Activity 1.3 : Regularization

Objective(s):

This activity aims to demonstrate how to apply regularization in neural networks

Intended Learning Outcomes (ILOs):

- Demonstrate how to build and train neural networks with regularization
- Demonstrate how to visualize the model with regularization
- Evaluate the result of model with regularization

Resources:

- Jupyter Notebook
- MNIST

Procedures

Load the necessary libraries

In [9]:
```
from __future__ import print_function

import keras
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import RMSprop

import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Load the data, shuffled and split between train and test sets

In [10]:
```
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz (https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz)
11490434/11490434 [==============================] - 0s 0us/step
```

Get the size of the sample train data

In [11]:
```
x_train[0].shape
```

Out[11]:
```
(28, 28)
```

Check the sample train data

In [12]:
```
x_train[333]
```

Out[12]: ndarray (28, 28) show data

```
array([[  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  87, 138,
        170, 253, 201, 244, 212, 222, 138,  86,  22,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  95, 253, 252,
        252, 252, 252, 253, 252, 252, 252, 252, 245,  80,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  68, 246, 205,  69,
         69,  69,  69,  69,  69,  69,  69, 205, 253, 240,  50,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0, 187, 252, 218,  34,
          0,   0,   0,   0,   0,   0,   0, 116, 253, 252,  69,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0, 116, 248, 252, 253,  92,
          0,   0,   0,   0,   0,   0,  95, 230, 253, 157,   6,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0, 116, 249, 253, 189,  42,
          0,   0,   0,   0,  36, 170, 253, 243, 158,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0, 133, 252, 245, 140,
         34,   0,   0,  57, 219, 252, 235,  60,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,  25, 205, 253, 252,
        234, 184, 184, 253, 240, 100,  44,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  21, 161, 219,
        252, 252, 252, 234,  37,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 203,
        252, 252, 252, 251, 135,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   9,  76, 255, 253,
        205, 168, 220, 255, 253, 137,   5,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0, 114, 252, 249, 132,
```

```
          25,   0,   0, 180, 252, 252,  45,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,  51, 220, 252, 199,   0,
           0,   0,   0,  38, 186, 252, 154,   7,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0, 184, 252, 252,  21,   0,
           0,   0,   0,   0,  67, 252, 252,  22,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0, 184, 252, 200,   0,   0,
           0,   0,   0,   0,  47, 252, 252,  22,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0, 185, 253, 201,   0,   0,
           0,   0,   0,   3, 118, 253, 245,  21,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0, 163, 252, 252,   0,   0,
           0,   0,   0,  97, 252, 252,  87,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,  51, 240, 252, 123,  70,
          70, 112, 184, 222, 252, 170,  13,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0, 165, 252, 253, 252,
         252, 252, 252, 245, 139,  13,   0,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   9,  75, 253, 252,
         221, 137, 137,  21,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0],
        [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
           0,   0]], dtype=uint8)
```

Check the corresponding label in the training set

In [13]:

```
y_train[333]
```

Out[13]: 8

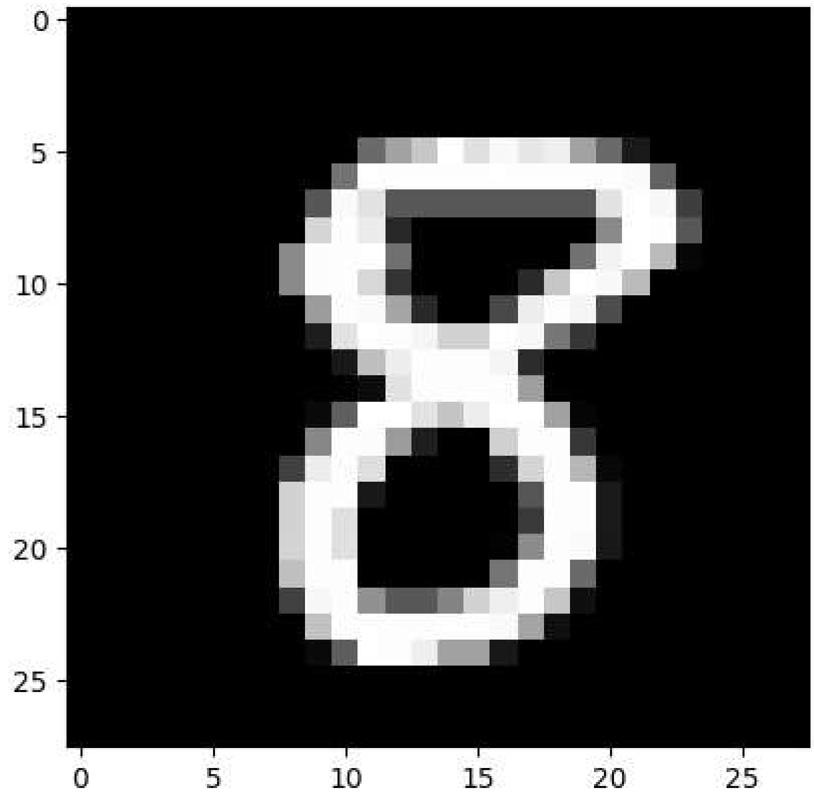
Check the actual image

In [14]:

