Introduction to Keras: Theory and Examples

IIT PATNA

OUTLINE

- Introduction to Google Colab
- Keras
  - Introduction
  - Fully Connected Neural Network
    - Classification
    - Regression

Confession

- Introductory (Hello World)
- Internet (sources at the end)

Part 0: Google Colab

Introduction to Google Colab

- Product by Google
- Google's free cloud service with GPU support for AI developers
  - CPU $\Leftrightarrow$ GPU $\Leftrightarrow$ TPU
  - Python programming language
  - Support to many neural network libraries such as Keras, PyTorch, OpenCV
- Files are stored on Drive

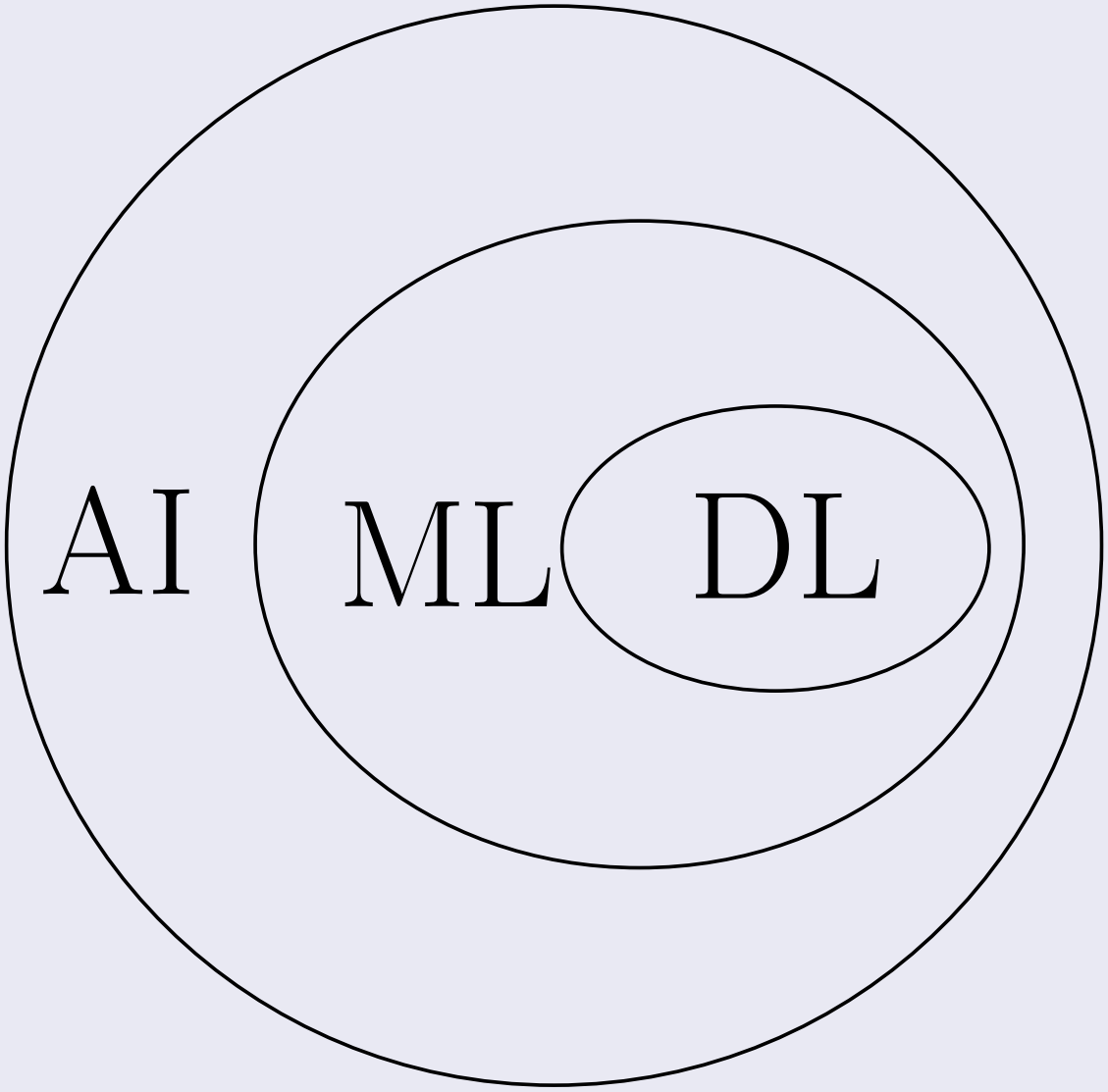
Introduction to Google Colab

- `https://github.com/`
  - nrjcs/mlip
- `https://colab.research.google.com/`

- Notebook: list of cells (code or text)
- can be shared
- collaborated
- GitHub
- Default folder is Colab Notebooks

