Deep neural network with stacked autoencoder on MNIST #358

Closed darzok opened this issue on Jul 8, 2015 · 33 comments

Labels stale

darzok commented on Jul 8, 2015

Helo,

firts of all sorry for my english, it's not my native language (I'm french)

As the tittle said, I'm trying to train deep neural network with stack autoencoder but I'm stuck ...
thanks to fchollet's exemple I managed to implement a simple deep neural network that is work thinks to ReLU activation function (Xavier Glorot thesis).
But now I want to compar the result I have with this simple deep neural network to a deep network with stack auto encoder pre training.

I start with this code but I don't know how I can continue ... and everytime I try to add code I have an error ... so this is my valid code :

from **future** import absolute_import
from **future** import print_function
import numpy as np
np.random.seed(1337) # for reproducibility

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation, AutoEncoder, Layer
from keras.optimizers import SGD, Adam, RMSprop, Adagrad, Adadelta
from keras.utils import np_utils
from keras.utils.dot_utils import Grapher
from keras.callbacks import ModelCheckpoint

batch_size = 10000
nb_classes = 10
nb_epoch = 1

the data, shuffled and split between train and test sets

(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype("float64")
X_test = X_test.astype("float64")
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

convert class vectors to binary class matrices

Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)

model = Sequential()

creating the autoencoder

first hidden layer

model.add(AutoEncoder(encoder=Dense(784, 700),
decoder=Dense(700, 784),
output_reconstruction=False, tie_weights=True))
model.add(Activation('tanh'))

model.compile(loss='mean_squared_error', optimizer=rms)
model.fit(X_train, X_train, batch_size=batch_size, nb_epoch=nb_epoch, show_accuracy=False, verbose=1, validation_data=None)

for layer in model.layers:

config=layer.get_config()

```
  weight=layer.get_weights()
```

print(weight)

second hidden layer

model.add(AutoEncoder(encoder=Dense(700, 600),
decoder=Dense(600, 700),
output_reconstruction=False, tie_weights=True))
model.add(Activation('tanh'))

model.compile(loss='mean_squared_error', optimizer=rms)
model.fit(X_train, X_train, batch_size=batch_size, nb_epoch=nb_epoch, show_accuracy=False, verbose=1, validation_data=None)

Pre-training

fine-tuning

autoencoder evaluation

mthrok commented on Jul 8, 2015

Not sure if this is what you are looking for but the following works.
(which is direct use of example in documentation http://keras.io/layers/core/#autoencoder)

```
from __future__ import absolute_import
from __future__ import print_function
import numpy as np
np.random.seed(1337) # for reproducibility

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import containers
from keras.layers.core import Dense, AutoEncoder
from keras.optimizers import RMSprop
from keras.utils import np_utils

batch_size = 64
nb_classes = 10
nb_epoch = 1
```

```
X_train = X_train.reshape(-1, 784)
X_test = X_test.reshape(-1, 784)
X_train = X_train.astype("float32") / 255.0
X_test = X_test.astype("float32") / 255.0
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

#creating the autoencoder
ae = Sequential()
encoder = containers.Sequential([Dense(784, 700), Dense(700, 600)])
decoder = containers.Sequential([Dense(600, 700), Dense(700, 784)])
ae.add(AutoEncoder(encoder=encoder, decoder=decoder,
                   output_reconstruction=True, tie_weights=True))

ae.compile(loss='mean_squared_error', optimizer=RMSprop())
ae.fit(X_train, X_train, batch_size=batch_size, nb_epoch=nb_epoch,
       show_accuracy=False, verbose=1, validation_data=[X_test, X_test])
```

The output

```
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 0
60000/60000 [==============================] - 36s - loss: 0.0371 - val_loss: 0.0229
```

mthrok commented on Jul 8, 2015

Updated the code to show how to use `validation_data`. `output_reconstruction` must be enabled.

darzok commented on Jul 8, 2015

thank you very much for your fast reply it's very apreciable. I can't test this code right now cause I haven't my laptop with me but I'll try it tonight.

If I don't misunderstood the method for training Deep neural network with autoencoder, the first step is to train one by one each autoencoder to encode and decode their input. After the pre training is done, I can set the weights of my DNN with the weights of all encoder. Then I can apply a simple SGD.

Is my understand right ?

So if i right, my goal is to train a second autoencoder with inputs of the firs autoencoder. And that's what I don't find the way to do it. But perhaps with your code I'm going to succeed.

mthrok commented on Jul 8, 2015

What you say sounds correct.

