Deep+learning+batch+size+20-achindha

December 10, 2017

1 Table of Contents

2 Installation of tensorflow and keras

```
In [1]: import pip
        pip.main(['install', '-U', 'keras'])
        #pip.main(['install', '-U', 'tensorflow'])
```

3 Importing standard deep learning libraries:

```
In [5]: import numpy as np
        import tensorflow as tf
        import scipy
        import keras as kr
        from keras.utils import np_utils
        from keras.models import Sequential
        from keras.layers import Dense, Dropout, Activation, Flatten, BatchNormalization
        from matplotlib import pyplot as plt
        from keras.layers import Convolution2D, MaxPooling2D
        from keras import optimizers
        from keras.datasets import mnist
        import time
        from keras import backend as K
```

In [3]:
```
np.random.seed(123)
# This makes sure that the same set of random variables is generated in each run
#Repeatability is maintained
```

4 Stochastic gradient descent optimizer:

In [4]:
```
t_SGD = time.time()
(X_train_SGD, y_train_SGD), (X_test_SGD, y_test_SGD) = mnist.load_data()

X_train_SGD = X_train_SGD.reshape(X_train_SGD.shape[0], 28, 28,1)
X_test_SGD = X_test_SGD.reshape(X_test_SGD.shape[0], 28, 28,1)
X_train_SGD = X_train_SGD.astype('float32')
X_test_SGD = X_test_SGD.astype('float32')
X_train_SGD /= 255
X_test_SGD /= 255

Y_train_SGD = np_utils.to_categorical(y_train_SGD, 10)
Y_test_SGD = np_utils.to_categorical(y_test_SGD, 10)

network_SGD = Sequential() #defining the type of neural network

#Adding layers to the neural network:
network_SGD.add(Convolution2D(32, 5, 5, activation='relu', input_shape=(28,28,1)))
network_SGD.add(BatchNormalization())
network_SGD.add(MaxPooling2D(pool_size=(2,2)))
network_SGD.add(Convolution2D(32, 5, 5, activation='relu', input_shape=(28,28,1)))
network_SGD.add(BatchNormalization())
network_SGD.add(MaxPooling2D(pool_size=(2,2)))
network_SGD.add(BatchNormalization())

#Post processing of layers:
network_SGD.add(Dropout(0.25))
network_SGD.add(Flatten())
network_SGD.add(Dense(128, activation='relu')) #activation function used
network_SGD.add(Dropout(0.5))
network_SGD.add(Dense(10, activation='softmax')) #activation function used

network_SGD.compile(loss='categorical_crossentropy',optimizer='sgd',metrics=['accuracy']

results_SGD = network_SGD.fit(X_train_SGD, Y_train_SGD, batch_size=20, nb_epoch=20, verb
elapsed_SGD = time.time() - t_SGD
print("Neural network training time is: {0:1.4f}".format(elapsed_SGD),"seconds")
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:17: UserWarning: Update your `Conv2
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:20: UserWarning: Update your `Conv2
/opt/conda/lib/python3.6/site-packages/keras/models.py:939: UserWarning: The `nb_epoch` argument
  warnings.warn('The `nb_epoch` argument in `fit` '
```

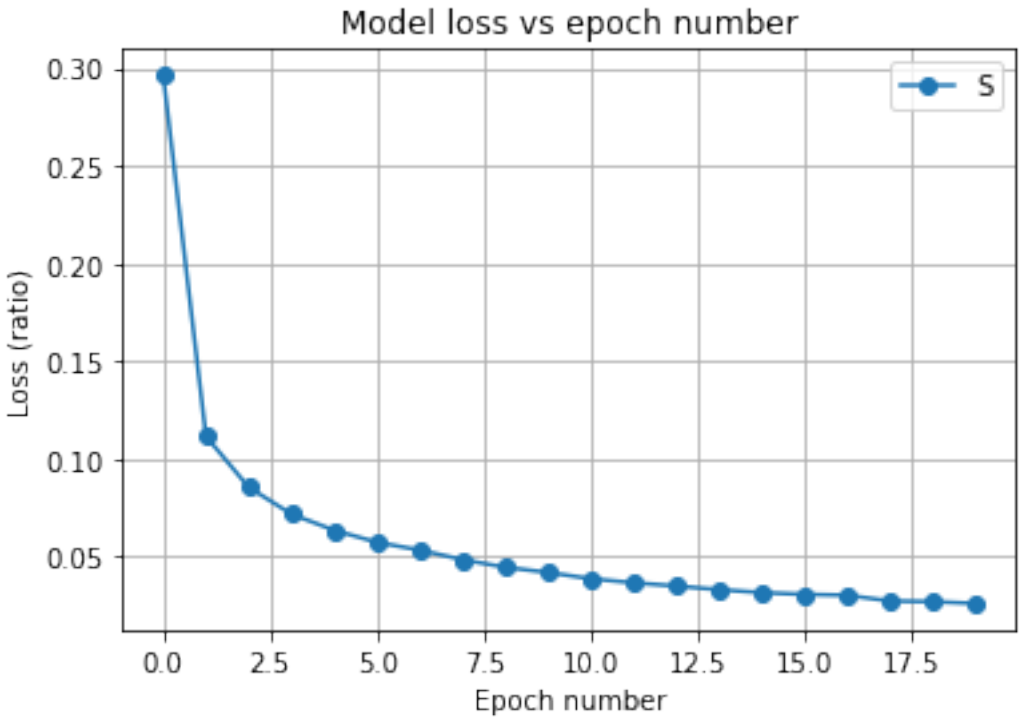
```
Epoch 1/20
60000/60000 [==============================] - 218s 4ms/step - loss: 0.2964 - acc: 0.9084
Epoch 2/20
60000/60000 [==============================] - 209s 3ms/step - loss: 0.1116 - acc: 0.9656
Epoch 3/20
60000/60000 [==============================] - 232s 4ms/step - loss: 0.0856 - acc: 0.9739
Epoch 4/20
60000/60000 [==============================] - 224s 4ms/step - loss: 0.0718 - acc: 0.9779
Epoch 5/20
60000/60000 [==============================] - 217s 4ms/step - loss: 0.0636 - acc: 0.9809
Epoch 6/20
60000/60000 [==============================] - 218s 4ms/step - loss: 0.0574 - acc: 0.9824
Epoch 7/20
60000/60000 [==============================] - 217s 4ms/step - loss: 0.0534 - acc: 0.9839
Epoch 8/20
60000/60000 [==============================] - 218s 4ms/step - loss: 0.0485 - acc: 0.9852
Epoch 9/20
60000/60000 [==============================] - 219s 4ms/step - loss: 0.0446 - acc: 0.9861
Epoch 10/20
60000/60000 [==============================] - 218s 4ms/step - loss: 0.0420 - acc: 0.9870
Epoch 11/20
60000/60000 [==============================] - 216s 4ms/step - loss: 0.0388 - acc: 0.9883
Epoch 12/20
60000/60000 [==============================] - 217s 4ms/step - loss: 0.0367 - acc: 0.9889
Epoch 13/20
60000/60000 [==============================] - 217s 4ms/step - loss: 0.0350 - acc: 0.9893
Epoch 14/20
60000/60000 [==============================] - 219s 4ms/step - loss: 0.0332 - acc: 0.9895
Epoch 15/20
60000/60000 [==============================] - 225s 4ms/step - loss: 0.0316 - acc: 0.9907
Epoch 16/20
60000/60000 [==============================] - 220s 4ms/step - loss: 0.0308 - acc: 0.9903
Epoch 17/20
60000/60000 [==============================] - 220s 4ms/step - loss: 0.0303 - acc: 0.9908
Epoch 18/20
60000/60000 [==============================] - 236s 4ms/step - loss: 0.0274 - acc: 0.9910
Epoch 19/20
60000/60000 [==============================] - 238s 4ms/step - loss: 0.0271 - acc: 0.9910
Epoch 20/20
60000/60000 [==============================] - 235s 4ms/step - loss: 0.0261 - acc: 0.9915
Neural network training time is: 4434.5606 seconds
```

