HEP(e)xML

Intro to Deep Learning

Johannes Wagner

Berkeley
UNIVERSITY OF CALIFORNIA

Last week recap

- We went through some of the basics of experimental particle physics
  - Particle detection, reconstruction and analysis
- Hands-on exercise showcased numpy/pyTorch fundamentals
- Today: Introduction of machine learning
  - Lecture will focus on broad concepts in ML and deep learning
  - Hands-on exercise will allow you to implement a simple deep neural network and apply it to a physics task

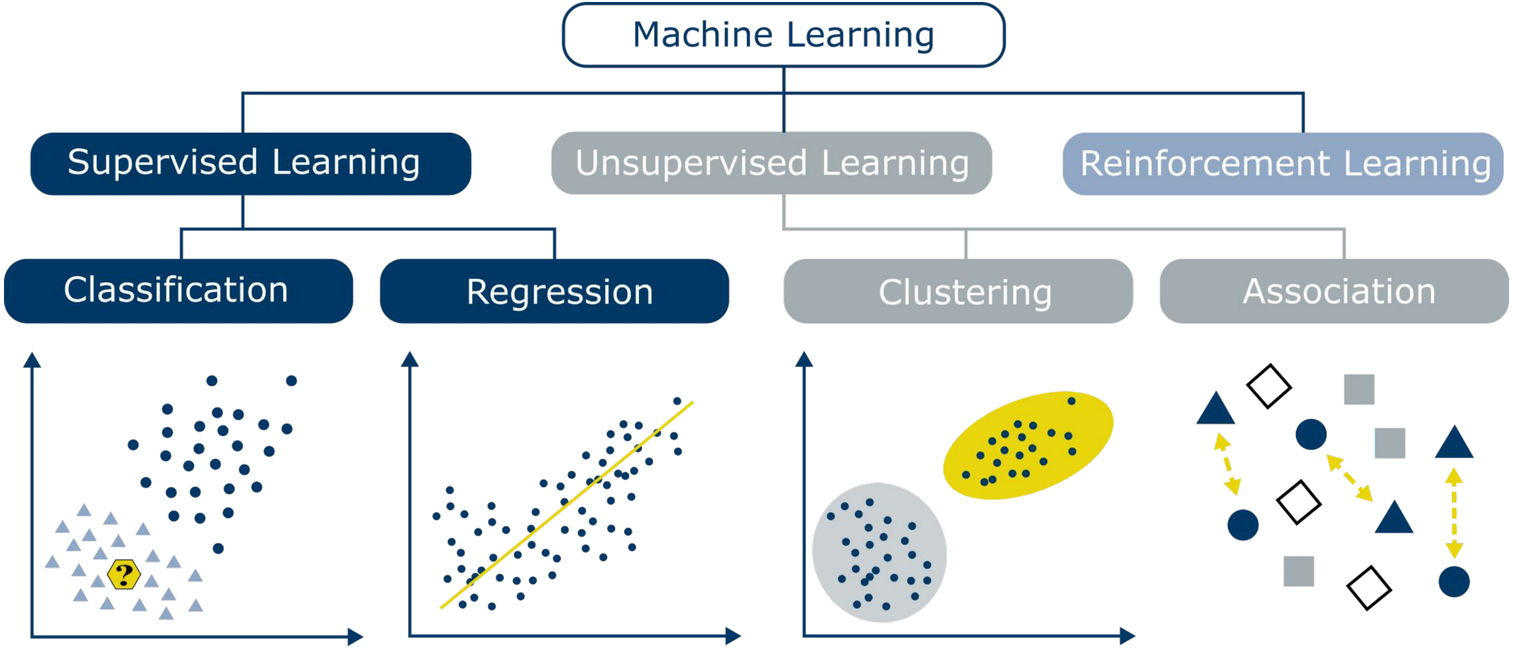
What is machine learning?

