

Artificial intelligence and machine learning in physics, introduction

Morten Hjorth-Jensen¹

Department of Physics and Center for Computing in Science Education,
University of Oslo, Norway¹

Geilo Winter school, March 10-20, 2025

What is this talk about?

The main emphasis is to give you a short and pedestrian introduction to the whys and hows we can use (with several examples) machine learning methods in physics. And why this could (or should) be of interest.

These slides and more at <http://mhjensenseminars.github.io/MachineLearningTalk/doc/pub/Geiloschool/>

Thanks to many

Jane Kim (MSU), Julie Butler (MSU), Patrick Cook (MSU), Danny Jammooa (MSU), Daniel Bazin (MSU), Dean Lee (MSU), Witek Nazarewicz (MSU), Michelle Kuchera (Davidson College), Even Nordhagen (UiO), Robert Solli (UiO, Expert Analytics), Bryce Fore (ANL), Alessandro Lovato (ANL), Stefano Gandolfi (LANL), Francesco Pederiva (UniTN), and Giuseppe Carleo (EPFL). Niyaz Beysengulov and Johannes Pollanen (experiment, MSU); Zachary Stewart, Jared Weidman, and Angela Wilson (quantum chemistry, MSU) Jonas Flaten, Oskar, Leinonen, Øyvind Sigmundson Schøyen, Stian Dysthe Bilek, and Håkon Emil Kristiansen (UiO). Marianne Bathen and Lasse Vines (experiments (UiO). Excuses to those I have omitted.

And sponsors

1. National Science Foundation, US (various grants)
2. Department of Energy, US (various grants)
3. Research Council of Norway (various grants) and my employers
University of Oslo and Michigan State University

AI/ML and some statements you may have heard (and what do they mean?)

1. Fei-Fei Li on ImageNet: **map out the entire world of objects** ([The data that transformed AI research](#))
2. Russell and Norvig in their popular textbook: **relevant to any intellectual task; it is truly a universal field** ([Artificial Intelligence, A modern approach](#))
3. Woody Bledsoe puts it more bluntly: **in the long run, AI is the only science** (quoted in Pamilla McCorduck, [Machines who think](#))

If you wish to have a critical read on AI/ML from a societal point of view, see [Kate Crawford's recent text Atlas of AI](#).

Here: with AI/ML we intend a collection of machine learning methods with an emphasis on statistical learning and data analysis

Types of machine learning

The approaches to machine learning are many, but are often split into two main categories. In *supervised learning* we know the answer to a problem, and let the computer deduce the logic behind it. On the other hand, *unsupervised learning* is a method for finding patterns and relationship in data sets without any prior knowledge of the system.

An important third category is *reinforcement learning*. This is a paradigm of learning inspired by behavioural psychology, where learning is achieved by trial-and-error, solely from rewards and punishment.

Main categories

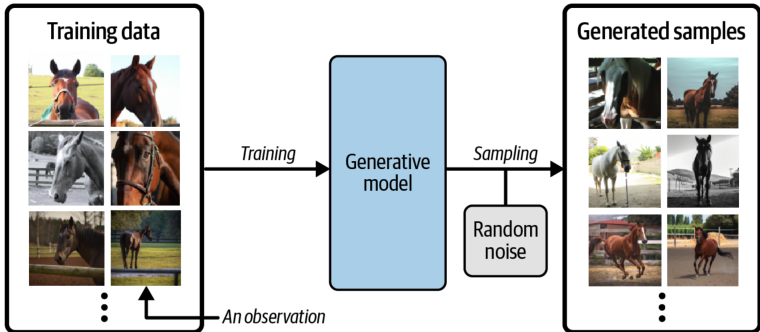
Another way to categorize machine learning tasks is to consider the desired output of a system. Some of the most common tasks are:

- ▶ **Classification:** Outputs are divided into two or more classes. The goal is to produce a model that assigns inputs into one of these classes. An example is to identify digits based on pictures of hand-written ones. Classification is typically supervised learning.
- ▶ **Regression:** Finding a functional relationship between an input data set and a reference data set. The goal is to construct a function that maps input data to continuous output values.
- ▶ **Clustering:** Data are divided into groups with certain common traits, without knowing the different groups beforehand. It is thus a form of unsupervised learning.