```
plt.imshow(x_train[333], cmap='Greys_r')
```

Out[14]:

```
<matplotlib.image.AxesImage at 0x7dc1316f5090>
```



Check the shape of the x_train and x_test

In [15]:

```
print(x_train.shape, 'train samples')
print(x_test.shape, 'test samples')
```

```
(60000, 28, 28) train samples
(10000, 28, 28) test samples
```

- Convert the x_train and x_test
- Cast the numbers to floats
- Normalize the inputs

In [16]:

```
x_train = x_train.reshape(len(x_train), 28*28)
x_test = x_test.reshape(len(x_test), 28*28)


x_train = x_train.astype('float32')
x_test = x_test.astype('float32')


x_train /= 255
x_test /= 255
```

Convert class vectors to binary class matrices

In [17]:

```
num_classes = 10
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)

y_train[333]  # now the digit k is represented by a 1 in the kth entry (0-inde
```

Out[17]:

```
array([0., 0., 0., 0., 0., 0., 0., 0., 1., 0.], dtype=float32)
```

- Build the model with two hidden layers of size 512.
- Use dropout of 0.2
- Check the model summary

In [18]:

```
model = Sequential()
model.add(Dense(64, activation='relu', input_shape=(784,)))
model.add(Dropout(0.2))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(10, activation='softmax'))
```

In [19]:
```
model.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 dense_3 (Dense)             (None, 64)                50240     
                                                                 
 dropout_2 (Dropout)         (None, 64)                0         
                                                                 
 dense_4 (Dense)             (None, 64)                4160      
                                                                 
 dropout_3 (Dropout)         (None, 64)                0         
                                                                 
 dense_5 (Dense)             (None, 10)                650       
                                                                 
=================================================================
Total params: 55050 (215.04 KB)
Trainable params: 55050 (215.04 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compile the model using learning rate of 0.001 and optimizer of RMSprop

In [20]:
```
learning_rate = .001
model.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(lr=learning_rate),
              metrics=['accuracy'])
batch_size = 128  # mini-batch with 128 examples
epochs = 30
history = model.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(x_test, y_test))
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate
` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

