PYTORCH CHEAT SHEET

SZNAJDMANCONSULTING

Imports

General

import torch	root package
from torch.utils.data import Dataset, DataLoader	dataset representation and loading

Neural nets

import torch.autograd as autograd	computation graph
from torch.autograd import Variable	variable node in computation graph
import torch.nn as nn	neural networks
import torch.nn.functional as F	layers, activations and more
import torch.optim as optim	optimizers e.g. gradient desc, ADAM, etc

Vision

from torchvision import datasets, models, transforms	vision datasets, architectures & transforms
import torchvision.transforms as transforms	composable transforms

Parallell

import torch.distributed as dist	distributed comunication
from torch.multiprocessing import Process	memory sharing processes

Tensors

Creation

torch.randn(*size)	tensor with independent N(0,1) entries
torch.[ones|zeros](*size)	tensor with all 1's [or 0's]
torch.Tensor(L)	create tensor from [nested] list or ndarray L
x.clone()	clone of x

Dimensionality

x.size()	return tuple-like object of dimensions
torch.cat(tensor_seq, dim=0)	concatenates tensors along dim
x.view(a,b,...)	reshapes x into size (a,b,...)
x.view(-1,a)	reshapes x into size (b,a) for some b
x.transpose(a,b)	swaps dimensions a and b
x.permute(*dims)	permutes dimensions
x.unsqueeze(dim)	tensor with added axis
x.unsqueeze(dim=2)	(a,b,c) tensor -> (a,b,1,c) tensor

Algebra

A.mm(B)	matrix multiplication
A.mv(x)	matrix-vector multiplication
x.t()	matrix transpose

GPU

torch.cuda.is_available()	check for cuda
x.cuda()	move x's data from CPU to GPU and return new object
x.cpu()	move x's data from GPU to CPU and return new object

Deep Learning

Layers

nn.Linear(m,n)	fully connected layer from m to n units
nn.ConvXd(m, n, s)	X dimensional conv layer from m to n channels where X∈{1,2,3} and kernel size is s
nn.MaxPoolXd(s)	X dimensional pooling layer (notation as above)
nn.BatchNorm	batch norm layer
nn.RNN/LSTM/GRU	recurrent layers
nn.Dropout(p=0.5, inplace=False)	dropout layer for any dimensional input
nn.Dropout2d(p=0.5, inplace=False)	2-dimensional channel-wise dropout
nn.Embedding(num_embeddings, embedding_dim)	(tensor-wise) mapping from indices to embedding vectors

Loss functions

nn.X where for example X is ...	BCELoss, CrossEntropyLoss, L1Loss, MSELoss, NLLLoss, SoftMarginLoss, MultiLabelSoftMarginLoss, CosineEmbeddingLoss, KLDivLoss, MarginRankingLoss, HingeEmbeddingLoss or CosineEmbeddingLoss

Activation functions

nn.X where for example X is ...	ReLU, ReLU6, ELU, SELU, PReLU, LeakyReLU, Threshold, Hardtanh, Sigmoid, Tanh, LogSigmoid, Softplus, Softshrink, Softsign, TanhShrink, Softmin, Softmax, Softmax2d or LogSoftmax

Optimizers

opt = optim.X(model.parameters(), ...)	create optimizer
opt.step()	update weights
optim.X where for example X is ...	SGD, Adadelta, Adagrad, Adam, SparseAdam, Adamax, ASGD, LBFGS, RMSProp or Rprop

Learning rate scheduling

scheduler = optim.X(optimizer,...)	create lr scheduler
scheduler.step()	update lr at start of epoch
optim.lr_scheduler.X where ...	LambdaLR, StepLR, MultiStepLR, ExponentialLR or ReduceLROnPlateau

Data - torch.utils.data.X

Datasets

Dataset	abstract class representing data set
TensorDataset	labelled data set in the form of tensors
ConcatDataset	concatation of iterable of Datasets

DataLoaders and DataSamplers

DataLoader(dataset, batch_size=1, ...)	loads data batches agnostically of structure of individual data points
sampler.Sampler(dataset,...)	abstract class dealing with ways to sample from dataset
sampler.XSampler where ...	Sequential, Random, Subset, WeightedRandom or Distributed

TensorFlow™

TensorFlow is an open-source software library for high-performance numerical computation. Its flexible architecture enables to easily deploy computation across a variety of platforms (CPUs, GPUs, and TPUs), as well as mobile and edge devices, desktops, and clusters of servers. TensorFlow comes with strong support for machine learning and deep learning.

High-Level APIs for Deep Learning

Keras is a handy high-level API standard for deep learning models widely adopted for fast prototyping and state-of-the-art research. It was originally designed to run on top of different low-level computational frameworks and therefore the TensorFlow platform fully implements it.

The Sequential API is the most common way to define your neural network model. It corresponds to the mental image we use when thinking about deep learning: a sequence of layers.