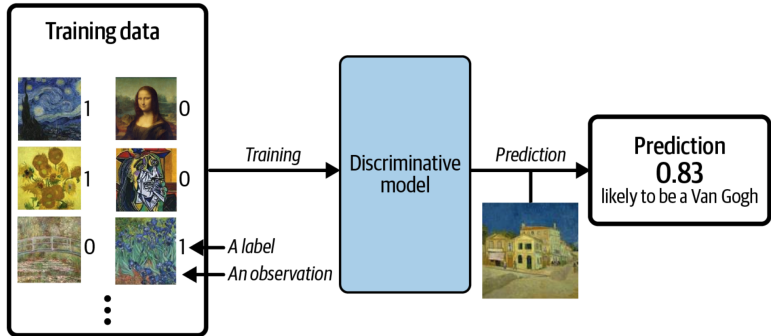
The plethora of machine learning algorithms/methods

1. Deep learning: Neural Networks (NN), Convolutional NN, Recurrent NN, Boltzmann machines, autoencoders and variational autoencoders and generative adversarial networks, stable diffusion and many more generative models
2. Bayesian statistics and Bayesian Machine Learning, Bayesian experimental design, Bayesian Regression models, Bayesian neural networks, Gaussian processes and much more
3. Dimensionality reduction (Principal component analysis), Clustering Methods and more
4. Ensemble Methods, Random forests, bagging and voting methods, gradient boosting approaches
5. Linear and logistic regression, Kernel methods, support vector machines and more
6. Reinforcement Learning; Transfer Learning and more

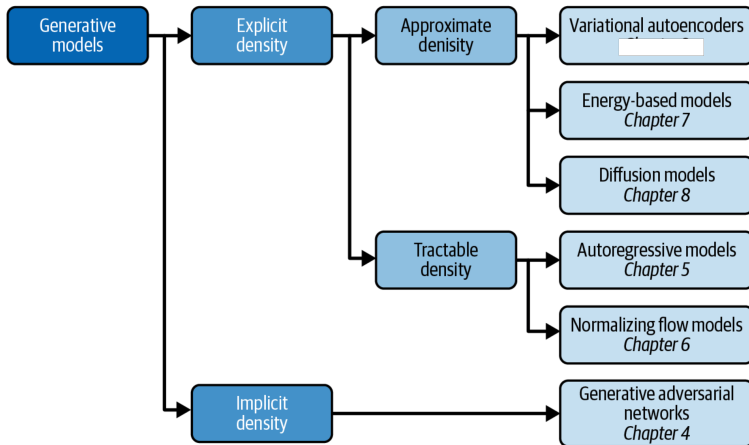
Example of generative modeling, taken from Generative Deep Learning by David Foster



Example of discriminative modeling, taken from Generative Deep Learning by David Foster



Taxonomy of generative deep learning, taken from Generative Deep Learning by David Foster



Good books with hands-on material and codes

- ▶ Sebastian Rashcka et al, Machine learning with Sickit-Learn and PyTorch
- ▶ David Foster, Generative Deep Learning with TensorFlow
- ▶ Bali and Gavras, Generative AI with Python and TensorFlow 2

All three books have GitHub addresses from where one can download all codes. We will borrow most of the material from these three texts as well as from Goodfellow, Bengio and Courville's text [Deep Learning](#)

What are the basic Machine Learning ingredients?

Almost every problem in ML and data science starts with the same ingredients:

- ▶ The dataset \mathbf{x} (could be some observable quantity of the system we are studying)
- ▶ A model which is a function of a set of parameters α that relates to the dataset, say a likelihood function $p(\mathbf{x}|\alpha)$ or just a simple model $f(\alpha)$
- ▶ A so-called **loss/cost/risk** function $\mathcal{C}(\mathbf{x}, f(\alpha))$ which allows us to decide how well our model represents the dataset.

We seek to minimize the function $\mathcal{C}(\mathbf{x}, f(\alpha))$ by finding the parameter values which minimize \mathcal{C} . This leads to various minimization algorithms. It may surprise many, but at the heart of all machine learning algorithms there is an optimization problem.

