DDA4220 · Fall 2026

Lecture 13

Deep Learning and Applications

Midterm review

Zhen Liu

CUHK-Shenzhen

Review scope

- L01–L12, through Transformers.
- Core concepts and their application to changed settings.
- Optional material and L14 onward are excluded.

The original core lecture slides remain the full study reference.

Exam format

Multiple-choice questions

Select all correct statements.
The number of correct options is not given.

Short-answer questions

Give the result and explain why.
Use a short calculation or labeled diagram when needed.

Core concepts, including changed settings. No cheat sheet.

Review outline

01 Networks, tensors and objectives

02 Learning and generalization

03 Convolutional networks

04 Visual tasks and transfer

05 Recurrent sequence models

06 Attention and Transformers

Capacity, optimization and generalization

- Capacity: does the model family contain a useful function?
- Optimization: can training find suitable parameters?
- Generalization: do the predictions work on unseen data?

The model family is also called the hypothesis class.

Tensors, axes and shapes

A tensor is an array of values. Its axes are numbered from 0.

Feature batch

```
x.shape          # (3, 5)
x.shape[0]       # 3
x.shape[-1]      # 5
B, F = x.shape   # B = 3, F = 5
```

Axis 0: examples
Axis 1: features
−1 selects the last axis.

Image batch: four axes

```
images.shape  # (8, 3, 32, 32)
```

Examples, channels, height, width
8 RGB images, each 32 × 32 pixels

Elementwise and matrix multiplication

Two-dimensional tensors: X has shape (3, 5), W has shape (5, 2).

Elementwise operations

```
X * X  # (3, 5)
```

$$Y_{ij} = X_{ij} X_{ij}$$

Combine corresponding entries.
+, − and / are also elementwise.

Matrix multiplication

```
X @ W  # (3, 2)
```

$$Y_{ij} = \sum_{k=1}^{5} X_{ik} W_{kj}$$

Match the inner dimensions.
Take a row–column dot product.

For matrices, torch.matmul(X, W) is equivalent to X @ W.

Broadcasting

Align shapes from the right. Each pair of sizes must match, or one must be 1.
Treat a missing leading axis as size 1.

One bias per feature

```
X      # (3, 5)
b      #    (5,)
X + b  # (3, 5)
```

The same five biases are reused for all three examples.

One scale per example

```
X      # (3, 5)
s      # (3, 1)
X * s  # (3, 5)
```

Each example's scale is reused across its five features.

Broadcasting: shape matching

Select all correct statements

X: (3, 5), with examples in rows.
b: (5,) s: (3, 1) u: (3,)

A. X + b adds the same feature-bias vector to every example.

B. X * s scales each example by one value.

C. X + u adds one value to each row.

D. s + b has shape (3, 5).

Broadcasting: answer

Correct: A, B, D

A b behaves as shape (1, 5): reuse its feature biases across rows.

B s has shape (3, 1): reuse each row's scale across columns.

C u aligns with the last axis: 3 does not match 5.
For one value per row, use u.reshape(3, 1).

D (3, 1) + (1, 5) gives (3, 5). Entry (i, j) uses s[i, 0] + b[j].

Sum and the reduction axis

dim selects the axis whose entries are combined.
Here, X has 2 example rows and 3 feature columns.

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

Scalar: one value

```
X.sum()  # 21
```

dim=1: combine features

```
X.sum(dim=1)
```

`[6, 15]`
shape (2,)

dim=0: combine examples

```
X.sum(dim=0)
```

`[5, 7, 9]`
shape (3,)

```
X.sum(dim=1, keepdim=True)  # [[6], [15]], shape (2, 1)
```

keepdim retains a size-1 axis, so row sums can broadcast across features.

keepdim, squeeze and unsqueeze

keepdim=True retains the reduced axis with size 1.
Using the same X of shape (2, 3):

```
r = X.sum(dim=1)                    # (2,)
c = X.sum(dim=1, keepdim=True)      # (2, 1)
```

```
r.unsqueeze(1)  # (2, 1): insert a size-1 axis
c.squeeze(1)    # (2,): remove that size-1 axis
```

The values stay the same. A (2, 1) column can broadcast across X.

squeeze() removes all size-1 axes, including a batch axis of size 1.

Concatenation with torch.cat

Join tensors along an existing axis. All other axis sizes must match.

Join features for the same examples

```
# A: (2, 3), B: (2, 2)
torch.cat([A, B], dim=1)  # (2, 5)
```

Combine examples with the same feature count

```
# A: (2, 3), C: (1, 3)
torch.cat([A, C], dim=0)  # (3, 3)
```

Reshape and flatten

x has shape (2, 3, 4): two examples, each a 3 × 4 array.

```
x.reshape(2, 12)          # (2, 12)
x.reshape(2, -1)          # (2, 12)
x.flatten(start_dim=1)    # (2, 12)
```

Keep all 24 entries. Group each example into one row.
One −1 may infer the missing size: 24 ÷ 2 = 12.

view can also reshape when the memory layout permits it.
reshape can copy the data if needed.

Image resizing resamples pixels. Reshaping only regroups existing entries.

Transpose and permute

transpose swaps two axes. permute specifies the complete axis order.

$$x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

```
x.transpose(0, 1)
```

$$\begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

```
x.reshape(3, 2)
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Same output shape, different arrangement of values.

```
batch.permute(0, 2, 1)  # (2, 3, 4) -> (2, 4, 3)
```

batch.permute(0, −1, 1) is equivalent here. −1 names the last axis.

Short answer: tensor transformations

x has shape (2, 3, 4), with distinct entries.
Axes: examples, rows, columns.

```
a = x.flatten(start_dim=1)
b = x.permute(0, 2, 1)
c = x.reshape(2, 4, 3)
```

- What are the shapes of a and b?
- Do b and c have the same arrangement of values? Explain.
- What size does −1 infer in x.reshape(2, -1)?

Tensor transformations: answer

```
a.shape  # (2, 12)
b.shape  # (2, 4, 3)
c.shape  # (2, 4, 3)
```

a keeps each example's 12 entries together.
b exchanges the row and column axes within each example.

b and c have the same shape, but different arrangements:

```
b[0, 0, 1] == x[0, 1, 0]
c[0, 0, 1] == x[0, 0, 1]
```

In x.reshape(2, -1), −1 infers 12. The number of entries stays 24.

Short answer: reading an MLP

Examples are rows. X has shape B × 6.
The hidden width is 4. There are 3 classes.

Network

$$H = \mathrm{ReLU}(XW_1^\mathsf{T} + b_1), \qquad S = HW_2^\mathsf{T} + b_2$$

- Find the shapes of W_1, b_1, W_2, b_2, H and S.
- How many trainable parameters are there? Does B change the count?

Reading the MLP: answer

Parameter shapes

$$W_1 : 4 \times 6, \quad b_1 : 4, \qquad W_2 : 3 \times 4, \quad b_2 : 3$$

Activation shapes

$$H : B \times 4, \qquad S : B \times 3$$

Trainable entries

$$4 \times 6 + 4 + 3 \times 4 + 3 = 43$$

B changes the number of examples, not the shared parameters.

MLP in PyTorch

Same MLP: 6 input features, 4 hidden units, 3 classes.
x has shape (B, 6), where B is the batch size.

```
model = nn.Sequential(
    nn.Linear(6, 4),  # 6 inputs to 4 hidden features
    nn.ReLU(),        # elementwise nonlinearity
    nn.Linear(4, 3),  # 3 class scores (logits)
)
logits = model(x)     # forward pass: (B, 6) to (B, 3)
```

nn.Sequential applies layers in order. Each nn.Linear learns weights and a bias.

Composition of linear layers

Why do hidden layers need nonlinear activations?

$$W_2(W_1x + b_1) + b_2$$

$$= (W_2W_1)x + (W_2b_1 + b_2)$$

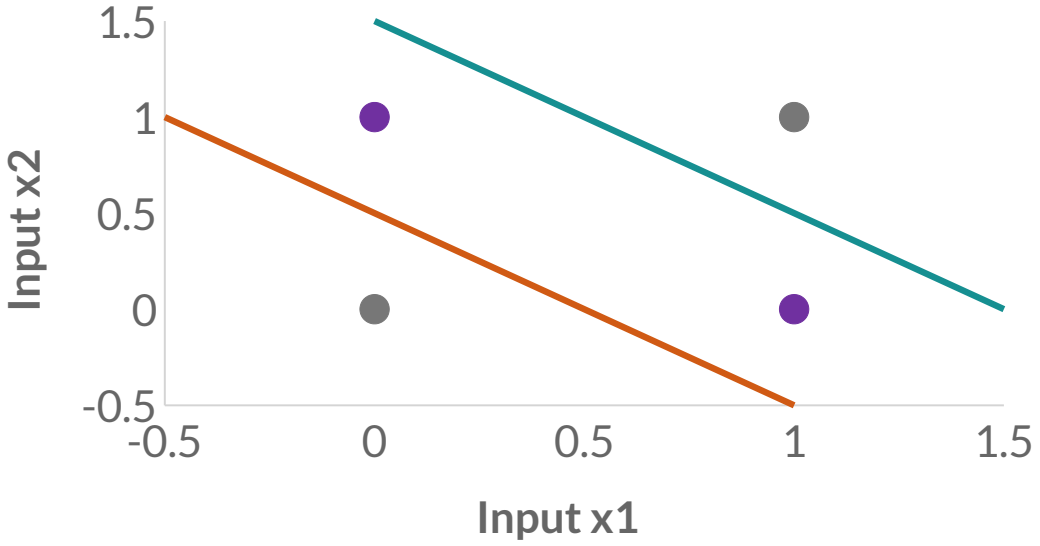
Stacking linear layers without activations gives one linear layer.

XOR in hidden coordinates

Thresholds: 1 if true, 0 if false

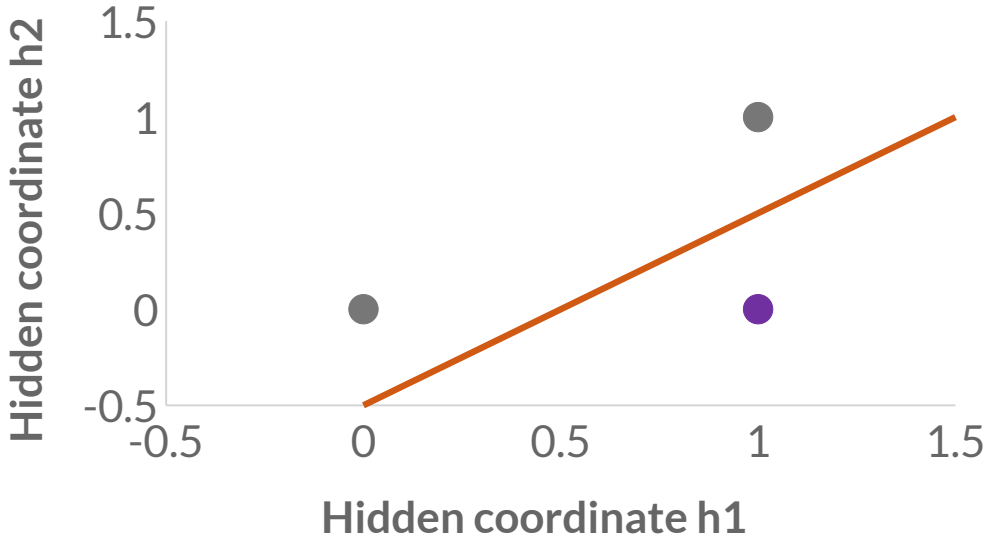
$$h_1 = \mathbf{1}\{x_1 + x_2 \geq \tfrac{1}{2}\}$$

$$h_2 = \mathbf{1}\{x_1 + x_2 \geq \tfrac{3}{2}\}$$



Linear output

$$\widehat{y} = h_1 - h_2$$



Classification scores and loss

s_k is the logit for class k. The observed class is y.

Probabilities

$$p_k = \frac{e^{s_k}}{\sum_j e^{s_j}}$$

Cross entropy loss

$$\ell_{\mathrm{CE}} = -\log p_y$$

- argmax selects the highest-scoring class. 0/1 loss is 0 if correct, 1 otherwise.

- Cross entropy is a smooth training surrogate for that hard decision.

Softmax and the class axis

Logits have shape (B, C): rows are examples, columns are classes.

```
B, C = logits.shape          # examples, classes
p = logits.softmax(dim=-1)   # shape (B, C)
p.sum(dim=1)                 # shape (B,), each entry is 1
```

dim=1 or −1: classes

Normalize each example's class scores.
Each row sums to 1.

dim=0: examples

Normalize across examples.
Each column sums to 1.

Both preserve shape (B, C). Only the first gives class probabilities per example.

Maximum, minimum and their indices

With dim, max/min return values and indices. Indices start at 0.

$$\text{scores} = \begin{bmatrix} 1 & 4 & 2 \\ 3 & 0 & 5 \end{bmatrix}$$

Rows: examples. Columns: classes.
dim=1 searches each row.
Each result has shape (2,).

```
v, i = scores.max(dim=1)  # v=[4, 5], i=[1, 2]
v, i = scores.min(dim=1)  # v=[1, 0], i=[0, 1]
scores.argmax(dim=1)      # [1, 2]: indices only
scores.argmin(dim=1)      # [0, 1]: indices only
```

Without dim, scores.max() and scores.min() each return one value.

Predictions and cross-entropy

Select all correct statements

Three classes. The correct label is class 2. The entries below are probabilities.

$$p_A = (0.10,\ 0.80,\ 0.10), \qquad p_B = (0.20,\ 0.60,\ 0.20)$$

A. Both models have zero classification error on this example.

B. Model B has the smaller cross-entropy loss.

C. Adding the same constant to every logit leaves its probabilities unchanged.

D. Lower cross-entropy must improve accuracy on this example.

Predictions and cross-entropy: answer

Correct: A, C

A	**Correct**	Both models assign their largest probability to the correct class.
B	**Incorrect**	Model A assigns more probability to class 2, so its negative log probability is smaller.
C	**Correct**	The common exponential factor cancels between the numerator and denominator of softmax.
D	**Incorrect**	Accuracy is already correct for both. The probability of the correct class can still increase.

Cross entropy and KL divergence

q: fixed target probabilities. p: model probabilities.

Entropy: target uncertainty

$$H(q) = -\sum_k q_k \log q_k$$

Cross entropy

$$H(q, p) = -\sum_k q_k \log p_k$$

$$H(q, p) = H(q) + D_{\mathrm{KL}}(q \| p)$$

H(q) is fixed. Minimizing cross entropy also minimizes forward KL.

For a one-hot target, H(q) = 0.

Forward and reverse KL

The target distribution q is fixed. The model distribution p can change.

Forward KL

$$\mathbb{E}_{x\sim q}\left[\log \frac{q(x)}{p_\theta(x)}\right]$$

Averages over target outcomes. Strongly penalizes missing their probability mass.

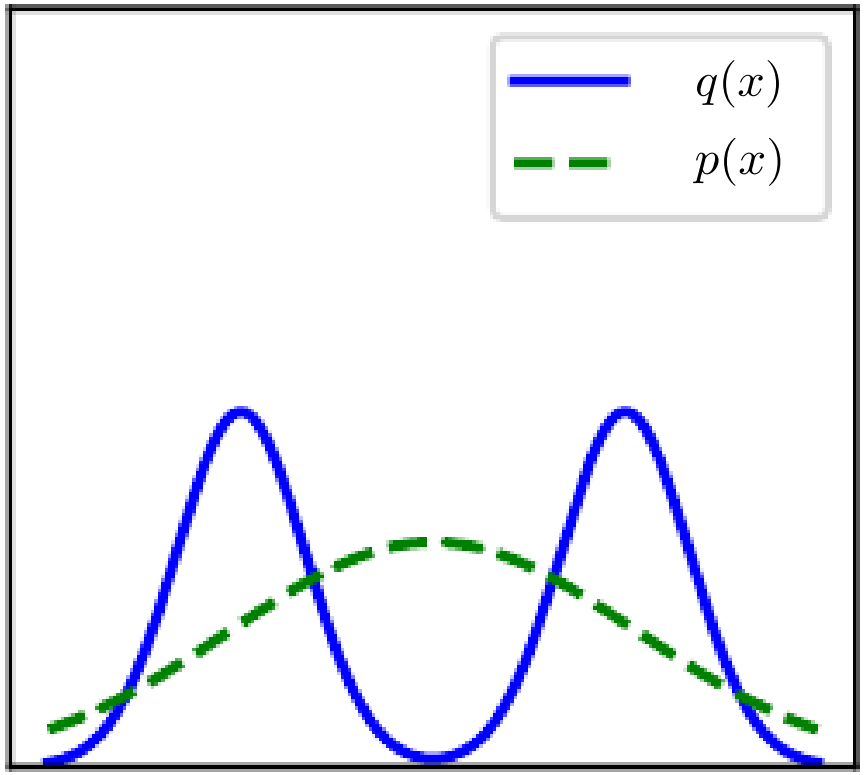
Reverse KL

$$\mathbb{E}_{x\sim p_\theta}\left[\log \frac{p_\theta(x)}{q(x)}\right]$$

Averages over model outcomes. Strongly penalizes placing mass where the target is tiny.

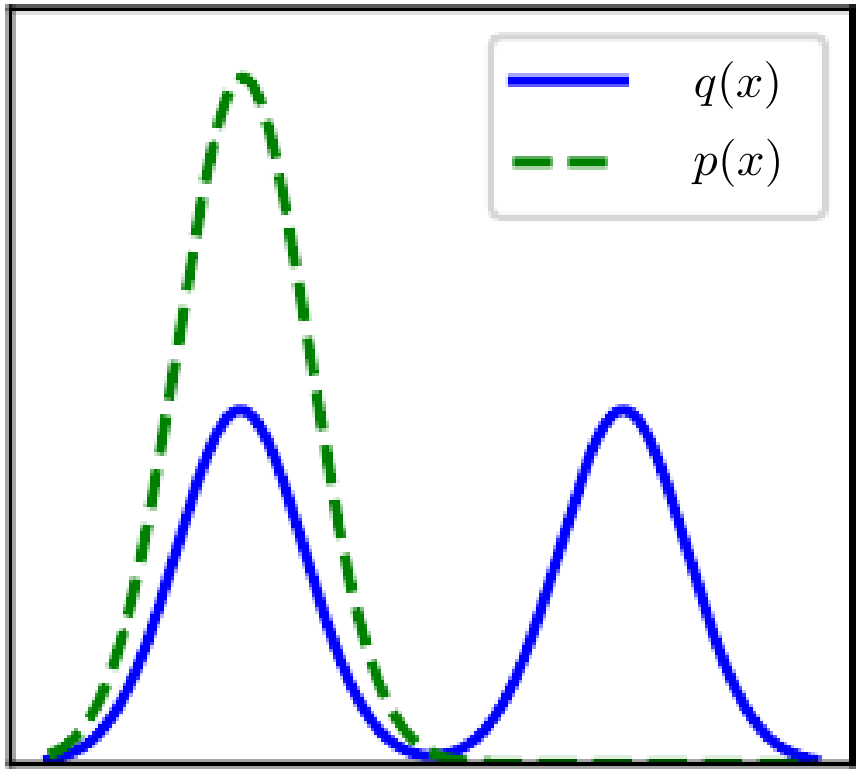
Two peaks, one-peak model

Forward KL: mode covering



Missing either target peak is costly.
The model broadens to cover both.

Reverse KL: mode seeking



Probability in the gap is costly.
The model concentrates on one peak.

Empirical loss

N training examples, each with an input x_i and target y_i.

θ collects the network's trainable weights and biases.

Average training loss

$$L_{\text{train}}(\theta) = \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(x_i), y_i)$$

$f_\theta(x_i)$: prediction ℓ : loss on one example

Empirical risk: average loss measured on the training data.

Review outline

01 Networks, tensors and objectives

02 Learning and generalization

03 Convolutional networks

04 Visual tasks and transfer

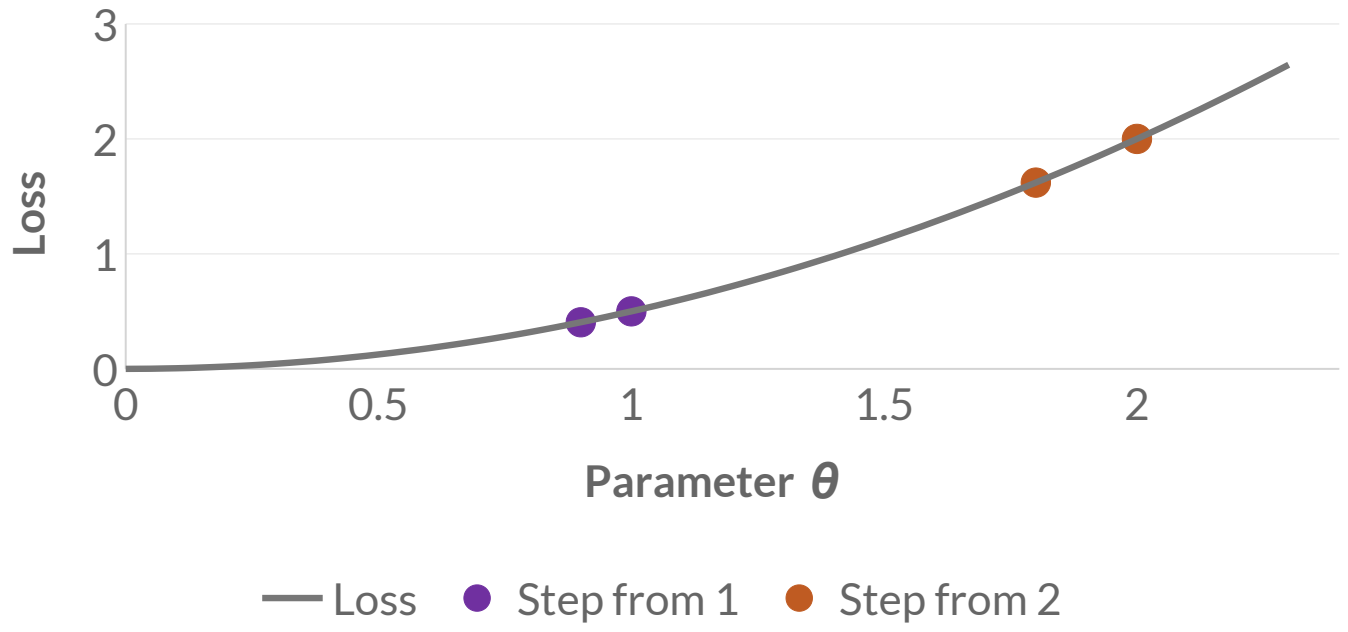
05 Recurrent sequence models

06 Attention and Transformers

Gradient descent

$$\theta_{t+1} = \theta_t - \eta g_t, \qquad \Delta\theta = -\eta g_t$$

g: loss gradient η : learning rate



$$\mathcal{L}(\theta) = \tfrac{1}{2}\theta^2, \quad \eta = 0.1$$

$$\theta = 1: \quad g = 1, \quad \Delta\theta = -0.1$$

$$\theta = 2: \quad g = 2, \quad \Delta\theta = -0.2$$

Same learning rate, larger slope, larger parameter step.

Minibatch SGD

Stochastic Gradient Descent (SGD)

Initialize the network parameters

For each update

1	$B_t \leftarrow \text{a sampled minibatch}$	**Input batch**
2	$g_t \leftarrow \frac{1}{\lvert B_t \rvert} \sum_{i \in B_t} \nabla_\theta \ell_i(\theta_t)$	Average gradient
3	$d_t \leftarrow g_t$	Update vector
4	$\eta_t \leftarrow \eta$	Learning rate
5	$\theta_{t+1} \leftarrow \theta_t - \eta_t d_t$	

End

Training the MLP

Each batch: x has shape (B, 6). y contains B integer labels: 0, 1 or 2.
lr is the learning rate. Cross-entropy takes the logits directly.

