# Mastering the Core of Deep Learning: A Playbook for Classification & Regression

Distilling the essential workflows from Chapter 4 of “Deep Learning with Python”

# Machine Learning Answers Three Fundamental Questions

## Is it A or B?

Binary Classification

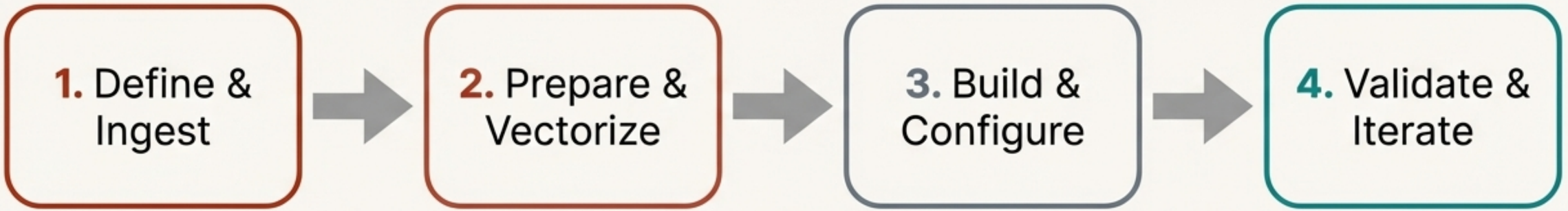
## Which of these N things is it?

Multiclass Classification

## How much or how many?

Scalar Regression

# A Universal Blueprint for Solving These Problems




Understand the problem and the raw data.

Transform data into tensors the network can understand.

Design the model architecture and choose the right tools (loss, optimizer).

Train the model, monitor for overfitting, and evaluate performance.

This core process is the key. While the tools may change for each problem, the workflow remains constant. We will now apply this blueprint to three distinct case studies.

# Case Study 1: Is the Review Positive or Negative?

1. Define & Ingest → 2. Prepare & Vectorize → 3. Build & Configure → 4. Validate & Iterate

## The Problem (Define & Ingest)

Classify 50,000 IMDb movie reviews as either positive (1) or negative (0). Data is split 25k for training and 25k for testing.

## The Data

Raw text is preprocessed into sequences of integers, keeping only the top 10,000 most frequent words.

? this film was just brilliant... ⟶ [1, 14, 22, 16, ...]

## The Transformation (Prepare & Vectorize)

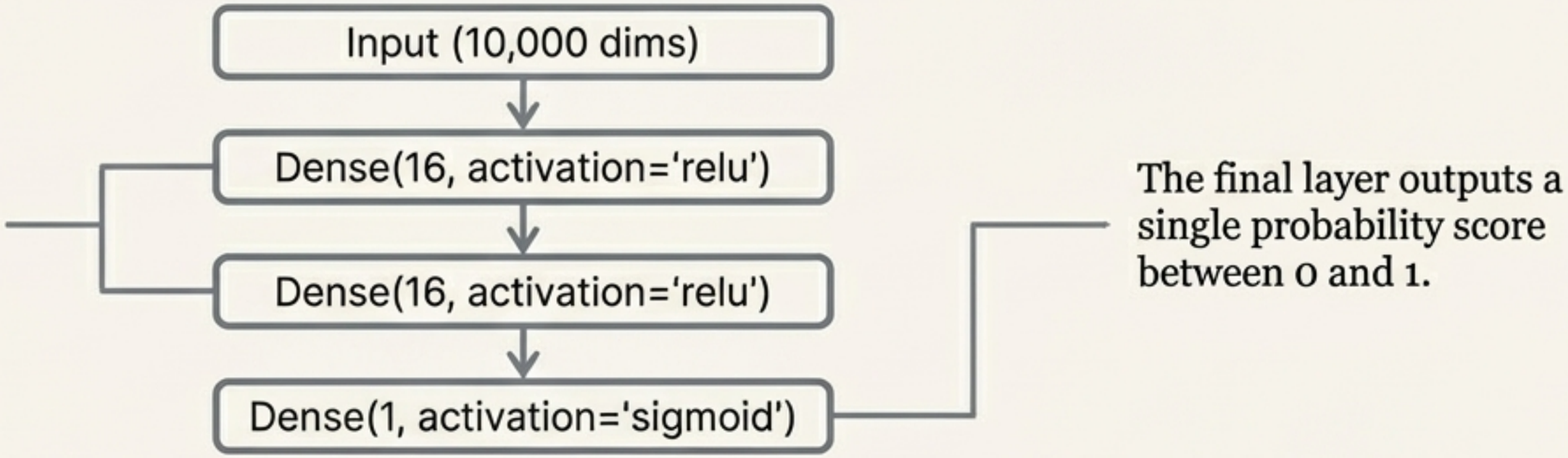
Step-by-step of a “multi-hot encoding”


