DDA4220 · Fall 2026

Lecture 5

Deep Learning and Applications

Training neural networks III: generalization and training practice

Zhen Liu

CUHK-Shenzhen

Lecture outline

01 Generalization

02 Dropout and input noise

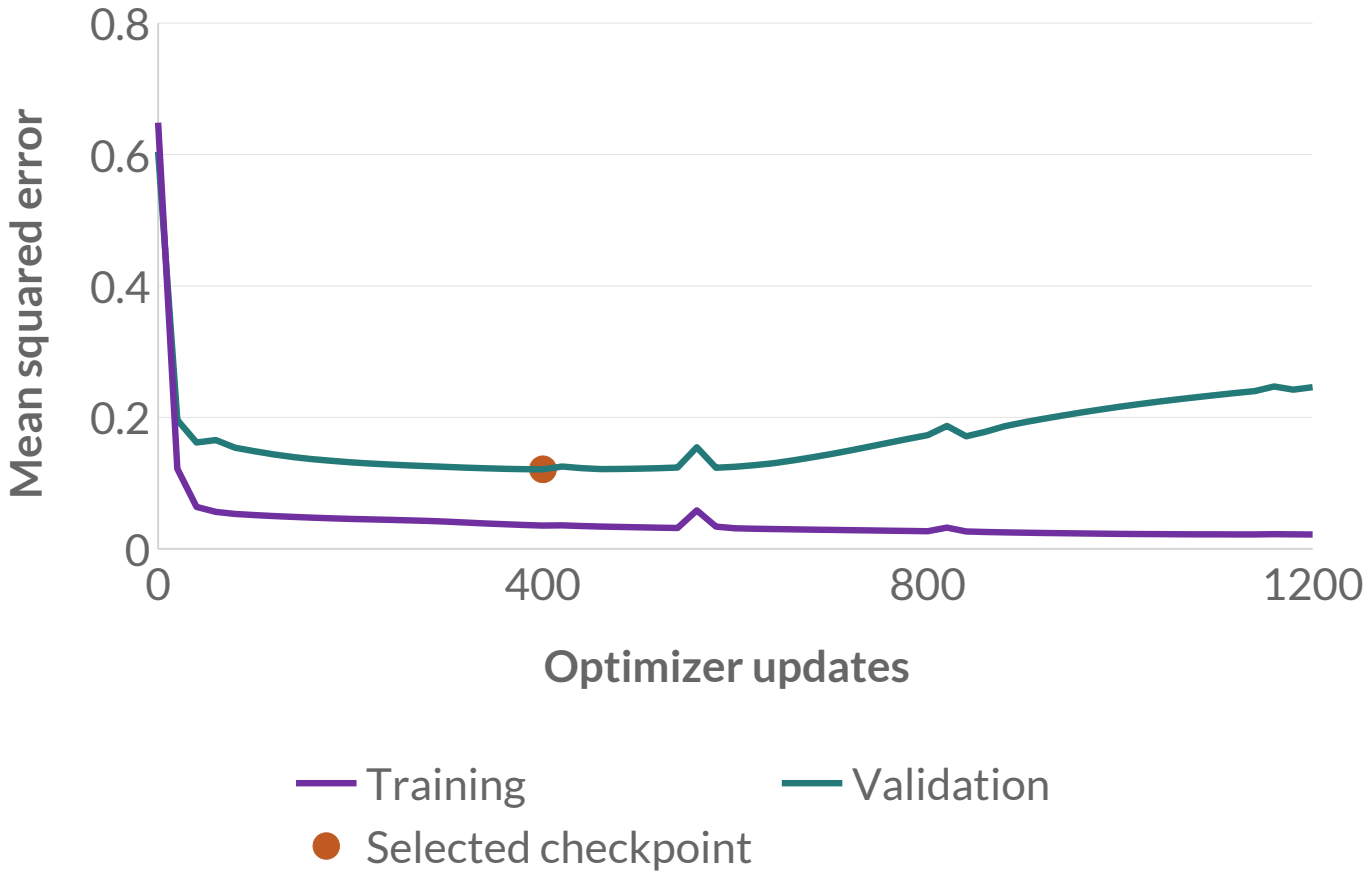
03 Soft targets and distillation

04 Training and evaluation

05 Memory and GPU parallelism

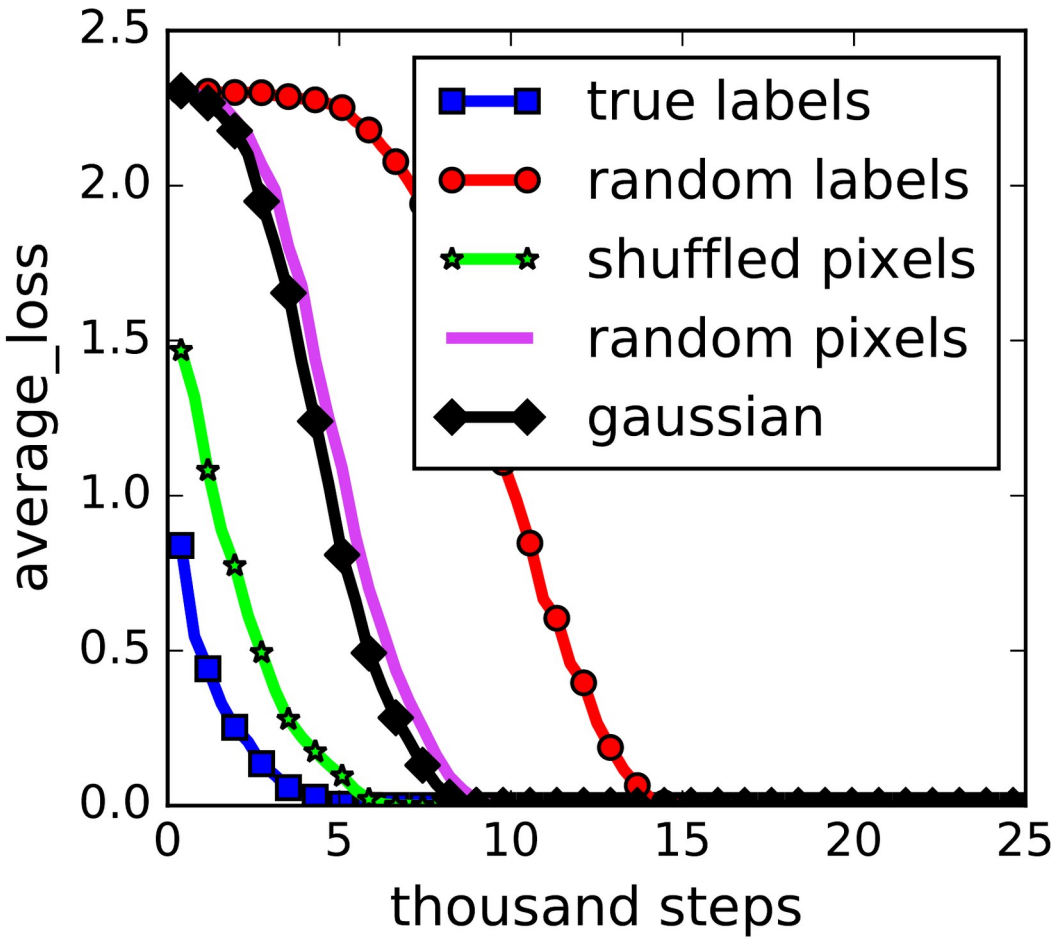
* Optional

Training fit and generalization



- Training observations supply gradients.
- Validation observations are held out from updates.

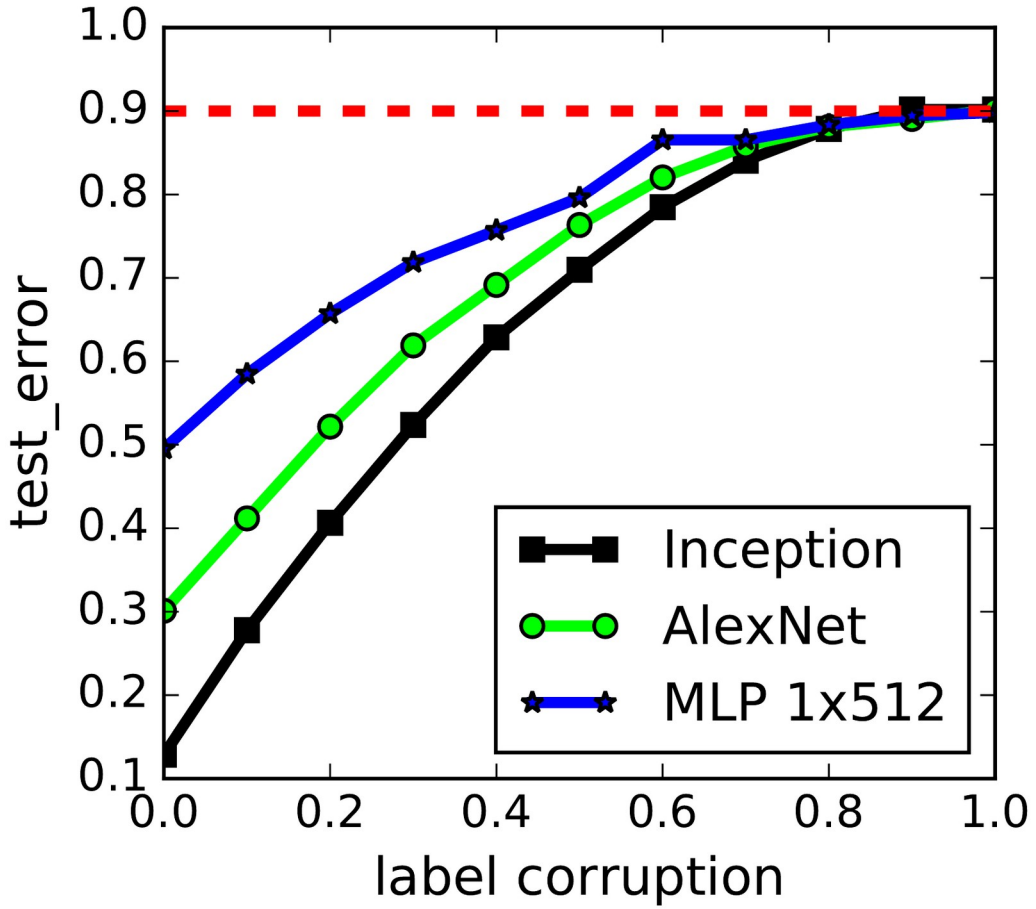
Fitting random labels



- Replace training labels with fixed random labels.
- The network can still drive training loss close to zero.

Source: Rethinking Generalization

Label noise and generalization

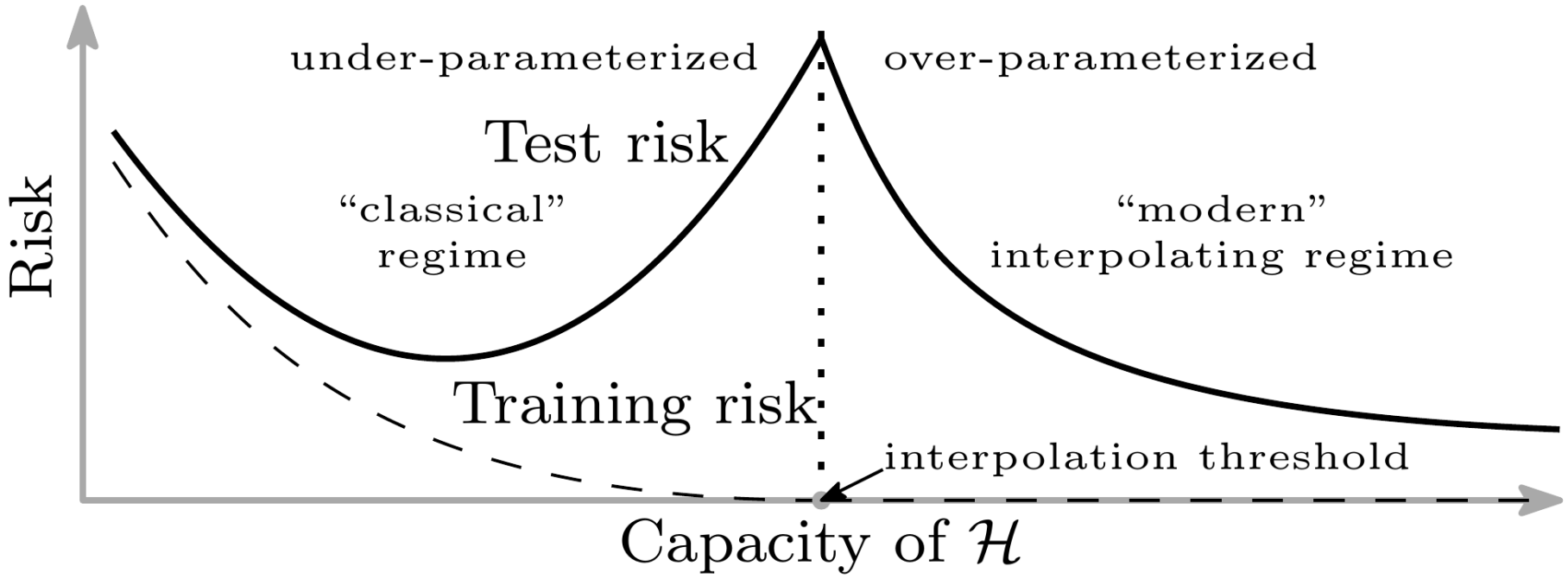


- Training error reaches zero across these noise levels.
- With fully random labels, test error approaches chance.

Source: Rethinking Generalization

Double descent

Test error can fall again after the model becomes large enough to fit all training data.

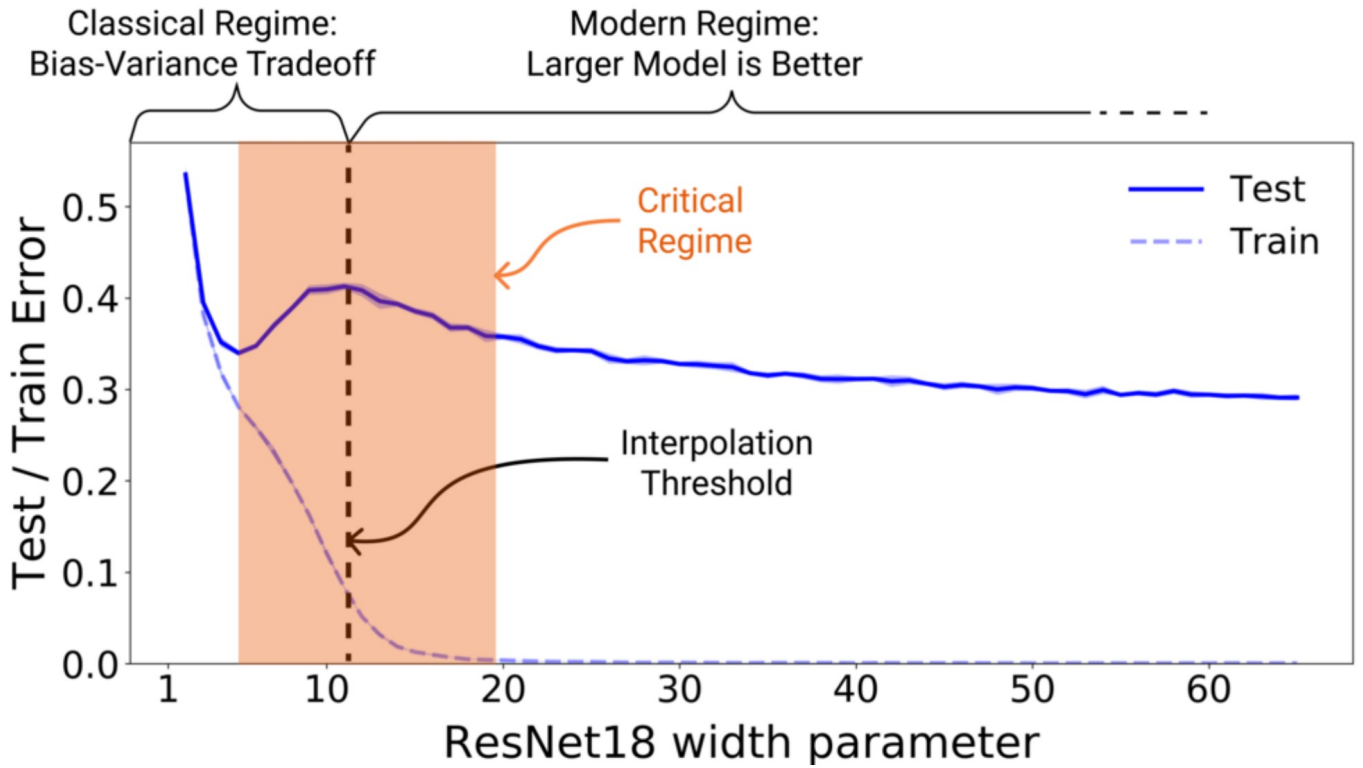


Interpolation threshold: the model can first fit every training observation.

