Neural Networks – Part I

PHYS 250 (Autumn 2024) – Lecture 15

David Miller

Department of Physics and the Enrico Fermi Institute
University of Chicago

November 27, 2024

Outline

Machine learning generally

Machine learning is an area of study based broadly on developments in computer science over the past 50 years that aims to use **algorithmic approaches to discovering, identifying, and analyzing, patterns of interest in datasets**.

- Key phrase here is **algorithmic**
- We want **models** that tell us something about a dataset

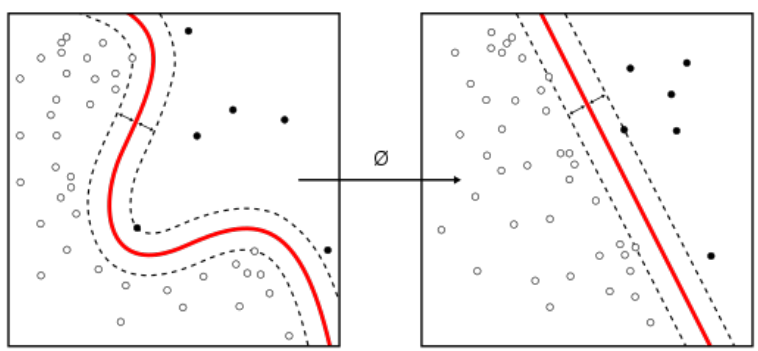
Sounds like something we'd be interested in!

$\rightarrow$ Definitely, but much broader concept here than just what we've been discussing so far in the course.

Machine learning applications

What are we using these algorithms to *do*?

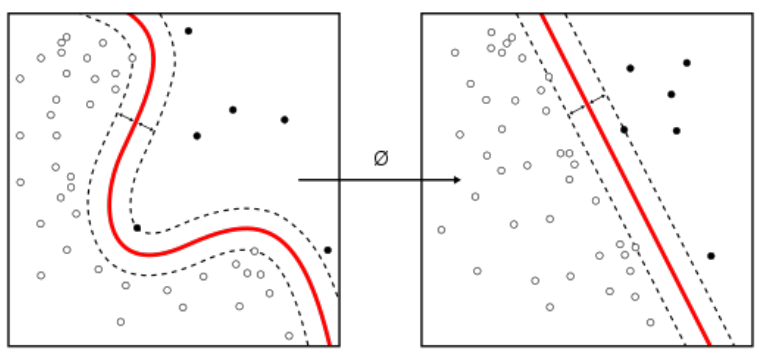
- **Supervised learning: machine is presented examples of multiple classes and learns to differentiate** (*discover known patterns on unknown data*)
  - **Classification:** identify categories of variables; e.g. signal vs. background
  - **Regression:** estimate relationships among variables; e.g. calibration
  - **Generation:** generate a set of variables from a category (e.g. image)
- **Unsupervised learning: machine is presented data and asked to provide multiple classes** (*discover unknown patterns on known data*)
  - **Clustering:** grouping set of objects in a dataset; e.g. jets of hadrons
  - **Anomaly detection:** identifying rare events or observations; e.g. errors



Machine learning applications

What are we using these algorithms to *do*?

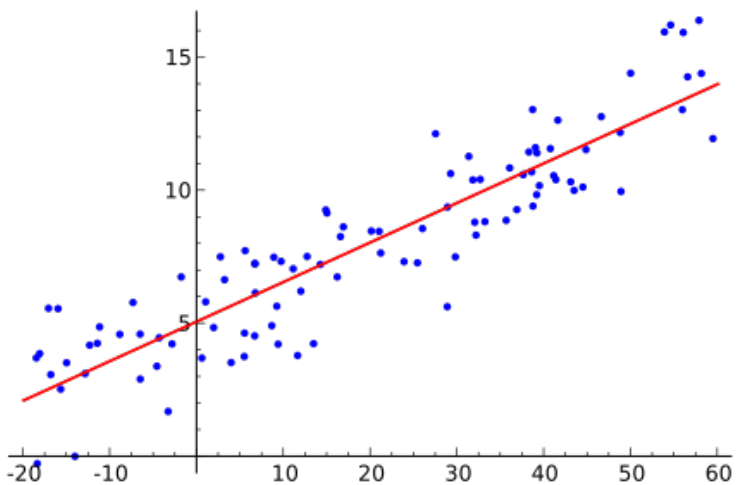
- **Supervised learning: machine is presented examples of multiple classes and learns to differentiate** (*discover known patterns on unknown data*)
  - **Classification:** identify categories of variables; e.g. signal vs. background
  - **Regression:** estimate relationships among variables; e.g. calibration
  - **Generation:** generate a set of variables from a category (e.g. image)
- **Unsupervised learning: machine is presented data and asked to provide multiple classes** (*discover unknown patterns on known data*)
  - **Clustering:** grouping set of objects in a dataset; e.g. jets of hadrons
  - **Anomaly detection:** identifying rare events or observations; e.g. errors



Machine learning applications

What are we using these algorithms to *do*?

- **Supervised learning: machine is presented examples of multiple classes and learns to differentiate** (*discover known patterns on unknown data*)
  - **Classification:** identify categories of variables; e.g. signal vs. background
  - **Regression:** estimate relationships among variables; e.g. calibration
  - **Generation:** generate a set of variables from a category (e.g. image)
- **Unsupervised learning: machine is presented data and asked to provide multiple classes** (*discover unknown patterns on known data*)
  - **Clustering:** grouping set of objects in a dataset; e.g. jets of hadrons
  - **Anomaly detection:** identifying rare events or observations; e.g. errors



Machine learning applications

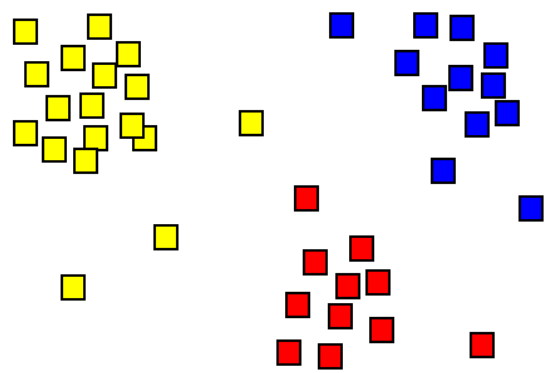
What are we using these algorithms to *do*?

- **Supervised learning:** **machine is presented examples of multiple classes and learns to differentiate** (*discover known patterns on unknown data*)
  - **Classification:** identify categories of variables; e.g. signal vs. background
  - **Regression:** estimate relationships among variables; e.g. calibration
  - **Generation:** generate a set of variables from a category (e.g. image)
- **Unsupervised learning:** **machine is presented data and asked to provide multiple classes** (*discover unknown patterns on known data*)
  - **Clustering:** grouping set of objects in a dataset; e.g. jets of hadrons
  - **Anomaly detection:** identifying rare events or observations; e.g. errors

Machine learning applications

What are we using these algorithms to *do*?

