1B) $p = \frac{1}{2}$

آموزش

$O = [1.5, 6, 3, -0.75, -4.5]$

اعمال ماسک $\longrightarrow O = [1.5, 0, 3, 0, -4.5]$

تقسیم بر $\frac{1}{2}$ $\longrightarrow O = [3, 0, 6, 0, -9]$

relu $\longrightarrow [3, 0, 6, 0, 0]$

activation $\longrightarrow 9$

آزمون

$$O = [1.5, 6, 3, -0.75, -4.5]$$

↳ relu $\Rightarrow O = [1.5, 6, 3, 0, 0]$

↳ activation $\rightarrow 10.5$

اگر هنگام آزمون بر $\frac{1}{2}$ تقسیم کردیم

در زمان آزمون کاری انجام نمی‌دهیم

3A

$$H_3 = \frac{1}{10} \times 0.35 + \frac{8}{10} \times 0.9$$

$$H_3 = 0.755$$

$$H_4 = \frac{4}{10} \times 0.35 + \frac{6}{10} \times 0.9$$

$$H_4 = 0.68$$

$$\sigma(H_3) \simeq 0.68 = y_3$$

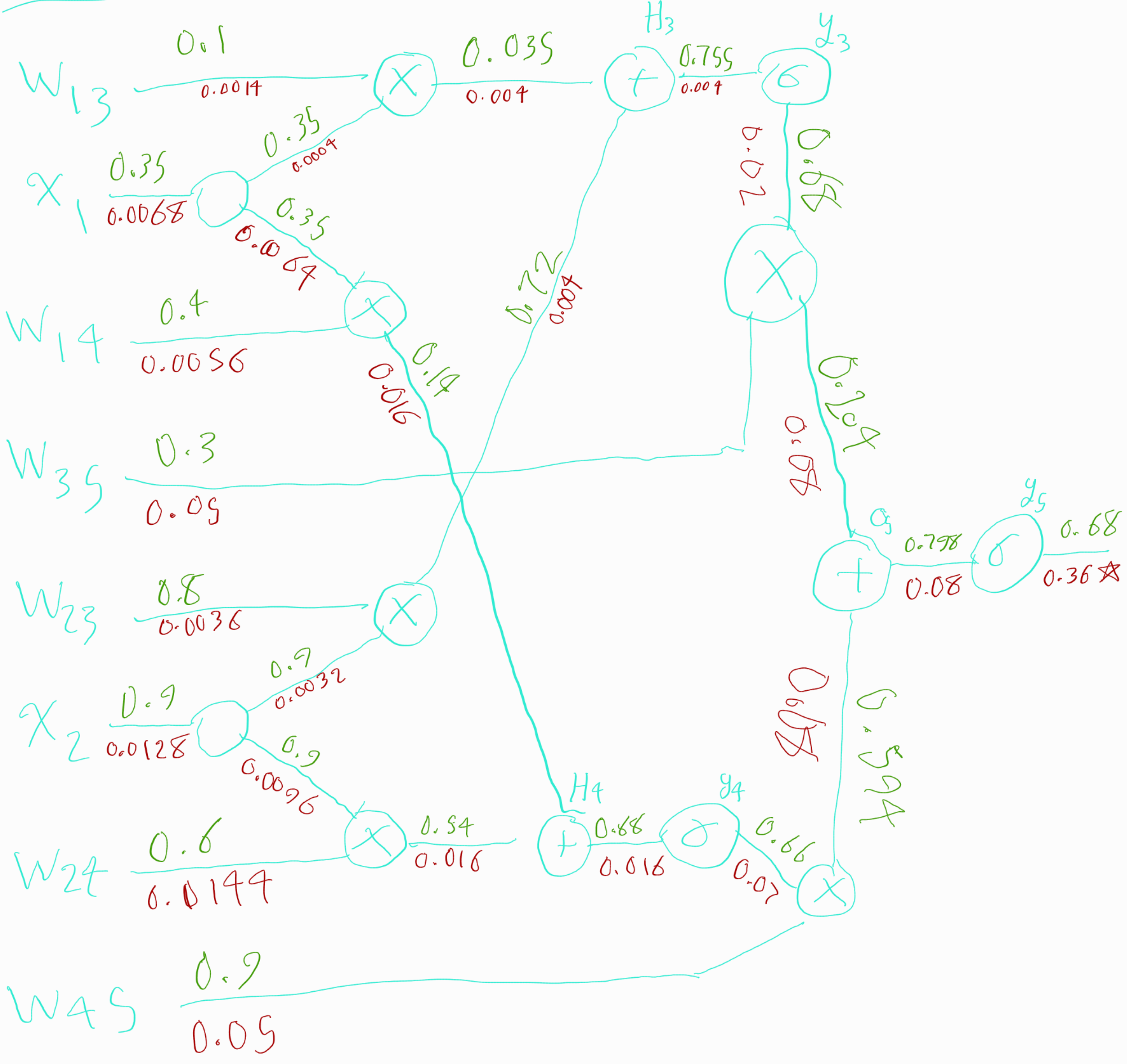
$$\sigma(H_4) \simeq 0.66 = y_4$$

$$O_5 = \frac{3}{10} \times 0.68 + \frac{9}{10} \times 0.66$$

$$O_5 = 0.798$$

$$y_5 = \sigma(O_5) \simeq 0.68$$

3B



$$l = (y - y_5)^2 \longrightarrow \frac{\partial l}{\partial y_5} = 2 \times -1 \times (y - y_5)$$

$$\frac{\partial l}{\partial y_5} = 2y_5 - 2y = 2(y_5 - y) = 2 \times (0.68 - 0.5) = 0.36^{\star}$$

$$W \leftarrow W - \nabla W$$

$$W_{13} \quad \left.\begin{matrix}0.1\\ 0.0014\end{matrix}\right\} 0.0986$$

$$W_{14} \quad \left.\begin{matrix}0.4\\ 0.0056\end{matrix}\right\} 0.3944$$

$$W_{35} \quad \left.\begin{matrix}0.3\\ 0.05\end{matrix}\right\} 0.25$$

$$W_{23} \quad \left.\begin{matrix}0.8\\ 0.0036\end{matrix}\right\} 0.7964$$

$$W_{24} \quad \left.\begin{matrix}0.6\\ 0.0144\end{matrix}\right\} 0.5856$$

$$W_{45} \quad \left.\begin{matrix}0.9\\ 0.05\end{matrix}\right\} 0.85$$

1A
We use activation functions in MLP to introduce nonlinearity into the output of each neuron. Without activation functions, the output of each neuron would be a linear combination of the input, which would result in the entire network being linear. This would make it difficult for the network to learn complex patterns and relationships in the data.

Activation functions are applied to the output of each neuron, transforming it into a non-linear form that allows the network to learn more complex features and patterns. There are several activation functions that can be used in MLPs, such as sigmoid, ReLU, and tanh, each with its own advantages and disadvantages. The choice of activation function depends on the specific problem being solved and the characteristics of the data being used.

1C
If we have a Multilayer Perceptron (MLP) with a single hidden layer, the size of the hidden layer can have a significant impact on the performance of the network.

