ELEC3544 · Fall 2026

Data Science with Foundation Models

Yang Liu

The University of Hong Kong



Deep Learning

Learning nonlinear representations with neural networks and gradient descent

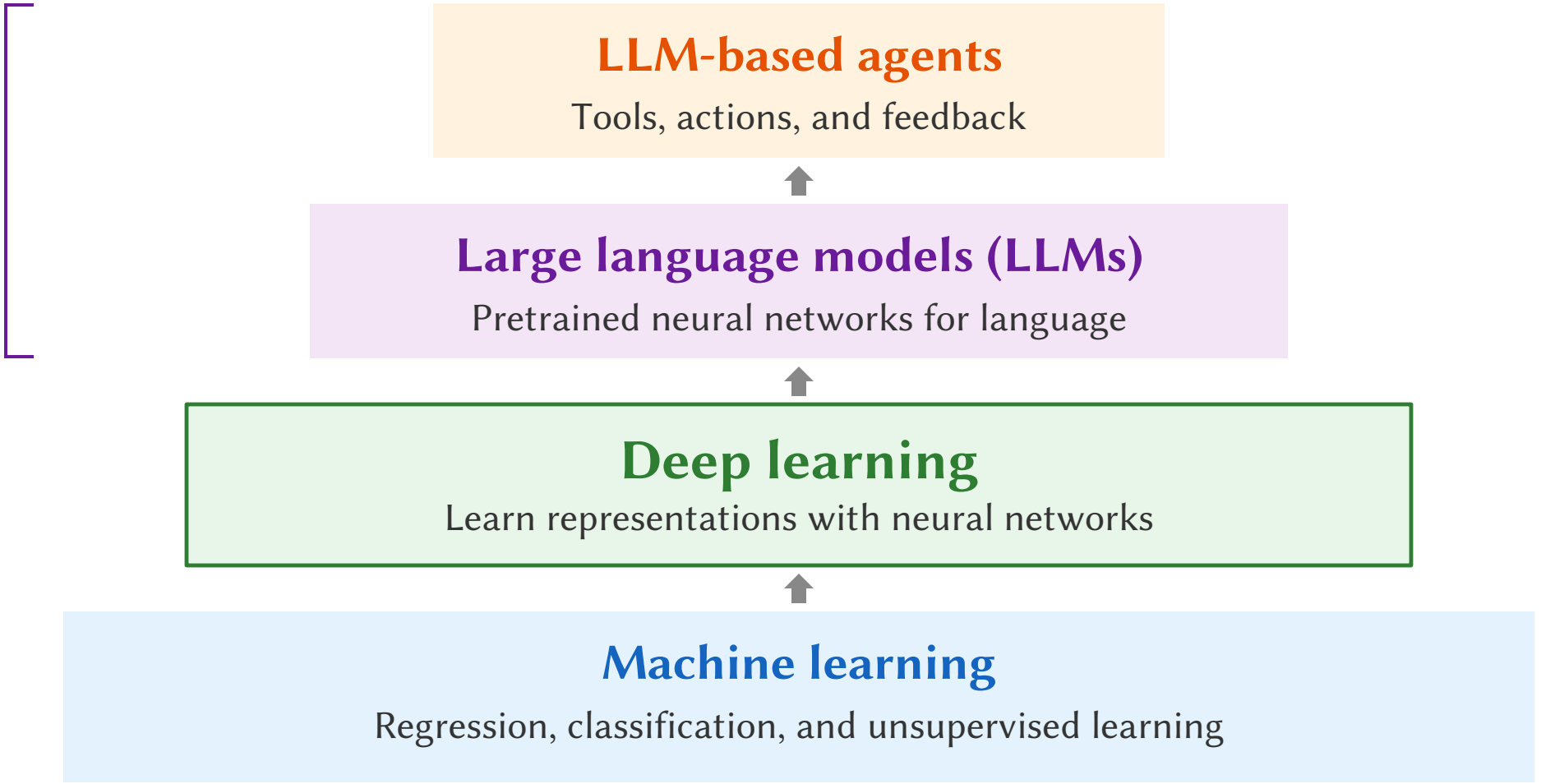
Slides adapted in part from MIT 6.7960 Deep Learning https://deeplearning6-7960.github.io/

Data Science Trajectory

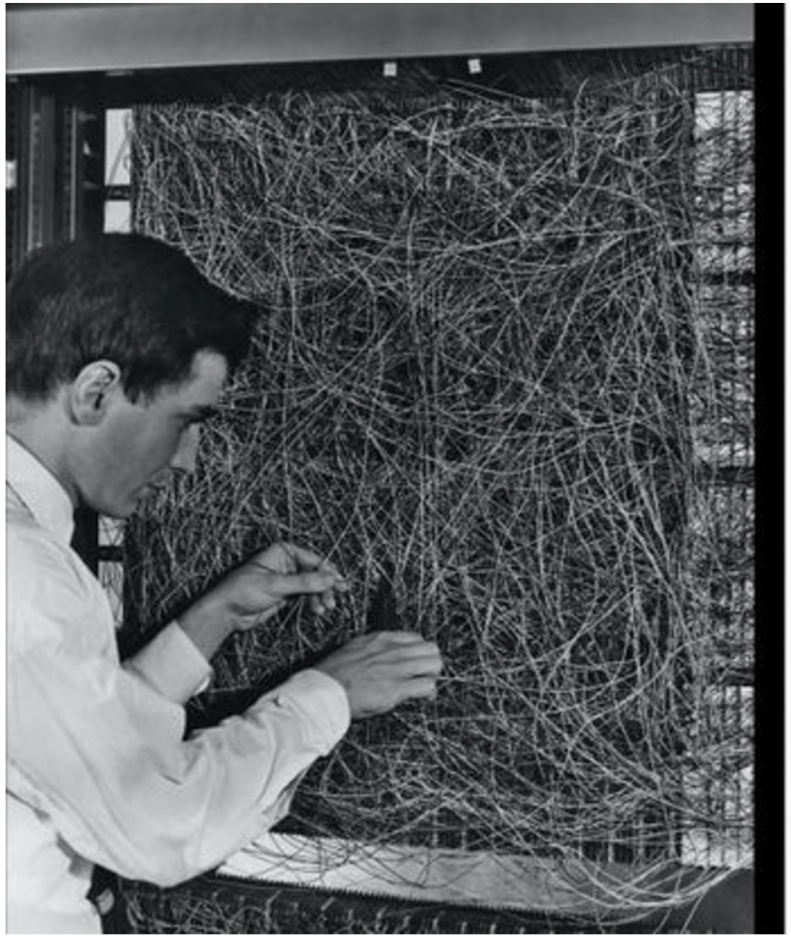
Next lecture
L9

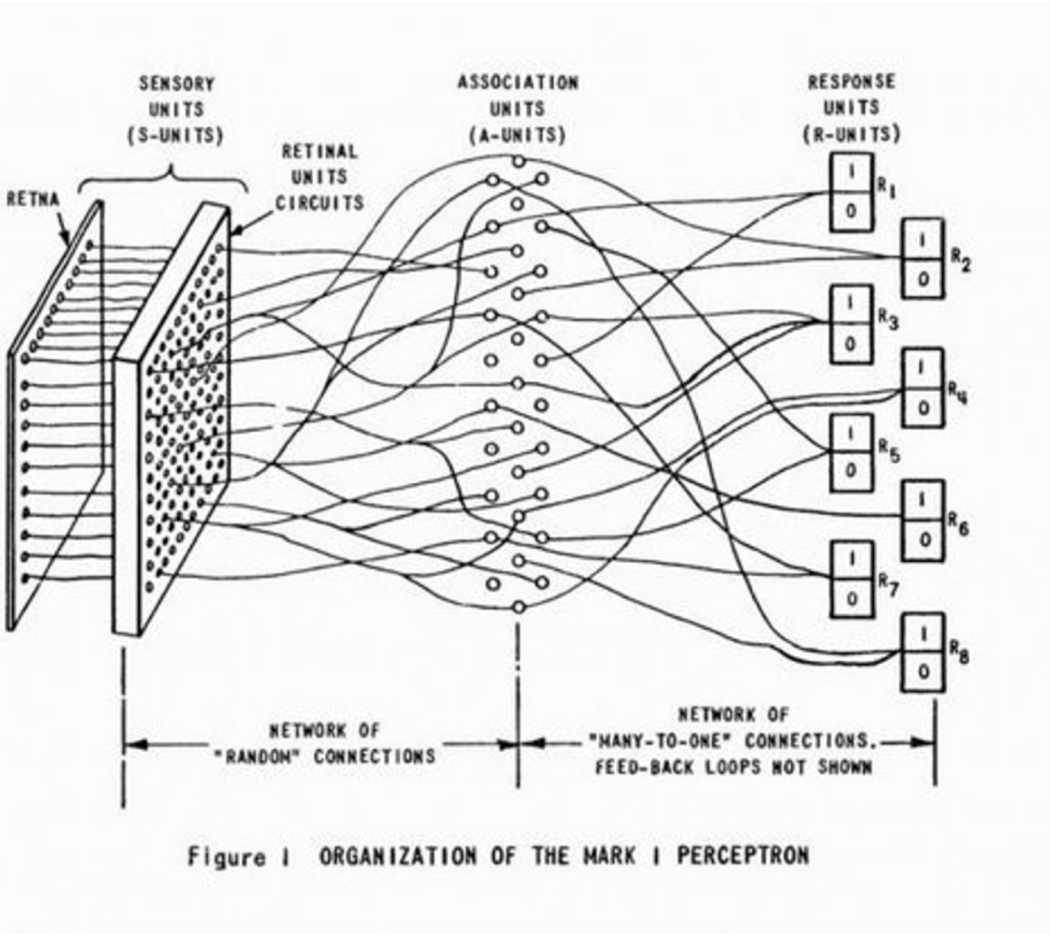
Today
L8

Previously
L5–L7



Perceptrons / MLPs: early learning machines



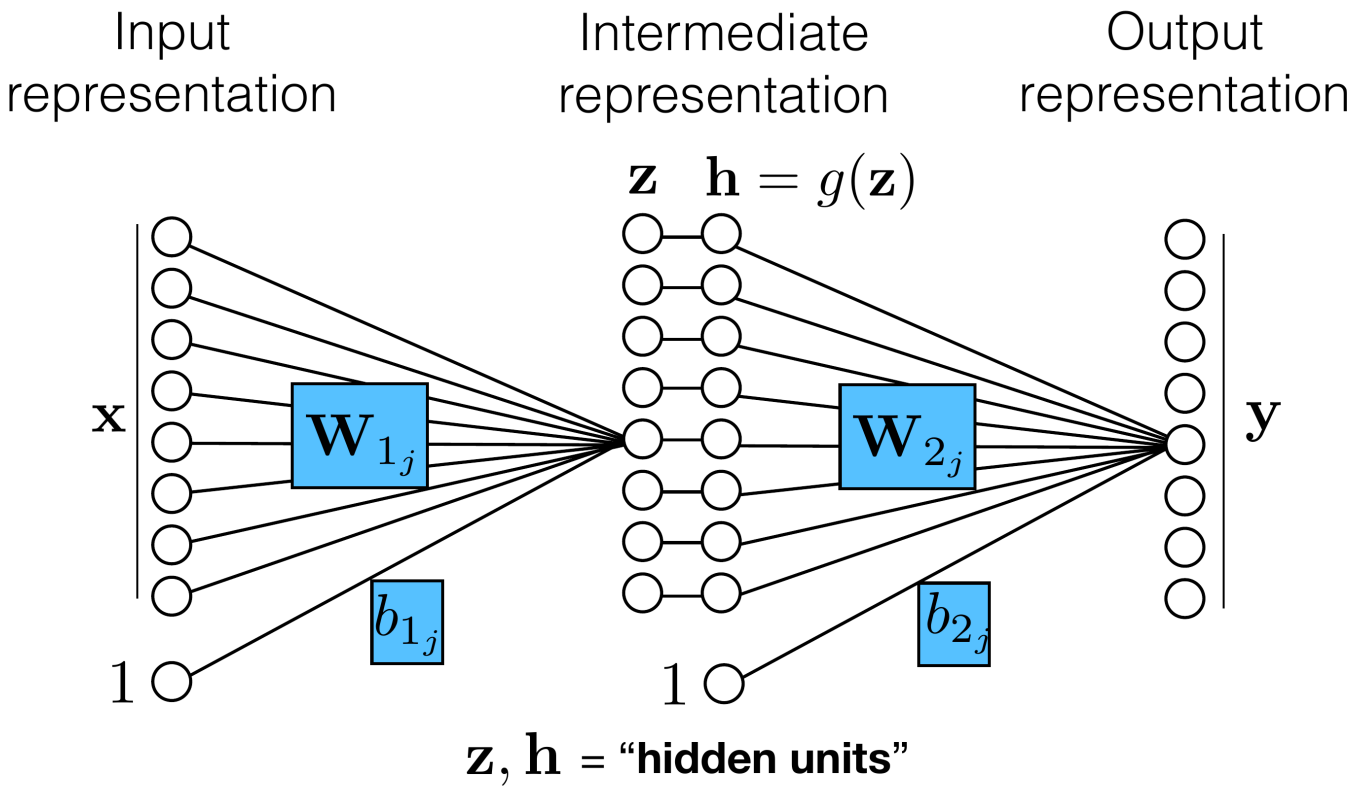


Mark I, c. 1960: visual input, wired features, and adjustable output weights.

Photo: Frederic Lewis / Archive Photos / Getty Images (c. 1960).

Diagram: Mark I Perceptron Operators' Manual (1960), Fig. 1.

Perceptrons / MLPs



Backpropagation: Rumelhart, Hinton & Williams (1986)

Loss gradients let the prediction task train hidden representations.

Hinton also shared the 2024 Physics Nobel with Hopfield for neural-network foundations.

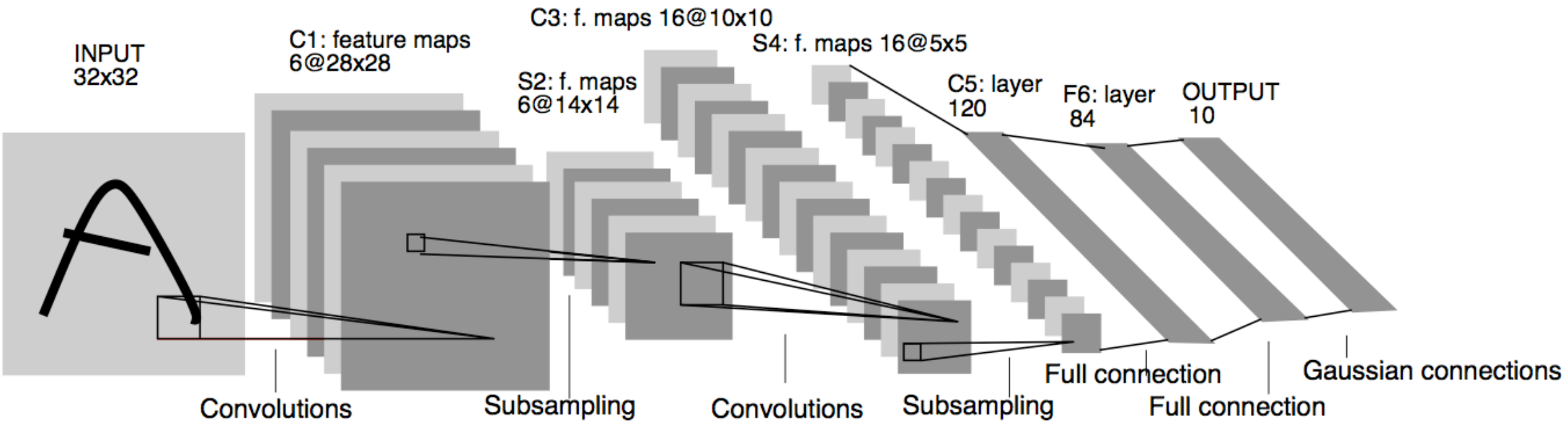
Diagram: K. He, MIT 6.7960 (2026).

Rumelhart et al., Nature (1986)

Nobel Prize in Physics (2024)

Deep networks: LeNet-5 learns visual features

1998 LeNet-5 **Pixels → learned feature maps → prediction**



Turing Award connection: LeCun's convolutional networks

Local connections, shared filters, and backpropagation learn visual features.

LeCun et al. (1998), LeNet-5, Fig. 2.

