```
%matplotlib inline
```

Double-click (or enter) to edit

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

```
import os
os.chdir('/content/drive/My Drive/cs505/char_rnn_tutorial')  #achange dir
!pwd
```

```
/content/drive/My Drive/cs505/char_rnn_tutorial
```

Classifying Names with a Character-Level RNN

Author: `Sean Robertson <https://github.com/spro/practical-pytorch>`_

We will be building and training a basic character-level RNN to classify words. A character-level RNN outputting a prediction and "hidden state" at each step, feeding its previous hidden state into each ne the output, i.e. which class the word belongs to.

Specifically, we'll train on a few thousand surnames from 18 languages of origin, and predict which la spelling:

::

```
$ python predict.py Hinton
(-0.47) Scottish
(-1.52) English
(-3.57) Irish

$ python predict.py Schmidhuber
(-0.19) German
(-2.48) Czech
(-2.68) Dutch
```

Recommended Reading:

I assume you have at least installed PyTorch, know Python, and understand Tensors:

- http://pytorch.org/ For installation instructions
- :doc: `/beginner/deep_learning_60min_blitz` to get started with PyTorch in general
- :doc: `/beginner/pytorch_with_examples` for a wide and deep overview

- :doc: `/beginner/former_torchies_tutorial` if you are former Lua Torch user

It would also be useful to know about RNNs and how they work:

- `The Unreasonable Effectiveness of Recurrent Neural Networks <http://karpathy.g` `effectiveness/>` __ shows a bunch of real life examples
- `Understanding LSTM Networks <http://colah.github.io/posts/2015-08-Understandin` but also informative about RNNs in general

Preparing the Data

.. Note:: Download the data from `here <https://download.pytorch.org/tutorial/data.zip>` _ a

Included in the `data/names` directory are 18 text files named as "[Language].txt". Each file contains a mostly romanized (but we still need to convert from Unicode to ASCII).

We'll end up with a dictionary of lists of names per language, `{language: [names ...]}`. The gene language and name in our case) are used for later extensibility.

```
from __future__ import unicode_literals, print_function, division
from io import open
import glob
import os

def findFiles(path): return glob.glob(path)

print(findFiles('data/cities_train/*.txt'))

import unicodedata
import string

all_letters = string.ascii_letters + " .,;'"
n_letters = len(all_letters)

# Turn a Unicode string to plain ASCII, thanks to http://stackoverflow.com/a/518232/2
def unicodeToAscii(s):
    return ''.join(
        c for c in unicodedata.normalize('NFD', s)
        if unicodedata.category(c) != 'Mn'
        and c in all_letters
    )

print(unicodeToAscii('Ślusàrski'))

# Build the category_lines dictionary, a list of names per language
category_lines = {}
val_category_lines = {}
all_categories = []
val_categories = []
val_categories = []

# Read a file and split into lines
def readLines(filename):
    lines = open(filename, encoding="ISO-8859-1").read().split('\n')
    return [unicodeToAscii(line) for line in lines]

for filename in findFiles('data/cities_train/*.txt'):
    category = os.path.splitext(os.path.basename(filename))[0]
    all_categories.append(category)
    lines = readLines(filename)[:-1]
    category_lines[category] = lines

n_categories = len(all_categories)

for filename in findFiles('data/cities_val/*.txt'):
    category = os.path.splitext(os.path.basename(filename))[0]
    val_categories.append(category)
    lines = readLines(filename)[:-1]
    val_category_lines[category] = lines
```

```
['data/cities_train/cn.txt', 'data/cities_train/za.txt', 'data/cities_train/de.tx
Slusarski
```

Now we have `category_lines`, a dictionary mapping each category (language) to a list of lines (nan `all_categories` (just a list of languages) and `n_categories` for later reference.

```
print(category_lines['cn'][-5:])
print(val_category_lines['cn'][-5:])
```

```
['cuizongzhuang', 'hetou', 'hulstai', 'shuanglazi', 'tebongori']
['xueguangzhang', 'ian', 'niujiaoxu', 'shuipo', 'daohugou']
```

Turning Names into Tensors

Now that we have all the names organized, we need to turn them into Tensors to make any use of the

To represent a single letter, we use a "one-hot vector" of size `<1 x n_letters>`. A one-hot vector is f current letter, e.g. `"b" = <0 1 0 0 0 ...>`.

To make a word we join a bunch of those into a 2D matrix `<line_length x 1 x n_letters>`.

That extra 1 dimension is because PyTorch assumes everything is in batches - we're just using a batc

```
import torch

# Find letter index from all_letters, e.g. "a" = 0
def letterToIndex(letter):
    return all_letters.find(letter)

# Just for demonstration, turn a letter into a <1 x n_letters> Tensor
def letterToTensor(letter):
    tensor = torch.zeros(1, n_letters)
    tensor[0][letterToIndex(letter)] = 1
    return tensor

# Turn a line into a <line_length x 1 x n_letters>,
# or an array of one-hot letter vectors
def lineToTensor(line):
    tensor = torch.zeros(len(line), 1, n_letters)
    for li, letter in enumerate(line):
        tensor[li][0][letterToIndex(letter)] = 1
    return tensor

print(letterToTensor('J'))

print(lineToTensor('Jones').size())
```

```
tensor([[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1.,
         0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
         0., 0., 0.]])
torch.Size([5, 1, 57])
```

Creating the Network

Before autograd, creating a recurrent neural network in Torch involved cloning the parameters of a lay hidden state and gradients which are now entirely handled by the graph itself. This means you can im regular feed-forward layers.

This RNN module (mostly copied from `the PyTorch for Torch users tutorial <http://pytorch.org/tutorials/beginner/former_torchies/ nn_tutorial.html#example-2-` which operate on an input and hidden state, with a LogSoftmax layer after the output.

.. figure:: https://i.imgur.com/Z2xbySO.png :alt:

```
import torch.nn as nn

class RNN(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(RNN, self).__init__()

        self.hidden_size = hidden_size

        self.i2h = nn.Linear(input_size + hidden_size, hidden_size)
        self.i2o = nn.Linear(input_size + hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)
    def forward(self, input, hidden):
        combined = torch.cat((input, hidden), 1)
        hidden = self.i2h(combined)
        output = self.i2o(combined)
        output = self.softmax(output)
        return output, hidden

    def initHidden(self):
        return torch.zeros(1, self.hidden_size)

n_hidden = 128
rnn = RNN(n_letters, n_hidden, n_categories)
```

To run a step of this network we need to pass an input (in our case, the Tensor for the current letter) initialize as zeros at first). We'll get back the output (probability of each language) and a next hidden

```
input = letterToTensor('A')
hidden =torch.zeros(1, n_hidden)

output, next_hidden = rnn(input, hidden)
print(output)
```

```
tensor([[-2.1508, -2.2374, -2.3111, -2.1435, -2.2026, -2.2217, -2.1695, -2.1645,
         -2.1848]], grad_fn=<LogSoftmaxBackward>)
```

