Can the Hidden Layer of an MLP act as a Kernel?

In Deep Learning by Goodfellow et al. [1], the idea of a feedforward network is introduced via learning the XOR function. The authors argue for an interpretation of a feedforward network as learning a representation of the data that allows the final linear layer to successfully predict. They consider a training set containing the four possible inputs to the XOR function – (0, 0), (0, 1), (1, 0), and (1, 1). The feedforward network they use to solve the XOR function has an input layer with two neurons, a single hidden layer with two neurons, and an output layer with 1 neuron. The RELU activation function is used in the hidden layer but not in the output layer. This means that the hidden-to-output transformation is a linear function, of the form w\_1\*h\_1 + w\_2\*h\_2 + b\_2.

The key observation is that the initial state of the inputs makes it impossible for the linear layer to learn the XOR function by itself. Let w\_1 and w\_2 denote the weights of the output layer, and suppose there is no hidden layer. Let (x\_1, x\_2) denote the input to the model. The input (x\_1 = 0, x\_2 = 0) produces the output 0, while the input (x\_1 = 0, x\_2 = 1) produces the output 1. Hence, the network’s output needs to increase while x\_2 increases, i.e., w\_2 should be positive. On the other hand, the input (x\_1 = 1, x\_2 = 0) produces the output 1 while the input (x\_1 = 1, x\_2 = 1) produces the output 0, so in this case the model’s output needs to decrease as x\_2 increases, i.e., w\_2 should be negative. But of course w\_2 cannot be positive and negative at the same time.

Hence, when we reintroduce the hidden layer, the goal of the hidden layer should be to transform the initial XOR inputs x\_1 and x\_2 into a hidden state h\_1 and h\_2 that the linear hidden-to-output transformation can successfully deal with. This means the function to be modeled should be able to be described as “increasing or decreasing in h\_1 and increasing or decreasing in h\_2.”

The goal of this note is to visualize whether this process actually occurs over time. We train the network with the MSE loss function and gradient descent.

The network is initialized with uniformly random weights and biases:

**Parameter Initialization of the Network:**

1. **First Hidden Layer (2 neurons)**:
   * Neuron 1:
     + Weights: 0.4353819571658071, 1.4491887901903806
     + Bias: -0.47002446404320053
   * Neuron 2:
     + Weights: 0.6907621337017469, -0.2429822886657198
     + Bias: 0.3183225318669727
2. **Output Layer (1 neuron)**:
   * Weights: 0.1238082116676491, -0.04459148399140078
   * Bias: 0.14278101806704635

A graph with numbers and dots

Description automatically generatedThe initial hidden layer feature space is shown below:

Figure - Epoch 0 ; loss=1.4545922727093285

The hidden layer feature space for intermediate epochs in training are now shown below. Please ignore the epoch numbers in the graph titles – they are wrong, the caption epoch number is correct.

A graph with numbers and dots

Description automatically generated

Figure - Epoch 15

A graph with numbers and symbols

Description automatically generated

Figure 3 - Epoch 60

A graph with numbers and symbols

Description automatically generated

Figure 4 - Epoch 90

At the 105th epoch of training, progress plateaus and we have the following hidden layer feature space:

A graph with numbers and symbols

Description automatically generated

Figure - Epoch 105 ; loss:0.19461413640473335

The hidden state may be described as “increasing in h2, decreasing in h1.” This is learnable by a linear function (w1 should be negative, w2 positive).