```
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
model.train()                       # select training behavior
for x, y in batches:                # one minibatch at a time
    optimizer.zero_grad()           # clear stored gradients
    logits = model(x)               # forward pass
    loss = loss_fn(logits, y)       # mean classification loss
    loss.backward()                 # add gradients to .grad
    optimizer.step()                # update parameters using .grad
```

backward() adds to each parameter's .grad. step() does not clear those gradients.

Averaging gradients and taking steps

Select all correct statements

Two equal minibatches use mean losses. Each example has a deterministic loss, independent of other examples. Use plain SGD.

A. Averaging their gradients at the same weights gives the combined-batch gradient.

B. Two successive minibatch updates always equal one combined-batch update.

C. Doubling the gradient and halving the learning rate gives the same SGD step.

D. Computing both gradients changes the weights even before an update.

Averaging gradients and taking steps: answer

Correct: A, C

A	**Correct**	Equal minibatches give equal weight to every example when their two mean gradients are averaged.
B	**Incorrect**	The second update evaluates its gradient at changed weights. The resulting updates need not match.
C	**Correct**	The SGD displacement is learning rate times gradient, so these two factors cancel.
D	**Incorrect**	Gradient computation and parameter updates are separate operations.

Short answer: one backward pass

Fixed input x = 3 and target y = 1. Parameters w = 2, b = −1.

Forward and loss

$$z = wx + b, \qquad h = \max(0, z), \qquad \ell = \frac{1}{2}(h - y)^2$$

- Find $\partial \ell / \partial w$ and $\partial \ell / \partial b$.
- What happens to this path's gradients when z < 0?

One backward pass: answer

Forward values

$$z = 5, \qquad h = 5, \qquad h - y = 4$$

Parameter gradients

$$\frac{\partial \ell}{\partial w} = 4 \times 1 \times 3 = 12, \qquad \frac{\partial \ell}{\partial b} = 4$$

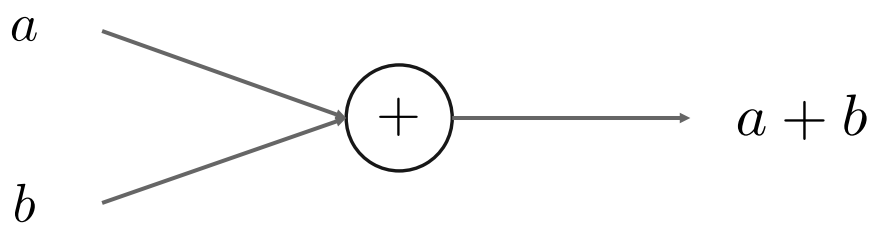
- If z < 0, the ReLU derivative is zero, so this path contributes zero gradient.

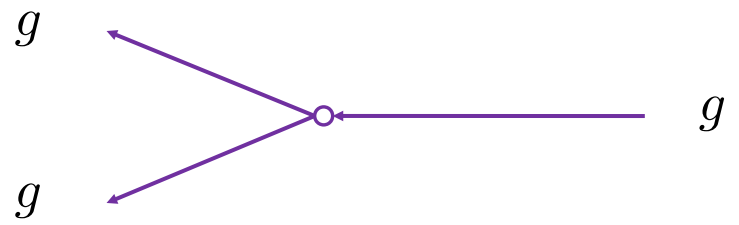
Forward and backward graph operations

Forward

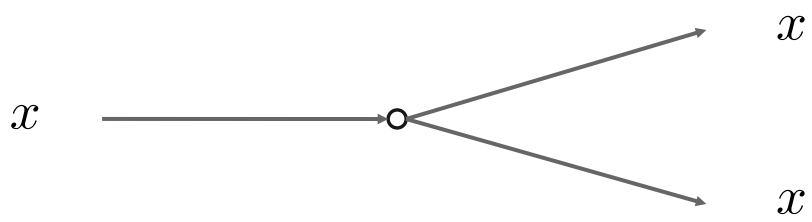
Backward

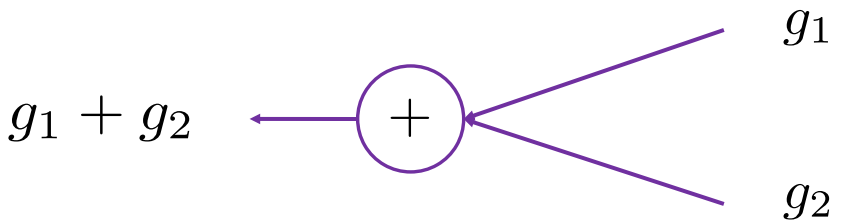
Sum





Copy





Adapted from CS 7643

Cross-entropy gradient

s: logits p: model probabilities q: target probabilities

$$\ell = \log \sum_j e^{s_j} - \sum_k q_k s_k$$

$$\frac{\partial \ell}{\partial s_k} = \frac{e^{s_k}}{\sum_j e^{s_j}} - q_k = p_k - q_k$$

One-hot target: hold the other logits fixed

$$k = y: \;\; p_y - 1 \leq 0$$

Raising it reduces loss.

$$k \neq y: \;\; p_k \geq 0$$

Lowering each reduces loss.

Learning rate on a quadratic loss

One parameter w. The minimum is at w = 0.
a > 0 controls the curvature: larger a means a steeper bowl.

Loss and gradient

$$L(w) = \frac{a}{2}w^2, \qquad g = aw$$

One gradient step

$$w_{\text{new}} = w - \eta a w = (1 - \eta a)w$$

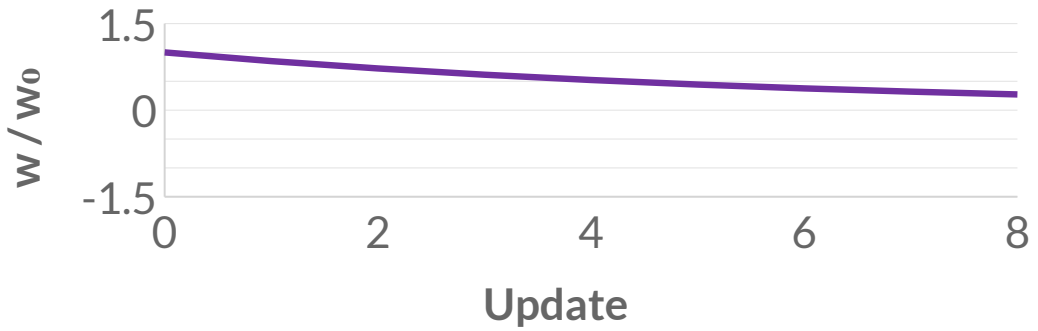
- The factor 1 – η a determines how the next step changes w.

Stable and unstable step sizes

w_0: starting parameter value

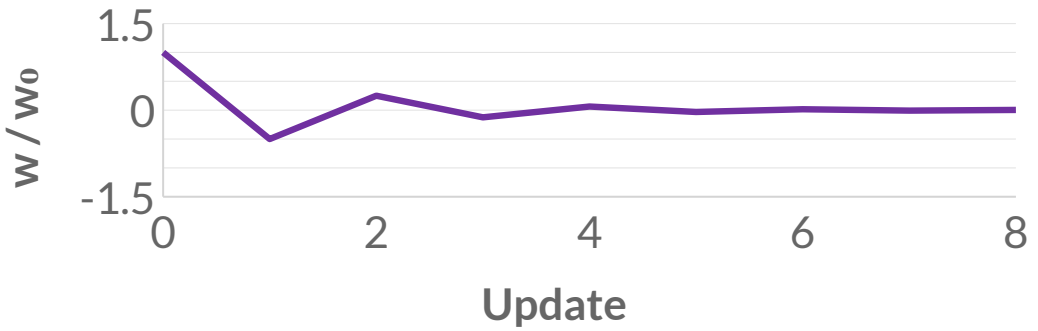
No oscillation

$0 < \eta a < 1$



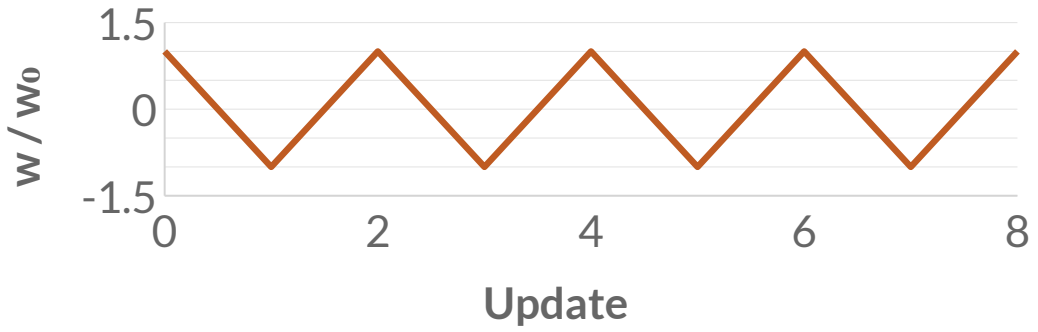
Shrinking oscillation

$1 < \eta a < 2$



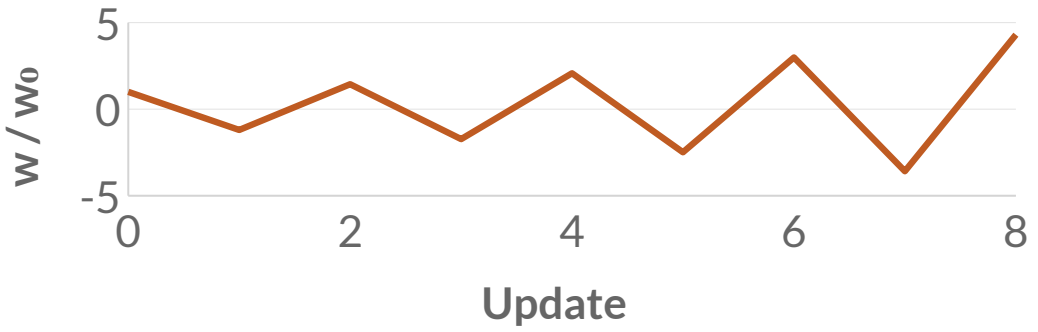
Mirror oscillation

$\eta a = 2$



Divergence

$\eta a > 2$



Can the parameter oscillate while the loss keeps decreasing?

Momentum and gradient history

g_t: current minibatch gradient. m_t: its moving average.
$0 \leq \beta < 1$ controls how long past gradients remain influential.

Why average gradients across updates?

$$m_t = \beta m_{t-1} + (1 - \beta) g_t$$

Parameter update

$$\theta_t = \theta_{t-1} - \eta m_t$$

- Persistent directions reinforce. Alternating or noisy directions partly cancel.

From AdaGrad to RMSProp

Each parameter has its own squared-gradient history.
The update divides g_t by $\sqrt{v_t} + \varepsilon$, then multiplies by the global learning rate.

AdaGrad

$$v_t = v_{t-1} + g_t^2$$

- Keeps every past squared gradient.
- The denominator cannot decrease.

RMSProp

$$v_t = \beta v_{t-1} + (1 - \beta) g_t^2$$

- Discounts older magnitudes.
- Adapts to changing gradient scales.

After a large gradient, which method can let its denominator shrink again?

Adam: direction and scale

g_t: current gradient η: learning rate β_1, β_2: decay factors

Start with $m_0 = v_0 = 0$. All operations below are elementwise.

Mean gradient

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$

Mean squared gradient

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

Bias correction

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t}, \qquad \widehat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Parameter update

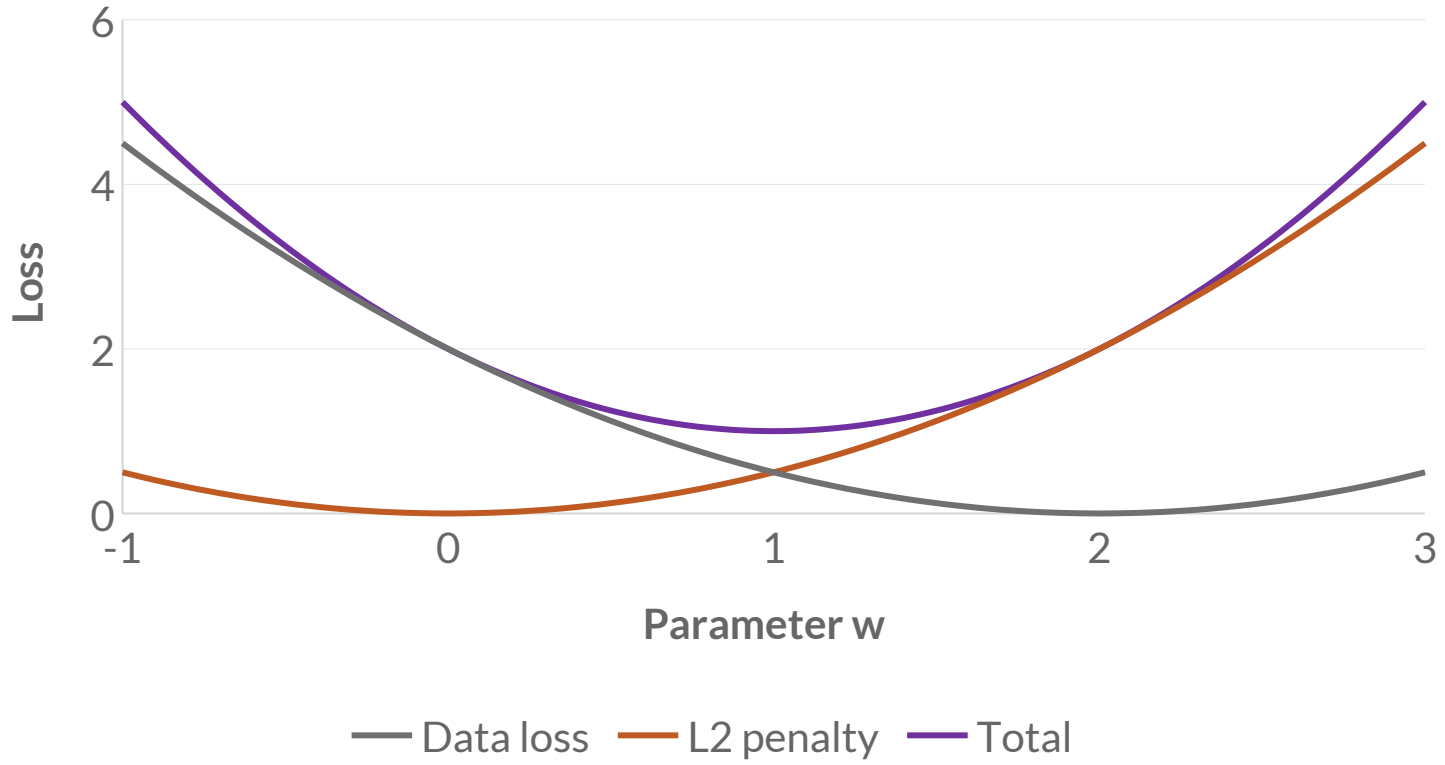
$$\theta_t = \theta_{t-1} - \eta \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \varepsilon}$$

ε is a small constant that keeps the denominator nonzero.

L2 regularization

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{data}} + \frac{\lambda}{2}\|\theta\|_2^2$$

Squared L2 norm: sum of squared parameters



- Penalize large parameter magnitudes.
- Larger λ penalizes large weights more.

Adam + L2 versus AdamW

g: data-loss gradient η : learning rate λ : penalty or decay coefficient

Adam + L2

$$g_{\text{reg}} = g + \lambda\theta$$

The penalty enters both moments and adaptive scaling.

AdamW

$$\theta_{\text{new}} = (1 - \eta\lambda)\theta - \eta\frac{\widehat{m}}{\sqrt{\widehat{v}} + \epsilon}$$

Both moment estimates use g only. Weight decay is applied separately.

- Which gradient enters the moving averages in each method?
- Why are L2 regularization and weight decay different for Adam?

Symmetry survives gradient descent

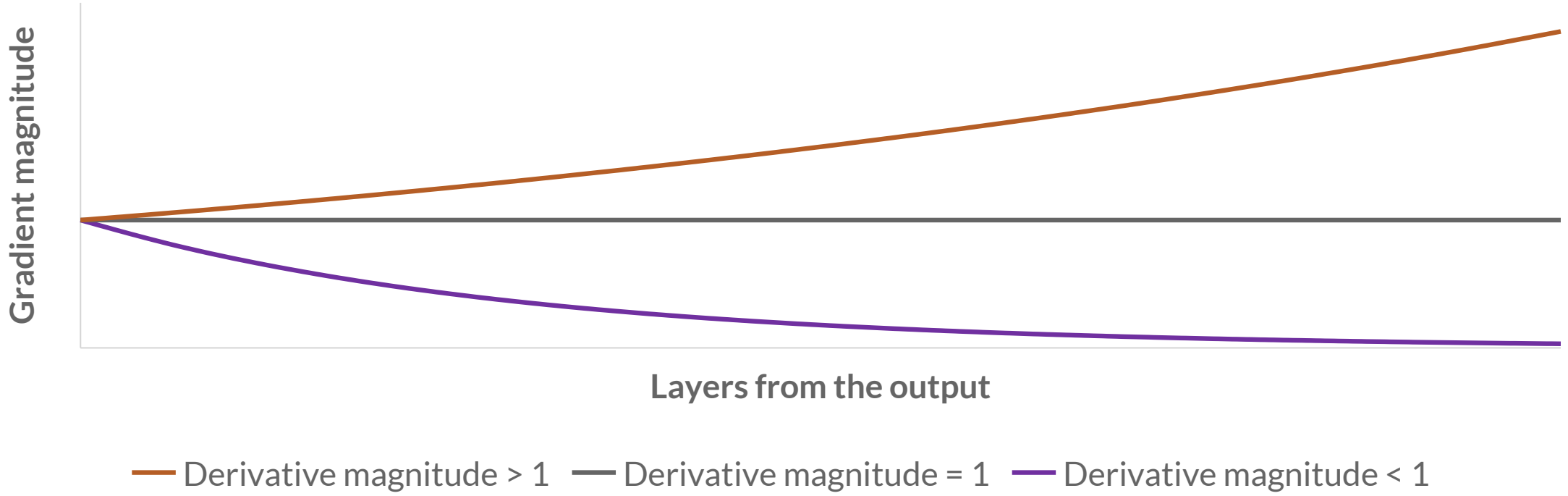
Symmetric incoming and outgoing connections

The units learn the same feature, reducing effective width.

Why initialize hidden units differently?

Vanishing and exploding gradients

In a scalar chain, each layer multiplies the gradient by its derivative.



Repeated multiplication can shrink or enlarge the gradient rapidly.

Choosing the initial scale

Fan-in: inputs to one unit. Fan-out: outputs reached by one unit.

Zero-mean weights: the formulas give variance, not standard deviation.

Xavier / Glorot

$$\mathrm{Var}(w) = \frac{2}{n_{\mathrm{in}} + n_{\mathrm{out}}}$$

Balances forward and backward scales.

He / Kaiming

$$\mathrm{Var}(w) = \frac{2}{n_{\mathrm{in}}}$$

For ReLU: compensates for the average loss of half the squared magnitude.

Normalization and learned scale

Why learn a scale and shift after normalization?

Normalize

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}}$$

μ, σ^2: mean and variance of the group
ε : small constant

Learn scale and shift

$$y = \gamma\hat{x} + \beta$$

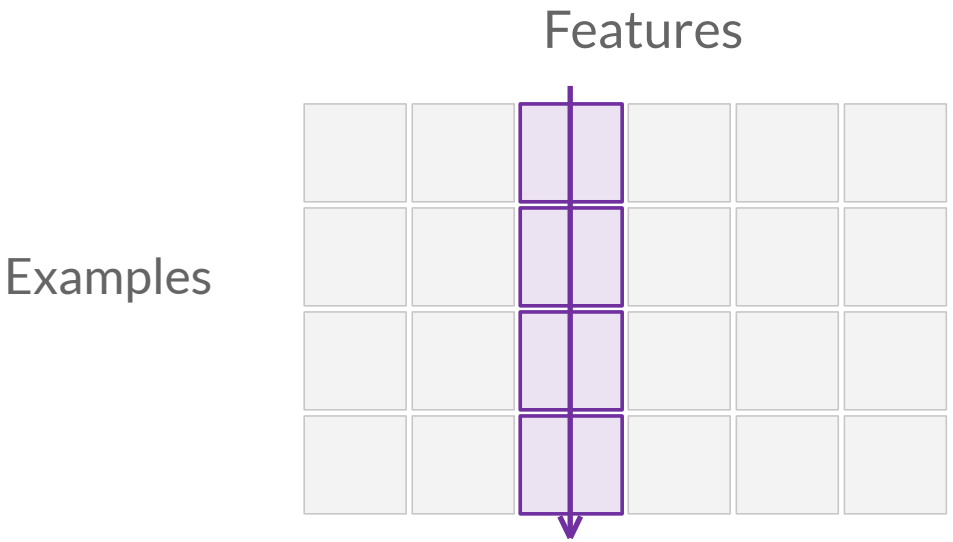
γ : learned scale
β : learned shift

The layer can learn the activation range needed by the next operation.

BatchNorm and LayerNorm

Example: a batch of feature vectors

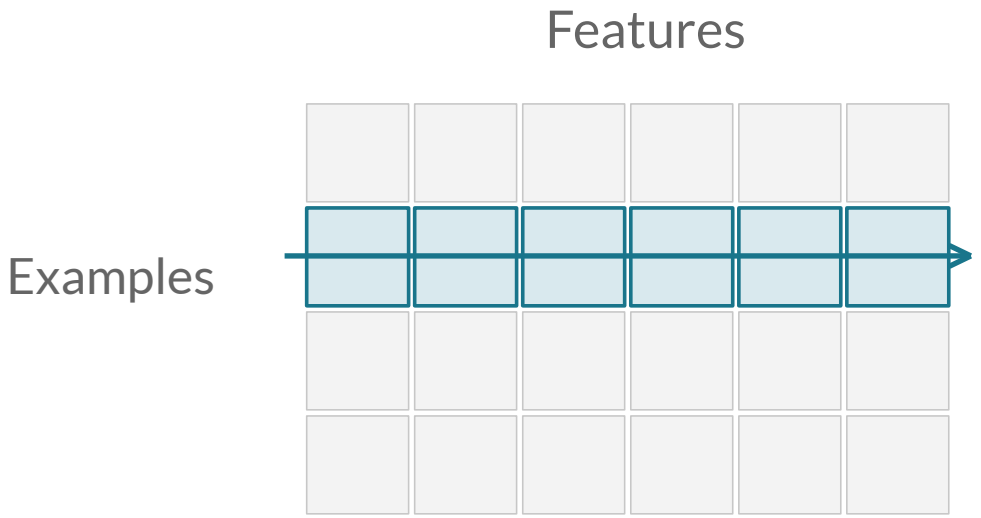
BatchNorm



One feature across examples

Training: current batch statistics
Evaluation: running estimates

LayerNorm



Features within one example

Current vector statistics
in both modes

Dropout in training and evaluation

Randomly mask each activation h to reduce reliance on particular units.

Common convention: scale during training (inverted dropout).

Training

$$\tilde{h} = \frac{mh}{q}$$

q: probability of keeping an activation
m = 1 with probability q, otherwise 0

Evaluation

$$\tilde{h} = h$$

Use every activation.
No mask or dropout scaling.

Dividing by q preserves the average activation during training.

Gradient tracking: no_grad and detach

Model parameters have requires_grad=True. In normal gradient mode, autograd records their operations so backward can compute gradients.

torch.no_grad()