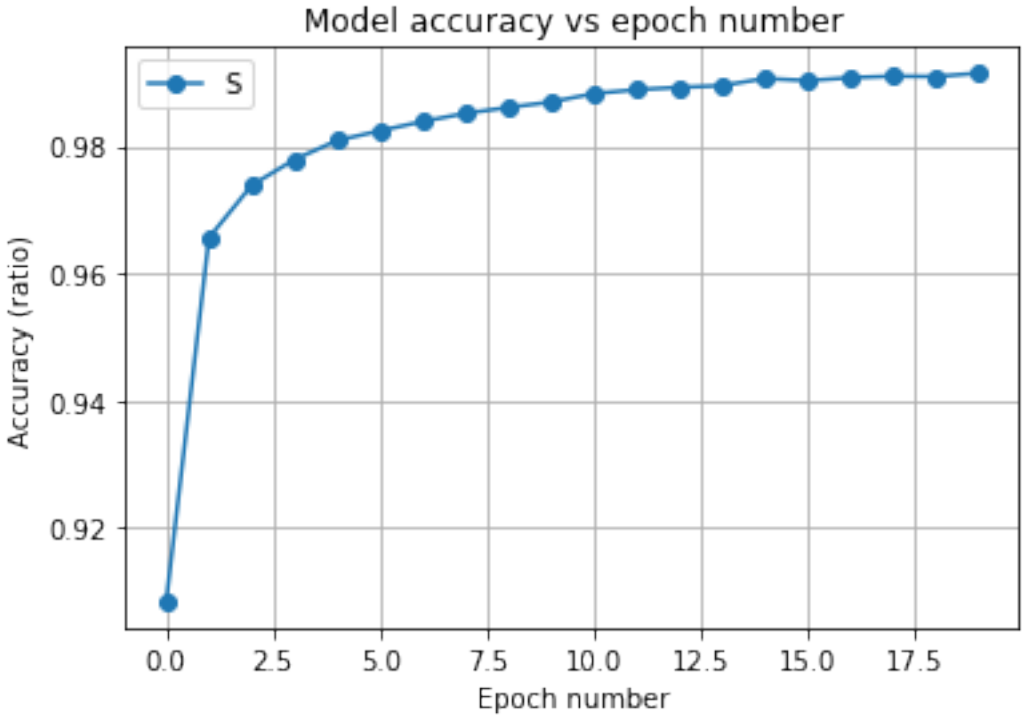
5 Testing trained data (SGD)

```
In [5]: t_test_SGD = time.time()
        score_SGD = network_SGD.evaluate(X_test_SGD, Y_test_SGD, verbose=0)
```

```
print("When evaluated on the MNIST test dataset, the loss is: {0:1.4f}".format(score_SGD
print("When evaluated on the MNIST test dataset, the acccuracy is: {0:1.4f}".format(scor
#Loss versus epochs
plt.plot(results_SGD.history['loss'],'-o')
plt.title("Model loss vs epoch number")
plt.xlabel("Epoch number")
plt.ylabel("Loss (ratio)")
plt.legend("SGD")
plt.grid()
plt.show()

#Accuracy versus epochs
plt.plot(results_SGD.history['acc'],'-o')
plt.xlabel("Epoch number")
plt.ylabel("Accuracy (ratio)")
plt.title("Model accuracy vs epoch number")
plt.legend("SGD")
plt.grid()
plt.show()

elapsed_test_SGD = time.time() - t_test_SGD
print("Evaluation and plotting runtime is: {0:1.4f}".format(elapsed_test_SGD),"seconds")
```

```
When evaluated on the MNIST test dataset, the loss is: 0.0210
When evaluated on the MNIST test dataset, the acccuracy is: 0.9931
```






```
Evaluation and plotting runtime is: 15.8835 seconds
```

6 SGD with momentum optimizer:

```
In [6]: t_SGD_mom = time.time()
        (X_train_SGD_mom, y_train_SGD_mom), (X_test_SGD_mom, y_test_SGD_mom) = mnist.load_data()

        X_train_SGD_mom = X_train_SGD_mom.reshape(X_train_SGD_mom.shape[0], 28, 28,1)
        X_test_SGD_mom = X_test_SGD_mom.reshape(X_test_SGD_mom.shape[0], 28, 28,1)
        X_train_SGD_mom = X_train_SGD_mom.astype('float32')
        X_test_SGD_mom = X_test_SGD_mom.astype('float32')
        X_train_SGD_mom /= 255
        X_test_SGD_mom /= 255

        Y_train_SGD_mom = np_utils.to_categorical(y_train_SGD_mom, 10)
        Y_test_SGD_mom = np_utils.to_categorical(y_test_SGD_mom, 10)

        network_SGD_mom = Sequential() #defining the type of neural network
```

```
        #Adding layers to the neural network:
        network_SGD_mom.add(Convolution2D(32, 5, 5, activation='relu', input_shape=(28,28,1)))
        network_SGD_mom.add(BatchNormalization())
        network_SGD_mom.add(MaxPooling2D(pool_size=(2,2)))
        network_SGD_mom.add(Convolution2D(32, 5, 5, activation='relu', input_shape=(28,28,1)))
        network_SGD_mom.add(BatchNormalization())
        network_SGD_mom.add(MaxPooling2D(pool_size=(2,2)))
        network_SGD_mom.add(BatchNormalization())

        #Post processing of layers:
        network_SGD_mom.add(Dropout(0.25))
        network_SGD_mom.add(Flatten())
        network_SGD_mom.add(Dense(128, activation='relu')) #activation function used
        network_SGD_mom.add(Dropout(0.5))
        network_SGD_mom.add(Dense(10, activation='softmax')) #activation function used

        opt = optimizers.SGD(lr=0.01, momentum=0.95, decay=1e-6, nesterov=False)

        network_SGD_mom.compile(loss='categorical_crossentropy',optimizer=opt,metrics=['accuracy

        results_SGD_mom = network_SGD_mom.fit(X_train_SGD_mom, Y_train_SGD_mom, batch_size=20, n
        elapsed_SGD_mom = time.time() - t_SGD_mom
        print("Neural network training time is: {0:1.4f}".format(elapsed_SGD_mom),"seconds")
        # see what happens when you use optimize.minimize
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:17: UserWarning: Update your `Conv2
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:20: UserWarning: Update your `Conv2
/opt/conda/lib/python3.6/site-packages/keras/models.py:939: UserWarning: The `nb_epoch` argument
  warnings.warn('The `nb_epoch` argument in `fit` '


Epoch 1/20
60000/60000 [==============================] - 234s 4ms/step - loss: 0.2344 - acc: 0.9309
Epoch 2/20
60000/60000 [==============================] - 225s 4ms/step - loss: 0.1048 - acc: 0.9703
Epoch 3/20
60000/60000 [==============================] - 218s 4ms/step - loss: 0.0888 - acc: 0.9755
Epoch 4/20
60000/60000 [==============================] - 218s 4ms/step - loss: 0.0802 - acc: 0.9777
Epoch 5/20
60000/60000 [==============================] - 219s 4ms/step - loss: 0.0739 - acc: 0.9794
Epoch 6/20
60000/60000 [==============================] - 218s 4ms/step - loss: 0.0689 - acc: 0.9806
Epoch 7/20
60000/60000 [==============================] - 219s 4ms/step - loss: 0.0599 - acc: 0.9833
Epoch 8/20
60000/60000 [==============================] - 218s 4ms/step - loss: 0.0615 - acc: 0.9834 0s - l
Epoch 9/20
```