For the sake of efficiency we don't want to be creating a new Tensor for every step, so we will use `li`
and use slices. This could be further optimized by pre-computing batches of Tensors.

```
input = lineToTensor('Albert')
hidden = torch.zeros(1, n_hidden)

output, next_hidden = rnn(input[0], hidden)
print(output)
```

```
tensor([[-2.1508, -2.2374, -2.3111, -2.1435, -2.2026, -2.2217, -2.1695, -2.1645,
         -2.1848]], grad_fn=<LogSoftmaxBackward>)
```

As you can see the output is a `<1 x n_categories>` Tensor, where every item is the likelihood of th

Training

Preparing for Training

Before going into training we should make a few helper functions. The first is to interpret the output o likelihood of each category. We can use `Tensor.topk` to get the index of the greatest value:

```
def categoryFromOutput(output):
    top_n, top_i = output.topk(1)
    category_i = top_i[0].item()
    return all_categories[category_i], category_i

print(categoryFromOutput(output))
```

```
('fr', 3)
```

We will also want a quick way to get a training example (a name and its language):

```
import random

def randomChoice(l):
    return l[random.randint(0, len(l) - 1)]

def randomTrainingExample():
    category = randomChoice(all_categories)
    line = randomChoice(category_lines[category])
    category_tensor = torch.tensor([all_categories.index(category)], dtype=torch.long
    line_tensor = lineToTensor(line)
    return category, line, category_tensor, line_tensor

def randomValidationExample():
    category = randomChoice(val_categories)
    line = randomChoice(val_category_lines[category])
    val_category_tensor = torch.tensor([val_categories.index(category)], dtype=torch.
    val_line_tensor = lineToTensor(line)
    return category, line, val_category_tensor, val_line_tensor

def shuffle_arrs(a,b,c,d):
    combined = list(zip(a, b, c, d))
    random.shuffle(combined)
    a, b, c, d = zip(*combined)
    return a,b,c,d

def genData(category_line_hash, categories_arr):
    x, y, x_tensor, y_tensor = [], [], [], []
    for y_category in category_line_hash.keys():
        for x_line in category_line_hash[y_category]:
            y.append(y_category)
            x.append(x_line)
            y_tensor.append(torch.tensor([categories_arr.index(y_category)], dtype=to
            x_tensor.append(lineToTensor(x_line))
    x, y, x_tensor, y_tensor = shuffle_arrs(x, y, x_tensor, y_tensor)
    return x, y, x_tensor, y_tensor

def TrainingData():
    return genData(category_lines, all_categories)
    # y = []
    # y = []
    # x = []
    # for y_category in category_lines.keys():
    #     for x_line in category_lines[y_category]:
    #         y.append(y_category)
    #         x.append(x_line)
    #         y_tensor.append(torch.tensor([val_categories.index(category)], dtype=to
    #         x_tensor.append(lineToTensor(x_line))
    # x, y, x_tensor, y_tensor = shuffle_arrs(x, y, x_tensor, y_tensor)
    # return x, y

def ValidationData():
    return genData(val_category_lines, val_categories)
    # y = []
    # x = []
    # y_tensor = []
    # x_tensor = []
    # for y_category in val_category_lines.keys():
    #     for x_line in val_category_lines[y_category]:
    #         y.append(y_category)
    #         x.append(x_line)
    #         y_tensor.append(torch.tensor([val_categories.index(category)], dtype=to
    #         x_tensor.append(lineToTensor(x_line))
    # x, y, x_tensor, y_tensor = shuffle_arrs(x, y, x_tensor, y_tensor)
    # return x, y

print("=== Train ===")
x,y,x_tensor,y_tensor= TrainingData()
print(x[:5])
print(y[:5])
# print(x_tensor[:1])
# print(y_tensor[:1])

print("=== Validation ===")
x,y,x_tensor,y_tensor = ValidationData()
print(x[:5])
print(y[:5])
# print(x_tensor[:1])
# print(y_tensor[:1])
```

```
=== Train ===
('eguenigue', 'hazar now', 'grosssteinbach', "podere sant'elisa", 'khanabade qotl
('fr', 'af', 'in', 'de', 'ir')
=== Validation ===
('khinddur', 'aubure', 'koppies irrigation settlement', 'kuth nari', 'lutcza')
('af', 'fr', 'za', 'pk', 'za')
```

Training the Network

Now all it takes to train this network is show it a bunch of examples, have it make guesses, and tell it

For the loss function `nn.NLLLoss` is appropriate, since the last layer of the RNN is `nn.LogSoftmax`.

```
criterion = nn.NLLLoss()
```

Each loop of training will:

- Create input and target tensors
- Create a zeroed initial hidden state
- Read each letter in and
    - Keep hidden state for next letter
- Compare final output to target
- Back-propagate
- Return the output and loss

```
learning_rate = 0.002 # If you set this too high, it might explode. If too low, it mi

def train(category_tensor, line_tensor):
    hidden = rnn.initHidden()

    rnn.zero_grad()

    # print("category_tensor={}, line_tensor.size()[0]={}".format(category_tensor, li
    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)

    loss = criterion(output, category_tensor)
    loss.backward()

    # Add parameters' gradients to their values, multiplied by learning rate
    for p in rnn.parameters():
        p.data.add_(-learning_rate, p.grad.data)

    return output, loss.item()

# Just return an output given a line
def evaluate(line_tensor, category_tensor):
    hidden = rnn.initHidden()

    # print("evaluate debug")
    # print(line_tensor.size()[0])
    # print("line_tensor")
    # print(line_tensor)
    for i in range(line_tensor.size()[0]):
        output, hidden = rnn(line_tensor[i], hidden)
    loss = criterion(output, category_tensor)
    loss = criterion(output, category_tensor)

    return output, loss.item()
```

