315_Hw_4

April 5, 2024

1 Homework 4: From Data To Model

In this homework we will focus on going from the data to a model that generalizes well.

Me: Qiulin Fan, uniquename: rynnefan ### Collaborator: Weibing Su, uniquename: weibinsu Weile Zheng, uniquename: weilez

1.1 Imports

[2]:
```
import tensorflow as tf
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
from matplotlib import pyplot as plt
from tensorflow.keras import regularizers
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from keras.utils import to_categorical
from tensorflow.keras.datasets import fashion_mnist
```

2 Part 1: Warmup

We will first start off again with the california housing dataset.

[3]:
```
california_housing = fetch_california_housing(return_X_y=True, as_frame=True)
X = california_housing[0]
y = california_housing[1]
X_train_unscaled, X_test_unscaled, y_train, y_test = train_test_split(X, y,␣
↪test_size=0.3, random_state=42)
sc=StandardScaler()
X_train=sc.fit_transform(X_train_unscaled)
X_test = sc.transform(X_test_unscaled)
```

2.0.1 Question 1 (9 pts): Redo model by tf.keras.Sequential

Recreate the regression model created in HW 3 using tf.keras.Sequential. While you will have to use the same amount of epochs and batch size, for the optimizer you can use RMSprop with the same learning rate as in HW 3.

[4]:
```
def q1():
    input_dim = 8
    model = tf.keras.Sequential([
        # relu
        tf.keras.layers.Dense(2*input_dim, activation='relu',␣
 ↪input_shape=(input_dim, )),
        # regression
        tf.keras.layers.Dense(1)
    ])
    model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.01),
                  loss='mean_squared_error')
    return model

model_q1 = q1()
model_q1.fit(X_train, y_train, epochs=100, validation_split=0.2,␣
 ↪batch_size=1000)
```

```
Epoch 1/100
12/12 [==============================] - 2s 19ms/step - loss: 2.2287 - val_loss:
1.1010
Epoch 2/100
12/12 [==============================] - 0s 6ms/step - loss: 0.9185 - val_loss:
0.7884
Epoch 3/100
12/12 [==============================] - 0s 6ms/step - loss: 0.6926 - val_loss:
0.6608
Epoch 4/100
12/12 [==============================] - 0s 8ms/step - loss: 0.5786 - val_loss:
0.5678
Epoch 5/100
12/12 [==============================] - 0s 5ms/step - loss: 0.5043 - val_loss:
0.5125
Epoch 6/100
12/12 [==============================] - 0s 6ms/step - loss: 0.4597 - val_loss:
0.4856
Epoch 7/100
12/12 [==============================] - 0s 6ms/step - loss: 0.4447 - val_loss:
0.4704
Epoch 8/100
12/12 [==============================] - 0s 5ms/step - loss: 0.4331 - val_loss:
0.4712
Epoch 9/100
12/12 [==============================] - 0s 6ms/step - loss: 0.4384 - val_loss:
0.4536
Epoch 10/100
12/12 [==============================] - 0s 6ms/step - loss: 0.4122 - val_loss:
```

```
0.4532
Epoch 11/100
12/12 [==============================] - 0s 6ms/step - loss: 0.4087 - val_loss:
0.4417
Epoch 12/100
12/12 [==============================] - 0s 6ms/step - loss: 0.4003 - val_loss:
0.4405
Epoch 13/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3960 - val_loss:
0.4411
Epoch 14/100
12/12 [==============================] - 0s 6ms/step - loss: 0.4013 - val_loss:
0.4315
Epoch 15/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3917 - val_loss:
0.4204
Epoch 16/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3848 - val_loss:
0.4215
Epoch 17/100
12/12 [==============================] - 0s 7ms/step - loss: 0.3832 - val_loss:
0.4224
Epoch 18/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3785 - val_loss:
0.4375
Epoch 19/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3815 - val_loss:
0.4435
Epoch 20/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3794 - val_loss:
0.4215
Epoch 21/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3718 - val_loss:
0.4152
Epoch 22/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3725 - val_loss:
0.4414
Epoch 23/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3734 - val_loss:
0.4136
Epoch 24/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3678 - val_loss:
0.4237
Epoch 25/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3664 - val_loss:
0.4334
Epoch 26/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3753 - val_loss:
```

```
0.3946
Epoch 27/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3635 - val_loss:
0.4082
Epoch 28/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3639 - val_loss:
0.3965
Epoch 29/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3571 - val_loss:
0.3928
Epoch 30/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3566 - val_loss:
0.4062
Epoch 31/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3524 - val_loss:
0.3880
Epoch 32/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3623 - val_loss:
0.3913
Epoch 33/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3522 - val_loss:
0.3877
Epoch 34/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3509 - val_loss:
0.3885
Epoch 35/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3494 - val_loss:
0.3909
Epoch 36/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3519 - val_loss:
0.3922
Epoch 37/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3468 - val_loss:
0.3839
Epoch 38/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3541 - val_loss:
0.3981
Epoch 39/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3474 - val_loss:
0.3792
Epoch 40/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3505 - val_loss:
0.3840
Epoch 41/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3500 - val_loss:
0.3713
Epoch 42/100
12/12 [==============================] - 0s 7ms/step - loss: 0.3533 - val_loss:
```

```
0.3699
Epoch 43/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3478 - val_loss:
0.3723
Epoch 44/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3400 - val_loss:
0.3839
Epoch 45/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3529 - val_loss:
0.3813
Epoch 46/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3414 - val_loss:
0.3714
Epoch 47/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3430 - val_loss:
0.3737
Epoch 48/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3405 - val_loss:
0.3696
Epoch 49/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3453 - val_loss:
0.3786
Epoch 50/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3435 - val_loss:
0.3744
Epoch 51/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3359 - val_loss:
0.4101
Epoch 52/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3457 - val_loss:
0.3681
Epoch 53/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3347 - val_loss:
0.3721
Epoch 54/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3420 - val_loss:
0.3676
Epoch 55/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3330 - val_loss:
0.3737
Epoch 56/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3392 - val_loss:
0.3795
Epoch 57/100
12/12 [==============================] - 0s 9ms/step - loss: 0.3342 - val_loss:
0.3760
Epoch 58/100
12/12 [==============================] - 0s 7ms/step - loss: 0.3437 - val_loss:
```

```
0.3671
Epoch 59/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3433 - val_loss:
0.3716
Epoch 60/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3396 - val_loss:
0.3725
Epoch 61/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3349 - val_loss:
0.3580
Epoch 62/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3329 - val_loss:
0.3696
Epoch 63/100
12/12 [==============================] - 0s 7ms/step - loss: 0.3356 - val_loss:
0.3711
Epoch 64/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3401 - val_loss:
0.3679
Epoch 65/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3305 - val_loss:
0.3569
Epoch 66/100
12/12 [==============================] - 0s 7ms/step - loss: 0.3327 - val_loss:
0.3700
Epoch 67/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3309 - val_loss:
0.3602
Epoch 68/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3293 - val_loss:
0.3671
Epoch 69/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3395 - val_loss:
0.4184
Epoch 70/100
12/12 [==============================] - 0s 7ms/step - loss: 0.3406 - val_loss:
0.3563
Epoch 71/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3292 - val_loss:
0.3602
Epoch 72/100
12/12 [==============================] - 0s 7ms/step - loss: 0.3256 - val_loss:
0.3613
Epoch 73/100
12/12 [==============================] - 0s 9ms/step - loss: 0.3309 - val_loss:
0.3577
Epoch 74/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3297 - val_loss:
```

```
0.3693
Epoch 75/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3326 - val_loss:
0.3650
Epoch 76/100
12/12 [==============================] - 0s 7ms/step - loss: 0.3258 - val_loss:
0.3561
Epoch 77/100
12/12 [==============================] - 0s 8ms/step - loss: 0.3329 - val_loss:
0.3653
Epoch 78/100
12/12 [==============================] - 0s 7ms/step - loss: 0.3394 - val_loss:
0.3771
Epoch 79/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3245 - val_loss:
0.3686
Epoch 80/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3258 - val_loss:
0.3618
Epoch 81/100
12/12 [==============================] - 0s 4ms/step - loss: 0.3361 - val_loss:
0.3512
Epoch 82/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3332 - val_loss:
0.3544
Epoch 83/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3227 - val_loss:
0.3856
Epoch 84/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3347 - val_loss:
0.3654
Epoch 85/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3259 - val_loss:
0.3603
Epoch 86/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3289 - val_loss:
0.3689
Epoch 87/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3309 - val_loss:
0.3909
Epoch 88/100
12/12 [==============================] - 0s 4ms/step - loss: 0.3364 - val_loss:
0.3531
Epoch 89/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3243 - val_loss:
0.3647
Epoch 90/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3269 - val_loss:
```

```
0.3619
Epoch 91/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3261 - val_loss:
0.3597
Epoch 92/100
12/12 [==============================] - 0s 4ms/step - loss: 0.3311 - val_loss:
0.3637
Epoch 93/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3215 - val_loss:
0.3736
Epoch 94/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3278 - val_loss:
0.3672
Epoch 95/100
12/12 [==============================] - 0s 7ms/step - loss: 0.3351 - val_loss:
0.3792
Epoch 96/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3250 - val_loss:
0.3476
Epoch 97/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3265 - val_loss:
0.3699
Epoch 98/100
12/12 [==============================] - 0s 4ms/step - loss: 0.3317 - val_loss:
0.3696
Epoch 99/100
12/12 [==============================] - 0s 5ms/step - loss: 0.3218 - val_loss:
0.3617
Epoch 100/100
12/12 [==============================] - 0s 6ms/step - loss: 0.3256 - val_loss:
0.3794
```

[4]:
```
<keras.src.callbacks.History at 0x7f30795433a0>
```

3 Part 2: MNIST to Model

Now we shall switch to the MNIST dataset

[5]:
```
(X, y_int), _ = fashion_mnist.load_data()
X = X.reshape(X.shape[0],-1)
y_one_hot = to_categorical(y_int, num_classes=10)
X_train_unscaled, X_test_unscaled, y_train, y_test = train_test_split(X,
 ↪y_one_hot, test_size=0.3, random_state=100)
sc=StandardScaler()
X_train=sc.fit_transform(X_train_unscaled)
X_test = sc.transform(X_test_unscaled)
```

```
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,␣
↪test_size=0.2, random_state=42)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-labels-idx1-ubyte.gz
29515/29515 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/train-images-idx3-ubyte.gz
26421880/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-labels-idx1-ubyte.gz
5148/5148 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/t10k-images-idx3-ubyte.gz
4422102/4422102 [==============================] - 0s 0us/step
```

[6]:
```
X_train.shape, X_test.shape
```

[6]:
```
((33600, 784), (18000, 784))
```

3.0.1 Question 2 (9 pts): Underfit

Create a model that underfits the data after at least 50 epochs.

