NN

August 10, 2020

1 Neural Networks

1.0.1 Comparing performance of "vanilla" multilayer perceptron (MLP) with "vanilla" MLP + dropout and "vanilla" MLP + batch normalization on the Fashion MNIST dataset

[1]:
```
import sklearn
import keras
import pandas as pd
import numpy as np
import tensorflow as tf
import seaborn as sns
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

[2]:
```
from keras.datasets import fashion_mnist

(X_train, y_train), (X_test, y_test) = fashion_mnist.load_data()
```

```
Downloading data from http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/train-labels-idx1-ubyte.gz
32768/29515 [==============================] - 0s 9us/step
Downloading data from http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 4s 0us/step
Downloading data from http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-labels-idx1-ubyte.gz
8192/5148 [===============================================] - 0s 0us/step
Downloading data from http://fashion-mnist.s3-website.eu-
central-1.amazonaws.com/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 2s 1us/step
```

[ ]:
```
X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
```

[ ]:
```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
```

[ ]:
```
# Divide by 255 since range is 0 to 255
X_train /= 255
X_test /= 255
```

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

[6]:
```
(60000, 784)
```

[7]:
```
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

```
60000 train samples
10000 test samples
```

[ ]:
```
num_classes = 10
# Convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

1.1 "Vanilla" model

[ ]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation

basic_model = Sequential(
    [
        # 1st hidden layer
        Dense(196, input_shape = (784, )),
        Activation('relu'),

        # 2nd hidden layer
        Dense(49),
        Activation('relu'),

        # Output layer
        Dense(10),
        Activation('softmax')
    ]
)
```

[ ]:
```
basic_model.compile("adam", "categorical_crossentropy", metrics=['accuracy'])
```

[16]:
```
# Validation split = 1/6 gives us 10000 samples since we 60000 training samples
basic_model_history = basic_model.fit(X_train, y_train, batch_size = 8, epochs␣
 ↪= 25, verbose = 1, validation_split = 1/6)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/25
50000/50000 [==============================] - 19s 376us/step - loss: 0.3341 -
accuracy: 0.8766 - val_loss: 0.3904 - val_accuracy: 0.8631
Epoch 2/25
50000/50000 [==============================] - 19s 374us/step - loss: 0.3157 -
accuracy: 0.8837 - val_loss: 0.3603 - val_accuracy: 0.8742
Epoch 3/25
50000/50000 [==============================] - 19s 385us/step - loss: 0.2959 -
accuracy: 0.8904 - val_loss: 0.3500 - val_accuracy: 0.8781
Epoch 4/25
50000/50000 [==============================] - 19s 378us/step - loss: 0.2830 -
accuracy: 0.8946 - val_loss: 0.3521 - val_accuracy: 0.8736
Epoch 5/25
50000/50000 [==============================] - 19s 373us/step - loss: 0.2718 -
accuracy: 0.8981 - val_loss: 0.3396 - val_accuracy: 0.8837
Epoch 6/25
50000/50000 [==============================] - 19s 380us/step - loss: 0.2659 -
accuracy: 0.9012 - val_loss: 0.3296 - val_accuracy: 0.8864
Epoch 7/25
50000/50000 [==============================] - 19s 386us/step - loss: 0.2524 -
accuracy: 0.9056 - val_loss: 0.4011 - val_accuracy: 0.8610
Epoch 8/25
50000/50000 [==============================] - 19s 373us/step - loss: 0.2461 -
accuracy: 0.9078 - val_loss: 0.3651 - val_accuracy: 0.8818
Epoch 9/25
50000/50000 [==============================] - 19s 377us/step - loss: 0.2394 -
accuracy: 0.9099 - val_loss: 0.3575 - val_accuracy: 0.8841
Epoch 10/25
50000/50000 [==============================] - 18s 366us/step - loss: 0.2342 -
accuracy: 0.9120 - val_loss: 0.3773 - val_accuracy: 0.8818
Epoch 11/25
50000/50000 [==============================] - 19s 370us/step - loss: 0.2287 -
accuracy: 0.9124 - val_loss: 0.3558 - val_accuracy: 0.8897
Epoch 12/25
50000/50000 [==============================] - 19s 376us/step - loss: 0.2250 -
accuracy: 0.9152 - val_loss: 0.3683 - val_accuracy: 0.8881
Epoch 13/25
50000/50000 [==============================] - 18s 367us/step - loss: 0.2157 -
accuracy: 0.9193 - val_loss: 0.3780 - val_accuracy: 0.8829
Epoch 14/25
50000/50000 [==============================] - 18s 364us/step - loss: 0.2136 -
accuracy: 0.9199 - val_loss: 0.3850 - val_accuracy: 0.8864
```

```
Epoch 15/25
50000/50000 [==============================] - 18s 363us/step - loss: 0.2086 -
accuracy: 0.9216 - val_loss: 0.3674 - val_accuracy: 0.8895
Epoch 16/25
50000/50000 [==============================] - 19s 381us/step - loss: 0.2046 -
accuracy: 0.9222 - val_loss: 0.3868 - val_accuracy: 0.8852
Epoch 17/25
50000/50000 [==============================] - 19s 378us/step - loss: 0.2002 -
accuracy: 0.9259 - val_loss: 0.3961 - val_accuracy: 0.8882
Epoch 18/25
50000/50000 [==============================] - 19s 381us/step - loss: 0.1968 -
accuracy: 0.9248 - val_loss: 0.3803 - val_accuracy: 0.8906
Epoch 19/25
50000/50000 [==============================] - 18s 363us/step - loss: 0.1925 -
accuracy: 0.9279 - val_loss: 0.3952 - val_accuracy: 0.8875
Epoch 20/25
50000/50000 [==============================] - 19s 372us/step - loss: 0.1926 -
accuracy: 0.9283 - val_loss: 0.4026 - val_accuracy: 0.8857
Epoch 21/25
50000/50000 [==============================] - 18s 365us/step - loss: 0.1872 -
accuracy: 0.9294 - val_loss: 0.4204 - val_accuracy: 0.8885
Epoch 22/25
50000/50000 [==============================] - 18s 366us/step - loss: 0.1807 -
accuracy: 0.9321 - val_loss: 0.4350 - val_accuracy: 0.8905
Epoch 23/25
50000/50000 [==============================] - 18s 363us/step - loss: 0.1827 -
accuracy: 0.9319 - val_loss: 0.4333 - val_accuracy: 0.8838
Epoch 24/25
50000/50000 [==============================] - 18s 367us/step - loss: 0.1796 -
accuracy: 0.9312 - val_loss: 0.4880 - val_accuracy: 0.8847
Epoch 25/25
50000/50000 [==============================] - 19s 385us/step - loss: 0.1739 -
accuracy: 0.9343 - val_loss: 0.4971 - val_accuracy: 0.8850
```

