به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

یادگیری عمیق با کاربردها

تمرین شماره یک

نام و نام خانوادگی	امید نائیج نژاد
شماره دانشجویی	610301189
تاریخ ارسال گزارش	1403/12/25

فهرست مطالب

فهرست شکل ها

فهرست جدول ها

1-1 بررسی تئوری آموزش شبکه های یادگیری عمیق

ورودی $x \in \mathbb{R}^d$: بعد داده ورودی = d

لایه پنهان $h \in \mathbb{R}^m$: تعداد نورون در لایه پنهان = m

لایه خروجی $y \in \mathbb{R}^k$: تعداد نورون در لایه خروجی = k

وزن های لایه پنهان $W^{(1)} \in \mathbb{R}^{m \times d}$

بایاس های لایه پنهان $b^{(1)} \in \mathbb{R}^m$

وزن های لایه خروجی $W^{(2)} \in \mathbb{R}^{k \times m}$

بایاس های لایه خروجی $b^{(2)} \in \mathbb{R}^k$

توابع فعالسازی : $\sigma(\cdot)$

تابع هزینه : $\mathcal{L} = \frac{1}{2}\|y - t\|^2$ (رابطه 1)

محاسبات پیشرو:

محاسبات مربوط به لایه پنهان

$$z^{(1)} = W^{(1)}x + b^{(1)} \quad \text{(رابطه 2)}$$

$$h = \sigma(z^{(1)}) \quad \text{(رابطه 3)}$$

محاسبات مربوط به لایه خروجی

$$z^{(2)} = W^{(2)}h + b^{(2)} \quad \text{(رابطه 4)}$$

$$y = \sigma(z^{(2)}) \quad \text{(رابطه 5)}$$

محاسبات پسرو:

با استفاده از قانون زنجیره ای مشتق، مقدار سهم هر پارامتر شبکه (وزن یا بایاس) را از مقدار خطا محاسبه میکنیم تا در قسمت بعدی بتوانیم پارامترها را به روزرسانی کنیم.

$$\frac{\partial \mathcal{L}}{\partial z^{(2)}} = \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial z^{(2)}} = (y - t)^T \sigma'\left(z^{(2)}\right) = \delta^{(2)} \quad \text{(رابطه 6)}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(2)}} = \frac{\partial \mathcal{L}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial b^{(2)}} = \delta^{(2)} \quad \text{(رابطه 7)}$$

$$\frac{\partial \mathcal{L}}{\partial W^{(2)}} = \frac{\partial \mathcal{L}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial W^{(2)}} = \delta^{(2)} h^T \quad \text{(رابطه 8)}$$

$$\frac{\partial \mathcal{L}}{\partial z^{(1)}} = \frac{\partial \mathcal{L}}{\partial z^{(2)}} \cdot \frac{\partial z^{(2)}}{\partial h} \cdot \frac{\partial h}{\partial z^{(1)}} = \left(W^{(2)}\right)^T \delta^{(2)} . \sigma'\left(z^{(1)}\right) = \delta^{(1)} \quad \text{(رابطه 9)}$$

$$\frac{\partial \mathcal{L}}{\partial b^{(1)}} = \frac{\partial \mathcal{L}}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial b^{(1)}} = \delta^{(1)} \quad \text{(رابطه 10)}$$

$$\frac{\partial \mathcal{L}}{\partial W^{(1)}} = \frac{\partial \mathcal{L}}{\partial z^{(1)}} \cdot \frac{\partial z^{(1)}}{\partial W^{(1)}} = \delta^{(1)} x^T \quad \text{(رابطه 11)}$$

به روزرسانی پارامترها:

با استفاده از گرادیان مقدار خطا نسبت به هر پارامتر که در قسمت پیشین محاسبه شد، اقدام به آپدیت کردن پارامترها میکنیم: (η = نرخ یادگیری)

$$b_{new}^{(2)} = b_{old}^{(2)} - \eta \frac{\partial \mathcal{L}}{\partial b^{(2)}} \quad \text{(رابطه 12)}$$

$$W_{new}^{(2)} = W_{old}^{(2)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(2)}} \quad \text{(رابطه 13)}$$

$$b_{new}^{(1)} = b_{old}^{(1)} - \eta \frac{\partial \mathcal{L}}{\partial b^{(1)}} \quad \text{(رابطه 14)}$$

$$W_{new}^{(1)} = W_{old}^{(1)} - \eta \frac{\partial \mathcal{L}}{\partial W^{(1)}} \quad \text{(رابطه 15)}$$

تابع اتلاف = CE:

تغییرات زیر ایجاد خواهدشد:

محاسبات پیشرو: خروجی شبکه به صورت زیر محاسبه میشود:

$$y = softmax(z^{(2)}) \quad \text{(رابطه 16)}$$

تغییرات محاسبات پسرو:

$$\frac{\partial \mathcal{L}}{\partial z^{(2)}} = \frac{\partial \mathcal{L}}{\partial y} \cdot \frac{\partial y}{\partial z^{(2)}} = y - t = \delta^{(2)} \quad \text{(رابطه 17)}$$

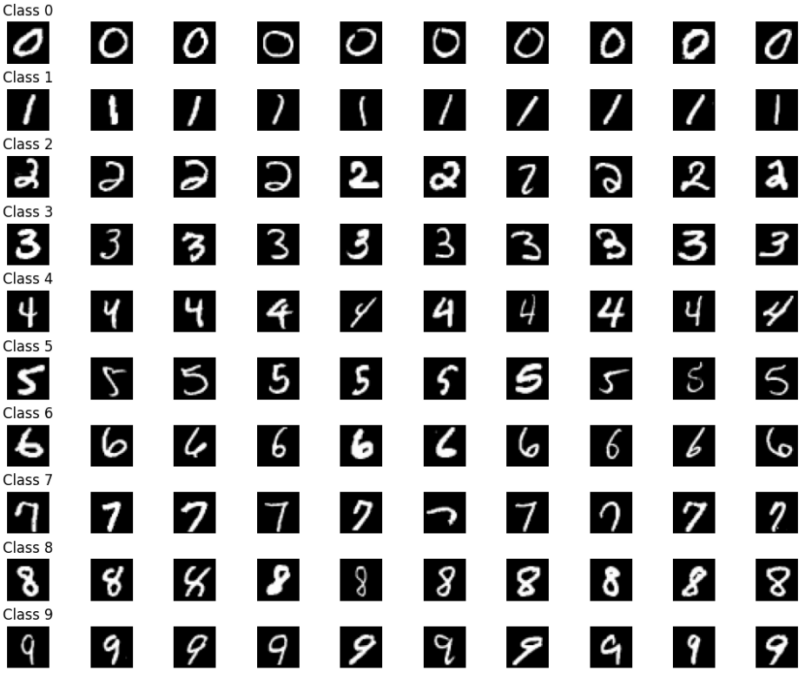
بقیه معادلات بدون تغییر باقی خواهد ماند.

1-2 پیاده سازی معماری و تنظیم ابرپارامتر ها

1-2-1 آماده سازی داده:

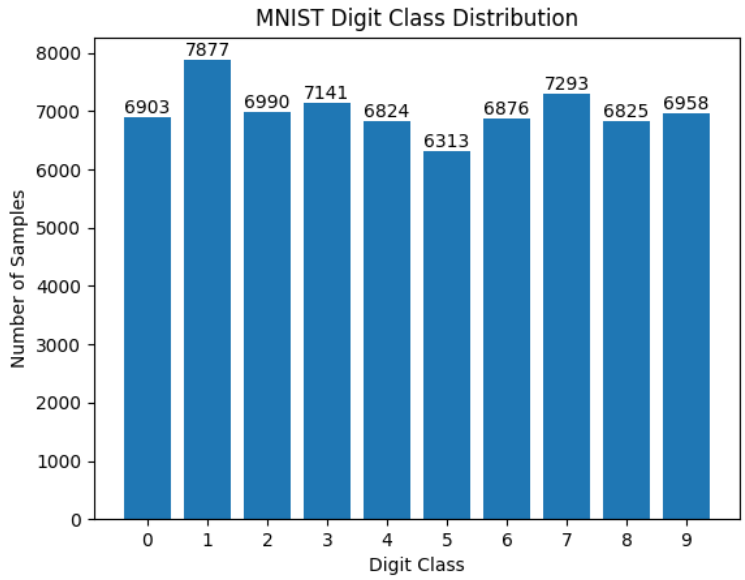
در گام اول به کاوش داده های ورودی می پردازیم، پس از دانلود کردن دیتاست توسط data_loader.py با استفاده از توابع پیاده سازی شده در visualization.py کار های خواسته شده در این بخش را انجام میدهیم. (توجه شود که همه فراخوانی های لازم از فایل های گفته شده در تابع توسعه داده شده به منظور انجام آن تسک که در فایل main.py قرار گرفته، صورت میپذیرد.)

- **تابع plot_sample_images در فایل visualization.py با گرفتن بردار X, Y که 10 نمونه تصادفی از هر کلاس را نمایش میدهد.**



شکل 1 - تصاویر تصادفی از mnist

- تابع hist_class_distribution در فایل visualization.py نیز با گرفتن بردار Y که همان لیبل تصاویر است هیستوگرام توزیع کلاس ها را رسم میکند. همانطور که در تصویر زیر مشاهده میکنید ناترازی قابل توجهی بین کلاسها وجود ندارد به عبارت دیگر کلاس ها نسبتا متعادل هستند.

