Physics 361 - Machine Learning in Physics

Lecture 6 – Basics of Machine Learning

Feb. 8th 2024

Moritz Münchmeyer
Unit 2: Machine Learning Basics

2.2 Neural Network Basics (Supervised Learning)
Supervised machine Learning

We want to learn a complicated non-linear function to map input $\hat{x}$ to output $\hat{y}$.

\[ \hat{x} \rightarrow f_{\theta}(\hat{x}) \rightarrow \hat{y} \]

$\theta$: model parameters = weights

We want to find the model parameters by minimizing some loss function on our training data.

E.g., \[
L^{\text{MSE}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} (y_i - f_{\theta}(\hat{x}_i))^2
\]
A very simple neural network

- We need some way to specify the function $f(x)$. It turns out that a very simple "architecture" work very well in practice:

  Linear transformations combined with element-wise non-linearities

- A linear transformation / linear layer* is given by
  \[
  \vec{y} = f(\vec{x}) = W \vec{x} + \vec{b}
  \]

  \[
  \vec{x} : N_{\text{dim}} \quad \vec{y} : M_{\text{dim}}
  \]

  \[
  \text{matrix of weights} \quad \text{bias}
  \]

- The most common non-linearity / activation function is the ReLU "rectified linear unit".*

* fully connected linear layer
• **ReLU** is a 1-dimensional function

![ReLU function diagram](image)

Piecewise linear function.

Alternatives: tanh, sigmoid

• A **multilayer perceptron (MLP)** is a "deep neural network" made out of a stack of the 2 building blocks:

\[
X \rightarrow \text{Linear} \rightarrow \text{ReLU} \rightarrow \text{Linear} \rightarrow \text{ReLU} \rightarrow \ldots \rightarrow \hat{y}
\]
The combination of ReLU and linear layer can be written as

\[ f^i(x) = \max \left( 0, W^i \cdot x^i + b^i \right) \]

If we stack these layers:

\[ \max (0, W^i_{n,j} \cdot \max (0, W^j_{n-1,k} \cdot \max (0, \cdots) + b^j_{n-1}) + b^i_n) \]

Now we have a function that we can "fit" to the training data.
Loss for classification

For classification we want the NN output to be the probabilities of the various classes. For example, given an image $x$, the output $y_j$ is transformed to $P_j(x)$ using the softmax function:

\[
P_j(x) = \frac{e^{y_j}}{\sum_i e^{y_i}}
\]

A probability distribution needs to have only positive probabilities and they must sum to 1. To achieve this we need the softmax function.
The typical loss function for classification is the negative log likelihood (cross-entropy) predicted by \( N \),

\[
L(\theta) = - \sum_{i=1}^{N_{\text{examples}}} \sum_{j=1}^{N_{\text{classes}}} y_{ij} \log(p_{ij}(\hat{x}_i))
\]

\( y \) is a one-hot vector which is 1 for the right class.
Optimization of the parameters

* The last element we need is the "optimization procedure" to find good weights $\theta^*$ from minimizing the loss function.

* Almost all algorithms used in practice to do this are versions of stochastic gradient descent:

$$\theta_{n+1} = \theta_n - \varepsilon \nabla_{\theta} L(\theta_n, i, x)$$

  - new parameters at step $n+1$
  - current parameters
  - learning rate
  - gradient of the loss function
  - stochastic means we evaluate the gradient at each step using only a subset of the training data, called the "mini-batch"
In practice, to calculate this gradient we use "auto-differentiation". We'll discuss optimization and auto-differentiation in more detail soon.
Unit 2: Machine Learning Basics

2.4 Training our first model
Train an MLP on SUSY data

• We will use the simulated collider data. The goal will be to discriminate SUSY (supersymmetry) events from non-SUSY events. The data is from the paper https://www.nature.com/articles/ncomms5308.

• We use a modified version of the code from https://physics.bu.edu/~pankajm/MLnotebooks.html which was written for the review https://arxiv.org/abs/1803.08823 A high-bias, low-variance introduction to Machine Learning for physicists.

• We will use python with the pytorch framework (it does the auto-differentiation). Pytorch is the most widely used tool in ML research (followed probably by JAX now).

• We will use Google Colab to run the python code. Colab provides free computational resources, including GPUs, with a nice interface based on python’s Jupyter.

• I uploaded a basic python tutorial to canvas in files/python/python tutorial (from a previous class, Physics 249). You will have to self study some python if you don’t have prior experience.
Physics Background

- Using the dataset from the UC Irvine ML repository produced by MC simulations to contain events with 2 leptons (electrons or muons)

- These events with 2 leptons with large $p_T$ can occur in SUSY models or within the SM.

- 18 kinematic variables ("features") are recorded for each event.

- We train a MLP classifier to classify the events into SUSY or SM background.

![Diagrams for SUSY benchmark](image)

**Figure 4 | Diagrams for SUSY benchmark.** Example diagrams describing the signal process involving hypothetical supersymmetric particles $\chi^\pm$ and $\chi^0$ along with charged leptons $\ell^\pm$ and neutrinos $\nu$ (a) and the background process involving $W$ bosons (b). In both cases, the resulting observed particles are two charged leptons, as neutrinos and $\chi^0$ escape undetected.

Baldi et al, Nature Communications, Volume 5, Article number: 4308 (2014)
The rest of this section will be presented in Colab. I will upload the Colab notebook on the course page, and you can download it from there, upload it to Colab, and run it yourself.

```python
[23] import torch.nn as nn  # construct NN

class model(nn.Module):
    def __init__(self, high_level_feats=None):
        # inherit attributes and methods of nn.Module
        super(model, self).__init__()

        # an affine operation: y = Wx + b
        if high_level_feats is None:
            self.fc1 = nn.Linear(18, 200) # all features
        elif high_level_feats:
            self.fc1 = nn.Linear(10, 200) # low-level only
        else:
            self.fc1 = nn.Linear(8, 200) # high-level only

        self.fc2 = nn.Linear(200, 100) # see forward function for dimensions
        self.fc3 = nn.Linear(100, 2)

    def forward(self, x):
        """Defines the feed-forward function for the NN."

        A backward function is automatically defined using `torch.autograd`

        Parameters
        ----------
        x : autograd.Tensor
            input data
```

3m 10s  completed at 9:19 AM
Course logistics

• Reading for this lecture:
  • For example: Deeplearningbook.org chapter 6.

• Problem set: First problem set to appear tomorrow.