Source: Belkin et al. (2019)

Double descent in neural networks

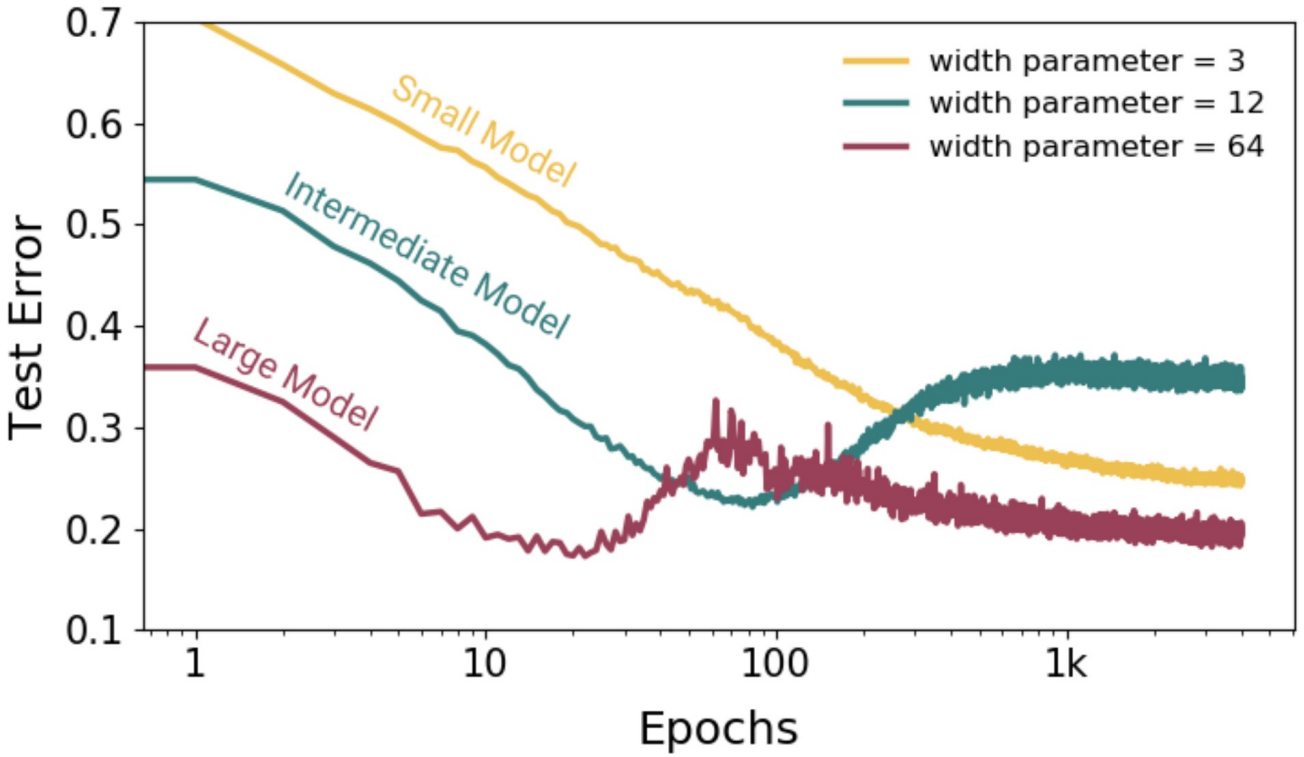
Image classification (CIFAR-10) with 15% label noise.
Vary network width under a fixed training budget.



Source: Deep Double Descent

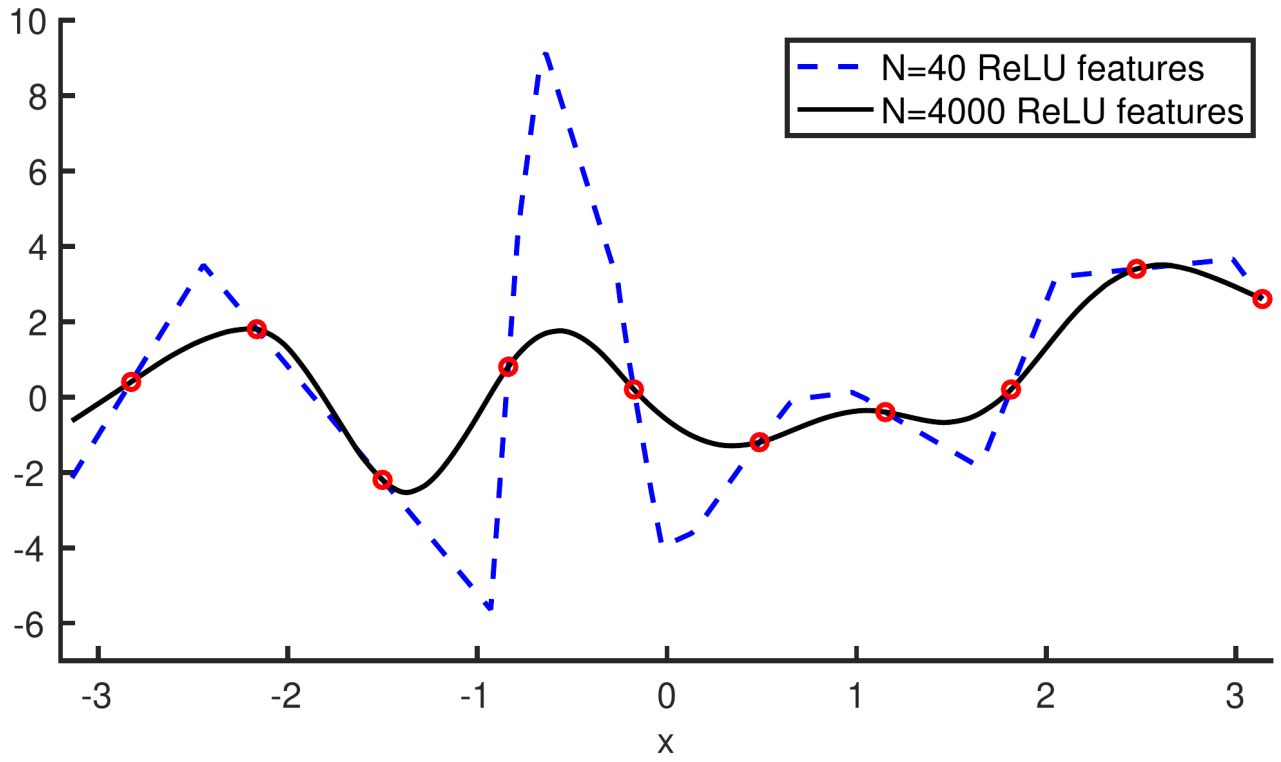
Epoch-wise double descent

CIFAR-10 with 20% label noise; three widths of the same model family.



Source: Deep Double Descent

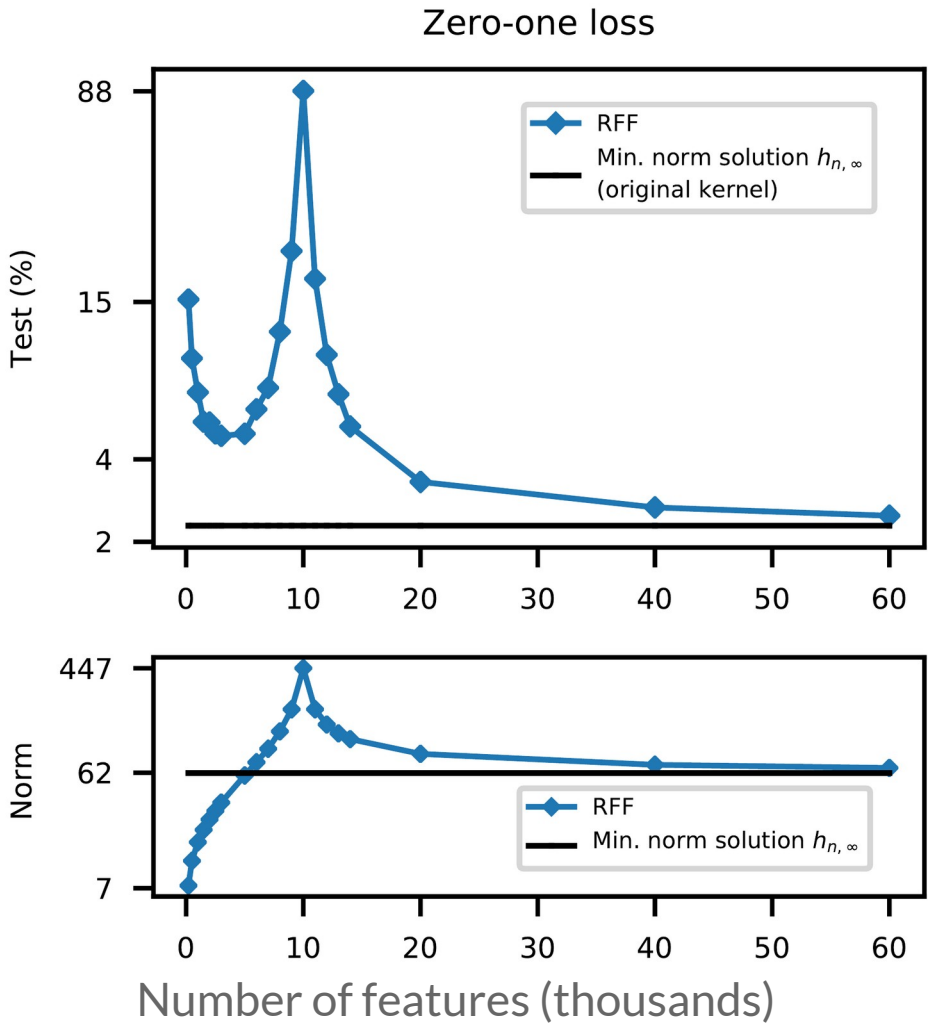
More parameters, simpler fits



- The same ten points lie on both curves.
- More features allow a less erratic fit.
- The fitting rule favors small coefficients. Extra capacity provides more choices.

Source: Belkin et al. (2019)

Fitting near the interpolation threshold



10,000 training examples

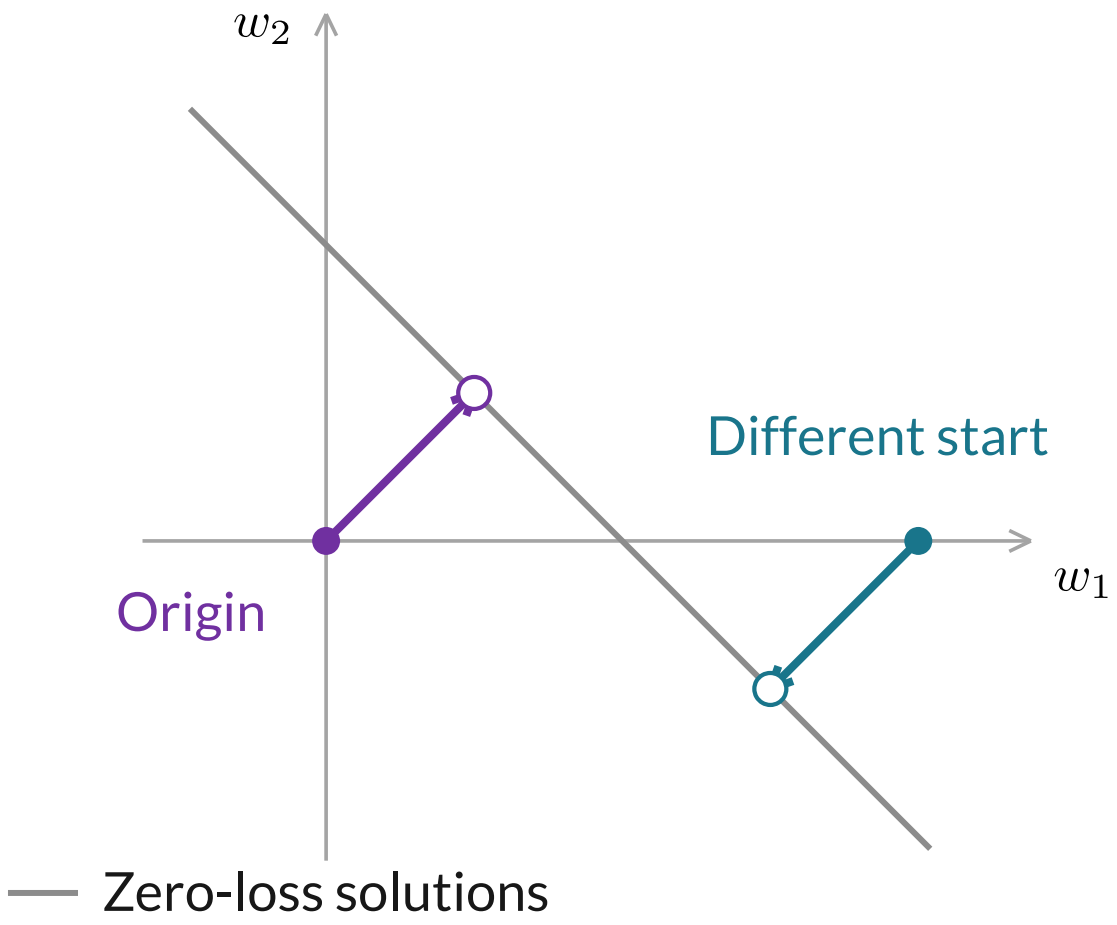
Norm: overall weight magnitude

- Near the threshold, an exact fit can require very large weights.
- Large weights can amplify noise and small changes in the data.
- More features permit a less extreme minimum-norm fit.

RFF: random Fourier features

Source: Belkin et al. (2019)

Implicit regularization



Same training fit

$$w_1 + w_2 = 1$$

$$L(w) = \frac{1}{2}(w_1 + w_2 - 1)^2$$

Even without an explicit penalty, initialization and gradient descent influence which solution is learned.

Regularization and Occam's razor

Fit to observed examples

$$\widehat{R}(\theta) = \frac{1}{N} \sum_i \ell(f_\theta(x_i), y_i)$$

Fit + penalty (e.g. L2 regularization)

$$\widehat{R}(\theta) + \lambda \Omega(\theta)$$

- Many models can fit a finite dataset.
- A preference for simpler or more stable solutions can guide learning.

