**Neural Network Architecture**

Borrowing the idea from ResNet[1, 2], I use two mechanisms here, the skip connection and the pre-activation, to construct my own neural network (See Figure 1). It is supposed that we are training a multi-class classifier with the cross-entropy loss. It can be seen from Figure 1 that all the intermediate layers, except activation functions, are fully connected layers.

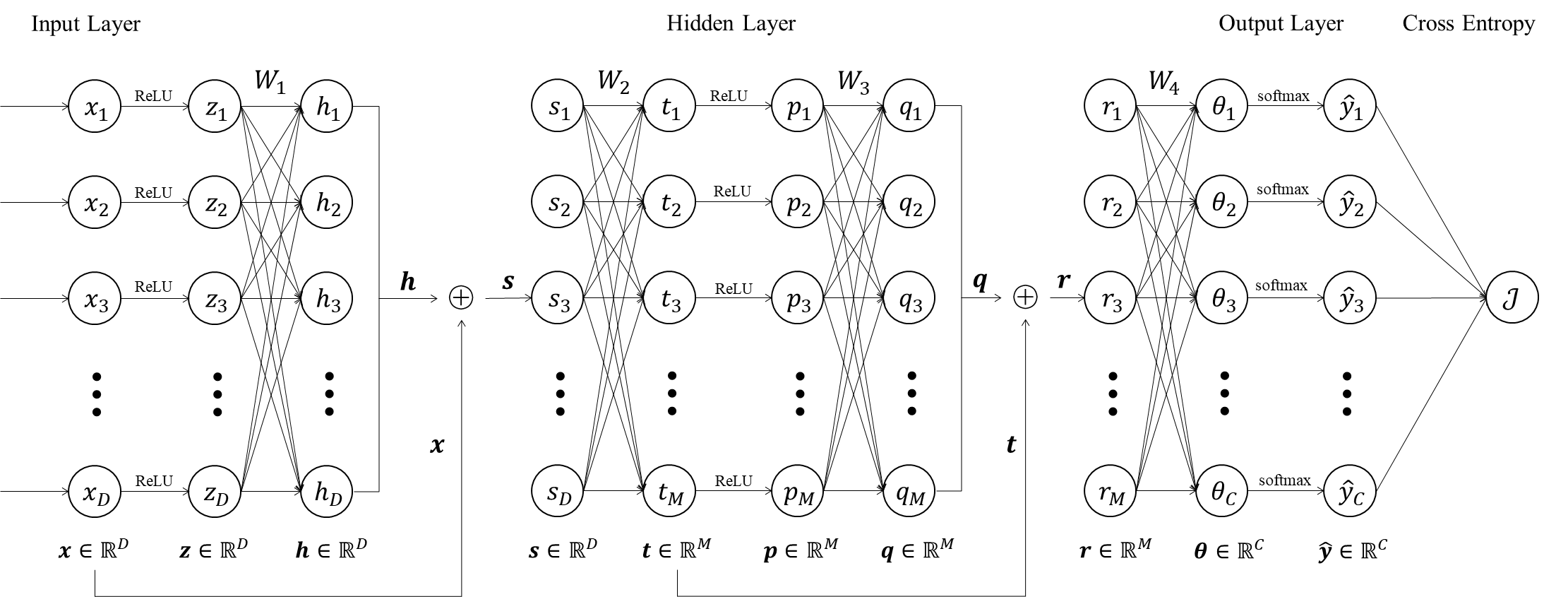


Figure 1: The architecture of this self-designed neural network.

A forward pass of a single input sample is as follows:

where , , , are weight matrices. is the true label vector associated with , that is, a one-hot vector indicting which class belongs to. The softmax activation of the -th output unit is

.

The cross-entropy loss for this single input sample is

.

**Gradients Derivation**

First, let us start with calculating the common parts shared by all four target gradients. we can see that

where the division in is an element-wise operator.

For the gradient of softmax function, we look at a single entry first:

By chain rule, we can compute

Recall that isa one-hot vector, that is, , we can get

and

Now we examine the gradient of ReLU function. Recall that , by extending that its derivative at equals , we get

where is an indicator function. Furthermore, we can extend ReLU function to a vector-valued function by applying ReLU to each entry of an input vector and calculate its gradient as

It is easy to see that and contain all the network parameters. So we just need to derive their gradients to update the whole neural network.