Low-level machine learning, the family of ordinary least squares methods

Our data which we want to apply a machine learning method on, consist of a set of inputs $\mathbf{x}^T = [x_0, x_1, x_2, \dots, x_{n-1}]$ and the outputs we want to model $\mathbf{y}^T = [y_0, y_1, y_2, \dots, y_{n-1}]$. We assume that the output data can be represented (for a regression case) by a continuous function f through

$$\mathbf{y} = f(\mathbf{x}) + \epsilon.$$

Setting up the equations

In linear regression we approximate the unknown function with another continuous function $\tilde{\mathbf{y}}(\mathbf{x})$ which depends linearly on some unknown parameters $\boldsymbol{\theta}^T = [\theta_0, \theta_1, \theta_2, \dots, \theta_{p-1}]$.

The input data can be organized in terms of a so-called design matrix with an approximating function $\tilde{\mathbf{y}}$

$$\tilde{\mathbf{y}} = \mathbf{X}\boldsymbol{\theta},$$

The objective/cost/loss function

The simplest approach is the mean squared error

$$C(\Theta) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 = \frac{1}{n} \left\{ (\mathbf{y} - \tilde{\mathbf{y}})^T (\mathbf{y} - \tilde{\mathbf{y}}) \right\},$$

or using the matrix \mathbf{X} and in a more compact matrix-vector notation as

$$C(\Theta) = \frac{1}{n} \left\{ (\mathbf{y} - \mathbf{X}\theta)^T (\mathbf{y} - \mathbf{X}\theta) \right\}.$$

This function represents one of many possible ways to define the so-called cost function.

Training solution

Optimizing with respect to the unknown parameters θ_j we get

$$\mathbf{X}^T \mathbf{y} = \mathbf{X}^T \mathbf{X} \boldsymbol{\theta},$$

and if the matrix $\mathbf{X}^T \mathbf{X}$ is invertible we have the optimal values

$$\hat{\boldsymbol{\theta}} = \left(\mathbf{X}^T \mathbf{X} \right)^{-1} \mathbf{X}^T \mathbf{y}.$$

We say we 'learn' the unknown parameters $\boldsymbol{\theta}$ from the last equation.

Ridge and LASSO Regression

Our optimization problem is

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n} \left\{ (\mathbf{y} - \mathbf{X}\boldsymbol{\theta})^T (\mathbf{y} - \mathbf{X}\boldsymbol{\theta}) \right\}.$$

or we can state it as

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 = \frac{1}{n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2,$$

where we have used the definition of a norm-2 vector, that is

$$\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}.$$

From OLS to Ridge and Lasso

By minimizing the above equation with respect to the parameters θ we could then obtain an analytical expression for the parameters θ . We can add a regularization parameter λ by defining a new cost function to be optimized, that is

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\theta\|_2^2 + \lambda \|\theta\|_2^2$$

which leads to the Ridge regression minimization problem where we require that $\|\theta\|_2^2 \leq t$, where t is a finite number larger than zero. We do not include such a constraints in the discussions here.

Lasso regression

Defining

$$C(\mathbf{X}, \boldsymbol{\theta}) = \frac{1}{n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1,$$

we have a new optimization equation

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^p} \frac{1}{n} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1$$

which leads to Lasso regression. Lasso stands for least absolute shrinkage and selection operator. Here we have defined the norm-1 as

$$\|\mathbf{x}\|_1 = \sum_i |x_i|.$$

Lots of room for creativity

Not all the algorithms and methods can be given a rigorous mathematical justification, opening up thereby for experimenting and trial and error and thereby exciting new developments.

