Pulling Threads:
Connectivity and Robustness in Neural Networks

BAMS project / course insight – Mathematical Modelling for Machine Learning

Abstract.

Imagine making two copies of a neural network that has learned a task. We remove the same number of connections from each copy, but choose different connections to remove. One copy might still work well, while the other might make many more mistakes. *Why should it matter which connections disappear, if the number removed is the same?*

A neural network makes predictions by passing information through layers of artificial neurons. The connections between them have numerical strengths, called *weights*, which are adjusted during training. Because a prediction depends on many weights working together, not every connection necessarily makes the same contribution. Some may be removed with little effect; losing others may change the output substantially. Understanding this difference is the central problem of the project.

We will train a small neural network on a simple classification task and then remove some of its learned connections, a procedure called *pruning*. To make the experiment concrete, the network will decide whether a collection of dots moves mainly left or right. We will generate these examples ourselves, using numbers to describe each dot's movement rather than working with video. The task is a way to test the network; the subject of the project is what happens *after* it has learned.

We will compare two ways of choosing connections to remove: selecting them at random, or removing those with the smallest absolute weights. At each step, we will measure how much of the network's accuracy remains. Finally, simple calculations and derivatives will help us investigate why changing a particular weight may affect a prediction more than changing another. The goal is to understand, in one carefully controlled model, **how connectivity supports a learned function and which kinds of changes it can withstand.**



Figure 1: An example of pruning. Both networks have the same neurons, but six of the sixteen connections are missing on the right. We then test whether the network's predictions change.

Prerequisites / Difficulty.

- **Mathematics:** vectors, matrices and matrix multiplication; basic derivatives; elementary probability and statistics.
- **Machine learning:** introductory knowledge of multilayer networks and training with gradient descent is useful. We will review the model before using it.
- **Programming:** basic Python and NumPy, with a library such as PyTorch for the experiments.

The core is accessible at an introductory–intermediate level; the derivative-based analysis is a more mathematical final step.

Starting materials.

Course foundation. MMML notes on multilayer neural networks, weight matrices and backpropagation. We will use these ideas to understand the network before changing its connections.

Hands-on guide. PyTorch, *Pruning Tutorial*. Shows how to remove weights by applying a mask; useful for implementing the experiments.

Mathematical extension. LeCun, Denker and Solla (1989), *Optimal Brain Damage*. An advanced example of studying how changes to weights affect a network's error. Selected ideas can be explored after the core work.

Background for the example task. Britten, Shadlen, Newsome and Movshon (1992), *The Analysis of Visual Motion*. A research example of decisions based on moving-dot displays; it motivates the task but is not required to carry out our simulations.

Possible directions.

The first four stages form one complete project. They move from a working network to a comparison of missing connections, and then to a mathematical explanation. The final stage is optional.

1. Train a small network and establish a baseline. We first need a network that performs a task reasonably well: otherwise, losing connections would tell us little. We will generate examples in which some dots move left or right and others move in random directions. Each example is represented by a list of horizontal and vertical displacements, so the model receives numbers rather than pictures or videos. We know the correct answer for each example because we generated the motion ourselves.

We will train a small feedforward network, with one hidden layer, to choose between the two directions. Its final output is a *score*: a positive score means "right" and a negative score means "left". One such model is

$$s_{\theta}(\mathbf{x}) = \mathbf{v}^{\top}\sigma(W\mathbf{x} + \mathbf{b}) + a, \qquad \widehat{y} = \operatorname{sign}\big(s_{\theta}(\mathbf{x})\big). \tag{1}$$

Here $\mathbf{x}$ is the input, $W, \mathbf{v}, \mathbf{b}, a$ are learned parameters, and σ is an activation function applied to each hidden neuron. We will check the model on new examples not used for training. Its accuracy on these examples is our *baseline*—the result we want to preserve as connections are removed.

2. Remove connections in two different ways. After training, we will freeze all the learned parameters. We then remove weights by setting them to zero, leaving the neurons in place. We can describe this using a *mask* of zeros and ones:

$$W_{\text{pruned}} = M \odot W, \qquad M_{ij} \in \{0, 1\}. \tag{2}$$

The symbol $\odot$ means multiplying entries one by one. A zero in M removes that connection; a one keeps it. The same idea applies to the weights of each layer.

We will compare two selection rules. In *random pruning*, we choose connections at random. In *magnitude pruning*, we remove those whose weights have the smallest absolute values. For a fair comparison, we will remove the same percentage of weights in each layer, keep biases unchanged, and test both methods at several percentages. Because a random choice can be unusually lucky or unlucky, we will repeat it several times.

3. Measure what the network can still do. After each round of pruning, we will test the network on the *same* examples as before. Let $A(0)$ be its original test accuracy and $A(q)$ its accuracy when a fraction q of the weights has been removed. Their difference,

$$\Delta A(q) = A(0) - A(q), \tag{3}$$

shows how much the accuracy has changed. A positive value means performance has fallen; a negative value means it has improved. We will plot accuracy against the percentage of weights removed for both pruning rules.

These curves let us ask two questions: *how many* connections can disappear before performance changes substantially, and *which* connections tend to matter? If time permits, we can also remove weights from one layer at a time to see whether some layers are more sensitive than others.

4. Use mathematics to understand the differences. Measuring performance is only the beginning. We also want to understand why removing one connection may matter more than removing another. We will start with a single neuron whose output score is

$$s(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b.$$

If we remove only weight w_j, the score changes by exactly $-w_j x_j$. This is a useful first observation: the effect of a weight depends on both its size and the input it multiplies. A large weight need not always be influential for every example.

In a multilayer network, connections interact and the calculation becomes harder. Derivatives provide an approximation for small changes to the parameters:

$$s_{\theta+\Delta\theta}(\mathbf{x}) - s_\theta(\mathbf{x}) \approx \nabla_\theta s_\theta(\mathbf{x})^\top \Delta\theta. \tag{4}$$

We will explain what this approximation means, evaluate it for a few individual inputs, and compare it with the actual change in the network's score. It is a *local* approximation, not a formula for test accuracy: removing many weights at once may change the network too much for it to remain accurate.

5. Are difficult decisions more sensitive? (If time allows.) Once we have answered the main questions, we can make the dot-motion task harder without changing the number of input dots: we replace some of the consistent movements with random ones. At each difficulty level, we would compare the pruned network with the unpruned one. This lets us ask whether missing connections are especially harmful when the input is ambiguous. It is an optional extension, not a second main objective.

Why robustness?

A network can learn to do something without relying equally on every connection. Studying what happens when weights disappear helps us understand how a learned computation is distributed and how sensitive its predictions are to changes in its parameters. It also connects to *model compression*: removing weights that contribute little might reduce the amount of information needed to store a model, although saving computation in practice can require additional engineering. Our small experiments will not produce a universal rule for every neural network. They will show how to ask and test a precise question about one model.

Expected outcome.

Participants will produce a small trained network, code that applies the two pruning rules, and plots comparing performance as more connections are removed. They will derive the masking equation and work through the exact one-neuron calculation before using the derivative approximation. A short report will explain the results, discuss whether the size of a weight was a reliable guide to its importance in the chosen network, and identify the limits of the conclusions. No particular method is assumed to win in advance.

Contacts.

For questions, ideas or further discussion, contact Virginia Benedetti:

virginia.benedetti@studbocconi.it

Phone / WhatsApp: +39 344 164 6581.