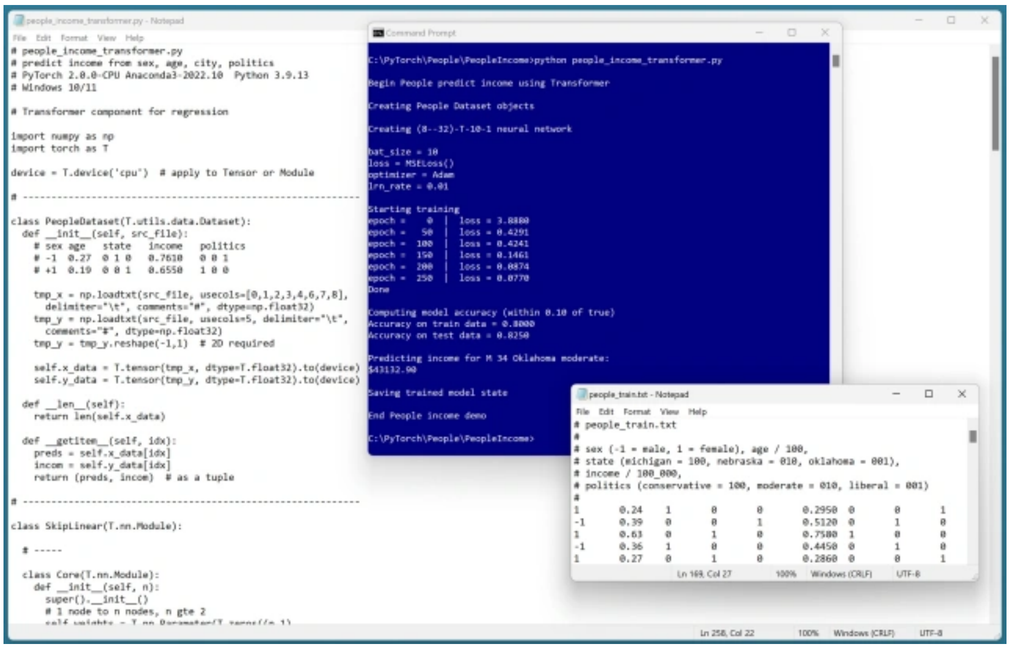
James D. McCaffrey

Software Research, Development, Testing, and Education

Regression Using a PyTorch Neural Network with a Transformer Component

Posted on December 1, 2023 by jamesdmccaffrey

Transformers are software modules that are the basis of large language models (GPT-4, Orca-2, LLaMA-2, etc.) A regression problem is one that predicts a single numeric value. Neural network regression systems are well-studied, but what happens if a Transformer is inserted into a regression system?