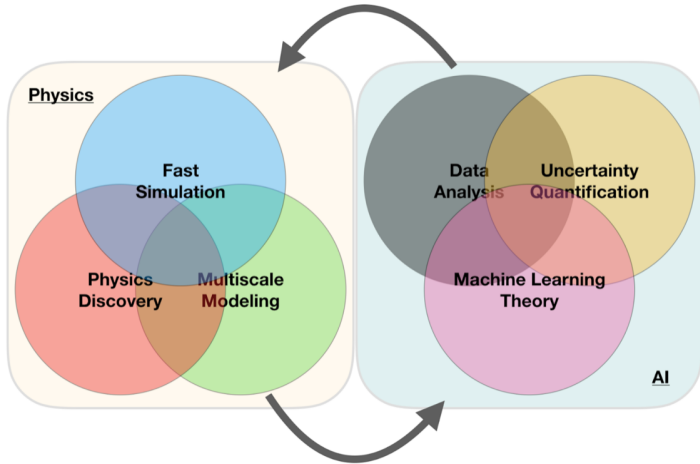
A solid command of linear algebra, multivariate theory, probability theory, statistical data analysis, optimization algorithms, understanding errors and Monte Carlo methods is important in order to understand many of the various algorithms and methods.

Job market, a personal statement: A familiarity with ML is almost becoming a prerequisite for many of the most exciting employment opportunities. And add quantum computing and there you are!

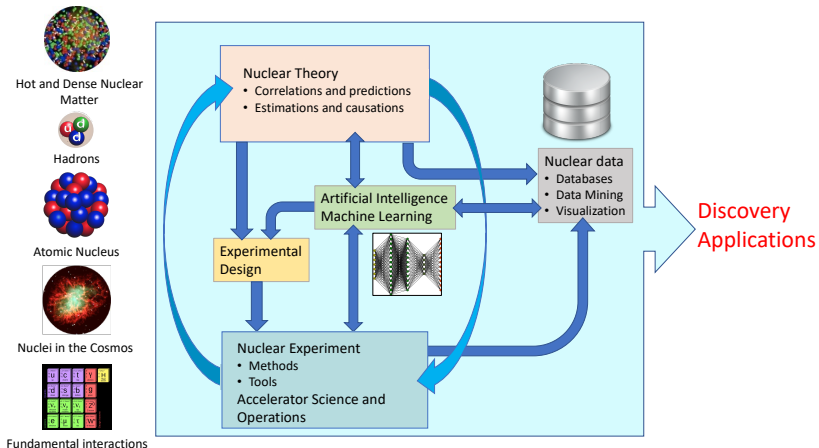
Selected references

- ▶ Mehta et al. and Physics Reports (2019).
- ▶ Machine Learning and the Physical Sciences by Carleo et al
- ▶ Artificial Intelligence and Machine Learning in Nuclear Physics, Amber Boehnlein et al., Reviews Modern of Physics 94, 031003 (2022)
- ▶ Dilute neutron star matter from neural-network quantum states by Fore et al, Physical Review Research 5, 033062 (2023)
- ▶ Neural-network quantum states for ultra-cold Fermi gases, Jane Kim et al, Nature Physics Communication, submitted
- ▶ Message-Passing Neural Quantum States for the Homogeneous Electron Gas, Gabriel Pescia, Jane Kim et al. arXiv.2305.07240,
- ▶ "Efficient solutions of fermionic systems using artificial neural networks, Nordhagen et al, Frontiers in Physics

Machine learning. A simple perspective on the interface between ML and Physics



ML in Nuclear Physics (or any field in physics)



Scientific Machine Learning

An important and emerging field is what has been dubbed as scientific ML, see the article by Deiana et al "Applications and Techniques for Fast Machine Learning in Science, Big Data 5, 787421 (2022):<https://doi.org/10.3389/fdata.2022.787421>"

The authors discuss applications and techniques for fast machine learning (ML) in science – the concept of integrating power ML methods into the real-time experimental data processing loop to accelerate scientific discovery. The report covers three main areas

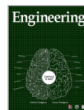
1. applications for fast ML across a number of scientific domains;
2. techniques for training and implementing performant and resource-efficient ML algorithms;
3. and computing architectures, platforms, and technologies for deploying these algorithms.



ELSEVIER



Engineering

Volume 6, Issue 3, March 2020, Pages 264-274



Research Artificial Intelligence—Review

A Survey of Accelerator Architectures for Deep Neural Networks

Yiran Chen^a  , Yuan Xie^b, Linghao Song^a, Fan Chen^a, Tianqi Tang^b

[Show more](#) 

 [Add to Mendeley](#)  [Share](#)  [Cite](#)

<https://doi.org/10.1016/j.eng.2020.01.007> 

[Get rights and content](#) 

Under a Creative Commons [license](#) 

 [open access](#)

Abstract

Physics driven Machine Learning

Another hot topic is what has loosely been dubbed **Physics-driven deep learning**. See the recent work on [Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators](#), Nature Machine Learning, vol 3, 218 (2021).

