Romeo Kienzler merge from romeokienzler repo 53f3fa0 on Jun 21

0 contributors

1 lines (1 sloc) 530 KB

In [5]:
```
%matplotlib inline
```

Introduction to PyTorch

PyTorch's tensor library

The most of PyTorch operations are running on **tensors**. A tensor is an multidimensional array. Lets have a look on some basic tensor operations. But first, lets import some important PyTorch libraries:

- **torch** - a Tensor library similar to NumPy, with strong GPU support
- **torch.autograd** - a "tape-based" (about this - later on) automatic differentiation library
- **torch.nn** - a neural networks library deeply integrated with autograd
- **torch.optim** - an optimization package to be used with torch.nn with standard optimization methods such as SGD, RMSProp, LBFGS, Adam etc.

We also set a seed to be able to reproduce the same results later.

In [44]:
```
import torch
import torch.autograd as autograd
import torch.nn as nn
import torch.optim as optim

torch.manual_seed(123)
```

Out[44]:
```
<torch._C.Generator at 0x7fe41fd172f0>
```

Creating Tensors

Tensors can be created from Python lists with the **torch.Tensor()** function.

In [47]:
```
# Create a torch.Tensor object from python list
v = [1, 2, 3]
print(type(v))
v_tensor = torch.Tensor(v)
print(v_tensor)

# Create a torch.Tensor object of size 2x3 from 2x3 matrix
m2x3 = [[1, 2, 3], [4, 5, 6]]
m2x3_tensor = torch.Tensor(m2x3)
print(m2x3_tensor)

# Create a 3D torch.Tensor object of size 3x3x3.
m3x3x3 = [[[1, 2, 3], [4, 5, 6], [7, 8, 9]],
          [[10, 11, 12],[13, 14, 15], [16, 17, 18]],
            [[19, 20, 21],[22, 23, 24], [25, 26, 27]]]
m3x3x3_tensor = torch.Tensor(m3x3x3)
print(m3x3x3_tensor)

#Create a 4Dtensor from random data and given dimensions (in this case 3x4x5x6) with torch.ra
ndn()
m4x3x3x3_tensor = torch.randn((4, 3, 3, 3))
m4x3x3x3_tensor.shape
print(m4x3x3x3_tensor)
```

```
<type 'list'>

 1
 2
 3
[torch.FloatTensor of size 3]


 1  2  3
 4  5  6
[torch.FloatTensor of size 2x3]


(0 ,.,.) =
   1   2   3
   4   5   6
   7   8   9

(1 ,.,.) =
  10  11  12
  13  14  15
  16  17  18

(2 ,.,.) =
  19  20  21
  22  23  24
```

```
  22  23  24
  25  26  27
[torch.FloatTensor of size 3x3x3]


(0 ,0 ,.,.) = 
  0.4934 -0.2766  0.2439
 -1.2116 -0.1520  0.1509
 -0.6251 -0.4416  0.3208

(0 ,1 ,.,.) = 
 -0.3273 -0.5305 -0.0172
  0.4719  0.5671  2.7930
  0.3229  0.8552  0.7492

(0 ,2 ,.,.) = 
 -1.7119  0.6025 -0.7018
 -1.3130  0.1574  2.0114
  0.1004  0.8222 -0.0176

(1 ,0 ,.,.) = 
  1.2481 -0.0710  2.1627
  1.5215 -1.0547  1.7822
  1.9736 -0.3101 -0.8211

(1 ,1 ,.,.) = 
  0.1315 -0.6948 -0.5823
  1.0035 -1.4613  0.8985
  0.6210 -0.9679  0.6740

(1 ,2 ,.,.) = 
 -1.2828 -0.5097  0.1464
 -0.4860 -0.7529  1.6989
  0.4991 -2.1702  0.5130

(2 ,0 ,.,.) = 
 -1.9029  0.8260 -0.6644
  1.6663 -0.5704  1.3906
 -1.4855  0.2987 -0.3029

(2 ,1 ,.,.) = 
  0.3354  1.0599  0.1941
 -0.9295  0.5329 -1.1307
  0.1024  2.5200 -1.2324

(2 ,2 ,.,.) = 
 -0.8294  0.4342 -0.7374
  0.1591 -1.3560  0.5513
  0.3732  1.4246 -0.6860

(3 ,0 ,.,.) = 
 -0.4791  2.2610  1.3191
  1.0130  1.6650  0.8469
  0.2564  0.8049 -0.3107

(3 ,1 ,.,.) = 
 -0.9208 -0.1507 -0.6977
  1.5227  2.4308  1.5695
 -0.0657 -0.4277  0.2537

(3 ,2 ,.,.) = 
  0.7337  1.1388  0.5516
 -0.6187 -0.6503  0.3017
  0.6113  0.4344 -0.3423
[torch.FloatTensor of size 4x3x3x3]
```

What is a multidimensional tensor?

