تمرین سری دوم درس هوش محاسباتی

کورش تقی پور پاسدار 400521207

۱۸ آبان ۱۴۰۳

فهرست مطالب

۱ سوال اول

۱.۱ بخش a

در شبکه‌های MLP که توابع فعالسازی ندارند، ساختار شبکه بصورت چند لایه خطی متصل به هم می‌باشد که از نظر ریاضی اثبات می‌شود چند لایه خطی پشت سر هم، مانند یک لایه خطی عمل می‌کنند و به عبارت دیگر، یک شبکه با چند لایه خطی پشت سر هم و بدون تابع فعالسازی، مانند یک شبکه با تنها یک لایه خطی عمل می‌کند و تنها قادر به جداسازی خطی می‌باشد. درحالی که وجود توابع فعالسازی (مانند ReLU, tanh, sigmoid) سبب غیرخطی شدن تابع شبکه شده و شبکه را پیچیده‌تر می‌کند.

۲.۱ بخش b

هر تابع غیرخطی نمی‌تواند به عنوان تابع فعالسازی مورد استفاده قرار بگیرد. در انتخاب تابع فعالسازی باید به نکاتی توجه داشته باشیم:

مشتق پذیری : درصورتی‌که تابع فعالسازی مشتق‌پذیر نباشد نمی‌توان از آن در روش‌های بهینه‌سازی مبتنی بر Back Propagation استفاده کرد.

محدودیت دامنه : برخی توابع مانندSigmoid یا tanh خروجی را به یک بازه محدود می‌کنند که به پایداری شبکه کمک می‌کند.

پیشگیری از محو شدن گرادیان : برخی توابع فعالسازی مانند ReLU بدلیل ویژگی‌های خود از مشکل محو شدن گرادیان جلوگیری کرده و به یادگیری کمک می‌کنند.

۳.۱ بخش c

۱.۳.۱ مزایا

افزودن لایه‌های بیشتر سبب پیچیده‌تر شدن مدل و در نتیجه، توانایی بیشتر آن در تشخیص الگوهای پیچیده‌تر و استفاده آن در مسائل دشوارتر باشد. همچنین در ابتدا، یکی از راهکارهای پیچیده‌تر کردن مدل، عریض‌تر کردن لایه‌ها بود. افزایش لایه‌ها بر این راهکار غلبه کرده و سبب شده تا همزمان با افزایش کمتر مقدار نورون‌ها نسبت به راهکار دیگر و کاهش هزینه‌های محاسباتی و حافظه‌ای، مدلی با قدرت بهتر ارائه دهد.

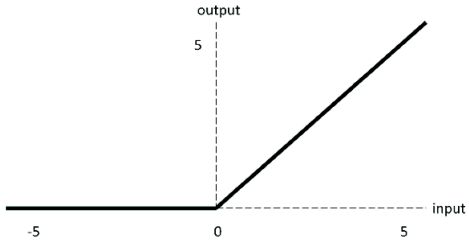
۲.۳.۱ معایب

افزودن لایه‌های بیشتر سبب افزایش تعداد نورون‌ها و همچنین افزایش هزینه محاسباتی و حافظه‌ای مدل می‌شود. از طرفی، مدل پیچیده‌تر شده و نیاز به مجموعه داده بیشتری برای آموزش بهتر است. همچنین با پیچیده‌تر شدن مدل، احتمال overfit مدل افزایش می‌یابد و نیاز به ارائه راهکار برای جلوگیری از آن هستیم. همچنین بدلیل وقوع Gradient Vanishing، احتمال آموزش دیرتر و کندتر نورون‌های لایه‌های ابتدایی‌تر هستیم. همچنین تنظیم پارامترهای شبکه‌های عمیق‌تر پیچیره‌تر بوده و نیازمند آزمایش و تجربه بیشتری هستیم.

۴.۱ بخش d

۱.۴.۱ تابع ReLU

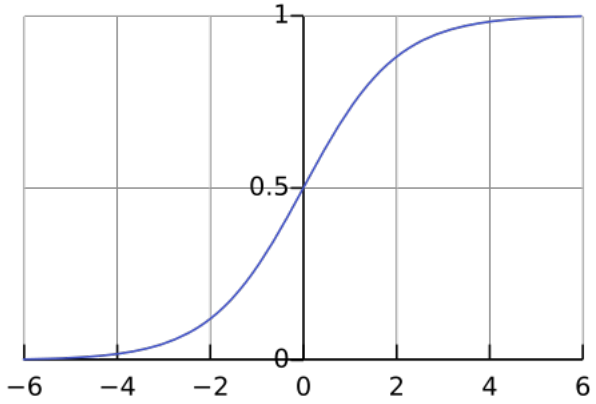
فرمول این تابع بصورت $ReLU(x) = max(0, x)$ می‌باشد و نمودار آن بصورت زیر است.