Now we just have to run that with a bunch of examples. Since the `train` function returns both the ou also keep track of loss for plotting. Since there are 1000s of examples we print only every `print_eve` loss.

```
# import time
# import math

# print_every = 1000 # total = 27000
# plot_every = 1000 # 5000

# # Keep track of losses for plotting
# current_loss = 0
# val_losses = 0.
# train_acc_thru_time_aggregate, val_acc_thru_time_aggregate = 0., 0.
# train_losses_thru_time = []
# val_losses_thru_time = []
# train_acc_thru_time = []
# val_acc_thru_time = []

# def timeSince(since):
#     now = time.time()
#     s = now - since
#     m = math.floor(s / 60)
#     s -= m * 60
#     return '%dm %ds' % (m, s)

# start = time.time()

# print("learning rate = ", learning_rate)

# x_train, y_train, x_train_tensor, y_train_tensor = TrainingData()
# x_val, y_val, x_val_tensor, y_val_tensor = ValidationData()

# x_train_len = len(x_train)
# x_val_len = 10 # len(x_val)
# print("x_train_len:", x_train_len, ", x_val_len:", x_val_len)

# for i in range(x_train_len):
#     # category, line, category_tensor, line_tensor = randomTrainingExample()  # TOD
#     category = y_train[i]
#     line = x_train[i]
#     category_tensor = y_train_tensor[i]
#     line_tensor = x_train_tensor[i]

#     output, loss = train(category_tensor, line_tensor)
#     current_loss += loss
#     val_loss_per_train_data = 0
#     val_correct_guess_count = 0
#     train_correct_guess_count = 0
#     # for j in range(x_val_len):
#     #     val_output, val_loss = evaluate(x_val_tensor[j], y_val_tensor[j])
#     #     val_loss_per_train_data += val_loss
#     for j in range(x_val_len):
#         # Train accuracy calc
#         train_category, _, train_category_tensor, train_line_tensor = randomTrainin
#         train_output, train_loss = evaluate(train_line_tensor, train_category_tenso
#         train_guess, _ = categoryFromOutput(train_output)
#         train_correct_guess_count += int(train_guess == train_category)

#         # Validation accuracy calc
#         val_category, _, val_category_tensor, val_line_tensor = randomValidationExa
#         val_output, val_loss = evaluate(val_line_tensor, val_category_tensor)
#         val_guess, _ = categoryFromOutput(val_output)
#         val_correct_guess_count += int(val_guess == val_category)

#         val_loss_per_train_data += val_loss

#     # Aggregate accuracy
#     train_acc_per_train_data = train_correct_guess_count / x_val_len
#     train_acc_thru_time_aggregate += train_acc_per_train_data
#     val_acc_per_train_data = val_correct_guess_count / x_val_len
#     val_acc_thru_time_aggregate += val_acc_per_train_data

#     # Aggregate validation loss
#     val_loss_per_train_data_ave = val_loss_per_train_data / x_val_len
#     val_losses += val_loss_per_train_data_ave

#     # Print iter number, loss, name and guess
#     if i % print_every == 0:
#         print("iter = {}({:d}%) | time taken = {} | train_loss={:.4f}, val_loss(ave
#         debug_x, debug_y, debug_x_tensor, debug_y_tensor = [], [], [], []

#     # Add current loss avg to list of losses
#     if i % plot_every == 0:
#         train_losses_thru_time.append(current_loss / plot_every)
#         val_losses_thru_time.append(val_losses / plot_every)
#         current_loss = 0
#         val_losses = 0

#         print("iter = {}({:d}%) | time taken = {} | train_acc_thru_time_ave={}, val

#         train_acc_thru_time.append(train_acc_thru_time_aggregate / plot_every)
#         val_acc_thru_time.append(val_acc_thru_time_aggregate / plot_every)
#         train_acc_thru_time_aggregate = 0
#         val_acc_thru_time_aggregate = 0
# import matplotlib.pyplot as plt
# import matplotlib.ticker as ticker

# plt.figure()
# train_loss_plot = plt.plot(train_losses_thru_time[1:], label='Train Loss')
# val_loss_plot = plt.plot(val_losses_thru_time[1:], label="Val Loss")
# plt.legend()

# print("train_losses_thru_time")
# print(train_losses_thru_time[1:])
# print("val_losses_thru_time")
# print(val_losses_thru_time[1:])


# import matplotlib.pyplot as plt
# import matplotlib.ticker as ticker

# plt.figure()
# train_acc_plot = plt.plot(train_acc_thru_time[1:], label='Train Accuracy')
# val_acc_plot = plt.plot(val_acc_thru_time[1:], label="Val Accuracy")
# plt.legend()

# print("train_acc_thru_time")
# print(train_acc_thru_time[1:])
# print("val_acc_thru_time")
# print(val_acc_thru_time[1:])


import time
import math

print_every = 100 # total = 27000
plot_every = 100 # 5000

# Keep track of losses for plotting
current_loss = 0
val_losses = 0.
train_acc_thru_time_aggregate, val_acc_thru_time_aggregate = 0., 0.
train_losses_thru_time = []
val_losses_thru_time = []
train_acc_thru_time = []
val_acc_thru_time = []

def timeSince(since):
    now = time.time()
    s = now - since
    m = math.floor(s / 60)
    s -= m * 60
    return '%dm %ds' % (m, s)

start = time.time()
```

```
    print("learning rate = ", learning_rate)

    x_train, y_train, x_train_tensor, y_train_tensor = TrainingData()
    x_val, y_val, x_val_tensor, y_val_tensor = ValidationData()

    x_train_len = len(x_train)
    x_val_len = 10 # len(x_val)
    print("x_train_len:", x_train_len, ", x_val_len:", x_val_len)

    for i in range(x_train_len):
        # category, line, category_tensor, line_tensor = randomTrainingExample()  # TODO:
        category = y_train[i]
        line = x_train[i]
        category_tensor = y_train_tensor[i]
        line_tensor = x_train_tensor[i]

        output, loss = train(category_tensor, line_tensor)