Labels (0s and 1s) are also cast to `float32`.

# The Anatomy of a Binary Classifier

1. Define & Ingest → 2. Prepare & Vectorize → **3. Build & Configure** → 4. Validate & Iterate

Intermediate layers with 16 units use `relu` to learn complex, non-linear patterns. Without a non-linearity, a deep stack of layers would only learn a single linear transformation.




The final layer outputs a single probability score between 0 and 1.

**Loss: `loss='binary_crossentropy'`**

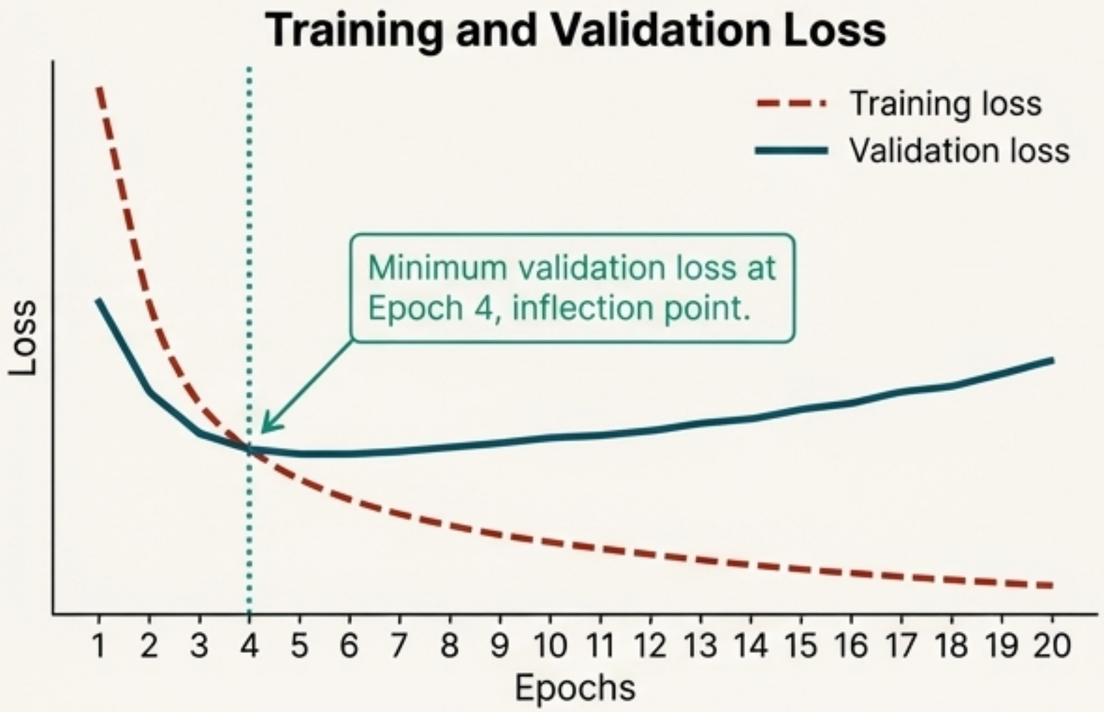
The ideal choice for a sigmoid output. It measures the distance between the predicted probability and the true label (0 or 1).

