

# Introductory Lecture Notes on Deep Learning

Stephen Gould\*  
[stephen.gould@anu.edu.au](mailto:stephen.gould@anu.edu.au)

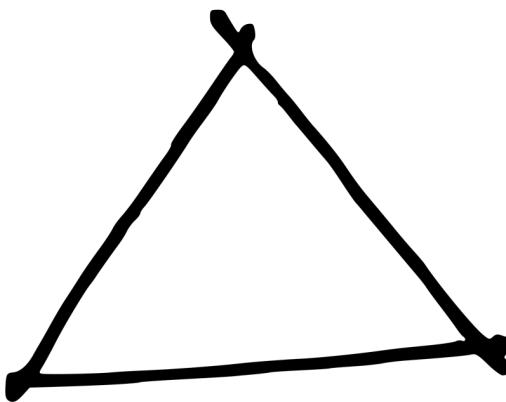
February 4, 2026

## Abstract

These lecture notes are for a one-semester (12-week) introductory course on deep learning, covering both theory and practice. Students will learn the foundational mathematics behind deep learning and explore topics such as multi-layer perceptrons (MLPs), back-propagation and automatic differentiation, convolutional neural networks (CNNs), recurrent neural networks (RNNs), and transformers. These techniques play a crucial role in modern artificial intelligence (AI) systems, including image and video understanding, natural language processing, generative AI, robotics, medicine and scientific discovery. The course includes various practical assessments to enhance student's understanding and intuition of deep learning and its diverse applications. Students are expected to have strong programming skills and previous exposure to linear algebra, differential calculus, and probability theory. Assessment details do not form part of these notes.

## Appetizer

Consider the two-dimensional shape depicted in Figure 1. Nobody would have trouble recognising the shape as a triangle despite the lines not being completely straight and them not meeting precisely at each corner. And when asked why the shape is a triangle, most people would answer that it's because the shape has three sides. Deep learning, however, takes a different perspective. Deep learning says that the figure is a triangle because it *looks* like a triangle. In the same way, a cat is a cat because it looks like a cat, and a dog is a dog because it looks like a dog. No further justification is needed.<sup>1</sup> This is because deep learning methods learn from data, i.e., from training examples, rather than from prescribed rules. In this course we will study deep learning and discover how this is at all possible.

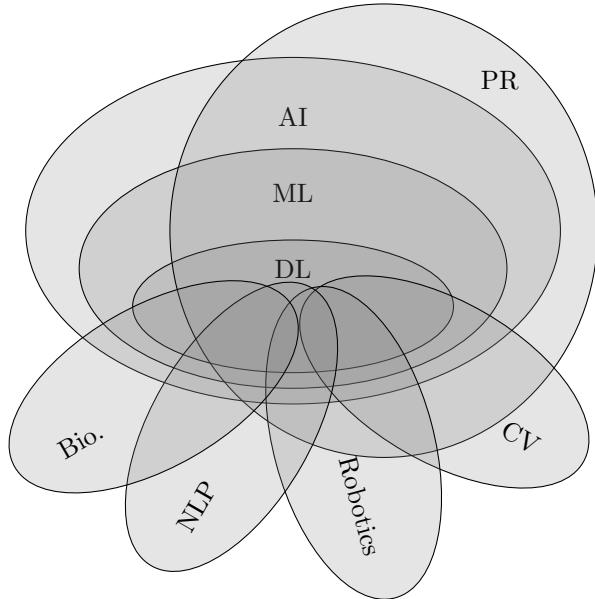


**Figure 1:** A triangular shape. But why? (Image credit: <https://www.kisscc0.com/>)

---

\*We are grateful to the many students and colleagues who have provided feedback and corrections on earlier versions of these notes.

<sup>1</sup>This is not completely true—there is an entire research field devoted to the important topic of explainable AI, which requires that a human understandable justification be given for the outputs, answers, predictions, and decisions produced by an AI algorithm.



**Figure 2:** A Venn diagram showing where deep learning fits within the broader field of artificial intelligence. Not shown are the foundational disciplines of mathematics and computer science.

## 1 Introduction

This lecture begins by contrasting deep learning against classical machine learning and related fields. It then gives a very quick tour of problems in certain influential application areas. The lecture ends by discussing how data is represented and processed within deep learning systems, namely, via the tensor data structure and review of basic linear algebra operations. It is assumed the material in this lecture is mostly review and that most students would have seen it before.

Let us begin by positioning the field of deep learning within the broader scientific discipline of artificial intelligence (AI). Figure 2 provides a good illustration. We can loosely define AI as being about building machines that behave in an intelligent way. Machine learning (ML) is a sub-field of AI where the machines that we build improve their performance on some task by learning from data or with gained experience. Deep learning (DL) then is a sub-field of machine learning where the models that make up these machines—often called **neural networks**—are learned in an end-to-end fashion (i.e., with limited human engineering of features or model architectures).<sup>2</sup> A large part of AI involves pattern recognition (PR), which is an older term that relates mostly to perception and prediction versus action and planning, another large part of AI. Not shown in the diagram are the mathematical and computer science foundations upon which deep learning is based—primarily, linear algebra, probability, calculus, optimisation, programming, algorithms, and data science. This course assumes that you have a basic understanding of these foundation topics, some of which will be reviewed briefly at the end of the lecture.

