به نام یگانه خالق هستی بخش

سفری به اعماق شبکه‌های عصبی

درس

هوش محاسباتی

استاد

دکتر حسین کارشناس

دستیاران آموزشی

رضا برزگر

علی شاه‌زمانی

آرمان خلیلی

نویسندگان

سید حسین حسینی دولت‌آبادی

محمدامین نصیری

دانشکده مهندسی کامپیوتر

دانشگاه دولتی اصفهان

بهار ۱۴۰۴

1. بخش اول) رقص یک نورون به سبک رگرسیون لجستیک

1.1. مقدمه(Introduction)

در این پروژه با هدف درک عمیق‌تر نحوه عملکرد یک نورون ساده، مدل Logistic Regression را بر پایه ریاضی و با پیاده‌سازی دستی بازسازی کردیم. این نورون، وظیفه‌ی تمایز بین تصاویر هواپیما (کلاس 0) و سایر کلاس‌های دیتاست CIFAR-10 را دارد. هدف اصلی این پروژه، آموزش یک مدل دودویی ساده برای طبقه‌بندی تصاویر و بررسی دقت و توانایی آن در حل یک مسئله بینایی ماشین است.

1.2. بارگذاری و نرمال‌سازی داده‌ها

داده‌ها شامل 60000 تصویر رنگی 32 * 32 در 10 کلاس هستند.

با تقسیم بر 255، تصاویر نرمال‌سازی می‌شوند تا در محدوده [0, 1] قرار بگیرند.

```
# Load in the data
cifar10 = tf.keras.datasets.cifar10

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
```

1.3. تبدیل مسئله به طبقه‌بندی دودویی

تنها کلاس 0 (هواپیما) در مقابل بقیه بررسی شده و برچسب‌ها به صورت زیر اصلاح شدند.

```
y_train = np.where(y_train == 0, 0, 1)
y_test = np.where(y_test == 0, 0, 1)
```

1.4. آماده‌سازی داده‌ها برای مدل

تصاویر 32×32×3 به بردارهای با طول 3072 تبدیل می‌شوند.

```
# reshape images from (32, 32, 3) to (3072,)
X_train = X_train.reshape((X_train.shape[0], -1))  # (50000, 3072)
X_test = X_test.reshape((X_test.shape[0], -1))     # (10000, 3072)

y_train, y_test = y_train.flatten(), y_test.flatten() # (50000,  -   10000,)
```

1.5. تعریف توابع مدل

1.5.1. تابع فعال‌سازی سیگموید

در اینجا باید تابع فعالسازی سیگموید را تعریف کنیم که در کلاسی به نام Activation قرار می گیرد تا بتوان در تمام مراحل از آن استفاده کرد و همچنین مشتق آن را هم قرار می دهیم برای اینکه بتوانیم مراحل مشتق زنجیره ای را به درستی و به صورت بهینه پیاده سازی کنیم:

```
@staticmethod
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

@staticmethod
def sigmoid_derivative(a):
    return a * (1 - a)
```

1.5.2. تابع پیش‌بینی

نکته ای که قابل ذکر است این است که در اینجا برای اینکه بتوانیم ساختار یک نورون و سپس یک لایه از نورون ها و سپس یک شبکه عصبی را تعریف کنیم، از یک ساختار منظم که هرکدام از این ها را تعریف میکند استفاده شده است که در بخش سوم به صورت مفصل تر توضیح داده خواهد شد:

```
def predict(self, X):
    y_pred = self.forward(X)
    return (y_pred >= 0.5).astype(int)
```

1.6. تعریف تابع هزینه(Loss Function)

تابع هزینه برای رگرسیون لجستیک کراس انتروپی هست، که در دل یکی از همین کلاس های ساختار یافته ساخته شده است، که برای جلوگیری از مقدار تعریف نشده تقسیم بر صفر، یک مقدار بسیار کوچک به اپسیلون داده شده است:

```
def compute_loss(self, y_true, y_pred):
    epsilon = 1e-8
    return -np.mean(y_true * np.log(y_pred + epsilon) + (1 - y_true) * np.log(1 - y_pred + epsilon))
```

1.7. الگوریتم Gradient Descent

در این قسمت الگوریتم گرادیان کاهشی را که وظیف آن آپدیت کردن ورن ها برای رسیدن به نقطه بهینه هست را تعریف میکنیم ولی به دلیل اینکه در شبکه های عصبی بر عملکرد بگ پرو پگیشن این کار انجام می شود اسم آن در اینجا به همان نام است ولی دقیقا همان کار را انجام میدهد:

```
def backward(self, da, lr):
    m = self.input.shape[0]
    if self.activation_name == 'sigmoid':
        dz = da * Activation.sigmoid_derivative(self.a)
    elif self.activation_name == 'relu':
        dz = da * Activation.relu_derivative(self.z)

    dw = np.dot(self.input.T, dz) / m
    db = np.sum(dz, axis=0, keepdims=True) / m
    da_prev = np.dot(dz, self.w.T)

    self.w -= lr * dw
    self.b -= lr * db

    return da_prev
```

1.8. آموزش مدل

یکی از چالش هایی که در فرایند آموزش مطرح است این است که در صورت داشتن حجم زیاد دیتا باید منابع سخت افزاری قوی تری استفاده کرد ولی با توجه به محدودیت ها و اینکه بتوانیم عملکرد مدل را در پیدا کردن نقطه بهینه تقویت کنیم از ساختاری استفاده میکنیم تا بتوانیم دیتا های آموزشی را برای آموزش به مدل بدهیم در این راستا از روشی به نام Mini Batch استفاده میکنیم که در آن به جای دادن تمام داده ها به مدل، در هر دور به صورت بخش بخش داده ها به مدل داده شده و وزن ها آپدیت میشود:

```python
def train(self, X, y, epochs=100, lr=0.01, batch_size=64, verbose=True):
    for epoch in range(epochs):
        # ✅ Shuffle the data at the beginning of each epoch
        indices = np.arange(X.shape[0])
        np.random.shuffle(indices)
        X_shuffled = X[indices]
        y_shuffled = y[indices]

        epoch_loss = []

        # ✅ Mini-batch gradient descent
        for i in range(0, X.shape[0], batch_size):
            x_batch = X_shuffled[i:i+batch_size]
            y_batch = y_shuffled[i:i+batch_size].reshape(-1, 1)

            # Forward pass
            output = x_batch
            for layer in self.layers:
                output = layer.forward(output)

            # ✅ Derivative of binary cross-entropy with sigmoid output (dz = a - y)
            dz = output - y_batch
            gradient = dz

            # Backward pass
            for layer in reversed(self.layers):
                gradient = layer.backward(gradient, lr)

            # Compute batch loss
            batch_loss = np.mean(
                -y_batch * np.log(output + 1e-8) - (1 - y_batch) * np.log(1 - output + 1e-8)
            )
            epoch_loss.append(batch_loss)
```

با استفاده از دستورات زیر فرایند آموزش را شروع می کنیم:

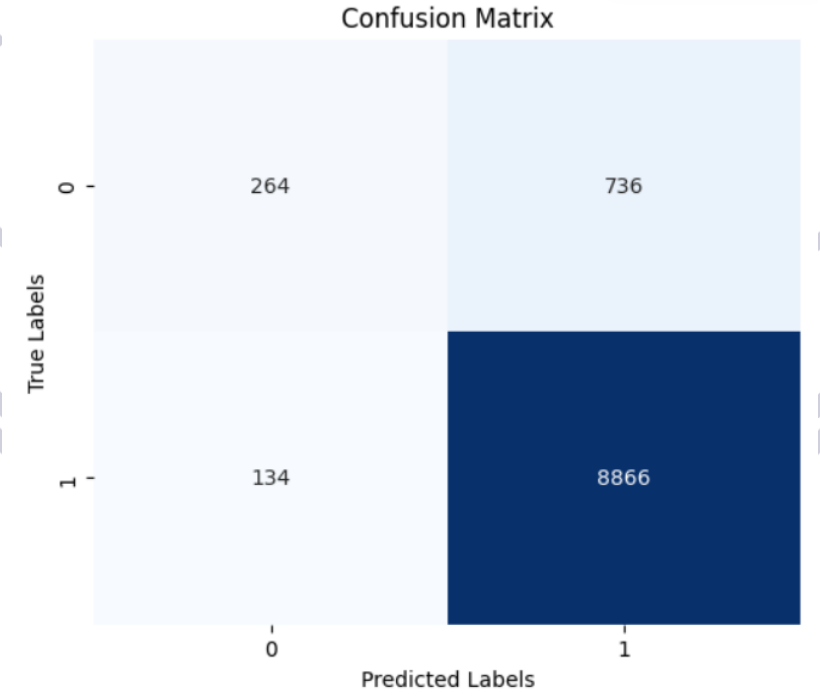
```
model = NeuralNetwork()
model.add(DenseLayer(3072, 1, activation='sigmoid'))

model.train(X_train, y_train, epochs=1000, lr=0.01, batch_size=64)
predictions = model.predict(X_test)
```

لازم به ذکر است که فرایند آموزش در این راستا نزدیک به 120 دقیقه به طول انجامیده است: (نمونه ای از دوره های آموزش)

```
Epoch 900: Loss = 0.2451
Epoch 910: Loss = 0.2450
Epoch 920: Loss = 0.2450
Epoch 930: Loss = 0.2449
Epoch 940: Loss = 0.2456
Epoch 950: Loss = 0.2454
Epoch 960: Loss = 0.2448
Epoch 970: Loss = 0.2451
Epoch 980: Loss = 0.2456
Epoch 990: Loss = 0.2449
```

1.9. ارزیابی عملکرد مدل

دقت، F1 Score، Precision و Recall به‌عنوان معیارهای ارزیابی نمایش داده می‌شوند. (مدل عمکرد خوبی داشته است)




```
F1 Score: 0.9532308353940436
Accuracy: 0.913
Precision: 0.9233493022287024
Recall: 0.9851111111111112
```

2. بخش دوم) اتحاد نورون‌ها در یک لایه پنهان به سبک دودویی

2.1. مقدمه(Introduction)

هدف این پروژه، پیاده‌سازی یک شبکه عصبی ساده با یک **لایه پنهان** برای انجام طبقه‌بندی **دودویی** است. به‌جای استفاده از فریمورک‌های سطح بالا مانند Keras یا PyTorch ، تمام عملیات (از جمله فورووارد، بک‌پراپ و آپدیت وزن‌ها) به‌صورت دستی انجام شده است تا درک عمیق‌تری از عملکرد نورون‌ها حاصل شود.

2.2. بارگذاری و نرمال‌سازی داده‌ها

داده‌ها شامل 60000 تصویر رنگی 32 * 32 در 10 کلاس هستند.

با تقسیم بر 255، تصاویر نرمال‌سازی می‌شوند تا در محدوده [0, 1] قرار بگیرند.

```
# Load in the data
cifar10 = tf.keras.datasets.cifar10

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
```

2.3. تبدیل مسئله به طبقه‌بندی دودویی

تنها کلاس 0 (هواپیما) در مقابل بقیه بررسی شده و برچسب‌ها به صورت زیر اصلاح شدند.

```
y_train = np.where(y_train == 0, 0, 1)
y_test = np.where(y_test == 0, 0, 1)
```

2.4. آماده‌سازی داده‌ها برای مدل

تصاویر 32×32×3 به بردارهای با طول 3072 تبدیل می‌شوند.

```
# reshape images from (32, 32, 3) to (3072,)
X_train = X_train.reshape((X_train.shape[0], -1))  # (50000, 3072)
X_test = X_test.reshape((X_test.shape[0], -1))     # (10000, 3072)

y_train, y_test = y_train.flatten(), y_test.flatten() # (50000,  -   10000,)
```

2.5. تعریف شبکه عصبی با یک لایه پنهان

در اینجا با استفاده از تمامی ساختار پروژه قبل که شامل پیاده سازی شبکه عصبی و لایه ها است استفاده میکنیم تا از تکرار مجدد و بازنویسی جلوگیری کنیم و کد را بهینه و تمیز بنویسیم.

2.5.1. ساختار مدل

- لایه ورودی: 3072 نورون + باياس (هر پیکسل از تصاویر به عنوان یک پارامتر در نظر گرفته میشود.)
- لایه پنهان: 64 نورون با سیگموید (برای پیدا کردن روابط پیچیده تر استفاده میشود)
- لایه خروجی: 1 نورون با سیگموید (پیش بینی شبکه را نشان میدهد)

```
model = NeuralNetwork()
model.add(DenseLayer(3072, 64, activation='sigmoid'))
model.add(DenseLayer(64, 1, activation='sigmoid'))

model.train(X_train, y_train, epochs=1000, lr=0.01, batch_size=64)
predictions = model.predict(X_test)
```

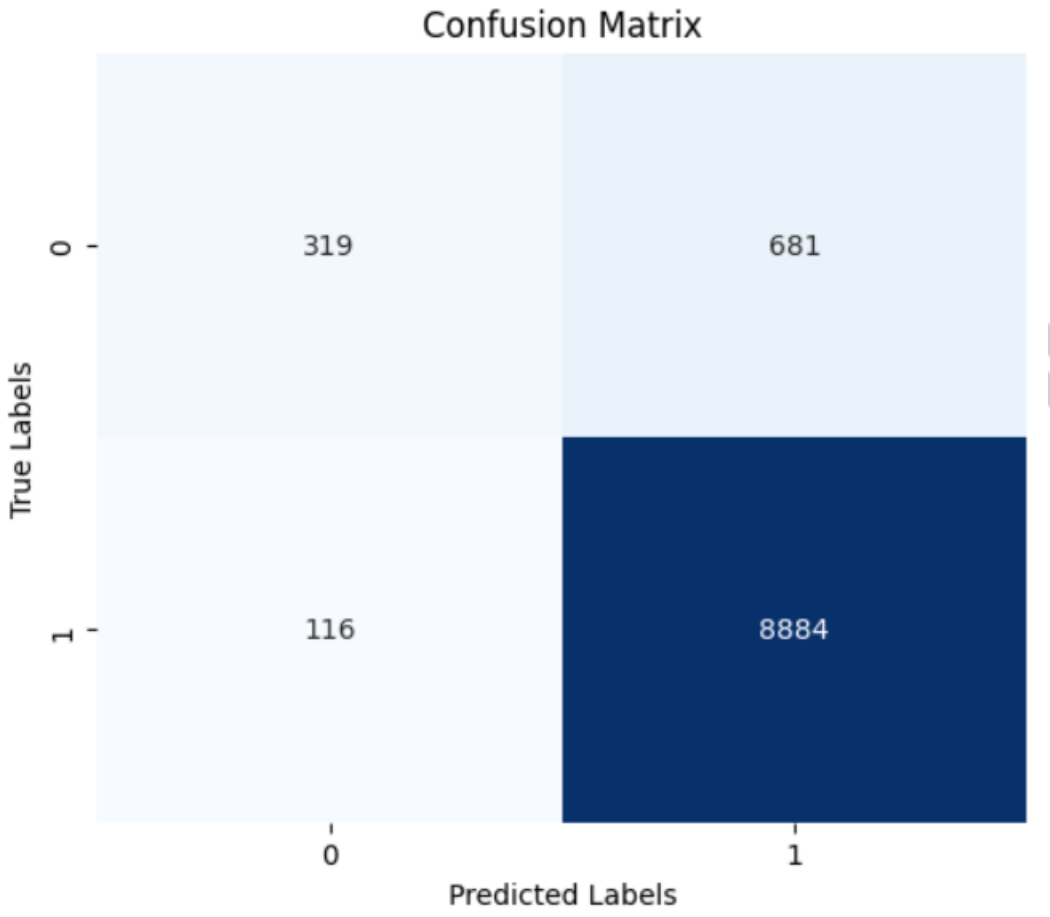
2.5.2. آموزش مدل

در اینجا با اجرای دستور مرحله قبل، مدل خود را آموزش می دهیم:

```
Epoch 900: Loss = 0.2396
Epoch 910: Loss = 0.2398
Epoch 920: Loss = 0.2389
Epoch 930: Loss = 0.2388
Epoch 940: Loss = 0.2383
Epoch 950: Loss = 0.2382
Epoch 960: Loss = 0.2377
Epoch 970: Loss = 0.2374
Epoch 980: Loss = 0.2373
Epoch 990: Loss = 0.2369
```