```
with torch.no_grad():
    y = model(x)
```

Do not record operations inside this block.

Tensor.detach()

```
h = model(x)
fixed = h.detach()
```

Keep h's values without its history.
h itself keeps its graph.
This shares data, rather than copying it.

Neither changes train/eval mode or clears existing gradients.

Copying tensors and detaching gradients

Copying data and stopping gradients are separate choices.

Assume x.requires_grad=True and normal gradient tracking.

```
a = x                    # same tensor
b = x.clone()            # copied data, keeps gradient path
c = x.detach()           # shared data, stops gradient path
d = x.detach().clone()   # copied data, stops gradient path
```

A loss using b can still backpropagate to x.
c shares x's data even though the gradient connection is removed.

PyTorch training and evaluation

model.train()

Dropout samples masks.
BatchNorm uses batch statistics and updates running statistics.

model.eval()

Dropout is inactive.
BatchNorm uses stored statistics.

These describe standard Dropout and BatchNorm with running statistics.

```
model.eval()              # choose evaluation behavior
with torch.no_grad():  # disable gradient recording
    prediction = model(x)
```

eval() alone does not disable gradients. train() alone does not update weights.

Training and evaluation modes

Select all correct statements

A trained model has standard dropout and BatchNorm with running statistics. It stays in training mode, inside no_grad().

A. no_grad() disables dropout.

B. Changing other examples in the batch can change this example's prediction.

C. BatchNorm's running statistics can still change.

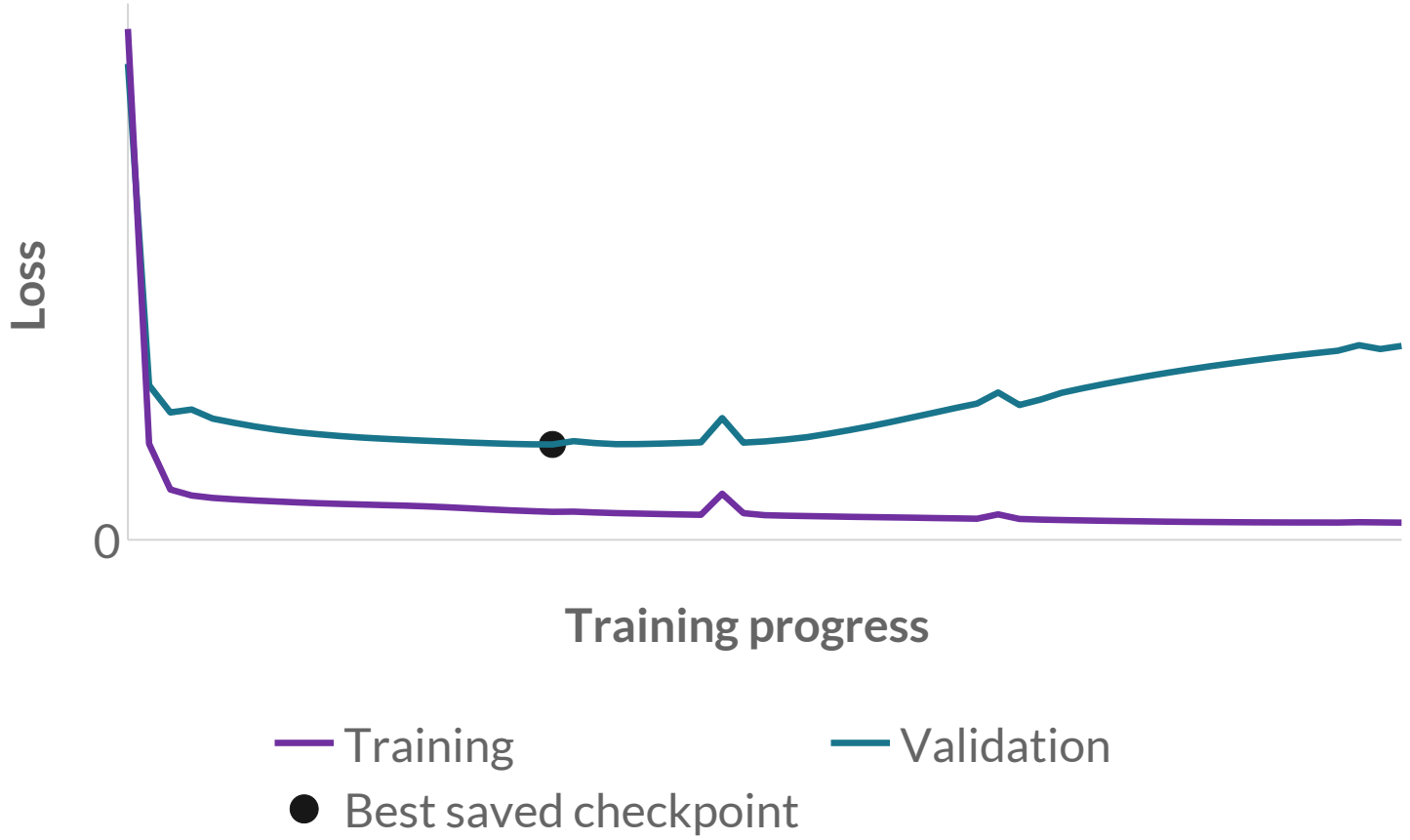
D. eval() changes these layers' behavior but does not itself disable gradients.

Training and evaluation modes: answer

Correct: B, C, D

A	**Incorrect**	no_grad() disables gradient recording. Dropout still follows the module's training mode.
B	**Correct**	Training-mode BatchNorm uses current batch statistics, which depend on the other examples.
C	**Correct**	Updating running statistics is part of the BatchNorm forward pass, even without backward.
D	**Correct**	eval() selects stored BatchNorm statistics and disables dropout masks. Gradient recording is a separate control.

Checkpoint selection and early stopping



Checkpoint selection

Keep the best saved weights by validation score.

Early stopping

Stop when validation has not improved for several checks.

Test set

Reserve test data for final evaluation after model selection.

Generalization and model selection

Select all correct statements

The training, validation and test sets contain independent examples from the intended task. Choose one final model.

A. Repeatedly choosing settings by test accuracy preserves an independent final test.

B. The lowest training loss identifies the model that will generalize best.

C. Early stopping can select a checkpoint using validation performance.

D. Stronger regularization can worsen training fit while improving validation performance.

Generalization and model selection: answer

Correct: C, D

A Incorrect	Those choices use information from the test set. It no longer gives an independent evaluation of the selection process.
B Incorrect	Training loss measures fit to observed training examples, not performance on new ones.
C Correct	Validation guides model selection. Keep the test set for the final evaluation.
D Correct	Restricting the fitted solution can reduce overfitting. Improvement is possible, not guaranteed.

Review outline

01 Networks, tensors and objectives

02 Learning and generalization

03 Convolutional networks

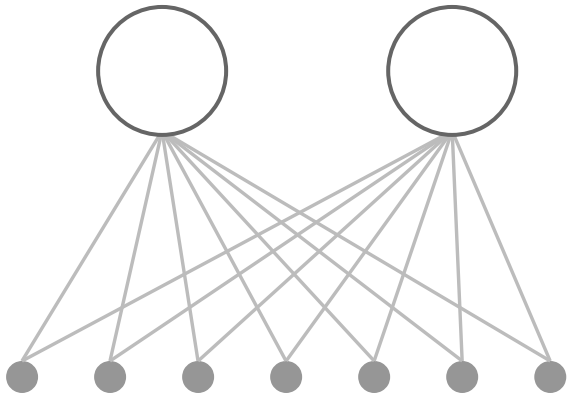
04 Visual tasks and transfer

05 Recurrent sequence models

06 Attention and Transformers

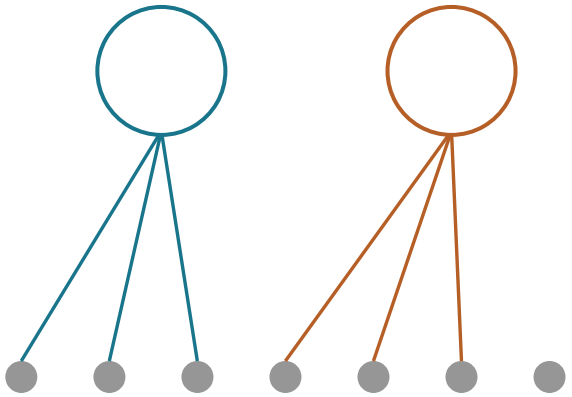
Dense, local and shared connections

Dense



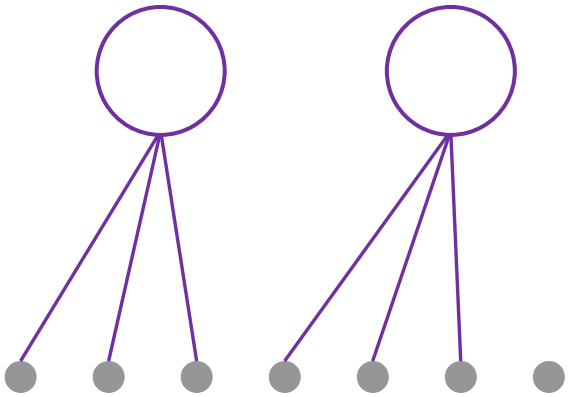
All input positions

Local



Nearby inputs

Local + shared



Same local detector

Why reuse the same filter at different positions?

A local pattern can be useful wherever it occurs.

Convolution: the weighted sum

One 2D image and one 2D kernel, stride 1

$$Z_{i,j} = \sum_{u,v} K_{u,v}\, X_{i+u,j+v} + b$$

$Z_{i,j}$

Output at (i, j)

$K_{u,v}\, X_{i+u,j+v}$

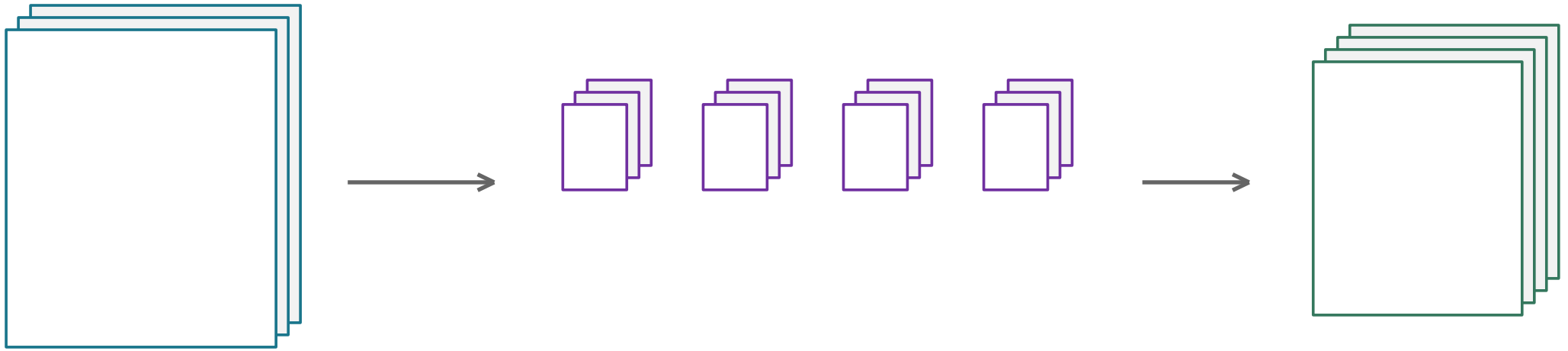
Kernel weight × input value

Sum over all positions (u, v) in the kernel.

X: input image K: kernel b: bias

Convolutional layer

Input channels | **Filter bank** | **Output channels**



3 channels

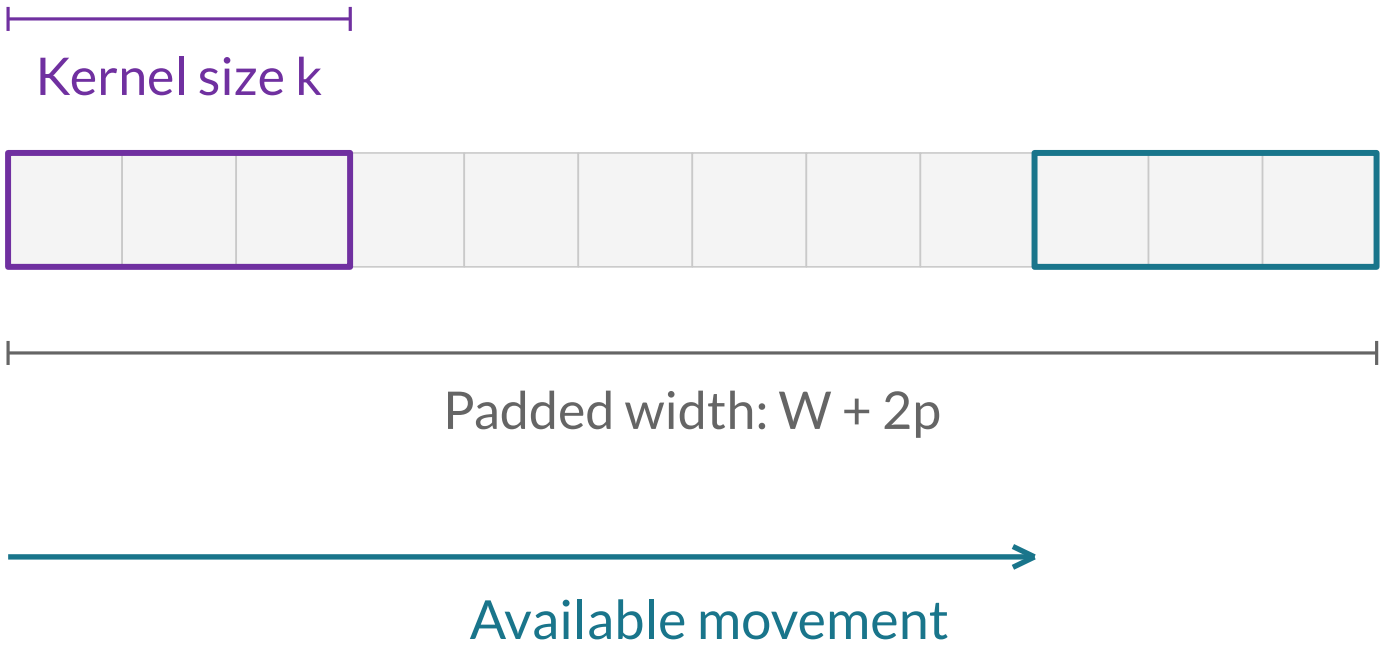
4 complete filters

Each filter spans all input channels

4 feature maps

Output dimensions

Count valid window placements (no dilation)



$$W_{\text{out}} = \left\lfloor \frac{W + 2p - k}{s} \right\rfloor + 1$$

Floor brackets round down to an integer.

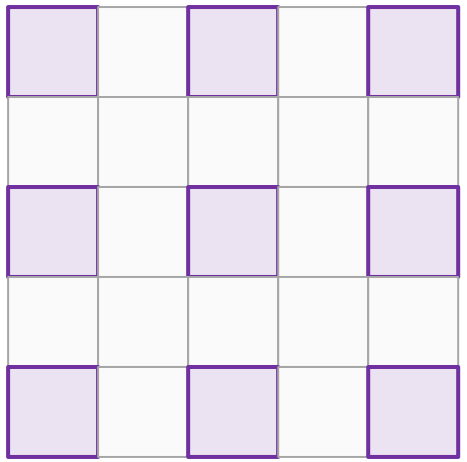
p: padding