- Machine learning (ML) is a field that lives in the intersection between computer science and statistics
    - Related to, but not synonymous with artificial intelligence (AI)
- Problem: Given some data, how do we make predictions about them?
    - Need some function that maps our inputs to outputs that we are interested in
- Machine learning focuses on algorithms that can learn these functions without explicit programming

Berkeley
UNIVERSITY OF CALIFORNIA

Types of ML problems

Source: [Towards Data Science]



Supervised learning

- Algorithm learns association of input data to known output labels
- Requires labeled training data
  - In particle physics, this comes from MC simulations
- Classification: place input data into discrete predefined categories
  - Commonly needed in image processing
- Regression: determine relationship between input data and continuous output
  - Linear regression is one of the simplest forms of ML

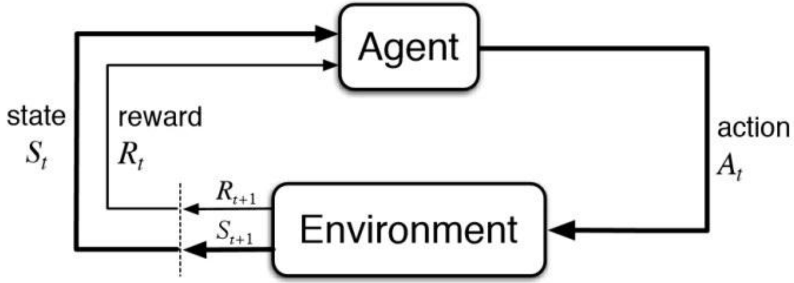
Berkeley
UNIVERSITY OF CALIFORNIA

Unsupervised learning

- Algorithm learns to make associations between input data
  - Patterns are learned more broadly
- Does not require labeled data
- Clustering: group inputs into a set number of “clusters”
  - Like classification but clusters are not predefined
- Anomaly detection: spot elements that are anomalous compared to the typical element
  - Useful in BSM searches where you don’t necessarily know what you’re looking for

Reinforcement learning

- Requires an agent and some predefined task we want to accomplish
    - This task represents a "goal state"
- Our agent has a space of actions they can take to change their state
    - Need a function to define the distance between their current state and the goal state
- Often used in robotics and with games



Source: KDNuggets

The beginnings of ML

- ML has its beginnings in the 1950's with the advent of computers
  - Desire to build artificial brain lead to development of neural network algorithms
- Invention of the "Perceptron" in 1958 constitutes the first artificial neural network
  - Perceptron is a linear binary classification algorithm
- Development of many ML algorithms throughout the 20th century
  - Various different kinds of neural network algorithms - focus of this workshop
  - Other ML algorithms such as k-nearest neighbors, decision trees, etc. (see CS 189A)

How a Perceptron "learns"

- Given some input data $\{\mathbf{x}_j\}$, a perceptron predicts output class labels $\{y_j\}$ where $y_j(t) = f[\mathbf{w}(t) \cdot \mathbf{x}_j]$ and $f(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{w} \cdot \mathbf{x} + b > 0, \\ 0 & \text{otherwise} \end{cases}$
- b is called the bias and shifts the decision boundary
- **w** represents a trainable vector of weights, which are updated by the rule $w_i(t+1) = w_i(t) + r \cdot (d_j - y_j(t))x_{j,i}$, where d_j are the true labels
- This process is repeated until the **w** vector no longer changes
- This is simple, but not very useful unless we have a problem that can be solved with a linear function