        current_loss += loss

        val_loss_per_train_data = 0
        val_correct_guess_count = 0
        train_correct_guess_count = 0
        # for j in range(x_val_len):
        #     val_output, val_loss = evaluate(x_val_tensor[j], y_val_tensor[j])
        #     val_loss_per_train_data += val_loss
        for j in range(x_val_len):
            # Train accuracy calc
            train_category, _, train_category_tensor, train_line_tensor = randomTrainingE
            train_output, train_loss = evaluate(train_line_tensor, train_category_tensor)
            train_guess, _ = categoryFromOutput(train_output)
            train_correct_guess_count += int(train_guess == train_category)

            # Validation accuracy calc
            val_category, _, val_category_tensor, val_line_tensor = randomValidationExamp
            val_output, val_loss = evaluate(val_line_tensor, val_category_tensor)
            val_guess, _ = categoryFromOutput(val_output)
            val_correct_guess_count += int(val_guess == val_category)

            val_loss_per_train_data += val_loss

        # Aggregate accuracy
        train_acc_per_train_data = train_correct_guess_count / x_val_len
        train_acc_thru_time_aggregate += train_acc_per_train_data
        val_acc_per_train_data = val_correct_guess_count / x_val_len
        val_acc_thru_time_aggregate += val_acc_per_train_data

        # Aggregate validation loss
        val_loss_per_train_data_ave = val_loss_per_train_data / x_val_len
        val_losses += val_loss_per_train_data_ave

        # Print iter number, loss, name and guess
        if i % print_every == 0:
            print("iter = {}({:d}%) | time taken = {} | train_loss={:.4f}, val_loss(ave)=
            debug_x, debug_y, debug_x_tensor, debug_y_tensor = [], [], [], []

        # Add current loss avg to list of losses
        if i % plot_every == 0:
            train_losses_thru_time.append(current_loss / plot_every)
            val_losses_thru_time.append(val_losses / plot_every)
            current_loss = 0
            val_losses = 0

            print("iter = {}({:d}%) | time taken = {} | train_acc_thru_time_ave={}, val_a

            train_acc_thru_time.append(train_acc_thru_time_aggregate / plot_every)
            val_acc_thru_time.append(val_acc_thru_time_aggregate / plot_every)
            train_acc_thru_time_aggregate = 0
            val_acc_thru_time_aggregate = 0
```

```
iter = 12300(45%) | time taken = 3m 58s | train_loss=2.2002, val_loss(ave)=2.2800
iter = 12300(45%) | train_acc_thru_time_ave=3m 58s, val_acc_thru_time_ave=0.33299
iter = 12400(45%) | time taken = 4m 0s | train_loss=2.2722, val_loss(ave)=2.3406
iter = 12400(45%) | train_acc_thru_time_ave=4m 0s, val_acc_thru_time_ave=0.314999
iter = 12500(46%) | time taken = 4m 2s | train_loss=2.1329, val_loss(ave)=2.3267
iter = 12500(46%) | train_acc_thru_time_ave=4m 2s, val_acc_thru_time_ave=0.282000
iter = 12600(46%) | time taken = 4m 4s | train_loss=1.8700, val_loss(ave)=2.3448
iter = 12600(46%) | train_acc_thru_time_ave=4m 4s, val_acc_thru_time_ave=0.311
iter = 12700(47%) | time taken = 4m 6s | train_loss=1.8203, val_loss(ave)=2.2659
iter = 12700(47%) | train_acc_thru_time_ave=4m 6s, val_acc_thru_time_ave=0.291999
iter = 12800(47%) | time taken = 4m 8s | train_loss=1.8880, val_loss(ave)=2.2464
iter = 12800(47%) | train_acc_thru_time_ave=4m 8s, val_acc_thru_time_ave=0.299000
iter = 12900(47%) | time taken = 4m 9s | train_loss=1.8384, val_loss(ave)=2.2957
iter = 12900(47%) | train_acc_thru_time_ave=4m 9s, val_acc_thru_time_ave=0.321000
iter = 13000(48%) | time taken = 4m 11s | train_loss=1.5378, val_loss(ave)=2.1822
iter = 13000(48%) | train_acc_thru_time_ave=4m 11s, val_acc_thru_time_ave=0.35
iter = 13100(48%) | time taken = 4m 13s | train_loss=1.7543, val_loss(ave)=2.5320
iter = 13100(48%) | train_acc_thru_time_ave=4m 13s, val_acc_thru_time_ave=0.325
iter = 13200(48%) | time taken = 4m 15s | train_loss=2.3358, val_loss(ave)=2.3173
iter = 13200(48%) | train_acc_thru_time_ave=4m 15s, val_acc_thru_time_ave=0.30900
iter = 13300(49%) | time taken = 4m 17s | train_loss=1.9725, val_loss(ave)=2.3769
iter = 13300(49%) | train_acc_thru_time_ave=4m 17s, val_acc_thru_time_ave=0.30299
iter = 13400(49%) | time taken = 4m 19s | train_loss=2.2557, val_loss(ave)=2.5178
iter = 13400(49%) | train_acc_thru_time_ave=4m 19s, val_acc_thru_time_ave=0.33899
iter = 13500(50%) | time taken = 4m 20s | train_loss=1.7968, val_loss(ave)=2.2874
iter = 13500(50%) | train_acc_thru_time_ave=4m 20s, val_acc_thru_time_ave=0.311
iter = 13600(50%) | time taken = 4m 22s | train_loss=1.5210, val_loss(ave)=2.2623
iter = 13600(50%) | train_acc_thru_time_ave=4m 22s, val_acc_thru_time_ave=0.31499
iter = 13700(50%) | time taken = 4m 24s | train_loss=2.1146, val_loss(ave)=2.4993
iter = 13700(50%) | train_acc_thru_time_ave=4m 24s, val_acc_thru_time_ave=0.31599
iter = 13800(51%) | time taken = 4m 26s | train_loss=2.0158, val_loss(ave)=2.4807
iter = 13800(51%) | train_acc_thru_time_ave=4m 26s, val_acc_thru_time_ave=0.30900
iter = 13900(51%) | time taken = 4m 28s | train_loss=1.6452, val_loss(ave)=2.6433
iter = 13900(51%) | train_acc_thru_time_ave=4m 28s, val_acc_thru_time_ave=0.317
iter = 14000(51%) | time taken = 4m 30s | train_loss=2.3315, val_loss(ave)=2.5099
iter = 14000(51%) | train_acc_thru_time_ave=4m 30s, val_acc_thru_time_ave=0.27899
iter = 14100(52%) | time taken = 4m 32s | train_loss=1.9080, val_loss(ave)=2.4434
iter = 14100(52%) | train_acc_thru_time_ave=4m 32s, val_acc_thru_time_ave=0.32800
iter = 14200(52%) | time taken = 4m 33s | train_loss=2.2569, val_loss(ave)=2.3588