Recognition for neural-network research

2018 ACM A.M. Turing Award



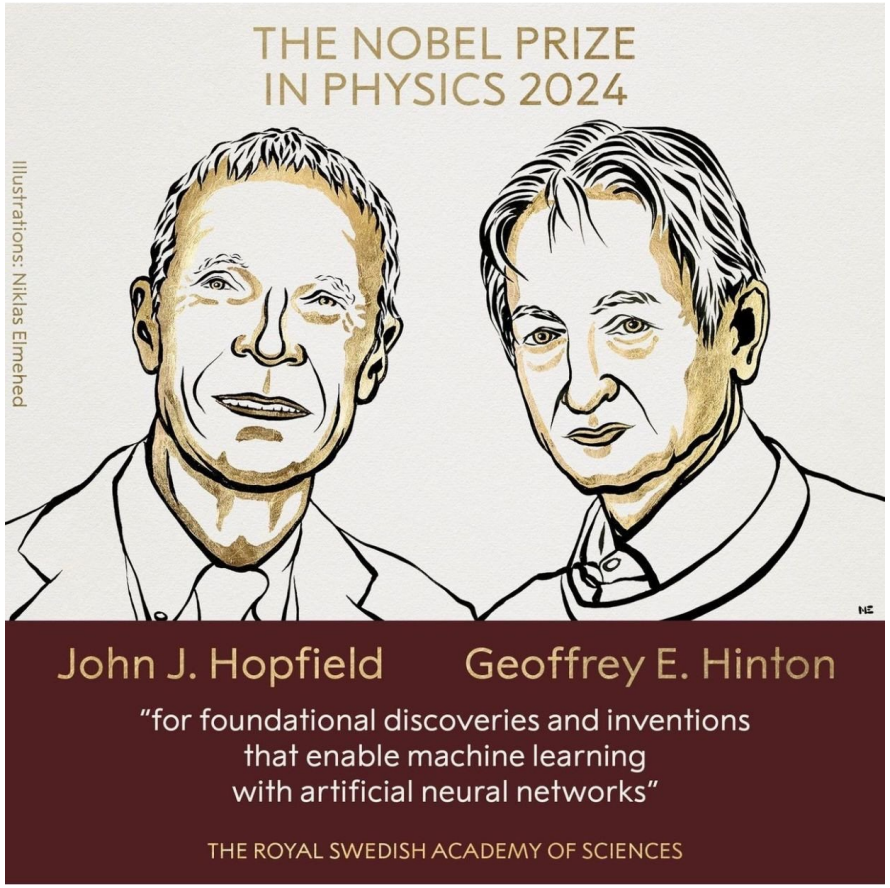
Yann LeCun, Geoffrey Hinton, Yoshua Bengio

Left to right

Deep-network training, convolutional networks, and learned representations.

Photo: ACM, FY2019 Annual Report

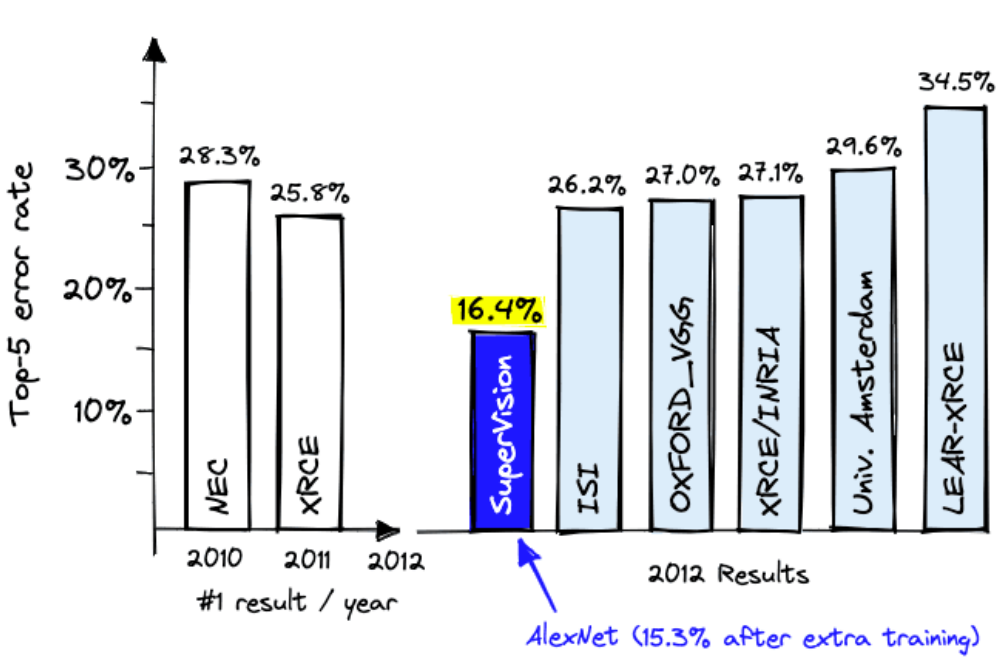
2024 Nobel Prize in Physics

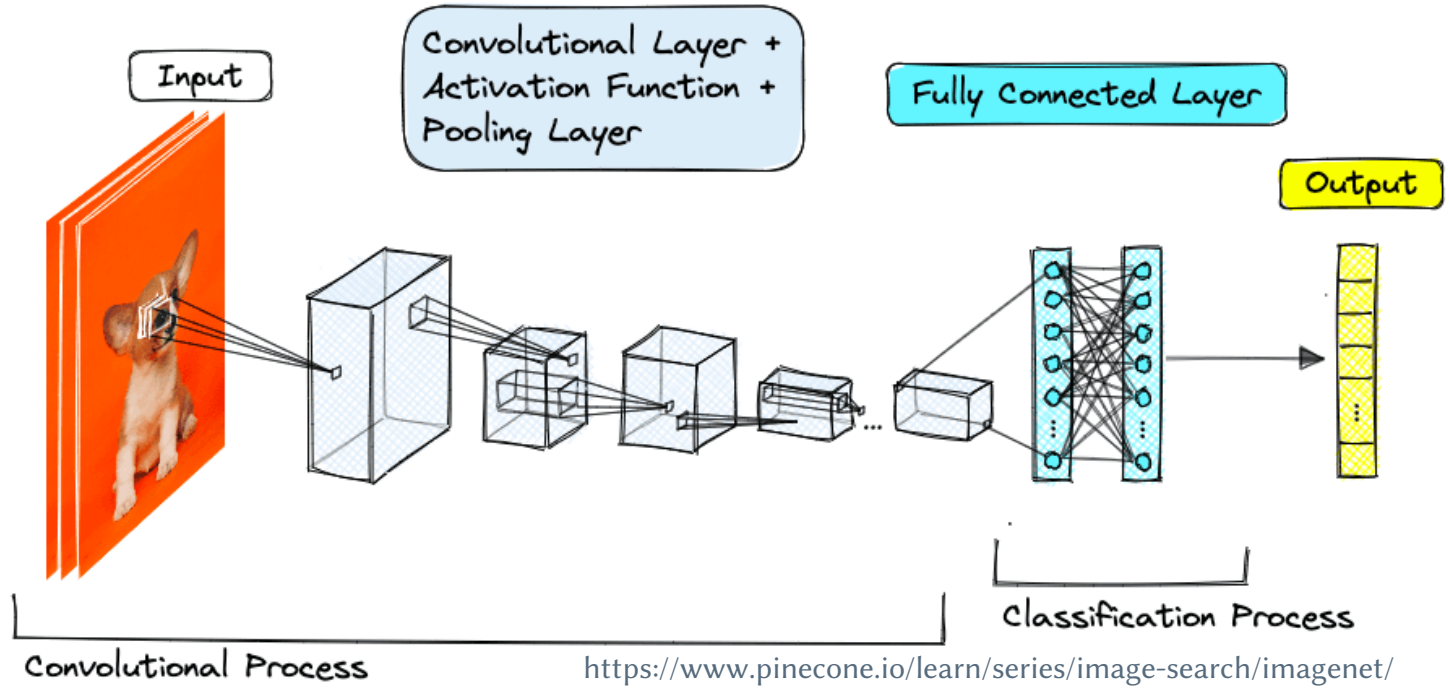


Hopfield networks and Boltzmann machines.

Illustration: Niklas Elmehed © Nobel Prize Outreach
Image via SNOLAB

Deep networks: the 2012 ImageNet breakthrough





https://www.pinecone.io/learn/series/image-search/imagenet/

Turing Award connection: ReLU and dropout improve deep CNN training

System enablers: 1.2 million images, two GPUs per network, and data augmentation.

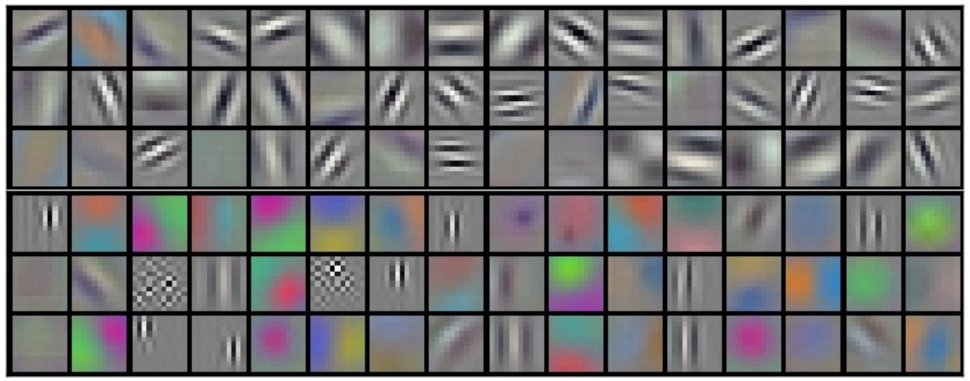
SIFT + Fisher ensemble	AlexNet: 5-CNN ensemble	Top-5 error reduction
26.2%	16.4%	9.8 points

Top-5 test error (lower is better).

Krizhevsky, Sutskever & Hinton (2012), Fig. 2 and Table 2.

AlexNet: learned filters and real predictions

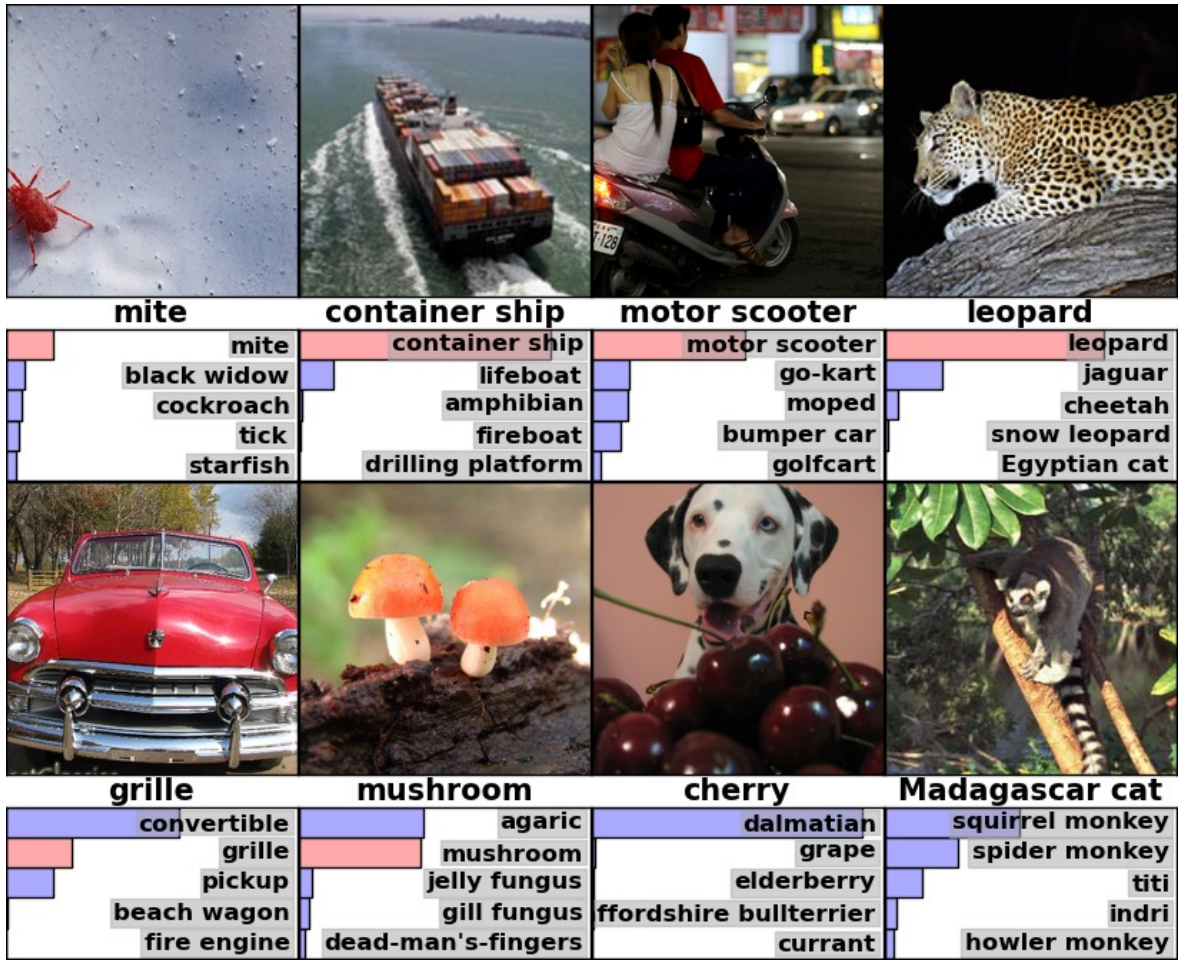
96 learned first-layer filters



Edges, orientations, and color patterns emerge from image–label training.

These are learned weights. Feature-map activations also depend on the input image.

Original top-5 predictions

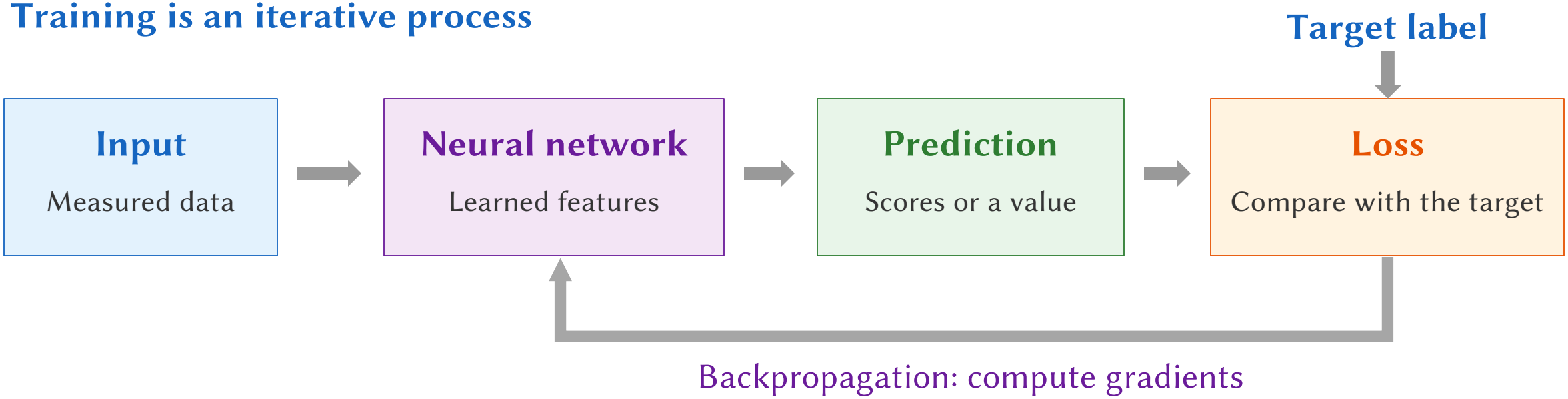


ILSVRC-2010 test images. Labels below photos show the ground truth.

Krizhevsky, Sutskever & Hinton (2012), Figs. 3–4. Original filters, images, labels, and probability bars.

Deep networks: learning end to end

Training is an iterative process



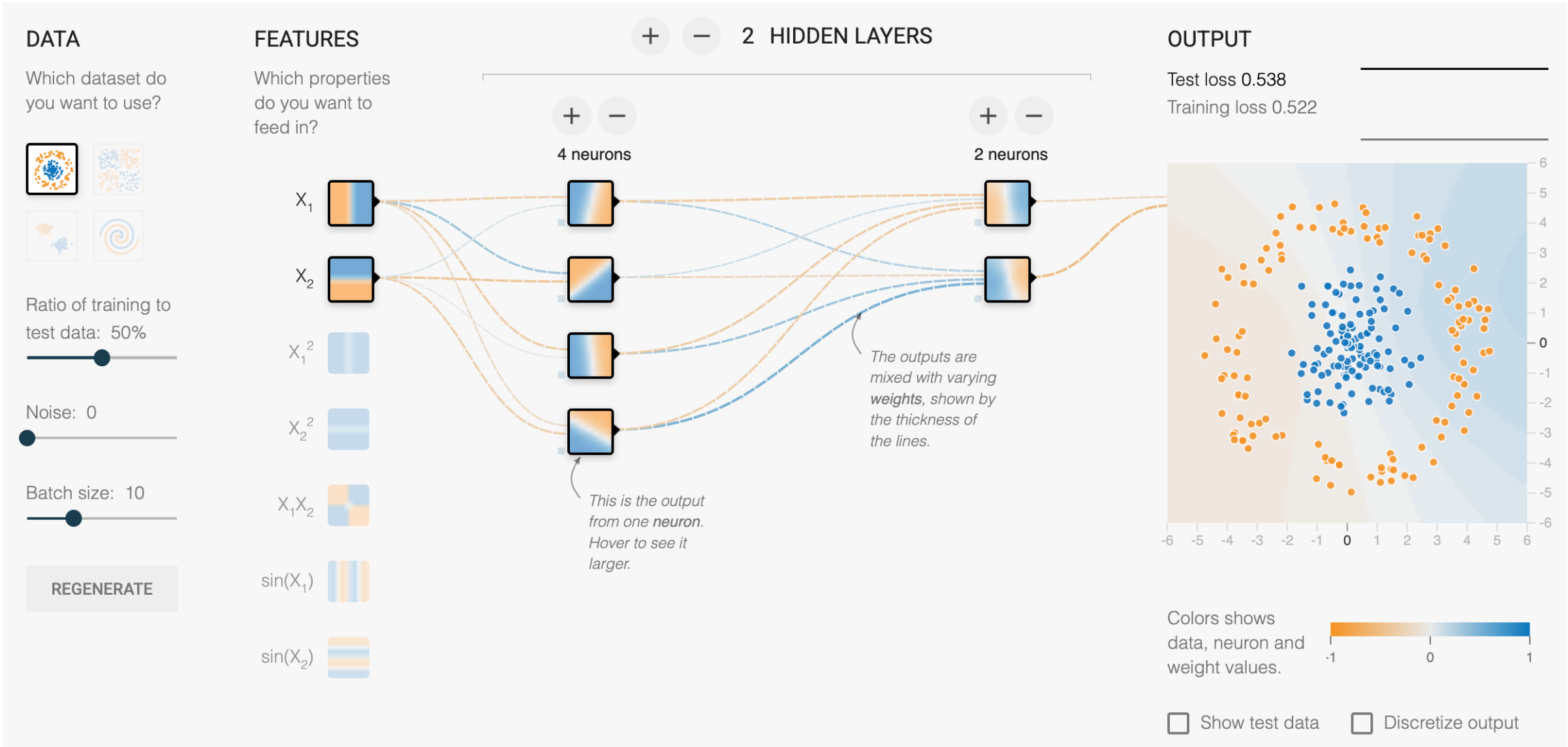
Optimizer: update weights, then repeat

Inference uses the learned weights. No target labels or weight updates are needed.

