Statistics in Particle Physicists
LPC 2021, Fermilab

Lecture 4: Overview of Machine Learning

Harrison B. Prosper

Department of Physics, Florida State University

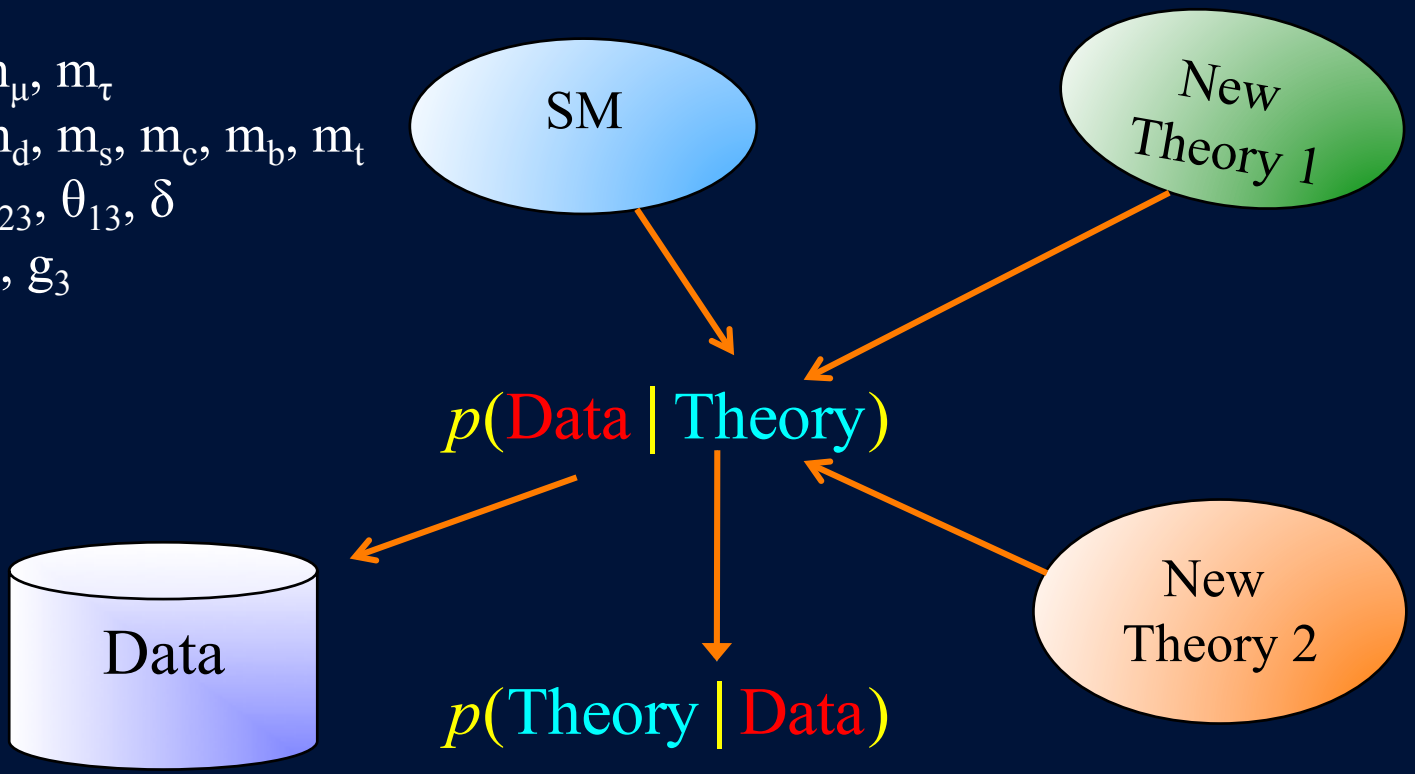
13 October, 2021

Outline

Recap

All interesting theories are ***multi-parameter*** models

m_e, m_μ, m_τ
m_u, m_d, m_s, m_c, m_b, m_t
θ_{12}, θ_{23}, θ_{13}, δ
g_1, g_2, g_3
θ_{QCD}
μ, λ



Basic *statistical* questions:

1. Which theories are preferred, given the data?
2. And which parameter sub-spaces within these theories?

Outline

What is machine learning?

According to IBM (https://www.ibm.com/cloud/learn/machine-learning):

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy.

However, I think a slightly better definition is:

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to build mathematical models that mimic intelligent behavior.