This course studies deep learning with a focus on certain application areas such as computer vision and natural language processing that have heavily influenced the models and methods in deep learning. In turn, these application areas have gained enormously from the results that deep learning has delivered. Computer vision (CV) is about developing algorithms for machines to understand the world behind images and videos. Some people think of this as inverse graphics. Natural language processing (NLP) studies the structure of language and develops algorithms for extracting information from language—both written and spoken. It enables machines to engage with humans in a natural way. We will discuss various problems studied in both CV and NLP later in the lecture.

It is important to note that there are many areas in computer vision and natural language processing that are not related to deep learning, for example, how cameras capture images of the world or the semantic and syntactic structure of language. Computer vision and natural language processing are still very much their own independent research fields. Likewise, there are many other application areas that have contributed to and benefited from deep learning. The most notable of these are robotics, bioinformatics, scientific discovery, and medicine.

Carl Sagan, the famous astronomer and science educator, once said, “You have to know the past to understand the present.” We can pinpoint a precise moment in time when deep learning became a viable and popular research field. Although research into neural networks can be traced back to the 1960s [27, 34], it wasn’t until 2012 where ideas from

---

<sup>2</sup>We will see a slightly different characterization shortly.

the past were put into a system that was truly competitive with other techniques for solving problems in computer vision—and shortly afterwards natural language processing—and the field that we call deep learning was born. Roughly speaking, research and methods in machine learning and its applications prior to 2012 were characterized by theory and provable algorithms. Systems involved a lot of feature engineering to get to work. Post 2012, deep learning methods have been characterized by data and compute, and building systems with end-to-end composeability. It is still not easy to get systems to work reliably all the time. But, as we will see, the knobs that we get to tune are different to the feature engineering of the past.

While deep learning is the dominant approach to solving problems in computer vision and natural language processing nowadays, it is important to mention that many classical ideas are still very relevant, either in their influence on deep learning techniques, in their adoption as components within larger deep learning systems, or as standalone algorithms. Moreover, developing theories and principles to better understand the behaviour and reliability of deep learning systems is becoming more and more urgent as deep learning algorithms are deployed in real-world engineering, economic, environmental, educational, societal, and health applications.

**So what happened in 2012?** A few years prior to 2012 a group of researchers from Stanford University led by Fei-Fei Li collected a massive dataset of images from the web (over one million). Each image in the dataset was annotated with a label that described its content employing the help of crowd-sourcing. There were images of dogs and cats of various breeds, cars, airplanes, foods, etc. One thousand different categories in total. The dataset was called ImageNet [6]. It was and remains one of the largest datasets of its kind. To go along with the dataset, the Stanford group started the ImageNet large-scale visual recognition challenge (ILSVRC), where researchers from around the world could compete on who had the best image recognition algorithm, that is, whose algorithm would correctly recognise the most number of images in a hold-out test set. In 2012, Alex Krizhevsky, a graduate student of Geoffrey Hinton from the University of Toronto, entered the challenge with a deep learning based model [15]. It significantly beat all other models. This was a watershed moment for deep learning. The following year the top five performing entries were based on deep learning. The year after that, and every year since, all entries used deep learning and the error rate steadily dropped. Today deep learning outperforms humans on the ILSVRC challenge.

Since 2012 progress has continued to accelerate thanks various initiatives and proven effectiveness across a large number of application domains. First, the development and support of high-quality open-source software libraries (supported by industry) such as PyTorch [22], TensorFlow [1] and JAX [2], has made it easy to develop models and reproduce results from other researchers. Second, the availability of large and diverse datasets necessary to train models. Third, the existence of fast and cheap compute such as GPUs for accelerating the core calculations used in deep learning models.<sup>3</sup> Third, a growing community of researchers and engineers that facilitate the rapid sharing of knowledge, ideas, and results. And last, the amazing success of recent large language models (such as ChatGPT, Copilot, Gemini and Claude) and tools for generative AI have attracted significant private and public investment in AI commercialisation and supporting infrastructure.

## 1.1 Tour of Application Domains

In this section we provide a very quick tour of problems studied in computer vision, natural language processing and other application domains. These fields are huge so this tour are just the tip of the iceberg to give you a flavour of the types of tasks being solved, and the language used to describe them. Many tasks are related and there are many variants that differ in underlying assumptions, categories to be recognized, availability and quality of training data, additional or different imaging modalities, etc. All have been advanced by deep learning in one way or another.

### 1.1.1 Computer Vision

Perhaps the most basic task in computer vision requiring machine learning is **image classification**.<sup>4</sup> Here the goal is to assign a single label to the entire image—the label can be binary (e.g., yes/no to a particular question) or multi-class where the system returns a label from a pre-defined set, typically of categories of objects (e.g., dog, cat, etc.). Algorithms often also provide a confidence score in addition to the label. These tasks are somewhat ill-defined and a critical inbuilt assumption is that the image neatly fits into one of the categories and the object (for object label sets) is dominant and well-framed within the image.

Going beyond a pre-defined set of categories, **open vocabulary image classification** allows images to be labeled with any noun, even those not seen during training of the model. A typical task of this type might be *what object is this?* This is a very challenging task and methods are usually augmented with an external knowledge base or pre-trained large vision-and-language model.

Structured prediction is a more refined machine learning approach outputs structured data rather than a single

<sup>3</sup>Though, even with GPU-acceleration training large deep learning models can take several weeks or more. The original AlexNet model took five to six days to train using two NVIDIA GTX 580 GPUs [15].

