Pre-Class Discovery: Solutions and Discussion Notes

Instructor copy – do not distribute before the lecture.

Data Science with Python – BSc Course

Task 1: XOR Problem by Hand

Model answer

(a) *No.* It is impossible to draw a single straight line that separates the two orange points at $(0, 1)$ and $(1, 0)$ from the two blue points at $(0, 0)$ and $(1, 1)$. Any straight line splits the plane into two half-planes — and the orange points lie on *opposite* corners of the square, so they cannot share a half-plane while excluding the blue points.

(b) A correct boundary is *not* a straight line. Reasonable student sketches include:

- Two lines forming a band or a V/X-shape.
- A curved (nonlinear) boundary.
- Two separate regions — one around each orange point.

(c) A single perceptron can only produce linear decision boundaries (half-planes). Since XOR is not linearly separable, a single perceptron *cannot* learn it. To solve XOR we need

- a **hidden layer** that transforms the inputs into a new feature space where the classes *become* linearly separable, or
- equivalently, a network with a **nonlinear activation** function and at least one hidden layer.

Common misconceptions

- Students sometimes draw a "diagonal wiggly line" and count it as a solution. Gently remind them: a perceptron is restricted to straight lines.
- Some students say the four points lie on a line, so a line "must exist." Lead them through the actual geometry: opposite corners $(0, 1)$ and $(1, 0)$ vs. $(0, 0)$ and $(1, 1)$ — no line can separate them.

Discussion note (links to lecture)

This task directly motivates the move from **L33 (Perceptron)** to **L34 (MLP and Activations)**: the XOR problem is the textbook demonstration that a single perceptron is insufficient and that we need hidden layers plus nonlinear activations. Revisit this XOR grid when introducing the universal approximation theorem.

Task 2: From Neuron to Network

Model answer

(a) Component roles:

- **Inputs (x_i):** the raw feature values of one example (pixel values, stock returns, customer features, etc.).
- **Weights (w_i):** learnable parameters that scale each input; they determine how strongly each input influences the output.
- **Sum-and-bias ($\Sigma + b$):** combines the weighted inputs and adds a shift so the neuron can activate even when all inputs are zero.
- **Activation:** a nonlinear function (ReLU, sigmoid, tanh, ...) that decides how strongly the neuron "fires." Without nonlinearity, stacked layers would collapse into a single linear map.

(b) Two-layer network: 3 inputs, 4 hidden neurons, 1 output.

- Input $\to$ hidden: $3 \times 4 = 12$ weights.
- Hidden $\to$ output: $4 \times 1 = 4$ weights.
- Biases: 4 (one per hidden neuron) + 1 (output) = 5.
- **Total parameters:** $12 + 4 + 5 = 21$.

(c) Doubling the hidden layer from 4 to 8 neurons:

- Input $\to$ hidden: $3 \times 8 = 24$ weights.
- Hidden $\to$ output: $8 \times 1 = 8$ weights.
- Biases: $8 + 1 = 9$.
- **New total:** $24 + 8 + 9 = 41$.

So the parameter count roughly doubles. Emphasize: wider/deeper networks grow very quickly in parameter count, which motivates the discussion of overfitting and regularization in L36.

Discussion note (links to lecture)

This task sets up **L33**'s diagram of the artificial neuron and **L34**'s MLP architecture. The parameter counting exercise naturally leads into the universal approximation theorem ("wider networks can represent more functions") and, later, the overfitting discussion ("more parameters mean more capacity to memorize noise").

Task 3: The Learning Loop

Model answer

(a) The *error* is the gap between where the ball landed and where the bucket is — distance and direction. The child uses the sign of this error ("too far" vs. "too short") to decide how to change the next throw.

(b) The *adjustment* is the change in throwing strength (and angle). Small adjustments are safer because a big change can easily overshoot the bucket in the opposite direction; gradual adjustments let the child home in on the right throw.

(c) In neural network training:

- *Error* corresponds to the **loss function** (e.g., squared error or cross-entropy).
- *Adjustment* corresponds to **weight updates** guided by the gradient of the loss.
- *Small adjustment* corresponds to a sensible **learning rate** — too large and training overshoots/oscillates; too small and training crawls.

The child behaviour we *do* want: iterating, measuring error, nudging parameters in the right direction. The behaviour we do *not* want: huge random jumps that overshoot (analogous to an oversized learning rate or unbounded gradients).

