Exercise 4: Neural Networks — Simulation Exercises

How to use this sheet (important)

Each exercise has the same structure:

1. **Given data**: what values to generate in Python (numbers, points, labels).
2. **Model equations**: how to compute predictions from parameters.
3. **Loss**: how to measure error.
4. **Gradients**: formulas you will implement to update parameters.
5. **Update rule**: how to update parameters in a loop.
6. **What to plot/print**: exactly what output is required.

Common symbols used everywhere.

- N = number of data points (samples).
- $x^{(i)}$ = input for sample i (a number or a vector).
- $y^{(i)}$ = true target/label for sample i.
- $\hat{y}^{(i)}$ = model prediction for sample i.
- η = learning rate (step size).
- k = training iteration number.

Common training rule (gradient descent). Parameters are collected into θ (weights + biases). Training uses gradient descent:

$$\theta^{(k+1)} = \theta^{(k)} - \eta \nabla_\theta J(\theta^{(k)})$$

Recommended default settings (use unless told otherwise).

- Set random seed: `np.random.seed(0)`
- Use $N = 200$ data points when possible.
- Use $K = 1000$ iterations for training loops.
- Start with learning rate $\eta = 0.1$ for simple problems; reduce if loss diverges.

Exercise 1 — Neuron forward computation (no training)

1) Given data (what to create in Python)

- Choose weights $w = (1, -2)^\top$ and bias $b = 0.5$.
- Generate N random inputs $x^{(i)} \in \mathbb{R}^2$. Example idea: each coordinate is sampled from a standard normal distribution.

2) Model equations (forward pass)

For each input $x^{(i)} = (x_1^{(i)}, x_2^{(i)})^\top$ compute:

$$z^{(i)} = w^\top x^{(i)} + b = 1 \cdot x_1^{(i)} + (-2) \cdot x_2^{(i)} + 0.5$$

Then compute output:

$$a^{(i)} = \sigma\left(z^{(i)}\right)$$

3) Activations to implement

Compute $a^{(i)}$ using each of:

- Identity: $\sigma(z) = z$
- Sigmoid: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- ReLU: $\sigma(z) = \max(0, z)$

4) What to plot/print

- Print a small table for the first 10 samples: (x_1, x_2, z, a) for each activation.
- Make 3 histograms (one for each activation) showing the distribution of $a^{(i)}$.

Exercise 2 — Activation shapes and derivatives (no training)

1) Given data

Create a grid of z values from -6 to 6 (many points, e.g. 1000).

2) Compute activation values

Compute:

- Sigmoid: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- Tanh: $\tanh(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$
- ReLU: $\mathrm{ReLU}(z) = \max(0, z)$

3) Compute derivatives (slopes)

Compute:

- Sigmoid slope: $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Tanh slope: $(\tanh z)' = 1 - \tanh^2(z)$
- ReLU slope:

$$\mathrm{ReLU}'(z) = \begin{cases} 0, & z < 0, \\ 1, & z > 0. \end{cases}$$

(At $z = 0$ you may choose either 0 or 1; it does not matter for plots.)

4) What to plot/print

- Plot 1: $\sigma(z)$ curves (sigmoid, tanh, ReLU) on the same figure.
- Plot 2: derivative curves on the same figure.
- Print one sentence: where do gradients become small for sigmoid or tanh?

Exercise 3 — Regression with one neuron (learn a straight line)

1) Given data

- Generate N input scalars $x^{(i)} \in \mathbb{R}$ (e.g. uniformly in $[-2, 2]$).
- Create targets:

$$y^{(i)} = 2x^{(i)} + 1 + \varepsilon^{(i)}$$

  where $\varepsilon^{(i)}$ is small noise (e.g. normal noise with standard deviation 0.1).

2) Model

$$\hat{y}^{(i)} = wx^{(i)} + b$$

Parameters to learn: w and b.