s: stride

Dilation and output size

Dilation spaces the sampling positions without adding kernel weights.

3 × 3 kernel, dilation 2



9 sampled positions
5 × 5 effective span

$$k_{\text{eff}} = d(k-1)+1$$

$$W_{\text{out}} = \left\lfloor \frac{W + 2p - k_{\text{eff}}}{s} \right\rfloor + 1$$

W: input width p: padding s: stride
k: kernel width d: dilation

$$W = 9,\ p = 0,\ s = 1 \quad \Rightarrow \quad W_{\text{out}} = 5$$

Short answer: convolution dimensions

Input axes: batch × channels × height × width. No dilation.

Input shape

$$X: \ 2 \times 3 \times 24 \times 30$$

Layer

$$8 \text{ filters}, \quad 3 \times 3 \text{ kernel}, \quad s = 2, \quad p = 1$$

- Find the output shape and parameter count, including biases.
- Which of these changes if the batch size doubles?

Convolution dimensions: solution

Spatial output

$$H_{\text{out}} = \left\lfloor \frac{24 + 2 - 3}{2} \right\rfloor + 1 = 12, \qquad W_{\text{out}} = 15$$

Output tensor

$$2 \times 8 \times 12 \times 15$$

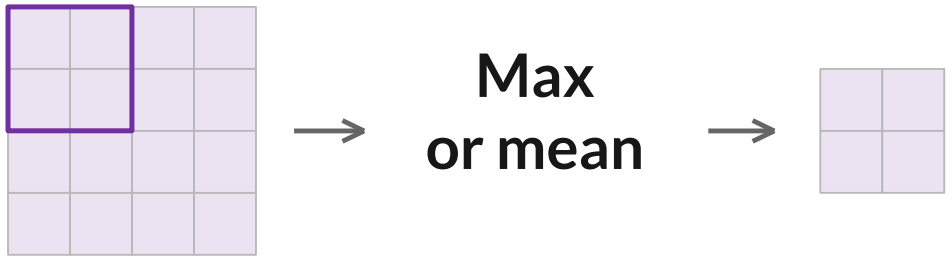
Parameters

$$8(3 \cdot 3 \cdot 3 + 1) = 224$$

- Doubling the batch changes 2 to 4. The parameter count stays 224.

Pooling and strided convolution

Pooling



Local summaries in each channel

No learned weights

Strided convolution

Can mix and change channels

Learned weights

Both can reduce spatial resolution.

Short answer: receptive-field size

Square kernels, no dilation. Consider an output away from image boundaries.

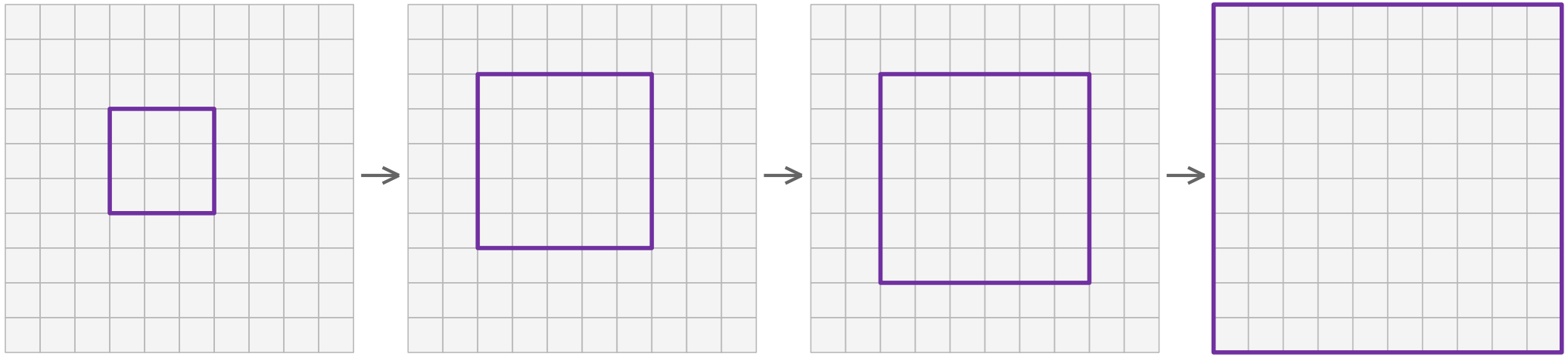
1. 3 × 3 convolution, stride 1
2. 3 × 3 convolution, stride 1
3. 2 × 2 max pooling, stride 2
4. 3 × 3 convolution, stride 1

Find the final theoretical receptive-field size in the original image.

Receptive-field size: solution

Theoretical receptive field in the input image

Conv 3×3 **Conv 3×3** **Pool 2×2, stride 2** **Conv 3×3**



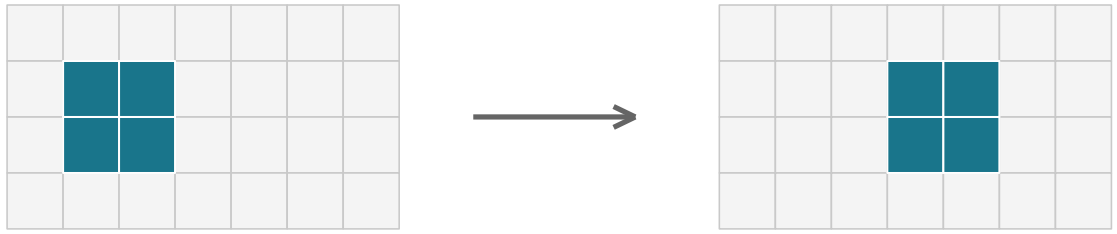
Input span: 3 **Input span: 5** **Input span: 6** **Input span: 10**

Center spacing: 1 Center spacing: 1 Center spacing: 2 Center spacing: 2

Final span: $6 + (3 - 1) \times 2 = 10$ input pixels.

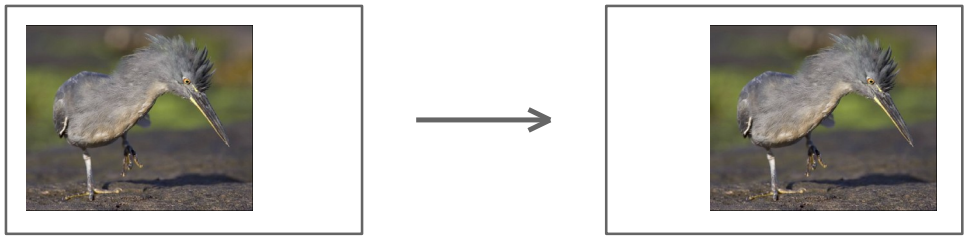
Shifts in features and class labels

Stride-1 convolution: equivariance



Responses shift with the input away from image boundaries.

Class label: desired invariance



Bird

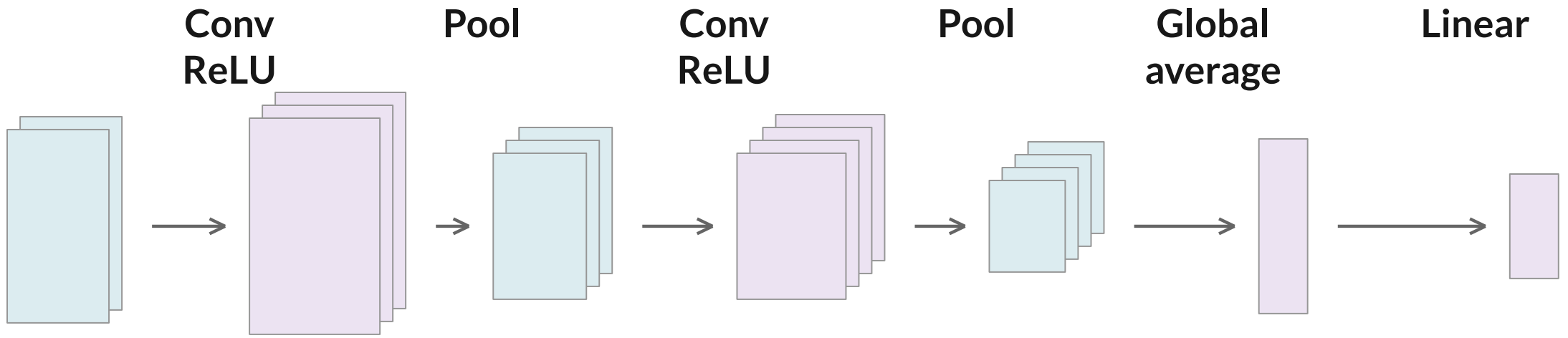
The class label stays the same

The classifier must combine responses across locations.

Source: MIT 6.7960

Example CNN classifier

One classical design for a 10-class image task.



Input

Class logits

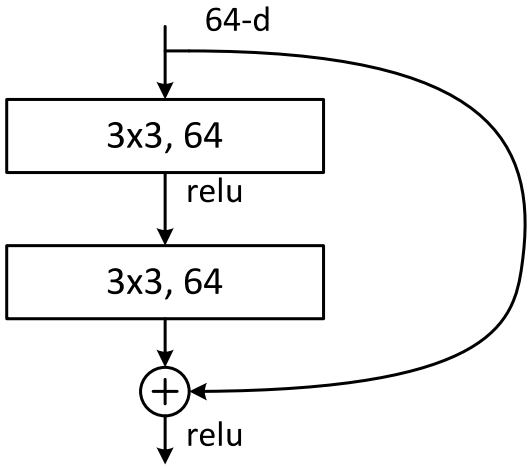
ResNet v1 blocks

Shortcut + learned residual branch

$$y = \mathrm{ReLU}(x + F(x))$$

Basic block example

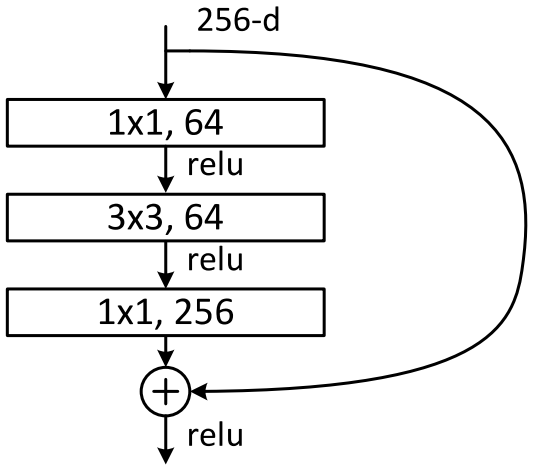
Two convolutions with BatchNorm



3 × 3 convolutions

Bottleneck block example

Three convolutions with BatchNorm



1 × 1 layers reduce and restore channels.

Source: arxiv.org/abs/1512.03385

Residual learning and optimization

Representing a good function

$$u = x + F(x)$$

- At the sum, F(x) = 0 preserves x.
- A larger hypothesis class can contain the smaller model's function.

Finding good parameters

- Worse training error can indicate optimization difficulty.
- The shortcut gives a direct gradient contribution through the sum.
- ResNet v1 applies ReLU after this sum.

If a deeper network has worse training error, does that by itself show overfitting?

Review outline

01 Networks, tensors and objectives

02 Learning and generalization

03 Convolutional networks

04 Visual tasks and transfer

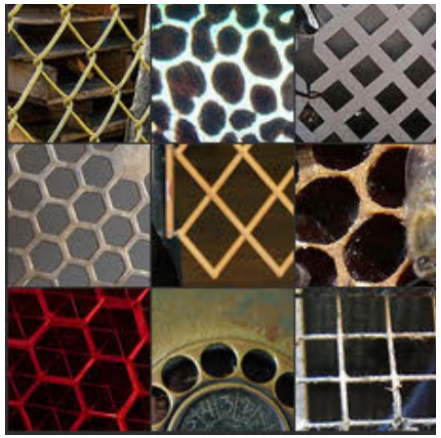
05 Recurrent sequence models

06 Attention and Transformers

Distributed features in a CNN

A class score combines many learned feature responses.

Examples from different layers



Layer 3

Layer 4

- One image activates several features.
- One feature can help several classes.
- Features have no predefined names.

Class score

$$s_k = \sum_c w_{kc} z_c + b_k$$

z: final feature vector
w: classifier weights

Source: Visualizing and Understanding Convolutional Networks

Short answer: frozen features and finetuning

Compare frozen-feature training with full finetuning.

$$h = f_\theta(x), \qquad s = Wh + b$$

θ : backbone parameters W, b: head parameters s: class logits

- Which parameters are updated in frozen-feature training?
- Which are updated in full finetuning?
- Why often use a small learning rate for the pretrained backbone?

Frozen features and finetuning: answer

Frozen-feature training

Update W and b. Keep θ fixed.
The head learns a new readout of the existing features.

Full finetuning

Update θ, W and b.
The target loss can adapt the features as well as the readout.

Small backbone learning rate

Often start small to limit disruption of useful pretrained features.

Gradients with respect to different variables

Training

$$\theta \leftarrow \theta - \eta \nabla_\theta L(\theta, x, y)$$

