Chapter 1

**Introduction to**

**DEEP LEARNING**

*Introduction to Neural Networks*

*Supervised Learning with Neural Networks*

*Why is Deep Learning taking off?*

**1.1 DEEP LEARNING**

* **What is Deep Learning?**

Deep learning is a specific subfield of machine learning: a new take on ***learning representations from data*** that puts an emphasis on "*learning successive layers of increasingly meaningful representations*".

* The “***deep***” in “deep learning” ***isn’t*** a reference to any kind of ***deeper understanding*** achieved by the approach; rather, it stands for this idea of ***successive layers of representations***. How many layers contribute to a model of the data is called the depth of the model.
* Other appropriate names for the field could have been ***layered representations learning*** or ***hierarchical representations learning***.
* Deep learning and Shallow Learning: Modern deep learning often involves tens or even ***hundreds*** of ***successive layers of representations***, and they’re all ***learned automatically*** from exposure to ***training data***. Meanwhile, other approaches to machine learning tend to focus on learning only one or two layers of representations of the data (say, taking a pixel histogram and then applying a classification rule); hence, they’re sometimes called ***shallow learning***.
* In deep learning, these ***layered representations*** are learned via models called ***neural networks***, structured in literal layers stacked on top of each other.
* The term “***neural network***” refers to ***neurobiology***, but although some of the central concepts in deep learning were developed in part by drawing ***inspiration*** from our ***brain*** (in particular, the visual cortex), ***deep learning models*** are ***not models of the brain***.
* There’s ***no evidence*** that the ***brain*** implements anything like the ***learning mechanisms*** used in ***modern deep learning models***.
* Deep learning is just a mathematical framework for learning representations from data. [Chollet, Deep Learning with Python]
* **Key characteristics of deep learning include:**

1. Hierarchical Learning: Deep learning architectures consist of ***multiple layers of interconnected neurons***, allowing them to learn *hierarchical representations of data*. Each layer extracts progressively more ***abstract features*** from the input data.
2. Feature Learning: Deep learning algorithms automatically learn ***relevant features*** from raw data, eliminating the need for ***manual feature engineering***. This ability to learn representations directly from data is one of the *key advantages of deep learning*.
3. End-to-End Learning: Deep learning models can be trained ***end-to-end***, meaning they take ***raw data*** as ***input*** and directly ***output*** ***predictions*** or ***decisions*** without the need for ***intermediate processing steps***.
4. Scalability: Deep learning models can scale to handle ***large*** and ***complex datasets***, making them suitable for a *wide range of applications*, including *computer vision*, *natural language processing*, *speech recognition*, and more.

**ChatGPT (What is 'Deep Learning'?) :**

**Deep learning** is a subset of **machine learning** that focuses on ***artificial neural networks*** and algorithms inspired by the structure and function of the **human brain**, known as artificial neural networks (ANNs). It involves training these neural networks with large sets of labeled data to make sense of patterns and features in the data, allowing them to make predictions, classifications, or decisions without being explicitly programmed for each task.

Some popular deep learning architectures include:

* **Convolutional Neural Networks** (CNNs) for image recognition,
* **Recurrent Neural Networks** (RNNs) for sequential data processing, and
* **Transformers** for natural language processing tasks.

***Deep learning*** *is a specialized area within the field of machine learning that involves training* ***artificial neural networks*** *to learn* ***intricate patterns*** *and* ***representations*** *directly from raw data. Unlike traditional machine learning approaches that often require manual* ***feature engineering****, deep learning algorithms* ***automatically******learn******hierarchical representations*** *through* ***multiple layers of interconnected Neurons****.*

*Deep neural networks effectively capture complex relationships in data.*

*Deep learning has shown remarkable success in various domains, including computer vision, natural language processing, and speech recognition, leading to advancements in tasks such as image classification, language translation, and voice synthesis.*

* **The reason why Deep Learning is so valuable today**

AI has many tools beyond ***Machine learning*** and ***Deep learning***. (Details on Stanford CS 221; Artificial Intelligence: Principles and Techniques). Let's consider the following branches of AI:

|  |  |
| --- | --- |
| * Deep Learning (DL / NN): *Deep learning* is a *method of machine learning* that teaches computers to understand and analyze data by using ***artificial*** ***neural networks*** with multiple ***layers***. * Machine Learning (ML): *Algorithms* and *techniques* that enable computers to ***learn from data*** and improve their performance over time, including *supervised* learning, *unsupervised* learning, *reinforcement* learning, and *deep learning* (DL is a subset of ML). * PGM (Details: Stanford CS 228 - Probabilistic Graphical Models): Probabilistic Graphical Models (PGM) are a way of *representing* and *reasoning* about *uncertainty in systems*. They use *graphs* to show *relationships* between *different variables* and *probabilities* to ***model uncertainty***. * Planning Algorithm: Planning algorithms typically involve ***modeling*** the ***environment***, the agent's capabilities, and the possible actions it can take. The algorithm then ***searches*** through ***possible sequences of actions*** to find a plan that leads to the desired goal while satisfying any constraints or objectives.   ***Eg:*** for self driving car we need different planning algorithms for motions.   * Planning and Decision Making: Techniques for generating sequences of actions to achieve goals or solve problems, including automated planning, reinforcement learning, and Markov decision processes. * Planning Agents: Planning agents are like decision-makers that figure out the best steps to take to reach a goal. They analyze the situation, consider possible actions, and then choose the actions that will lead to the desired outcome. * Search Algorithms: Techniques like *depth-first search*, *breadth-first search*, *A\* search*, and *heuristic search* algorithms are used to ***explore*** and ***navigate problem spaces*** to ***find solutions***. | * Knowledge Representation: It’s another interesting field. It’s basically knowledge graphs. But it's mostly used in industries but often under-appreciated in academia. * Knowledge representation involves the process of ***organizing*** and ***structuring*** information in a way that ***computers can understand*** and ***reason*** with. * ***Knowledge graphs*** are one method of knowledge representation, where entities (such as objects, concepts, or individuals) are represented as nodes, and the relationships between them are represented as edges.   E.g., if you search a web engine for room price with Wi-Fi, swimming pool, etc., that’s actually a knowledge graph (knowledge representation) done by the web engine.  Knowledge Representation is used by many companies which have huge databases. Sadly, it's underappreciated in academia.   * Game theory: Game theory is applied in AI to model and analyze strategic interactions between rational agents. It provides a mathematical framework for understanding and predicting the behavior of intelligent decision-makers in competitive or cooperative situations. Here are some ways game theory is applied in AI: * Multi-Agent Systems * Strategic Decision Making * Game Playing * Auction Design * Cooperative Decision Making * Security and Defense * Reinforcement Learning |

* **Applications of Machine Learning & Deep-Learning:**

|  |  |
| --- | --- |
| * Health-care * Robotics * Computational Biology | * Machine Translation System * Face Recognition system * Art-Generation system |

* Among other tools of AI, ***Deep Learning*** and ***Machine Learning*** have dramatically improved in performance. This is due to the *increasing scale* of *data* and *computing power* each year (i.e., we can access vast amounts of data and feed it to large neural networks for better performance). Additionally, more people are becoming involved in Deep Learning compared to other branches of AI.
* We notice more and more *artificial intelligence* in different industries nowadays, but we're not there yet. Just because a company uses *Neural Networks* doesn't mean it's an *Artificial Intelligence Company*. The top AI companies are skilled at:
* Gathering data smartly,
* Organizing it,
* Finding ways to automate tasks, and
* Hiring experts in cutting-edge technology.

AI > ML > Representation Learning > DL .