...as in the following example...

ARTICLE

doi:10.1038/nature24270

Mastering the game of Go without human knowledge

David Silver[1]*, Julian Schrittwieser[1]*, Karen Simonyan[1]*, Ioannis Antonoglou[1], Aja Huang[1], Arthur Guez[1], Thomas Hubert[1], Lucas Baker[1], Matthew Lai[1], Adrian Bolton[1], Yutian Chen[1], Timothy Lillicrap[1], Fan Hui[1], Laurent Sifre[1], George van den Driessche[1], Thore Graepel[1] & Demis Hassabis[1]

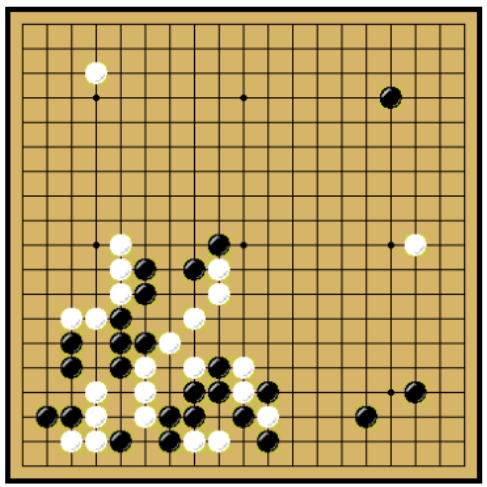
A long-standing goal of artificial intelligence is an algorithm that learns, *tabula rasa*, superhuman proficiency in challenging domains. Recently, AlphaGo became the first program to defeat a world champion in the game of Go. The tree search in AlphaGo evaluated positions and selected moves using deep neural networks. These neural networks were trained by supervised learning from human expert moves, and by reinforcement learning from self-play. Here we introduce an algorithm based solely on reinforcement learning, without human data, guidance or domain knowledge beyond game rules. AlphaGo becomes its own teacher: a neural network is trained to predict AlphaGo's own move selections and also the winner of AlphaGo's games. This neural network improves the strength of the tree search, resulting in higher quality move selection and stronger self-play in the next iteration. Starting *tabula rasa*, our new program AlphaGo Zero achieved superhuman performance, winning 100–0 against the previously published, champion-defeating AlphaGo.

David Silver *et al.* Nature, vol. **550**, 19 October 2017.