Lecture outline

01 Generalization

02 Dropout and input noise

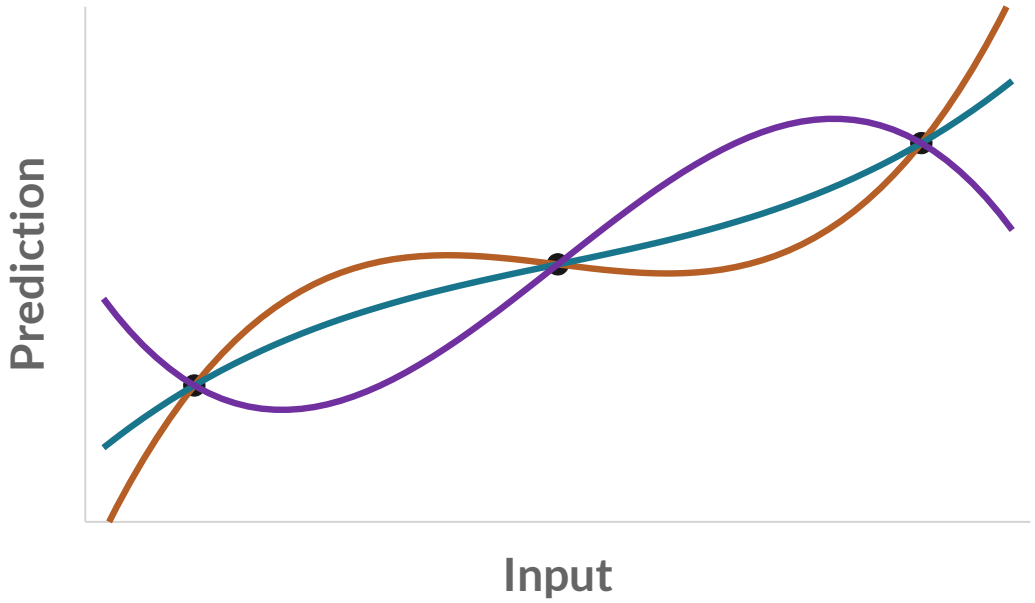
03 Soft targets and distillation

04 Training and evaluation

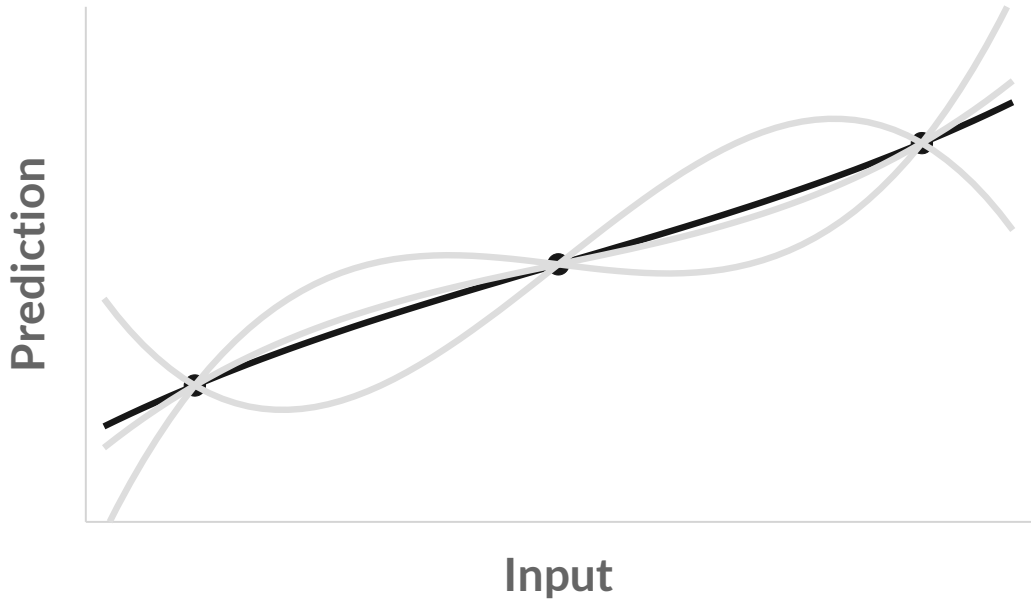
05 Memory and GPU parallelism

Ensemble averaging

Individual predictors



Average prediction

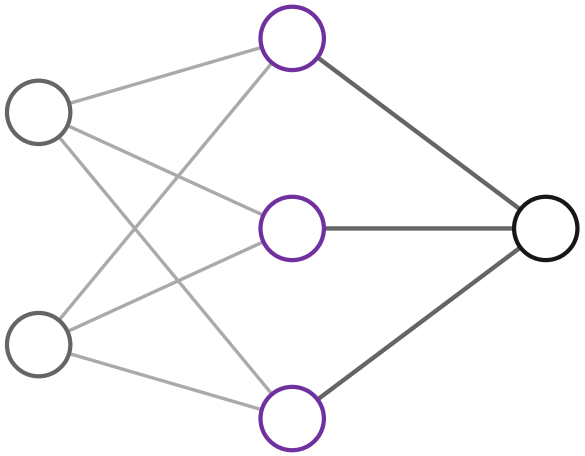


$$\overline{f}(x) = \frac{1}{M}\sum_{m=1}^{M} f_m(x)$$

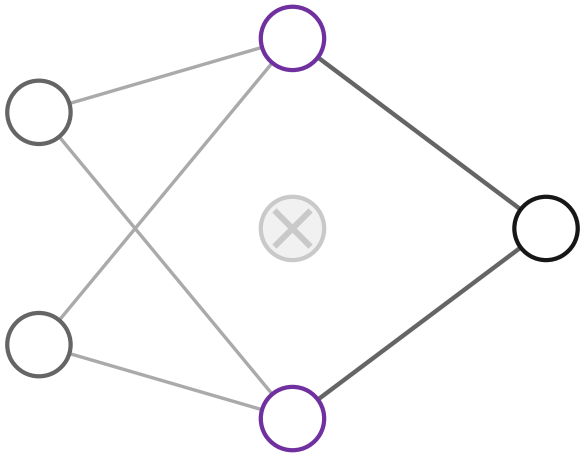
Dropout masks

Randomly omit activations to reduce reliance on a few hidden units.

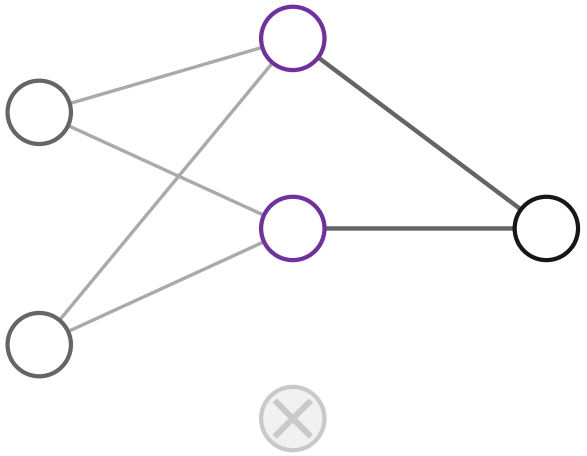
Full computation



Mask A



Mask B



A new mask changes the active computation, not the stored weights.

Inverted dropout

Retention probability

$$q = 1 - p_{\text{drop}}$$

Mask: 1 with probability q, else 0

$$r_j \sim \text{Bernoulli}(q)$$

Training activation

$$\widetilde{h}_j = \frac{r_j h_j}{q}$$

Retained

$$q: \quad h_j/q$$

Dropped

$$1-q: \quad 0$$

$$\mathbb{E}[\widetilde{h}_j \mid h_j] = q\frac{h_j}{q} + (1-q)\,0 = h_j$$

Dropout at evaluation

Training uses random masks and inverted scaling.

Evaluation uses every activation without that scaling.

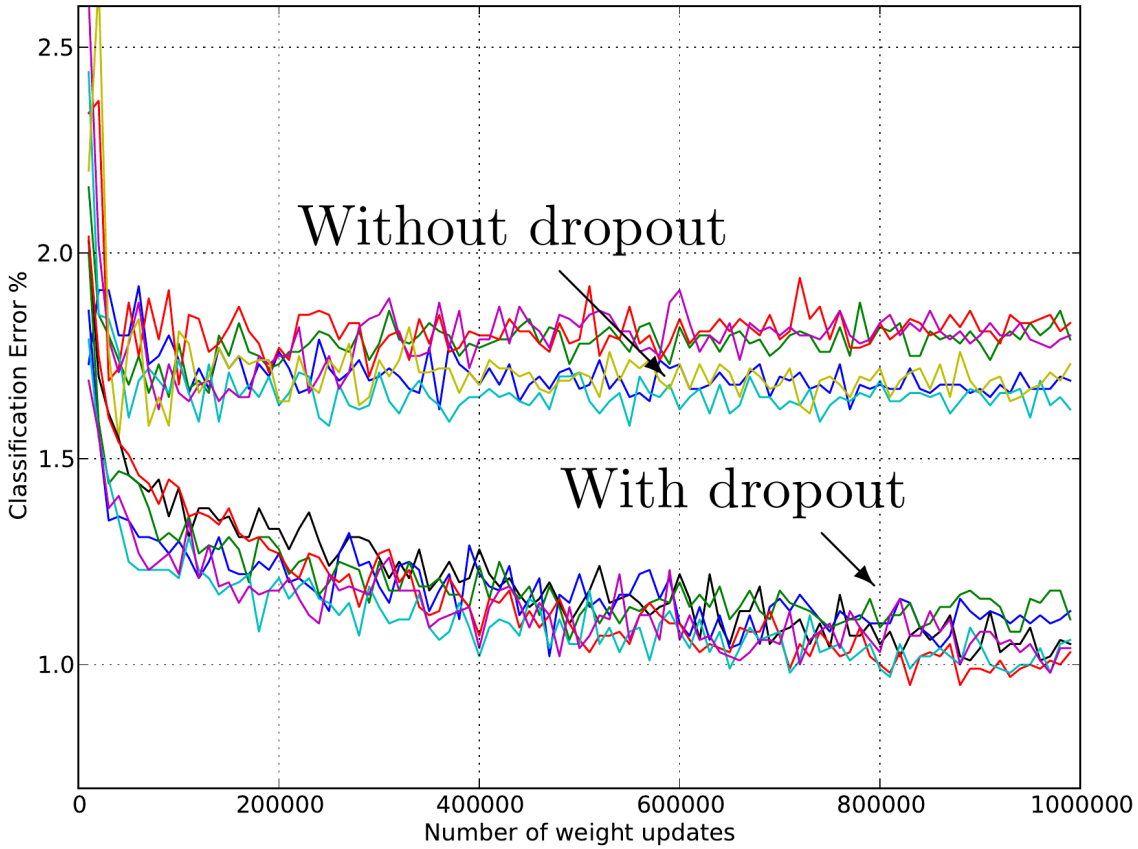
$$\text{training: } \widetilde{h} = r \odot h/q$$

$$\text{evaluation: } \widetilde{h} = h$$

Dropout in PyTorch

```
drop = nn.Dropout(p=0.2)  # probability of dropping

drop.train()              # enable random masks
y_train = drop(h)         # retained values scaled automatically

drop.eval()               # ordinary evaluation
y_eval = drop(h)          # unchanged activations
```

Dropout on MNIST

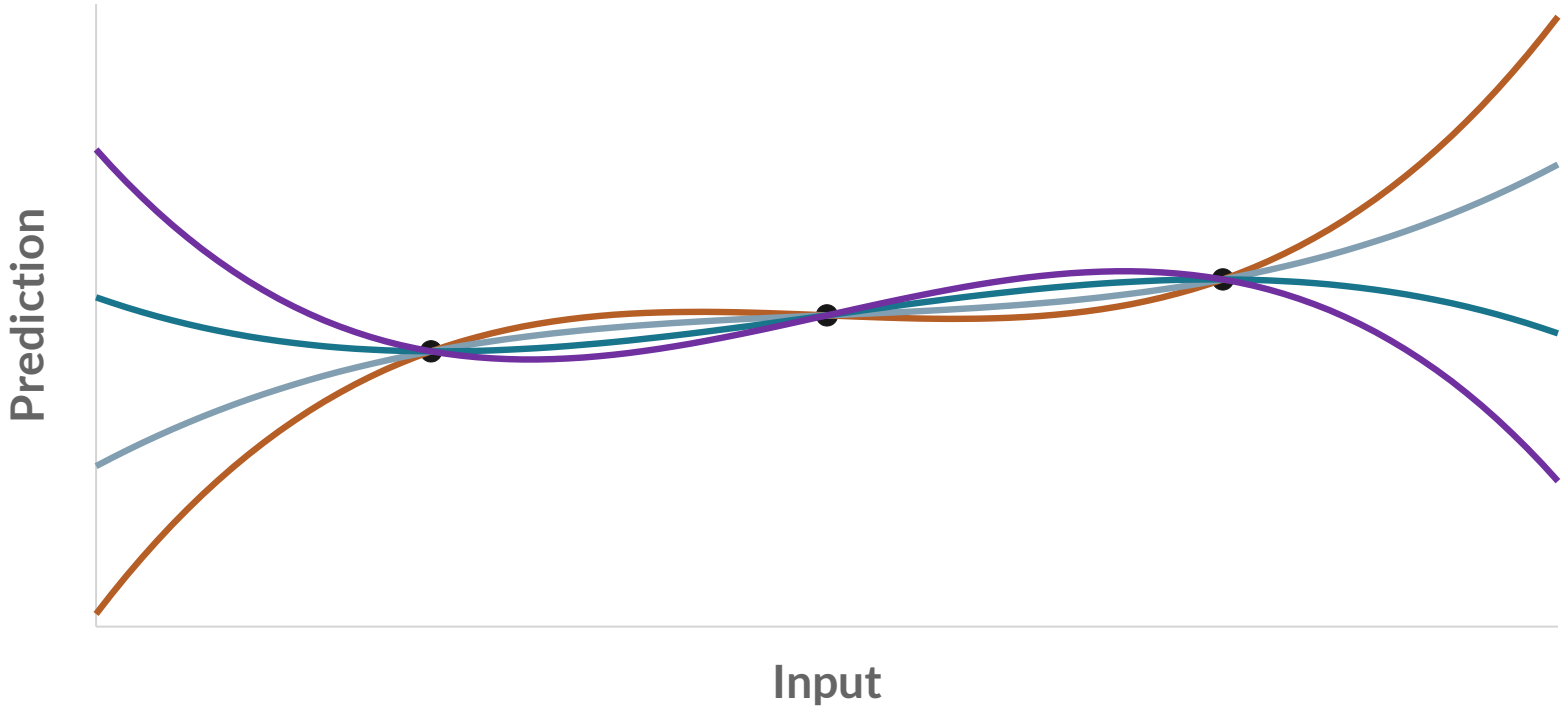


- Fully connected networks with and without dropout.
- The dropout runs achieve lower test error in this comparison.

Source: Dropout (2014)

Bayesian intuition for dropout

Several predictors can agree on the observed data.



Observed points

- Same fit

New inputs

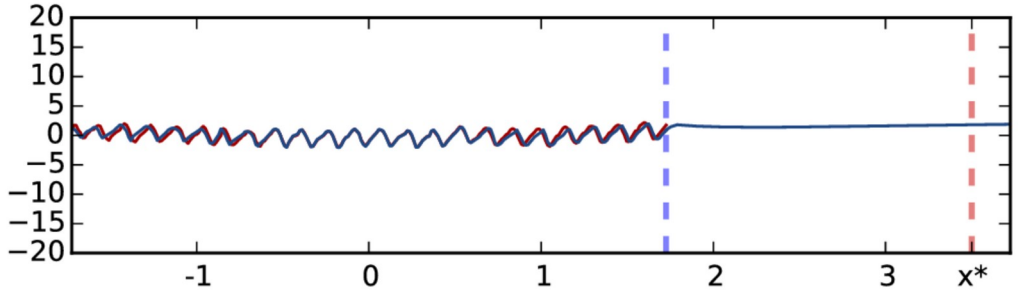
Predictions can disagree.

Illustration of predictor uncertainty

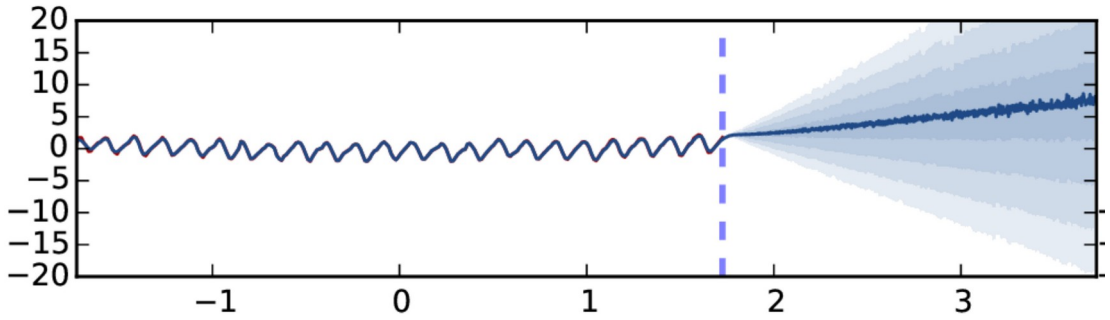
Predictive uncertainty with dropout

Repeat prediction with different masks: Monte Carlo (MC) dropout.

