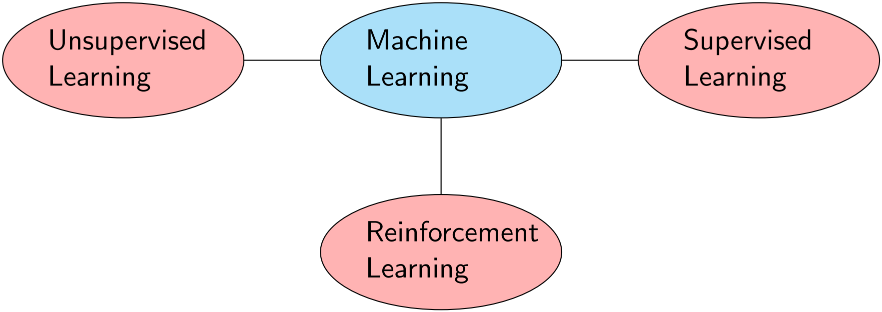
String Theory meets Machine Learning - Neural Networks

Robin Schneider

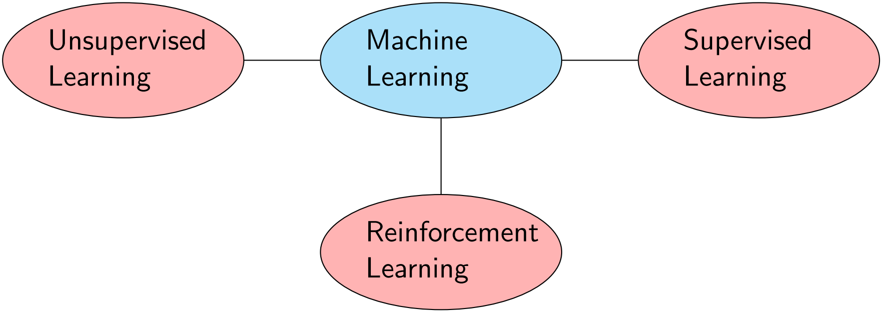
Uppsala University

November 2020

What is machine learning?



What is machine learning?



It's really just curve fitting, or 'regression', with a very, very large number of parameters.

Jared Kaplan

Why is ML useful for String theory?

- Universal Approximation theorem - A sufficient wide and deep NN can approximate almost any function we are interested in.

Why is ML useful for String theory?

- Universal Approximation theorem - A sufficient wide and deep NN can approximate almost any function we are interested in.
- Computations are (NP) hard [1809.08279].
  - Gröbner basis are double exponential
  - Non linear Diophantine equations are undecidable

Why is ML useful for String theory?

- Universal Approximation theorem - A sufficient wide and deep NN can approximate almost any function we are interested in.
- Computations are (NP) hard [1809.08279].
  - Gröbner basis are double exponential
  - Non linear Diophantine equations are undecidable
- Computations are plenty full.
  - Up to 10^{428} topological inequivalent CY 3-folds [2008.01730]
  - F-theory: $10^{272.000}$ flux vacua [1511.03209]

Overview - Five Sessions

1. Neural Networks and Experiments
   - Learning stability and Hodge numbers.
2. Regularization and Convolutional Neural Networks
   - Learning more Hodge numbers.
3. Hyperparameter optimization
   - Learning Ricci flat metrics
4. (Variational) Autoencoder
   - Clustering of standard like models
5. Reinforcement learning
   - Exploring standard like models

Literature

Physics:

- Fabian Ruehle - Data science applications to string theory
- Jared Kaplan - Notes on Contemporary Machine Learning for Physicists
- Pankaj Mehta et al. - A high-bias, low-variance introduction to Machine Learning for physicists

Literature

Physics:

- Fabian Ruehle - Data science applications to string theory
- Jared Kaplan - Notes on Contemporary Machine Learning for Physicists
- Pankaj Mehta et al. - A high-bias, low-variance introduction to Machine Learning for physicists

Machine Learning:

- Christoper M. Bishop - Pattern Recognition and Machine Learning
- Ian Goodfellow and Yoshua Bengio and Aaron Courville - Deep Learning

Probability Distributions

- likelihood function $p(D|\theta)$, observing data D, given parameters θ.
- joint probability distribution $p(D, \theta) = p(D|\theta)p(\theta)$.
- prior $p(\theta)$ belief
- posterior distribution $p(\theta|D)$, having parameters θ given data D.