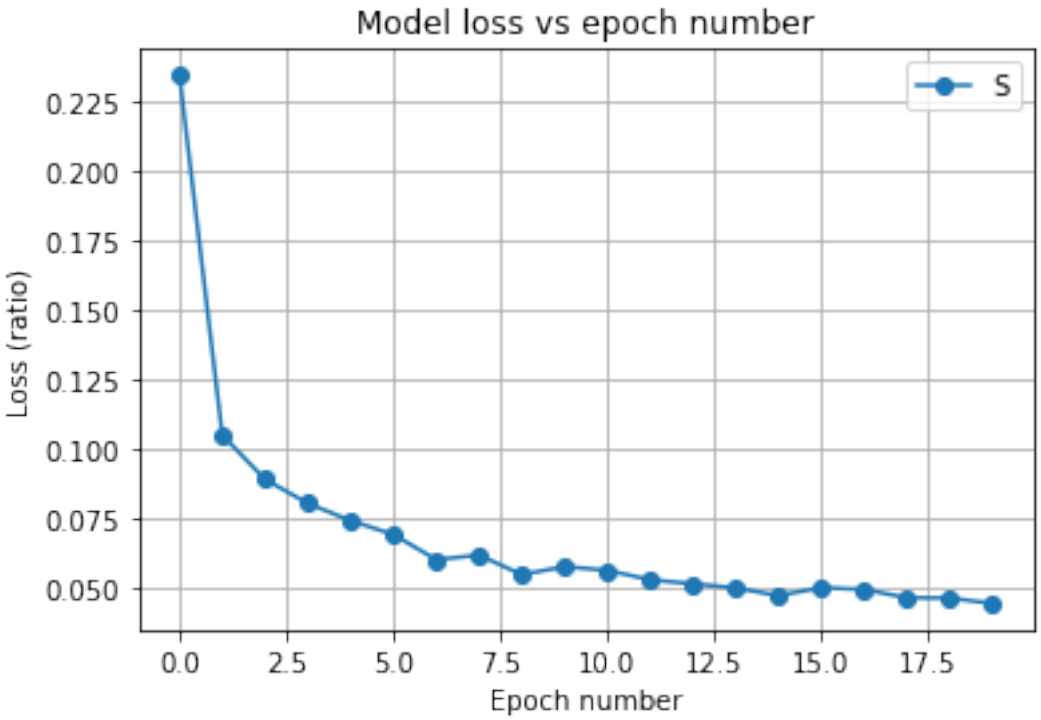
```
60000/60000 [==============================] - 218s 4ms/step - loss: 0.0545 - acc: 0.9851
Epoch 10/20
60000/60000 [==============================] - 218s 4ms/step - loss: 0.0574 - acc: 0.9844
Epoch 11/20
60000/60000 [==============================] - 239s 4ms/step - loss: 0.0560 - acc: 0.9849
Epoch 12/20
60000/60000 [==============================] - 232s 4ms/step - loss: 0.0526 - acc: 0.9852
Epoch 13/20
60000/60000 [==============================] - 221s 4ms/step - loss: 0.0512 - acc: 0.9860
Epoch 14/20
60000/60000 [==============================] - 221s 4ms/step - loss: 0.0497 - acc: 0.9864
Epoch 15/20
60000/60000 [==============================] - 220s 4ms/step - loss: 0.0469 - acc: 0.9874
Epoch 16/20
60000/60000 [==============================] - 221s 4ms/step - loss: 0.0498 - acc: 0.9869
Epoch 17/20
60000/60000 [==============================] - 221s 4ms/step - loss: 0.0491 - acc: 0.9866
Epoch 18/20
60000/60000 [==============================] - 221s 4ms/step - loss: 0.0462 - acc: 0.9883
Epoch 19/20
60000/60000 [==============================] - 219s 4ms/step - loss: 0.0461 - acc: 0.9877
Epoch 20/20
60000/60000 [==============================] - 219s 4ms/step - loss: 0.0441 - acc: 0.9878
Neural network training time is: 4441.6443 seconds
```