Representation: express a function. Optimization: fit it. Generalization: test it.

Backpropagation landmark: Rumelhart, Hinton & Williams (1986), Learning representations by back-propagating errors.

Deep networks: visualization



https://playground.tensorflow.org/

Regression models and their representations

Model	Representation	Parameters learned from data
Linear regression	Original measured inputs	Slope / weights and intercept
Polynomial regression	Fixed powers and interactions	Coefficients of those fixed features
Neural regression	Nonlinear hidden activations	Hidden features and output weights

$$J(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{i=1}^{n} \left(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i) - y_i \right)^2$$

Hold the target, split, and loss fixed to compare model classes.

Regression, classification, and one neural layer

All three start with learned weights and a bias.

Linear regression

Real-valued prediction

$$\hat{y} = \boldsymbol{w}^\top \boldsymbol{x} + b$$

Identity activation

Uses the score directly as the prediction.

Binary classification

Logistic regression

$$\hat{p} = \sigma(\boldsymbol{w}^\top \boldsymbol{x} + b)$$

Sigmoid output

Predicts the probability of class 1.

One neural layer

Learned representation

$$\boldsymbol{h} = \varphi(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

Sigmoid or ReLU

Produces a feature vector for the next layer.

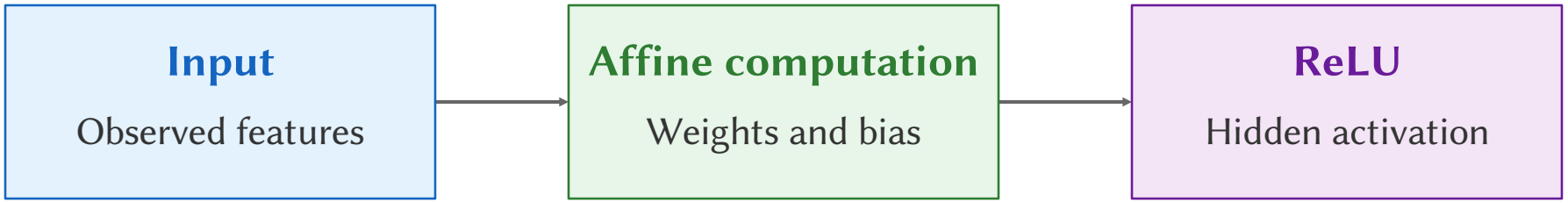
Nonlinear activations

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

$$\mathrm{ReLU}(z) = \max(0, z)$$

A single sigmoid output neuron is logistic regression. Hidden layers learn features.

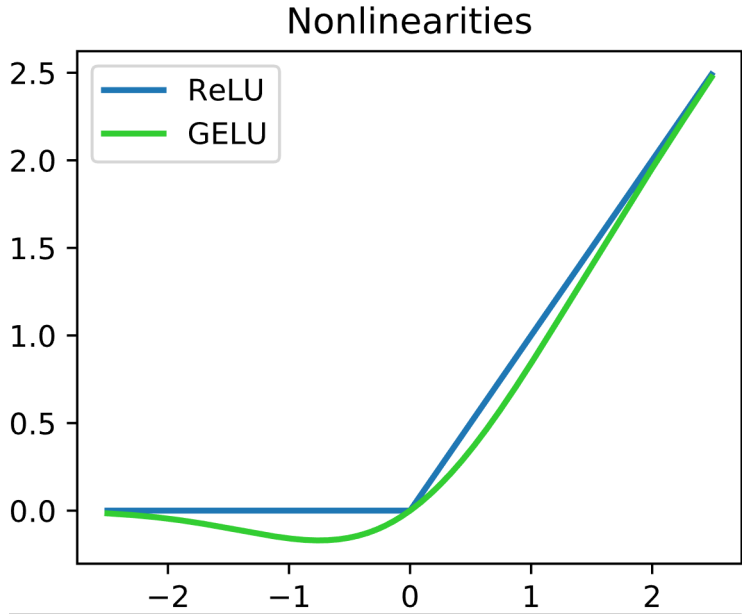
A neuron: weighted sum and nonlinear activation



$$z = w^{\mathsf{T}} x + b, \qquad h = \mathrm{ReLU}(z)$$

Weighted sum followed by nonlinear activation

$$\mathrm{ReLU}(z) = \begin{cases} 0, & z \le 0, \\ z, & z > 0, \end{cases} \qquad \mathrm{ReLU}'(z) = \begin{cases} 0, & z < 0, \\ 1, & z > 0. \end{cases}$$



Parameters are shared across examples. Activations depend on the input.

Nonlinearity changes the function family

Two affine layers still produce one affine map

$$h^{(2)} = W^{(2)}\left(W^{(1)}x + b^{(1)}\right) + b^{(2)}$$

$$= \underbrace{W^{(2)}W^{(1)}}_{\widetilde{W}}\,x + \underbrace{W^{(2)}b^{(1)} + b^{(2)}}_{\widetilde{b}}$$



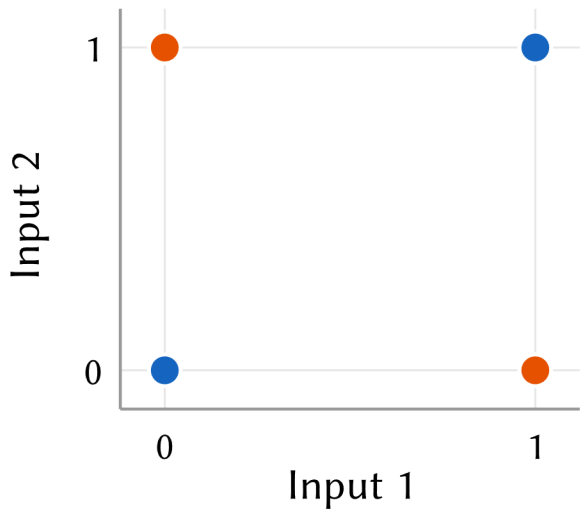
Nonlinear activations allow different input regions to use different slopes.

A representation can make a difficult boundary simple

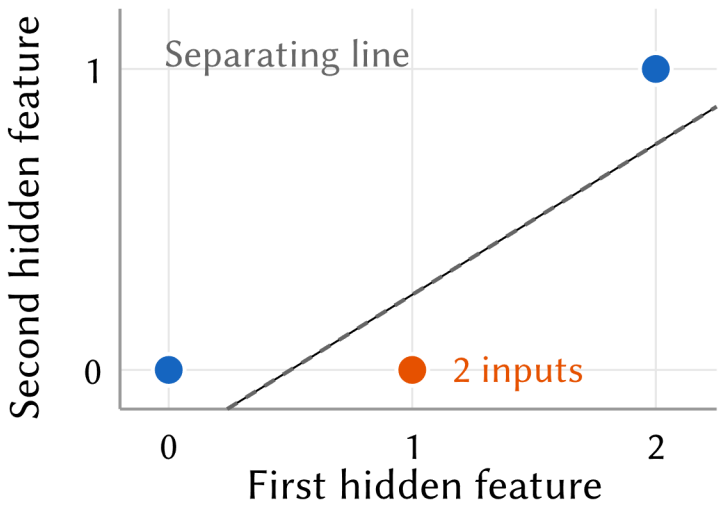
$$h_1 = \mathrm{ReLU}(x_1 + x_2), \qquad h_2 = \mathrm{ReLU}(x_1 + x_2 - 1)$$

$$\hat{y} = h_1 - 2h_2$$

Input space



Constructed hidden features



Linear output

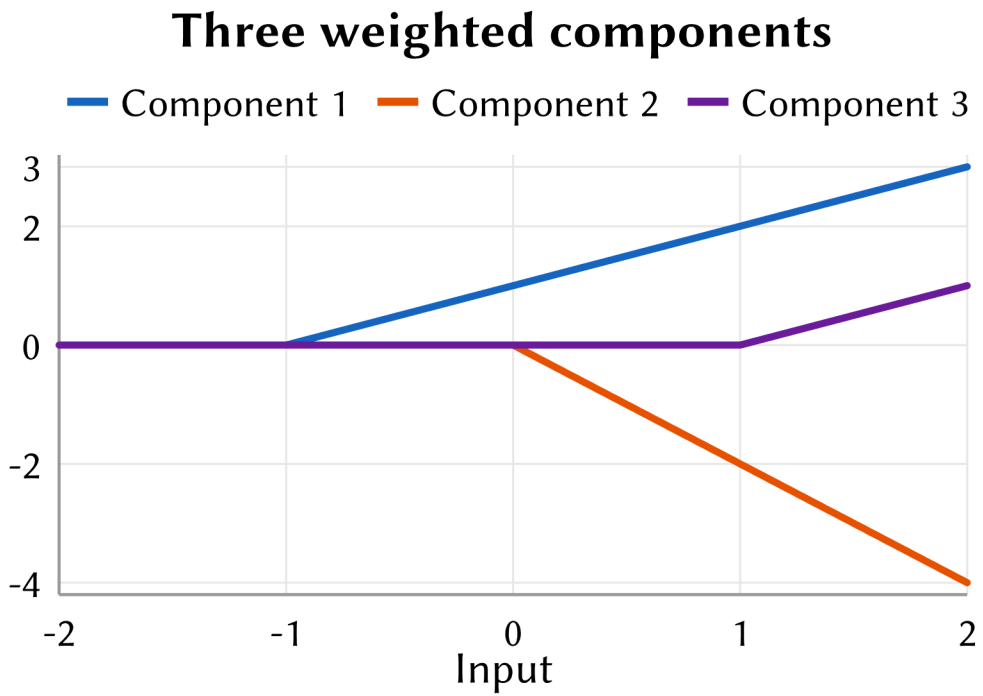
Input pair	Score
(0, 0)	0
(0, 1)	1
(1, 0)	1
(1, 1)	0

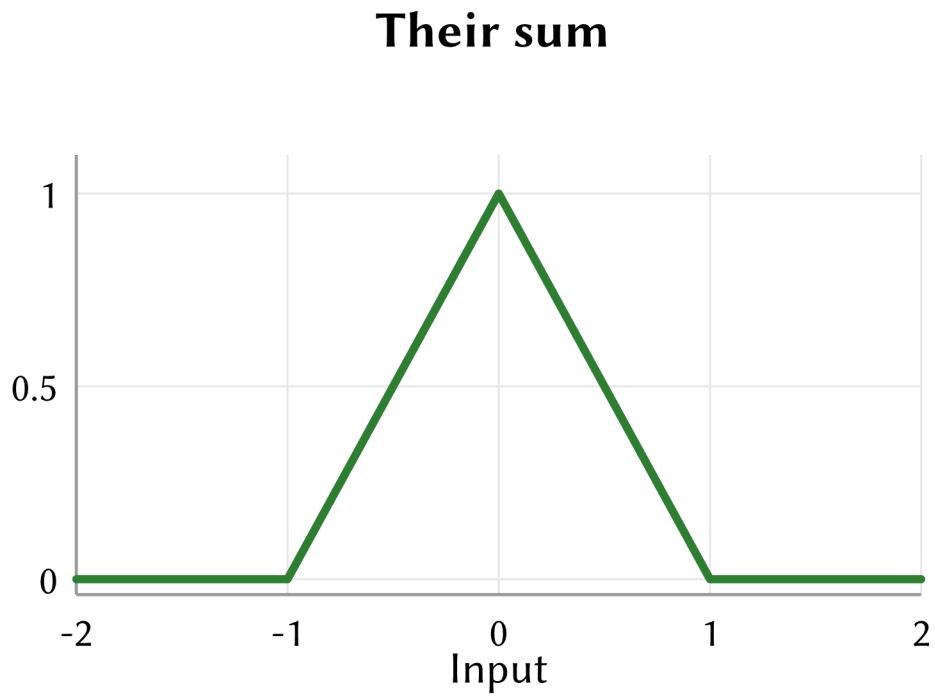
Class 0 Class 1

Hidden features can turn a nonlinear decision into a linear readout.

A curve built from simple ReLU pieces

$$f(x) = \text{ReLU}(x + 1) - 2\,\text{ReLU}(x) + \text{ReLU}(x - 1)$$





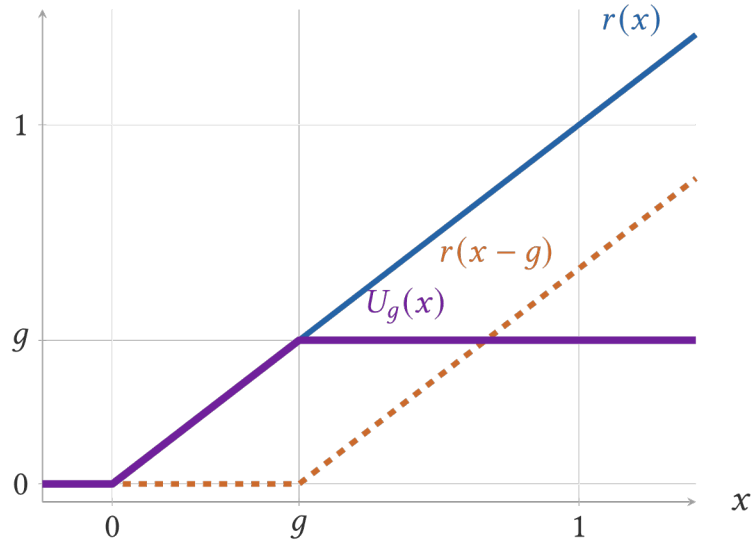
Fitting the hidden weights and biases moves the pieces used to build the curve.