From their abstract

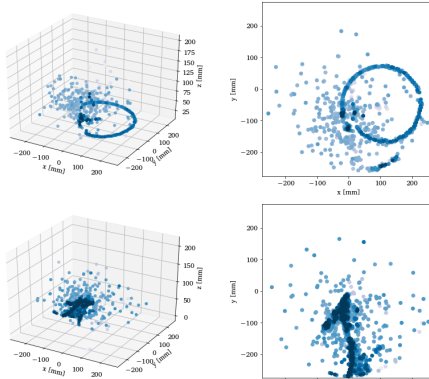
A less known but powerful result is that an NN with a single hidden layer can accurately approximate any nonlinear continuous operator. This universal approximation theorem of operators is suggestive of the structure and potential of deep neural networks (DNNs) in learning continuous operators or complex systems from streams of scattered data. ... We demonstrate that DeepONet can learn various explicit operators, such as integrals and fractional Laplacians, as well as implicit operators that represent deterministic and stochastic differential equations.

And more

- ▶ An important application of AI/ML methods is to improve the estimation of bias or uncertainty due to the introduction of or lack of physical constraints in various theoretical models.
- ▶ In theory, we expect to use AI/ML algorithms and methods to improve our knowledge about correlations of physical model parameters in data for quantum many-body systems. Deep learning methods show great promise in circumventing the exploding dimensionalities encountered in quantum mechanical many-body studies.
- ▶ Merging a frequentist approach (the standard path in ML theory) with a Bayesian approach, has the potential to infer better probability distributions and error estimates.
- ▶ Machine Learning and Quantum Computing is a very interesting avenue to explore.

Argon-46 by Solli et al., NIMA 1010, 165461 (2021)

Representations of two events from an Argon-46 experiment at MSU. Each row is one event in two projections, where the color intensity of each point indicates higher charge values recorded by the detector. The bottom row illustrates a carbon event with a large fraction of noise, while the top row shows a proton event almost free of noise.



Why Feed Forward Neural Networks (FFNN)?

According to the *Universal approximation theorem*, a feed-forward neural network with just a single hidden layer containing a finite number of neurons can approximate a continuous multidimensional function to arbitrary accuracy, assuming the activation function for the hidden layer is a **non-constant, bounded and monotonically-increasing continuous function**.

Universal approximation theorem

The universal approximation theorem plays a central role in deep learning. [Cybenko \(1989\)](#) showed the following:

Let σ be any continuous sigmoidal function such that

$$\sigma(z) = \begin{cases} 1 & z \rightarrow \infty \\ 0 & z \rightarrow -\infty \end{cases}$$

Given a continuous and deterministic function $F(\mathbf{x})$ on the unit cube in d -dimensions $F \in [0, 1]^d$, $\mathbf{x} \in [0, 1]^d$ and a parameter $\epsilon > 0$, there is a one-layer (hidden) neural network $f(\mathbf{x}; \Theta)$ with $\Theta = (\mathbf{W}, \mathbf{b})$ and $\mathbf{W} \in \mathbb{R}^{m \times n}$ and $\mathbf{b} \in \mathbb{R}^n$, for which

$$|F(\mathbf{x}) - f(\mathbf{x}; \Theta)| < \epsilon \quad \forall \mathbf{x} \in [0, 1]^d.$$

The approximation theorem in words

Any continuous function $y = F(\mathbf{x})$ supported on the unit cube in d -dimensions can be approximated by a one-layer sigmoidal network to arbitrary accuracy.

[Hornik \(1991\)](#) extended the theorem by letting any non-constant, bounded activation function to be included using that the expectation value

$$\mathbb{E}[|F(\mathbf{x})|^2] = \int_{\mathbf{x} \in D} |F(\mathbf{x})|^2 p(\mathbf{x}) d\mathbf{x} < \infty.$$

Then we have

$$\mathbb{E}[|F(\mathbf{x}) - f(\mathbf{x}; \Theta)|^2] = \int_{\mathbf{x} \in D} |F(\mathbf{x}) - f(\mathbf{x}; \Theta)|^2 p(\mathbf{x}) d\mathbf{x} < \epsilon.$$

More on the general approximation theorem

None of the proofs give any insight into the relation between the number of hidden layers and nodes and the approximation error ϵ , nor the magnitudes of \mathbf{W} and \mathbf{b} .