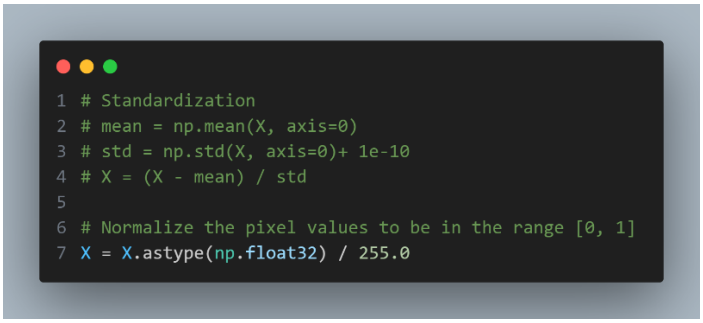


شکل 2 - هیستوگرام توزیع کلاس ها

پیش پردازش داده ها:

چون X_shape = (70000, 784) بود، تصاویر تک کاناله دانلود شده اند همچنین flat هستند.

با سعی و خطای فراوان، به جای تبدیل استانداردسازی که در فایل تمرین توصیه شده بود از تبدیل min max normalization استفاده کردم.



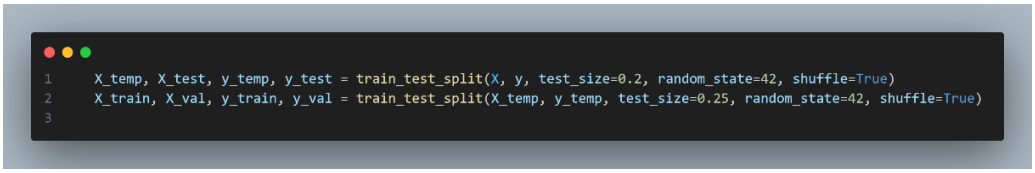
شکل 3- پیش پردازش تصاویر

تقسیم داده ها به سه گروه:

اگر از تمام داده برای آموزش استفاده شود، نمیتوان نتیجه مدل را روی داده های دیده نشده سنجید چون هیچ تضمینی وجود ندارد که شبکه در مواجهه با داده های جدید، همان رفتار داده های قبلی را داشته باشد. پس شبکه عصبی صرفا با استفاده از داده های آموزشی، یاد میگیرد.

همانطور که گفته شد لازم است علاوه بر داده های آموزشی که برای یادگیری پارامترهای شبکه بکار می روند، دسته ای از داده ها وجود داشته باشند که در آموزش شبکه دخالت نداشته اند تا بتوانیم از آنها برای پیداکردن مقدار مناسب برای ابرپارامترها استفاده کنیم. به این صورت که با آموزش چندباره شبکه با ابرپارامتر های گوناگون و سنجیدن عملکرد مدل روی داده های اعتبارسنجی، بهترین مقدار برای هر ابرپارامتر به دست می آید.

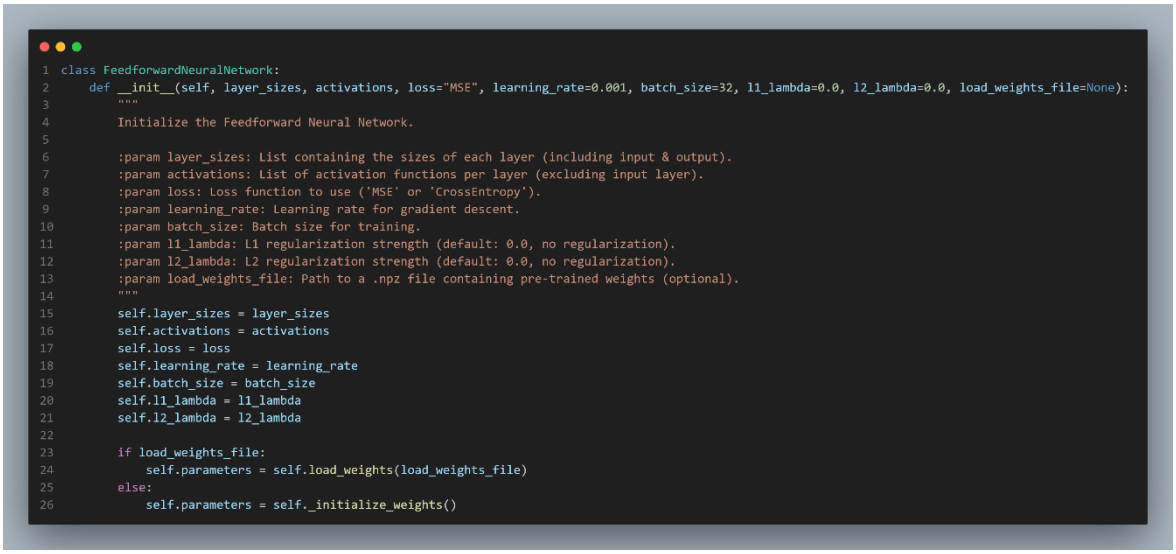
همچنین نیاز است دسته ای دیگر از داده ها وجود داشته باشند، که نه در آموزش و نه در انتخاب ابرپارامترها استفاده نشده باشند و صرفا برای تست کردن مدل روی داده هایی که در آینده برای ما می آیند استفاده شوند،که این دسته داده های آزمایش نامیده می شوند.



شکل 4 – تقسیم داده ها به 3 گروه

1-2-2 طراحی شبکه از ابتدا:

همانطور که در تکه کد زیر میبینید، کلاس FeedforwardNeuralNetwork طوری پیاده سازی شده که شما میتوانید هنگام ساختن یک آبجکت از آن تمام ابرپارامترهای شبکه را به صورت دلخواه مقدار دهی کنید. به عنوان مثال تعداد لایه ها و تعداد پرسپترون در هر لایه با لیست layer_sizes قابل تنظیم است و تابع فعالسازی پس از هر لایه به وسیله لیست activations مشخص میشود.

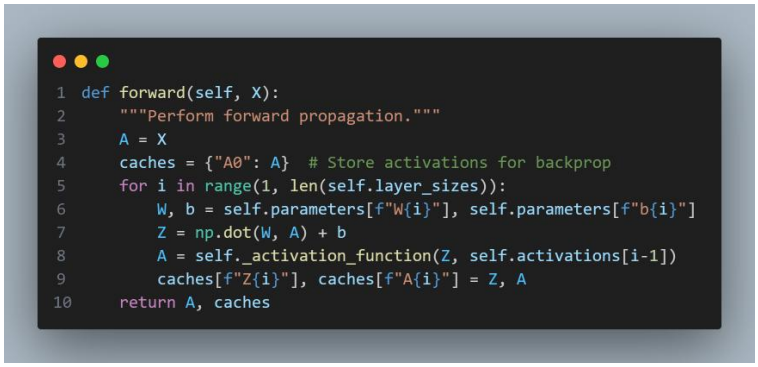


```python
class FeedforwardNeuralNetwork:
    def __init__(self, layer_sizes, activations, loss="MSE", learning_rate=0.001, batch_size=32, l1_lambda=0.0, l2_lambda=0.0, load_weights_file=None):
        """
        Initialize the Feedforward Neural Network.

        :param layer_sizes: List containing the sizes of each layer (including input & output).
        :param activations: List of activation functions per layer (excluding input layer).
        :param loss: Loss function to use ('MSE' or 'CrossEntropy').
        :param learning_rate: Learning rate for gradient descent.
        :param batch_size: Batch size for training.
        :param l1_lambda: L1 regularization strength (default: 0.0, no regularization).
        :param l2_lambda: L2 regularization strength (default: 0.0, no regularization).
        :param load_weights_file: Path to a .npz file containing pre-trained weights (optional).
        """
        self.layer_sizes = layer_sizes
        self.activations = activations
        self.loss = loss
        self.learning_rate = learning_rate
        self.batch_size = batch_size
        self.l1_lambda = l1_lambda
        self.l2_lambda = l2_lambda

        if load_weights_file:
            self.parameters = self.load_weights(load_weights_file)
        else:
            self.parameters = self._initialize_weights()
```

شکل 5 - سازنده کلاس **FeedforwardNeuralNetwork**

طراحی مسیر پیشرو:

کل محاسبات مسیر پیشرو توسط تابع forward صورت میپذیرد. هر لایه از لایه پایین دستی ورودی میگیرد و با انجام dot product بین ماتریس ورودی و ماتریس وزن های آن لایه و اضافه کردن بایاس به آن و در نهایت با اثر دادن تابع فعالسازی آن لایه، خروجی لایه مشخص میشود و به لایه بالادستی داده میشود. همچنین مقدار هر پرسپترون ذخیره و برگردانده میشود تا در فرایند پس انتشار خطا مورد استفاده قرار گیرد.



```python
def forward(self, X):
    """Perform forward propagation."""
    A = X
    caches = {"A0": A}  # Store activations for backprop
    for i in range(1, len(self.layer_sizes)):
        W, b = self.parameters[f"W{i}"], self.parameters[f"b{i}"]
        Z = np.dot(W, A) + b
        A = self._activation_function(Z, self.activations[i-1])
        caches[f"Z{i}"], caches[f"A{i}"] = Z, A
    return A, caches
```