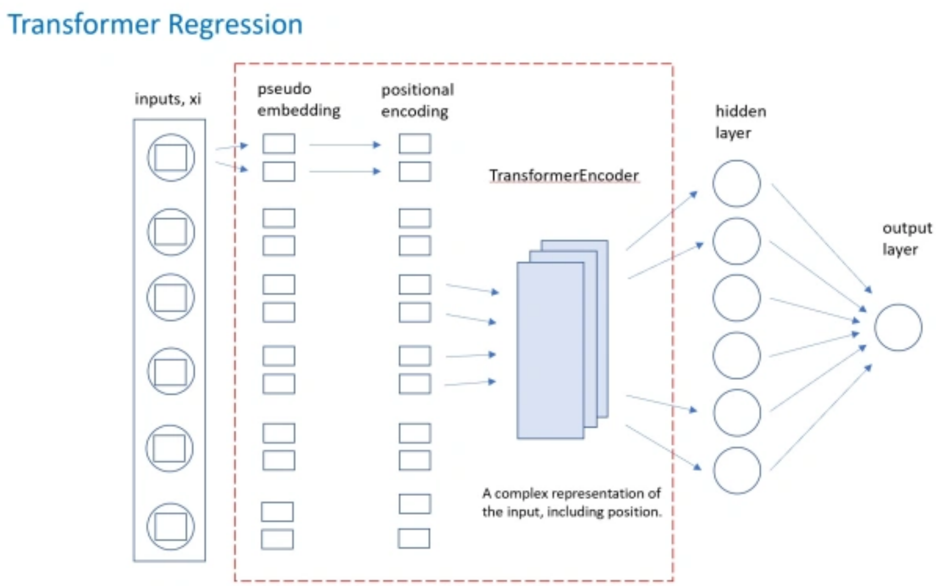


I put together a demo using one of my standard synthetic datasets. The goal is to predict a person's income (divided by $100,000) from sex (male = -1, female = +1), age (divided by 100), State (Michigan = 100, Nebraska = 010, Oklahoma = 001), and political leaning (conservative = 100, moderate – 010, liberal = 001). The tab-delimited data looks like:

```
 1  0.24  1  0  0  0.2950  0  0  1
-1  0.39  0  0  1  0.5120  0  1  0
 1  0.63  0  1  0  0.7580  1  0  0
-1  0.36  1  0  0  0.4450  0  1  0
 1  0.27  0  1  0  0.2860  0  0  1
. . .
```

There are 200 training items and 40 test items. The technique works but it's not clear if adding a Transformer to a regression system improves the system enough to warrant the added complexity.

For natural language problems, a Transformer component expects data in a very specific format. Each input sentence is broken into integer tokens, each token is projected to an embedding vector, and each embedding vector is augmented with positional encoding. Therefore, my biggest challenge was to create a custom PyTorch layer module that transforms my input into a form that can be consumed by the Transformer component. I did so by implementing a custom SkipLinear module (I'm not sure why I used such a weird name).



From a practical point of view, the major downside to adding a Transformer to a regression system is dealing with the five additional hyperparameters: pseudo-embedding size, positional dropout rate, number heads, internal feedforward size , number Transformer layer. I suspect that a regression system that incorporates a Transformer is most likely useful for very difficult problems with a very large dataset and a large number of predictor variables.

Neural systems have an element of randomness due to the random initialization of weights and biases. Here are three interesting sets of dice. Left: A pair of 7-sided dice. This design has generated a lot of commentary. I'm highly skeptical the dice are completely fair but they're probably good enough for board games where money isn't involved. Center: Skewed dice. Each side has the same shape so the dice are fair, subject to manufacturing tolerances. Right: Designer AKO ("another kind of dice") dice. Experiments show the dice are close to, but not completely, fair.