* "Artificial Intelligence: Principles and Techniques" encompasses a broad range of concepts and methodologies aimed at developing intelligent systems capable of performing tasks that typically require human intelligence. Here are some examples:
* Other subfields of AI are:
* Expert Systems: Expert systems are AI systems that emulate the decision-making abilities of human experts in specific domains. They use rules and knowledge bases to provide advice, make decisions, or solve problems.
* Multi-Agent Systems: Multi-agent systems involve the study of systems composed of multiple interacting agents that can perceive their environment, make decisions, and take actions to achieve individual or collective goals.
* Reinforcement Learning: Reinforcement learning is a subset of machine learning that focuses on learning optimal decision-making strategies by interacting with an environment and receiving feedback in the form of rewards or penalties.
* Cognitive Computing: Cognitive computing aims to simulate human thought processes by combining AI techniques with insights from cognitive psychology and neuroscience. It focuses on building systems that can understand, reason, and learn in a human-like manner.
* Knowledge Representation and Reasoning: Methods for representing knowledge in a computer-understandable format, such as logic-based representations, semantic networks, and ontologies, along with reasoning mechanisms for deriving conclusions from the represented knowledge.
* Natural Language Processing (NLP): Techniques for analyzing and generating human language, including syntactic and semantic parsing, sentiment analysis, named entity recognition, machine translation, and dialogue systems.
* Computer Vision: Methods for enabling machines to interpret and understand visual information from the real world, such as image classification, object detection, image segmentation, and image generation.
* Robotics: Methods for designing, constructing, and programming robots to perform various tasks autonomously or with human assistance, including robot motion planning, localization and mapping, and robot control.
* Genetic Algorithms and Evolutionary Computing: Optimization techniques inspired by the process of natural selection and genetics used to evolve solutions to optimization and search problems.
* **" Practical-know-how " vs "** **Technical Aspects "**

In this course we'll also focus on ***practical-know-how aspects*** of Machine learning not just the ***technical aspects***. What we can learn from the *internet*, *pure academic sources*, and *research papers* is a lot about the ***technical aspects*** of ML and DL. However, there are also other ***practical aspects*** concerning how to make those ***ML-DL algorithms*** work.

* For example: There's a huge difference between a "Junior Software Engineer" and a "Senior Software Engineer" (Book of John Ousterhout: Software Architecture). Everyone understands C++/Python/Java syntax, but there are also *high-level judgment decisions* like:
* How to architect the systems?
* Which kinds of abstractions to use?
* How to define interfaces?

that define a "Good Software Engineer" vs a "Less Experienced Software Engineer"; it's not just understanding the C++ syntax.

* Similarly, in the ML-DL field, there are many ways to learn the *technical tools of ML-DL* (such as how to *train NN*, the latest *optimization algorithms*, understanding *ConvNets*, *RNNs*, *LSTMs*, *attention models*, and applications in *NLP*, *computer vision*, *speech recognition*, etc.). Simultaneously, with these ***technical aspects***, we need to know the ***"practical know-how"*** so that when we're building an ML system, we can efficiently decide things like:
* ***Should we collect more data or not?*** (The answer is not always yes). However, more data is always good (it won't hurt us), but it won't always help us.
* ***So for an ML team, sometimes they have to decide:***
* Should we spend time collecting more data for the next 1 or 2 weeks, or
* Should we tune hyper-parameters for the NN?

Making these kinds of decisions correctly will make your team *2 times or even 10 times more efficient*. So our goal is to *gain this kind of knowledge* as well as learning the ***technical details***.

* The difference between a ***Good ML team*** and a "***Less experienced ML team***" is not just knowing how to implement things like LSTM in TensorFlow or Keras; you have to know those things, but there are other things to master so that we can lead an ML engineering team more efficiently.

***mlyearning.org:*** *This is a book by Andrew Ng, written over a few months. It turns ML principles from a black art into a systematic engineering discipline. Get the book: MACHINE LEARNING YEARNING: Technical Strategy for AI Engineers, In the Era of Deep Learning. We'll use this book in this course as well.*

* **What is it that makes a great AI company?**

Let's begin with a similar question: ***What makes an internet company?***

* The internet era:

Let's consider a shopping mall that *builds its own website*. That website doesn't make the shopping mall an internet company.

* The shopping mall can sell its goods on the website, but there's a *huge difference between a shopping mall with a website compared to a true internet* *company like Amazon or Alibaba*.
* A shopping mall plus a website does not make it an internet company. Any shopping mall can have its website and sell products online, but it doesn't make them an internet company like Amazon.
* ***So what's the difference?***
* Defining an internet company by its website is wrong; it's about their team.
* You have to organize your team or company to do things that the internet lets you do pretty well: What defines an internet company is not just their website, but how they organize their *team* or *company* to do the things that the *internet lets you do really well*. For example, the following things can be done better using the internet:
* ***For example:***
* A/B testing: Internet teams engage in pervasive A/B testing. For instance, they can launch two *versions of websites* simultaneously and *determine* which one works *better*, allowing for *faster learning*.

For traditional shopping malls, conducting A/B testing isn't feasible as you cannot have *two shopping malls* in *parallel* *universes*. (Traditional shopping malls usually redesign every three months.)

* ***Short shipping times*** enable ***quicker learning***. Products are shipped daily or weekly, facilitating faster feedback.
* ***Teams are organized differently***, with decision-making often led by *engineers* or *engineers and product managers*. In contrast, *traditional shopping companies* have decision-making led by ***CEOs***. However, in the ***internet era***, decisions are made by *individuals with a better understanding of technology*, such as ***engineers*** or ***product managers***.
* AI era (Rise of AI in multiple companies):

Simply using ***Neural Networks*** in a traditional company doesn't automatically make them an AI company. What defines a great AI team is:

* Their ability to organize their own work and
* Also organize their team's work to leverage the capabilities of AI, ML, and DL effectively.

Most AI companies haven't completely figured out the principles to organize their AI teams. Some potential principles could include:

* ***The team has to be very good at "strategic data acquisition":*** You may observe that some AI companies engage in practices that may *not always make sense to you (user)*. For instance, *they may offer free products*, prompting the question: ***Why don't they make any money from these free products?***
* However, some of these products are designed to *acquire data*, enabling *monetization through other means* such as ***advertising*** or ***gaining*** ***insights into user behavior***.
* Therefore, there are *various data acquisition strategies* that may *not make sense at first glance*, but they can be valuable when applied with ***Deep Learning Algorithms***.
* ***Unified data warehouses:*** AI companies strive to organize data differently. The ***AI team*** has to excel at ***consolidating all data***.
* Before the rise of Deep Learning, many companies had *fragmented data warehouses*. If one company has *50 large databases across 50 different divisions*, it's very difficult for an ***engineer*** to ***consolidate all the data*** to train a ***Learning Algorithm*** for valuable tasks.
* Therefore, leading AI companies tend to have ***unified data warehouses***. Many of the big companies today are investing in unified data warehouses to lay the foundations for learning algorithms.
* *They tend to be very good at spotting pervasive automation opportunities:* They should excel at identifying opportunities where *instead of having people perform a task, a Deep Learning algorithm or AI can handle that task*.
* *They also have new job descriptions:* They can begin creating lots of new roles for engineers.
* Once, the world was so simple that we only needed a software engineer to do the job. But now, we need ***specialized people*** for ***internet***, ***front-end***, ***back-end***, ***mobile***, and other roles like ***QA***, ***DevOps***, and ***IT***. In other words, we need people specialized in specific knowledge.
* So, due to the rise of AI, we start creating new job roles like:
  + ML engineer
  + ML research scientists
  + Product managers in AI teams
* **One more analogy:**

What are similar techniques in machine learning, like the software development techniques such as agile development, the waterfall model, or the Scrum process?

* Just as a software engineer takes a long time to understand
* *agile development* or the *pros and cons of the waterfall model vs. agile*.
* What is Scrum process?
* Is code review a good idea?

Even after different programming languages are invented, we still have to figure out various ways to help individuals and teams to write software effectively.