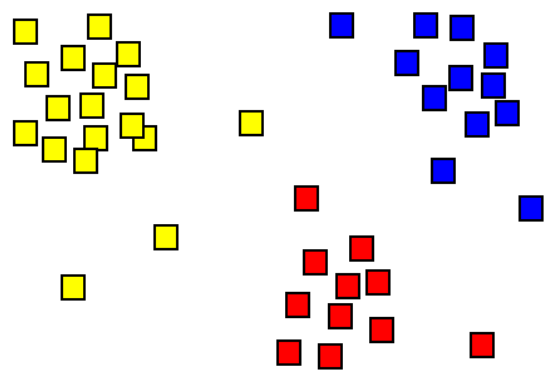
- **Supervised learning: machine is presented examples of multiple classes and learns to differentiate** (*discover known patterns on unknown data*)
  - **Classification:** identify categories of variables; e.g. signal vs. background
  - **Regression:** estimate relationships among variables; e.g. calibration
  - **Generation:** generate a set of variables from a category (e.g. image)
- **Unsupervised learning: machine is presented data and asked to provide multiple classes** (*discover unknown patterns on known data*)
  - **Clustering:** grouping set of objects in a dataset; e.g. jets of hadrons
  - **Anomaly detection:** identifying rare events or observations; e.g. errors



Machine learning applications

What are we using these algorithms to *do*?

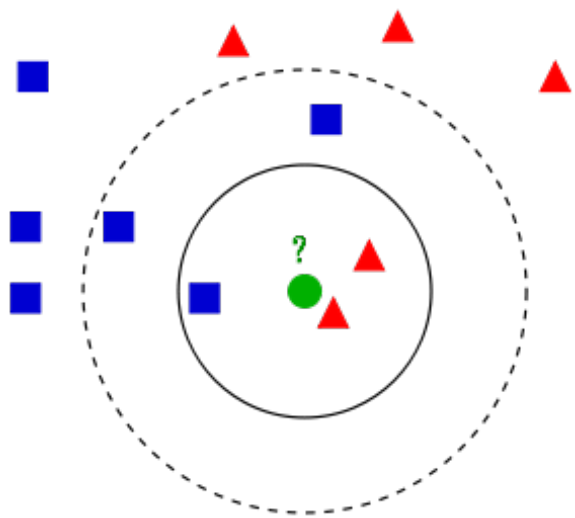
- **Supervised learning: machine is presented examples of multiple classes and learns to differentiate** (*discover known patterns on unknown data*)
  - **Classification:** identify categories of variables; e.g. signal vs. background
  - **Regression:** estimate relationships among variables; e.g. calibration
  - **Generation:** generate a set of variables from a category (e.g. image)
- **Unsupervised learning: machine is presented data and asked to provide multiple classes** (*discover unknown patterns on known data*)
  - **Clustering:** grouping set of objects in a dataset; e.g. jets of hadrons
  - **Anomaly detection:** identifying rare events or observations; e.g. errors



Machine learning applications

What are we using these algorithms to *do*?

- **Supervised learning: machine is presented examples of multiple classes and learns to differentiate** (*discover known patterns on unknown data*)
  - **Classification:** identify categories of variables; e.g. signal vs. background
  - **Regression:** estimate relationships among variables; e.g. calibration
  - **Generation:** generate a set of variables from a category (e.g. image)
- **Unsupervised learning: machine is presented data and asked to provide multiple classes** (*discover unknown patterns on known data*)
  - **Clustering:** grouping set of objects in a dataset; e.g. jets of hadrons
  - **Anomaly detection:** identifying rare events or observations; e.g. errors



Where do neural networks fit in?

There are many **tools** that we can use to **learn** how to perform these tasks.

Neural networks

Neural networks provide a framework for non-linear fitting and parameter estimation with limited input data but for cases in which the input data are understood and the categories (for example) are known.

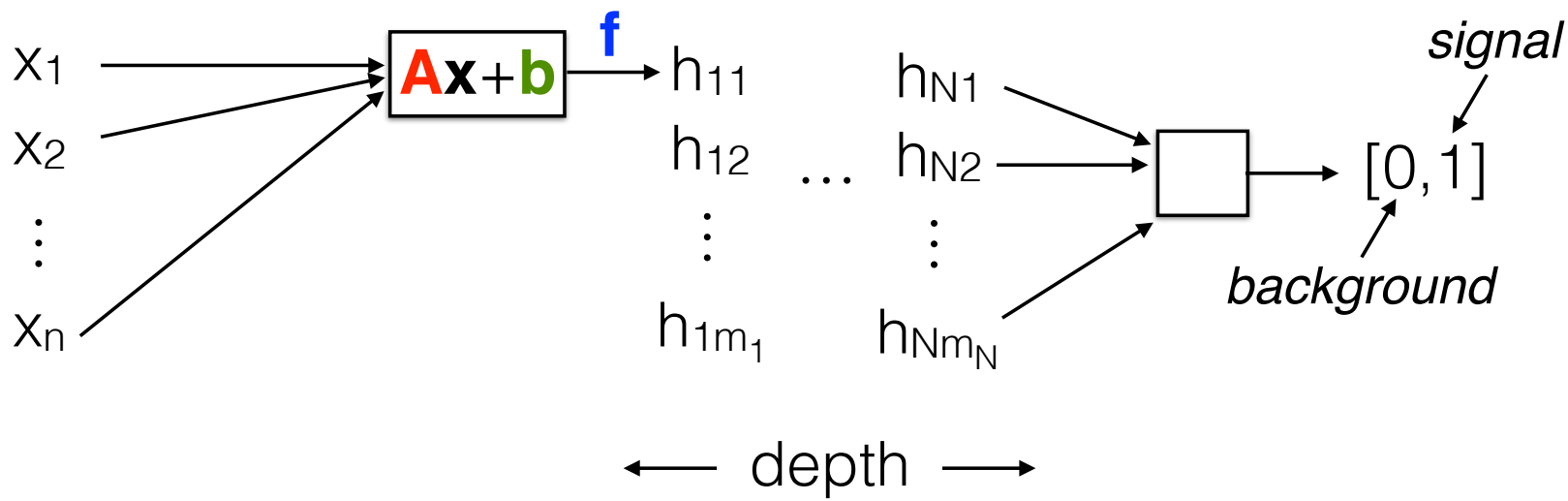
Neural networks are proven to be "universal approximators"

See Kurt Hornik, "Approximation Capabilities of Multilayer Feedforward Networks", Neural Networks, 4, 251-257 (1991).

One straightforward way to think of this is as a composition of functions $f(\mathbb{A}\mathbf{x} + \mathbf{b})$ for inputs $\mathbf{x}$ (features) matrix $\mathbb{A}$ (weights), bias $\mathbf{b}$, non-linearity f. Given some data, the algorithm should tell us which of a pre-defined set of categories the data fall into, even if it's a complicated non-linear function that describes the boundary between these categories.