Part I: Fully Connected Neural Network

Introduction



Architecture of a Neural Network

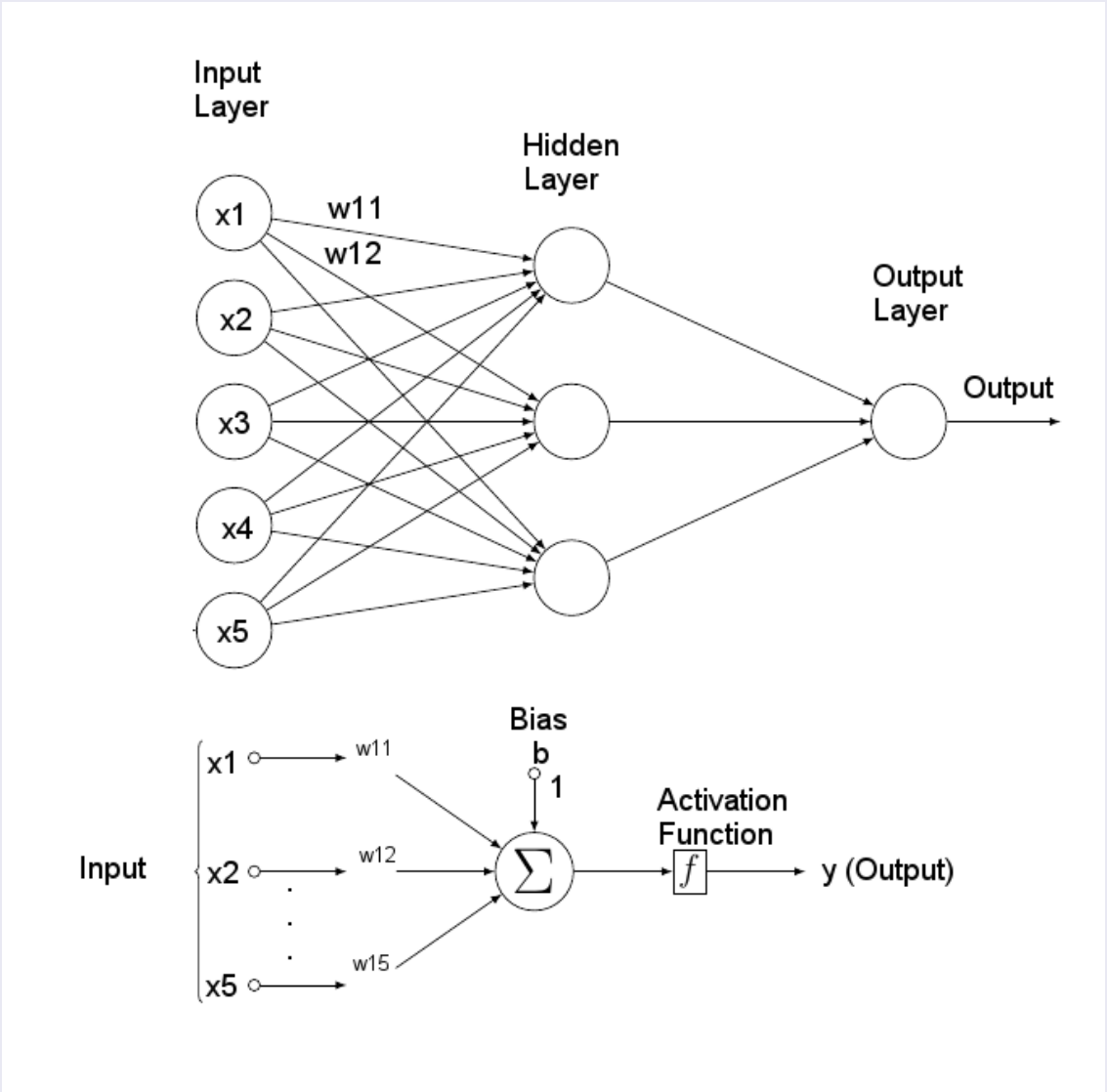


Figure: A Neural Network

Architecture of a Neural Network

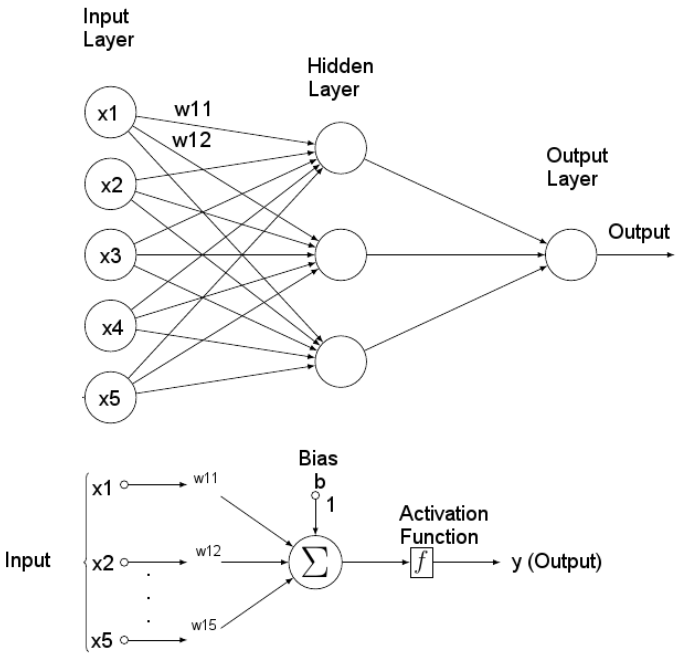


Figure: A Neural Network

Learning Steps (Decisions to be made):

1. Application (Problem)
2. Type of model
3. No. of layers
4. No. of nodes
5. Initialization of weights
6. Activation Function
7. Optimization Function
8. Evaluation Metrics
9. Dataset
10. Testing and Training Data
11. Batch size
12. Epoch

Keras

- NN: development (implementation and experimentation) is difficult.

Keras is

- high-level neural networks library
- written in Python
- capable of running on top of
  - TensorFlow (open source software library for numerical computation)
  - Theano (numerical computation library for Python)
  - CNTK (Microsoft Cognitive Toolkit): Deep learning framework
- developed with a focus on enabling fast experimentation (through user friendliness, modularity, and extensibility)
- and much more visit

Guiding principles

- Modularity
  - configurable modules
    - neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models
- Minimalism
  - Each module should be kept short and simple
- Easy extensibility
