DEEP LEARNING

Lecture 2

Simple Neural Network

Instructor: Zafar Iqbal

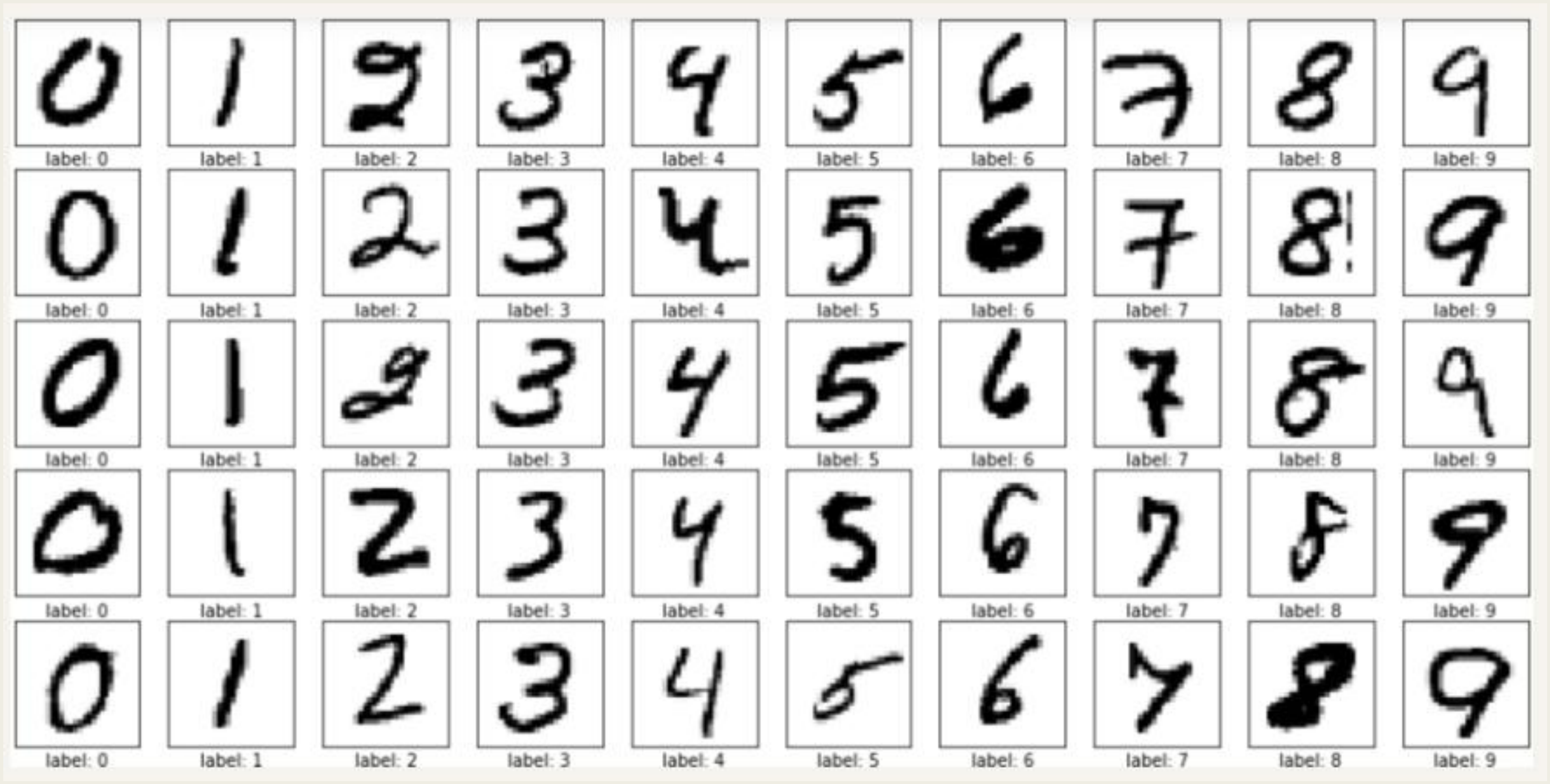
First Steps in Deep Learning

- Choose a framework (e.g., PyTorch or TensorFlow)
- Start with a simple dataset (e.g., MNIST)
- Experiment with a small neural network
- Visualize results to understand model behavior