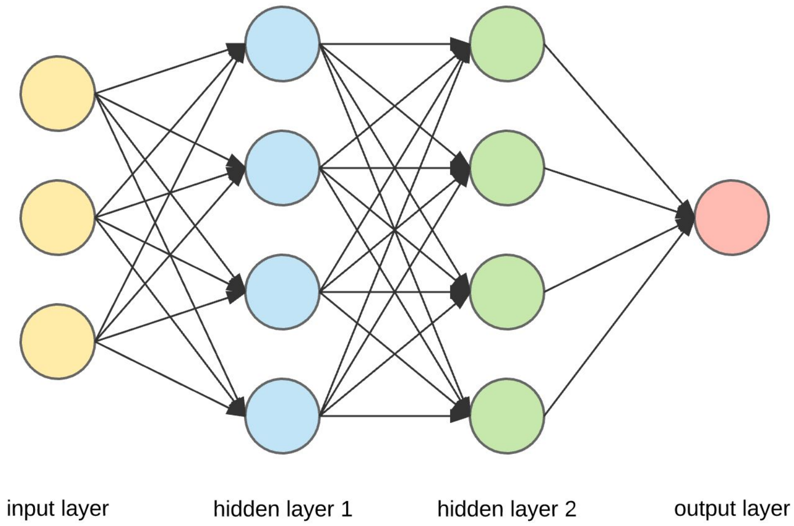
Berkeley
UNIVERSITY OF CALIFORNIA

The “deep learning” revolution

- “Deep neural networks” are the most basic “deep learning” algorithms
  - Generalization of perceptrons that were not feasible to use until ~15 years ago due to lack of computing power
  - They are able to express arbitrary functions, not just linear ones
- Idea: stack multiple linear layers (i.e. perceptrons) together and connect them via “activation functions”
- Updating the weights to train the network is harder since the relationship between inputs and outputs is more complicated

Overview of DNN's

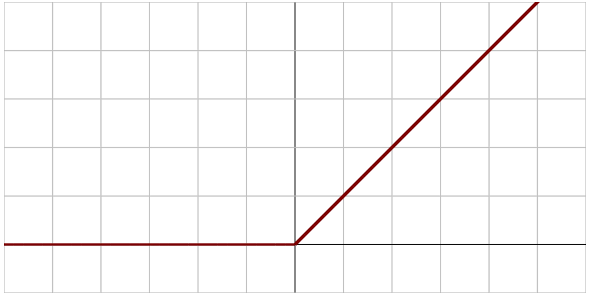
- Each node in the input layer represents one input feature
- Each arrow in the diagram represents a weighted connection to the next layer
- At each node, incoming weighted connections are summed and passed through an activation function to the next layer



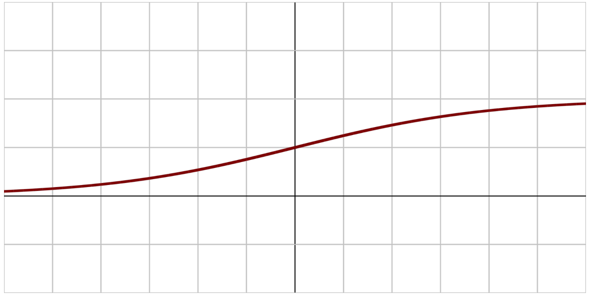
Source: [Towards Data Science]

Activation functions

- Activation functions allow deep neural networks to model nonlinear functions
  - Successive linear matrix multiplications are still linear
- Should ideally be continuously differentiable
  - It’s advantageous if the gradients don’t tend to 0 or blow up too quickly anywhere in the domain
- Popular choices are ReLU or Sigmoid functions
  - Often used for regression/classification respectively