Probability Distributions

- likelihood function $p(D|\theta)$, observing data D, given parameters θ.
- joint probability distribution $p(D,\theta) = p(D|\theta)p(\theta)$.
- prior $p(\theta)$ belief
- posterior distribution $p(\theta|D)$, having parameters θ given data D.

Central limit theorem the sum of a set of random variables has a distribution that becomes increasingly Gaussian:

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{1}{2\sigma^2}(x-\mu)^2\right] \tag{1}$$

Bayesian Inference and MLE

Sum and Product rule

$$\underbrace{p(D) = \sum_{\theta} p(D,\theta)}_{\text{sum rule}} \rightarrow \int p(D,\theta)d\theta = \int \underbrace{p(D|\theta)p(\theta)}_{\text{product rule}} d\theta = \mathbb{E}_{\theta}[p(D|\theta)] \tag{2}$$

Bayesian Inference and MLE

Sum and Product rule

$$\underbrace{p(D) = \sum_\theta p(D,\theta)}_{\text{sum rule}} \rightarrow \int p(D,\theta)d\theta = \int \underbrace{p(D|\theta)p(\theta)}_{\text{product rule}} d\theta = \mathbb{E}_\theta[p(D|\theta)] \tag{2}$$

Bayes theorem

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\int d\theta' p(D|\theta')p(\theta')} \tag{3}$$

Bayesian Inference and MLE

Sum and Product rule

$$\underbrace{p(D) = \sum_\theta p(D, \theta)}_{\text{sum rule}} \rightarrow \int p(D, \theta) d\theta = \int \underbrace{p(D|\theta)p(\theta)}_{\text{product rule}} d\theta = \mathbb{E}_\theta[p(D|\theta)] \tag{2}$$

Bayes theorem

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} = \frac{p(D|\theta)p(\theta)}{\int d\theta' p(D|\theta')p(\theta')} \tag{3}$$

Maximum (Log) Likelihood Estimation

$$\hat{\theta} = \arg\max_\theta \log p(D|\theta) \tag{4}$$

What is a Neural Network?

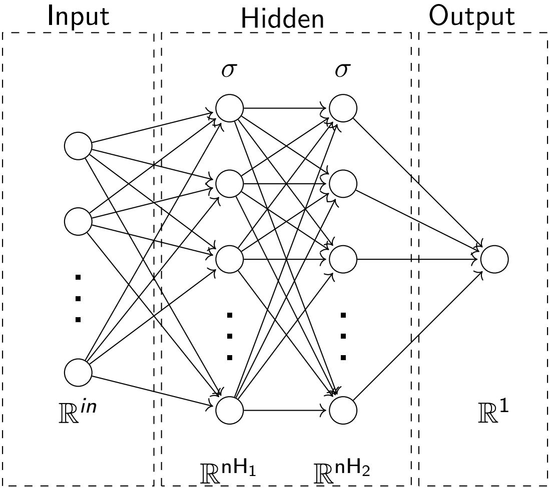


Figure: *A fully connected Neural Network of a regression problem.*

What is a Neural Network?

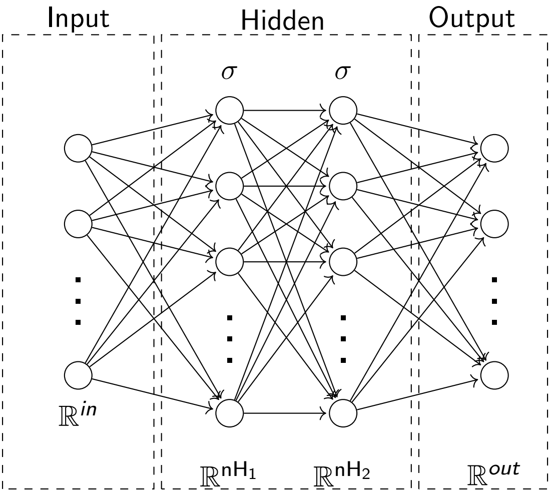


Figure: *A fully connected Neural Network of a classification problem.*

A Neural Network

We have a map $f(x; \{W, b\}) \simeq p(y|x, W, b)$:

$$f : \mathbb{R}^{in} \to \begin{cases} \mathbb{R}^1 & \text{regression} \\ \mathbb{R}^{out} & \text{classification} \end{cases} \tag{5}$$

at each hidden layer we have

$$H_1 : a_1 = \sigma_1(z_1 = W_1 x + b_1) \tag{6}$$

$$\vdots$$

and at the output

$$out = \begin{cases} W_n a_{n-1} + b_n, & \text{regression} \\ \text{softmax}(W_n a_{n-1} + b_n) & \text{classification} \end{cases} \tag{7}$$

Activation Functions

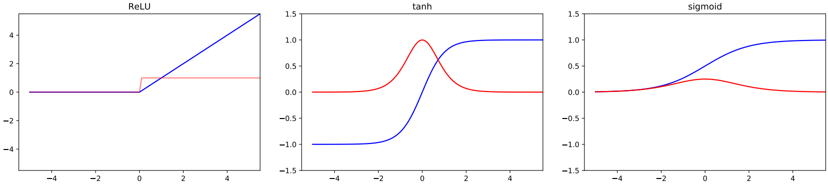


Figure: *ReLU, tanh and sigmoid function in blue, their derivatives in red.*

Typically we have, ReLU, tanh or sigmoid:

$$\sigma(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases}, \qquad \sigma(x) = \tanh(x), \qquad \sigma(x) = \frac{1}{1+e^{-x}}. \quad (8)$$

Optimization and loss function

We optimize with respect to some objective J. Take a classical regression problem, i.e. curve fitting with just one layer

$$J = \frac{1}{2} \sum_i (y_i - f(x_i; \theta))^2 \tag{9}$$

is just least square. Generally we denote NN parameters as $\theta = \{W_1, b_1, \dots\}$.

Optimization and loss function

We optimize with respect to some objective J. Take a classical regression problem, i.e. curve fitting with just one layer

$$J = \frac{1}{2}\sum_i (y_i - f(x_i;\theta))^2 \tag{9}$$

is just least square. Generally we denote NN parameters as $\theta = \{W_1, b_1, \dots\}$.

Parameter space of NN is non convex, which makes rigorous mathematical results more tricky.
Nevertheless, we can motivate least square with maximum likelihood estimation. Assume NN consisting of a single layer, i.e. simple regression.

Maximum Likelihood Estimator

Start from a Gaussian

$$p(y_i|x_i, \theta) = \mathcal{N}(y_i|\mu(x_i), \sigma^2(x_i)) \tag{10}$$

the MLE tells us for some data

$$\hat{\theta} = \arg\max_{\theta} \log p(y_i|x_i, \theta) \tag{11}$$

assume samples are independent and identically distributed

$$p(y|x, \theta) = \Pi_{i=1}^{n} \mathcal{N}(y_i|\mu(x_i), \sigma^2(x_i)) \tag{12}$$

Maximum Likelihood Estimator

Start from a Gaussian

$$p(y_i|x_i,\theta) = \mathcal{N}(y_i|\mu(x_i),\sigma^2(x_i)) \tag{10}$$

the MLE tells us for some data

$$\hat{\theta} = \arg\max_{\theta} \log p(y_i|x_i,\theta) \tag{11}$$

assume samples are independent and identically distributed

$$p(y|x,\theta) = \Pi_{i=1}^{n}\mathcal{N}(y_i|\mu(x_i),\sigma^2(x_i)) \tag{12}$$

we find the log likelihood

$$\begin{aligned} l(\theta) = \log p(y|x,\theta) &= \sum_{i=1}^{n} \log p(y_i|x_i,\theta) \\ &\overset{10}{=} -\frac{1}{2\sigma^2}\sum_{i}^{n}(y_i - \underbrace{\theta x_i}_{f(x_i;\theta)})^2 - \frac{n}{2}\log(2\pi\sigma^2) \end{aligned} \tag{13}$$

where we used $\mu = \theta x$ and some fixed variance $\sigma(x)^2 = \sigma^2$.