If you need to do layer-by-layer pre-training, then I think you need to write similar scripts for each stage, save the trained weight with `save_weight` function and load it at the next stage with `load_weight` function.
This however might not work, since the documentation says that when you load saved weight with `load_weight` function, the architecture of model must be identical. But looking at the source code this might not be the case and you can simply use the weight from the previous stage.

If your goal is to do experiment with pre-trainng, you are doing it right. But if your goal is to train a network, then keep in mind that by applying glorot initialization (which is default initialization scheme in Keras), you don't need to do pre-training. You can normally directly start training the network.

jramapuram commented on Jul 8, 2015

To do layer by layer pretraining you will currently need to run fit() (or train) on each model and then couple them later after training is done using `output_reconstruction=True`

mthrok mentioned this issue on Jul 8, 2015

Multinode output and hidden layer output #365

Closed

mthrok commented on Jul 8, 2015

It looks like, I didn't put activation function. (Sorry, I have not used Keras' AE before.)
This may be more correct architecture. It needs to be checked though.

```
encoder = containers.Sequential([Dense(784, 700, activation='sigmoid'),
                                 Dense(700, 600, activation='sigmoid')])
decoder = containers.Sequential([Dense(600, 700, activation='sigmoid'),
                                 Dense(700, 784, activation='sigmoid')])
```

As a matter of fact, it certainly changes the output
Without activation

```
Epoch 0
60000/60000 [==============================] - 37s - loss: 0.0371 - val_loss: 0.0229
Epoch 1
60000/60000 [==============================] - 36s - loss: 0.0191 - val_loss: 0.0157
Epoch 2
60000/60000 [==============================] - 36s - loss: 0.0144 - val_loss: 0.0126
Epoch 3
 3136/60000 [>.............................] - ETA: 36s - loss: 0.0130
```

With activation

```
60000/60000 [==============================] - 41s - loss: 0.0719 - val_loss: 0.0677
Epoch 1
60000/60000 [==============================] - 42s - loss: 0.0674 - val_loss: 0.0676
Epoch 2
60000/60000 [==============================] - 42s - loss: 0.0673 - val_loss: 0.0675
Epoch 3
 1216/60000 [..............................] - ETA: 42s - loss: 0.0669
```

jramapuram commented on Jul 8, 2015

@mthrok : yes you can stack the layers like that, but it is not doing greedy layerwise training. Just so you are aware.

darzok commented on Jul 8, 2015

I try do something like that to do greedy layerwise but it's not working...

from **future** import absolute_import
from **future** import print_function
import numpy as np
np.random.seed(1337) # for reproducibility

```
from keras.optimizers import SGD, Adam, RMSprop, Adagrad, Adadelta
from keras.utils import np_utils
from keras.utils.dot_utils import Grapher
from keras.callbacks import ModelCheckpoint
from keras.layers import containers

batch_size = 10000
nb_classes = 10
nb_epoch = 1

adg = Adagrad()
sgd = SGD()
rms = RMSprop()

#the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()

X_train = X_train.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
X_train = X_train.astype("float64")
X_test = X_test.astype("float64")
X_train /= 255
X_test /= 255
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

#convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)

#creating the autoencoder

#first autoencoder
AE1_output_reconstruction = True
ae1 = Sequential()
encoder1 = containers.Sequential([Dense(784, 700, activation='tanh'), Dense(700, 600, activation='tanh')])
decoder1 = containers.Sequential([Dense(600, 700, activation='tanh'), Dense(700, 784, activation='tanh')])
ae1.add(AutoEncoder(encoder=encoder1, decoder=decoder1,
output_reconstruction=AE1_output_reconstruction, tie_weights=True))

#training the first autoencoder
ae1.compile(loss='mean_squared_error', optimizer=RMSprop())
ae1.fit(X_train, X_train, batch_size=batch_size, nb_epoch=nb_epoch,
show_accuracy=False, verbose=1, validation_data=[X_test, X_test])
```

```
autoencoder
FirstAeOutput = ae1.predict(X_train)

#second autoencoder
AE2_output_reconstruction = True
ae2 = Sequential()
encoder2 = containers.Sequential([Dense(600, 500, activation='tanh'), Dense(500, 400, activation='tanh')])
decoder2 = containers.Sequential([Dense(400, 500, activation='tanh'), Dense(500, 600, activation='tanh')])
ae2.add(AutoEncoder(encoder=encoder2, decoder=decoder2,
output_reconstruction=AE2_output_reconstruction, tie_weights=True))

#training the second autoencoder
ae2.compile(loss='mean_squared_error', optimizer=RMSprop())
ae2.fit(FirstAeOutput, FirstAeOutput, batch_size=batch_size, nb_epoch=nb_epoch,
show_accuracy=False, verbose=1)

#getting output of the second autoencoder to connect to the input of the
#third autoencoder
AE2_output_reconstruction = False
SecondAeOutput = ae2.predict(FirstAeOutput)

#third autoencoder
AE3_output_reconstruction = True
ae3 = Sequential()
encoder3 = containers.Sequential([Dense(400, 300, activation='tanh'), Dense(300, 200, activation='tanh')])
decoder3 = containers.Sequential([Dense(200, 300, activation='tanh'), Dense(300, 400, activation='tanh')])
ae3.add(AutoEncoder(encoder=encoder3, decoder=decoder3,
output_reconstruction=AE3_output_reconstruction, tie_weights=True))

#training the third autoencoder
ae2.compile(loss='mean_squared_error', optimizer=RMSprop())
ae2.fit(SecondAeOutput, SecondAeOutput, batch_size=batch_size, nb_epoch=nb_epoch,
show_accuracy=False, verbose=1)

#creating the Deep neural network with all encoder of each autoencoder trained before

model = Sequential()
model.add(ae1[0].encoder)
model.add(ae2[0].encoder)
model.add(ae3[0].encoder)
model.add(Dense(200, 10))
model.add(Activation('softmax'))

model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=nb_epoch, show_accuracy=True, verbose=2,
validation_data=(X_test, Y_test))
score = model.evaluate(X_test, Y_test, show_accuracy=True, verbose=0)
print('Test score:', score[0])
print('Test accuracy:', score[1])
```