iter = 14200(52%) | train_acc_thru_time_ave=4m 33s, val_acc_thru_time_ave=0.30300
iter = 14300(52%) | time taken = 4m 35s | train_loss=2.0032, val_loss(ave)=2.2868
iter = 14300(52%) | train_acc_thru_time_ave=4m 35s, val_acc_thru_time_ave=0.305
iter = 14400(53%) | time taken = 4m 37s | train_loss=2.2629, val_loss(ave)=2.9534
iter = 14400(53%) | train_acc_thru_time_ave=4m 37s, val_acc_thru_time_ave=0.33699
iter = 14500(53%) | time taken = 4m 39s | train_loss=2.0681, val_loss(ave)=2.2724
iter = 14500(53%) | train_acc_thru_time_ave=4m 39s, val_acc_thru_time_ave=0.30100
iter = 14600(54%) | time taken = 4m 41s | train_loss=1.7303, val_loss(ave)=2.4939
iter = 14600(54%) | train_acc_thru_time_ave=4m 41s, val_acc_thru_time_ave=0.32099
iter = 14700(54%) | time taken = 4m 43s | train_loss=1.8400, val_loss(ave)=2.4820
iter = 14700(54%) | train_acc_thru_time_ave=4m 43s, val_acc_thru_time_ave=0.326
iter = 14800(54%) | time taken = 4m 45s | train_loss=2.1921, val_loss(ave)=2.0390
iter = 14800(54%) | train_acc_thru_time_ave=4m 45s, val_acc_thru_time_ave=0.32499
iter = 14900(55%) | time taken = 4m 46s | train_loss=1.6754, val_loss(ave)=2.3825
iter = 14900(55%) | train_acc_thru_time_ave=4m 46s, val_acc_thru_time_ave=0.344
iter = 15000(55%) | time taken = 4m 48s | train_loss=1.9468, val_loss(ave)=2.1685
iter = 15000(55%) | train_acc_thru_time_ave=4m 48s, val_acc_thru_time_ave=0.32
iter = 15100(55%) | time taken = 4m 50s | train_loss=1.7500, val_loss(ave)=2.2437
iter = 15100(55%) | train_acc_thru_time_ave=4m 50s, val_acc_thru_time_ave=0.31699
iter = 15100(55%) | train_acc_thru_time_ave=4m 50s, val_acc_thru_time_ave=0.3109
iter = 15200(56%) | time taken = 4m 52s | train_loss=2.0191, val_loss(ave)=2.4282
iter = 15200(56%) | train_acc_thru_time_ave=4m 52s, val_acc_thru_time_ave=0.3279
iter = 15300(56%) | time taken = 4m 54s | train_loss=1.4058, val_loss(ave)=2.2684
iter = 15300(56%) | train_acc_thru_time_ave=4m 54s, val_acc_thru_time_ave=0.312
iter = 15400(57%) | time taken = 4m 56s | train_loss=1.9848, val_loss(ave)=2.3435
iter = 15400(57%) | train_acc_thru_time_ave=4m 56s, val_acc_thru_time_ave=0.29300
iter = 15500(57%) | time taken = 4m 58s | train_loss=3.1281, val_loss(ave)=2.5012
iter = 15500(57%) | train_acc_thru_time_ave=4m 58s, val_acc_thru_time_ave=0.32300
iter = 15600(57%) | time taken = 4m 59s | train_loss=2.2698, val_loss(ave)=2.4336
iter = 15600(57%) | train_acc_thru_time_ave=4m 59s, val_acc_thru_time_ave=0.31600
iter = 15700(58%) | time taken = 5m 1s | train_loss=1.4748, val_loss(ave)=2.3805
iter = 15700(58%) | train_acc_thru_time_ave=5m 1s, val_acc_thru_time_ave=0.316
iter = 15800(58%) | time taken = 5m 3s | train_loss=2.0722, val_loss(ave)=2.8802
iter = 15800(58%) | train_acc_thru_time_ave=5m 3s, val_acc_thru_time_ave=0.339999
iter = 15900(58%) | time taken = 5m 5s | train_loss=1.5647, val_loss(ave)=2.2978
iter = 15900(58%) | train_acc_thru_time_ave=5m 5s, val_acc_thru_time_ave=0.334999
iter = 16000(59%) | time taken = 5m 7s | train_loss=1.2696, val_loss(ave)=2.7273
iter = 16000(59%) | train_acc_thru_time_ave=5m 7s, val_acc_thru_time_ave=0.343999
iter = 16100(59%) | time taken = 5m 9s | train_loss=1.5769, val_loss(ave)=2.2471
iter = 16100(59%) | train_acc_thru_time_ave=5m 9s, val_acc_thru_time_ave=0.348000
iter = 16200(60%) | time taken = 5m 10s | train_loss=1.8372, val_loss(ave)=2.3777
iter = 16200(60%) | train_acc_thru_time_ave=5m 10s, val_acc_thru_time_ave=0.33199
iter = 16300(60%) | time taken = 5m 12s | train_loss=1.9412, val_loss(ave)=2.0383
iter = 16300(60%) | train_acc_thru_time_ave=5m 12s, val_acc_thru_time_ave=0.35399
iter = 16400(60%) | time taken = 5m 14s | train_loss=2.1943, val_loss(ave)=2.5166
iter = 16400(60%) | train_acc_thru_time_ave=5m 14s, val_acc_thru_time_ave=0.35100
iter = 16500(61%) | time taken = 5m 16s | train_loss=1.8320, val_loss(ave)=1.9845
iter = 16500(61%) | train_acc_thru_time_ave=5m 16s, val_acc_thru_time_ave=0.32700
iter = 16600(61%) | time taken = 5m 18s | train_loss=2.3783, val_loss(ave)=2.4579
iter = 16600(61%) | train_acc_thru_time_ave=5m 18s, val_acc_thru_time_ave=0.33699
iter = 16700(61%) | time taken = 5m 20s | train_loss=2.0086, val_loss(ave)=2.6477
iter = 16700(61%) | train_acc_thru_time_ave=5m 20s, val_acc_thru_time_ave=0.32599
iter = 16800(62%) | time taken = 5m 22s | train_loss=2.1705, val_loss(ave)=2.5903
iter = 16800(62%) | train_acc_thru_time_ave=5m 22s, val_acc_thru_time_ave=0.333
iter = 16900(62%) | time taken = 5m 23s | train_loss=1.7867, val_loss(ave)=2.1766
iter = 16900(62%) | train_acc_thru_time_ave=5m 23s, val_acc_thru_time_ave=0.33199
iter = 17000(62%) | time taken = 5m 25s | train_loss=1.6074, val_loss(ave)=2.3057
iter = 17000(62%) | train_acc_thru_time_ave=5m 25s, val_acc_thru_time_ave=0.35299
iter = 17100(63%) | time taken = 5m 27s | train_loss=1.6309, val_loss(ave)=2.6039
iter = 17100(63%) | train_acc_thru_time_ave=5m 27s, val_acc_thru_time_ave=0.34700
iter = 17200(63%) | time taken = 5m 29s | train_loss=3.1790, val_loss(ave)=2.0832
iter = 17200(63%) | train_acc_thru_time_ave=5m 29s, val_acc_thru_time_ave=0.345
iter = 17300(64%) | time taken = 5m 31s | train_loss=1.3906, val_loss(ave)=2.1576
iter = 17300(64%) | train_acc_thru_time_ave=5m 31s, val_acc_thru_time_ave=0.29999
iter = 17400(64%) | time taken = 5m 32s | train_loss=1.0294, val_loss(ave)=2.5278
iter = 17400(64%) | train_acc_thru_time_ave=5m 32s, val_acc_thru_time_ave=0.32800
iter = 17500(64%) | time taken = 5m 34s | train_loss=2.0128, val_loss(ave)=2.4684
iter = 17500(64%) | train_acc_thru_time_ave=5m 34s, val_acc_thru_time_ave=0.30600