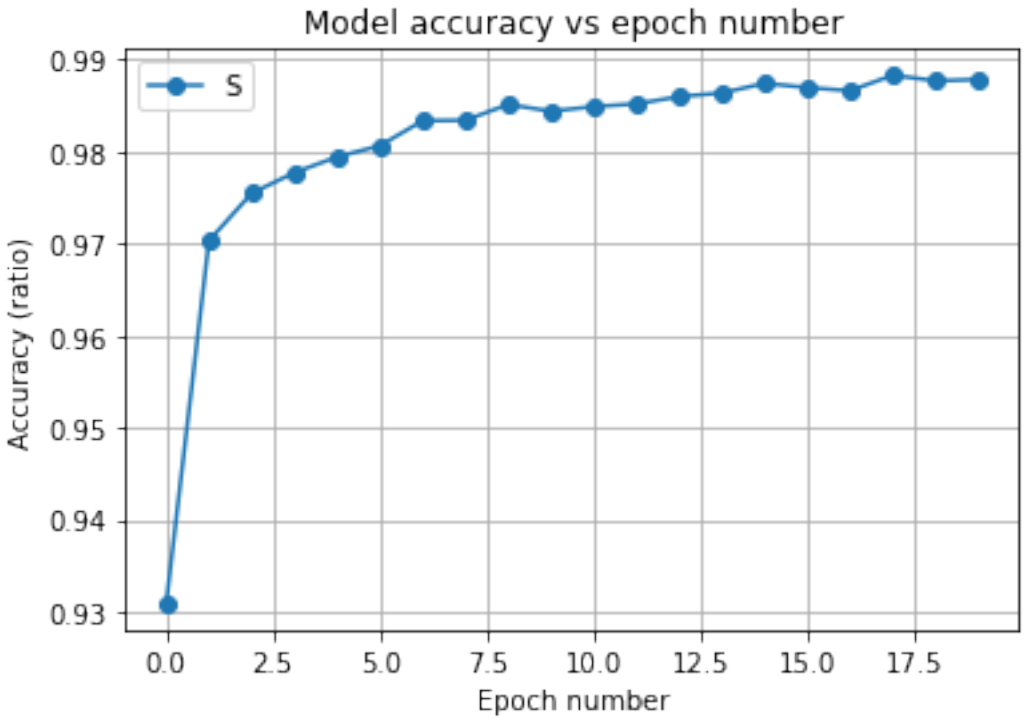
7 Testing trained data (SGD w/ momentum)

```
In [7]: t_test_SGD_mom = time.time()
        score_SGD_mom = network_SGD_mom.evaluate(X_test_SGD_mom, Y_test_SGD_mom, verbose=0)
        print("When evaluated on the MNIST test dataset, the loss is: {0:1.4f}".format(score_SGD
        print("When evaluated on the MNIST test dataset, the acccuracy is: {0:1.4f}".format(scor
        #Loss versus epochs
        plt.plot(results_SGD_mom.history['loss'],'-o')
        plt.title("Model loss vs epoch number")
        plt.xlabel("Epoch number")
        plt.ylabel("Loss (ratio)")
        plt.legend("SGD with Momentum")
        plt.grid()
        plt.show()

        #Accuracy versus epochs
        plt.plot(results_SGD_mom.history['acc'],'-o')
        plt.xlabel("Epoch number")
        plt.ylabel("Accuracy (ratio)")
        plt.title("Model accuracy vs epoch number")
        plt.legend("SGD with Momentum")
```

```
        plt.grid()
        plt.show()

        elapsed_test_SGD_mom = time.time() - t_test_SGD_mom
        print("Evaluation and plotting runtime is: {0:1.4f}".format(elapsed_test_SGD_mom),"secon
```

```
When evaluated on the MNIST test dataset, the loss is: 0.0391
When evaluated on the MNIST test dataset, the acccuracy is: 0.9919
```






```
Evaluation and plotting runtime is: 15.6856 seconds
```

8 Adam optimizer (Neural network training):

```
In [8]: t_adam = time.time()
        (X_train_adam, y_train_adam), (X_test_adam, y_test_adam) = mnist.load_data()

        X_train_adam = X_train_adam.reshape(X_train_adam.shape[0], 28, 28,1)
        X_test_adam = X_test_adam.reshape(X_test_adam.shape[0], 28, 28,1)
        X_train_adam = X_train_adam.astype('float32')
        X_test_adam = X_test_adam.astype('float32')
        X_train_adam /= 255
        X_test_adam /= 255

        Y_train_adam = np_utils.to_categorical(y_train_adam, 10)
        Y_test_adam = np_utils.to_categorical(y_test_adam, 10)

        network = Sequential() #defining the type of neural network

        #Adding layers to the neural network:
        network.add(Convolution2D(32, 5, 5, activation='relu', input_shape=(28,28,1)))
        network.add(BatchNormalization())
```

```
        network.add(MaxPooling2D(pool_size=(2,2)))
        network.add(Convolution2D(32, 5, 5, activation='relu', input_shape=(28,28,1)))
        network.add(BatchNormalization())
        network.add(MaxPooling2D(pool_size=(2,2)))
        network.add(BatchNormalization())

        #Post processing of layers:
        network.add(Dropout(0.25))
        network.add(Flatten())
        network.add(Dense(128, activation='relu')) #activation function used
        network.add(Dropout(0.5))
        network.add(Dense(10, activation='softmax')) #activation function used

        network.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])

        results_adam = network.fit(X_train_adam, Y_train_adam, batch_size=20, nb_epoch=20, verbo
        elapsed_adam = time.time() - t_adam
        print("Neural network training time is: {0:1.4f}".format(elapsed_adam),"seconds")
```

```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:17: UserWarning: Update your `Conv2
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:20: UserWarning: Update your `Conv2
/opt/conda/lib/python3.6/site-packages/keras/models.py:939: UserWarning: The `nb_epoch` argument
  warnings.warn('The `nb_epoch` argument in `fit` '


Epoch 1/20
60000/60000 [==============================] - 225s 4ms/step - loss: 0.1910 - acc: 0.9428
Epoch 2/20
60000/60000 [==============================] - 221s 4ms/step - loss: 0.0873 - acc: 0.9752
Epoch 3/20
60000/60000 [==============================] - 219s 4ms/step - loss: 0.0708 - acc: 0.9793
Epoch 4/20
60000/60000 [==============================] - 220s 4ms/step - loss: 0.0625 - acc: 0.9816
Epoch 5/20
60000/60000 [==============================] - 227s 4ms/step - loss: 0.0524 - acc: 0.9847
Epoch 6/20
60000/60000 [==============================] - 236s 4ms/step - loss: 0.0517 - acc: 0.9851
Epoch 7/20
60000/60000 [==============================] - 235s 4ms/step - loss: 0.0431 - acc: 0.9874
Epoch 8/20
60000/60000 [==============================] - 240s 4ms/step - loss: 0.0421 - acc: 0.9874
Epoch 9/20
60000/60000 [==============================] - 239s 4ms/step - loss: 0.0370 - acc: 0.9889
Epoch 10/20
60000/60000 [==============================] - 243s 4ms/step - loss: 0.0370 - acc: 0.9892
Epoch 11/20
60000/60000 [==============================] - 222s 4ms/step - loss: 0.0351 - acc: 0.9895
Epoch 12/20
```