ReLU steps and sampling blocks

$r(x) = \mathrm{ReLU}(x), \quad 0 < g < 1$

1 Cancel the slopes

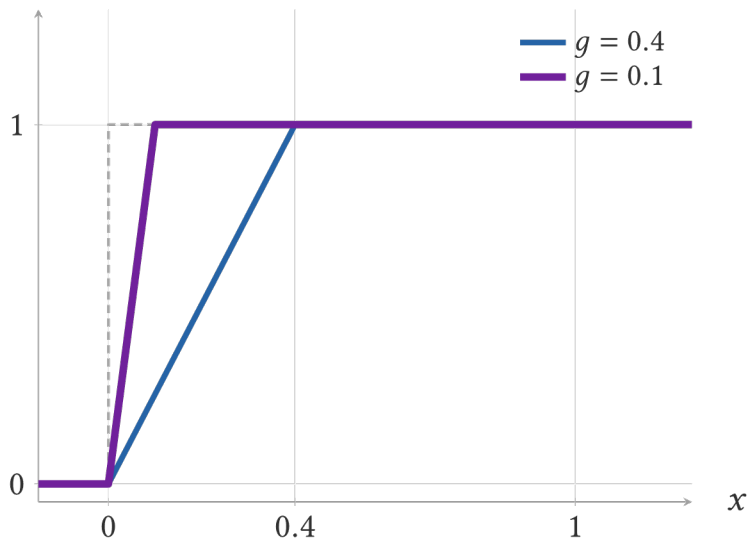
$$U_g(x) = r(x) - r(x-g)$$



Plateau height g

2 Normalize and sharpen

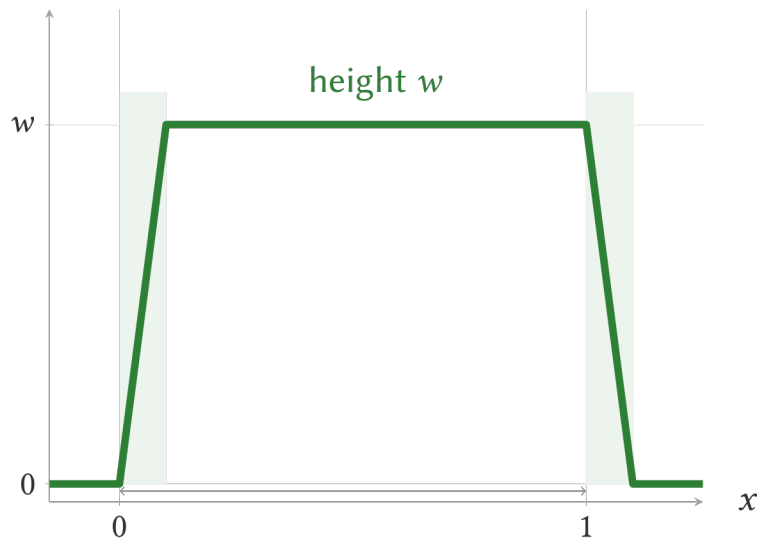
$$S_g(x) = U_g(x)/g$$



Smaller g: sharper edge

3 Make a sampling block

$$B_g(x) = w\,[S_g(x) - S_g(x-1)]$$



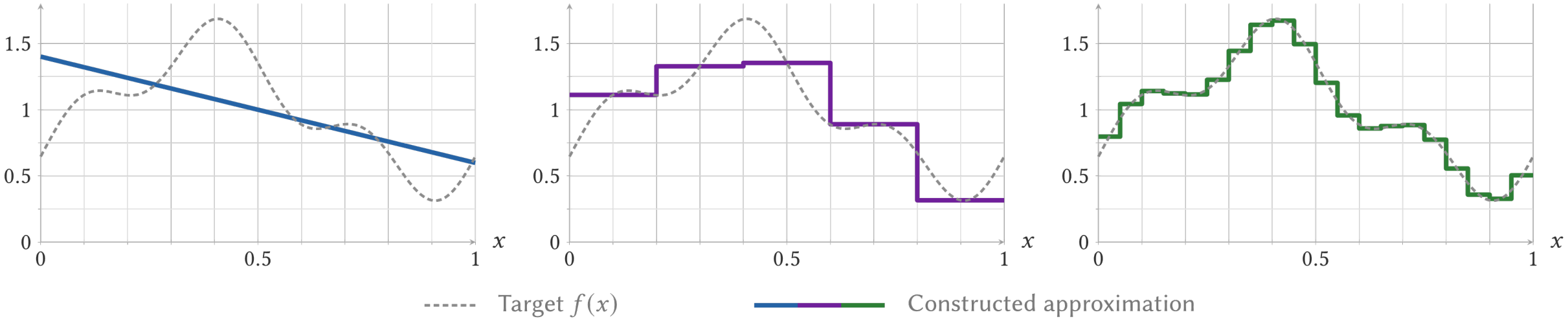
Four ReLUs; continuous edges

Four ReLUs make one local block; shift, weight, and add blocks to approximate a curve.

Universal approximation: what the theorem says

Constructed approximations: more local pieces follow the target more closely.

One straight line | **5 constant pieces** | **20 constant pieces**



B bins on $[0, 1]$, midpoint samples, $|f'| \le L$: $\max_x |f(x) - f_B(x)| \le L/(2B) = O(1/B)$

A continuous target on a compact domain; sufficient width and suitable nonlinear activations.

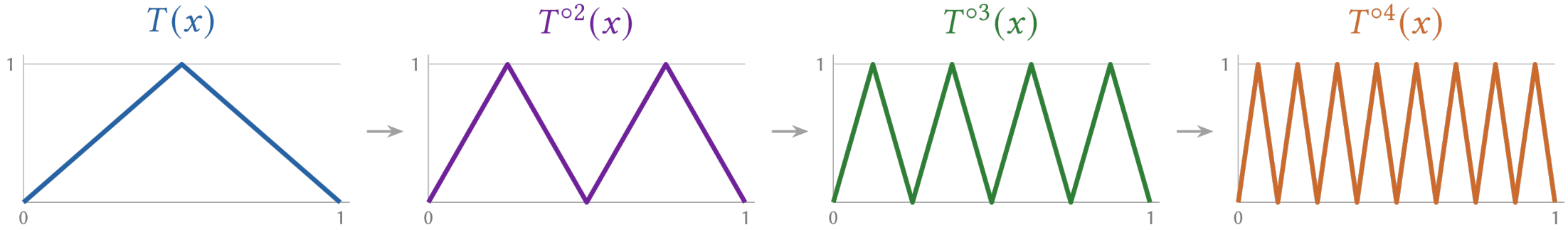
$$\sup_{\mathbf{x} \in K} |f(\mathbf{x}) - f_\theta(\mathbf{x})| < \varepsilon$$

Existence of suitable weights does not guarantee successful training or generalization.

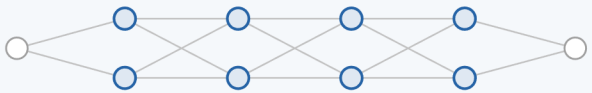
Illustration adapted from P. Isola, MIT 6.7960, Approximation Theory (2026). Constructed examples.

Depth vs. width: the same curve, fewer neurons

$$T(x) = 2\,\mathrm{ReLU}(x) - 4\,\mathrm{ReLU}(x - \tfrac{1}{2}), \qquad 0 \le x \le 1$$



Deep: compose the same block

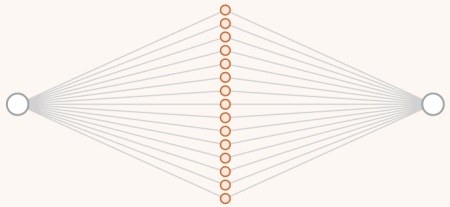


8 ReLU units

$2k$ units $\to 2^k$ pieces

4 hidden layers × width 2

Shallow: add separate hinges



≥15 ReLU units

for the same 16 pieces

One hidden layer: m units, at most $m + 1$ pieces

For repeated structure, depth can use exponentially fewer neurons.

Adapted from P. Isola, MIT 6.7960, Approximation Theory.

Related result: Telgarsky, COLT 2016.

CTA Demo 1: Regression with learned features

What We'll Explore

What changes when the model learns the nonlinear features?

1 **Predict**

Compare a line, a degree-5 polynomial, and a ReLU regressor on the same data.

2 **Run**

Compare in-domain test RMSE. Inspect extrapolation curves separately.

3 **Explain**

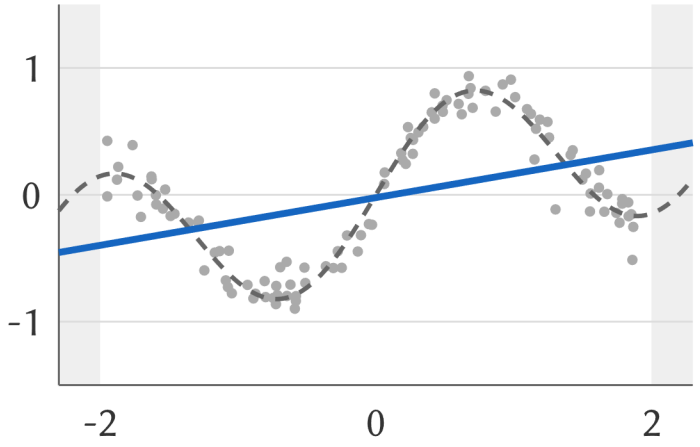
Identify which features are fixed and which are learned. Repeat with the Auto MPG example.

VERIFY Keep the data and split fixed. Explain a neural model's failure as carefully as its improvement.

Collaboration pattern: predict → request computation → inspect evidence → explain

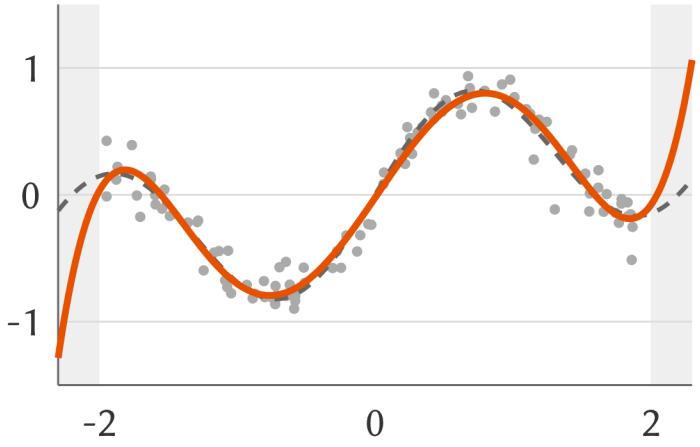
Regression results: the baseline still matters

Linear
2 parameters



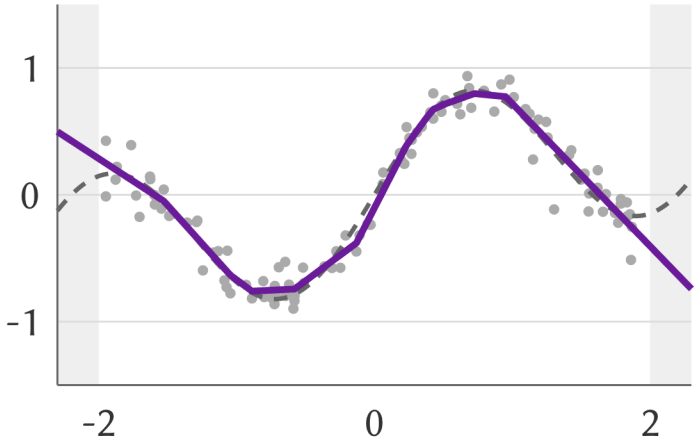
Test RMSE 0.501

Polynomial
6 parameters



Test RMSE 0.115

ReLU network
49 parameters



Test RMSE 0.128

Synthetic data. Gray points: training. Dashed: target. Shaded: extrapolation.

Real data: Auto MPG	Line	Quadratic	ReLU regressor
Test RMSE (MPG) ↓	4.97	4.52	4.66

219 training / 55 validation / 118 test cars. Horsepower predicts MPG.

Training: predictions, losses, and updates

Regression

$$\hat{y} = f_\theta(\mathbf{x}), \qquad L_i = \frac{1}{2}(\hat{y}_i - y_i)^2$$

A real-valued prediction; penalize its distance from the target.

Multiclass classification

$$p_k = \frac{e^{z_k}}{\sum_j e^{z_j}}, \qquad L_i = -\log p_{y_i}$$

A probability for each class; reward probability on the true class.

Training minimizes the average loss on a mini-batch

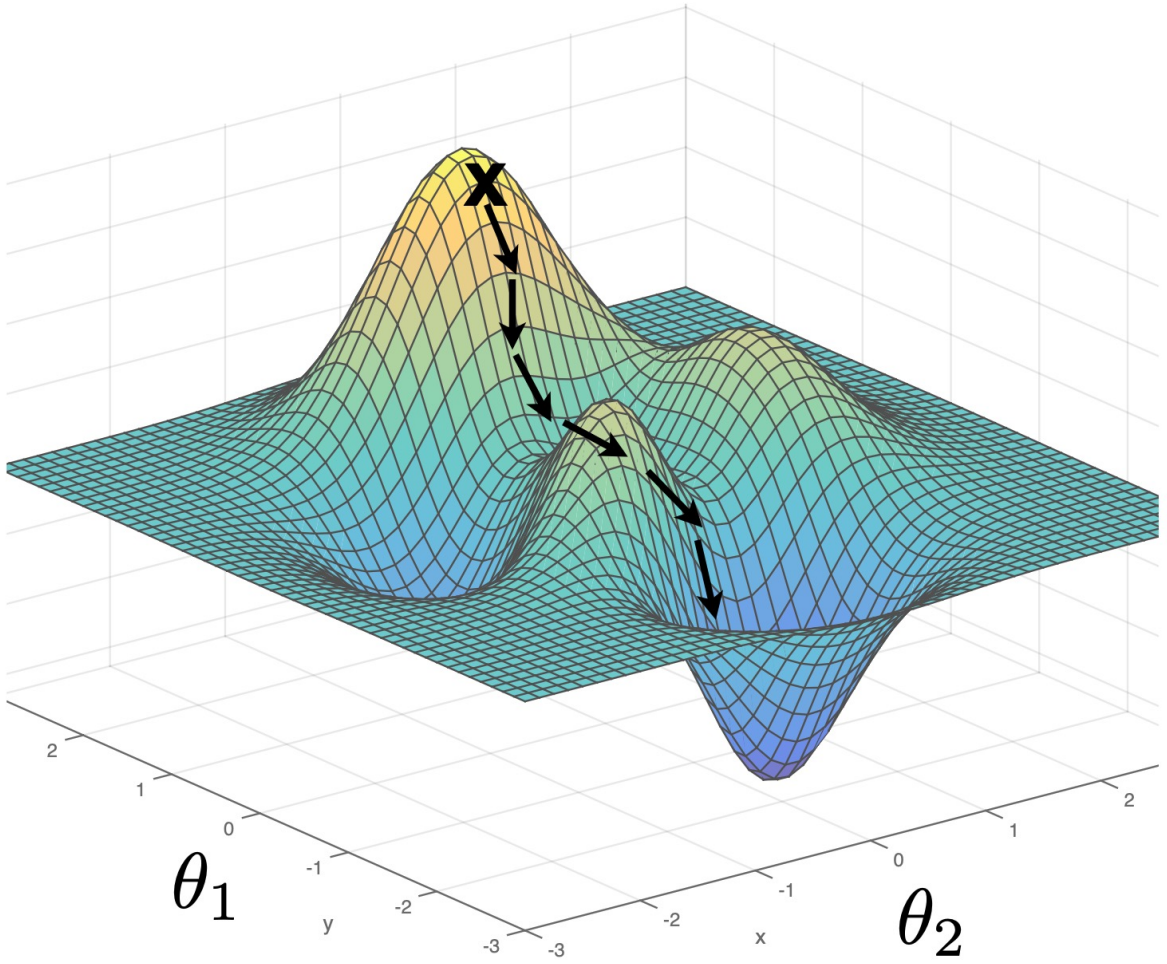
$$J_{\mathcal{B}}(\theta) = \frac{1}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} L_i \quad \Rightarrow \quad \theta_{\text{new}} = \theta - \eta \nabla_\theta J_{\mathcal{B}}(\theta)$$

learning rate

The gradient gives local sensitivity. The learning rate sets the step size.

Training: gradient descent

$J(\theta)$



$$\theta^* = \arg\min_{\theta} J(\theta)$$

One training example: the forward pass

A 2-input, 2-hidden-unit, 1-output network

Inputs and parameters

$$W = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}, \; x = \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \; b = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$v = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \qquad c = 0, \qquad y = 2$$

Hidden computation

$$z = Wx + b = \begin{bmatrix} 1 \\ -1 \end{bmatrix}, \qquad h = \mathrm{ReLU}(z) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

Prediction and loss

$$\hat{y} = v^{\top} h + c = 1, \qquad L_i = \frac{1}{2}(\hat{y} - y)^2 = 0.5$$

The second ReLU is inactive for this example.

Chain rule: how an early weight affects the loss

$$u = g(x), \quad y = f(u) \qquad \Longrightarrow \qquad \frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

Local sensitivities along one path

$$W_{11} \xrightarrow{2} z_1 \xrightarrow{1} h_1 \xrightarrow{1} \hat{y} \xrightarrow{-1} L_i$$

Multiply the local sensitivities to get the effect on the loss.

$$\frac{\partial L_i}{\partial W_{11}} = \frac{\partial L_i}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial h_1}\frac{\partial h_1}{\partial z_1}\frac{\partial z_1}{\partial W_{11}} = (-1)(1)(1)(2) = -2$$

What does –2 mean?

A +0.01 weight change predicts about –0.02 change in loss.
Actual loss: 0.5000 to 0.4802.

Compute once, reuse

$$\delta_1 = \frac{\partial L_i}{\partial z_1} = -1$$

Reuse this sensitivity for both incoming weights and the bias.

Multiply along a path. Add contributions when paths meet.

Adapted: P. Isola, MIT 6.7960, How to Train a Neural Net (2026)

One training example: gradients and an update

Loss and hidden sensitivities

$$\frac{\partial L_i}{\partial \hat{y}} = \hat{y} - y = -1$$

$$\boldsymbol{\delta} = (\hat{y} - y)\mathbf{v} \odot \mathrm{ReLU}'(\mathbf{z}) = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

Output-parameter gradients

$$\frac{\partial L_i}{\partial \mathbf{v}} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}, \qquad \frac{\partial L_i}{\partial c} = -1$$

Hidden-parameter gradients

$$\frac{\partial L_i}{\partial W} = \begin{bmatrix} -2 & -1 \\ 0 & 0 \end{bmatrix}, \qquad \frac{\partial L_i}{\partial \mathbf{b}} = \begin{bmatrix} -1 \\ 0 \end{bmatrix}$$

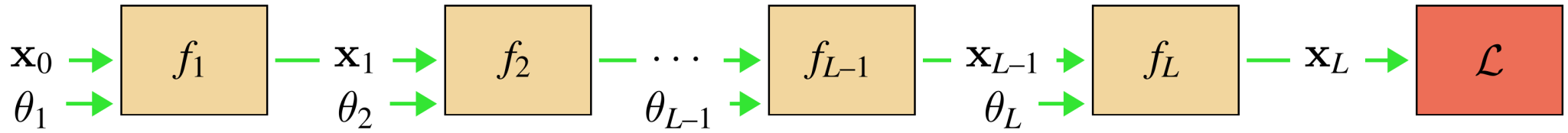
Update all parameters using the old gradients (learning rate 0.1).

$$W_{11,\text{new}} = 1 - 0.1(-2) = 1.2, \qquad \hat{y}_{\text{new}} = 1.86, \qquad L_{i,\text{new}} = 0.0098$$

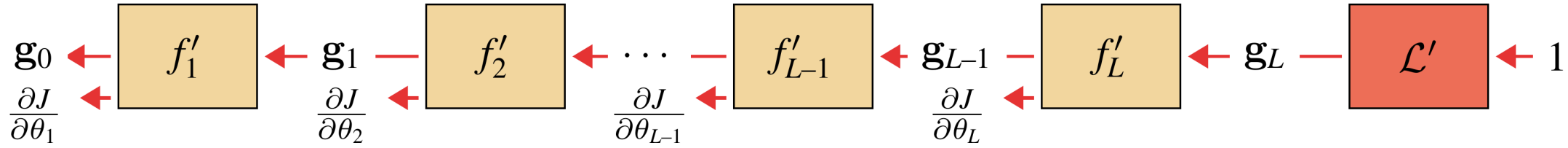
Reverse-mode autodiff and the chain rule

Chain rule at every block: multiply incoming sensitivity by the local derivative.