darzok closed this on Jul 8, 2015

darzok commented on Jul 8, 2015

```
Traceback (most recent call last):
  File "/home/papin/Documents/MLP_MNIST (pre trainning).py", line 74, in <module>
    show_accuracy=False, verbose=1)
  File "build/bdist.linux-x86_64/egg/keras/models.py", line 205, in fit
    loss = self._train(*ins)
  File "/usr/local/lib/python2.7/dist-packages/theano/compile/function_module.py",
line 606, in __call__
    storage_map=self.fn.storage_map)
  File "/usr/local/lib/python2.7/dist-packages/theano/compile/function_module.py",
line 595, in __call__
    outputs = self.fn()
ValueError: Shape mismatch: x has 784 cols (and 10000 rows) but y has 600 rows (and
500 cols)
Apply node that caused the error: Dot22(<TensorType(float64, matrix)>, <TensorType(
float64, matrix)>)
Inputs types: [TensorType(float64, matrix), TensorType(float64, matrix)]
Inputs shapes: [(10000, 784), (600, 500)]
Inputs strides: [(6272, 8), (4000, 8)]
Inputs values: ['not shown', 'not shown']

HINT: Re-running with most Theano optimization disabled could give you a back-trace
of when this node was created. This can be done with by setting the Theano flag 'op
timizer=fast_compile'. If that does not work, Theano optimizations can be disabled
with 'optimizer=None'.
HINT: Use the Theano flag 'exception_verbosity=high' for a debugprint and storage m
ap footprint of this apply node.
```

darzok reopened this on Jul 8, 2015

mthrok commented on Jul 9, 2015

Here is a layer-by-layer example.
But one thing I am not sure is if I am reusing encoder weight correctly, because the output before fine turning is almost same as no training at all. Maybe I need to do get_weight and set_weight manually.
I will look into it later.

```
from __future__ import absolute_import
from __future__ import print_function
import numpy as np
np.random.seed(1337) # for reproducibility

from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import containers
```

```
batch_size = 64
nb_classes = 10
nb_epoch = 1
nb_hidden_layers = [784, 600, 500, 400]

# the data, shuffled and split between train and test sets
(X_train, y_train), (X_test, y_test) = mnist.load_data()
X_train = X_train.reshape(-1, 784)
X_test = X_test.reshape(-1, 784)
X_train = X_train.astype("float32") / 255.0
X_test = X_test.astype("float32") / 255.0
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# convert class vectors to binary class matrices
Y_train = np_utils.to_categorical(y_train, nb_classes)
Y_test = np_utils.to_categorical(y_test, nb_classes)

# Layer-wise pretraining
encoders = []
nb_hidden_layers = [784, 600, 500, 400]
X_train_tmp = np.copy(X_train)
for i, (n_in, n_out) in enumerate(zip(nb_hidden_layers[:-1], nb_hidden_layers[1:]), start=1):
    print('Training the layer {}: Input {} -> Output {}'.format(i, n_in, n_out))
    # Create AE and training
    ae = Sequential()
    encoder = containers.Sequential([Dense(n_in, n_out, activation='sigmoid')])
    decoder = containers.Sequential([Dense(n_out, n_in, activation='sigmoid')])
    ae.add(AutoEncoder(encoder=encoder, decoder=decoder,
                       output_reconstruction=False, tie_weights=True))
    ae.compile(loss='mean_squared_error', optimizer='rmsprop')
    ae.fit(X_train_tmp, X_train_tmp, batch_size=batch_size, nb_epoch=nb_epoch)
    # Store trainined weight and update training data
    encoders.append(ae.layers[0].encoder)
    X_train_tmp = ae.predict(X_train_tmp)

# Fine-turning
model = Sequential()
for encoder in encoders:
    model.add(encoder)
model.add(Dense(nb_hidden_layers[-1], nb_classes, activation='softmax'))

model.compile(loss='categorical_crossentropy', optimizer='rmsprop')
score = model.evaluate(X_test, Y_test, show_accuracy=True, verbose=0)
print('Test score before fine turning:', score[0])
print('Test accuracy after fine turning:', score[1])
model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=nb_epoch,
          show_accuracy=True, validation_data=(X_test, Y_test))
score = model.evaluate(X_test, Y_test, show_accuracy=True, verbose=0)
print('Test score after fine turning:', score[0])
print('Test accuracy after fine turning:', score[1])
```