Since we frequently deal with n > 3 dimensional tensors, its understanding is very important. The best way to think of a higher (n) dimensional object (and tensor in particular) is as of a container which keeps a series of n-1 dimensional objects "inside" of it. We can "pull out" these "inner" objects by indexing in to higher dimensional tensor container. Let's have a look on some examples:

- For a vector v (dim(v)=1), indexing into it ("pulling out of it") returns its "slice" - a scalar s (dim(s)=0).
- For a matrix, indexing into it returns its "slice" - a (row or column) vector.
- 3D tensor can be seen as a cube or 3D rectangular consisting of horizontally "stacked" matrices. So if we index into a such tensor it will give us its slice which is a matrix!
- We can't easily visualize 5D (or n-D) tensors, but the idea is actually the same. If we index in to them, we will pull out an object of dimension n-1.
- E.g. a 4D tensor can be seen as a list of cubes or 3D reactangulars. If we index in to a 4D tensor, we will get 3D rectangulars.

IMG_0120.png

In [46]:
```python
# Index into v_tensor and get a scalar
print(v_tensor[0])

# Index into m2x3_tensor and get a vector
print(m2x3_tensor[0])

# Index into m3x3x3_tensor and get a matrix
print(m3x3x3_tensor[0])

# Index into m4x3x3x3_tensor and get a 3D rectangular of size 4x5x6
print(m4x3x3x3_tensor[0])
```

```
1.0

 1
 2
 3
[torch.FloatTensor of size 3]


 1  2  3
 4  5  6
 7  8  9
[torch.FloatTensor of size 3x3]


(0 ,.,.) =
 -0.1115  0.1204 -0.3696
 -0.2404 -1.1969  0.2093
 -0.9724 -0.7550  0.3239

(1 ,.,.) =
 -0.1085  0.2103 -0.3908
  0.2350  0.6653  0.3528
  0.9728 -0.0386 -0.8861

(2 ,.,.) =
 -0.4709 -0.4269 -0.0283
  1.4220 -0.3886 -0.8903
 -0.9601 -0.4087  1.0764
[torch.FloatTensor of size 3x3x3]
```

Operations with Tensors

You can operate on tensors in the ways you would expect. See the documentation http://pytorch.org/docs/torch.html (http://pytorch.org/docs/torch.html) for a complete list of operations.

Simple mathematical operations: **Addition, Multiplication**

In [76]:
```python
x = torch.Tensor([1, 2, 3])
y = torch.Tensor([4, 5, 6])
print(x)
print(y)

w = torch.matmul(x, y)
print(w)
```

```
 1
 2
 3
[torch.FloatTensor of size 3]


 4
 5
 6
[torch.FloatTensor of size 3]

32.0
```

Helpful operation: **Concatenation**

In [78]:
```python
# By default, it concatenates along the axis with 0 (rows). It's "stacking" the rows.

x_1 = torch.randn(2, 5)
print(x_1)
y_1 = torch.randn(3, 5)
print(y_1)
z_1 = torch.cat([x_1, y_1])
print(z_1)

# Concatenate columns:
x_2 = torch.randn(2, 3)
```

```python
x_2 = torch.randn(2, 5)
print(x_2)
y_2 = torch.randn(3, 5)
print(y_2)
# second arg specifies which axis to concat along. Here we select 1 (columns). It's attaching
 the columns.
z_2 = torch.cat([x_2, y_2], 1)
print(z_2)

# If your tensors are not compatible, torch will complain.  Uncomment to see the error
torch.cat([x_1, x_2])
```

```
 0.5374 -0.0899 -0.7969 -0.7968 -0.2611
 0.4808 -0.1458 -0.5357 -0.7330  2.3465
[torch.FloatTensor of size 2x5]


-0.4654  1.4965  2.7607 -1.4606  1.0825
-0.9363 -0.2594 -1.3465  0.5388  0.0382
-0.1434  1.4049  0.0889  0.5171 -0.0214
[torch.FloatTensor of size 3x5]


 0.5374 -0.0899 -0.7969 -0.7968 -0.2611
 0.4808 -0.1458 -0.5357 -0.7330  2.3465
-0.4654  1.4965  2.7607 -1.4606  1.0825
-0.9363 -0.2594 -1.3465  0.5388  0.0382
-0.1434  1.4049  0.0889  0.5171 -0.0214
[torch.FloatTensor of size 5x5]


-0.2001 -0.6503 -0.2019
 0.1262 -0.0708  1.4592
[torch.FloatTensor of size 2x3]


 0.5823  0.0048 -0.2069 -0.5939 -1.3076
 0.9156  0.3748 -1.3108 -2.6207 -1.2696
 0.9036 -1.3043  0.3300 -0.6928 -1.5125
[torch.FloatTensor of size 3x5]


RuntimeErrorTraceback (most recent call last)
<ipython-input-78-c85240a5062e> in <module>()
     14 print(y_2)
     15 # second arg specifies which axis to concat along. Here we select 1 (columns). It's at
taching the columns.
---> 16 z_2 = torch.cat([x_2, y_2], 1)
     17 print(z_2)
     18

RuntimeError: inconsistent tensor sizes at /pytorch/torch/lib/TH/generic/THTensorMath.c:2864
```