Forward:



Backward:



Here, J denotes the loss. Each block has its own activations and parameters.

Reuse chain-rule calculations: one backward pass computes all parameter gradients.

Original diagram: P. Isola, MIT 6.7960, How to Train a Neural Net.

Foundations of Computer Vision, Ch. 14.

Backpropagation through network layers

$$z^{(\ell)} = W^{(\ell)} h^{(\ell-1)} + b^{(\ell)}$$

$$h^{(\ell)} = \sigma\left(z^{(\ell)}\right), \qquad h^{(0)} = x$$

$$W^{(\ell)} \in \mathbb{R}^{d_\ell \times d_{\ell-1}}, \quad b^{(\ell)} \in \mathbb{R}^{d_\ell}, \quad h^{(\ell)} \in \mathbb{R}^{d_\ell}$$

Backward sensitivity

$$\delta^{(\ell)} = \frac{\partial L_i}{\partial z^{(\ell)}}$$

$$\delta^{(\ell)} = \left(W^{(\ell+1)}\right)^{\mathsf{T}} \delta^{(\ell+1)} \odot \sigma'\left(z^{(\ell)}\right)$$

Parameter gradients

$$\frac{\partial L_i}{\partial W^{(\ell)}} = \delta^{(\ell)} \left(h^{(\ell-1)}\right)^{\mathsf{T}}, \qquad \frac{\partial L_i}{\partial b^{(\ell)}} = \delta^{(\ell)}$$

Then update every layer using the mini-batch loss:

$$W^{(\ell)} \leftarrow W^{(\ell)} - \eta \nabla_{W^{(\ell)}} J_{\mathcal{B}}, \qquad \mathbf{b}^{(\ell)} \leftarrow \mathbf{b}^{(\ell)} - \eta \nabla_{\mathbf{b}^{(\ell)}} J_{\mathcal{B}}$$

Backpropagation computes gradients; the optimizer updates every layer afterward.

Further explanation: Torralba, Isola & Freeman, Foundations of Computer Vision, Chapter 14.

CTA Demo 2: Verify a gradient step

What We'll Explore

Can we explain every parameter change before trusting automatic differentiation?

1. **Predict the sign**

   For target 2 and prediction 1, should the output bias increase or decrease?

2. **Check automatic differentiation**

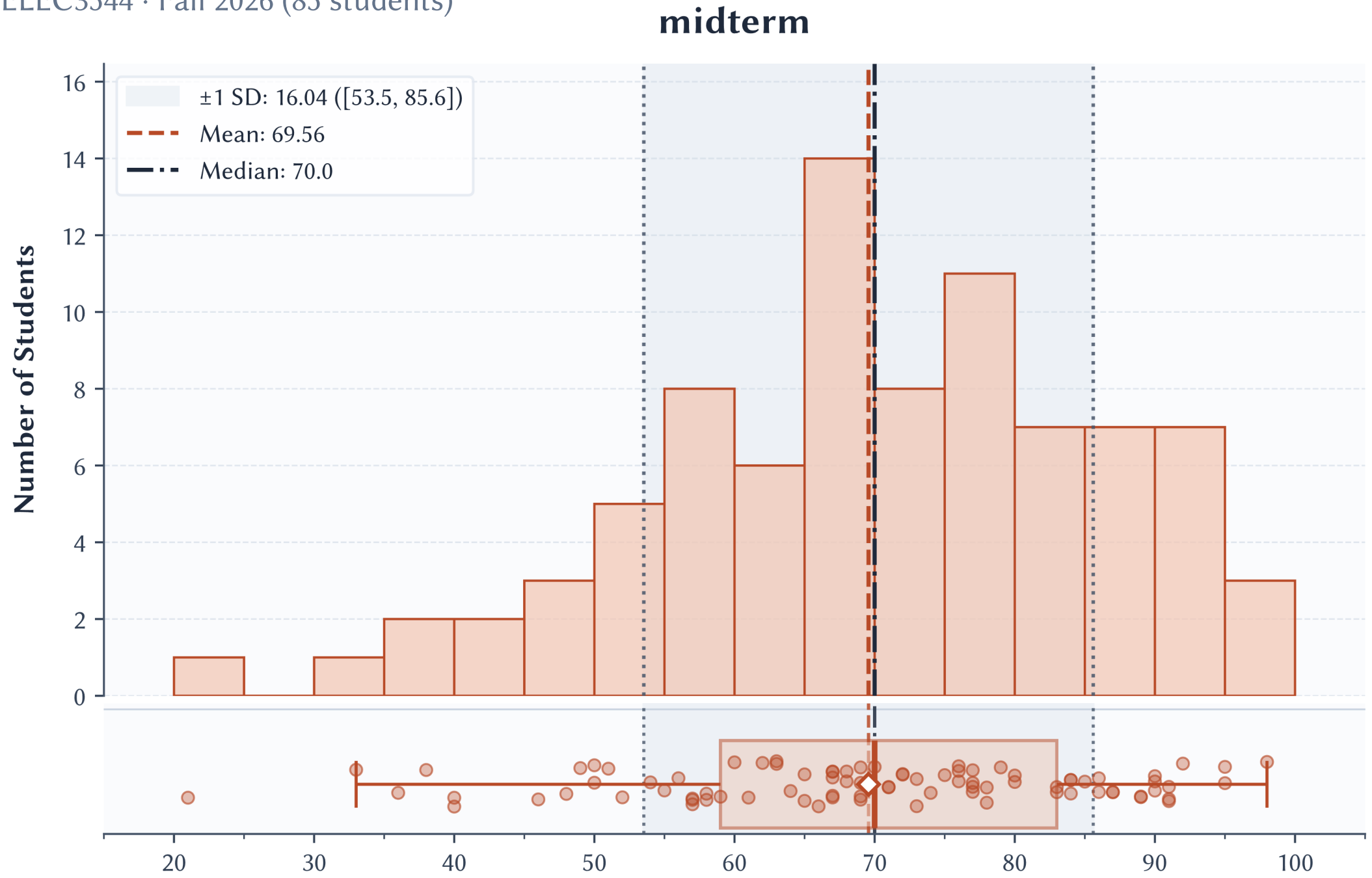
   Compare autodiff, analytic gradients, and central finite differences.

3. **Apply one simultaneous update**

   Use learning rate 0.1, then recompute the prediction and loss. Try a much larger step.

VERIFY All gradients use the old parameters. The checked step changes loss from 0.5 to 0.0098.

Collaboration pattern: reason first → verify numerically → diagnose the update


midterm
±1 SD: 16.04 ([53.5, 85.6])
Mean: 69.56
Median: 70.0
Number of Students
0
2
4
6
8
10
12
14
16
20
30
40
50
60
70
80
90
100

Homework 3, Group Project checkpoint (brief statements of progress)

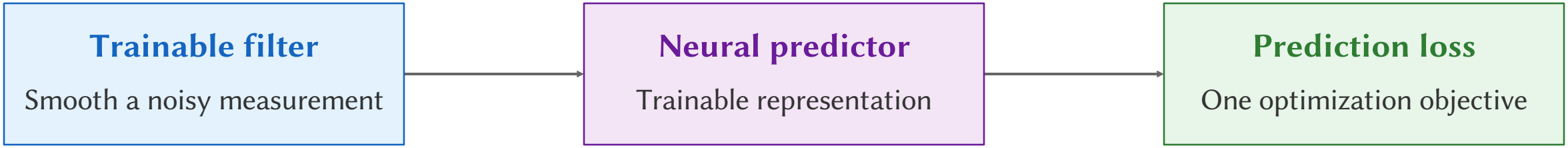
Group Project

- Presentations Nov 18 and Nov 25. ~10 min for each group
- Report and slides due Nov 17 at 11:59PM; requirements will be released

See Moodle for the confirmed schedule.

Check the current course announcements for submission deadlines, release dates, and group project details.

Differentiable programming



PyTorch pattern

```
optimizer.zero_grad()
prediction = model(transform(x))
loss = criterion(prediction, target)
loss.backward()
optimizer.step()
```

A connected gradient path

Usable backward rules
Recorded computation
Both modules in the optimizer

The task loss can tune both the filter and the predictor.

PyTorch documentation: Automatic Differentiation with torch.autograd.

The same gradient, different optimizers

A mini-batch gives a noisy estimate of the full-data gradient:

$$g_t = \nabla_\theta J_{\mathcal{B}}(\theta_t)$$

SGD

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

Follow the current batch's gradient.

Momentum

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

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

Retain useful motion across recent gradients.

Muon adds matrix orthogonalization to the momentum update (via Newton–Schulz).

Adam

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

Momentum plus a separate scale for each parameter.

Adam averages gradients and their squares; hats correct initialization bias.

The learning rate and validation behavior still matter.

Adam: Kingma & Ba, ICLR 2015, Algorithm 1.

Muon: Jordan et al. (2024)

Deriving practical optimizers

The norm defines the size of a step and changes the steepest descent direction.

Norm (step geometry)		Descent method	Practical relatives
Euclidean norm	$\|\Delta \mathbf{w}\|_2$	Gradient descent	**SGD / momentum**
Infinity norm	$\|\Delta \mathbf{w}\|_\infty$	Sign descent	**Adam***
Spectral norm	$\|\Delta W\|_{\text{op}}$	Spectral descent	**Muon**

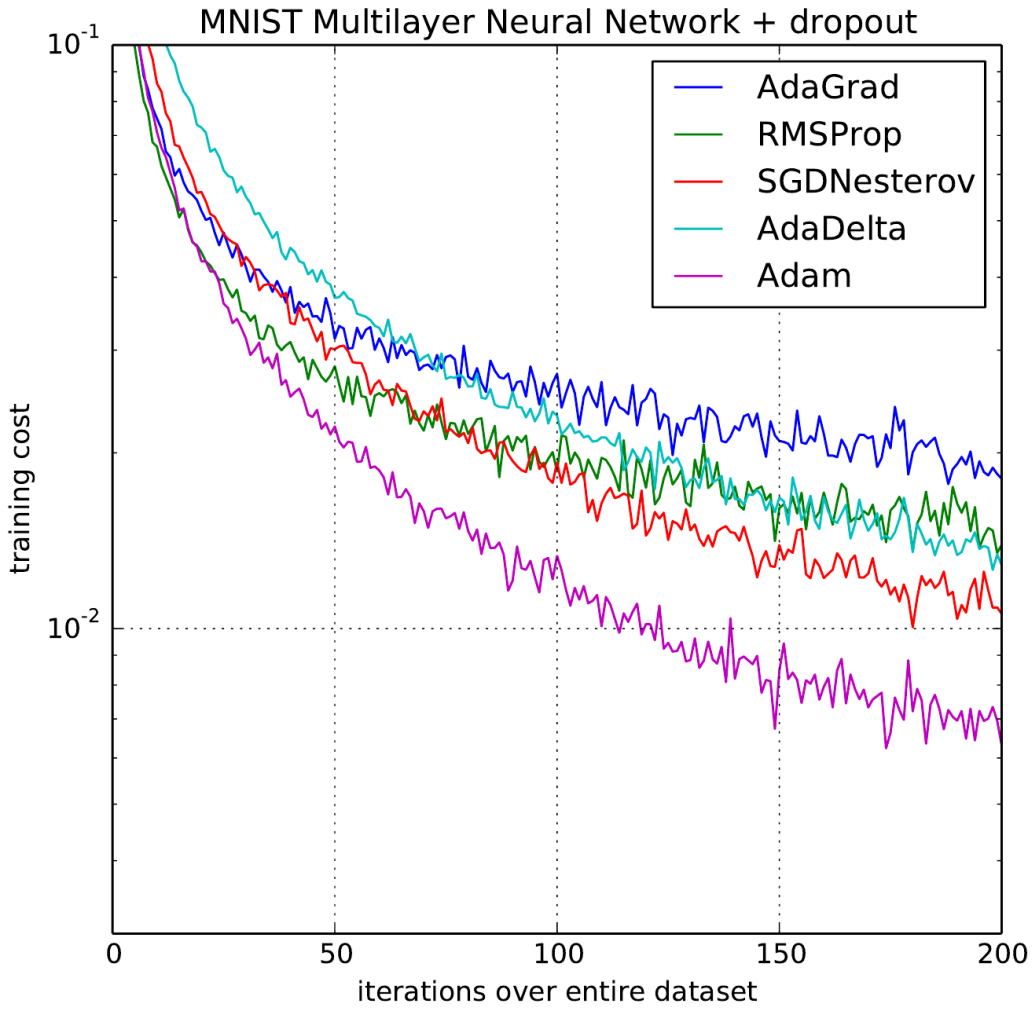
Practical optimizers add mini-batches, averaging, and scaling.

Muon = MomentUm Orthogonalized by Newton–Schulz.

* Adam uses adaptive scaling. It is related to sign descent, but is not generally identical.

Adapted: P. Isola, MIT 6.7960 (2026) Bernstein & Newhouse (2024) Muon: Jordan et al. (2024)

Optimizer choice changes training progress



One published experiment

MNIST classification
Two 1,000-unit ReLU hidden layers
Dropout; mini-batches of 128

Read the axes first

Horizontal: passes over the dataset
Vertical: training cost, on a log scale

Adam reduces training cost faster here. Validation and test performance require separate checks.

Kingma & Ba, Adam: A Method for Stochastic Optimization, ICLR 2015, Fig. 2(a). Original experimental curves.

Initialization and normalization control signal scale

He initialization for ReLU

$$\mathrm{Var}(W_{jk}) = \frac{2}{d_{\text{in}}}, \qquad \mathbb{E}[W_{jk}] = 0$$

Fan-in is the number of inputs to a unit.

Operation	What it controls	When it is determined
Input standardization	Scale of measured inputs	Fit on training data
Weight initialization	Initial activation / gradient scale	Before parameter updates
Normalization layers	Intermediate activation scale	During training and evaluation

Batch normalization changes its statistical behavior between training and evaluation.

He et al., ICCV 2015 (initialization).

Ioffe & Szegedy, ICML 2015 (batch normalization).

