Intro to Deep Learning

AI Seminar

Overview

- Background
- Basics
  - Neural network representation and inference
- Optimization (ie, training)
  - Training Algorithms
  - Momentum
- Hands-on Examples and Demos

Background

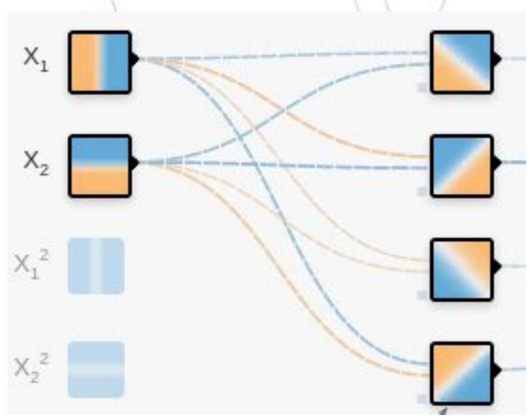
History

- Initially introduced in 1943 by McCulloch and Pitts
- Inspired by biological neurons
- Fell out of popularity for awhile in favor of approaches like SVMs
- Gained popularity due to:
    - larger datasets
    - better GPU performance
    - impressive empirical performance

Basics

Basics

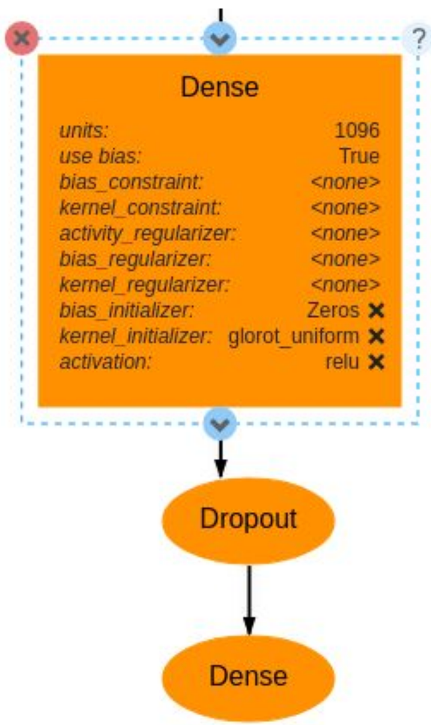
- Deep learning is the use of an ANN with more than two hidden layers. Most commonly used architectures are now deep.
- Each layer is essentially a differentiable operation which includes both linear and nonlinear operations.
- The most common layer is simply a matrix multiplication followed by a nonlinearity (activation function)



Visualization from https://playground.tensorflow.org

Representations

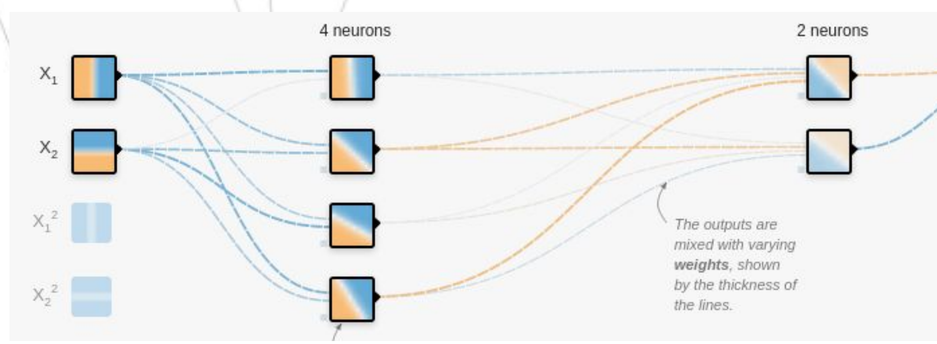
- These operations have multiple, equivalent representations: layers, functions, neurons, etc



$$l_n(l_{n-1}(\ldots l_2(l_1(X)))) = \hat{Y}$$

Or, equivalently

$$f_n(f_{n-1}(\ldots f_2(f_1(X)))) = \hat{Y}$$



Visualizations from DeepForge and https://playground.tensorflow.org

Regression or Classification?

- Neural networks can be used for both regression and classification.
- Essentially, this depends on two things:
  - network architecture
    - What is the output of the network?
      - Scalars are expected for binary classification or regression. Vectors can be used to represent class probabilities or some target vector.
      - Activation functions can ensure a probability distribution is produced (ie, softmax or sigmoid)
  - loss function (discussed more in the next section)

Optimization (ie, Training)

At a (very) high level...

- Since the neural network is entirely differentiable, we can simply compute the error for some training data.
- Then we can compute the derivative with respect to the network weights to determine how each weight contributed to the error and use this to improve the model.
- This enables us to perform gradient descent.

Training

- The standard approach to training a neural network entails:
  - Feeding data into the network to produce some output
    $$f_n(f_{n-1}(...f_2(f_1(X)))) = \hat{Y}$$
  - Determining the output's discrepancy using a loss function
    $$l(\hat{Y}, Y) = L$$
  - Backpropagating the discrepancy through the network (ie, $\frac{dL}{dW}$)
  - Using the discrepancy to inform the updates to the networks weights

Filling in some details...

- How exactly do we update the neural network weights?
  - Naively, we can decrease them by the gradient (scaled by the learning rate).
  - In practice, it is good to use momentum - like Adam - to help convergence and avoiding local minima. For a detailed comparison, check out https://ruder.io/optimizing-gradient-descent/
- What loss function should be used?
  - Depends on the problem. Mean squared error is often suitable for regression problems. Cross Entropy Loss is often suitable for classification.

Demos and Hands-On Examples

Demos and Resources

- Training a network in the browser: A Neural Network Playground
- Visualizing Learned Features: https://dev.deepforge.org/
- Training an image classifier with PyTorch
- Unreasonable Effectiveness of Recurrent Neural Networks
- A Recipe for Training Neural Networks
- Yes, you should understand backprop
- Automatic Differentiation in NetsBlox

Additional Topics

- Training Neural Networks with CMAES
- Dynamic vs Static Computational Graph
- Visualizing Learned Features