Learning a Classifier

Practise image vectors and class scores, then investigate how a model could improve its predictions.

Scope. Part A practises slides 1–21 of the Week 5 lecture. Part B is a guided preview of selected ideas from slides 22–90. Each new rule is supplied; gradients and optimisation are not prerequisites.

Work individually. Use one Python script named STUDENTID_CV_Lab5.py. Replace STUDENTID with your own ID. Complete the TODOs. Record answers as # Task N comments in your script; use paper for initial hand calculations. All task data are included.

Timing. 120 minutes: setup and Part A 60; break 5; Part B 47; final explanation 8. Use NumPy and Matplotlib in your usual Python environment.

Task 1 Read an image as numbers

PART A • Previously taught • 10 minutes after 5 minutes of setup

This image contains 8 rows and 8 columns. Values run from 0 to 16; larger values mean more foreground in that cell. The correct digit label is 3. [See sources on page 8.]

```
import numpy as np
import matplotlib.pyplot as plt

image = np.array([
    [0, 0, 4,14,16, 5, 0,0],
    [0, 4,16,16,16, 8, 0,0],
    [0,12,12, 0,15, 8, 0,0],
    [0, 2, 1, 5,16,13, 1,0],
    [0, 0, 0, 1,11,15,11,0],
    [0, 0, 0, 0, 0,11,12,0],
    [0, 0, 2,13,12,16, 7,0],
    [0, 0, 3,16,15, 8, 0,0]], dtype=float)
plt.imshow(image, cmap="gray_r", vmin=0, vmax=16)
plt.axis("off")
plt.show()
# TODO: create vector using image.reshape(-1)
# TODO: create scaled by dividing vector by 16
# TODO: print image.shape, vector.shape, and scaled[:8]
```

Computer Vision · NUU

1. Run the display code. Flatten the image row by row, then scale its values to the interval from 0 to 1. Complete the three TODOs.

2. Write the two shapes and the first eight values before and after scaling. Which row of the original image do they represent?

3. What do raw values 12 and 16 become? Does flattening or scaling change the correct digit label? Explain.

Think. If training flattens row by row but prediction flattens column by column, will a weight still refer to the same pixel?

If imports are missing in your local environment: python -m pip install numpy matplotlib

Task 2 Calculate scores and a prediction

PART A • Previously taught • 15 minutes

Use a separate two-feature example from the lecture. It is not the 64-pixel vector from Task 1. The class order is A, B, C; the true class for this example is B.

$$z = Wx + b$$

A weight multiplies an input feature; a bias is added afterwards. Each row of W produces one class score, also called a logit.

Class	Feature 1 weight	Feature 2 weight	Bias
A	1	0	0
B	0	1	0
C	-1	1	1

The input is x = [2, 1]. For each class, multiply its two weights by the corresponding features, add the results, then add its bias.

Class	Your calculation	Score
A	1 × 2 + 0 × 1 + 0	
B		
C		

1. Complete the table by hand. Circle the largest score and write the predicted class. Is that prediction correct?

```
classes = np.array(["A", "B", "C"])
x = np.array([2., 1.])
W0 = np.array([[1., 0.], [0., 1.], [-1., 1.]])
b0 = np.array([0., 0., 1.])
# TODO: compute scores using W0 @ x + b0
# TODO: compute predicted_index using np.argmax(scores)
# TODO: print scores and classes[predicted_index]
# TODO: print W0.shape, x.shape, b0.shape, scores.shape
```

2. Complete the code and compare it with your hand calculation. Explain what each dimension of W0 represents.

3. Compare np.max(scores) with np.argmax(scores). Which gives a score, and which gives an index into classes?

Check. There must be one score for each of the three classes. The true label is used to judge the prediction; it is not an input feature.

Task 3 Separate a weight from a bias

PART A • Previously taught • 15 minutes

Keep the original W0 and b0 from Task 2. Change only one parameter per trial. The first feature weight for class B is W0[1, 0]: class B has row index 1, and feature 1 has column index 0.

1. Before coding, predict the change in B's score for all four rows. Write your predictions in the table.

Input x	Change from the original model	Predicted change in B score	Measured change
[2, 1]	Add 0.25 to B feature 1 weight		
[2, 1]	Add 0.25 to B bias		
[0, 1]	Add 0.25 to B feature 1 weight		
[0, 1]	Add 0.25 to B bias		

2. Run the first trial below. Edit the input and the single marked parameter line to check the other rows. Start each trial with fresh copies so changes do not accumulate.

```
x_trial = np.array([2., 1.])
before = W0 @ x_trial + b0
W_trial = W0.copy()
b_trial = b0.copy()
W_trial[1, 0] += 0.25   # one parameter change
after = W_trial @ x_trial + b_trial
print("Before:", before, "After:", after)
print("Change in B score:", after[1] - before[1])
# For a bias trial, replace the parameter-change line
# with: b_trial[1] += 0.25
```

3. Which change has a larger effect for input [2, 1]? Why does the weight change have a different effect for input [0, 1]?