* So, if you're working with a high-performing industrial AI team in a big company, these software engineering practices (code review, agile or water-fall etc.) will help you.
* Building an effective software team involves more than just knowing programming syntax (C++/Python/Java, etc.).

|  |
| --- |
| * So, in the ***Machine Learning world***, we're still in the process of inventing these types of ***methodologies***: * What's the ***Scrum process*** for machine learning? * What's the equivalent of ***agile development*** for ML? * What's the equivalent of ***code review*** for developing ML algorithms?   We'll try to systematically learn these tools so that we're not only good at *deriving and implementing learning algorithms*, but also effective in terms of how we go about *building these systems*. |

* Related Courses: Consider these as Pokémon; you should collect them all.
* ***CS229:*** Machine Learning. It's the most mathematical course, delving deeper into the derivations of the algorithms.
* ***CS230:*** Deep Learning. Moderately mathematical and practical (somewhat in between CS229 and CS229A). It focuses on deep learning, practical know-how, and how to apply deep-learning algorithms. (Not enough math.)
* ***CS229A***: Applied Machine Learning. Less mathematical but more practical.

Deep learning is just a subset of machine learning. There are other tools like ***PCA***, ***K-Means***, ***Recommender Systems***, and support vector machines. These are not included in CS230 but in CS229.

* Use CS229, CS230, and CS229A as foundations, then go deeper to learn about *Computer Vision*, *NLP*, *Robotics*, or *Deep Reinforcement Learning*. First, take a quick overview of every ML subfield, like *Computer Vision*, *NLP*, *Speech Recognition*, *Self-Driving Cars*, and *Reinforcement Learning*, to get the foundation. Then, go deeper into the specific subfield (NLP, Robotics, or Deep Reinforcement Learning) you're interested in.
* Our objective for this course:
* Being Expert in Deep Learning algorithms, know the state of the arts
* Getting know-how to apply these algorithms to solve a problem

Software engineering methodologies

**Agile Development :**

***Agile development*** is a way of creating software that focuses on small, manageable pieces of work done in short cycles called sprints. This method allows teams to *quickly adapt to changes* and get *continuous feedback* *from users* and stakeholders. Key points include:

* **Iterative Work:** Develop software in small steps with regular updates.
* **Collaboration:** Work closely with all team members and stakeholders.
* **Flexibility:** Easily adapt to changes in requirements.
* **Continuous Improvement:** Regularly review and improve the process.
* **Frequent Feedback:** Get ongoing feedback to ensure the product meets user needs.

Common frameworks within Agile include **Scrum** and **Kanban**, each with its own practices and tools to support these principles.

* Agile development is a methodology used in software engineering that emphasizes ***iterative development***, ***collaboration***, and ***flexibility***. It is designed to accommodate changing requirements and foster continuous improvement. Here are some key aspects of agile development:
* ***Iterative and Incremental:*** Agile development breaks down a project into small, manageable units called ***iterations*** or ***sprints***, which typically last between ***one to four weeks***. Each iteration results in a working product increment that adds to the previous iteration.
* ***Collaboration:*** Agile emphasizes close collaboration between ***cross-functional teams***, including developers, testers, designers, and stakeholders. This helps ensure that the project is aligned with user needs and business goals.
* ***Flexibility:*** Agile methodologies are designed to *accommodate changes in requirements*, even ***late*** in the development process. This flexibility allows teams to respond to ***new information*** and ***evolving customer needs***.
* ***Customer Feedback:*** Frequent *feedback* from customers and stakeholders is integral to the agile process. This ensures that the product meets *user expectations* and can be adjusted based on *real-world usage* and *feedback*.
* ***Self-Organizing Teams:*** Agile teams are typically *self-organizing* and *empowered to make decisions*. This encourages ***ownership***, ***accountability***, and ***creativity*** among team members.
* ***Focus on Quality:*** Agile practices emphasize the importance of delivering *high-quality software* through *continuous integration*, *testing*, and *code reviews*.
* ***Transparency and Visibility:*** Agile methodologies promote *transparency* and *visibility* through *regular meetings* such as ***daily stand-ups***, ***sprint reviews***, and ***retrospectives***. These meetings help track ***progress***, address ***issues***, and ***improve*** processes.
* ***Common Agile Frameworks:***
* ***Scrum:*** Scrum is one of the most ***popular*** agile frameworks. It involves
  + specific roles (Scrum Master, Product Owner, and Development Team),
  + artifacts (Product Backlog, Sprint Backlog, and Increment), and
  + ceremonies (Sprint Planning, Daily Stand-up, Sprint Review, and Sprint Retrospective).
* ***Kanban:*** Kanban focuses on visualizing the workflow, limiting work in progress (WIP), and optimizing the flow of tasks. It uses a Kanban board to represent tasks and their status.
* ***Extreme Programming (XP):*** XP emphasizes technical excellence and includes practices such as pair programming, test-driven development (TDD), and continuous integration.

Overall, agile development aims to deliver functional software more quickly and efficiently while adapting to changing requirements and improving continuously through feedback and collaboration.

**Waterfall Model :**

|  |
| --- |
| The Waterfall Model is a **traditional** software development ***methodology*** that follows a sequential, linear approach to software development. In the Waterfall Model, the development process is divided into distinct phases, with each phase occurring in a sequential order, like water flowing downwards, hence the name "Waterfall." Here are the typical phases of the Waterfall Model:   * **Requirements Gathering:** The first phase involves *gathering* and *documenting the requirements* for the software system. This phase focuses on understanding ***what the software needs to do*** and ***what features*** it should have. * **System Design:** Once the requirements are gathered, the system design phase begins. In this phase, the overall system architecture and design are planned and documented. This includes defining the ***software's structure***, ***components***, ***interfaces***, and ***data flow***. * **Implementation:** After the system design is completed, the implementation phase begins. In this phase, the actual code for the software is written based on the design specifications. Programmers write the code following the design documents. * **Testing:** Once the code is implemented, the software undergoes testing to identify and fix any defects or bugs. Testing is typically divided into different types, such as:  1. unit testing (testing individual components), 2. integration testing (testing how components work together), and 3. system testing (testing the entire system).  * **Deployment:** After testing, the software is deployed to users or customers. This phase involves installing the software on the users' systems and ensuring that it works as expected in the real-world environment. * **Maintenance:** The final phase of the Waterfall Model is maintenance. In this phase, the software is maintained and updated as needed to address bugs, add new features, or make improvements based on **user feedback**. |

* One of the key characteristics of the Waterfall Model is its linear nature, where each phase must be completed before moving on to the next. This means that changes to requirements or design later in the development process ***can be costly and time-consuming to implement***, as they may ***require revisiting earlier*** ***phases***.
* While the **Waterfall M**odel was once **widely used**, many software development teams now prefer more flexible and iterative approaches, such as **Agile** **methodologies**, which allow for greater adaptability and responsiveness to change.

**Scrum process :**

"Scrum" is a concept called in software development, which is a widely used Agile framework. *Scrum* is a *lightweight and flexible approach* to software development that emphasizes iterative and incremental delivery.

* It *breaks down* the *development process* into small iterations called "sprints", typically lasting ***two to four weeks***.
* During each ***sprint***, a ***cross-functional team*** works *collaboratively* to deliver a potentially *shippable product increment*.
* In Scrum, there are defined roles, including the:
* Product Owner: The ***Product Owner*** is responsible for representing the stakeholders and prioritizing the work.
* Scrum Master: The ***Scrum Master*** serves as a facilitator and coach for the team.
* Development Team: The Development Team is responsible for delivering the work agreed upon in each sprint.
* Key ceremonies in Scrum include:
* ***Sprint Planning:*** At the beginning of each sprint, the team plans the work to be done and selects the items from the product backlog to work on.
* ***Daily Stand-up:*** *Daily meetings* where team members *synchronize* their *work* and discuss *progress*, *challenges*, and *plans* for the day.
* ***Sprint Review:*** At the end of each sprint, the team presents the completed work to stakeholders and gathers feedback.
* ***Sprint Retrospective:*** A meeting where the team reflects on the sprint and identifies areas for improvement.