شکل 6 - مسیر پیشرو

طراحی مسیر پسرو:

در مسیر پسرو باید ابتدا گرادیان تابع اتلاف حساب شود و سپس این گرادیان تا ورودی، بازگردانده شود (براساس قاون زنجیره ای مشتق). پس در گام نخست باید گرادیان را برای تابع اتلاف حساب کنیم سپس با پیمایش لایه ها از انتها به ابتدا گرادیان تابع اتلاف به هر پارامتر را محاسبه (با قاون زنجیره ای مشتق) و ذخیره کنیم تا در تابع update_parameters از مقادیر ذخیره شده استفاده کرده و پارامتر ها را آپدیت کنیم.

```python
def backward(self, X, Y, caches):
    grads = {}
    L = len(self.layer_sizes) - 1
    m = X.shape[1]

    # Compute dA for the output layer
    if self.loss == "CrossEntropy":
        if self.activations[-1] == "softmax":
            dZ = caches[f"A{L}"] - Y  # Simplified gradient for softmax + cross-entropy
        else:
            dZ = -(Y / (caches[f"A{L}"] + 1e-8))  # Avoid division by zero
    elif self.loss == "MSE":
        dZ = 2 * (caches[f"A{L}"] - Y) / m
    else:
        raise ValueError("Unsupported loss function")

    # Loop backward through layers
    for i in reversed(range(1, L + 1)):
        if i != L or self.activations[i-1] != "softmax":
            dZ = dZ * self._activation_function(caches[f"Z{i}"], self.activations[i-1], deriv=True)

        grads[f"dW{i}"] = np.dot(dZ, caches[f"A{i-1}"].T) / m
        grads[f"db{i}"] = np.sum(dZ, axis=1, keepdims=True) / m

        # Add L1 and L2 regularization gradients
        if self.l1_lambda > 0:
            grads[f"dW{i}"] += self.l1_lambda * np.sign(self.parameters[f"W{i}"])
        if self.l2_lambda > 0:
            grads[f"dW{i}"] += self.l2_lambda * 2 * self.parameters[f"W{i}"]

        if i > 1:
            dZ = np.dot(self.parameters[f"W{i}"].T, dZ)  # Update dZ for next layer

    return grads

```

شکل 7 - مسیر پسرو

به روز رسانی وزن های شبکه در طی مسیر پسرو

وزن های شبکه عصبی با استفاده از روش نزول گرادیان دسته ای و طبق رابطه زیر آپدیت میشود.

رابطه 1:

$$w_{new} = w_{old} - \eta \frac{\partial l}{\partial w}$$

```python
def update_parameters(self, grads):
    """Update parameters using gradient descent."""
    for i in range(1, len(self.layer_sizes)):
        self.parameters[f"W{i}"] -= self.learning_rate * grads[f"dW{i}"]
        self.parameters[f"b{i}"] -= self.learning_rate * grads[f"db{i}"]
```

شکل 8 - به روز رسانی وزن های شبکه در مسیر پسرو

```python
def train(self, X_train, Y_train, X_val, Y_val, epochs=20, classification=True):
    """
    Train the model without shuffling to reduce time complexity.
    """
    # Arrays to store training and validation metrics
    train_losses = np.empty(epochs)
    train_accuracies = np.empty(epochs)
    val_losses = np.empty(epochs)
    val_accuracies = np.empty(epochs)

    m = X_train.shape[1]
    for epoch in range(1, epochs + 1):
        for i in range(0, m, self.batch_size):
            # Extract mini-batch
            X_batch = X_train[:, i:i + self.batch_size]
            Y_batch = Y_train[:, i:i + self.batch_size]

            Y_pred, caches = self.forward(X_batch)
            loss = self._compute_loss(Y_pred, Y_batch)

            grads = self.backward(X_batch, Y_batch, caches)
            self.update_parameters(grads)

        # Compute validation loss and accuracy
        Y_val_pred, _ = self.forward(X_val)
        val_loss = self._compute_loss(Y_val_pred, Y_val)
        val_acc = accuracy(np.argmax(Y_val, axis=0), np.argmax(Y_val_pred, axis=0))

        # Compute training loss and accuracy
        Y_true = np.argmax(Y_train, axis=0)
        Y_pred, _ = self.forward(X_train)
        acc = accuracy(Y_true, np.argmax(Y_pred, axis=0))

        if classification:
            print(f"Epoch {epoch}/{epochs} - Train Loss: {loss:.4f} - Train Accuracy: {100 * acc:.4f}% - Val Loss: {val_loss:.4f} - Val Accuracy: {100 * val_acc:.4f}%")
        else:
            print(f"Epoch {epoch}/{epochs} - Train Loss: {loss:.4f} - Val Loss: {val_loss:.4f}")

        train_losses[epoch - 1] = loss
        train_accuracies[epoch - 1] = acc
        val_losses[epoch - 1] = val_loss
        val_accuracies[epoch - 1] = val_acc

    if classification:
        plot_loss_acc(train_losses, val_losses, train_accuracies, val_accuracies)
    else:
        plot_loss_curve(train_losses, val_losses, self.loss)
```

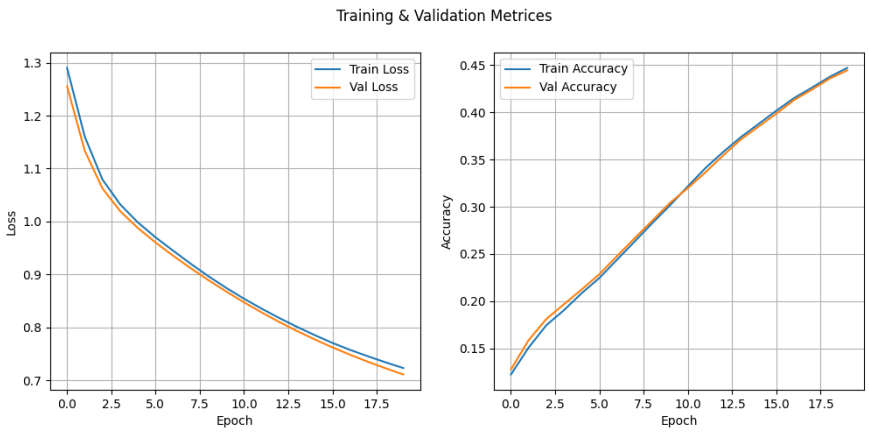
شکل 9 – کل فرایند آموزش شبکه عصبی

1-2-3 تنظیم ابر پارامتر ها:

مدل پایه:

با توجه به روند نمودار های زیر، بنظر میرسد شبکه به ایپاک های بیشتری برای آموزش نیاز دارد یعنی 20 ایپاک برای یادگیری 25450 پارامتر کافی نبوده است و یا اینکه نرخ یادگیری باید بیشتر از 0.001 باشد تا فرایند بهینه سازی به خوبی انجام بگیرد.

$$(784 + 1) \times 32 + (32 + 1) \times 10 = 25450\ parameters$$

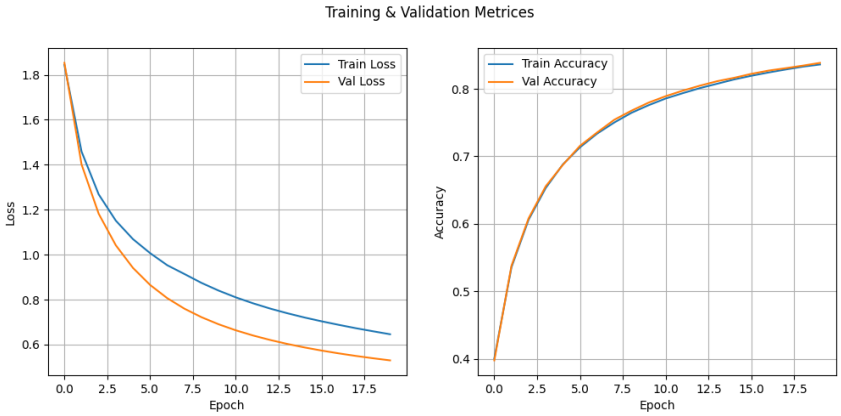


شکل 10 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل پایه

بررسی عملکرد تابع هزینه :

به جای تابع هزینه میانگین مربع خطا به کمک آنتروپی متقابل آموزش میدهیم.

مطابق انتظار با تابع هزینه آنتروپی متقابل، آموزش شبکه بسیار بهتر از حالت پایه انجام شده است چون طبیعت این تابع به گونه ای است که با تابع فعالساز سافتمکس به همگرا شدن پارامترها به مقادیر بهینه شان کمک میکند. همانطور که میبینید دقت مدل روی داده های اعتبار سنجی بسیار نزدیک به دقت آن روی داده های آموزشی است و این یعنی بیش برازش اتفاق نیفتاده و حتی میتوان آموزش را در ایپاک های بیشتری دنبال کرد.