از مزایای آن می‌توان به محاسبات ساده و سریع آن اشاره کرد. همچنین به دلیل مشتق ثابت در نواحی مثبت، از ناپدید شدن گرادیان در لایه‌های عمیق‌تر پیشگیری می‌کند و سبب یادگیری بهتر مدل می‌شود. از معایب آن می‌توان به مشتق صفر در نواحی منفی اشاره کرد و همچنین اینکه خروجی آن بدون محدودیت بوده و می‌تواند مقادیر بسیار بزرگ را خروجی دهد.

۲.۴.۱ تابع Sigmoid

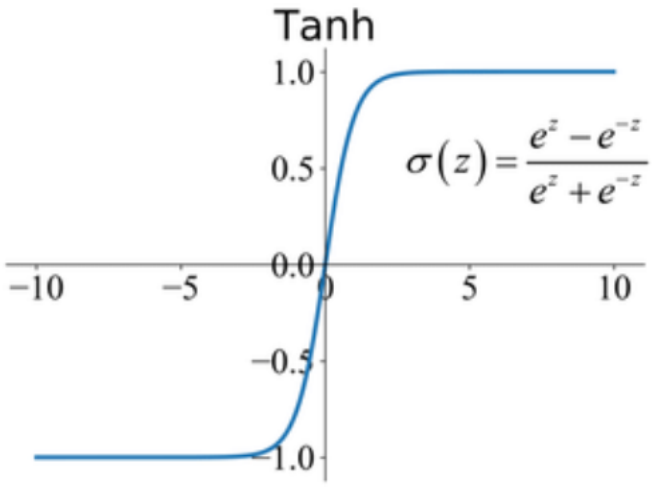
فرمول این تابع بصورت $Sigmoid(x) = \frac{1}{1+e^{-x}}$ می‌باشد و نمودار آن بصورت زیر می‌باشد.



از مزایای آن می‌توان به خروجی محدود آن (در بازه ۰ و ۱) اشاره کرد و همچنین برای دسته‌بندی دوتایی (Binary Classification) مناسب می‌باشد.از معایب آن نیز به محاسبات پیچیده‌تر آن نسبت به ReLU ، کوچک شدن گرادیان در نواحی اشباع اشاره کرد.

۳.۴.۱ تابع Tanh

فرمول این تابع بصورت $f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$ می‌باشد و نمودار آن بصورت زیر می‌باشد.



از مزایای آن می‌توان به محدود بودن خروجی‌ها اشاره کرد (بین ۱ و -۱) و همچنین خروجی ها بصورت متقارن حول صفر تعیین می‌شوند. از معایب آن نیز مشابه Sigmoid به پیچیده‌تر بودن محاسبات نسبت به ReLU و همچنین ناپدید شدن گرادیان در نواحی اشباع اشاره کرد.

۴.۴.۱ تابع Softmax

فرمول این تابع بصورت $Softmax(x) = \frac{e_i^x}{\sum_{i=1}^{n} e_i^x}$ می‌باشد. از مزایای آن می‌توان به محدود کردن خروجی به بازه ۰ تا ۱ اشاره کرد و همچنین برای دسته‌بندی چندکلاسه (Multiclass Classification) مناسب است. از معایب آن نیز محاسبات بیشتر نسبت به سایر توابع و همچنین حساسیت نسبت به مقادیر بزرگ می‌توان گفت.

۲ سوال دوم

تابع ReLU بدلیل صفر کردن مقادیر منفی می‌تواند سبب از دست رفتن اطلاعات مهمی برای بروزرسانی شبکه شود و ممکن است که به گرادیان‌های مرده سبب شود که در آن نورون‌ها دیگر بروزرسانی نشده و فرایند آموزش متوقف می‌گردد. همچنین بدلیل استفاده همزمان از ReLU و Sigmoid می‌تواند شبکه را به شدت غیرخطی کرده و همگرایی را دشوار سازد. همچنین آستانه ۵.۰ می‌تواند سبب تصمیم‌گیری های نادرست شود و به عنوان مثال، هر دو خروجی ۰۱.۰ و ۴۹.۰ را به عنوان کلاس صفر خروجی دهد درحالی‌که تفاوت زیادی بین آنها وجود دارد.