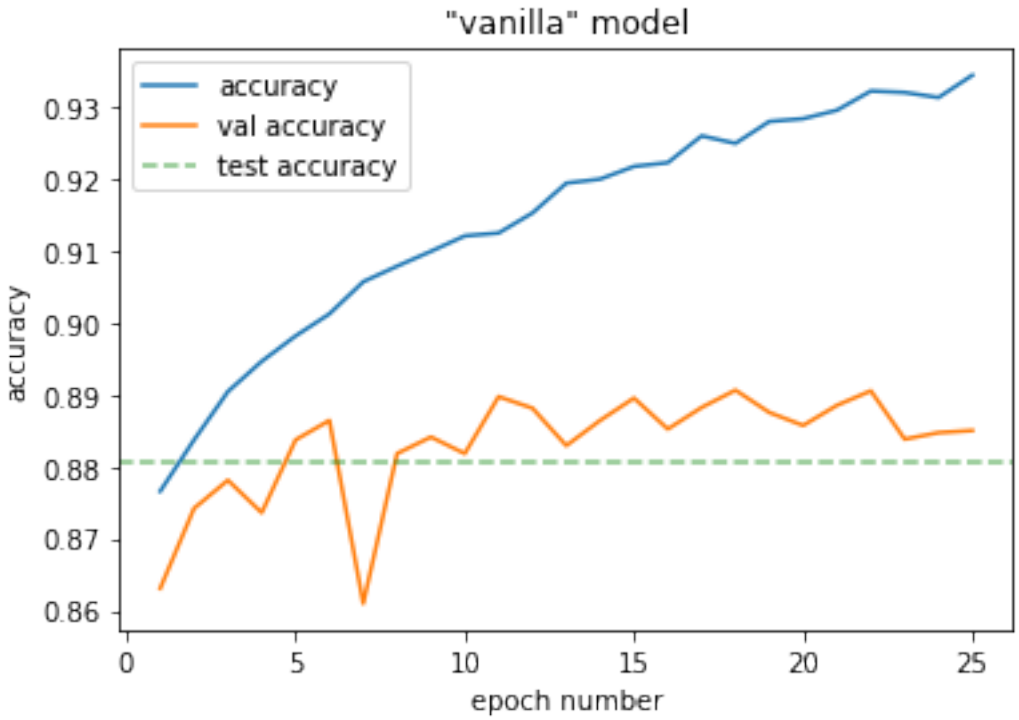
[49]:
```
basic_model_score = basic_model.evaluate(X_test, y_test)
print("\"Vanilla\" model test loss: {:.3f}".format(basic_model_score[0]))
print("\"Vanilla\" model test accuracy: {:.3f}".format(basic_model_score[1]))
```

```
10000/10000 [==============================] - 0s 48us/step
"Vanilla" model test loss: 0.537
"Vanilla" model test accuracy: 0.881
```

[50]:
```
basic_accuracy = basic_model_history.history['accuracy']
basic_val_accuracy = basic_model_history.history['val_accuracy']
basic_loss = basic_model_history.history['loss']
basic_val_loss = basic_model_history.history['val_loss']
```

```
plt.plot(range(1, len(basic_accuracy) + 1), basic_accuracy, label = 'accuracy')
plt.plot(range(1, len(basic_val_accuracy) + 1), basic_val_accuracy, label =␣
↪'val accuracy')
plt.axhline(y = basic_model_score[1], color = 'g', linestyle='--', label =␣
↪'test accuracy', alpha = 0.5)
plt.xlabel('epoch number')
plt.ylabel('accuracy')
plt.title('\"vanilla\" model')
plt.legend()
```

[50]: <matplotlib.legend.Legend at 0x7fbeda44ef98>



The training accuracy does not align well with the validation or test accuracy as the number of epochs increases.

It takes a few epochs for the training accuracy to ramp up.

This could mean that there is some level of overfitting occuring on the training set.

1.2 Drop-out model

[ ]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation, Dropout
```

```
dropout_model = Sequential(
    [
        # 1st hidden layer
        Dense(392, input_shape = (784, )),
        Activation('relu'),
        Dropout(0.5),

        # 2nd hidden layer
        Dense(98),
        Activation('relu'),
        Dropout(0.5),

        # 3rd hidden layer
        Dense(24),
        Activation('relu'),
        Dropout(0.5),

        # Output layer
        Dense(10),
        Activation('softmax')
    ]
)
```

[ ]:
```
dropout_model.compile("adam", "categorical_crossentropy", metrics=['accuracy'])
```

[30]:
```
# Validation split = 1/6 gives us 10000 samples since we 60000 training samples
dropout_model_history = dropout_model.fit(X_train, y_train, batch_size = 8,␣
 ↪epochs = 25, verbose = 1, validation_split = 1/6)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/25
50000/50000 [==============================] - 24s 476us/step - loss: 1.0363 -
accuracy: 0.5962 - val_loss: 0.6172 - val_accuracy: 0.7254
Epoch 2/25
50000/50000 [==============================] - 23s 469us/step - loss: 0.7838 -
accuracy: 0.6883 - val_loss: 0.5846 - val_accuracy: 0.7349
Epoch 3/25
50000/50000 [==============================] - 23s 462us/step - loss: 0.7407 -
accuracy: 0.7046 - val_loss: 0.5580 - val_accuracy: 0.7644
Epoch 4/25
50000/50000 [==============================] - 23s 463us/step - loss: 0.7177 -
accuracy: 0.7197 - val_loss: 0.5467 - val_accuracy: 0.7862
Epoch 5/25
50000/50000 [==============================] - 23s 462us/step - loss: 0.6847 -
accuracy: 0.7381 - val_loss: 0.5273 - val_accuracy: 0.8031
Epoch 6/25
50000/50000 [==============================] - 23s 460us/step - loss: 0.6678 -
```