Scrum promotes *transparency*, *inspection*, and *adaptation*, allowing teams to quickly respond to changes and deliver value to customers efficiently. It is widely used in various industries for managing complex projects and developing high-quality software products.

* Here are some more **examples** of **applications** of ML:
* **Web search:** There’s a learning algorithm that improves the search results and another that shows the most relevant results.
* **Voice search:** 10% of web searches are now voice searches, which heavily involve speech recognition.
* **Recommender systems:** Websites like Amazon and Netflix use ML to build recommender systems.
* **Fraudulent transaction detection:** ML is used to detect credit card fraud.
* **Spam detection:** Email companies use ML to detect spam and remove it from your inbox, so spam filters now use learning algorithms.
* **Shortest route:** While the *shortest route algorithm* provides the shortest route on a map, a *learning algorithm* predicts probable traffic on that route and calculates the ***estimated time***.
* **Healthcare:** ML is increasingly used to diagnose cancer and estimate infectious diseases.

All of the above cases use learning algorithms behind the scenes. These ML applications are now integrated into our everyday lives, including in agriculture and education.

|  |
| --- |
| In the *next two sections and the next chapter*, we'll learn about the *most important foundations*—the building blocks of deep learning. We'll learn how to build and get a deep neural network to work by following these steps:   * ***Step 1:*** * Introduction to deep learning and how it works. * Basics of neural network programming. * ***Step 2:*** * Learn the structure of the *forward propagation* and *backpropagation* steps of the algorithm. * *Implement neural networks* efficiently. * Complete programming exercises to *implement the algorithms* and see them work for yourself. * ***Step 3:*** * After learning the *framework* for *neural network programming*, we'll code a *single hidden layer neural network*. * Learn all the *key concepts* needed to *implement* and *operate* a neural network. * ***Step 4:*** * Finally, we'll build a *deep neural network* with *many layers* and see it work for ourselves. |

**1.2 NEURAL NETWORKS**

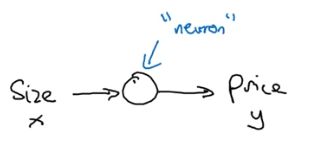
* **What is a Neural Network?**

The term, ***Deep Learning***, refers to *training Neural Networks*, sometimes very *large Neural Networks*. Let's consider the Housing Price Prediction example:

|  |  |
| --- | --- |
| * ***Dataset:*** Let's say you have a dataset with six houses. It consists of the size *of the houses* in square feet (or square meters) and the price *of the houses*. * ***Objective:*** *Fit a function* (a function of the size) to *predict* house *prices* based on their *sizes*. * ***Approach:*** We can use different kinds of approaches for this. The simplest one is the Linear Regression approach, where we'll simply fit a *straight line* to these *data points* as shown below. | * However, to be more practical, you might say that: *"we know prices can never be negative"*. So instead of using a *straight line fit*, which will eventually become negative, let's *bend the curve* here. This curve *ends at zero*, ensuring that prices do not go below zero. |

* ***Non-Negative Price Constraint:*** Following is the *adjusted curve with non-negative prices*, combining a *curve* that *flattens to zero* (curve remains at zero for smaller sizes) and a *linear fit for higher values*. The ***thick blue line*** represents your function for predicting house prices based on size.

|  |  |
| --- | --- |
| * ***Problem:*** Linear regression may predict negative prices, which is unrealistic. * ***Solution:*** Modify the curve to ensure prices do not go negative.   ***Simple Neural Network Representation:*** The adjusted function for predicting housing prices can be seen as a very simple neural network.   * We can think of this *function* that we've just *fit the housing prices* as *a very simple neural network*. It's almost as simple as possible neural network. Let me draw it here: |  |



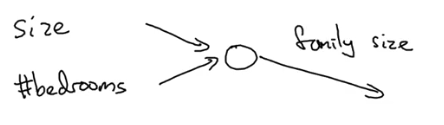
* ***How it works:*** The input to the neural network is the *size of a house*, which one we call **x**. It goes into the node, (the little circle) and then it outputs the *price*, which we call **y**.
* ***Neuron:*** The little circle represents a *single neuron* in this neural network. It implements the function (Linear Regression) that we drew above. And all the neuron does is:
* It inputs the size,
* Computes the linear function,
* Takes a max of zero i.e. , and
* Then outputs the estimated price.
* Note that, in neural network literature, you often see the following function a lot, which is very similar to the above function. This function stays *constant at zero for some time* and then *takes off as a straight line*. It's called ReLU (Rectified Linear Unit).

|  |  |
| --- | --- |
| * ReLU: Rectify means taking the **max** of 0, which is why you get a function shape like this   In other words, the ReLU function *"rectifies" the input* by ***setting*** any ***negative*** *values to* ***zero*** and ***passing*** through any ***positive*** *values* ***unchanged***.  This is a common *activation function* used in *neural networks* because it *introduces* ***non-linearity*** while being **computationally efficient** and easy to implement |  |

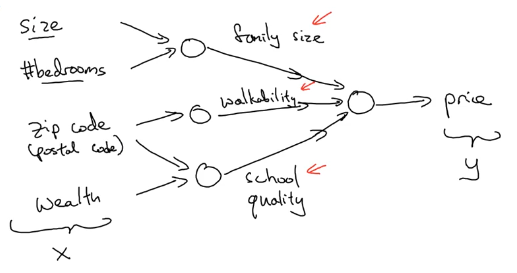
* + Mathematically, the ReLU function can be described as:
* We'll discuss the ReLU in the upcoming chapters.
* ***Larger neural network:*** In a *simple neural network*, a ***single neuron*** can be seen as the ***smallest building block*** (like a tiny Lego brick). A larger neural network is then formed by *combining* many of these *single neurons* and *stacking them together*.
* So, if you think of a neuron as a single Lego brick, you create a bigger neural network by stacking together many of these Lego bricks.

**Definition:** Neural networks refer to broad type of **non-linear models/parametrizations** hθ(x) that involve combinations of *matrix multiplications* and other entrywise *non-linear operations*. We will start small and slowly build up a neural network, step by step.

* ***Let's see an example:*** Instead of *predicting the price* of a house based solely on its *size*, you can have other features.
* You can use other things about the house, such as the number of bedrooms.
* You might think that one of the key factors affecting the price of a house is family size. For example, *Can a house accommodate a family of three, four, or five?*
* So the size in square feet or square meters, along with the number of bedrooms(#bedrooms), determines whether a house can fit your family's needs (i.e. family size). These two inputs are related.



* There can be other parameters, like the zip code (or postal code in different countries). The zip code can provide information about the neighborhood's walkability. For instance:
* Is the neighborhood *highly walkable*?
* Can you easily walk to the *grocery store* or *school*?
* Do you need to *drive*?
* Some people prefer highly walkable neighborhoods.
* The zip code can also be combined with other features like the area's wealth. For example, it might indicate the quality of schools in the area (certainly in the United States and in some other countries as well).