3) Loss (mean squared error)

$$J(w, b) = \frac{1}{N} \sum_{i=1}^{N} \left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

4) Gradients

Let $e^{(i)} = \hat{y}^{(i)} - y^{(i)}$.

$$\frac{\partial J}{\partial w} = \frac{2}{N} \sum_{i=1}^{N} e^{(i)} x^{(i)}, \qquad \frac{\partial J}{\partial b} = \frac{2}{N} \sum_{i=1}^{N} e^{(i)}$$

5) Update rule (training loop)

Initialize $w = 0$, $b = 0$. For $k = 0, 1, \ldots, K - 1$:

1. Compute all predictions $\hat{y}^{(i)}$.
2. Compute loss $J(w, b)$ and store it.
3. Compute gradients $\partial J/\partial w$, $\partial J/\partial b$.
4. Update w, b using learning rate η.

6) What to plot/print

- Plot loss J vs iteration k.
- Plot data points $(x^{(i)}, y^{(i)})$ and the final fitted line $\hat{y} = wx + b$.
- Print final w, b and compare to $(2, 1)$.

Exercise 4 — Binary classification with one neuron (sigmoid)

1) Given data

- Generate $N/2$ points near $(-2, -2)$ and label them 0.
- Generate $N/2$ points near $(2, 2)$ and label them 1.
- Store all points in $X \in \mathbb{R}^{N\times 2}$ and labels in $y \in \{0, 1\}^N$.

2) Model

For each sample:

$$z^{(i)} = w^\top x^{(i)} + b, \qquad p^{(i)} = \sigma(z^{(i)}) = \frac{1}{1 + e^{-z^{(i)}}}$$

Interpretation: $p^{(i)}$ is the predicted probability of class 1.

3) Loss (cross-entropy)

$$J(w, b) = -\frac{1}{N}\sum_{i=1}^{N}\left[y^{(i)}\log p^{(i)} + (1 - y^{(i)})\log(1 - p^{(i)})\right]$$

Important for coding: use a small ε to avoid $\log(0)$.

4) Gradients

$$\frac{\partial J}{\partial w} = \frac{1}{N}\sum_{i=1}^{N}(p^{(i)} - y^{(i)})x^{(i)}, \qquad \frac{\partial J}{\partial b} = \frac{1}{N}\sum_{i=1}^{N}(p^{(i)} - y^{(i)})$$

5) Update rule (training loop)

Initialize $w = (0, 0)$ and $b = 0$. For $k = 0, 1, \ldots, K - 1$:

1. Compute $z^{(i)}$ for all points.
2. Compute probabilities $p^{(i)}$.
3. Compute loss $J(w, b)$ and store it.
4. Compute gradients and update w, b.

6) Prediction rule and accuracy

$$\hat{y}^{(i)} = \mathbf{1}_{p^{(i)}\geq 0.5}, \qquad \text{accuracy} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}_{\hat{y}^{(i)} = y^{(i)}}$$

7) What to plot/print (very explicit instructions)

- **(A) Plot loss vs iteration (one curve).**
  During training you run a loop for $k = 0, 1, \ldots, K - 1$. At each iteration:

  1. Compute probabilities $p^{(i)}$ for all training points using

  $$p^{(i)} = \sigma\Big(w^\top x^{(i)} + b\Big) = \frac{1}{1 + e^{-(w^\top x^{(i)} + b)}}.$$

2. Compute the cross-entropy loss

$$J^{(k)} = -\frac{1}{N}\sum_{i=1}^{N}\left[y^{(i)}\log p^{(i)} + (1-y^{(i)})\log(1-p^{(i)})\right].$$

3. Store the value $J^{(k)}$ in a list/array.

After training, plot the stored values $J^{(k)}$ (vertical axis) against iteration number k (horizontal axis). The loss should usually decrease.

- **(B) Plot the dataset points (scatter plot).**
  Make a 2D scatter plot in the (x_1, x_2) plane:
  - Plot all class 0 points (where $y^{(i)} = 0$) using one marker/color.
  - Plot all class 1 points (where $y^{(i)} = 1$) using a different marker/color.

  Add a legend showing which marker/color corresponds to class 0 and class 1.

- **(C) Plot the decision boundary line on the same figure.**
  After training finishes, you have final parameters (w, b) with $w = (w_1, w_2)^\top$. The decision boundary is the set of points $x = (x_1, x_2)$ such that

  $$w^\top x + b = 0.$$

  To draw it as a line on your plot:

  1. Choose a range of x_1 values covering your data (for example from $\min_i x_1^{(i)}$ to $\max_i x_1^{(i)}$).
  2. For each chosen x_1, compute the corresponding x_2 value:

  $$x_2 = -\frac{w_1}{w_2}x_1 - \frac{b}{w_2} \qquad \text{(only if } w_2 \neq 0\text{)}.$$

  3. Plot the curve (x_1, x_2) as a line over the scatter plot.

  Special case: if $w_2 = 0$, the boundary is a vertical line:

  $$w_1 x_1 + b = 0 \;\Rightarrow\; x_1 = -\frac{b}{w_1}.$$

  Plot this vertical line instead.

