Week 3 flow

- NN crash course 2
- Lab 1 FAQ
- Lab 2 brief

Neural networks crash course 2

IOAI Training and Selection Programme 2025

Apr 5, 2025 Sat

Table of contents

Recap

Starting point: Equation of a straight line

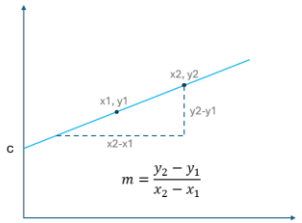
https://spmaddmaths.blog.onlinetuition.com.my/2020/08/6-4-equation-of-straight-lines-part-2.html

Equation in the gradient form

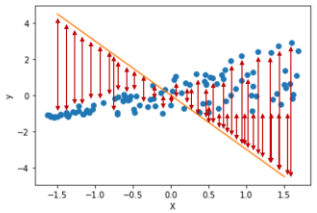
$y = mx + c$

m = gradient

c = y-intercept



Mean squared error (MSE) loss function



Mean Squared Error

$L = \sum (y - \hat{y})^2$

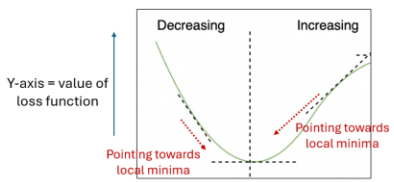
Calculate difference between true labels and prediction, then take sum of their squares

Smaller is better. Hence, we want to **minimize loss** by adjusting a

Gradient descent: find local minima w/ slope

If we can calculate a loss function, we can calculate its **derivative** i.e. rate of change

https://en.wikipedia.org/wiki/File:Tangent_function_animation.gif



https://www.geeksforgeeks.org/increasing-and-decreasing-intervals/

Iteratively adjust the value of a until we arrive at minimal loss
We call this **gradient descent**

Very rudimentary optimization loop

To find the best value of

```python
def f(a, x):
    return a * x

def mse(y, y_hat):
    return (y - y_hat) ** 2

def dl_da(a, x, y):
    return -2*x*y + 2*a*x ** 2
```

```python
history = []
a = -3 # Initial estimate
step_size = 1e-2 # Arbitrary step size

# Arbitrary num of loops over dataset i.e. epochs
for _ in range(5):
    # Iterate over every single data point
    for x, y in zip(X_sel, y_sel):
        # Get y_hat
        y_hat = f(a, x)

        # Calculate Loss
        current_loss = mse(y, y_hat)

        # Calculate Loss derivative
        current_loss_d = dl_da(a, x, y)

        # Modify a using loss derivative
        a = a - step_size * current_loss_d

        # Store values for plotting
        history.append((current_loss, a))
```

Optimization is a complex science. Google or ask ChatGPT about "operations research"

Deriving a general expression

2 features

$\hat{y} = a_1 x_1 + a_2 x_2 + b$

$\hat{y} = [x_1 \quad x_2] \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} + b$

3 features

$\hat{y} = a_1 x_1 + a_2 x_2 + a_3 x_3 + b$

$\hat{y} = [x_1 \quad x_2 \quad x_3] \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + b$

3 features, 2 labels

3 features but calculate 2 data points in one go

$\widehat{y_1} = a_1 x_{11} + a_2 x_{12} + a_3 x_{13} + b$

$\widehat{y_2} = a_1 x_{21} + a_2 x_{22} + a_3 x_{23} + b$

Original eqns as reference

$\begin{bmatrix} \widehat{y_1} \\ \widehat{y_2} \end{bmatrix}$ = (2, 1)

Need a (2,1) here!
Only possible if (2,3) x (3,1)

$+ \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$ (2, 1)

Hence $\begin{bmatrix} \widehat{y_1} \\ \widehat{y_2} \end{bmatrix} = \begin{bmatrix} x_{11} & x_{12} & x_{13} \\ x_{21} & x_{22} & x_{23} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$

General form

$\hat{\boldsymbol{Y}} = \boldsymbol{X}\,\boldsymbol{W} + \boldsymbol{b}$

for any dim of inputs and outputs

Chapter 2

Extending NNs to classification

Lets predict types of flowers

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
95	5.7	3.0	4.2	1.2	1
96	5.7	2.9	4.2	1.3	1
97	6.2	2.9	4.3	1.3	1
98	5.1	2.5	3.0	1.1	1
99	5.7	2.8	4.1	1.3	1