Ordinary dropout prediction



Monte Carlo dropout



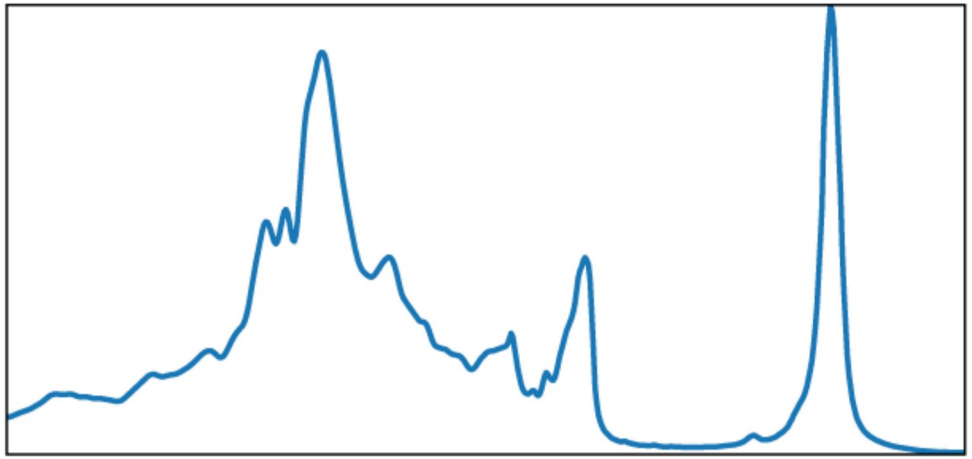
Time (normalized)

- Training data end at the blue dashed line.
- A wider blue band means more uncertainty.

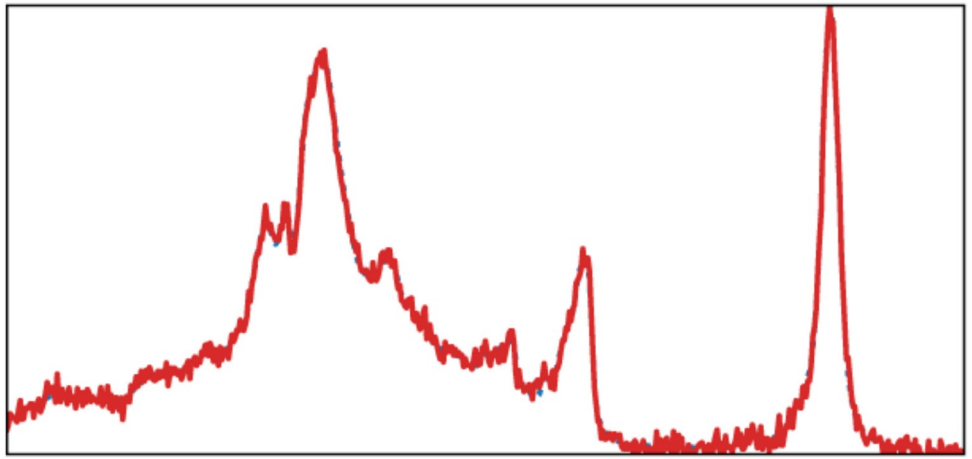
Source: Gal & Ghahramani (2016)

Adding input noise

Original measurement



Noisy input, same target



$$\widetilde{x} = x + \epsilon, \qquad \epsilon \sim \mathcal{N}(0, \sigma^2 I)$$

$$\mathbb{E}_\epsilon[\ell(f_\theta(x + \epsilon), y)]$$

Source: Time Series Data Augmentation

Augmentation depends on the task

Same speech recording · lower volume

Recognize the spoken word

The word stays the same.

Estimate sound level

The target must change.

Mixup

Knowledge extension

$$\widetilde{x} = \lambda x_i + (1 - \lambda) x_j, \qquad 0 \leq \lambda \leq 1$$

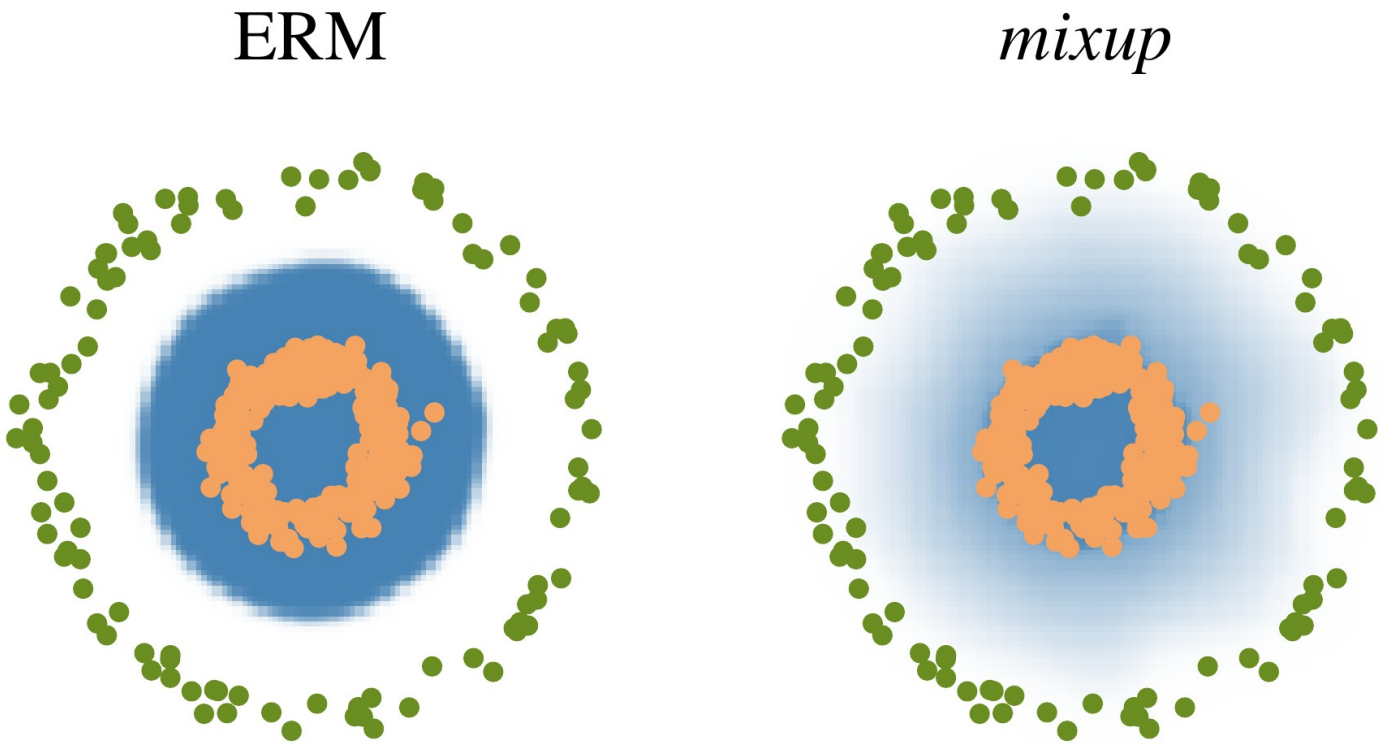
$$\widetilde{y} = \lambda y_i + (1 - \lambda) y_j$$

(x_i, y_i) $(\widetilde{x}, \widetilde{y})$ (x_j, y_j)

Use the same mixing coefficient for the input and its target vector.

Predictions between training examples

Illustrative two-dimensional classification problem: blue shading shows predicted class probability.



Source: mixup

Mixup on tabular data

Classification error (%); the same two-hidden-layer MLP in both methods.

Dataset	ERM	*mixup*
Abalone	74.0	73.6
Arcene	57.6	**48.0**
Arrhythmia	56.6	**46.3**

Dataset	ERM	*mixup*
Htru2	2.0	2.0
Iris	21.3	**17.3**
Phishing	16.3	15.2

Source: mixup

Lecture outline

01 Generalization

02 Dropout and input noise

03 Soft targets and distillation

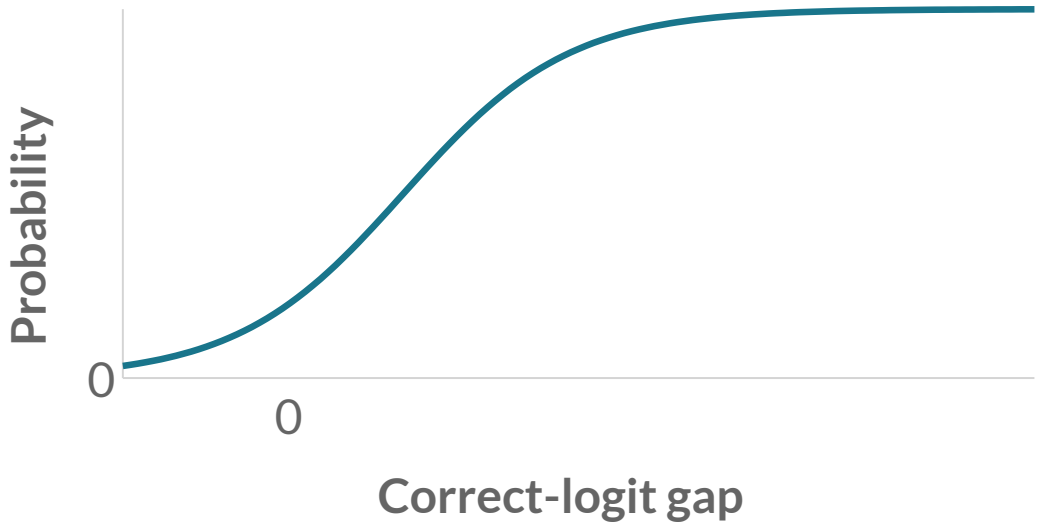
04 Training and evaluation

05 Memory and GPU parallelism

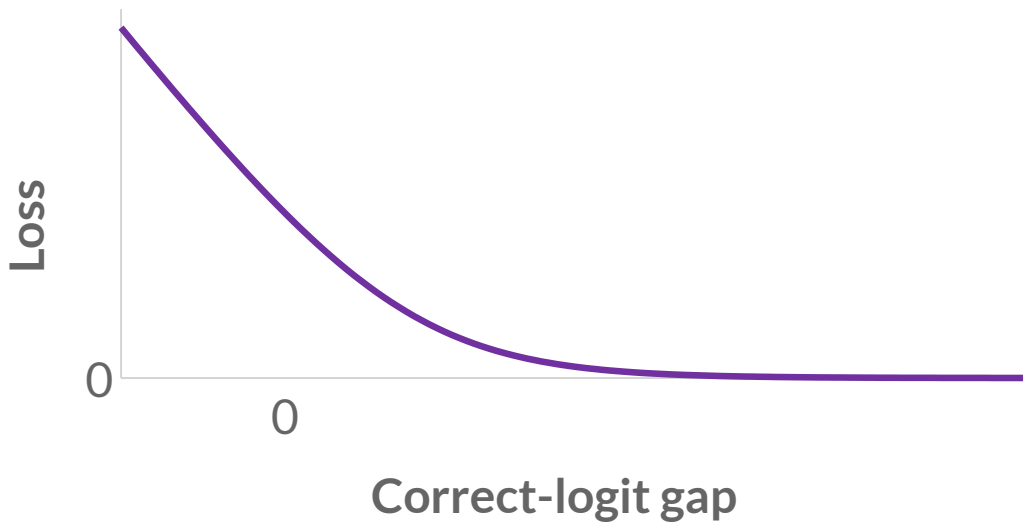
Cross-entropy and confidence

With a one-hot target, the loss falls as the correct-class probability rises.

Correct-class probability



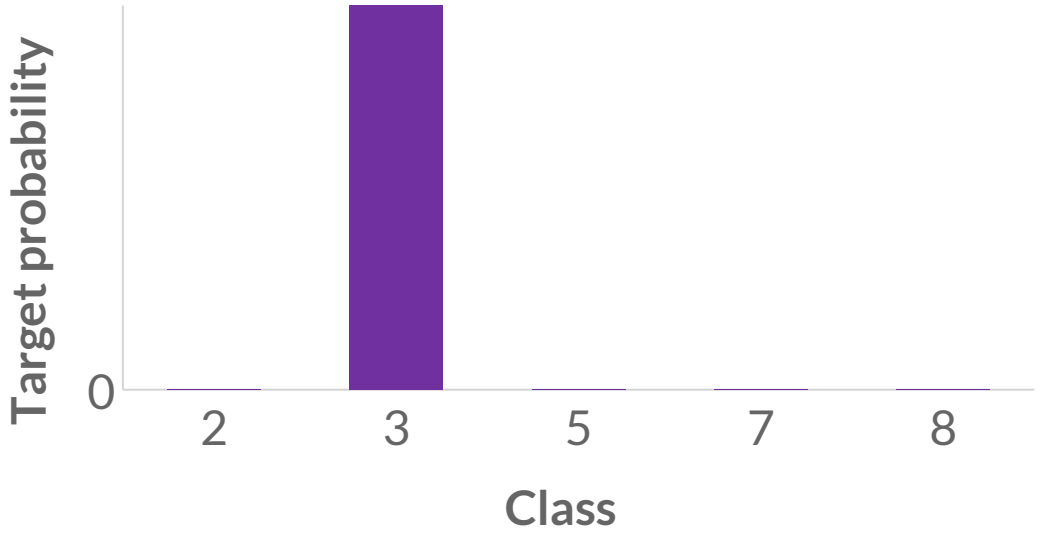
Cross-entropy loss



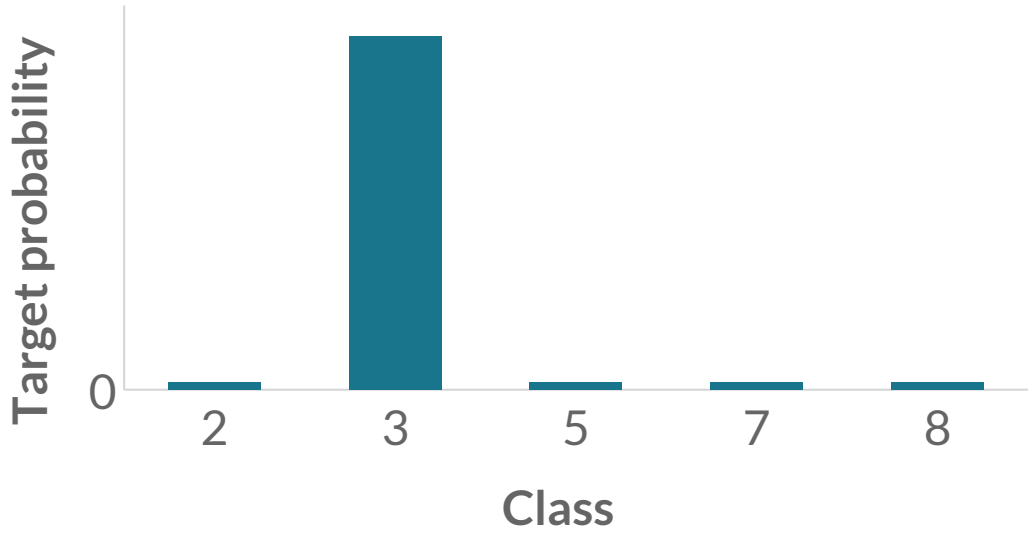
$$\mathcal{L}_{\mathrm{CE}} = -\log p_y$$

Label smoothing

One-hot target



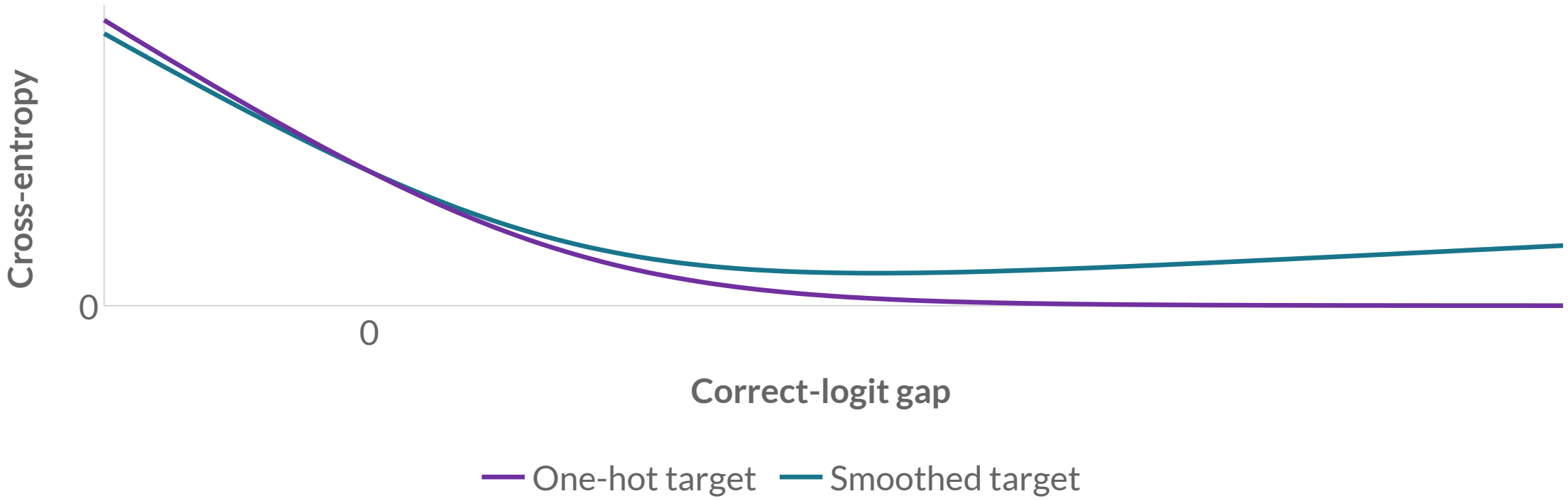
Smoothed target



$$q_k = (1-\varepsilon)\mathbf{1}[k=y] + \varepsilon/K$$

Spread a fraction ε uniformly across all K classes.
The indicator selects the correct class: 1 if k = y, otherwise 0.