```
Epoch 1/30
469/469 [==============================] - 4s 6ms/step - loss: 0.5245 - accur
acy: 0.8414 - val_loss: 0.2130 - val_accuracy: 0.9357
Epoch 2/30
469/469 [==============================] - 2s 4ms/step - loss: 0.2577 - accur
acy: 0.9237 - val_loss: 0.1580 - val_accuracy: 0.9529
Epoch 3/30
469/469 [==============================] - 2s 4ms/step - loss: 0.2059 - accur
acy: 0.9390 - val_loss: 0.1224 - val_accuracy: 0.9618
Epoch 4/30
469/469 [==============================] - 2s 5ms/step - loss: 0.1776 - accur
acy: 0.9474 - val_loss: 0.1163 - val_accuracy: 0.9629
Epoch 5/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1619 - accur
acy: 0.9529 - val_loss: 0.1059 - val_accuracy: 0.9669
Epoch 6/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1475 - accur
acy: 0.9562 - val_loss: 0.1049 - val_accuracy: 0.9689
Epoch 7/30
469/469 [==============================] - 3s 6ms/step - loss: 0.1351 - accur
acy: 0.9587 - val_loss: 0.1040 - val_accuracy: 0.9686
Epoch 8/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1289 - accur
acy: 0.9612 - val_loss: 0.0964 - val_accuracy: 0.9702
Epoch 9/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1209 - accur
acy: 0.9637 - val_loss: 0.0920 - val_accuracy: 0.9728
Epoch 10/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1176 - accur
acy: 0.9649 - val_loss: 0.0923 - val_accuracy: 0.9715
Epoch 11/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1121 - accur
acy: 0.9664 - val_loss: 0.0902 - val_accuracy: 0.9733
Epoch 12/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1078 - accur
acy: 0.9671 - val_loss: 0.0919 - val_accuracy: 0.9738
Epoch 13/30
469/469 [==============================] - 2s 4ms/step - loss: 0.1038 - accur
acy: 0.9683 - val_loss: 0.0909 - val_accuracy: 0.9761
Epoch 14/30
469/469 [==============================] - 2s 5ms/step - loss: 0.1026 - accur
acy: 0.9691 - val_loss: 0.0896 - val_accuracy: 0.9758
Epoch 15/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0982 - accur
acy: 0.9715 - val_loss: 0.0891 - val_accuracy: 0.9761
Epoch 16/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0957 - accur
acy: 0.9710 - val_loss: 0.0915 - val_accuracy: 0.9759
Epoch 17/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0944 - accur
acy: 0.9717 - val_loss: 0.0886 - val_accuracy: 0.9753
Epoch 18/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0908 - accur
acy: 0.9721 - val_loss: 0.0914 - val_accuracy: 0.9767
Epoch 19/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0913 - accur
acy: 0.9727 - val_loss: 0.0882 - val_accuracy: 0.9751
```

```
Epoch 20/30
469/469 [==============================] - 3s 5ms/step - loss: 0.0860 - accur
acy: 0.9742 - val_loss: 0.0870 - val_accuracy: 0.9762
Epoch 21/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0861 - accur
acy: 0.9737 - val_loss: 0.0918 - val_accuracy: 0.9760
Epoch 22/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0861 - accur
acy: 0.9739 - val_loss: 0.0952 - val_accuracy: 0.9755
Epoch 23/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0862 - accur
acy: 0.9744 - val_loss: 0.0912 - val_accuracy: 0.9756
Epoch 24/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0834 - accur
acy: 0.9751 - val_loss: 0.0946 - val_accuracy: 0.9768
Epoch 25/30
469/469 [==============================] - 3s 6ms/step - loss: 0.0841 - accur
acy: 0.9753 - val_loss: 0.0958 - val_accuracy: 0.9746
Epoch 26/30
469/469 [==============================] - 3s 6ms/step - loss: 0.0804 - accur
acy: 0.9756 - val_loss: 0.0993 - val_accuracy: 0.9746
Epoch 27/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0821 - accur
acy: 0.9748 - val_loss: 0.0999 - val_accuracy: 0.9754
Epoch 28/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0816 - accur
acy: 0.9764 - val_loss: 0.0980 - val_accuracy: 0.9753
Epoch 29/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0800 - accur
acy: 0.9760 - val_loss: 0.1030 - val_accuracy: 0.9759
Epoch 30/30
469/469 [==============================] - 2s 4ms/step - loss: 0.0787 - accur
acy: 0.9763 - val_loss: 0.1031 - val_accuracy: 0.9750
```

Use Keras evaluate function to evaluate performance on the test set

In [21]:

```
score = model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.10310786217451096
Test accuracy: 0.9750000238418579
```

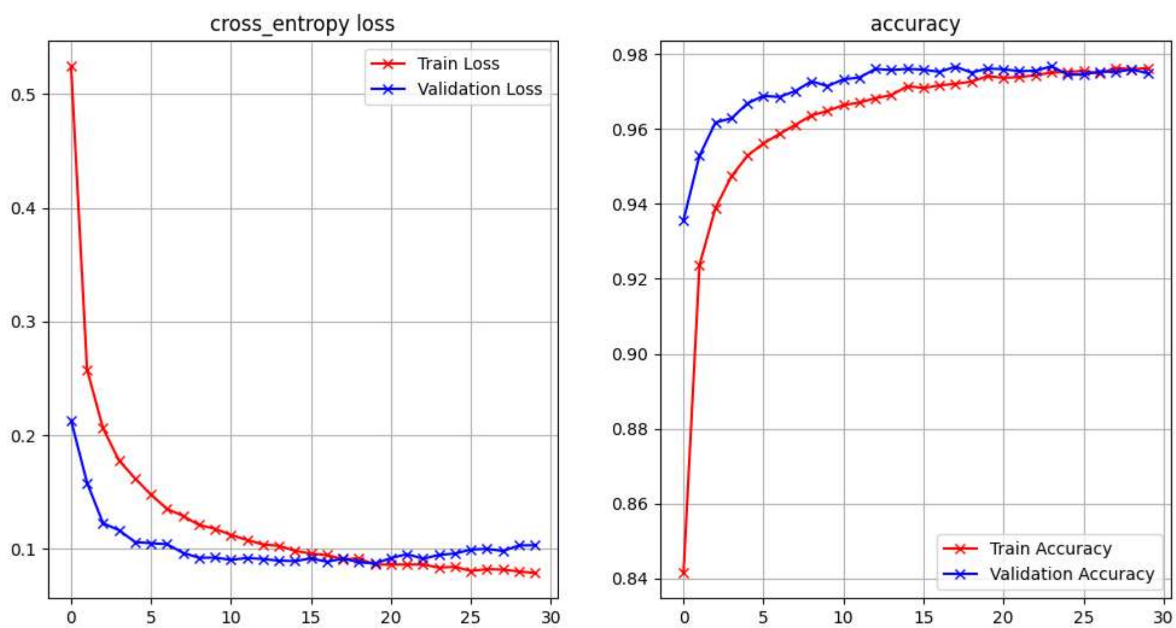
Interpret the result

This model uses three layers which are, first, one to receive input, two hidden layers to process the data, and one to produce output. The dropout layers help prevent the model from becoming too specialized to the training data. The model is trained to improve its ability to correctly categorize data by adjusting its parameters over multiple iterations. It showed impressive performance on unseen data, as shown in a test loss of approximately 0.0907 and a test accuracy of roughly 97.69%. The low test loss means that the model's predictions closely align with the actual values, showing its capability to make accurate predictions.

In [22]:
```
def plot_loss_accuracy(history):
    fig = plt.figure(figsize=(12, 6))
    ax = fig.add_subplot(1, 2, 1)
    ax.plot(history.history["loss"],'r-x', label="Train Loss")
    ax.plot(history.history["val_loss"],'b-x', label="Validation Loss")
    ax.legend()
    ax.set_title('cross_entropy loss')
    ax.grid(True)


    ax = fig.add_subplot(1, 2, 2)
    ax.plot(history.history["accuracy"],'r-x', label="Train Accuracy")
    ax.plot(history.history["val_accuracy"],'b-x', label="Validation Accuracy"
    ax.legend()
    ax.set_title('accuracy')
    ax.grid(True)


plot_loss_accuracy(history)
```



Interpret the result

In here, the training loss is lower than the validation loss, it tells me that my model might be getting too good at the training data but struggles with new data. Even though the accuracy looks good for both the training and validation sets, the higher validation loss shows that my model isn't doing as well with new data. To fix this, I might need to use some other adjustment in the hyperparams like dropout or adjust the model to help it do better with new data.

Supplementary Activity

- Use the Keras "Sequential" functionality to build a new model (model_1) with the following specifications:

1. Two hidden layers.
2. First hidden layer of size 400 and second of size 300
3. Dropout of .4 at each layer
4. How many parameters does your model have? How does it compare with the previous model?
5. Train this model for 20 epochs with RMSProp at a learning rate of .001 and a batch size of 128
6. Use at least two regularization techniques and apply it to the new model (model_2)
7. Train this model for your preferred epochs , learning rate, batch size and optimizer
8. Compare the accuracy and loss (training and validation) of model_1 and model_2

model_1

In [23]:
```
model_1 = Sequential()
model_1.add(Dense(400, activation='relu', input_shape=(784,)))
model_1.add(Dropout(0.4))
model_1.add(Dense(300, activation='relu'))
model_1.add(Dropout(0.4))
model_1.add(Dense(10, activation='softmax'))
```

In [24]:
```
model_1.summary()
```