Example: MNIST Dataset

- Dataset of 70,000 handwritten digits (0–9)
- Task: Classify images into correct digit
- Commonly used for benchmarking neural networks [Image: Sample MNIST digits]

Simple Neural Network

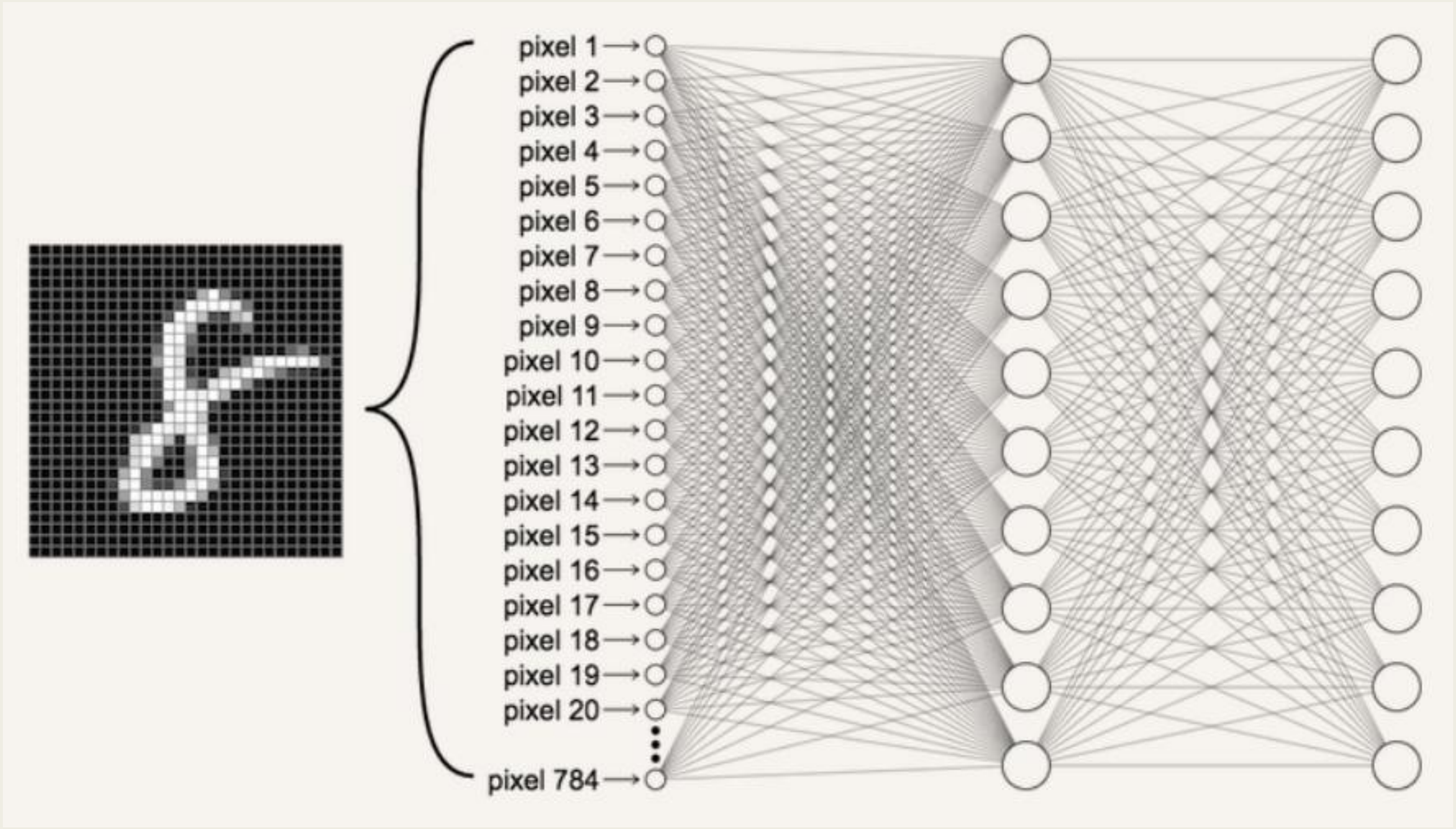


Simple Neural Network

- Example: Classify handwritten digits (MNIST)
- Layers: Dense → ReLU → Dense → Softmax
- Output: Predicted digit (0–9)