4 features

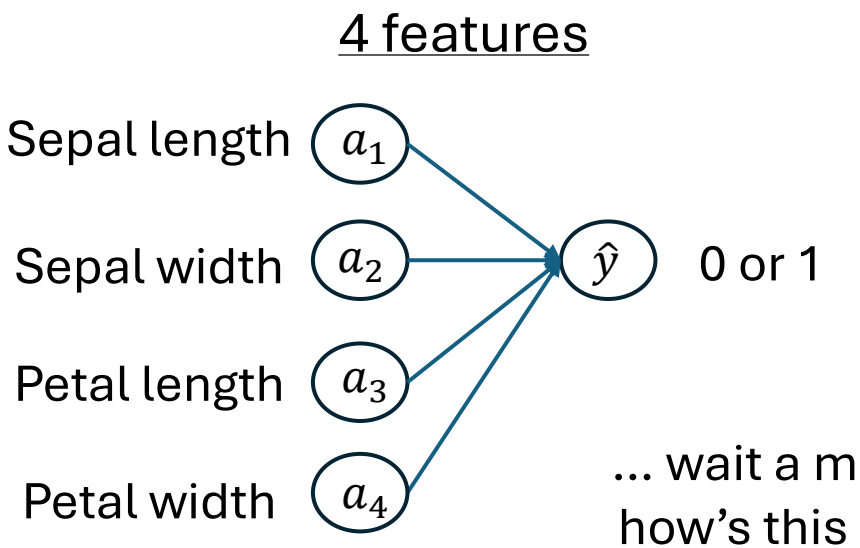
Class 0: iris setosa

https://en.wikipedia.org/wiki/Iris_setosa

Class 1: iris versicolor

https://plants.ces.ncsu.edu/plants/iris-versicolor/

Our architecture



… wait a minute, how's this going to work?

Class 0: iris setosa

https://en.wikipedia.org/wiki/Iris_setosa

Class 1: iris versicolor

https://plants.ces.ncsu.edu/plants/iris-versicolor/

Classification vs regression

Regression = predict real-valued numbers

Diabetes dataset from scikit-learn

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641	135.0

Iris dataset from scikit-learn

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
...	...	...	...	...	...
95	5.7	3.0	4.2	1.2	1
96	5.7	2.9	4.2	1.3	1
97	6.2	2.9	4.3	1.3	1

Classification = predict discrete classes

Transforming a regressor to a classifier

Technically speaking you can use any regressor as a classifier!
Most classifiers are regressors under the hood.
Just have to bucket the outputs into a range of classes

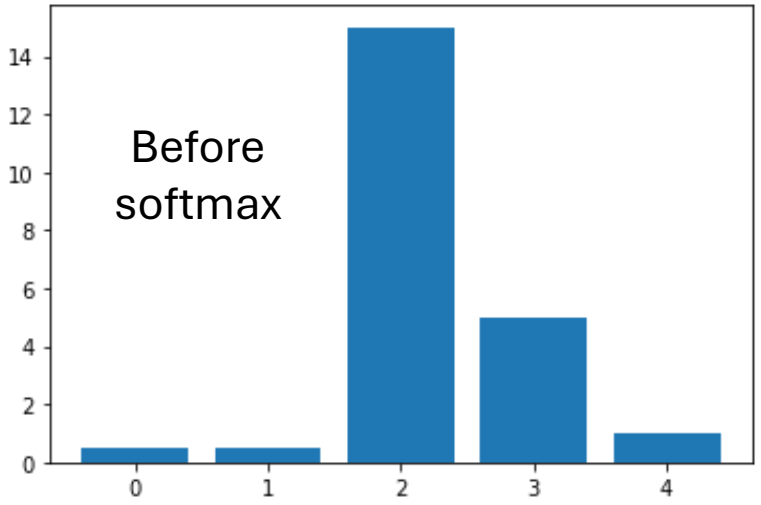
- But to do it properly, you need to constrain the [-inf, inf] output of a regression neuron into a bounded range.
- We can pipe all class activations through softmax and take the max.
- For predicting just two classes, piping output into a sigmoid* layer will directly give us 0 or 1. This is the special case of binary classification