[7]:
```
def q2():
    input_dim = 28*28  #784

    model_digits = tf.keras.Sequential([
        # Use simple layers to underfit
        tf.keras.layers.Dense(10, activation='relu', input_shape=(input_dim,)),
        # softmax
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model_digits.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.
↪0005),
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model_digits
```

[8]:
```
model_q2 = q2()
history_q2 = model_q2.fit(X_train,y_train, epochs=50, batch_size=2048,␣
↪validation_data=(X_valid, y_valid))
```

```
Epoch 1/50
17/17 [==============================] - 1s 30ms/step - loss: 1.9289 - accuracy:
0.2995 - val_loss: 1.6419 - val_accuracy: 0.4165
Epoch 2/50
```

```
17/17 [==============================] - 0s 12ms/step - loss: 1.5081 - accuracy:
0.4615 - val_loss: 1.4206 - val_accuracy: 0.4911
Epoch 3/50
17/17 [==============================] - 0s 13ms/step - loss: 1.3156 - accuracy:
0.5367 - val_loss: 1.2573 - val_accuracy: 0.5689
Epoch 4/50
17/17 [==============================] - 0s 8ms/step - loss: 1.1675 - accuracy:
0.6077 - val_loss: 1.1295 - val_accuracy: 0.6215
Epoch 5/50
17/17 [==============================] - 0s 8ms/step - loss: 1.0516 - accuracy:
0.6561 - val_loss: 1.0282 - val_accuracy: 0.6654
Epoch 6/50
17/17 [==============================] - 0s 11ms/step - loss: 0.9584 - accuracy:
0.6933 - val_loss: 0.9452 - val_accuracy: 0.7027
Epoch 7/50
17/17 [==============================] - 0s 8ms/step - loss: 0.8815 - accuracy:
0.7262 - val_loss: 0.8725 - val_accuracy: 0.7355
Epoch 8/50
17/17 [==============================] - 0s 8ms/step - loss: 0.8161 - accuracy:
0.7536 - val_loss: 0.8127 - val_accuracy: 0.7575
Epoch 9/50
17/17 [==============================] - 0s 8ms/step - loss: 0.7586 - accuracy:
0.7733 - val_loss: 0.7578 - val_accuracy: 0.7748
Epoch 10/50
17/17 [==============================] - 0s 8ms/step - loss: 0.7073 - accuracy:
0.7889 - val_loss: 0.7108 - val_accuracy: 0.7883
Epoch 11/50
17/17 [==============================] - 0s 8ms/step - loss: 0.6629 - accuracy:
0.8003 - val_loss: 0.6696 - val_accuracy: 0.7989
Epoch 12/50
17/17 [==============================] - 0s 8ms/step - loss: 0.6248 - accuracy:
0.8070 - val_loss: 0.6358 - val_accuracy: 0.8052
Epoch 13/50
17/17 [==============================] - 0s 8ms/step - loss: 0.5930 - accuracy:
0.8131 - val_loss: 0.6094 - val_accuracy: 0.8075
Epoch 14/50
17/17 [==============================] - 0s 8ms/step - loss: 0.5667 - accuracy:
0.8187 - val_loss: 0.5852 - val_accuracy: 0.8119
Epoch 15/50
17/17 [==============================] - 0s 8ms/step - loss: 0.5450 - accuracy:
0.8231 - val_loss: 0.5669 - val_accuracy: 0.8170
Epoch 16/50
17/17 [==============================] - 0s 8ms/step - loss: 0.5260 - accuracy:
0.8276 - val_loss: 0.5519 - val_accuracy: 0.8183
Epoch 17/50
17/17 [==============================] - 0s 8ms/step - loss: 0.5106 - accuracy:
0.8314 - val_loss: 0.5387 - val_accuracy: 0.8211
Epoch 18/50
```

```
17/17 [==============================] - 0s 10ms/step - loss: 0.4983 - accuracy:
0.8332 - val_loss: 0.5276 - val_accuracy: 0.8226
Epoch 19/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4874 - accuracy:
0.8354 - val_loss: 0.5193 - val_accuracy: 0.8240
Epoch 20/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4785 - accuracy:
0.8371 - val_loss: 0.5115 - val_accuracy: 0.8260
Epoch 21/50
17/17 [==============================] - 0s 7ms/step - loss: 0.4701 - accuracy:
0.8391 - val_loss: 0.5037 - val_accuracy: 0.8295
Epoch 22/50
17/17 [==============================] - 0s 7ms/step - loss: 0.4632 - accuracy:
0.8414 - val_loss: 0.4994 - val_accuracy: 0.8315
Epoch 23/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4567 - accuracy:
0.8435 - val_loss: 0.4920 - val_accuracy: 0.8315
Epoch 24/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4504 - accuracy:
0.8445 - val_loss: 0.4873 - val_accuracy: 0.8356
Epoch 25/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4450 - accuracy:
0.8465 - val_loss: 0.4843 - val_accuracy: 0.8333
Epoch 26/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4402 - accuracy:
0.8484 - val_loss: 0.4841 - val_accuracy: 0.8325
Epoch 27/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4351 - accuracy:
0.8493 - val_loss: 0.4795 - val_accuracy: 0.8365
Epoch 28/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4308 - accuracy:
0.8504 - val_loss: 0.4718 - val_accuracy: 0.8414
Epoch 29/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4269 - accuracy:
0.8518 - val_loss: 0.4704 - val_accuracy: 0.8389
Epoch 30/50
17/17 [==============================] - 0s 10ms/step - loss: 0.4232 - accuracy:
0.8531 - val_loss: 0.4669 - val_accuracy: 0.8399
Epoch 31/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4194 - accuracy:
0.8549 - val_loss: 0.4616 - val_accuracy: 0.8435
Epoch 32/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4162 - accuracy:
0.8557 - val_loss: 0.4622 - val_accuracy: 0.8412
Epoch 33/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4129 - accuracy:
0.8566 - val_loss: 0.4576 - val_accuracy: 0.8446
Epoch 34/50
```

```
17/17 [==============================] - 0s 8ms/step - loss: 0.4100 - accuracy:
0.8576 - val_loss: 0.4553 - val_accuracy: 0.8437
Epoch 35/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4071 - accuracy:
0.8593 - val_loss: 0.4561 - val_accuracy: 0.8446
Epoch 36/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4044 - accuracy:
0.8593 - val_loss: 0.4525 - val_accuracy: 0.8458
Epoch 37/50
17/17 [==============================] - 0s 8ms/step - loss: 0.4019 - accuracy:
0.8594 - val_loss: 0.4514 - val_accuracy: 0.8462
Epoch 38/50
17/17 [==============================] - 0s 7ms/step - loss: 0.3995 - accuracy:
0.8601 - val_loss: 0.4496 - val_accuracy: 0.8455
Epoch 39/50
17/17 [==============================] - 0s 10ms/step - loss: 0.3972 - accuracy:
0.8623 - val_loss: 0.4503 - val_accuracy: 0.8455
Epoch 40/50
17/17 [==============================] - 0s 8ms/step - loss: 0.3946 - accuracy:
0.8629 - val_loss: 0.4460 - val_accuracy: 0.8496
Epoch 41/50
17/17 [==============================] - 0s 8ms/step - loss: 0.3926 - accuracy:
0.8641 - val_loss: 0.4457 - val_accuracy: 0.8485
Epoch 42/50
17/17 [==============================] - 0s 8ms/step - loss: 0.3915 - accuracy:
0.8627 - val_loss: 0.4457 - val_accuracy: 0.8479
Epoch 43/50
17/17 [==============================] - 0s 8ms/step - loss: 0.3891 - accuracy:
0.8651 - val_loss: 0.4427 - val_accuracy: 0.8504
Epoch 44/50
17/17 [==============================] - 0s 8ms/step - loss: 0.3873 - accuracy:
0.8647 - val_loss: 0.4413 - val_accuracy: 0.8496
Epoch 45/50
17/17 [==============================] - 0s 10ms/step - loss: 0.3853 - accuracy:
0.8655 - val_loss: 0.4394 - val_accuracy: 0.8524
Epoch 46/50
17/17 [==============================] - 0s 8ms/step - loss: 0.3846 - accuracy:
0.8661 - val_loss: 0.4407 - val_accuracy: 0.8500
Epoch 47/50
17/17 [==============================] - 0s 10ms/step - loss: 0.3823 - accuracy:
0.8666 - val_loss: 0.4391 - val_accuracy: 0.8511
Epoch 48/50
17/17 [==============================] - 0s 8ms/step - loss: 0.3808 - accuracy:
0.8676 - val_loss: 0.4395 - val_accuracy: 0.8499
Epoch 49/50
17/17 [==============================] - 0s 9ms/step - loss: 0.3788 - accuracy:
0.8680 - val_loss: 0.4359 - val_accuracy: 0.8508
Epoch 50/50
```

```
17/17 [==============================] - 0s 7ms/step - loss: 0.3775 - accuracy:
0.8684 - val_loss: 0.4367 - val_accuracy: 0.8526
```