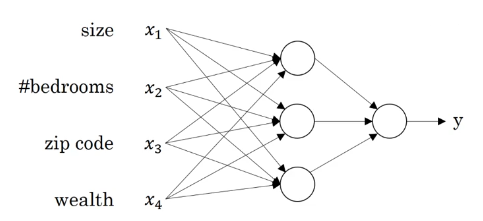


* Each of these *little circles* in above drawing can represent one of those ReLU (rectified linear units) or some other *slightly non-linear function*. So that
* Based on the size and number of bedrooms, you can estimate the family size.
* Using the zip code, you can estimate walkability.
* Based on the zip code and wealth, you can estimate school quality.
* Finally, people decide how much they are willing to pay for a house (price) by considering the *factors* that *matter to them*. In this case, family size, walkability, and school quality help predict the price.
* In the example, x represents all of these four inputs: size, number of bedrooms (#bedrooms), zip code, and wealth, y is the price (output) we're trying to *predict*.
* By *stacking* together a few of the ***single neurons*** or simple *predictors* from the previous example, we now have a slightly larger neural network.
* **Managing neural networks**

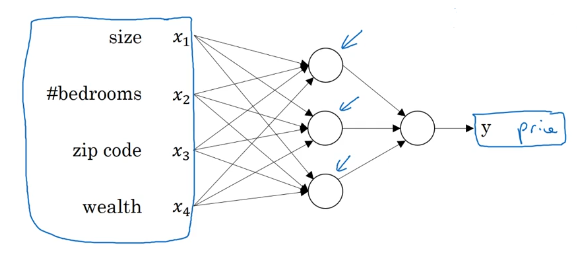
When you implement a neural network, you need to provide it with the **input x** and the **output y** for *several examples* in your *training set*. And all the *nodes (neurons)* in the middle, they will figure out the relationships by itself

|  |  |
| --- | --- |
| * What we actually implement is looks like below, where we have a neural network with four inputs. |  |

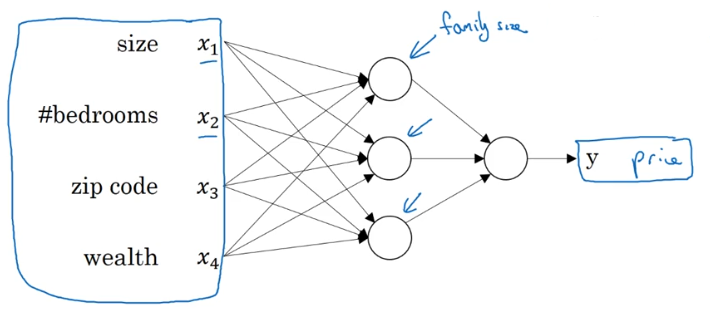
* The input features might be the size of the house, the number of bedrooms, the zip code or postal code, and the wealth of the neighborhood. Given these input features, the job of the neural network is to predict the price .



* Notice that each of the *circles* in the *middle* are called *hidden units* in the neural network. Each *hidden unit* takes *all four input features* as its input.



* For example, instead of saying these *first nodes* represent family size and that family size depends only on features and . Instead we gonna say, *"Neural network, you decide whatever this node should represent, and we'll give you all four input-features to compute whatever you want."*



* We also say that the "*input layer*" and the "*layer in the middle*" of the neural network are densely connected because every input feature is connected to every node (neuron) in the middle layer (hidden layer).
* One remarkable thing about neural networks is that, given *enough data* on both x and y, and *enough training examples* with both (x, y), neural networks are exceptionally good at *figuring out* functions that accurately map from x to y.

So, that's a basic neural network. It turns out that as you build your own neural networks, you will find them most useful and powerful in supervised learning settings. This means you're trying to take an input x and map it to some output y, using Neural Network.

In the following section, we'll discuss more examples of supervised learning where neural networks can be incredibly helpful.

**1.3 SUPERVISED LEARNING with Neural Networks**

There's been a lot of hype about neural networks, and some of that hype is indeed justified, given their impressive performance.

* However, it's important to note that almost all the *economic value* created by Neural Networks so far has been through one type of machine learning called- ***supervised learning***. Let's explore what that means with some examples.
* ***Goal of supervised learning:*** In supervised learning, you have some input x, and the goal is to *learn a function* that maps it to some output y.

So, we start with a *set of inputs*, each *paired* with a *corresponding output*. Our goal is to learn a *function* that accurately *maps* these *inputs* to their respective *outputs*.

* Here are some examples where **neural networks** have been applied very effectively:
* Price Prediction: We previously discussed the *housing price prediction* application, where you input some features of a home and try to predict or estimate the price
* Ad Selection: One of the most lucrative applications of deep learning today is in **online advertising**. Although it may not be the most inspiring, it is certainly very profitable.
* By inputting *information about an ad* and *details about the user*, neural networks have become highly effective at predicting *whether or not a user will click on an ad*.
* Showing users the ads they are *most likely to click* on has had a direct impact on the *revenue* of major *online advertising companies*.
* Image Recognition: Computer vision has also seen significant advancements in recent years, largely due to *deep learning*. For example, you might *input an image* and the neural network could *output a label*, such as an index from 1 to 1,000, identifying the object in the picture *(it might be any one of, say a 1000 different images of object)*. This is commonly used in photo tagging.
* Audio/Speech to Text: Recent progress in speech recognition has been very exciting. You can now input an audio clip into a *neural network* and receive a text transcript as the output.
* Translation: Machine translation has made huge strides thanks to deep learning. Now, a neural network can input a sentence in *English* and directly output a translated sentence in another language, such as *Chinese*.
* Autonomous Driving: In autonomous driving, you might input an *image* (e.g., a picture of what's in front of your car) along with *data* from a *radar*. A Neural Network can be trained to analyze this information and *determine the positions of other vehicles* on the road.
* Selecting Input x (image) and Output y (position of the car), is a critical component of autonomous driving systems. Much of the value created by neural networks has come from *cleverly selecting* what should be x and what should be y for a particular problem. This *supervised learning component* is often integrated into larger systems, such as autonomous vehicles."

|  |  |  |
| --- | --- | --- |
| Input | Output | Application |
| Features of a House | Price | Real Estate |
| Advertise, user info | Click on ad? (0/1) | Online Advertising |
| Image | Object (1, . . . , 1000) | Photo tagging |
| Audio | Text transcript | Speech recognition |
| English | Chinese | Machine translation |
| Image, Radar info | Position of other cars | Autonomous driving |

* **Different types of neural networks**

Different types of neural networks are suited for different applications. It turns out that *slightly different types of neural networks* are useful for different applications.

For example, in the real estate application we discussed earlier, a *standard neural network architecture* was used. This type of network is commonly applied in areas like real estate and online advertising.

* ***Image Applications:*** When it comes to *image-related tasks*, *Convolutional Neural Networks* (CNNs) are often preferred. CNNs are designed to process *grid-like data*, such as images, and they excel at tasks like *image recognition* and *classification*.
* ***Sequence Data (sound/audio):*** For tasks involving sequence data—where information is ordered over time, such as in *audio* (audio is played out over time, i.e. it has a temporal component) or *language processing*—Recurrent Neural Networks (RNNs) are commonly used.
* Audio, for example, is naturally represented as a *one-dimensional time series* (or as a one-dimensional temporal sequence), making RNNs a good fit.
* Similarly, *language data (like English or Chinese text)* is also sequential, with words or characters following one another in a specific order (the alphabets or the words come one at a time). More advanced versions of RNNs are often employed for such tasks.
* ***Complex Applications:*** For more complex applications, like *autonomous driving*, you may need a *hybrid neural network architecture*. For instance, an *image from a camera* might be processed using a *CNN*, while *radar data*, which is quite different in nature, might be handled by *another type of neural network*. Combining these different networks can result in a more sophisticated, custom architecture (complex, hybrid neural network architecture) tailored to the specific demands of the application.
* ***Neural Network Examples:*** To clarify the differences between Standard, CNN, and RNN architectures, let's look at some examples:

|  |  |  |
| --- | --- | --- |
|  |  |  |

* ***Standard /Artificial Neural Network (ANN):*** You might be familiar with this basic architecture, often depicted in diagrams showing *layers of nodes connected by lines*.
* ***Convolutional Neural Network (CNN):*** This type of network is widely used for *image processing* tasks. We'll delve deeper into CNNs in later chapters, where we'll explore how they work and how to implement them.
* ***Recurrent Neural Network (RNN):*** RNNs are good at handling *sequential data with temporal components*, such as audio or language (one-dimensional sequence data that has a temporal component). We'll learn about RNNs implementation in upcoming chapters.
* **Structured Data vs. Unstructured Data**

You may have heard about machine learning applications in both *structured* data and *unstructured* data.