  - New modules are simple to add (as new classes and functions)
  - suitable for advanced research
- Work with Python
  - Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility
- User friendliness

Installation and Dependencies

- No worries
  - Google Colab
- You may visit Keras Installation Page @ keras.io

Keras Toolbox

What is in the toolbox ?

- Models
- Layers
- Preprocessing
- Metrics
- Optimizers
- Activations
- Datasets
- Constraints
- Initializers
- Loss (Objecitve) Function
- and many more...

Model

- Model
  - core data structure of Keras
  - a way to organize layers
- Two types:
  - Sequential
  - Model class API
- Sequential Model: a linear stack of layers
- functional API: for defining complex models, such as models with shared layers

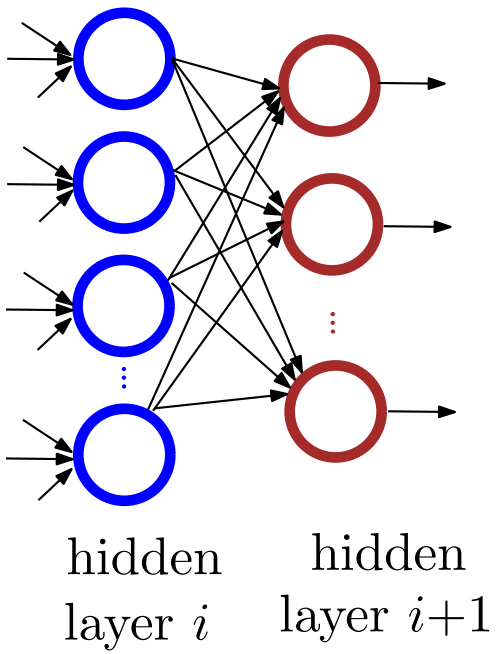
Layers

- Core Layers
  - Dense
  - Activation
  - Dropout
  - Flatten
  - many more ...
- Convolutional Layers
- Pooling Layers
- Recurrent Layers
- Your own Keras layers
- and many more ...

Core Layers

Dense

- fully connected NN layer: connection to all activations from previous layer



Core Layers

Activation

- Applies an activation function
  - detailed next

Dropout

- Applies Dropout to the input
- randomly setting a fraction p of input units to 0
- prevent overfitting