- Change model parameters θ.
- Keep the example x and label y fixed.

Untargeted attack

$$\text{increase } L(\theta, x, y) \text{ by changing } x$$

- Keep the trained model fixed.
- Constrain the input change to the allowed budget.

Fast gradient sign method (FGSM)

Image values lie in [0, 1]. Keep the model weights fixed.

1 Compute the input gradient

$$g = \nabla_x \mathcal{L}(f_\theta(x), y)$$

2 Step along the gradient sign

$$x_{\text{adv}} = \text{clip}_{[0,1]}(x + \epsilon \ \text{sign}(g))$$

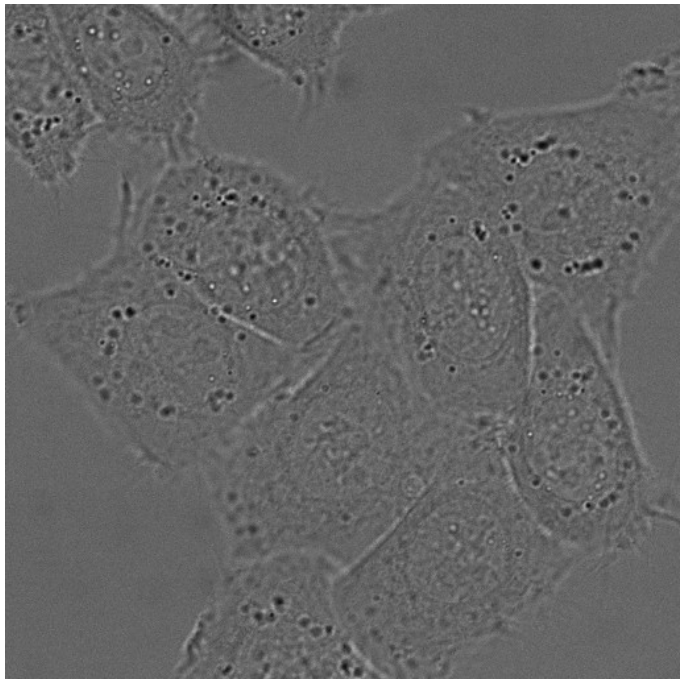
sign: +1 for positive slope, −1 for negative slope

Clip each value to [0, 1]

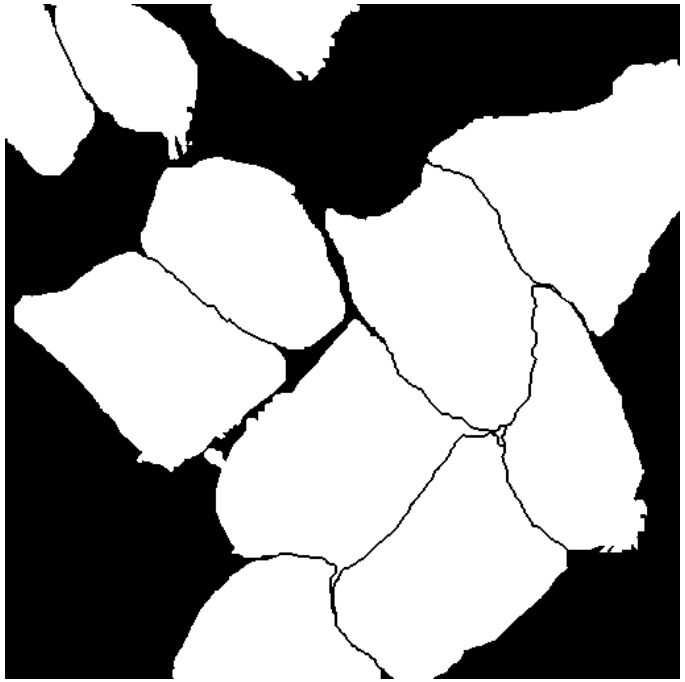
ϵ : largest allowed change in one image value

Semantic and instance masks

Image

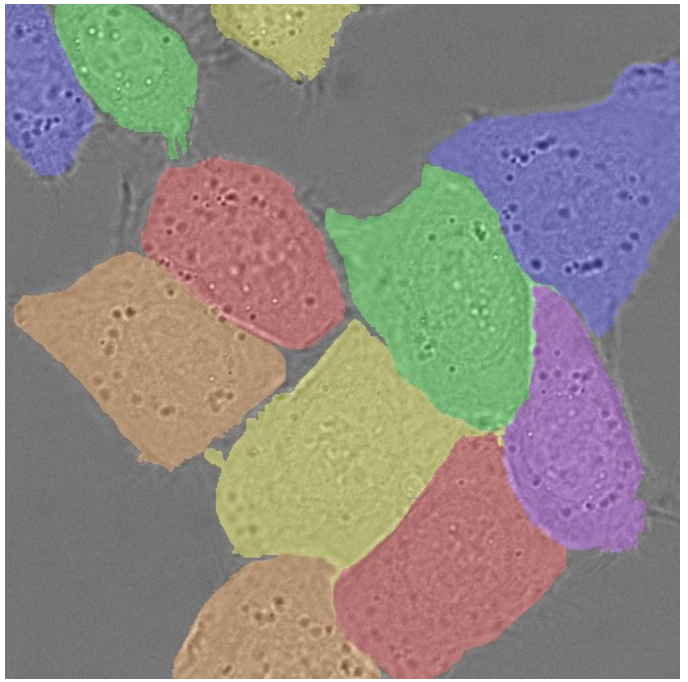


Cell / background



Semantic labels

Individual cells



Instance IDs

Source: U-Net

FCN with bilinear upsampling

One baseline: resize the class scores

Image

CNN backbone

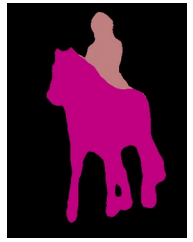
1 × 1 conv

Coarse logits

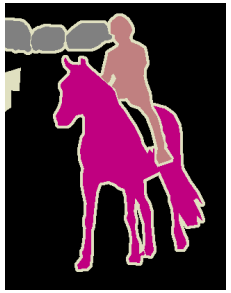
Bilinear resize

Pixel logits

argmax



Prediction



Target mask

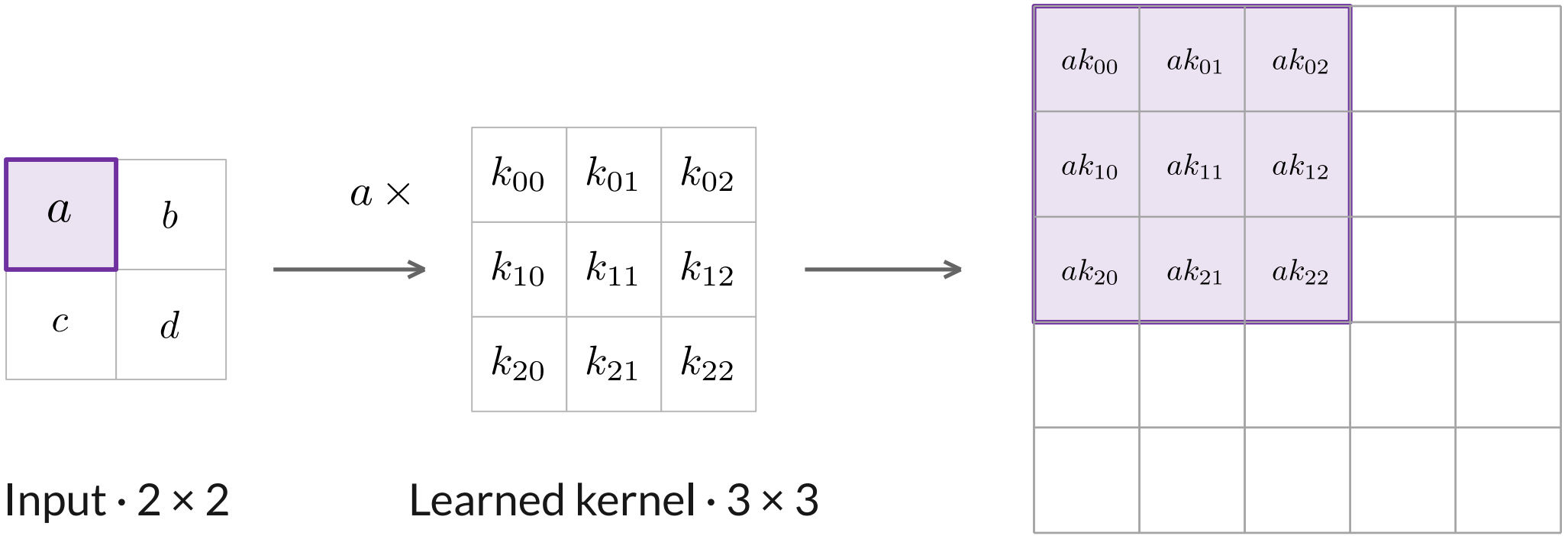
Cross-entropy

Training

Source: FCN

Transposed convolution

Place a scaled kernel at each input position. Add overlaps.



Input · 2 × 2

Learned kernel · 3 × 3

Subscripts: kernel row, column

Stride 2 · no padding

Contribution to a 5 × 5 output

Source: CS231n

Short answer: receptive field after upsampling

Kernel 3 × 3, stride 2, no padding or dilation.

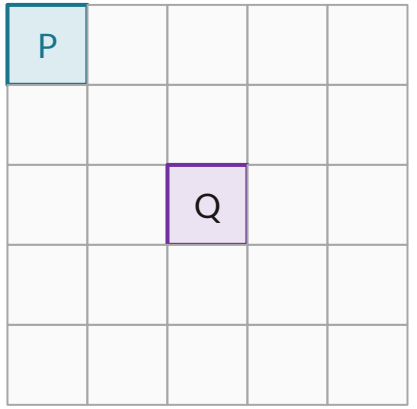
Each input feature covers 5 × 5 original-image pixels.
Adjacent centers are 2 original-image pixels apart. Ignore boundary clipping.

Input features

a	b
c	d

Transposed convolution →

Output map: 5 × 5

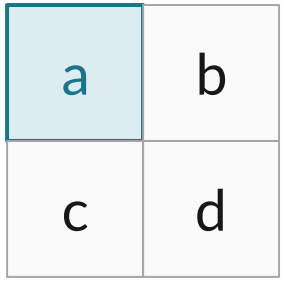


For P and Q: which input features can contribute?

What is each receptive-field size in the original image?

Transposed convolution: receptive-field solution

Top-left output P

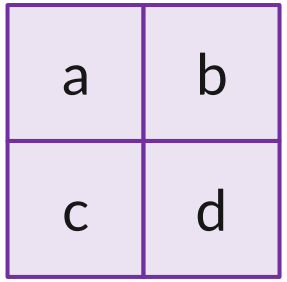


Only a contributes.

Receptive field

$$5 \times 5$$

Center output Q



All four features contribute.

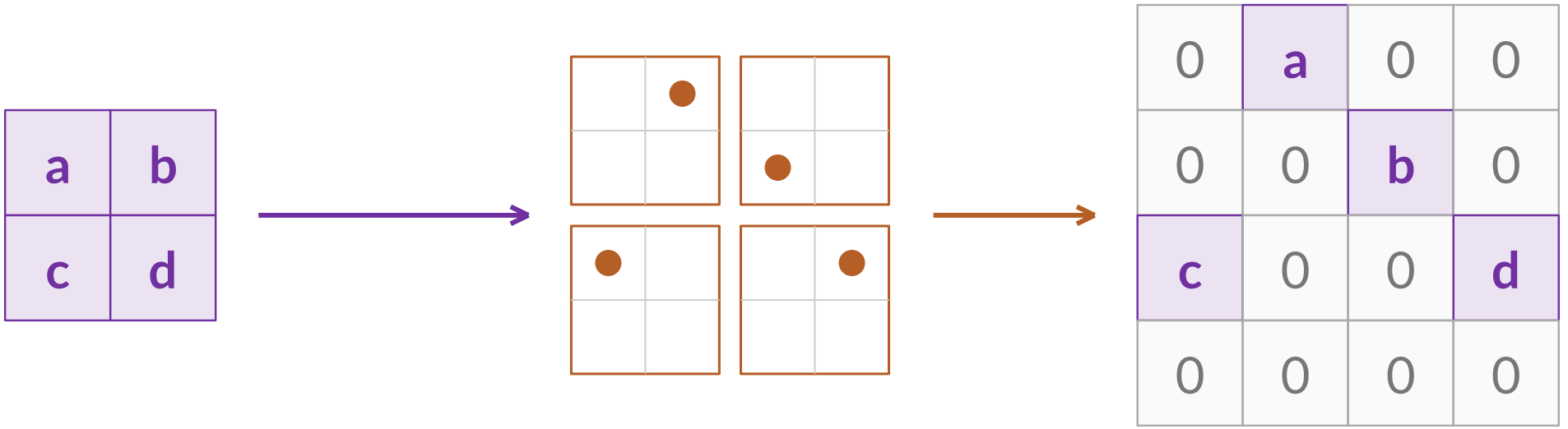
Receptive field

$$5 + 2 = 7 \quad \Rightarrow \quad 7 \times 7$$

For Q, two 5-pixel spans are shifted by 2 pixels along each axis.

Max unpooling

Current decoder values **Saved locations** **Unpooled map**

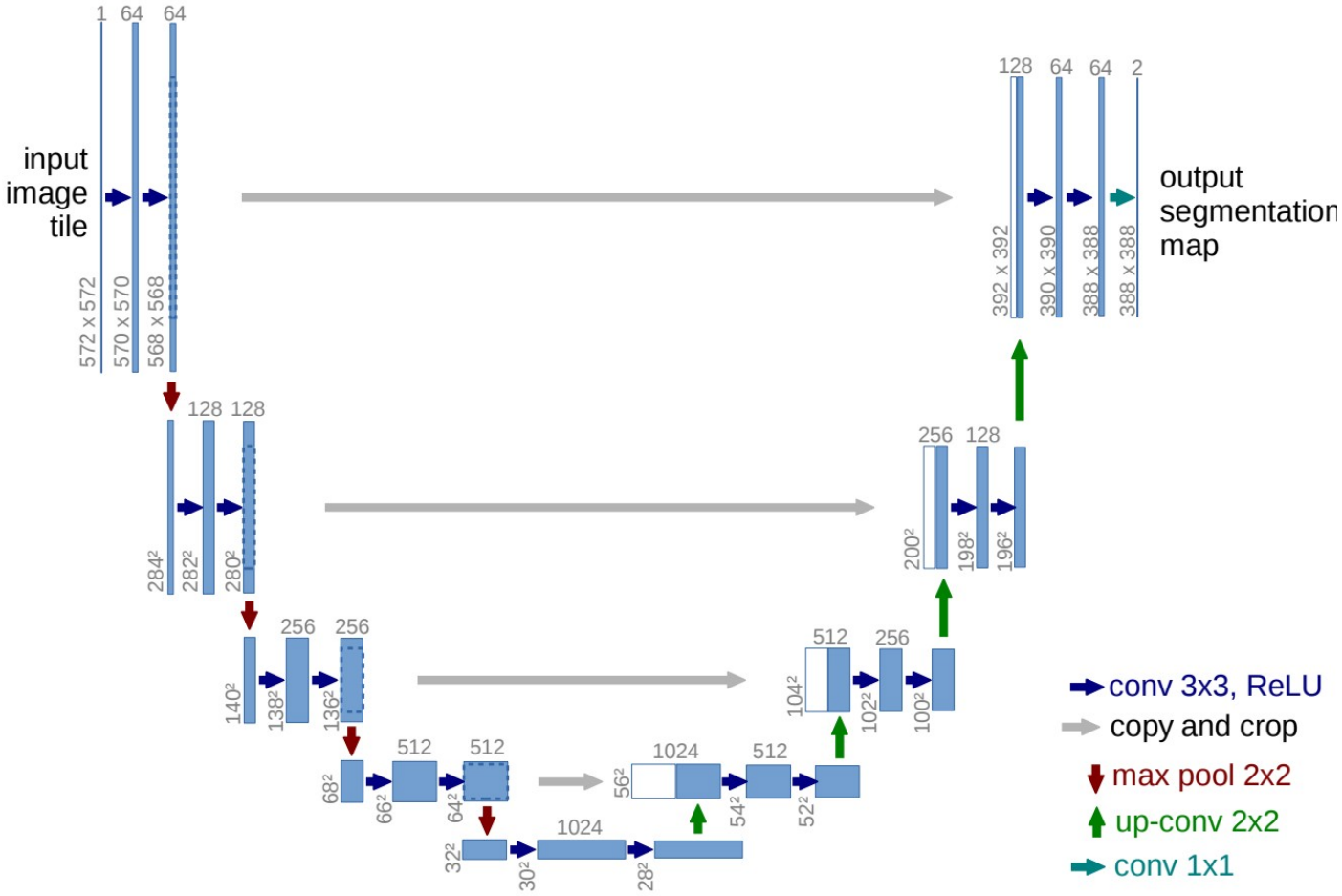


Place values at the locations saved during max pooling.

Can saved locations recover the discarded values? Why?

Source: SegNet

U-Net



Why reuse features?

Concatenate encoder and decoder features.

Bring fine spatial detail into the decoder.

Source: U-Net

Short answer: pixel predictions and evaluation

Four images, three classes. Every pixel has a valid class label.

Logits: batch × classes × height × width

$$Z: \ 4 \times 3 \times 16 \times 20$$

Foreground pixel counts

$$\text{truth} = 20, \quad \text{prediction} = 25, \quad \text{overlap} = 15$$

- State the target shape and the prediction axis.
- Find the foreground IoU. Explain why pixel accuracy can hide a missed object.

Pixel predictions: solution

Target shape and class prediction

$$Y: \ 4 \times 16 \times 20, \qquad \hat{Y} = \underset{\text{class}}{\operatorname{argmax}} Z$$

Foreground overlap

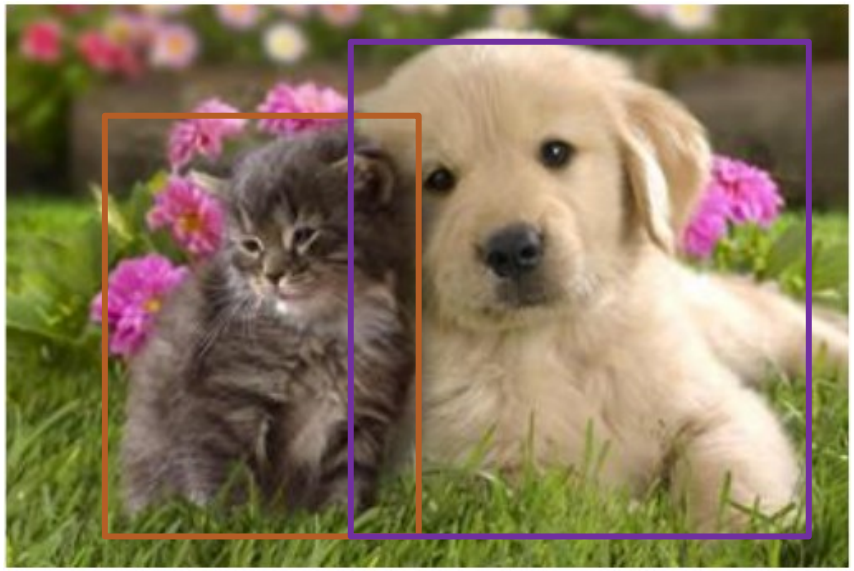
$$\text{IoU} = \frac{15}{20 + 25 - 15} = 0.5$$

- argmax selects the class with the largest logit at each pixel.
- A large background region can dominate pixel accuracy.

Fast R-CNN: shared image features

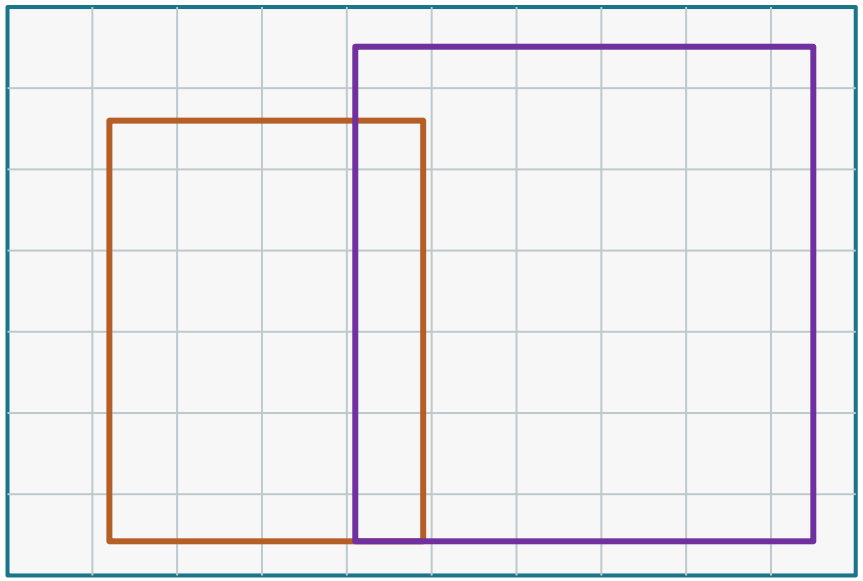
Share one CNN pass across all proposals.
Proposal boxes still come from a separate method.

Image and proposal coordinates



One CNN pass

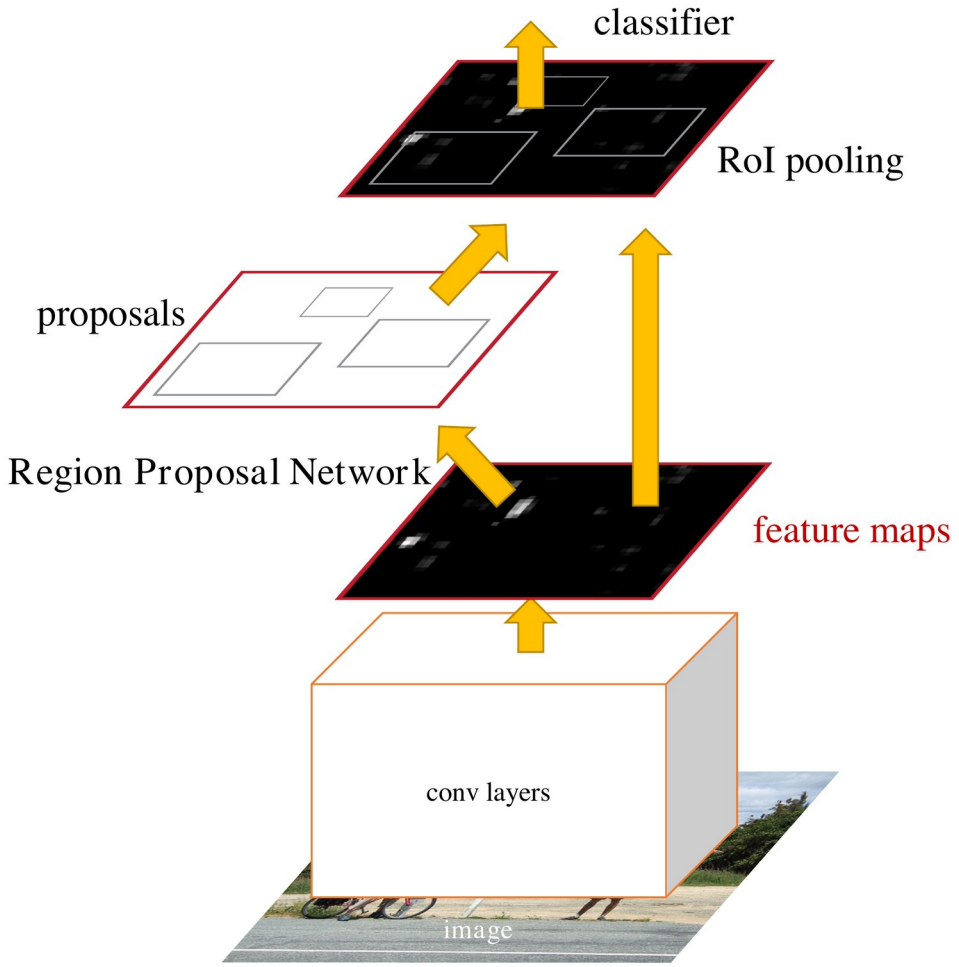
Shared feature map



RoI (region of interest): the feature region selected by a proposal.

Source: Stanford CS231n

Faster R-CNN



2 Classify and refine

Pool features for each proposal.
Predict category and a refined box.

1 Learn to propose regions

RPN predicts object/background scores and corrected boxes.

Both stages use the same image features.

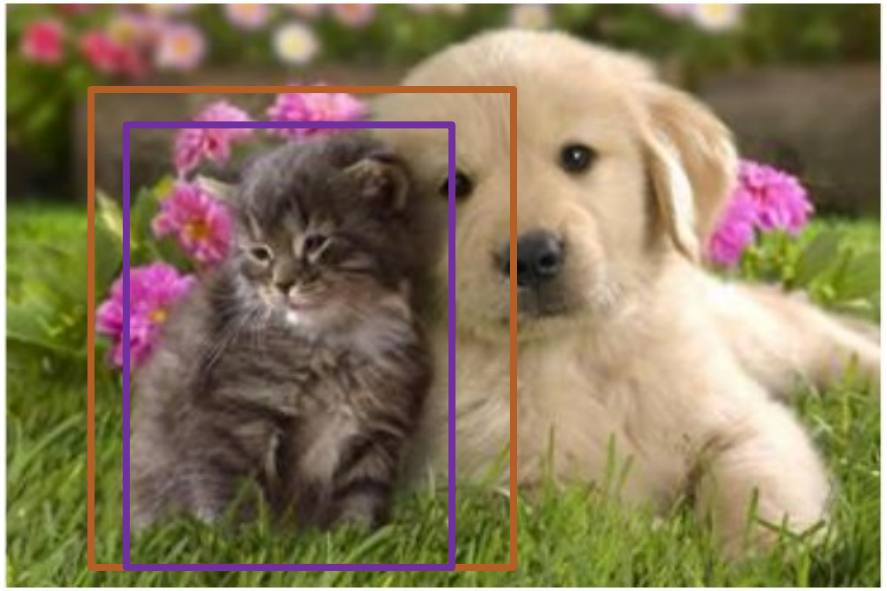
Source: Faster R-CNN

Training Faster R-CNN

Why is box loss used only for positive matches?

RPN

Anchor Annotated box

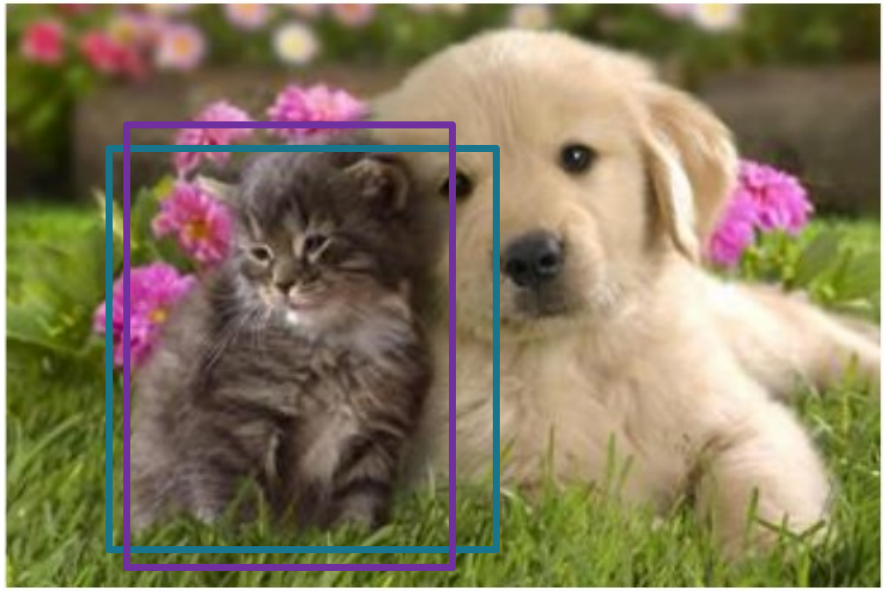


Objectness CE (target: object)

Box loss: fit the annotated box

Region detector

Proposal Annotated box



Category CE (target: cat)

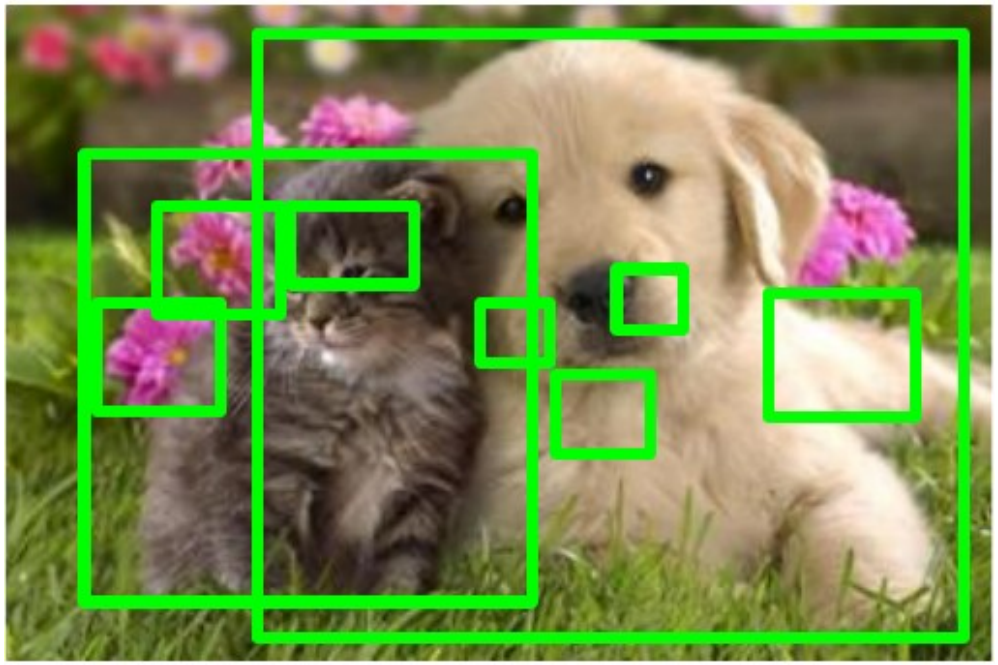
Box loss: fit the annotated box

Box loss uses positive matches only. All losses train the shared CNN.

Source: CS231n

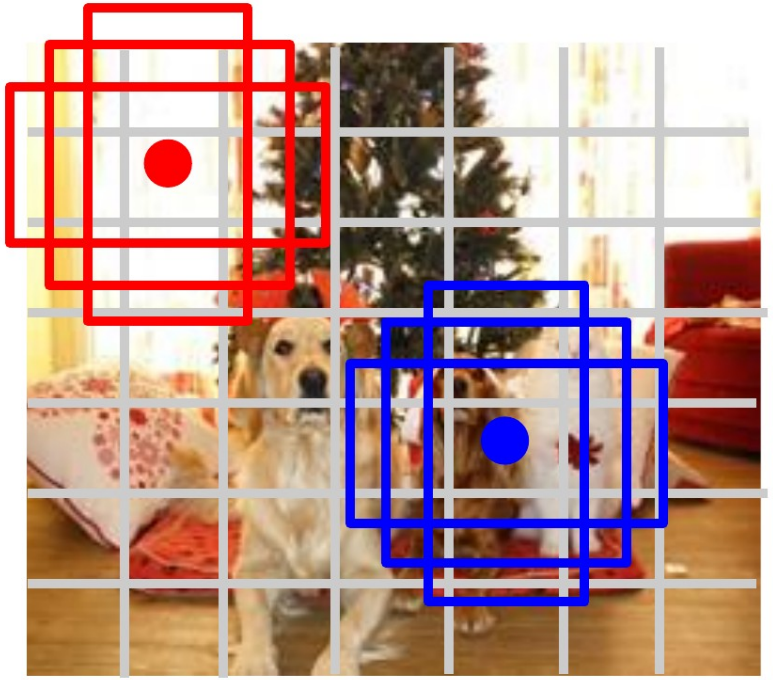
One-stage object detection

Two-stage



Select proposals, then classify and refine each region.

One-stage example with anchors