- **(D) Print final accuracy (one number).**
  After training, compute for each training point the predicted probability $p^{(i)}$ and convert it into a predicted class using threshold 0.5:

  $$\hat{y}^{(i)} = \begin{cases} 1 & \text{if } p^{(i)} \geq 0.5, \\ 0 & \text{if } p^{(i)} < 0.5. \end{cases}$$

  Then compute accuracy:

  $$\text{accuracy} = \frac{1}{N}\sum_{i=1}^{N}\mathbf{1}_{\hat{y}^{(i)}=y^{(i)}}.$$

  Print this accuracy value (example: `accuracy = 0.95`).

Exercise 5 — 1-hidden-layer regression network (tanh)

1) Given data

Generate inputs $x^{(i)} \in [-2, 2]$ and targets:

$$y^{(i)} = \sin(3x^{(i)}) + \varepsilon^{(i)}$$

2) Model (1 → L → 1)

Parameters:

$$W_1 \in \mathbb{R}^{1\times L}, \quad b_1 \in \mathbb{R}^{1\times L}, \quad W_2 \in \mathbb{R}^{L\times 1}, \quad b_2 \in \mathbb{R}.$$

Forward:

$$z_1 = xW_1 + b_1, \quad h = \tanh(z_1), \quad \hat{y} = hW_2 + b_2$$

3) Loss

$$J = \frac{1}{N}\sum(\hat{y} - y)^2$$

4) Gradients (use directly)

Define:

$$d\hat{y} = \frac{2}{N}(\hat{y} - y)$$

Then:

$$\frac{\partial J}{\partial W_2} = h^\top d\hat{y}, \quad \frac{\partial J}{\partial b_2} = \sum d\hat{y}$$
$$dh = d\hat{y}\, W_2^\top, \qquad dz_1 = dh \odot (1 - h^2)$$
$$\frac{\partial J}{\partial W_1} = x^\top dz_1, \quad \frac{\partial J}{\partial b_1} = \sum dz_1$$

5) Update rule

Initialize parameters with small random values and train for K iterations using learning rate η.

6) What to plot/print

- Plot true function $y = \sin(3x)$ and learned prediction $\hat{y}$.
- Plot loss J vs iteration.
- Repeat for $L = 5$ and $L = 20$ and compare.

Exercise 6 — Same network but ReLU hidden layer

1) Given data (what to create in Python)

Use the same dataset as in Exercise 5.

- Choose N input points on an interval, for example

$$x^{(i)} \in [-2, 2], \qquad i = 1, \ldots, N,$$

(you can take equally spaced points).

- Create target outputs using
$$y^{(i)} = \sin(3x^{(i)}) + \varepsilon^{(i)},$$
where $\varepsilon^{(i)}$ is a small noise term (optional; you may also set $\varepsilon^{(i)} = 0$).

2) Model (network structure: $1 \to L \to 1$)

This is the same architecture as Exercise 5: one hidden layer with L neurons and one output neuron.

Parameters (what you must store and update).

- $W_1 \in \mathbb{R}^{1\times L}$ (weights from input to hidden layer)
- $b_1 \in \mathbb{R}^{1\times L}$ (biases of hidden layer)
- $W_2 \in \mathbb{R}^{L\times 1}$ (weights from hidden layer to output)
- $b_2 \in \mathbb{R}$ (bias of output layer)

Forward pass (compute prediction $\hat{y}$). For all input points x (as a column vector of shape $N \times 1$):

$$\begin{aligned} z_1 &= xW_1 + \mathbf{1}b_1 \quad \in \mathbb{R}^{N\times L}, \\ h &= \mathrm{ReLU}(z_1) = \max(0, z_1) \quad \in \mathbb{R}^{N\times L}, \\ \hat{y} &= hW_2 + \mathbf{1}b_2 \quad \in \mathbb{R}^{N\times 1}. \end{aligned}$$

Here $\mathbf{1}$ is the $N \times 1$ vector of ones (used to add the bias to every row).