Label smoothing and confidence



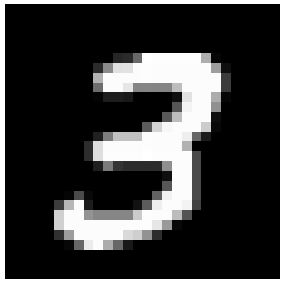
$$\frac{\partial \mathcal{L}}{\partial z_k} = p_k - q_k$$

Above target → lower the logit
Below target → raise the logit

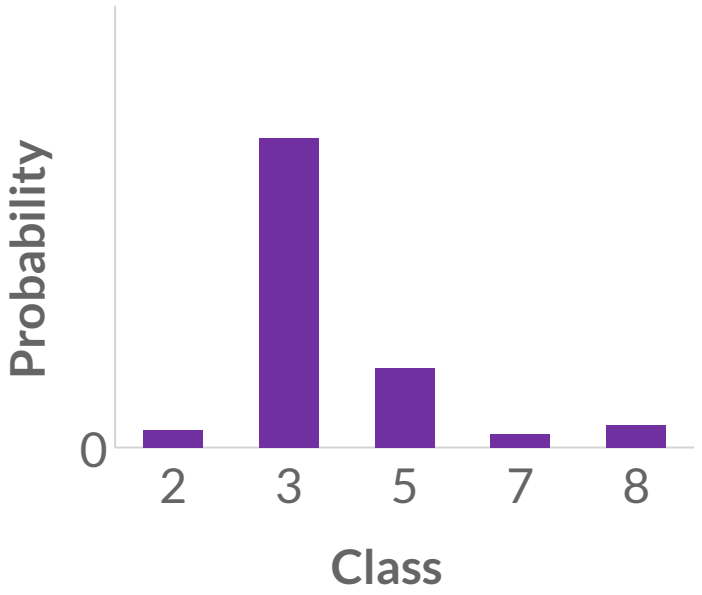
Teacher targets

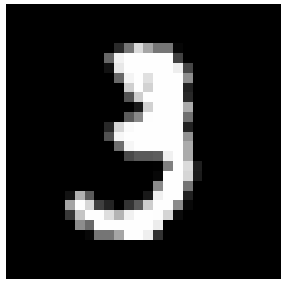
The teacher is a trained model that supplies soft targets.

Illustration: unlike label smoothing, targets depend on the input.

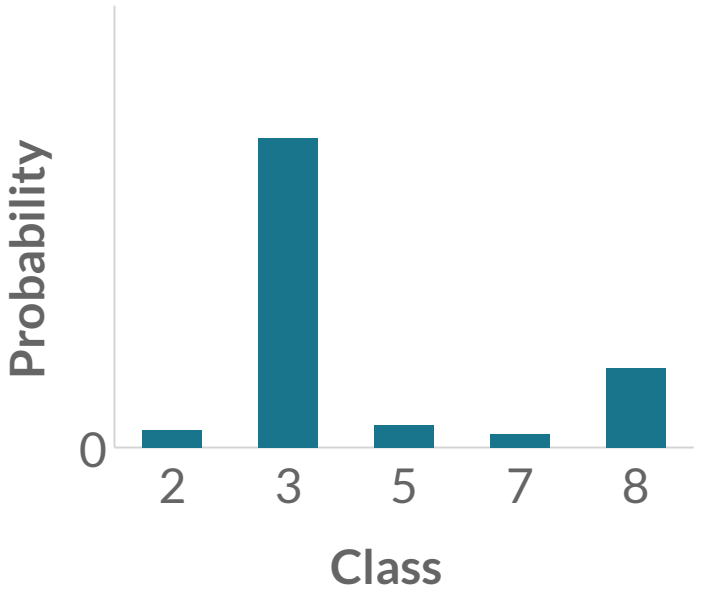


Label: 3





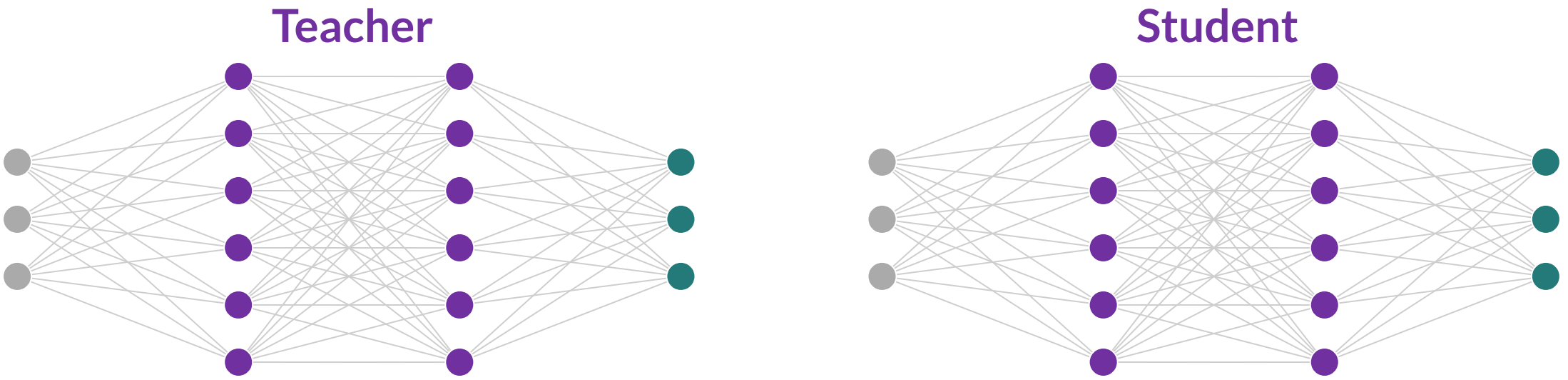
Label: 3



Source: MNIST

Knowledge distillation

Train using soft targets obtained from a trained model's logits.



Frozen parameters

Updated by backpropagation

Here: the same architecture, with a newly initialized student.

Teacher targets can help the student generalize.

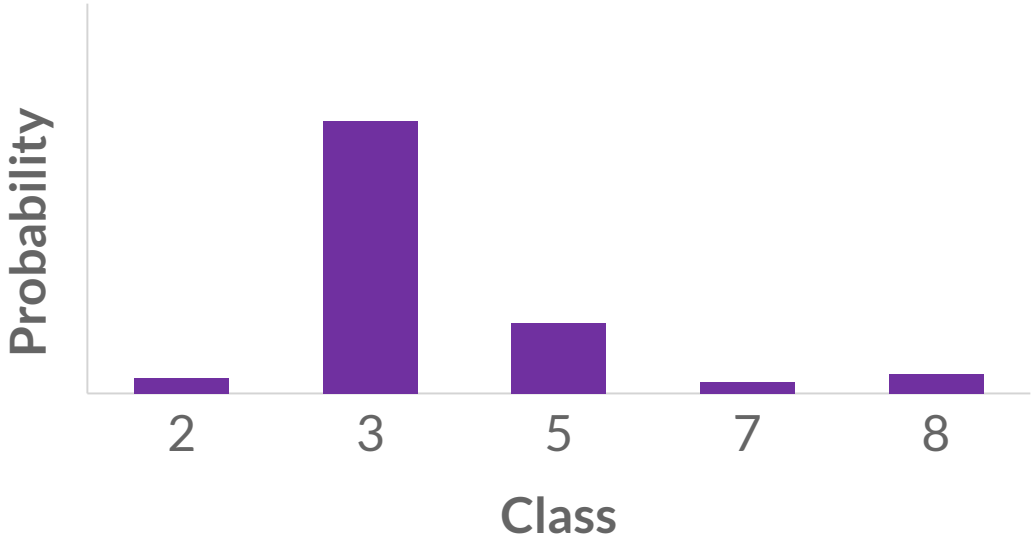
Temperature

$$p_k^{(T)} = \frac{\exp(z_k/T)}{\sum_j \exp(z_j/T)}$$

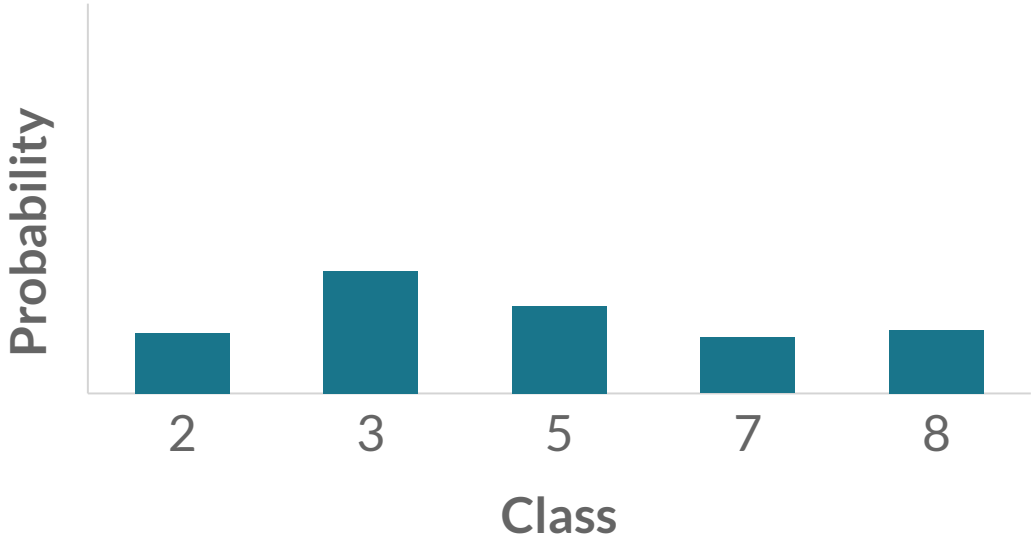
Larger temperature: softer targets

The class order stays the same.

Same illustrative logits $T = 1$



$T = 4$



Recall: cross-entropy and KL divergence

q Target probabilities

p Student probabilities

Cross-entropy (CE)

Penalizes low probability on target classes.

$$\mathrm{CE}(q, p) = -\sum_k q_k \log p_k$$

KL divergence

Measures the mismatch between two distributions.

$$D_{\mathrm{KL}}(q \| p) = \sum_k q_k \log \frac{q_k}{p_k}$$

Fixed teacher targets: CE and KL give the same student gradients.

The distillation objective

Observed labels

$$\mathcal{L}_{\text{hard}} = \text{CE}(y, p^{(1)}_{\text{student}})$$

Frozen teacher targets

$$\mathcal{L}_{\text{soft}} = T^2 D_{\text{KL}}\left(p^{(T)}_{\text{teacher}} \| p^{(T)}_{\text{student}}\right)$$

$$\mathcal{L} = (1-\alpha)\mathcal{L}_{\text{hard}} + \alpha\mathcal{L}_{\text{soft}}$$

α : teacher-target weight

T^2: keeps gradient magnitudes comparable across temperatures

Distillation and generalization

CIFAR-100 test error (%). Same architecture for teacher and student.

Network	Teacher	BAN	BAN+L
DenseNet-112-33	18.25	**16.95**	17.68
DenseNet-90-60	17.69	**16.69**	16.93
DenseNet-80-80	17.16	**16.36**	16.5
DenseNet-80-120	16.87	**16.00**	16.41

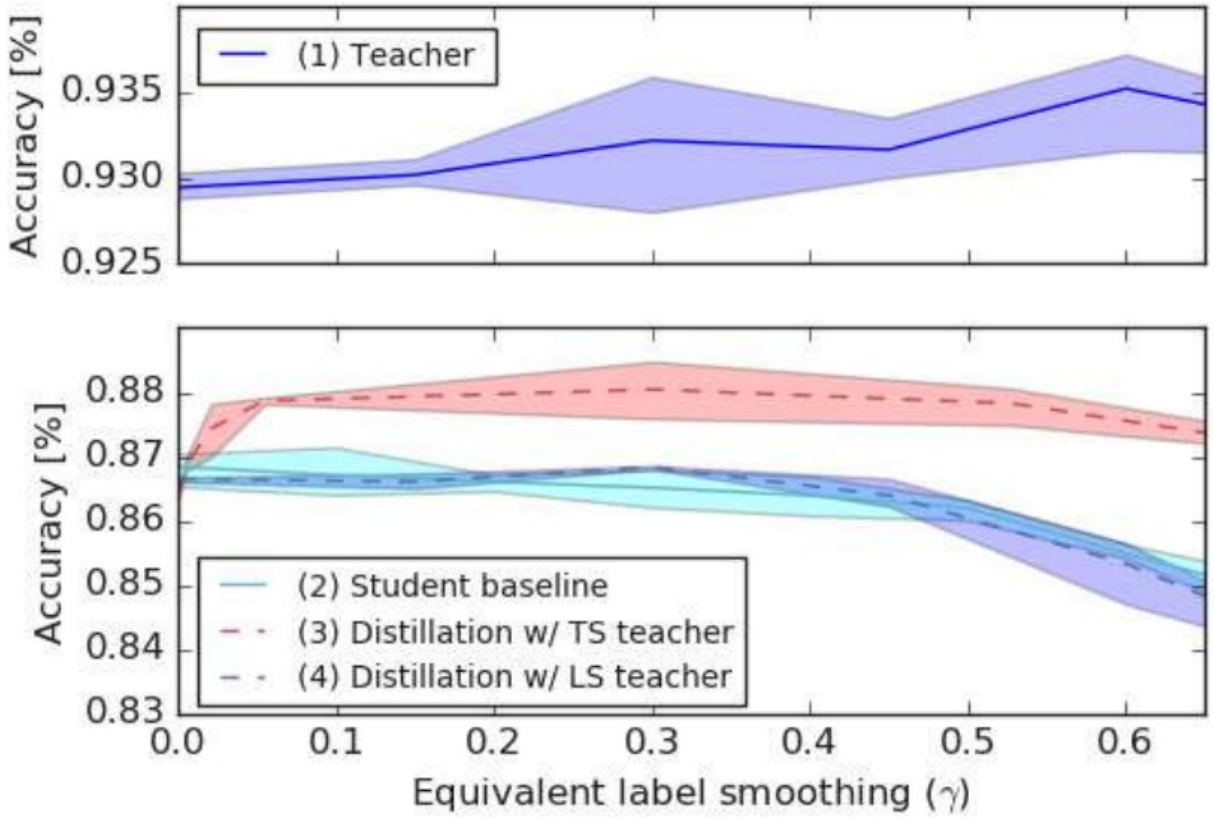
BAN: student trained on teacher targets alone.