Demo code. Replace "lt" with Boolean less-than operator symbol. (My lame blog editor often chokes on symbols).

```
# people_income_transformer.py
```

```
# predict income from sex, age, city, politics
# PyTorch 2.0.0-CPU Anaconda3-2022.10  Python 3.9.13
# Windows 10/11

# Transformer component for regression

import numpy as np
import torch as T

device = T.device('cpu')  # apply to Tensor or Module

# -----------------------------------------------------------

class PeopleDataset(T.utils.data.Dataset):
  def __init__(self, src_file):
    # sex age   state   income   politics
    # -1  0.27  0 1 0   0.7610   0 0 1
    # +1  0.19  0 0 1   0.6550   1 0 0

    tmp_x = np.loadtxt(src_file, usecols=[0,1,2,3,4,6,7,8],
      delimiter="\t", comments="#", dtype=np.float32)
    tmp_y = np.loadtxt(src_file, usecols=5, delimiter="\t",
      comments="#", dtype=np.float32)
    tmp_y = tmp_y.reshape(-1,1)  # 2D required

    self.x_data = T.tensor(tmp_x, dtype=T.float32).to(device)
    self.y_data = T.tensor(tmp_y, dtype=T.float32).to(device)

  def __len__(self):
    return len(self.x_data)

  def __getitem__(self, idx):
    preds = self.x_data[idx]
    incom = self.y_data[idx]
    return (preds, incom)  # as a tuple

# -----------------------------------------------------------

class SkipLinear(T.nn.Module):

  # -----

  class Core(T.nn.Module):
    def __init__(self, n):
      super().__init__()
      # 1 node to n nodes, n gte 2
      self.weights = T.nn.Parameter(T.zeros((n,1),
        dtype=T.float32))
      self.biases = T.nn.Parameter(T.tensor(n,
        dtype=T.float32))
      lim = 0.01
      T.nn.init.uniform_(self.weights, -lim, lim)
      T.nn.init.zeros_(self.biases)

    def forward(self, x):
      wx= T.mm(x, self.weights.t())
      v = T.add(wx, self.biases)
      return v
```

```
  # -----

  def __init__(self, n_in, n_out):
    super().__init__()
    self.n_in = n_in; self.n_out = n_out
    if n_out  % n_in != 0:
      print("FATAL: n_out must be divisible by n_in")
    n = n_out // n_in  # num nodes per input

    self.lst_modules = \
      T.nn.ModuleList([SkipLinear.Core(n) for \
        i in range(n_in)])

  def forward(self, x):
    lst_nodes = []
    for i in range(self.n_in):
      xi = x[:,i].reshape(-1,1)
      oupt = self.lst_modules[i](xi)
      lst_nodes.append(oupt)
    result = T.cat((lst_nodes[0], lst_nodes[1]), 1)
    for i in range(2,self.n_in):
      result = T.cat((result, lst_nodes[i]), 1)
    result = result.reshape(-1, self.n_out)
    return result

# -----------------------------------------------------------

class PositionalEncoding(T.nn.Module):  # documentation code
  def __init__(self, d_model: int, dropout: float=0.1,
   max_len: int=5000):
    super(PositionalEncoding, self).__init__()  # old syntax
    self.dropout = T.nn.Dropout(p=dropout)
    pe = T.zeros(max_len, d_model)  # like 10x4
    position = \
      T.arange(0, max_len, dtype=T.float).unsqueeze(1)
    div_term = T.exp(T.arange(0, d_model, 2).float() * \
      (-np.log(10_000.0) / d_model))
    pe[:, 0::2] = T.sin(position * div_term)
    pe[:, 1::2] = T.cos(position * div_term)
    pe = pe.unsqueeze(0).transpose(0, 1)
    self.register_buffer('pe', pe)  # allows state-save

  def forward(self, x):
    x = x + self.pe[:x.size(0), :]
    return self.dropout(x)

# -----------------------------------------------------------

class TransformerNet(T.nn.Module):
  def __init__(self):
    super(TransformerNet, self).__init__()
    self.embed = SkipLinear(8, 32)  # 8 inputs, each goes to 4
    self.pos_enc = \
      PositionalEncoding(4, dropout=0.20)  # positional
    self.enc_layer = T.nn.TransformerEncoderLayer(d_model=4,
      nhead=2, dim_feedforward=10,
      batch_first=True)  # d_model divisible by nhead
    self.trans_enc = T.nn.TransformerEncoder(self.enc_layer,
      num_layers=2)  # 6 layers default
```

```
    self.fc1 = T.nn.Linear(32, 10)  # 8--32-T-10-1
    self.fc2 = T.nn.Linear(10, 1)

    # default weight and bias initialization

  def forward(self, x):
    z = self.embed(x)  # 8 inpts to 32 embed
    z = z.reshape(-1, 8, 4)  # bat seq embed
    z = self.pos_enc(z)
    z = self.trans_enc(z)
    z = z.reshape(-1, 32)  # torch.Size([bs, xxx])
    z = T.tanh(self.fc1(z))