2.08×10^{170}

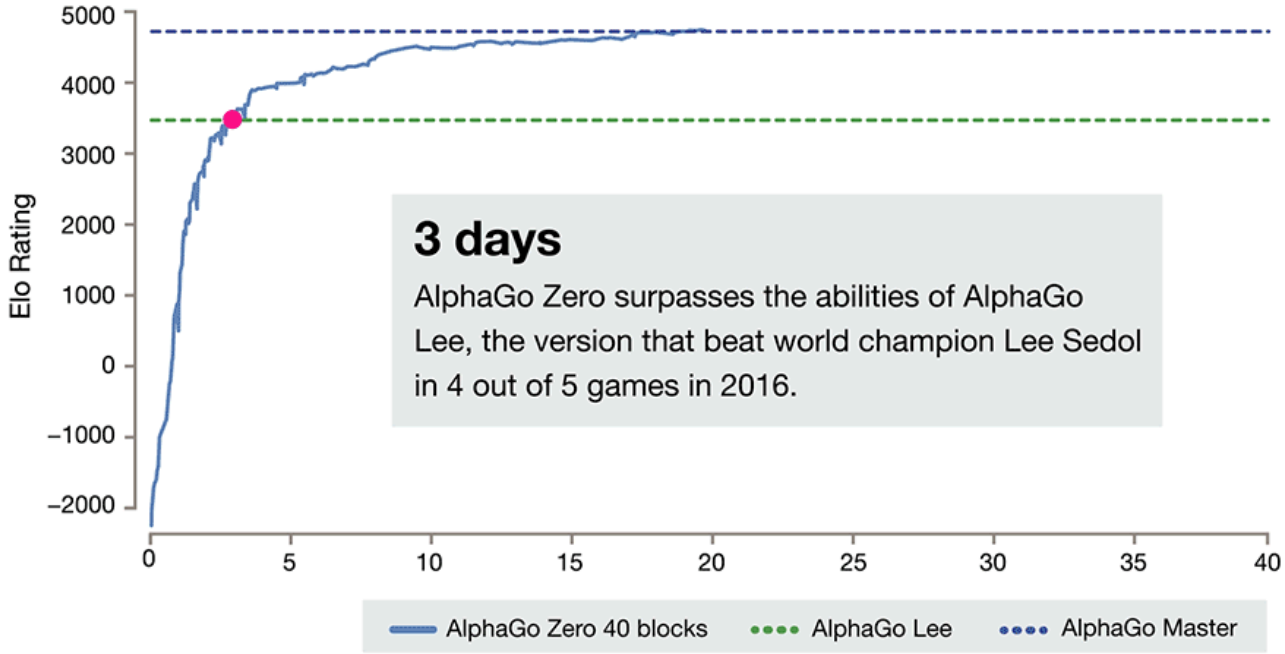
AlphaGo 4, Homo sapiens 1

2016 – Google's AlphaGo program beats Go champion Lee Sodol.



Photograph: Yonhap/Reuters

AlphaGo Zero



"Starting from random play, and given no domain knowledge except the game rules, AlphaZero achieved within 24 hours a superhuman level of play in the games of chess and shogi (Japanese chess) as well as Go, and convincingly defeated a world-champion program in each case."[1]

[1] D. Silver et al. "*Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm*", arXiv:1712.01815v1 [cs.AI] 5 Dec 2017.

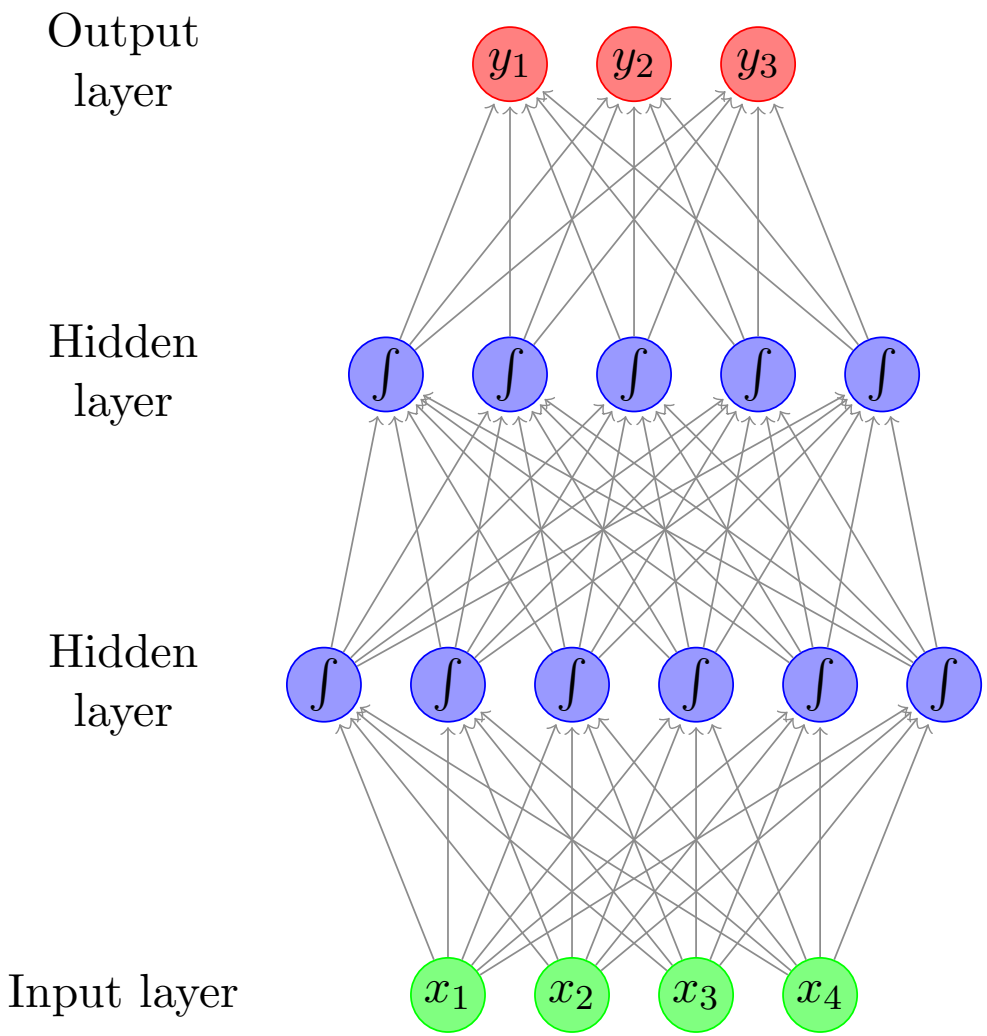
The AlphaZero breakthrough and the impressive developments in domains like natural language processing are primarily due to:

- increased computational power;
- the use of huge data sets;
- highly non-linear models with large numbers of parameters, and
- the use of *stochastic gradient descent* and its many variants.