4. Write one sentence completing this rule: "Changing a weight by an amount changes its class score by that amount multiplied by …" Then write the corresponding rule for a bias.

Check. Only B's score can change in these trials. If A or C changes, check the row you edited and whether W0 was accidentally modified.

Optional challenge. Predict the effect of the same weight change when feature 1 is −2, then verify with input [−2, 1]. Explain the sign.

Task 4 Turn scores into probabilities

PART A • Previously taught • 15 minutes

Softmax converts a score vector into positive probabilities that sum to 1. Use the supplied function. It subtracts the largest score before exponentiating to keep the calculation numerically stable.

```
def softmax(scores):
    shifted = np.asarray(scores, dtype=float)
    shifted = shifted - shifted.max()
    e = np.exp(shifted)
    return e / e.sum()

test_scores = np.array([2., 1., 0.])
# TODO: compute probs using softmax(test_scores)
# TODO: print np.round(probs, 4) and probs.sum()
# TODO: print classes[np.argmax(probs)]
```

1. Predict which rows below should give the same probabilities. Then complete the TODOs and test each row. Record probabilities to four decimal places and the sum before rounding.

Scores in A B C order	Probabilities in A B C order	Sum	Largest probability class or tie
[2, 1, 0]			
[1002, 1001, 1000]			
[4, 2, 0]			
[0, 0, 0]			

2. Compare the first two rows. Does adding the same constant to every score change the probabilities? What values does the function exponentiate in each case?

3. Compare the first and third rows. Has the winner changed? Has the probability assigned to the winner changed? Explain using the gaps between scores.

Check. Equal scores give probabilities of 1/3, 1/3, 1/3. This is a three-way tie. NumPy argmax returns the first maximum, so its output alone does not report every tied class.

A winner can have less than half

Now consider the already calculated probability vector [0.40, 0.35, 0.25]. Do not apply softmax to it again.

4. Which class wins? Must a winning class have probability above 0.50? Does the winning probability prove that the prediction is correct?

Take a 5 minute break before Part B.

Task 5 Discover a prediction penalty

PART B • Guided preview for the next lecture • 12 minutes

This is new material. We will explain it fully in the next lecture. Today, investigate a supplied rule using a calculator or the one-line Python helper below. No derivation is required.

Every row below describes the same example, whose true class is B. First judge each prediction from its probabilities; then measure it with a penalty.

$$L = -\ln(p_{true})$$

Use the probability assigned to the true class, which is p(B) here. The symbol ln means natural logarithm. A smaller penalty is better. For one labelled example, this rule is called cross-entropy loss.

Worked check. If p(B) = 0.50, then L = −ln(0.50) ≈ 0.6931. Use positive probabilities; all values in this task are above zero.

Model	Probabilities A B C	Predicted class	p of true class	Penalty L
1	[0.34, 0.36, 0.30]			
2	[0.55, 0.40, 0.05]			
3	[0.25, 0.60, 0.15]			
4	[0.95, 0.04, 0.01]			
5	[0.05, 0.90, 0.05]			

1. Before calculating, rank the models from smallest expected penalty to largest. Explain what quantity you used to order them.

2. Fill the predicted class and p of true class columns. Calculate the penalties to four decimal places and check your ranking.

```
def penalty(p_true):
    return -np.log(p_true)

print(penalty(0.50))   # worked check only
# Replace 0.50 with each row's probability of class B.
```

3. Compare Models 1 and 2. Which predicts correctly? Which has the smaller penalty? Explain how both observations can hold.

4. Why does Model 4 receive a large penalty? Which number in its probability vector matters for this calculation?

Preview question. If we can measure a prediction, how might we use that number to choose a better weight?

Task 6 Improve one weight by trying changes

PART B • Guided preview for the next lecture • 15 minutes

Return to the Task 2 model with input [2, 1] and true class B. You will try candidate changes, then judge them using the penalty from Task 5. This experiment does not require a gradient.

Change only the first-feature weight for B, originally 0. Let δ (delta) mean the amount added to that original weight. Each trial starts from W0, not the result of the previous trial.

$$z = [2,\ 1 + 2\delta,\ 0]$$

1. First calculate B's score and the largest-score class by hand for every row. Mark any tie explicitly. Predict which direction of change will help this example.

