Deep Learning 101

Flatiron Wide Algorithms and Mathematics (FWAM!)

Gabriella Contardo (Flatiron Institute, Center for Computational Astrophysics)
gecontardo@flatironinstitute.org
Outline of tutorial part I

- Machine learning principles
- What is *deep learning*?
- Neural networks:
  - Components
  - Training
  - Fully Connected / Multi-Layer Perceptron (MLP)
  - Convolutional neural networks
A bit of definitions

Machine Learning (ML):

- Relying on data analysis to automate model building to perform certain tasks.
- Model learns from data, extracts patterns, and makes some ‘prediction’
- Learning can be supervised, unsupervised, or semi-supervised.
- Various type of models: SVM, Random Forest, KNN, neural networks, etc.

Deep Learning:

- Subgroup of ML based on artificial neural networks
- Strongly rely on “pattern” extraction and representation learning
- Very good ability to approximate complex functions and generalize
- Needs (lots of) labeled data
Some Machine Learning principles

3 main “components”:

- Data
- Machine or model
- Criterion (learning and evaluation)

Goal: Automatically extract relevant information from the data that generalize well to infer on new data (‘test’ data from similar distribution, but different from training examples)

Learn a model on training data to optimize some criterion, select the best on validation data, then infer on test data.
Supervised Learning

Training data is composed of

- Examples (“inputs”)
- ‘Targets’ --or label(s)-- (i.e outputs) for each example.

E.g. image classification.

Different type of targets => different learning problem:

- Limited set of values, categorical : classification (multi-class, multi-label,...)
- Continuous values : regression
- Other type e.g. noisy targets, weak supervision etc.

Goal: Find an approximation of the labeling process.
What are neural networks?

- Non-linear, highly parametric, functions’ approximators
- Loosely inspired from brain: network of interconnected neurons that “activate” or not.
- Composition of stacked ‘layers’ of weights (parameters) producing neurons.
Let’s start with a very simple one: Perceptron

- Like linear regression: $\sum_i w_i x_i$
- But with an additional activation function

Here: “step” function, threshold to obtain binary outputs.
‘Vanilla’ Neural Networks: Multi Layer Perceptron (MLP)

- Also called ‘fully-connected’ feed-forward neural networks.
- Stack ‘perceptron’ on several layers.
- Composition of functions:

\[
\begin{align*}
out &= \sigma(W_H * l_{H-1} + b_H) \\
l_h &= \sigma(W_h * l_{h-1} + b_h) \\
out &= \sigma(W_H * \sigma(W_{H-1} * \sigma(\ldots \sigma(W_1 * input + b_1) + \ldots) + b_{H-1}) + b_H)
\end{align*}
\]

- Each hidden layer has a latent size (output size)

Glossary:
- Input nodes, Output nodes
- Connections / weights
- Activation function
- Hidden layers
(Some) Activation Functions:

- **Sigmoid:**

- **Tanh**

- **Rectified Linear Unit (ReLU) / LeakyReLU:**

- **Softmax:** all neurons in [0,1] and sum to 1 (good for probability distribution).
Training : Finding the “best” weights (= parameters)

Remember that ‘cost’ function (criterion) ?

=> Goal is to minimize that

How ? Using gradient descent and backpropagation of the gradients !
Learning algorithm

- “Forward pass”: Feed an example (or more) to the network
- Compute error with regard to your criterion (compared to expected targets)
- Compute gradients
- Update weights
- Repeat until stopping criterion
Backpropagation?

- Goal of gradient: update the weights in the “right” direction
  - Which weight is responsible for the error and on what ‘amount’?

- Backpropagation relies on the chain rules to compute the gradients of a layer’s weight using the delta’s of next layer.

- Pytorch / Tensorflow: auto-diff
Different Losses

- **Regression**:
  - Mean-square error (MSE)
  - (Smooth) L1-Loss

- **Classification**:
  - Negative Log-Likelihood
  - Cross-Entropy

- And many others defined for specific problems e.g. semi supervised embeddings, generative methods, etc.

- + add some regularization constraint on the parameters of the network
https://playground.tensorflow.org/
Convolutional Neural Networks
Convolutional Neural Networks

- Use local connections instead of fully-connected

Credit: Goodfellow et al 2016
Convolutional Neural Networks

- Share weights (parameters)

Convolution shares the same parameters across all spatial locations.

Traditional matrix multiplication does not share any parameters.
- Filters extend the full depth of the input volume
- Convolve the filter with the image: Slide over the image spatially, computing dot products.
- Each layer can train several convolution filters: channels
- Stack the activation map produced by each filter; repeat.

Credit Fei-Fei Li's course
Growing receptive field

Credit: Goodfellow et al 2016
Pooling

- Number of features extracted can be very large (increasing number of channels / filters)
- To reduce the size of the activation maps, one can:
  - Use strides
  - Downsample using Max or AveragePooling - applied on each channel independently

Pooled feature (max or mean operator)

Convolutional features

Credit Fei-Fei Li's course
Convolutional Neural Networks
“Inside” deep learning: Representation Learning
“Inside” deep learning: Representation Learning

Credit: Goodfellow et al 2016
A mostly complete chart of Neural Networks

- Backfed Input Cell
- Input Cell
- Noisy Input Cell
- Hidden Cell
- Probabilistic Hidden Cell
- Spiking Hidden Cell
- Output Cell
- Match Input Output Cell
- Recurrent Cell
- Memory Cell
- Different Memory Cell
- Kernel
- Convolutional Pool

- Perceptron (P)
- Feed Forward (FF)
- Radial Basis Network (RBF)
- Recurrent Neural Network (RNN)
- Long / Short Term Memory (LSTM)
- Gated Recurrent Unit (GRU)
- Auto Encoder (AE)
- Variational AE (VAE)
- Denoising AE (DAE)
- Sparse AE (SAE)
- Markov Chain (MC)
- Hopfield Network (HN)
- Boltzmann Machine (BM)
- Restricted BM (RBM)
- Deep Belief Network (DBN)

- Deep Convolutional Network (DCN)
- Deconvolutional Network (DN)
- Deep Convolutional Inverse Graphics Network (DCiGAN)
- Generative Adversarial Network (GAN)
- Liquid State Machine (LSM)
- Extreme Learning Machine (ELM)
- Echo State Network (ESN)

- Deep Residual Network (DRN)
- Kohonen Network (KN)
- Support Vector Machine (SVM)
- Neural Turing Machine (NTM)
Neural nets are composition of functions and extract high levels representations of the data. Good tool to approximate ‘functions’ on complex / high dim data. ... if you have the right data + give the model the right information.
  ○ Pre-processing data correctly + calibrating the problem is most of the work...
Can lack ‘interpretability’ at first glance / in vanilla mode...but some tricks and tools exist (e.g. saliency maps, heatmap backprop,...)
Practical: Pytorch / Tensorflow
Go beyond ‘just’ classification / regression:
  ○ unsupervised / weakly supervised x representation learning, disentanglement, generative models, transfer learning, etc.
Some Resources

- Pattern Recognition and Machine Learning by Bishop
- Machine Learning Andrew Ng’s class on coursera [https://www.coursera.org/learn/machine-learning](https://www.coursera.org/learn/machine-learning)
- PyTorch / Tensorflow - Keras tutorials