Flatten

- Flattens the input

- many more

Activation Function: Sigmoid

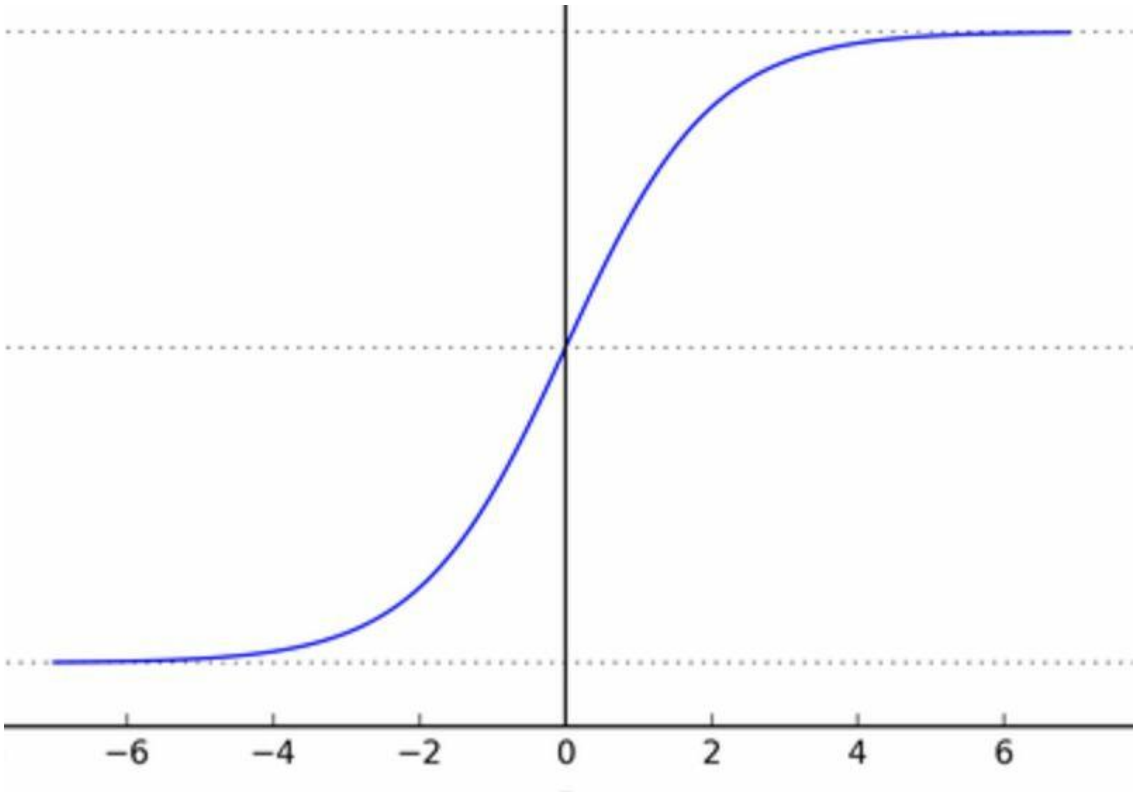


Figure: Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \qquad (1)$$

Activation Function: ReLU (rectified linear unit)

Figure: ReLU

$$f(x) = max(0, x) \quad (2)$$

Activation Function: softmax

- usually used on the output layer to turn the outputs into probability-like values
- Sigmoid: two class
- softmax: multiclass

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{3}$$

for i=1 to K and K is number of output units in output layer

Activation Function

linear

$$f(x) = x \tag{4}$$

- and many more...

Keras provides

Optimizer

- the specific algorithm used to update weights while we train our model
- such as *sgd* (Stochastic gradient descent optimizer)

Objective function or loss function

- used by the optimizer to navigate the space of weights
- such as *mse* (mean squared error)