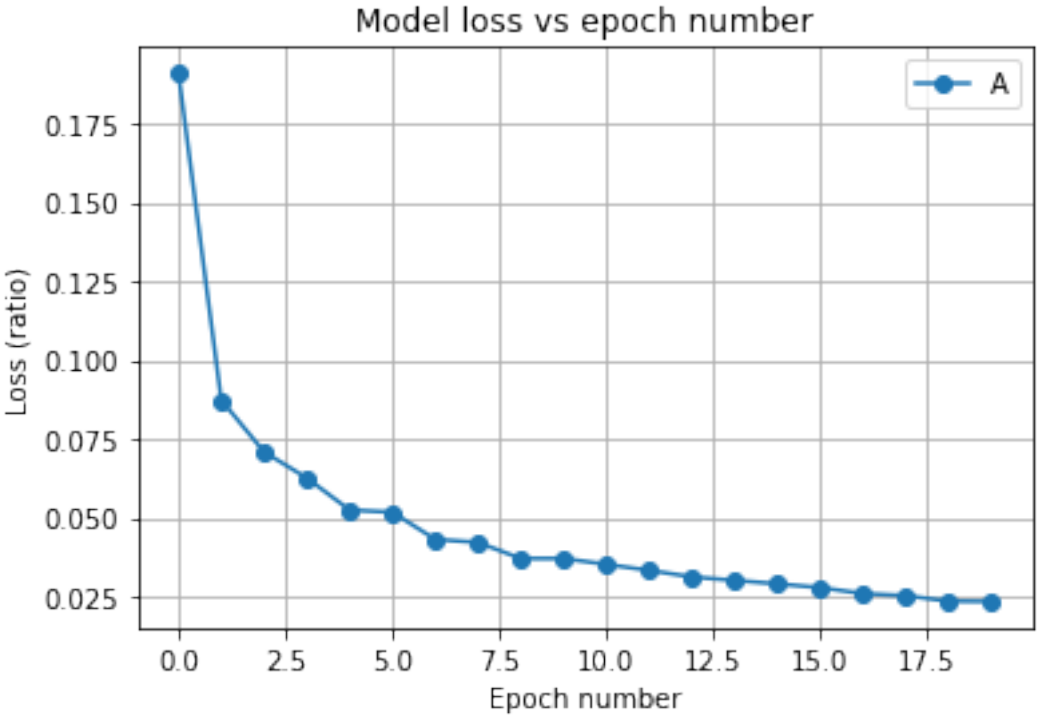
```
60000/60000 [==============================] - 222s 4ms/step - loss: 0.0332 - acc: 0.9903
Epoch 13/20
60000/60000 [==============================] - 223s 4ms/step - loss: 0.0312 - acc: 0.9910
Epoch 14/20
60000/60000 [==============================] - 221s 4ms/step - loss: 0.0301 - acc: 0.9908
Epoch 15/20
60000/60000 [==============================] - 221s 4ms/step - loss: 0.0290 - acc: 0.9914
Epoch 16/20
60000/60000 [==============================] - 223s 4ms/step - loss: 0.0278 - acc: 0.9917
Epoch 17/20
60000/60000 [==============================] - 223s 4ms/step - loss: 0.0259 - acc: 0.9922
Epoch 18/20
60000/60000 [==============================] - 216s 4ms/step - loss: 0.0252 - acc: 0.9923
Epoch 19/20
60000/60000 [==============================] - 228s 4ms/step - loss: 0.0235 - acc: 0.9925
Epoch 20/20
60000/60000 [==============================] - 240s 4ms/step - loss: 0.0234 - acc: 0.9929
Neural network training time is: 4554.9762 seconds
```

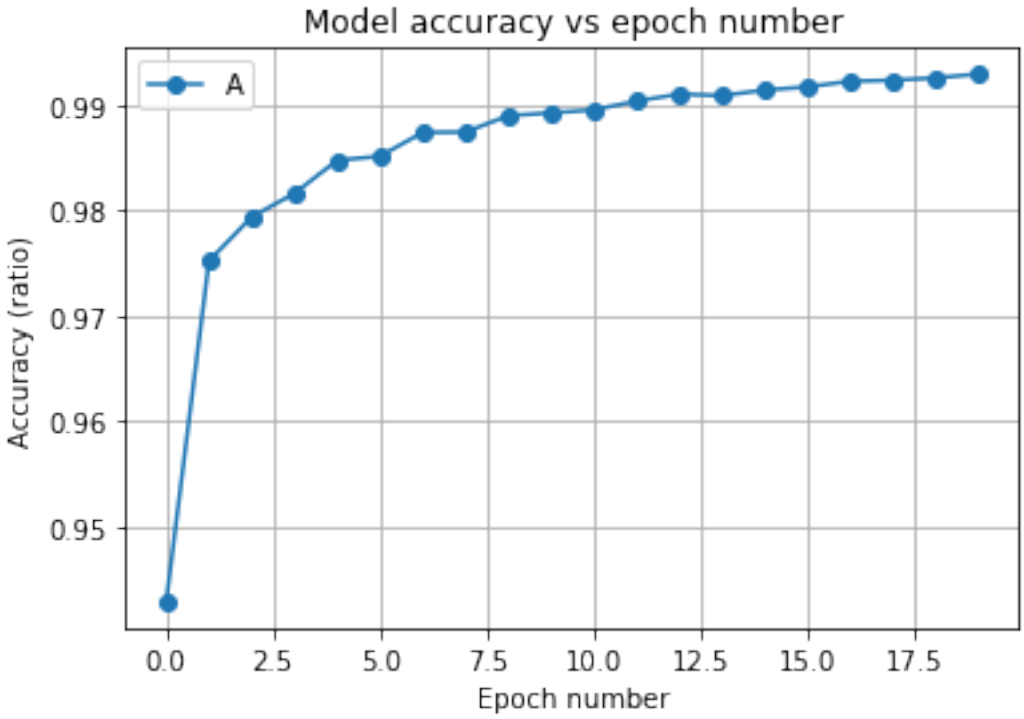
9 Testing trained data (adam)

```
In [9]: t_test_adam = time.time()
        score_adam = network.evaluate(X_test_adam, Y_test_adam, verbose=0)
        print("When evaluated on the MNIST test dataset, the loss is: {0:1.4f}".format(score_ada
        print("When evaluated on the MNIST test dataset, the acccuracy is: {0:1.4f}".format(scor
        #Loss versus epochs
        plt.plot(results_adam.history['loss'],'-o')
        plt.title("Model loss vs epoch number")
        plt.xlabel("Epoch number")
        plt.ylabel("Loss (ratio)")
        plt.legend("Adam")
        plt.grid()
        plt.show()

        #Accuracy versus epochs
        plt.plot(results_adam.history['acc'],'-o')
        plt.xlabel("Epoch number")
        plt.ylabel("Accuracy (ratio)")
        plt.title("Model accuracy vs epoch number")
        plt.legend("Adam")
        plt.grid()
        plt.show()

        elapsed_test_adam = time.time() - t_test_adam
        print("Evaluation and plotting runtime is: {0:1.4f}".format(elapsed_test_adam),"seconds"
```

```
When evaluated on the MNIST test dataset, the loss is: 0.0281
```

```
When evaluated on the MNIST test dataset, the acccuracy is: 0.9936
```





```
Evaluation and plotting runtime is: 15.9780 seconds
```

