AI Research Labs

Deep Learning Foundations · Practice test 9 · 20 questions

Regularisation

How to use this test. Answer all 20 questions without notes, then mark yourself with the answer key at the back. Each question has exactly one best answer. Score 75% or more and you've mastered the material; below that, reread the lessons for the questions you missed.

9.1 L2 and L1 Penalties

Q1 `calculate` With L2 weight decay, $\eta = 0.1$ and $\lambda = 0.01$, by what factor is each weight shrunk before the usual gradient step?

- A 0.001
- B 0.99
- C 0.9
- D 0.999

Q2 `concept` Which penalty tends to set many weights to exactly zero?

- A Neither: penalties only shrink
- B Both, about equally often
- C L1, the sum of absolute weights
- D L2, the sum of squared weights

Q3 `concept` Why does L2 shrink small weights but almost never make them exactly zero?

- A It is turned off once the training loss stops falling
- B Its pull λw weakens as the weight gets smaller
- C It adds a constant to every weight at each step
- D It only applies to weights larger than one in size

Q4 `apply` You have 500 input features and suspect only a handful matter. Which penalty fits?

- A L1, to drive the useless weights to zero
- B L2 on the biases only, to shift each unit
- C L2, to make every weight equal in size
- D No penalty, since all features carry some signal

Q5 `concept` In the penalised objective $J(\theta) + \lambda\Omega(\theta)$, what happens as $\lambda \to \infty$?

- A The biases are driven to zero along with the weights
- B The model fits the training data perfectly
- C The penalty stops having any effect on training
- D All the penalised weights are driven towards zero

9.2 Early Stopping

Q6 `concept` In early stopping, which weights do you keep at the end?

- A Those from the epoch with the lowest training cost
- B Those from the very last epoch that was trained
- C The average of the weights over all of the epochs
- D Those from the epoch with the lowest validation cost

Q7 `calculate` The validation cost is lowest at epoch 31. With a patience of 15, when does training stop?

- A Epoch 46
- B Epoch 15
- C Epoch 31
- D Epoch 16

Q8 `concept` Why does early stopping count as regularisation?

- A It adds a penalty on large weights directly into the loss
- B It limits how far weights move from their small initial values
- C It randomly drops units in the network during training
- D It removes the hidden units that contribute least to the output

Q9 `apply` The validation cost ticks up for one epoch, then improves again. Why is a patience period useful here?

- A It avoids stopping on a brief noisy blip
- B It replaces the need for a validation set
- C It increases the learning rate gradually
- D It makes the training loss fall faster

Q10 `interpret` Besides preventing overfitting, what else does early stopping decide for you?

- A The learning rate for each layer
- B The size of each mini-batch
- C The number of hidden units
- D The number of training epochs

9.3 Dropout

Q11 `calculate` Inverted dropout with $p = 0.5$: activations $h = (2, 4, 1, 3)$, mask $(1, 0, 1, 0)$. What does the layer output during training?

- A $(2, 0, 1, 0)$
- B $(4, 8, 2, 6)$
- C $(4, 0, 2, 0)$
- D $(1, 0, 0.5, 0)$

Q12 `concept` In the lectured (inverted) dropout, what happens at test time?

- A Activations are multiplied by $1 - p$
- B Nothing is dropped and nothing is rescaled
- C Units are dropped with probability p
- D Only half of the units are used

Q13 `concept` What does dropout prevent, according to the usual explanation?

- A Dead ReLU units that output zero for every input
- B Exploding gradients caused by large learning rates
- C Co-adaptation: units relying on specific other units
- D Vanishing gradients in very deep sigmoid networks

Q14 `apply` Your validation accuracy looks noisy and lower than expected, and predictions change each run on the same input. What did you probably forget?

- A `optimizer.zero_grad()` before evaluating
- B To add dropout to the output layer too
- C To raise the dropout probability p
- D `model.eval()` before evaluating

Q15 `interpret` An older answer key says "each neuron is kept with probability p and activations are scaled at test time". How does it relate to the lectured version?

- A It is wrong: dropout never rescales anything
- B Different convention; same expected activations
- C It applies only to convolutional networks
- D It is the same convention in different words

9.4 Data Augmentation

Q16 `apply` Which augmentation would break the labels of a handwritten-digit dataset?

- A Slight zooms of a few percent
- B Small brightness changes
- C Shifts of a pixel or two
- D 180° rotations

Q17 `concept` When are augmentations applied?

- A Only at test time, to make predictions stable
- B Randomly to each training mini-batch only
- C Only to the validation set, to tune them
- D Once, before training, to the test set too

Q18 `concept` What does augmentation teach the network?

- A To reproduce the exact transformations on its output
- B To memorise every training image pixel by pixel
- C To ignore the labels and learn only from the inputs
- D Invariance to changes that shouldn't affect the label

Q19 apply Which augmentation suits an audio keyword-spotting dataset?