```
accuracy: 0.7578 - val_loss: 0.4960 - val_accuracy: 0.8308
Epoch 7/25
50000/50000 [==============================] - 23s 468us/step - loss: 0.6470 -
accuracy: 0.7753 - val_loss: 0.4701 - val_accuracy: 0.8400
Epoch 8/25
50000/50000 [==============================] - 23s 464us/step - loss: 0.6328 -
accuracy: 0.7808 - val_loss: 0.4651 - val_accuracy: 0.8390
Epoch 9/25
50000/50000 [==============================] - 23s 470us/step - loss: 0.6308 -
accuracy: 0.7842 - val_loss: 0.4658 - val_accuracy: 0.8447
Epoch 10/25
50000/50000 [==============================] - 23s 457us/step - loss: 0.6235 -
accuracy: 0.7880 - val_loss: 0.4443 - val_accuracy: 0.8450
Epoch 11/25
50000/50000 [==============================] - 23s 467us/step - loss: 0.6094 -
accuracy: 0.7905 - val_loss: 0.4447 - val_accuracy: 0.8499
Epoch 12/25
50000/50000 [==============================] - 23s 461us/step - loss: 0.6015 -
accuracy: 0.7944 - val_loss: 0.4379 - val_accuracy: 0.8524
Epoch 13/25
50000/50000 [==============================] - 23s 465us/step - loss: 0.6032 -
accuracy: 0.7967 - val_loss: 0.4281 - val_accuracy: 0.8571
Epoch 14/25
50000/50000 [==============================] - 24s 475us/step - loss: 0.5974 -
accuracy: 0.7987 - val_loss: 0.4495 - val_accuracy: 0.8458
Epoch 15/25
50000/50000 [==============================] - 23s 464us/step - loss: 0.5831 -
accuracy: 0.8015 - val_loss: 0.4512 - val_accuracy: 0.8479
Epoch 16/25
50000/50000 [==============================] - 23s 461us/step - loss: 0.5884 -
accuracy: 0.8014 - val_loss: 0.4385 - val_accuracy: 0.8552
Epoch 17/25
50000/50000 [==============================] - 24s 473us/step - loss: 0.5859 -
accuracy: 0.8026 - val_loss: 0.4436 - val_accuracy: 0.8509
Epoch 18/25
50000/50000 [==============================] - 24s 476us/step - loss: 0.5853 -
accuracy: 0.8034 - val_loss: 0.4290 - val_accuracy: 0.8595
Epoch 19/25
50000/50000 [==============================] - 23s 468us/step - loss: 0.5753 -
accuracy: 0.8074 - val_loss: 0.4493 - val_accuracy: 0.8515
Epoch 20/25
50000/50000 [==============================] - 23s 461us/step - loss: 0.5687 -
accuracy: 0.8077 - val_loss: 0.4259 - val_accuracy: 0.8589
Epoch 21/25
50000/50000 [==============================] - 23s 466us/step - loss: 0.5804 -
accuracy: 0.8080 - val_loss: 0.4171 - val_accuracy: 0.8620
Epoch 22/25
50000/50000 [==============================] - 24s 474us/step - loss: 0.5720 -
```

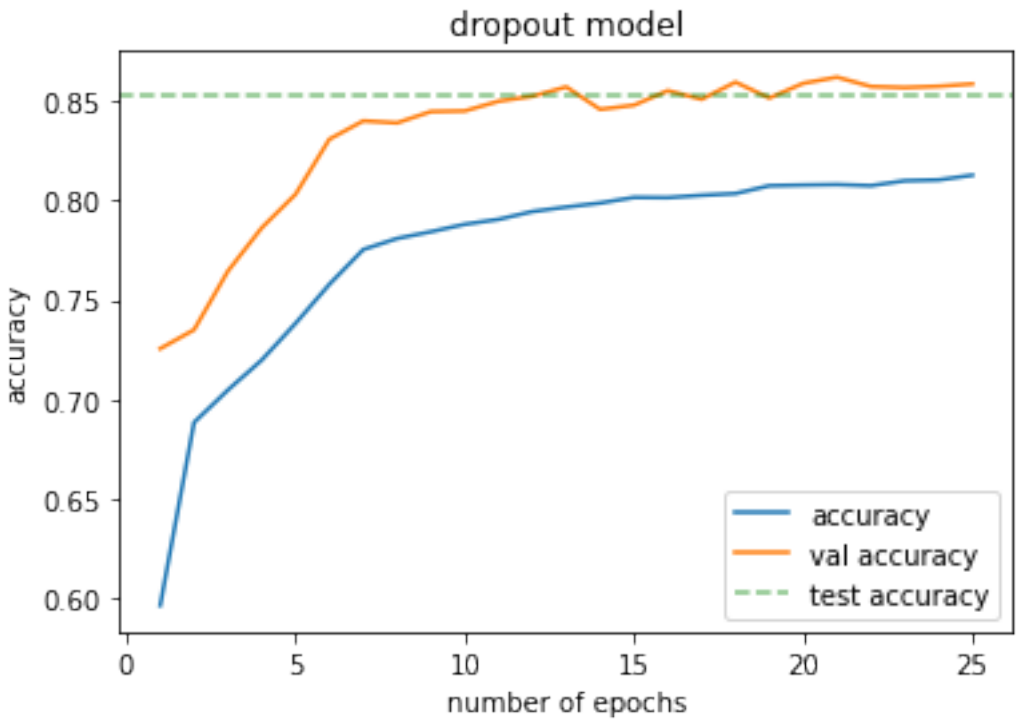
```
accuracy: 0.8074 - val_loss: 0.4313 - val_accuracy: 0.8573
Epoch 23/25
50000/50000 [==============================] - 23s 467us/step - loss: 0.5691 -
accuracy: 0.8099 - val_loss: 0.4409 - val_accuracy: 0.8568
Epoch 24/25
50000/50000 [==============================] - 23s 459us/step - loss: 0.5740 -
accuracy: 0.8103 - val_loss: 0.4424 - val_accuracy: 0.8574
Epoch 25/25
50000/50000 [==============================] - 23s 462us/step - loss: 0.5608 -
accuracy: 0.8127 - val_loss: 0.4305 - val_accuracy: 0.8586
```

[51]:
```
dropout_model_score = dropout_model.evaluate(X_test, y_test)
print("Drop-out model test loss: {:.3f}".format(dropout_model_score[0]))
print("Drop-out model test accuracy: {:.3f}".format(dropout_model_score[1]))
```