[9]:
```
print(history_q2.history)
```

{'loss': [1.9288711547851562, 1.5081032514572144, 1.315558671951294,
1.1675034761428833, 1.051605224609375, 0.9583876132965088, 0.8814641237258911,
0.8160650134086609, 0.7586497068405151, 0.7073211669921875, 0.6629389524459839,
0.6247860789299011, 0.5930214524269104, 0.566676139831543, 0.5450043082237244,
0.5260263085365295, 0.510614812374115, 0.49825137853622437, 0.4873904287815094,
0.47847747802734375, 0.47009941935539246, 0.46319442987442017,
0.4566786289215088, 0.45040732622146606, 0.44500818848609924,
0.44017019867897034, 0.4351077973842621, 0.4307532012462616,
0.42691144347190857, 0.42317304015159607, 0.4193999171257019,
0.41624853014945984, 0.4129059612751007, 0.4099704325199127,
0.40710946917533875, 0.4044053852558136, 0.4019085466861725, 0.3995043933391571,
0.39715883135795593, 0.39462795853614807, 0.3925551474094391,
0.39147403836250305, 0.3890562653541565, 0.387264221906662, 0.38528546690940857,
0.384644091129303, 0.3822512924671173, 0.3808186650276184, 0.37879085540771484,
0.37745875120162964], 'accuracy': [0.29949405789375305, 0.461517870426178,
0.536726176738739, 0.6076785922050476, 0.6560714244842529, 0.6933333277702332,
0.7262202501296997, 0.7536309361457825, 0.7733333110809326, 0.7888988256454468,
0.8002976179122925, 0.8069642782211304, 0.813095211982727, 0.8186607360839844,
0.8230952620506287, 0.8275595307350159, 0.8313690423965454, 0.833184540271759,
0.835357129573822, 0.837113082408905, 0.8391368985176086, 0.8414285778999329,
0.84354168176651, 0.8445237874984741, 0.846547603607177, 0.8483631014823914,
0.8493154644966125, 0.8503571152687073, 0.8518154621124268, 0.8530952334403992,
0.8548511862754822, 0.855654776096344, 0.8566368818283081, 0.8575595021247864,
0.8593452572822571, 0.8592559695243835, 0.8594047427177429, 0.8601487874984741,
0.8623214364051819, 0.8629166483879089, 0.8641369342803955, 0.862708330154419,
0.8651487827301025, 0.8647024035453796, 0.8655059337615967, 0.8661309480667114,
0.8666369318962097, 0.867559552192688, 0.867976188659668, 0.868363082408905],
'val_loss': [1.641914963722229, 1.4205549955368042, 1.25730299949646,
1.1295181512832642, 1.0282026529312134, 0.9451822638511658, 0.8724880218505859,
0.8126883506774902, 0.7578064203262329, 0.7108434438705444, 0.6696445345878601,
0.6357702016830444, 0.6093555688858032, 0.5851646661758423, 0.5669193267822266,
0.5518868565559387, 0.5386661887168884, 0.5276150107383728, 0.5193021297454834,
0.5115480422973633, 0.503667950630188, 0.49944934248924255, 0.49202218651771545,
0.4872981011867523, 0.4842926859855652, 0.4840870499610901, 0.4795241355895996,
0.4717732071876526, 0.470409095287323, 0.466913640499115, 0.4616324007511139,
0.46223923563957214, 0.45759356021881104, 0.4552648961544037,
0.4560765326023102, 0.45245838165283203, 0.45142027735710144,
0.4496140778064728, 0.45029744505882263, 0.4460361897945404, 0.4456797242164612,
0.4456821084022522, 0.4426749348640442, 0.44130706787109375, 0.4393972158432007,
0.4407111704349518, 0.43911033868789673, 0.4395406246185303,
0.43587154150009155, 0.43666934967041016], 'val_accuracy': [0.41654762625694275,
0.4910714328289032, 0.568928599357605, 0.6215476393699646, 0.6653571724891663,
0.7027381062507629, 0.7354761958122253, 0.7574999928474426, 0.7747619152069092,

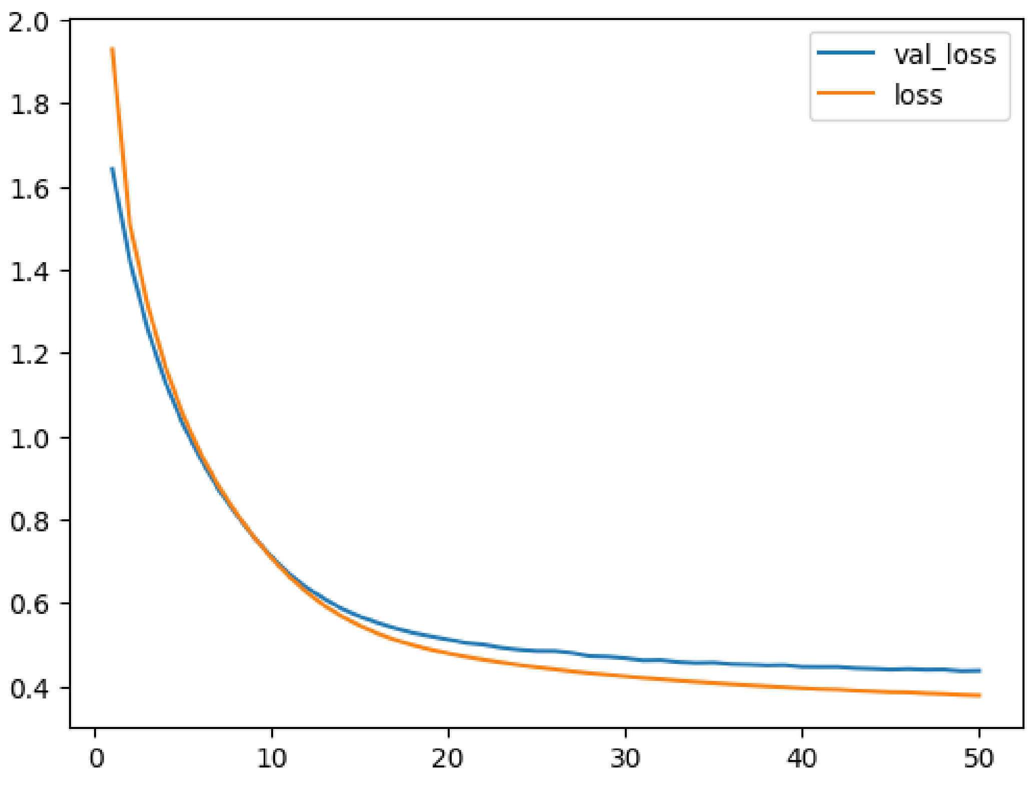
```
0.7883333563804626, 0.7989285588264465, 0.8052380681037903, 0.8075000047683716,
0.811904788017273, 0.8170238137245178, 0.8183333277702332, 0.821071445941925,
0.8226190209388733, 0.8240476250648499, 0.825952410697937, 0.8295238018035889,
0.8315476179122925, 0.8315476179122925, 0.8355952501296997, 0.8333333134651184,
0.8324999809265137, 0.8365476131439209, 0.8414285778999329, 0.8389285802841187,
0.8398809432983398, 0.8434523940086365, 0.8411904573440552, 0.8446428775787354,
0.8436904549598694, 0.8446428775787354, 0.8458333611488342, 0.8461904525756836,
0.8454762101173401, 0.8454762101173401, 0.8496428728103638, 0.8484523892402649,
0.8478571176528931, 0.8503571152687073, 0.8496428728103638, 0.8523809313774109,
0.8500000238418579, 0.8510714173316956, 0.8498809337615967, 0.8508333563804626,
0.8526190519332886]}
```

[10]:

```
val_loss = history_q2.history["val_loss"]
loss = history_q2.history["loss"]
plt.plot(np.arange(1,len(val_loss)+1), val_loss, label="val_loss")
plt.plot(np.arange(1,len(loss)+1), loss, label="loss")
plt.legend()
```

[10]:

```
<matplotlib.legend.Legend at 0x7f30239a12a0>
```



3.0.2 Question 3 (9 pts): Overfit

Take the model from question 2 and change it/tweak hyperparameters until it is able to overfit the data. The resulting model does not need to be similar to the answer in question 2.

[11]:
```
def q3():
    input_dim = 28*28

    model = tf.keras.Sequential([
        # Increased layers to overfit
        tf.keras.layers.Dense(2 * input_dim, activation='relu',␣
 ↪input_shape=(input_dim, )),
        # softmax
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0005),
                  loss='categorical_crossentropy', metrics=['accuracy'])

    return model
```

[12]:
```
ep = 20
bs = 512
```

[13]:
```
model_q3 = q3()
history_q3 = model_q3.fit(X_train,y_train, epochs=ep, batch_size=bs,␣
 ↪validation_data=(X_valid, y_valid))
```

```
Epoch 1/20
66/66 [==============================] - 2s 12ms/step - loss: 0.6240 - accuracy:
0.7885 - val_loss: 0.4719 - val_accuracy: 0.8238
Epoch 2/20
66/66 [==============================] - 1s 9ms/step - loss: 0.4217 - accuracy:
0.8492 - val_loss: 0.4178 - val_accuracy: 0.8495
Epoch 3/20
66/66 [==============================] - 1s 9ms/step - loss: 0.3562 - accuracy:
0.8717 - val_loss: 0.4721 - val_accuracy: 0.8437
Epoch 4/20
66/66 [==============================] - 1s 13ms/step - loss: 0.3147 - accuracy:
0.8865 - val_loss: 0.4237 - val_accuracy: 0.8470
Epoch 5/20
66/66 [==============================] - 1s 16ms/step - loss: 0.2914 - accuracy:
0.8918 - val_loss: 0.3540 - val_accuracy: 0.8717
Epoch 6/20
66/66 [==============================] - 1s 12ms/step - loss: 0.2624 - accuracy:
0.9037 - val_loss: 0.3752 - val_accuracy: 0.8648
Epoch 7/20
66/66 [==============================] - 1s 10ms/step - loss: 0.2406 - accuracy:
0.9124 - val_loss: 0.3587 - val_accuracy: 0.8736
```