*Note: Sigmoid (logistic function) has many wonderful uses! Look it up

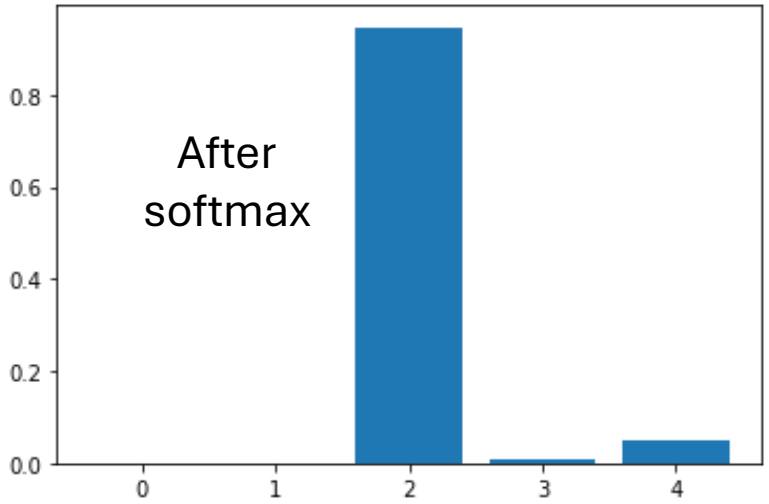
Softmax intuition

Allows us to think of outcomes as probability*

Rescales all class activations such that they sum to 1




```
a = torch.tensor([0.5, 0.5, 10, 5, 7])
```




```
import torch.nn.functional as F
print(F.softmax(a, dim=0))
# Pretty print in a more human-readable form
print([f"{i:.3f}" for i in F.softmax(a, dim=0)])
```

```
tensor([7.0837e-05, 7.0837e-05, 9.4636e-01, 6.3766e-03, 4.7117e-02])
['0.000', '0.000', '0.946', '0.006', '0.047']
```

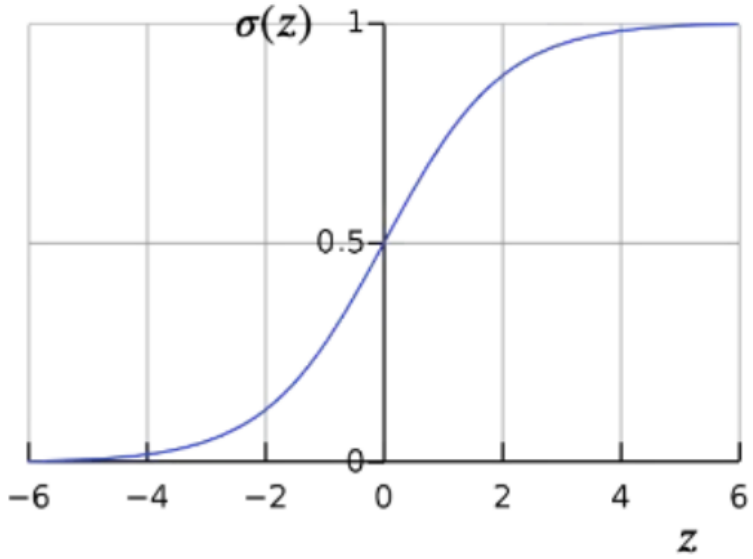
$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

https://en.wikipedia.org/wiki/Softmax_function

*Not strictly true but it's a convenient mental picture. See:
https://ai.stackexchange.com/questions/37889/are-softmax-outputs-of-classifiers-true-probabilities
https://stats.stackexchange.com/questions/309642/why-is-softmax-output-not-a-good-uncertainty-measure-for-deep-learning-models

Sigmoid (logistic function) intuition

Given class labels 0 and 1:



$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Predict class 1 if > 0.5
Predict class 0 if <= 0.5

Fig 4. Output sigmoid/logistic node

https://towardsdatascience.com/nothing-but-numpy-understanding-creating-binary-classification-neural-networks-with-e746423c8d5c

Why classification losses are different from regression losses

You can technically use MSE loss for classification, but you shouldn't* because :

https://towardsdatascience.com/why-using-mean-squared-error-mse-cost-function-for-binary-classification-is-a-bad-idea-933089e90df7