Metrics

- used to judge the performance of your model
- such as *accuracy*

API

- Keras provides nice API
- documentation
  - A tour of `https://keras.io`

Building a Simple Deep Learning Network Using Keras

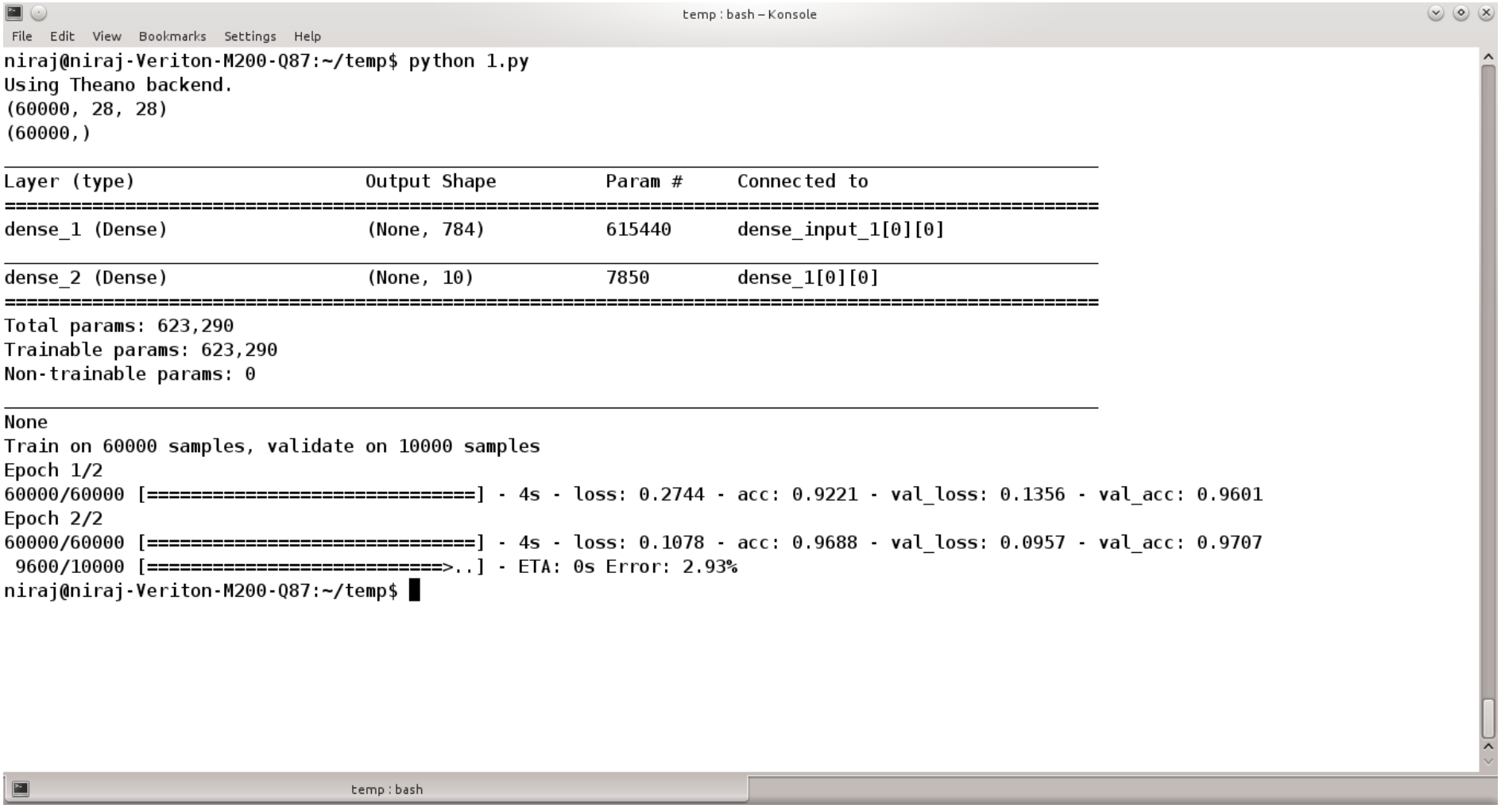
Steps

- Import libraries and modules
- Load image data
- Pre-process data
- Define model architecture
- Compile model
- Fit and evaluate Model
- Improvements