BAN+L: teacher targets combined with the original labels.

Source: Born Again Neural Networks

Label smoothing and distillation

Image classification · teacher and student use different networks



Teacher label smoothing

Improved teacher accuracy in this experiment.

But its targets produced worse distilled students.

When Does Label Smoothing Help?

Distillation in PyTorch

The optimizer updates only student.parameters().

```
teacher.eval()                                  # fixed teacher behavior
student.train()
with torch.no_grad():
    q = F.softmax(teacher(x) / T, dim=1)  # fixed targets
z = student(x)                                  # student gradient graph
log_p = F.log_softmax(z / T, dim=1)
soft = F.kl_div(log_p, q, reduction="batchmean")
hard = F.cross_entropy(z, y)                    # ordinary logits
loss = (1 - alpha) * hard + alpha * T**2 * soft
optimizer.zero_grad()                           # student optimizer
loss.backward()
optimizer.step()
```

Lecture outline

01 Generalization

02 Dropout and input noise

03 Soft targets and distillation

04 Training and evaluation

05 Memory and GPU parallelism

Comparable training and validation loss

Training log

Dropout active
Soft targets or teacher supervision

Evaluation measurement

Dropout inactive
Original task labels

For a direct comparison: same model behavior, same loss.

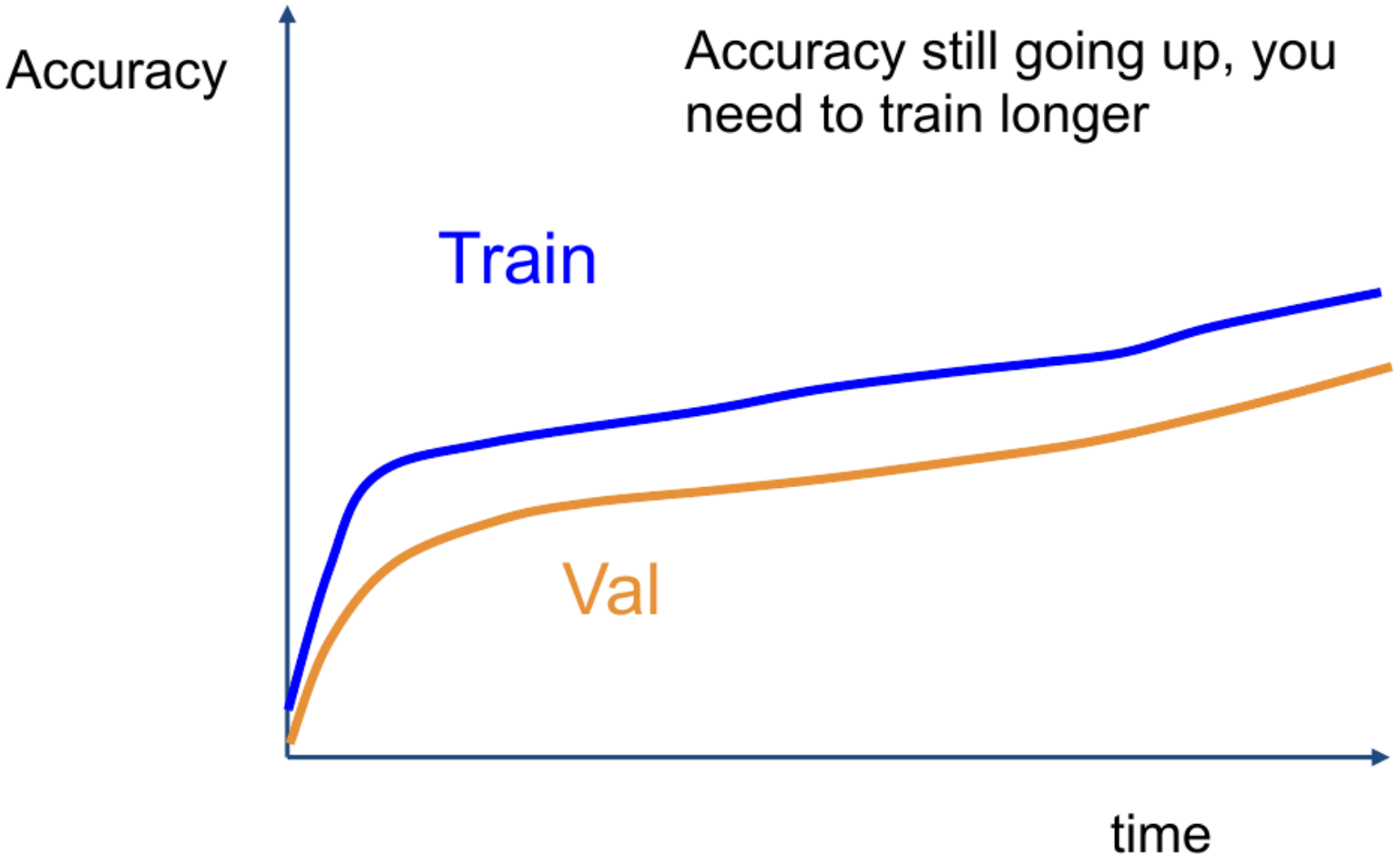
$$\frac{1}{N_{\text{train}}} \sum_{i \in \text{train}} \ell(f_\theta(x_i), y_i)$$

Training examples

$$\frac{1}{N_{\text{val}}} \sum_{i \in \text{val}} \ell(f_\theta(x_i), y_i)$$

Validation examples

Learning curves during training

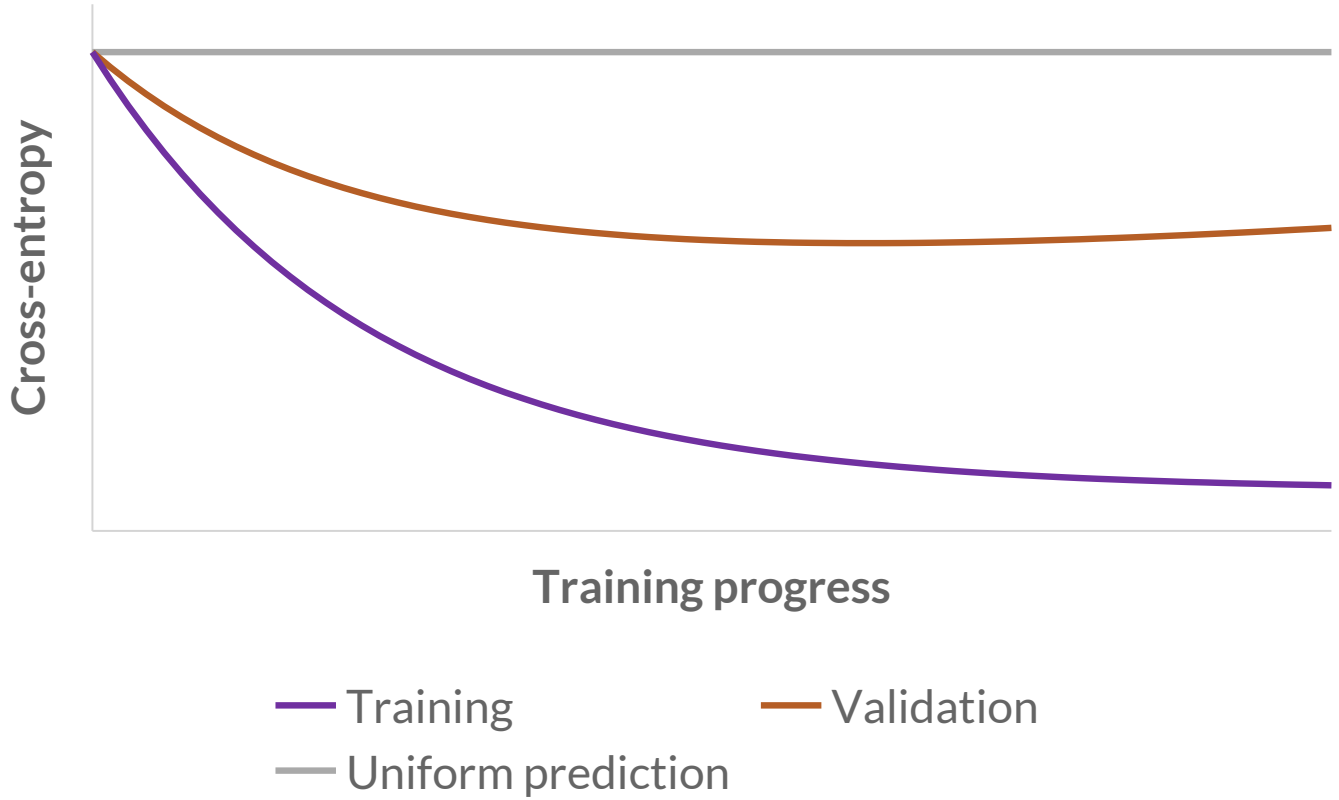


Continued improvement

Validation gains can justify a longer run.

A small gap alone does not imply underfitting.

Training–validation gap



Overfitting

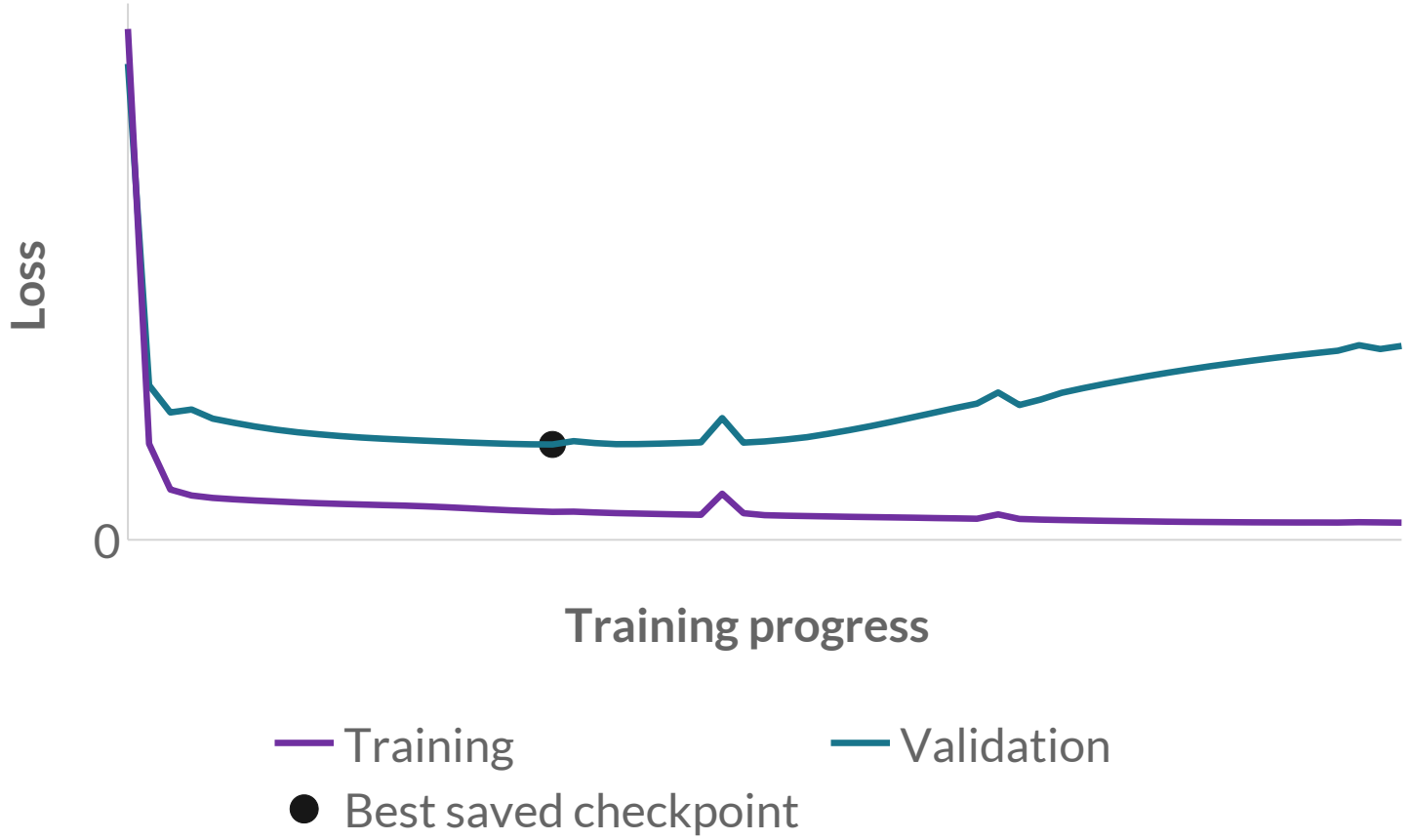
Training loss falls while validation loss rises.

Next comparison

Tune regularization or add training data.

Adapted from CS 7643

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.

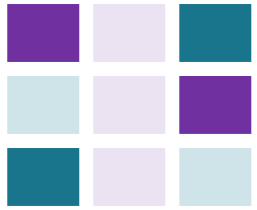
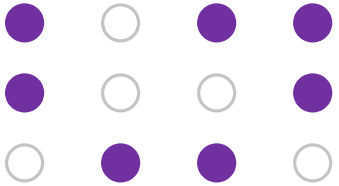
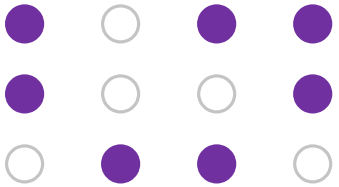
Fixed budget

Use a preset number of steps or a fixed time budget.

Random seeds

A seed fixes the starting point of a random-number stream.

Replay the same code and sequence of random operations.

	Initial weights	Data order	Dropout masks
First run `seed = s`		C A D B	
Replay `seed = s`		C A D B	

Set the seed once before the run, not before every batch.

Fixing random seeds

```
import random
import numpy as np
import torch
from torch.utils.data import DataLoader

seed = 42
random.seed(seed)          # Python randomness
np.random.seed(seed)       # NumPy randomness
torch.manual_seed(seed)    # PyTorch: CPU and CUDA

model = MLP()              # initialize after seeding
loader = DataLoader(
    train_set, batch_size=batch_size,
    shuffle=True, num_workers=0,
)
```

Some operations can remain nondeterministic; the environment also matters.