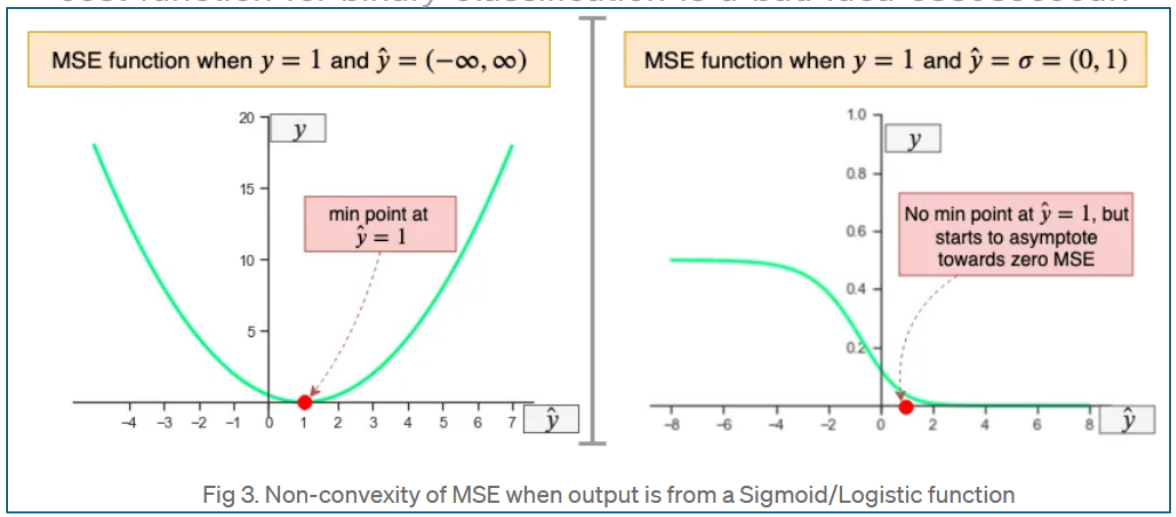


Fig 3. Non-convexity of MSE when output is from a Sigmoid/Logistic function

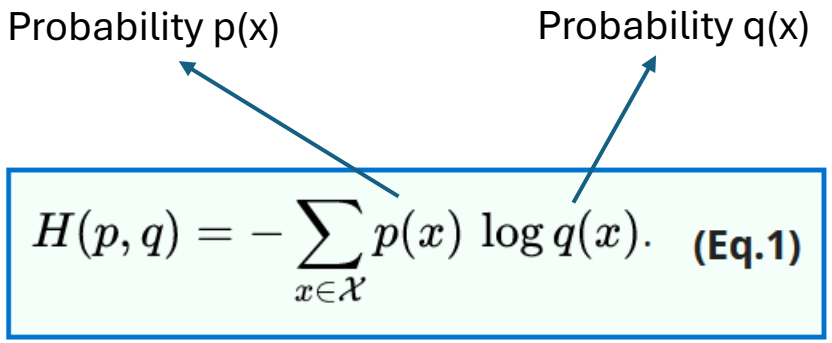
No guarantee of U-shaped valley discoverable by gradient descent, i.e. might be non-convex

*unless you know what you're doing and you find yourself in very specific situations where the drawbacks are nullified.

It can happen! https://ai.stackexchange.com/a/39802

Also called log loss, xent loss

Cross-entropy loss



https://en.wikipedia.org/wiki/Cross-entropy

Cross-entropy is a measure of

Likelihood of actual event (Our data, p(x)) X Likelihood of predicting event to be true (Our prediction, q(x))

I am choosing to completely skip the mathematical explanation for cross-entropy loss.
Learn more here: https://d2l.ai/chapter_linear-classification/softmax-regression.html

