Session 9 notebook

The notebook has been written during the session
please watch the video on "Course Materials" section of iLearn for the full description

May 29, 2020

0.0.1 A few announcements:

1) The final exam will be held on June 4 (Next Thursday).
2) The exam will cover all the material in 18 lectures and 9 discussion sessions.
3) You will not be asked to write a Python code. However, you should be familiar with the algorithms we used during the discussion sessions and assignments.
4) The exam will be available on June 4, 8:00 a.m. until June 5, 8 a.m. (all in California time zone).
5) Although the exam will be available for 24 hours, it is time-limited. In other words, you should complete the exam in 45 minutes once you started it. You cannot pause the exam and it will automatically be unavailable after 45 minutes. So, please make sure that you have a stable internet connection before starting the exam.
6) Exam will be accessible on iLearn: https://ilearn.ucr.edu/ —–> Assignments —–> Final Exam.
7) It is an open-book exam.

8) The final project is due on June 12, 11:59 p.m. You will submit 1) a written report (2-5 pages) which includes a brief description of the project and the data set, data processing and cleaning procedure, plots and figures, explaining the method you used to build a model and evaluating your model's performance. 2) .ipynb file to show your code work.
9) Please let me know if you have any questions. nima.chartab@email.ucr.edu

Now, let's continue our discussion on ANN:

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pylab as plt
from keras.models import Sequential
from keras.layers import Dense,Dropout,Flatten
from keras.utils import np_utils
from keras.optimizers import Adam
```

Load MNIST data

[5]:
```
from keras.datasets import mnist
(X_train,y_train),(X_test,y_test)=mnist.load_data()
```

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

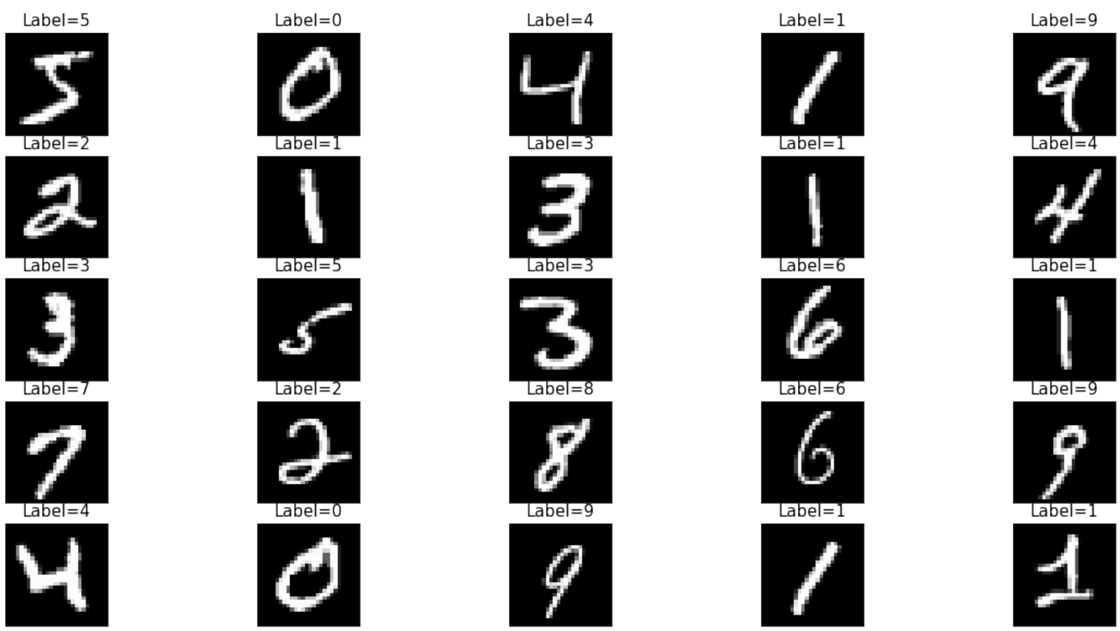
[6]:
```
(60000, 28, 28)
```

[7]:
```
X_test.shape
```

[7]:
```
(10000, 28, 28)
```

[8]:
```
plt.figure(figsize=(20,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(X_train[i],cmap='gray')
    plt.title('Label={}'.format(y_train[i]),fontsize=15)