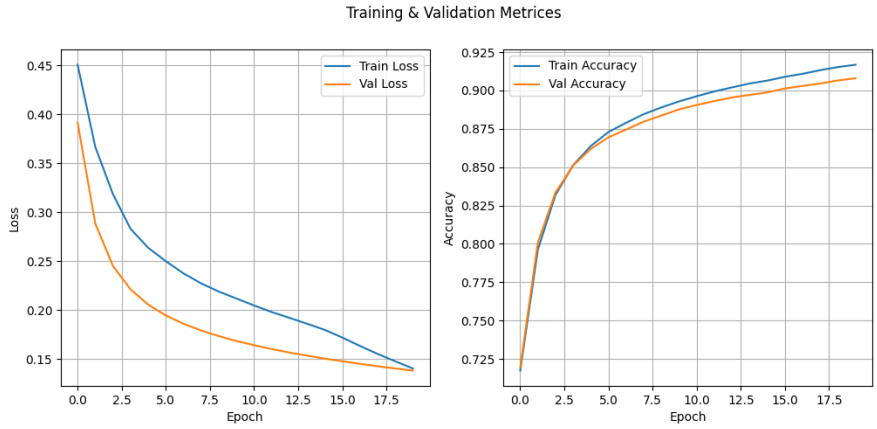


شکل 11 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با تابع هزینه آنتروپی متقابل

نرخ یادگیری:

ابتدا با نرخ یادگیری 0.1 شبکه را آموزش میدهیم:

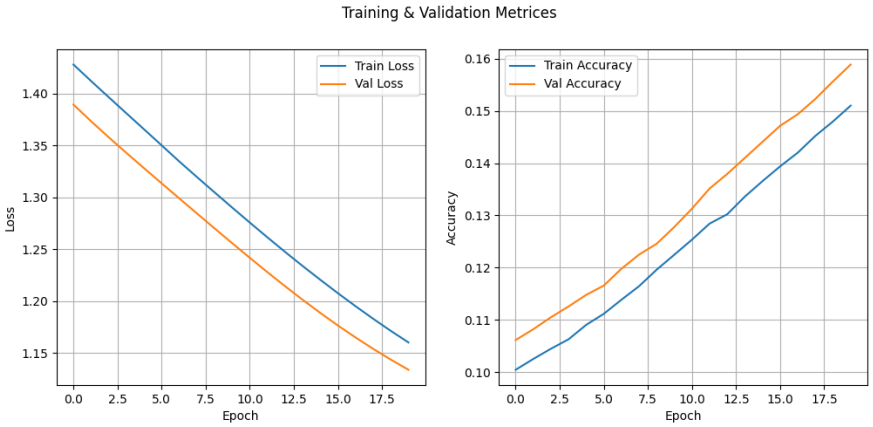
همانطور که در نمودار های زیر میبینید، با افزایش نرخ یادگیری به 0.1 روند بهینه سازی با سرعت بیشتری همراه شده و به نتیجه بهتری نسبت به حالت های قبل رسیده است؛ در نتیجه میتواند مقدار مناسبی برای این ابرپارامتر در این مسئله باشد. اما باید توجه داشته باشید که اگر مقدار آن بیش از اندازه زیاد شود، ریسک گیر افتادن در بهینه محلی بیشتر میشود و حتی ممکن است باعث واگرا شدن (diverge) الگوریتم بهینه سازی شود.



شکل 12 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با نرخ یادگیری 0.1

حالا با نرخ یادگیری 0.0001 شبکه را آموزش میدهیم:

همانطور که در نمودار های زیر میبینید، با کاهش نرخ یادگیری به 0.0001 روند بهینه سازی شدیدا با کاهش سرعت همراه شده و واضحا مینیمم سازی مقدار تابع اتلاف به زمان بسیار زیادی نیاز دارد. در نتیجه اصلا مقدار خوبی برای نرخ یادگیری نیست.

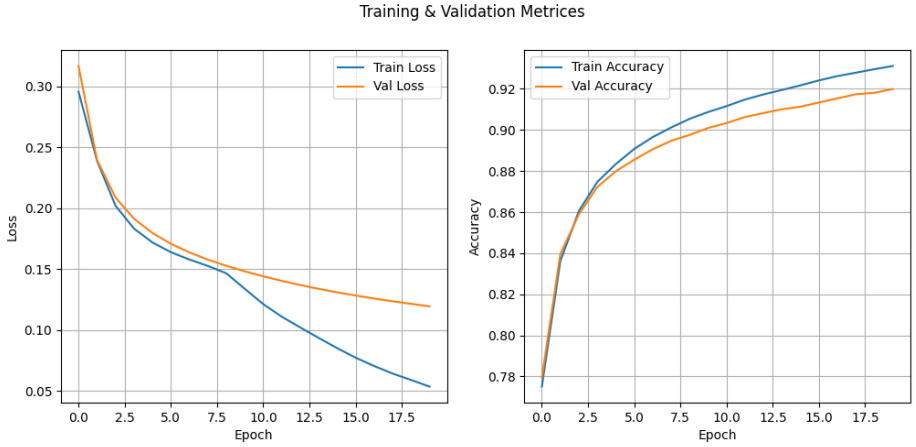


شکل 13 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با نرخ یادگیری 0.0001

اندازه دسته:

ابتدا با اندازه دسته برابر 8 شبکه را آموزش میدهیم:

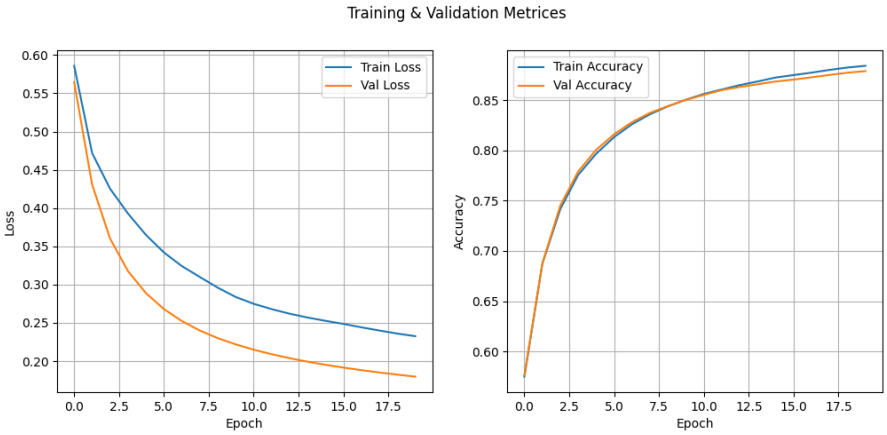
با توجه به نمودار ها، کاهش اندازه دسته به 8 باعث افزایش دقت مدل روی داده های آموزشی و اعتبارسنجی نسبت به مدل پایه شده اما بعد از ایپاک دوازدهم گپ قابل توجهی بین نمودار تابع هزینه برای داده های آموزشی و تابع هزینه برای داده های اعتبارسنجی ایجاد شده است که به معنای زنگ خطری برای رخ دادن بیش برازش در صورت انجام آموزش در ایپاک های بیشتر است. پس باید حواسمان به این نکته باشد!



شکل 14 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با اندازه دسته 8

حالا با اندازه دسته برابر 16 شبکه را آموزش میدهیم:

در این حالت هم گپ قابل توجهی بین نمودار تابع هزینه برای داده های آموزشی و تابع هزینه برای داده های اعتبارسنجی ایجاد شده است که به معنای زنگ خطری برای رخ دادن بیش برازش در صورت انجام آموزش در ایپاک های بیشتر است. همچنین دقت مدل نسبت به حالت پیش کمتر است پس 16 گزینه مناسبی برای این ابرپارامتر نیست.



شکل 15 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با اندازه دسته 16

حالا با اندازه دسته برابر 128 شبکه را آموزش میدهیم:

در این حالت هم روند بهینه سازی شدیدا با کاهش سرعت همراه شده و واضحا مینیمم سازی مقدار تابع اتلاف به زمان بسیار زیادی نیاز دارد. پس 128 هم گزینه مناسبی برای این ابرپارامتر نیست.

شکل 16 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با اندازه دسته 128

تعداد واحدهای لایه پنهان:

ابتدا با 8 پرسپترون در لایه مخفی شبکه را آموزش میدهیم:

با توجه به نمودار ها، کاهش تعداد واحدهای لایه پنهان به 8 باعث افزایش دقت مدل روی داده های آموزشی و اعتبارسنجی نسبت به مدل پایه شده چون تعداد پارامترهای مدل بسیار کمتر($^1/_4$ برابر) شده است، مدل بهتر آموزش دیده است. بدیهی است که زمان مورد نیاز برای آموزش شبکه نیز به طور قابل توجهی کمتر شده است.