Keras examples

- Fully Connected Neural Network with MNIST dataset

Sample Output



Improving Performance of Simple Network: additional hidden layers

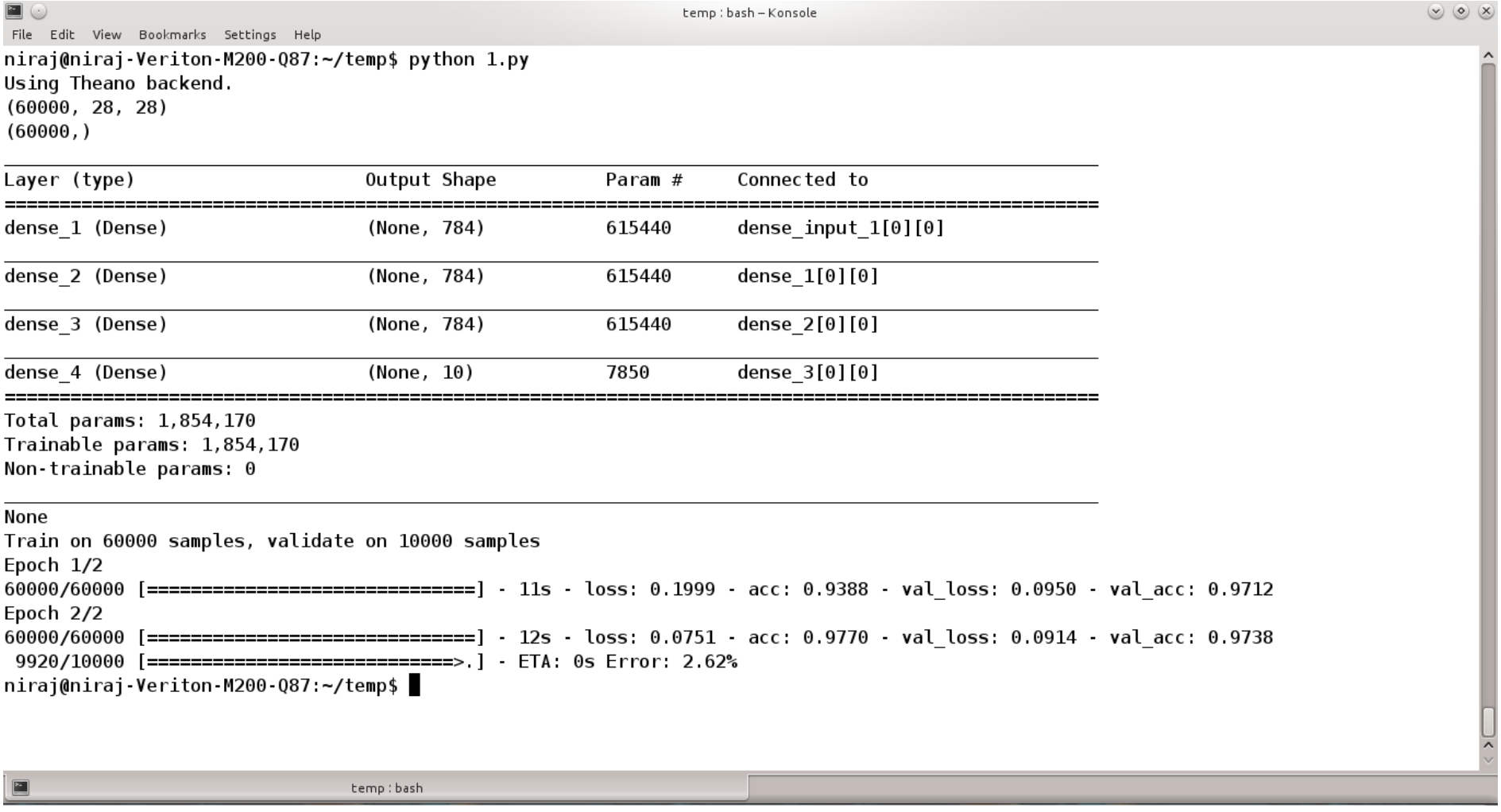
temp : bash – Konsole

File Edit View Bookmarks Settings Help

```
niraj@niraj-Veriton-M200-Q87:~/temp$ python 1.py
Using Theano backend.
(60000, 28, 28)
(60000,)
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 784)               615440
_________________________________________________________________
dense_2 (Dense)              (None, 784)               615440
_________________________________________________________________
dense_3 (Dense)              (None, 10)                7850
=================================================================
Total params: 1,238,730
Trainable params: 1,238,730
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - 8s - loss: 0.2184 - acc: 0.9354 - val_loss: 0.1094 - val_acc: 0.9639
Epoch 2/2
60000/60000 [==============================] - 8s - loss: 0.0755 - acc: 0.9767 - val_loss: 0.0852 - val_acc: 0.9720
 9824/10000 [============================>.] - ETA: 0s Error: 2.80%
niraj@niraj-Veriton-M200-Q87:~/temp$
```