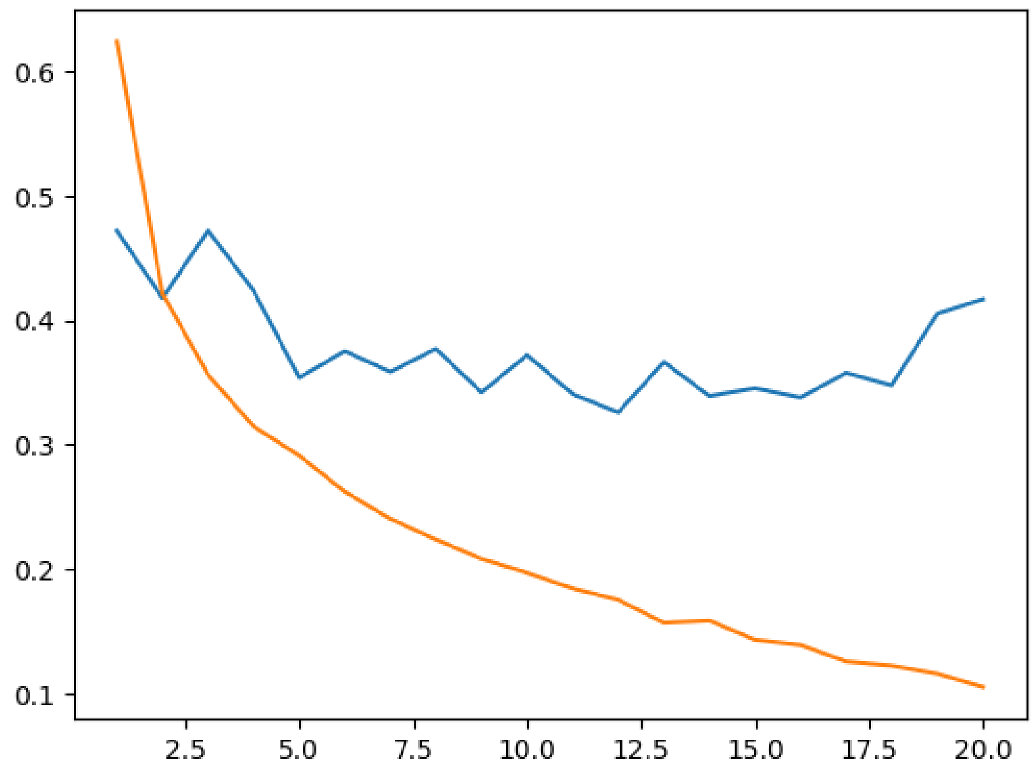
```
Epoch 8/20
66/66 [==============================] - 1s 10ms/step - loss: 0.2240 - accuracy:
0.9177 - val_loss: 0.3770 - val_accuracy: 0.8621
Epoch 9/20
66/66 [==============================] - 1s 11ms/step - loss: 0.2085 - accuracy:
0.9243 - val_loss: 0.3421 - val_accuracy: 0.8764
Epoch 10/20
66/66 [==============================] - 1s 11ms/step - loss: 0.1972 - accuracy:
0.9289 - val_loss: 0.3721 - val_accuracy: 0.8729
Epoch 11/20
66/66 [==============================] - 1s 9ms/step - loss: 0.1847 - accuracy:
0.9336 - val_loss: 0.3407 - val_accuracy: 0.8843
Epoch 12/20
66/66 [==============================] - 1s 9ms/step - loss: 0.1755 - accuracy:
0.9375 - val_loss: 0.3260 - val_accuracy: 0.8931
Epoch 13/20
66/66 [==============================] - 1s 11ms/step - loss: 0.1572 - accuracy:
0.9438 - val_loss: 0.3666 - val_accuracy: 0.8785
Epoch 14/20
66/66 [==============================] - 1s 10ms/step - loss: 0.1589 - accuracy:
0.9461 - val_loss: 0.3393 - val_accuracy: 0.8868
Epoch 15/20
66/66 [==============================] - 1s 11ms/step - loss: 0.1434 - accuracy:
0.9493 - val_loss: 0.3455 - val_accuracy: 0.8882
Epoch 16/20
66/66 [==============================] - 1s 11ms/step - loss: 0.1392 - accuracy:
0.9504 - val_loss: 0.3380 - val_accuracy: 0.8885
Epoch 17/20
66/66 [==============================] - 1s 10ms/step - loss: 0.1261 - accuracy:
0.9563 - val_loss: 0.3578 - val_accuracy: 0.8842
Epoch 18/20
66/66 [==============================] - 1s 10ms/step - loss: 0.1226 - accuracy:
0.9564 - val_loss: 0.3478 - val_accuracy: 0.8877
Epoch 19/20
66/66 [==============================] - 1s 10ms/step - loss: 0.1162 - accuracy:
0.9604 - val_loss: 0.4054 - val_accuracy: 0.8771
Epoch 20/20
66/66 [==============================] - 1s 12ms/step - loss: 0.1056 - accuracy:
0.9651 - val_loss: 0.4169 - val_accuracy: 0.8730
```

[14]:

```
val_loss = history_q3.history["val_loss"]
loss = history_q3.history["loss"]
plt.plot(np.arange(1,len(val_loss)+1),val_loss)
plt.plot(np.arange(1,len(val_loss)+1), loss)
```

[14]:

```
[<matplotlib.lines.Line2D at 0x7f30040cbc70>]
```



3.0.3 Question 4 (9 pts): Early Stopping

From your model in question 3, what do you think is the best stopping point? Note your answer below and why you think it. Then, use keras early stopping on your model from question 3 to have keras automatically do it for you.

Write what you think here:

By my observation, the validation loss of the loss would be increasingly unstable and start to increase instead of decreasing at epoch 12. So I think the model should early stop there.

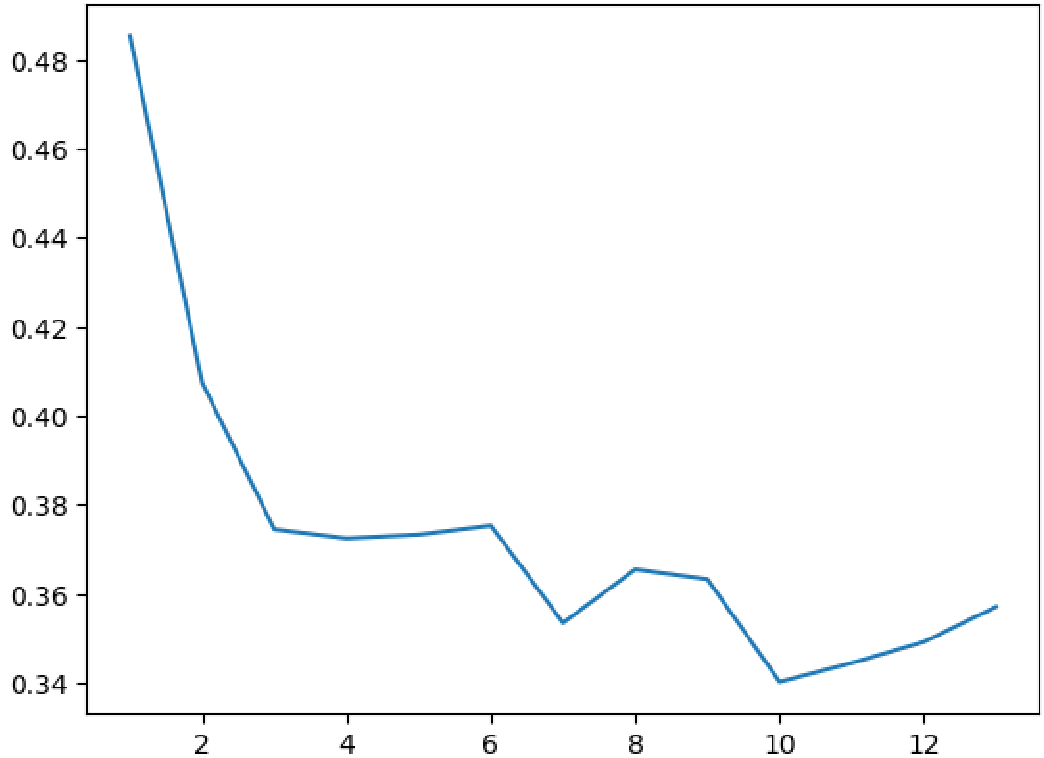
[15]:
```
from tensorflow.keras.callbacks import EarlyStopping
model_q4 = q3()

early_stopping = EarlyStopping(monitor='val_loss', patience=3,␣
 ↪restore_best_weights=True)

history_q4 = model_q4.fit(X_train, y_train, epochs=ep,
                          validation_data=(X_valid, y_valid),
                          batch_size=bs,
                          callbacks=[early_stopping])
```

```
Epoch 1/20
```

```
66/66 [==============================] - 2s 9ms/step - loss: 0.6375 - accuracy:
0.7849 - val_loss: 0.4853 - val_accuracy: 0.8180
Epoch 2/20
66/66 [==============================] - 0s 6ms/step - loss: 0.4248 - accuracy:
0.8477 - val_loss: 0.4074 - val_accuracy: 0.8464
Epoch 3/20
66/66 [==============================] - 0s 6ms/step - loss: 0.3554 - accuracy:
0.8730 - val_loss: 0.3745 - val_accuracy: 0.8662
Epoch 4/20
66/66 [==============================] - 0s 6ms/step - loss: 0.3244 - accuracy:
0.8805 - val_loss: 0.3725 - val_accuracy: 0.8651
Epoch 5/20
66/66 [==============================] - 0s 6ms/step - loss: 0.2910 - accuracy:
0.8929 - val_loss: 0.3733 - val_accuracy: 0.8687
Epoch 6/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2672 - accuracy:
0.9018 - val_loss: 0.3753 - val_accuracy: 0.8714
Epoch 7/20
66/66 [==============================] - 0s 6ms/step - loss: 0.2485 - accuracy:
0.9092 - val_loss: 0.3535 - val_accuracy: 0.8715
Epoch 8/20
66/66 [==============================] - 0s 6ms/step - loss: 0.2305 - accuracy:
0.9157 - val_loss: 0.3654 - val_accuracy: 0.8652
Epoch 9/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2096 - accuracy:
0.9245 - val_loss: 0.3633 - val_accuracy: 0.8730
Epoch 10/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2051 - accuracy:
0.9260 - val_loss: 0.3402 - val_accuracy: 0.8806
Epoch 11/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1870 - accuracy:
0.9320 - val_loss: 0.3444 - val_accuracy: 0.8820
Epoch 12/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1752 - accuracy:
0.9377 - val_loss: 0.3492 - val_accuracy: 0.8807
Epoch 13/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1616 - accuracy:
0.9416 - val_loss: 0.3571 - val_accuracy: 0.8780
```

[16]:
```
val_loss = history_q4.history["val_loss"]
plt.plot(np.arange(1,len(val_loss)+1),val_loss)
```

[16]:
```
[<matplotlib.lines.Line2D at 0x7f307012d3f0>]
```



3.0.4 Question 5 (9 pts): Reducing size of network

Starting with your model from question 3, try to regularize it by reducing size of network. State the process you went through on how you settled on your model.

