Commented Answer Key

Final Exam (FE) · example 2026.2

14 multiple-choice questions (7.0) + 1 open-ended question in three parts (3.0) = 10.0 points

How to use it. Solve the whole exam on paper, timing 120 minutes, *before* opening this answer key. Check the multiple-choice questions against the table and read the comment on each one, including the ones you got right. Then grade the open-ended question with the criteria of each part.

How to read it. ✓ **Why** explains the right answer; × **Pitfalls** shows what is wrong with the most tempting alternatives. Each question names the class to review; the links open the class on the site.

Multiple-choice questions

Q1 to Q14 · 0.5 point each

question	1	2	3	4	5	6	7	8	9	10	11	12	13	14
answer	B	D	A	E	C	B	E	A	D	C	A	C	B	D

Q1 B — Two updates; $z = 0$ counts as 1

review: **4. Perceptron**

✓ **Why.** Examples 1 ($z = 1$) and 2 ($z = -1$) already come out right. At 3, $z = -1$ gives $\hat{y} = 0$ with $y = 1$: error $+1$, $\mathbf{w} \leftarrow (1, -1) + 0.5\,(0, 1) = (1, -0.5)$ and $b \leftarrow 0.5$. At 4, $z = 1 - 1.5 + 0.5 = 0$; since $z \geq 0$ gives $\hat{y} = 1$, there is an error of -1: $\mathbf{w} \leftarrow (1, -0.5) - 0.5\,(1, 3) = (0.5, -2)$ and $b \leftarrow 0.5 - 0.5 = 0$.

× **Pitfalls. A**: treating $z = 0$ as class 0 skips the second update; the step from class returns 1 at $z \geq 0$. **C**: using the error with the sign flipped ($\hat{y} - y$ instead of $y - \hat{y}$). **E**: updating $\mathbf{w}$ and forgetting the bias at example 4. **D**: adding $\eta\,\mathbf{x}$ on a false positive, when the -1 error calls for subtracting.

Q2 D — XOR with one hidden layer

review: **5. Multi-Layer Perceptron**, **3. Neural Networks**

✓ **Why.** h_1 fires only at $(1, 0)$ and h_2 only at $(0, 1)$; the output is the OR of the two. The column is 0, 1, 1, 0: XOR, the non-linearly separable function that Minsky and Papert (1969) used against the single-layer perceptron. With one hidden layer it comes out; what made it possible to train such networks from data was the backpropagation of Rumelhart, Hinton and Williams (1986).

× **Pitfalls. E**: XOR is not linearly separable; no line leaves $(0, 1)$ and $(1, 0)$ on one side with $(0, 0)$ and $(1, 1)$ on the other. **C**: the sigmoid is nonlinear too; universal approximation (Cybenko, 1989) was proved precisely for it. **B**: at $(1, 1)$ both hidden units stay at 0.

Q3 A — The largest that does not blow up, divided by two

review: **6. Optimization**

✓ **Why.** It is the search the class recommends: sweep η in powers of ten for a few hundred steps, keep the largest one that does not blow up and divide it by two. 10^{-1} went past the stability limit (the loss rises from step 12 on and becomes NaN); 10^{-2} goes down fast and stably; hence about $5 \cdot 10^{-3}$ for the long run.

× **Pitfalls. C**: a loss falling slowly and smoothly is the symptom of an η that is too small. **B**: the initial drop of 10^{-1} does not last; the loss climbs back before step 20. **E**: in AdamW the step has a size of the order of η whatever the scale of the gradient (it divides by the root of the mean of the squares), so clipping the gradient does not shorten the step; the class says to divide η by 10 first.

Q4 E Augmented validation measures something else

review: **7. Regularization**

✓ **Why.** C and D have the same training (92%); only the validation changes. In C, each validation photo is randomly cropped and recolored, so the number measures the model on distorted photos, not on the ones it will receive. With clean validation (D), the same model scores 91%, 5 points above B: data augmentation helped. The rule from class is never to augment the validation set.

× **Pitfalls. A**: compares B with a broken measurement. **C**: validation should mimic the photos that arrive, with no random draws; crop and color drawn anew at each evaluation are not what the user sends. **B** and **D**: the target is validation, not training; 100% on training with 81% on validation is overfitting.

Q5 C The factor of one layer is $n\sigma^2\rho$

review: **9. Deep Networks**

✓ **Why.** Each layer multiplies $\mathbb{E}[a^2]$ by $n\sigma^2\rho$, with $\rho = \frac{1}{2}$ for ReLU. With $\sigma^2 = 0.01$ fixed: $100 \cdot 0.01 \cdot \frac{1}{2} = 0.5$, then 1 and 2. Over 19 crossings, $0.5^{19} \approx 2 \cdot 10^{-6}$ and $2^{19} \approx 5 \cdot 10^5$, the orders of magnitude in the plot. Xavier and He put n in the denominator precisely to cancel the width: with $\sigma^2 = 2/n$ the factor equals 1 for any n.