Cross-entropy loss example

$$H(p, q) = -\sum_{x \in \mathcal{X}} p(x) \log q(x). \quad \textbf{(Eq.1)}$$

https://en.wikipedia.org/wiki/Cross-entropy

y_{apple}	1	p_{apple}	0.8	$\hat{y}_{apple}$	1
y_{banana}	0	p_{banana}	0.1	$\hat{y}_{banana}$	0
y_{orange}	0	p_{orange}	0.1	$\hat{y}_{orange}$	0

$$Xent = -(y_a \ln(p_a) + y_b \ln(p_b) + y_o \ln(p_o))$$
$$= -(1 \times \ln(0.8) + 0 \times \ln(0.1) + 0 \times \ln(0.1))$$
$$= -\ln(0.8)$$
$$= 0.223$$

p_{apple}	0.25	$\hat{y}_{apple}$	0
p_{banana}	0.25	$\hat{y}_{banana}$	0
p_{orange}	0.5	$\hat{y}_{orange}$	1

$$Xent = -(y_a \ln(p_a) + y_b \ln(p_b) + y_o \ln(p_o))$$
$$= -(0 \times \ln(0.25) + 0 \times \ln(0.25) + 1 \times \ln(0.5))$$
$$= -\ln(0.5)$$
$$= 0.693$$

If you see 0.693 in the wild, it is from a perfectly confused binary classifier!

$$-\log(p_{true\ class})$$

Notice how Xent loss pushes classifiers to be more confident!

Derivative of binary cross-entropy loss

$$BCE\ loss, L = -(y \cdot \ln(\hat{p}) + (1 - y) \cdot \ln(1 - \hat{p}))$$

This is cross-entropy expanded for the special case where y can only be 0 or 1

where:

y is the ground truth class label

$\hat{p}$ is our predicted probability from sigmoid

$$\hat{p} = \frac{1}{1 + e^{-z}}$$

where:

z is the output from the previous layer

$$z = X \cdot W + b$$

where:

X is a matrix of features

W is a matrix of weights

b is the bias

Hold on! Are we seriously going to find the derivative of L as a function of $\hat{p}$, as a function of z, as a function of W?

Chain rule

Ultimately, we want to find the derivative of loss w.r.t our variables, W and b
We can build a chain of derivatives to arrive at what we need!
It's a bit like cancelling out fractions* **drastic but useful oversimplification*

Formulating what we need:

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial \hat{p}} \cdot \frac{d\hat{p}}{dz} \cdot \frac{\partial z}{\partial W}$$

$$\frac{\partial L}{\partial b} = \frac{\partial L}{\partial \hat{p}} \cdot \frac{d\hat{p}}{dz} \cdot \frac{\partial z}{\partial b}$$

Solving individually:

$$\frac{\partial L}{\partial \hat{p}} = \frac{1-y}{1-\hat{p}} - \frac{y}{\hat{p}}$$

$$\frac{d\hat{p}}{dz} = \hat{p}(1-\hat{p})$$

$$\frac{\partial z}{\partial W} = X \qquad \frac{\partial z}{\partial b} = 1$$

Piecing all expressions together:

$$\frac{\partial L}{\partial W} = \left(\frac{1-y}{1-\hat{p}} - \frac{y}{\hat{p}}\right) \cdot (\hat{p}(1-\hat{p})) \cdot X$$

$$\frac{\partial L}{\partial b} = \left(\frac{1-y}{1-\hat{p}} - \frac{y}{\hat{p}}\right) \cdot (\hat{p}(1-\hat{p}))$$

Footnotes

1. You must have a lot of questions of where all these expressions are popping up as I omit numerous derivation steps here. Be my guest if you want to work out the math, though I prefer you to spend time working on something else ☺
2. Note that we can specify dp/dz as a total derivative as the sigmoid function only has one variable

Forward prop and backward prop

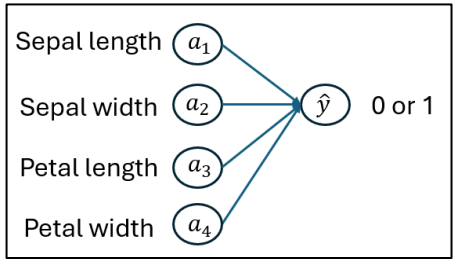
- Each layer boils down to a fwd prop and bwd prop operation
- Possible to stack a new layer just by defining its fwd prop and bwd prop!

$$z = X \cdot W + b \qquad \hat{p} = \frac{1}{1 + e^{-z}} \qquad L = -(y \cdot \ln(\hat{p}) + (1 - y) \cdot \ln(1 - \hat{p}))$$