Motivations for neural networks

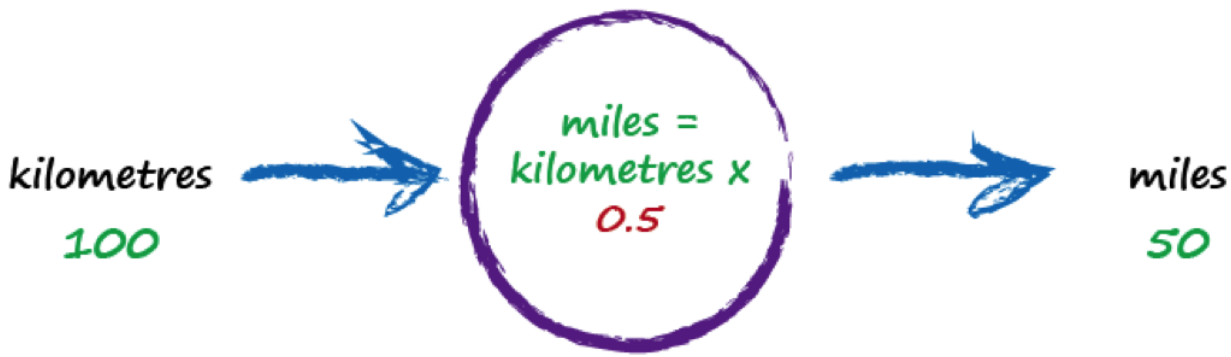
If we can use a neural network to approximate any (incl. non-linear) function, then this is strong motivation for using it to learn complicated functions that describe classes of objects, potentially in high-dimensional spaces.



Very, very naive strawman example

Suppose the "function" we want to learn is to map kilometers$\rightarrow$ miles.

Given a **"training dataset"** of input and output pairs, we can test a simple function with a parameter the defines the conversion, and then **iteratively update** the parameters based on how far from the **"true"** value we end up.



Obviously, we want to extend this to non-linear cases.

Very, very naive strawman example

Suppose the "function" we want to learn is to map kilometers→ miles.

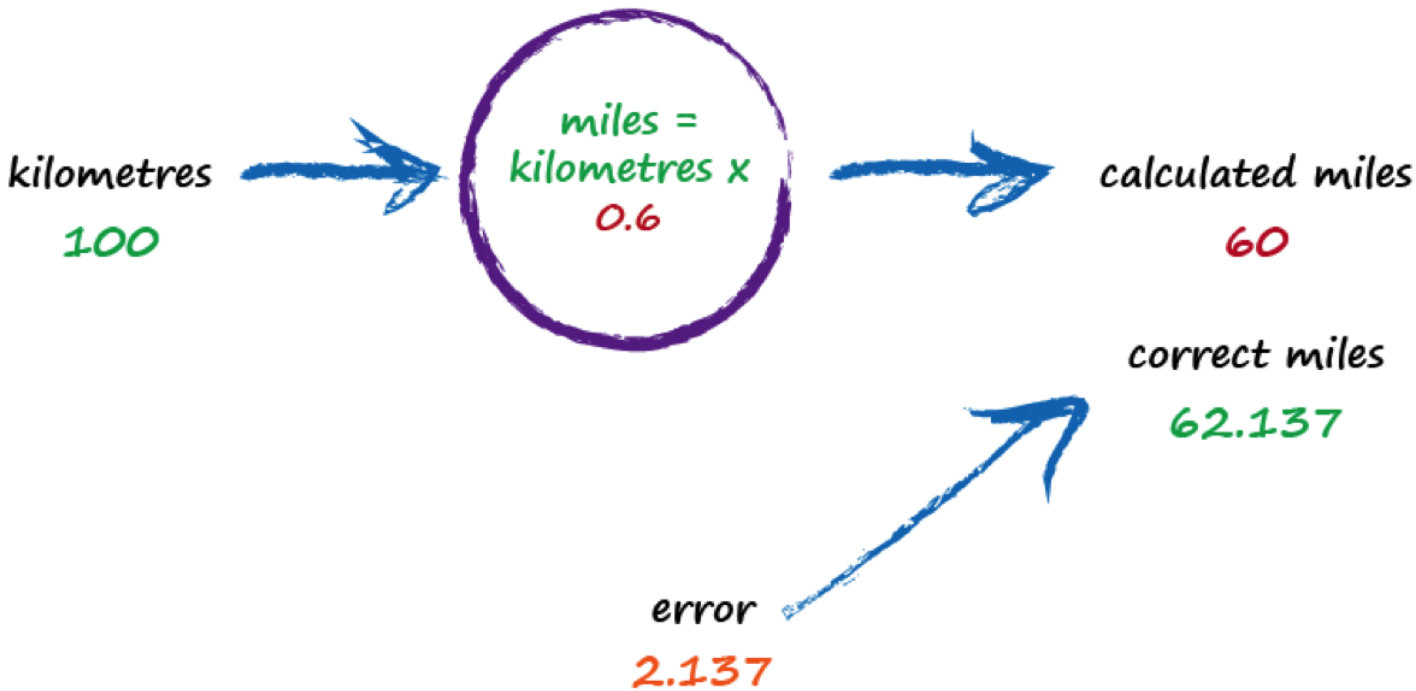
Given a **"training dataset"** of input and output pairs, we can test a simple function with a parameter the defines the conversion, and then **iteratively update** the parameters based on how far from the **"true"** value we end up.

Obviously, we want to extend this to non-linear cases.

Very, very naive strawman example

Suppose the "function" we want to learn is to map kilometers→ miles.

Given a **"training dataset"** of input and output pairs, we can test a simple function with a parameter the defines the conversion, and then **iteratively update** the parameters based on how far from the **"true"** value we end up.