<sup>4</sup>There are many other tasks in computer vision, such as involving multi-view geometry, that do not require machine learning.

categorical label or regression value. In the context of computer vision the canonical example is **object detection** where the algorithm is expected to output a set of object labels and bounding box pairs corresponding to objects found in the image. Another example is the **(human) pose estimation** task, where we are asking to *locate all body joints* such as hands, elbows, shoulders, etc. Here a skeleton model defines the joints and how they are connected in the model, which constrains the location of one joint with respect to another.

There is a very large and specialized research area on **text recognition**, which aims to *locate and read text* within the image. This has applications from number plate recognition for automating parking lot ticketing, to interpreting receipts for financial systems, to reading text in-the-wild for place recognition or language translation (e.g., of menus at foreign restaurants).

Pixel labeling tasks also go beyond giving a single label to the entire image and instead assign one or more class labels to every pixel in the image. These labels can represent semantics, such as a sky pixel, or indicate membership such as all pixels having the same (integer) label belong to the same object. The most basic example of the latter type is **(unsupervised) image segmentation** or over-segmentation, which *breaks images up into meaningful regions* sometimes called **superpixels**. This is often a pre-processing building block for some more sophisticated downstream algorithm.

**Panoptic segmentation**, for example, labels each pixel with a semantic category (from a pre-defined set) and an instance identifier. This allows separation of multiple instances of the same object, e.g., multiple people in a crowd. Background categories are often considered to not have different instances. Sometimes researchers will use the terms **things** and **stuff** to distinguish between categories with distinct well-defined boundaries (and hence countable instances) versus those with amorphous shape and extent such as sky and grass.

Pixel labeling tasks are not restricted to semantic categories. **Monocular depth estimation** is a task that labels every pixel with its depth to the camera. If we have two cameras (or two views from the same camera) with known relative pose then the problem becomes **stereo depth estimation**, which can be solved by understanding the geometry of image formation and using traditional computer vision techniques [11].

A large area of computer vision involves searching and sorting operations on which many downstream tasks are built. In **feature matching** we are interested in finding corresponding pixels between two (or more) images, i.e., pixels that represent the same location in 3D space. This helps with applications such as camera calibration, object tracking, and 3D reconstruction. In **visual search** we are interested in finding information based on a query image such as the name of a famous landmark. In **attribute ranking** a set of images is sorted according to some visual attribute, which is useful for applications like online shopping.

There are several tasks in computer vision that require reasoning over temporally evolving scenes. In **tracking** we are asked to detect and track objects as they move through an environment where they may sometimes be occluded by other objects in the scene. The task of **activity recognition** is akin to image classification but for video data where we ask what activity is happening in a (short) video clip. Just like image classification this can be extended to activity detection and activity segmentation for longer video clips.

Several computer vision tasks are concerned with editing or synthesizing images as the output. There has been a long history of work focused on **image denoising**, which aims to remove noise and restore corrupted images, **image colorization**, which aims to convert a black and white image to colour, and **image inpainting**, which aims to remove an object from an image and fill in the background with a plausible reconstruction. In **video stabilization**, post-processing is used to remove jitter caused by a handheld camera and motion blur caused by fast moving objects.

More recently, deep learning has enabled tasks such as **novel view synthesis**, which depicts a scene from a viewpoint not previously seen by the model, i.e., different to where the image (or set of images) was originally taken. Neural radiance fields (NeRFs) are a popular class of deep learning model for novel view synthesis. Another task is **restyling**, which changes the style but not the viewpoint or content of a scene, e.g., *depict an adult as a child*. We can also create completely novel (fake) images and videos using **generative AI**, which involves both vision and language understanding.

### 1.1.2 Natural Language Processing

One of the core areas of natural language processing (NLP) is understanding the structure and meaning of language, including morphology, syntax and semantics. **Morphology** characterises the formation of words, e.g., “running”, “ran” and “runs” are all related to the dictionary word “run.” **Syntax** is about how words are combined into grammatically valid sentences. It involves identifying parts-of-speech such as nouns and verbs, and parsing sentences to understand the relationship between words. **Semantics** is concerned with the meaning of words, sentences and larger pieces of text and spoken language. For example, recognising names and associating pronouns with a particular person or people (**named entity recognition**) and distinguishing between the meanings of words that have the same spelling or sound (**word sense disambiguation**), e.g., “bank” as a financial institution or the side of a river. The ability to automatically parse text and perform **information extraction** comes from our ability to understand the

syntax and semantics of language.

The task of **speech recognition** is a long standing AI problem that involves transcribing a speech signal into written language, and is now largely solved by deep learning methods. One of the key challenges is that different words can sound the same, as alluded to above, that different speakers pronounce words differently, and that background noise and context all play an important role in us understanding speech. The inverse problem of **speech synthesis**—generating a speech signal from text—is much easier although still a challenge for natural sounding speech.

Another well-studied problem is NLP is **machine translation**, e.g., translating from English to French, or from Chinese to German. Here the problem is that different languages have different rules and constructs so we cannot simply build a one-to-one mapping of words. Methods initially invented for solving machine translation are now at the heart of large language models (LLMs) that enable tasks such as **dialogue** (i.e., chatbots), **question answering**, **text generation** and **summarization**. Some claim that these models are capable of **commonsense reasoning**. Indeed, large language models exhibit remarkable fluency when applied to these tasks, even if their accuracy is sometimes lacking.