۳ سوال سوم

در ابتدا فرمول‌های هریک از نورون‌ها بصورت زیر نوشته می‌شود:

$$h_j = \sum_{i=1}^{n} x_i w_{ij} \quad (۱)$$

$$y_j = \sum_{i=1}^{m} h_i w_{ij} \quad (۲)$$

$$Sigmoid = \frac{1}{1+e^{-x}} \quad (۳)$$

در ابتدا forward feed حساب می‌شود.

$$input1: \quad h_1 = Sigmoid(3 \times 6 + 1 \times -6) = 0.99 \quad (۴)$$

$$h_2 = Sigmoid(3 \times -3 + 1 \times 5) = 0.01 \quad (۵)$$

$$y_1 = Sigmoid(h_1 \times 1 + h_2 \times 0.25) = 0.73 \quad (۶)$$

$$y_2 = Sigmoid(h_1 \times -2 + h_2 \times 2) = 0.27 \quad (۷)$$

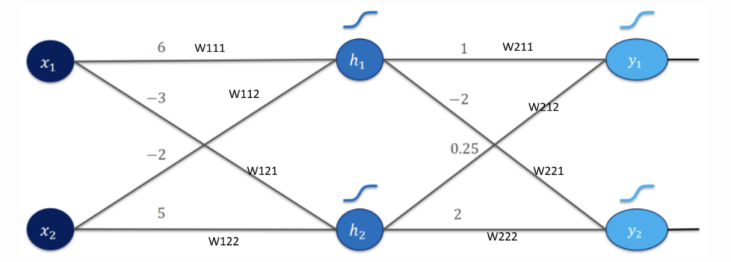
$$input2: \quad h_1 = Sigmoid(-1 \times 6 + 4 \times -2) = 0.01 \quad (۸)$$

$$h_2 = Sigmoid(-1 \times -3 + 4 \times 5) = 0.99 \quad (۹)$$

$$y_1 = Sigmoid(h_1 \times 1 + h_2 \times 0.25) = 0.56 \quad (۱۰)$$

$$y_2 = Sigmoid(h_1 \times -2 + h_2 \times 2) = 0.88 \quad (۱۱)$$

حال به محاسبه Backpropagation می‌پردازیم. در ابتدا، وزن‌ها را مطابق با شکل زیر نام‌گذاری می‌کنیم.



حال مراحل محاسبه آموزش وزن‌ها به صورت زیر است. همچنین نرخ آموزش بدلیل عدم ذکر در صورت سوال برابر با 0.1 درنظر گرفته می‌شود.
در ابتدا حالت کلی فرمول‌ها بیان می‌شود سپس به محاسبه گرادیان‌ها می‌پردازیم.

$$\Delta W_{2**} = lr \times Sigmoid \times (1 - Sigmoid) \times (d - y) \times h$$

$$\Delta W_{1**} = lr \times ((d_1 - y_1) \times Sigmoid \times (1 - Sigmoid) \times W_{2**} + (d_0 - y_0) \times Sigmoid \times (1 - Sigmoid) \times W_{2**}) \times h$$

$$
\begin{aligned}
input1: \Delta W_{211} &= 0.1 \times 0.73 \times 0.27 \times (1-0.73) \times 0.99 = 5.26 \times 10^{-3} \\
\Delta W_{212} &= 0.1 \times 0.73 \times 0.27 \times (1-0.73) \times 0.01 = 5.32 \times 10^{-5} \\
\Delta W_{221} &= 0.1 \times 0.27 \times 0.73 \times (0-0.27) \times 0.99 = -5.26 \times 10^{-3} \\
\Delta W_{222} &= 0.1 \times 0.27 \times 0.73 \times (0-0.27) \times 0.01 = -5.32 \times 10^{-5} \\
\Delta W_{111} &= 0.1 \times (((1-0.73) \times 0.73 \times 0.27 \times 1 + (0-0.27) \times 0.27 \times 0.73 \times -2) \\
&\times \ 0.99 \times 0.01 \times 3) = 4.74 \times 10^{-4} \\
\Delta W_{112} &= 0.1 \times (((1-0.73) \times 0.73 \times 0.27 \times 1 + (0-0.27) \times 0.27 \times 0.73 \times -2) \\
&\times \ 0.99 \times 0.01 \times 1) = 1.58 \times 10^{-4} \\
\Delta W_{121} &= 0.1 \times (((1-0.73) \times 0.73 \times 0.27 \times 0.25 + (0-0.27) \times 0.27 \times 0.73 \times 2) \\
&\times \ 0.01 \times 0.99 \times 3) = -2.76 \times 10^{-4} \\
\Delta W_{122} &= 0.1 \times (((1-0.73) \times 0.73 \times 0.27 \times 0.25 + (0-0.27) \times 0.27 \times 0.73 \times 2) \\
&\times \ 0.01 \times 0.99 \times 1) = 9.21 \times 10^{-5} \\
input2: \Delta W_{211} &= 0.1 \times 0.56 \times 0.44 \times (0-0.56) \times 0.01 = -1.37 \times 10^{-4} \\
\Delta W_{212} &= 0.1 \times 0.56 \times 0.44 \times (0-0.56) \times 0.99 = -1.36 \times 10^{-2} \\
\Delta W_{221} &= 0.1 \times 0.88 \times 0.12 \times (1-0.88) \times 0.01 = 1.26 \times 10^{-5} \\
\Delta W_{222} &= 0.1 \times 0.88 \times 0.12 \times (1-0.88) \times 0.99 = 1.25 \times 10^{-3} \\
\Delta W_{111} &= 0.1 \times (((0-0.56) \times 0.56 \times 0.44 \times 1 + (1-0.88) \times 0.88 \times 0.12 \times -2) \\
&\times \ 0.99 \times 0.01 \times -1) = 1.61 \times 10^{-4} \\
\Delta W_{112} &= 0.1 \times (((0-0.56) \times 0.56 \times 0.44 \times 1 + (1-0.88) \times 0.88 \times 0.12 \times -2) \\
&\times \ 0.99 \times 0.01 \times 4) = -6.46 \times 10^{-4} \\
\Delta W_{121} &= 0.1 \times (((0-0.56) \times 0.56 \times 0.44 \times 0.25 + (1-0.88) \times 0.88 \times 0.12 \times 2) \\
&\times \ 0.99 \times 0.01 \times -1) = 9.06 \times 10^{-6} \\
\Delta W_{122} &= 0.1 \times (((0-0.56) \times 0.56 \times 0.44 \times 0.25 + (1-0.88) \times 0.88 \times 0.12 \times 2) \\
&\times \ 0.99 \times 0.01 \times 4) = -3.62 \times 10^{-5}
\end{aligned}
$$