    z = self.fc2(z)  # regression: no activation
    return z

# -----------------------------------------------------------

def accuracy(model, ds, pct_close):
  # assumes model.eval()
  # correct within pct of true income
  n_correct = 0; n_wrong = 0

  for i in range(len(ds)):
    X = ds[i][0].reshape(1,-1)  # make it a batch
    Y = ds[i][1].reshape(1)
    with T.no_grad():
      oupt = model(X)         # computed income

    if T.abs(oupt - Y) "lt" T.abs(pct_close * Y):
      n_correct += 1
    else:
      n_wrong += 1
  acc = (n_correct * 1.0) / (n_correct + n_wrong)
  return acc

# -----------------------------------------------------------

def accuracy_x(model, ds, pct_close):
  # all-at-once (quick)
  # assumes model.eval()
  X = ds.x_data  # all inputs
  Y = ds.y_data  # all targets
  n_items = len(X)
  with T.no_grad():
    pred = model(X)  # all predicted incomes

  n_correct = T.sum((T.abs(pred - Y) "lt" \
    T.abs(pct_close * Y)))
  result = (n_correct.item() / n_items)  # scalar
  return result

# -----------------------------------------------------------

def train(model, ds, bs, lr, me, le, test_ds):
  # dataset, bat_size, lrn_rate, max_epochs, log interval
  train_ldr = T.utils.data.DataLoader(ds, batch_size=bs,
    shuffle=True)
  loss_func = T.nn.MSELoss()
```

```
  optimizer = T.optim.Adam(model.parameters(), lr=lr)

  for epoch in range(0, me):
    epoch_loss = 0.0  # for one full epoch
    for (b_idx, batch) in enumerate(train_ldr):
      X = batch[0]  # predictors
      y = batch[1]  # target income
      optimizer.zero_grad()
      oupt = model(X)
      loss_val = loss_func(oupt, y)  # a tensor
      epoch_loss += loss_val.item()  # accumulate
      loss_val.backward()  # compute gradients
      optimizer.step()     # update weights

    if epoch % le == 0:
      print("epoch = %4d  |  loss = %0.4f" % \
        (epoch, epoch_loss))
      # model.eval()
      # print("-------------")
      # acc_train = accuracy(model, ds, 0.10)
      # print("Accuracy on train data = %0.4f" % acc_train)
      # acc_test = accuracy(model, test_ds, 0.10)
      # print("Accuracy on test data = %0.4f" % acc_test)
      # model.train()
      # print("-------------")

# -----------------------------------------------------------

def main():
  # 0. get started
  print("\nBegin People predict income using Transformer ")
  T.manual_seed(0)
  np.random.seed(0)

  # 1. create Dataset objects
  print("\nCreating People Dataset objects ")
  train_file = ".\\Data\\people_train.txt"
  train_ds = PeopleDataset(train_file)  # 200 rows

  test_file = ".\\Data\\people_test.txt"
  test_ds = PeopleDataset(test_file)  # 40 rows

  # 2. create network
  print("\nCreating (8--32)-T-10-1 neural network ")
  net = TransformerNet().to(device)

# -----------------------------------------------------------

  # 3. train model
  print("\nbat_size = 10 ")
  print("loss = MSELoss() ")
  print("optimizer = Adam ")
  print("lrn_rate = 0.01 ")

  print("\nStarting training")
  net.train()
  train(net, train_ds, bs=10, lr=0.01, me=300,
    le=50, test_ds=test_ds)
  print("Done ")
```

```
# -----------------------------------------------------------

  # 4. evaluate model accuracy
  print("\nComputing model accuracy (within 0.10 of true) ")
  net.eval()
  acc_train = accuracy(net, train_ds, 0.10)  # item-by-item
  print("Accuracy on train data = %0.4f" % acc_train)

  acc_test = accuracy_x(net, test_ds, 0.10)  # all-at-once
  print("Accuracy on test data = %0.4f" % acc_test)

# -----------------------------------------------------------

  # 5. make a prediction
  print("\nPredicting income for M 34 Oklahoma moderate: ")
  x = np.array([[-1, 0.34, 0,0,1,  0,1,0]],
    dtype=np.float32)
  x = T.tensor(x, dtype=T.float32).to(device)

  with T.no_grad():
    pred_inc = net(x)
  pred_inc = pred_inc.item()  # scalar
  print("$%0.2f" % (pred_inc * 100_000))  # un-normalized

# -----------------------------------------------------------

  # 6. save model (state_dict approach)
  print("\nSaving trained model state")
  fn = ".\\Models\\people_income_model.pt"
  T.save(net.state_dict(), fn)

  # model = Net()
  # model.load_state_dict(T.load(fn))
  # use model to make prediction(s)

  print("\nEnd People income demo ")

if __name__ == "__main__":
  main()
```