Reshaping Tensors

We can use the `.view()` method to reshape a tensor. Often we will need to reshape our data before passing it to a neuronal network.

Let's assume we have 64000 RGB images with the size of 28x28 pixels. We can define an array fo shape (64000, 3, 28, 28) to hold them, where 3 is number of color channels:

In [97]:
```python
x = torch.randn(64000, 3, 28, 28)
# Now we want to add a batch dimension of size 32. We can then infer the second dimension by
 placing -1:
x_rehsaped = x.view(32, -1, 3, 28, 28)
print(x_rehsaped.shape)
```

```
torch.Size([32, 2000, 3, 28, 28])
```

Computation Graphs and Automatic Differentiation

A computation graph is a specification of what parameters with which operations are involved in the computation to give the output.

The fundamental class of Pytorch `autograd.Variable` keeps track of how it was created.

In [100]:
```python
# Variables wrap tensor objects
x = autograd.Variable(torch.Tensor([1, 2, 3]), requires_grad=True)
# You can access the data with the .data attribute
print(x.data)

y = autograd.Variable(torch.Tensor([4, 5, 6]), requires_grad=True)

# With autograd.Variable you can also perform all the same operations you did with tensors
z = x + y
```

```
z = x + y
print(z.data)

#  w knows also that it's result of addition of z lements (AddBackward)
operation = z.grad_fn
print(operation)
```

```
 1
 2
 3
[torch.FloatTensor of size 3]


 5
 7
 9
[torch.FloatTensor of size 3]

<AddBackward1 object at 0x7fe41361ffd0>
```

The autograd.Variable knows which operation has created it. But how does that help **compute a gradient**?

In [101]:
```
# Lets sum up all the entries in z
s = z.sum()
print(s)
print(s.grad_fn)
```

```
Variable containing:
 21
[torch.FloatTensor of size 1]

<SumBackward0 object at 0x7fe413507cd0>
```

Gradient

So now, what is the derivative of this sum with respect to the first component of x? Remember, that x is a tensor of 3 elements: $x = (x_0, x_1, x_2)$

In math, we want a partial derivative of s with respect to x_0: $\frac{\partial s}{\partial x_0}$

Well, s knows that it was created as a sum of the tensor z elements (z_0, z_1, z_2). z knows that it was the sum $x + y$. So

$$s = x_0 \mp y_0{}^{\$z\_0\$} + x_1 \mp y_1{}^{\$z\_1\$} + x_2 \mp y_2{}^{\$z\_2\$} \quad (1)$$

And so s contains enough information to determine that the derivative of s with respect to x_0 is 1!

Reminder: If you compute the partial derivative with respekt to one variable, you handle all other variables as constants. Therefore they all $(x_1, x_2, y_0, y_1, y_2)$ get zeroes, and the derivative of $f(x_0) = x_0$ is 1.

First we need to run **backpropagation** and calculate gradients with respect to every variable. *Note:* if you run `backward` multiple times, the gradient will increment. That is because Pytorch *accumulates* the gradient into the **.grad property**, since for many models this is very convenient. Lets now have Pytorch compute the gradient, and see that we were right with our guess of 1:

In [106]:
```
# calling .backward() on any variable will run backprop, starting from it.
s.backward(retain_graph=True)
```

In [107]:
```
print(x)
print(x.grad)
print(y.grad)
```

```
Variable containing:
 1
 2
 3
[torch.FloatTensor of size 3]

Variable containing:
 3
 3
 3
[torch.FloatTensor of size 3]

Variable containing:
 3
 3
 3
[torch.FloatTensor of size 3]
```

How NOT to break the computational graph

Let's create two torch tensors and add them up:

In [109]:
```
x = torch.randn((2, 2))
y = torch.randn((2, 2))
z = x + y  # These are Tensor types, and backprop would not be possible

print(z)
```

```
 0.1730  1.8913
 0.1251  0.1286
[torch.FloatTensor of size 2x2]
```

Now we wrap the torch tensors in `autograd.Variable`. The `var_z` contains the information for backpropagation:

In [111]:
```
var_x = autograd.Variable(x, requires_grad=True)
var_y = autograd.Variable(y, requires_grad=True)
# var_z contains enough information to compute gradients, as we saw above
var_z = var_x + var_y
print(var_z.grad_fn)
```

```
<AddBackward1 object at 0x7fe41352c110>
```

But what happens if we extract the wrapped tensor object out of `var_z` and re-wrap the tensor in a new `autograd.Variable`?