```
Model: "sequential_3"

 Layer (type)                Output Shape              Param #   
=================================================================
 dense_6 (Dense)             (None, 400)               314000    
                                                                 
 dropout_4 (Dropout)         (None, 400)               0         
                                                                 
 dense_7 (Dense)             (None, 300)               120300    
                                                                 
 dropout_5 (Dropout)         (None, 300)               0         
                                                                 
 dense_8 (Dense)             (None, 10)                3010      
                                                                 
=================================================================
Total params: 437310 (1.67 MB)
Trainable params: 437310 (1.67 MB)
Non-trainable params: 0 (0.00 Byte)
```

This model (model_1) has a higher number of parameters of 437,000 compared to the previous model, which had around 55,000 parameters. This is mainly due to the larger size of the hidden layers in this model that are 400 and 300 neurons compared to the previous model. Later on, I can assess their performance and the impact of regularization on the model's generalization ability.

In [25]:
```
learning_rate = .001
model_1.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(lr=learning_rate),
              metrics=['accuracy'])
batch_size = 128
epochs = 20
history_1 = model_1.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(x_test, y_test))
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate
` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

```
Epoch 1/20
469/469 [==============================] - 7s 13ms/step - loss: 0.3363 - accu
racy: 0.8970 - val_loss: 0.1273 - val_accuracy: 0.9622
Epoch 2/20
469/469 [==============================] - 5s 11ms/step - loss: 0.1548 - accu
racy: 0.9549 - val_loss: 0.0924 - val_accuracy: 0.9719
Epoch 3/20
469/469 [==============================] - 6s 13ms/step - loss: 0.1203 - accu
racy: 0.9636 - val_loss: 0.0909 - val_accuracy: 0.9713
Epoch 4/20
469/469 [==============================] - 5s 11ms/step - loss: 0.1008 - accu
racy: 0.9700 - val_loss: 0.0762 - val_accuracy: 0.9778
Epoch 5/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0878 - accu
racy: 0.9738 - val_loss: 0.0705 - val_accuracy: 0.9798
Epoch 6/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0805 - accu
racy: 0.9756 - val_loss: 0.0693 - val_accuracy: 0.9804
Epoch 7/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0724 - accu
racy: 0.9781 - val_loss: 0.0738 - val_accuracy: 0.9811
Epoch 8/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0695 - accu
racy: 0.9798 - val_loss: 0.0674 - val_accuracy: 0.9814
Epoch 9/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0639 - accu
racy: 0.9808 - val_loss: 0.0618 - val_accuracy: 0.9817
Epoch 10/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0591 - accu
racy: 0.9824 - val_loss: 0.0684 - val_accuracy: 0.9821
Epoch 11/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0542 - accu
racy: 0.9835 - val_loss: 0.0702 - val_accuracy: 0.9825
Epoch 12/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0523 - accu
racy: 0.9842 - val_loss: 0.0706 - val_accuracy: 0.9827
Epoch 13/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0500 - accu
racy: 0.9853 - val_loss: 0.0745 - val_accuracy: 0.9817
Epoch 14/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0493 - accu
racy: 0.9855 - val_loss: 0.0641 - val_accuracy: 0.9846
Epoch 15/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0477 - accu
racy: 0.9860 - val_loss: 0.0657 - val_accuracy: 0.9838
Epoch 16/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0451 - accu
racy: 0.9866 - val_loss: 0.0692 - val_accuracy: 0.9838
Epoch 17/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0431 - accu
racy: 0.9875 - val_loss: 0.0662 - val_accuracy: 0.9834
Epoch 18/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0413 - accu
racy: 0.9876 - val_loss: 0.0715 - val_accuracy: 0.9832
Epoch 19/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0384 - accu
racy: 0.9889 - val_loss: 0.0758 - val_accuracy: 0.9835
```

```
Epoch 20/20
469/469 [==============================] - 5s 11ms/step - loss: 0.0375 - accu
racy: 0.9886 - val_loss: 0.0722 - val_accuracy: 0.9845
```