حال به آپدیت وزن‌ها درحالت چندتایی (Batch Mode) می‌پردازیم. فرمول کلی آپدیت وزن‌ها بصورت

$$W = W + \sum_x \Delta W^x$$

است. وزن‌های آپدیت شده را در ادامه خواهیم داشت.

$$
\begin{aligned}
W_{111} &= 6 + (4.74 \times 10^{-4} + 1.61 \times 10^{-4}) = 6.000635 \\
W_{112} &= -2 + (1.58 \times 10^{-4} + -6.46 \times 10^{-4}) = -2.000488 \\
W_{121} &= -3 + (-2.76 \times 10^{-4} + 9.06 \times 10^{-6}) = -3.00026694 \\
W_{122} &= 5 + (9.21 \times 10^{-5} + -3.62 \times 10^{-5}) = 5.0000559 \\
W_{211} &= 1 + (5.26 \times 10^{-3} + -1.37 \times 10^{-4}) = 1.005123 \\
W_{212} &= 0.25 + (5.32 \times 10^{-5} + -1.36 \times 10^{-2}) = 0.2364532 \\
W_{221} &= -2 + (-5.26 \times 10^{-3} + 1.26 \times 10^{-5}) = -2.0052474 \\
W_{222} &= 2 + (-5.32 \times 10^{-5} + 1.25 \times 10^{-3}) = 2.0011968
\end{aligned}
$$

۴ سوال چهارم

در ابتدا بخش‌های تکمیل شده شرح داده می‌شوند. قابل به ذکر است که به دلیل اینکه بسیاری از این توابع تنها پیاده‌سازی فرمول‌های ریاضیاتی آن‌ها است و نکته خاصی در آنها وجود ندارد از توضیح صرف نظر شده و تنها تصاویر تکمیل شده آنها آورده می‌شود. همچنین فایل Jupyter تکمیل شده آن در پوشه زیپ قرار دارد.

```python
def relu(x, deriv=False):
    """
    Args:
      x: A numpy array of any shape
      deriv: True or False. determines if we want the derivative of the function

    Returns:
      relu_out: A numpy array of the same shape as x.
        Basically relu function or its derivative applied to every element of x

    """

    x_shape = x.shape

    if deriv:
        x_return = np.array([1 if y >= 0 else 0 for y in x.flatten()])
        return x_return.reshape(x_shape)
    else:
        x_return = np.array([max(y, 0) for y in x.flatten()])
        return x_return.reshape(x_shape)

    return relu_out
```

(آ) تابع ReLU

```python
def sigmoid(x, deriv=False):
    """
    Args:
      x: A numpy array of any shape
      deriv: True or False. determines if we want the deriv

    Returns:
      sig_out: A numpy array of the same shape as x.
        Basically sigmoid function or its derivative applie

    """

    x_shape = x.shape

    if deriv:
        sig = 1 / (1+np.exp(-x))
        return sig * (1- sig)
    else:
        return 1 / (1+np.exp(-x))

    return sig_out
```