Generalization and regularization

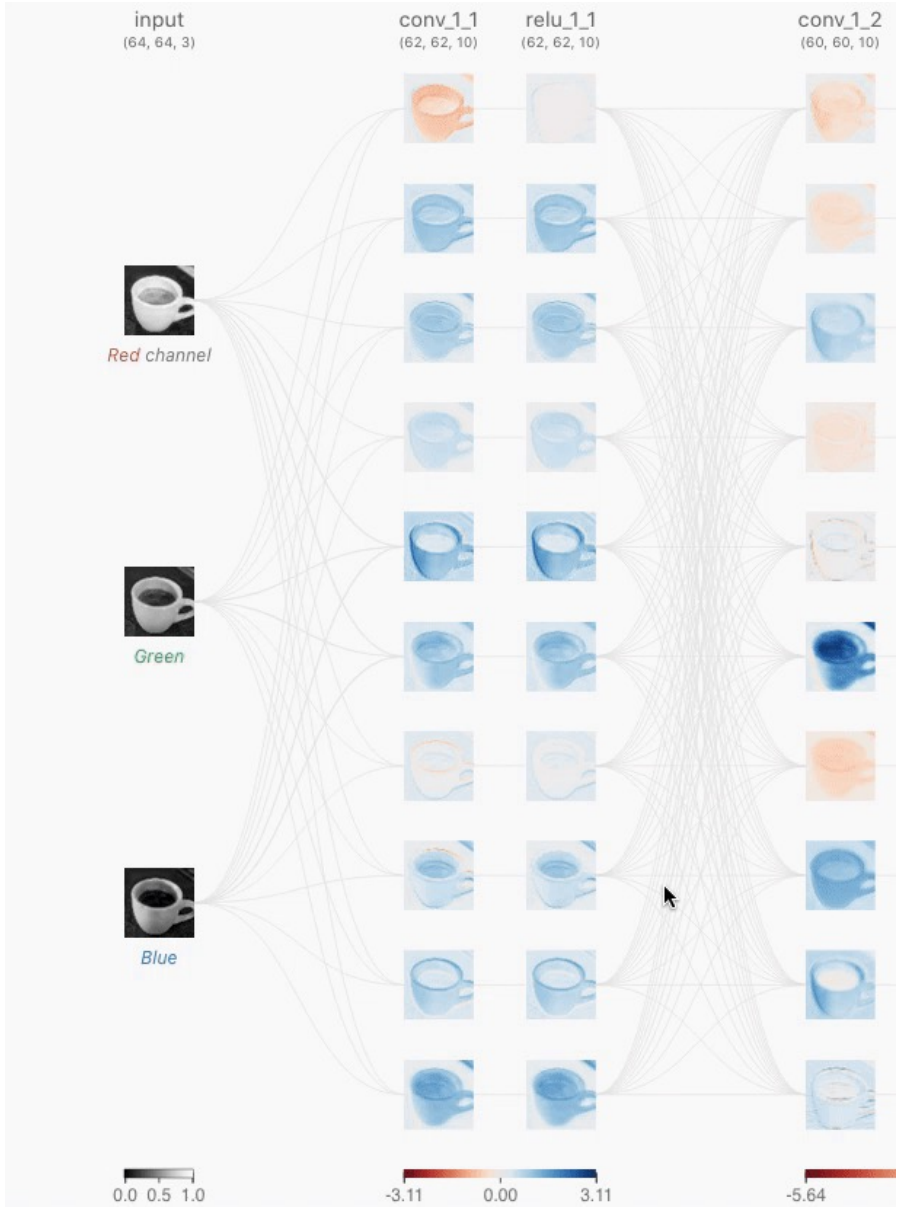
Method	What changes
Weight decay	The training objective / parameter update
Early stopping	The checkpoint selected using validation data
Label-preserving augmentation	The training examples seen by the model
Dropout	The activations used during training

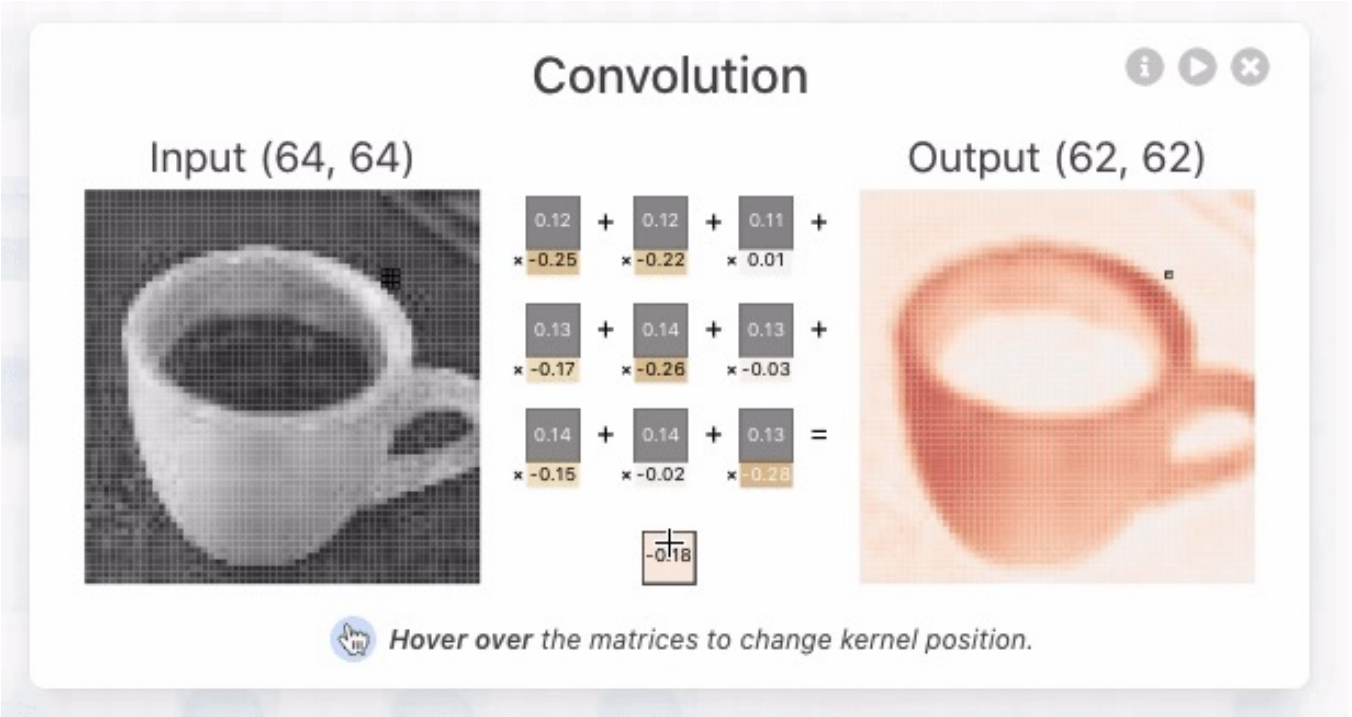
Checkpoint question

Is a failure due to model capacity, training, or generalization?

Next: exploit image structure with convolutional networks.

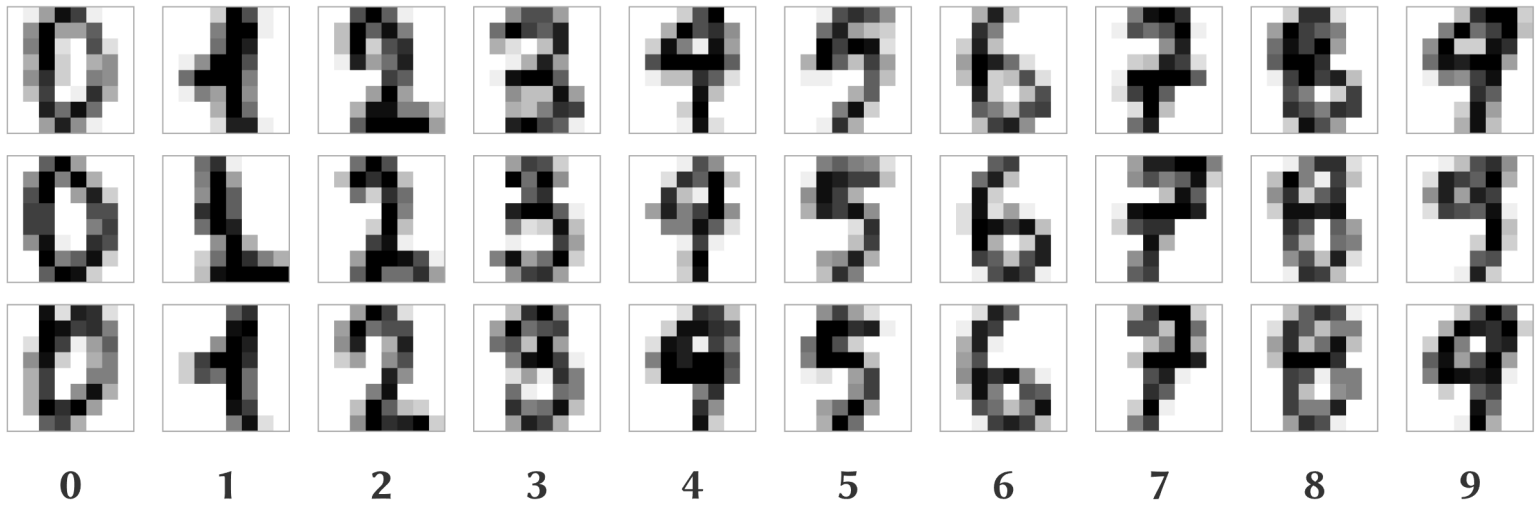
Convolutional Nets





https://poloclub.github.io/cnn-explainer/

Images make the representation problem visible



Data and task

8 × 8 grayscale pixels
10 digit classes

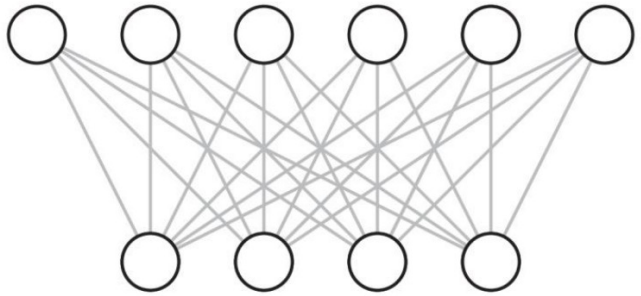
3,060 training
763 validation
1,797 official test

Nearby pixels form patterns. A useful model can exploit that spatial structure.

Convolution: locality and shared weights

Restrict the connections, then reuse the same coefficients.

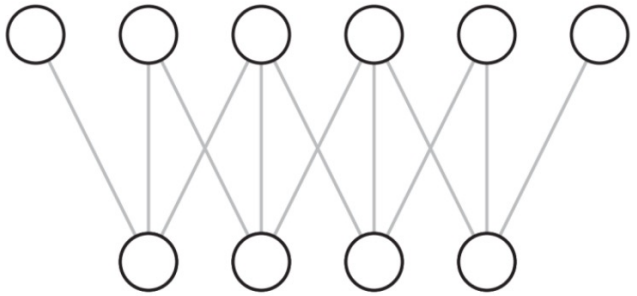
Fully connected



24 weights

Every output uses every input.

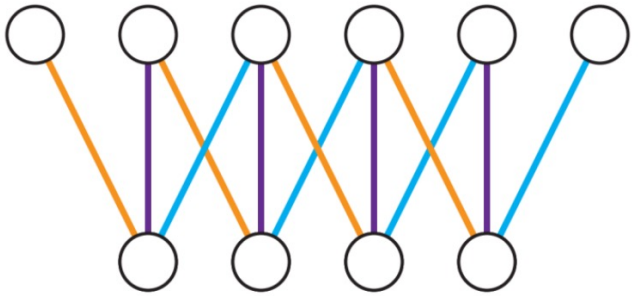
Locally connected



12 weights

Each output uses a nearby window.

Convolution



3 shared weights

The same filter is used at every position.

Convolution combines locality and shared weights. Biases are omitted here.

Adapted from K. He, MIT 6.7960, ConvNets (2026). Original connectivity graphs, rearranged.

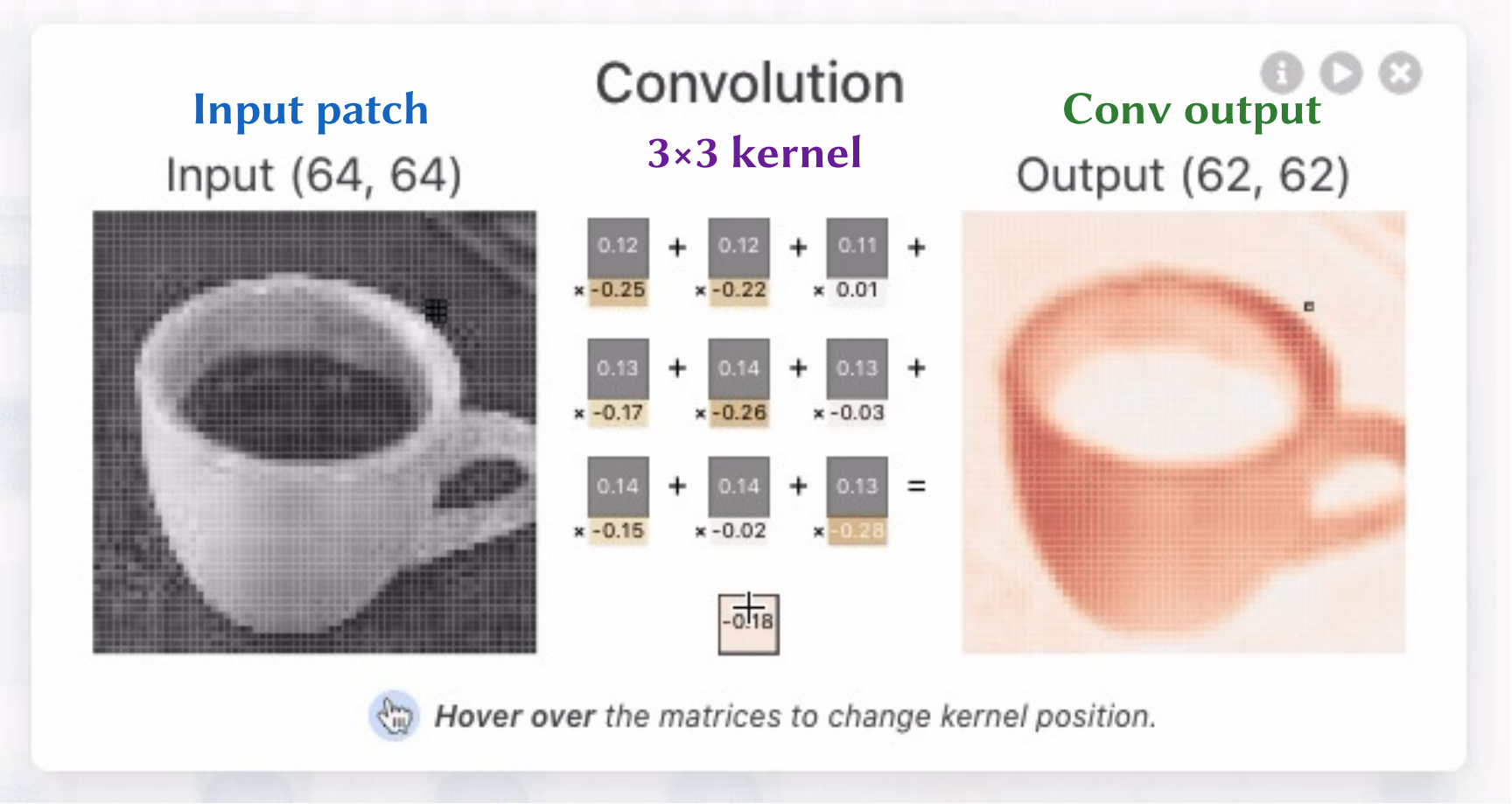
One convolution output, step by step

Input patch				Shared filter / Kernel				Elementwise products		
1	0	2	×	1	0	−1	=	1	0	−2
2	1	0		1	0	−1		2	0	0
0	1	1		1	0	−1		0	0	−1

Sum = 0 **Bias = 0.5** **ReLU output = 0.5**

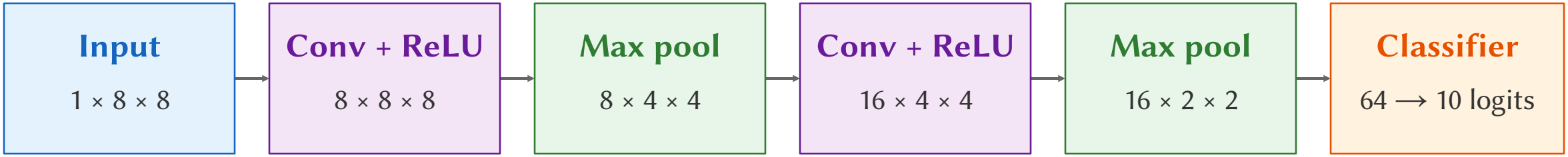
Teaching filter, not learned. Cross-correlation convention: no kernel flip.

One convolution output, step by step



https://poloclub.github.io/cnn-explainer/

A CNN combines convolution, activation, and pooling



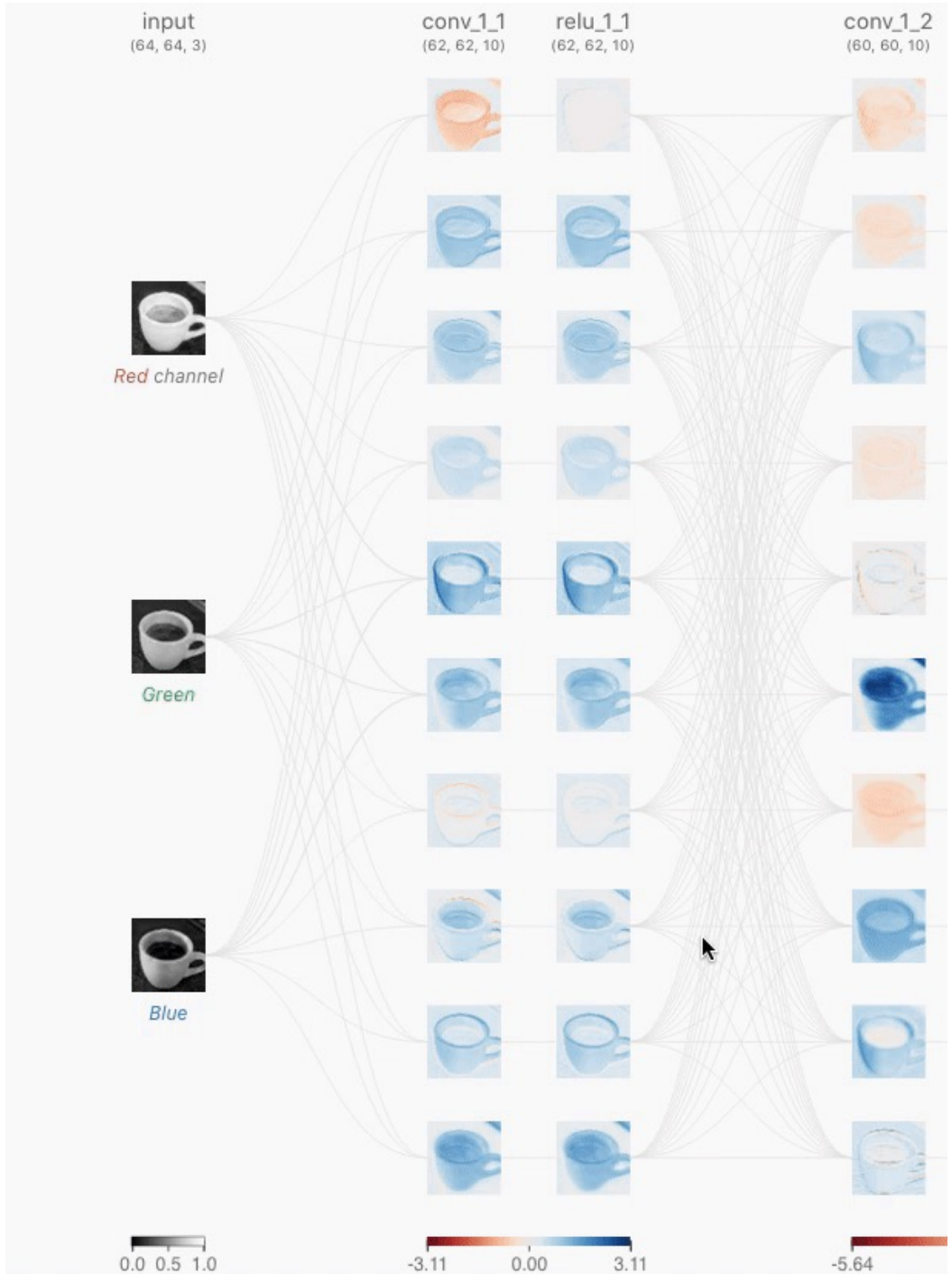
What max pooling does

$$\begin{bmatrix} 1 & 4 \\ 2 & 0 \end{bmatrix} \xrightarrow{\text{max pool}} 4$$

2 × 2 window, stride 2
Apply separately to each channel
No learned weights

The classification loss updates the filters and the final classifier together.