### 1.1.3 Multimodal Problems (Vision and Language)

One of the more impressive feats that deep learning has been able to achieve is unified reasoning over multiple data modalities such as vision and language. This manifests itself in many different tasks. In **image captioning** the goal is to describe an image in words. This is refined in **visual question answering** (VQA) where we ask specific free-form questions (i.e., not necessarily seen during training) of an image. For example, given an appropriate image where such questions make sense, *what color are her eyes? what is the mustache made of?*

**Visual and language navigation** (VLN) is a relatively new area of research where we ask a robot (or simulated agent) to follow a natural language navigation instruction using visual cues, e.g., *exit the dining room via the door next at the far end of the table, walk to the kitchen and stop in front of the fridge*.

Another popular vision and language task is to **locate a specific object in an image**, such as to *find the light blue truck*. Here the algorithm is expected to return a bounding box or segmentation mask of the object being localized. The task can be combined with VLN to navigate to a particular location and then identify a target object in the scene. Continuing from the example above we might ask the robot to *open the fridge and locate the bottle of milk*.

As mentioned above, in addition to basic image editing tasks, we can combine vision and language to create completely novel (fake) images and videos using **generative AI** from text prompts such as *an image of an astronaut riding a horse* and *a video of a woolly mammoth*.

### 1.1.4 Robotics and Sequential Decision Making

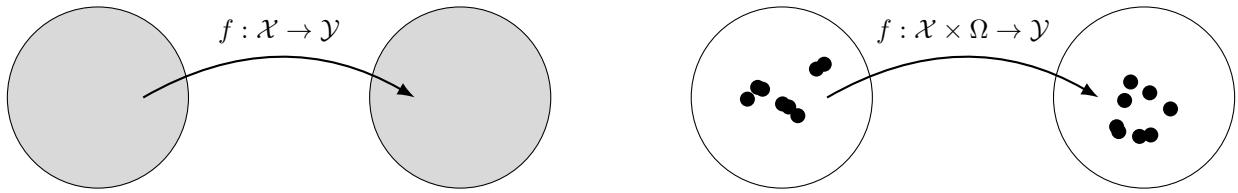
Game play has been a long-standing test of AI systems. Research in this area includes state representation, search and planning under uncertainty, learning of policies, opponent modeling, and long-horizon credit assignment. Games can be deterministic with perfect information (e.g., chess or Go) or stochastic and partially observable (e.g., poker). **Reinforcement learning** is a classic machine learning approach to develop policies for sequential decision making that occurs in complex games. Deep learning variants [18, 31] have demonstrated amazing results in this area.

Planning tasks generalise beyond simple games and focus choosing a sequence of actions to achieve some goal while respecting constraints and optimizing objectives. Markov decision processes (MDPs) and partially-observable MDPs are classic models that solve the problem of planning under uncertainty and usually involve reinforcement learning to solve. Planning underpins many high-level AI applications, from logistics and scheduling to decision-making in robotics and autonomous systems.

In robotic movement and manipulation, perception (e.g., object detection and pose estimation), planning, and control are combined to navigate and interact with objects in a physical environment. Unlike games or abstract planning, manipulation must contend with noisy sensors, imperfect actuation, deformable objects, and complex physical interactions. Autonomous driving [30] is a special type of robotic task with the goal of developing vehicles that can drive themselves either in controlled situations or in open-ended environments with many other moving vehicles and pedestrians. Research into autonomous driving has brought many safety and driver assist technologies into regular vehicles (such as lane tracking and collision detection).

### 1.1.5 Scientific Discovery and Medicine

Science and medicine has seen an increasing application of deep learning methods over the past several years. One significant area of research is in understanding biological processes and novel drug discovery. The 2024 Nobel Prize in Chemistry awarded to David Baker, Demis Hassabis and John Jumper, was in acknowledgment of their work in computational protein design and protein structure prediction, enabling the creation of entirely new proteins. Hassabis is the founder of DeepMind where work initially focusing on game play (e.g., AlphaGo) has been applied to numerous



**Figure 3:** Machine learning from 10,000ft.

scientific endeavours including AlphaFold for predicting the structure of proteins.

Other areas of deep learning in science includes weather forecasting, medical image analysis, clinical decision support, and surrogate models for complex physical processes such as fluid dynamics. In the not-too-distant future we may see brain-computer interfaces driven by deep learning models that interpret brain activity and link your thoughts directly to controlling your phone and other devices.

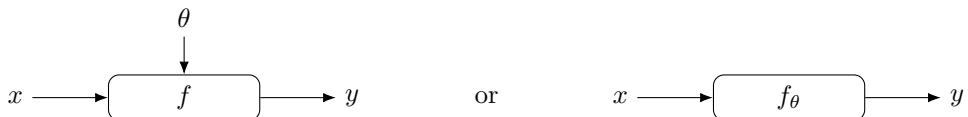
## 1.2 Machine Learning from 10,000ft

At its core machine learning is about finding a function  $f$  that maps from an input space  $\mathcal{X}$  to an output space  $\mathcal{Y}$  (see Figure 3). Since we can't practically search over all possible mappings we define a function class parametrized by  $\theta$  and train a model on a dataset of samples from  $\mathcal{X}$  and  $\mathcal{Y}$ , denoted  $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}_{i=1}^N$ . For example, the function class may be restricted to only linear functions or only polynomials or neural networks of a given architecture. The goal is for the model perform well over the entire space—it is no good to just memorize the training samples. This is, of course, very difficult since much of the space is unseen (i.e., not covered by samples in  $\mathcal{D}$ ). The mapping  $f$  is learned by minimizing (over the parameters  $\theta$  of the model) a loss function  $L$ , that in some sense measures the inaccuracy of the model. Typically, the loss function decomposes over sampled input-output pairs,

