AI TRAILBLAZERS · SELF-GUIDED STUDENT PACK

Session 1

Machines That Learn — train your first real AI, by yourself

This pack belongs to: _____

Read this first: how to use this pack alone

You don't need a teacher for this. Everything a coach would say is written right here. By the end you will have **trained a real neural network in Python** — and understood what that actually means.

- **What you need:** a computer with a web browser and internet, and a free Google account. That's it. No installing anything.
- **How long:** about 90–120 minutes. Go at your own pace. You can stop and come back.
- **How it works:** you'll read a step, **copy a block of code** into Google Colab, run it, look at what happens, and write your numbers in this pack.
- **Write in this pack.** The blanks and boxes aren't decoration — your numbers become your evidence (and later, your resume).

> **The one golden rule of this whole bootcamp:** you may ask an AI (ChatGPT/Claude/Gemini) to *explain* an error message — never to *write* your code for you. Copying an answer teaches you nothing. Struggling for four minutes and getting it teaches you everything. **Error messages are clues, not verdicts** — read the last line out loud.

GOAL **What you'll build today**

Three real wins — check them off as you earn them:

- ☐ **Train & evaluate** a real image classifier and report its held-out accuracy.
- ☐ **Tune two knobs** (settings) and explain what changed and why.
- ☐ **Speak the language:** explain features, labels, and loss — and how data shapes behavior.

By the end you'll be able to honestly say: **"I trained a neural network in code today."**

THE CONTRACT Four words and nothing is magic anymore

Learn these four and you can explain *any* AI. We'll use today's example (teaching a computer to recognize handwritten digits):

Word	What it means	In today's project
Features	The numbers going *in*	The 64 pixel brightness values of each tiny image
Label	The correct answer	Which digit it actually is (0–9)
Loss	The "wrongness" score	How far the model's guesses are from the real answers
Training	Nudging the model to shrink loss	Adjusting the model until it guesses digits well

Say them out loud right now: *features, label, loss, training*. You'll use all four in the next hour.

STEP 0 Open your lab (Google Colab)

Google Colab is a free website where you write and run Python — nothing to install. Do this once:

1. Go to `colab.research.google.com` in your browser.
2. Sign in with your Google account if it asks.
3. Click **File → New notebook**. A blank page with an empty grey box appears. That grey box is a **cell** — where code goes.
4. Click inside the cell, type `print("hello AI")`, then press **Shift + Enter** to run it. You should see `hello AI` appear below.

> **You should see:** the words `hello AI` printed just under the cell, and a new empty cell appear below. If so — your lab works. 🎉 To add another cell any time, click **+ Code**.

> **How to run any code in this pack:** copy the whole grey block, paste it into a new Colab cell, and press **Shift + Enter**. Run them *in order*, top to bottom.

STEP 1 Load the data — and actually LOOK at it

Rule one of machine learning: **look at your data before you model it.** We'll use a famous set of 1,797 tiny (8×8 pixel) images of handwritten digits that comes built into Python.

COLAB CELL 1

```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

digits = load_digits()
print("We have", len(digits.images), "images")
print("Each image is", digits.images[0].shape, "pixels")

# Look at the first 10 digits and their real labels
fig, axes = plt.subplots(1, 10, figsize=(12, 2))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.imshow(image, cmap="gray")
    ax.set_title(label)
    ax.axis("off")
plt.show()
```

> **You should see:** `We have 1797 images`, `Each image is (8, 8) pixels`, and a row of 10 fuzzy little digits with their numbers on top.

What just happened: each image is really just 64 numbers (an 8×8 grid of brightness values from 0–16). Those 64 numbers are the **features**. The digit written on top is the **label**. That's the whole dataset: features + labels.

✓ Checkpoint 1

How many images are there? _____ How many numbers (features) describe each one? _____

STEP 2 The sacred split (train vs. test)

Here's the most important idea in all of machine learning. If you let the model *study* every example and then *test* it on those same examples, of course it looks perfect — it's an open-book test on questions it memorized. So we **split** the data: most for studying (**train**), some hidden away for the real exam (**test**).

COLAB CELL 2

```
from sklearn.model_selection import train_test_split

X = digits.data      # features: 64 numbers per image
y = digits.target    # labels: the correct digit (0–9)

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=450, random_state=42)

print("Studying with (train):", len(X_train), "images")
print("Hidden exam (test):   ", len(X_test), "images")
assert len(X_train) == 1347 and len(X_test) == 450
print("Split passed ✓")
```

You should see: `Studying with (train): 1347`, `Hidden exam (test): 450`, and `Split passed ✓`.

Why `assert`? It's a tripwire — if the split were ever wrong, Python would stop and shout. Real engineers leave tripwires so mistakes can't sneak through. If you see `AssertionError`, your numbers didn't match — re-run cell 1 first.