10 Adadelta optimizer

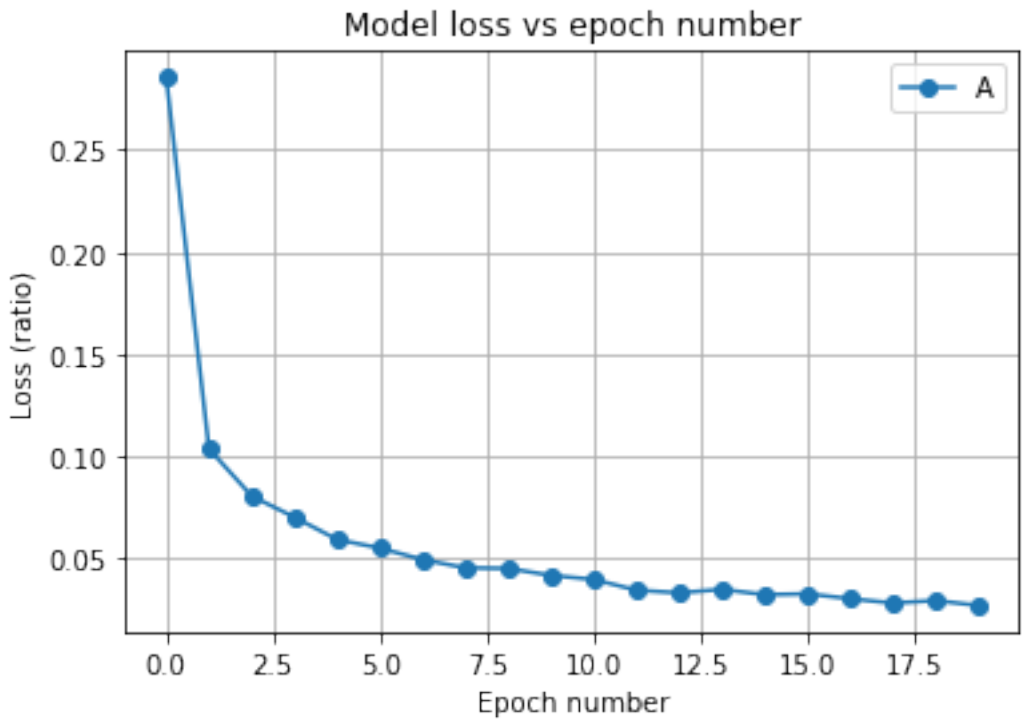
```
In [95]: t_adadelta = time.time()
         (X_train_adadelta, y_train_adadelta), (X_test_adadelta, y_test_adadelta) = mnist.load_d

         X_train_adadelta = X_train_adadelta.reshape(X_train_adadelta.shape[0], 28, 28,1)
         X_test_adadelta = X_test_adadelta.reshape(X_test_adadelta.shape[0], 28, 28,1)
         X_train_adadelta = X_train_adadelta.astype('float32')
         X_test_adadelta = X_test_adadelta.astype('float32')
         X_train_adadelta /= 255
         X_test_adadelta /= 255

         Y_train_adadelta = np_utils.to_categorical(y_train_adadelta, 10)
         Y_test_adadelta = np_utils.to_categorical(y_test_adadelta, 10)

         network_adadelta = Sequential() #defining the type of neural network

         #Adding layers to the neural network:
         network_adadelta.add(Convolution2D(32, 5, 5, activation='relu', input_shape=(28,28,1)))
         network_adadelta.add(BatchNormalization())
         network_adadelta.add(MaxPooling2D(pool_size=(2,2)))
         network_adadelta.add(Convolution2D(32, 5, 5, activation='relu', input_shape=(28,28,1)))
         network_adadelta.add(BatchNormalization())
         network_adadelta.add(MaxPooling2D(pool_size=(2,2)))
         network_adadelta.add(BatchNormalization())

         #Post processing of layers:
         network_adadelta.add(Dropout(0.25))
         network_adadelta.add(Flatten())
         network_adadelta.add(Dense(128, activation='relu')) #activation function used
         network_adadelta.add(Dropout(0.5))
         network_adadelta.add(Dense(10, activation='softmax')) #activation function used


         network_adadelta.compile(loss='categorical_crossentropy',optimizer='adadelta',metrics=[

         results_adadelta = network_adadelta.fit(X_train_adadelta, Y_train_adadelta, batch_size=
         elapsed_adadelta = time.time() - t_adadelta
         print("Neural network training time is: {0:1.4f}".format(elapsed_adadelta),"seconds")
         # see what happens when you use optimize.minimize

/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:17: UserWarning: Update your `Conv2
```

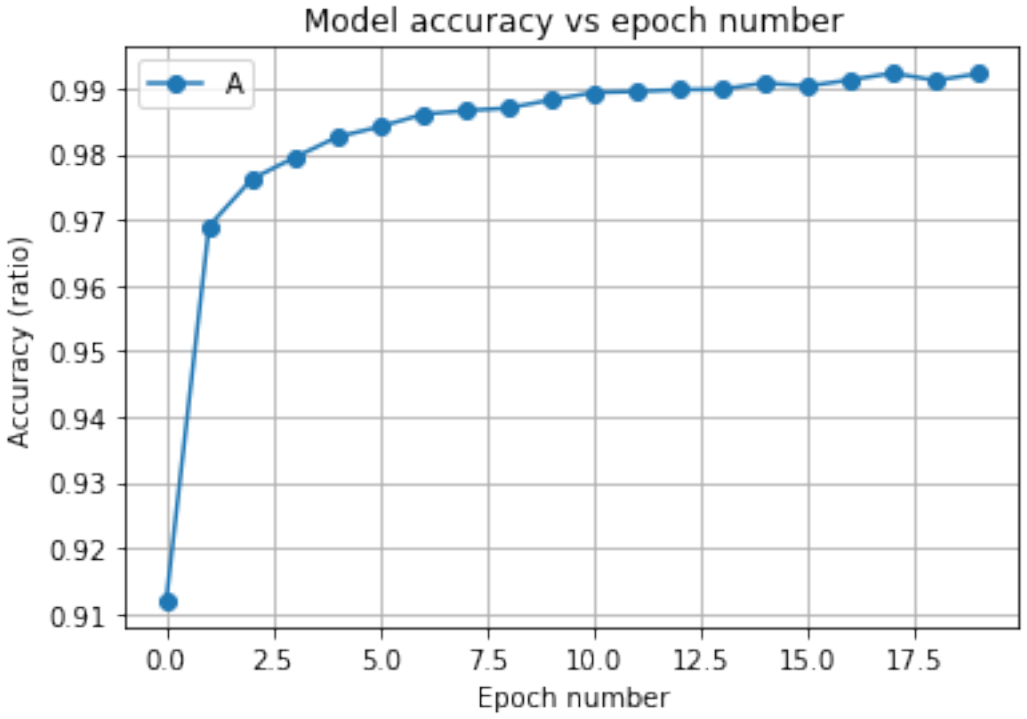
```
/opt/conda/lib/python3.6/site-packages/ipykernel_launcher.py:20: UserWarning: Update your `Conv2
/opt/conda/lib/python3.6/site-packages/keras/models.py:939: UserWarning: The `nb_epoch` argument
  warnings.warn('The `nb_epoch` argument in `fit` '


Epoch 1/20
60000/60000 [==============================] - 129s 2ms/step - loss: 0.2855 - acc: 0.9119
Epoch 2/20
60000/60000 [==============================] - 139s 2ms/step - loss: 0.1038 - acc: 0.9690
Epoch 3/20
60000/60000 [==============================] - 205s 3ms/step - loss: 0.0807 - acc: 0.9761
Epoch 4/20
60000/60000 [==============================] - 234s 4ms/step - loss: 0.0704 - acc: 0.9794
Epoch 5/20
60000/60000 [==============================] - 129s 2ms/step - loss: 0.0595 - acc: 0.9826
Epoch 6/20
60000/60000 [==============================] - 129s 2ms/step - loss: 0.0553 - acc: 0.9841
Epoch 7/20
60000/60000 [==============================] - 154s 3ms/step - loss: 0.0496 - acc: 0.9860
Epoch 8/20
60000/60000 [==============================] - 143s 2ms/step - loss: 0.0456 - acc: 0.9866
Epoch 9/20
60000/60000 [==============================] - 131s 2ms/step - loss: 0.0453 - acc: 0.9870
Epoch 10/20
60000/60000 [==============================] - 113s 2ms/step - loss: 0.0419 - acc: 0.9882
Epoch 11/20
60000/60000 [==============================] - 122s 2ms/step - loss: 0.0399 - acc: 0.9893
Epoch 12/20
60000/60000 [==============================] - 127s 2ms/step - loss: 0.0347 - acc: 0.9895
Epoch 13/20
60000/60000 [==============================] - 129s 2ms/step - loss: 0.0334 - acc: 0.9898
Epoch 14/20
60000/60000 [==============================] - 137s 2ms/step - loss: 0.0351 - acc: 0.9898
Epoch 15/20
60000/60000 [==============================] - 136s 2ms/step - loss: 0.0325 - acc: 0.9908
Epoch 16/20
60000/60000 [==============================] - 132s 2ms/step - loss: 0.0329 - acc: 0.9903
Epoch 17/20
60000/60000 [==============================] - 130s 2ms/step - loss: 0.0305 - acc: 0.9912
Epoch 18/20
60000/60000 [==============================] - 121s 2ms/step - loss: 0.0284 - acc: 0.9923
Epoch 19/20
60000/60000 [==============================] - 136s 2ms/step - loss: 0.0296 - acc: 0.9912
Epoch 20/20
60000/60000 [==============================] - 140s 2ms/step - loss: 0.0272 - acc: 0.9922
Neural network training time is: 2815.9537 seconds
```