× **Pitfalls. D**: forgetting the $\frac{1}{2}$ of ReLU. **B**: ReLU does not saturate on the positive side. **E**: nothing was trained; the learning rate plays no part in initialization. **A**: the depth is the same in all three networks.

Q6 B Shapes from the formula; global pooling

review: **10. Convolutional Networks**

✓ **Why.** With stride 2 and padding 1: $\lfloor (32 + 2 - 3)/2 \rfloor + 1 = 16$, so $32 \times 16 \times 16$; the second pooling leads to $32 \times 8 \times 8$ and the `Flatten` to 2,048 numbers. The `Linear` has $2048 \cdot 10 + 10 = 20{,}490$ parameters. Global pooling averages each map: 32 numbers are left, with $32 \cdot 10 + 10 = 330$ parameters.

× **Pitfalls. A**: forgetting the second `MaxPool2d`. **C**: forgetting the padding in the stride-2 convolution ($\lfloor 29/2 \rfloor + 1 = 15$, then 7). **D**: forgetting the bias and treating global pooling as a single number (it gives one per channel). **E**: global pooling changes precisely the input of the head.

Q7 E LoRA trains $r(d + k)$ per matrix

review: **14. Transfer Learning**, **13. Vision Transformers**

✓ **Why.** Each 768×768 matrix gains B (768×4) and A (4×768): $r(d + k) = 4 \cdot 1536 = 6{,}144$. There are 4 matrices per block over 12 blocks: $48 \cdot 6{,}144 = 294{,}912$, about 0.3% of the 86.6 million. The AdamW state follows the *trainable* parameters: $294{,}912 \cdot 12 \approx 3.5$ MB, against 86.6 million $\cdot 12 \approx 1$ GB in full fine-tuning.

× **Pitfalls. A**: counting only one of the two factors. **B**: $\Delta W = BA$ is never stored or trained; what is trained is B and A. **C**: freezing means precisely having no gradient and no optimizer state. **D**: the 12 bytes already include the two moments, stored in fp32 along with the master copy.

Q8 A Picking modes or covering everything

review: **16. Generative Models**

✓ **Why.** Z costs 50 passes per sample: it is the flow matching model, whose cost is the number of steps. Of the two one-pass models, X puts sharp samples on only 3 of the 8 modes: mode collapse, the behavior of the reverse KL, typical of the GAN. Y covers the whole ring, including the valley between the clusters: the blur of maximum likelihood (forward KL), typical of the VAE.

× **Pitfalls. C**: Y covers all the modes; collapse is the opposite. **B**: blur spreads the samples out, it does not concentrate them. **E**: blur and collapse are properties of the objective, not of short training. **D**: 50 passes per sample is the cost of diffusion or flow; the VAE generates in one pass.

Q9 D Zero KL in a dimension: it carries nothing

review: **17. Variational Autoencoders**

✓ **Why.** The KL of a single dimension is zero only when $q(z_i \mid x) = \mathcal{N}(0,1)$, the prior itself, for every image: there the code is the same for all images and says nothing about x. This is posterior collapse in 4 of the 8 dimensions; the other 4 (about 10 nats in total) carry the information.

× **Pitfalls. C**: $\sigma \to 0$ makes the KL blow up (the $-\log\sigma^2$ term), not vanish. **A**: KL near zero means precisely carrying no information. **B**: total collapse would have a total KL near zero; here half of the dimensions are at work. **E**: a larger β pushes more dimensions toward the prior; to reactivate them one uses a smaller β, KL annealing or free bits.

Q10 C The minimax loss saturates when D wins

review: **18. GANs**

✓ **Why.** `bce(`ℓ`, 0)` $= -\log(1-D)$, so `loss_G` $= \log(1 - D(G(z)))$: the generator's minimax loss. Its gradient at the logit is $-D(G(z)) \approx -0.02$, almost nothing, precisely when the discriminator is winning. The non-saturating loss, `bce(D(fake), ones)` $= -\log D(G(z))$, has the same fixed point and a gradient of $D - 1 \approx -0.98$.

× **Pitfalls. A**: it is the paper's objective, but the paper itself replaces this loss with the non-saturating one in practice. **D**: the sign is right (minimizing $\log(1-D)$ pushes $D(G(z))$ toward 1); removing the minus would make the generator help the discriminator. **E**: at equilibrium $D(G(z))$ stays near 0.5, not 0.02. **B**: the `detach` belongs to the discriminator's step.