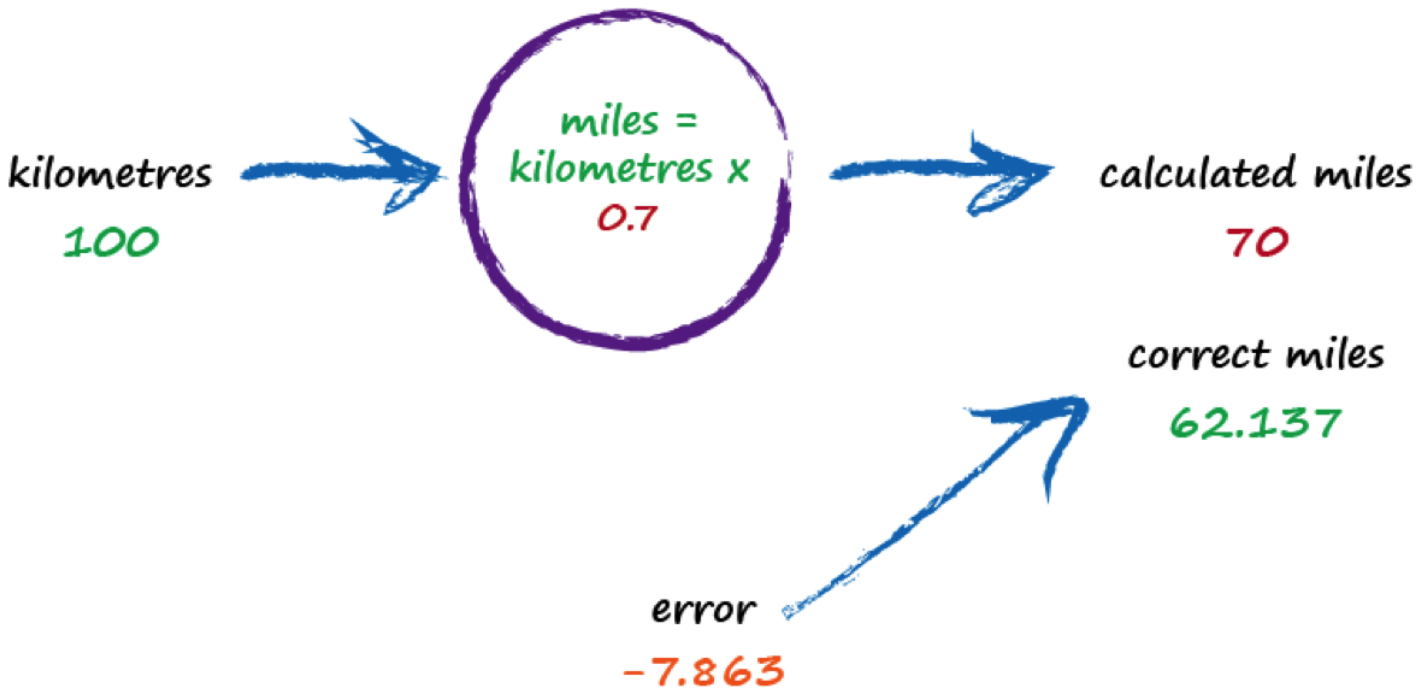


Obviously, we want to extend this to non-linear cases.

Very, very naive strawman example

Suppose the "function" we want to learn is to map kilometers→ miles.

Given a **"training dataset"** of input and output pairs, we can test a simple function with a parameter the defines the conversion, and then **iteratively update** the parameters based on how far from the **"true"** value we end up.

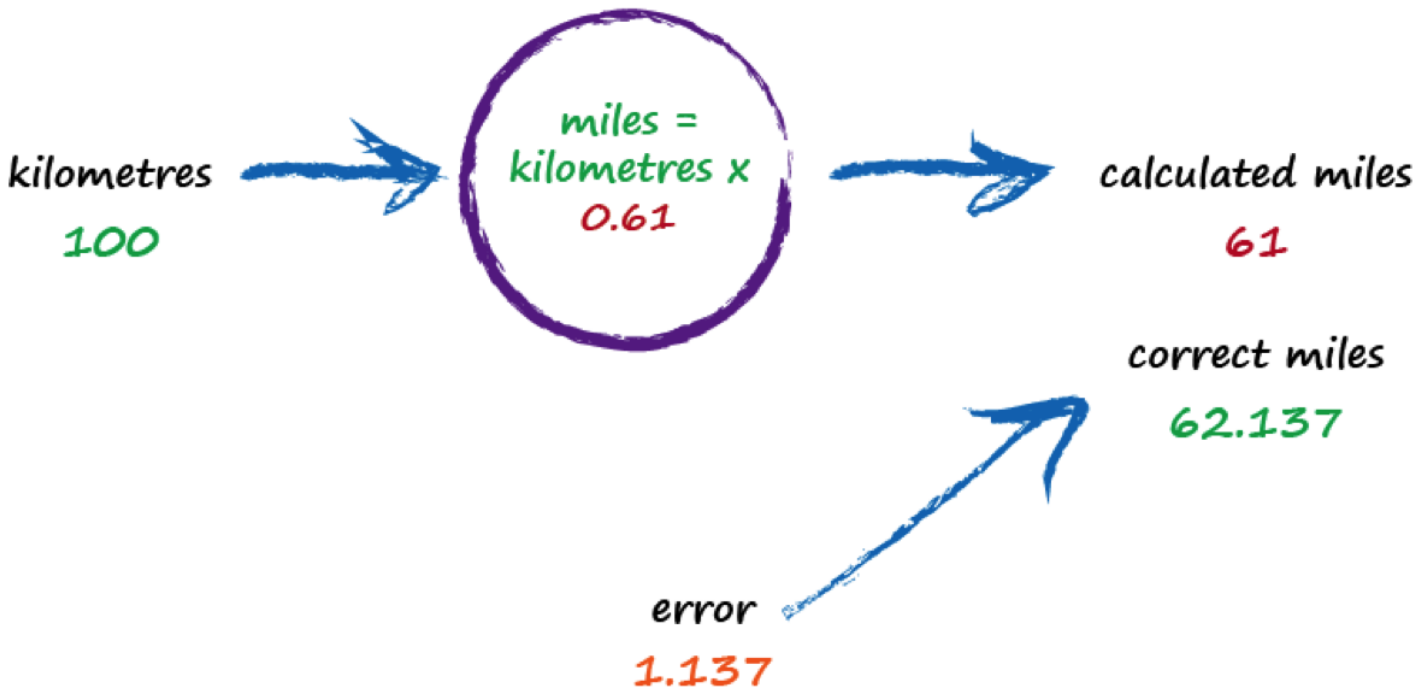


Obviously, we want to extend this to non-linear cases.

Very, very naive strawman example

Suppose the "function" we want to learn is to map kilometers→ miles.

Given a **"training dataset"** of input and output pairs, we can test a simple function with a parameter the defines the conversion, and then **iteratively update** the parameters based on how far from the **"true"** value we end up.