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models

# Load data set
mnist = datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

# Construct a neural network model
model = models.Sequential()
model.add(layers.Flatten(input_shape=(28, 28)))
model.add(layers.Dense(512, activation=tf.nn.relu))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(10, activation=tf.nn.softmax))
model.compile(optimizer=’adam’,
              loss=’sparse_categorical_crossentropy’,
              metrics=[’accuracy’])

# Train and evaluate the model
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

The Functional API enables engineers to define complex topologies, including multi-input and multi-output models, as well as advanced models with shared layers and models with residual connections.

```
from tensorflow.keras.layers import Flatten, Dense, Dropout
from tensorflow.keras.models import Model

# Loading data set must be here <...>

inputs = tf.keras.Input(shape=(28, 28))
x = Flatten()(inputs)
x = Dense(512, activation=’relu’)(x)
x = Dropout(0.2)(x)
predictions = Dense(10, activation=’softmax’)(x)
model = Model(inputs=inputs, outputs=predictions)

# Compile, train and evaluate the model here <...>
```

A layer instance is called on a tensor and returns a tensor. An input tensor and output tensor can then be used to define a `Model`, which is compiled and trained just as a `Sequential` model. Models are callable by themselves and can be stacked the same way while reusing trained weights.

Transfer learning and fine-tuning of pretrained models saves your time if your data set does not differ significantly from the original one.

```
import tensorflow as tf
import tensorflow_datasets as tfds

dataset = tfds.load(name=’tf_flowers’, as_supervised=True)
NUMBER_OF_CLASSES_IN_DATASET = 5
IMG_SIZE = 160

def preprocess_example(image, label):
   image = tf.cast(image, tf.float32)
   image = (image / 127.5) - 1
   image = tf.image.resize(image, (IMG_SIZE, IMG_SIZE))
   return image, label

DATASET_SIZE = 3670
BATCH_SIZE = 32
train = dataset[’train’].map(preprocess_example)
train_batches = train.shuffle(DATASET_SIZE).batch(BATCH_SIZE)

# Load MobileNetV2 model pretrained on ImageNet data
model = tf.keras.applications.MobileNetV2(
    input_shape=(IMG_SIZE, IMG_SIZE, 3),
    include_top=False, weights=’imagenet’, pooling=’avg’)
model.trainable = False

# Add a new layer for multiclass classification
new_output = tf.keras.layers.Dense(
    NUMBER_OF_CLASSES_IN_DATASET, activation=’softmax’)
new_model = tf.keras.Sequential([model, new_output])
new_model.compile(
    loss=tf.keras.losses.categorical_crossentropy,
    optimizer=tf.keras.optimizers.RMSprop(lr=1e-3),
    metrics=[’accuracy’])

# Train the classification layer
new_model.fit(train_batches.repeat(), epochs=10,
              steps_per_epoch=DATASET_SIZE // BATCH_SIZE)
```

After the execution of the given transfer learning code, you can make MobileNetV2 layers trainable and perform fine-tuning of the resulting model to achieve better results.

Jupyter Notebook

Jupyter Notebook is a web-based interactive computational environment for data science and scientific computing.

Google Colaboratory is a free notebook environment that requires no setup and runs entirely in the cloud. Use it for jump-starting a machine learning project.

Python For Data Science *Cheat Sheet*

Keras

Learn Python for data science **Interactively** at www.DataCamp.com

Keras

Keras is a powerful and easy-to-use deep learning library for Theano and TensorFlow that provides a high-level neural networks API to develop and evaluate deep learning models.

A Basic Example

```
>>> import numpy as np
>>> from keras.models import Sequential
>>> from keras.layers import Dense
>>> data = np.random.random((1000,100))
>>> labels = np.random.randint(2,size=(1000,1))
>>> model = Sequential()
>>> model.add(Dense(32,
                    activation='relu',
                    input_dim=100))
>>> model.add(Dense(1, activation='sigmoid'))
>>> model.compile(optimizer='rmsprop',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
>>> model.fit(data,labels,epochs=10,batch_size=32)
>>> predictions = model.predict(data)
```

Data

Also see NumPy, Pandas & Scikit-Learn

Your data needs to be stored as NumPy arrays or as a list of NumPy arrays. Ideally, you split the data in training and test sets, for which you can also resort to the `train_test_split` module of `sklearn.cross_validation`.

Keras Data Sets

```
>>> from keras.datasets import boston_housing,
                               mnist,
                               cifar10,
                               imdb
>>> (x_train,y_train),(x_test,y_test) = mnist.load_data()
>>> (x_train2,y_train2),(x_test2,y_test2) = boston_housing.load_data()
>>> (x_train3,y_train3),(x_test3,y_test3) = cifar10.load_data()
>>> (x_train4,y_train4),(x_test4,y_test4) = imdb.load_data(num_words=20000)
>>> num_classes = 10
```

Other

```
>>> from urllib.request import urlopen
>>> data = np.loadtxt(urlopen("http://archive.ics.uci.edu/
ml/machine-learning-databases/pima-indians-diabetes/
pima-indians-diabetes.data"),delimiter=",")
>>> X = data[:,0:8]
>>> y = data [:,8]
```

stop
[: :] ⇒ per dimension
start step

Model Architecture

Sequential Model

```
>>> from keras.models import Sequential
>>> model = Sequential()
>>> model2 = Sequential()
>>> model3 = Sequential()
```