temp : bash

Improving Performance of Simple Network: additional hidden layers



Improving Performance of Simple Network: introducing dropout layer

temp : bash – Konsole

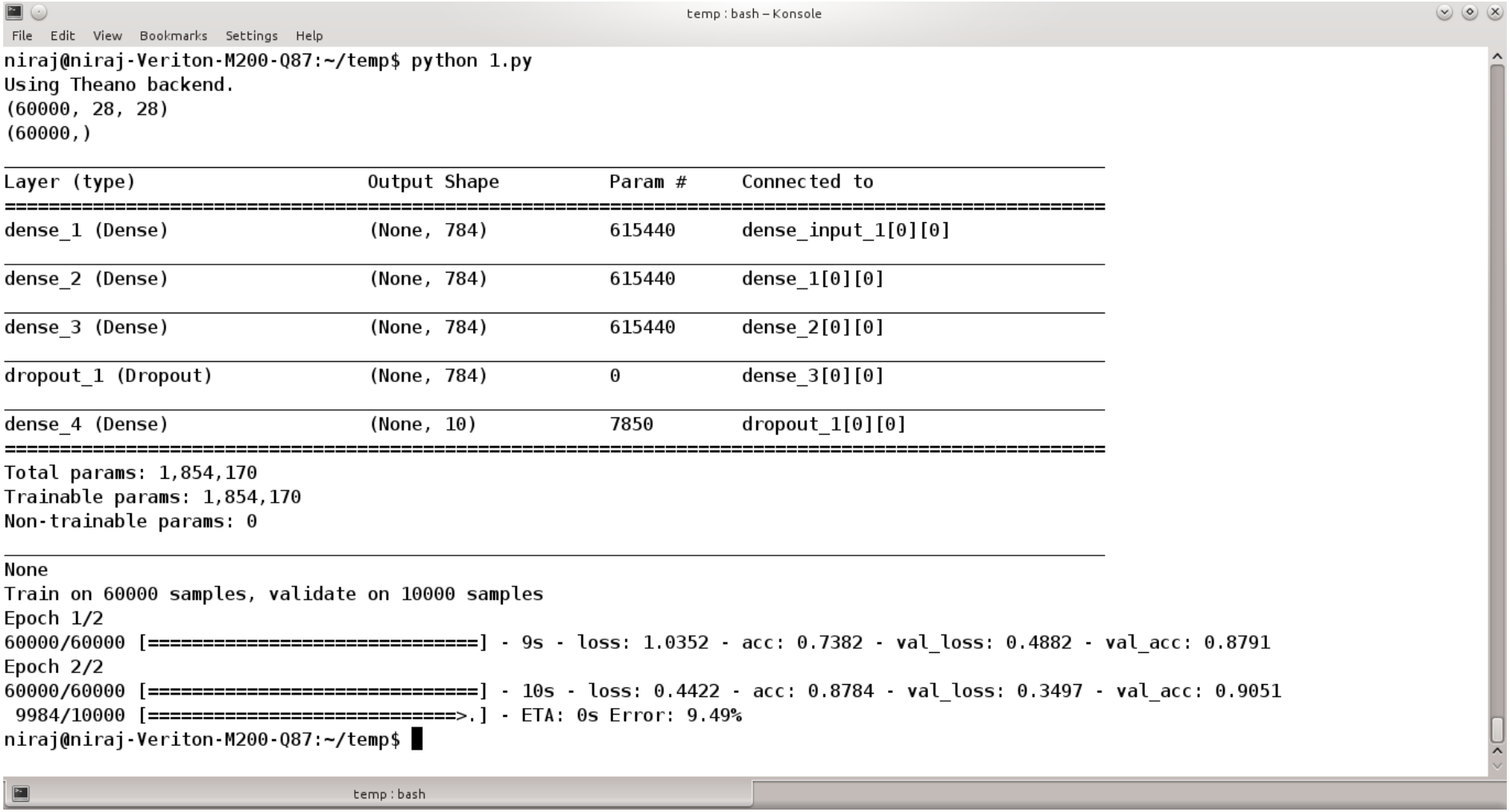
File Edit View Bookmarks Settings Help

```
niraj@niraj-Veriton-M200-Q87:~/temp$ python 1.py
Using Theano backend.
(60000, 28, 28)
(60000,)

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 784)               615440
_________________________________________________________________
dense_2 (Dense)              (None, 784)               615440
_________________________________________________________________
dense_3 (Dense)              (None, 784)               615440
_________________________________________________________________
dropout_1 (Dropout)          (None, 784)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                7850
=================================================================
Total params: 1,854,170
Trainable params: 1,854,170
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/2
60000/60000 [==============================] - 13s - loss: 0.2014 - acc: 0.9386 - val_loss: 0.1017 - val_acc: 0.9697
Epoch 2/2
60000/60000 [==============================] - 14s - loss: 0.0771 - acc: 0.9760 - val_loss: 0.0811 - val_acc: 0.9740
10000/10000 [==============================] - 1s
 Error: 2.60%
niraj@niraj-Veriton-M200-Q87:~/temp$
```