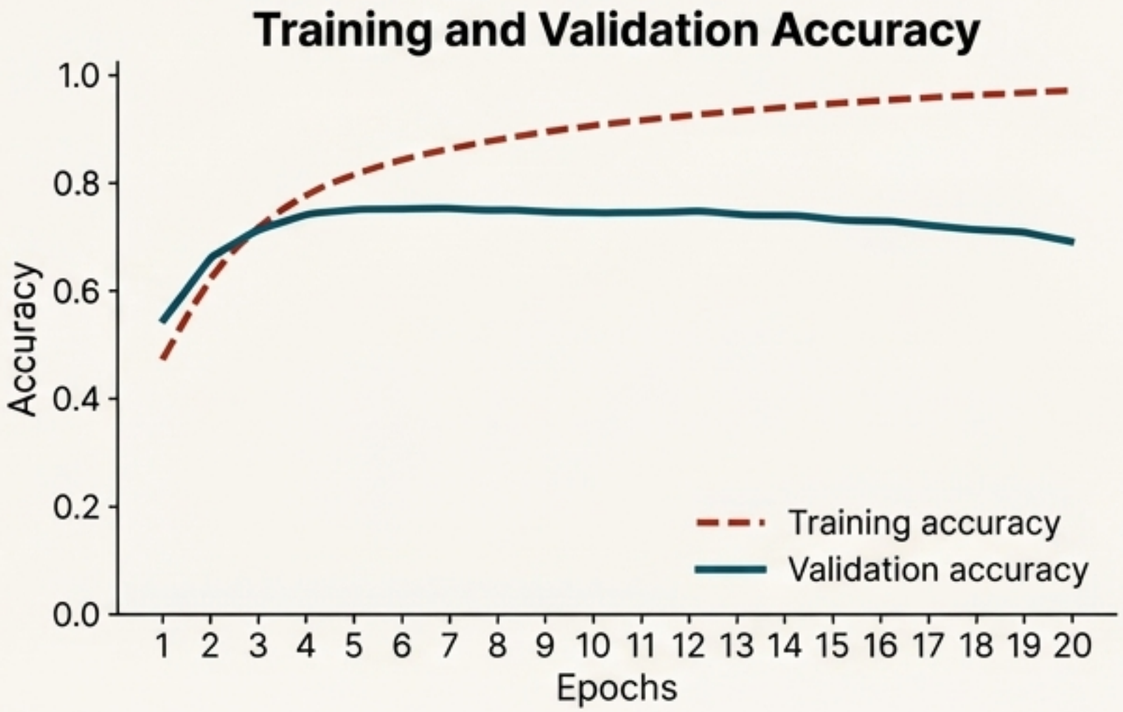
**Optimizer: `optimizer='adam'`**

A robust, effective default choice for most problems.

# The Primary Enemy: Overfitting

1. Define & Ingest → 2. Prepare & Vectorize → 3. Build & Configure → **4. Validate & Iterate**




Performance on training data is not the goal. Notice how validation performance peaks at Epoch 4 and then degrades. This is overfitting. To maximize generalization, we retrain a final model for only 4 epochs.

Final Result: This approach achieves **88% accuracy** on the test set.

# Case Study 2: What's the News Topic?

1. Define & Ingest → 2. Prepare & Vectorize → 3. Build & Configure → 4. Validate & Iterate

## The Problem

Classify 8,982 Reuters newswires into one of 46 mutually exclusive topics. This is single-label, multiclass classification.