Outputs

```
Epoch 0
60000/60000 [==============================] - 18s - loss: 0.0733
Training the layer 2: Input 600 -> Output 500
Epoch 0
60000/60000 [==============================] - 11s - loss: 0.0022
Training the layer 3: Input 500 -> Output 400
Epoch 0
60000/60000 [==============================] - 8s - loss: 0.0003
Test score before fine turning: 2.36135076835
Test accuracy after fine turning: 0.1028
Train on 60000 samples, validate on 10000 samples
Epoch 0
60000/60000 [==============================] - 18s - loss: 1.3828 - acc: 0.4964 - val_loss:
0.6930 - val_acc: 0.7838
Test score before fine turning: 0.692979552291
Test accuracy after fine turning: 0.7838
```

👍 3

darzok commented on Jul 9, 2015

I'm reading an article (thesis of LISA labs) about different method to train deep neural networks. This thesis explain that to train a network with autoencoder we should use crossentropy for eache autoencoder.

jramapuram commented on Jul 17, 2015

Cross entropy is for classification (ie you need classes). Autoencoders are purely MSE based.

xypan1232 commented on Sep 9, 2015

But when i use parameter tie_weights
ae.add(AutoEncoder(encoder=encoder, decoder=decoder, output_reconstruction=False, tie_weights=True))

it gives the error:
TypeError: **init**() got an unexpected keyword argument 'tie_weights'

jramapuram commented on Sep 9, 2015

This is because weight tying has been removed

xypan1232 commented on Sep 9, 2015

thanks, so what can we do if i want to use tie_weights?

Nidhi1211 commented on Sep 22, 2015

```
Traceback (most recent call last):
File "/home/nidhi/Documents/project/SAE.py", line 18, in
(X_train,y_train),(X_test,y_test)=mnist.load_data()
File "/usr/local/lib/python2.7/dist-packages/keras/datasets/mnist.py", line 17, in load_data
data = six.moves.cPickle.load(f)
File "/usr/lib/python2.7/gzip.py", line 455, in readline
c = self.read(readsize)
File "/usr/lib/python2.7/gzip.py", line 261, in read
self._read(readsize)
File "/usr/lib/python2.7/gzip.py", line 308, in _read
self._read_eof()
File "/usr/lib/python2.7/gzip.py", line 347, in _read_eof
hex(self.crc)))
IOError: CRC check failed 0x7603be46 != 0x4bbebed3L
```

jramapuram commented on Sep 22, 2015

@Nidhi1211 : This is unrelated. Your error is clearly in your data load....
@xypan1232 : You will have to write your own extend Layer and write your own autoencoder.

Traceback (most recent call last):
File "/home/nidhi/Documents/project/SAE.py", line 40, in
ae.add(AutoEncoder(encoder=encoder,decoder=decoder,output_reconstruction=False,tie_weights=True))
TypeError: **init**() got an unexpected keyword argument 'tie_weights'

jramapuram commented on Sep 22, 2015

@Nidhi1211 : I suggest you learn how to read stack traces.
The first stack trace is clearly not the same as the second.
The second is also mentioned above if you spend a few seconds to read the context.