1,898 parameters. Both convolutions use 3 × 3 filters, stride 1, padding 1.



https://poloclub.github.io/cnn-explainer/

Tensor shapes and parameter counts

$$N_{\text{parameters}} = C_{\text{out}}\left(C_{\text{in}}K^2 + 1\right)$$

$$H_{\text{out}} = \left\lfloor \frac{H + 2P - K}{S} \right\rfloor + 1$$

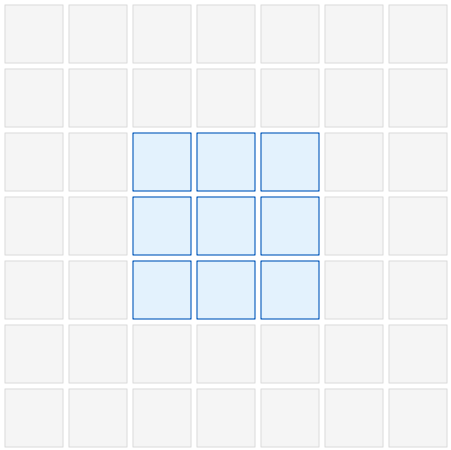
Square kernels, dilation 1, symmetric padding.

Layer in our CNN	Input ⟶ output	Trainable parameters
First 3 × 3 convolution	1 × 8 × 8 ⟶ 8 × 8 × 8	8 × (1 × 9 + 1) = 80
Second 3 × 3 convolution	8 × 4 × 4 ⟶ 16 × 4 × 4	16 × (8 × 9 + 1) = 1,168
Final affine classifier	64 ⟶ 10	10 × (64 + 1) = 650

Total: 1,898 parameters. ReLU and max pooling add no trainable parameters.

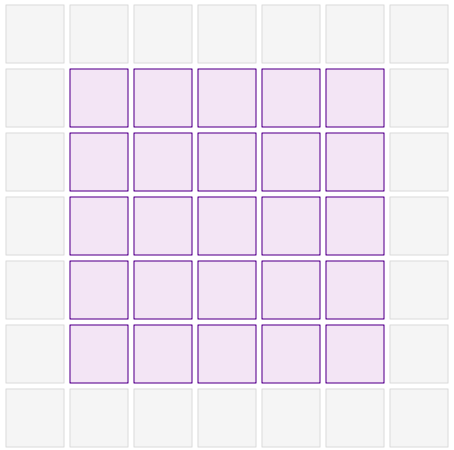
Depth expands the receptive field

1 convolution



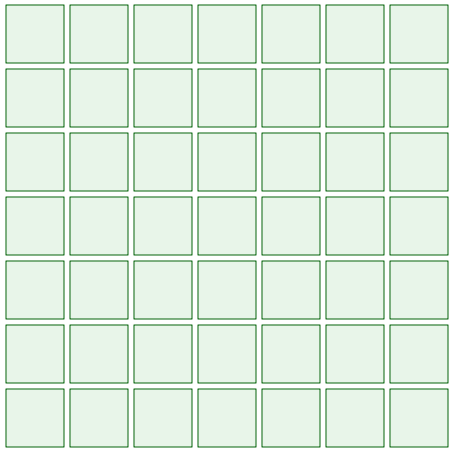
3 × 3 input region

2 convolutions



5 × 5 input region

3 convolutions



7 × 7 input region

3 × 3 kernels, stride 1, dilation 1, no pooling

Consecutive 3 × 3 layers combine information over progressively larger regions.

CTA Demo 3: Does image structure help?

What We'll Explore

How do linear, multilayer, and convolutional models compare on the same real images?

1 **Predict and compare**

Inspect the parameter counts. Predict which model will generalize to unseen writers.

2 **Run or load the fixed experiment**

Use the same split and 3 initialization seeds. Inspect validation loss and final test results.

3 **Inspect what the score hides**

Examine learned filters, feature maps, errors, and the separate one-pixel shift test.

VERIFY Select checkpoints by validation loss. Keep official test images out of all training choices.

Collaboration pattern: state a hypothesis → compare fairly → inspect failures

Measured comparison on unseen writers

Model	Parameters	Mean test accuracy	Across 3 seeds
Linear softmax	650	94.03%	93.99–94.05%
MLP: 64 → 64 → 32 → 10	6,570	95.86%	95.44–96.27%
CNN: 2 convolution layers	1,898	96.62%	96.05–97.16%

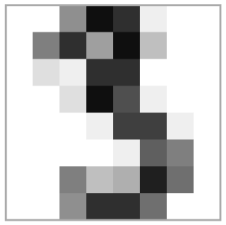
3,060 training / 763 validation / 1,797 official test images
Same data split, 80 epochs, Adam, and 3 fixed seeds.

The CNN uses spatial structure to improve this result with fewer parameters than the MLP.

Seed range is variability across runs, not a confidence interval or an exhaustive benchmark.

Learned filters and input-dependent activations

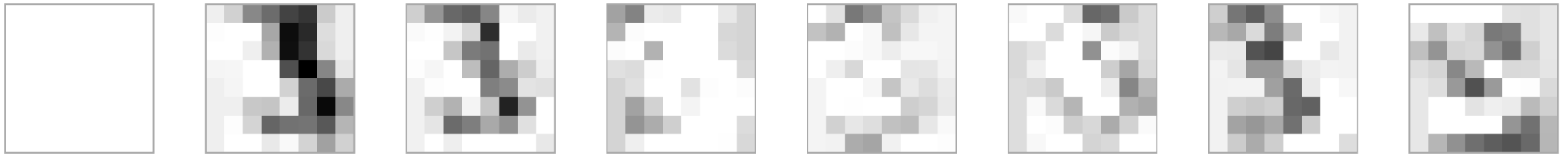
Input



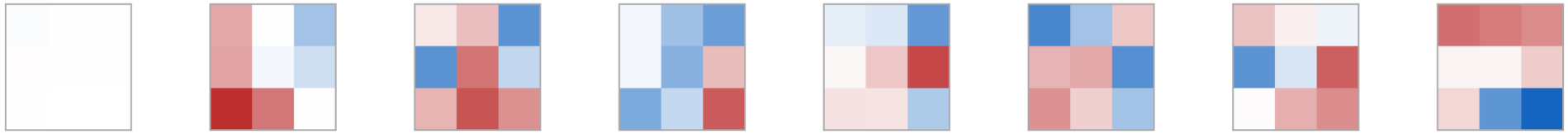
Digit 3

ReLU feature maps

Channel 1 Channel 2 Channel 3 Channel 4 Channel 5 Channel 6 Channel 7 Channel 8



Learned 3 × 3 kernels



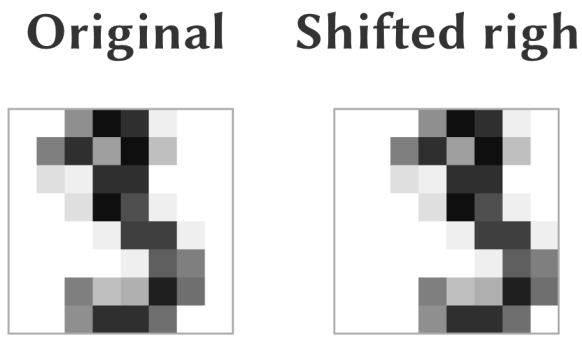
Same input, eight different learned responses

Shared scales across channels; blue/red kernel weights are negative/positive.

Filters are shared parameters. Feature maps depend on the particular image.

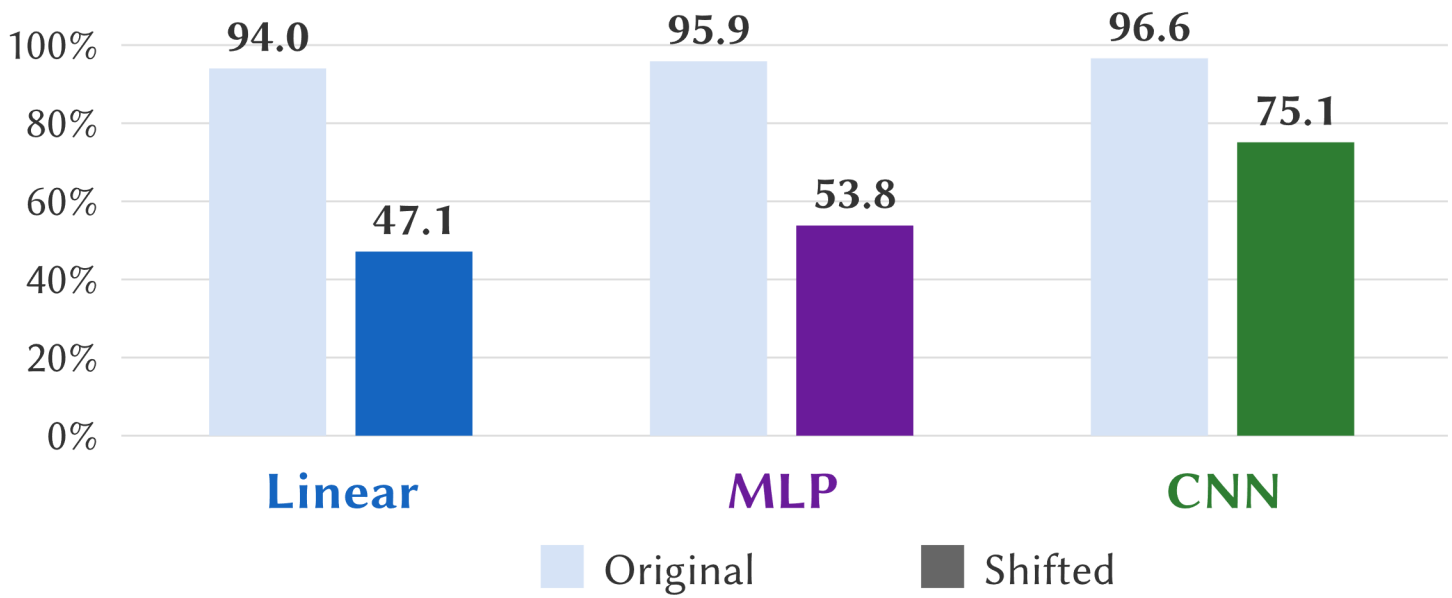
Equivariance and robustness to shifts

Equivariance shifts the feature map. Invariance preserves the final prediction.



One pixel; zero fill

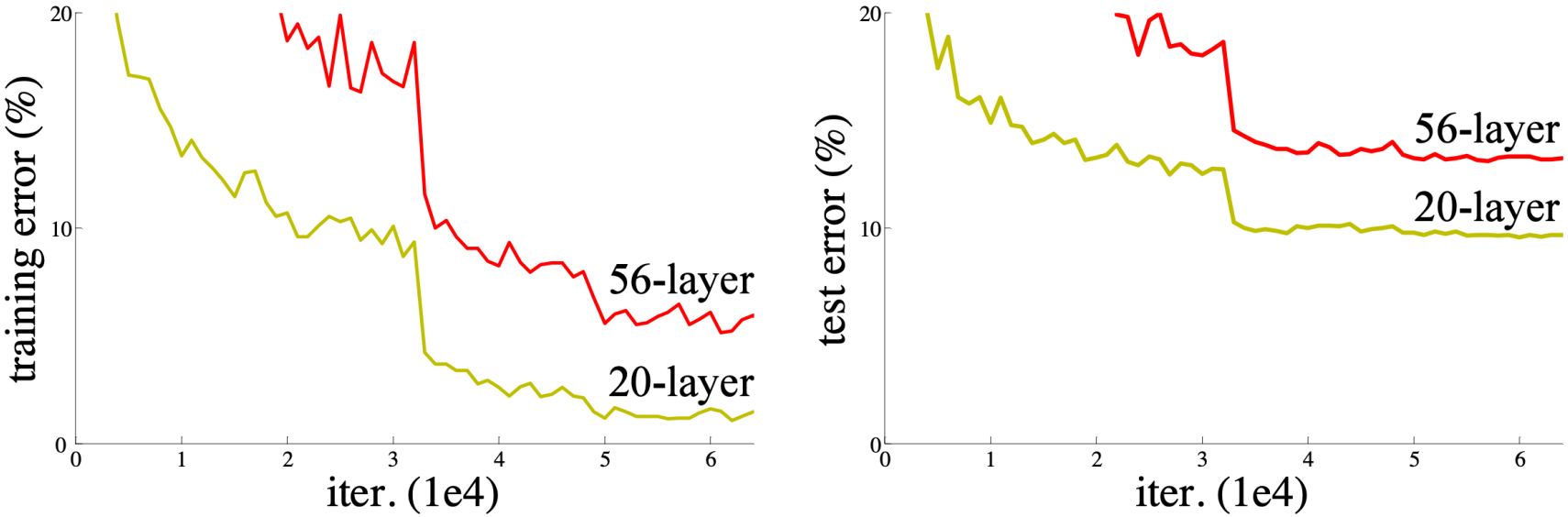
Right edge is cropped



Exploratory stress test: shift right by 1 pixel, zero fill, crop the right edge. No retraining.

Deeper networks can be harder to optimize

CIFAR-10: measured error for plain 20-layer and 56-layer networks



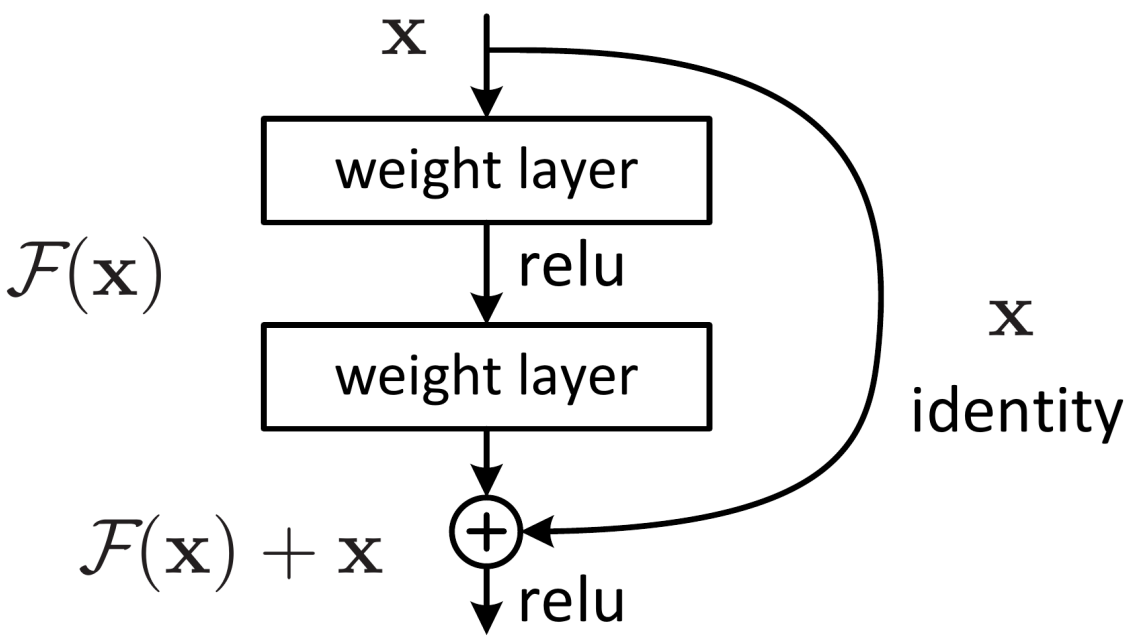
Add blocks that copy their input: the deeper model could preserve the same predictions.

The deeper model has higher training error: an optimization problem.

Original curves: He, Zhang, Ren & Sun, Deep Residual Learning for Image Recognition, CVPR 2016, Fig. 1.

Residual learning in the original ResNet block

Original post-activation block



The original block applies ReLU after the sum.