    plt.tick_params(axis='both',which='both',bottom=False,␣
 ↪left=False,labelbottom=False, labelleft=False)
```



Build one-hot encoded vectors

[9]:
```
y_train=np_utils.to_categorical(y_train)
```

[13]:
```
y_test=np_utils.to_categorical(y_test)
```

Build an NN model:

[16]:
```
model=Sequential()

model.add(Flatten())

model.add(Dense(200,activation='relu'))

model.add(Dense(10,activation='softmax'))

model.
↪compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

[18]:
```
model.
↪fit(x=X_train,y=y_train,validation_data=(X_test,y_test),batch_size=32,epochs=10)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 5s 90us/step - loss: 2.3722 -
accuracy: 0.8804 - val_loss: 0.4270 - val_accuracy: 0.8995
Epoch 2/10
60000/60000 [==============================] - 5s 88us/step - loss: 0.3455 -
accuracy: 0.9215 - val_loss: 0.3277 - val_accuracy: 0.9203
Epoch 3/10
60000/60000 [==============================] - 5s 90us/step - loss: 0.2640 -
accuracy: 0.9345 - val_loss: 0.2571 - val_accuracy: 0.9420
Epoch 4/10
60000/60000 [==============================] - 5s 89us/step - loss: 0.2495 -
accuracy: 0.9407 - val_loss: 0.2742 - val_accuracy: 0.9353
Epoch 5/10
60000/60000 [==============================] - 6s 94us/step - loss: 0.2319 -
accuracy: 0.9443 - val_loss: 0.3509 - val_accuracy: 0.9378
Epoch 6/10
60000/60000 [==============================] - 5s 91us/step - loss: 0.2135 -
accuracy: 0.9498 - val_loss: 0.2720 - val_accuracy: 0.9411
Epoch 7/10
60000/60000 [==============================] - 5s 91us/step - loss: 0.2016 -
accuracy: 0.9523 - val_loss: 0.2782 - val_accuracy: 0.9406
Epoch 8/10
60000/60000 [==============================] - 5s 91us/step - loss: 0.2020 -
accuracy: 0.9536 - val_loss: 0.2388 - val_accuracy: 0.9487
Epoch 9/10
60000/60000 [==============================] - 6s 92us/step - loss: 0.1847 -
accuracy: 0.9568 - val_loss: 0.2701 - val_accuracy: 0.9454
Epoch 10/10
60000/60000 [==============================] - 6s 93us/step - loss: 0.1779 -
accuracy: 0.9584 - val_loss: 0.2589 - val_accuracy: 0.9520
```

[18]: <keras.callbacks.callbacks.History at 0x1404f2a30>

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

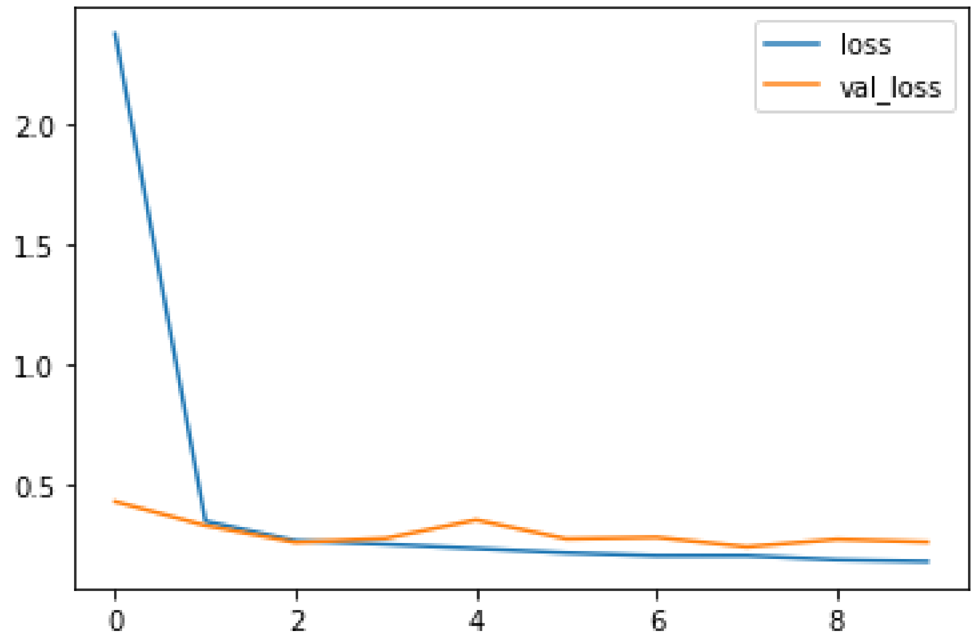
```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_1 (Dense)              (None, 200)               157000
_________________________________________________________________
dense_2 (Dense)              (None, 10)                2010
=================================================================
Total params: 159,010
Trainable params: 159,010
Non-trainable params: 0
_________________________________________________________________
```