Neural networks (NNs) have what we may call a kind of universality no matter what function we want to compute.

It does not mean that an NN can be used to exactly compute any function. Rather, we get an approximation that is as good as we want.

Class of functions we can approximate

The class of functions that can be approximated are the continuous ones. If the function $F(\mathbf{x})$ is discontinuous, it won't in general be possible to approximate it. However, an NN may still give an approximation even if we fail in some points.

Illustration of a single perceptron model and an FFNN

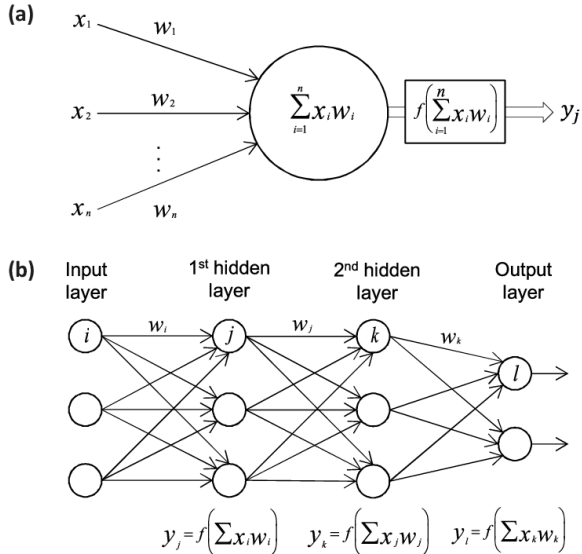


Figure: In a) we show a single perceptron model while in b) we display a network with two hidden layers, an input layer and an output layer.

Our network example, simple perceptron with one input

As a simple example we define now a simple perceptron model with all quantities given by scalars. We consider only one input variable x and one target value y . We define an activation function σ_1 which takes as input

$$z_1 = w_1 x + b_1,$$

where w_1 is the weight and b_1 is the bias. These are the parameters we want to optimize. This output is then fed into the **cost/loss** function, which we here for the sake of simplicity just define as the squared error

$$C(x; w_1, b_1) = \frac{1}{2}(a_1 - y)^2.$$

Optimizing the parameters

In setting up the feed forward and back propagation parts of the algorithm, we need now the derivative of the various variables we want to train.

We need

$$\frac{\partial C}{\partial w_1} \text{ and } \frac{\partial C}{\partial b_1}.$$

Using the chain rule we find

$$\frac{\partial C}{\partial w_1} = \frac{\partial C}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial w_1} = (a_1 - y)\sigma'_1 x,$$

and

$$\frac{\partial C}{\partial b_1} = \frac{\partial C}{\partial a_1} \frac{\partial a_1}{\partial z_1} \frac{\partial z_1}{\partial b_1} = (a_1 - y)\sigma'_1,$$

which we later will just define as

$$\frac{\partial C}{\partial a_1} \frac{\partial a_1}{\partial z_1} = \delta_1.$$

Implementing the simple perceptron model

In the example code here we implement the above equations (with explicit expressions for the derivatives) with just one input variable x and one output variable. The target value $y = 2x + 1$ is a simple linear function in x . Since this is a regression problem, we define the cost function to be proportional to the least squares error

$$C(y, w_1, b_1) = \frac{1}{2}(a_1 - y)^2,$$

with a_1 the output from the network.

```
# import necessary packages
import numpy as np
import matplotlib.pyplot as plt

def feed_forward(x):
    # weighted sum of inputs to the output layer
    z_1 = x*output_weights + output_bias
    # Output from output node (one node only)
    # Here the output is equal to the input
    a_1 = z_1
    return a_1

def backpropagation(x, y):
    a_1 = feed_forward(x)
    # derivative of cost function
    derivative_cost = a_1 - y
```

Central magic

Automatic differentiation