

Topic 18: Artificial Intelligence

How learning really works, how game-playing programs prune the search, and classification by closeness.

Optional stretch material. This goes beyond the 9618 syllabus, so you will not be asked about it directly in the exam. Its purpose is to deepen ideas you already know. **Builds on:** 18.1 Artificial intelligence (machine learning, search algorithms). Try the challenge questions on the last pages before you read the answers.

Gradient descent: learning as rolling downhill

Training a model means finding weights that make its **loss** (error) as small as possible. **Gradient descent** does this repeatedly: calculate how the loss changes with each weight (the gradient) and nudge the weight in the downhill direction. With learning rate η : $\mathbf{w} \leftarrow \mathbf{w} - \eta \times (\mathbf{dL}/\mathbf{dw})$.

Example. Loss $L(w) = (w - 3)^2$, so the gradient is $2(w - 3)$ and the best weight is 3. Start at $w = 0$ with $\eta = 0.25$:

Step	w	Gradient $2(w-3)$	New $w = w - 0.25 \times \text{gradient}$
1	0	-6	1.5
2	1.5	-3	2.25
3	2.25	-1.5	2.625

The steps shrink as the slope flattens, converging on 3. With η too small learning crawls; with η too large it overshoots and may diverge (try $\eta = 1.1$: $0 \rightarrow 6.6 \rightarrow -1.3 \rightarrow \dots$). Back-propagation is just an efficient way to compute these gradients for every weight in a neural network.

Minimax and alpha-beta pruning

To play a two-player game a program searches a **game tree** of possible moves. **Minimax** assumes both sides play perfectly: at MAX's turn pick the child with the highest value, at MIN's turn the lowest, with leaf positions scored by an evaluation function.

Alpha-beta pruning gives the same answer while skipping branches that cannot matter. Example: MAX chooses between two MIN nodes. Left MIN has leaves 3 and 5, so its value is 3. Right MIN's first leaf is 2. Since MIN will pick 2 or lower, the right node can never beat 3, so its remaining leaves need not be examined. With good move ordering, pruning lets a program search about twice as deep in the same time.

k-nearest neighbours: classify by closeness

The simplest learning method: store all the labelled examples. To classify a new point, find the **k closest** examples (usually by Euclidean distance) and take a majority vote.

Point	Class	Distance to new point (4, 4)
A (1, 1)	Red	$\sqrt{18} \approx 4.24$
B (2, 1)	Red	$\sqrt{13} \approx 3.61$
C (5, 5)	Blue	$\sqrt{2} \approx 1.41$
D (6, 5)	Blue	$\sqrt{5} \approx 2.24$
E (5, 6)	Blue	$\sqrt{5} \approx 2.24$

With $k = 3$ the nearest are C, D, E, all blue, so (4, 4) is Blue. The choice of k balances two failures: a very small k follows noise (**overfitting**); a very large k blurs real boundaries (**underfitting**).

Challenge questions

1. $L(w) = (w - 4)^2$; start at $w = 0$ with $\eta = 0.1$. Perform two steps of gradient descent.
2. MAX picks between three MIN nodes with leaves (4, 6), (3, 8) and (5, 2) in left-to-right order. Evaluate with alpha-beta pruning. Which leaf can be skipped?
3. For a noisy dataset, which is more likely to overfit: $k = 1$ or $k = 15$ nearest neighbours? Explain.

Worked answers

Only look once you have had a proper go at each question.

1. Gradient = $2(w - 4)$. Step 1: gradient -8 , $w = 0 - 0.1 \times (-8) = 0.8$. Step 2: gradient $2(0.8 - 4) = -6.4$, $w = 0.8 + 0.64 = \mathbf{1.44}$.
2. First MIN node = $\min(4, 6) = 4$, so MAX has found at least 4. Second MIN node: first leaf $3 \leq 4$, so it can't beat 4: **the 8 is skipped**. Third node: leaf 5 then 2 gives 2. MAX's best option is the first node, so the root value is **4** (the pruned node is known to be at most 3, which cannot change that).
3. $k = 1$. It copies the label of the single closest example, so one mislabelled or unusual point creates a wrong region around it. With $k = 15$ the vote averages out individual noise, giving a smoother boundary (though too large a k underfits).