```
10000/10000 [==============================] - 0s 48us/step
Drop-out model test loss: 0.466
Drop-out model test accuracy: 0.853
```

[52]:
```
dropout_accuracy = dropout_model_history.history['accuracy']
dropout_val_accuracy = dropout_model_history.history['val_accuracy']
dropout_loss = dropout_model_history.history['loss']
dropout_val_loss = dropout_model_history.history['val_loss']

plt.plot(range(1, len(dropout_accuracy) + 1), dropout_accuracy, label =␣
↪'accuracy')
plt.plot(range(1, len(dropout_val_accuracy) + 1), dropout_val_accuracy, label =␣
↪'val accuracy')
plt.axhline(y = dropout_model_score[1], color = 'g', linestyle='--', label =␣
↪'test accuracy', alpha = 0.5)
plt.xlabel('number of epochs')
plt.ylabel('accuracy')
plt.title('dropout model')
plt.legend()
```

[52]:
```
<matplotlib.legend.Legend at 0x7fbeda3ff1d0>
```



It takes a few epochs (5-7) for the training accuracy to ramp up.

The test accuracy is higher than the training accuracy.

This could mean that the model is more robust and does not overfit on the training set when compared to the "vanilla" model.

1.3 Batch Normalization model

[ ]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation, BatchNormalization

bn_model = Sequential(
    [
        # 1st hidden layer
        Dense(196, input_shape = (784, )),
        Activation('relu'),
        BatchNormalization(),

        # 2nd hidden layer
        Dense(49),
        Activation('relu'),
        BatchNormalization(),

        # Output layer
```

```
        Dense(10),
        Activation('softmax')
    ]
)
```

[ ]:
```
bn_model.compile("adam", "categorical_crossentropy", metrics=['accuracy'])
```

[35]:
```
# Validation split = 1/6 gives us 10000 samples since we 60000 training samples
bn_model_history = bn_model.fit(X_train, y_train, batch_size = 8, epochs = 25,␣
↪verbose = 1, validation_split = 1/6)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/25
50000/50000 [==============================] - 30s 594us/step - loss: 0.6259 -
accuracy: 0.7822 - val_loss: 0.4426 - val_accuracy: 0.8358
Epoch 2/25
50000/50000 [==============================] - 30s 598us/step - loss: 0.5141 -
accuracy: 0.8203 - val_loss: 0.4619 - val_accuracy: 0.8413
Epoch 3/25
50000/50000 [==============================] - 30s 595us/step - loss: 0.4788 -
accuracy: 0.8310 - val_loss: 0.4149 - val_accuracy: 0.8556
Epoch 4/25
50000/50000 [==============================] - 30s 596us/step - loss: 0.4583 -
accuracy: 0.8380 - val_loss: 0.4319 - val_accuracy: 0.8470
Epoch 5/25
50000/50000 [==============================] - 30s 607us/step - loss: 0.4444 -
accuracy: 0.8419 - val_loss: 0.3860 - val_accuracy: 0.8603
Epoch 6/25
50000/50000 [==============================] - 29s 582us/step - loss: 0.4309 -
accuracy: 0.8468 - val_loss: 0.3997 - val_accuracy: 0.8605
Epoch 7/25
50000/50000 [==============================] - 30s 595us/step - loss: 0.4201 -
accuracy: 0.8483 - val_loss: 0.3894 - val_accuracy: 0.8643
Epoch 8/25
50000/50000 [==============================] - 29s 576us/step - loss: 0.4051 -
accuracy: 0.8560 - val_loss: 0.3873 - val_accuracy: 0.8673
Epoch 9/25
50000/50000 [==============================] - 29s 585us/step - loss: 0.4044 -
accuracy: 0.8564 - val_loss: 0.3954 - val_accuracy: 0.8683
Epoch 10/25
50000/50000 [==============================] - 30s 590us/step - loss: 0.3982 -
accuracy: 0.8574 - val_loss: 0.3827 - val_accuracy: 0.8683
Epoch 11/25
50000/50000 [==============================] - 29s 588us/step - loss: 0.3941 -
accuracy: 0.8578 - val_loss: 0.3774 - val_accuracy: 0.8653
Epoch 12/25
50000/50000 [==============================] - 30s 608us/step - loss: 0.3862 -
```

```
accuracy: 0.8613 - val_loss: 0.3933 - val_accuracy: 0.8609
Epoch 13/25
50000/50000 [==============================] - 30s 595us/step - loss: 0.3827 -
accuracy: 0.8628 - val_loss: 0.3584 - val_accuracy: 0.8723
Epoch 14/25
50000/50000 [==============================] - 29s 588us/step - loss: 0.3804 -
accuracy: 0.8640 - val_loss: 0.3928 - val_accuracy: 0.8656
Epoch 15/25
50000/50000 [==============================] - 29s 589us/step - loss: 0.3713 -
accuracy: 0.8654 - val_loss: 0.3550 - val_accuracy: 0.8726
Epoch 16/25
50000/50000 [==============================] - 29s 576us/step - loss: 0.3713 -
accuracy: 0.8653 - val_loss: 0.3748 - val_accuracy: 0.8679
Epoch 17/25
50000/50000 [==============================] - 29s 578us/step - loss: 0.3750 -
accuracy: 0.8636 - val_loss: 0.3787 - val_accuracy: 0.8702
Epoch 18/25
50000/50000 [==============================] - 29s 575us/step - loss: 0.3748 -
accuracy: 0.8657 - val_loss: 0.3906 - val_accuracy: 0.8625
Epoch 19/25
50000/50000 [==============================] - 29s 579us/step - loss: 0.3739 -
accuracy: 0.8660 - val_loss: 0.3705 - val_accuracy: 0.8665
Epoch 20/25
50000/50000 [==============================] - 29s 589us/step - loss: 0.3621 -
accuracy: 0.8689 - val_loss: 0.3668 - val_accuracy: 0.8696
Epoch 21/25
50000/50000 [==============================] - 29s 581us/step - loss: 0.3561 -
accuracy: 0.8703 - val_loss: 0.3601 - val_accuracy: 0.8751
Epoch 22/25
50000/50000 [==============================] - 29s 587us/step - loss: 0.3516 -
accuracy: 0.8732 - val_loss: 0.3747 - val_accuracy: 0.8692
Epoch 23/25
50000/50000 [==============================] - 29s 584us/step - loss: 0.3505 -
accuracy: 0.8729 - val_loss: 0.3799 - val_accuracy: 0.8690
Epoch 24/25
50000/50000 [==============================] - 29s 587us/step - loss: 0.3479 -
accuracy: 0.8750 - val_loss: 0.3739 - val_accuracy: 0.8663
Epoch 25/25
50000/50000 [==============================] - 28s 567us/step - loss: 0.3502 -
accuracy: 0.8740 - val_loss: 0.3667 - val_accuracy: 0.8720
```