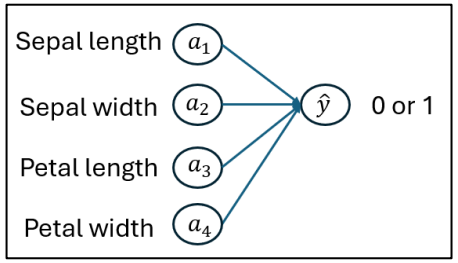
$$\frac{\partial z}{\partial W} = X \qquad \frac{d\hat{p}}{dz} = \hat{p}(1 - \hat{p}) \qquad \frac{\partial L}{\partial \hat{p}} = \frac{1 - y}{1 - \hat{p}} - \frac{y}{\hat{p}}$$

$$\frac{\partial z}{\partial b} = 1$$

Forward propagation to get predictions



Backward propagation to get gradients



In practice, you don't worry about any of this! Autograd takes care of all of it

Optimization loop for classification

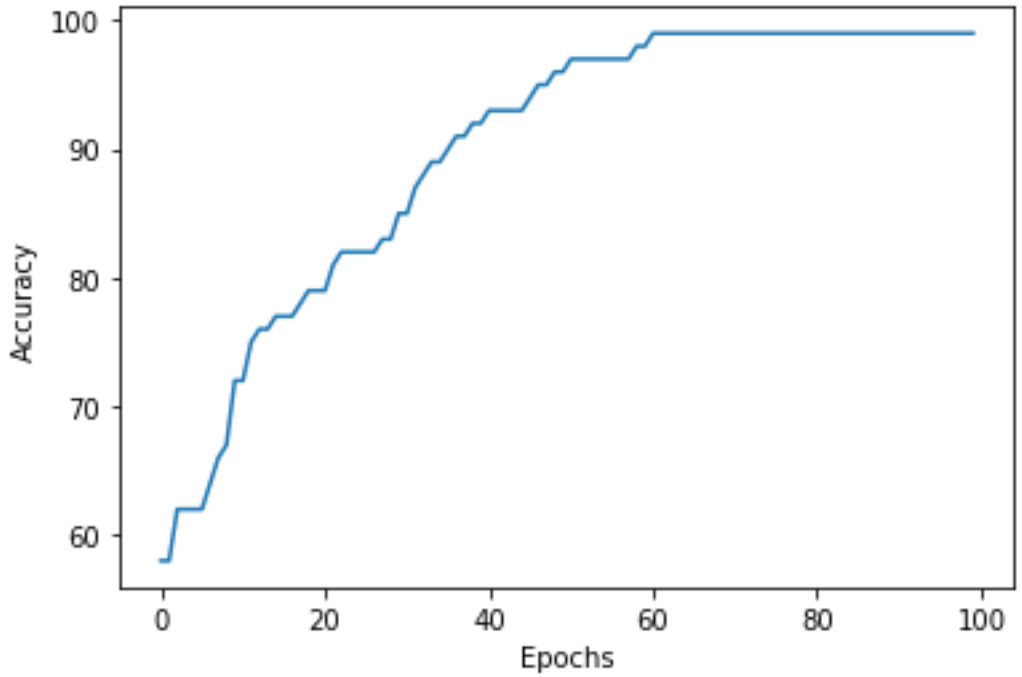
```
history = []

m = len(y)

# Initialize parameters with estimates
# -0.5 to center on -0.5 to 0.5
W = np.random.random(size=(4,)) - 0.5
b = np.random.random() - 0.5

# Arbitrary step sizes
step_size_W = 1e-2
step_size_b = 1e-2
```

```
def dL_dphat(y, phat):
    return (1 - y) / (1 - phat) - y / phat

def dphat_dz(phat):
    return phat * (1 - phat)
```