2.5.3. ارزیابی عملکرد مدل

دقت، F1 Score، Precision و Recall به‌عنوان معیارهای ارزیابی نمایش داده می‌شوند. (مدل عمکرد خوبی داشته است)




```
F1 Score: 0.957069754915163
Accuracy: 0.9203
Precision: 0.9288029273392577
Recall: 0.9871111111111112
```

2.6. نتیجه‌گیری

با توجه به نتیجه این بخش و بخش قبلی میتوان متوجه این شد که اضافه کردن یک لایه به شبکه عصبی و افزایش تعداد نورون ها میتواند توابع پیچیده و ساختار های پیچیده را بهتر درک کند و بتواند پیش بینی دقیق تری داشته باشد و خوب در قسمت ارزیابی پروژه اول و پروژه دوم میتوان این موضوع را به راحتی درک کرد ولی چالشی که با افزایش تعداد لایه ها و نورون ها همراه است این هست که زمان آموزش مدل به شدت افزایش میابد که این موضوع نیاز به داشتن منابع سخت افزاری بیشتر را برایمان آشکار میکند.

3. بخش سوم) ورود به دنیای طبقه‌بندی‌کننده‌های چندکلاسه

3.1. مقدمه(Introduction)

در این پروژه، یک شبکه عصبی چندکلاسه (Multi-class Neural Network) از صفر با استفاده از NumPy پیاده‌سازی شده است تا بر روی دیتاست CIFAR-10 آموزش ببیند. برخلاف مدل‌های دودویی، این پروژه طبقه‌بندی 10 کلاس متفاوت را بدون استفاده از فریم‌ورک‌های آماده مانند Keras یا TensorFlow پیاده‌سازی می‌کند. هدف اصلی، درک عمیق از نحوه عملکرد لایه‌های Dense، توابع Softmax و Backpropagation در مسئله‌های چندکلاسه است.

3.2. بارگذاری و نرمال‌سازی داده‌ها

داده‌ها شامل 60000 تصویر رنگی 32 * 32 در 10 کلاس هستند.

با تقسیم بر 255، تصاویر نرمال‌سازی می‌شوند تا در محدوده [0, 1] قرار بگیرند.

```
# Load in the data
cifar10 = tf.keras.datasets.cifar10

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
```

3.3. تبدیل مسئله به طبقه‌بندی دودویی

تنها کلاس 0 (هواپیما) در مقابل بقیه بررسی شده و برچسب‌ها به صورت زیر اصلاح شدند.

```
y_train = np.where(y_train == 0, 0, 1)
y_test = np.where(y_test == 0, 0, 1)
```

3.4. آماده‌سازی داده‌ها برای مدل

تصاویر 32×32×3 به بردارهای با طول 3072 تبدیل می‌شوند.

```
# reshape images from (32, 32, 3) to (3072,)
X_train = X_train.reshape((X_train.shape[0], -1))  # (50000, 3072)
X_test = X_test.reshape((X_test.shape[0], -1))     # (10000, 3072)

y_train, y_test = y_train.flatten(), y_test.flatten() # (50000,  -   10000,)
```

3.5. تبدیل برچسب‌ها به One-Hot

برای اینکه بتوانیم مقیاس بندی مناسب تری داشته باشیم و به طور صحیح مدل خود را آموزش دهیم نیاز است که کلاس ها را انکود کرده و برچسب های خروجی را با قاعده One-Hot انکود کنیم و سپس وارد فرایند آموزش مدل کنیم:

```
y_train = to_categorical(y_train, num_classes=10)
```

3.6. تعریف ساختار شبکه

- لایه ورودی: 3072 + 1 نورون (با بایاس)
- لایه پنهان: 64 نورون با تابع فعال‌سازی Sigmoid
- لایه خروجی: 10 نورون با تابع Softmax

3.7. توابع اصلی

با توجه به اینکه یکسری از توابع قبلا پیاده سازی شده است فقط از توابع جدید در این راستا استفاده میکنیم:

3.7.1. Softmax

کاربرد این تابع این است که بر خلاف سیگموید این قابلیت را دارد که بتواند کلاس های بیشتری را پیش بینی کند و اصطلاحا به آن چند کلاسه میگویند:

```
@staticmethod
def softmax(z):
    exps = np.exp(z - np.max(z, axis=1, keepdims=True))  # For numerical stability
    return exps / np.sum(exps, axis=1, keepdims=True)

@staticmethod
def softmax_derivative(output):
    # The exact derivative of softmax is a Jacobian matrix.
    # However, when used with cross-entropy loss, the gradient simplifies to:
    # ∂L/∂z = y_pred - y_true
    # So this method is typically not needed explicitly during backpropagation.
    pass
```

3.7.2. تابع هزینه (Loss Function)

```
    else:  # Multi-class classification
        return -np.mean(np.sum(y_true * np.log(y_pred + epsilon), axis=1))
```

3.8. فاز آموزش (Training)

شامل:

- فورووارد پروپگیشن(Forward Pass)

در این مرحله با استفاده از توابع فعالسازی که برای هر لایه انتخاب شده مقدار پیش بینی شده به لایه بعدی منتقل میشود.

```
def forward(self, x):
    self.input = x
    self.z = np.dot(x, self.w) + self.b
    if self.activation_name == 'sigmoid':
        self.a = Activation.sigmoid(self.z)
    elif self.activation_name == 'relu':
        self.a = Activation.relu(self.z)
    elif self.activation_name == 'softmax':
        self.a = Activation.softmax(self.z)
    else:
        raise ValueError(f"Unsupported activation function: {self.activation_name}")
    return self.a
```

- بک‌پروپگیشن(Backward Pass)

در این مرحله با توجه به گرادیان نزولی که به صورت مشتق زنجیره ای در شبکه گرفته میشود، خطا به جهت آپدیت وزن ها به عقب بر میگردد.

```
def backward(self, da, lr):
    m = self.input.shape[0]

    # Derivative depending on activation function
    if self.activation_name == 'sigmoid':
        dz = da * Activation.sigmoid_derivative(self.a)
    elif self.activation_name == 'relu':
        dz = da * Activation.relu_derivative(self.z)
    elif self.activation_name == 'softmax':
        # With softmax + cross-entropy, da is already (y_pred - y_true)
        dz = da
    else:
        raise ValueError(f"Unsupported activation function: {self.activation_name}")
```