Q11 A Softmax of 100 × cosine

review: **19. CLIP**

✓ **Why.** The logits are 29, 27 and 22. Dividing everything by the largest, $p = 1/(1 + e^{-2} + e^{-7}) \approx 1/1.141 \approx 0.88$. The new class has logit 20 and adds $e^{-9} \approx 0.0001$ to the denominator: the value barely changes. In zero-shot, a new class is just one more prompt and nothing is trained.

× **Pitfalls. B** and **C**: the cosine is not a probability; normalizing the cosines ignores the temperature. **E**: cosines from 0.2 to 0.35 are typical of a correct image–text pair (the modality gap). **D**: the weights of the zero-shot classifier are the embeddings of the prompts.

Q12 C Extrapolating from the unconditional

review: **20. Stable Diffusion**

✓ **Why.** $\tilde{\epsilon} = \epsilon_\theta(z_t, \varnothing) + w\,(\epsilon_\theta(z_t, c) - \epsilon_\theta(z_t, \varnothing))$. Component 1: $0.10 + 7.5 \cdot 0.20 = 1.60$. Component 2: the difference is zero, so $\tilde{\epsilon} = -0.20$. Guidance amplifies only the direction in which the prompt changes the prediction.

× **Pitfalls. D**: starting from the conditional prediction ($0.30 + 1.50$) instead of the unconditional one. **A**: that is $w = 1$. **B** and **E**: multiplying by w the whole prediction, or the term that does not depend on the prompt.

Q13 B The convention changes x_t and the direction of the target

review: **21. Flow Matching**

✓ **Why.** In class, $x_t = (1-t)\,x_0 + t\,x_1 = 0.75\,(1, -2) + 0.25\,(3, 2) = (1.5, -1)$ and the target is $x_1 - x_0 = (2, 4)$. In the colleague's code, $x_t = 0.75\,(3, 2) + 0.25\,(1, -2) = (2.5, 1)$ and the velocity goes from data to noise: $(1, -2) - (3, 2) = (-2, -4)$. Mixing the two conventions (a sampler from one with a model from the other) gives garbage; it is the first thing to check in someone else's code.

× **Pitfalls. E**: switching the convention without reversing the direction of the velocity. **A**: swapping the weights of the interpolant. **C**: confusing the velocity with the data point or the noise, which are other possible targets. **D**: the interpolant is not symmetric; $t = 0.25$ lies near one of the ends.

Q14 **D** **Quality per compute, or a single sequence**

review: **22. Diffusion Transformers**, **23. Autoregressive Generation**

✓ **Why.** For stand-alone text-to-image, the strongest open models are diffusion or flow transformers (SD3, FLUX.1, Qwen-Image): it is the path to the best image per unit of compute. When the image must live in the same context as the text (chatting, reasoning, editing), the natural fit is a single model over a sequence of text and image: autoregressive over tokens (Chameleon, Janus-Pro) or with a diffusion loss inside the language model (Transfusion, BAGEL).

× **Pitfalls. B**: in MMDiT the image rewrites the text tokens, but the model only predicts the image; it does not generate text. **A**: exact likelihood is not image quality. **C**: discrete tokens have the ceiling of the tokenizer; the open models that lead in quality use continuous latents. **E**: MaskGIT and VAR cut sequential passes; neither of them beats "any DiT".

Question 15 · An end-to-end deep learning project

3.0 points · 1.0 per part

Expected answer, one sentence per part.

■ **Model and transfer.** Supervised classification into 5 classes on top of a pretrained backbone (ResNet-50 is the safe choice with 2,400 photos), with a new 5-output head and cross-entropy; first a linear probe, then fine-tuning with a small rate or LoRA.

■ **Training.** AdamW with warmup and cosine, data augmentation that preserves color, validation in `model.eval()` without augmentation and early stopping that restores the best checkpoint.

■ **Diagnosis.** Unfreezing the backbone with $\eta = 10^{-2}$ destroyed the pretrained features; the first step is to go back to the epoch-6 checkpoint and tune the backbone with a rate 100 to 1,000 times smaller (or with LoRA).