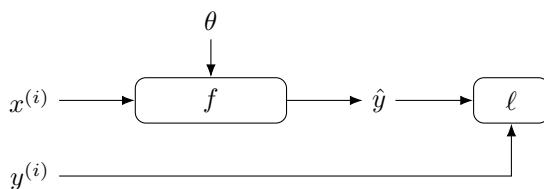
$$\text{minimize}_\theta \quad L(\theta; \mathcal{D}) \triangleq \sum_{(x,y) \in \mathcal{D}} \ell(f(x; \theta), y) \quad (1)$$

This is called **empirical risk minimization** [32].<sup>5</sup> Here the loss function  $L$  tells us **what** to do, and the parametrized function  $f(\cdot; \theta)$  tells us **how** to do it.<sup>6</sup> The objective (composed of the loss  $L$  and the mapping function  $f$ ) is, in general, nonconvex in the parameters  $\theta$ . As such, we can only hope to find a local minima of the learning problem, which we do using **gradient descent** or one of its variants.<sup>7</sup>

It is common to represent the machine learning model diagrammatically as,<sup>8</sup>



We can also incorporate the loss function in the diagram when we want to stress how the model is trained,



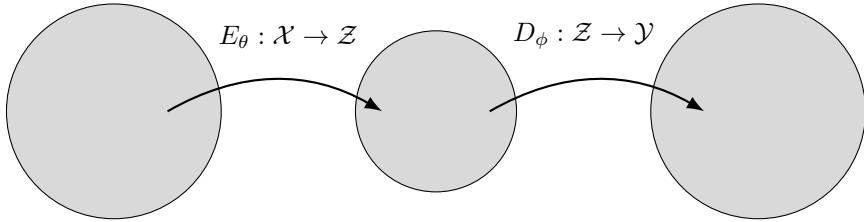
To summarise, empirical risk minimization (ERM) [32] is a principled framework for choosing model parameters in machine learning. The theory of empirical risk minimization states that we should choose the parameters of the model

<sup>5</sup>The function  $L$  is often scaled by  $1/N$ . Technically it is then the expected loss called **risk** with the function  $\ell$  being the **loss**. However, most researchers will use the terms interchangeably, or simply refer to any function being minimized in deep learning a loss function.

<sup>6</sup>Often you'll see notation  $f_\theta$ , indicating that  $\theta$  indexes  $f$  from some function class  $\mathcal{F}$ . However, in these notes we prefer  $f(\cdot; \theta)$  to make it clear that the function takes parameters as an argument and avoid confusion when discussing function compositions in later lectures.

<sup>7</sup>In practical applications suboptimal solutions are often desirable since finding the globally optimal solution can result in poor generalization of the model to unseen test samples. Techniques such as regularization, data-augmentation, and model selection on a hold-out validation set all help in this regard, and will be discussed in later lectures.

<sup>8</sup>In the sequel we will sometimes use symbol  $y$  to denote the target output (i.e., groundtruth) and sometimes the estimated output of the network (i.e., as a shorthand for  $f(x; \theta)$ ). At other times we will use  $\hat{y}$  for the estimated output. The meaning should be clear from the context.



**Figure 4:** A popular paradigm in deep learning involves an encoder  $E_\theta$  that maps the input into some latent space followed by a decoder  $D_\phi$  that maps from the latent space to the output space. This has numerous applications from language translation to image generation.

$\theta$  that minimize  $L$  over all possible parameters (i.e., the “hypothesis” class). To prevent overfitting on the training dataset or to incorporate prior information into the model we often include a **regularization term** on  $\theta$ ,

$$L(\theta; \mathcal{D}) = \sum_{i=1}^N \ell(f(x^{(i)}; \theta), y^{(i)}) + \lambda R(\theta) \quad (2)$$

where  $\lambda$  is a hyperparameter that controls the regularization strength. This is known as regularized empirical risk minimization. The most common regularization function is the squared-norm of the parameters,  $R(\theta) = \frac{1}{2}\|\theta\|_2^2$ , which has a nice theoretical interpretation and also results in a simple modification to gradient descent based update rules.

A very common pattern in deep learning is the **encoder-decoder** paradigm. Here one model, the encoder, maps from the input space  $\mathcal{X}$  to some latent space  $\mathcal{Z}$ . A second model then maps from the latent space to the output space  $\mathcal{Y}$ , as shown in Figure 4. The encoder and decoder models each have their own parameters and can be trained together or separately. Usually the latent space is smaller than the input or output spaces, and hence acts to compress the representation of the data. As a graphical illustration of this, we often draw encoder functions ( $E$ ) and decoder functions ( $D$ ) as follows,



Parameters are often omitted from these diagrams for brevity. Owing to the shape when the diagrams are combined, encoder-decoder models are sometimes called **hourglass** models.