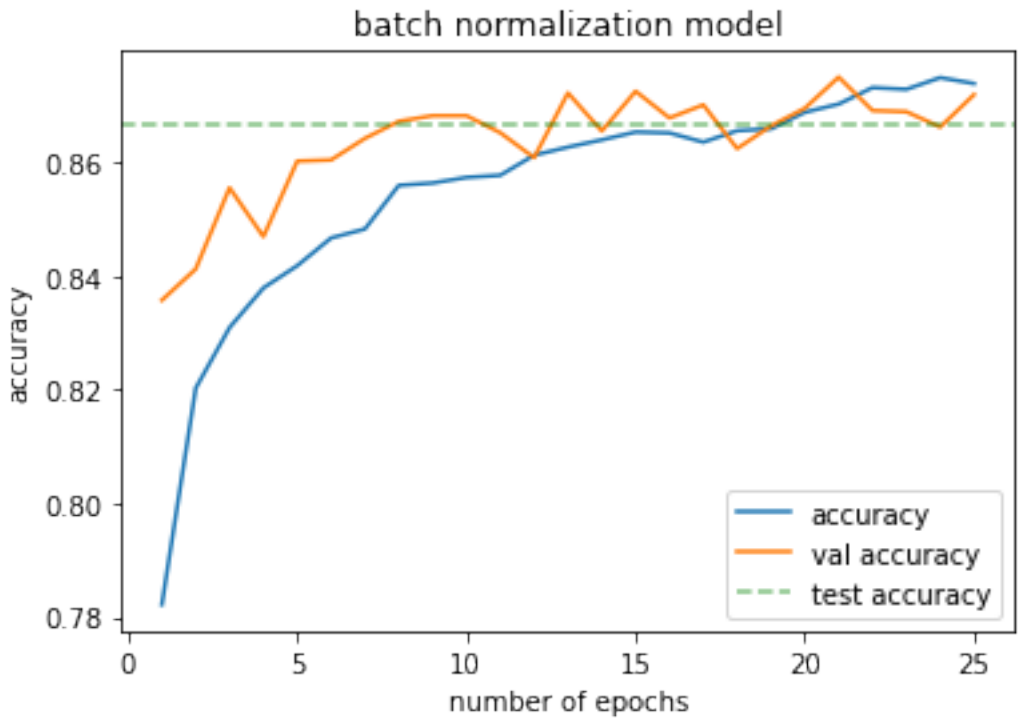
[53]:
```
bn_model_score = bn_model.evaluate(X_test, y_test)
print("Batch Normalization model test loss: {:.3f}".format(bn_model_score[0]))
print("Batch Normalization model test accuracy: {:.3f}".
 ↪format(bn_model_score[1]))
```

```
10000/10000 [==============================] - 1s 59us/step
Batch Normalization model test loss: 0.372
```

```
Batch Normalization model test accuracy: 0.867
```

[54]:
```
bn_accuracy = bn_model_history.history['accuracy']
bn_val_accuracy = bn_model_history.history['val_accuracy']
bn_loss = bn_model_history.history['loss']
bn_val_loss = bn_model_history.history['val_loss']

plt.plot(range(1, len(bn_accuracy) + 1), bn_accuracy, label = 'accuracy')
plt.plot(range(1, len(bn_val_accuracy) + 1), bn_val_accuracy, label = 'val␣
↪accuracy')
plt.axhline(y = bn_model_score[1], color = 'g', linestyle='--', label = 'test␣
↪accuracy', alpha = 0.5)
plt.xlabel('number of epochs')
plt.ylabel('accuracy')
plt.title('batch normalization model')
plt.legend()
```

[54]:
```
<matplotlib.legend.Legend at 0x7fbeda30e940>
```



The training accuracy ramps up very quickly compared to the previous models (2-3 epochs compared to the dropout model).

The test accuracy is higher than the training accuracy for up to 20 epochs. This could mean that the model is more robust and does not overfit on the training set when compared to the "vanilla" model.

1.4 Batch Normalization with Drop-out model

[ ]:
```
from keras.models import Sequential
from keras.layers import Dense, Activation, BatchNormalization

bn_dp_model = Sequential(
    [
        # 1st hidden layer
        Dense(196, input_shape = (784, )),
        Activation('relu'),
        Dropout(0.5),
        BatchNormalization(),

        # 2nd hidden layer
        Dense(49),
        Activation('relu'),
        Dropout(0.5),
        BatchNormalization(),

        # Output layer
        Dense(10),
        Activation('softmax')
    ]
)
```

[ ]:
```
bn_dp_model.compile("adam", "categorical_crossentropy", metrics=['accuracy'])
```