11 Testing trained data (Adadelta)

In [97]:
```
t_test_adadelta = time.time()
score_adadelta = network_adadelta.evaluate(X_test_adadelta, Y_test_adadelta, verbose=0)
print("When evaluated on the MNIST test dataset, the loss is: {0:1.4f}".format(score_ad
print("When evaluated on the MNIST test dataset, the acccuracy is: {0:1.4f}".format(sco
#Loss versus epochs
plt.plot(results_adadelta.history['loss'],'-o')
plt.title("Model loss vs epoch number")
plt.xlabel("Epoch number")
plt.ylabel("Loss (ratio)")
plt.legend("Adadelta")
plt.grid()
plt.show()

#Accuracy versus epochs
plt.plot(results_adadelta.history['acc'],'-o')
plt.xlabel("Epoch number")
plt.ylabel("Accuracy (ratio)")
plt.title("Model accuracy vs epoch number")
plt.legend("Adadelta")
plt.grid()
plt.show()

elapsed_test_adadelta = time.time() - t_test_adadelta
print("Evaluation and plotting runtime is: {0:1.4f}".format(elapsed_test_adadelta),"sec
```

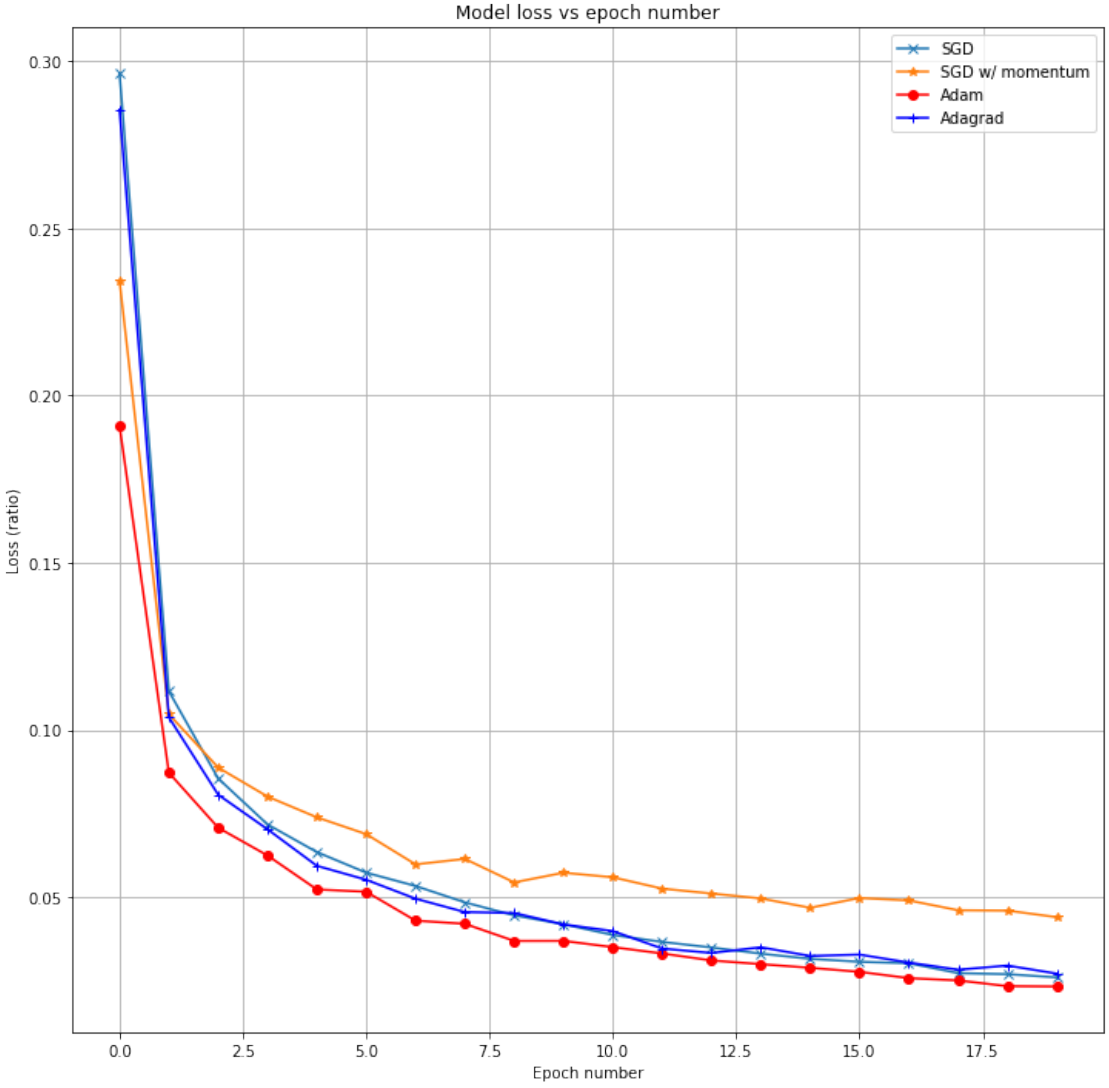
```
When evaluated on the MNIST test dataset, the loss is: 0.0268
When evaluated on the MNIST test dataset, the acccuracy is: 0.9933
```