- آپدیت وزن‌ها با گرادیان نزولی

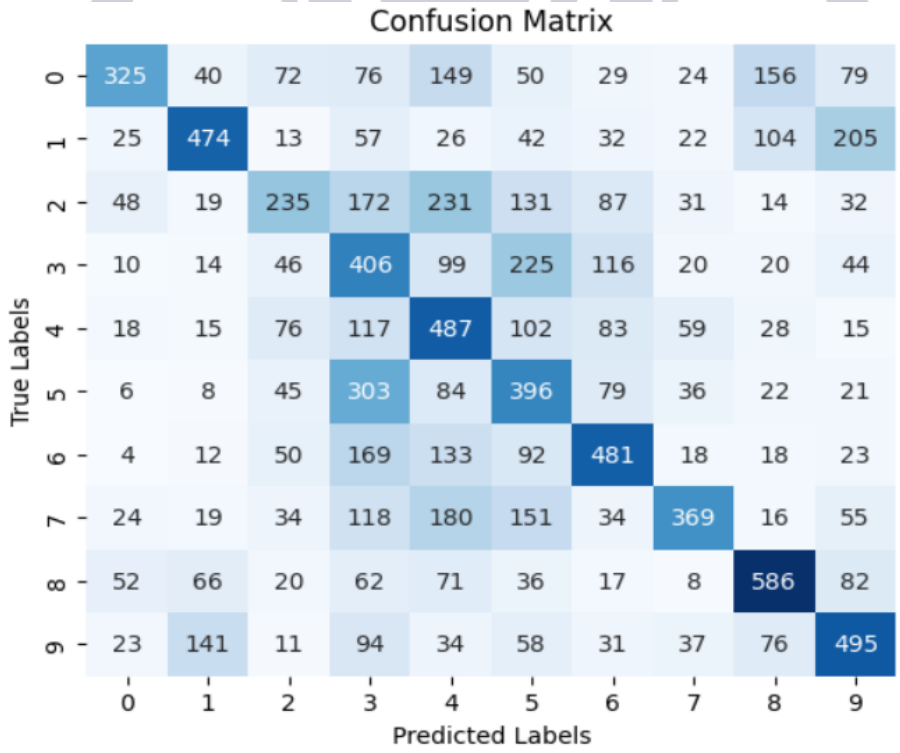
در این مرحله با استفاده از اطلاعات مرحله قبل وزن ها آپدیت میشود.

```
# Compute gradients
dw = np.dot(self.input.T, dz) / m
db = np.sum(dz, axis=0, keepdims=True) / m
da_prev = np.dot(dz, self.w.T)

# Update weights
self.w -= lr * dw
self.b -= lr * db

return da_prev
```

نکته) فرایند چیدمان لایه ها و آموزش مثل قبل انجام میشود و در ادامه به روند بررسی و ارزیابی توجه خواهیم کرد.

3.9. ارزیابی عملکرد مدل

دقت، F1 Score، Precision و Recall به‌عنوان معیارهای ارزیابی نمایش داده می‌شوند.




```
F1 Score (macro): 0.4289378687974864
Accuracy: 0.4254
Precision (macro): 0.4589177003924503
Recall (macro): 0.42539999999999994
```

3.10. نتیجه گیری

با توجه به آموزش های قبلی دیدم که در صورت داشتن مدل دودویی میزان پیش بینی از دقت بالایی برخوردار بود ولی در اینجا با افت شدیدی در دقت و پیش بینی مدل همراه هستیم که این نشان دهنده این است که در روند پیش بینی پیچیدگی داریم که با دولایه مدنظر ما این پیچیده قابل ارزیابی به شکل درست نبوده است و نیاز است که تعداد لایه های شبکه افزایش یابد و روند انتخاب توابع فعالسازی در شبکه با دید بهتری انتخاب شود.

چالشی که مطرح است این است که ما روش ثابتی برای تعیین تعداد لایه ها و تعداد نورون ها و یا نوع توابع فعالسازی نداریم و این مستلزم این است که به روند آموزش و تست بتوانیم به نتیجه مطلوبی برسیم.

4. بخش چهارم) تنوع شبکه‌ها و بهینه‌سازها برای پیشرفت

4.1. مقدمه(Introduction)

در دنیای یادگیری ماشین و به‌ویژه یادگیری عمیق، ساختار شبکه عصبی و نوع بهینه‌سازی که برای آموزش استفاده می‌شود، نقشی حیاتی در عملکرد نهایی مدل دارد. این پروژه با هدف بررسی تأثیر **تنوع در معماری شبکه‌های عصبی** و **انواع الگوریتم‌های بهینه‌سازی (Optimizers)** بر عملکرد مدل در دسته‌بندی تصاویر دیتاست مشهور **CIFAR-10** طراحی شده است.

ایده‌ی اصلی این است که با ثابت نگه داشتن سایر مؤلفه‌ها (مثل داده‌ها و معیارهای ارزیابی)، تنها با تغییر نوع شبکه یا الگوریتم یادگیری، بررسی کنیم کدام ترکیب بهترین نتایج را ارائه می‌دهد.

4.2. بارگذاری و نرمال‌سازی داده‌ها

داده‌ها شامل 60000 تصویر رنگی 32 * 32 در 10 کلاس هستند.

با تقسیم بر 255، تصاویر نرمال‌سازی می‌شوند تا در محدوده [0, 1] قرار بگیرند.

```
# Load in the data
cifar10 = tf.keras.datasets.cifar10

(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train, X_test = X_train / 255.0, X_test / 255.0
```

4.3. تبدیل مسئله به طبقه‌بندی دودویی

تنها کلاس 0 (هواپیما) در مقابل بقیه بررسی شده و برچسب‌ها به صورت زیر اصلاح شدند.

```
y_train = np.where(y_train == 0, 0, 1)
y_test = np.where(y_test == 0, 0, 1)
```

4.4. آماده‌سازی داده‌ها برای مدل

تصاویر 32×32×3 به بردارهای با طول 3072 تبدیل می‌شوند.

```
# reshape images from (32, 32, 3) to (3072,)
X_train = X_train.reshape((X_train.shape[0], -1))  # (50000, 3072)
X_test = X_test.reshape((X_test.shape[0], -1))     # (10000, 3072)

y_train, y_test = y_train.flatten(), y_test.flatten() # (50000,  -   10000,)
```

4.5. تبدیل برچسب‌ها به One-Hot

برای اینکه بتوانیم مقیاس بندی مناسب تری داشته باشیم و به طور صحیح مدل خود را آموزش دهیم نیاز است که کلاس ها را انکود کرده و برچسب های خروجی را با قاعده One-Hot انکود کنیم و سپس وارد فرایند آموزش مدل کنیم:

```
y_train = to_categorical(y_train, num_classes=10)
```

4.6. تعریف توابع فعال‌سازی(Activation Functions)

```
class Activation:
    sigmoid, relu, softmax, tanh
```

چهار تابع مهم برای فعال‌سازی پیاده‌سازی شده‌اند:

- Sigmoid: برای داده‌های دودویی، اما با مشکل vanishing gradient
- ReLU: بسیار محبوب، سریع و مؤثر در شبکه‌های عمیق
- Softmax: برای لایه خروجی در مسائل طبقه‌بندی چندکلاسه
- Tanh: نسخه بهبودیافته sigmoid ، اما همچنان دارای gradient ضعیف برای مقادیر بزرگ

4.7. ساختار شبکه عصبی سفارشی

شبکه عصبی به صورت کلاس محور و دستی (from scratch) بدون استفاده از ماژول آماده Keras ساخته شده است:

اجزای اصلی:

- DenseLayer: لایه‌ی کامل با قابلیت وزن‌دهی و bias

```
class DenseLayer:
    def __init__(self, input_size, output_size, activation='sigmoid'):
        self.w = np.random.randn(input_size, output_size) * np.sqrt(2.0 / input_size) # He Initialization
        self.b = np.zeros((1, output_size))
        self.activation_name = activation
        self.z = None
        self.a = None
        self.input = None
        self.dw = None
        self.db = None
```

- NeuralNetwork: ترکیب چند لایه، آموزش، پیش‌بینی و ارزیابی

```
class NeuralNetwork:
    def __init__(self, optimizer):
        self.layers = []
        self.optimizer = optimizer
        self.loss_history = []
        self.accuracy_history = []
```