(ب) تابع Sigmoid

```python
def feed_forward(self, X):
    """
    Propagate the inputs forward using this function
    Args:
      X: A numpy array of shape (b, input_dim) where b is the batch size and input_dim is the dimension of the input

    Returns:
      mlp_out: A numpy array of shape (b, out_dim) where b is the batch size and out_dim is the dimension of the output

      Hint: Don't forget to store weighted inputs and outputs of each layer in self.weighted_ins and self.activations respectively

    """
      self.activations = []
      self.weighted_ins = []
      mlp_out = X

      for layer in range(len(self.parameters)):
        layer_output = np.dot(mlp_out, self.parameters[layer]['w']) + self.parameters[layer]['b']
        self.weighted_ins.append(layer_output)
        mlp_out = self.act_funcs[layer](layer_output)
        self.activations.append(mlp_out)

      return mlp_out
```

(ج) تابع feed forward

```python
def softmax(y_hat):
    """
      Apply softmax to the inputs
    Args:
      y_hat: A numpy array of shape (b, out_dim) where b is the batch size and out_dim is the ou

    Returns:
      soft_out: A numpy array of shape (b, out_dim)

    """
    batch_size = y_hat.shape[0]
    exp_sums = np.sum(np.exp(y_hat), axis=1)
    soft_out = np.array([np.exp(y_hat[i]) / exp_sums[i] for i in range(batch_size)])
    return soft_out
```

(د) تابع Softmax

```python
def categorical_cross_entropy(y, y_soft):
    """
      Compute the categorical cross entropy loss
    Args:
      y: A numpy array of shape (b, out_dim). Target labels of network.
      y_soft: A numpy array of shape (b, out_dim). Output of the softmax activatio

    Returns:
      loss: A scaler of type float. Average loss over a batch.

    Hint: Use np.mean to compute average loss of a batch

    """
    softmax_log = np.log(y_soft)
    loss = np.mean(-np.sum(np.multiply(softmax_log,y),axis=1))

    return loss
```

(ه) تابع Categorical Cross Entropy

```python
    gradients = []

    ### get the output of the network
    y_hat = mlp.activations[-1]
    num_layers = len(mlp.parameters)

    ### compute gradient of the loss with respect to network output
    g = loss_function(y, y_hat, return_grad=True)

    ### You'll need the input in the last step of backprop so let's make a new l
    activations = [x] + mlp.activations

    for i in reversed(range(num_layers)):
        act_func_grad = mlp.act_funcs[i](mlp.weighted_ins[i], deriv=True)
        dx = np.multiply(g, act_func_grad)
        dw = np.dot(activations[i].T, dx)
        db = np.sum(dx, axis=0)
        gradients.append({"w": dw, "b": db})
        g = np.dot(dx, mlp.parameters[i]["w"].T)
    gradients.reverse()

    return gradients
```

(و) تابع Back Propagation

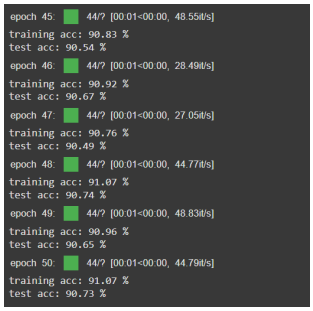
```python
mlp = MLP(x_train.shape[-1])

w1 = np.random.random(size=(mlp.input_dim, 4))
b1 = np.random.random(size=(4,))
layer_1 = {'w' : w1, 'b':b1}
w2 = np.random.random(size=(4, 10))
b2 = np.random.random(size=(10,))
layer_2 = {'w' : w2, 'b':b2}
mlp.parameters.append(layer_1)
mlp.parameters.append(layer_2)
mlp.act_funcs.append(relu)
mlp.act_funcs.append(linear)
```

(ح) تعریف مدل MLP

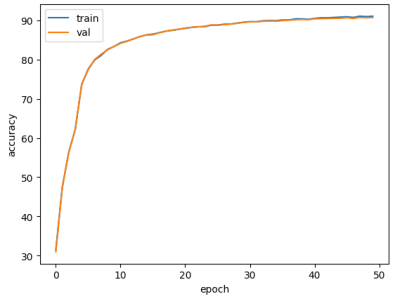
```python
def step(self, parameters, grads):
    """
      Perform a gradient descent step
    Args:
      parameters: A list of dictionaries {'w': weights , 'b': bias}. MLP's p
      grads: A list of dictionaries {'w': dw, 'b': db}. gradient of MLP's pa

    Returns:
      Updated_parameters: A list of dictionaries {'w': weights , 'b': bias}

    """
    Updated_parameters = []
    layer_num = len(parameters)

    for layer in range(layer_num):
        updated_w = parameters[layer]["w"] - self.lr * grads[layer]["w"]
        updated_b = parameters[layer]["b"] - self.lr * grads[layer]["b"]
        Updated_parameters.append({"w": updated_w, "b": updated_b})

    return Updated_parameters
```

(ز) تابع Step

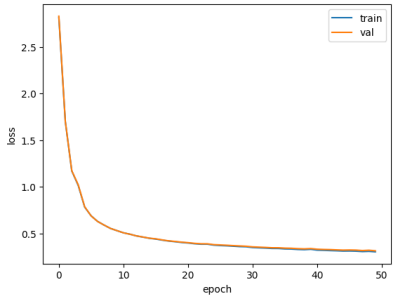
در ادامه میزان خطا، دقت و ۵ epoch آخر را نمایش می‌دهیم.