model_2

In [26]:

```
from keras import regularizers
```

In [34]:

```
model_2 = Sequential()
model_2.add(Dense(400, activation='relu', input_shape = (784, ), kernel_regula
model_2.add(Dropout(0.4))
model_2.add(Dense(300, activation='relu', input_shape = (784, ), kernel_regula
model_2.add(Dropout(0.4))
model_2.add(Dense(10, activation='softmax', kernel_regularizer = regularizers.
```

In [35]:
```
learning_rate = 0.01
model_2.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(lr=learning_rate),
              metrics=['accuracy'])
batch_size = 100
epochs = 20
history_2 = model_2.fit(
    x_train, y_train,
    batch_size=batch_size,
    epochs=epochs,
    verbose=1,
    validation_data=(x_test, y_test))
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use `learning_rate
` or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.RMSprop.
```

```
Epoch 1/20
600/600 [==============================] - 8s 12ms/step - loss: 0.4080 - accu
racy: 0.9027 - val_loss: 0.2279 - val_accuracy: 0.9554
Epoch 2/20
600/600 [==============================] - 6s 10ms/step - loss: 0.2307 - accu
racy: 0.9545 - val_loss: 0.1744 - val_accuracy: 0.9675
Epoch 3/20
600/600 [==============================] - 7s 12ms/step - loss: 0.1928 - accu
racy: 0.9630 - val_loss: 0.1613 - val_accuracy: 0.9712
Epoch 4/20
600/600 [==============================] - 6s 10ms/step - loss: 0.1740 - accu
racy: 0.9665 - val_loss: 0.1449 - val_accuracy: 0.9754
Epoch 5/20
600/600 [==============================] - 7s 12ms/step - loss: 0.1607 - accu
racy: 0.9704 - val_loss: 0.1317 - val_accuracy: 0.9774
Epoch 6/20
600/600 [==============================] - 6s 11ms/step - loss: 0.1520 - accu
racy: 0.9724 - val_loss: 0.1326 - val_accuracy: 0.9767
Epoch 7/20
600/600 [==============================] - 7s 11ms/step - loss: 0.1451 - accu
racy: 0.9742 - val_loss: 0.1263 - val_accuracy: 0.9795
Epoch 8/20
600/600 [==============================] - 7s 11ms/step - loss: 0.1416 - accu
racy: 0.9742 - val_loss: 0.1284 - val_accuracy: 0.9795
Epoch 9/20
600/600 [==============================] - 6s 11ms/step - loss: 0.1358 - accu
racy: 0.9756 - val_loss: 0.1253 - val_accuracy: 0.9799
Epoch 10/20
600/600 [==============================] - 7s 11ms/step - loss: 0.1339 - accu
racy: 0.9755 - val_loss: 0.1222 - val_accuracy: 0.9804
Epoch 11/20
600/600 [==============================] - 6s 10ms/step - loss: 0.1319 - accu
racy: 0.9762 - val_loss: 0.1196 - val_accuracy: 0.9806
Epoch 12/20
600/600 [==============================] - 7s 12ms/step - loss: 0.1300 - accu
racy: 0.9766 - val_loss: 0.1246 - val_accuracy: 0.9789
Epoch 13/20
600/600 [==============================] - 6s 10ms/step - loss: 0.1281 - accu
racy: 0.9772 - val_loss: 0.1229 - val_accuracy: 0.9802
Epoch 14/20
600/600 [==============================] - 7s 12ms/step - loss: 0.1284 - accu
racy: 0.9765 - val_loss: 0.1176 - val_accuracy: 0.9820
Epoch 15/20
600/600 [==============================] - 6s 11ms/step - loss: 0.1278 - accu
racy: 0.9774 - val_loss: 0.1231 - val_accuracy: 0.9789
Epoch 16/20
600/600 [==============================] - 7s 12ms/step - loss: 0.1263 - accu
racy: 0.9785 - val_loss: 0.1236 - val_accuracy: 0.9796
Epoch 17/20
600/600 [==============================] - 6s 10ms/step - loss: 0.1269 - accu
racy: 0.9775 - val_loss: 0.1200 - val_accuracy: 0.9808
Epoch 18/20
600/600 [==============================] - 7s 12ms/step - loss: 0.1251 - accu
racy: 0.9783 - val_loss: 0.1237 - val_accuracy: 0.9806
Epoch 19/20
600/600 [==============================] - 6s 10ms/step - loss: 0.1224 - accu
racy: 0.9790 - val_loss: 0.1248 - val_accuracy: 0.9787
```

```
Epoch 20/20
600/600 [==============================] - 7s 11ms/step - loss: 0.1236 - accu
racy: 0.9784 - val_loss: 0.1169 - val_accuracy: 0.9819
```