Training data. Tab-delimited. Replace space-space with tab or comma.

```
# people_train.txt
#
# sex (-1 = male, 1 = female), age / 100,
# state (michigan = 100, nebraska = 010, oklahoma = 001),
# income / 100_000,
# politics (conservative = 100, moderate = 010, liberal = 001)
#
1  0.24  1  0  0  0.2950  0  0  1
-1  0.39  0  0  1  0.5120  0  1  0
1  0.63  0  1  0  0.7580  1  0  0
-1  0.36  1  0  0  0.4450  0  1  0
1  0.27  0  1  0  0.2860  0  0  1
1  0.50  0  1  0  0.5650  0  1  0
```

```
1  0.50  0  0  1  0.5500  0  1  0
-1  0.19  0  0  1  0.3270  1  0  0
1  0.22  0  1  0  0.2770  0  1  0
-1  0.39  0  0  1  0.4710  0  0  1
1  0.34  1  0  0  0.3940  0  1  0
-1  0.22  1  0  0  0.3350  1  0  0
1  0.35  0  0  1  0.3520  0  0  1
-1  0.33  0  1  0  0.4640  0  1  0
1  0.45  0  1  0  0.5410  0  1  0
1  0.42  0  1  0  0.5070  0  1  0
-1  0.33  0  1  0  0.4680  0  1  0
1  0.25  0  0  1  0.3000  0  1  0
-1  0.31  0  1  0  0.4640  1  0  0
1  0.27  1  0  0  0.3250  0  0  1
1  0.48  1  0  0  0.5400  0  1  0
-1  0.64  0  1  0  0.7130  0  0  1
1  0.61  0  1  0  0.7240  1  0  0
1  0.54  0  0  1  0.6100  1  0  0
1  0.29  1  0  0  0.3630  1  0  0
1  0.50  0  0  1  0.5500  0  1  0
1  0.55  0  0  1  0.6250  1  0  0
1  0.40  1  0  0  0.5240  1  0  0
1  0.22  1  0  0  0.2360  0  0  1
1  0.68  0  1  0  0.7840  1  0  0
-1  0.60  1  0  0  0.7170  0  0  1
-1  0.34  0  0  1  0.4650  0  1  0
-1  0.25  0  0  1  0.3710  1  0  0
-1  0.31  0  1  0  0.4890  0  1  0
1  0.43  0  0  1  0.4800  0  1  0
1  0.58  0  1  0  0.6540  0  0  1
-1  0.55  0  1  0  0.6070  0  0  1
-1  0.43  0  1  0  0.5110  0  1  0
-1  0.43  0  0  1  0.5320  0  1  0
-1  0.21  1  0  0  0.3720  1  0  0
1  0.55  0  0  1  0.6460  1  0  0
1  0.64  0  1  0  0.7480  1  0  0
-1  0.41  1  0  0  0.5880  0  1  0
1  0.64  0  0  1  0.7270  1  0  0
-1  0.56  0  0  1  0.6660  0  0  1
1  0.31  0  0  1  0.3600  0  1  0
-1  0.65  0  0  1  0.7010  0  0  1
1  0.55  0  0  1  0.6430  1  0  0
-1  0.25  1  0  0  0.4030  1  0  0
1  0.46  0  0  1  0.5100  0  1  0
-1  0.36  1  0  0  0.5350  1  0  0
1  0.52  0  1  0  0.5810  0  1  0
1  0.61  0  0  1  0.6790  1  0  0
1  0.57  0  0  1  0.6570  1  0  0
-1  0.46  0  1  0  0.5260  0  1  0
-1  0.62  1  0  0  0.6680  0  0  1
1  0.55  0  0  1  0.6270  1  0  0
-1  0.22  0  0  1  0.2770  0  1  0
