsewhere:**
  - Stanford (Fei-Fei, Karpathy, et al.) [https://cs231n.stanford.edu/]
  - Michigan (Johnson) [https://web.eecs.umich.edu/~justincj/teaching/eecs498/FA2020/]
  - York University (Derpanis) [https://www.eecs.yorku.ca/~kosta/Courses/EECS6322/]

Course Assessment

- **Assessment:**
  - twelve weekly lab assignments (30%)
    - 3% each, starting Week 1, two are redeemable (we take the best 10 of 12)
    - mostly follows lecture material
    - submit as single Python file via Canvas; automatically graded
  - group research project (40%)
    - 5-minute video presentation/demo due end of Week 12 (40%)
    - 8-page report and code due first week of exams (60%)
  - final written exam (30%)
    - 3 hours, open book, no electronic devices
- **Policy on group work:**
  - you are encouraged to study in groups but must work independently on lab assignments and each submit your own work
  - we reserve the right to ask you to orally explain your solutions
- **Late policy:**
  - 0% unless accompanied by a doctor's certificate

Policy on Generative AI

The use of Generative AI tools (e.g., ChatGPT) is **permitted** in this course, given that proper citation and prompts are provided, along with a description of how the tool contributed to the work *and how you verified the accuracy of its output*.

Guidelines regarding appropriate citation and use can be found on the ANU library website (`https://libguides.anu.edu.au/generative-ai`).

Marks will reflect the contribution of the student rather than the contribution of the tools. Further guidance on appropriate use should be directed to the course staff.

Grievance Policy

- **Grievance resolution:**
  - You can contact the course staff in the first instance
  - Alternatively, contact the Dean of Students for advice
  - If unresolved, you can lodge a formal complaint: `https://www.anu.edu.au/students/contacts/student-complaint-resolution`

- **Warning:** This is a fast moving course. You are expected to know your own abilities (and whether or not you have the right background for the course). You are expected work diligently throughout the semester. We will endeavour to answer questions and give help in lectures, labs and online forums, but you are also expected to do a lot of reading and research on your own. This is part of the challenge of learning. In the end you will be rewarded by mastering material at the cutting-edge of AI.

Tentative Course Overview

1. Introduction and Propaedeutics
2. Linear Classifiers and Mult-layer Perceptrons
3. Automatic Differentiation and Back-propagation
4. Image Classification, Object Detection and Convolutional Neural Networks
5. Image Segmentation and Deconvolutions
6. Sequence Models and Back-propagation through Time

   —

7. Debugging Algorithms
8. Attention and Transformers
9. Contrastive Learning
10. Deep Reinforcement Learning
11. Generative AI
12. Review

Learning Outcomes

By the end of the course you should be able to:

- describe and apply the theory of convolutional neural networks and the universal approximation theorem
- apply the theory of training a convolutional neural network or transformer model
- write programs to design, train and test neural networks to solve problems in a given application area
- competently apply programming and software tools to write programs to solve realistic problems using deep neural networks
- understand the principals of testing and evaluating the performance of deep neural networks, and apply these principals in practice
- understand the role of data in training deep learning systems including ethical considerations and potential biases
- conduct a group/research project in applying deep learning methods to a given problem

Big picture: Where we fit (research field)

Big picture: Where we fit in time

"You have to know the past to understand the present" — Carl Sagan.

- categorized by theory and algorithms
- involved a lot of feature engineering
- e.g., **COMP4670** and **COMP8600**

- categorized by data and compute
- end-to-end learning and composeability
- emergent behaviours

- both continue to be important active research fields

What happened in 2012?

- since 2012 rapid progress has continued to accelerate thanks to:
  - open-source software
  - availability of (internet-scale) datasets
  - fast (and cheap?) compute
  - sharing of knowledge and results
  - investment ($$$)

Overview for Remainder of Lecture/Week

- tour of application areas
  - computer vision
  - natural language processing
  - multimodal (vision and language)
  - robotic and sequential decision making
  - scientific discovery and medicine
- machine learning from 10,000ft
- basic data structures in DL
- linear algebra review
- data processing in PyTorch