Stochastic Gradient Descent

Stochastic - we use (mini)batches.
Gradient descent, we compute the direction towards the next minimum w.r.t. the weights

$$g_\theta^k = \nabla_\theta J \tag{14}$$

and then update the weights

$$\theta_{k+1} = \theta_k - \epsilon g_\theta^k \tag{15}$$

where ϵ is called the learning rate.

Stochastic Gradient Descent

Stochastic - we use (mini)batches.
Gradient descent, we compute the direction towards the next minimum w.r.t. the weights

$$g_\theta^k = \nabla_\theta J \tag{14}$$

and then update the weights

$$\theta_{k+1} = \theta_k - \epsilon g_\theta^k \tag{15}$$

where ϵ is called the learning rate.
Momentum

$$g_\theta^k = \nabla_\theta J - \alpha g_\theta^{k-1} \tag{16}$$

Backpropagation I

How to optimize in practice? Compute derivatives

$$
\begin{aligned}
\partial_\theta J &= \sum_i^{\mathrm{nB}} \partial_\theta J(x_i, \theta) \\
&= \sum_i^{\mathrm{nB}} (y_i - f(x_i, \theta)) \partial_\theta f(x_i, \theta) \\
&= \sum_i^{\mathrm{nB}} (y_i - f(x_i, \theta)) x_i
\end{aligned} \tag{17}
$$

Backpropagation I

How to optimize in practice? Compute derivatives

$$\begin{aligned}
\partial_\theta J &= \sum_i^{\mathrm{nB}} \partial_\theta J(x_i, \theta) \\
&= \sum_i^{\mathrm{nB}} (y_i - f(x_i, \theta)) \partial_\theta f(x_i, \theta) \\
&= \sum_i^{\mathrm{nB}} (y_i - f(x_i, \theta)) x_i
\end{aligned} \tag{17}$$

Consider more layers

$$\begin{aligned}
&= \sum_i^{\mathrm{nB}} (y_i - f(x_i, \theta)) \partial_\theta a_{n-1} \\
&= \sum_i^{\mathrm{nB}} (y_i - f(x_i, \theta)) \sigma'(z_{n-1}) \partial_\theta z_{n-1} = \qquad \ldots
\end{aligned} \tag{18}$$

Backpropagation II

This brings us to backpropagation. We compute chain rule of each part, save results and multiply them together. Typical derivatives of activation functions

$$\partial_x \text{Relu}(x) = \begin{cases} 1 \text{ if } x > 0 \\ 0 \text{ else} \end{cases} \tag{19}$$

$$\partial_x \tanh(x) = 1 - \tanh^2(x) \tag{20}$$

$$\partial_x \text{sigmoid}(x) = \text{sigmoid}(x)(1 - \text{sigmoid}(x)) \tag{21}$$

Classification

Take only two classes with *cross entropy* as loss function

$$J = \frac{1}{n}\sum_i H(p(x_i), q(x_i)) \text{ with } H(p,q) = -\sum_j p_j(x)\log(q_j(x)) \tag{22}$$

where $p_j(x)$ are the discrete true labels and $q_j(x)$ the predictions. Consider logistic

$$q_{y=1} = \sigma(x;\theta) = \frac{1}{1+e^{-\theta x}} \text{ and } q_{y=0} = 1 - q_{y=1} \tag{23}$$

Then rewrite cross entropy as average over data points

$$J = -\frac{1}{n}\sum_i y_i \log(\sigma(x_i;\theta)) + (1-y_i)\log(1-\sigma(x_i;\theta)). \tag{24}$$

