CHAPTER 4 SOLUTION

---

1. What Linear Regression training algorithm can you use if you have a training set with millions of features?

If there are millions of features, using the **Normal Equation** is inefficient and consumes too much memory. Instead, we use **Stochastic Gradient Descent (SGD)** or **Mini-batch Gradient Descent**, which work well even with large feature sets because they update the model step-by-step and don't require inverting huge matrices.

---

2. Suppose the features in your training set have very different scales. What algorithms might suffer from this, and how? What can you do about it?

Algorithms like **Gradient Descent**, **SVMs**, and **KNN** can suffer when features have different scales. This is because features with larger ranges dominate the learning process or distance calculations. To fix this, we **standardize or normalize** the features so that all of them have the same scale (usually mean 0 and standard deviation 1).

---

3. Can Gradient Descent get stuck in a local minimum when training a Logistic Regression model?

No, because the **cost function for Logistic Regression is convex**. That means it has only one global minimum. So, Gradient Descent will never get stuck in a local minimum — it always converges to the global minimum (if learning rate and other parameters are set properly).

---

4. Do all Gradient Descent algorithms lead to the same model provided you let them run long enough?

Yes — if the learning rate is well-tuned and you let them run long enough, **all Gradient Descent variants (Batch, Stochastic, Mini-batch)** will eventually reach the **same or similar optimal model**. The paths and speed might differ, but the destination is the same (especially for convex problems).

---

5. Suppose you use Batch Gradient Descent and you plot the validation error at every epoch. If you notice that the validation error consistently goes up, what is likely going on? How can you fix this?

If validation error keeps increasing, it usually means the model is **overfitting** or **learning too aggressively**. A common fix is to use **early stopping**, which stops training when the validation error starts rising. You can also **reduce the learning rate**, add **regularization**, or use **Mini-batch GD** for better generalization.

---

6. Is it a good idea to stop Mini-batch Gradient Descent immediately when the validation error goes up?

No — Mini-batch GD has **fluctuations** due to randomness. A single spike doesn't mean the model is overfitting. Instead, use **early stopping with patience**, where training only stops if the validation error fails to improve for several epochs in a row.

---

7. Which Gradient Descent algorithm will reach the vicinity of the optimal solution the fastest? Which will actually converge? How can you make the others converge as well?

- **Stochastic Gradient Descent (SGD)** reaches the solution area fastest due to frequent updates.
- **Batch Gradient Descent** converges more smoothly and reliably.
- To make SGD or Mini-batch converge, we can **gradually reduce the learning rate** (called a **learning schedule**) over time.

---

8. Suppose you are using Polynomial Regression and see a large gap between training and validation error. What is happening? What are three ways to solve this?

This is a sign of **overfitting**. The model fits training data too well but fails on unseen data.

Three ways to fix this:

1. **Use regularization** (like Ridge or Lasso).
2. **Reduce the model complexity** (lower polynomial degree).
3. **Add more training data** or use **data augmentation**.

---

9. Suppose you are using Ridge Regression and the training and validation errors are both high and very close. What's the issue and what should you change?

This means the model is **underfitting** — it has **high bias**. Since Ridge adds regularization, too much of it can make the model too simple. So, we should **reduce the regularization hyperparameter α** to let the model learn more complex patterns.

---

10. Why would you use:

- **Ridge Regression instead of plain Linear Regression?**
  To **prevent overfitting** by penalizing large weights. It adds L2 regularization.

- **Lasso instead of Ridge?**
  Lasso uses L1 regularization, which can **shrink some weights to 0** — useful for **feature selection**.
- **Elastic Net instead of Lasso?**
  Lasso can behave badly when features are correlated. Elastic Net combines **L1 + L2**, so it handles correlated features better and still performs feature selection.

---

11. You want to classify pictures as outdoor/indoor and daytime/nighttime. Should you use two Logistic Regression classifiers or one Softmax?

These are **two separate binary classifications**, so it's better to use **two Logistic Regression classifiers**. Softmax is used for **multi-class problems** with **mutually exclusive** classes, which isn't the case here.

---

12. Implement Batch Gradient Descent with early stopping for Softmax Regression (without using Scikit-Learn)

```
import numpy as np

def softmax(z):
    exp = np.exp(z - np.max(z, axis=1, keepdims=True))
    return exp / np.sum(exp, axis=1, keepdims=True)

def cross_entropy_loss(y_true, y_pred):
    m = y_true.shape[0]
    return -np.sum(y_true * np.log(y_pred + 1e-15)) / m

def one_hot(y, num_classes):
    return np.eye(num_classes)[y]

def softmax_regression(X, y, lr=0.1, epochs=1000, patience=10):
    m, n = X.shape
    k = len(np.unique(y))
    X = np.c_[np.ones((m, 1)), X]  # Add bias
    y_onehot = one_hot(y, k)
    theta = np.random.randn(n + 1, k)

    best_loss = np.inf
    wait = 0

    for epoch in range(epochs):
        logits = X.dot(theta)
        y_pred = softmax(logits)
        loss = cross_entropy_loss(y_onehot, y_pred)

        if loss < best_loss:
            best_loss = loss
            best_theta = theta.copy()
            wait = 0
        else:
            wait += 1
            if wait >= patience:
                print(f"Early stopping at epoch {epoch}")
                break
```

```
        gradient = X.T.dot(y_pred - y_onehot) / m
        theta -= lr * gradient

    return best_theta
```