vkuznet commented on Dec 30, 2015

Hi,
I'm having trouble to understand how properly configure AutoEncoder for non MNIST dataset. Let's say I have a dataset with N rows and M features and I'm trying to perform unsupervised learning to extract features from it. For that I setup simple autoencoder code following keras documentation example (http://keras.io/layers/core/#autoencoder). Here it is:

```
    # input_dim, output_dim, h1_dim below are
    # input, output and hidden layer dimensions, respectively
    encoder = containers.Sequential([Dense(output_dim=h1_dim, input_dim=input_dim)])
    decoder = containers.Sequential([Dense(output_dim=input_dim, input_dim=h1_dim)])
    model = Sequential()
    model.add(AutoEncoder(encoder=encoder, decoder=decoder,
                          output_reconstruction=True))

    # optimizer
    optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-06)

    # compile model
    model.compile(loss='mean_squared_error', optimizer=optimizer)

    print "model", pprint.pformat(model.to_json())
    model.fit(X, X, nb_epoch = epochs, batch_size = batch_size,
            validation_split=0.2,
            verbose = 2, shuffle=shuffle, show_accuracy = False)
```

```
output_reconstruction: If this is False,
the output of the autoencoder is the output of the deepest hidden layer.
Otherwise, the output of the final decoder layer is returned.
```

So I though I'll use output_reconstructions=False and then I'll be able to extract hidden layer. But I got the following error when I used this option in my model:

```
Epoch 1/50
Traceback (most recent call last):
  File "./autoenc.py", line 186, in <module>
    main()
  File "./autoenc.py", line 183, in main
    float(opts.split_size))
  File "./autoenc.py", line 151, in neuralnet
    verbose = 2, shuffle=shuffle, show_accuracy = False)
  File "build/bdist.linux-x86_64/egg/keras/models.py", line 581, in fit
  File "build/bdist.linux-x86_64/egg/keras/models.py", line 254, in _fit
  File "build/bdist.linux-x86_64/egg/keras/models.py", line 308, in _test_loop
  File "build/bdist.linux-x86_64/egg/keras/backend/theano_backend.py", line 365, in __call__
  File
"/nfs/cor/user/vk/DataAnalysis/CUDA/homesite/src/theano/theano/compile/function_module.py",
line 618, in __call__
    storage_map=self.fn.storage_map)
  File "/nfs/cor/user/vk/DataAnalysis/CUDA/homesite/src/theano/theano/gof/link.py", line 297,
in raise_with_op
    reraise(exc_type, exc_value, exc_trace)
  File
"/nfs/cor/user/vk/DataAnalysis/CUDA/homesite/src/theano/theano/compile/function_module.py",
line 607, in __call__
    outputs = self.fn()
ValueError: dimension mismatch in args to gemm (10000,301)x(301,100)->(10000,301)
Apply node that caused the error: GpuGemm{inplace}(GpuFromHost.0, TensorConstant{1.0},
GpuFromHost.0, <CudaNdarrayType(float32, matrix)>, TensorConstant{-1.0})
Toposort index: 8
Inputs types: [CudaNdarrayType(float32, matrix), TensorType(float32, scalar),
CudaNdarrayType(float32, matrix), CudaNdarrayType(float32, matrix), TensorType(float32,
scalar)]
Inputs shapes: [(10000, 301), (), (10000, 301), (301, 100), ()]
Inputs strides: [(301, 1), (), (301, 1), (100, 1), ()]
Inputs values: ['not shown', array(1.0, dtype=float32), 'not shown', 'not shown', array(-1.0,
dtype=float32)]
Outputs clients: [[GpuElemwise{Add}[(0, 1)](GpuDimShuffle{x,0}.0, GpuGemm{inplace}.0)]]

HINT: Re-running with most Theano optimization disabled could give you a back-trace of when
this node was created. This can be done with by setting the Theano flag
'optimizer=fast_compile'. If that does not work, Theano optimizations can be disabled with
'optimizer=None'.
HINT: Use the Theano flag 'exception_verbosity=high' for a debugprint and storage map
footprint of this apply node.
```

Please note, that my data X is a dataset without labels, I used 10000 as a batch size and my dataset has 301 features. I used hidden layer with 100 neurons and run keras version 0.3.0 on GPU.

- even though this ticket and most examples use standard dataset like MNIST, I don't see any difference between MNIST and any other dataset, therefore I presume the code should work out of the box. Am I wrong on this statement, if so can someone explain the reason.
- why using output_reconstructions=False gives dimension mismatch?
- if I'll use activation='tanh' I got slightly different error: `ValueError: GpuElemwise. Input dimension mis-match. Input 2 (indices start at 0) has shape[1] == 301, but the output's size on that axis is 100.` This is particular strange since it says that my output size is a size of my hidden layer, but this is what docs tell would happen.