Furthermore, a large number of different classes of models have been developed, each with thousands of variants.

Many of these models are variations and combinations of the deep feed forward neural network model shown on the following slide.

Deep feed forward neural network[2]



[2] https://web.stanford.edu/class/cs224n/

Machine learning models are trained, that is, fitted, by optimizing (usually minimizing) a function $R(\theta)$ called the empirical risk (aka: loss function, average loss, cost function, objective function).

Optimization is akin to hiking without a map in a highly corrugated landscape – for example, the Jura mountains (near CERN) – in dense fog.

All you can see is the terrain in your immediate vicinity and, on that basis, you must decide what step to take.

The optimization algorithms in machine learning are variations of Algorithm 1:

Algorithm 1 Gradient descent

Model $f(x, \theta)$
Data $T \leftarrow \{(x_i, y_i), i = 1, \ldots, n\}, \quad f_i \equiv f(x_i, \theta)$
Loss $L(y, f)$
while not converged **do**
    $b \leftarrow \{(x_i, y_i), i = 1, \ldots, m\} \sim T, \quad 1 \leq m \leq n$
    $\ell \leftarrow 0$
    for x_i, y_i in b **do**
        $\ell \leftarrow \ell + L(y_i, f_i)$
    end for
    $R(\theta) \leftarrow \ell \,/\, m$
    $\theta \leftarrow \theta - \eta \nabla R$
end while

Why does **Algorithm 1** work?

The update rule for the parameter vector θ,

$$\theta_{t+1} = \theta_t - \eta \nabla R,$$

implies

$$R(\theta_{t+1}) = R(\theta_t - \eta \nabla R),$$

$$= R(\theta_t) - \eta \nabla R \cdot \nabla R + \mathcal{O}(\eta^2).$$

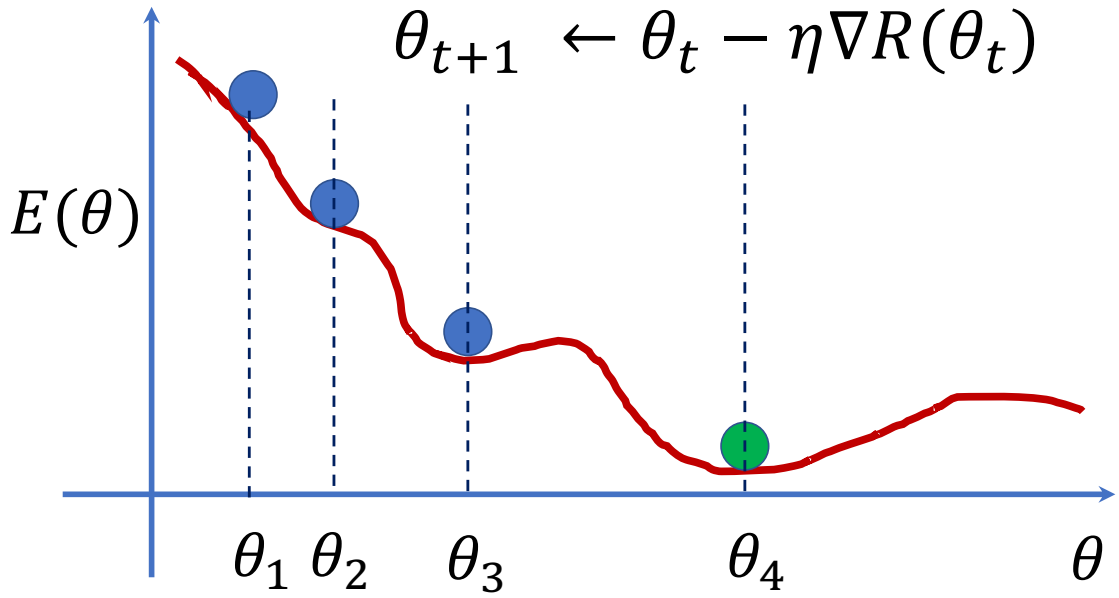
If the terms $\mathcal{O}(\eta^2)$ can be neglected, then it follows that

$$R(\theta_{t+1}) < R(\theta_t),$$

that is, Algorithm 1 moves to lower values of the empirical risk $R(\theta)$.