At the addition, before ReLU

$$\mathbf{u} = \mathbf{x} + F(\mathbf{x}), \qquad \mathbf{y} = \mathrm{ReLU}(\mathbf{u})$$

A zero residual preserves the input at the sum.

$$F(\mathbf{x}) = \mathbf{0} \quad \Longrightarrow \quad \mathbf{u} = \mathbf{x}$$

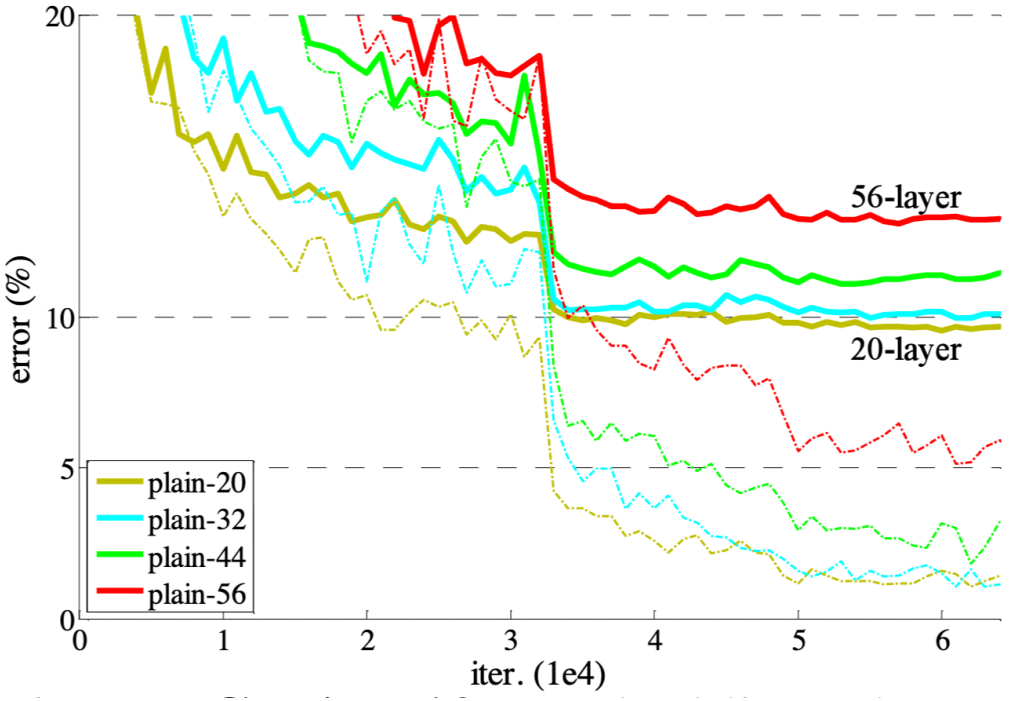
Scalar intuition for the backward path

$$\frac{du}{dx} = 1 + F'(x)$$

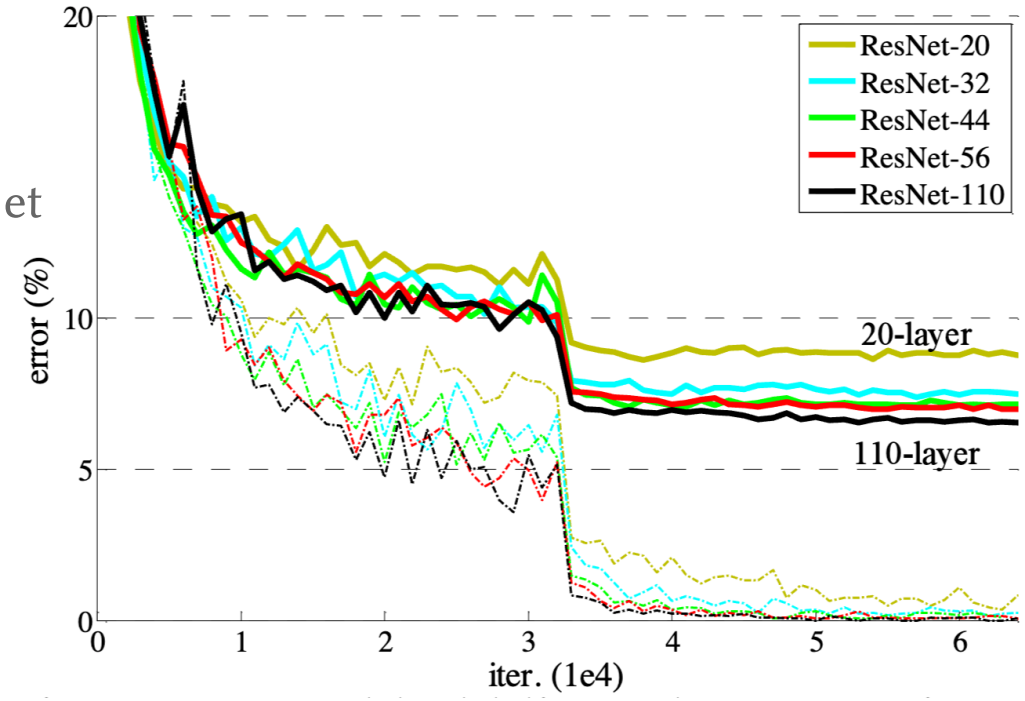
The identity term helps optimization; it does not guarantee stable gradients.

Residual learning in the original ResNet block

Learn minimal (via residuals), go deeper!

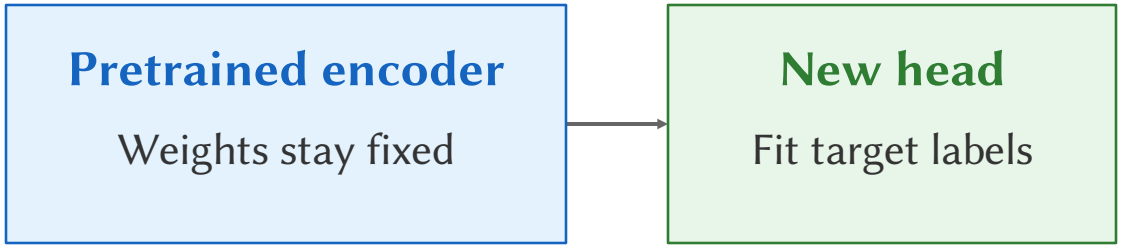


After ResNet



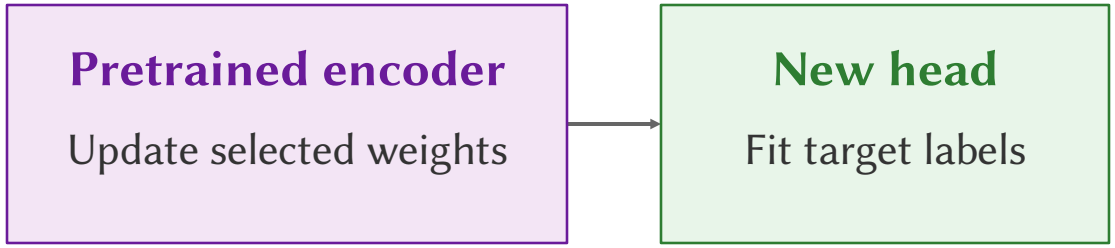
Transfer learning reuses representations

Frozen representation



Less training and fewer fitted parameters.
Useful when labels are scarce.

Fine-tuning



More task adaptation and more risk of overfitting or forgetting.

L7 used pretrained embeddings. L9 explains how larger pretrained models are built and used.

Architecture choice follows the data and task

Data / constraint	Useful starting point	Evidence to check
Small tabular data	Linear or tree baseline	Validation gain over the baseline
Images and spatial signals	CNN candidate	Accuracy, spatial shifts, acquisition changes
Few labels, relevant pretrained model	Frozen features, then fine-tuning	Transfer benefit and data leakage
Sequences and long-range context	Context-aware architectures in L9	Order, context limits, task-specific errors

A useful representation must justify its data, computation, and deployment costs.

AlphaGo: a public turning point for deep learning

March 2016: 4–1 against Lee Sedol. More than 200 million viewers.



AlphaGo vs Lee Sedol match, Seoul (2016)

Why it mattered

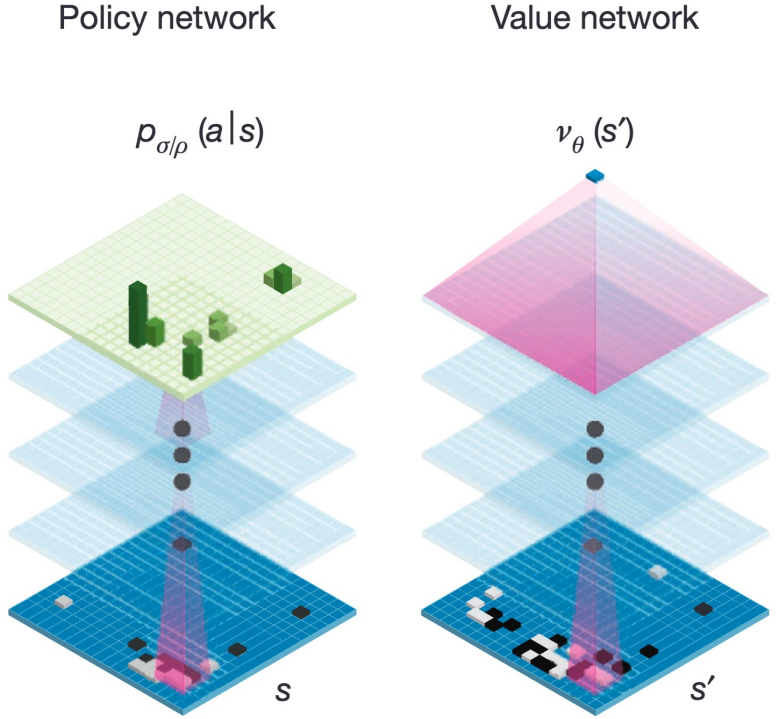
Made strategic AI visible worldwide.
Raised ambitions in research and industry.

Earlier: AlexNet (2012), Deep Q-network (2015).
Later: AlphaZero (2017), AlphaFold 2 (2020).

Data science perspective

Collect Huaman expert and self-play positions

Model Policy and value networks; Tree search and self plays



The goal is winning games. Predicting expert moves is an intermediate objective.

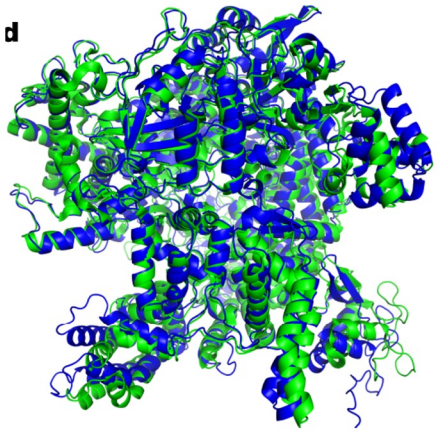
AlphaFold: predicting protein structure

Input: amino-acid sequence. Output: a protein's 3D structure.

2024 Nobel Prize in Chemistry: Demis Hassabis and John Jumper

Blind accuracy + complex structure

RNA polymerase
2,180 residues



Post-CASP example: T1044
$RMSD_{95}$ = 2.2 Å

Blind CASP14 test
87 protein domains

Median backbone error (Å)
Lower is better

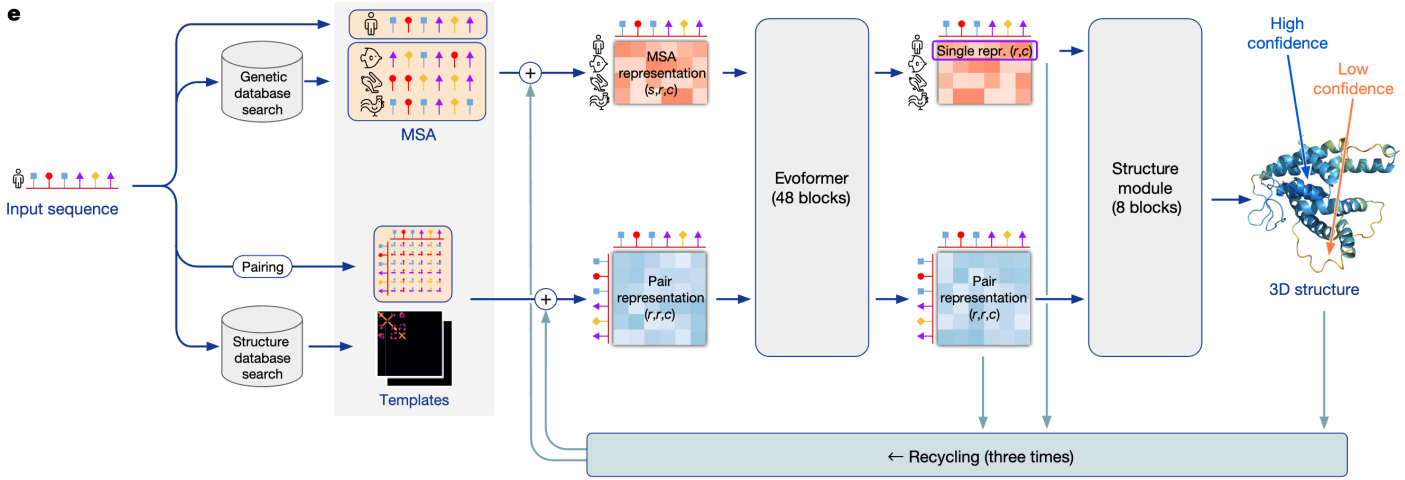


Cα RMSD at 95% coverage
Unpublished structures
kept hidden for evaluation

Data science perspective

Model

Multi-sequence alignments (MSAs), Evoformer (attention-based, with residual connections - ResNet)



Key insight: learn sequence-to-structure geometry end to end; use confidence to guide experiments.

Representation, optimization, and generalization

1 Representation

What can nonlinear layers represent, and how does depth help?

2 Optimization

How do backpropagation, the optimizer, and residual connections help us train?

3 Generalization

What evidence shows that learned representations help on new data?

Next: foundation models, attention, pretraining, and agentic data-science workflows.

Open questions to explore with CTA

Ask CTA for competing explanations, then design an experiment to distinguish them.

1 **If a shallow network is universal, why go deep?**

Compare shallow and deep models on a compositional function.
Separate approximation accuracy, parameter efficiency, and trainability.

2 **Why can a network memorize and still generalize?**

Compare learning with true and shuffled labels. What selects the solution?
Explore the roles of data structure, SGD, and regularization.

3 **Why can an identity shortcut make a deep network trainable?**

Compare plain and residual networks. Trace the chain rule through both.
When does the shortcut help, and what can it fail to fix?

4 **Has the network learned the concept, or a shortcut?**

Change image backgrounds or shift digits while preserving their labels.
What evidence distinguishes useful representations from spurious cues?

VERIFY Which claim is a theorem, an empirical finding, or still a hypothesis?

Start from the lecture. Ask what changes when its assumptions fail.

References: course, books, and demos

Course and books

MIT 6.7960 Deep Learning

2026 course materials 2025 archive

Understanding Deep Learning

Simon J. D. Prince (2023)
Intuition, worked examples, and notebooks.

Deep Learning

Goodfellow, Bengio & Courville (2016)
Core theory, optimization, and architectures.

Foundations of Computer Vision

Torralba, Isola & Freeman (2024)
Visual explanations of networks and CNNs.

Interactive visualizations

CNN Explainer

Trace convolution, ReLU, pooling,
and feature maps layer by layer.

poloclub.github.io/cnn-explainer/

TensorFlow Playground

Change depth, width, activation, and
learning rate. Compare training and test loss.

playground.tensorflow.org

Grand Tour (Distill, 2020)

See learned representations evolve
across layers and during training.

distill.pub/2020/grand-tour/

References: foundational papers

Foundations and training

Rosenblatt (1958)

The perceptron: a probabilistic model for information storage and organization in the brain

Rumelhart, Hinton & Williams (1986)

Learning representations by back-propagating errors

Cybenko (1989)

Approximation by superpositions of a sigmoidal function

Baydin et al. (2018)

Automatic Differentiation in Machine Learning: a Survey

Vision and optimization

LeCun et al. (1998)

Gradient-Based Learning Applied to Document Recognition

Krizhevsky, Sutskever & Hinton (2012)

ImageNet Classification with Deep Convolutional Neural Networks

Kingma & Ba (2015)

Adam: A Method for Stochastic Optimization

He et al. (2016)

Deep Residual Learning for Image Recognition

Questions?

CONTACT INFORMATION



yangliu.mit.edu

Instructor: Prof. Yang Liu

Email: yliu12@hku.hk

Office Hours: 5:00 - 5:50 PM, Thursdays

Location: CB-714, Chow Yei Ching Building

Let me know if you want remote!

🤖 **CTA is your 24/7 interactive and responsive learning partner!**

TA contacts

Tuesday 4-5PM

Yushi Du:
u3012725@connect.hku.hk
https://us04web.zoom.us/j/78561597481

Zhengyang Hu:
u3010250@connect.hku.hk
https://hku.zoom.us/j/9021481973

Thursday 3-4PM

Jun Dai:
jundai332@connect.hku.hk
https://rochester.zoom.us/j/97202452759

Lulu Cai:
u303011209@connect.hku.hk
https://hku.zoom.us/j/5010926726

TA office hours on Zoom! Two slots (1 hour each) on Tuesdays and Thursdays