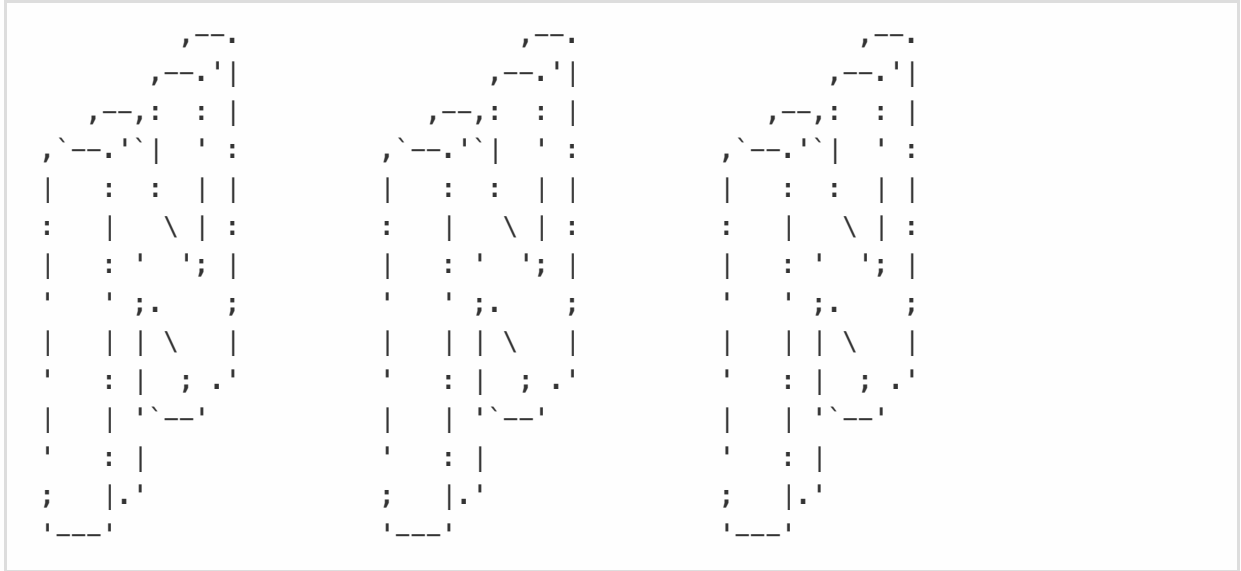


Numpy Neural Network



Numpy Neural Network(NNN) is neural network framework based on only [numpy](#) which is fundamental package for scientific computing with Python.

NNN support *activations, metrics, layers, optimizers* and *loss functions*. You can build network with high-level expressions like [keras](#) on [tensorflow](#) or [PyTorch](#).

Below code, create two-layer network using *sigmoid* as activation, solving problem given as assignment. You can use the same interface for *numpy* and *tensorflow*, depending on whether you define nn as `tf.keras` or `nnn`.

```
model = nn.Sequential([
    nn.layers.Dense(2, input_dim=2, activation='sigmoid'),
    nn.layers.Dense(1, input_dim=2, activation='sigmoid'),
    # more layers if you need like
    # nn.layers.Dense(2, input_dim=2, activation='relu'),
])
```

After compile you model, use `fit` and `evaluate` to train and test model.

```
model.fit(train.X, train.Y, epochs=100)
loss, acc = model.evaluate(test.X, test.Y)
```

Run!

There are several arguments for define task. You can run *numpy* and *tensorflow* implementations using the `--mode` flag as `np` or `tf`. Also you can set `epoch`, `size`, `lr`, `optimizer`, `loss`, `layer`, `repeat` and `seed` flags.

```
python main.py
--mode tf
--lr .1
--optimizer SGD
--loss binary_crossentropy
```