- Optimizer: پیاده‌سازی momentum و بهینه‌سازهای دستی

```
class SGD:
    def __init__(self, lr=0.01):
        self.lr = lr
```

```
class Momentum:
    def __init__(self, lr=0.01, beta=0.9):
        self.lr = lr
        self.beta = beta
        self.v_w = {}
        self.v_b = {}
```

```
class Adam:
    def __init__(self, lr=0.001, beta1=0.9, beta2=0.999, epsilon=1e-8):
        self.lr = lr
        self.beta1 = beta1
        self.beta2 = beta2
        self.epsilon = epsilon
        self.m_w, self.v_w = {}, {}
        self.m_b, self.v_b = {}, {}
        self.t = {}
```

ویژگی‌ها:

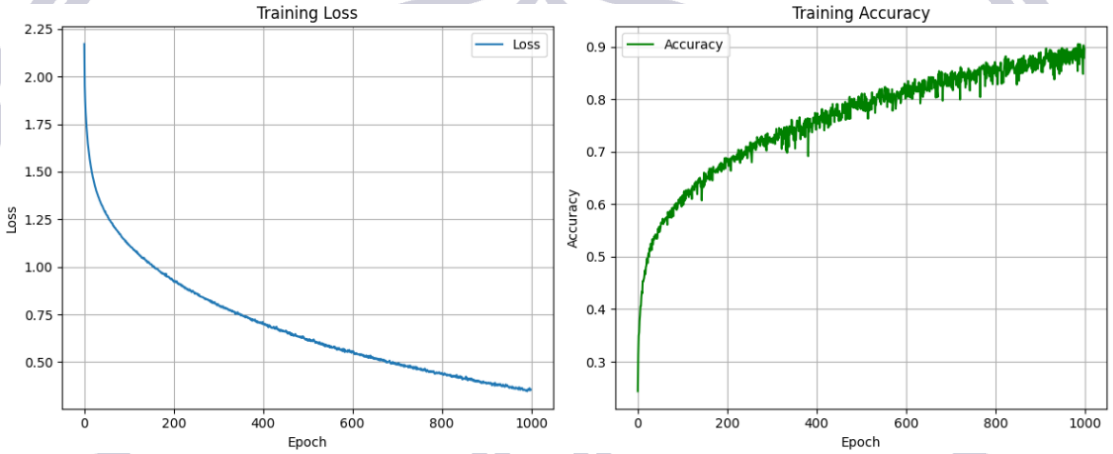
- پشتیبانی از تابع فعال‌سازی سفارشی
- پیاده‌سازی دستی backward propagation
- استفاده از **He Initialization** برای افزایش پایداری در ReLU

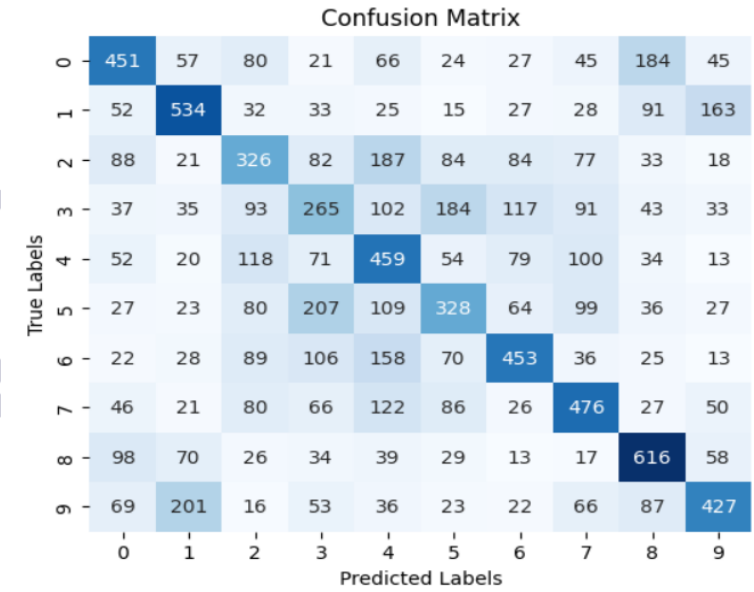
4.8. تحلیل آموزش و مقایسه معماری‌ها

برای مقایسه دقیق‌تر، چندین سناریو آموزشی اجرا شده‌اند که تفاوت آن‌ها در انتخاب تابع فعال‌سازی و نوع initial weightsاست:

1. Sigmoid + Momentum + Random Init

```
optimizer = Momentum(lr=0.05)
nn = NeuralNetwork(optimizer=optimizer)

nn.add(DenseLayer(3072, 64, activation='sigmoid'))
nn.add(DenseLayer(64, 10, activation='softmax'))

nn.train(X_train, y_train, epochs=1000, batch_size=64)
predictions = nn.predict(X_test)
```



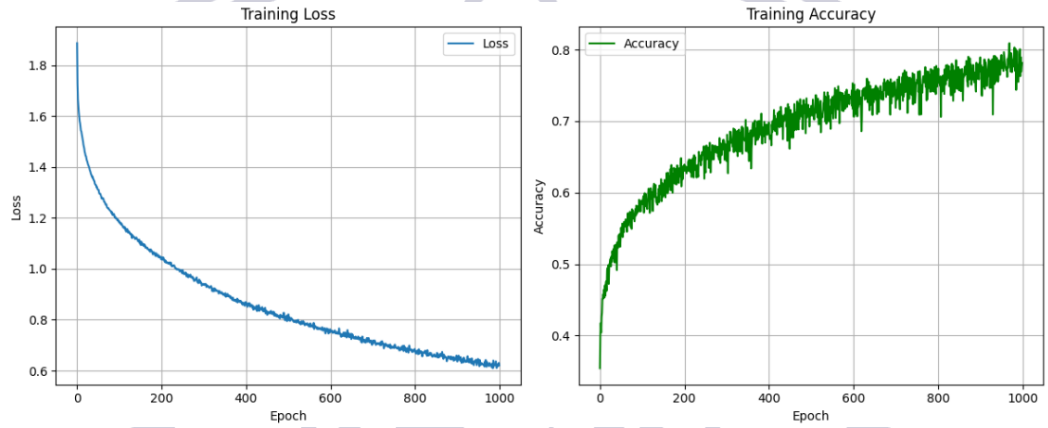


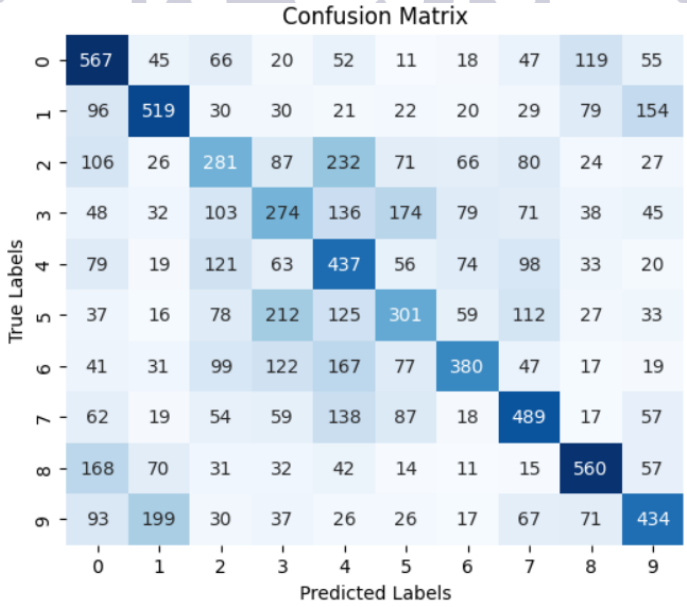

```
F1 Score (macro): 0.432000339323514
Accuracy: 0.4335
Precision (macro): 0.43392889423888575
Recall (macro): 0.4335
```

میتوان مشاهده کرد که به نسبت گرادیان کاهشی معمولی عملکردی بهتر و مناسب تر داشته است.