-1  0.50  1  0  0  0.6290  1  0  0
-1  0.32  0  1  0  0.4180  0  1  0
-1  0.21  0  0  1  0.3560  1  0  0
1  0.44  0  1  0  0.5200  0  1  0
1  0.46  0  1  0  0.5170  0  1  0
1  0.62  0  1  0  0.6970  1  0  0
1  0.57  0  1  0  0.6640  1  0  0
```

```
-1  0.67  0  0  1  0.7580  0  0  1
1  0.29  1  0  0  0.3430  0  0  1
1  0.53  1  0  0  0.6010  1  0  0
-1  0.44  1  0  0  0.5480  0  1  0
1  0.46  0  1  0  0.5230  0  1  0
-1  0.20  0  1  0  0.3010  0  1  0
-1  0.38  1  0  0  0.5350  0  1  0
1  0.50  0  1  0  0.5860  0  1  0
1  0.33  0  1  0  0.4250  0  1  0
-1  0.33  0  1  0  0.3930  0  1  0
1  0.26  0  1  0  0.4040  1  0  0
1  0.58  1  0  0  0.7070  1  0  0
1  0.43  0  0  1  0.4800  0  1  0
-1  0.46  1  0  0  0.6440  1  0  0
1  0.60  1  0  0  0.7170  1  0  0
-1  0.42  1  0  0  0.4890  0  1  0
-1  0.56  0  0  1  0.5640  0  0  1
-1  0.62  0  1  0  0.6630  0  0  1
-1  0.50  1  0  0  0.6480  0  1  0
1  0.47  0  0  1  0.5200  0  1  0
-1  0.67  0  1  0  0.8040  0  0  1
-1  0.40  0  0  1  0.5040  0  1  0
1  0.42  0  1  0  0.4840  0  1  0
1  0.64  1  0  0  0.7200  1  0  0
-1  0.47  1  0  0  0.5870  0  0  1
1  0.45  0  1  0  0.5280  0  1  0
-1  0.25  0  0  1  0.4090  1  0  0
1  0.38  1  0  0  0.4840  1  0  0
1  0.55  0  0  1  0.6000  0  1  0
-1  0.44  1  0  0  0.6060  0  1  0
1  0.33  1  0  0  0.4100  0  1  0
1  0.34  0  0  1  0.3900  0  1  0
1  0.27  0  1  0  0.3370  0  0  1
1  0.32  0  1  0  0.4070  0  1  0
1  0.42  0  0  1  0.4700  0  1  0
-1  0.24  0  0  1  0.4030  1  0  0
1  0.42  0  1  0  0.5030  0  1  0
1  0.25  0  0  1  0.2800  0  0  1
1  0.51  0  1  0  0.5800  0  1  0
-1  0.55  0  1  0  0.6350  0  0  1
1  0.44  1  0  0  0.4780  0  0  1
-1  0.18  1  0  0  0.3980  1  0  0
-1  0.67  0  1  0  0.7160  0  0  1
1  0.45  0  0  1  0.5000  0  1  0
1  0.48  1  0  0  0.5580  0  1  0
-1  0.25  0  1  0  0.3900  0  1  0
-1  0.67  1  0  0  0.7830  0  1  0
1  0.37  0  0  1  0.4200  0  1  0
-1  0.32  1  0  0  0.4270  0  1  0
1  0.48  1  0  0  0.5700  0  1  0
-1  0.66  0  0  1  0.7500  0  0  1
1  0.61  1  0  0  0.7000  1  0  0
-1  0.58  0  0  1  0.6890  0  1  0
1  0.19  1  0  0  0.2400  0  0  1
1  0.38  0  0  1  0.4300  0  1  0
-1  0.27  1  0  0  0.3640  0  1  0
1  0.42  1  0  0  0.4800  0  1  0
1  0.60  1  0  0  0.7130  1  0  0
-1  0.27  0  0  1  0.3480  1  0  0
```