ReLU



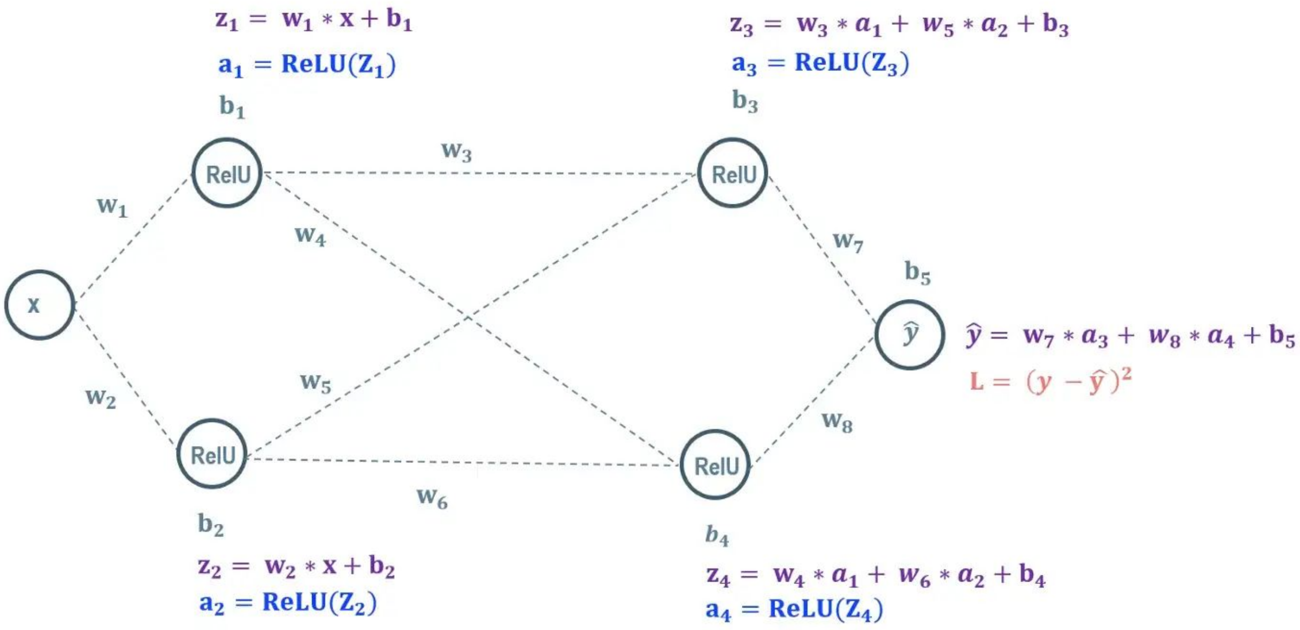
Sigmoid

Berkeley UNIVERSITY OF CALIFORNIA

The forward pass

Source: [Towards Data Science]()

- Evaluation consists of matrix algebra
- Weights are updated based on backpropagation gradient descent



Berkeley UNIVERSITY OF CALIFORNIA

Loss functions

- In order to understand how to update our network, we need some measure of how wrong our current predictions are: Loss functions!
  - Function that evaluates to a high number when our prediction is far off and a low number when it’s close
- Classification: binary cross-entropy loss

$$L_{BCE} = -\frac{1}{N}[\Sigma_{j=1}^{N} y_j log(\hat{y}_j) + (1 - y_j) log(1 - \hat{y}_j)]$$

- Regression: mean-squared error loss

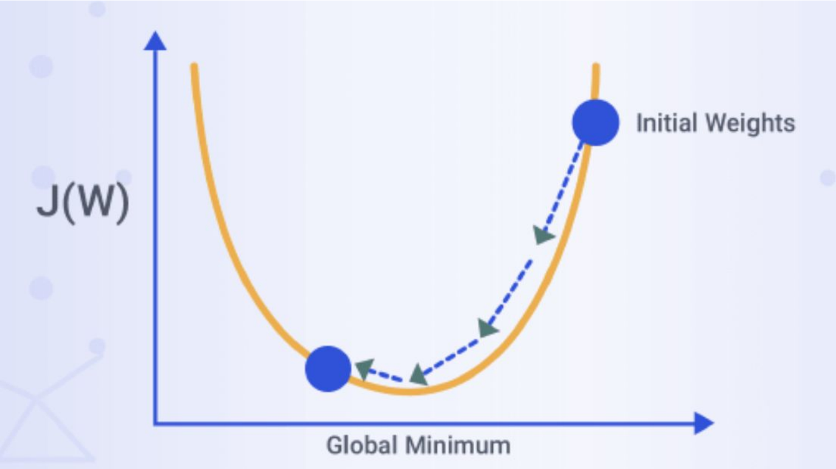
$$L_{MSE} = \frac{1}{N}\Sigma_{j=1}^{N}(\hat{y_j} - y_j)^2$$