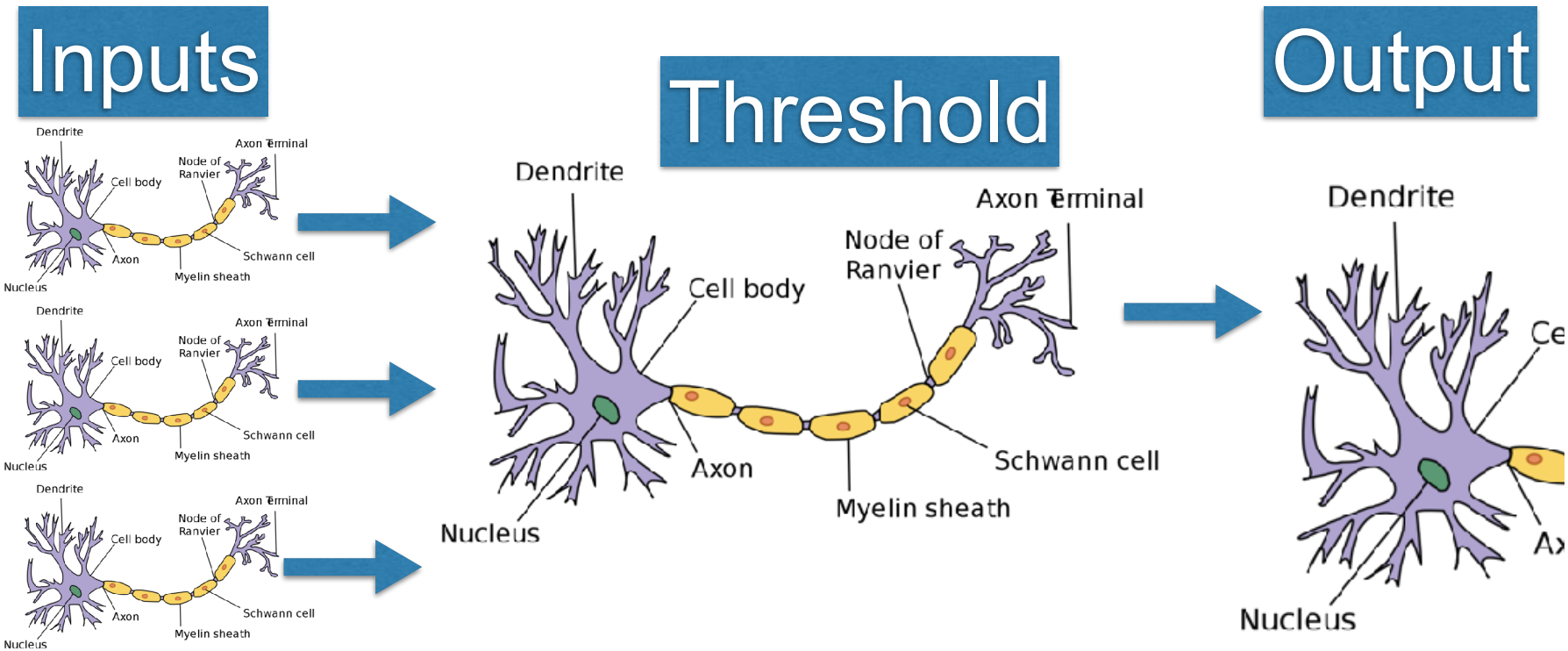


Obviously, we want to extend this to non-linear cases.

The biological model

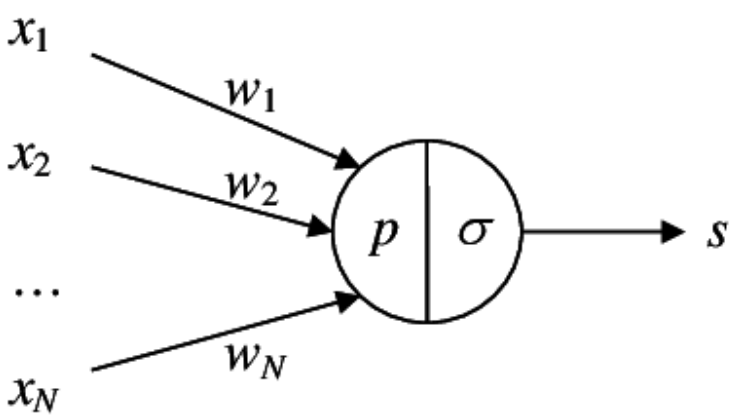
The way brains work is by establishing connections between various neurons.

Once connections are made, initial stimuli give similar (though not always identical) results



Artificial neuron model

McCulloch and Pitts (1943) model of a single neuron. Rosenblatt (1958) then developed the **perceptron** based on this, the oldest neural network still in use today; provided an algorithm to train the perceptron network, built it in electronics, and proved convergence in linearly separable case.



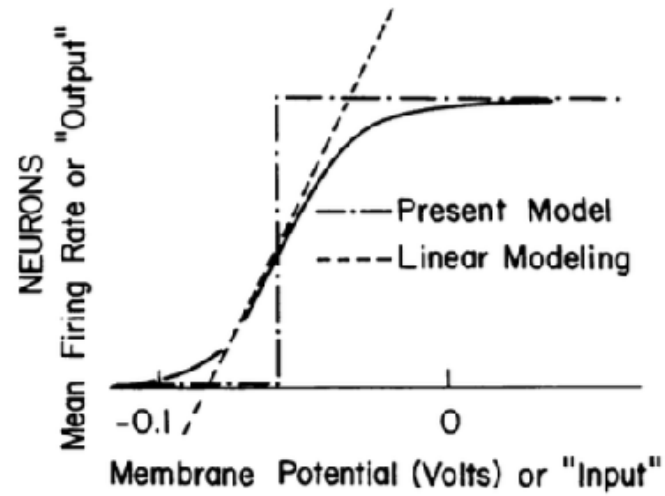
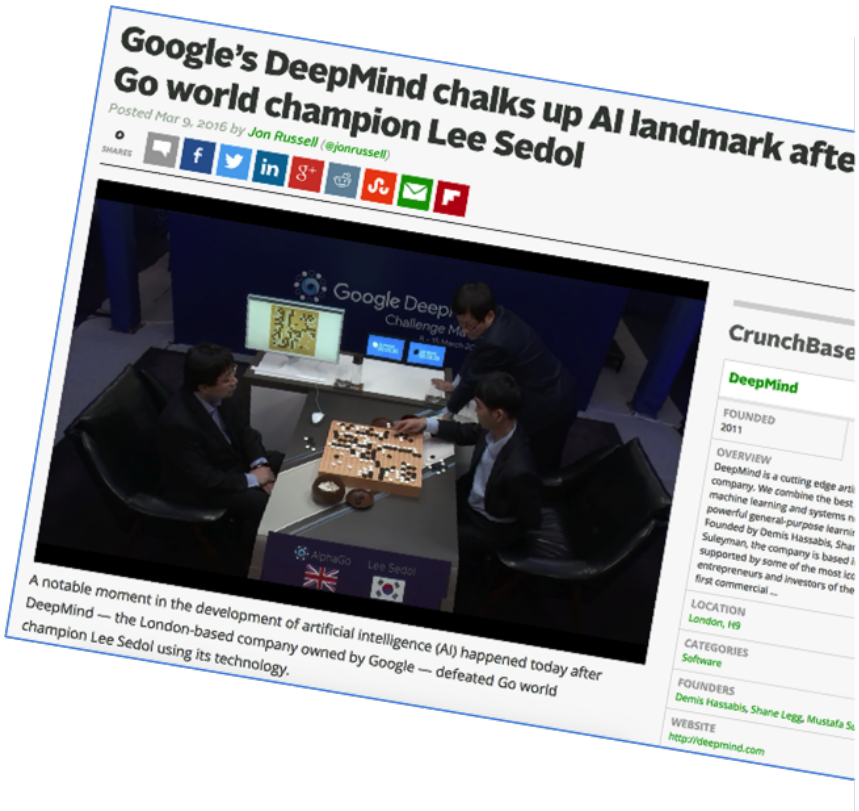


FIG. 1. Firing rate versus membrane voltage for a typical neuron (solid line), dropping to 0 for large negative potentials and saturating for positive potentials. The broken lines show approximations used in modeling.

Neural networks are, maybe literally, everywhere



Google's DeepMind chalks up AI landmark after Go world champion Lee Sedol

Posted Mar 9, 2016 by Jon Russell (@jonrussell)

A notable moment in the development of artificial intelligence (AI) happened today after DeepMind — the London-based company owned by Google — defeated Go world champion Lee Sedol using its technology.