3) Loss (what you minimize)

Use Mean Squared Error (MSE), same as Exercise 5:

$$J = \frac{1}{N}\sum_{i=1}^{N}\left(\hat{y}^{(i)} - y^{(i)}\right)^2.$$

4) Gradients (formulas you implement to update parameters)

Step 1: derivative of loss w.r.t. output. Define the vector

$$d\hat{y} = \frac{\partial J}{\partial \hat{y}} = \frac{2}{N}(\hat{y} - y) \quad \in \mathbb{R}^{N\times 1}.$$

Step 2: gradients for the output layer (W_2, b_2).

$$\frac{\partial J}{\partial W_2} = h^\top d\hat{y} \quad \in \mathbb{R}^{L\times 1}, \qquad \frac{\partial J}{\partial b_2} = \sum_{i=1}^{N} d\hat{y}^{(i)} \quad \in \mathbb{R}.$$

Step 3: pass gradient back to hidden layer output h.

$$dh = d\hat{y}\, W_2^\top \quad \in \mathbb{R}^{N\times L}.$$

Step 4: ReLU derivative to get gradient w.r.t. z_1. ReLU is $h = \max(0, z_1)$, so its derivative is:

$$\frac{\partial h}{\partial z_1} = \mathbf{1}_{z_1>0}$$

(elementwise). Therefore:

$$dz_1 = dh \odot \mathbf{1}_{z_1>0} \quad \in \mathbb{R}^{N\times L},$$

where $\odot$ means elementwise multiplication, and $\mathbf{1}_{z_1>0}$ is a matrix with entries

$$(\mathbf{1}_{z_1>0})_{ij} = \begin{cases} 1 & \text{if } (z_1)_{ij} > 0, \\ 0 & \text{if } (z_1)_{ij} \le 0. \end{cases}$$

Step 5: gradients for the hidden layer (W_1, b_1).

$$\frac{\partial J}{\partial W_1} = x^\top dz_1 \quad \in \mathbb{R}^{1\times L}, \qquad \frac{\partial J}{\partial b_1} = \sum_{i=1}^{N} (dz_1)_{i,:} \quad \in \mathbb{R}^{1\times L}.$$

(Here $(dz_1)_{i,:}$ means the i-th row of dz_1.)

5) Update rule (what you do in the training loop)

Choose a learning rate $\eta > 0$ and number of iterations K (for example, $\eta = 0.01$ and $K = 5000$). Initialize W_1, W_2 with small random values and b_1, b_2 with zeros.

For each iteration $k = 0, 1, \ldots, K-1$:

1. Compute forward pass: z_1, h, $\hat{y}$.
2. Compute loss J and store it in a list (for plotting).
3. Compute gradients using the formulas in Section 4.
4. Update parameters (gradient descent):

$$W_1 \leftarrow W_1 - \eta \frac{\partial J}{\partial W_1}, \quad b_1 \leftarrow b_1 - \eta \frac{\partial J}{\partial b_1},$$
$$W_2 \leftarrow W_2 - \eta \frac{\partial J}{\partial W_2}, \quad b_2 \leftarrow b_2 - \eta \frac{\partial J}{\partial b_2}.$$

6) What to plot/print (clear deliverables)

- **Plot the learned curve:** after training, plot the target function $y = \sin(3x)$ (using the same x points) and on the same figure plot your network prediction $\hat{y}$.
- **Plot the loss curve:** plot the stored loss values $J^{(k)}$ versus iteration k.
- **Compare with Exercise 5:** use the same dataset, the same hidden width L, and similar learning rate. Write 2–3 lines describing which activation (tanh vs ReLU) gave a better fit or faster decrease of the loss.

7) Recommended settings (to avoid confusion)

- Use $N = 200$ points, equally spaced in $[-2, 2]$.
- Start with hidden width $L = 20$ (then try $L = 5$).
- Initialize weights with small random numbers (for example scale 0.1).
- Use $K = 5000$ iterations.
- Try learning rate $\eta = 0.01$ first. If loss explodes (increases a lot), reduce η to 0.001.