Applications — Computer Vision

image classification: *is this a dog or a cat or a bird or . . . ?*

identity recognition: *who is this?*

object detection: *what and where are the objects?*

(human) pose estimation: *locate all body joints*

panoptic segmentation: *label classes and instances*

feature matching: *find corresponding pixels in two (or more) images*

camera localization: *where are all the cameras wrt each other?*

tracking: *detect and track objects in a video stream*

Applications — Natural Language Processing

parsing: *syntax (nouns, verbs, adjectives)*

named entity recognition: *are "Stephen" and "Prof Gould" the same?*

speech recognition: *what song do you want me to play?*

machine translation: *convert this document to French*

dialogue: *e.g., chat bots*

question answering: *what is the capital of Australia?*

text generation: *write an email for a job application*

summarization: *what are the main points in this document?*

Applications — Multimodal Problems (Vision and Language) and GenAI

image captioning: *describe the image in words*

visual question answering: *what color are her eyes?*

visual and language navigation: *follow a language instruction using visual cues*

locate a specific object in an image: *find the light blue truck*

novel view synthesis: *show scene from a different viewpoint*

restyle: *show person as a child*

image generation: *an astronaut riding a horse*

video generation: *generate a video of a woolly mammoth*

Applications — Robotics and Sequential Decision Making

manipulation: *pick up objects from a work surface*

planning: *create a construction gantt chart*

multi-agent systems: *plan way points for a drone show*

navigation: *traverse an environment using a map*

autonomous driving: *self-driving taxis and driver assist technologies*

game play: *play the ancient Chinese game "Go"*

trajectory prediction: *predict trajectory of vehicles and pedestrians*

domain adaptation: *train in simulated environment, run in real-world*

Applications — Scientific Discovery and Medicine

protein structure prediction: *e.g., DeepMind's AlphaFold*

medical image analysis: *e.g., radiology report*

drug discovery: *predict drug interactions and side effects*

gene expression modeling: *discover gene expression pathways*

surrogate physics simulations: *e.g., fluid dynamics*

weather prediction: *predict pressure, temperature and rainfall maps*

clinical decision support: *treatment recommendation and patient monitoring*

brain-computer interface: *the future?*

Machine Learning from 10,000ft

minimize (over θ) $\sum_{(x,y)\sim\mathcal{X}\times\mathcal{Y}} \ell(f(x;\theta), y)$

- loss ℓ quantifies performance — **what to do**
- model $f(\cdot;\theta)$ (sometimes written f_θ) — **how to do it**
- composition of ℓ and $f(\cdot;\theta)$ non-convex in general
- optimized by gradient descent (or a variant)

Empirical Risk Minimization

- principled optimization framework for machine learning
- defines an empirical measure of performance over a given finite dataset of examples, $\mathcal{D} = \left\{ \left(x^{(i)}, y^{(i)} \right) \right\}_{i=1}^{N}$, called the risk[1]

$$L(\theta; \mathcal{D}) = \sum_{i=1}^{N} \ell(f(x^{(i)}; \theta), y^{(i)})$$

- empirical risk minimization states that we should choose the parameters that minimize L over all possible parameters (the "hypothesis" class)
- to prevent overfitting or incorporate prior information we often include a regularization term on θ,

$$L(\theta; \mathcal{D}) = \sum_{i=1}^{N} \ell(f(x^{(i)}; \theta), y^{(i)}) + \lambda R(\theta)$$

[1] Technically, L is called the **risk** and ℓ the **loss**. However, most researchers will also call L the **loss**.

Encoder-Decoder Paradigm

A very common paradigm in deep learning is the encoder-decoder model:

Examples.

- auto-encoders ($y = x$)
  - attempt to learn a low-dimensional (and sometimes sparse) representation of data
  - does not need explicit supervision; z is called a **bottleneck** between x and y
- cross-encoders ($y \neq x$) give rise to interesting applications
  - use a CNN encoder-decoder for semantic segmentation or style-transfer
  - use an RNN encoder-decoder for language translation
  - use a mixed CNN-RNN encoder-decoder for image captioning
  - learn joint image-language embedding (e.g., CLIP and BLIP)

Ingredients of a Machine Learning System

- data and labels (supervision)
- representation (datastructures)
- model/network architectures
- objectives (losses and priors)
- learning algorithms
- evaluation metrics

(https://xkcd.com/1838/)

Each one of these ingredients encode **(inductive) biases** that affect system behaviour and provide levers to change behaviour. We will study each of them in this course in the context of deep learning systems.