|  |  |
| --- | --- |
| **Structured Data :**  Structured data refers to organized data , typically stored in databases.   * For example, in Housing Price Prediction, you might have a *database* where *each column* represents *features* like the size of a house or the number of bedrooms. * Another example is predicting whether a user will click on an ad, where you might have *structured data about the user (information about the user such as age) and the ad*, with the *label* (e.g., whether the ad was clicked) being the target to predict. * Information about the user (i.e. X), such as the *age* and some *information about the ad*, * Labels (i.e. Y)   In structured data, each feature—such as the size of a house, the number of bedrooms, or a user’s age & type of Ad—has a clearly defined meaning. | Housing price problem    Ad clicking data |
| **Unstructured Data :**  In contrast, unstructured data includes raw *audio*, *images*, or *text*, where the data *doesn't have a predefined structure*. For example, in *image* *recognition*, the *features* might be the *pixel values* in an image, or in *text processing*, the features could be the *individual words in a sentence*.   * Historically, computers have *struggled* more with *unstructured data* than with *structured data*. This is because humans are naturally adept at understanding *audio cues*, *visual images*, and *text*, making it challenging to replicate this understanding in machines. * ***Advances in Unstructured Data Processing:*** One of the most exciting developments with the rise of neural networks is that computers have become much better at *interpreting unstructured data*. Thanks to ***deep learning***, tasks like *speech recognition*, *image recognition*, and *natural language processing (NLP)* have seen significant improvements, leading to new applications that weren't possible just a few years ago. * You might often hear about the successes of neural networks in unstructured data in the media, as it's exciting when a neural network can, for instance, recognize a cat in an image—something we can easily relate to. |

**Economic Value of Structured Data:**

While the *breakthroughs* in *unstructured data* are impressive, it's important to note that a significant amount of the *economic value* created by *Neural* *Networks* has also come from improvements in processing *structured data*. This includes:

* More effective advertising systems,
* Better product recommendations, and
* Improved capabilities for analyzing large databases to make accurate predictions.
* ***Learning Both Structured and Unstructured Data Techniques:*** Throughout our discussions, we’ll cover *techniques* that apply to both *structured* and *unstructured data*. While we might focus more on examples involving *unstructured data* to explain *certain algorithms*, it's crucial to recognize that *neural networks are valuable for both types of data*. As you consider neural network applications for your own work, you’ll likely find opportunities to use them with both ***structured*** and ***unstructured*** data.

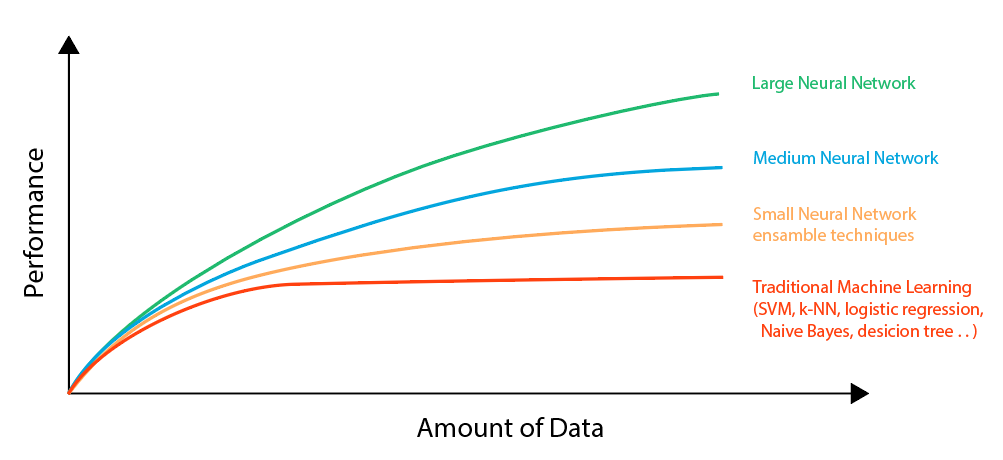
Neural networks have transformed supervised learning and are creating tremendous economic value across various industries.

**1.4 Why is Deep Learning taking off?**

Briefly discussed in Section 1.1. The *basic technical ideas* behind Neural Networks have been around for decades, so why are they only now becoming so powerful? Why are they only now gaining traction? In this section, we'll explore the key factors that have led to the recent success of neural networks.

* If the concepts behind deep learning and neural networks have existed for so long, why are they only now gaining traction? We'll go over the main drivers behind the rise of deep learning, which will help you identify the best opportunities to apply these tools within your organization.
* **Performance of a Model vs. Amount of Data**

Over the past few years, many have wondered why deep learning is suddenly performing so well. A useful way to understand this is through a simple diagram: on the horizontal axis, the amount of data available for a task is plotted, while the vertical axis represents the performance of a learning algorithm—such as the accuracy of a *spam classifier*, an *ad click predictor*, or a neural network that determines the *position of other* *cars for a self-driving car application*.



* When plotting the *performance* of a *traditional learning algorithm*, like a Support Vector Machine (SVM) or logistic regression, as a function of the amount of data available, the curve (red) typically shows that *performance improves as more data is added*.
* However, after a certain point, performance plateaus.
* These traditional algorithms struggle to effectively handle very large datasets.
* ***Vast amount of data:*** In the past decade, the amount of available data has grown significantly due to the digitization of society. Much of human activity now occurs in the digital realm—on computers, websites, and mobile apps—generating vast amounts of data.
* Additionally, the availability of inexpensive *cameras*, *sensors*, and *devices* connected through the Internet of Things (IOT) has led to the collection of even more data. As a result, many applications now have access to far more data than traditional learning algorithms can efficiently process.
* ***Use NN to handle vast amount of data:*** When training a *small neural network* (instead of traditional learning algorithm), its performance initially improves, but not significantly. A *larger* *neural network*, often referred to as a ***medium-sized network***, performs better, while a very large neural network tends to show continuous improvement as more data is provided.
* To achieve very high levels of performance you need two things:
* First, you need to be *able to train a sufficiently large neural network* capable of leveraging vast amounts of data
* Second, you need a lot of data
* This is why scale has been a major driver of progress in deep learning—scale in terms of *both* the *size of the neural network* *(with many hidden units, parameters, and connections)* and the *volume of data*.

One of the most reliable ways to improve a neural network's performance today is to either *increase its size (train a bigger network)* or provide *more data*. However, this approach has *limitations*, you may *run out of data*, or the *network* may become *too large*, making *training* time excessively *long*. Nonetheless, scaling up has significantly advanced deep learning.

* Note that in the above plot the **X-axis** represents the ***Amount Of Labeled Data***. By labeled data, we mean *training examples* where both the **input** and the **label** are available *i.e (input, label) training examples*.
* The *size of the training set* (or the number of training examples) is denoted by m, which corresponds to the **X-axis**, the ***Amount Of Labeled Data***

|  |  |
| --- | --- |
|  | * In the region of smaller training sets (toward the left side of the diagram), the relative performance of different algorithms is not well-defined. * That is, if the amount of data is small, traditional algorithms and neural networks, regardless of their *size* or *architecture*, tend to give similar results. In this case, a larger or better-designed neural network doesn't provide any advantage. * When there's *limited training data*, the *performance* often *depends* on how well features are *"****Hand-Engineered****"*. It's possible that in such cases, a ***well-crafted SVM*** with *good* ***Feature Engineering*** can *outperform* even ***Large Neural Networks***. * So, in this region (the left side of the plot), the *"relative ranking"* between algorithms *isn't well defined*, and success relies more on skill in feature engineering and other algorithm details. |