Stochastic gradient descent (SGD)

Unless the function $R(\theta)$ is simple, every optimization algorithm runs the risk of getting stuck in a local minimum that is a poor approximation to the global minimum.



The use of noisy gradients rather than exact ones increases the chance that the algorithm will escape from a sub-optimal local minimum and drastically increases the speed of convergence.

Outline

Machine learning (ML) has been used in particle physics and astronomy for decades. However, the exponential rise use of ML can be traced to a breakthrough that occurred in 2006.

- In 2006, the field of machine learning became HOT when AI pioneers Hinton, Osindero and Teh[3] succeeded in training a deep neural network (DNN) by initializing its parameters layer by layer.
- Each layer was trained to produce a representation of its inputs that served as the training data for the next layer. Then the network was refined using stochastic gradient descent.
- This breakthrough was viewed as compelling evidence that the training of DNNs requires careful initialization of parameters and sophisticated training algorithms.

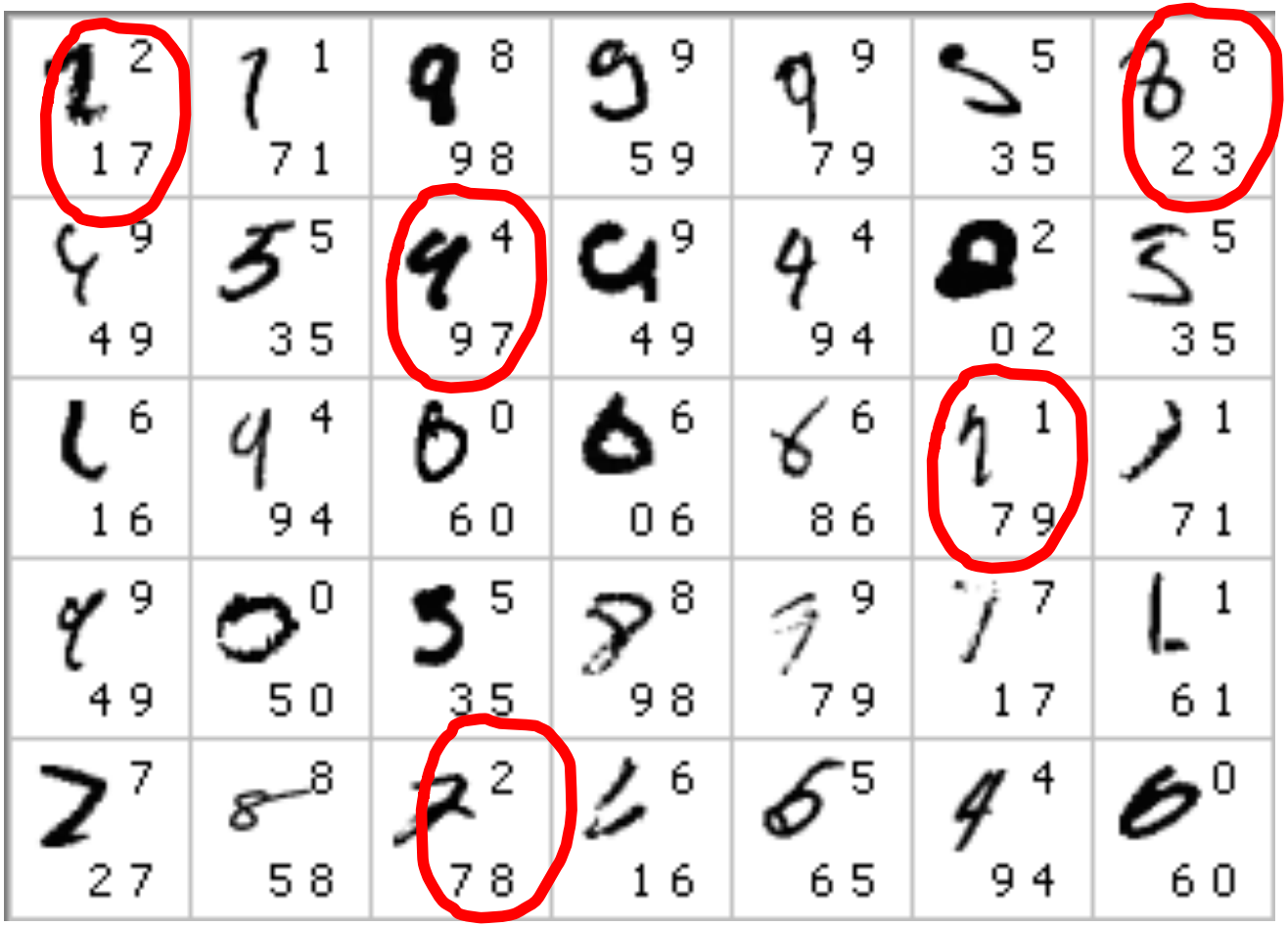
[3] G. E. Hinton, S. Osindero, and Y. Teh, *A fast learning algorithm for deep belief nets*, Neural Computation **18**, 1527-1554.