[46]:
```
# Validation split = 1/6 gives us 10000 samples since we 60000 training samples
bn_dp_model_history = bn_dp_model.fit(X_train, y_train, batch_size = 8, epochs␣
 ↪= 25, verbose = 1, validation_split = 1/6)
```

```
Train on 50000 samples, validate on 10000 samples
Epoch 1/25
50000/50000 [==============================] - 30s 610us/step - loss: 1.3851 -
accuracy: 0.4891 - val_loss: 0.8219 - val_accuracy: 0.7360
Epoch 2/25
50000/50000 [==============================] - 31s 620us/step - loss: 1.1802 -
accuracy: 0.5575 - val_loss: 0.7442 - val_accuracy: 0.7584
Epoch 3/25
50000/50000 [==============================] - 31s 620us/step - loss: 1.1105 -
accuracy: 0.5884 - val_loss: 0.7264 - val_accuracy: 0.7558
Epoch 4/25
50000/50000 [==============================] - 32s 634us/step - loss: 1.1004 -
accuracy: 0.5929 - val_loss: 0.7413 - val_accuracy: 0.7495
Epoch 5/25
50000/50000 [==============================] - 31s 618us/step - loss: 1.0750 -
accuracy: 0.6034 - val_loss: 0.8034 - val_accuracy: 0.7114
Epoch 6/25
```

```
50000/50000 [==============================] - 30s 602us/step - loss: 1.0852 -
accuracy: 0.5984 - val_loss: 0.7065 - val_accuracy: 0.7343
Epoch 7/25
50000/50000 [==============================] - 31s 616us/step - loss: 1.0411 -
accuracy: 0.6172 - val_loss: 0.6892 - val_accuracy: 0.7426
Epoch 8/25
50000/50000 [==============================] - 31s 623us/step - loss: 1.0412 -
accuracy: 0.6157 - val_loss: 0.6734 - val_accuracy: 0.7550
Epoch 9/25
50000/50000 [==============================] - 31s 626us/step - loss: 1.0501 -
accuracy: 0.6124 - val_loss: 0.6755 - val_accuracy: 0.7623
Epoch 10/25
50000/50000 [==============================] - 30s 609us/step - loss: 1.0263 -
accuracy: 0.6256 - val_loss: 0.6590 - val_accuracy: 0.7540
Epoch 11/25
50000/50000 [==============================] - 31s 625us/step - loss: 1.0292 -
accuracy: 0.6186 - val_loss: 0.6720 - val_accuracy: 0.7708
Epoch 12/25
50000/50000 [==============================] - 31s 621us/step - loss: 1.0452 -
accuracy: 0.6156 - val_loss: 0.7124 - val_accuracy: 0.7727
Epoch 13/25
50000/50000 [==============================] - 31s 623us/step - loss: 1.0224 -
accuracy: 0.6217 - val_loss: 0.6864 - val_accuracy: 0.7585
Epoch 14/25
50000/50000 [==============================] - 31s 629us/step - loss: 1.0431 -
accuracy: 0.6162 - val_loss: 0.6763 - val_accuracy: 0.7652
Epoch 15/25
50000/50000 [==============================] - 30s 610us/step - loss: 1.0185 -
accuracy: 0.6271 - val_loss: 0.6933 - val_accuracy: 0.7820
Epoch 16/25
50000/50000 [==============================] - 30s 609us/step - loss: 1.0060 -
accuracy: 0.6333 - val_loss: 0.7301 - val_accuracy: 0.7346
Epoch 17/25
50000/50000 [==============================] - 32s 637us/step - loss: 1.0000 -
accuracy: 0.6323 - val_loss: 0.6722 - val_accuracy: 0.7719
Epoch 18/25
50000/50000 [==============================] - 31s 628us/step - loss: 0.9815 -
accuracy: 0.6388 - val_loss: 0.6421 - val_accuracy: 0.7547
Epoch 19/25
50000/50000 [==============================] - 30s 606us/step - loss: 0.9842 -
accuracy: 0.6404 - val_loss: 0.6367 - val_accuracy: 0.7764
Epoch 20/25
50000/50000 [==============================] - 30s 609us/step - loss: 0.9765 -
accuracy: 0.6414 - val_loss: 0.6464 - val_accuracy: 0.7728
Epoch 21/25
50000/50000 [==============================] - 30s 592us/step - loss: 0.9783 -
accuracy: 0.6394 - val_loss: 0.6529 - val_accuracy: 0.7765
Epoch 22/25
```

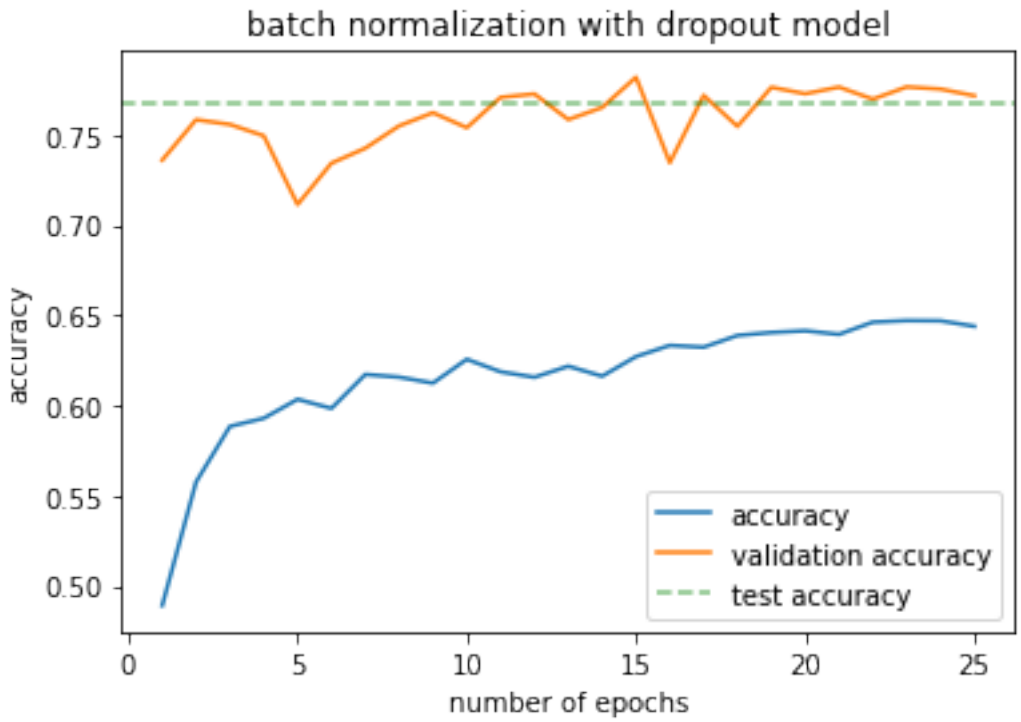
```
50000/50000 [==============================] - 30s 593us/step - loss: 0.9806 -
accuracy: 0.6462 - val_loss: 0.7581 - val_accuracy: 0.7698
Epoch 23/25
50000/50000 [==============================] - 30s 594us/step - loss: 0.9673 -
accuracy: 0.6470 - val_loss: 0.6397 - val_accuracy: 0.7766
Epoch 24/25
50000/50000 [==============================] - 31s 614us/step - loss: 0.9728 -
accuracy: 0.6469 - val_loss: 0.6426 - val_accuracy: 0.7754
Epoch 25/25
50000/50000 [==============================] - 31s 621us/step - loss: 0.9728 -
accuracy: 0.6439 - val_loss: 0.6519 - val_accuracy: 0.7717
```

[47]:
```
bn_dp_model_score = bn_dp_model.evaluate(X_test, y_test)
print("Batch Normalization with Drop-out model test loss: {:.3f}".
 ↪format(bn_dp_model_score[0]))
print("Batch Normalization with Drop-out model test accuracy: {:.3f}".
 ↪format(bn_dp_model_score[1]))
```

```
10000/10000 [==============================] - 1s 64us/step
Test loss: 0.646
Test Accuracy: 0.768
```