2. ReLU + Momentum + He Init

```
optimizer = Momentum(lr=0.05)
nn = NeuralNetwork(optimizer=optimizer)

nn.add(DenseLayer(3072, 64, activation='relu'))
nn.add(DenseLayer(64, 10, activation='softmax'))

nn.train(X_train, y_train, epochs=1000, batch_size=64)
predictions = nn.predict(X_test)
```



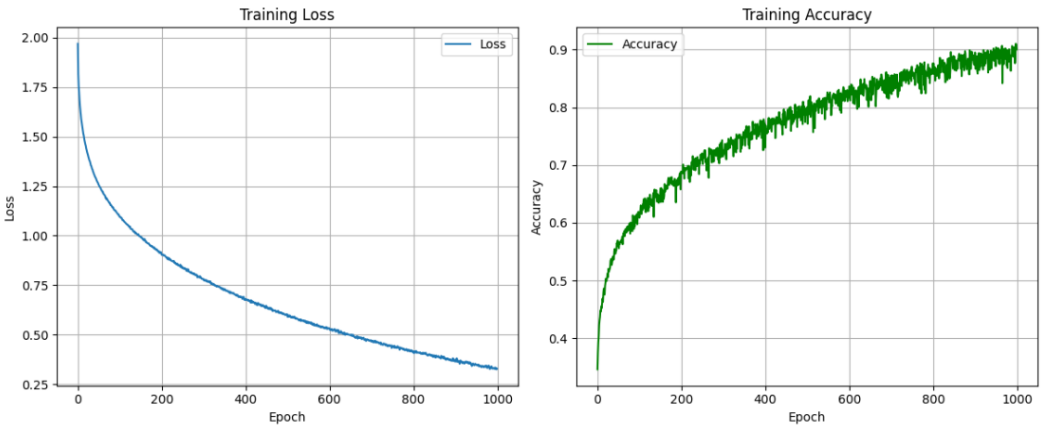


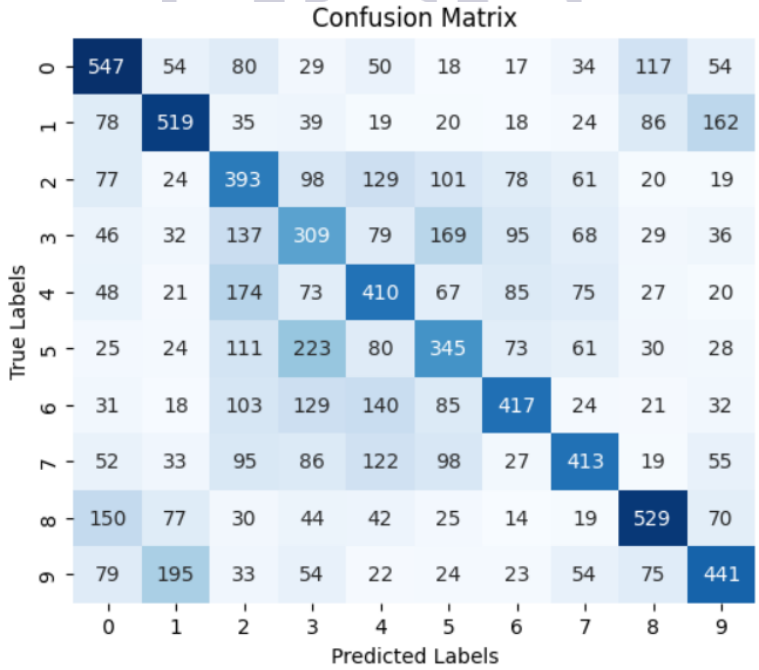

```
F1 Score (macro): 0.4227155180016916
Accuracy: 0.4242
Precision (macro): 0.4278528734440787
Recall (macro): 0.4242
```

در این پروژه عملکرد دقیق‌تر و پایدارتری ارائه داده است نسبت به ReLU + Softmax، که Sigmoid + Softmax برخلاف انتظار رایج در شبکه‌های عمیق است، اما نشان می‌دهد در پروژه‌های با ساختار ساده‌تر، تابع فعال‌سازی sigmoid همچنان می‌تواند مؤثر باشد. (همچنین مقدار دهی اولیه تاثیر زیادی در روند آموزش دارد)

3. Sigmoid + Momentum + He Init

```
optimizer = Momentum(lr=0.05)
snn = NeuralNetwork(optimizer=optimizer)

snn.add(DenseLayer(3072, 64, activation='sigmoid'))
snn.add(DenseLayer(64, 10, activation='softmax'))

snn.train(X_train, y_train, epochs=1000, batch_size=64)
predictions = snn.predict(X_test)
```






```
F1 Score (macro): 0.43372184502938566
Accuracy: 0.4323
Precision (macro): 0.438001962226447
Recall (macro): 0.43229999999999996
```

اگر از تابع فعال‌سازی sigmoid استفاده می‌کنی، مقداردهی اولیه‌ی ساده و کم‌دامنه (Randomبا std کوچک) عملکرد بسیار بهتری دارد نسبت به He Initialization ، چون مانع از اشباع شدن نورون‌ها می‌شود و گرادیان مؤثر حفظ می‌شود. (نتیجه این تغییرات را تا حد محدودی میتوان مشاهده کرد.)

4. Tanh + Momentum + He Init

```
optimizer = Momentum(lr=0.05)
tnn = NeuralNetwork(optimizer=optimizer)

tnn.add(DenseLayer(3072, 64, activation='tanh'))
tnn.add(DenseLayer(64, 10, activation='softmax'))

tnn.train(X_train, y_train, epochs=1000, batch_size=64)
predictions = tnn.predict(X_test)
```

مزایا:

- تابع tanh برخلاف sigmoid خروجی را در بازه [1, -1]نگه می‌دارد، که نرمال‌سازی بهتری برای داده‌ها در شبکه فراهم می‌کند.
- مشتق tanh در بازه مرکزی (اطراف ۰) بزرگ‌تر از sigmoid است → گرادیان مؤثرتر.
- ترکیب با softmax در خروجی مشکلی ایجاد نمی‌کند چون tanh فقط در لایه‌های پنهان استفاده شده.

معایب:

- He Initializationمناسب tanh نیست (برای ReLU طراحی شده) → در برخی نورون‌ها باعث ورود به ناحیه اشباع می‌شود.
- نسبت به sigmoid + random init همگرایی کندتری داشت، زیرا ورودی‌ها سریع‌تر به لبه‌های بازه tanh می‌رسند.
- نسبت به ReLU نیز سرعت آموزش پایین‌تر ولی پایداری بیشتر بود.

ترکیب	دقت نهایی	پایداری یادگیری	سرعت همگرایی	توضیح
Sigmoid + Softmax + Random	⭐ بیشتر	خوب	خوب	ورودی‌ها در ناحیه فعال نگه داشته می‌شوند
ReLU + Softmax + He	کم‌تر	ناپایدارتر	سریع‌تر (ولی ناپایدار)	نورون‌های مرده، نوسان شدید
Tanh + Softmax + He	متوسط	متوسط	کندتر	به -1 تا 1 tanh مقیاس خروجی مناسب آن He کمک می‌کند، اما نیست

5. بخش پنجم) ورود به دنیای شبکه‌های عصبی عمیق کانولوشنی(CNN)

این پروژه به بررسی و پیاده‌سازی یک شبکه عصبی کانولوشنی (CNN) برای طبقه‌بندی مجموعه داده تصاویر CIFAR-10 می‌پردازد. در ابتدا، یک ساختار CNN ساده پیاده‌سازی شد که عملکرد ضعیفی (دقت حدود 10٪) از خود نشان داد. سپس، با اعمال بهبودهای معماری کلیدی شامل افزودن لایه‌هایBatch Normalization ، Dropout، افزایش عمق شبکه و تعداد فیلترها، یک مدل CNN بهبودیافته (ImprovedCNN) توسعه داده شد. این مدل بهبودیافته توانست به دقت قابل توجهی حدود 86.80٪ بر روی داده‌های آزمون دست یابد. در نهایت، عملکرد و ویژگی‌های CNN با یک پرسپترون چندلایه (MLP)برای وظایف مشابه مقایسه شده و برتری ذاتی CNN برای پردازش تصویر تشریح گردیده است.

