Lecture 5

Nonlinear neural networks

We start this lecture with a remark from lecture 4.

Why the more general three-layer network still doesn't work as expected?

The answer is simple: mathematical trivial result

. For any two consecutive weighted sums of the input, there exists a single weighted sum with identical behavior. So anything that the three-layer network can do, the two-layer network can also do.

It is clear, that the main aim in training any neural networks is to find correlations between the input and output. Only in this case we can expect that this neural network can predict results for ~~once~~ input data.

So we want that middle layer to sometimes correlate with an input layer, and sometimes not correlate.

The middle layer will have the opportunity not just always be correlated to some input.

It can be correlated to one input only when it wants to be and other times not be correlated at all (conditional correlation).

This approach is based on nonlinear neurons. Let us take a first, simple but very important example.

If the node's value dropped below 0, normaly for linear neurons it would still have the same correlation to the input. But now we turn of the node setting it to 0, when it would be negative.

We have zero correlation to any inputs whenever they are negative.

Thus, for example, we can construct a nonlinear NN for which some node on level 1 (middle layer) is perfectly correlated to input node 1 (on layer level 0) only when another input node 2 is turned off.

We see that by turning off any middle node whenever it would be negative we allow the NN to sometimes to correlation from various inputs. This result is impossible for two-layer NN.

There are many kinds of nonlinearities. We started from one simple but very popular example called relu activation f (rectified Linear Unit)

```
def relu(x):
    return (x > 0) * x
```

This function sets all negative numbers to 0.

Also we will need the derivative of this function (in order to calculate the gradient)

```
def relu2deriv(input):
    return input > 0
```

It returns 1 for input > 0 and returns 0 otherwise.

Now we are ready to modify the three-level NN proposed in Lecture 3.

This linear deep NN not converged during training stage. Let us check of application of relu type nonlinear neurons can change this situation.

```
N=3, M=4
hidden_size = 4    # P=4
weights_01 = ...   # random [N×P]
weights_12 = ...   # random [P×1]
for iteration in range(60):
    layer_2_error = 0
    for i in range(len(inputs)):
        layer_0 = inputs[i:i+1]          #M
        layer_1 = relu(np.dot(layer_0,
                              weights_01))
        layer_2 = np.dot(layer_1,
                         weights_12)
        layer_2_error += np.sum(
            (layer_2 - goals[i:i+1])**2)
    layer_2_delta = layer_2 - goals[i:i+1]
```

```
# Back propagation
    layer1_delta = layer2_delta.
        dot(weights_12.T)*relu2deriv
                (layer_1)
    weights_12 += alpha * layer1.T.dot(
                    layer_2_delta)
    weights_01 += alpha * layer0.T.dot(
                    layer_1_delta)

    if (iteration % 10 == 9):
        print ("Error:" +str(layer2_error))
```

Let us make experiments - the following results are obtained

```
Error: 0.634231
Error: 0.358384
Error: 0.083018
Error: 0.006467
Error: 0.000329
Error: 1.505e-05.
```

After inclusion of nonlinear activation function the three-level NN ~~o~~ have new possibilities ~~to~~ predict results for ~~non~~ data sets not used during training. The iterative process is converging fastly

The relu2deriv function defines the slope (derivative) of the relu activation function and it regularizes update of weights_01 Remember, that weights_01 are used to calculate a prediction on level 1 and relu function is used in this update step.

How we compute the deltas for layer 1.

- first we take the standar step for three-layer NN : multiplay the output (Layer 2) deltas by the weights 12.
- second, we should take into account that we applied relu activation function in computation of Layer 1 nodes. and set value 0 if the predicted value was negative. Thus now we apply relu2deriv to regularize layer 1 delta values, by seting the delta of that node to 0.