[48]:
```
bn_dp_accuracy = bn_dp_model_history.history['accuracy']
bn_dp_val_accuracy = bn_dp_model_history.history['val_accuracy']
bn_dp_loss = bn_dp_model_history.history['loss']
bn_dp_val_loss = bn_dp_model_history.history['val_loss']

plt.plot(range(1, len(bn_dp_accuracy) + 1), bn_dp_accuracy, label = 'accuracy')
plt.plot(range(1, len(bn_dp_val_accuracy) + 1), bn_dp_val_accuracy, label =␣
 ↪'validation accuracy')
plt.axhline(y = bn_dp_model_score[1], color = 'g', linestyle='--', label =␣
 ↪'test accuracy', alpha = 0.5)
plt.xlabel('number of epochs')
plt.ylabel('accuracy')
plt.title('batch normalization with dropout model')
plt.legend()
```

[48]:
```
<matplotlib.legend.Legend at 0x7fbedb6534e0>
```

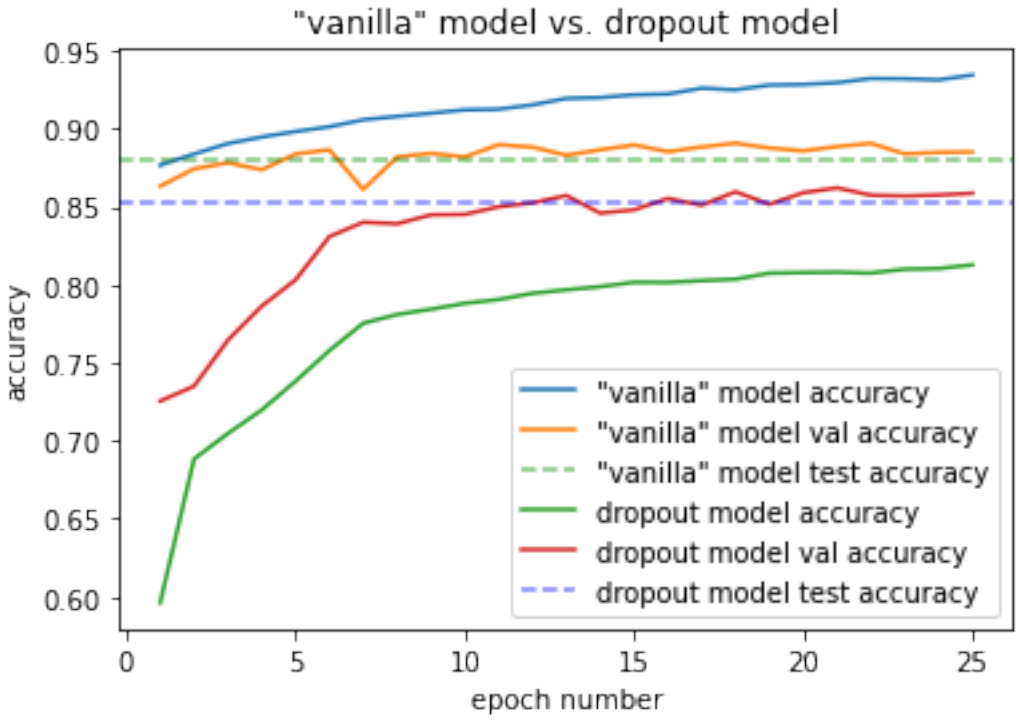


1.5 Comparison of "vanilla" model with the other models

[55]:
```
# "vanilla" model vs. dropout model

# "vanilla" model
plt.plot(range(1, len(basic_accuracy) + 1), basic_accuracy, label =␣
 ↪'\"vanilla\" model accuracy')
plt.plot(range(1, len(basic_val_accuracy) + 1), basic_val_accuracy, label =␣
 ↪'\"vanilla\" model val accuracy')
plt.axhline(y = basic_model_score[1], color = 'g', linestyle='--', label =␣
 ↪'\"vanilla\" model test accuracy', alpha = 0.5)

# dropout model
plt.plot(range(1, len(dropout_accuracy) + 1), dropout_accuracy, label =␣
 ↪'dropout model accuracy')
plt.plot(range(1, len(dropout_val_accuracy) + 1), dropout_val_accuracy, label =␣
 ↪'dropout model val accuracy')
plt.axhline(y = dropout_model_score[1], color = 'b', linestyle='--', label =␣
 ↪'dropout model test accuracy', alpha = 0.5)

plt.xlabel('epoch number')
plt.ylabel('accuracy')
plt.title('\"vanilla\" model vs. dropout model')
```

```
plt.legend()
```

[55]: <matplotlib.legend.Legend at 0x7fbeda274e10>



Training the dropout model took a bit longer than training the "vanilla" model since we had added more layer(s).

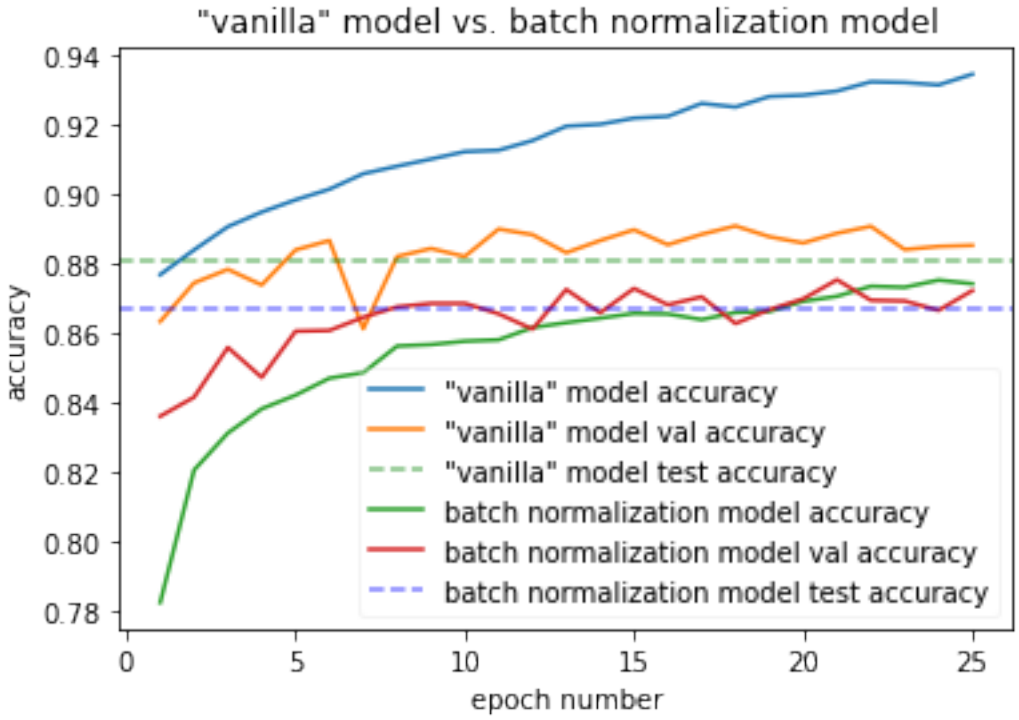
The test accuracy for the "vanilla" model is higher than the test accuracy for the dropout model which might indicate that our dropout model needs to be trained for a larger number of epochs.