If the hidden layer size is 2 neurons, this would result in a very simple network that would be limited in its ability to learn complex patterns and relationships in the data. This could lead to underfitting, where the network is not able to capture all of the relevant features in the data and thus, cannot make accurate predictions.

If the hidden layer size is 256 neurons, this would result in a more complex network that could potentially perform better on more challenging tasks. However, a very large hidden layer size can also lead to overfitting, where the network is too complex and learns to fit the training data too closely, resulting in poor generalization to new, unseen data.

If the hidden layer size is 1 million neurons, this would result in an extremely large and computationally intensive network. In practice, it is very rare to use such a large hidden layer size, as it is unlikely to provide significant benefits compared to a more modestly sized hidden layer. Additionally, training such a large network can be very difficult and may require specialized hardware.

Overall, the choice of hidden layer size depends on the specific problem being solved and the characteristics of the data being used. It is often a matter of trial and error to find the optimal size that balances the trade-off between model complexity and generalization performance.

1D
The vanishing gradient problem in deep neural networks occurs when gradients become very small as they propagate through the network during training. This can make it difficult for the network to learn and can result in poor performance. The main causes of the vanishing gradient problem are:

Activation functions with small derivatives: When activation functions with small derivatives, such as the sigmoid function, are used in deep neural networks, the gradients can become very small as they propagate through the network, resulting in the vanishing gradient problem.

Deep networks: As the number of layers in a deep neural network increases, the gradients can become smaller and smaller as they propagate through the network, which can also lead to the vanishing gradient problem.

Here are some techniques that can be used to solve the vanishing gradient problem:

ReLU activation function: Rectified Linear Unit (ReLU) is an activation function that has a derivative that is always 1 for positive inputs. ReLU can help mitigate the vanishing gradient problem because it does not cause gradients to shrink as they propagate through the network.

Weight initialization: Properly initializing the weights in the network can help prevent the vanishing gradient problem. For example, using Xavier or He initialization can help ensure that the weights are not too small or too large, which can contribute to the vanishing gradient problem.