[21]:
```
print('Number of params for the second layer:', 784*200+200)
print('Number of params for the last layer:', 10*200+10)
```

```
Number of params for the second layer: 157000
Number of params for the last layer: 2010
```

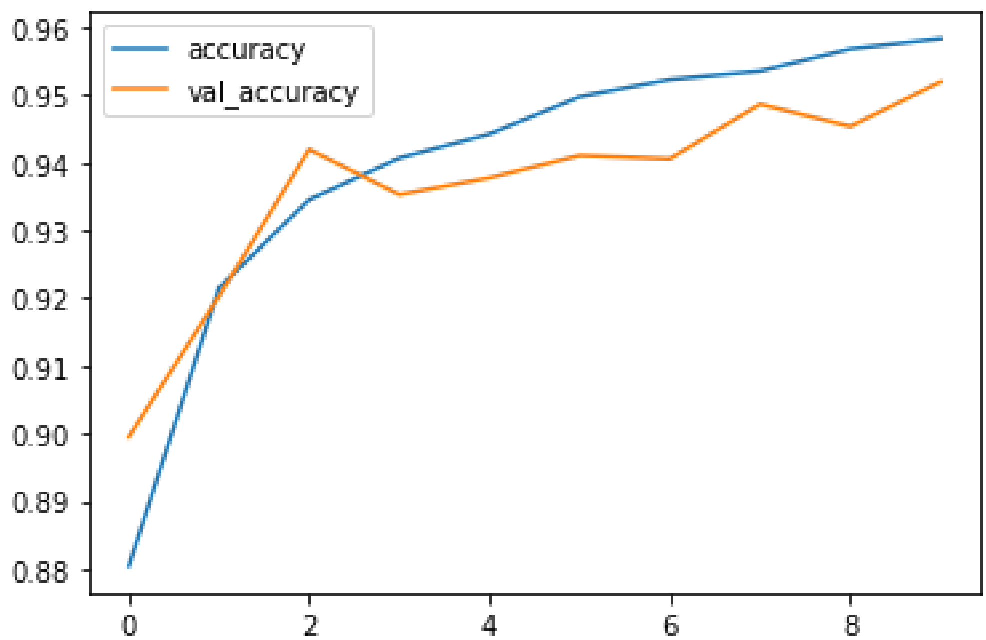
[24]:
```
history=pd.DataFrame(model.history.history)
history[['loss','val_loss']].plot()
```

[24]: <matplotlib.axes._subplots.AxesSubplot at 0x145c33b80>



[26]:
```
history[['accuracy','val_accuracy']].plot()
```

[26]: <matplotlib.axes._subplots.AxesSubplot at 0x145d222b0>



Can we do better?

[34]:
```
X_train=np_utils.normalize(X_train,axis=1)
X_test=np_utils.normalize(X_test,axis=1)
```