1	0.29	0	1	0	0.3710	1	0	0
-1	0.43	1	0	0	0.5670	0	1	0
1	0.48	1	0	0	0.5670	0	1	0
1	0.27	0	0	1	0.2940	0	0	1
-1	0.44	1	0	0	0.5520	1	0	0
1	0.23	0	1	0	0.2630	0	0	1
-1	0.36	0	1	0	0.5300	0	0	1
1	0.64	0	0	1	0.7250	1	0	0
1	0.29	0	0	1	0.3000	0	0	1
-1	0.33	1	0	0	0.4930	0	1	0
-1	0.66	0	1	0	0.7500	0	0	1
-1	0.21	0	0	1	0.3430	1	0	0
1	0.27	1	0	0	0.3270	0	0	1
1	0.29	1	0	0	0.3180	0	0	1
-1	0.31	1	0	0	0.4860	0	1	0
1	0.36	0	0	1	0.4100	0	1	0
1	0.49	0	1	0	0.5570	0	1	0
-1	0.28	1	0	0	0.3840	1	0	0
-1	0.43	0	0	1	0.5660	0	1	0
-1	0.46	0	1	0	0.5880	0	1	0
1	0.57	1	0	0	0.6980	1	0	0
-1	0.52	0	0	1	0.5940	0	1	0
-1	0.31	0	0	1	0.4350	0	1	0
-1	0.55	1	0	0	0.6200	0	0	1
1	0.50	1	0	0	0.5640	0	1	0
1	0.48	0	1	0	0.5590	0	1	0
-1	0.22	0	0	1	0.3450	1	0	0
1	0.59	0	0	1	0.6670	1	0	0
1	0.34	1	0	0	0.4280	0	0	1
-1	0.64	1	0	0	0.7720	0	0	1
1	0.29	0	0	1	0.3350	0	0	1
-1	0.34	0	1	0	0.4320	0	1	0
-1	0.61	1	0	0	0.7500	0	0	1
1	0.64	0	0	1	0.7110	1	0	0
-1	0.29	1	0	0	0.4130	1	0	0
1	0.63	0	1	0	0.7060	1	0	0
-1	0.29	0	1	0	0.4000	1	0	0
-1	0.51	1	0	0	0.6270	0	1	0
-1	0.24	0	0	1	0.3770	1	0	0
1	0.48	0	1	0	0.5750	0	1	0
1	0.18	1	0	0	0.2740	1	0	0
1	0.18	1	0	0	0.2030	0	0	1
1	0.33	0	1	0	0.3820	0	0	1
-1	0.20	0	0	1	0.3480	1	0	0
1	0.29	0	0	1	0.3300	0	0	1
-1	0.44	0	0	1	0.6300	1	0	0
-1	0.65	0	0	1	0.8180	1	0	0
-1	0.56	1	0	0	0.6370	0	0	1
-1	0.52	0	0	1	0.5840	0	1	0
-1	0.29	0	1	0	0.4860	1	0	0
-1	0.47	0	1	0	0.5890	0	1	0
1	0.68	1	0	0	0.7260	0	0	1
1	0.31	0	0	1	0.3600	0	1	0
1	0.61	0	1	0	0.6250	0	0	1
1	0.19	0	1	0	0.2150	0	0	1
1	0.38	0	0	1	0.4300	0	1	0
-1	0.26	1	0	0	0.4230	1	0	0
1	0.61	0	1	0	0.6740	1	0	0
1	0.40	1	0	0	0.4650	0	1	0

```
-1  0.49  1  0  0  0.6520  0  1  0
1  0.56  1  0  0  0.6750  1  0  0
-1  0.48  0  1  0  0.6600  0  1  0
1  0.52  1  0  0  0.5630  0  0  1
-1  0.18  1  0  0  0.2980  1  0  0
-1  0.56  0  0  1  0.5930  0  0  1
-1  0.52  0  1  0  0.6440  0  1  0
-1  0.18  0  1  0  0.2860  0  1  0
-1  0.58  1  0  0  0.6620  0  0  1
-1  0.39  0  1  0  0.5510  0  1  0
-1  0.46  1  0  0  0.6290  0  1  0
-1  0.40  0  1  0  0.4620  0  1  0
-1  0.60  1  0  0  0.7270  0  0  1
1  0.36  0  1  0  0.4070  0  0  1
1  0.44  1  0  0  0.5230  0  1  0
1  0.28  1  0  0  0.3130  0  0  1
1  0.54  0  0  1  0.6260  1  0  0
```