Gradient clipping: Gradient clipping involves setting a threshold on the gradients during training to prevent them from becoming too large or too small. This can help prevent the vanishing gradient problem and also prevent the exploding gradient problem.

Batch normalization: Batch normalization is a technique that can be used to normalize the inputs to each layer in the network. This can help prevent the vanishing gradient problem by ensuring that the inputs to each layer have a reasonable range of values.

Overall, the choice of technique depends on the specific problem being solved and the characteristics of the data being used. It is often a matter of trial and error to find the optimal solution that balances the trade-off between model complexity and performance.

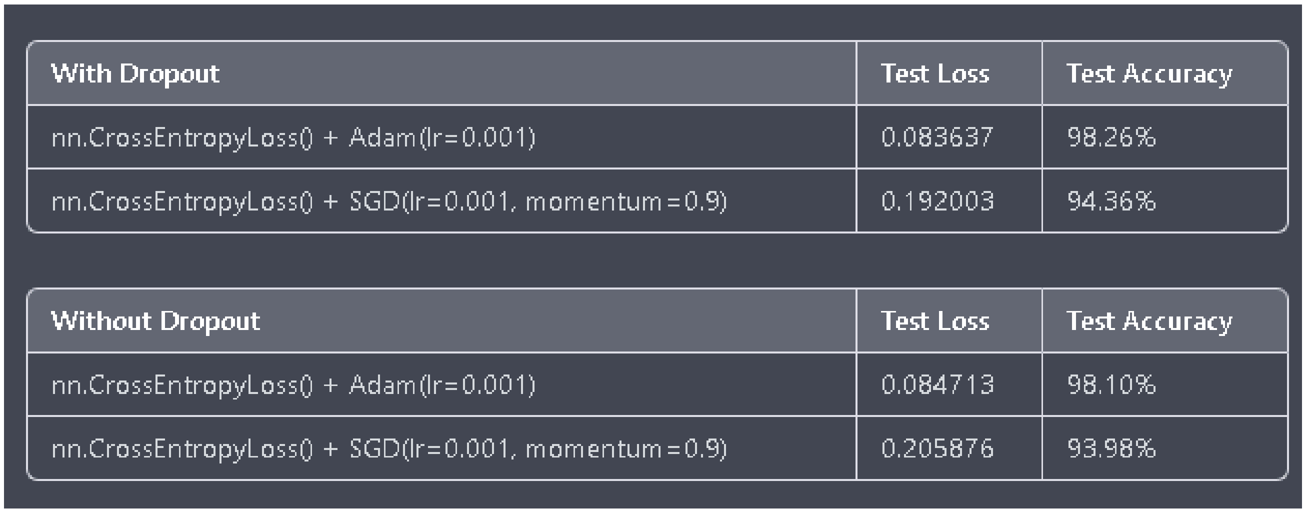
2A
false
Using regularization can help prevent overfitting and improve the generalization performance of the model. However, using too much regularization can cause the model to underfit and result in poor performance. Therefore, finding the right amount of regularization is important for achieving the best possible performance.

2B
false
Overfitting occurs when a model becomes too complex and fits the training data too closely, resulting in poor performance on new data. Adding more features to the input dataset can increase the complexity of the model, which can actually exacerbate the overfitting problem if not done carefully. While adding informative features can improve the performance of the model, adding irrelevant or redundant features can actually harm performance. Therefore, it is important to carefully select the features that are most relevant for the problem being solved and to use regularization techniques to prevent overfitting.

2C
Increasing the strength of regularization can help prevent overfitting by further penalizing complex models and encouraging simpler models. Therefore, increasing regularization should generally improve the generalization performance of the model and prevent overfitting.

However, it is possible that using too much regularization can cause the model to underfit, which occurs when the model is not complex enough to capture the underlying patterns in the data. This can result in poor performance on both the training data and new data. Therefore, finding the right amount of regularization is important for achieving the best possible performance.

4
Using drop out incrases the performace.



With Dropout	Test Loss	Test Accuracy
nn.CrossEntropyLoss() + Adam(lr=0.001)	0.083637	98.26%
nn.CrossEntropyLoss() + SGD(lr=0.001, momentum=0.9)	0.192003	94.36%

Without Dropout	Test Loss	Test Accuracy
nn.CrossEntropyLoss() + Adam(lr=0.001)	0.084713	98.10%
nn.CrossEntropyLoss() + SGD(lr=0.001, momentum=0.9)	0.205876	93.98%

Adam is better than SGD for this task, because it created a better model with higher perforamce