In [112]:
```
var_z_data = var_z.data
new_var_z = autograd.Variable(var_z_data)
print(new_var_z.grad_fn)
```

```
None
```

The variable chain is not existing anymore, since we have extracted only data and the whole operations chain was lost. If we try now to compute `backward` on `new_var_z`, it will throw an error:

In [113]:
```
new_var_z.backward(retain_graph=True)
```

```

RuntimeErrorTraceback (most recent call last)
<ipython-input-113-9385e2013798> in <module>()
----> 1 new_var_z.backward(retain_graph=True)

/gpfs/fs01/user/s5b4-5d8779714d1900-f211816cda7e/.local/lib/python2.7/site-packages/torch/auto
grad/variable.pyc in backward(self, gradient, retain_graph, create_graph, retain_variables)
    165                 Variable.
    166         """
--> 167         torch.autograd.backward(self, gradient, retain_graph, create_graph, retain_var
iables)
    168 
    169     def register_hook(self, hook):

/gpfs/fs01/user/s5b4-5d8779714d1900-f211816cda7e/.local/lib/python2.7/site-packages/torch/auto
grad/__init__.pyc in backward(variables, grad_variables, retain_graph, create_graph, retain_va
riables)
     97 
     98     Variable._execution_engine.run_backward(
---> 99         variables, grad_variables, retain_graph)
    100 
    101 

RuntimeError: element 0 of variables does not require grad and does not have a grad_fn
```

CUDA

Check wether GPU accelaration with **CUDA** is available

In [42]:
```
# let us run this cell only if CUDA is available
if torch.cuda.is_available():
    # creates a LongTensor and transfers it
    # to GPU as torch.cuda.LongTensor
    a = torch.LongTensor(10).fill_(3).cuda()
    print(type(a))
    b = a.cpu()
    # transfers it to CPU, back to
    # being a torch.LongTensor
```

Linear Model

In [1]:
```
import torch
import torch.nn as nn
```

```
from torch.autograd import Variable
import numpy as np
```

In [2]:
```
x = [i for i in range(20)] #list comprehention
x_train = np.array(x, dtype=np.float32)
x_train = x_train.reshape(-1, 1)
print(x)
print(x_train.shape)

y = [(5*i + 2) for i in x] #list comprehention
y_train = np.array(y, dtype=np.float32)
y_train = y_train.reshape(-1, 1)
print(y)
print(y_train.shape)
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
(20, 1)
[2, 7, 12, 17, 22, 27, 32, 37, 42, 47, 52, 57, 62, 67, 72, 77, 82, 87, 92, 97]
(20, 1)
```

Create Model Class

In [6]:
```
class LinearRegressor(nn.Module):
    def __init__(self, input_dim, output_dim):
        super(LinearRegressor, self).__init__()
        self.linear = nn.Linear(input_dim, output_dim)

    def forward(self, x):
        out = self.linear(x)
        return out

input_dim = 1
output_dim = 1

model = LinearRegressor(input_dim, output_dim)

model
```

Out[6]:
```
LinearRegressor(
  (linear): Linear(in_features=1, out_features=1)
)
```

Loss & Optimizer

In [7]:
```
loss_function = nn.MSELoss()


optimizer = torch.optim.SGD(model.parameters(), lr=0.001)
optimizer
loss_function
```