- Example coded in Keras (using tesorflow backend).
- Input layer 28*28 = 784
- First layer: Dense 512, activation: relu
- Second layer: Dense 512, activation: relu
- Final layer: Dense 10, activation: softmax
- Function loss: cross entropy
- Gradient algorithm: Minibatch 128 size and RMSprop
- Num epochs: 20

Simple Neural Network



Code

- import tensorflow as tf
- from tensorflow.keras import layers, models
- # Load and preprocess MNIST dataset
- (x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
- x_train, x_test = x_train / 255.0, x_test / 255.0 # Normalize pixel values to [0, 1]
- x_train = x_train.reshape(-1, 28 * 28) # Flatten 28x28 images to 784
- x_test = x_test.reshape(-1, 28 * 28)
- # Build simple neural network
- model = models.Sequential([
-   layers.Dense(128, activation='relu', input_shape=(784,)), # Dense + ReLU
-   layers.Dense(10, activation='softmax') # Dense + Softmax (10 classes)
-])

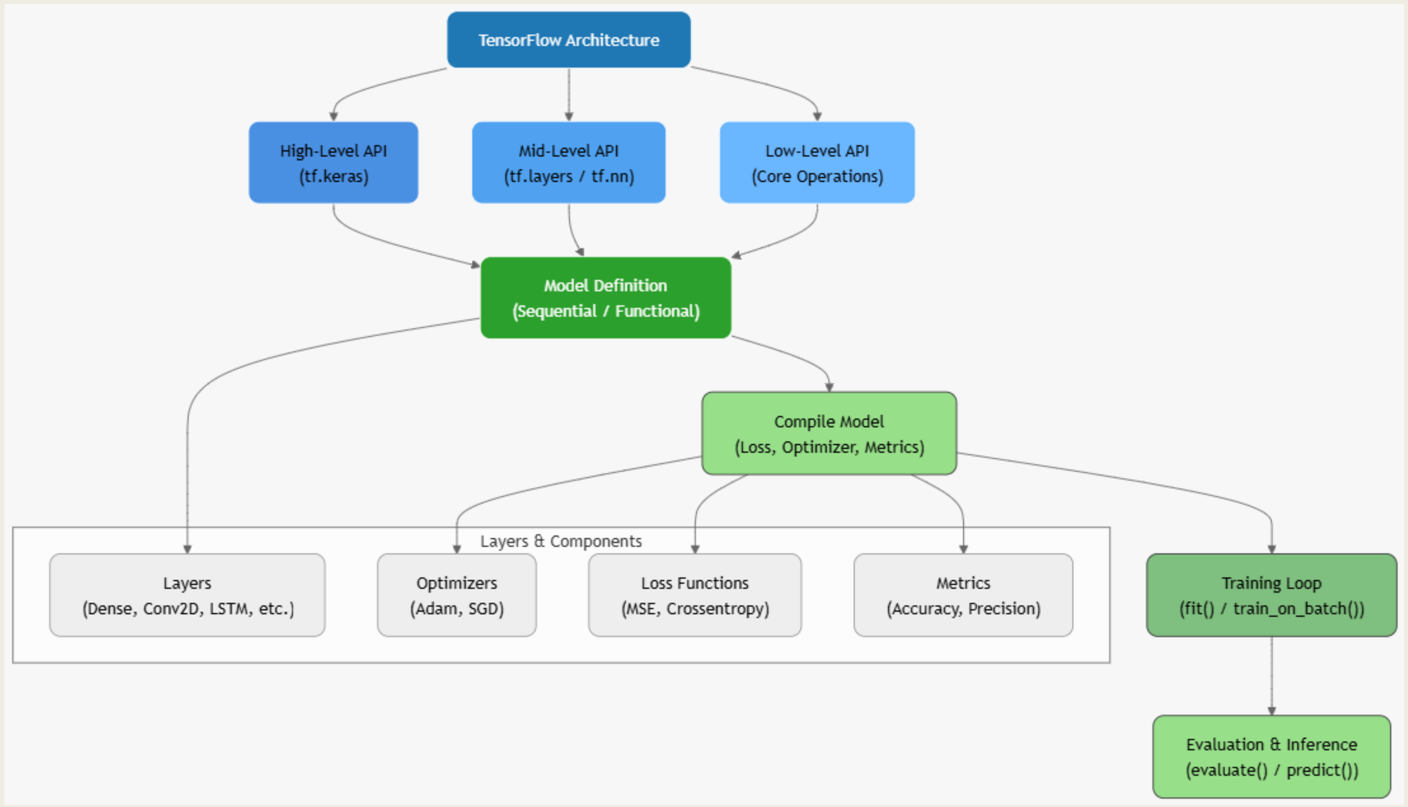
Code

- # Compile and train
- model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy’])
- model.fit(x_train, y_train, epochs=5, batch_size=32, verbose=1)
- # Evaluate on test data
- test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
- print(f"Test accuracy: {test_acc:.4f}")

Importing Libraries

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

- **tensorflow** — main deep learning framework.
- **layers** — defines neural network layers (Dense, Conv, etc.).
- **models** — builds and manages neural network architectures.


TensorFlow Architecture
High-Level API (tf.keras)
Mid-Level API (tf.layers / tf.nn)
Low-Level API (Core Operations)
Model Definition (Sequential / Functional)
Compile Model (Loss, Optimizer, Metrics)
Layers & Components
Layers (Dense, Conv2D, LSTM, etc.)
Optimizers (Adam, SGD)
Loss Functions (MSE, Crossentropy)
Metrics (Accuracy, Precision)
Training Loop (fit() / train_on_batch())
Evaluation & Inference (evaluate() / predict())

Loading the MNIST Dataset

```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data()
```

- MNIST = **handwritten digits (0–9)** dataset.