[38]:
```
model=Sequential()

model.add(Flatten())

model.add(Dense(200,activation='relu'))

#To avoid overfitting we want that 20% of the weights do not get updated in each␣
↪iteration
model.add(Dropout(0.2))
model.add(Dense(10,activation='softmax'))

model.
↪compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

[39]:
```
model.
↪fit(x=X_train,y=y_train,validation_data=(X_test,y_test),batch_size=32,epochs=10)
```

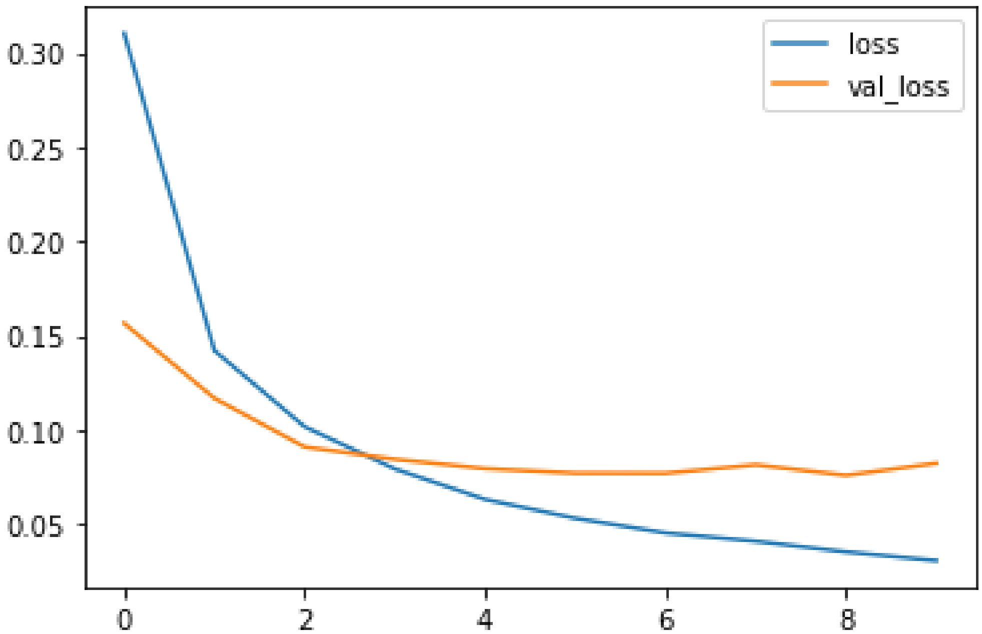
```
Train on 60000 samples, validate on 10000 samples
Epoch 1/10
60000/60000 [==============================] - 6s 107us/step - loss: 0.3104 -
accuracy: 0.9116 - val_loss: 0.1565 - val_accuracy: 0.9550
Epoch 2/10
60000/60000 [==============================] - 6s 104us/step - loss: 0.1421 -
accuracy: 0.9580 - val_loss: 0.1167 - val_accuracy: 0.9663
Epoch 3/10
60000/60000 [==============================] - 7s 121us/step - loss: 0.1016 -
accuracy: 0.9693 - val_loss: 0.0909 - val_accuracy: 0.9737
Epoch 4/10
60000/60000 [==============================] - 6s 100us/step - loss: 0.0792 -
accuracy: 0.9756 - val_loss: 0.0843 - val_accuracy: 0.9745
Epoch 5/10
60000/60000 [==============================] - 6s 102us/step - loss: 0.0630 -
accuracy: 0.9803 - val_loss: 0.0794 - val_accuracy: 0.9763
Epoch 6/10
60000/60000 [==============================] - 6s 103us/step - loss: 0.0530 -
accuracy: 0.9833 - val_loss: 0.0770 - val_accuracy: 0.9767
Epoch 7/10
60000/60000 [==============================] - 6s 102us/step - loss: 0.0452 -
accuracy: 0.9854 - val_loss: 0.0769 - val_accuracy: 0.9778
Epoch 8/10
60000/60000 [==============================] - 6s 100us/step - loss: 0.0408 -
accuracy: 0.9867 - val_loss: 0.0814 - val_accuracy: 0.9769
Epoch 9/10
60000/60000 [==============================] - 6s 100us/step - loss: 0.0351 -
```