Out[7]:
```
MSELoss(
)
```

In [8]:
```
epochs = 500

for epoch in range(epochs):
    epoch += 1
    #Convert inputs and outputs to torch variable
    inputs = Variable(torch.from_numpy(x_train))

    real_outputs = Variable(torch.from_numpy(y_train))

    # Reset Gradients
    optimizer.zero_grad()

    # Forward - compute the output
    pred_outputs = model(inputs)

    # Loss
    loss = loss_function(pred_outputs, real_outputs)

    # Backword - compute gradients
    loss.backward()

    # Update parameters
    optimizer.step()

    print('epoch {}, loss {}'.format(epoch, loss.data[0]))
```

```
epoch 1, loss 3330.96435547
epoch 2, loss 1881.8626709
epoch 3, loss 1063.4017334
epoch 4, loss 601.130065918
```

```
epoch 5, loss 340.03616333
epoch 6, loss 192.568328857
epoch 7, loss 109.277175903
epoch 8, loss 62.2333488464
epoch 9, loss 35.6621856689
epoch 10, loss 20.6541633606
epoch 11, loss 12.1769609451
epoch 12, loss 7.38844966888
epoch 13, loss 4.68333339691
epoch 14, loss 3.15493202209
epoch 15, loss 2.29113745689
epoch 16, loss 1.80271029472
epoch 17, loss 1.52630400658
epoch 18, loss 1.36964297295
epoch 19, loss 1.28061759472
epoch 20, loss 1.22979319096
epoch 21, loss 1.20054602623
epoch 22, loss 1.18348562717
epoch 23, loss 1.17330884933
epoch 24, loss 1.16701996326
epoch 25, loss 1.16292822361
epoch 26, loss 1.16007828712
epoch 27, loss 1.15793061256
epoch 28, loss 1.15617823601
epoch 29, loss 1.15465211868
epoch 30, loss 1.15325260162
epoch 31, loss 1.15192627907
epoch 32, loss 1.15064108372
epoch 33, loss 1.14937984943
epoch 34, loss 1.14813303947
epoch 35, loss 1.14689469337
epoch 36, loss 1.14566218853
epoch 37, loss 1.14443290234
epoch 38, loss 1.14320623875
epoch 39, loss 1.14198327065
epoch 40, loss 1.14075922966
epoch 41, loss 1.13953721523
epoch 42, loss 1.13831853867
epoch 43, loss 1.13709974289
epoch 44, loss 1.13588225842
epoch 45, loss 1.13466560841
epoch 46, loss 1.13345181942
epoch 47, loss 1.13223946095
epoch 48, loss 1.13102686405
epoch 49, loss 1.12981712818
epoch 50, loss 1.12860763073
epoch 51, loss 1.12740015984
epoch 52, loss 1.12619328499
epoch 53, loss 1.12498831749
epoch 54, loss 1.12378358841
epoch 55, loss 1.12258088589
epoch 56, loss 1.12137913704
epoch 57, loss 1.12017941475
epoch 58, loss 1.11898028851
epoch 59, loss 1.11778283119
epoch 60, loss 1.11658644676
epoch 61, loss 1.11539149284
epoch 62, loss 1.11419713497
epoch 63, loss 1.11300492287
epoch 64, loss 1.11181366444
epoch 65, loss 1.11062431335
epoch 66, loss 1.10943520069
epoch 67, loss 1.10824799538
epoch 68, loss 1.10706198215
epoch 69, loss 1.10587668419
epoch 70, loss 1.10469269753
epoch 71, loss 1.10351061821
epoch 72, loss 1.10232949257
epoch 73, loss 1.10114979744
epoch 74, loss 1.09997189045
epoch 75, loss 1.09879422188
epoch 76, loss 1.09761846066
epoch 77, loss 1.09644293785
epoch 78, loss 1.09526956081
epoch 79, loss 1.09409844875
epoch 80, loss 1.09292626381
epoch 81, loss 1.09175646305
epoch 82, loss 1.09058868885
epoch 83, loss 1.0894215107
epoch 84, loss 1.08825469017
epoch 85, loss 1.08709037304
epoch 86, loss 1.085927248
epoch 87, loss 1.0847645998
epoch 88, loss 1.08360350132
epoch 89, loss 1.08244431019
epoch 90, loss 1.08128559589
epoch 91, loss 1.08012807369
```

```
epoch 92, loss 1.07897210121
epoch 93, loss 1.07781684399
epoch 94, loss 1.07666349411
epoch 95, loss 1.07551205158
epoch 96, loss 1.07435965538
epoch 97, loss 1.0732101202
epoch 98, loss 1.07206165791
epoch 99, loss 1.07091379166
epoch 100, loss 1.06976830959
epoch 101, loss 1.06862294674
epoch 102, loss 1.06747913361
epoch 103, loss 1.06633603573
epoch 104, loss 1.0651961565
epoch 105, loss 1.06405568123
epoch 106, loss 1.06291675568
epoch 107, loss 1.06177866459