- x_train → images for training.
- y_train → labels for training.
- x_test, y_test → images and labels for testing.
- Dataset size: **60,000 training + 10,000 testing samples.**

Normalizing Pixel Values

```
x_train, x_test = x_train / 255.0, x_test / 255.0
```

- Converts pixel range from 0–255 → 0–1.
- Helps model train faster and perform better.
- Normalization ensures **numerical stability** and smoother gradients.

Flattening Images

```
x_train = x_train.reshape(-1, 28 * 28)
x_test = x_test.reshape(-1, 28 * 28)
```

- Each image = **28 × 28 pixels → 784 features.**
- Converts 2D image into a 1D vector for input to the dense layers.
- -1 lets NumPy automatically calculate batch size.

Building the Neural Network

```
model = models.Sequential([
  layers.Dense(128, activation='relu', input_shape=(784,)),
  layers.Dense(10, activation='softmax')
])
```

- **Sequential model:** stack of layers in order.
- **Layer 1:** 128 neurons, ReLU activation → learns features.
- **Layer 2:** 10 neurons, Softmax → outputs class probabilities (digits 0–9).

Compiling the Model

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

- **Optimizer:** Adam (adaptive learning rate).
- **Loss:** Sparse categorical cross-entropy (for integer labels).
- **Metric:** Accuracy (percentage of correct predictions).

Training the Model

```
model.fit(x_train, y_train, epochs=5, batch_size=32, verbose=1)
```

- **epochs=5:** train over full dataset 5 times.
- **batch_size=32:** updates weights every 32 samples.
- **verbose=1:** shows progress (e.g., 1500/1500 steps).

Evaluating the Model

```
test_loss, test_acc = model.evaluate(x_test, y_test, verbose=0)
print(f"Test accuracy: {test_acc:.4f}")
```

- Uses unseen test data.
- Returns **test loss** and **test accuracy**.
- Accuracy typically ~0.97–0.98 (97–98%).

Summary

- MNIST is a great dataset to start with.
- Neural networks learn from features → outputs probabilities.
- Proper normalization, model structure, and tuning are essential.
- Achieved ~98% accuracy with simple architecture.