Vector, Matrix, Tensor Basics

for the purposes of this course (mostly),

- a (column) **vector** is a one-dimensional array of numbers
- a **matrix** is a two-dimensional array of numbers
- a **tensor** is an arbitrary-dimensional array of numbers
  - the number of dimensions is called the **order** of the tensor

- **dimension** can also be used to specify the number of entries a vector, e.g., a 3-dimensional vector
- we describe vectors, matrices and tensors by their **size** or **shape**, e.g., a 2-by-3 matrix has shape `(2, 3)`

Images, Feature Maps and Videos

- an image is an array of red, green and blue (RGB) colour pixels

- as such it can be represented by a third-order tensor with shape
  - $(H, W, 3)$ in OpenCV and numpy or $(3, H, W)$ in PyTorch
- and values in the range 0–255 (integer) or 0.0–1.0 (floating point)
- this is called a 3-**channel** or 3-**plane** image

- instead of colour we can generalize to C-dimensional features per pixel to give a shape (C, H, W) feature map
- a video can be represented as a $(3, H, W, T)$-tensor

Linear Algebra Review

- matrix operations
  - transpose
  - addition
  - multiplication
- inner products and norms

... many, many topics not covered. Some great online resources are:

- Gilbert Strang's MIT lectures,
  `https://www.youtube.com/playlist?list=PL49CF3715CB9EF31D`
- 3Blue1Brown animated math videos,
  `https://www.3blue1brown.com/topics/linear-algebra`

Transpose

- The transpose of a an $m \times n$ matrix, A, is an $n \times m$ matrix, A^T, in which the rows and columns are swapped, e.g.,

$$A = \begin{bmatrix} 8 & 5 \\ 3 & 6 \end{bmatrix} \qquad A^T = \begin{bmatrix} 8 & 3 \\ 5 & 6 \end{bmatrix}$$

- The transpose of a column vector is a row vector and vice versa, e.g.,

$$a = \begin{bmatrix} 8 \\ 5 \\ 3 \\ 6 \end{bmatrix} \qquad a^T = \begin{bmatrix} 8 & 5 & 3 & 6 \end{bmatrix}$$

- The following properties of transposes are true:
  - $\left(A^T\right)^T = A$
  - $(AB)^T = B^T A^T$
  - $(A + B)^T = A^T + B^T$

Special Vectors and Matrices

- The **zero matrix** is a matrix with all entries equal to zero and written as $\mathbf{0}_{m\times n}$.
- The **all-ones vector** is a vector with all entries equal to one and written as $\mathbf{1}_n$.
- A **canonical vector** e_i is a vector with i-th entry one and all other entries zero.
  - Also called a **indicator** or **one-hot** vector.
- A **diagonal matrix** is a matrix where the off-diagonal entries are zero,

$$A_{ij} = \begin{cases} d_i & \text{if } i = j \\ 0 & \text{if } i \neq j \end{cases}$$

- The **identity matrix**, denoted $\mathbf{I}_{n\times n}$, is a square matrix whose diagonal entries are one and whose off-diagonal entries are zero,

$$I = \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix}$$

```
I = torch.eye(3)        # creates 3-by-3 identity matrix
```

Matrix Addition

Two matrices of the *same size* can be added together in a componentwise fashion.

$$C = A + B$$

means

$$C_{ij} = A_{ij} + B_{ij} \quad \forall i = 1, \ldots m \text{ and } j = 1, \ldots n$$

Matrix-Vector Multiplication

The product of a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $x \in \mathbb{R}^n$, written $y = Ax$, is an m-dimensional vector with elements

$$y_i = \sum_{j=1}^{n} A_{ij} x_j$$

- cost of matrix-vector multiplication is $O(mn)$
- multiplication can be generalized to matrix multiplicands

```
y = torch.matmul(A, x)      # or y = A @ x
```

Matrix-Matrix Multiplication

The product of two matrices $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$ is an $m \times p$ matrix with elements

$$C_{ij} = \sum_{k=1}^{n} A_{ik} B_{kj}$$

Note that the order of the matrices matters and for the product to exist we need the number of columns of A to equal the number of rows of B.