Last 5 Epochs (ک)



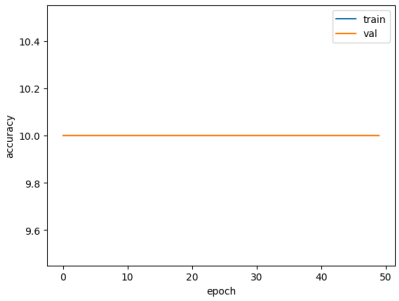
Accuracy (ی)



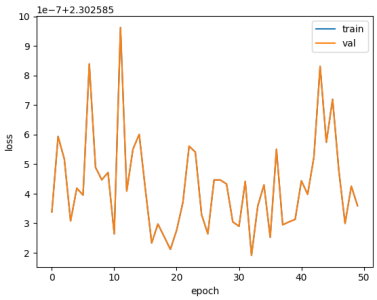
Loss (ط)

۱.۴ بخش a

در ابتدای کار، بجای مقداردهی رندوم وزن‌ها، آنها را با صفر مقداردهی کردیم و نمودار خطا و دقت را بدست آوردیم.



(م) مقادیر دقت



(ل) مقادیر خطا

همانگونه که مشاهده می‌شود، با وزن‌دهی وزن‌ها با صفر، تمام لایه‌ها متقارن شده و خروجی لایه‌ها و مشتق به‌طورکلی صفر شده و فرآیند آموزش متوقف می‌شود.

۲.۴ بخش b

در الگوریتم‌های گرادیان کاهشی، گرادیان لزوما کاهش نمی‌یابد زیرا ممکن است براثر زیاد بودن نرخ یادگیری، مدل از یک مینیمم محلی یا کلی عبور کرده یا به اصطلاح پرش کرده و به مقدار خطای بیشتری برسد که دراینصورت گرادیان بدلیل خطای بیشتر مدل در محل جدید نه تنها کاهش نیافته که افزایش نیز یافته است. این افزایش گرادیان هم می‌تواند خوب باشد (مثلا از مینیمم محلی عبور کرده و به مینیمم جهانی برسد) و هم بد باشد (در یک نقطه مناسب نتواند به مینیمم برسد و در اطراف آن پرش کند).

۳.۴ بخش c

Overfit : در اینحالت مدل داده‌های آموزشی را به خوبی یاد می‌گیرد ولی در داده‌های تست عملکرد ضعیفی دارد. به عبارتی مدل به جزئیات و نویزهای داده‌های آموزشی بیش از حد حساس شده است. برای رفع این مشکل می‌توان از راهکارهایی مانند کاهش پیچیدگی مدل، استفاده از Drop out برای جلوگیری از وابستگی مدل به برخی نورون‌های خاص، استفاده از داده‌های بیشتر یا افزایش تنوع داده‌ها بهره برد.

Underfit : در اینحالت مدل نتوانسته داده‌های آموزشی را به خوبی یاد بگیرد و در داده‌های تست نیز عملکرد ضعیفی دارد. به عبارت دیگر مدل به قدر کافی پیچیده نیست تا الگوهای موجود در داده‌ها را یاد بگیرد. برای رفع این مشکل می‌توان از پیچیده‌تر کردن مدل (افزایش تعداد لایه‌ها و نورون‌ها)، استفاده از توابع فعالسازی پیچیده‌تر، افزایش تعداد epoch ها استفاده کرد.

۴.۴ بخش d

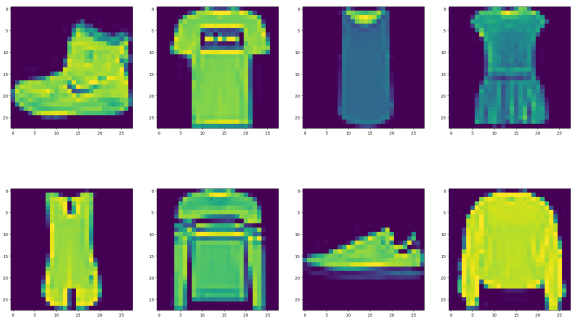
با بررسی نمودار Loss نهایی درمی‌یابیم که به دلیل آنکه مقدار loss برای آموزش و تست تقریبا برابر بوده و مدل به دقت بالای ٪۹۰ دست یافته پس می‌توان گفت که مدل درحالت fit بوده و فرآیند آموزش به خوبی پیش رفته است.