Write about the process here:

I altered the hyperparameter to have found that 16 layers of relu causes underfitting after 50 epoches, and 2*28*28 = 1568 layers cause overfitting such that the validation loss is out of control over time. I think 128 layers should be a better size to fit the model.

[17]:
```
def q5():
    input_dim = 28*28

    model = tf.keras.Sequential([
        # I think this is the suitiable fitting layer quantity
        tf.keras.layers.Dense(128, activation='relu', input_shape=(input_dim,)),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0005),
                  loss='categorical_crossentropy', metrics=['accuracy'])
```

```
    return model
```

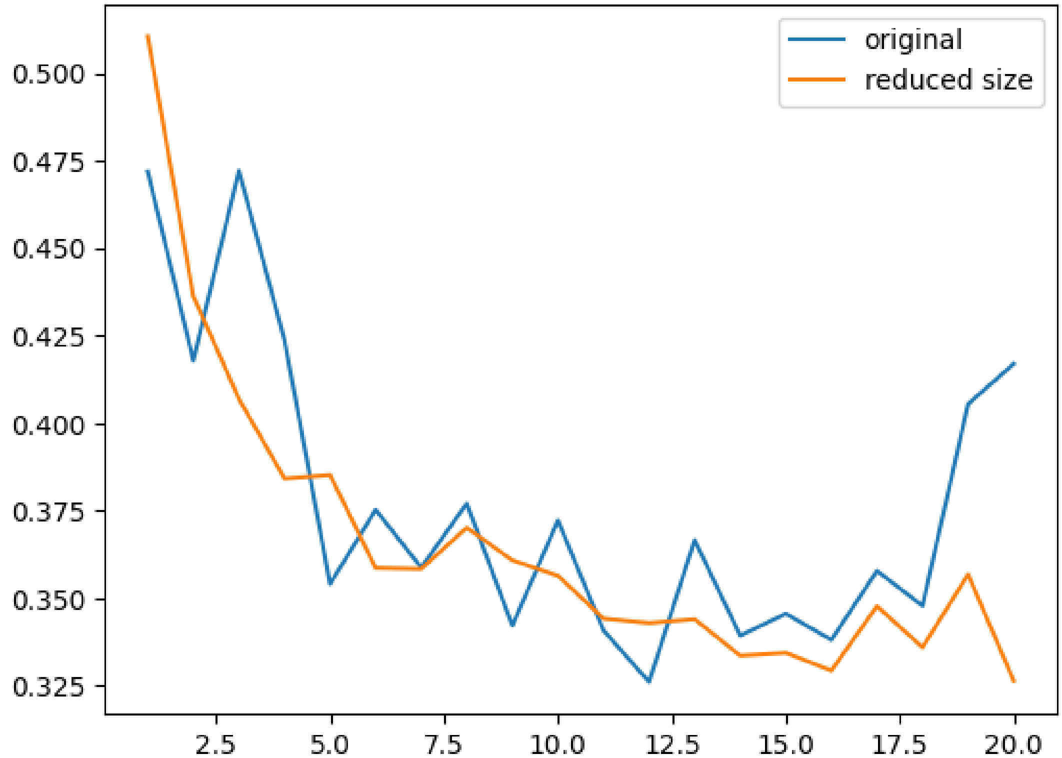
[18]:
```
model_q5 = q5()
history_q5 = model_q5.fit(X_train,y_train, epochs=ep, batch_size=bs,␣
 ↪validation_data=(X_valid, y_valid))
```

```
Epoch 1/20
66/66 [==============================] - 1s 8ms/step - loss: 0.7299 - accuracy:
0.7532 - val_loss: 0.5105 - val_accuracy: 0.8186
Epoch 2/20
66/66 [==============================] - 0s 5ms/step - loss: 0.4361 - accuracy:
0.8458 - val_loss: 0.4362 - val_accuracy: 0.8451
Epoch 3/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3827 - accuracy:
0.8624 - val_loss: 0.4069 - val_accuracy: 0.8529
Epoch 4/20
66/66 [==============================] - 0s 4ms/step - loss: 0.3505 - accuracy:
0.8755 - val_loss: 0.3841 - val_accuracy: 0.8631
Epoch 5/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3296 - accuracy:
0.8823 - val_loss: 0.3852 - val_accuracy: 0.8600
Epoch 6/20
66/66 [==============================] - 0s 4ms/step - loss: 0.3100 - accuracy:
0.8882 - val_loss: 0.3586 - val_accuracy: 0.8717
Epoch 7/20
66/66 [==============================] - 0s 4ms/step - loss: 0.2957 - accuracy:
0.8950 - val_loss: 0.3583 - val_accuracy: 0.8706
Epoch 8/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2833 - accuracy:
0.8975 - val_loss: 0.3701 - val_accuracy: 0.8664
Epoch 9/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2731 - accuracy:
0.9017 - val_loss: 0.3608 - val_accuracy: 0.8706
Epoch 10/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2623 - accuracy:
0.9074 - val_loss: 0.3564 - val_accuracy: 0.8723
Epoch 11/20
66/66 [==============================] - 0s 4ms/step - loss: 0.2518 - accuracy:
0.9112 - val_loss: 0.3441 - val_accuracy: 0.8758
Epoch 12/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2443 - accuracy:
0.9124 - val_loss: 0.3429 - val_accuracy: 0.8774
Epoch 13/20
66/66 [==============================] - 0s 4ms/step - loss: 0.2339 - accuracy:
0.9175 - val_loss: 0.3440 - val_accuracy: 0.8761
Epoch 14/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2277 - accuracy:
```

```
0.9196 - val_loss: 0.3335 - val_accuracy: 0.8800
Epoch 15/20
66/66 [==============================] - 0s 4ms/step - loss: 0.2210 - accuracy:
0.9235 - val_loss: 0.3344 - val_accuracy: 0.8787
Epoch 16/20
66/66 [==============================] - 0s 4ms/step - loss: 0.2151 - accuracy:
0.9248 - val_loss: 0.3293 - val_accuracy: 0.8813
Epoch 17/20
66/66 [==============================] - 0s 4ms/step - loss: 0.2080 - accuracy:
0.9279 - val_loss: 0.3477 - val_accuracy: 0.8788
Epoch 18/20
66/66 [==============================] - 0s 4ms/step - loss: 0.2024 - accuracy:
0.9299 - val_loss: 0.3359 - val_accuracy: 0.8813
Epoch 19/20
66/66 [==============================] - 0s 6ms/step - loss: 0.1966 - accuracy:
0.9311 - val_loss: 0.3567 - val_accuracy: 0.8743
Epoch 20/20
66/66 [==============================] - 0s 6ms/step - loss: 0.1909 - accuracy:
0.9331 - val_loss: 0.3264 - val_accuracy: 0.8845
```

[19]:
```
val_loss_q5 = history_q5.history["val_loss"]
val_loss_q3 = history_q3.history["val_loss"]
plt.plot(np.arange(1,len(val_loss_q3)+1),val_loss_q3, label="original")
plt.plot(np.arange(1,len(val_loss_q5)+1),val_loss_q5, label="reduced size")
plt.legend()
```

[19]:
```
<matplotlib.legend.Legend at 0x7f2ff776eb90>
```



3.0.5 Question 6 (9 pts): L1 Regularization

Starting with your model from question 3, try to regularize it by using by L1 regularization. State the process you went through on how you settled on your model.

Write about the process here: Note that our size of network is determined by the number of layers of relu. Here we have used too many layers of relu to have caused overfitting, so I added a l1 regularization to increase the sparseness. Through altering hyperparameter, I found 1e-7 a good parameter for l2 regularization.