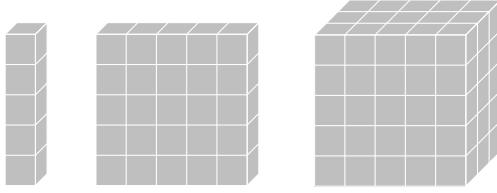
Encoder-decoder models can be divided into **auto-encoders**, where the input and output spaces are the same and **cross-encoders**, where the input and output spaces are different. One use of an auto-encoder is to learn a low-dimensional (and sometimes sparse) representation of the data. We do not need explicit supervision for this task since the aim is to reconstruct the input  $x$  from its encoded version  $z = E(x; \theta)$ , i.e., we want  $y = D(E(x; \theta); \phi)$  to be close to  $x$ . The latent variable  $z$  is called a **bottleneck** between  $x$  and  $y$ .

Cross-encoders have numerous applications. Indeed, many of the recent advances in deep learning and generative AI can be considered as a cross-encoder. With convolutional neural network (CNN) encoders and decoders we can implement semantic segmentation and style transfer algorithms; with recurrent neural network (RNN) encoders and decoders we can implement language translation models; and with a mixed CNN encoder and RNN decoder we can implement image captioning. Multiple different encoders can also be used to learn **joint embedding** spaces for images and language (e.g., CLIP [25] and BLIP [16]), allowing us to map from one semantic space to another.

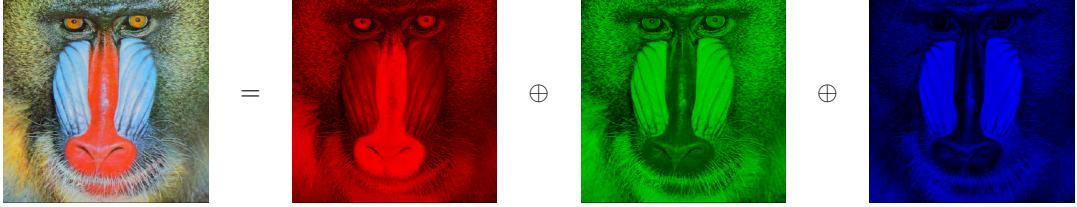
### 1.2.1 Ingredients of a Machine Learning System

There are several ingredients that make up a machine learning system. They should be considered carefully whenever designing, debugging or deploying a system for a specific task. Perhaps the most obvious ingredient is the **data**, which includes the input signals (i.e., set of images) and the target labels for supervised tasks. The data is used for training as well as evaluating the system, and we need to be sure there is sufficient quantity and quality for both of these roles. How these are **represented**, that is what datastructures are used for storing and processing, plays an important part of the system design. For example, labels can be represented as text strings, integers (each indexing a list of categories), or one-hot vectors; videos can be stored in a compressed encoded format or as a sequence of extracted frames.

Next, the **model architectures** (also known as networks) and **training objectives** (i.e., losses and priors), which we touched on above, are central to the workings of the system. Equally important is the **learning algorithm** used



**Figure 5:** The basic datastructures in deep learning are vectors, matrices and tensors.



**Figure 6:** A colour image is represented using a third-order tensor composed of red, green and blue channels. (Mandrill image from USC Image Database, <https://sipi.usc.edu/database/>)

to tune the model’s parameters using the training data and guided by the training objective. A crucial element of the learning algorithm for deep learning systems is how the parameters are initialised as this can have a big affect on the resulting performance and convergence rate of the algorithm.

Last, is the **evaluation metrics**. These map our judgment of how a system should perform into quantifiable scores. Using a single measure of performance can often be misleading. For example, reporting detection rates for a positive class (e.g., detecting cancer) and ignoring false positives can be misled by a system that always returns positive. It is important to think critically of the evaluation metrics, what they mean and where they may fail.

Each one of these ingredients—data, representations, architectures, objectives, and metrics—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 computer vision and deep learning.

### 1.3 Data Representation and Linear Algebra Basics

Deep learning builds heavily on linear algebra (and a bit of differential calculus) for much of its numerical processing. Linear algebra is an incredibly rich area of mathematics and we will only review a few key topics here. We recommend the following online resources for students interested in delving more deeply into the field:

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

The first thing we need to consider is how data is represented and stored. 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, and a **tensor** is an arbitrary-dimensional array of numbers (see Figure 5). The number of dimensions is called the **order** of the tensor, so an first-order tensor is just a vector.

The term **dimension** can also be used to specify the number of entries a vector, e.g., a 3-dimensional vector, but the meaning should be clear from the context. In deep learning we tend to describe vectors, matrices and tensors by their **size** or **shape**, e.g., a matrix with two rows and three columns, that is, a 2-by-3 matrix, has shape  $(2, 3)$ . A matrix with the same number of rows as columns is said to be **square**.

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 integer values in the range 0–255 or floating-point values in the range 0.0–1.0 (see Figure 6). The order in which data is stored is not consistent across software frameworks: In OpenCV and **numpy** images have shape  $(H, W, 3)$ , whereas in PyTorch they have shape  $(3, H, W)$ , where  $H$  and  $W$  are the height and width of the image, respectively.

We will often refer to a colour image as a **3-channel** or **3-plane** image. Instead of colour, which can be thought of as a 3-dimensional vector, we can generalize to  $C$ -dimensional feature vectors for each pixel assembled into a **feature map** of size  $(C, H, W)$ , also called a  $C$ -channel feature map. A video then can be represented using a fourth-order tensor with shape  $(3, H, W, T)$ , where  $T$  denotes the number of frames in the video.