I would appreciate any suggestions and explanations even using some dummy example.
Thanks,
Valentin.

antoniosehk commented on Dec 31, 2015

> why using output_reconstruction=True flags works and False value does not?
> It is because you ask the "fit" function to do validation as well. The following would work even for output_reconstruction - False

model.fit(X, X, nb_epoch = epochs, batch_size = batch_size,
validation_data=None,
verbose = 2, shuffle=shuffle, show_accuracy = False)

> even though this ticket and most examples use standard dataset like MNIST, I don't see any difference between MNIST and any other dataset, therefore I presume the code should work out of the box. Am I wrong on this statement, if so can someone explain the reason.

No difference between MNIST and any other dataset.

> why using output_reconstructions=False gives dimension mismatch
> It is mainly because of the "fit" function
>
> if I'll use activation='tanh' I got slightly different error: ValueError: GpuElemwise. Input dimension mis-match. Input 2 (indices start at 0) has shape[1] == 301, but the output's size on that axis is 100. This is particular strange since it says that my output size is a size of my hidden layer, but this is what docs tell would happen.

Cannot understand why. Any more detailed explanation?

vkuznet commented on Dec 31, 2015

Thanks for your help.
Valentin.

On 0, Tenkawa Akito notifications@github.com wrote:

> > why using output_reconstruction=True flags works and False value does not?
> > It is because you ask the "fit" function to do validation as well. The following would work even for output_reconstruction - False
>
> model.fit(X, X, nb_epoch = epochs, batch_size = batch_size,
> validation_data=None,
> verbose = 2, shuffle=shuffle, show_accuracy = False)
>
> > even though this ticket and most examples use standard dataset like MNIST, I don't see any difference between MNIST and any other dataset, therefore I presume the code should work out of the box. Am I wrong on this statement, if so can someone explain the reason.
>
> No difference between MNIST and any other dataset.
>
> > why using output_reconstructions=False gives dimension mismatch
> > It is mainly because of the "fit" function
> >
> > if I'll use activation='tanh' I got slightly different error: ValueError: GpuElemwise. Input dimension mis-match. Input 2 (indices start at 0) has shape[1] == 301, but the output's size on that axis is 100. This is particular strange since it says that my output size is a size of my hidden layer, but this is what docs tell would happen.
>
> Cannot understand why. Any more detailed explanation?
>
> Reply to this email directly or view it on GitHub:
> #358 (comment)

dancivitarese commented on Mar 18, 2016

Hi all,
I'm trying to stack some auto encoders, but without success. Here is my code:

```
# from https://github.com/fchollet/keras/issues/358

from __future__ import absolute_import
from __future__ import print_function

import numpy as np
from keras import models
from keras.layers.core import Dense, AutoEncoder

np.random.seed(1337)
batch_size = 100
nb_classes = 10
nb_epoch = 1

(X_train, _), (X_test, _) = mnist.load_data()
X_train = X_train.reshape(-1, 784)
X_test = X_test.reshape(-1, 784)
X_train = X_train.astype("float32") / 255.0
X_test = X_test.astype("float32") / 255.0
print(X_train.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')

# Autoencoder 1
ae1 = models.Sequential()
ae1.add(AutoEncoder(encoder=containers.Sequential([Dense(500, input_dim=784)]),
                    decoder=containers.Sequential([Dense(784, input_dim=500)]),
                    output_reconstruction=True))
ae1.compile(loss='mse', optimizer=optimizers.RMSprop())
ae1.fit(X_train, X_train, batch_size=batch_size, nb_epoch=nb_epoch,
        show_accuracy=True, verbose=1, validation_data=[X_test, X_test])
X_train_tmp = ae1.predict(X_train)
print("Autoencoder data format: {0} - should be (60000, 500)".format(X_train_tmp.shape))

# Autoencoder 2
ae2 = models.Sequential()
ae2.add(AutoEncoder(encoder=containers.Sequential([Dense(400, input_dim=500)]),
                    decoder=containers.Sequential([Dense(500, input_dim=400)]),
                    output_reconstruction=True))
ae2.compile(loss='mse', optimizer=optimizers.RMSprop())
ae2.fit(X_train_tmp, X_train_tmp, batch_size=batch_size, nb_epoch=nb_epoch)
X_train_tmp = ae1.predict(X_train_tmp)
```