Exercise 7 — XOR with a small network ($2 \to 2 \to 1$)

1) Given data

Use four XOR points and labels:

$$(0,0) \mapsto 0,\ (0,1) \mapsto 1,\ (1,0) \mapsto 1,\ (1,1) \mapsto 0.$$

2) Model

Hidden:

$$z_1 = XW_1 + b_1, \quad h = \tanh(z_1)$$

Output:

$$z_2 = hW_2 + b_2, \quad p = \sigma(z_2)$$

3) Loss (cross-entropy)

$$J = -\frac{1}{N}\sum [y \log p + (1-y)\log(1-p)]$$

4) Gradients (use directly)

Key shortcut:

$$\frac{\partial J}{\partial z_2} = \frac{1}{N}(p-y)$$

Then:

$$\frac{\partial J}{\partial W_2} = h^\top \frac{\partial J}{\partial z_2}, \quad \frac{\partial J}{\partial b_2} = \sum \frac{\partial J}{\partial z_2}$$

$$dh = \frac{\partial J}{\partial z_2} W_2^\top, \quad dz_1 = dh \odot (1-h^2)$$

$$\frac{\partial J}{\partial W_1} = X^\top dz_1, \quad \frac{\partial J}{\partial b_1} = \sum dz_1$$

5) What to plot/print (very explicit instructions)

- **(A) Print predicted probabilities for the 4 XOR points.**
  Use the four input points

  $$(0,0),\ (0,1),\ (1,0),\ (1,1).$$

  After training is finished, compute the network output for each point step-by-step:

  $$z_1 = xW_1 + b_1, \qquad h = \tanh(z_1),$$

  $$z_2 = hW_2 + b_2, \qquad p = \sigma(z_2) = \frac{1}{1+e^{-z_2}}.$$

  Here $p \in (0,1)$ is the predicted probability of class 1 (label 1). **Print** a small table like:

  $$x = (0,0) \Rightarrow p = \dots, \quad x = (0,1) \Rightarrow p = \dots, \quad x = (1,0) \Rightarrow p = \dots, \quad x = (1,1) \Rightarrow p = \dots$$

  Expected: for true label 1, p should be close to 1 (e.g. > 0.9), and for true label 0, p should be close to 0 (e.g. < 0.1).

- **(B) Print accuracy (one number).**
  Convert each probability $p^{(i)}$ into a predicted class using threshold 0.5:

$$\hat{y}^{(i)} = \begin{cases} 1 & \text{if } p^{(i)} \geq 0.5, \\ 0 & \text{if } p^{(i)} < 0.5. \end{cases}$$

  Then compute accuracy:

$$\text{accuracy} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{1}_{\hat{y}^{(i)}=y^{(i)}},$$

  where $\mathbf{1}_{\hat{y}^{(i)}=y^{(i)}}$ equals 1 if the prediction is correct and 0 otherwise. **Print** the final accuracy as a number between 0 and 1 (example: `accuracy = 1.00`).
  Note: If you train only on the 4 XOR points (so $N = 4$), then accuracy = 1.00 means you classified all four points correctly.

- **(C) Plot loss vs iteration (one curve).**
  During training, at every iteration $k = 0, 1, \ldots, K-1$, compute the cross-entropy loss using the current network outputs $p^{(i)}$:

$$J^{(k)} = -\frac{1}{N} \sum_{i=1}^{N} \left[y^{(i)} \log(p^{(i)}) + (1 - y^{(i)}) \log(1 - p^{(i)}) \right].$$

  Store $J^{(0)}, J^{(1)}, \ldots, J^{(K-1)}$ in a list/array and **plot** $J^{(k)}$ (vertical axis) against the iteration number k (horizontal axis). The curve should usually decrease during training.

Exercise 8 — Check your gradients (numerical vs analytic)

Given

Choose Exercise 5 or 7.

Numerical gradient

Pick one parameter value θ and small $\delta = 10^{-5}$:

$$g_{\text{num}} = \frac{J(\theta + \delta) - J(\theta - \delta)}{2\delta}$$

Compare

Let g_{bp} be your gradient formula result:

$$\text{rel\_err} = \frac{|g_{\text{bp}} - g_{\text{num}}|}{|g_{\text{bp}}| + |g_{\text{num}}|}$$

What to print

Print g_{bp}, g_{num}, and rel_err. If rel_err is large, your gradients likely contain a mistake.