Berkeley
UNIVERSITY OF CALIFORNIA

Backpropagation gradient descent

- Training involves minimizing loss function
  - Loss is a function of NN weights, biases
- If we take the gradient with respect to the weights and biases, we get a set of directions we need to take a step in for loss minimization
  - Learning rate determines step size
- Usually this is done by taking the average gradient for some “batch” of data

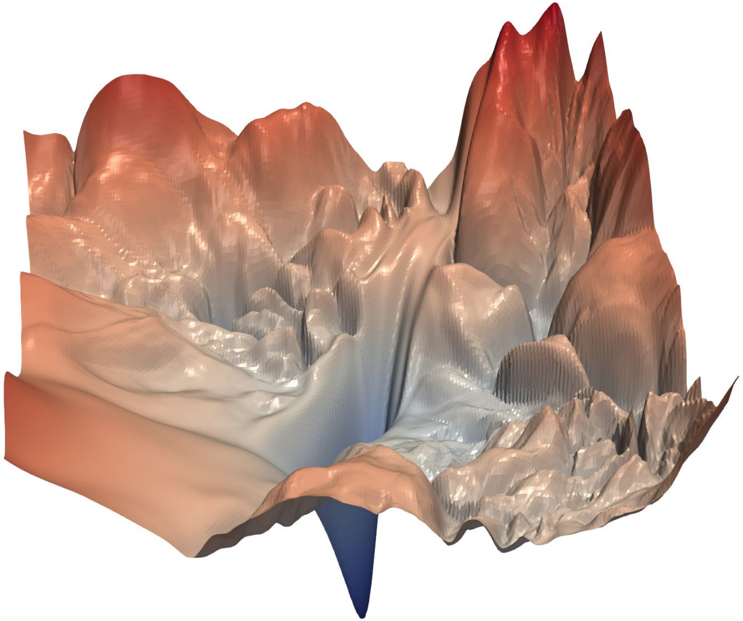
Source: akira.ai



Berkeley
UNIVERSITY OF CALIFORNIA

The loss landscape

- Loss landscape is a many-dimensional space (lots of weights in a NN) with a very complicated structure
  - Lots of local minima
- Need to find reliable ways to approach the global minimum without getting stuck
  - Choose the right learning rate



Source: [arxiv]