- But, in 2010, a counter example to that conventional wisdom was demonstrated by Cireșan *et al.*[4].
- The authors trained deep neural networks to classify the handwritten digits in the MNIST[5] data set, which comprises 60,000 $28 \times 28 = 784$ pixel images for training and 10,000 images for testing.
- The authors' model, with structure (784, 2500, 2000, 1500, 1000, 500, 10), outperformed all other methods that had been applied to the MNIST data set as of 2010. The error rate of this ∼12 million parameter model was 35 images out of 10,000.

[4] Cireșan DC, Meier U, Gambardella LM, Schmidhuber J. ,*Deep, big, simple neural nets for handwritten digit recognition.* Neural Comput. 2010 Dec; 22 (12): 3207-20.

[5] `http://yann.lecun.com/exdb/mnist/`

(784, 2500, 2000, 1500, 1000, 500, 10)



Upper right: correct answer; lower left answer of highest DNN output; lower right answer of next highest DNN output.

The lessons drawn were:

1) models need to be deep (hence deep learning),

2) huge amounts of data are needed to fit them, and

3) huge amounts of computing is a necessity.

During the last decade, these lessons have driven practice in AI and machine learning.

The largest natural language processing model, GPT-3 (released by OpenAI in 2020), has 175 billion parameters and was trained on most of the data on the Web[6].

This ferociously complicated model was optimized using variants of Algorithm 1.

But what exactly do ML models approximate?

[6] Bhavika Kanani, https://studymachinelearning.com/gpt-3/, 6 Oct. 2021

As noted earlier, the training of most ML models can be cast as an optimization problem whose goal is to minimize the average

$$R(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, f(x_i, \theta)) + C(\theta)$$

of a suitable loss function $L(y, f)$, possibly subject to a constraint $C(\theta)$.

The key point to note is that $R(\theta)$ approximates the functional

$$\begin{aligned} R[f] &= \int \left[\int dy\, L(y, f(x, \theta))\, p(y, x) \right] dx, \\ &\equiv \int G(f, x)\, dx, \end{aligned}$$

where $p(y, x)$ is the probability density of the targets y and inputs x. The goal of the optimization is to approximate the minimum of $R[f]$.

Let's consider an example.

Example (Quadratic Loss: $L(y, f) = (y - f)^2$)

For the quadratic loss,

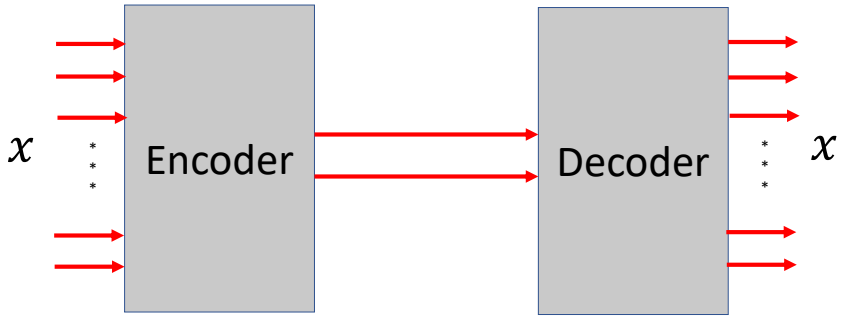
$$
\begin{aligned}
G(f, x) &= \int (y - f)^2 \, p(y, x) \, dy \\
&= p(x) \int (y - f)^2 \, p(y|x) \, dy, \\
\frac{\delta G}{\delta f} &= -2p(x) \int (y - f) \, p(y|x) \, dy = 0,
\end{aligned}
$$

which implies $\boxed{f(x, \theta^*) = \int y \, p(y|x) \, dy \text{ for some value of } \theta = \theta^*.}$

Conclusion If 1) the training data are sufficient and 2) $f(x, \theta)$ is sufficiently flexible (i.e., $\exists$ an $f(x, \theta)$ such that the functional derivative $\delta G / \delta f$ reaches zero) and 3) we use the quadratic loss then $f(x, \theta)$ will approximate the *mean* of the conditional density $p(y|x)$.