temp : bash

Improving Performance of Simple Network: using different optimizers



Improving Performance of Simple Network: training for more number of epochs

temp : bash – Konsole

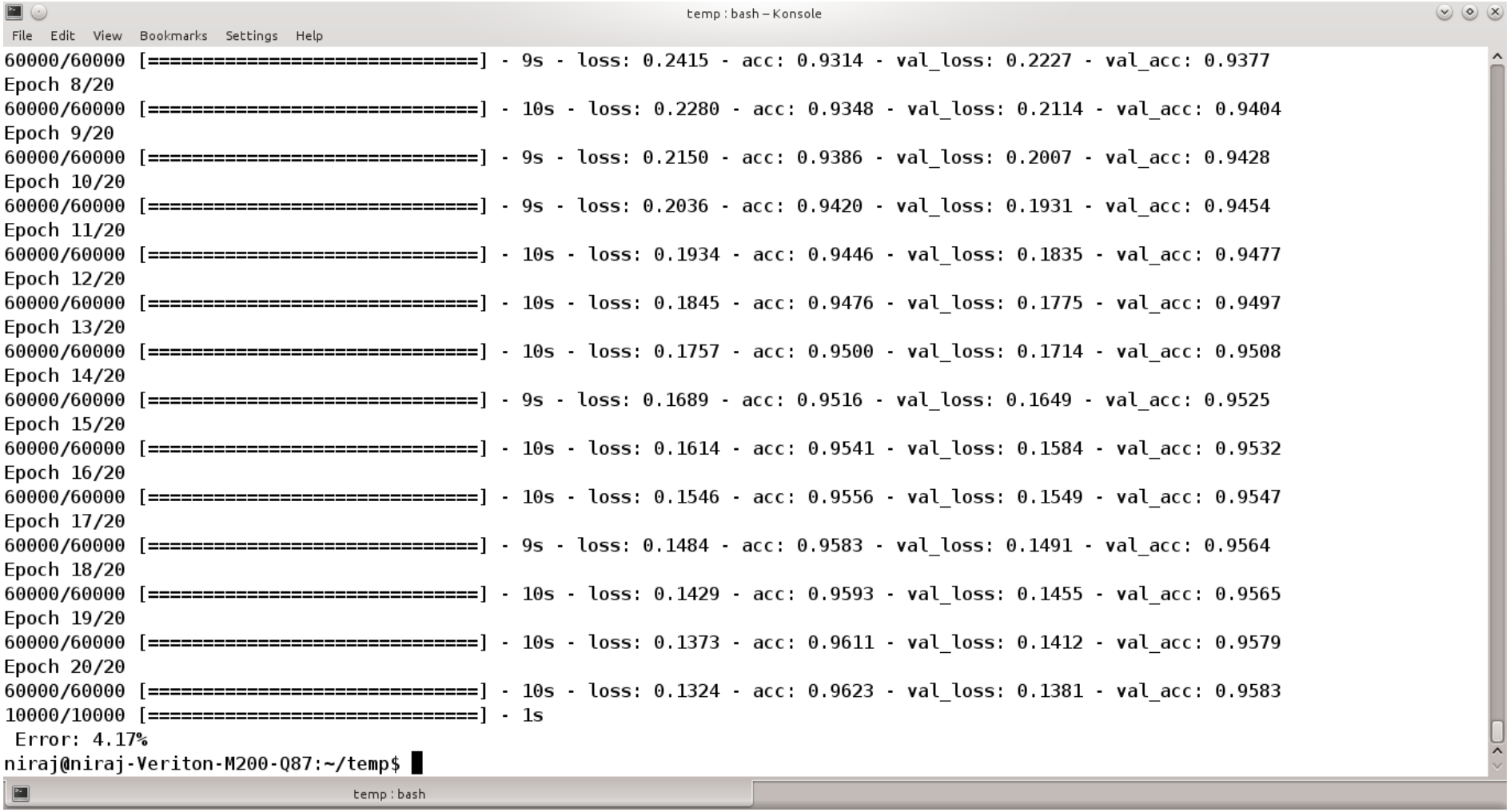
File Edit View Bookmarks Settings Help

```
niraj@niraj-Veriton-M200-Q87:~/temp$ python 1.py
Using Theano backend.
(60000, 28, 28)
(60000,)

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 784)               615440
_________________________________________________________________
dense_2 (Dense)              (None, 784)               615440
_________________________________________________________________
dense_3 (Dense)              (None, 784)               615440
_________________________________________________________________
dropout_1 (Dropout)          (None, 784)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                7850
=================================================================
Total params: 1,854,170
Trainable params: 1,854,170
Non-trainable params: 0
_________________________________________________________________
None
Train on 60000 samples, validate on 10000 samples
Epoch 1/20
60000/60000 [==============================] - 10s - loss: 1.0537 - acc: 0.7378 - val_loss: 0.4933 - val_acc: 0.8815
Epoch 2/20
60000/60000 [==============================] - 9s - loss: 0.4407 - acc: 0.8813 - val_loss: 0.3505 - val_acc: 0.9054
Epoch 3/20
60000/60000 [==============================] - 10s - loss: 0.3512 - acc: 0.9016 - val_loss: 0.3008 - val_acc: 0.9187
Epoch 4/20
```

temp : bash

Improving Performance of Simple Network: training for more number of epochs



Keras examples

- Regression Example

Improving Performance of Simple Network

other options to explore

- additional hidden layers
- dropout
- different optimizers
- more number of epochs
- optimizer learning rate
- number of internal hidden neurons
- batch size

Important Links

Links

1. Keras Official Documentation Page
2. Keras official github
3. Another GitHub Page
4. GitHub Page MNIST example
5. Keras Tutorial
6. An Example
7. Another Example
8. Deep Learning with Keras (Book)

The End