### 1.3.1 Transpose

Given an  $m \times n$  matrix,  $A$ , we can define its transpose,  $A^T$ , as the  $n \times m$  matrix where the rows and columns of  $A$  have been swapped. For example,

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} 8 & 5 \\ 3 & 6 \end{bmatrix} \quad A^T = \begin{bmatrix} A_{11} & A_{21} \\ A_{12} & A_{22} \end{bmatrix} = \begin{bmatrix} 8 & 3 \\ 5 & 6 \end{bmatrix} \quad (3)$$

It should be clear from this definition that the transpose of a column vector is a row vector and vice versa, e.g.,

$$a = \begin{bmatrix} 8 \\ 5 \\ 3 \\ 6 \end{bmatrix} \quad a^T = [8 \ 5 \ 3 \ 6] \quad (4)$$

A square matrix that is equal to its transpose is called **symmetric**. The following properties of transpose are true:

- $(A^T)^T = A$
- $(AB)^T = B^T A^T$
- $(A + B)^T = A^T + B^T$

The idea of transpose extends to tensors where we then also need to specify exactly which channels are being swapped.

### 1.3.2 Special Vectors and Matrices

Some special vectors up often and can be used to simplify notation, e.g., the all-zeros vector  $\mathbf{0}_n$  and all-ones vector  $\mathbf{1}_n$ . The canonical vector  $e_i$  is a vector with all zeros except for the  $i$ -th location, which contains a one. This sometimes also called an indicator vector or one-hot vector and can be used to denote a category or class label in classification problems.

The same ideas extend to matrices, e.g.,  $\mathbf{0}_{m \times n}$ ,  $\mathbf{1}_{m \times n}$  and  $E_{ij}$  for the zero matrix, all-ones matrix, and indicator matrix with a one for the  $(i, j)$ -th component and zeros elsewhere, respectively. Another special matrix that you should know about is the **diagonal matrix** which is a square matrix where the off-diagonal entries are all zeros. The diagonal entries may or may not be zero. When the diagonal entries are all ones the matrix is called the identity matrix and denoted by  $I_{n \times n}$ . The identity matrix serves the role of one in scalar arithmetic—multiplying by the identity matrix copies the multiplicand to the output.

Subscripts denoting the size of the (zero, ones and identity) vectors and matrices are often omitted when it is clear from the context.

### 1.3.3 Addition and Multiplication

Two matrices of the *same size* can be added together in a componentwise fashion so that  $C = A + B$  means

$$C_{ij} = A_{ij} + B_{ij} \quad \forall i = 1, \dots, m \text{ and } j = 1, \dots, n \quad (5)$$

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

$$y_i = \sum_{j=1}^n A_{ij}x_j \quad (6)$$

for  $i = 1, \dots, m$ . The cost of matrix-vector multiplication is  $O(mn)$ .

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

$$C_{ij} = \sum_{k=1}^n A_{ik}B_{kj} \quad (7)$$

for  $i = 1, \dots, m$  and  $j = 1, \dots, p$ .

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  $B$ . However, matrix multiplication is associative,  $(AB)C = A(BC)$ , and distributive,  $A(B + C) = AB + BC$ . Matrix multiplication is not, in general, commutative,  $AB \neq BA$ .

The cost of matrix multiplication is  $O(mnp)$ .

Consider a matrix  $A \in \mathbb{R}^{m \times n}$  and vector  $x \in \mathbb{R}^n$ . 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  $A$  (denoted by  $a_i^T \in \mathbb{R}^{1 \times n}$  here) 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  $A$  (denoted by  $a_i \in \mathbb{R}^m$  here for the  $i$ -th column)

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

### 1.3.4 Inner Product and Euclidean Norm

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

$$\langle x, y \rangle = x_1 y_1 + \cdots + x_n y_n \quad (8)$$

$$= x^T y \quad (9)$$

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  $\mathbb{R}^n \rightarrow \mathbb{R}$ .

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

$$\|x\|_2 = \sqrt{x_1^2 + \cdots + x_n^2} \quad (10)$$

$$= \sqrt{x^T x} \quad (11)$$

The quantity  $\|x\|_2$  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 (vector) norms are  $\|x\|_1 = |x_1| + \cdots + |x_n|$  and  $\|x\|_\infty = \max\{x_1, \dots, x_n\}$ .

### 1.3.5 Batch Processing and Broadcasting

Deep learning frameworks process data in batches, passed as tensors. The first dimension of the tensor is the batch dimension. So, for example, a batch of  $n$ -dimensional vectors has shape  $(B, N)$ , and a batch of colour images has shape  $(B, 3, H, W)$ .