Lecture outline

01 Generalization

02 Dropout and input noise

03 Soft targets and distillation

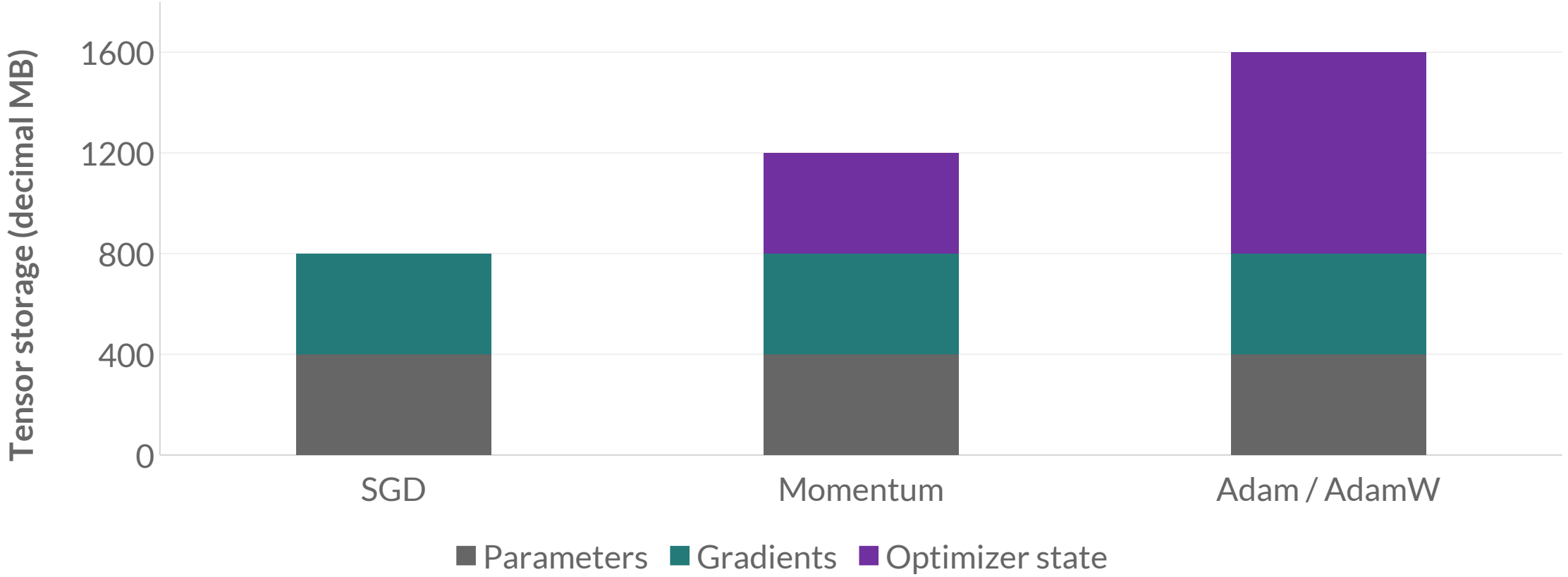
04 Training and evaluation

05 Memory and GPU parallelism

Optimizer-state memory

100 million parameters

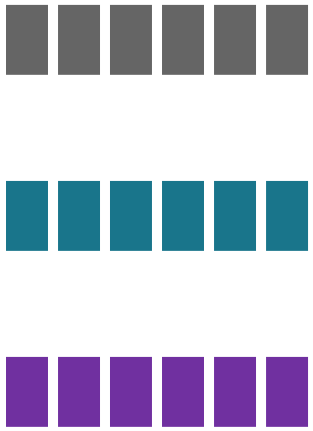
Assume 4 bytes per value (32-bit storage).



Model state and saved activations

Model state

Parameter-sized arrays



Parameters

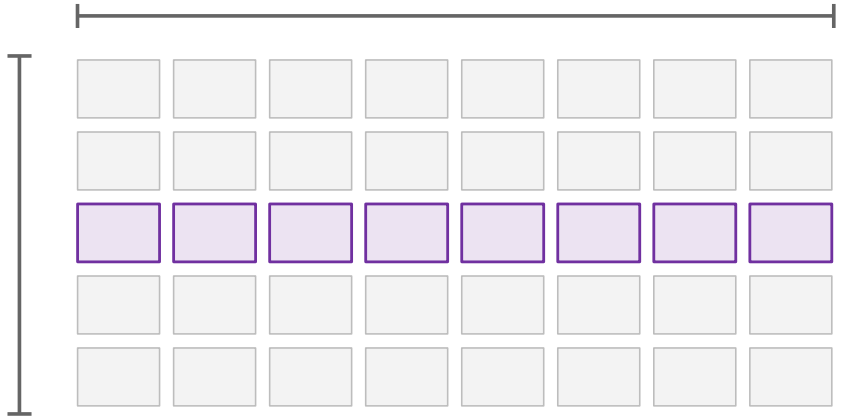
Gradients

Optimizer history

Scales with model size

Saved activations

Values from the current batch



Depends on batch and layer sizes

Forward values used by backward

Linear layer

$$z = Wh + b$$

$$\frac{\partial \mathcal{L}}{\partial W} = \frac{\partial \mathcal{L}}{\partial z} h^\top$$

Uses the saved input

ReLU

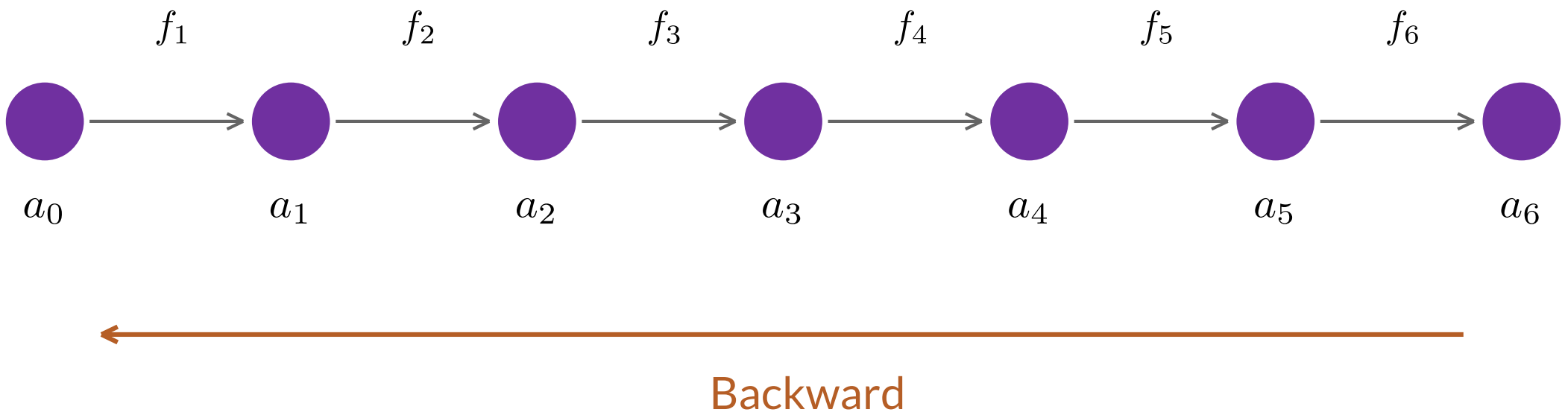
$$h = \mathrm{ReLU}(z)$$

$$\frac{\partial \mathcal{L}}{\partial z} = \frac{\partial \mathcal{L}}{\partial h} \odot \mathbf{1}[z > 0]$$

Indicator: 1 for positive entries,
0 otherwise

Saved activations

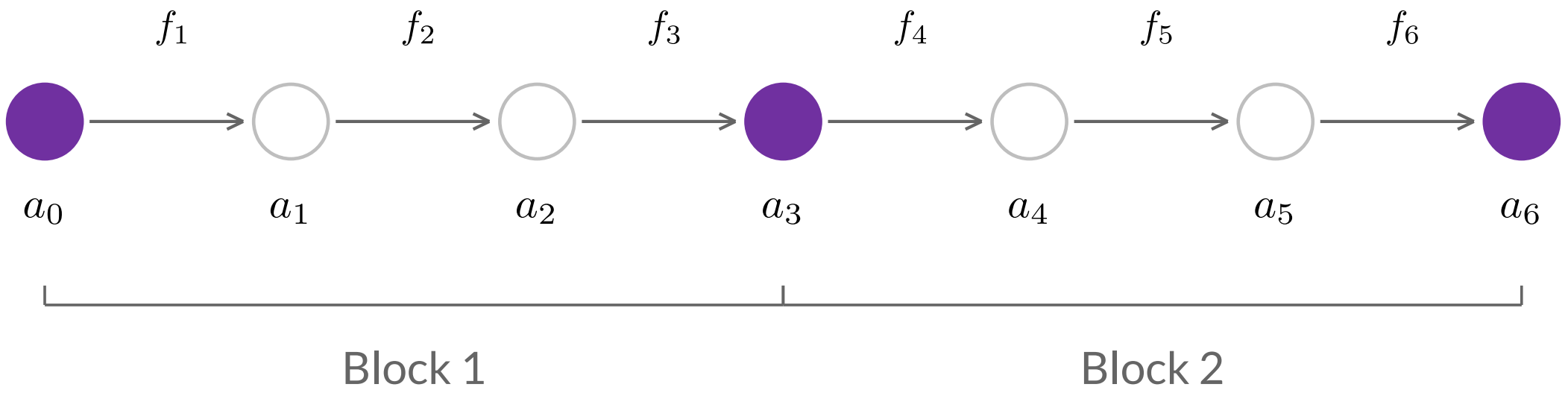
Each circle is an activation array.



- Retained until its backward computation

Activation checkpointing

Keep selected boundaries after the forward pass.

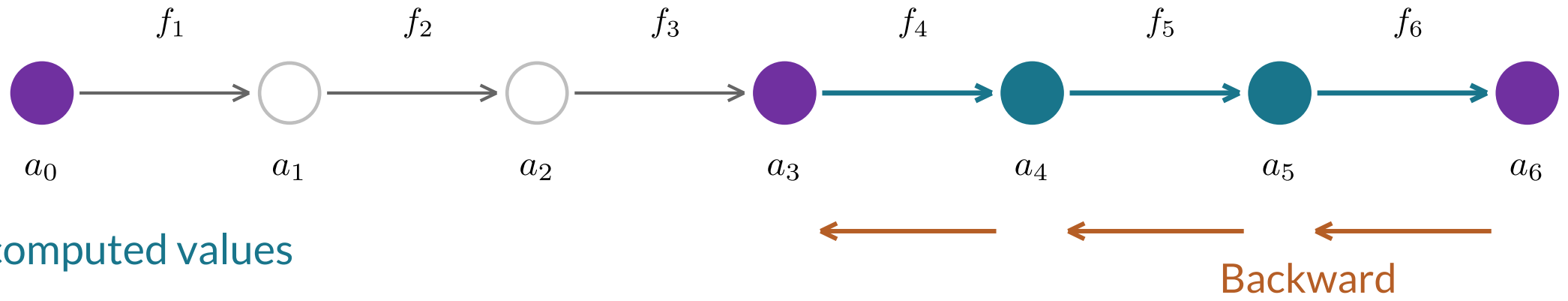


● Retained boundary

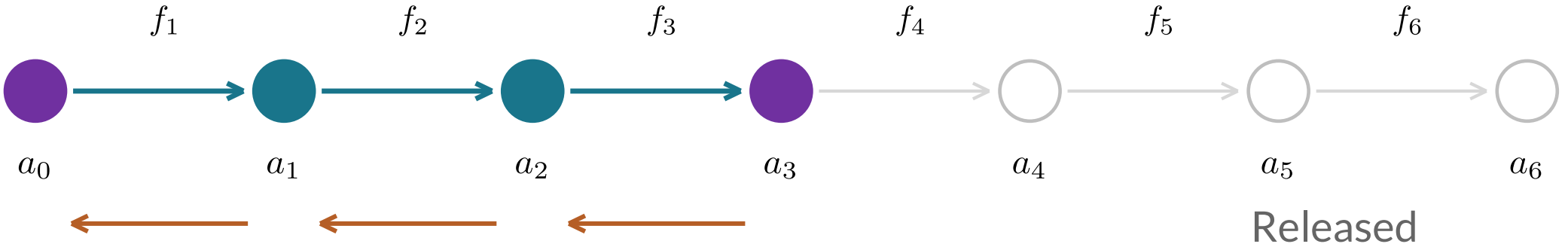
○ Released intermediate

Recomputation and release

1 Reconstruct the last block, then backpropagate



2 Release that block; reconstruct the preceding block



Accumulation and checkpointing

Gradient accumulation

Fewer examples at once

Separate microbatch passes

Activation checkpointing

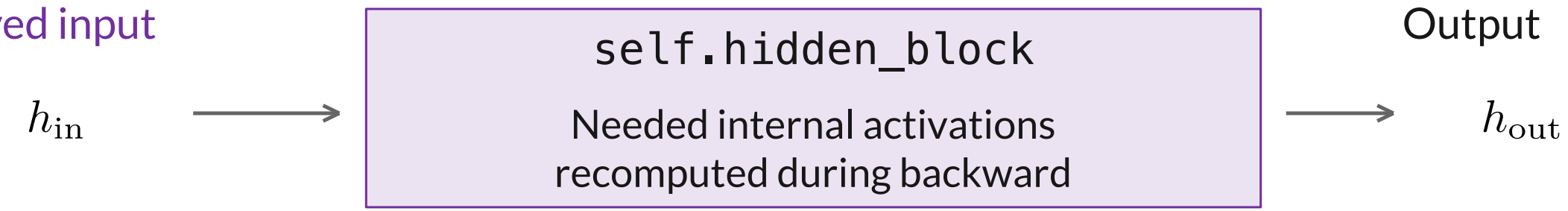
Fewer retained values

Extra forward computation

Both retain the parameters and optimizer history.

Checkpointing in PyTorch

```python
from torch.utils.checkpoint import checkpoint

# Inside the model's forward method:
h_in = self.input_layer(x)
h_out = checkpoint(self.hidden_block, h_in,
                   use_reentrant=False)
output = self.output_layer(h_out)
```



Outside this block, autograd saves tensors as usual.

Saving training state

```
state = {
    "model": model.state_dict(),
    "optimizer": optimizer.state_dict(),
    "scheduler": scheduler.state_dict(),
    "update": update,
}
torch.save(state, "training_state.pt")
```

Training on a GPU

GPU: graphics processing unit

- Training repeatedly applies large matrix operations in the forward and backward passes.
- GPUs perform many of these calculations in parallel, often making training much faster.
- PyTorch runs these operations on the GPU. We use the same network, loss and optimizer.

GPU training in PyTorch

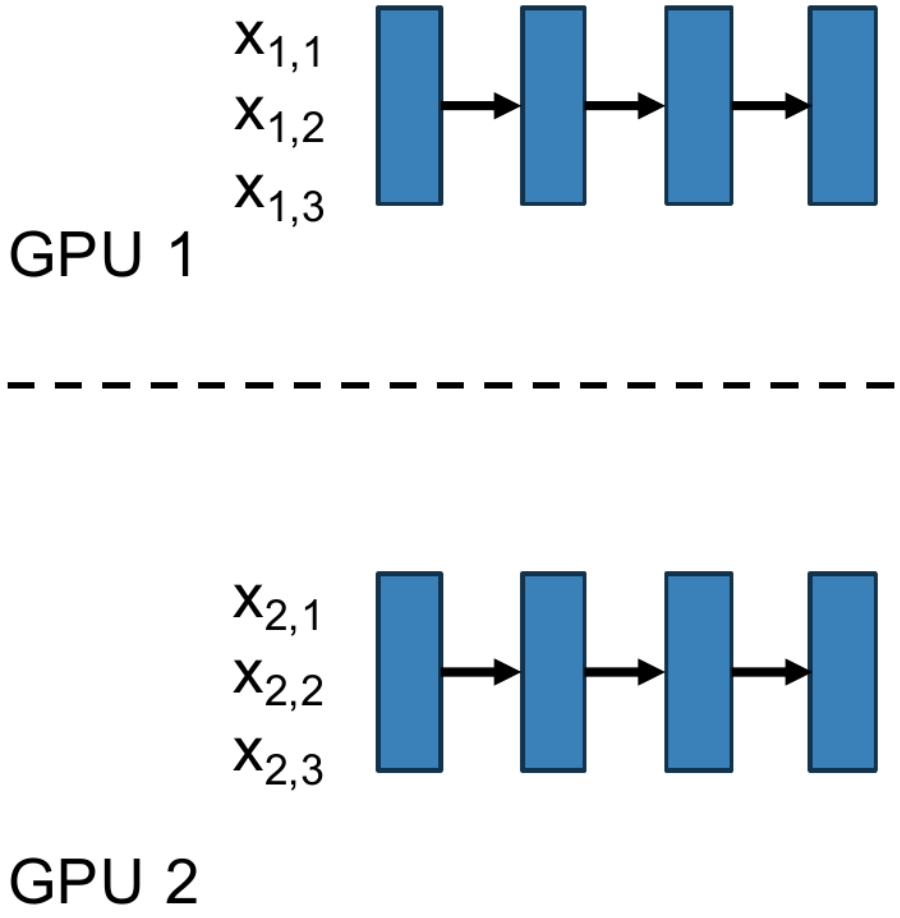
```
device = torch.device("cuda:0")          # first GPU
model = MLP().to(device)                 # move parameters
optimizer = torch.optim.AdamW(model.parameters(), lr=1e-3)
model.train()

for x, y in loader:
    x, y = x.to(device), y.to(device)  # move this batch
    optimizer.zero_grad()              # clear gradients
    loss = criterion(model(x), y)
    loss.backward()                    # compute gradients
    optimizer.step()                   # update parameters
```

`x.device` reports location; `x.to(device)` moves the tensor.