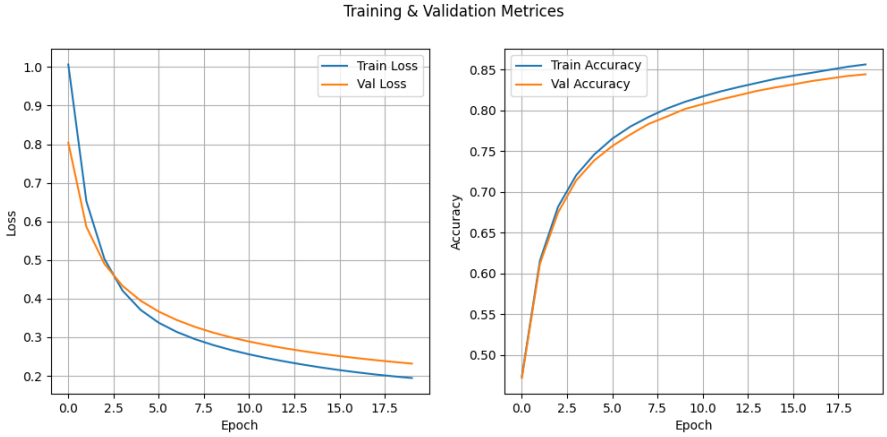
$$(784 + 1) \times 8 + \ (8 + 1) \times 10 = 6370 \ parameters$$

شکل 17 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با 8 پرسپترون در لایه مخفی

حالا با 128 پرسپترون در لایه مخفی شبکه را آموزش میدهیم:

با افزایش تعداد واحدهای لایه پنهان به 128 ، دقت مدل روی داده های آموزشی و اعتبارسنجی نسبت به مدل پایه بیشتر شده چون تعداد پارامترهای مدل تقریبا 4 برابر شده است و این موضوع باعث بیشتر شدن قدرت شبکه برای پیداکردن روندها و ارتباطات بین ورودی ها شده است. اما باید توجه داشته باشید که پیچیده کردن مدل (افزایش تعداد پارامترها = وزن های شبکه عصبی) ریسک بیش برازش را زیاد میکند هرچند اینجا این اتفاق نیفتاده اما این موضوع محتمل است. بدیهی است که زمان مورد نیاز برای آموزش شبکه نیز به طور قابل توجهی بیشتر شده است.

$$(784 + 1) \times 128 + \ (128 + 1) \times 10 = 101770 \ parameters$$



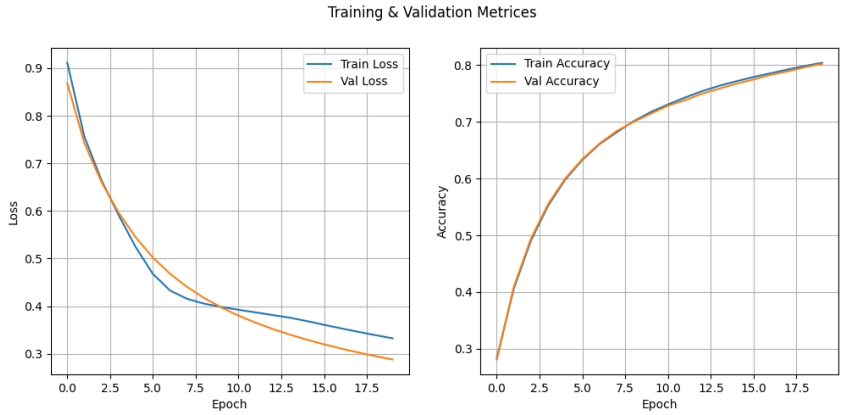
شکل 18 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با 128 پرسپترون در لایه مخفی

تعداد لایه های پنهان:

با اضافه کردن یک لایه پنهان دیگر، دقت مدل روی داده های آموزشی و اعتبارسنجی نسبت به مدل پایه بیشتر شده چون تعداد پارامترهای مدل کمی بیشتر شده است (حدودا 1000) و این موضوع باعث بیشتر شدن قدرت شبکه برای پیداکردن روندها و ارتباطات بین ورودی ها شده است. اما باید توجه داشته باشید که پیچیده کردن مدل (افزایش تعداد پارامترها = وزن های شبکه عصبی) ریسک بیش برازش را زیاد میکند هرچند اینجا این اتفاق نیفتاده اما این موضوع محتمل است. بدیهی است که زمان مورد نیاز برای آموزش شبکه نیز به طور قابل توجهی بیشتر شده است.

$$Base\ Model: (784+1)\times 32+\ (32+1)\times 10=25450\ parameters$$

$$NN\ with\ 2\ Hidden\ Layer: (784+1)\times 32+\ (32+1)\times 32+\ (32+1)\times 10=26506\ parameters$$



شکل 19 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با دو لایه مخفی

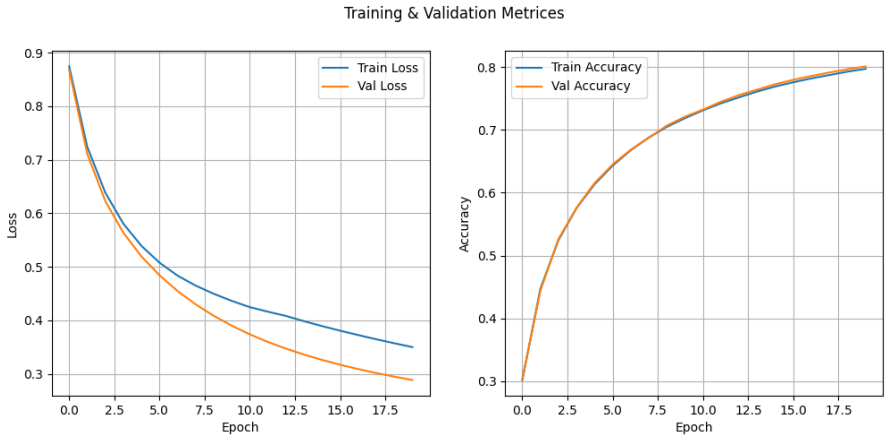
روش بهینه سازی:

با استفاده از روش کاهش گرادیان تصادفی با گشتاور قابل انتظار است که فرایند بهینه سازی نسبت به روش کاهش گرادیان تصادفی ساده بهتر شود چون با درنظر گرفتن ضریبی از بردار گرادیان مرحله قبل در به روز رسانی وزن های شبکه، نویز را کاهش میدهد. بنابراین مسیر حرکت به سمت نقطه مینیمم تابع هزینه نویزی و تحت تاثیر داده های پرت نخواهد بود و احتمال گیر نیفتادن در مینیمم لوکال و همگرا شدن به سمت مینیمم سراسری بیشتر خواهد شد.

منظم ساز L2 :

در ابتدا l2_lambda را برابر 0.01 قرار میدهیم و شبکه را آموزش میدهیم:

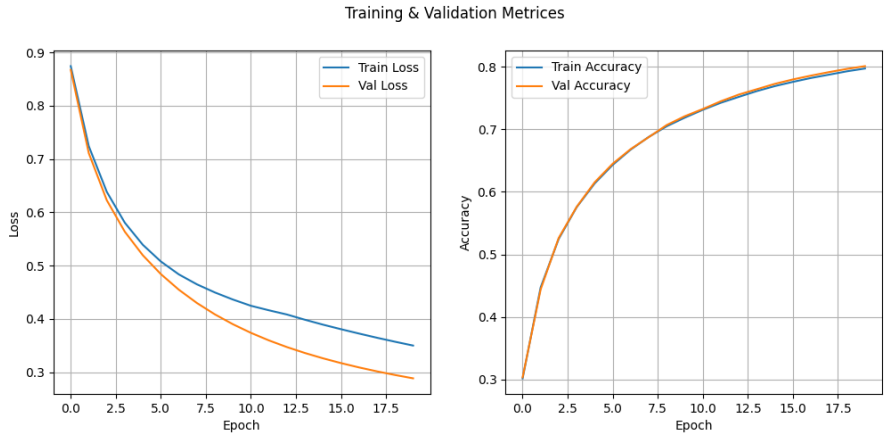
طبق نمودارهای زیر، از نظر دقت، بهترین مدل آموزش داده شده نیست اما منطبق بودن منحنی دقت داده های آموزشی و منحنی دقت داده های اعتبارسنجی نشان دهنده تاثیر اضافه کردن ترم منظم ساز برای جلوگیری از بیش برازش است.



شکل 20 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با نرخ 0.01 منظم ساز L2

حالا l2_lambda را برابر 0.05 قرار میدهیم و شبکه را آموزش میدهیم:

نتیجه کاملا مشابه l2_lambda = 0.01 است.

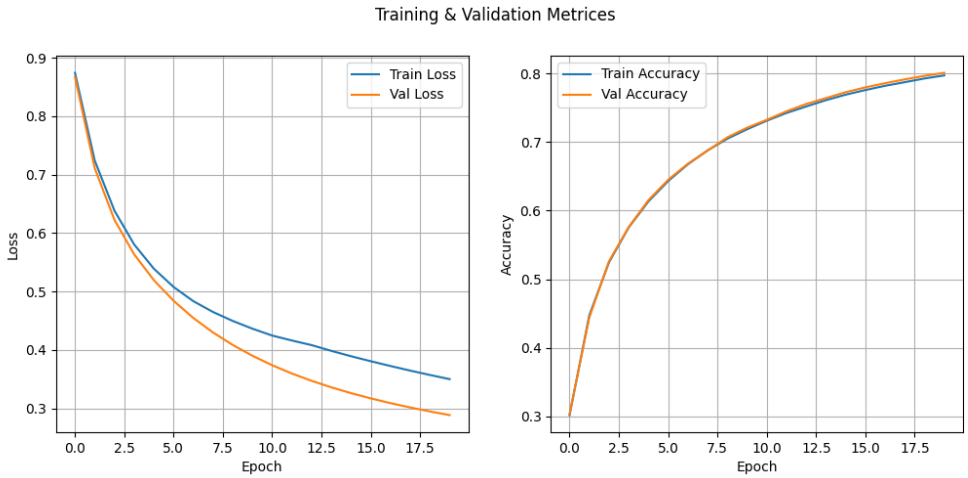


شکل 21 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با نرخ 0.05 منظم ساز L2

منظم ساز L1:

l1_lambda را برابر 0.1 قرار میدهیم و شبکه را آموزش میدهیم:

نتیجه فرقی با دو حالت قبل (استفاده از منظم ساز L2) ندارد!