[42]:
```
def q6():
    input_dim = 28*28

    model = tf.keras.Sequential([
        # L1 regularization
        tf.keras.layers.Dense(2*input_dim, activation='relu',␣
↪input_shape=(input_dim,),
                              kernel_regularizer=regularizers.l1(1e-8)),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0005),
                  loss='categorical_crossentropy', metrics=['accuracy'])
```

```
    return model
```

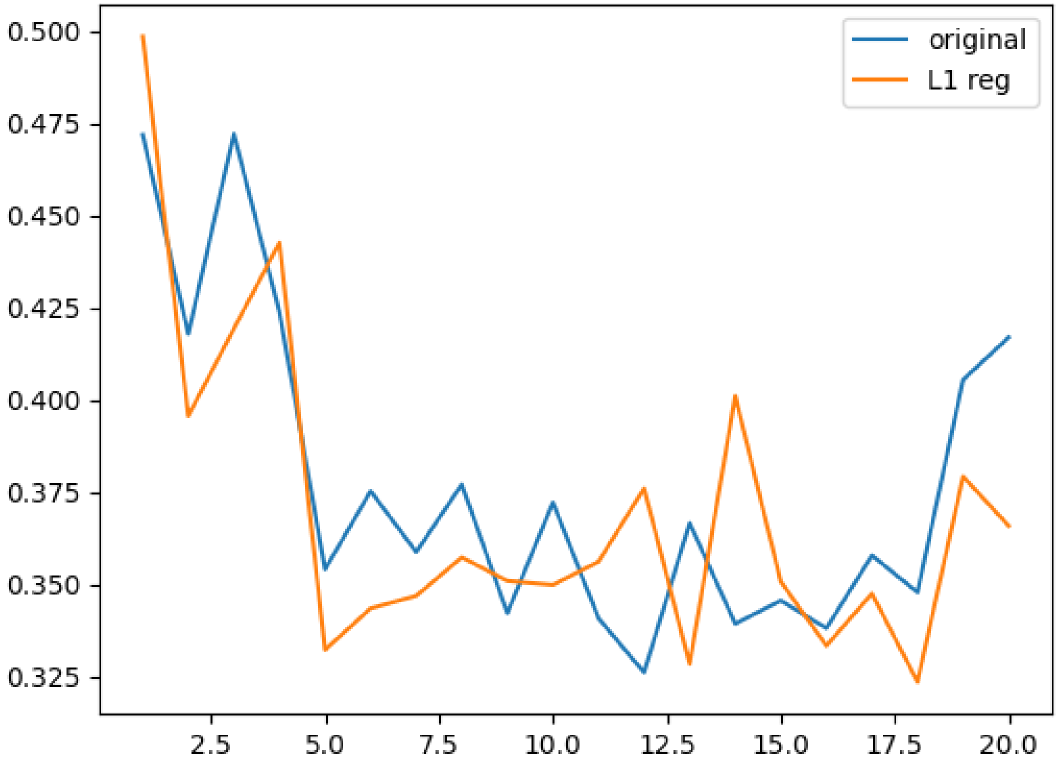
[46]:
```
model_q6 = q6()
history_q6 = model_q6.fit(X_train,y_train, epochs=ep, batch_size=bs,␣
 ↪validation_data=(X_valid, y_valid))
```

```
Epoch 1/20
66/66 [==============================] - 1s 12ms/step - loss: 0.6427 - accuracy:
0.7841 - val_loss: 0.4985 - val_accuracy: 0.8240
Epoch 2/20
66/66 [==============================] - 0s 7ms/step - loss: 0.4200 - accuracy:
0.8510 - val_loss: 0.3956 - val_accuracy: 0.8543
Epoch 3/20
66/66 [==============================] - 0s 8ms/step - loss: 0.3617 - accuracy:
0.8689 - val_loss: 0.4193 - val_accuracy: 0.8539
Epoch 4/20
66/66 [==============================] - 0s 7ms/step - loss: 0.3098 - accuracy:
0.8873 - val_loss: 0.4426 - val_accuracy: 0.8476
Epoch 5/20
66/66 [==============================] - 0s 6ms/step - loss: 0.2929 - accuracy:
0.8929 - val_loss: 0.3322 - val_accuracy: 0.8790
Epoch 6/20
66/66 [==============================] - 0s 6ms/step - loss: 0.2614 - accuracy:
0.9049 - val_loss: 0.3434 - val_accuracy: 0.8781
Epoch 7/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2441 - accuracy:
0.9133 - val_loss: 0.3468 - val_accuracy: 0.8730
Epoch 8/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2234 - accuracy:
0.9192 - val_loss: 0.3573 - val_accuracy: 0.8738
Epoch 9/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2101 - accuracy:
0.9235 - val_loss: 0.3509 - val_accuracy: 0.8782
Epoch 10/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2003 - accuracy:
0.9267 - val_loss: 0.3498 - val_accuracy: 0.8773
Epoch 11/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1858 - accuracy:
0.9337 - val_loss: 0.3561 - val_accuracy: 0.8739
Epoch 12/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1764 - accuracy:
0.9356 - val_loss: 0.3760 - val_accuracy: 0.8724
Epoch 13/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1604 - accuracy:
0.9432 - val_loss: 0.3283 - val_accuracy: 0.8860
Epoch 14/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1532 - accuracy:
```

```
0.9451 - val_loss: 0.4010 - val_accuracy: 0.8718
Epoch 15/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1433 - accuracy:
0.9489 - val_loss: 0.3507 - val_accuracy: 0.8798
Epoch 16/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1366 - accuracy:
0.9521 - val_loss: 0.3333 - val_accuracy: 0.8864
Epoch 17/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1264 - accuracy:
0.9569 - val_loss: 0.3474 - val_accuracy: 0.8851
Epoch 18/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1211 - accuracy:
0.9583 - val_loss: 0.3235 - val_accuracy: 0.8942
Epoch 19/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1127 - accuracy:
0.9629 - val_loss: 0.3792 - val_accuracy: 0.8732
Epoch 20/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1060 - accuracy:
0.9648 - val_loss: 0.3658 - val_accuracy: 0.8848
```

[47]:
```python
val_loss_q3 = history_q3.history["val_loss"]
plt.plot(np.arange(1,len(val_loss_q3)+1),val_loss_q3, label="original")
val_loss_q6 = history_q6.history["val_loss"]
plt.plot(np.arange(1,len(val_loss_q6)+1),val_loss_q6, label="L1 reg")
plt.legend()
```

[47]:
```
<matplotlib.legend.Legend at 0x7f2ff81afe50>
```



3.0.6 Question 7 (9 pts): L2 Regularization

Starting with your model from question 3, try to regularize it by using L2 regularization. State the process you went through on how you settled on your model.

Write about the process here: It turns out that through l1 regularization, the model would still be overfitting, though better than q3. So the lack of sparseness is not the only problem. Therefore we can try l2 regularization to distribute the weights better, increasing steadiness. Through altering hyperparameter, I found 1e-8 a good parameter for l2 regularization.

[25]:
```
def q7():
    input_dim = 28*28

    model = tf.keras.Sequential([
        # L2 regularization
        tf.keras.layers.Dense(2 * input_dim, activation='relu',␣
 ↪input_shape=(input_dim,),
                              kernel_regularizer=regularizers.l2(1e-8)),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0005),
                  loss='categorical_crossentropy', metrics=['accuracy'])
```

```
    return model
```

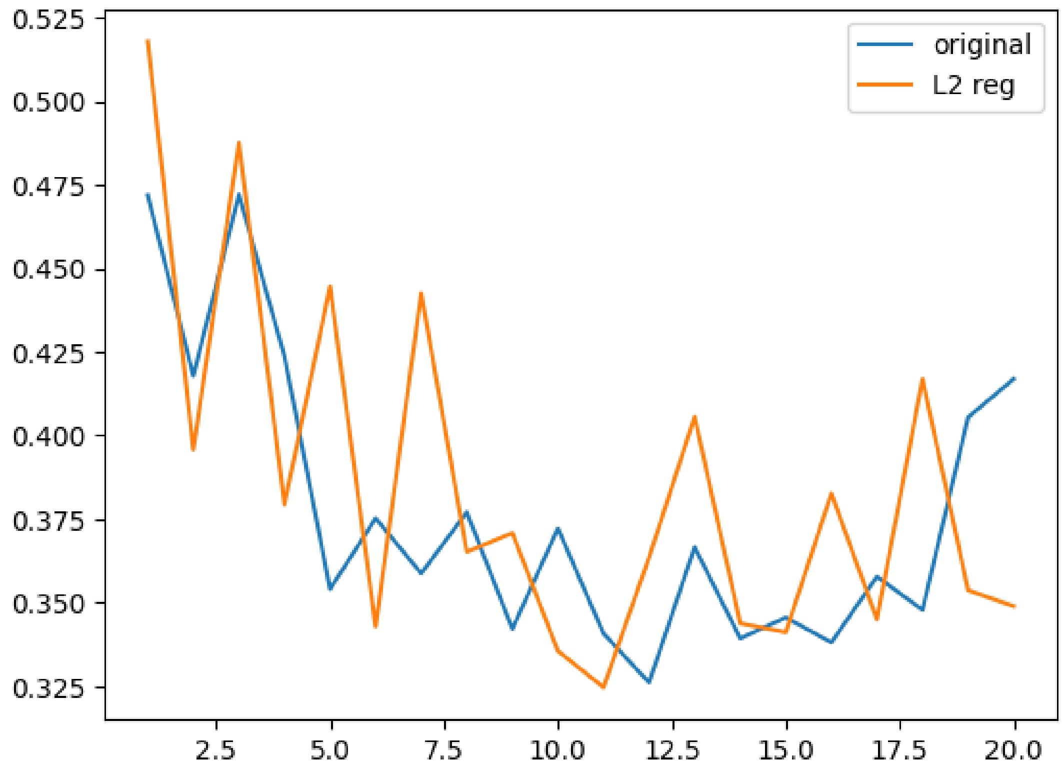
[33]:
```
model_q7 = q7()
history_q7 = model_q7.fit(X_train,y_train, epochs=ep, batch_size=bs,␣
 ↪validation_data=(X_valid, y_valid))
```

```
Epoch 1/20
66/66 [==============================] - 1s 12ms/step - loss: 0.6282 - accuracy:
0.7851 - val_loss: 0.5180 - val_accuracy: 0.8119
Epoch 2/20
66/66 [==============================] - 1s 8ms/step - loss: 0.4177 - accuracy:
0.8497 - val_loss: 0.3958 - val_accuracy: 0.8598
Epoch 3/20
66/66 [==============================] - 0s 6ms/step - loss: 0.3515 - accuracy:
0.8725 - val_loss: 0.4877 - val_accuracy: 0.8182
Epoch 4/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3183 - accuracy:
0.8854 - val_loss: 0.3793 - val_accuracy: 0.8614
Epoch 5/20
66/66 [==============================] - 0s 6ms/step - loss: 0.2872 - accuracy:
0.8943 - val_loss: 0.4446 - val_accuracy: 0.8452
Epoch 6/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2679 - accuracy:
0.9017 - val_loss: 0.3429 - val_accuracy: 0.8767
Epoch 7/20
66/66 [==============================] - 0s 6ms/step - loss: 0.2433 - accuracy:
0.9107 - val_loss: 0.4425 - val_accuracy: 0.8523
Epoch 8/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2252 - accuracy:
0.9179 - val_loss: 0.3652 - val_accuracy: 0.8743
Epoch 9/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2081 - accuracy:
0.9242 - val_loss: 0.3708 - val_accuracy: 0.8710
Epoch 10/20
66/66 [==============================] - 0s 6ms/step - loss: 0.1950 - accuracy:
0.9306 - val_loss: 0.3355 - val_accuracy: 0.8830
Epoch 11/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1861 - accuracy:
0.9325 - val_loss: 0.3246 - val_accuracy: 0.8863
Epoch 12/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1694 - accuracy:
0.9392 - val_loss: 0.3635 - val_accuracy: 0.8744
Epoch 13/20
66/66 [==============================] - 0s 6ms/step - loss: 0.1601 - accuracy:
0.9423 - val_loss: 0.4056 - val_accuracy: 0.8658
Epoch 14/20
66/66 [==============================] - 0s 6ms/step - loss: 0.1570 - accuracy:
```

```
0.9449 - val_loss: 0.3438 - val_accuracy: 0.8856
Epoch 15/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1425 - accuracy:
0.9502 - val_loss: 0.3411 - val_accuracy: 0.8827
Epoch 16/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1380 - accuracy:
0.9516 - val_loss: 0.3827 - val_accuracy: 0.8751
Epoch 17/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1255 - accuracy:
0.9556 - val_loss: 0.3449 - val_accuracy: 0.8888
Epoch 18/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1191 - accuracy:
0.9580 - val_loss: 0.4169 - val_accuracy: 0.8730
Epoch 19/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1118 - accuracy:
0.9613 - val_loss: 0.3537 - val_accuracy: 0.8873
Epoch 20/20
66/66 [==============================] - 0s 5ms/step - loss: 0.1124 - accuracy:
0.9606 - val_loss: 0.3489 - val_accuracy: 0.8894
```