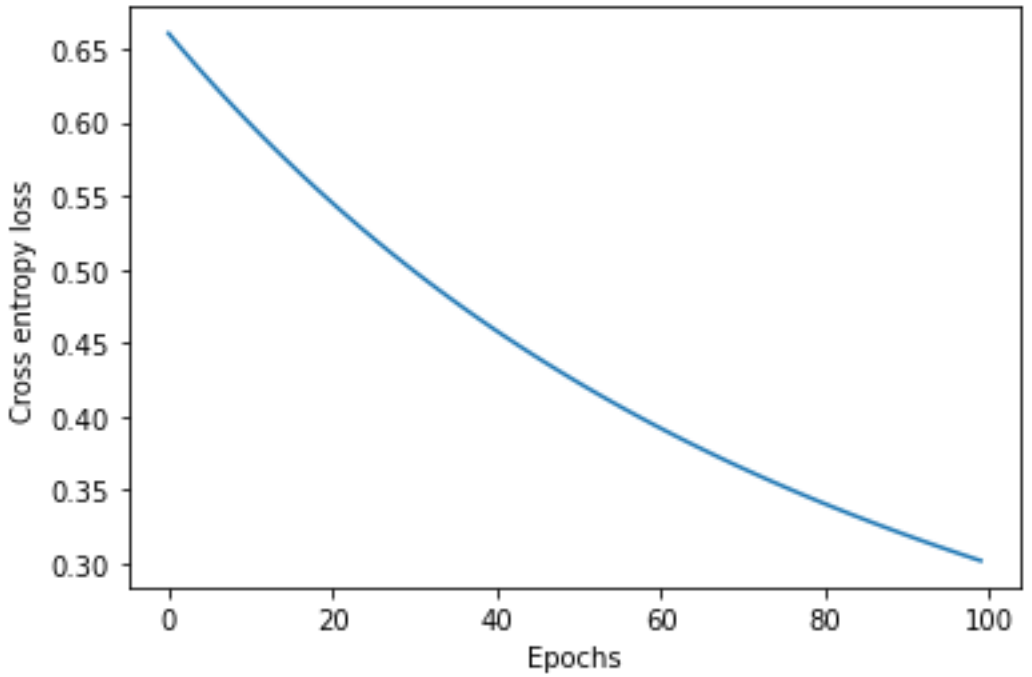
```
for _ in range(100):
    # Forward prop
    z = np.dot(X, W) + b
    phat = 1 / (1 + np.exp(-z))

    # Loss fxn and prediction
    xent_loss = -(y * np.log(phat) + (1 - y) * np.log(1 - phat))
    y_hat = np.greater(phat, 0.5).astype(float)

    # Backward prop
    dL_dW = np.expand_dims(dL_dphat(y, phat) * dphat_dz(phat), axis=-1) * X
    dL_db = dL_dphat(y, phat) * dphat_dz(phat)

    # Scale the gradients over all samples
    dL_dW_avg = np.mean(dL_dW, axis=0, keepdims=False)
    dL_db_avg = np.mean(dL_db, axis=0, keepdims=False)

    # Optimization step
    W = W - step_size_W * dL_dW_avg
    b = b - step_size_b * dL_db_avg

    # Calc accuracy and store values for plotting
    acc = accuracy_score(y, y_hat)
    history.append((np.mean(xent_loss, axis=0, keepdims=False), acc))
```

Optimization results for classification

Accuracy



Xent loss



Lab 1 discussion

IOAI Training and Selection Programme 2025

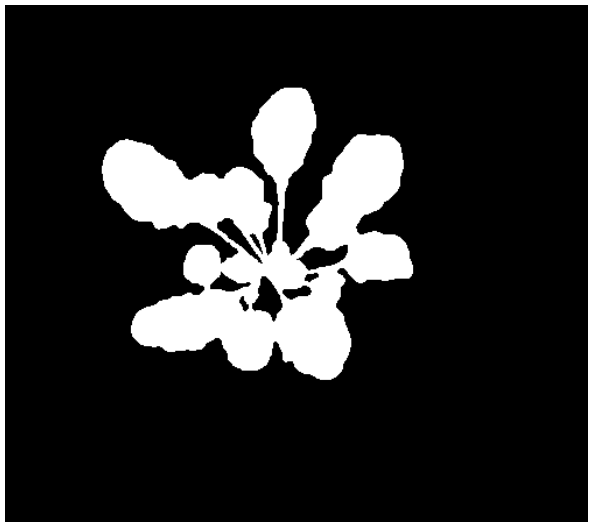
Apr 5, 2025 Sat

Lab 1 FAQ

- What am I supposed to do?
- Is it easy to get perfect performance ? Yes
- Why does my network predict 1000 classes?
- Why does my network have perfect performance without finetuning?
- How do I submit my work?

- What is 151?