Berkeley
UNIVERSITY OF CALIFORNIA

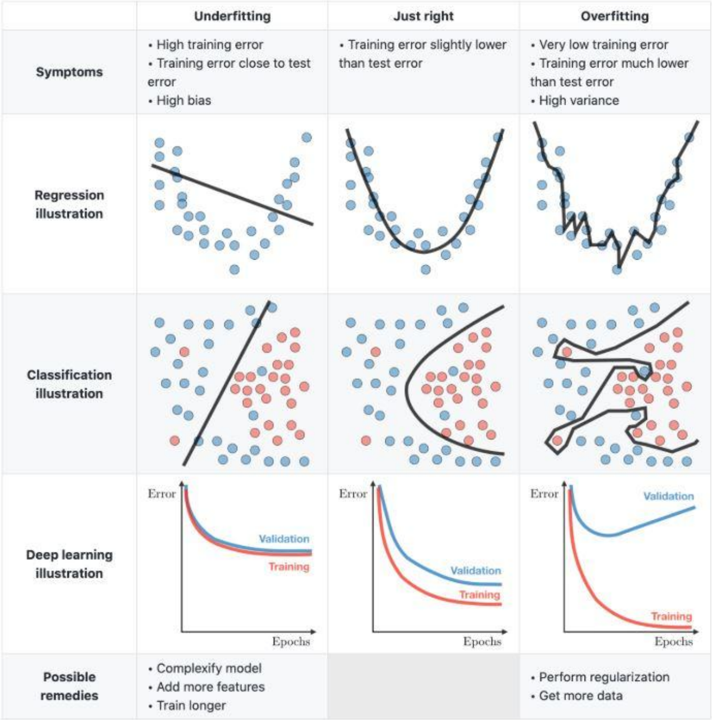
Beyond standard gradient descent

- Optimizers can use additional techniques to make loss minimization more effective
- Momentum: Incorporate previous gradient values into calculation of next step - keep “momentum” in some direction over time
- Adaptive gradients: Adapt step size along each dimension automatically based on smoothness of past gradients (smoother = larger step size)
- ADAM optimizer used commonly in modern applications
    - Gradient descent + momentum + adaptive gradients

Berkeley
UNIVERSITY OF CALIFORNIA

Overfitting

- Overfitting happens when we train our neural network to match our training data too well
- Our data will have statistical fluctuations -> we don't want to fit these!
- Can check for this by separating out a slice of the dataset for "validation"
    - Other techniques such as "dropout" also help

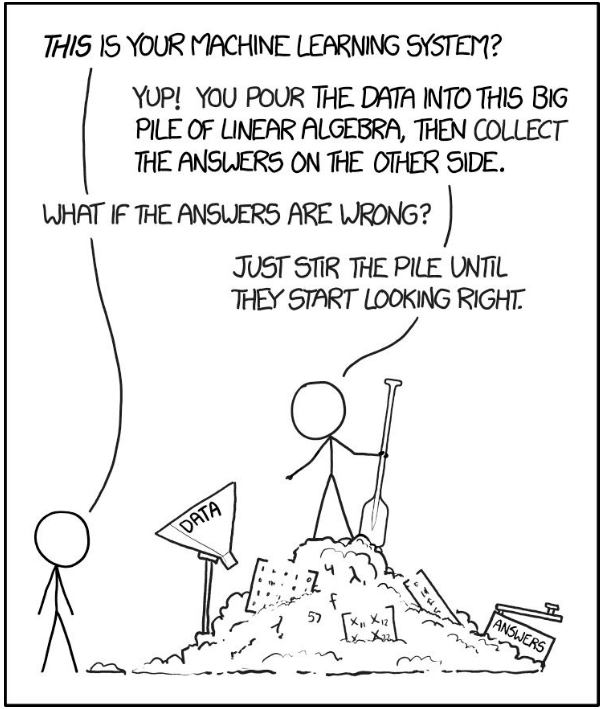


Source: Kaggle

Berkeley UNIVERSITY OF CALIFORNIA

The learning process

- Neural networks start with randomly initialized weights -> random predictions
- Since we have true labels, we can quantify how wrong we are and how we need to update our network weights to get better predictions
- If we follow the gradient of our loss function, our results will improve!
  - Usually multiple passes through the dataset ("epochs")



Source: xkcd

Training data selection

- Once we have a trained neural network, we can apply it to new data we don’t have the labels for!
- Neural network predictions are entirely dependent on what its shown during training
  - It won’t be able to generalize if given an unreliable/biased training sample
- Carefully selecting your training data is one of the most important tasks when developing a NN algorithm

Recap: What is a deep neural network?

- A highly nonlinear function with many parameters that takes a vector as input and returns a scalar or vector
- A loose model of the neuronal connections in a human brain
- A trainable algorithm that can be applied to data to extract some relevant quantity from its inputs

Hands-on exercise #2

Deep Neural Networks

Berkeley
UNIVERSITY OF CALIFORNIA

Exercise agenda

- Exercise #1: pyTorch (Credit: pyTorch Team)
    - We will finish the rest of the chapters starting from “Build Model”
- Exercise #2: “Intro to Deep Learning” Jupyter notebook
- Please take a moment to fill out the feedback form when you’re done

Berkeley
UNIVERSITY OF CALIFORNIA