epoch 108, loss 1.06064271927
epoch 109, loss 1.05950784683
epoch 110, loss 1.05837345123
epoch 111, loss 1.05724036694
epoch 112, loss 1.05610907078
epoch 113, loss 1.0549788475
epoch 114, loss 1.05385005474
epoch 115, loss 1.05272185802
epoch 116, loss 1.05159509182
epoch 117, loss 1.0504693985
epoch 118, loss 1.04934501648
epoch 119, loss 1.04822170734
epoch 120, loss 1.04710018635
epoch 121, loss 1.04597961903
epoch 122, loss 1.04486060143
epoch 123, loss 1.04374194145
epoch 124, loss 1.04262447357
epoch 125, loss 1.04150927067
epoch 126, loss 1.04039406776
epoch 127, loss 1.03928053379
epoch 128, loss 1.03816819191
epoch 129, loss 1.03705775738
epoch 130, loss 1.03594756126
epoch 131, loss 1.03483831882
epoch 132, loss 1.03373169899
epoch 133, loss 1.03262424469
epoch 134, loss 1.03151917458
epoch 135, loss 1.03041529655
epoch 136, loss 1.02931237221
epoch 137, loss 1.02821099758
epoch 138, loss 1.02711033821
epoch 139, loss 1.02601122856
epoch 140, loss 1.02491283417
epoch 141, loss 1.02381563187
epoch 142, loss 1.02271962166
epoch 143, loss 1.0216255188
epoch 144, loss 1.02053201199
epoch 145, loss 1.01943993568
epoch 146, loss 1.01834869385
epoch 147, loss 1.01725888252
epoch 148, loss 1.01616990566
epoch 149, loss 1.0150822401
epoch 150, loss 1.01399600506
epoch 151, loss 1.01291131973
epoch 152, loss 1.01182675362
epoch 153, loss 1.01074433327
epoch 154, loss 1.00966215134
epoch 155, loss 1.00858151913
epoch 156, loss 1.00750219822
epoch 157, loss 1.0064227581
epoch 158, loss 1.00534558296
epoch 159, loss 1.00427126884
epoch 160, loss 1.00319552422
epoch 161, loss 1.00212168694
epoch 162, loss 1.00104928017
epoch 163, loss 0.99997740984
epoch 164, loss 0.998907446861
epoch 165, loss 0.997838199139
epoch 166, loss 0.996771931648
epoch 167, loss 0.995703995228
epoch 168, loss 0.994637966156
epoch 169, loss 0.993573665619
epoch 170, loss 0.992510199547
epoch 171, loss 0.991447746754
epoch 172, loss 0.990386784077
epoch 173, loss 0.989326655865
epoch 174, loss 0.988267123699
epoch 175, loss 0.987210392952
epoch 176, loss 0.986153483391
epoch 177, loss 0.985097527504
epoch 178, loss 0.984043121338
```

```
epoch 179, loss 0.982990145683
epoch 180, loss 0.981939017773
epoch 181, loss 0.980888009071
epoch 182, loss 0.979837536812
epoch 183, loss 0.978789806366
epoch 184, loss 0.977740943432
epoch 185, loss 0.976695179939
epoch 186, loss 0.975649952888
epoch 187, loss 0.974604964256
epoch 188, loss 0.973562121391
epoch 189, loss 0.972519397736
epoch 190, loss 0.971478819847
epoch 191, loss 0.9704387784
epoch 192, loss 0.969401478767
epoch 193, loss 0.968362212181
epoch 194, loss 0.967327296734
epoch 195, loss 0.966291606426
epoch 196, loss 0.965257465839
epoch 197, loss 0.964224636555
epoch 198, loss 0.963192462921
epoch 199, loss 0.962162315845
epoch 200, loss 0.961131751537
epoch 201, loss 0.960103809834
epoch 202, loss 0.959075450897
epoch 203, loss 0.958049118519
epoch 204, loss 0.957024395466
epoch 205, loss 0.95599925518
epoch 206, loss 0.954976439476
epoch 207, loss 0.953954398632
epoch 208, loss 0.952932834625
epoch 209, loss 0.951912999153
epoch 210, loss 0.95089495182
epoch 211, loss 0.949876189232
epoch 212, loss 0.948859095573
epoch 213, loss 0.947844982147
epoch 214, loss 0.946830153465
epoch 215, loss 0.945816338062
epoch 216, loss 0.944804489613
epoch 217, loss 0.943793416023
epoch 218, loss 0.942782759666
epoch 219, loss 0.941774070263
epoch 220, loss 0.940765857697
epoch 221, loss 0.939758956432
epoch 222, loss 0.938753008842
epoch 223, loss 0.937748432159
epoch 224, loss 0.936745524406
epoch 225, loss 0.935741782188
epoch 226, loss 0.934740841389
epoch 227, loss 0.933740496635
epoch 228, loss 0.932740986347
epoch 229, loss 0.931742370129