On ImageNet: https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world



Lab 2 assigned

IOAI Training and Selection Programme 2025

Apr 5, 2025 Sat

Post-class addendum

IOAI Training and Selection Programme 2025

Apr 5, 2025 Sat

Entropy calculation example (p(x) = q(x))

$$H(p) = -\sum p(x) \log p(x)$$

Will it rain tomorrow in the Sahara?

$$P(rain) = 0.01$$
$$P(no\ rain) = 0.99$$
$$H = -(0.01 \times \ln(0.01) + 0.99 \times \ln(0.99))$$
$$= -(0.01 \times -4.605 + 0.99 \times -0.01)$$
$$= 0.036$$

Low information entropy! High certainty

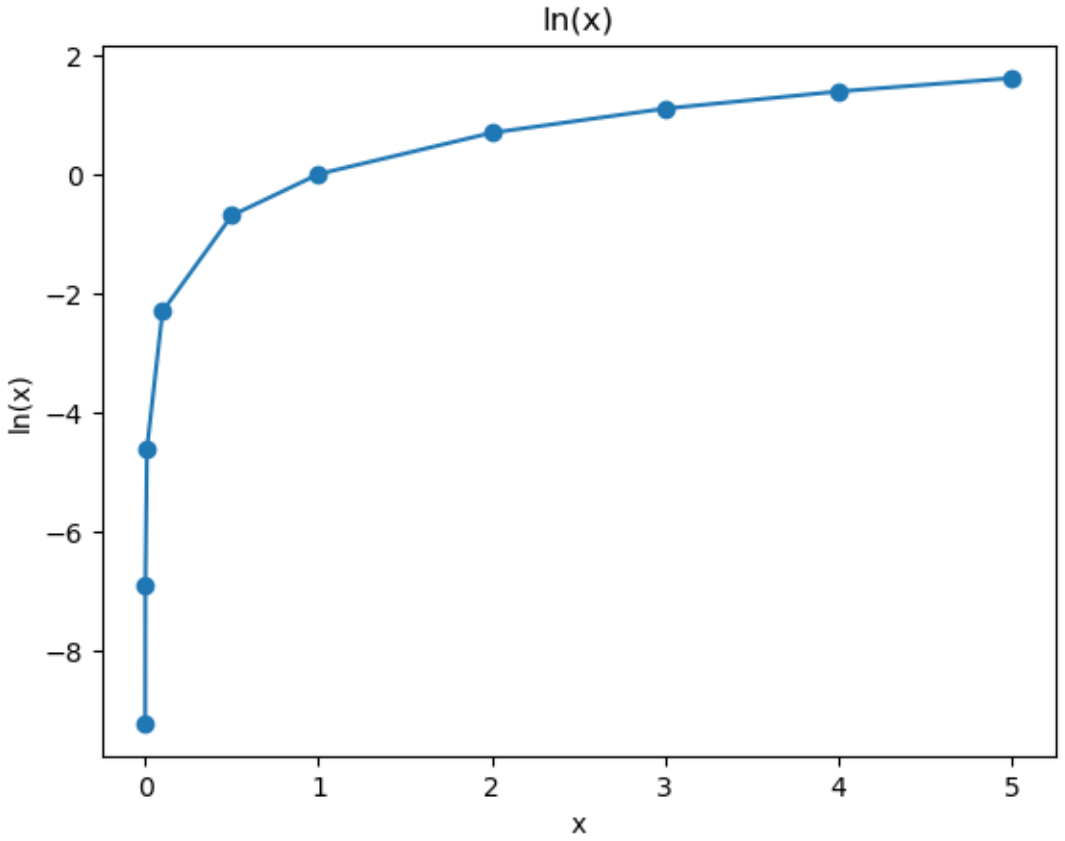
Will it rain tomorrow in KL?

$$P(rain) = 0.5$$
$$P(no\ rain) = 0.5$$
$$H = -(0.5 \times \ln(0.5) + 0.5 \times \ln(0.5))$$
$$= -(0.5 \times -0.693 + 0.5 \times -0.693)$$
$$= 0.693$$

High information entropy! Low certainty

In information theory, log base 2 is commonly used to express everything in number of bits. In machine learning we use natural log because the derivative is easier to express. The difference is just a constant scaling factor

Behaviour of natural log



- Derivative of ln(x) = 1/x
- X must be > 0
- ln(1) = 0
- If 0 < x < 1, ln(x) is negative -> most of the time in machine learning we are in this regime!