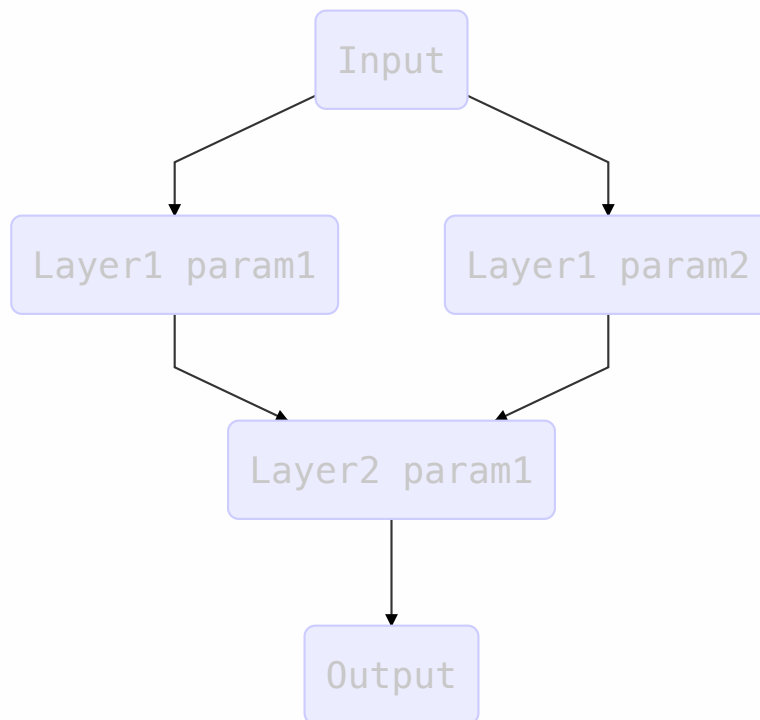
Also, You can find help and possible options by typing

```
python main.py --help
```

For dependencies, install package list on `requirements.txt`

```
pip install -r requirements.txt
```

Base Network Architecture



In this assignment, use SGD as optimizer, BCE as loss function and sigmoid as activation function. Each test performed on 128 dataset with 5000 epoch, learning rate 0.1. The random seed is fixed at 2.

Data

`utils.data.Dataset` generate a random sample from *standard normal* distribution using `np.random.randn`. Set normalized dataset with `--normal` flag.

`utils.data.Dataset` generate labels from X, Y where $X^2 > Y$. If you want custom dataset, override `utils.data.Dataset` and define X and Y .

Comparison Results

%	Base	LR[.5]	LR[.01]	Loss[MSE]	Data[Normal]	Data[256]	Data[2048]
TF	96.95	97.97	96.88	97.11	97.27	97.27	99.28
NNN	97.27	96.88	69.92	97.27	95.94	95.86	96.45

Implementation

The **Layer** must determine the size of *input dimension* and the size of the *output dimension* and also specify activation function. `forward` and `backward` is default method for network for forward propagation and backward propagation. Also **Layer** calls are replaced with `forward`. `after_forward` is method for keep *last output* and pass the output to *activation* if exists. `update` call optimizer to update parameters. `parameters` is getter and setter, which returns all *parameters* belongs to layer and update it. You can make custom layer by inheriting the **Layer**.

In **Dense(Layer)**, `forward` just dot product the *input value* by the *parameters* and add *bias*. `backward` calculate gradient of *parameters* and *bias* using last saved *output value* and passed *gradient*.

Activation works like a **Layer**, but basically it does not need an *initial parameters* such as *input dimension* or *output dimension*, so it can be called like a function even though it still performs forward and backward. Sigmoid and ReLU are implemented. You can make custom activation function by inheriting the **Activation**.

Optimizer update parameters by *learning rate* or other *hyper parameters*. SGD and Adam are implemented.

NNN support optimizers (from now, only support SGD and Adam) Loss (support MSE and BCE) and Metric(only accuracy now). You can easily define it as you do in *tensorflow*.

```
optimizer = nn.optimizers.sgd(lr=.1)
loss_function = nn.losses.MSE()
metric = nn.metrics.accuracy()
```

Save parameter retrival

Numpy support `np.ndarray` save method. `parameters.npz` is saved *paramters*.

```
parameters = np.load('parameters.npz')

for i, layer in enumerate(model.layers):
    layer.parameters = parameters[str(i)]
```

parameters	param1	param2	bias
Layer1-1	-5.4167	4.1429	1.1458
Layer1-2	-5.3711	4.1334	1.1175
Layer2	-6.4002	-6.3592	5.0729

```
base maybe ~/Workspace/ITE4053 master • python main.py --mode np --verbose --repeat 1
100%| 1/1 [00:02<00:00, 2.41s/it, loss=0.0861, mean=98.44%, score=98.44%]
100%| 5000/5000 [00:02<00:00, 2073.06it/s, loss=0.1502]

== Parameters
0: Dense [[-5.41671808  4.14299408  1.14582771]
 [-5.37105625  4.13338613  1.11745175]]
1: Dense [[-6.40020313 -6.35924756  5.07291324]]

base maybe ~/Workspace/ITE4053 master • 5s
```