```
accuracy: 0.9885 - val_loss: 0.0758 - val_accuracy: 0.9781
Epoch 10/10
60000/60000 [==============================] - 6s 98us/step - loss: 0.0305 -
accuracy: 0.9901 - val_loss: 0.0822 - val_accuracy: 0.9775
```

[39]: <keras.callbacks.callbacks.History at 0x1461ee6a0>

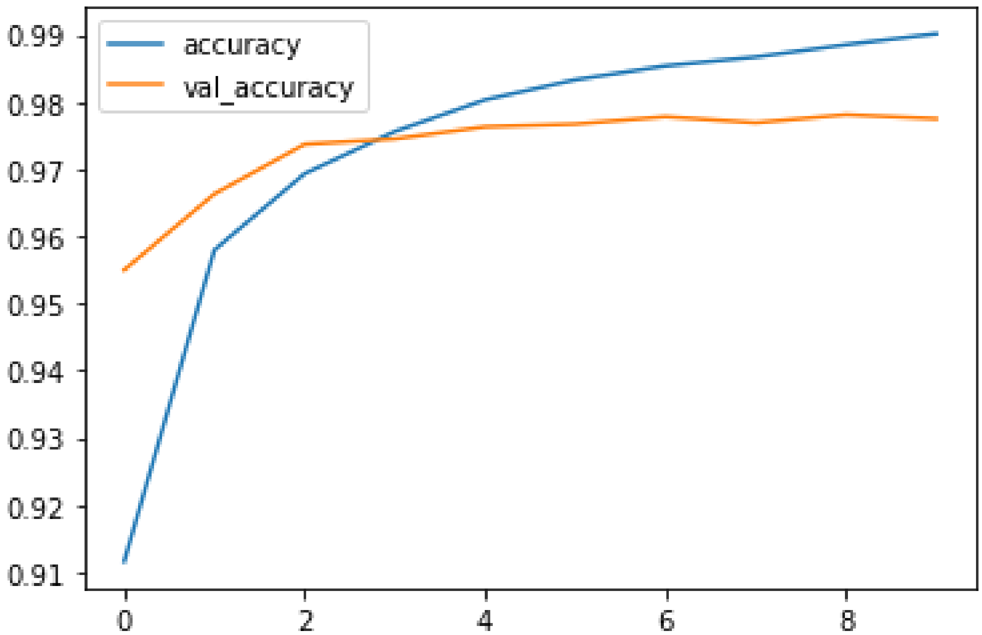
[40]:
```
history=pd.DataFrame(model.history.history)
history[['loss','val_loss']].plot()
```

[40]: <matplotlib.axes._subplots.AxesSubplot at 0x146237460>



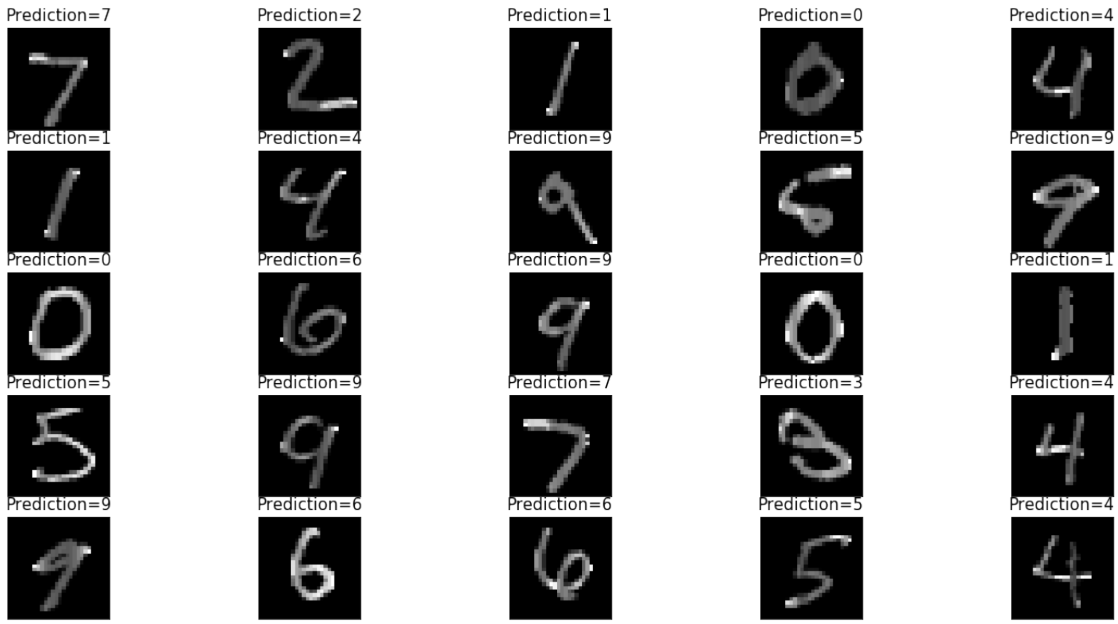
[42]:
```
history[['accuracy','val_accuracy']].plot()
```

[42]: <matplotlib.axes._subplots.AxesSubplot at 0x14648b1f0>



[52]:
```
plt.figure(figsize=(20,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.imshow(X_test[i],cmap='gray')
    plt.title('Prediction={}'.format(np.argmax(model.
 ↪predict([X_test])[i])),fontsize=15)