✓ Checkpoint 2

Train size: _____ Test size: _____ Did the split pass? ☐ yes

STEP 3 Train your first model

Now we train a simple model called *logistic regression*. Don't worry about the name — the two lines that matter are `.fit()` (study the training data) and `.score()` (take the hidden exam).

COLAB CELL 3

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=10000)
model.fit(X_train, y_train)        # study (this is TRAINING)

accuracy = model.score(X_test, y_test)   # the hidden exam
print("Held-out accuracy:", round(accuracy * 100, 1), "%")
```

> **You should see:** `Held-out accuracy: 97.3 %` (because we set `random_state=42`, you should get this exact number).

Write your number: My logistic-regression held-out accuracy = _____ %

That percent is the fraction of the 450 *never-before-seen* digits it got right. This is the **only honest grade** — it was measured on the hidden exam, not the study set.

STEP 4 Where did it go wrong? (misclassification gallery)

A good scientist studies the failures. Let's look at digits the model got wrong.

COLAB CELL 4

```
import numpy as np

predictions = model.predict(X_test)
wrong = np.where(predictions != y_test)[0]
print("It got", len(wrong), "wrong out of", len(y_test))

fig, axes = plt.subplots(1, 5, figsize=(11, 2))
for ax, idx in zip(axes, wrong[:5]):
    ax.imshow(X_test[idx].reshape(8, 8), cmap="gray")
    ax.set_title(f"real {y_test[idx]}, said {predictions[idx]}")
    ax.axis("off")
plt.show()
```

> **You should see:** how many it missed, then 5 blurry digits labeled with the truth and the model's wrong guess.

✓ Checkpoint 3 — your weirdest miss

My strangest mistake was a real _____ that the model called a _____. Looking at the picture, I can/can't see why:

_____

STEP 5 Model Autopsy — the confusion matrix

A single accuracy number hides *which* digits it struggles with. A **confusion matrix** is a report card with detail: rows are the truth, columns are what the model said. The diagonal is where it was right.

COLAB CELL 5

```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

cm = confusion_matrix(y_test, predictions)
ConfusionMatrixDisplay(cm).plot(cmap="Greens")
plt.show()
```

You should see: a 10×10 grid. Big numbers down the diagonal = mostly right. Any stray number off the diagonal is a confusion (e.g., a real 8 the model called a 9).

Now read it like an engineer: recall by hand

The most useful question isn't "how accurate overall?" It's **per-class recall**: *of all the true 8s, what fraction did the model catch?* Here is a small real example (just digits 1, 7, 8, 9):

true ↓ / said →	1	7	8	9	row total
1	**42**	3	0	0	45
7	4	**39**	0	2	45
8	1	0	**38**	6	45
9	0	1	5	**40**	46

Worked example — recall for digit 8: the model correctly caught **38** of the **45** true 8s. Recall = 38 ÷ 45 = **0.844 = 84%**. The other 7 true 8s were mislabeled (six became 9s, one became a 1).

✓ Your turn — compute two recalls

Recall for digit **1** = 42 ÷ 45 = _____ % Recall for digit **9** = 40 ÷ 46 = _____ %

Which single digit here has the *lowest* recall (the model's weakest)? _____

You just did real data-science by hand. Notice the two digits confused most are **8 and 9** — both are round on top with a loop/tail, so their pixel patterns overlap. The model confuses *shapes*, not meanings. Next week you'll use this exact tool to catch a model that *hides* its harm behind a good overall score.

STEP 6 Train an actual neural network

Time for the headline. A **neural network** is a more powerful model, loosely inspired by brain cells ("neurons"). Watch its **loss** (wrongness) fall as it trains — that falling curve *is* learning happening.

COLAB CELL 6

```
from sklearn.neural_network import MLPClassifier

nn = MLPClassifier(hidden_layer_sizes=(64,), max_iter=500, random_state=42)
nn.fit(X_train, y_train)
print("Neural net accuracy:", round(nn.score(X_test, y_test) * 100, 1), "%")