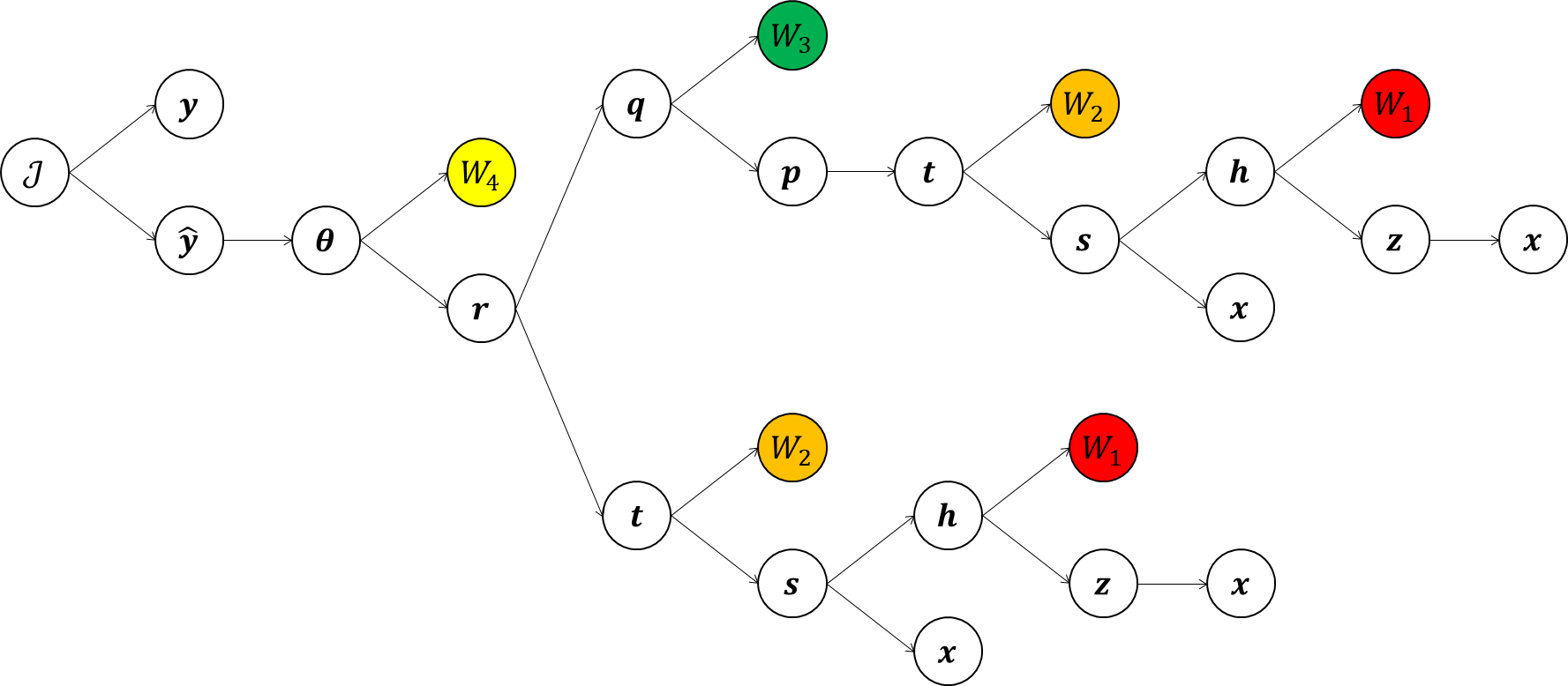


Figure 2: The computation graph of this self-designed neural network.

However, due to the complexity of this neural network, it is hard to capture directly how the gradients flow just by observing its architecture figure. So I show the computation graph of it to make the process of gradients derivation more clearly (See Figure 2).

It can be seen obviously from Figure 2 that there is only one path to approach and , but two to approach and . With the help of this computation graph, we can write down the gradients for all the network parameters by using the chain rule as

Subscript 1 means the gradient flows through the upper branch of the computation graph and subscript 2 the lower. We use subscripts here to avoid confusion, that is, .

It is noticed that , , and are used multiple times, so it is convenient to calculate them firstly:

where .

Now we can rewrite the target gradients as

Then we continue to calculate the rest parts. For the gradient , let us start by analyzing the gradient . By definition, this gradient is the collection of the partial derivatives:

We can write in this form:

where is the -th row of . So we can explicitly write out as

that is, a matrix with -th row being and other rows being . Combining them together, we can get

Similarly, we can get that

At the end, we show the final results as follows:

**Training Equations**

In real practice, we never use just a single input to update the neural network. Instead, we train a neural network with a training set in most cases. To make this assignment more practical, we suppose that we use the training set to train the neural network.

As a result, the final loss function needs to be modified as

where is the cross-entropy loss w.r.t a single input , whose gradients are shown in the above section. And the gradients of it are

For the training process, we start by randomly generating ~. At step , we use the following training equations to update the parameters:

where is the learning rate.

**Reference**

[1] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[2] He, Kaiming, et al. "Identity mappings in deep residual networks." European conference on computer vision. Springer, Cham, 2016.