## The Key Adaptations

| Before | | After |
|---|---|---|
| Dense(1, 'sigmoid') | evolves to | Dense(46, 'softmax') |

Softmax distributes 100% probability across all 46 classes, giving a score for each.

| | | |
|---|---|---|
| binary_crossentropy | evolves to | categorical_crossentropy |

Measures the distance between the model's predicted probability distribution and the true one-hot encoded label.

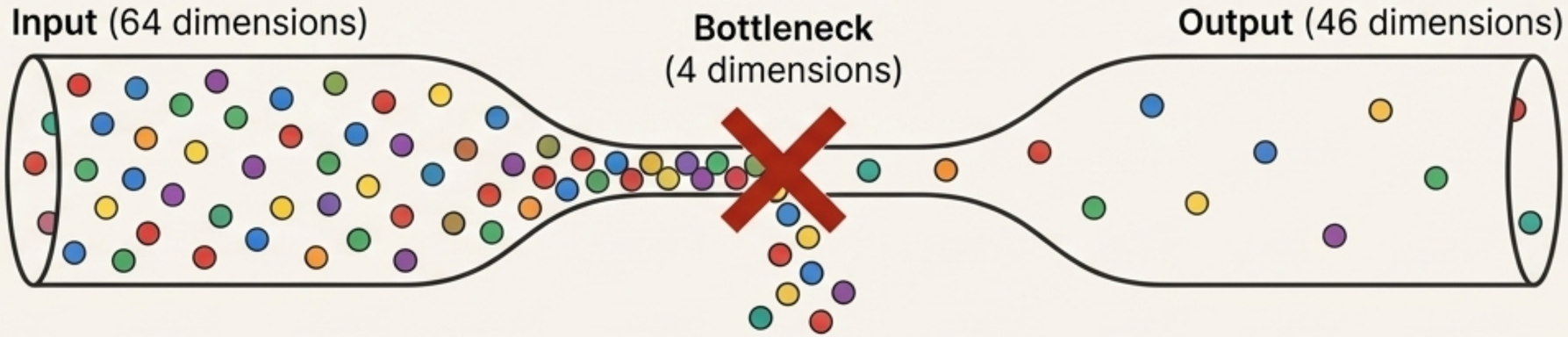
| | | |
|---|---|---|
| Dense(16) | evolves to | Dense(64) |

Larger layers are needed to learn representations for 46 classes without creating an information bottleneck.

# Why Model Shape Matters: The Information Bottleneck

1. Define & Ingest → 2. Prepare & Vectorize → 3. Build & Configure → 4. Validate & Iterate

What happens if an intermediate layer is too small?
We tested a model with a **Dense(4)** layer, trying to compress information for 46 classes into a 4-dimensional space.




Validation accuracy peaked at **~71%**.
This is an **8% absolute drop** from the ~79% accuracy of the 64-unit model.

If you need to classify data into many categories, avoid creating information bottlenecks with intermediate layers that are too small.

# Case Study 3: What's the House Price?

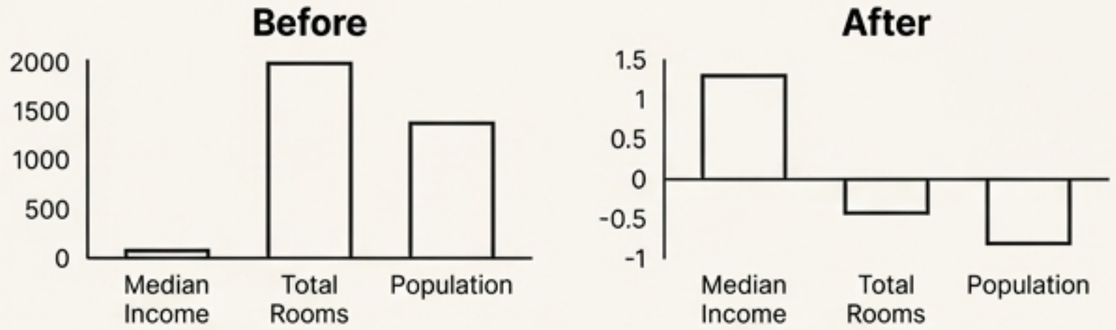
1. Define & Ingest
2. Prepare & Vectorize
3. Build & Configure
4. Validate & Iterate

**A New Kind of Problem:** We are now predicting a continuous value (scalar regression), not a discrete class.

**The Data:** The California Housing dataset. We use a small version with only 480 training samples and 8 features per sample (longitude, population, median income, etc.).