5.2. مقدمه

هدف این پروژه، پیاده‌سازی و آموزش یک شبکه عصبی عمیق برای طبقه‌بندی تصاویر رنگی کوچک از مجموعه داده CIFAR-10 بود. شامل 60,000 تصویر 32 * 32 پیکسلی در 10 کلاس مختلف است (هواپیما، ماشین، پرنده، گربه، گوزن، سگ، قورباغه، اسب، کشتی، کامیون) که 50,000 تصویر برای آموزش و 10,000 تصویر برای آزمون در نظر گرفته شده است. شبکه‌های عصبی کانولوشنی (CNNs) به دلیل توانایی‌شان در یادگیری ویژگی‌های سلسله‌مراتبی از داده‌های تصویری، معماری استاندارد برای این نوع وظایف محسوب می‌شوند.

5.3. متدولوژی:

- **مجموعه دادهCIFAR-10 :**
- **پیش‌پردازش داده‌ها:**
  - تبدیل تصاویر به تنسورهای.PyTorch
  - نرمال‌سازی مقادیر پیکسل‌ها با استفاده از میانگین و انحراف معیار استاندارد مجموعه داده CIFAR-10 ((0.4914, 0.4822, 0.4465), (0.2023, 0.1994, 0.2010)).
  - *نکته: در ساختار بهبودیافته، افزایش داده (Data Augmentation) مانند چرخش تصادفی یا برش تصادفی برای داده‌های آموزشی پیاده‌سازی نشد، که می‌توانست به بهبود بیشتر تعمیم‌پذیری مدل کمک کند.*

- **معماری مدل‌ها:**
  - ساختار اولیه (CNN ساده):
    - لایه کانولوشن اول: 3 ورودی(RGB) ، 32 فیلتر خروجی، کرنل 3*3، پدینگ 1، فعال‌ساز ReLU.
    - لایه Max Pooling اول: کرنل 2*2، استراید 2.
    - لایه کانولوشن دوم: 32 ورودی، 64 فیلتر خروجی، کرنل 3*3، پدینگ 1، فعال‌ساز ReLU .
    - لایه Max Pooling دوم: کرنل 2*2، استراید 2.
    - لایه Flatten .
    - لایه Fully Connected (FC) اول: 128 نورون، فعال‌ساز ReLU .
    - لایه FC دوم (خروجی): 10 نورون (متناسب با تعداد کلاس‌ها).
  - ساختار بهبودیافته:(ImprovedCNN)
    - بلوک کانولوشنی 1:

```
self.conv_block1 = nn.Sequential(
    nn.Conv2d(in_channels=3, out_channels=64, kernel_size=3, padding=1, bias=False), # bias=False because we have BN
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, padding=1, bias=False),
    nn.BatchNorm2d(64),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2) # 64x16x16
```

    - بلوک کانولوشنی 2:

```
self.conv_block2 = nn.Sequential(
    nn.Conv2d(in_channels=64, out_channels=128, kernel_size=3, padding=1, bias=False),
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.Conv2d(in_channels=128, out_channels=128, kernel_size=3, padding=1, bias=False),
    nn.BatchNorm2d(128),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2) # 128x8x8
```

    - بلوک کانولوشنی 3:

```
self.conv_block3 = nn.Sequential(
    nn.Conv2d(in_channels=128, out_channels=256, kernel_size=3, padding=1, bias=False),
    nn.BatchNorm2d(256),
    nn.ReLU(),
    nn.Conv2d(in_channels=256, out_channels=256, kernel_size=3, padding=1, bias=False),
    nn.BatchNorm2d(256),
    nn.ReLU(),
    nn.MaxPool2d(kernel_size=2, stride=2) # 256x4x4
```

- لایه Flatten.

```
self.flatten = nn.Flatten()
# Flattened size: 256 * 4 * 4 = 4096
```

- بلوک Fully Connected:

```
self.fc_block = nn.Sequential(
    nn.Linear(256 * 4 * 4, 512, bias=False), # You can put 1024 or 512 neurons
    nn.BatchNorm1d(512), # BN for FC layers
    nn.ReLU(),
    nn.Dropout(0.5), # Dropout with probability 0.5
    nn.Linear(512, num_classes)
)
```

- **تنظیمات آموزش:**
  - **تعداد Epoch ها: 100**
  - **اندازه Batch: 128**
  - **نرخ یادگیری: 0.001**
  - **بهینه‌ساز (Optimizer): Adam**
  - **تابع هزینه (Loss Function): CrossEntropyLoss**
  - **سخت‌افزار: GPU (NVIDIA T4 در محیط Colab)**

5.4. نتایج و بحث:

- **عملکرد ساختار اولیه:**
  - مدل اولیه CNN عملکرد بسیار ضعیفی داشت و دقت آن بر روی داده‌های آزمون حدود 10٪ بود. این دقت معادل حدس تصادفی برای یک مسئله 10 کلاسه است.
  - تابع هزینه آموزشی (Training Loss) در طول آموزش تقریباً ثابت و حدود 2.3 باقی ماند (که برابر با ln(10) است)، نشان‌دهنده عدم یادگیری مدل بود.
  - دقت بر روی هر کلاس نیز نامطلوب بود و مدل تمایل به پیش‌بینی تنها یک کلاس (car) برای اکثر نمونه‌ها داشت.

- **دلایل احتمالی عملکرد ضعیف:**
  - عدم وجودBatch Normalization : این امر می‌توانست منجر به ناپایداری در آموزش، مشکل "internal covariate shift" و کندی همگرایی شود.
  - عدم وجود Dropout یا سایر روش‌های منظم‌سازی(Regularization) : احتمال بیش‌برازش (overfitting)حتی در مراحل اولیه وجود داشت، یا مدل در مینیمم‌های محلی نامطلوب گیر کرده بود.
  - معماری نسبتاً ساده :ظرفیت مدل برای یادگیری ویژگی‌های پیچیده تصاویر CIFAR-10 کافی نبود.

- **عملکرد ساختار بهبودیافته:(ImprovedCNN)**
  - مدل ImprovedCNNپیشرفت چشمگیری را نشان داد و به دقت نهایی 86.80%بر روی مجموعه داده آزمون دست یافت.
  - نمودار هزینه آموزشی (Training Loss) کاهش مداوم و معنی‌داری را در طول 100 epoch نشان داد و از مقادیر اولیه بالای 1.0 به حدود 0.0079 رسید.
  - نمودار هزینه اعتبارسنجی (Validation Loss) نیز در ابتدا به سرعت کاهش یافت و به حداقل حدود 0.48 در epoch ششم رسید، سپس به تدریج افزایش یافت و در انتهای 100 epoch به حدود 0.98 رسید. این رفتار (کاهش و سپس افزایش Validation Loss در حالی که Training Loss همچنان در حال کاهش است) نشانه‌ای از شروع بیش‌برازش (overfitting) مدل بر روی داده‌های آموزشی پس از حدود 10–20 epoch اولیه است.
  - دقت اعتبارسنجی (Validation Accuracy) به سرعت افزایش یافت و به بیشینه حدود 87.01٪ در 88 epochرسید و سپس کمی نوسان داشت.