It may be the case that the dropout model is more robust than the "vanilla" model since it forces the neurons in the hidden layers to account for the neurons being dropped leading to higher generalization.

[61]:
```
# "vanilla" model vs. batch normalization model

# "vanilla" model
plt.plot(range(1, len(basic_accuracy) + 1), basic_accuracy, label =␣
 ↪'\"vanilla\" model accuracy')
plt.plot(range(1, len(basic_val_accuracy) + 1), basic_val_accuracy, label =␣
 ↪'\"vanilla\" model val accuracy')
plt.axhline(y = basic_model_score[1], color = 'g', linestyle='--', label =␣
 ↪'\"vanilla\" model test accuracy', alpha = 0.5)

# batch normalization model
```

```
plt.plot(range(1, len(bn_accuracy) + 1), bn_accuracy, label = 'batch␣
 ↪normalization model accuracy')
plt.plot(range(1, len(bn_val_accuracy) + 1), bn_val_accuracy, label = 'batch␣
 ↪normalization model val accuracy')
plt.axhline(y = bn_model_score[1], color = 'b', linestyle='--', label = 'batch␣
 ↪normalization model test accuracy', alpha = 0.5)

plt.xlabel('epoch number')
plt.ylabel('accuracy')
plt.title('\"vanilla\" model vs. batch normalization model')
plt.legend(framealpha = 0.2)
```

[61]: <matplotlib.legend.Legend at 0x7fbed9ec1e80>



Training the batch normalization model took a lot longer than training the "vanilla" model since we added another step in-between the hidden layers.

However, as mentioned above, it "comes online" very quickly (2-3 epochs) compared to the "vanilla" model and the dropout model.

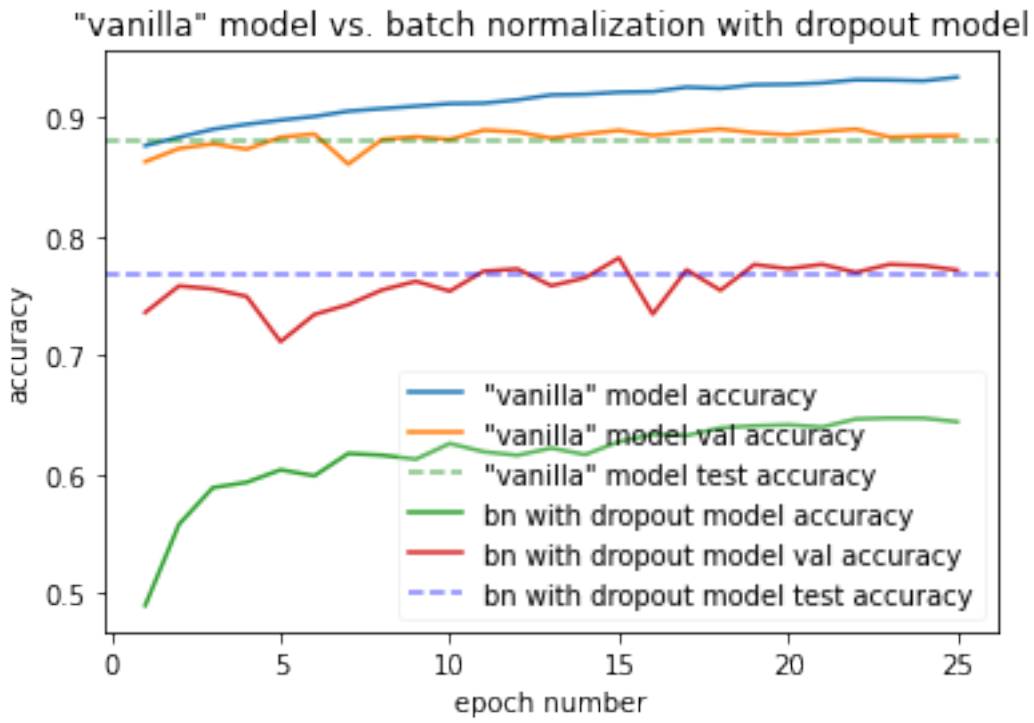
Similar to the dropout model, the test accuracy for the batch normalization model is lower than that of the "vanilla" model which might indicate that our batch normalization model needs to be trained for a larger number of epochs.

Also, it may be the case that the batch normalization model is more robust than the "vanilla"

model since we're basically applying a StandardScaler to the activation in each layer allowing for quicker convergence to the optimum.

[60]:
```
# "vanilla" model vs. batch normalization with dropout model

# "vanilla" model
plt.plot(range(1, len(basic_accuracy) + 1), basic_accuracy, label =␣
↪'\"vanilla\" model accuracy')
plt.plot(range(1, len(basic_val_accuracy) + 1), basic_val_accuracy, label =␣
↪'\"vanilla\" model val accuracy')
plt.axhline(y = basic_model_score[1], color = 'g', linestyle='--', label =␣
↪'\"vanilla\" model test accuracy', alpha = 0.5)

# batch normalization with dropout model
plt.plot(range(1, len(bn_dp_accuracy) + 1), bn_dp_accuracy, label = 'bn with␣
↪dropout model accuracy')
plt.plot(range(1, len(bn_dp_val_accuracy) + 1), bn_dp_val_accuracy, label = 'bn␣
↪with dropout model val accuracy')
plt.axhline(y = bn_dp_model_score[1], color = 'b', linestyle='--', label = 'bn␣
↪with dropout model test accuracy', alpha = 0.5)

plt.xlabel('epoch number')
plt.ylabel('accuracy')
plt.title('\"vanilla\" model vs. batch normalization with dropout model')
plt.legend(loc = 'lower right', framealpha = 0.2)
```

[60]:
```
<matplotlib.legend.Legend at 0x7fbed9f63588>
```



Training the batch normalization with dropout model took almost the same time as training the batch normalization model since we did not add any additional layers and only included dropout in each hidden layer.

It performs very poorly when compared to the “vanilla” model.

Perhaps it can be improved by choosing different number of hidden layer nodes and different number of layers.

As it stands now, this is not a good model when compared to the “vanilla” model.

2 References

Dataset

Drop-out