- A Swapping the labels of similar keywords
- B Small time shifts and added background noise
- C Reversing each recording backwards in time
- D Horizontal flips of the spectrogram image

Q20 interpret After adding augmentation, training error rises slightly but validation error falls. What does that show?

- A The validation set is now leaking into training
- B The network has become underfitted and too small
- C The regularisation is working as intended
- D The augmentation is broken and should be removed

Answer key

Each answer comes with the reasoning, including why the most tempting wrong option is wrong.

Q1 **D** · lesson 9.1

$W \leftarrow (1-\eta\lambda)W - \eta\,\partial J/\partial W$, and $1 - 0.1 \times 0.01 = 0.999$. 0.99 uses λ alone, 0.9 uses η alone, and 0.001 is $\eta\lambda$ itself, the amount removed rather than kept.

Q2 **C** · lesson 9.1

L1's gradient $\lambda\,\mathrm{sign}(w)$ pulls with constant force even on tiny weights, driving them to exactly zero (sparsity). L2's pull λw fades as w shrinks, so weights get small but rarely reach zero.

Q3 **B** · lesson 9.1

The gradient of $\frac{\lambda}{2}w^2$ is λw: as $w \to 0$, so does the pull, and the weight approaches zero without reaching it. L1's pull stays at size λ, which is why L1 produces exact zeros.

Q4 **A** · lesson 9.1

L1's sparsity switches off unhelpful inputs, a form of feature selection. L2 shrinks all weights but keeps every feature in play; penalising biases doesn't address irrelevant inputs.

Q5 **D** · lesson 9.1

An enormous penalty outweighs the data term, so the weights shrink to zero and the model underfits. $\lambda = 0$ is the unpenalised problem. Biases aren't penalised, so they're free to fit the mean.

Q6 **D** · lesson 9.2

You save the weights whenever the validation cost reaches a new low and restore them at the end. The last epoch has trained past the best point; the lowest training cost usually means the most overfitted model.

Q7 **A** · lesson 9.2

Training continues until 15 epochs pass without improvement: $31 + 15 = 46$. You then restore the weights from epoch 31. Stopping at 31 would require knowing in advance that no later epoch improves.

Q8 **B** · lesson 9.2

Weights start small and grow during training; stopping early keeps them closer to their start, which limits how complex the function becomes. For a linear model it's equivalent to L2. Dropping units is dropout; a loss penalty is L1/L2.

Q9 **A** · lesson 9.2

Validation cost is noisy, especially with mini-batches; patience waits a few epochs before concluding training has stopped helping. Since the best weights are restored, waiting costs only time.

Q10 **D** · lesson 9.2

Stopping at the best validation epoch effectively chooses how long to train, one hyperparameter fewer to tune. It doesn't set the learning rate, width or batch size.

Q11 **C** · lesson 9.3

Kept units are scaled by $1/(1-p) = 2$: $(2 \cdot 2, 0, 2 \cdot 1, 0) = (4, 0, 2, 0)$. $(2, 0, 1, 0)$ forgets the scaling, $(1, 0, 0.5, 0)$ scales the wrong way, and $(4, 8, 2, 6)$ ignores the mask.

Q12 **B** · lesson 9.3

The $1/(1-p)$ scaling during training already keeps the expected activations right, so the full network is used unchanged at test time. Scaling by $1-p$ at test time is the older convention.

Q13 **C** · lesson 9.3

If any unit may vanish, none can depend on particular partners, so each must be useful alone, and units can't jointly fit noise. Dropout also trains an implicit ensemble of sub-networks. It isn't a fix for gradient pathologies.

Q14 **D** · lesson 9.3

In training mode, dropout keeps dropping units, so evaluation is random and worse. `model.eval()` switches dropout off. `zero_grad` affects training, and dropout never belongs on the output layer.

Q15 **B** · lesson 9.3

The older form uses p = keep probability and scales at test time; the lectured inverted dropout uses p = drop probability and scales by $1/(1-p)$ in training. Both preserve expected activations; use the lecture's convention and say which you use.

Q16 **D** · lesson 9.4

Rotating a 6 by 180° turns it into a 9, so the label is wrong. Small shifts, brightness and zoom changes keep the digit's identity, which is the requirement for any augmentation.

Q17 **B** · lesson 9.4

Applying fresh random transformations each mini-batch means the network rarely sees the same example twice. Evaluation must use the real inputs, so validation and test data are never augmented.

Q18 **D** · lesson 9.4

By seeing shifted, cropped or brightened copies with the same label, the network learns that those changes don't matter. Augmentation keeps the labels; it discourages memorisation rather than encouraging it.

Q19 **B** · lesson 9.4

Shifting a recording slightly or adding noise doesn't change which word was spoken. Reversing the audio (or flipping a spectrogram in time) produces a different sound, and swapping labels corrupts the data.

Q20 **C** · lesson 9.4

Regularisation trades a little training performance for better generalisation: the task got harder, so training error rose, and the model generalises better, so validation error fell. That's success, not a bug.