epoch 230, loss 0.930745720863
epoch 231, loss 0.92974960804
epoch 232, loss 0.928754210472
epoch 233, loss 0.927759647369
epoch 234, loss 0.926767170429
epoch 235, loss 0.925775527954
epoch 236, loss 0.924784362316
epoch 237, loss 0.923794746399
epoch 238, loss 0.922805786133
epoch 239, loss 0.921818375587
epoch 240, loss 0.920831859112
epoch 241, loss 0.919846057892
epoch 242, loss 0.918861567974
epoch 243, loss 0.917877852917
epoch 244, loss 0.91689568758
epoch 245, loss 0.915914237499
epoch 246, loss 0.914934277534
epoch 247, loss 0.913955032825
epoch 248, loss 0.912976861
epoch 249, loss 0.91199952364
epoch 250, loss 0.911023259163
epoch 251, loss 0.910047829151
epoch 252, loss 0.909074306488
epoch 253, loss 0.908101379871
epoch 254, loss 0.907129406929
epoch 255, loss 0.906159043312
epoch 256, loss 0.905188858509
epoch 257, loss 0.904220223427
epoch 258, loss 0.903252482414
epoch 259, loss 0.902285277843
epoch 260, loss 0.901319384575
epoch 261, loss 0.900355517864
epoch 262, loss 0.899390697479
epoch 263, loss 0.898429691792
epoch 264, loss 0.897467255592
epoch 265, loss 0.896506667137
```

```
epoch 266, loss 0.895547032356
epoch 267, loss 0.89458835125
epoch 268, loss 0.893631160259
epoch 269, loss 0.892674803734
epoch 270, loss 0.891718745232
epoch 271, loss 0.890764534473
epoch 272, loss 0.889811813831
epoch 273, loss 0.888859570026
epoch 274, loss 0.887908160686
epoch 275, loss 0.886957943439
epoch 276, loss 0.886007785797
epoch 277, loss 0.885059535503
epoch 278, loss 0.884112656116
epoch 279, loss 0.883166611195
epoch 280, loss 0.882221400738
epoch 281, loss 0.881277084351
epoch 282, loss 0.880333602428
epoch 283, loss 0.879391491413
epoch 284, loss 0.8784506917
epoch 285, loss 0.877509474754
epoch 286, loss 0.876570820808
epoch 287, loss 0.875633537769
epoch 288, loss 0.874695897102
epoch 289, loss 0.873760223389
epoch 290, loss 0.87282449007
epoch 291, loss 0.871890425682
epoch 292, loss 0.870957255363
epoch 293, loss 0.870025336742
epoch 294, loss 0.869093596935
epoch 295, loss 0.868164181709
epoch 296, loss 0.867234230042
epoch 297, loss 0.866306006908
epoch 298, loss 0.865379035473
epoch 299, loss 0.864453196526
epoch 300, loss 0.863526701927
epoch 301, loss 0.862603843212
epoch 302, loss 0.861680030823
epoch 303, loss 0.860757827759
epoch 304, loss 0.85983645916
epoch 305, loss 0.858916163445
epoch 306, loss 0.857997298241
epoch 307, loss 0.85707873106
epoch 308, loss 0.856161415577
epoch 309, loss 0.855246067047
epoch 310, loss 0.854329943657
epoch 311, loss 0.853415310383
epoch 312, loss 0.852502346039
epoch 313, loss 0.851590156555
epoch 314, loss 0.850677967072
epoch 315, loss 0.849767506123
epoch 316, loss 0.848858356476
epoch 317, loss 0.847949504852
epoch 318, loss 0.84704208374
epoch 319, loss 0.846135795116
epoch 320, loss 0.845229744911
epoch 321, loss 0.844325423241
epoch 322, loss 0.843421339989
epoch 323, loss 0.842518806458
epoch 324, loss 0.841616034508
epoch 325, loss 0.840716183186
epoch 326, loss 0.839816212654
epoch 327, loss 0.838917732239
epoch 328, loss 0.838019549847
epoch 329, loss 0.837123036385
epoch 330, loss 0.836227238178
epoch 331, loss 0.835331439972
epoch 332, loss 0.834438204765
epoch 333, loss 0.833545207977
epoch 334, loss 0.832652568817
epoch 335, loss 0.831760764122
epoch 336, loss 0.830871462822
epoch 337, loss 0.829982161522
epoch 338, loss 0.829094231129
epoch 339, loss 0.828205943108
epoch 340, loss 0.827320098877
epoch 341, loss 0.826433956623
epoch 342, loss 0.825549960136
epoch 343, loss 0.824666500092
epoch 344, loss 0.823783993721
epoch 345, loss 0.822902083397
epoch 346, loss 0.822021186352
epoch 347, loss 0.821141421795
epoch 348, loss 0.820262610912
epoch 349, loss 0.819384694099
epoch 350, loss 0.818507790565