evaluating the accuracy and loss (training and validation) of model_1 and model_2

In [36]:
```
score = model_1.evaluate(x_test, y_test, verbose=0)
print('model_1 Test loss:', score[0])
print('model_1 Test accuracy:', score[1])
```

```
model_1 Test loss: 0.07218511402606964
model_1 Test accuracy: 0.984499990940094
```

In [37]:
```
score = model_2.evaluate(x_test, y_test, verbose=0)
print('model_2 Test loss:', score[0])
print('model_2 Test accuracy:', score[1])
```

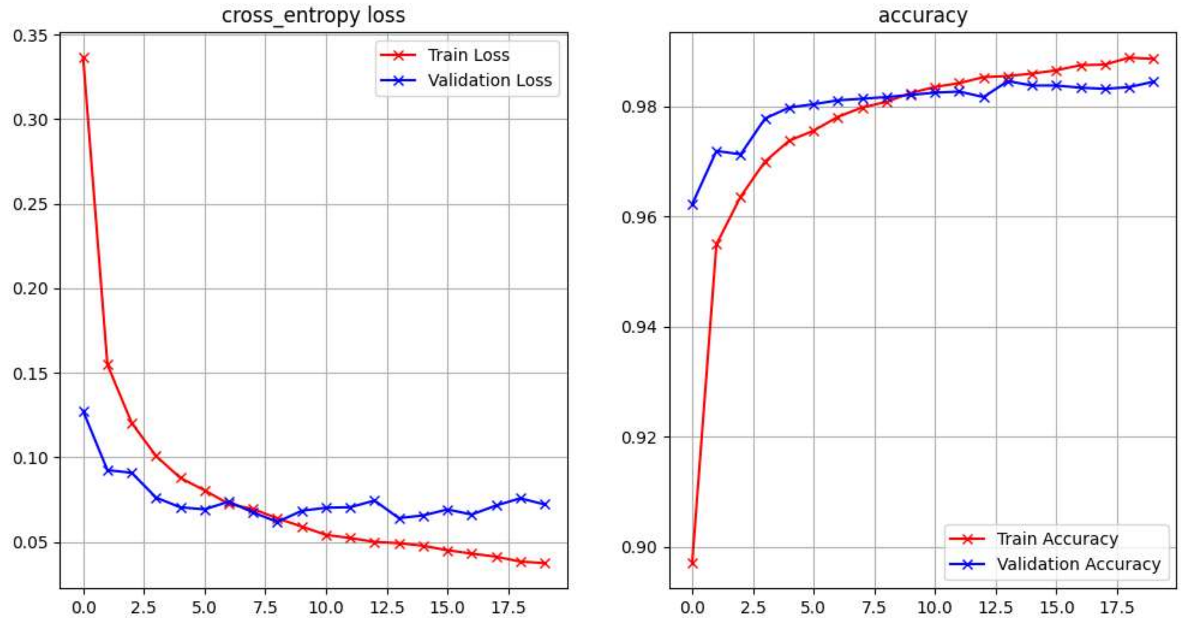
```
model_2 Test loss: 0.11694423854351044
model_2 Test accuracy: 0.9818999767303467
```

For model_1, the test loss is 0.0703 with a test accuracy of 98.36%, while for model_2, the test loss is 0.1169 with a test accuracy of 98.14%. These metrics are evaluated on a separate test dataset, unseen during training. In this comparison, model_1 shows a better performance over model_2 just a little bit, as it achieves a lower test loss and a higher test accuracy, suggesting better generalization and predictive capability on unseen data. This means that the hyperparameters for model could be done better to produce better results.