    plt.tick_params(axis='both',which='both',bottom=False,␣
 ↪left=False,labelbottom=False, labelleft=False)
```



0.0.2 Convolutional neural network(CNN):

[54]:
```
X_train=X_train.reshape(X_train.shape[0],X_train.shape[1],X_train.shape[2],1)
X_test=X_test.reshape(X_test.shape[0],X_test.shape[1],X_test.shape[2],1)
```

[55]:
```
X_train.shape
```

[55]:
```
(60000, 28, 28, 1)
```

[56]:
```
from keras.layers import Conv2D, MaxPooling2D
```

[60]:
```
model=Sequential()

model.add(Conv2D(64,(3,3),input_shape=(28,28,1),activation='relu'))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(200,activation='relu'))
model.add(Dense(10,activation='softmax'))

model.
 ↪compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

[61]:
```
model.
 ↪fit(x=X_train,y=y_train,validation_data=(X_test,y_test),batch_size=32,epochs=3)
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 72s 1ms/step - loss: 0.1611 -
accuracy: 0.9514 - val_loss: 0.0753 - val_accuracy: 0.9757
Epoch 2/3
60000/60000 [==============================] - 72s 1ms/step - loss: 0.0577 -
accuracy: 0.9822 - val_loss: 0.0603 - val_accuracy: 0.9797
Epoch 3/3
60000/60000 [==============================] - 78s 1ms/step - loss: 0.0364 -
accuracy: 0.9887 - val_loss: 0.0631 - val_accuracy: 0.9806
```

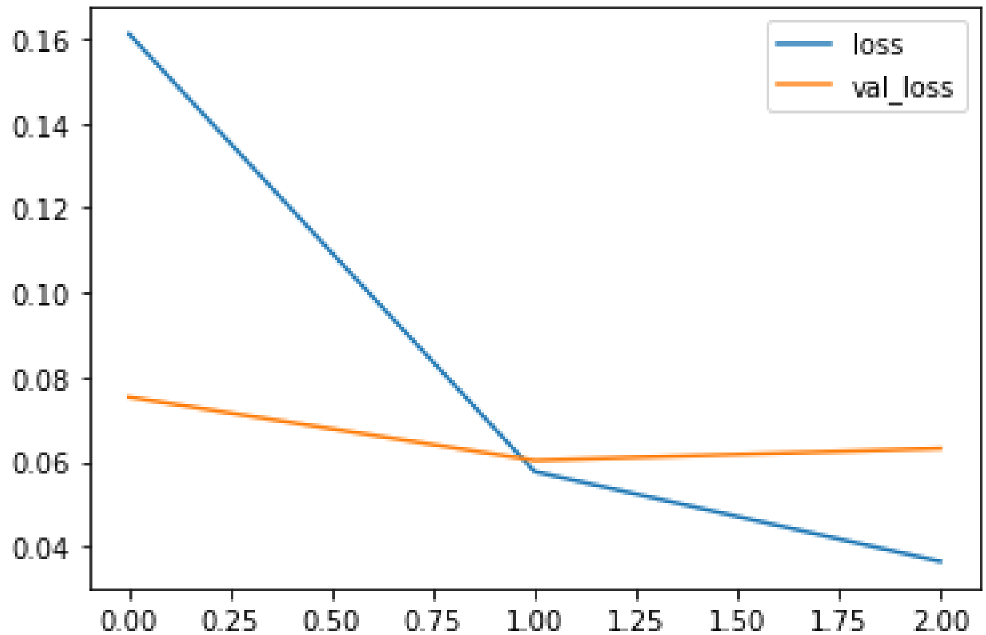
[61]:
```
<keras.callbacks.callbacks.History at 0x15e0de970>
```

[62]:
```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_4 (Conv2D)            (None, 26, 26, 64)        640
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 10816)             0
_________________________________________________________________
dense_5 (Dense)              (None, 200)               2163400
_________________________________________________________________
dense_6 (Dense)              (None, 10)                2010
=================================================================
Total params: 2,166,050
Trainable params: 2,166,050
Non-trainable params: 0
_________________________________________________________________
```