epoch 351, loss 0.817631900311
```

```
epoch 352, loss 0.816756844521
epoch 353, loss 0.815882503986
epoch 354, loss 0.815009713173
epoch 355, loss 0.814137279987
epoch 356, loss 0.813265919685
epoch 357, loss 0.812395095825
epoch 358, loss 0.811525523663
epoch 359, loss 0.810657143593
epoch 360, loss 0.809789180756
epoch 361, loss 0.808922946453
epoch 362, loss 0.808056950569
epoch 363, loss 0.807192504406
epoch 364, loss 0.806328296661
epoch 365, loss 0.805464625359
epoch 366, loss 0.804603278637
epoch 367, loss 0.803742587566
epoch 368, loss 0.802882015705
epoch 369, loss 0.802022814751
epoch 370, loss 0.801163971424
epoch 371, loss 0.800305664539
epoch 372, loss 0.799450039864
epoch 373, loss 0.798594295979
epoch 374, loss 0.797739744186
epoch 375, loss 0.796885728836
epoch 376, loss 0.796033501625
epoch 377, loss 0.7951810956
epoch 378, loss 0.794329583645
epoch 379, loss 0.79347962141
epoch 380, loss 0.792630374432
epoch 381, loss 0.79178249836
epoch 382, loss 0.790934920311
epoch 383, loss 0.790088713169
epoch 384, loss 0.789242625237
epoch 385, loss 0.788398146629
epoch 386, loss 0.78755402565
epoch 387, loss 0.786711573601
epoch 388, loss 0.785869300365
epoch 389, loss 0.785028576851
epoch 390, loss 0.78418803215
epoch 391, loss 0.783348798752
epoch 392, loss 0.782510399818
epoch 393, loss 0.781673669815
epoch 394, loss 0.780836701393
epoch 395, loss 0.780000209808
epoch 396, loss 0.779165744781
epoch 397, loss 0.778331875801
epoch 398, loss 0.77749890089
epoch 399, loss 0.776666760445
epoch 400, loss 0.775835394859
epoch 401, loss 0.775004982948
epoch 402, loss 0.77417576313
epoch 403, loss 0.773346424103
epoch 404, loss 0.772519230843
epoch 405, loss 0.771692216396
epoch 406, loss 0.770866215229
epoch 407, loss 0.770041763783
epoch 408, loss 0.769217371941
epoch 409, loss 0.768394172192
epoch 410, loss 0.767570853233
epoch 411, loss 0.766750276089
epoch 412, loss 0.765929281712
epoch 413, loss 0.765109777451
epoch 414, loss 0.764290452003
epoch 415, loss 0.763472735882
epoch 416, loss 0.762655556202
epoch 417, loss 0.761839568615
epoch 418, loss 0.761023163795
epoch 419, loss 0.760208964348
epoch 420, loss 0.759395718575
epoch 421, loss 0.758583366871
epoch 422, loss 0.757771253586
epoch 423, loss 0.756960272789
epoch 424, loss 0.756150126457
epoch 425, loss 0.755340695381
epoch 426, loss 0.754532814026
epoch 427, loss 0.753725349903
epoch 428, loss 0.752918124199
epoch 429, loss 0.752112329006
epoch 430, loss 0.751306772232
epoch 431, loss 0.750502943993
epoch 432, loss 0.749699950218
epoch 433, loss 0.748897194862
epoch 434, loss 0.748095393181
epoch 435, loss 0.747294843197
epoch 436, loss 0.746495068073
epoch 437, loss 0.745696365833
epoch 438, loss 0.744898676872
```

```
epoch 439, loss 0.744100928307
epoch 440, loss 0.74330496788
epoch 441, loss 0.742509245872
epoch 442, loss 0.741714060307
epoch 443, loss 0.740920722485
epoch 444, loss 0.740127801895
epoch 445, loss 0.739335119724
epoch 446, loss 0.738543987274
epoch 447, loss 0.737753987312
epoch 448, loss 0.73696398735
epoch 449, loss 0.736175179482
epoch 450, loss 0.735387027264
epoch 451, loss 0.734600186348
epoch 452, loss 0.73381459713
epoch 453, loss 0.733029007912
epoch 454, loss 0.732244610786
epoch 455, loss 0.731460571289
epoch 456, loss 0.73067766428
epoch 457, loss 0.729895353317
epoch 458, loss 0.729114890099
epoch 459, loss 0.728334486485
epoch 460, loss 0.727554738522
epoch 461, loss 0.72677564621
epoch 462, loss 0.725997924805
epoch 463, loss 0.725220680237
epoch 464, loss 0.72444498539
epoch 465, loss 0.723669946194
epoch 466, loss 0.722895383835
epoch 467, loss 0.722121119499
epoch 468, loss 0.72134822607
epoch 469, loss 0.720576047897
epoch 470, loss 0.719805121422
```