iter = 17600(65%) | time taken = 5m 36s | train_loss=1.4292, val_loss(ave)=2.7424
iter = 17600(65%) | train_acc_thru_time_ave=5m 36s, val_acc_thru_time_ave=0.33499
iter = 17700(65%) | time taken = 5m 38s | train_loss=1.5184, val_loss(ave)=2.2860
iter = 17700(65%) | train_acc_thru_time_ave=5m 38s, val_acc_thru_time_ave=0.336
iter = 17800(65%) | time taken = 5m 40s | train_loss=1.2573, val_loss(ave)=2.4766
iter = 17800(65%) | train_acc_thru_time_ave=5m 40s, val_acc_thru_time_ave=0.343
iter = 17900(66%) | time taken = 5m 42s | train_loss=1.4314, val_loss(ave)=2.2565
iter = 17900(66%) | train_acc_thru_time_ave=5m 42s, val_acc_thru_time_ave=0.34199
iter = 18000(66%) | time taken = 5m 43s | train_loss=1.7756, val_loss(ave)=2.5801
iter = 18000(66%) | train_acc_thru_time_ave=5m 43s, val_acc_thru_time_ave=0.35699
```

```
        iter = 18000(66%) | train_acc_thru_time_ave=5m 43s, val_acc_thru_time_ave=0.35699
        iter = 18100(67%) | time taken = 5m 45s | train_loss=1.9601, val_loss(ave)=3.0338
        iter = 18100(67%) | train_acc_thru_time_ave=5m 45s, val_acc_thru_time_ave=0.345
        iter = 18200(67%) | time taken = 5m 47s | train_loss=0.9575, val_loss(ave)=2.1005
        iter = 18200(67%) | train_acc_thru_time_ave=5m 47s, val_acc_thru_time_ave=0.34399
        iter = 18300(67%) | time taken = 5m 49s | train_loss=1.6310, val_loss(ave)=2.1908
        iter = 18300(67%) | train_acc_thru_time_ave=5m 49s, val_acc_thru_time_ave=0.34399
        iter = 18400(68%) | time taken = 5m 51s | train_loss=1.9476, val_loss(ave)=2.5692
        iter = 18400(68%) | train_acc_thru_time_ave=5m 51s, val_acc_thru_time_ave=0.32799
        iter = 18500(68%) | time taken = 5m 53s | train_loss=3.4008, val_loss(ave)=2.7739
        iter = 18500(68%) | train_acc_thru_time_ave=5m 53s, val_acc_thru_time_ave=0.35999
        iter = 18600(68%) | time taken = 5m 54s | train_loss=1.7850, val_loss(ave)=2.2558
        iter = 18600(68%) | train_acc_thru_time_ave=5m 54s, val_acc_thru_time_ave=0.39399
        iter = 18700(69%) | time taken = 5m 56s | train_loss=3.1697, val_loss(ave)=2.3454
        iter = 18700(69%) | train_acc_thru_time_ave=5m 56s, val_acc_thru_time_ave=0.37399
        iter = 18800(69%) | time taken = 5m 58s | train_loss=1.3339, val_loss(ave)=2.4799
        iter = 18800(69%) | train_acc_thru_time_ave=5m 58s, val_acc_thru_time_ave=0.38499
        iter = 18900(70%) | time taken = 6m 0s | train_loss=2.8672, val_loss(ave)=2.7980
        iter = 18900(70%) | train_acc_thru_time_ave=6m 0s, val_acc_thru_time_ave=0.371999
        iter = 19000(70%) | time taken = 6m 2s | train_loss=1.5832, val_loss(ave)=2.3897
        iter = 19000(70%) | train_acc_thru_time_ave=6m 2s, val_acc_thru_time_ave=0.369999
        iter = 19100(70%) | time taken = 6m 4s | train_loss=1.2720, val_loss(ave)=3.3501
        iter = 19100(70%) | train_acc_thru_time_ave=6m 4s, val_acc_thru_time_ave=0.373
        iter = 19200(71%) | time taken = 6m 5s | train_loss=0.6569, val_loss(ave)=2.5742
        iter = 19200(71%) | train_acc_thru_time_ave=6m 5s, val_acc_thru_time_ave=0.335000
        iter = 19300(71%) | time taken = 6m 7s | train_loss=1.8828, val_loss(ave)=2.4055
        iter = 19300(71%) | train_acc_thru_time_ave=6m 7s, val_acc_thru_time_ave=0.367000
        iter = 19400(71%) | time taken = 6m 9s | train_loss=1.4897, val_loss(ave)=2.7761
        iter = 19400(71%) | train_acc_thru_time_ave=6m 9s, val_acc_thru_time_ave=0.353999
        iter = 19500(72%) | time taken = 6m 11s | train_loss=1.4574, val_loss(ave)=2.5699
        iter = 19500(72%) | train_acc_thru_time_ave=6m 11s, val_acc_thru_time_ave=0.353
        iter = 19600(72%) | time taken = 6m 13s | train_loss=1.0938, val_loss(ave)=2.2041
        iter = 19600(72%) | train_acc_thru_time_ave=6m 13s, val_acc_thru_time_ave=0.33599
        iter = 19700(72%) | time taken = 6m 14s | train_loss=1.6390, val_loss(ave)=2.6427
        iter = 19700(72%) | train_acc_thru_time_ave=6m 14s, val_acc_thru_time_ave=0.344
        iter = 19800(73%) | time taken = 6m 16s | train_loss=1.8259, val_loss(ave)=2.6950
        iter = 19800(73%) | train_acc_thru_time_ave=6m 16s, val_acc_thru_time_ave=0.361
        iter = 19900(73%) | time taken = 6m 18s | train_loss=1.4627, val_loss(ave)=2.6592
        iter = 19900(73%) | train_acc_thru_time_ave=6m 18s, val_acc_thru_time_ave=0.333
        iter = 20000(74%) | time taken = 6m 20s | train_loss=1.2091, val_loss(ave)=2.6960
        iter = 20000(74%) | train_acc_thru_time_ave=6m 20s, val_acc_thru_time_ave=0.35700
        iter = 20100(74%) | time taken = 6m 22s | train_loss=2.4690, val_loss(ave)=2.6709
        iter = 20100(74%) | train_acc_thru_time_ave=6m 22s, val_acc_thru_time_ave=0.37799
        iter = 20200(74%) | time taken = 6m 23s | train_loss=1.7508, val_loss(ave)=2.7360
        iter = 20200(74%) | train_acc_thru_time_ave=6m 23s, val_acc_thru_time_ave=0.37199
        iter = 20300(75%) | time taken = 6m 25s | train_loss=2.0950, val_loss(ave)=2.4790
        iter = 20300(75%) | train_acc_thru_time_ave=6m 25s, val_acc_thru_time_ave=0.38699
        iter = 20400(75%) | time taken = 6m 27s | train_loss=1.4475, val_loss(ave)=2.6798
        iter = 20400(75%) | train_acc_thru_time_ave=6m 27s, val_acc_thru_time_ave=0.36399
        iter = 20500(75%) | time taken = 6m 29s | train_loss=2.2695, val_loss(ave)=2.5023
        iter = 20500(75%) | train_acc_thru_time_ave=6m 29s, val_acc_thru_time_ave=0.347
        iter = 20600(76%) | time taken = 6m 30s | train_loss=1.2968, val_loss(ave)=2.7411
        iter = 20600(76%) | train_acc_thru_time_ave=6m 30s, val_acc_thru_time_ave=0.347