* However, in the ***big data region*** (the right side of the plot), with very large training sets, larger neural networks consistently outperform other methods.
* **Scale drives deep learning progress**
* Data
* Computation
* Algorithms
* In the early stages of modern deep learning, progress was driven by the *scale* of data and computation. The *ability* to train *large neural networks* on *CPUs* or *GPUs* played a key role in this advancement.
* In recent years, there has also been *significant algorithmic innovation*, especially in making neural networks *run faster*. These algorithms are more focused on computations.
* As a concrete example, one major breakthrough in neural networks was the shift from using the *sigmoid function* to the *ReLU function*. The sigmoid has regions where the *gradient* is *nearly zero*, which *slows down learning*. *ReLU*, on the other hand, avoids this issue and *speeds up training*.

|  |  |
| --- | --- |
|  |  |

* When using ***Gradient Descent***, if the gradient is near zero (as with the sigmoid function), the *parameters update very slowly*, which makes ***learning slow***. By switching the *activation function of the neural network* to the *ReLU (Rectified Linear Unit)* function, where the *gradient is 1 for positive input values*, learning becomes *faster*. The gradient doesn't shrink to zero as easily (also slope is zero on the left side), and this simple *algorithmic change* has made *Gradient Descent* much more efficient, greatly speeding up neural network training.
* There are many examples that show how *algorithmic changes* have been made to *speed up code*, allowing for the training of larger neural networks.

|  |  |
| --- | --- |
| * ***Fast computation helps iterative training:*** *Fast computation* is also crucial because *training* a *neural network* is an *iterative process*. Typically, you come up with an idea for a neural network architecture, implement it in code, run an experiment to see how well it performs, then *adjust the network* based on the results. This cycle (**idea** -> **code** -> **experiment**) is repeated many times. |  |

* ***Faster computation increase your productivity:*** When ***training*** a neural network takes a ***long time***, the entire development process slows down. There's a significant difference in productivity when you can *test an idea* and see the results in *10 minutes* or a *day* versus waiting for a *month*.
* Faster feedback allows for more experimentation, helping to *discover effective architectures more quickly*. Fast computation has greatly improved the speed at which *neural network practitioners and researchers* can test and refine ideas, leading to faster innovation and better results.
* So this has been a major boost for the entire deep learning research community, enabling *rapid invention* of *new algorithms* and continuous progress, especially in the *area of computation*.

|  |
| --- |
| These are some of the key factors driving the rise of deep learning, and the good news is that these 3 forces (data, algorithm & computation) continue to improve it.   * ***More data:*** As society generates more digital data and * ***Computational power:*** advances in specialized hardware like GPUs and faster networking,   *More data* and *Computational power* makes our ability to train larger neural networks from a computational standpoint and it's expected to keep advancing.   * ***Advanced algorithms:*** The deep learning research community has been phenomenal at continuously innovating on the "algorithms front."   Because of this, there's reason to be optimistic that deep learning will keep improving for many years to come. |

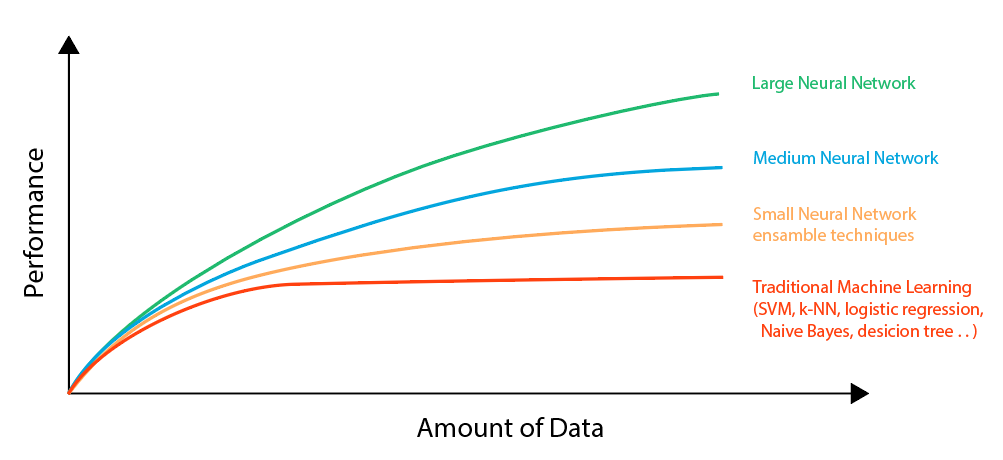
* **Why is Deep Learning taking off?**

Many of the ideas (the basic ideas) behind the **Deep Learning** have been around for several decades. For example: ***Several fundamental ideas*** behind deep learning have been around for decades. Some of these include:

1. ***Neural Networks:*** The concept of ***Artificial Neural Networks***, which mimic the *structure* and *function* of the *human brain*, dates back to the 1940s and 1950s.
2. ***Backpropagation:*** The backpropagation algorithm, used for ***training*** neural networks by adjusting their weights based on error gradients, was proposed in the 1970s and 1980s.
3. ***Universal Function Approximation:*** The ***universal approximation theorem***, which states that a Feedforward Neural Network with a ***single hidden layer*** can *approximate any continuous function*, was established in the 1980s.
4. ***Convolutional Neural Networks (CNNs):*** CNNs, specialized neural networks designed for ***processing Grid-Like data*** such as images, were introduced in the late 1980s and early 1990s.
5. ***Recurrent Neural Networks (RNNs):*** RNNs, which have ***loops*** to allow ***information*** to ***persist over time***, were proposed in the 1980s for processing sequential data.

These ideas laid the foundation for modern deep learning techniques and have been refined and expanded upon over the years, leading to the development of advanced deep learning models and algorithms that we see today.

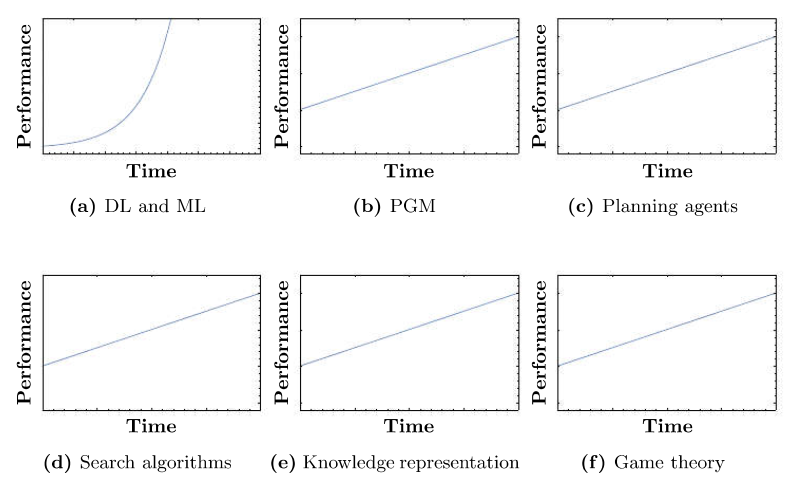
* **So why is deep learning suddenly taking off now?**
* Over the last couple of decades, the digitalization of society has led to the ***accumulation*** of ***vast amounts of data***. With most people spending a significant amount of time on *computers* and *smartphones*, nearly every activity generates data.
* What was once represented on paper is now stored digitally. For instance, X-rays are now digital images rather than physical films, and ***online*** ***purchases*** are recorded as ***digital transactions***. Just a decade ago, ordering or buying something was documented on paper. As a result, we generate a considerable volume of digital data on a daily basis.



* So, the amount of data has nearly exploded in the last 20 years.
* The performance of **traditional Machine learning algorithms** such as: ***SVM, k-NN, logistic regression, decision trees*** is a plateau even if we feed them more data.
* But if we use ***Neural Networks***, the ***performance increases*** with more data. Nevertheless, we observed that as we began adding data to ***Neural Networks***, their *performance improved significantly* compared to *Traditional Machine Learning Algorithms*.
* However the performance can't go further than a typical limit called ***Base Error Rate*** (discussed later).

The term **"Base Error Rate"** typically refers to the minimum achievable error rate or performance level for a given task or dataset. In the context of neural networks, the **Base Error Rate** represents the **lowest level of error** that can be attained, often due to inherent noise or irreducible error in the data.