CrunchBase

DeepMind

FOUNDED 2011

OVERVIEW DeepMind is a cutting edge artificial intelligence company. We combine the best techniques from machine learning and systems neuroscience to build powerful general-purpose learning algorithms. Founded by Demis Hassabis, Shane Legg and Mustafa Suleyman, the company is based in London and is supported by some of the most successful entrepreneurs and investors of the world. Its first commercial ...

LOCATION London, HQ

CATEGORIES Software

FOUNDERS Demis Hassabis, Shane Legg, Mustafa Su

WEBSITE http://deepmind.com

nature International weekly journal of science

Home | News & Comment | Research | Careers & Jobs | Current Issue | Archive | Audio & Video | For

Archive > Volume 529 > Issue 7587 > Articles > Article

ARTICLE PREVIEW

view full access options

NATURE | ARTICLE

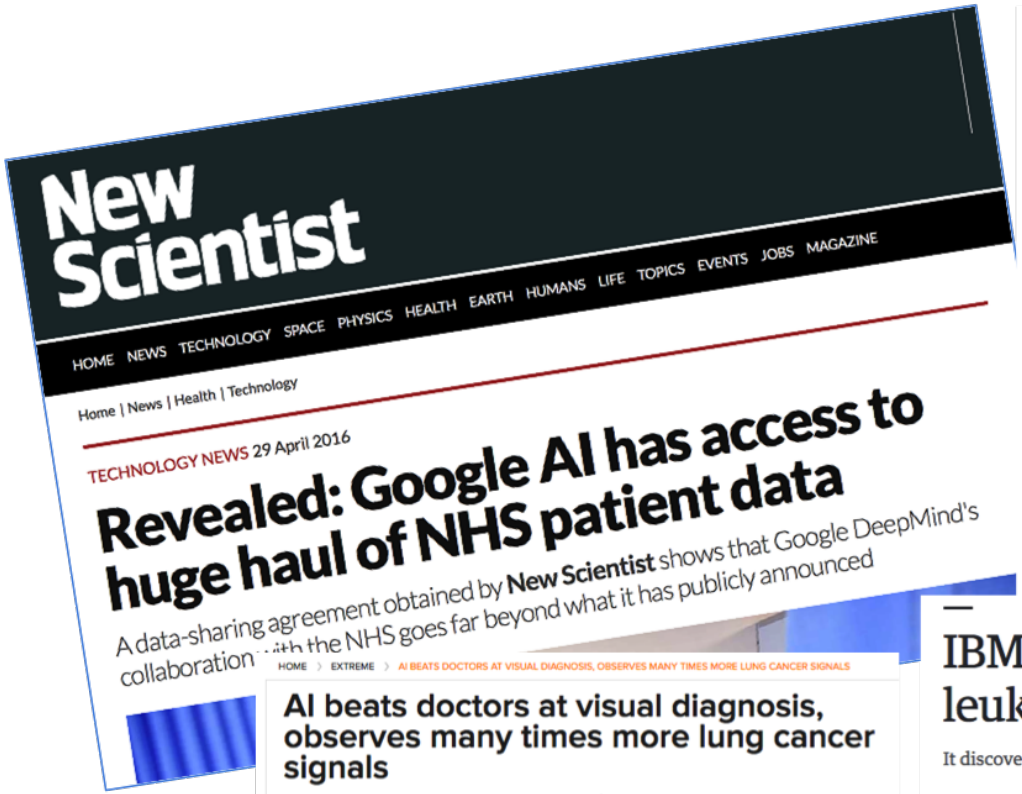
日本語要約

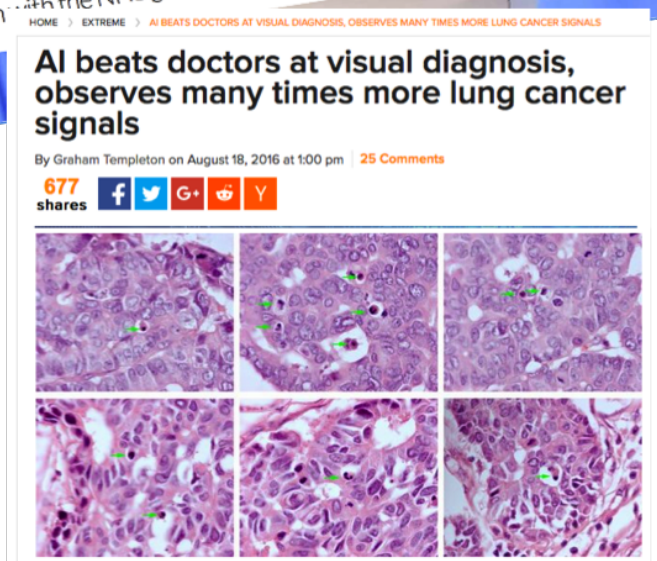
Mastering the game of Go with deep neural networks and tree search

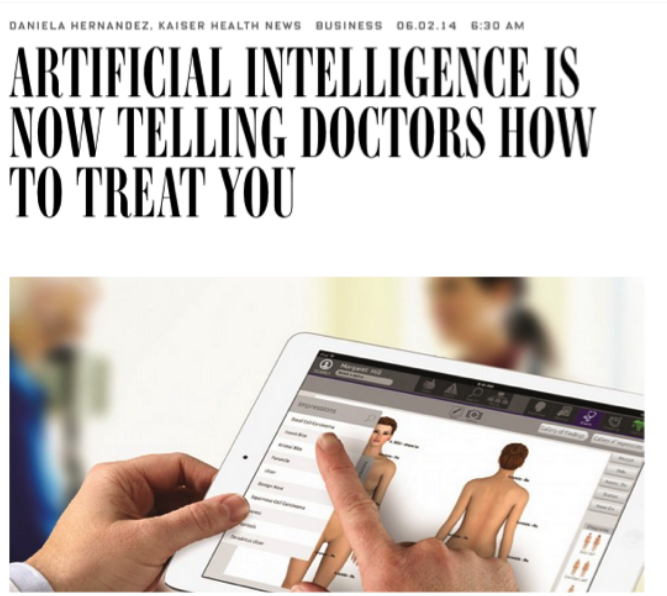
David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel & Demis Hassabis

Affiliations | Contributions | Corresponding authors

Neural networks are, maybe literally, everywhere



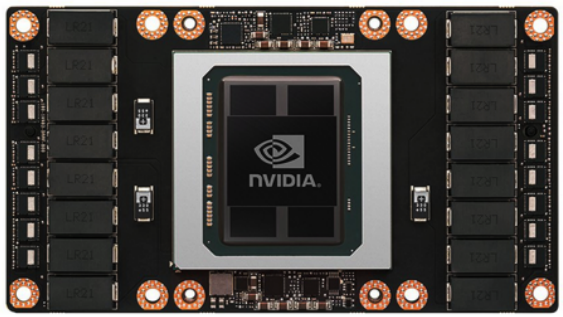




Neural networks are, maybe literally, everywhere

TECHNOLOGY

Nvidia Unveils Artificial-Intelligence Tech For Baidu Self-Driving Cars



Nvidia releases Pascal GPUs for neural networks

Nvidia has launched a pair of GPUs that the chip giant claims are four times faster than last year's offering.