۵ سوال پنجم

در ابتدا کد نوشته شده برای این سوال شرح داده می‌شود. همچنین فایل Jupyter این سوال در پوشه زیپ شده قرار دارد.
در ابتدا دیتاست Fashion MNIST لود می‌شود و ۸ تصویر رندوم از این دیتاست نمایش داده می‌شود.

```
[2]  (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
     assert x_train.shape == (60000, 28, 28)
     assert x_test.shape == (10000, 28, 28)
     assert y_train.shape == (60000,)
     assert y_test.shape == (10000,)

[3]  fig, axs = plt.subplots(2, 4, figsize=(25, 15))

     axs = axs.ravel()

     for i, image in enumerate(x_train[:8]):
         axs[i].imshow(image)

     plt.show()
```

(ن) کد مربوط به لود کردن دیتاست و نمایش چند نمونه



(س) چند نمونه از دیتاست Fashion MNIST

سپس مدل را بصورت یک مدل ۵ لایه تعریف می‌کنیم که درلایه اول ۳۲ نورون، در لایه دوم و سوم ۶۴ نورون، در لایه چهارم ۳۲ نورون و درلایه پنجم و آخر ۱۰ نورون قرار دارند. همچنین توابع فعالسازی تمام لایه‌ها بجز لایه آخر ReLU بوده و در لایه آخر تابع فعالسازی Softmax قرار دارد (۱ف). با توجه به اینکه تصاویر دیتاست از جزئیات چندان زیادی مانند رنگ، تصاویر پس زمینه و ... برخوردار نیست به‌همین دلیل تعداد نورون‌ها در لایه‌های ابتدایی چندان زیاد نیست زیرا جزئیات زیادی برای استخراج وجود ندارد. درعوض سعی شده است تا با افزایش تعداد لایه‌ها، مدل بتواند ارتباط بین این ویژگی‌ها را بهتر درک و استخراج کند. همچنین بدلیل چند کلاسه بودن مساله،از تابع فعالسازی Softmax در لایه آخر استفاده شده است و برای حفظ گرادیان، در لایه‌های میانی از تابع فعالسازی ReLU استفاده شده است. همچنین تابع ضرر مناسب برای مسائل چند کلاسه، تابع ضرر Cross Entropy بود که استفاده شده است.

```
[5]  model = keras.Sequential([
         keras.Input(shape=input_shape),
         keras.layers.Flatten(),
         keras.layers.Dense(32, activation="relu"),
         keras.layers.Dense(64,activation="relu"),
         keras.layers.Dense(64,activation="relu"),
         keras.layers.Dense(32,activation="relu"),
         keras.layers.Dense(10,activation="softmax")
     ])

[6]  model.summary()
```

(ع) تعریف مدل عمیق و ترسیم پارامترهای مدل در هر لایه

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 32)	25,120
dense_1 (Dense)	(None, 64)	2,112
dense_2 (Dense)	(None, 64)	4,160
dense_3 (Dense)	(None, 32)	2,080
dense_4 (Dense)	(None, 10)	330

```
Total params: 33,802 (132.04 KB)
Trainable params: 33,802 (132.04 KB)
Non-trainable params: 0 (0.00 B)
```

(ف) ویژگی‌های مدل

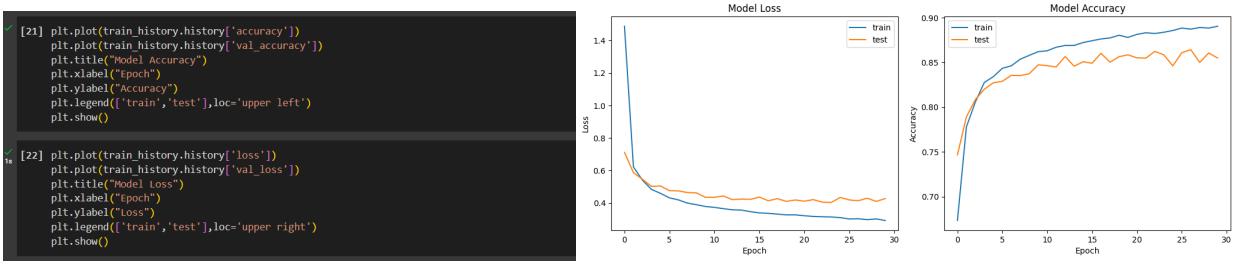
سپس به آموزش مدل می‌پردازیم.

```
[7]  batch_size = 128
     epochs = 30

     model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

     train_history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_test, y_test))
```