* *One thing to point out is that:* we didn't hit the limit of scale yet, i.e. amount of data is useful for the problem according to the size of the Neural Network. In other words, the neural network is still able to effectively ***utilize additional data*** to improve its ***performance*** without encountering limitations in ***scalability***.
* Increased training capabilities with GPU: GPU computing (using CUDA) is a large part that enables us to transition from ***small neural*** ***networks*** to ***large neural networks***. The power of computers increases rapidly day by day, and you don’t need to spend a lot of money to buy *expensive GPUs or machines*. One can use online cloud computing systems like **Google Colab** and others, allowing you to rent GPUs from other companies. Therefore, almost everyone has access to the computational power needed to train neural networks.
* With advanced GPU and more data, training larger neural networks is easier than ever. Larger networks consistently outperform other machine learning algorithms.



Phases of the Course

Phases: We break down our course notes into 5 phases. The ***number of chapters*** may increase or decrease according to the phases.

1. Phase A: Neural Networks and Deep Learning

* What is a neuron?
* How to build layers using those neurons?
* How can we stack those layers on top of each other to build a Neural Network (small or deep)?

1. Phase B: Improving Deep Neural Networks

* *Hyperparameter Tuning*, *Regularization*, and *Optimization*
* It's not enough to just build and deploy NNs; we have to use methods to tune the NNs in order to improve their performance.

1. Phase C: **Structuring** Machine Learning Projects

* Learn about industrial applications
* How the industry works in AI?
* Understand how to strategize your project
* Understand how an AI developer team works.
* If you have an algorithm, you have to identify why does that algorithm works/ doesn't work,
* If it doesn't work, you have to identify why is doesn't wok
* What are the parts you should improve inside the algorithm?

In following two phases we'll focus on ***two fields*** that are defined by the ***two types of algorithms***

1. Phase D: Convolutional Neural Networks

* For image processing or videos

1. Phase E: Sequence Models

* Including RNNs, for Natural Language Processing and speech recognition

[**Specific Notation**: for example ***C2M3*** stands for ***3rd module of Course 2***,

i.e. **3rd section of Phase 2**, we can rewrite as **P2S3** or we can specify chapter wise.]

|  |  |
| --- | --- |
| **1 week of the course :**   * **1 module** = 10 online videos (1h 30min) + solving a quiz (20min) + completing programming assignments (1-3h) * **1 week** = 2 modules + 1 class lecture on advanced topics (1h 20min) + 1 TA lecture (1h) + 15min project mentorship * 10 weeks = **full course** | Grades: The weights determine where you should focus more:  Grade = 0.02**A** + 0.08**Q** + 0.25**Pa** + 0.25**M** + 0.40**Pr**   * **A** = Attendance (2%) * **Q** = Quizzes (8%) * **Pa** = Programming assignments (25%) * **M** = Midterm (25%) * **Pr** = Final project (40%)   Active Piazza participation = 1% bonus (Stanford) |

Programming Assignments

|  |  |
| --- | --- |
| * Project 1: There is a project of ***converting sign languages*** in vector form. * We're going to learn logistic regression, CNN in order to solve these projects. |  |
| * Project 2: Smile/Happiness Detector:   Imagine there is a house, and *we let in* only the people who are happy.   * We'll build a network that runs on a camera (in front of the house), which will determine whether to let people in or not. |  |
| * Project 3: Object Detection   This project is about object detection. We will use deep learning to *detect* *objects* in *real time*.   * We'll use a *deep-learning architecture* called YOLO-V2, an object detection algorithm that runs in real time and can detect 9,000 objects. * Check the paper and video: * Joseph Redmon, Ali Farhad: YOLO9000; Better, Faster, Stronger, 2016 * Another fun video generated with Y0L0v2 by J Redmon: <https://youtu.be/VOC3huqHrss> |  |

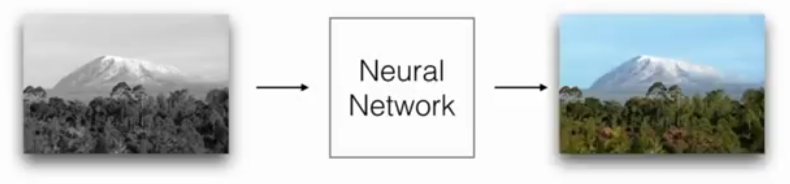
* Other Projects: You should build all of them. The more advanced projects are:
* ***Optimal Goalkeeper Shot Prediction:*** You want to decide where *you should shoot the ball in order to make it reach one of your teammates*. You have to find the exact line on the field where the goalkeeper should try to shoot the ball.

|  |  |
| --- | --- |
| * ***Face Recognition System:*** We are going to do face verification, for example: Is this person the right person? Or, who is this person? It's a little bit complex, but we'll try to solve it. * ***Music Generation:*** We are going to generate jazz music (in Phase 5: Sequence Model). * ***Generate Text:*** Given a huge corpus written by Shakespeare (poems), train an algorithm that can generate poems as if written by Shakespeare. You can give it a first sentence, and then the algorithm continues. * ***Emojifier:*** Write a sentence, and the algorithm will suggest an emoji to follow that sentence. * ***Trigger Word Detection:*** This involves detecting a single word. To detect the voice, we should use trigger words. We'll build the algorithm for the trigger word "activate." |  |

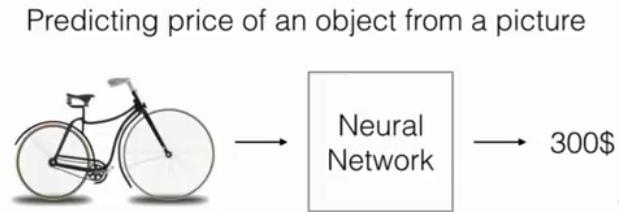
|  |  |
| --- | --- |
| * ***Car Detection:*** We'll apply the CNN (Phase 4) to build an autonomous driving application that will detect cars, stop signs, lights, pedestrians, and all other objects related to road features. You will generate some pictures like this ones, where a camera is placed in front of a car using the application Drive.ai. * ***Machine Translation:*** We're not going to translate one language to another. We'll try to build an algorithm that will convert human-readable dates to machine-readable dates. It will take all different dates in various formats and convert them into the correct format. |  |
| * ***Art Generation:*** We'll use a *neural style transfer algorithm* to ***generate art***. In general, we'll give a "content image" (e.g., Golden Gate Bridge) to our model and provide a "style image" (that was painted by someone, or an image from which we want to ***extract the styles***). Then we ***extract*** the ***style*** from the "style image" and apply those styles to the "content image". The algorithm is going to generate a new image by mixing the content of the first image and the styles from the second image.   ***Check out the paper:*** L. Gatys et al.: *Image Style Transfer Using Convolutional Neural Networks*. 2015. |  |

**More projects :**

* ***Coloring*** *black and white pictures with* ***deep learning*** *(using neural networks):* This project will help to colorize old *black and white movies* into color movies.



* ***Predicting*** *the* ***price*** *of an object from a* ***picture****:* For example, we provide our neural network with an *image of a bike*, and it predicts the *price* of that bike. We also try to determine which feature of the bike is related to its price. We need to find out if it's the steering wheel, the wheels, or the body of the bike that *contributes* to the high price.



* There are many other projects as well (these are very big, 10-week long projects):
* Predicting atom energy based on atomic structure
* Visual question answering
* Cancer/Parkinson/Alzheimer detection
* Brain tumor detection: We have brain tumor segmentation. Segmentation is a problem where an image is classified by every pixel, and the model will tell you which pixels correspond to the tumor.
* Activity recognition in video
* Music genre classification / Music compression
* Accent transfer in speech
* Generating images based on a given legend
* Detecting earthquake precursor signals with sequence models

In the next Chapter, we'll learn in detail: Basics of deep learning and neural networks