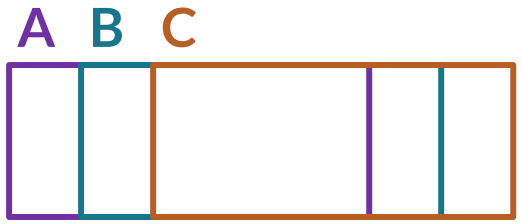
Predict class scores and boxes directly at feature locations.

Source: CS231n 2025 · CS231n 2026

Non-maximum suppression

Select all correct statements

Three boxes predict the same category. Scores rank A above B above C. Greedy NMS suppresses overlaps above 0.5.



$$\mathrm{IoU}(A, B) = \frac{2}{3}, \quad \mathrm{IoU}(B, C) = \frac{2}{3}, \quad \mathrm{IoU}(A, C) = \frac{3}{7}$$

A. The algorithm keeps A and C.

B. B suppresses C even after B has been removed.

C. Keeping a box does not establish that it matches a real object.

D. Increasing the threshold to 0.8 still suppresses B.

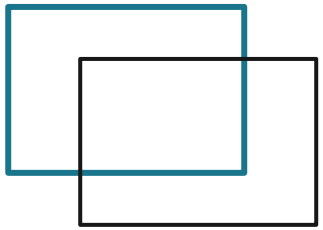
Non-maximum suppression: answer

Correct: A, C

A	**Correct**	Keep A and suppress B. The overlap between A and C is below the threshold, so C survives.
B	**Incorrect**	Only a kept box suppresses remaining candidates. A suppressed box is not used later.
C	**Correct**	NMS compares predicted boxes with one another. Correctness needs comparison with annotated objects.
D	**Incorrect**	All three pairwise overlaps are below 0.8, so all three boxes survive.

Three uses of IoU

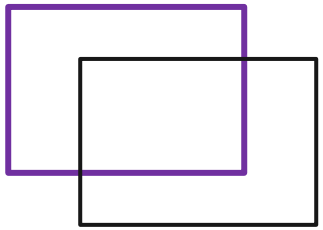
Training



Candidate → target

Assign supervision

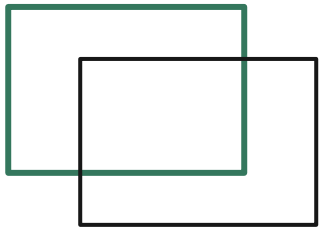
NMS



Prediction → prediction

Remove duplicates

Evaluation



Prediction → target

Count correct objects

Precision and recall

Example: person detection

predictions

ground truth

1 true positive (TP): correct detection

2 false positives (FP): false alarms

1 false negative (FN): missed person

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} = \frac{1}{3}$$

1 correct out of 3 predicted detections

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} = \frac{1}{2}$$

1 found out of 2 annotated people

Source: UW CSE455

Choosing a confidence threshold

Keep the trained detector fixed.
Retain a detection if its score s reaches the chosen threshold τ.

$$s \geq \tau$$

Higher threshold

Fewer retained detections.
Can miss more true objects.

Lower threshold

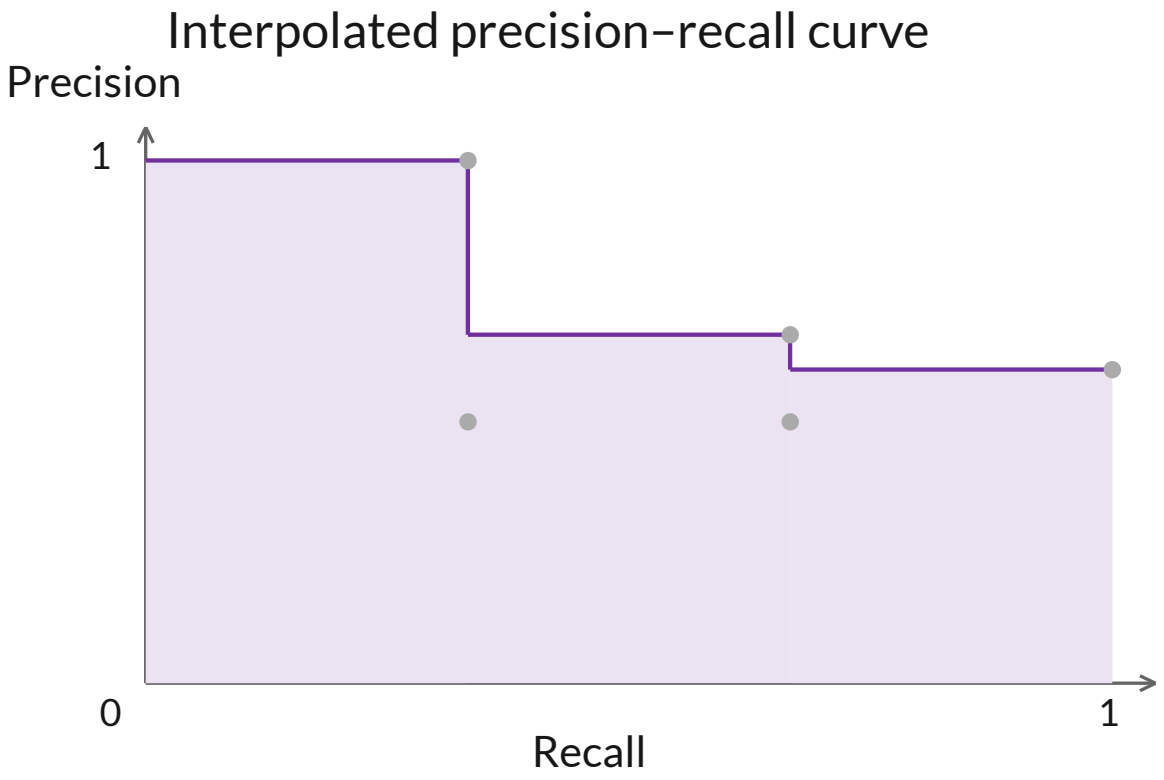
More retained detections.
Can admit more false alarms.

Choose τ on validation data for the required miss / false-alarm trade-off.

Detection evaluation: AP and mAP

Correct detection: same class, enough overlap, object not already matched.

Vary the confidence threshold to trace the curve. Keep the IoU rule fixed.



$$\text{Precision} = \frac{\text{correct detections}}{\text{predicted detections}}$$

$$\text{Recall} = \frac{\text{detected objects}}{\text{annotated objects}}$$

Average precision (AP)

Area under the curve for one class

Mean average precision (mAP)

Mean of AP across classes

Review outline

01 Networks, tensors and objectives

02 Learning and generalization

03 Convolutional networks

04 Visual tasks and transfer

05 Recurrent sequence models

06 Attention and Transformers

Recurrent updates, states and parameters

General recurrent update

$$h_t = f_\theta(x_t, h_{t-1})$$

x_t: current input h_{t-1}: previous state h_t: updated state

Common vanilla-RNN implementation

$$h_t = \tanh(W_x x_t + W_h h_{t-1} + b)$$

This example uses W_x, W_h and b as its learned parameters.

Parameters θ are learned and shared across time.
The state changes as a sequence is processed.

Autoregressive language modeling

Autoregressive: predict the next token from the sequence's own past

[BOS]: beginning of sequence [EOS]: end of sequence

$$p(\text{birds}, \text{fly}, [\text{EOS}] \mid [\text{BOS}])$$

$$= p(\text{birds} \mid [\text{BOS}])$$
$$\times\, p(\text{fly} \mid [\text{BOS}], \text{birds})$$
$$\times\, p([\text{EOS}] \mid [\text{BOS}], \text{birds}, \text{fly})$$

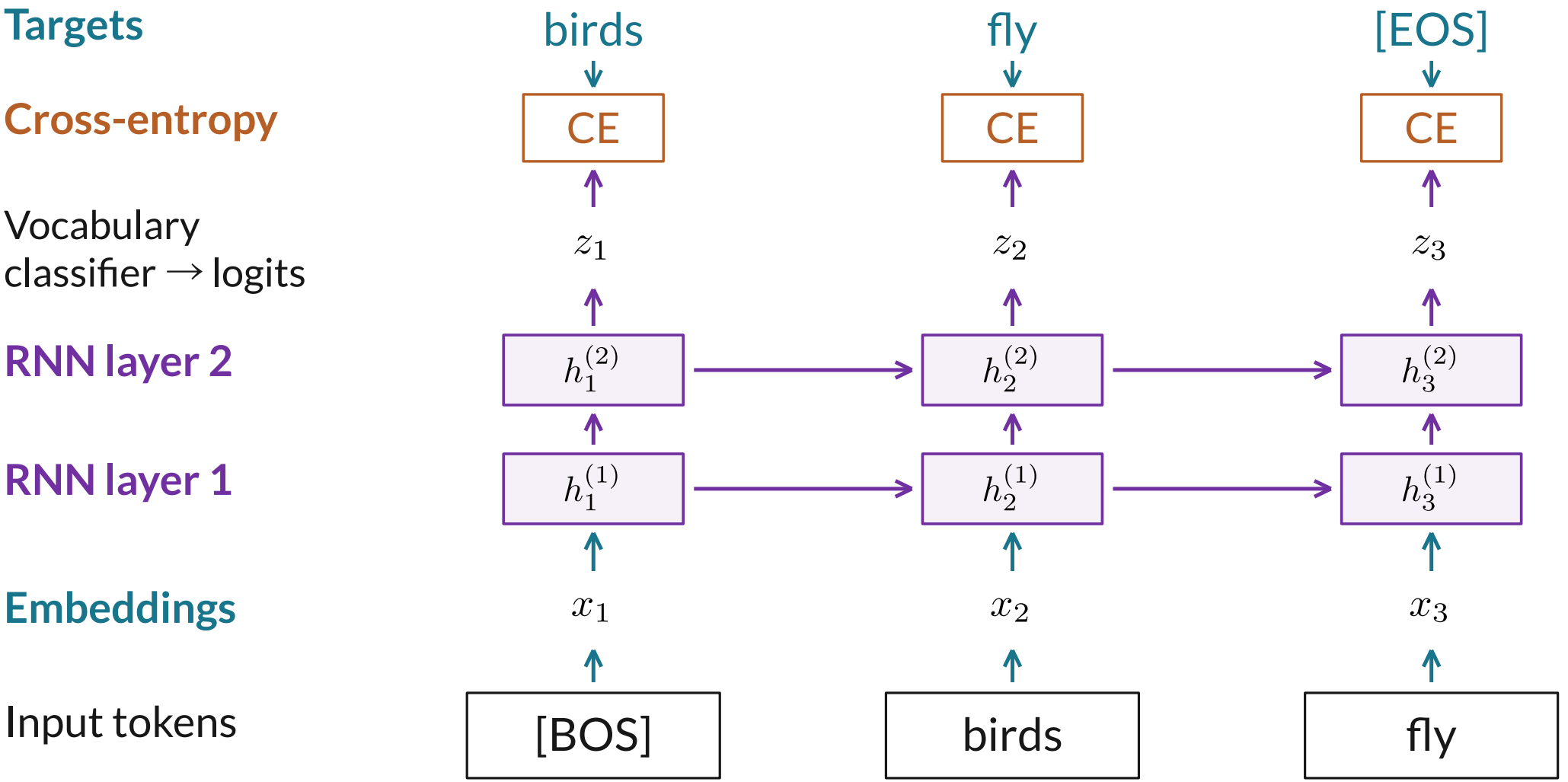
Token sequence:

$$u_0 = [\text{BOS}], \quad u_{1:T} = (u_1, \ldots, u_T), \quad u_T = [\text{EOS}]$$

$$p_\theta(u_{1:T} \mid u_0) = \prod_{t=1}^{T} p_\theta(u_t \mid u_{0:t-1})$$

Multiply the next-token probabilities

The recurrent language model



Short answer: training prefixes

Observed sequence: [BOS] birds sing outside [EOS].

- A proposed training loop feeds back its own token sleep instead of the observed token sing.
- It still trains the following prediction against outside. Which prefix is that loss now conditioned on?
- Write the inputs used by teacher forcing, and contrast them with generation.

Training prefixes: solution

Teacher-forced training

- Inputs: [BOS], birds, sing, outside.
- Targets: birds, sing, outside, [EOS].
- Each loss uses its recorded prefix.

Autoregressive generation

- Append the model's chosen token.
- Use this prefix for the next token.
- Parameters stay fixed. States and inputs change.

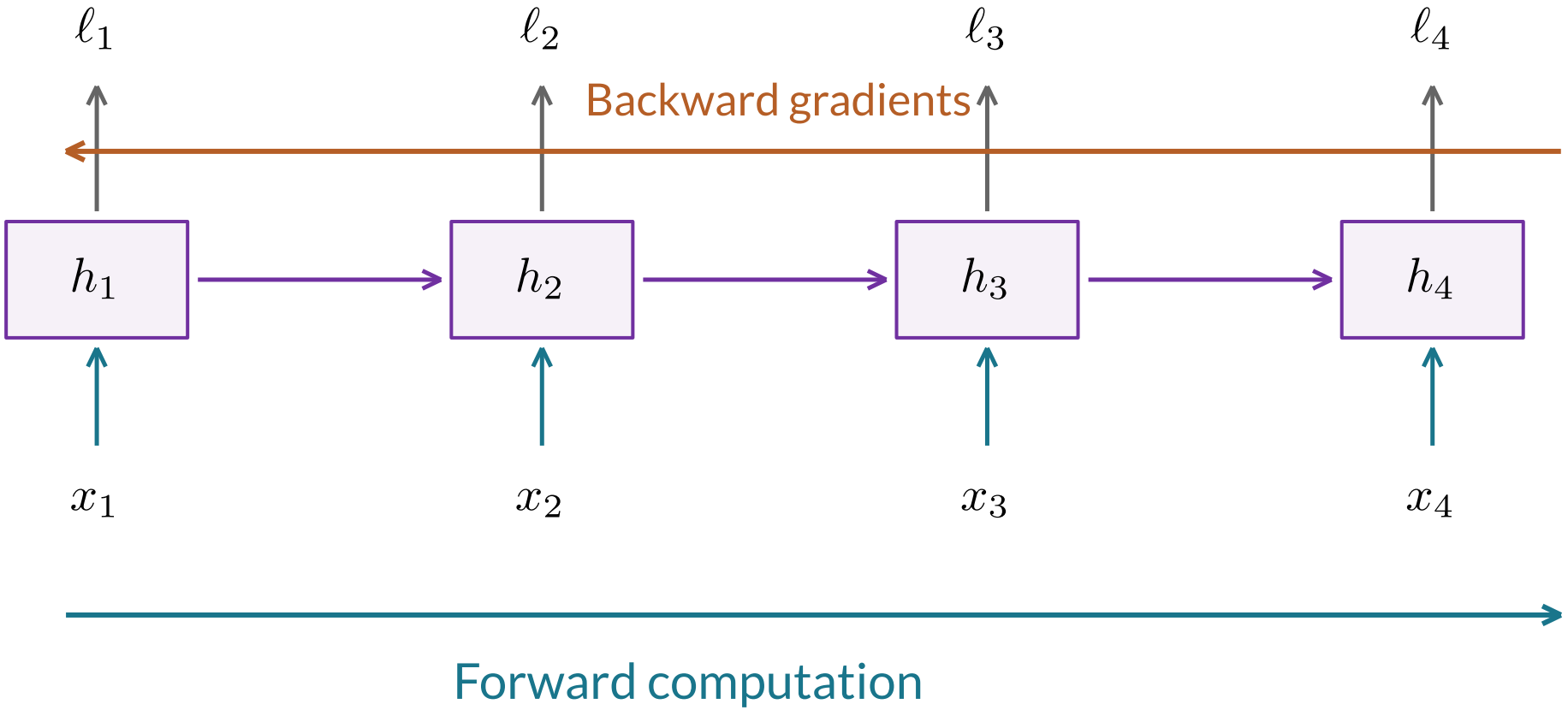
The proposed loop predicts "outside" from "[BOS] birds sleep".

Wrong generated tokens become inputs to later steps.

Backpropagation through time (BPTT)