CRUNCH NETWORK

The driverless truck is coming, and it's going to automate millions of jobs

Posted Apr 25, 2016 by Ryan Petersen (@typesfast)

Pittsburgh residents get ready for driverless Uber cars

Business

Self-driving cars set to disrupt UK's £14bn motor insurance industry

share

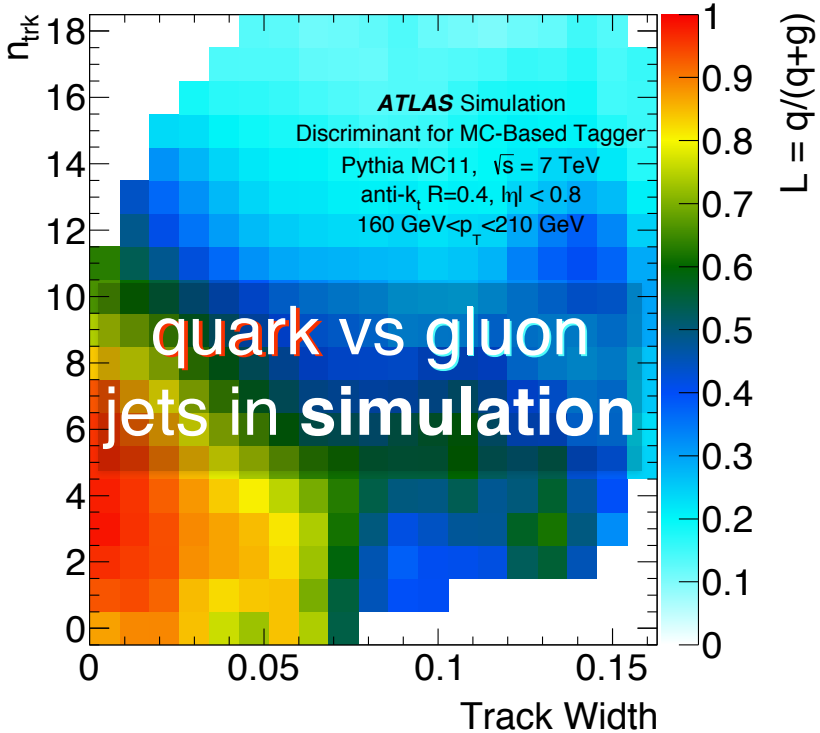
Self-driving cars will result in an 80pc drop in crashes by 2035 CREDIT: ALAMY