        iter = 20700(76%) | time taken = 6m 32s | train_loss=1.5432, val_loss(ave)=2.3825
        iter = 20700(76%) | train_acc_thru_time_ave=6m 32s, val_acc_thru_time_ave=0.39199
        iter = 20800(77%) | time taken = 6m 34s | train_loss=2.1458, val_loss(ave)=2.2047
        iter = 20800(77%) | train_acc_thru_time_ave=6m 34s, val_acc_thru_time_ave=0.35099
        iter = 20900(77%) | time taken = 6m 36s | train_loss=1.1018, val_loss(ave)=2.4886
        iter = 20900(77%) | train_acc_thru_time_ave=6m 36s, val_acc_thru_time_ave=0.36199
        iter = 21000(77%) | time taken = 6m 38s | train_loss=1.9379, val_loss(ave)=2.9478
        iter = 21000(77%) | train_acc_thru_time_ave=6m 38s, val_acc_thru_time_ave=0.382
        iter = 21100(78%) | time taken = 6m 39s | train_loss=1.1685, val_loss(ave)=2.3252
        iter = 21100(78%) | train_acc_thru_time_ave=6m 39s, val_acc_thru_time_ave=0.37299
        iter = 21200(78%) | time taken = 6m 41s | train_loss=1.3580, val_loss(ave)=3.1043
        iter = 21200(78%) | train_acc_thru_time_ave=6m 41s, val_acc_thru_time_ave=0.35800
        iter = 21300(78%) | time taken = 6m 43s | train_loss=1.6205, val_loss(ave)=3.0578
        iter = 21300(78%) | train_acc_thru_time_ave=6m 43s, val_acc_thru_time_ave=0.37499
        iter = 21400(79%) | time taken = 6m 45s | train_loss=0.9315, val_loss(ave)=2.9917
        iter = 21400(79%) | train_acc_thru_time_ave=6m 45s, val_acc_thru_time_ave=0.37599
        iter = 21500(79%) | time taken = 6m 46s | train_loss=2.3089, val_loss(ave)=2.5668
        iter = 21500(79%) | train_acc_thru_time_ave=6m 46s, val_acc_thru_time_ave=0.37
        iter = 21600(80%) | time taken = 6m 48s | train_loss=1.5007, val_loss(ave)=2.4439
        iter = 21600(80%) | train_acc_thru_time_ave=6m 48s, val_acc_thru_time_ave=0.39899
        iter = 21700(80%) | time taken = 6m 50s | train_loss=1.8719, val_loss(ave)=2.6746
        iter = 21700(80%) | train_acc_thru_time_ave=6m 50s, val_acc_thru_time_ave=0.34699
        iter = 21800(80%) | time taken = 6m 52s | train_loss=1.4589, val_loss(ave)=2.1134
        iter = 21800(80%) | train_acc_thru_time_ave=6m 52s, val_acc_thru_time_ave=0.36999
        iter = 21900(81%) | time taken = 6m 54s | train_loss=1.7356, val_loss(ave)=2.6316
        iter = 21900(81%) | train_acc_thru_time_ave=6m 54s, val_acc_thru_time_ave=0.384
        iter = 22000(81%) | time taken = 6m 56s | train_loss=1.0889, val_loss(ave)=3.3154
        iter = 22000(81%) | train_acc_thru_time_ave=6m 56s, val_acc_thru_time_ave=0.37999
        iter = 22100(81%) | time taken = 6m 57s | train_loss=0.9760, val_loss(ave)=2.3220
        iter = 22100(81%) | train_acc_thru_time_ave=6m 57s, val_acc_thru_time_ave=0.333
        iter = 22200(82%) | time taken = 6m 59s | train_loss=1.9148, val_loss(ave)=2.5772
        iter = 22200(82%) | train_acc_thru_time_ave=6m 59s, val_acc_thru_time_ave=0.37199
        iter = 22300(82%) | time taken = 7m 1s | train_loss=2.1747, val_loss(ave)=2.7972
        iter = 22300(82%) | train_acc_thru_time_ave=7m 1s, val_acc_thru_time_ave=0.36
        iter = 22400(82%) | time taken = 7m 3s | train_loss=1.6137, val_loss(ave)=3.5009
        iter = 22400(82%) | train_acc_thru_time_ave=7m 3s, val_acc_thru_time_ave=0.352999
        iter = 22500(83%) | time taken = 7m 4s | train_loss=1.8429, val_loss(ave)=2.4811
        iter = 22500(83%) | train_acc_thru_time_ave=7m 4s, val_acc_thru_time_ave=0.379000
        iter = 22600(83%) | time taken = 7m 6s | train_loss=1.7330, val_loss(ave)=2.5446
        iter = 22600(83%) | train_acc_thru_time_ave=7m 6s, val_acc_thru_time_ave=0.369999
        iter = 22700(84%) | time taken = 7m 8s | train_loss=1.8996, val_loss(ave)=2.8270
        iter = 22700(84%) | train_acc_thru_time_ave=7m 8s, val_acc_thru_time_ave=0.371000
        iter = 22800(84%) | time taken = 7m 10s | train_loss=1.6847, val_loss(ave)=3.1092
        iter = 22800(84%) | train_acc_thru_time_ave=7m 10s, val_acc_thru_time_ave=0.37999
        iter = 22900(84%) | time taken = 7m 12s | train_loss=0.7048, val_loss(ave)=3.0991
        iter = 22900(84%) | train_acc_thru_time_ave=7m 12s, val_acc_thru_time_ave=0.345
        iter = 23000(85%) | time taken = 7m 13s | train_loss=0.8224, val_loss(ave)=2.9675
        iter = 23000(85%) | train_acc_thru_time_ave=7m 13s, val_acc_thru_time_ave=0.38099
        iter = 23100(85%) | time taken = 7m 15s | train_loss=2.8651, val_loss(ave)=2.7467
        iter = 23100(85%) | train_acc_thru_time_ave=7m 15s, val_acc_thru_time_ave=0.37899
        iter = 23200(85%) | time taken = 7m 17s | train_loss=2.5489, val_loss(ave)=2.8095
        iter = 23200(85%) | train_acc_thru_time_ave=7m 17s, val_acc_thru_time_ave=0.35200
        iter = 23300(86%) | time taken = 7m 19s | train_loss=0.7354, val_loss(ave)=2.2656
        iter = 23300(86%) | train_acc_thru_time_ave=7m 19s, val_acc_thru_time_ave=0.37499
        iter = 23400(86%) | time taken = 7m 21s | train_loss=0.5311, val_loss(ave)=2.6708
        iter = 23400(86%) | train_acc_thru_time_ave=7m 21s, val_acc_thru_time_ave=0.37099
        iter = 23500(87%) | time taken = 7m 23s | train_loss=3.0536, val_loss(ave)=2.6790
        iter = 23500(87%) | train_acc_thru_time_ave=7m 23s, val_acc_thru_time_ave=0.37399
        iter = 23600(87%) | time taken = 7m 25s | train_loss=1.8817, val_loss(ave)=2.1731
        iter = 23600(87%) | train_acc_thru_time_ave=7m 25s, val_acc_thru_time_ave=0.38399
        iter = 23700(87%) | time taken = 7m 26s | train_loss=0.2969, val_loss(ave)=2.6987
        iter = 23700(87%) | train_acc_thru_time_ave=7m 26s, val_acc_thru_time_ave=0.37899