MLE

Likelihood of observing data $D = (x_i, y_i)$ is

$$P(D|\theta) = \Pi_i^n \sigma(x_i; \theta)^{y_i} (1 - \sigma(x_i; \theta))^{1-y_i} \tag{25}$$

Thus the log likelihood becomes

$$l(\theta) = \sum_{i=1}^{n} y_i \log(\sigma(x_i; \theta)) + (1 - y_i) \log(1 - \sigma(x_i; \theta)) \tag{26}$$

MLE

Likelihood of observing data $D = (x_i, y_i)$ is

$$P(D|\theta) = \Pi_i^n \sigma(x_i;\theta)^{y_i}(1-\sigma(x_i;\theta))^{1-y_i} \tag{25}$$

Thus the log likelihood becomes

$$l(\theta) = \sum_{i=1}^{n} y_i \log(\sigma(x_i;\theta)) + (1-y_i)\log(1-\sigma(x_i;\theta)) \tag{26}$$

Compute gradient for backpropagation

$$\partial_x \text{sigmoid}(x) = \text{sigmoid}(x)(1-\text{sigmoid}(x)) \tag{27}$$

such that

$$\partial_\theta J = \sum_{i=1} x_i(y_i - \sigma(x_i;\theta)) \tag{28}$$

More Classes and Layers - Softmax

One hot encoding. Treat N classes as vector $y_j \in \mathbb{Z}^N$ with all 0 except of a single 1. Use softmax activation function in last layer:

$$p_j(y|x;\theta) = \frac{e^{a_j}}{\sum_k^N e^{a_k}} \tag{29}$$

which has derivatives

$$\frac{\partial p_j(a)}{\partial a_k} = p_j(a)(\delta_{jk} - p_k(a)). \tag{30}$$

More Classes and Layers - Softmax

One hot encoding. Treat N classes as vector $y_j \in \mathbb{Z}^N$ with all 0 except of a single 1. Use softmax activation function in last layer:

$$p_j(y|x;\theta) = \frac{e^{a_j}}{\sum_k^N e^{a_k}} \tag{29}$$

which has derivatives

$$\frac{\partial p_j(a)}{\partial a_k} = p_j(a)(\delta_{jk} - p_k(a)). \tag{30}$$

Again take crossentropy as a loss function

$$J = -\frac{1}{n}\sum_{i=1}^{n}\sum_{j=1}^{N} y_{ij}\log(p_j(x_i;\theta)) \tag{31}$$

we take derivatives and find

$$\frac{\partial J}{\partial a_j} = \frac{1}{n}\sum_i p_j(a_i) - y_{ij}. \tag{32}$$

Experiment set up

How to conduct a ML experiment.

1. Define the problem

Experiment set up

How to conduct a ML experiment.

1. Define the problem
2. Create some data

Experiment set up

How to conduct a ML experiment.

1. Define the problem
2. Create some data
3. Make a train test split

Experiment set up

How to conduct a ML experiment.

1. Define the problem
2. Create some data
3. Make a train test split
4. Run many experiments with different NNs

Experiment set up

How to conduct a ML experiment.

1. Define the problem
2. Create some data
3. Make a train test split
4. Run many experiments with different NNs
5. Repeat
6. (Profit)

Application: Learning Stability

Reproduce some results of the early literature

- F. Ruehle - *Evolving neural networks with genetic algorithms to study the String Landscape*
- We will predict stability of line bundles over a CY manifolds using a fully connected NN.
- We will implement the fully connected NN from scratch.

Application: Learning Stability

Consider the following Complete Intersection Calabi Yau (bicubic):

$$M = \left[\begin{array}{c||c} 2 & 3 \\ 2 & 3 \end{array} \right] \tag{33}$$

The slope of a line bundle over it is given by

$$\mu(L) = d_{ijk} q^i t^j t^k = 6q_0 t_0 t_1 + 3q_0 t_1^2 + 3q_1 t_0^2 + 6q_1 t_0 t_1 \tag{34}$$

a line bundle is slope stable iff

$$\mu(L) = 0 \qquad \forall t^j > 0 \tag{35}$$

somewhere in the Kähler cone.