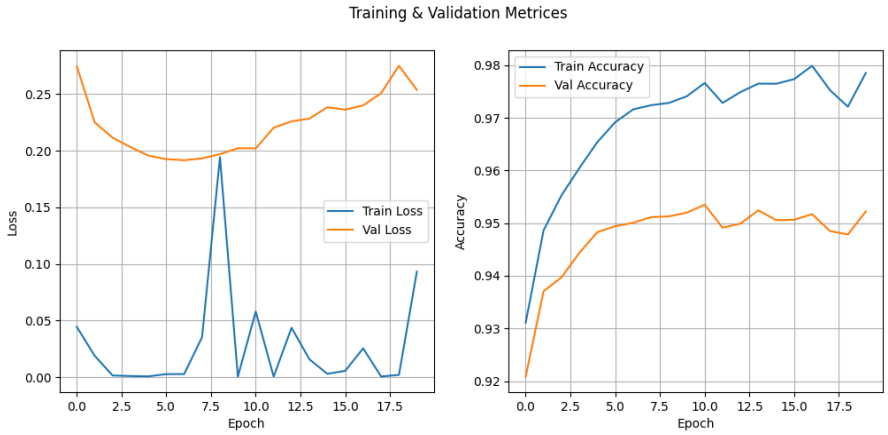
شکل 22 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل با نرخ **0.1** منظم ساز **L1**

جمع بندی نتایج و آزمایش بهترین مدل

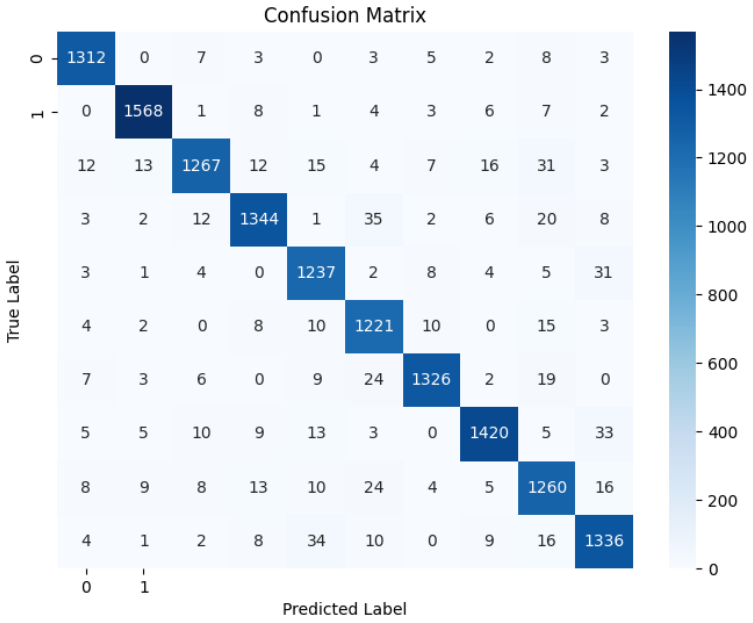
بعد از انجام سعی و خطا روی مقدار هر ابرپارامتر و بررسی نتایج و تحلیل هایی که بالاتر ارائه شد، جدول زیر را به عنوان بهترین configuration برای ابرپارامترهای مدل بدست آوردم و بعد از آموزش مدل نتیجه آن روی داده های آزمایشی را در قالب ماتریس سردرگمی میبینیم.

جدول 1 – بهترین مدل

نرخ یادگیری	اندازه دسته	لایه پنهان	تابع فعالسازی	تابع اتلاف	ایپاک	نرخ منظم ساز L1	نرخ منظم ساز L2	روش بهینه سازی
0.1	8	32	relu, softmax	CrossEntropy	20	0	0.1	SGD



شکل 23 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در بهترین مدل

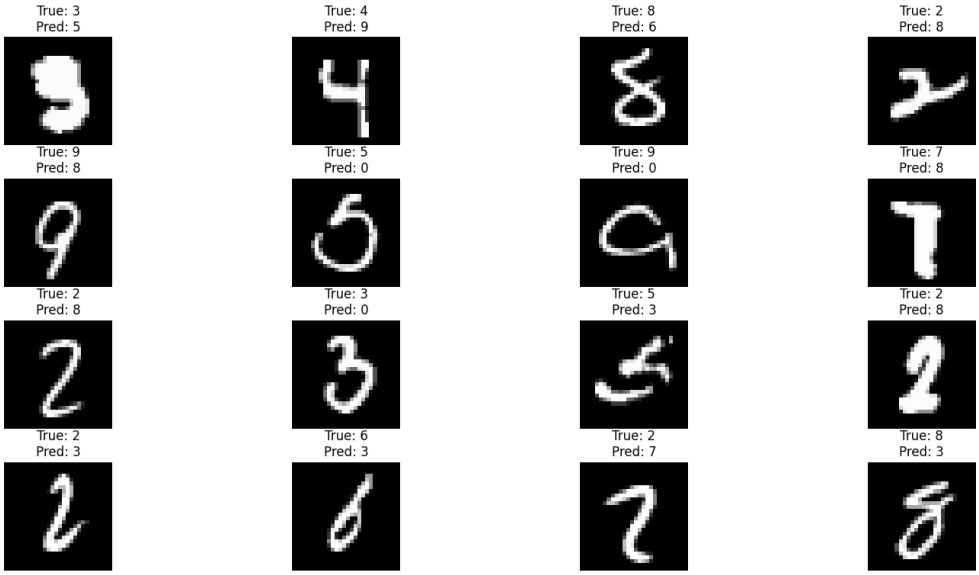


شکل 24 – ماتریس سردرگمی بهترین مدل برای تصاویر آزمایشی

تحلیل خطا

تحلیل ماتریس سردرگمی:

- برای مدل تشخیص عدد یک ساده تر از بقیه بوده است که به دلیل پیچیده نبودن فرم آن اتفاق افتاده.
- تفکیک کردن عدد 4 و 9 از هر دو عدد دیگری سخت تر بوده جایی که بیشترین خطای مدل در مقایسه های دوتایی رخ داده؛ 31بار رقم 4 را به اشتباه 9 تشخیص داده و 34بار رقم 9 را به اشتباه 4 تشخیص داده است.
- شبکه 35بار رقم 3 را به اشتباه 5 تشخیص داده که نشان دهنده مشکل شبکه های عصبی پیشخور در پیدا کردن تفاوت های محلی است؛ نکته ای که نقطه قوت شبکه های کانولوشنی محسوب میشود.



شکل 25 – 16 نمونه از تصاویری که خطا تشخیص داده شده اند

1-3 مواجهه با داده جدید و انتقال یادگیری

پیش پردازش داده

همه کار های لازم برای پیش پردازش دیتاست SVHN و آماده کردن آن برای استفاده در شبکه عصبی تعریف شده در تابع preprocess_svhn که در فایل دیتالودر قابل دسترسی و فراخوانی است صورت میگیرد و شما برای لود کردن تصاویر صرفا کافیست تابع get_svhn را فراخوانی کنید تا در صورت نبودن دیتاست در دایرکتوری اعلام شده، آن را دانلود کند و با انجام مراحل پیش پردازش داده ها یعنی نرمال سازی تصاویر، تبدیل آن ها به تصاویر تک کاناله و تغییر ابعاد آن و فلت سازی تصاویر آنها را برای استفاده در فاز آموزش یا تست شبکه عصبی آماده کند.

```python
def get_svhn(data_dir="data/svhn"):
    """
    Download, load, and preprocess the SVHN dataset.
    """
    download_svhn(data_dir)
    X_train, Y_train, X_test, Y_test = load_svhn(data_dir)
    X_train, Y_train, X_test, Y_test = preprocess_svhn(X_train, Y_train, X_test, Y_test)

    # print(f"X_train shape: {X_train.shape}, X_test shape: {X_test.shape}")
    # print(f"Y_train shape: {Y_train.shape}, Y_test shape: {Y_test.shape}")
    return X_train.T, Y_train, X_test.T, Y_test
```

شکل 26 – تابع **get_svhn**

```python
def preprocess_svhn(X_train, Y_train, X_test, Y_test, resize_shape=(28, 28)):
    """
    Preprocess the SVHN dataset.
    """
    # Resize images to 28x28, convert to grayscale, and flatten them
    def resize_to_grayscale_and_flatten(images, resize_shape):
        resized_images = []
        for img in images:
            img_pil = Image.fromarray((img * 255).astype("uint8"))

            img_resized = img_pil.resize(resize_shape, Image.BILINEAR)
            img_grayscale = img_resized.convert("L")
            img_flattened = np.array(img_grayscale).flatten()

            resized_images.append(img_flattened)
        return np.array(resized_images)

    # Normalize pixel values to [0, 1]
    X_train = X_train.astype("float32") / 255.0
    X_test = X_test.astype("float32") / 255.0

    # Resize, convert to grayscale, and flatten images
    X_train = resize_to_grayscale_and_flatten(X_train, resize_shape)
    X_test = resize_to_grayscale_and_flatten(X_test, resize_shape)

    Y_train = np.eye(10)[Y_train].T.astype(np.float32)  # Convert labels to one-hot (10, samples)
    Y_test = np.eye(10)[Y_test].T.astype(np.float32)

    # Split training data into training and validation sets
    # X_train, X_val, Y_train, Y_val = train_test_split(X_train, Y_train, test_size=val_size, random_state=42)

    return X_train, Y_train, X_test, Y_test
```