# New Data Prep Challenges

## 1. Feature Normalization



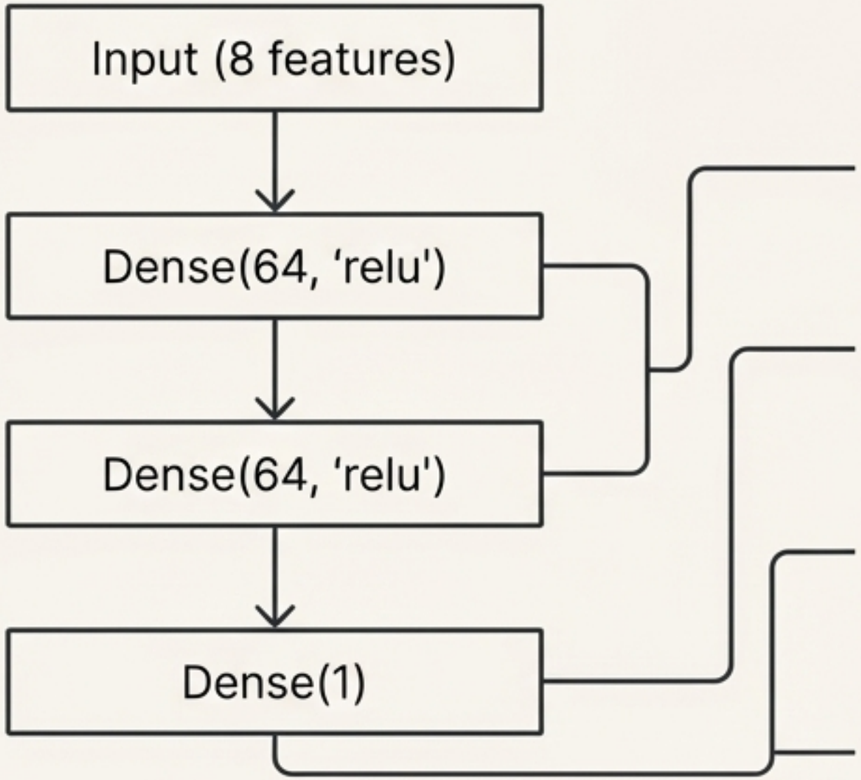

We normalize each feature by subtracting its mean and dividing by its standard deviation. This helps the model learn efficiently.

## 2. Small Dataset

**Only 480 training samples**

# The Anatomy of a Regression Model

1. Define & Ingest → 2. Prepare & Vectorize → 3. Build & Configure → 4. Validate & Iterate




**Small Model:** With so few samples, we use a small model (two Dense(64) layers) to mitigate severe overfitting.

**Final Layer:** Dense(1): The final layer has a single unit and no activation function. This is critical. It allows the model to predict any continuous value, not constrained to a specific range like sigmoid or softmax.