- cost of matrix multiplication in $O(mnp)$
- matrix multiplication is associative: $(AB)C = A(BC)$
- matrix multiplication is distributive: $A(B + C) = AB + BC$
- matrix multiplication is not, in general, commutative: $AB \neq BA$

Matrix Multiplication Interpretation

Consider a matrix $A \in \mathbb{R}^{m \times n}$ and vector $x \in \mathbb{R}^n$. Then we can interpret the product $y = Ax \in \mathbb{R}^m$ in the following ways:

- the i-th entry of y is the inner product of the i-th row of A and x

$$y = \begin{bmatrix} — & a_1^T & — \\ — & a_2^T & — \\ & \vdots & \\ — & a_m^T & — \end{bmatrix} x = \begin{bmatrix} a_1^T x \\ a_2^T x \\ \vdots \\ a_m^T x \end{bmatrix}$$

- y is a linear combination of the columns of A

$$y = \begin{bmatrix} | & | & & | \\ a_1 & a_2 & \cdots & a_n \\ | & | & & | \end{bmatrix} x = \sum_{i=1}^{n} x_i a_i$$

Inner Product

The **inner product** between two vectors (of equal length) is defined by

$$\begin{aligned}\langle x, y\rangle &= x_1 y_1 + \cdots + x_n y_n \\ &= x^T y\end{aligned}$$

Some important properties of inner product are:

- $\langle \alpha x, y\rangle = \alpha \langle x, y\rangle$
- $\langle x + y, z\rangle = \langle x, z\rangle + \langle y, z\rangle$
- $\langle x, y\rangle = \langle y, x\rangle$
- $\langle x, x\rangle \geq 0$
- $\langle x, x\rangle = 0$ if and only if $x = 0$

The row vector x^T represents a linear function $\mathbb{R}^n \to \mathbb{R}$.

Euclidean Norm

For $x \in \mathbb{R}^n$, we define the (Euclidean) **norm** as

$$\begin{aligned} \|x\|_2 &= \sqrt{x_1^2 + \cdots + x_n^2} \\ &= \sqrt{x^T x} \end{aligned}$$

$\|x\|_2$ measures the length of the vector (from the origin).

Some important properties of norm are:

- $\|\alpha x\| = |\alpha| \|x\|$
- $\|x + y\| \leq \|x\| + \|y\|$ (triangle inequality)
- $\|x\| \geq 0$
- $\|x\| = 0$ if and only if $x = 0$

Other important norms are $\|\cdot\|_1$ and $\|\cdot\|_\infty$

Batch Processing and Broadcasting

Deep learning frameworks process data in batches, passed as tensors. The first dimension of the tensor is the batch dimension. E.g.,

- a batch of n-dimensional vectors has shape (B, N)
- a batch of colour images has shape $(B, 3, H, W)$

Example. For the operation $y = Ax + b$ we might have

$$X = \{x^{(1)}, \ldots, x^{(B)}\} \qquad \text{(input)}$$
$$Y = \{Ax^{(1)} + b, \ldots, Ax^{(B)} + b\} \qquad \text{(output)}$$

Many PyTorch functions are batch-aware and support broadcasting, e.g.,

```
y = torch.matmul(A.view(1, M, N), x.view(B, N, 1)).view(B, M) + b.view(1, M)
# could also be done with: y = torch.einsum("ij,kj->ki", A, x) + b
```

computes $y^{(k)} = Ax^{(k)} + b$ on each element $k = 1, \ldots, B$ of the batch.

Always read the API docs to know how broadcasting applies to different operations.

Tensors in PyTorch

In addition to their shape (`size`), tensors in PyTorch have several other properties:

- `dtype` to specify numerical precision, e.g., 32-bit float
- `device`, e.g., on CPU or GPU
- `requires_grad` will be discussed later
- `grad` holds gradient for training (will be discussed later)

There are lots of ways to create a tensor. For example,

```
A = torch.tensor([[8, 5], [3, 6]])
```

creates the matrix

$$A = \begin{bmatrix} 8 & 5 \\ 3 & 6 \end{bmatrix}$$

A `Parameter` is a type of learnable `Tensor` used within PyTorch modules that you will encounter later in the course.

Reshaping

- it will often prove useful to be able to **reshape** a tensor
  - transposing a vector is a prototypical example (but not transposing a matrix)
- always possible so long as the total number of elements does not change, e.g.,

```
A = torch.tensor([[8, 5, 1], [3, 6, 2]])
B = A.reshape([3, 2])
```

  creates matrices

$$A = \begin{bmatrix} 8 & 5 & 1 \\ 3 & 6 & 2 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 8 & 5 \\ 1 & 3 \\ 6 & 2 \end{bmatrix}$$

- the method **view** can also be used whenever the memory layout of the underlying data does not need to change, e.g., `B = A.view([3, 5])`
- a very common reshaping is to **flatten** the tensor (sometimes called vectorizing), e.g., `C = A.flatten()` produces

$$C = \begin{bmatrix} 8 & 5 & 1 & 3 & 6 & 2 \end{bmatrix}$$

- note that elements are rearranged rowwise (called row-major ordering)