Change δ	B score	p(B)	Penalty L	Largest score class or tie
−0.25				
0.00	1.00	0.2447	1.4076	A
+0.25				
+0.50				
+0.60				

2. Use the supplied softmax and penalty helpers to fill the remaining columns. The baseline row is provided as a check. Change only delta between runs.

```
delta = 0.25
W_trial = W0.copy()
W_trial[1, 0] += delta
z_trial = W_trial @ np.array([2., 1.]) + b0
p_trial = softmax(z_trial)
print("Scores:", z_trial)
print("p(B):", p_trial[1], "Penalty:", penalty(p_trial[1]))
print("argmax class:", classes[np.argmax(z_trial)])
```

3. Compare δ = 0 and δ = +0.25. Has the predicted class changed? Has the penalty improved? What does this tell you about measuring progress?

4. Find the change at which A and B tie. What condition on δ makes B the unique winner? If argmax prints A at a tie, is B's score smaller?

Think. A change that helps this one example may affect another example differently. What additional evidence would you want before accepting it as a good model change?

Optional check. Try input [3, 1] with true class A, before and after δ = +0.25. Compare p(A) and its penalty. Reuse the helpers; choose index 0 for the true class A.

Task 7 Discover why step size matters

PART B • Guided preview for the next lecture • 13 minutes

Use a separate one-number toy problem. The goal is to make E(w) small. The best value is w = 0. This example isolates the size of an update; it is not the classifier from Task 6.

$$E(w) = \frac{1}{2}w^2; w_0 = 4$$

Use the following move rule, supplied for this activity. The symbol η (eta) is the step size, also called the learning rate. The next lecture will explain how such update rules are obtained.

$$w_{next} = w - \eta w$$

Worked step. With η = 0.5 and w = 4, the next value is 4 − 0.5 × 4 = 2. Its penalty is E(2) = 0.5 × 2² = 2.

1. Start again at w = 4 for every row. Calculate two successive moves by hand, then calculate the penalty after the second move. Use the new w for each successive move.

Step size η	Start w_0	After move 1 w_1	After move 2 w_2	Penalty E(w_2)
0.1	4			
0.5	4	2		
1.5	4			
2.2	4			

2. Compare the distance from zero and the penalty. Which step sizes make progress? Which makes the values grow farther from zero?

3. Compare η = 0.5 with η = 1.5. One can cross zero and still improve. What evidence in your calculations shows this?

Verify after the hand calculation

This short loop checks four moves. Change eta to examine the other rows. Predict the next value before running each case.

```
eta = 0.5
w = 4.0
for step in range(1, 5):
    w = w - eta * w
    print(step, "w =", round(w, 4),
          "penalty =", round(0.5 * w**2, 4))
```

Explain. Why might a very small move be inconvenient, and why might a large move fail? Use this experiment as your evidence. Do not assume its numeric step sizes work for every learning problem.

Task 8 Move an image without changing its label

PART B • Guided preview for the next lecture • 7 minutes plus 8 minutes to explain your work

Here is a new binary image example with two classes: Vertical and Horizontal. Image C is A shifted one column to the right; no line pixel is lost. The fixed weights below were chosen for this activity, not trained.

Original A

0	1	0	0
0	1	0	0
0	1	0	0
0	1	0	0

Right-shifted C

0	0	1	0
0	0	1	0
0	0	1	0
0	0	1	0

Vertical weights

0	+1	0	0
-1	0	-1	-1
0	+1	0	0
0	+1	0	0

Images: 0 = background, 1 = line.

Use the familiar weighted score rule: multiply each pixel by the weight in the same cell, then add all products. Both biases are zero. The Horizontal weights are the negatives of the Vertical weights, so its score is the negative of the Vertical score.

1. The pixels with value 0 contribute nothing. Add only the weights under the four pixels with value 1. Calculate the two class scores and predicted class for each image.

Image	Vertical score	Horizontal score	Predicted class	True class
A				Vertical
C				Vertical

2. Did the line's orientation change? Did the model's prediction change? Explain why these particular pixel weights can react differently after a shift.

3. Suggest one way a future model or its training examples might help it recognise a vertical line in different positions. A plain-language idea is enough; no CNN knowledge is required.

Explain your results

Use the final 8 minutes to finish your explanations and be ready to discuss three points when asked: how a score differs from a probability; how a penalty can improve before the winning class changes; and why a new image position can challenge fixed pixel weights.

Bring one question to the next lecture about how a model could find useful weight changes automatically. Keep your calculations and your Python script for the discussion.

Sources and data

Aligned with Mohammad Sabokrou, Week 5 Learning a Classifier, 90-slide course deck. Part A follows slides 1–21; Part B previews selected ideas from slides 22–90.

Task 1 image: UCI Optical Recognition of Handwritten Digits, E. Alpaydin and C. Kaynak, 1998, CC BY 4.0; reproduced as an 8 × 8 array. https://doi.org/10.24432/C50P49

NumPy argmax reference: https://numpy.org/doc/stable/reference/generated/numpy.argmax.html. Tasks 2–7 use small numerical teaching examples. Task 8 images and fixed weights are synthetic.