Multilayer Perceptron (MLP)

Binary Classification

```
>>> from keras.layers import Dense
>>> model.add(Dense(12,
                    input_dim=8,
                    kernel_initializer='uniform',
                    activation='relu'))
>>> model.add(Dense(8,kernel_initializer='uniform',activation='relu'))
>>> model.add(Dense(1,kernel_initializer='uniform',activation='sigmoid'))
```

Multi-Class Classification

```
>>> from keras.layers import Dropout
>>> model.add(Dense(512,activation='relu',input_shape=(784,)))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(512,activation='relu'))
>>> model.add(Dropout(0.2))
>>> model.add(Dense(10,activation='softmax'))
```

Regression

```
>>> model.add(Dense(64,activation='relu',input_dim=train_data.shape[1]))
>>> model.add(Dense(1))
```

Convolutional Neural Network (CNN)

```
>>> from keras.layers import Activation,Conv2D,MaxPooling2D,Flatten
>>> model2.add(Conv2D(32,(3,3),padding='same',input_shape=x_train.shape[1:]))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(32,(3,3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Conv2D(64,(3,3), padding='same'))
>>> model2.add(Activation('relu'))
>>> model2.add(Conv2D(64,(3, 3)))
>>> model2.add(Activation('relu'))
>>> model2.add(MaxPooling2D(pool_size=(2,2)))
>>> model2.add(Dropout(0.25))

>>> model2.add(Flatten())
>>> model2.add(Dense(512))
>>> model2.add(Activation('relu'))
>>> model2.add(Dropout(0.5))
>>> model2.add(Dense(num_classes))
>>> model2.add(Activation('softmax'))
```

Recurrent Neural Network (RNN)

```
>>> from keras.klayers import Embedding,LSTM
>>> model3.add(Embedding(20000,128))
>>> model3.add(LSTM(128,dropout=0.2,recurrent_dropout=0.2))
>>> model3.add(Dense(1,activation='sigmoid'))
```

Preprocessing

Also see NumPy & Scikit-Learn

Sequence Padding

```
>>> from keras.preprocessing import sequence
>>> x_train4 = sequence.pad_sequences(x_train4,maxlen=80)
>>> x_test4 = sequence.pad_sequences(x_test4,maxlen=80)
```

One-Hot Encoding

```
>>> from keras.utils import to_categorical
>>> Y_train = to_categorical(y_train, num_classes)
>>> Y_test = to_categorical(y_test, num_classes)
>>> Y_train3 = to_categorical(y_train3, num_classes)
>>> Y_test3 = to_categorical(y_test3, num_classes)
```

Train and Test Sets

```
>>> from sklearn.model_selection import train_test_split
>>> X_train5,X_test5,y_train5,y_test5 = train_test_split(X,
                                                         y,
                                                         test_size=0.33,
                                                         random_state=42)
```

Standardization/Normalization

```
>>> from sklearn.preprocessing import StandardScaler
>>> scaler = StandardScaler().fit(x_train2)
>>> standardized_X = scaler.transform(x_train2)
>>> standardized_X_test = scaler.transform(x_test2)
```

Inspect Model

`>>> model.output_shape`	Model output shape
`>>> model.summary()`	Model summary representation
`>>> model.get_config()`	Model configuration
`>>> model.get_weights()`	List all weight tensors in the model

Compile Model

MLP: Binary Classification

```
>>> model.compile(optimizer='adam',
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
```

MLP: Multi-Class Classification

```
>>> model.compile(optimizer='rmsprop',
                  loss='categorical_crossentropy',
                  metrics=['accuracy'])
```

MLP: Regression

```
>>> model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])
```

Recurrent Neural Network

```
>>> model3.compile(loss='binary_crossentropy',
                   optimizer='adam',
                   metrics=['accuracy'])
```

Model Training

```
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               verbose=1,
               validation_data=(x_test4,y_test4))
```

Evaluate Your Model's Performance

```
>>> score = model3.evaluate(x_test,
                            y_test,
                            batch_size=32)
```

Prediction

```
>>> model3.predict(x_test4, batch_size=32)
>>> model3.predict_classes(x_test4,batch_size=32)
```

Save/ Reload Models

```
>>> from keras.models import load_model
>>> model3.save('model_file.h5')
>>> my_model = load_model('my_model.h5')
```

Model Fine-tuning

Optimization Parameters

```
>>> from keras.optimizers import RMSprop
>>> opt = RMSprop(lr=0.0001, decay=1e-6)
>>> model2.compile(loss='categorical_crossentropy',
                   optimizer=opt,
                   metrics=['accuracy'])
```

Early Stopping

```
>>> from keras.callbacks import EarlyStopping
>>> early_stopping_monitor = EarlyStopping(patience=2)
>>> model3.fit(x_train4,
               y_train4,
               batch_size=32,
               epochs=15,
               validation_data=(x_test4,y_test4),
               callbacks=[early_stopping_monitor])
```