Test data. Tab-delimited. Replace space-space with tab or comma.

```
# people_test.txt
#
-1  0.51  1  0  0  0.6120  0  1  0
-1  0.32  0  1  0  0.4610  0  1  0
1  0.55  1  0  0  0.6270  1  0  0
1  0.25  0  0  1  0.2620  0  0  1
1  0.33  0  0  1  0.3730  0  0  1
-1  0.29  0  1  0  0.4620  1  0  0
1  0.65  1  0  0  0.7270  1  0  0
-1  0.43  0  1  0  0.5140  0  1  0
-1  0.54  0  1  0  0.6480  0  0  1
1  0.61  0  1  0  0.7270  1  0  0
1  0.52  0  1  0  0.6360  1  0  0
1  0.3  0  1  0  0.3350  0  0  1
1  0.29  1  0  0  0.3140  0  0  1
-1  0.47  0  0  1  0.5940  0  1  0
1  0.39  0  1  0  0.4780  0  1  0
1  0.47  0  0  1  0.5200  0  1  0
-1  0.49  1  0  0  0.5860  0  1  0
-1  0.63  0  0  1  0.6740  0  0  1
-1  0.3  1  0  0  0.3920  1  0  0
-1  0.61  0  0  1  0.6960  0  0  1
-1  0.47  0  0  1  0.5870  0  1  0
1  0.3  0  0  1  0.3450  0  0  1
-1  0.51  0  0  1  0.5800  0  1  0
-1  0.24  1  0  0  0.3880  0  1  0
-1  0.49  1  0  0  0.6450  0  1  0
1  0.66  0  0  1  0.7450  1  0  0
-1  0.65  1  0  0  0.7690  1  0  0
-1  0.46  0  1  0  0.5800  1  0  0
-1  0.45  0  0  1  0.5180  0  1  0
-1  0.47  1  0  0  0.6360  1  0  0
-1  0.29  1  0  0  0.4480  1  0  0
-1  0.57  0  0  1  0.6930  0  0  1
-1  0.2  1  0  0  0.2870  0  0  1
-1  0.35  1  0  0  0.4340  0  1  0
```

```
-1  0.61  0  0  1  0.6700  0  0  1
-1  0.31  0  0  1  0.3730  0  1  0
1  0.18  1  0  0  0.2080  0  0  1
1  0.26  0  0  1  0.2920  0  0  1
-1  0.28  1  0  0  0.3640  0  0  1
-1  0.59  0  0  1  0.6940  0  0  1
```

This entry was posted in PyTorch. Bookmark the permalink.