Model loss vs epoch number
A
Loss (ratio)
0.25
0.20
0.15
0.10
0.05
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch number

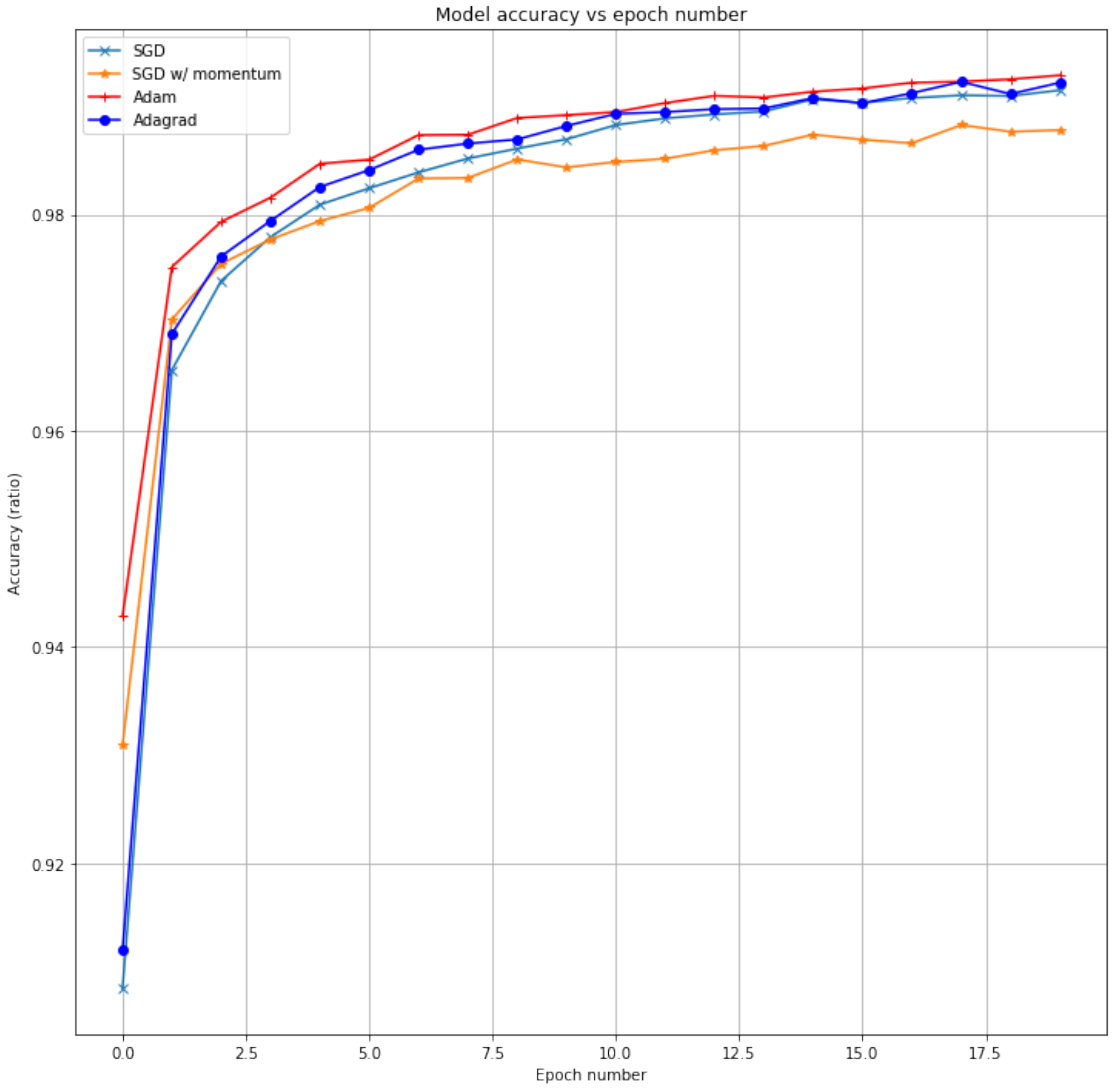

Model accuracy vs epoch number
A
Accuracy (ratio)
0.99
0.98
0.97
0.96
0.95
0.94
0.93
0.92
0.91
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Epoch number

```
Evaluation and plotting runtime is: 8.2450 seconds
```

```
In [98]: height =12
         width = 12
         plt.figure(1,figsize = (height,width))
         plt.plot(results_SGD.history['loss'],'-x',results_SGD_mom.history['loss'],'-*',results_
         plt.title("Model loss vs epoch number")
         plt.xlabel("Epoch number")
         plt.ylabel("Loss (ratio)")
         plt.legend(['SGD','SGD w/ momentum','Adam','Adagrad'])
         plt.savefig('loss 20.png')
         plt.grid()
         plt.show()
```



In [99]:
```
height =12
width = 12
plt.figure(2,figsize = (height,width))
plt.plot(results_SGD.history['acc'],'-x',results_SGD_mom.history['acc'],'-*',results_ad
plt.xlabel("Epoch number")
plt.ylabel("Accuracy (ratio)")
plt.title("Model accuracy vs epoch number")
plt.legend(['SGD','SGD w/ momentum','Adam','Adagrad'])
plt.savefig('acc 20.png')
plt.grid()
plt.show()
```



In [86]:
```
print('SGD:\n',results_SGD.history['acc'],'\n')
print('SGDmom:\n',results_SGD_mom.history['acc'],'\n')
```

```
         print('Adam:\n',results_adam.history['acc'],'\n')
         print('Adadelta:\n',results_adadelta.history['acc'],'\n')
```

```
SGD:
 [0.90841666079436745, 0.96563332678874336, 0.97388332794109977, 0.97793332890669504, 0.98093332
```

```
SGDmom:
 [0.93088332715133826, 0.97031666102011993, 0.97546666145324712, 0.97771666183074313, 0.97941666
```

```
Adam:
 [0.94284999372685951, 0.97516666169961297, 0.97934999553362534, 0.9815833292603493, 0.98471666
```

```
Adadelta:
 [0.94043332700928051, 0.97301666120688124, 0.97669999514023464, 0.97939999552567802, 0.98103332
```

```
In [87]: print('SGD:\n',results_SGD.history['loss'],'\n')
         print('SGDmom:\n',results_SGD_mom.history['loss'],'\n')
         print('Adam:\n',results_adam.history['loss'],'\n')
         print('Adadelta:\n',results_adadelta.history['loss'],'\n')
```

```
SGD:
 [0.29635741568496449, 0.1116401872189017, 0.085560050723870518, 0.071822709713218497, 0.0635559
```

```
SGDmom:
 [0.23435007755252202, 0.10477428774478539, 0.088812210901063282, 0.080152728033641318, 0.073945
```

```
Adam:
 [0.19096954527073345, 0.087286187406966448, 0.070833746635545439, 0.062535459251339492, 0.05238
```

```
Adadelta:
 [0.20400659434741827, 0.097181705248592459, 0.084476095580785115, 0.072491785509067999, 0.06717
```