Exercise 9 — Learning rate experiment

Given

Use Exercise 3 or Exercise 5.

Task

Train the same model three times with:

$$\eta \in \{0.001,\ 0.01,\ 0.1\}$$

What to plot/print

- Plot loss vs iteration for all three learning rates on the same graph.
- Explain in 2–3 lines: which learning rate is too small (slow) and which is too large (unstable).

Exercise 10 — 3-class softmax classifier (linear)

1) Given data

Generate 3 clusters in $\mathbb{R}^2$ and labels $y^{(i)} \in \{0, 1, 2\}$.

2) Model

Parameters: $W \in \mathbb{R}^{2\times 3}$, $b \in \mathbb{R}^3$.
Scores:

$$S = XW + \mathbf{1}b^\top$$

Softmax:

$$P_{ik} = \frac{e^{S_{ik}}}{\sum_{j=1}^{3} e^{S_{ij}}}$$

One-hot labels Y:

$$Y_{ik} = 1 \text{ if } y_i = k,\ \text{else } 0.$$

3) Loss

$$J = -\frac{1}{N}\sum_{i=1}^{N}\sum_{k=1}^{3} Y_{ik}\log(P_{ik})$$

4) Gradients

$$\frac{\partial J}{\partial S} = \frac{1}{N}(P - Y)$$

$$\nabla_W J = X^\top \frac{\partial J}{\partial S}, \qquad \nabla_b J = \sum_{i=1}^{N}\left(\frac{\partial J}{\partial S}\right)_{i,:}$$

5) Prediction

$$\hat{y}^{(i)} = \arg\max_k P_{ik}$$

6) What to plot/print (very explicit instructions)

- **(A) Print training accuracy (one number).**
After training, for each data point $x^{(i)}$ compute the softmax probabilities P_{i0}, P_{i1}, P_{i2}. Then predict the class as the index of the largest probability:

$$\hat{y}^{(i)} = \arg\max_{k\in\{0,1,2\}} P_{ik}.$$

Count how many predictions are correct and divide by N:

$$\text{accuracy} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}_{\hat{y}^{(i)}=y^{(i)}}.$$

Print this accuracy value (for example: `accuracy = 0.93`).

- **(B) Plot loss vs iteration (one curve).**
During training, at every iteration $k = 0, 1, \ldots, K-1$, compute the cross-entropy loss

$$J^{(k)} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{m=0}^{2} Y_{im}\log(P_{im}),$$

using the current parameters $W^{(k)}, b^{(k)}$. Store the values $J^{(0)}, J^{(1)}, \ldots, J^{(K-1)}$ in a list/array. **Plot** the stored loss values on the vertical axis versus the iteration number k on the horizontal axis. The curve should usually go down.

- **(C) Plot decision regions (a colored background showing predicted class).**
This plot shows *which class the model would predict* at every location in the plane. Follow these steps:

1. Find the minimum and maximum of the data coordinates:

$$x_1^{\min}, x_1^{\max} \quad \text{and} \quad x_2^{\min}, x_2^{\max}.$$

Add a small margin (for example ± 1) so the plot has space around the data.

2. Create a grid of points covering that rectangle. For example, create 200 equally spaced values between $x_1^{\min} - 1$ and $x_1^{\max} + 1$, and 200 equally spaced values between $x_2^{\min} - 1$ and $x_2^{\max} + 1$. Each grid location is one 2D point $\tilde{x} = (\tilde{x}_1, \tilde{x}_2)$.

3. For each grid point $\tilde{x}$, compute its scores and softmax probabilities:

$$\tilde{s} = \tilde{x}^\top W + b^\top, \qquad \tilde{p}_k = \frac{e^{\tilde{s}_k}}{\sum_{j=0}^{2} e^{\tilde{s}_j}}.$$

Predict the class:

$$\tilde{y} = \arg\max_{k\in\{0,1,2\}} \tilde{p}_k.$$

4. Color the background according to $\tilde{y}$ (three colors: one for each class).

5. On top of this colored background, plot the original training points $x^{(i)}$ using markers/colors according to their true labels $y^{(i)}$.

Result: you will see three colored regions separated by boundaries. Those boundaries are where the predicted class changes.