# Watch the loss curve fall = watch it learn
plt.plot(nn.loss_curve_)
plt.xlabel("training round")
plt.ylabel("loss (wrongness)")
plt.title("Watch the loss fall")
plt.show()
```

> **You should see:** an accuracy around 97–98%, and a curve that starts high on the left and *slides down* toward zero. (A yellow "convergence" warning is normal — it just means it could keep improving with more rounds.)

Write your number: My neural-network accuracy = _____ % — higher or lower than logistic regression? _____

Stop and notice: you just *trained a neural network in code.* That sentence is true now in a way it wasn't an hour ago. Write your version of it here:

STEP 7 Knob experiments — PREDICT before you run

Real understanding comes from breaking things on purpose. For each change below, **first write your prediction**, then run it, then explain what happened. Copy cell 6, change the one line shown, and re-run.

Change this line	My prediction	What happened (accuracy)	Why?
`hidden_layer_sizes=(4,)`			
`hidden_layer_sizes=(256,)`			
`max_iter=5`			
train on 100 examples*			

*For the last one, change `nn.fit(X_train, y_train)` to `nn.fit(X_train[:100], y_train[:100])` — it studies only 100 images instead of 1,347.

What you'll usually find: (4,) too few neurons → accuracy drops (underfitting). **(256,)** more power → usually a bit higher, but slower. **max_iter=5** → stops too soon, the loss curve never falls, accuracy drops. **100 examples** → accuracy drops *hard* — a worse teacher makes a worse model.

LAW #1 — EARNED TODAY

The data is the teacher.

100 examples made a worse model than 1,347. The model is only as smart — and only as fair — as what you feed it. Now connect it: **what happens when a face-recognition system is trained mostly on one demographic?** Hold that thought. It becomes very real in Session 2.

THINK Confidently wrong vs. unsurely wrong

A model can be wrong *and sure of itself*, or wrong *and hesitant*. In the real world, one is far more dangerous. Imagine a self-driving car's vision system.

Circle-up question

Which scares you more — a model that is **99% sure and wrong**, or **55% sure and wrong**? Why?

(A hint for your thinking: a hesitant model can ask for help or hand off to a human. A confident-and-wrong model just acts — and nobody double-checks it.)

STUCK? Troubleshooting — errors are clues

Errors are normal and good — every engineer reads dozens a day. **Read the last line of the error first.** Here are the common ones:

Error message contains…	What it means & the fix
`NameError: 'digits' is not defined`	You ran a later cell before an earlier one. Re-run the cells *in order* from cell 1.
`ModuleNotFoundError`	A tool didn't load. In Colab everything is pre-installed — just re-run the cell's `import` line.
`AssertionError`	Your split numbers didn't match 1347/450. Re-run cell 1, then cell 2.
`IndentationError`	Python cares about spaces. Make sure lines inside a `for` loop are indented the same (paste the whole block again).
A yellow `ConvergenceWarning`	Not an error! It just means the model could keep improving. Your results are still valid.

> **Allowed:** paste an error into ChatGPT/Claude and ask "*what does this error mean?*" **Not allowed:** asking it to write the fix for you. Understand it, then fix it yourself.

FINISH LINE Self-check & your badge

Tick each one you can truly say yes to:

- [ ] I loaded 1,797 images and looked at them.
- [ ] I split the data 1347/450 and the tripwire passed.
- [ ] I trained a model and wrote down its held-out accuracy.
- [ ] I read a confusion matrix and computed recall by hand.
- [ ] I trained a neural network and watched its loss curve fall.
- [ ] I changed two knobs and can explain what happened.

🎯 Badge: Model Trainer

You've earned it if: a digit classifier is trained in code; accuracy came from a proper split; two knobs are tuned and explained; and your numbers are written in this pack. **Earned?**

☐ **Yes.**

RECORD 1 Your 4-H Record Sheet — Session 1

This becomes part of your 4-H record book and your capstone story. Numbers are evidence — always write the numbers.

Name: _____ Date: _____

What I DID (tools, notebooks, checkpoints passed):

What I LEARNED (one surprise counts double):

MY NUMBERS (accuracies, recalls, loss behavior):

HOMEWORK Mission before Session 2 (1–2 hrs)

1. **One stretch experiment:** make a plot of accuracy vs. how many training examples you use (try 100, 300, 700, 1347). Watch accuracy climb as data grows — that's Law #1 as a picture.
2. **Choose & save your Session 3 corpus:** 300+ words you love and have the right to use (song lyrics, your own writing, or public-domain text). You'll build a language model from it.
3. **Find one real-world AI-failure article** and bring the link/source.

Mission complete? ☐ Yes ☐ Partly — what I got done: _____

KEEP THIS Session 1 glossary

Word	Plain-English meaning
Features	The input numbers (here, 64 pixels per image).
Label	The correct answer the model is trying to predict.
Loss	A score for how wrong the model is right now. Training shrinks it.
Training / `.fit()`	The model studying the training data and adjusting itself.
Train / test split	Study set vs. hidden exam. The test grade is the only honest one.
Held-out accuracy	Percent correct on data the model never saw while training.
Confusion matrix	A grid of truth (rows) vs. the model's guesses (columns).
Recall (per class)	Of all the true X's, the fraction the model caught. Finds hidden weak spots.
Hyperparameter	A setting *you* choose (like hidden-layer size or number of rounds).
Neural network	A flexible model of connected math "neurons" that learns patterns.