[34]:
```
val_loss_q3 = history_q3.history["val_loss"]
plt.plot(np.arange(1,len(val_loss_q3)+1),val_loss_q3, label="original")
val_loss_q7 = history_q7.history["val_loss"]
plt.plot(np.arange(1,len(val_loss_q7)+1),val_loss_q7, label="L2 reg")
plt.legend()
```

[34]:
```
<matplotlib.legend.Legend at 0x7f2ff84beef0>
```



3.0.7 Question 8 (9 pts): Dropout Regularization

Starting with your model from question 3, try to regularize it by using dropout. State the process you went through on how you settled on your model.

Write about the process here:

It turns out that through l2 regularization, the model would still be overfitting. So we can try another dropout regularation after relu the model. I keep altering hyperparameter to have found that dropout 0.5 of features after relu is a good parameter.

[30]:
```
from tensorflow.keras.layers import Dropout

def q8():
    input_dim = 28*28

    model = tf.keras.Sequential([
        tf.keras.layers.Dense(2 * input_dim, activation='relu',␣
 ↪input_shape=(input_dim,)),
        Dropout(0.5),
        Dropout(0.5),
        tf.keras.layers.Dense(10, activation='softmax')
    ])
```

```
    model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0005),
                  loss='categorical_crossentropy', metrics=['accuracy'])
    return model
```

[31]:
```
model_q8 = q8()
history_q8 = model_q8.fit(X_train,y_train, epochs=ep, batch_size=bs,␣
 ↪validation_data=(X_valid, y_valid))
```

```
Epoch 1/20
66/66 [==============================] - 2s 9ms/step - loss: 0.7805 - accuracy:
0.7492 - val_loss: 0.4670 - val_accuracy: 0.8393
Epoch 2/20
66/66 [==============================] - 0s 6ms/step - loss: 0.5243 - accuracy:
0.8216 - val_loss: 0.4296 - val_accuracy: 0.8493
Epoch 3/20
66/66 [==============================] - 0s 6ms/step - loss: 0.4681 - accuracy:
0.8376 - val_loss: 0.3913 - val_accuracy: 0.8576
Epoch 4/20
66/66 [==============================] - 0s 6ms/step - loss: 0.4319 - accuracy:
0.8491 - val_loss: 0.3920 - val_accuracy: 0.8575
Epoch 5/20
66/66 [==============================] - 0s 6ms/step - loss: 0.4061 - accuracy:
0.8560 - val_loss: 0.4031 - val_accuracy: 0.8573
Epoch 6/20
66/66 [==============================] - 0s 6ms/step - loss: 0.3875 - accuracy:
0.8636 - val_loss: 0.3580 - val_accuracy: 0.8725
Epoch 7/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3706 - accuracy:
0.8674 - val_loss: 0.3662 - val_accuracy: 0.8701
Epoch 8/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3538 - accuracy:
0.8739 - val_loss: 0.3545 - val_accuracy: 0.8732
Epoch 9/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3404 - accuracy:
0.8796 - val_loss: 0.3402 - val_accuracy: 0.8806
Epoch 10/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3309 - accuracy:
0.8824 - val_loss: 0.3474 - val_accuracy: 0.8751
Epoch 11/20
66/66 [==============================] - 0s 6ms/step - loss: 0.3271 - accuracy:
0.8822 - val_loss: 0.3388 - val_accuracy: 0.8793
Epoch 12/20
66/66 [==============================] - 0s 6ms/step - loss: 0.3133 - accuracy:
0.8885 - val_loss: 0.3429 - val_accuracy: 0.8769
Epoch 13/20
66/66 [==============================] - 0s 6ms/step - loss: 0.3089 - accuracy:
0.8888 - val_loss: 0.3347 - val_accuracy: 0.8819
```

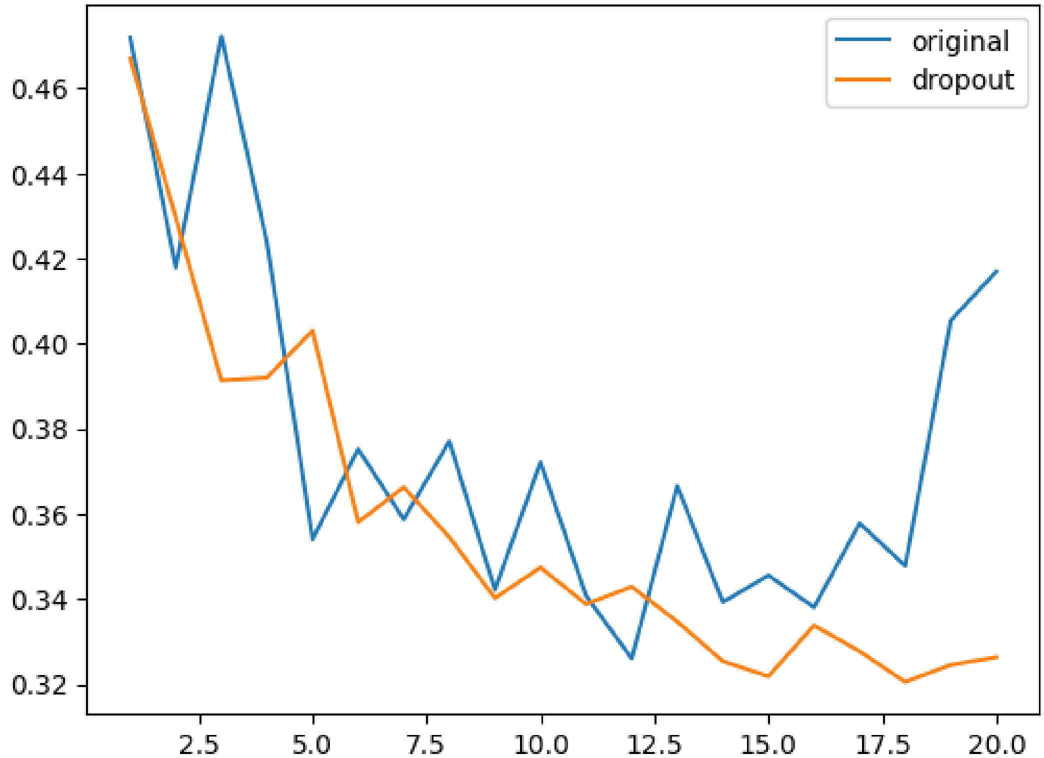
```
Epoch 14/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3000 - accuracy:
0.8911 - val_loss: 0.3254 - val_accuracy: 0.8865
Epoch 15/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2930 - accuracy:
0.8952 - val_loss: 0.3218 - val_accuracy: 0.8871
Epoch 16/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2867 - accuracy:
0.8944 - val_loss: 0.3338 - val_accuracy: 0.8861
Epoch 17/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2825 - accuracy:
0.8981 - val_loss: 0.3277 - val_accuracy: 0.8862
Epoch 18/20
66/66 [==============================] - 0s 5ms/step - loss: 0.2749 - accuracy:
0.9007 - val_loss: 0.3205 - val_accuracy: 0.8893
Epoch 19/20
66/66 [==============================] - 0s 6ms/step - loss: 0.2743 - accuracy:
0.9014 - val_loss: 0.3245 - val_accuracy: 0.8882
Epoch 20/20
66/66 [==============================] - 0s 6ms/step - loss: 0.2713 - accuracy:
0.9020 - val_loss: 0.3263 - val_accuracy: 0.8865
```

[32]:

```python
val_loss_q3 = history_q3.history["val_loss"]
plt.plot(np.arange(1,len(val_loss_q3)+1),val_loss_q3, label="original")
val_loss_q8 = history_q8.history["val_loss"]
plt.plot(np.arange(1,len(val_loss_q8)+1),val_loss_q8, label="dropout")
plt.legend()
```

[32]:

```
<matplotlib.legend.Legend at 0x7f2ff7c71c00>
```



3.0.8 Question 9 (9 pts): Combine 5-8

Starting with your model from question 3, try to regularize it by using a combination of the methods used in questions 5-8. Make changes to hyperparameters and have the model stop at a good epoch. State the process you went through on how you settled on your model.