- دقت بر روی هر کلاس نیز به طور قابل توجهی بهبود یافت:
  - **plane: 88.00%**
  - **car: 96.10%**
  - **bird: 82.70%**
  - **cat: 73.00%**
  - **deer: 85.80%**
  - **dog: 81.70%**
  - **frog: 90.40%**
  - **horse: 88.00%**
  - **ship: 92.10%**
  - **truck: 90.20%**

مشاهده می‌شود که کلاس‌هایی مانند 'car', 'frog', 'ship', 'truck' دقت بالاتری دارند، در حالی که 'cat' و 'dog' که ممکن است از نظر بصری شباهت بیشتری به سایر کلاس‌ها یا تنوع درون‌کلاسی بیشتری داشته باشند، دقت کمتری کسب کرده‌اند.

- **تحلیل بهبودها در ImprovedCNN :**
  - **Batch Normalization**: نقش حیاتی در تثبیت آموزش، کاهش حساسیت به مقادیر اولیه وزن‌ها، امکان استفاده از نرخ یادگیری بالاتر و تسریع همگرایی داشته است. همچنین به عنوان یک نوع منظم‌ساز عمل می‌کند.
  - افزایش عمق و تعداد فیلترها :سه بلوک کانولوشنی با تعداد فیلترهای افزایشی (25664 <- 128 <-) ظرفیت مدل را برای یادگیری ویژگی‌های پیچیده‌تر و سلسله‌مراتبی از تصاویر افزایش داده است.
  - Dropout: با نرخ 0.5 در بلوک Fully Connected به کاهش بیش‌برازش کمک کرده است، اگرچه نمودار Validation Loss نشان می‌دهد که هنوز مقداری بیش‌برازش در انتهای آموزش رخ داده است.

5.5. مقایسه CNN با پرسپترون چندلایه(MLP) :

پرسپترون چندلایه(MLP) ، نوعی شبکه عصبی پیشخور است که از چندین لایه نورون کاملاً متصل (Fully Connected)تشکیل شده است. برای استفاده از MLP در طبقه‌بندی تصاویر، ابتدا تصویر ورودی باید به یک بردار یک‌بعدی مسطح (flatten) تبدیل شود.

ویژگی	شبکه عصبی کانولوشنی(CNN)	پرسپترون چندلایه(MLP)
ورودی	داده‌های با ساختار شبکه‌ای) مانند تصاویر 2(D/3D	بردارهای ویژگی یک‌بعدی
حفظ ساختار فضایی	بله، لایه‌های کانولوشن و پولینگ اطلاعات مکانی را حفظ می‌کنند.	خیر، با مسطح کردن تصویر، اطلاعات مکانی پیکسل‌ها از بین می‌رود.
استخراج ویژگی	به طور خودکار ویژگی‌های سلسله‌مراتبی (لبه، بافت، اشیاء) را یاد می‌گیرد.	مستقیماً بر روی پیکسل‌های مسطح شده عمل می‌کند؛ یادگیری ویژگی‌های فضایی دشوار است.
اشتراک پارامتر	بله، کرنل‌های کانولوشن در سراسر تصویر به اشتراک گذاشته می‌شوند که تعداد پارامترها را به شدت کاهش می‌دهد.	خیر، هر نورون به تمام نورون‌های لایه قبلی متصل است (تعداد پارامترها زیاد).
حساسیت به مکان	به دلیل لایه‌های پولینگ، تا حدی به جابجایی‌های کوچک ویژگی‌ها مقاوم است (Translation Invariance).	به مکان ویژگی‌ها بسیار حساس است.
تعداد پارامترها	معمولاً کمتر از MLP برای ورودی‌های تصویری با ابعاد مشابه (به دلیل اشتراک پارامتر).	برای تصاویر با ابعاد متوسط تا بزرگ، بسیار زیاد می‌شود.
عملکرد در تصویر	بسیار بالا، معماری استاندارد برای وظایف بینایی ماشین.	معمولاً ضعیف‌تر ازCNN ، به خصوص برای تصاویر پیچیده. مستعد بیش‌برازش.

چرا CNN برای CIFAR-10 بهتر از MLP است؟

- یادگیری ویژگی‌های فضایی CNN :ها با استفاده از فیلترهای کانولوشنی، الگوهای محلی مانند لبه‌ها، گوشه‌ها و بافت‌ها را تشخیص می‌دهند. این ویژگی‌ها سپس در لایه‌های بالاتر ترکیب شده و ویژگی‌های پیچیده‌تری را تشکیل می‌دهند MLP .این توانایی را ندارد.
- کاهش تعداد پارامترها :اشتراک پارامتر در CNN ها باعث می‌شود که با تعداد پارامترهای بسیار کمتری نسبت به MLPبتوانند مدل‌های عمیق و کارآمدی ساخت. برای یک تصویر x32x332، ورودی مسطح شده به MLP دارای 3072 ویژگی است. یک لایه FC اولیه در MLP با تعداد نورون‌های معقول، پارامترهای زیادی خواهد داشت.
- مقاومت به جابجایی :لایه‌های Max Pooling به CNN کمک می‌کنند تا نسبت به جابجایی‌های کوچک اشیاء در تصویر مقاوم باشند.

اگر یک MLP برای CIFAR-10 آموزش داده می‌شد، انتظار می‌رفت که دقت آن به مراتب کمتر از 86.80٪ باشد و به سرعت دچار بیش‌برازش شود، مگر اینکه از تعداد لایه‌ها و نورون‌های بسیار کمی استفاده شود که در آن صورت ظرفیت یادگیری آن نیز محدود می‌شد.

5.6. نتیجه‌گیری:

این پروژه با موفقیت نشان داد که چگونه می‌توان با طراحی یک معماری CNN مناسب و استفاده از تکنیک‌هایی مانند Batch Normalizationو Dropout ، به دقت بالایی در طبقه‌بندی تصاویر CIFAR-10 دست یافت. مدل اولیه با دقت 10٪ نشان‌دهنده اهمیت انتخاب معماری و تکنیک‌های آموزشی صحیح بود. مدل بهبودیافته ImprovedCNNبا دقت 86.80٪ عملکرد بسیار خوبی از خود نشان داد. مقایسه با MLP نیز برتری ذاتی CNN ها را برای وظایف پردازش تصویر به دلیل توانایی آن‌ها در یادگیری ویژگی‌های فضایی و سلسله‌مراتبی و همچنین مدیریت کارآمدتر پارامترها، برجسته کرد.

5.7. پیشنهادات برای کارهای آینده:

- افزایش داده :(Data Augmentation) اعمال تکنیک‌هایی مانند برش تصادفی، چرخش تصادفی، و تغییر رنگ برای داده‌های آموزشی می‌تواند به کاهش بیش‌برازش و بهبود تعمیم‌پذیری مدل کمک شایانی کند.
- تنظیم نرخ یادگیری :(Learning Rate Scheduling) استفاده از یک زمان‌بند برای کاهش نرخ یادگیری در طول آموزش (مثلاً StepLRیا ReduceLROnPlateau)می‌تواند به همگرایی بهتر و یافتن مینیمم‌های بهتر کمک کند.
- توقف زودهنگام(Early Stopping) : پایش Validation Loss و توقف آموزش زمانی که این هزینه شروع به افزایش می‌کند، می‌تواند از بیش‌برازش شدید جلوگیری کند.
- تنظیم هایپرپارامترها :آزمایش با نرخ‌های یادگیری مختلف، اندازه‌های Batch متفاوت، و نرخ‌های Dropout متفاوت.
- معماری‌های پیشرفته‌تر :پیاده‌سازی یا استفاده از معماری‌های شناخته‌شده و قدرتمندتر مانندResNet ، VGG، یا DenseNetکه نتایج بهتری بر روی CIFAR-10 کسب کرده‌اند.
- افزایش تعداد Epoch ها :با وجود Early Stopping و Data Augmentation ، می‌توان مدل را برای تعداد Epochهای بیشتری آموزش داد.