        iter = 23800(88%) | time taken = 7m 28s | train_loss=2.1149, val_loss(ave)=2.5141
```

```
    iter = 23800(88%) | train_acc_thru_time_ave=7m 28s, val_acc_thru_time_ave=0.37799
    iter = 23900(88%) | time taken = 7m 30s | train_loss=2.4242, val_loss(ave)=2.3827
    iter = 23900(88%) | train_acc_thru_time_ave=7m 30s, val_acc_thru_time_ave=0.37099
    iter = 24000(88%) | time taken = 7m 32s | train_loss=2.3396, val_loss(ave)=2.6043
    iter = 24000(88%) | train_acc_thru_time_ave=7m 32s, val_acc_thru_time_ave=0.37400
    iter = 24100(89%) | time taken = 7m 34s | train_loss=1.8689, val_loss(ave)=2.7602
    iter = 24100(89%) | train_acc_thru_time_ave=7m 34s, val_acc_thru_time_ave=0.38399
    iter = 24200(89%) | time taken = 7m 35s | train_loss=2.1569, val_loss(ave)=2.6754
    iter = 24200(89%) | train_acc_thru_time_ave=7m 35s, val_acc_thru_time_ave=0.38899
    iter = 24300(90%) | time taken = 7m 37s | train_loss=1.1251, val_loss(ave)=2.6127
    iter = 24300(90%) | train_acc_thru_time_ave=7m 37s, val_acc_thru_time_ave=0.37799
    iter = 24400(90%) | time taken = 7m 39s | train_loss=2.1828, val_loss(ave)=2.1801
    iter = 24400(90%) | train_acc_thru_time_ave=7m 39s, val_acc_thru_time_ave=0.41
    iter = 24500(90%) | time taken = 7m 41s | train_loss=1.6761, val_loss(ave)=2.6082
    iter = 24500(90%) | train_acc_thru_time_ave=7m 41s, val_acc_thru_time_ave=0.37799
    iter = 24600(91%) | time taken = 7m 43s | train_loss=1.0393, val_loss(ave)=2.6825
    iter = 24600(91%) | train_acc_thru_time_ave=7m 43s, val_acc_thru_time_ave=0.41099
    iter = 24700(91%) | time taken = 7m 45s | train_loss=2.0749, val_loss(ave)=2.2552
    iter = 24700(91%) | train_acc_thru_time_ave=7m 45s, val_acc_thru_time_ave=0.40299
    iter = 24800(91%) | time taken = 7m 46s | train_loss=0.7032, val_loss(ave)=2.3686
    iter = 24800(91%) | train_acc_thru_time_ave=7m 46s, val_acc_thru_time_ave=0.40499
    iter = 24900(92%) | time taken = 7m 48s | train_loss=1.5661, val_loss(ave)=2.6037
    iter = 24900(92%) | train_acc_thru_time_ave=7m 48s, val_acc_thru_time_ave=0.33099
    iter = 25000(92%) | time taken = 7m 50s | train_loss=2.2296, val_loss(ave)=2.8762
    iter = 25000(92%) | train_acc_thru_time_ave=7m 50s, val_acc_thru_time_ave=0.36300
    iter = 25100(92%) | time taken = 7m 52s | train_loss=1.1190, val_loss(ave)=2.1530
    iter = 25100(92%) | train_acc_thru_time_ave=7m 52s, val_acc_thru_time_ave=0.398
    iter = 25200(93%) | time taken = 7m 54s | train_loss=1.7067, val_loss(ave)=2.2141
    iter = 25200(93%) | train_acc_thru_time_ave=7m 54s, val_acc_thru_time_ave=0.36800
    iter = 25300(93%) | time taken = 7m 55s | train_loss=1.8808, val_loss(ave)=2.2143
    iter = 25300(93%) | train_acc_thru_time_ave=7m 55s, val_acc_thru_time_ave=0.35499
    iter = 25400(94%) | time taken = 7m 57s | train_loss=1.4532, val_loss(ave)=2.6144
    iter = 25400(94%) | train_acc_thru_time_ave=7m 57s, val_acc_thru_time_ave=0.40799
    iter = 25500(94%) | time taken = 7m 59s | train_loss=1.7864, val_loss(ave)=2.5152
    iter = 25500(94%) | train_acc_thru_time_ave=7m 59s, val_acc_thru_time_ave=0.37699
    iter = 25600(94%) | time taken = 8m 1s | train_loss=1.5119, val_loss(ave)=2.2826
    iter = 25600(94%) | train_acc_thru_time_ave=8m 1s, val_acc_thru_time_ave=0.360999
    iter = 25700(95%) | time taken = 8m 3s | train_loss=1.8001, val_loss(ave)=3.6471
    iter = 25700(95%) | train_acc_thru_time_ave=8m 3s, val_acc_thru_time_ave=0.404999
    iter = 25800(95%) | time taken = 8m 4s | train_loss=3.3583, val_loss(ave)=2.7407
    iter = 25800(95%) | train_acc_thru_time_ave=8m 4s, val_acc_thru_time_ave=0.404999
    iter = 25900(95%) | time taken = 8m 6s | train_loss=2.3825, val_loss(ave)=2.5842
    iter = 25900(95%) | train_acc_thru_time_ave=8m 6s, val_acc_thru_time_ave=0.414
    iter = 26000(96%) | time taken = 8m 8s | train_loss=2.0782, val_loss(ave)=2.8047
    iter = 26000(96%) | train_acc_thru_time_ave=8m 8s, val_acc_thru_time_ave=0.395
    iter = 26100(96%) | time taken = 8m 10s | train_loss=1.4913, val_loss(ave)=2.8816
    iter = 26100(96%) | train_acc_thru_time_ave=8m 10s, val_acc_thru_time_ave=0.40299
    iter = 26200(97%) | time taken = 8m 12s | train_loss=1.3295, val_loss(ave)=2.2469
    iter = 26200(97%) | train_acc_thru_time_ave=8m 12s, val_acc_thru_time_ave=0.38499
    iter = 26300(97%) | time taken = 8m 13s | train_loss=1.5638, val_loss(ave)=2.4293
    iter = 26300(97%) | train_acc_thru_time_ave=8m 13s, val_acc_thru_time_ave=0.33900
    iter = 26400(97%) | time taken = 8m 15s | train_loss=0.8817, val_loss(ave)=2.2633
    iter = 26400(97%) | train_acc_thru_time_ave=8m 15s, val_acc_thru_time_ave=0.39299
    iter = 26500(98%) | time taken = 8m 17s | train_loss=1.4422, val_loss(ave)=3.1558
    iter = 26500(98%) | train_acc_thru_time_ave=8m 17s, val_acc_thru_time_ave=0.37499
    iter = 26600(98%) | time taken = 8m 19s | train_loss=2.4650, val_loss(ave)=2.7682
    iter = 26600(98%) | train_acc_thru_time_ave=8m 19s, val_acc_thru_time_ave=0.37499
    iter = 26700(98%) | time taken = 8m 21s | train_loss=3.6292, val_loss(ave)=2.9509
```

```
iter = 26700(98%) | time taken = 8m 21s | train_loss=3.6292, val_loss(ave)=2.950
iter = 26700(98%) | train_acc_thru_time_ave=8m 21s, val_acc_thru_time_ave=0.392
iter = 26800(99%) | time taken = 8m 22s | train_loss=0.5329, val_loss(ave)=3.154
iter = 26800(99%) | train_acc_thru_time_ave=8m 22s, val_acc_thru_time_ave=0.3809
iter = 26900(99%) | time taken = 8m 24s | train_loss=1.6085, val_loss(ave)=2.851
iter = 26900(99%) | train_acc_thru_time_ave=8m 24s, val_acc_thru_time_ave=0.3959
```