(a) Model and transfer — 1.0 point

Supervised learning (the agronomists provided the labels): classification into 5 classes. The features come learned from pretraining, not designed by hand.	0.2
A pretrained backbone, never trained from scratch on 2,400 photos. ResNet-50 is the safe choice: the strong convolutional prior wins with little data and it is cheaper. ViT-B/16 also counts if the justification is pretraining at scale (without it, ViT loses to the CNN with little data).	0.25
Order: first a linear probe (frozen encoder, features computed once, a cheap baseline); then fine-tuning, full or of the last blocks, with a small rate, or LoRA. With 1,920 training photos, either of the two, justified, earns the credit.	0.25
Input $3 \times 224 \times 224$, normalized as in pretraining; output of 5 logits (the old 1,000-class head is replaced by `Linear(2048, 5)` or `Linear(768, 5)`); cross-entropy loss with softmax.	0.3

(b) Training — 1.0 point

Optimizer and schedule: AdamW with weight decay, a small rate for the backbone (around 10^{-5}) and a larger one for the head, a short warmup followed by cosine, with the scheduler stepping at every step.	0.25
Regularization justified by the domain: data augmentation on training only (flipping and rotating the leaf does not change the disease), without touching the hue, because color is what separates rust from cercospora; weight decay.	0.25
Validation with `model.eval()` and `torch.no_grad()`, without augmentation, switching back to `model.train()` for training.	0.25
Early stopping with patience, saving the best validation checkpoint and restoring it at the end.	0.25

(c) Diagnosis **1.0 point**

Reading: everything was going well up to epoch 6 (validation around 83%); at epoch 7 *both* losses jump to around $\ln 5 \approx 1.61$ (that of a guess among 5 classes) and accuracy drops to around 30% without returning to the previous level. It is not overfitting: the training loss got worse too. **0.35**

Cause: $\eta = 10^{-2}$ was fine for a new head, but it is too large for pretrained weights; the first steps throw away the features that made the model work (catastrophic forgetting). **0.3**

First change: go back to the epoch-6 checkpoint and tune the backbone with a rate 100 to 1,000 times smaller (10^{-5} to 10^{-4}, in a parameter group separate from the head), with a short warmup; or use LoRA, or unfreeze only the last blocks. Because the pretrained weights are already close to a good solution: fine-tuning should move them only a little. **0.35**

Syntax details are not penalized. Training from scratch zeroes the backbone item in (a). Applying data augmentation to validation zeroes the validation item in (b). In (c), calling the drop overfitting zeroes the reading item.

A code that meets (b), among other possible ones (the `imports` and the setup of the `DataLoaders` were omitted):

```
model = torchvision.models.resnet50(weights='IMAGENET1K_V2')
model.fc = nn.Linear(2048, 5)                          # new head: 5 classes
train_tf = T.Compose([T.RandomResizedCrop(224, scale=(0.7, 1.0)),
    T.RandomHorizontalFlip(), T.RandomVerticalFlip(), T.RandomRotation(20),
    T.ColorJitter(brightness=0.2, contrast=0.2),       # no hue: color is the cue
    T.ToTensor(), T.Normalize(MEAN, STD)])             # ImageNet MEAN and STD
val_tf = T.Compose([T.Resize(256), T.CenterCrop(224),  # validation without augmentation
    T.ToTensor(), T.Normalize(MEAN, STD)])

backbone = [p for n, p in model.named_parameters() if not n.startswith('fc.')]
opt = torch.optim.AdamW([{'params': backbone, 'lr': 3e-5},
                         {'params': model.fc.parameters(), 'lr': 1e-3}], weight_decay=0.05)
total, warmup = EPOCHS * len(train_dl), len(train_dl)
def factor(s):                                         # linear warmup, then cosine
    if s < warmup: return (s + 1) / warmup
    return 0.5 * (1 + math.cos(math.pi * (s - warmup) / (total - warmup)))
sched = torch.optim.lr_scheduler.LambdaLR(opt, factor)
loss_fn = nn.CrossEntropyLoss()

best, patience, no_improve = float('inf'), 5, 0
for epoch in range(EPOCHS):
    model.train()
    for x, y in train_dl:
        loss = loss_fn(model(x), y)
        opt.zero_grad(); loss.backward(); opt.step(); sched.step()   # sched per step
    model.eval()
    with torch.no_grad():
        val = sum(loss_fn(model(x), y).item() * len(y) for x, y in val_dl) / len(val_dl.dataset)
    if val < best:
        best, no_improve = val, 0
        torch.save(model.state_dict(), 'best.pt')      # save the best checkpoint
    else:
        no_improve += 1
        if no_improve == patience:
            break
model.load_state_dict(torch.load('best.pt'))           # restore the best
```

Common mistakes. Training the network from scratch on 2,400 photos. Data augmentation with strong `hue`, which erases the color cue. Applying augmentation to validation as well. Forgetting `model.eval()`. Stopping training without restoring the best checkpoint. Using the next harvest's test set to choose the model. In (c), calling the drop overfitting or "bad data" without noticing that it starts exactly at the epoch in which the backbone was unfrozen.

Review: **1. Concepts**, **6. Optimization**, **7. Regularization**, **10. Convolutional Networks**, **13. Vision Transformers** and **14. Transfer Learning**.