The output is:

```
Using Theano backend.
60000 train samples
10000 test samples
Train on 60000 samples, validate on 10000 samples
Epoch 1/1
60000/60000 [==============================] - 6s - loss: 0.0442 - acc: 0.0139 - val_loss:
0.0245 - val_acc: 0.0127
Autoencoder data format: (60000, 784) - should be (60000, 500)
Epoch 1/1
Traceback (most recent call last):
  File "/Users/sallesd/PycharmProjects/keras_tutorials/autoencoder.py", line 43, in <module>
    ae2.fit(X_train_tmp, X_train_tmp, batch_size=batch_size, nb_epoch=nb_epoch)
  File "/Users/sallesd/anaconda/envs/keras/lib/python2.7/site-packages/keras/models.py", line
646, in fit
    shuffle=shuffle, metrics=metrics)
  File "/Users/sallesd/anaconda/envs/keras/lib/python2.7/site-packages/keras/models.py", line
280, in _fit
    outs = f(ins_batch)
  File "/Users/sallesd/anaconda/envs/keras/lib/python2.7/site-
```

```
    storage_map=self.fn.storage_map)
  File "/Users/sallesd/anaconda/envs/keras/lib/python2.7/site-
packages/theano/compile/function_module.py", line 595, in __call__
    outputs = self.fn()
ValueError: Shape mismatch: x has 784 cols (and 100 rows) but y has 500 rows (and 400 cols)
Apply node that caused the error: Dot22(<TensorType(float32, matrix)>, <TensorType(float32,
matrix)>)
Inputs types: [TensorType(float32, matrix), TensorType(float32, matrix)]
Inputs shapes: [(100, 784), (500, 400)]
Inputs strides: [(3136, 4), (1600, 4)]
Inputs values: ['not shown', 'not shown']

HINT: Re-running with most Theano optimization disabled could give you a back-trace of when
this node was created. This can be done with by setting the Theano flag
'optimizer=fast_compile'. If that does not work, Theano optimizations can be disabled with
'optimizer=None'.
HINT: Use the Theano flag 'exception_verbosity=high' for a debugprint and storage map
footprint of this apply node.
```

Can someone help me?

Thanks,
Daniel

dibenedetto commented on Mar 21, 2016

@dchevitarese you are trying to fit your second autoencoder with an input with size 784, while it expects one of 500.

If I get it right, you want to sneak on the innermost layer, so take care of what data are you dealing with. here is some hint:

```
# ...

# Autoencoder 1
realAE = AutoEncoder(encoder=containers.Sequential([Dense(500, input_dim=784)]),
                    decoder=containers.Sequential([Dense(784, input_dim=500)]),
                    output_reconstruction=True))

ae1 = models.Sequential([realAE])
ae1.compile(loss='mse', optimizer=optimizers.RMSprop())
ae1.fit(X_train, X_train, batch_size=batch_size, nb_epoch=nb_epoch,
        show_accuracy=True, verbose=1, validation_data=[X_test, X_test])

realAE.output_reconstruction = False                     # grab inner representation ...
ae1.compile(loss='mse', optimizer=optimizers.RMSprop())  # ... but you have to recompile,
don't re-train, weights are ketp (someone can confirm?)
X_train_tmp = ae1.predict(X_train)                       # X_train_tmp is now sized at 500
print("Autoencoder data format: {0} - should be (60000, 500)".format(X_train_tmp.shape))
```

I hope this helps.

👍 2

dancivitarese commented on Mar 21, 2016

Hi **@dibenedetto**, I didn't know that I would have to recompile, but it did the trick. In the end, I got ~91% of accuracy. I could use a CNN to do the same job, but I am investigating this AE's to pre-train layers - and this also explains my next question: What do you mean with "take care of what data are you dealing with"?
Thank you

P.S.: I am trying to recreate this: http://www.sciencedirect.com/science/article/pii/S0031320315001181

isalirezag commented on Apr 4, 2016

@mthrok Thanks for your help and your code!
would you please explain, why you have `encoder = containers.Sequential([Dense(784, 700), Dense(700, 600)])` and in Keras page they have `encoder = containers.Sequential([Dense(16, input_dim=32), Dense(8)])` .I mean what is the difference, and also whenever I use `encoder = containers.Sequential([Dense(784, 700), Dense(700, 600)])` I receive an error, idk why...
Does anyone have any sample code to visualize the layers and output please?
Thanks in advance!

dancivitarese commented on Apr 4, 2016

Hi **@isalirezag**, you can get all configuration by using `model.get_config()` that will give you something like this:

```
'input_shape': (860,), 'name': 'Dense', 'output_dim': 784, 'trainable': True}], 'name':
'Sequential'}, 'encoder_config': {'layers': [{'W_constraint': None, 'W_regularizer': None,
'activation': 'sigmoid', 'activity_regularizer': None, 'b_constraint': None, 'b_regularizer': None,
'cache_enabled': True, 'custom_name': 'dense', 'init': 'glorot_uniform', 'input_dim': None,
'input_shape': (784,), 'name': 'Dense', 'output_dim': 860, 'trainable': True}], 'name':
'Sequential'}, 'name': 'AutoEncoder', 'output_reconstruction': True}], 'loss':
'binary_crossentropy', 'name': 'Sequential', 'optimizer': {'epsilon': 1e-06, 'lr':
0.0010000000474974513, 'name': 'RMSprop', 'rho': 0.8999999761581421}, 'sample_weight_mode': None}
```