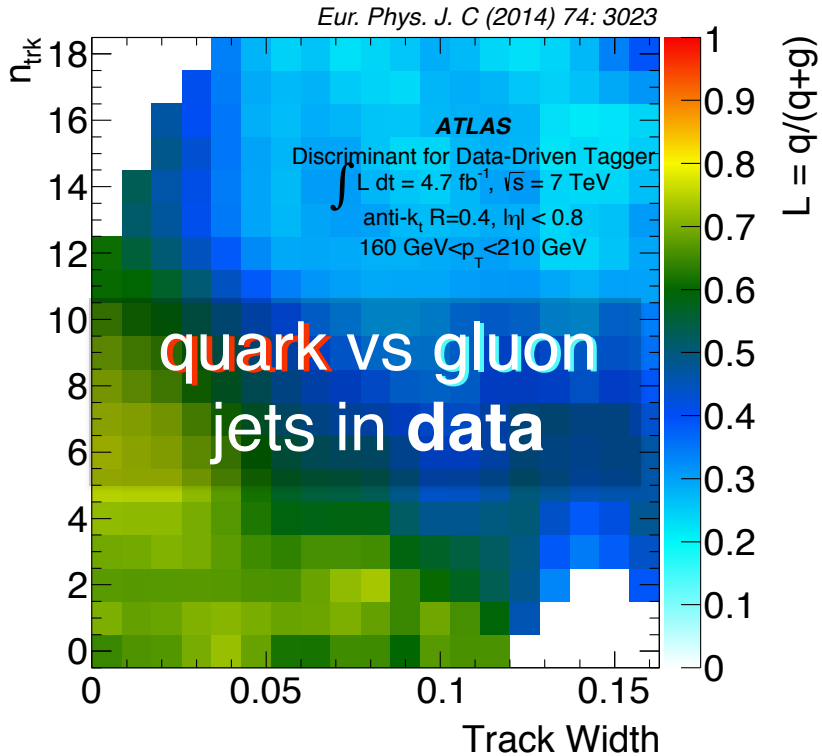
MOTHERBOARD Watch Sections

The Tradeoffs of Imbuing Self-Driving Cars With Human Morality

August 25, 2016 // 07:00 AM EST

Neural networks are, maybe literally, everywhere

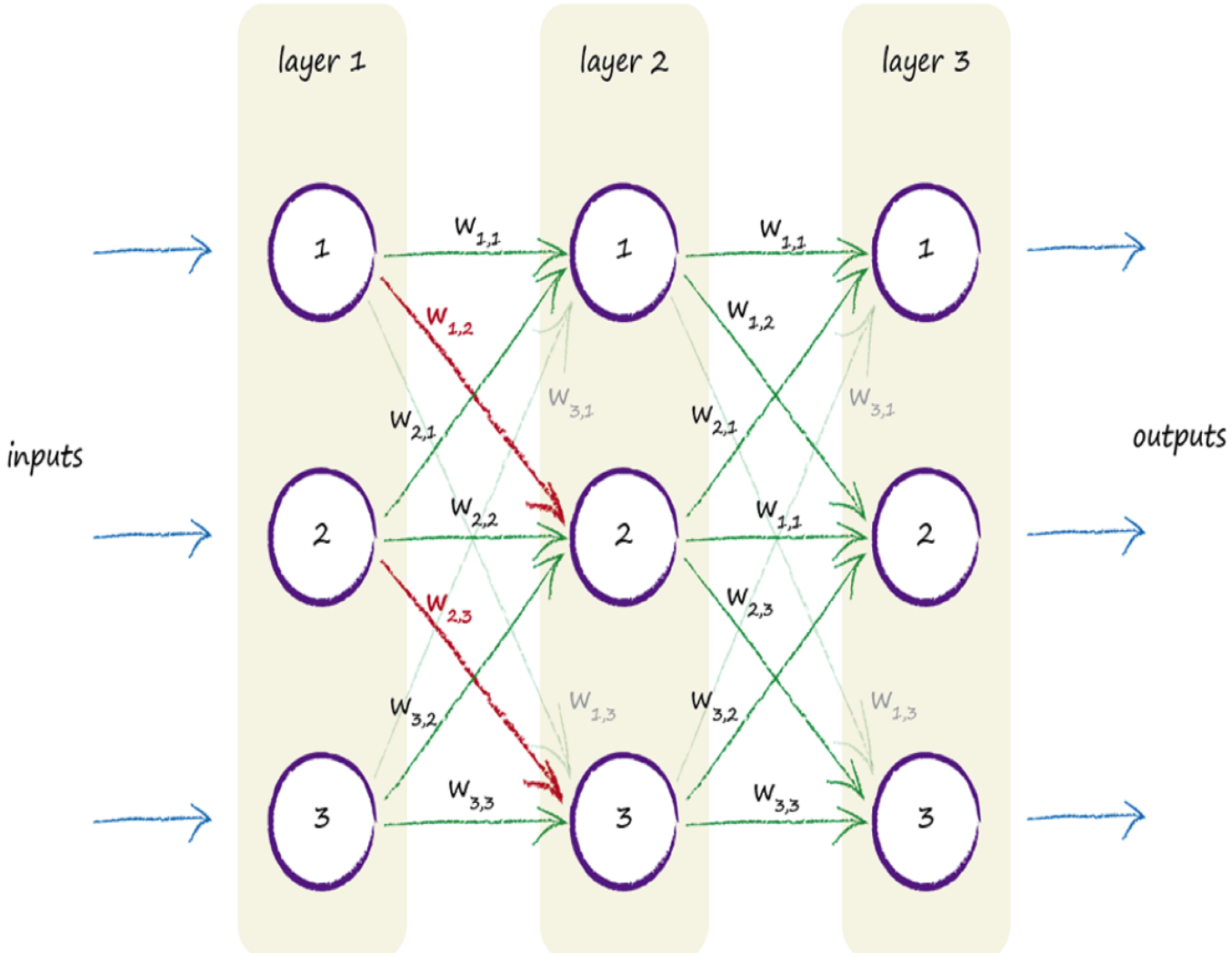




Outline

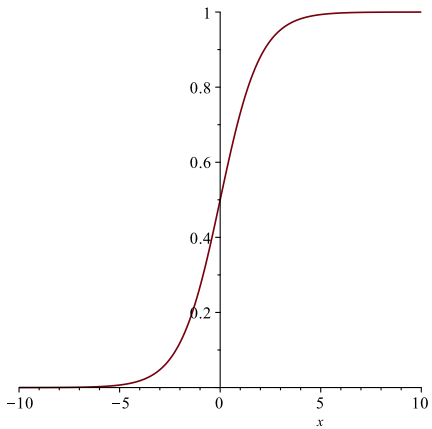
Activation functions

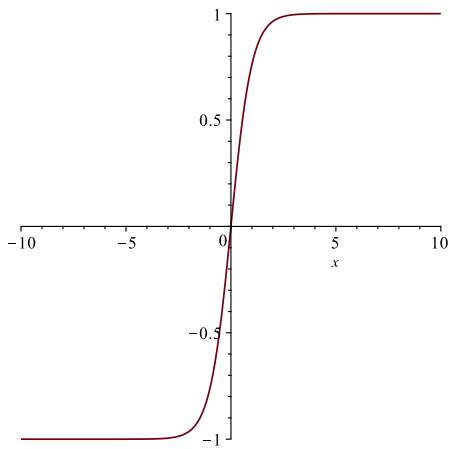
We need a network that can effectively learn a complex function by weighting the individual outputs of each node.



Non-linearity functions for learning

This is the function that defines the output of a node. Typically, we want to map the resulting values into a range between $[0, 1]$, or $[-1, 1]$





Logistic (aka Sigmoid): one of the most widely-used functions in the past, no basically only used for the last layer.

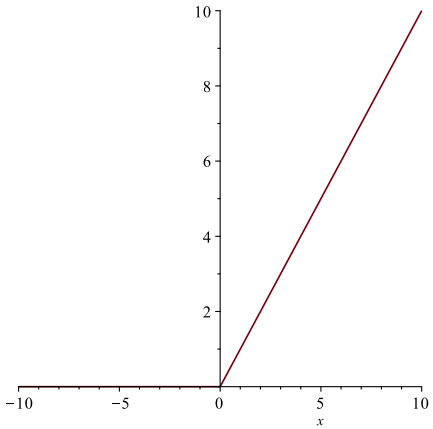
generalization to multi-dimensional input: softmax

$$\mathbf{f}(\vec{x}) = e^{x_i} / \sum_i e^{x_i}$$

tanh: similar story to sigmoid.

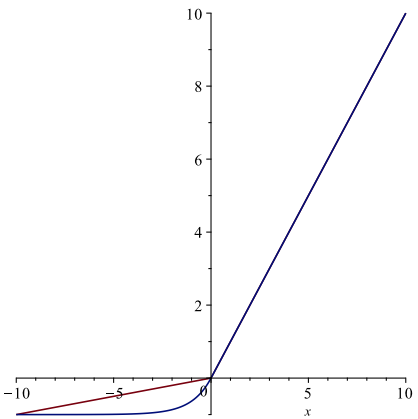
Non-linearity functions for learning

This is the function that defines the output of a node. Typically, we want to map the resulting values into a range between $[0, 1]$, or $[-1, 1]$



Rectified Linear Unit **ReLU**: one of the most widely-used functions now.

do not suffer from the vanishing gradient problem



Leaky ReLU / Exponential LU (ELU): variations on the ReLU that are popular.

Training a neural network

The **"objective function"** is minimized using **"stochastic gradient decent"**:

- iterative method for optimizing the "loss" or the error that we are trying to minimize

Training proceeds by minimizing a loss function.

Typical loss functions

Squared error: $(y_i - \hat{y}_i)^2$

Cross-entropy: $-y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i)$

True label (0 or 1)

NN output

Back-propagation

For the matrix that defines the **weights** we want to update those weights based on

$$A_{ij} \mapsto A_{ij} - \eta \nabla_{ij} \mathcal{L}$$

learning rate

back-propagation: weights updated backwards and gradients are recycled.

Here, **back-propagation** is shorthand for "the backward propagation of errors" since an error is computed at the output and distributed backwards throughout the network's layers.

Really, **back-propagation** is **just a special case of computational differentiation!**.

Back-propagation

Phase 1: propagation

- Propagation forward through the network to generate the output value(s)
- Calculation of the cost (error term)
- Propagation of the output activations back through the network using the training pattern target to generate the deltas (the difference between the targeted and actual output values) of all output and hidden neurons.

Phase 2: weight update

- The weight's output delta and input activation are multiplied to find the gradient of the weight.
- A ratio (percentage) of the weight's gradient is subtracted from the weight.

This ratio (percentage) influences the speed and quality of learning; it is called the learning rate (η on the previous slide). The greater the ratio, the faster the neuron trains, but the lower the ratio, the more accurate the training is. The sign of the gradient of a weight indicates whether the error varies directly with, or inversely to, the weight. Therefore, the weight must be updated in the opposite direction, "descending" the gradient.

```
initialize network weights (often small random values)
```

Training and validation

Training proceeds multiple times (epochs), reshuffling the data.

Early stopping: stop at the epoch where the validation error starts to increase