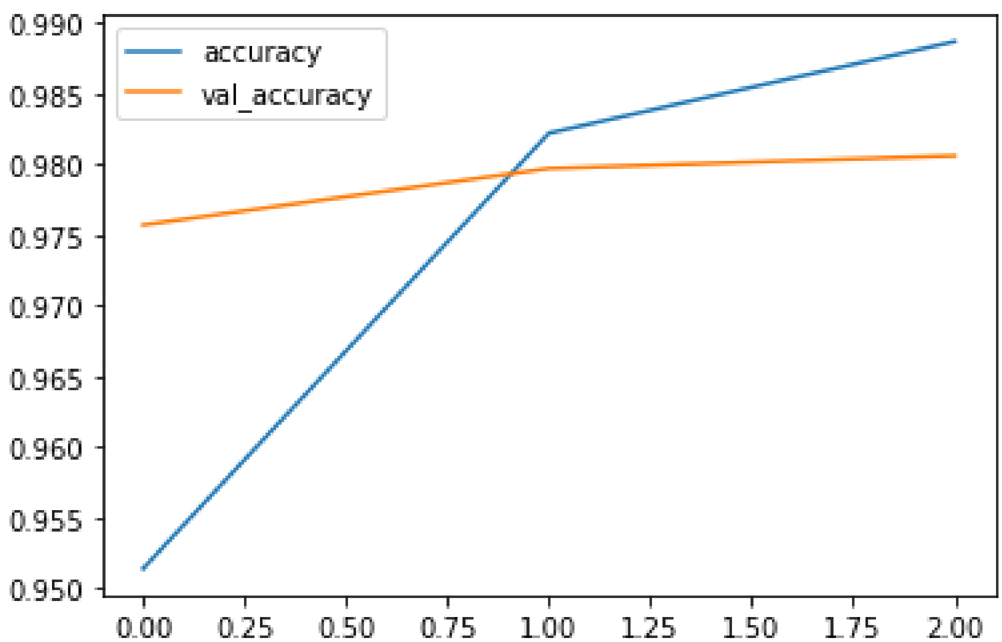
[63]:
```
history=pd.DataFrame(model.history.history)
history[['loss','val_loss']].plot()
```

[63]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x145c10250>
```



[64]: `history[['accuracy','val_accuracy']].plot()`

[64]: `<matplotlib.axes._subplots.AxesSubplot at 0x145ad8850>`



We have reached accuracy of 98% in 3 epochs.

0.0.3 TensorBoard: a powerful visualization tool

[65]:
```
from keras.callbacks import TensorBoard
```

[67]:
```
import os
%load_ext tensorboard
```

[68]:
```
os.mkdir('log_session9')
```

[71]:
```
log_dir='log_session9/'
```

[72]:
```
tensorboard_callback=TensorBoard(log_dir=log_dir,histogram_freq=1)
```

[73]:
```
model=Sequential()

model.add(Conv2D(64,(3,3),input_shape=(28,28,1),activation='relu'))

model.add(MaxPooling2D(pool_size=(2,2)))

model.add(Flatten())

model.add(Dense(200,activation='relu'))
model.add(Dense(10,activation='softmax'))

model.
 ↪compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

[74]:
```
model.
 ↪fit(x=X_train,y=y_train,validation_data=(X_test,y_test),batch_size=32,epochs=3,callbacks=[ten
```

```
Train on 60000 samples, validate on 10000 samples
Epoch 1/3
60000/60000 [==============================] - 72s 1ms/step - loss: 0.1639 -
accuracy: 0.9506 - val_loss: 0.0716 - val_accuracy: 0.9776
Epoch 2/3
60000/60000 [==============================] - 73s 1ms/step - loss: 0.0569 -
accuracy: 0.9827 - val_loss: 0.0572 - val_accuracy: 0.9825
Epoch 3/3
60000/60000 [==============================] - 76s 1ms/step - loss: 0.0353 -
accuracy: 0.9886 - val_loss: 0.0596 - val_accuracy: 0.9802
```

[74]:
```
<keras.callbacks.callbacks.History at 0x144018040>
```

[76]:
```
%tensorboard --logdir log_session9
```

```
<IPython.core.display.HTML object>
```

```
[ ]:
```