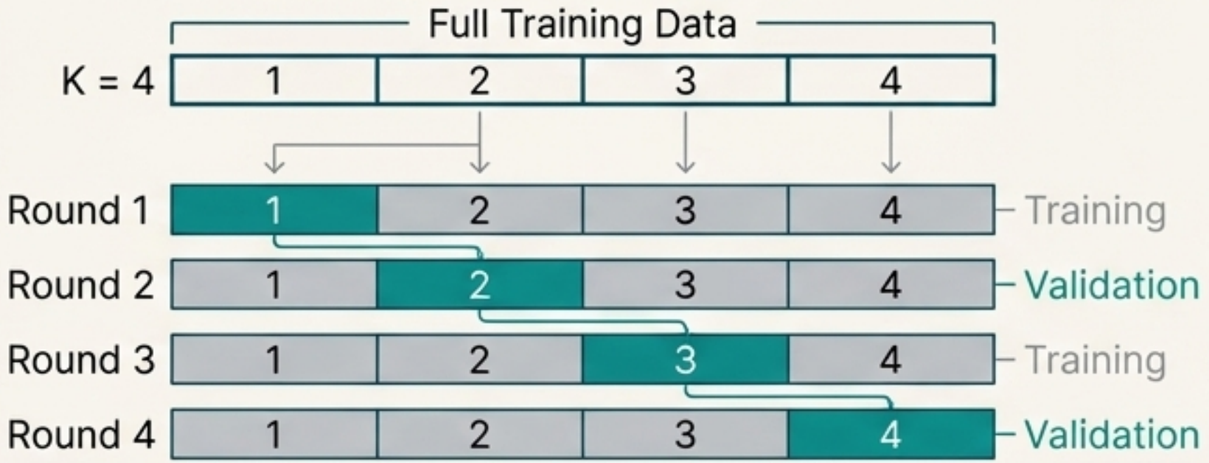
**Loss Function:** loss='mean_squared_error': MSE is a standard loss for regression. It's the square of the difference between the prediction and the target, penalizing larger errors more heavily.

**Metric:** metrics=['mean_absolute_error']: We monitor Mean Absolute Error (MAE). An MAE of 0.31 means our predictions are off by an average of $31,000.

# Reliable Evaluation for Small Datasets: K-Fold Validation

With only 480 samples, a standard train/validation split is unreliable. The validation score would have high variance depending on which samples ended up in the split.

## The Solution: K-Fold Validation




1. Split data into K (e.g., 4) partitions.
2. Train K identical models.
3. Each model trains on K–1 partitions and validates on the remaining one.
4. The final validation score is the average of the K scores.

The average MAE score provides a much more reliable estimate of the model's performance. In this case, validation MAE stopped improving after ~130 epochs.

# The Decision Matrix: Choosing the Right Tools for the Job

| Feature | Case 1: Binary Classification | Case 2: Multiclass Classification | Case 3: Scalar Regression |
|---|---|---|---|
| **Goal** | Predict one of two classes | Predict one of N classes | Predict a continuous value |
| **Final Layer** | `Dense(1, 'sigmoid')` | `Dense(N, 'softmax')` | `Dense(1)` (no activation) |
| **Loss Function** | `binary_crossentropy` | `categorical_crossentropy` | `mean_squared_error` |
| **Key Challenge** | Overfitting | Information Bottleneck | Small Data / Normalization |
| **Evaluation Method** | Validation Set Split | Validation Set Split | K-Fold Cross-Validation |

# The Playbook: Key Principles for Your Projects

- ✓ Always perform feature-wise normalization when input features have different ranges.
- ✓ For small datasets, prefer smaller models (fewer layers and units) to combat severe overfitting.
- ✓ When data is scarce, K-fold validation is the most reliable way to evaluate your model.
- ✓ For binary classification, end your model with a `Dense(1, 'sigmoid')` layer and use `binary_crossentropy` loss.
- ✓ For single-label, N-class classification, end your model with a `Dense(N, 'softmax')` layer and use `categorical_crossentropy` loss.
- ✓ For scalar regression, end your model with a `Dense(1)` layer (no activation) and use `mean_squared_error` loss.
- ✓ Monitor performance on a validation set to detect overfitting. Training performance alone is misleading.

# You now have the playbook for the three most common tasks in deep learning.

Based on the workflows and principles in Chapter 4 of *Deep Learning with Python* by François Chollet
Manning Publications