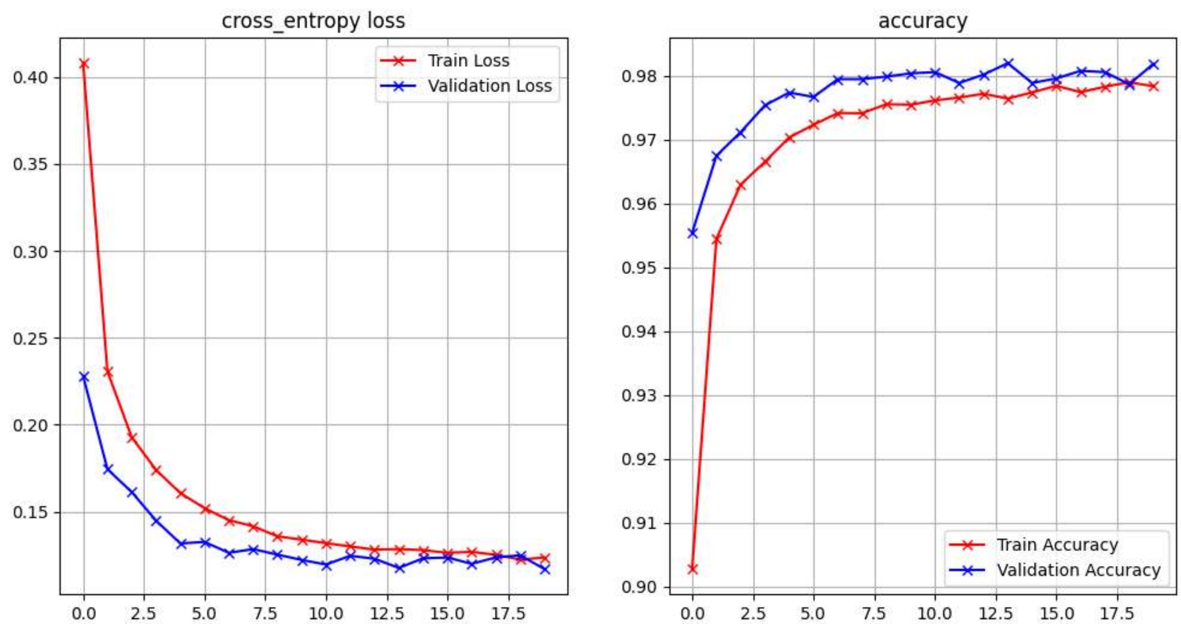
In [38]:
```
def plot_loss_accuracy(history):
    fig = plt.figure(figsize=(12, 6))
    ax = fig.add_subplot(1, 2, 1)
    ax.plot(history.history["loss"],'r-x', label="Train Loss")
    ax.plot(history.history["val_loss"],'b-x', label="Validation Loss")
    ax.legend()
    ax.set_title('cross_entropy loss')
    ax.grid(True)


    ax = fig.add_subplot(1, 2, 2)
    ax.plot(history.history["accuracy"],'r-x', label="Train Accuracy")
    ax.plot(history.history["val_accuracy"],'b-x', label="Validation Accuracy"
    ax.legend()
    ax.set_title('accuracy')
    ax.grid(True)
```

In [39]:
```
plot_loss_accuracy(history_1) # model_1
```



In [40]:
```
plot_loss_accuracy(history_2) # model_2
```



Looking at the graphs, it's clear that the first model is better than the second one. In the loss graph, the first model consistently has a lower loss, showing that it's better at learning from the data. Meanwhile, the accuracy graph also confirms this. model_1 has higher accuracy throughout the training process, meaning it makes fewer mistakes. So, even though both models do okay, the first one is definitely the better one.

conclusion

Overall, regularization is a technique used in machine learning to prevent overfitting by adding a penalty term to the model's loss function, discouraging overly complex models. Using MNIST, I was able to create a few models by applying regularization techniques.

In [40]:

```
```