Machine Learning for Social Sciences
Part 5: Deep Learning

Felix Hagemeister

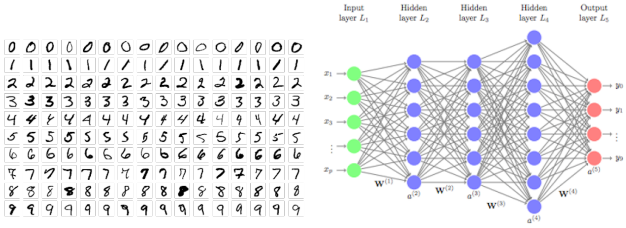
TUM School of Social Sciences and Technology
April 2022

Deep learning

Most ML models have the ability to only use one or two layers of data transformation to learn the output representation ("*shallow models*").

Deep Learning: Multi-layer approach to learn data representations, best used for big (n and p) data

- Deep neural networds (DNN): mapping features to targets through a process of simple data transformations and feedback signals
- Three kind of layers: an input layer, hidden layers, and an output layer



Neural Networks

Feedforward DNNs: ("multilayer perceptron") densely connected layers where inputs influence each successive layer which then influences the output layer

Four key components to build feedforward DNN:

1. Input data
2. Pre-defined network structure: architecture and activation
3. Feedback mechanism to help the network learn
4. Nodel training approach.

Network Architecture

Dense layers: fully connected successive *Capacity:* more layers and node mean more opportunities to learn new features

- Goal is to find simplest model with optimal performance
- Regression and binary classification have one output node: the predicted value or probability of success.
- Multinomial output will contain the same number of output nodes as classes.
- In *R*, the first layer needs the *input_shape* argument to equal the number of features in your data.

Activation

Each connection gets a weight and then that node adds all the incoming inputs multiplied by its corresponding connection weight plus an extra bias parameter w_0.

The summed total of these inputs become an input to an activation function: a mathematical function that determines whether or not there is enough informative input at a node to fire a signal to the next layer, e.g.

- Linear (identity): $f(x) = x$
- Rectified linear unit (ReLU): f(x) $= 0$ for $x > 0$, x for $x \geq 0$
- Sigmoid: $f(x) = \frac{1}{1+e^{-x}}$
- Softmax: $f(x) = \frac{e^{x_i}}{\sum_j e^{x_j}}$

Backpropagation

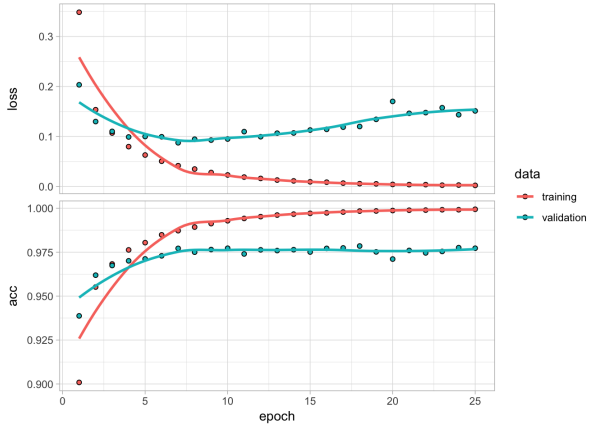
First run ("forward pass"): the DNN selects a batch of observations, randomly assigns weights across all the node connections, and predicts the output.

Backpropagation: assess own accuracy and automatically adjust weights

1. Need to establish an objective (loss) function to measure performance (e.g. MSE for regression)
2. *Mini-batch stochastic gradient descent:* after each run,
   - work backwards through layers,
   - compute gradient of loss w.r.t. weights,
   - adjust a little in opposite direction of gradient
3. Rinse and repeat until loss function is minimized.

Model Training

In this example, the DNN's performance is optimized at 5–10 epochs. Then, the DNN then proceeds to overfit.



Model Tuning

Next task is to find an optimal DNN by tuning different hyperparameters. The general steps are

1. Adjust number layers and nodes ("memorisation capacity")
2. Add batch normalisation (help with gradient propagation)
3. Add regularisation (help with overfitting: either add penalty or use dropout, i.e. randomly removing different nodes)
4. Adjust learning rate (size of incremental steps)

Could use *h2o.deeplearning()* function for grid search of optimal combination of parameters.

The tfruns package adds flexibility for tracking, visualising, and managing training runs.

Exercise

Let's implement an DNN in R ourselves and try to learn digit recognition using the MNIST database.

Here is the R code, based on this tutorial.