شکل 27 – تابع **preprocess_svhn**

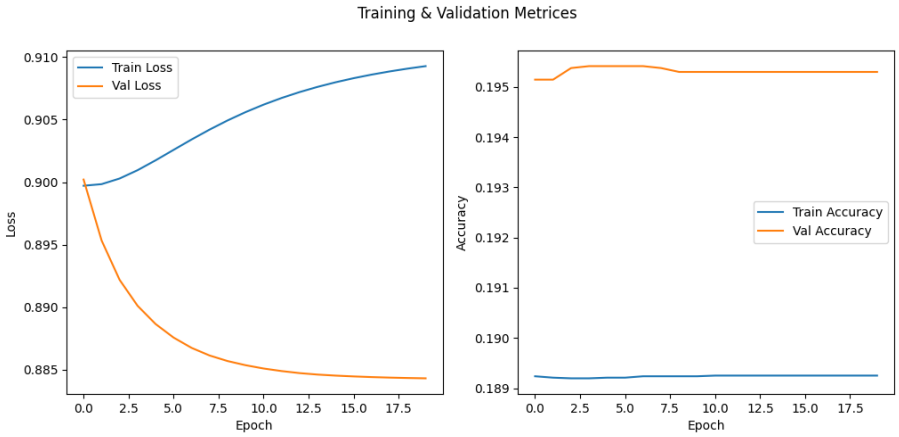
شکل 28 - تصاویر تصادفی از دیتاست **SVHN**

آموزش و تست شبکه با دیتاست جدید (با وزن دهی اولیه رندوم):

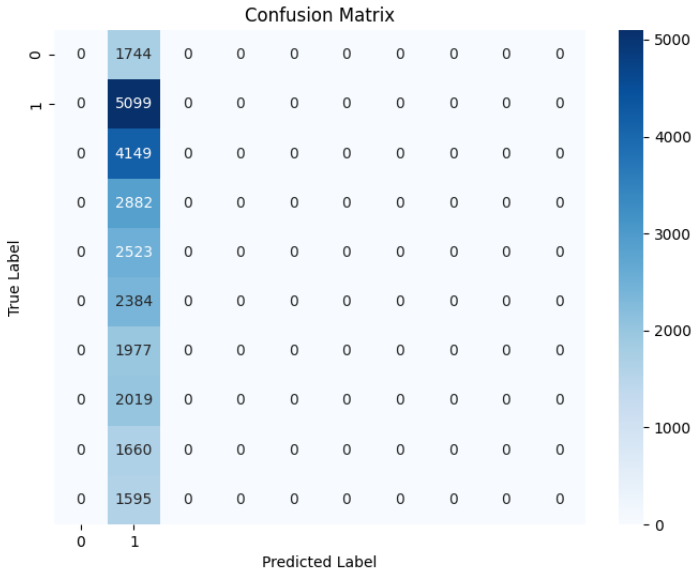
عملکرد مدل فاجعه بود! بطوریکه همه ورودی ها رو یک تشخیص میداد!! ولی به لطف این 19.59% دقت داشت که همین موضوع نشان دهنده بالانس نبودن دیتاست جدید است. نکته عجیب عدم بهبود مدل (کم شدن تابع اتلاف) در طی فرایند آموزش بود.

فارغ از عملکرد فاجعه بار مدل توسعه داده شده، بطور کلی نباید توقع زیادی از شبکه پرسپترون چندلایه در برخورد با این دیتاست داشت که پیچیدگی بسیار بیشتری نسبت به دیتاست MNIST دارد از جمله:

1. وجود ارقام در طرفین رقم اصلی که باعث فریب مدل میشود.
2. متغیر بودن رزولوشن و کیفیت تصاویر
3. متغیر بودن رنگ پس زمینه تصاویر
4. تفاوت های ساختاری زیاد بین فونت های مختلف موجود



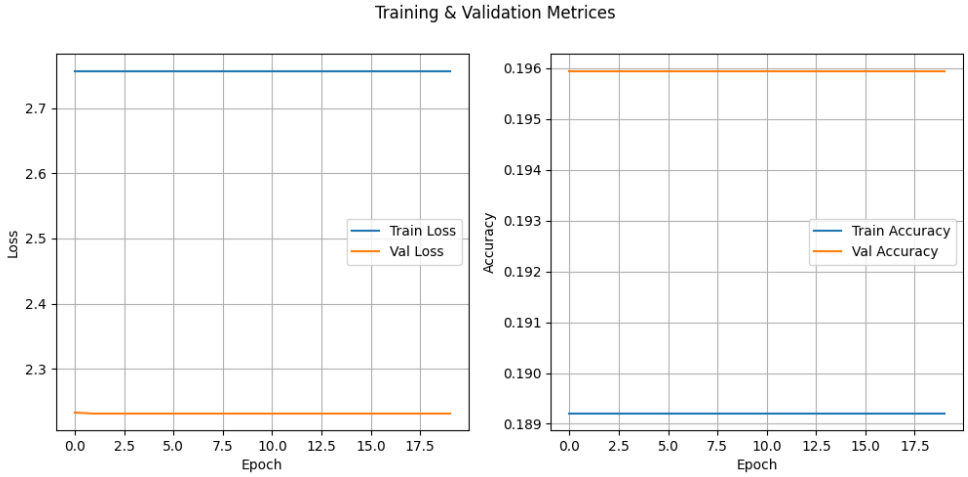
شکل 29 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی در مدل از ابتدا آموزش دیده با **SVHN**



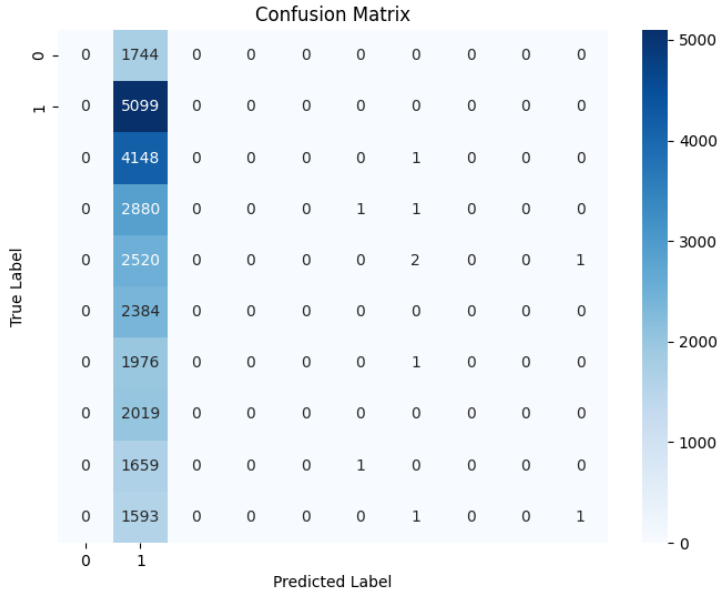
شکل 30 – ماتریس سردرگمی مدل از ابتدا آموزش دیده با **SVHN**

آموزش و تست شبکه با دیتاست جدید (با وزن دهی اولیه حاصل از آموزش با MNIST):

تغییر آنچنانی ایجاد نشد صرفا دقت اعتبارسنجی از 19.5874% به 19.5951% رسید!



شکل 31 – نمودار های اتلاف و دقت برای داده های آموزش و اعتبارسنجی با دیتاست **SVHN** (با لود کردن مقادیر اولیه وزنها از بخش قبل)



شکل 32 – ماتریس سردرگمی مدل روی دیتاست **SVHN** (با لود کردن مقادیر اولیه وزنها از بخش قبل)

2-تخمین امید به زندگی با استفاده از شبکه های عصبی

پیش پردازش و تقسیم داده ها

با فراخوانی تابع load_life_expectancy_dataset :