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

$$X = \{x^{(1)}, \dots, x^{(B)}\} \quad (\text{input}) \quad (12)$$

$$Y = \{Ax^{(1)} + b, \dots, Ax^{(B)} + b\} \quad (\text{output}) \quad (13)$$

Many PyTorch functions are batch-aware and support broadcasting where data along singleton dimensions is automatically replicated to match arguments within an operation, e.g., for  $(M \times N)$ -matrix  $A$ ,  $M$ -vector  $b$ , and batch of  $N$ -vectors  $x$ ,

```

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

```

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

### 1.3.6 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 many ways to create a tensor. For example,

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

creates the matrix

$$A = \begin{bmatrix} 8 & 5 \\ 3 & 6 \end{bmatrix} \quad (14)$$

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

It will often prove useful to be able to **reshape** a tensor when implementing deep learning models. Reshaping is always possible so long as the total number of elements does not change, e.g.,

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

creates matrices

$$A = \begin{bmatrix} 8 & 5 & 1 \\ 3 & 6 & 2 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 8 & 5 \\ 1 & 3 \\ 6 & 2 \end{bmatrix} \quad (15)$$

Note that this is different to transpose, which would create

$$A^T = \begin{bmatrix} 8 & 3 \\ 5 & 6 \\ 1 & 2 \end{bmatrix} \quad (16)$$

from PyTorch code

```
1 A.transpose(0, 1)
```

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

$$C = [8 \ 5 \ 1 \ 3 \ 6 \ 2] \quad (17)$$

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

### 1.3.7 Coordinate Systems

A final comment on indexing and coordinate systems. In mathematics we generally index from 1, whereas in code we generally index from 0. This is a source of endless confusion and bugs. Now you know you wont fall into this trap.

To add further confusion, in linear algebra we index matrices and tensors from top-left to bottom-right, i.e., the top-left element of an  $m \times n$  matrix is the  $(1, 1)$ -th entry and the bottom-right element is the  $(m, n)$ -th entry. Since we represent images as tensors, this will also be the default indexing for pixels. So the top-left pixel is the first pixel in the image. However, in geometry, coordinate systems increase upwards from the origin. In this coordinate system it is the bottom-left pixel that is the first pixel in the image. Be aware that different tools and different authors may use different coordinate systems. If you view an image and everything is upside down then flip the coordinate system.



## 1.4 Further Reading and Resources

The following is a list of recent textbooks that augment that material covered in these lecture notes:

- Goodfellow et al., *Deep Learning*, 2016 (<http://www.deeplearningbook.org/>)
- Zhang et al., *Dive into Deep Learning*, 2023 (<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>)

You should also familiarize yourself with the PyTorch deep learning library (<https://pytorch.org/>), which has excellent documentation and tutorials to work through.

Finally, there are some very high quality lectures and courses online from various institutions, including:

- 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/>
- University of Amsterdam (Lippe), <https://uvadlc-notebooks.readthedocs.io/en/latest/>

## References

- [1] M. Abadi and et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.
- [2] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, G. Necula, A. Paszke, J. VanderPlas, S. Wanderman-Milne, and Q. Zhang. JAX: composable transformations of Python+NumPy programs, 2018.
- [3] N. Carion, A. Kirillov, F. Massa, G. Synnaeve, N. Usunier, and S. Zagoruyko. End-to-end object detection with transformers. In *ECCV*, 2020.
- [4] M. Caron, H. Touvron, I. Misra, H. Jégou, J. Mairal, P. Bojanowski, and A. Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021.
- [5] X. Chen and K. He. Exploring simple siamese representation learning. In *CVPR*, 2021.
- [6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.
- [7] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *ACL*, 2019.
- [8] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.
- [9] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.
- [10] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. Richemond, E. Buchatskaya, C. Doersch, B. Avila Pires, Z. Guo, M. Gheshlaghi Azar, et al. Bootstrap your own latent—a new approach to self-supervised learning. *NeurIPS*, 2020.
- [11] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.
- [12] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.
- [13] D. P. Kingma and M. Welling. Auto-encoding variational bayes. In *ICLR*, 2014.
- [14] D. P. Kingma and M. Welling. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, 12(4):307–392, 2019.
- [15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *NeurIPS*, 2012.
- [16] J. Li, D. Li, C. Xiong, and S. Hoi. BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *CVPR*, 2022.
- [17] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. In *ICLR*, 2013.
- [18] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013.
- [19] A. v. d. Oord, Y. Li, and O. Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.
- [20] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, M. Assran, N. Ballas, W. Galuba, R. Howes, P.-Y. Huang, S.-W. Li, I. Misra, M. Rabbat, V. Sharma, G. Synnaeve, H. Xu, H. Jegou, J. Mairal, P. Labatut, A. Joulin, and P. Bojanowski. DINoV2: Learning robust visual features without supervision, 2023.
- [21] G. Papamakarios, E. Nalisnick, D. J. Rezende, S. Mohamed, and B. Lakshminarayanan. Normalizing flows for probabilistic modeling and inference. *JMLR*, 22(1), Jan 2021. ISSN 1532-4435.
- [22] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NeurIPS Autodiff Workshop*, 2017.
- [23] J. Pennington, R. Socher, and C. Manning. GloVe: Global vectors for word representation. In *EMNLP*, 2014.
- [24] S. J. Prince. *Understanding Deep Learning*. The MIT Press, 2023.

- [25] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.
- [26] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.
- [27] F. Rosenblatt. *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms*. Spartan Books, 1962.
- [28] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *NeurIPS*, 2015.
- [29] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
- [30] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian, and P. Mahoney. *Stanley: The Robot That Won the DARPA Grand Challenge*, pages 1–43. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-73429-1\_1.
- [31] H. van Hasselt, A. Guez, and D. Silver. Deep reinforcement learning with double Q-learning. In *AAAI*, 2015.
- [32] V. N. Vapnik. *Statistical Learning Theory*. Wiley-Interscience, 1998.
- [33] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, 2017.
- [34] B. Widrow and M. E. Hoff. Adaptive switching circuits. *IRE WESCON Convention Record*, 1960.
- [35] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2023.