Data parallelism

One computer, two GPUs

- Copy the network and weights.
- Split the batch between GPUs.
- Combine gradients for one update.

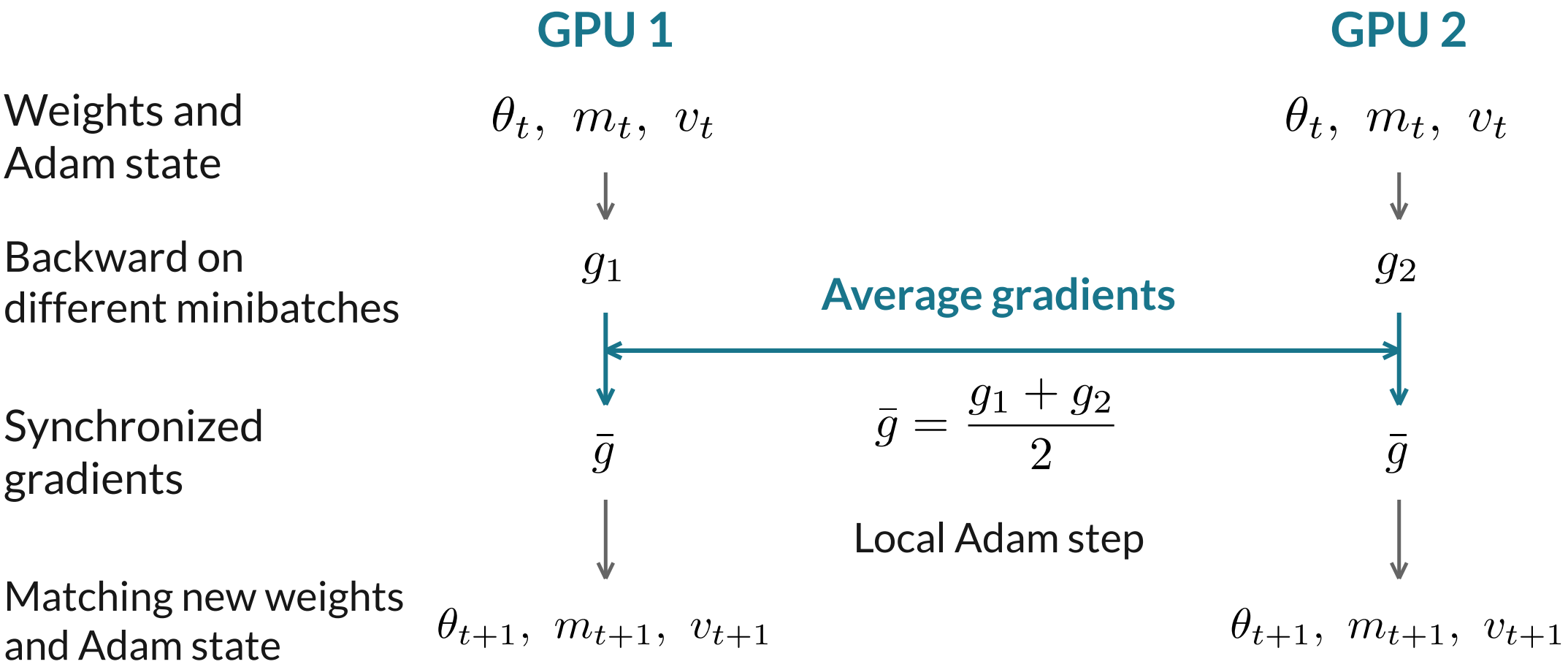


PyTorch: DistributedDataParallel (DDP)

Adapted from Stanford CS231n

Gradient synchronization

Equal local batch sizes. Same starting weights, state and settings.



Batch size in PyTorch DDP

One process per GPU

$$B_{\text{effective}} = b \times G \times K$$

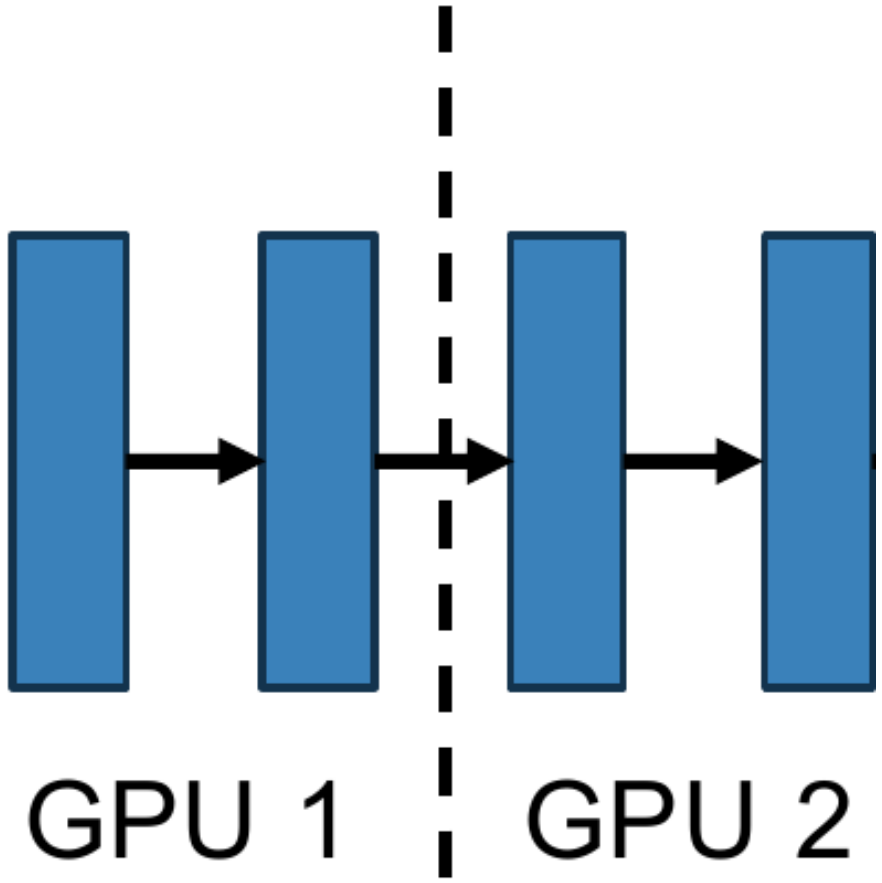
b Batch per GPU G Number of GPUs K Accumulation steps

```
DataLoader(..., batch_size=b)  # batch per GPU
```

More GPUs with unchanged settings increase the effective batch.

To preserve it, reduce the batch per GPU or accumulation steps.

Model parallelism



Adapted from Stanford CS231n

Split the layers of one network.

Forward Pass activations to GPU 2.

Backward Return activation gradients to GPU 1.

Distributes model storage.

Successive layers still depend on earlier outputs.

Training and selecting a model

1. Fix the dataset version and input-processing configuration
2. Fix the evaluation metric and record random seeds
3. Train with the chosen loss and regularization
4. Evaluate periodically in the correct mode
5. Compare methods with a well-trained baseline
6. Restore the checkpoint selected using validation
7. Evaluate the selected procedure on test data

Next: convolutional neural networks

- Dense layers connect every input feature to every output unit.
- Images and other structured inputs contain local, repeated patterns.
- Convolution builds that structure into the model.

Choosing a project topic

Start from a problem you want to solve or understand.

New methods	Develop or improve a model or training method.
Applications	Apply deep learning to a meaningful problem, with suitable data and evaluation.
Empirical studies	Investigate a clear question about models, data or training through experiments.

Reading the closest papers

First pass

Abstract, figures
and conclusion

Identify the task, main idea and principal result.

For a relevant paper

Method, experiments
and implementation

Check the data, comparisons and computing requirements.

Proposal: identify what you plan to read and why.

Your technical contribution

Starting point

Papers, datasets, code and pretrained models

Running the existing implementation establishes a starting point.

Your work

- Adapt or implement the method for the task.
- Justify the important design choices.
- Carry out experiments and analysis.

A new architecture or a leading score is not required.

Baselines and evaluation

Better prediction	→	Task performance against a credible baseline
Faster inference	→	Measured runtime and memory, alongside prediction quality
Empirical study	→	Comparisons that distinguish plausible explanations

Understanding the results

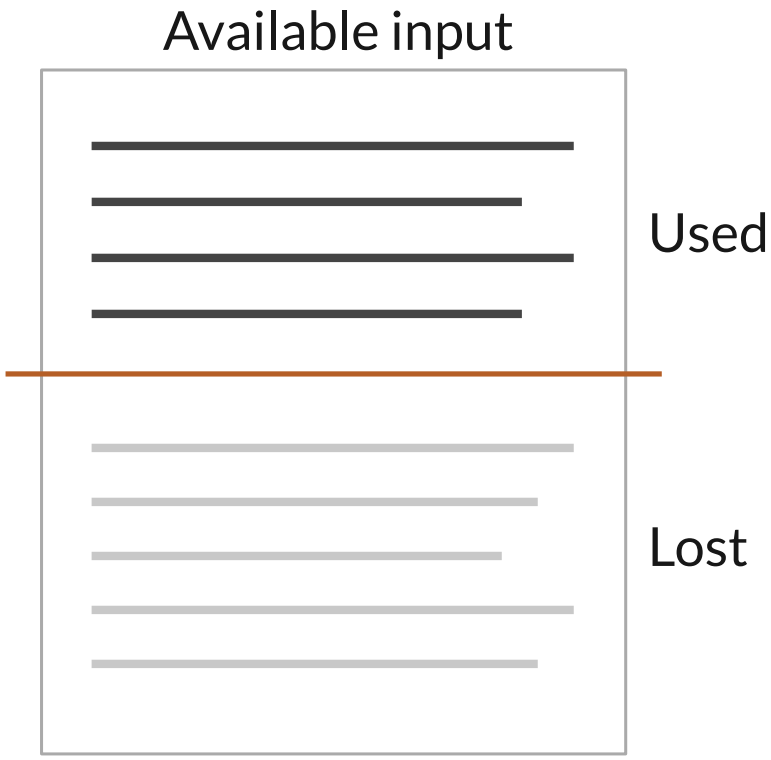
Suppose errors are more common on long inputs.

Check the pipeline

Is relevant information discarded before the model receives it?

Test the explanation

Compare with an input representation that preserves that information.



Keeping the project feasible

Data and code	Verify access and inspect actual examples.
Computation	Estimate memory and runtime from a small complete run.
Core project	Reserve time for comparisons, analysis and the GPU queue.

Project proposal

Explain what you plan to do.

- The problem and why it matters.
- Readings you plan to examine and a possible starting method.
- Planned data, implementation and evaluation.
- Feasibility, risks and the team's next steps.
- Form a team of 3–4 and discuss the scope early.

Midterm report

A sanity check that your team is making real progress.

Examples of work to report

- Progress on the problem, data or implementation.
- Experiments, preliminary results or analysis.
- Difficulties encountered and your next steps.

You do not need to have completed every item.

Final report

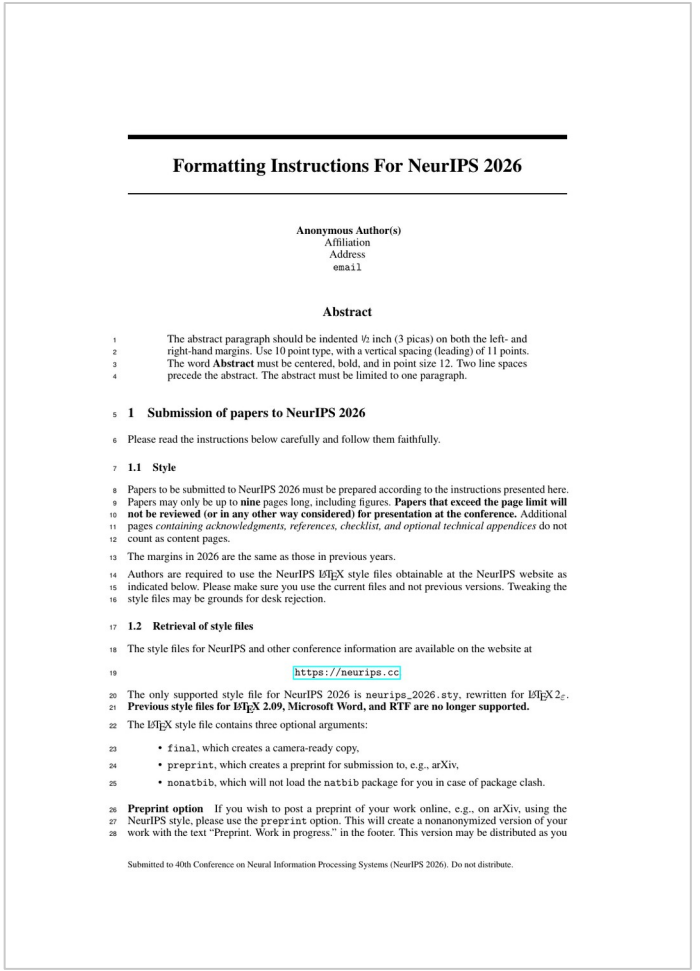
Explain the completed work and what the evidence shows.

- Problem, motivation, related work and your contribution.
- Method, data and experimental setup.
- Results and meaningful comparisons.
- Analysis, limitations and conclusions.
- Team contributions and sources of reused work.

Report format

Use a conference-paper template, such as NeurIPS.

- LaTeX is recommended.
- Submit the report as a PDF.
- Give figures and tables clear captions.
- Include references and citations.
- List team member names and contributions.



Formatting Instructions For NeurIPS 2026

Anonymous Author(s)
Affiliation
Address
email

Abstract

The abstract paragraph should be indented ½ inch (3 picas) on both the left- and right-hand margins. Use 10 point type, with a vertical spacing (leading) of 11 points. The word **Abstract** must be centered, bold, and in point size 12. Two line spaces precede the abstract. The abstract must be limited to one paragraph.

1 Submission of papers to NeurIPS 2026

Please read the instructions below carefully and follow them faithfully.

1.1 Style

Papers to be submitted to NeurIPS 2026 must be prepared according to the instructions presented here. Papers may only be up to **nine** pages long, including figures. **Papers that exceed the page limit will not be reviewed (or in any other way considered) for presentation at the conference.** Additional pages *containing acknowledgments, references, checklist, and optional technical appendices* do not count as content pages.

The margins in 2026 are the same as those in previous years.

Authors are required to use the NeurIPS LaTeX style files obtainable at the NeurIPS website as indicated below. Please make sure you use the current files and not previous versions. Tweaking the style files may be grounds for desk rejection.

1.2 Retrieval of style files

The style files for NeurIPS and other conference information are available on the website at

`https://neurips.cc`

The only supported style file for NeurIPS 2026 is `neurips_2026.sty`, rewritten for LaTeX 2ε. **Previous style files for LaTeX 2.09, Microsoft Word, and RTF are no longer supported.**

The LaTeX style file contains three optional arguments:

- `final`, which creates a camera-ready copy,
- `preprint`, which creates a preprint for submission to, e.g., arXiv,
- `nonatbib`, which will not load the `natbib` package for you in case of package clash.

Preprint option If you wish to post a preprint of your work online, e.g., on arXiv, using the NeurIPS style, please use the `preprint` option. This will create a nonanonymized version of your work with the text "Preprint. Work in progress." in the footer. This version may be distributed as you

Submitted to 40th Conference on Neural Information Processing Systems (NeurIPS 2026). Do not distribute.

NeurIPS template · NeurIPS 2026 Program Chairs · CC BY 4.0