Introduction To Deep Learning & Neural Networks with keras (M2) (1)

Created	@December 10, 2025 11:50 AM
Module Code	IBMM02: Introduction to Deep Learning

Module 3: Keras and Deep Learning Libraries

Deep Learning Libraries

1. TensorFlow

- **Developer:** Google Brain (released 2015)
- **Type:** Low-level + high-level deep learning framework
- **Backend:** Uses dataflow graphs for computation
- **Execution:** Supports both *eager execution* and *graph execution*
- **Strengths:**
  - Highly scalable for production environments
  - Strong GPU/TPU support
  - Very large ecosystem (TensorBoard, TensorHub, TF Lite, TF Serving)
  - Most widely used in **industry**
- **Weakness:**
  - More complex syntax
  - Steeper learning curve than Keras

2. PyTorch

- **Developer:** Facebook/Meta AI (released 2016)
- **Type:** Low-level deep learning framework
- **Backend:** Uses dynamic computation graphs ("define-by-run")
- **Execution:** Eager by default
- **Strengths:**
    - Very flexible for custom architectures
    - Preferred in **research** because debugging is easy
    - Pythonic syntax
    - Strong GPU acceleration via CUDA
- **Weakness:**
    - Historically less production tooling (though now TorchServe exists)

---

3. Keras

- **Developer:** Initially independent; now tightly integrated with TensorFlow
- **Type:** High-level API for neural network construction
- **Backend:** Runs **on top of TensorFlow** (TF 2.x includes tf.keras directly)
- **Strengths:**
    - Extremely simple and clean API
    - Fast prototyping
    - Minimal code to build complex models
- **Weakness:**
    - Less control over low-level operations
    - Not ideal for highly experimental/custom layers compared to PyTorch

---

4. Theano

- **Developer:** MILA (Montreal Institute for Learning Algorithms)
- **Type:** Low-level symbolic computation library
- **Backend:** Static computation graph
- **Strengths:**
    - Historically the first widely used deep learning framework
    - Enabled GPU-accelerated matrix computation
- **Weakness:**
    - Officially discontinued
    - Replaced by more modern frameworks

Regression Models with Keras

```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from keras.layers import Input
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

df = pd.read_csv("concrete_data.csv")   change filename if needed

X = df.drop("Concrete_compressive_strength", axis=1).values
y = df["Concrete_compressive_strength"].values

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
```

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

def regression_model():
    # create model
    model = Sequential()
    model.add(Input(shape=(n_cols,)))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(50, activation='relu'))
    model.add(Dense(1))

model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='mse',
    metrics=['mae']
)

model.summary()
model = regression_model()

history = model.fit(predictors_norm, target, validation_split=0.3, epochs=100,
verbose=2

test_loss, test_mae = model.evaluate(X_test, y_test, verbose=0)
print(f"Test MAE: {test_mae:.3f}")

predictions = model.predict(X_test)
print("Sample predictions:", predictions[:5].flatten())
```

Classification Models with Keras

	price_high	price_low	price_med	maintenance_high	maintenance_low	maintenance_med	persons_2	persons_more	decision
0	1	0	0	1	0	0	1	0	0
1	1	0	0	1	0	0	1	0	0
2	1	0	0	1	0	0	1	0	0
3	1	0	0	1	0	0	1	0	0
4	1	0	0	1	0	0	1	0	0

```
import numpy as np
import pandas as pd
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split

df = pd.read_csv("car_data.csv")

# predictors (features)
X = df.drop("decision", axis=1).values

# target (0,1,2,3)
y = df["decision"].values

# convert to one-hot encoding
y_cat = to_categorical(y)
# to convert categorial to binary array



X_train, X_test, y_train, y_test = train_test_split(
    X, y_cat, test_size=0.2, random_state=42



model = Sequential()
model.add(Dense(5, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dense(5, activation='relu'))
model.add(Dense(4, activation='softmax'))   # 4 classes

model.compile(
```

```
    optimizer='adam',
    loss='categorical_crossentropy', # Used for multiclass classification
    metrics=['accuracy']
)

model.summary()

history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=16,
    validation_split=0.2
)

loss, acc = model.evaluate(X_test, y_test)
print(f"Test Accuracy: {acc:.3f}")

pred = model.predict(X_test[:5])
print("Raw class probabilities:\n", pred)
print("Predicted class labels:", np.argmax(pred, axis=1))
```

```
model.predict(test_data)
```

```
array([[9.9978679e-01, 2.1028408e-04, 1.0243932e-06, 1.8633460e-06],
       [9.9978679e-01, 2.1028408e-04, 1.0243932e-06, 1.8633460e-06],
       [9.9978679e-01, 2.1028408e-04, 1.0243932e-06, 1.8633460e-06],
       ...,
       [4.9434581e-01, 4.9560347e-01, 4.4486104e-03, 5.6021004e-03],
       [4.9434581e-01, 4.9560347e-01, 4.4486104e-03, 5.6021004e-03],
       [4.9434581e-01, 4.9560347e-01, 4.4486104e-03, 5.6021004e-03]],
      dtype=float32)
```