Write about the process here:

L2 regularization does not seem to be helpful, while l1 is helpful but overfitting still exist by merely using l1 regularizaion. Drop out regularizaion is helpful, so we can first pick a suitable size of network, then combine l1 and dropout regularization, and then alter hyperparameters.

[56]:
```
from tensorflow.keras.layers import Dropout
def q9():
    input_dim = 28*28
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(128, activation='relu', input_shape=(input_dim,)
        ,kernel_regularizer=regularizers.l1(1e-7)),
        Dropout(0.5),
        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer=tf.keras.optimizers.RMSprop(learning_rate=0.0005),
```

```
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])

    return model
```

[57]:
```
model_q9 = q9()

# fill in the arguments for model_q9.fit() below
history_q9 = model_q9.fit(X_train,y_train, epochs=ep, batch_size=bs,
                          validation_data=(X_valid, y_valid),␣
 ↪callbacks=[early_stopping])
```

```
Epoch 1/20
66/66 [==============================] - 1s 7ms/step - loss: 0.9727 - accuracy:
0.6780 - val_loss: 0.5187 - val_accuracy: 0.8133
Epoch 2/20
66/66 [==============================] - 0s 4ms/step - loss: 0.6067 - accuracy:
0.7915 - val_loss: 0.4559 - val_accuracy: 0.8350
Epoch 3/20
66/66 [==============================] - 0s 5ms/step - loss: 0.5280 - accuracy:
0.8173 - val_loss: 0.4279 - val_accuracy: 0.8435
Epoch 4/20
66/66 [==============================] - 0s 5ms/step - loss: 0.4889 - accuracy:
0.8301 - val_loss: 0.4097 - val_accuracy: 0.8525
Epoch 5/20
66/66 [==============================] - 0s 4ms/step - loss: 0.4560 - accuracy:
0.8388 - val_loss: 0.3949 - val_accuracy: 0.8590
Epoch 6/20
66/66 [==============================] - 0s 5ms/step - loss: 0.4325 - accuracy:
0.8469 - val_loss: 0.3893 - val_accuracy: 0.8614
Epoch 7/20
66/66 [==============================] - 0s 4ms/step - loss: 0.4208 - accuracy:
0.8502 - val_loss: 0.3808 - val_accuracy: 0.8648
Epoch 8/20
66/66 [==============================] - 0s 4ms/step - loss: 0.4028 - accuracy:
0.8574 - val_loss: 0.3711 - val_accuracy: 0.8679
Epoch 9/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3927 - accuracy:
0.8613 - val_loss: 0.3683 - val_accuracy: 0.8685
Epoch 10/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3866 - accuracy:
0.8630 - val_loss: 0.3643 - val_accuracy: 0.8692
Epoch 11/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3749 - accuracy:
0.8650 - val_loss: 0.3639 - val_accuracy: 0.8668
Epoch 12/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3688 - accuracy:
0.8697 - val_loss: 0.3570 - val_accuracy: 0.8717
```

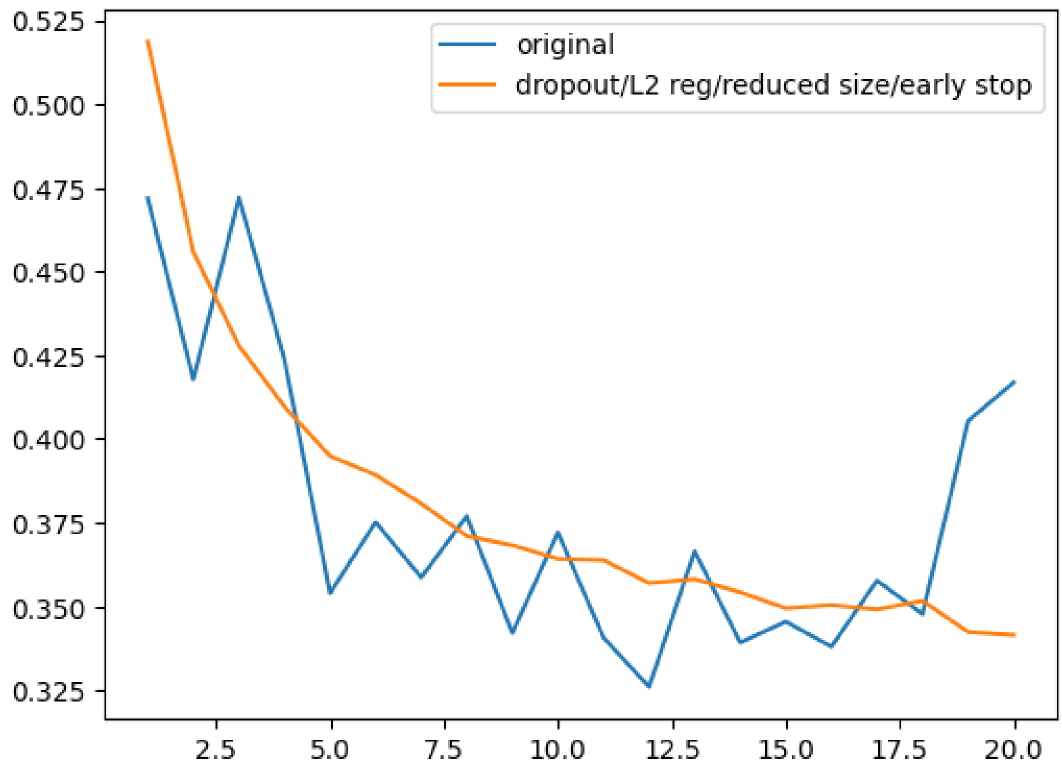
```
Epoch 13/20
66/66 [==============================] - 0s 4ms/step - loss: 0.3580 - accuracy:
0.8715 - val_loss: 0.3582 - val_accuracy: 0.8737
Epoch 14/20
66/66 [==============================] - 0s 4ms/step - loss: 0.3503 - accuracy:
0.8765 - val_loss: 0.3543 - val_accuracy: 0.8714
Epoch 15/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3444 - accuracy:
0.8772 - val_loss: 0.3495 - val_accuracy: 0.8742
Epoch 16/20
66/66 [==============================] - 0s 5ms/step - loss: 0.3389 - accuracy:
0.8789 - val_loss: 0.3505 - val_accuracy: 0.8752
Epoch 17/20
66/66 [==============================] - 0s 4ms/step - loss: 0.3337 - accuracy:
0.8815 - val_loss: 0.3492 - val_accuracy: 0.8745
Epoch 18/20
66/66 [==============================] - 0s 6ms/step - loss: 0.3319 - accuracy:
0.8814 - val_loss: 0.3517 - val_accuracy: 0.8736
Epoch 19/20
66/66 [==============================] - 0s 6ms/step - loss: 0.3262 - accuracy:
0.8822 - val_loss: 0.3425 - val_accuracy: 0.8782
Epoch 20/20
66/66 [==============================] - 0s 6ms/step - loss: 0.3225 - accuracy:
0.8851 - val_loss: 0.3415 - val_accuracy: 0.8774
```

[58]:

```
val_loss_q3 = history_q3.history["val_loss"]
plt.plot(np.arange(1,len(val_loss_q3)+1),val_loss_q3, label="original")
val_loss_q9 = history_q9.history["val_loss"]
plt.plot(np.arange(1,len(val_loss_q9)+1),val_loss_q9, label="dropout/L2 reg/
 ↪reduced size/early stop")
plt.legend()
```

[58]:

```
<matplotlib.legend.Legend at 0x7f2ff76576d0>
```



3.0.9 Question 10 (9 pts): Test loss of Model 2-9

Take the models from question 2 to 9 and find their test loss.

[59]:
```
models_list = [globals()[f'model_q{i}'] for i in range(2, 10)]
```

[60]:
```
test_losses = []
i = 2
for model in models_list:
    test_loss, test_accuracy = model.evaluate(X_test, y_test)
    test_losses.append(round(test_loss,5))
    print(f"Test Loss of model {i}: {test_loss}, Test Accuracy of model {i}:␣
 ↪{test_accuracy}")
    i+=1
```

```
563/563 [==============================] - 1s 2ms/step - loss: 0.4167 -
accuracy: 0.8524
Test Loss of model 2: 0.416743665933609, Test Accuracy of model 2:
0.8524444699287415
563/563 [==============================] - 2s 3ms/step - loss: 0.4328 -
accuracy: 0.8769
Test Loss of model 3: 0.4328499138355255, Test Accuracy of model 3:
0.8769444227218628
```

```
563/563 [==============================] - 1s 2ms/step - loss: 0.3489 -
accuracy: 0.8799
Test Loss of model 4: 0.34888505935668945, Test Accuracy of model 4:
0.8799444437026978
563/563 [==============================] - 1s 2ms/step - loss: 0.3278 -
accuracy: 0.8855
Test Loss of model 5: 0.3278467655181885, Test Accuracy of model 5:
0.8855000138282776
563/563 [==============================] - 2s 3ms/step - loss: 0.3670 -
accuracy: 0.8883
Test Loss of model 6: 0.36703479290008545, Test Accuracy of model 6:
0.8882777690887451
563/563 [==============================] - 1s 2ms/step - loss: 0.3577 -
accuracy: 0.8901
Test Loss of model 7: 0.35768556594848633, Test Accuracy of model 7:
0.8900555372238159
563/563 [==============================] - 2s 4ms/step - loss: 0.3362 -
accuracy: 0.8877
Test Loss of model 8: 0.3362028896808624, Test Accuracy of model 8:
0.8877221941947937
563/563 [==============================] - 1s 3ms/step - loss: 0.3403 -
accuracy: 0.8826
Test Loss of model 9: 0.34026598930358887, Test Accuracy of model 9:
0.8825555443763733
```

3.0.10 Question 11 (10 pts):

If you had to use one of these models, which one would you use and why?

Write answer here:

I would use the combined model. Though the early stopping has a better performace on the testing, combined model is model stable as we can view from the loss graph that the validation loss goes down smoothly.