Summed step losses

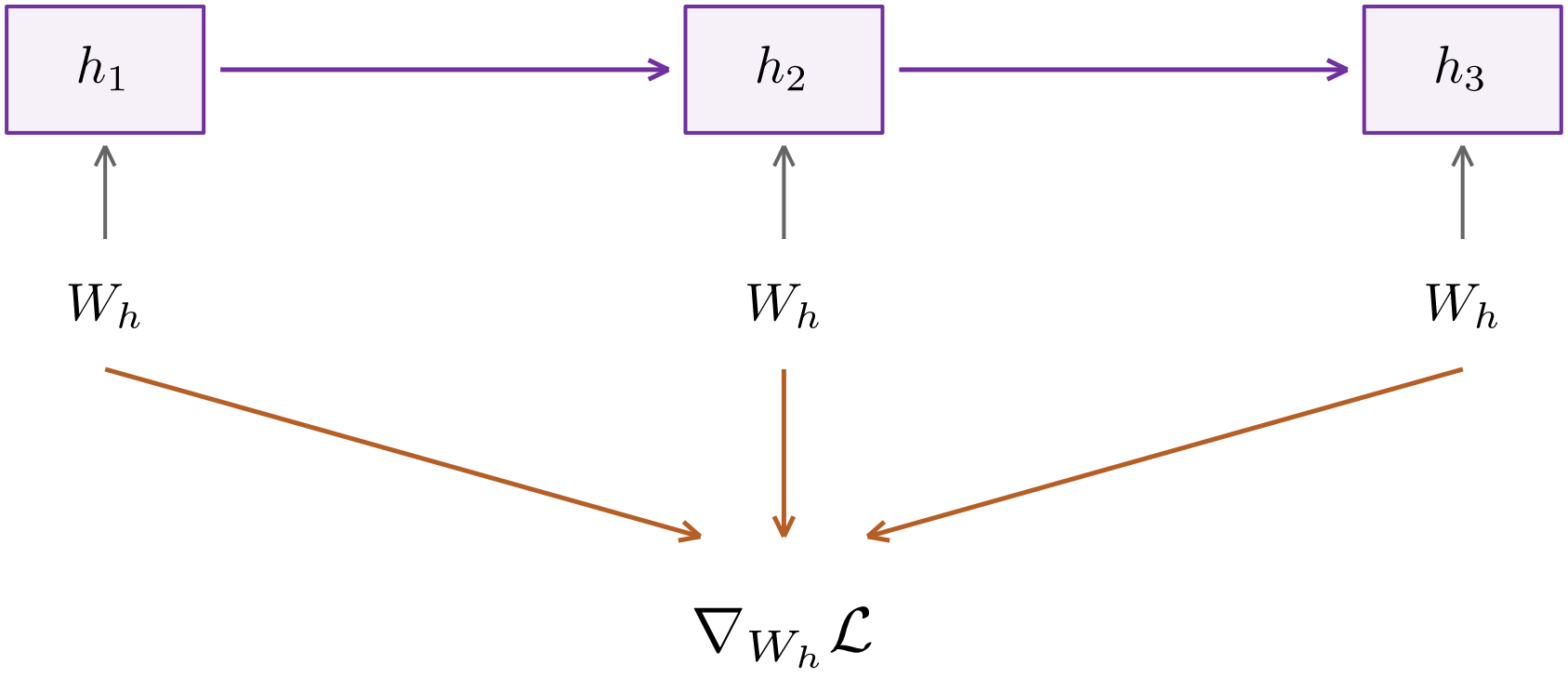
$$\mathcal{L} = \sum_t \ell_t$$



Source: Adapted from Georgia Tech CS 7643

Gradients of shared parameters

Same parameter tensor at every step



Add the contributions from every use.

Source: Adapted from Berkeley CS 182

Long gradient paths and clipping

Repeated derivative factors can make long-range gradients vanish or explode.

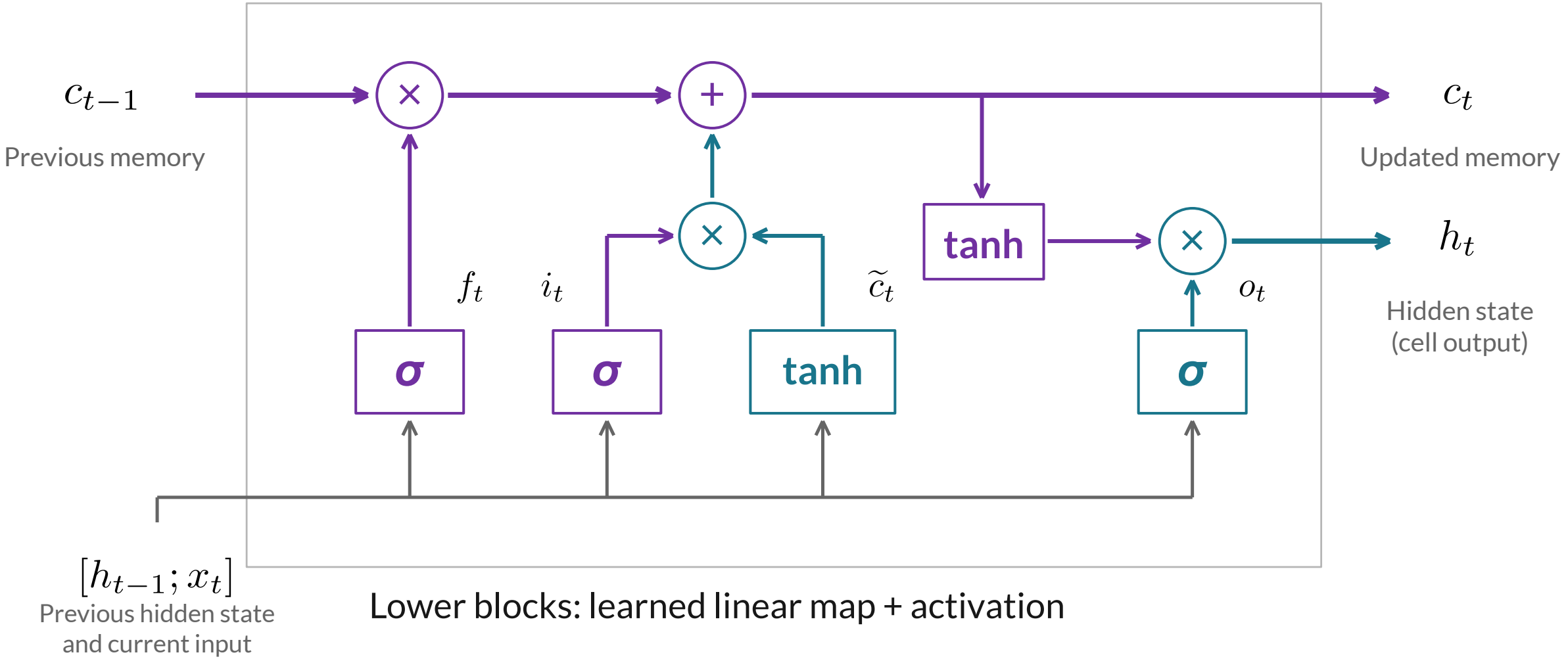
Gradient-norm clipping

$$g_{\text{clip}} = \begin{cases} g, & \|g\|_2 \le c, \\ \dfrac{c}{\|g\|_2} g, & \|g\|_2 > c. \end{cases}$$

g: gradient before the update
$\| g \|_2$: its length
c: maximum allowed length

- Scale down a large gradient without changing its direction.
- Clipping limits explosions. It does not repair vanishing gradients.

LSTM cell



Lower blocks: learned linear map + activation

LSTM equations

Gates and proposed content

Forget $$f_t = \sigma(W_f[h_{t-1}; x_t] + b_f)$$

Input $$i_t = \sigma(W_i[h_{t-1}; x_t] + b_i)$$

Output $$o_t = \sigma(W_o[h_{t-1}; x_t] + b_o)$$

Candidate $$\widetilde{c}_t = \tanh(W_c[h_{t-1}; x_t] + b_c)$$

State updates

Stored memory

$$c_t = f_t \odot c_{t-1} + i_t \odot \widetilde{c}_t$$

Hidden state

$$h_t = o_t \odot \tanh(c_t)$$

W, b: learned parameters x: input h: hidden state c: cell memory

LSTM memory and hidden output

Select all correct statements

Consider one nonzero cell-memory component. For this question, use ideal gate values of 0 or 1.

$$c_t = f_t c_{t-1} + i_t \widetilde{c}_t, \qquad h_t = o_t \tanh(c_t)$$

A. Closing the output gate erases the stored cell memory.

B. Forget gate 1 and input gate 0 retain the old cell memory.

C. The forget and input gates must add to 1.

D. Opening the output gate can reveal memory without rewriting it.

LSTM memory and hidden output: answer

Correct: B, D

A	Incorrect	The output gate controls the hidden output, not the cell-memory update.
B	Correct	The old memory is kept and no new candidate is written.
C	Incorrect	The gates are separate. Their values need not add to one.
D	Correct	The output gate exposes a transformed version of the stored cell state.

Sequence boundaries

Which sequence does the state belong to?

- Independent examples: reset h and c.
- Consecutive parts of one stream: carrying state can preserve context.

Which positions contain real observations?

- Ignore padded target losses.
- Use the last valid recurrent state for a sequence summary.
- A zero padding input is not automatically a no-op.

Review outline

01 Networks, tensors and objectives

02 Learning and generalization

03 Convolutional networks

04 Visual tasks and transfer

05 Recurrent sequence models

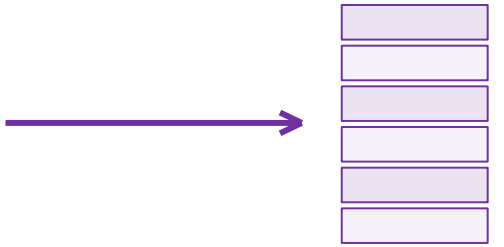
06 Attention and Transformers

The fixed-summary bottleneck

Source sentence

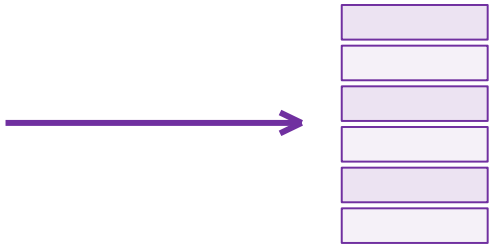
Fixed-size context

The red car stopped.



c

The red car beside the blue van did not stop.

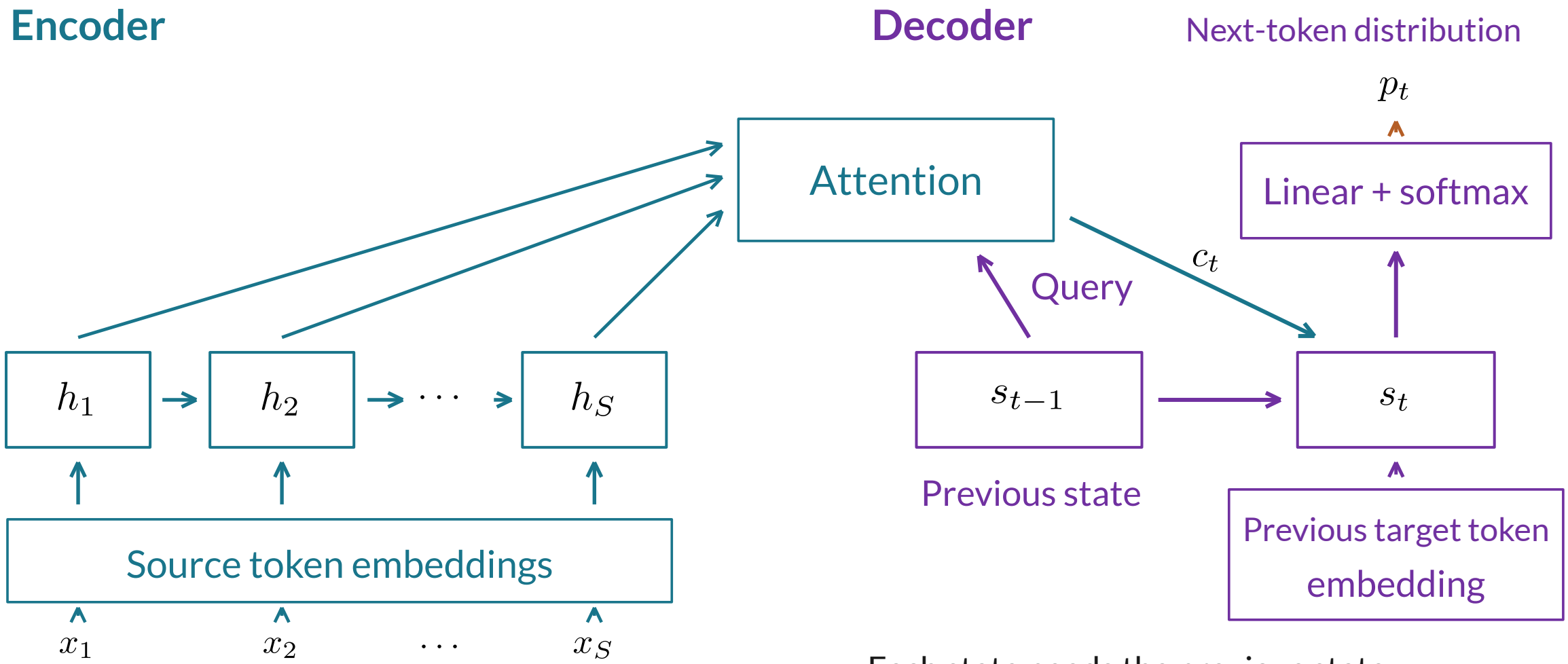


c

More source details, the same number of stored values.

Why let each decoder step retrieve different source details?

Attention in a recurrent decoder



Each state needs the previous state.

Source: Adapted from CMU 11-785

Queries, keys and values

U: query features X: source features One vector per row

$Q = UW_Q$	**Queries**	Information requested
$K = XW_K$	**Keys**	Features used for matching
$V = XW_V$	**Values**	Features to combine

Projection weights are learned. Q, K and V depend on the current input.

Self-attention uses the same sequence for queries and source: U = X.

Scaled dot-product attention

Q, K and V contain one vector per row.

$$S = \frac{QK^\mathsf{T}}{\sqrt{d_k}}$$

$$A = \mathrm{softmax}_{\mathrm{row}}(S)$$

$$O = AV$$

Compare queries with keys

d_k is the query/key width. Scaling controls the size of the matching scores.

Normalize each query row

Each row gives weights over source positions.

Combine the values

One output feature vector for each query.

Short answer: queries, keys and values

Encoder–decoder cross-attention. The full source is available.

3 target queries. 5 source rows, with row 5 padded.
Matching width 4. Value width 6.

- What are the score-matrix and output shapes? Which rows must correspond between keys and values?
- Does replacing the padded key vector with zero exclude it?
- May every target query read all four real source rows?

Queries, keys and values: solution

Each key row and value row describe the same source position.

Inputs

$$Q : 3 \times 4, \qquad K : 5 \times 4, \qquad V : 5 \times 6$$

Outputs

$$QK^{\mathsf{T}} : 3 \times 5, \qquad AV : 3 \times 6$$

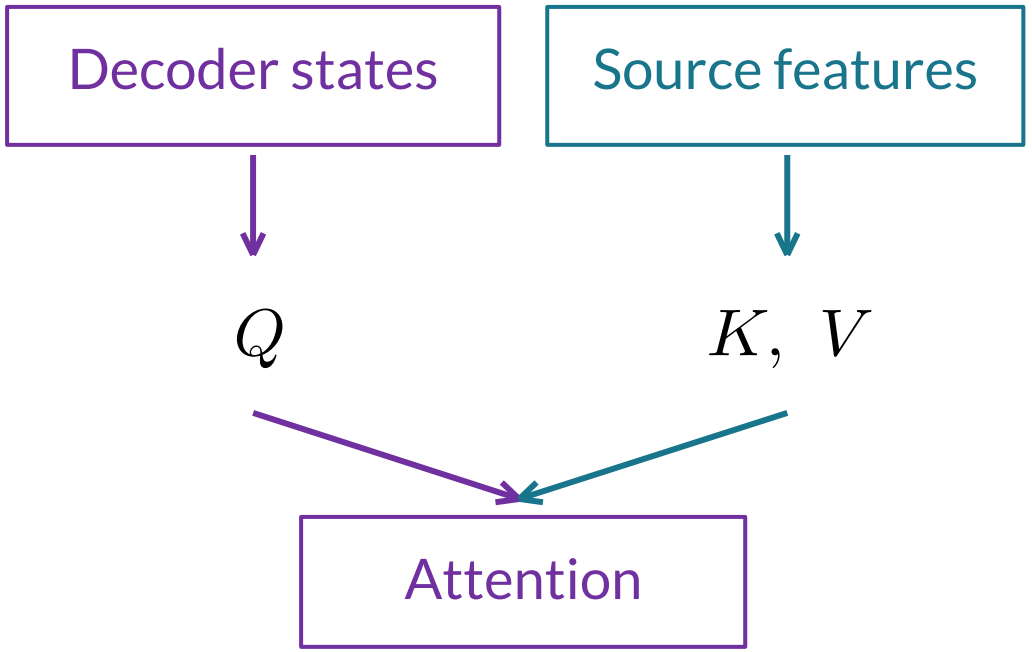
- A contains attention weights. Mask score column 5 with −∞ before softmax.

- A zero key still receives softmax weight. All four real source rows can be read.

Cross-attention and self-attention

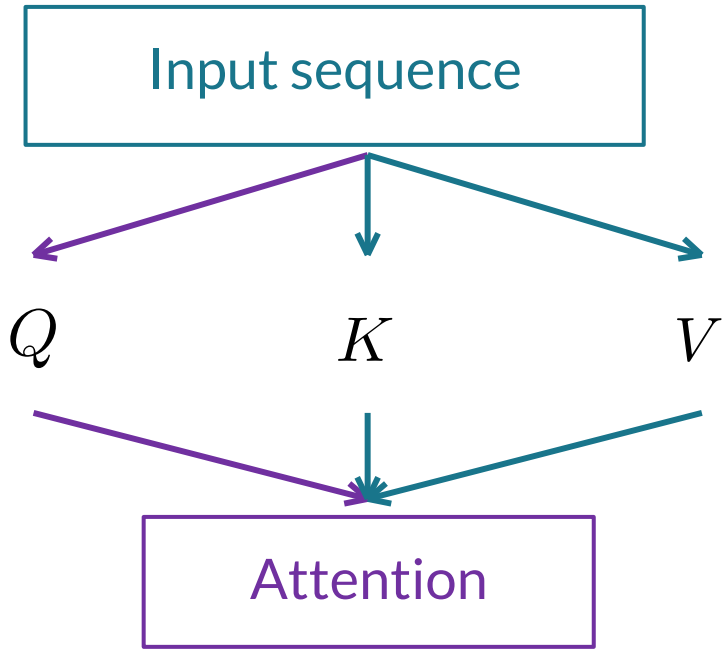