Unfortunately, I don't think keras has a good visualization functionality. The only thing you get is a very simple graphviz plot, which is not helpful.

muyinanhai commented on Apr 16, 2016

It seems the code can only work on keras==0.3.0

soloice commented on Jun 9, 2016 • edited

I just wanna know if the `AutoEncoder` has been removed from the newest Keras?
I have no idea why I cannot import `AutoEncoder` and containers... (Even if I reinstalled theano and Keras)

AdityaGudimella commented on Jun 9, 2016

It has been removed. You can easily accomplish it using the functional api.
Check it the blog for an example.
On Jun 9, 2016 2:56 AM, "lurker" notifications@github.com wrote:

> I just wanna know if the AutoEncoder has been removed from the newest Keras?
> I have no idea why I cannot import AutoEncoder and containers...

> #358 (comment), or mute
> the thread
> https://github.com/notifications/unsubscribe/AFHcNR8-Avd6cXVOPkKFAm4-EXoE5FQUks5qJ7kjgaJpZM4FT7x6
> .

zgbkdlm commented on Feb 23, 2017 • edited

Hey, guys, I am also working on how to layer-by-layer train AE, and I'm new to Keras.

Will such code work ?

```
from keras.layers import Input, Dense
from keras.models import Model
import numpy as np
import matplotlib.pyplot as plt
encoding_dim = 64

input_img = Input(shape=(867,))
encoded_l1 = Dense(512, activation='relu')(input_img)
encoded_l2 = Dense(256, activation='relu')(encoded_l1)
encoded_l3 = Dense(128, activation='relu')(encoded_l2)
encoded = Dense(64, activation='relu')(encoded_l3)

decoded = Dense(64, activation='relu')(encoded)
decoded_l1 = Dense(128, activation='relu')(decoded)
decoded_l2 = Dense(256, activation='relu')(decoded_l1)
decoded_l3 = Dense(512, activation='relu')(decoded_l2)
decoded_l4 = Dense(867, activation='sigmoid')(decoded_l3)


# Layerwise Trainning
encoder_l1 = Model(input=input_img, output=Dense(867, activation='sigmoid')(encoded_l1))
encoder_l1.compile(optimizer='Adadelta', loss='binary_crossentropy')
encoder_l1.fit(x_train, x_train, nb_epoch=50, batch_size=256, shuffle=True, validation_data=
(x_test, x_test))

encoder_l2 = Model(input=input_img, output=Dense(867, activation='sigmoid')(encoded_l2))
encoder_l2.compile(optimizer='Adadelta', loss='binary_crossentropy')
encoder_l2.fit(x_train, x_train, nb_epoch=50, batch_size=256, shuffle=True, validation_data=
(x_test, x_test))

encoder_l3 = Model(input=input_img, output=Dense(867, activation='sigmoid')(encoded_l3))
encoder_l3.compile(optimizer='Adadelta', loss='binary_crossentropy')
encoder_l3.fit(x_train, x_train, nb_epoch=50, batch_size=256, shuffle=True, validation_data=
(x_test, x_test))

encoder_l4 = Model(input=input_img, output=Dense(867, activation='sigmoid')(encoded))
encoder_l4.compile(optimizer='Adadelta', loss='binary_crossentropy')
encoder_l4.fit(x_train, x_train, nb_epoch=50, batch_size=256, shuffle=True, validation_data=
encoder = Model(input=input_img, output=encoded)

autoencoder.compile(optimizer='Adadelta', loss='binary_crossentropy')

autoencoder.fit(x_train, x_train,
                nb_epoch=100,
                batch_size=256,
                shuffle=True,
                validation_data=(x_test, x_test))
```

stale bot added the stale label on May 24, 2017

stale bot commented on May 24, 2017

This issue has been automatically marked as stale because it has not had recent activity. It will be closed after 30 days if no further activity occurs, but feel free to re-open a closed issue if needed.

stale bot closed this on Jun 24, 2017

vade mentioned this issue on Sep 7, 2019

Error when load combined mobile-net model #10428

Closed

Assignees

No one assigned

Labels

stale

Projects

None yet

Linked pull requests

Successfully merging a pull request may close this issue.

None yet

14 participants