Example (Quadratic Loss: Mapping galactic photometric data to 2-D)

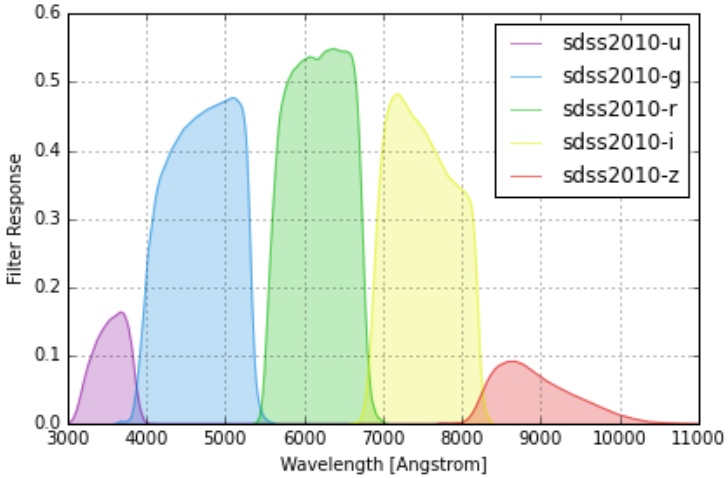
The model below is an example of an autoencoder, which we train



by minimizing the quadratic loss between the input to the encoder and the output of the decoder in order to map $x \rightarrow x$, via a 2-D space, where $x = u, g, r, i, z$ are SDSS photometric data for 50,000 galaxies and quasars.

Since we are using the quadratic loss function, $f(x, \theta)$ will approximate the integral $\int y\, p(y \,|\, x)\, dy = \int y\, \delta(y - x)\, dy = x$ provided that the conditions on the previous slide are met, *irrespective of the details of the autoencoder* $f(x, \theta)$. Note, by the way, that for autoencoders, the quadratic loss yields functions that are *unbiased* estimates of the data x.

Example (Quadratic Loss: Mapping galactic photometric data to 2-D)