در اینجا نتیجه‌ی ۵ epoch آخر نمایش داده شده است.

```
Epoch 25/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8882 - loss: 0.3023 - val_accuracy: 0.8460 - val_loss: 0.4339
Epoch 26/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8887 - loss: 0.3028 - val_accuracy: 0.8607 - val_loss: 0.4186
Epoch 27/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.8896 - loss: 0.3001 - val_accuracy: 0.8642 - val_loss: 0.4137
Epoch 28/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8896 - loss: 0.2935 - val_accuracy: 0.8499 - val_loss: 0.4291
Epoch 29/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.8862 - loss: 0.3071 - val_accuracy: 0.8605 - val_loss: 0.4095
Epoch 30/30
469/469 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.8915 - loss: 0.2911 - val_accuracy: 0.8548 - val_loss: 0.4267
```

در این بخش نیز نتیجه‌ی نهایی نمایش داده می‌شود.



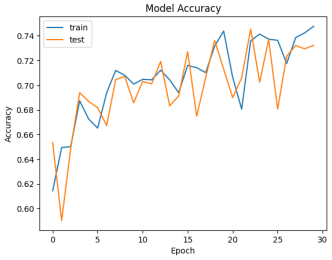
(ص) نمودار دقت مدل در هر epoch (ق) نمودار خطای مدل در هر epoch

```
[21] plt.plot(train_history.history['accuracy'])
     plt.plot(train_history.history['val_accuracy'])
     plt.title("Model Accuracy")
     plt.xlabel("Epoch")
     plt.ylabel("Accuracy")
     plt.legend(['train','test'],loc='upper left')
     plt.show()

[22] plt.plot(train_history.history['loss'])
     plt.plot(train_history.history['val_loss'])
     plt.title("Model Loss")
     plt.xlabel("Epoch")
     plt.ylabel("Loss")
     plt.legend(['train','test'],loc='upper right')
     plt.show()
```

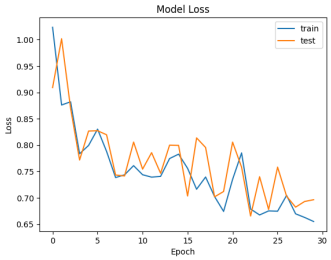
(ر) کد مربوط به این بخش

با توجه به نمودار دقت و ضرر برای آموزش و تست، فاصله بین این دو نمودار مقدار خوب و قابل قبولی است و فاصله‌ی کم آنها نشان‌دهنده عدم رخداد overfitting است.

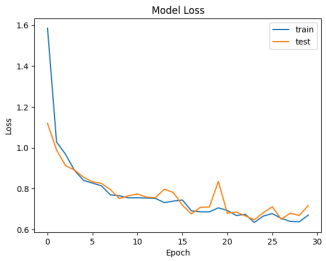
حال در ادامه تابع فعالسازی ReLU با تابع فعالسازی Tanh جایگزین می‌شود.

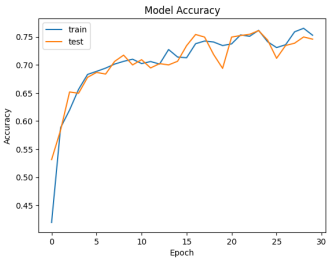



```
[25] model = keras.Sequential([
        keras.Input(shape=input_shape),
        keras.layers.Flatten(),
        keras.layers.Dense(32, activation="tanh"),
        keras.layers.Dense(64,activation="tanh"),
        keras.layers.Dense(64,activation="tanh"),
        keras.layers.Dense(32,activation="tanh"),
        keras.layers.Dense(10,activation="softmax")
    ])
```



همانطور که مشاهده می‌شود، مدل علاوه بر کاهش دقت هم در آموزش و هم تست، در این حالت رفتارهای ناپایدارتری نیز نشان می‌دهد.
در ادامه بجای تابع فعالسازی ReLU از تابع فعالسازی Sigmoid استفاده می‌شود.






```
[32] model = keras.Sequential([
        keras.Input(shape=input_shape),
        keras.layers.Flatten(),
        keras.layers.Dense(32, activation="sigmoid"),
        keras.layers.Dense(64,activation="sigmoid"),
        keras.layers.Dense(64,activation="sigmoid"),
        keras.layers.Dense(32,activation="sigmoid"),
        keras.layers.Dense(10,activation="softmax")
    ])
```

طبق مشاهدات، مدل در این حالت بیشتر پایدار بوده ولی سرعت آموزش آن کاهش یافته است که دلیل آن مقدار کم مشتق تابع Sigmoid می‌باشد.