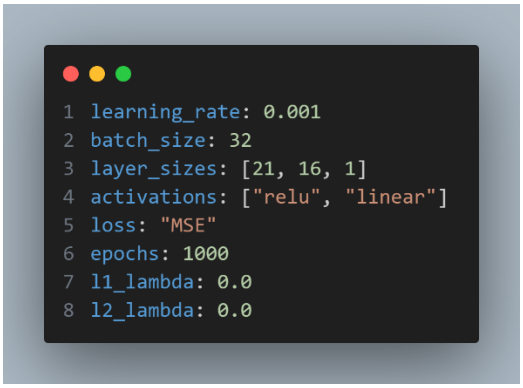
1. دیتاست از فایل موجود در دایرکتوری لود شده،
2. مقادیر گمشده جاگذاری میشود (میانگین برای متغیرهای عددی و مد برای متغیرهای کیفی)
3. متغیرهای عددی استاندارد سازی میشوند
4. متغیر های کیفی به اعداد صحیح انکود میشوند تا امکان ورودی دادن انها به شبکه عصبی فراهم شود.
5. داده های مربوط به قبل از سال 2010 به عنوان داده های آموزشی و داده های بعد از آن به عنوان داده آزمایشی برگردانده میشود.

```python
def load_life_expectancy_dataset(path="data/life expectancy/Life Expectancy Data.csv"):
    data = pd.read_csv(path)

    print("Missing values:", data.isnull().sum())

    categorical_columns = data.select_dtypes(include=['object', 'category']).columns
    numerical_columns = data.select_dtypes(include=[np.number]).columns

    data[numerical_columns] = data[numerical_columns].fillna(data[numerical_columns].mean())

    for col in categorical_columns:
        data[col] = data[col].fillna(data[col].mode()[0])

    print("After filling missing values:", data.isnull().sum())


    label_encoder = LabelEncoder()
    for col in categorical_columns:
        data[col] = label_encoder.fit_transform(data[col])

    print(np.unique(data["Year"]))

    train_data = data[data["Year"] <= 2010]
    test_data = data[data["Year"] > 2010]

    scaler = StandardScaler()
    train_data[numerical_columns] = scaler.fit_transform(train_data[numerical_columns])
    test_data[numerical_columns] = scaler.fit_transform(test_data[numerical_columns])

    X_train = train_data.drop(columns=["Life expectancy "]).values
    y_train = train_data["Life expectancy "].values
    X_test = test_data.drop(columns=["Life expectancy "]).values
    y_test = test_data["Life expectancy "].values

    y_train = y_train.reshape(1, -1)  # Reshape y_train to (1, num_samples)
    y_test = y_test.reshape(1, -1)

    print(f"X_train shape: {X_train.shape}, y_train shape: {y_train.shape}")
    print(f"X_test shape: {X_test.shape}, y_test shape: {y_test.shape}")

    return X_train.T, y_train, X_test.T, y_test

```

شکل 33 - تابع **load_life_expectancy_dataset**

ساخت و آموزش مدل

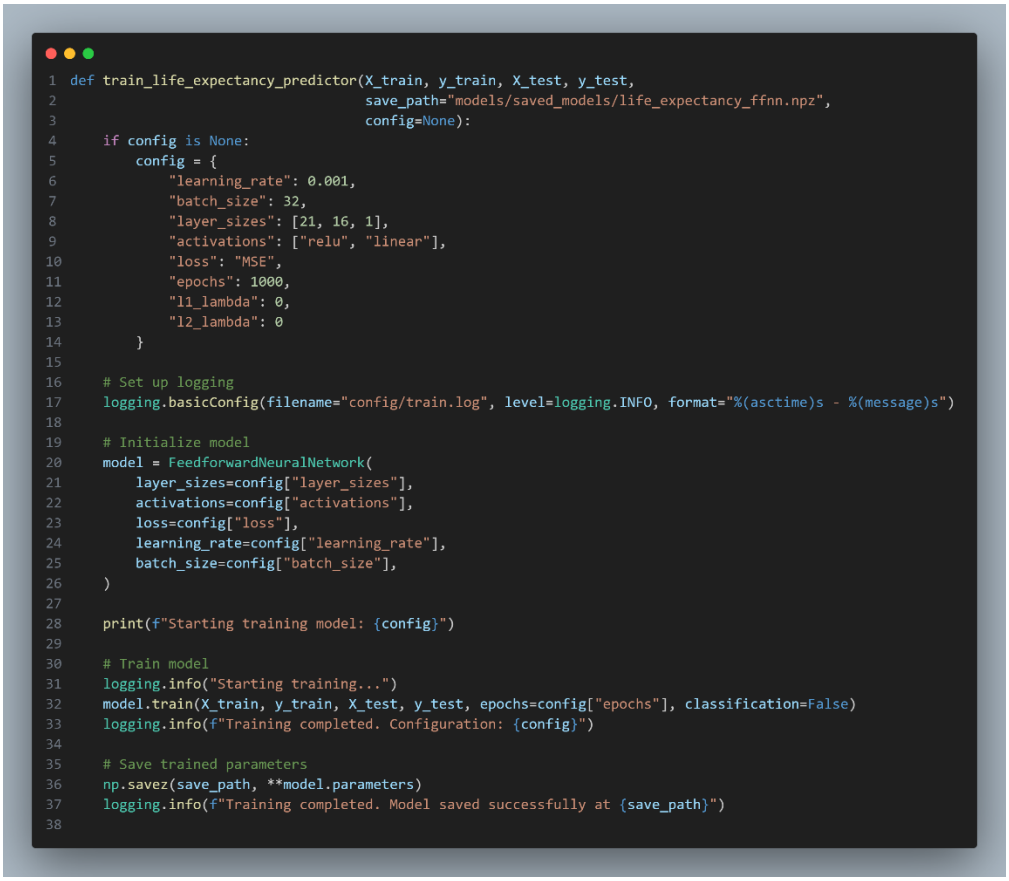
طبق توضیحات موجود در فایل تمرین، کانفیگ مدل بصورت زیر خواهد بود که در فایل q2_config.yaml ذخیره شده است:



```yaml
learning_rate: 0.001
batch_size: 32
layer_sizes: [21, 16, 1]
activations: ["relu", "linear"]
loss: "MSE"
epochs: 1000
l1_lambda: 0.0
l2_lambda: 0.0
```

شکل 34 - فایل **q2_config.yaml**

با فراخوانی تابع train_life_expectancy_predictor که در فایل train.py قرار دارد مدل آموزش میبیند و وزنهای مدل ذخیره میشود تا در زمان ارزیابی مدل بتوانیم از آنجا لود کنیم.

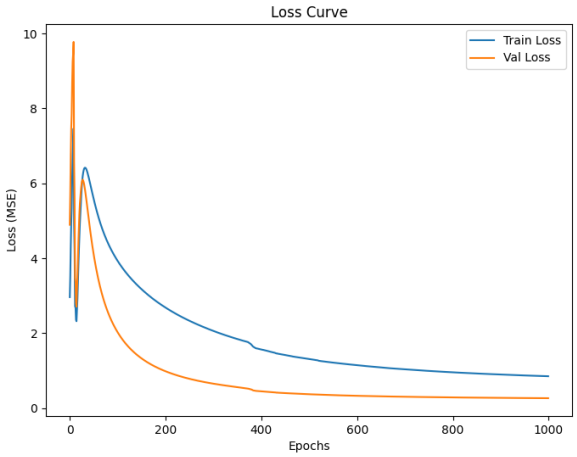


```python
def train_life_expectancy_predictor(X_train, y_train, X_test, y_test,
                                    save_path="models/saved_models/life_expectancy_ffnn.npz",
                                    config=None):
    if config is None:
        config = {
            "learning_rate": 0.001,
            "batch_size": 32,
            "layer_sizes": [21, 16, 1],
            "activations": ["relu", "linear"],
            "loss": "MSE",
            "epochs": 1000,
            "l1_lambda": 0,
            "l2_lambda": 0
        }

    # Set up logging
    logging.basicConfig(filename="config/train.log", level=logging.INFO, format="%(asctime)s - %(message)s")

    # Initialize model
    model = FeedforwardNeuralNetwork(
        layer_sizes=config["layer_sizes"],
        activations=config["activations"],
        loss=config["loss"],
        learning_rate=config["learning_rate"],
        batch_size=config["batch_size"],
    )

    print(f"Starting training model: {config}")

    # Train model
    logging.info("Starting training...")
    model.train(X_train, y_train, X_test, y_test, epochs=config["epochs"], classification=False)
    logging.info(f"Training completed. Configuration: {config}")

    # Save trained parameters
    np.savez(save_path, **model.parameters)
    logging.info(f"Training completed. Model saved successfully at {save_path}")

```

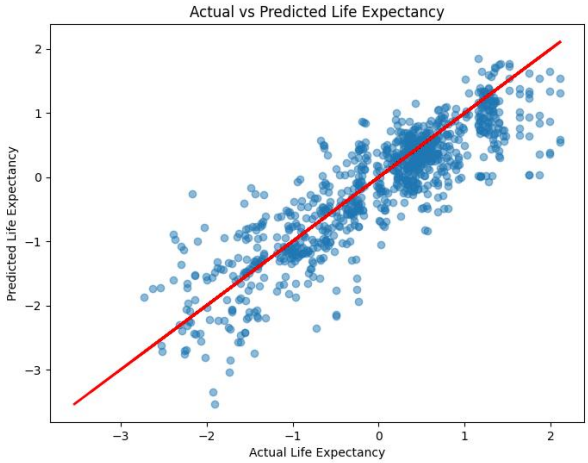
شکل 35 – تابع **train_life_expectancy_predictor**

نمایش منحنی خطا و نتایج شبکه

مقدار میانگین مربعات خطا بر روی مجموعه داده تست برابر 0.2626 است. همچنین منحنی خطا در طی فرایند اموزش بصورت زیر بوده است:



شکل 36 – نمودار میانگین مربعات خطا برای داده های آموزش و تست دیتاست امید به زندگی



شکل 37 – مقادیر تخمین زده شده و مقادیر واقعی امید به زندگی داده های تست

مراجع