Common misconceptions

- Students think the network "knows" what to adjust. Clarify: the gradient (computed via backpropagation, L35) tells the network which direction to nudge each weight.
- Some students conflate "small adjustment" with "slow learning." In practice, the right learning rate balances stability and speed.

Discussion note (links to lecture)

Use this analogy when introducing **gradient descent** in L35 and **learning rate** in the MLP training loop. Return to it when discussing Adam, momentum, and learning rate scheduling.

Task 4: Sign of Trouble

Model answer

(a) Network B is overfitting. Training loss keeps decreasing while validation loss *increases* — the classic divergence signature. Network A shows a healthy gap (val slightly above train, both stable) which is normal.

(b) In the later epochs Network B is *memorizing* the specific patterns (and noise) in the training set. Its predictions fit the training examples almost perfectly but generalize worse and worse to new (validation) data. The model has become too specific to the training set.

(c) A simple **early stopping** rule:

> *"Stop training when validation loss has not improved for* `patience` *consecutive epochs, and restore the weights from the epoch with the best validation loss."*

For Network B, the best model is the checkpoint just *before* the validation loss started rising.

Common misconceptions

- Students sometimes think overfitting means "high training loss." It is the opposite: training loss is suspiciously low while validation loss is high.
- Some say Network A is "not learning" because training loss is higher than B's. Correct: A has *generalized* better; its train-val gap is small and stable.

Discussion note (links to lecture)

This task sets up the overfitting section in **L36**: the train/validation loss plot, the definition of the generalization gap, and the menu of countermeasures (early stopping, dropout, L2 regularization, data augmentation, batch normalization). Revisit the exact divergence shape when showing real training curves.

Task 5: Droplets of Noise

Model answer

(a) The *class average* should become more independent. Each student gets repeatedly forced to solve problems without their usual neighbours — they cannot free-ride on the same two loud classmates. When all students are present at the real exam, each has broader competence.

(b) Random silencing breaks *co-dependence*. If students always rely on the same two experts, the class fails when those experts are absent. Randomly silencing forces every student to build their own skills, so the class becomes *robust* to any particular member being unavailable.

(c) Translation to a neural network:

- Students ↔ **neurons** (or activations) in a hidden layer.
- Silencing ↔ **zeroing out** a random subset of activations during training (the exam analogy: at prediction/evaluation time, all neurons are active).
- Exam ↔ the **loss** or **validation** evaluation.

This is the intuition behind **dropout** (Srivastava et al., 2014). By randomly zeroing activations during training, the network cannot rely on any particular neuron being present, which prevents *co-adaptation* and acts as a strong regularizer.

Common misconceptions

- Students worry that zeroing out neurons "destroys information." Clarify: dropout is applied only during training; at test time the full network is used (with rescaled activations, or equivalently, inverted-dropout scaling during training).
- Some think dropout replaces all other regularization. Emphasize: dropout is one tool among several (L2, early stopping, batch norm, data augmentation). Modern practice often combines them.

Discussion note (links to lecture)

This task sets up the **dropout** section in **L36**. Use the classroom analogy when introducing the dropout layer and the inverted-scaling trick. Extend to ensemble intuition: "dropout trains an exponential family of sub-networks and averages their predictions at test time."

Task 6: Three Questions

Typical student questions and where they are answered

- "Why does deep learning need so much data?" — Lecture slides on capacity, overfitting, and the double-descent picture (L36).
- "How do you know how many hidden layers or neurons to use?" — Universal approximation and architecture guidelines (L34), model selection and regularization (L36).
- "What is backpropagation, really?" — Chain-rule walkthrough on a tiny network (L35).
- "Why do we need ReLU — wasn't sigmoid fine?" — Vanishing gradient problem and activation function comparison (L34).
- "Why does the network not just overfit?" — Regularization menu: dropout, L2, early stopping, batch normalization (L36).

- "Is deep learning always better than XGBoost?" — Tabular vs. image/text benchmarks (Grinsztajn 2022 result); discussed as a closing caveat in L36.
- "How does the network get its initial weights?" — Xavier/Glorot and He initialization (L35).
- "How fast is training in practice?" — Minibatch SGD, Adam, GPU acceleration (L35).

Instructor tip

Collect student questions at the start of the lecture and write them on the board. Revisit each question after the relevant section to confirm it has been addressed. Unanswered questions are a useful source of exam review topics and a signal of where the lecture may need more depth next semester.