Does self-attention require Q, K and V to be identical?

Cross-attention



Read a separate source

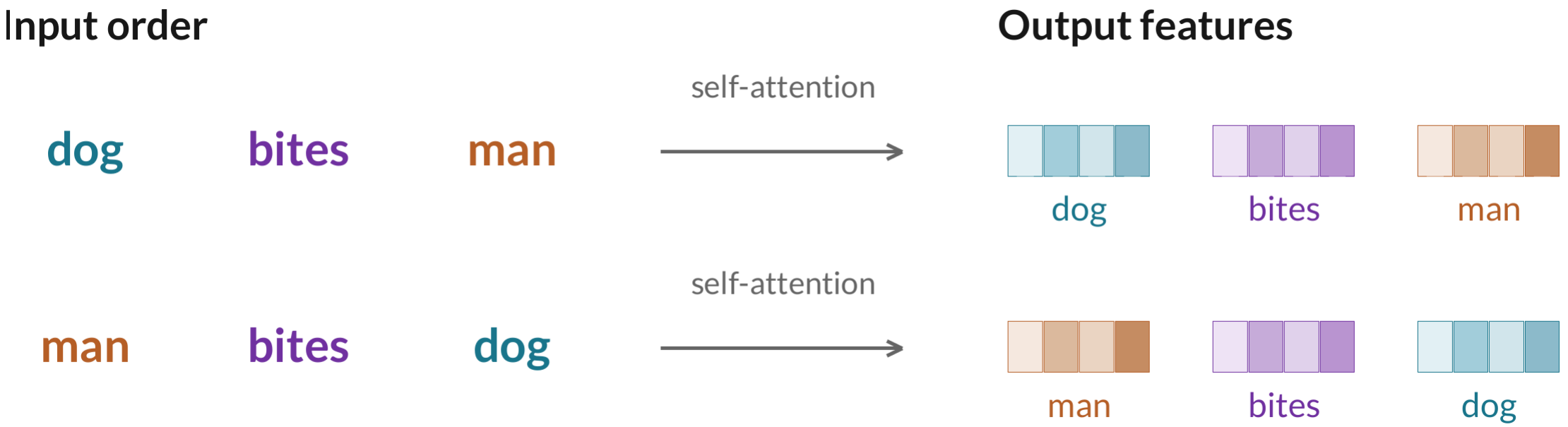
Self-attention



Build context within a sequence

Why supply positional information?

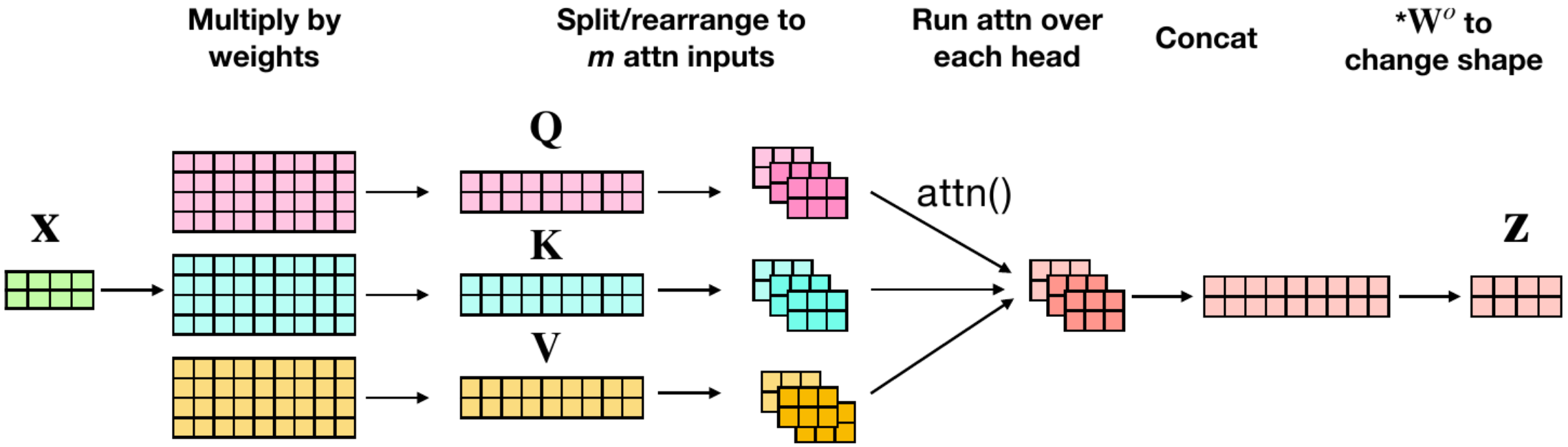
Unrestricted self-attention, without position information



Reordering inputs only reorders outputs: permutation equivariance.

$z_i = x_i + p_i$ One choice: token embedding + position vector

Multihead attention



Concat joins the head outputs for each token.

$$\text{MultiHead}(X) = \text{Concat}(H_1, \ldots, H_m)W_O$$

$X,\ Z$ input / output token vectors

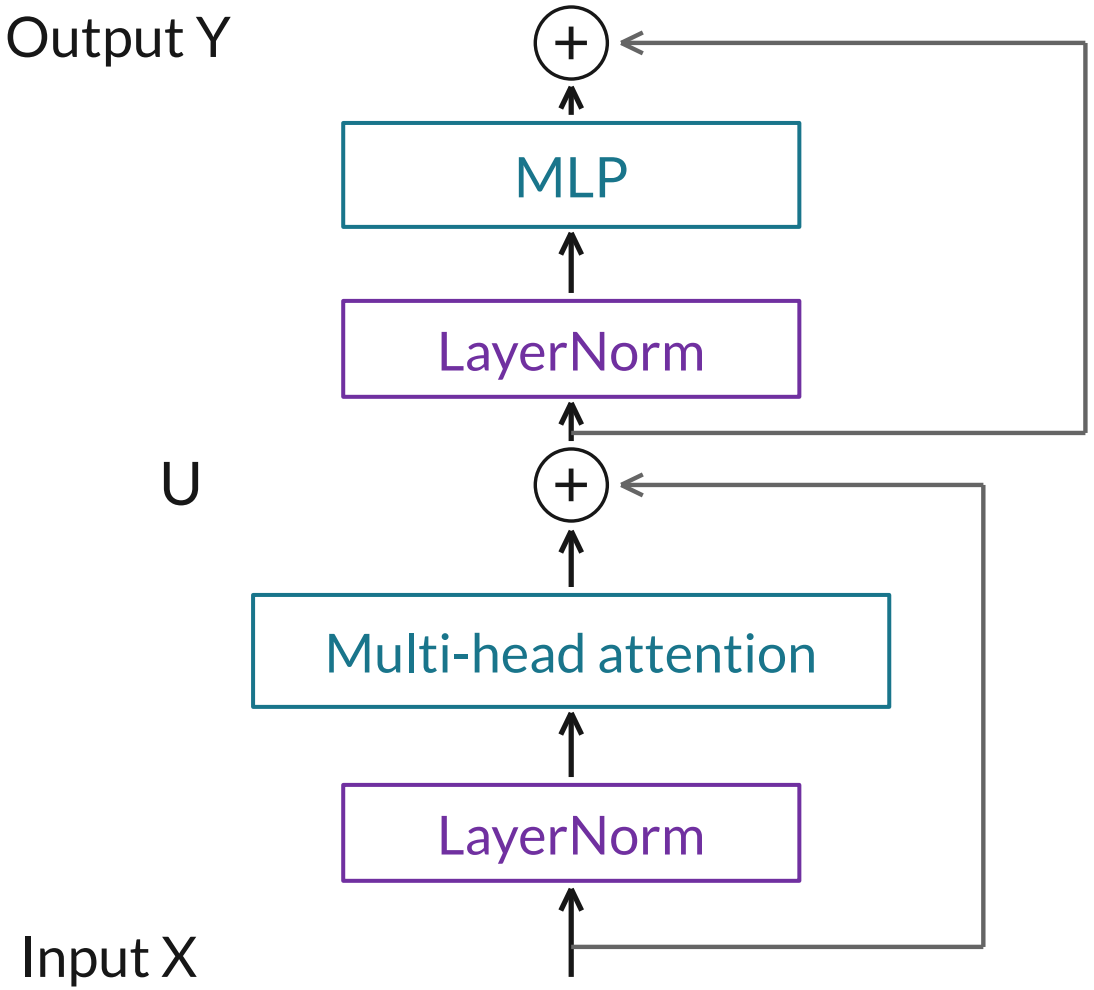
m number of heads

H_h output of head h

$W_O = W^O$ learned output projection

Source: Berkeley CS288

Transformer encoder block



Pre-normalized block

Attention update

$$U = X + \mathrm{MHA}(\mathrm{LN}(X))$$

Feature update

$$Y = U + \mathrm{MLP}(\mathrm{LN}(U))$$

LN: LayerNorm
MHA: multi-head self-attention

Roles within a Transformer block

Across positions

- Attention mixes token features according to the query.
- Heads form separate mixtures before concatenation and output projection.

Within each position

- The same MLP transforms each token separately.
- LayerNorm normalizes that token's feature coordinates.
- Residual connections add each token's input to its update.

Position vectors encode location. Adding them does not mix token features.

Causal attention across layers

Select all correct statements

A two-block decoder predicts the next token. Shifted inputs are [BOS], the, quiet, room. The input quiet predicts room.

A. Setting a future attention score to zero excludes that token.

B. A causal mask in only the final block prevents future-token leakage.

C. The quiet position may attend to its own input position.

D. A causal mask in each block prevents leakage along attention paths.

Causal attention across layers: answer

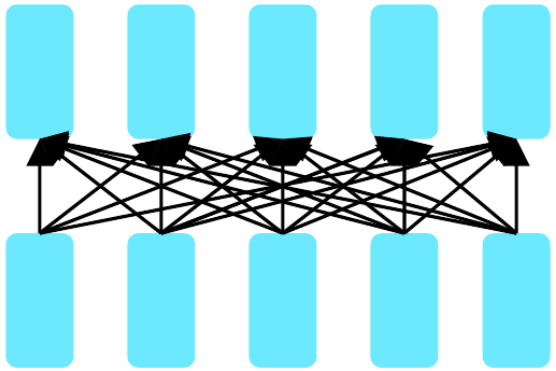
Correct: C, D

A	Incorrect	Softmax gives a zero score a positive weight. Set forbidden scores to $-\infty$ before softmax.
B	Incorrect	An earlier block can already mix future information into a position that the final block may read.
C	Correct	The current input is known in next-token prediction. Its target is the following token.
D	Correct	Each block restricts the positions it reads. Earlier features therefore cannot pass future information onward.

Transformer architectures

Encoder-only

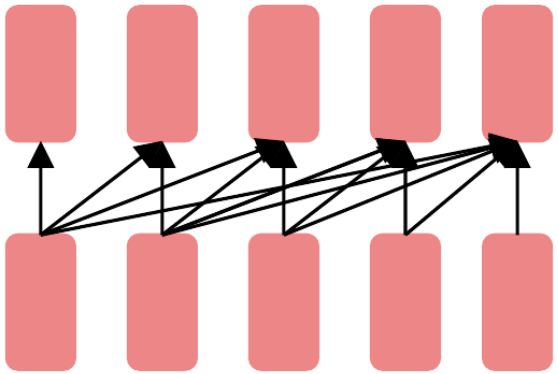
BERT



Full input context

Decoder-only

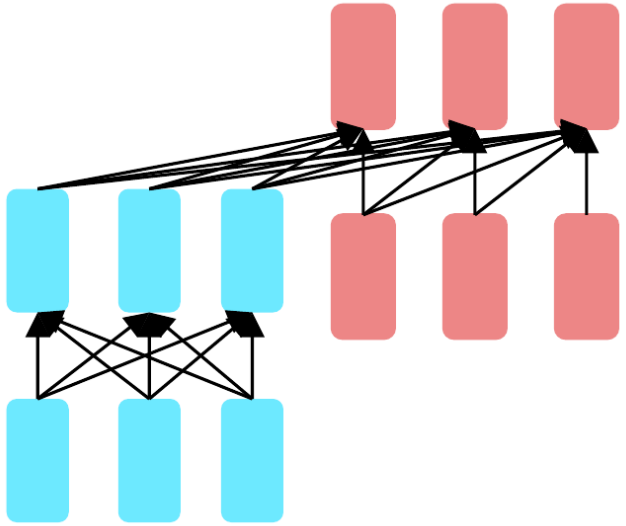
GPT



Prefix only

Encoder–decoder

T5



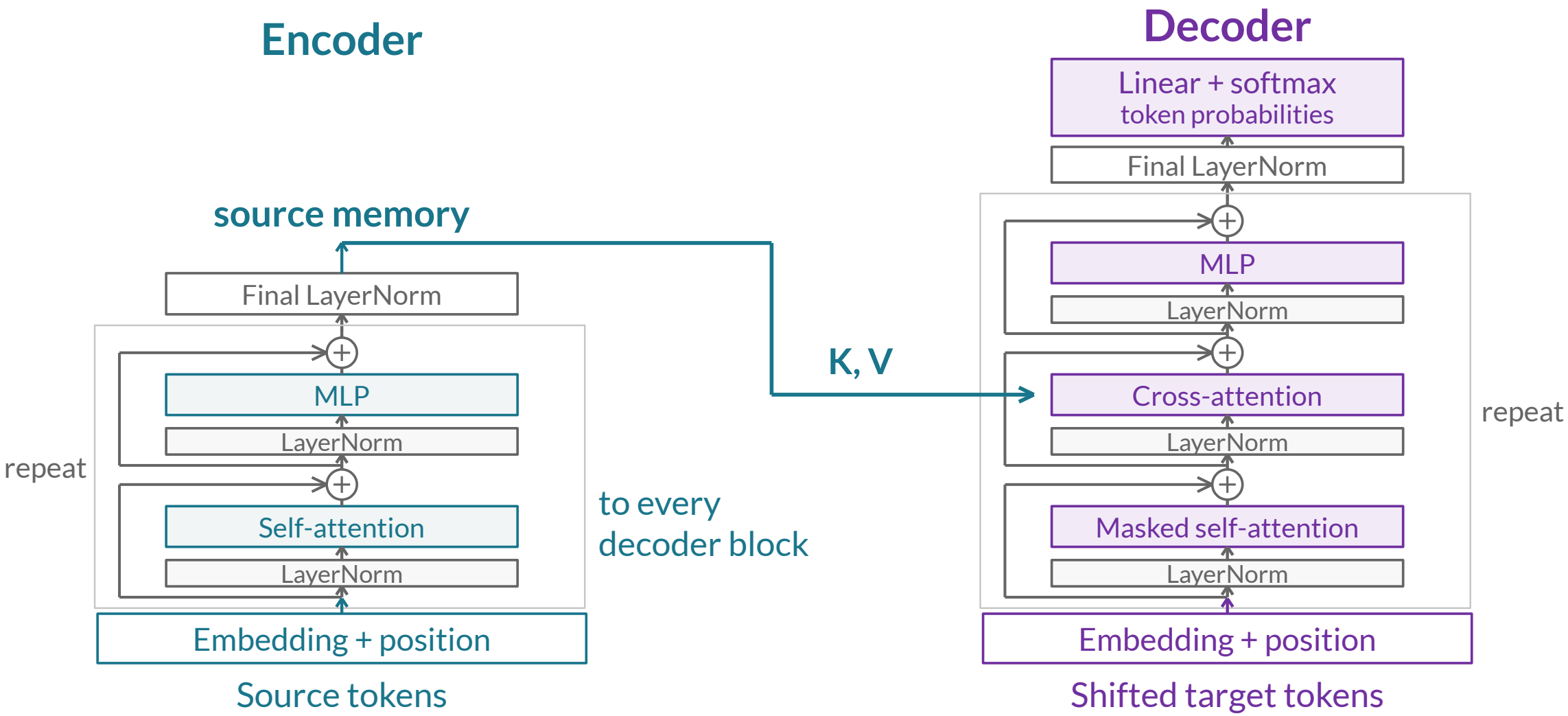
Source + target prefix

Rectangles: token features. Arrows: information access.

Source: Stanford CS224n

Encoder–decoder Transformer

Example with pre-LayerNorm blocks



Transformer training and generation

Teacher-forced training

- All shifted input tokens are known.
- Positions compute together within each layer under causal masking.
- Process the layers in sequence.

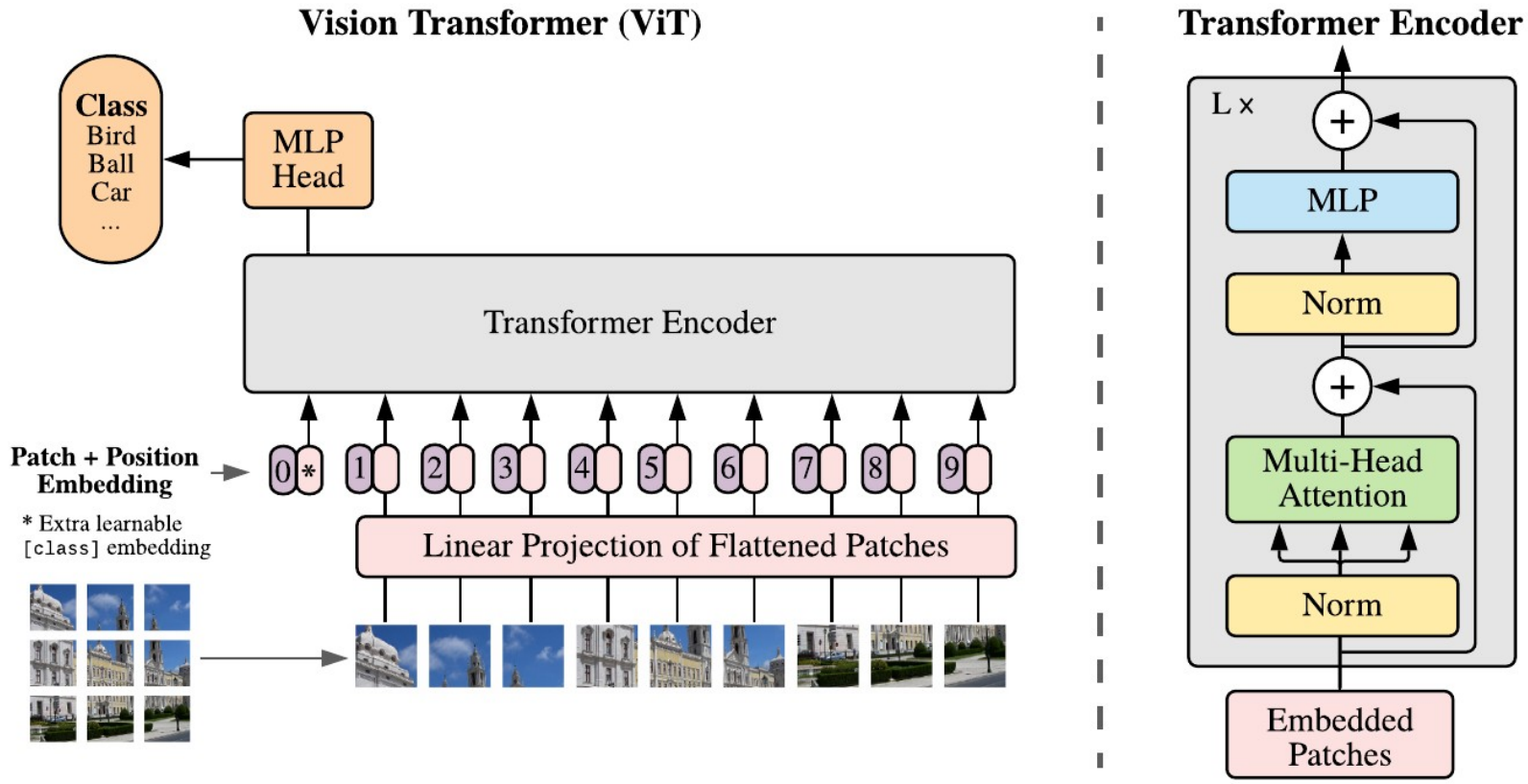
Autoregressive generation

- The next token is not known yet.
- Append the selected token, then predict the next.
- Model weights stay fixed.

Why does teacher forcing still leave an RNN sequential?

Each RNN state needs the previous state. Transformer positions read features from the previous layer.

Vision Transformer (ViT)



In the original ViT, [CLS] starts as a learned input token.
Its final feature gathers patch information for classification.

Source: An Image Is Worth 16×16 Words

Changing the patch size

Select all correct statements

Compare two ViT designs using non-overlapping square patches of the same RGB image. Halve the patch edge. Both sizes divide the image dimensions. Keep the model width fixed.

A. The number of patch tokens becomes four times larger.

B. Each flattened patch has one quarter as many raw RGB values.

C. Each token’s feature width must become one quarter as large.

D. With one [CLS] token, the total token count becomes exactly four times larger.

Changing the patch size: answer

Correct: A, B

A	**Correct**	Twice as many patches fit along each image axis, giving four times as many patch positions.
B	**Correct**	Each patch covers one quarter of the area. The number of color channels stays fixed.
C	**Incorrect**	The learned patch projection maps the raw patch into the chosen, unchanged model width.
D	**Incorrect**	If there were N patches, the total changes from N + 1 to 4N + 1.

Preparing for the midterm

- Explain the operation and why the model needs it.
- Explain why the main design choices are useful.
- For each multiple-choice option, identify the rule that makes it true or false.
- For short answers, connect the conclusion to a mechanism or calculation.

Try the questions before reading the worked answers.