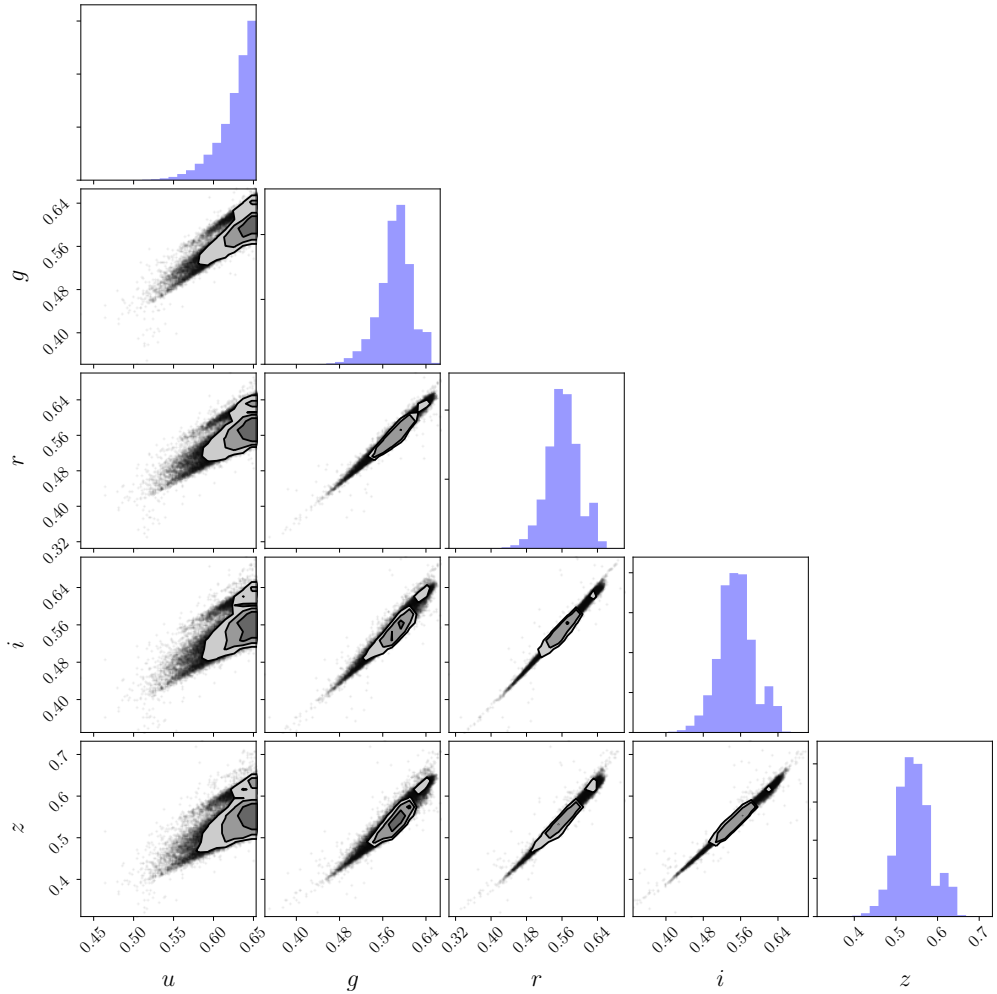


https://speclite.readthedocs.io/en/latest/filters.html

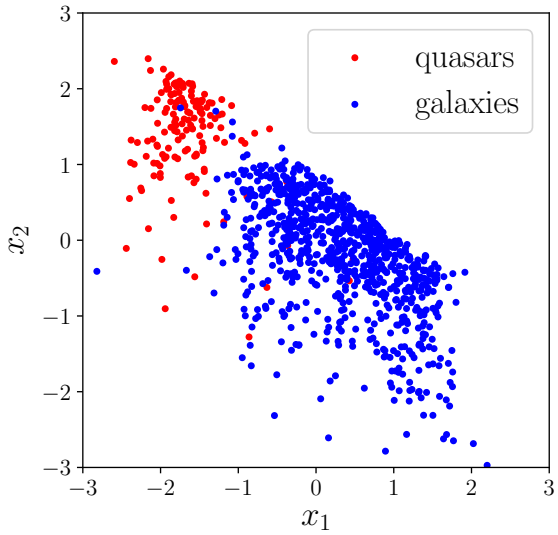
https://www.sdss.org/instruments/camera

The SDSS filters are shown above, while the data used in this example are taken from

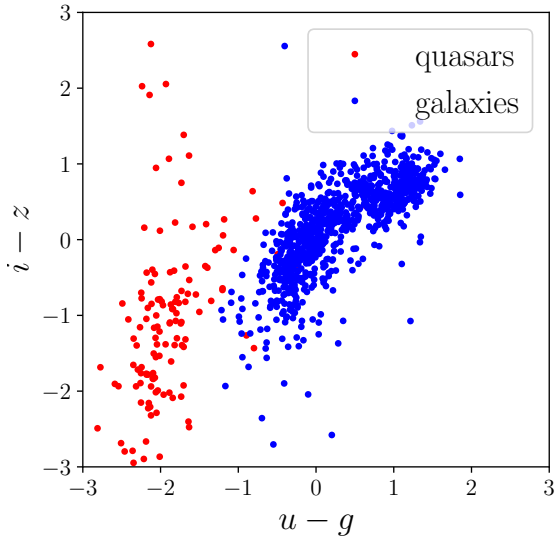
https://www.astroml.org/index.html



Example (Quadratic Loss: Mapping galactic photometric data to 2-D)



Autoencoder The plot to the left shows the mapping of the 5-D photometric data to the 2-D latent space of the autoencoder.



Astronomers Astronomers have known for a long time how to cluster photometric data in order to separate quasars from galaxies, as shown in the lower plot. However, the clustering by the autoencoder was *unsupervised*.

Outline

- Machine learning models are highly non-linear functions, which are functions of a large number of free parameters.
- These models are typically trained, that is, fitted, using a variation of stochastic gradient descent: 1) to speed up the training and 2) to introduce noise into the gradient calculations. The introduction of noise increases the chance that the optimization algorithm will escape from local minima.
- Whatever the complexity of an ML model, the quantity it approximates is determined largely by the choice of loss function.