Assignment 5.1

1.
In the squared_loss function, the reshape function serves to ensure that both y and y hat have the same shape before performing the subtraction and square operations. This is crucial for element-wise operations to work correctly. If y has a shape of [n,1] (a column vector with n rows) and y hat has a shape of [n] (a 1D array with n elements). In this case, the subtraction operation would result in a broadcasting error. Because the shapes are not compatible for element-wise subtraction based on the broadcasting rules of tensors. By using reshape, we make sure that both tensors have exactly the same shape. This ensures that element-wise subtraction (and subsequently, the squaring operation) happens without any hitches. If you remove the reshape operation, the code may throw an error or produce unexpected results depending on the shapes of y and y hat. For example, if y is [n,1] and y hat is [n], then y - y hat would fail because the shapes are not compatible for element-wise operations according to the broadcasting rules. So, the reshape operation is more of a safeguard to ensure that the loss function operates correctly regardless of the shape of the input tensors.

2.
I tested two weight initialization methods for a simple linear regression model:

Random Initialization: Weights were initialized using a normal distribution with a mean of 0 and a standard deviation of 0.01.

Zero Initialization: Weights were initialized as zeros.

Metrics Observed
Convergence Speed: Measured by observing the loss value over epochs.
Final Parameters: The weights w and bias b after training.

Results
Convergence Speed

Epoch	Random Initialization Loss	Zero Initialization Loss
1	0.030702	0.030701
2	0.000108	0.000102
3	0.000049	0.000049
4	0.000049	0.000049
5	0.000049	0.000049
6	0.000049	0.000049
7	0.000049	0.000049
8	0.000049	0.000049
9	0.000049	0.000049
10	0.000049	0.000049

Both initialization methods showed a declining trend in loss, indicating convergence. The Zero Initialization approach reached a stable loss by epoch 3, maintaining it throughout the remaining epochs.

Final Parameters

Parameter	Random Initialization	Zero Initialization	True Value
$w1$	2.0001	1.9997	2
$w2$	-3.3997	-3.3998	-3.4
b	4.2007	4.2008	4.2

The final parameters for both initialization methods were extremely close to the true values, suggesting that the model was effectively able to learn the underlying data distribution.

For the linear regression model under study, both random and zero weight initialization methods yielded highly accurate and effective models. This is in line with theoretical expectations, given the nature of the loss function in linear regression. Both initialization methods were effective in training the model to a state very close to the true values. The loss values indicate that the model with zero initialization converges quickly and maintains stability. The final weights and biases are practically indistinguishable between the two initialization methods and are very close to the true values.

3.

Learning Rate	Epoch	Loss
LR: 0.01	Epoch 1	Loss: 2.1372909545898438
LR: 0.01	Epoch 2	Loss: 0.266950786113739
LR: 0.01	Epoch 3	Loss: 0.033471863716840744
LR: 0.01	Epoch 4	Loss: 0.0042547001503407955
LR: 0.01	Epoch 5	Loss: 0.0005827781278640032
LR: 0.01	Epoch 6	Loss: 0.00011653260298771784
LR: 0.01	Epoch 7	Loss: 5.777967453468591e-05
LR: 0.01	Epoch 8	Loss: 4.99391790071968e-05
LR: 0.01	Epoch 9	Loss: 4.884734153165482e-05
LR: 0.01	Epoch 10	Loss: 4.8712987336330116e-05
LR: 0.03	Epoch 1	Loss: 0.030827268958091736

LR: 0.03	Epoch 2	Loss: 0.00010674486838979647
LR: 0.03	Epoch 3	Loss: 4.880351116298698e-05
LR: 0.03	Epoch 4	Loss: 4.8700352635933086e-05
LR: 0.03	Epoch 5	Loss: 4.8773017624625936e-05
LR: 0.03	Epoch 6	Loss: 4.8756894102552906e-05
LR: 0.03	Epoch 7	Loss: 4.8847810830920935e-05
LR: 0.03	Epoch 8	Loss: 4.873523721471429e-05
LR: 0.03	Epoch 9	Loss: 4.8765126848593354e-05
LR: 0.03	Epoch 10	Loss: 4.877367973676883e-05
LR: 0.1	Epoch 1	Loss: 4.878137769992463e-05
LR: 0.1	Epoch 2	Loss: 4.930340583086945e-05
LR: 0.1	Epoch 3	Loss: 4.879319021711126e-05
LR: 0.1	Epoch 4	Loss: 4.9371588829671964e-05
LR: 0.1	Epoch 5	Loss: 4.952047311235219e-05
LR: 0.1	Epoch 6	Loss: 4.889177580480464e-05
LR: 0.1	Epoch 7	Loss: 4.893436926067807e-05
LR: 0.1	Epoch 8	Loss: 4.9477050197310746e-05
LR: 0.1	Epoch 9	Loss: 4.906012327410281e-05
LR: 0.1	Epoch 10	Loss: 4.8772530135465786e-05

Analysis and Interpretation
Learning Rate: 0.01 The loss starts from 2.137 and takes 10 epochs to reach 4.87×10−5. The descent is gradual, indicating a conservative update in each iteration.

Learning Rate: 0.03 The loss starts from 0.031 and rapidly falls to 4.88×10−5 within just 2 epochs. This is faster compared to a learning rate of 0.01, reflecting a more aggressive update strategy.

Learning Rate: 0.1 The loss starts already low at 4.88×10−5 and hovers around that level throughout all 10 epochs. This indicates that the model converged almost immediately, but it's worth noting that the loss doesn't improve significantly after the first epoch.

Takeaways
A lower learning rate (η=0.01) will make the model learn slowly, which might be good for highly sensitive optimization landscapes but could be computationally expensive.
A moderate learning rate (η=0.03) appears to be a good compromise between speed of convergence and computational efficiency in this case.
A higher learning rate (η=0.1) leads to rapid convergence, but too high a learning rate might overshoot the optimal solution in more complex landscapes.

4.
In the data_iter function, the slicing mechanism indices[i:min(i + batch_size, num_examples)] ensures that the last batch contains the remaining examples, which may be fewer than the specified batch_size. Specifically, the last batch will contain num_examples % batch_size examples. The computational optimization for matrix operations often expects uniform tensor sizes. The last batch's smaller size could slightly derail these optimizations, although the impact is typically marginal. The model parameters' update will be based on a smaller batch. This could slightly skew the gradient and, consequently, the model update direction for that particular iteration. The data_iter function is designed to handle the "leftover" examples by creating a smaller final batch, ensuring that no example is left out during the training process. The effect on most models is often negligible, but it's a crucial detail to be aware of, especially when debugging or fine-tuning more complex models.