# Neural Networks: Theory, Training, and Architectures

## Deep Learning and Reinforcement Learning — Lecture 4

Wenhao Yang

# Deep learning applications

# Vision: locating objects in an image




Image $\longrightarrow$ object labels and locations

Application: Ren et al. (2015). Image and detection boxes are illustrative.

# Speech: turning sound into text



Audio $\longrightarrow$ written transcript

Application: Amodei et al. (2016). Original illustrative image.

# Vision and language: describing an image



Image $\longrightarrow$ "A child and a dog play in the rain."

Application: Xu et al. (2015). Image and example caption are illustrative.

# Generation: drawing from a text prompt

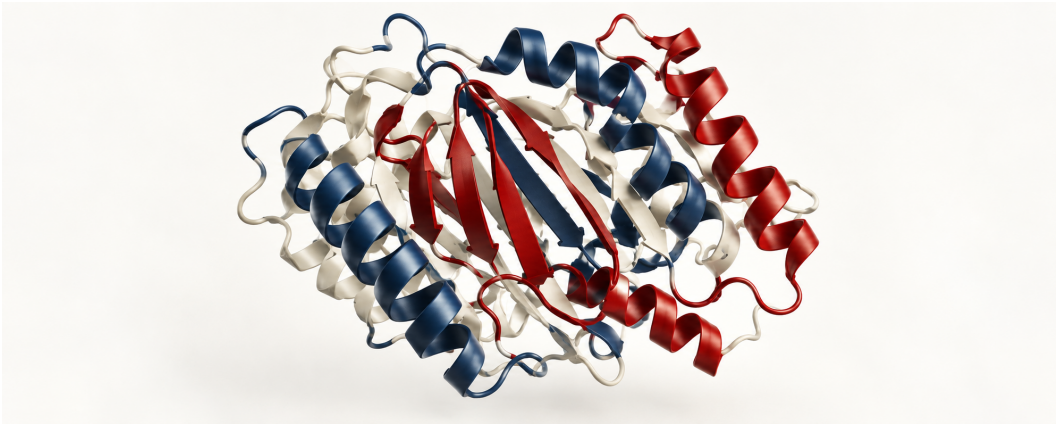


"A lighthouse on a floating island above the clouds at sunset."

Text prompt $\longrightarrow$ new image

Application: Ramesh et al. (2021). Illustration generated for this lecture, not by the cited model.

# Science: predicting protein structure



Protein sequence data $\longrightarrow$ 3D structure

Application: Jumper et al. (2021). Schematic rendering, not an AlphaFold prediction.

# Decision making: choosing a Go move



Board position $\longrightarrow$ move

AlphaGo combines neural networks and tree search: Silver et al. (2016). Original illustrative image.

# Start with one task: classify a new catalog photo



new product photo $x$

$\longrightarrow$

network scores
$a_\theta(x)$, then softmax

$\longrightarrow$

| | |
|---|---|
| **coat** | **0.84** |
| shirt | 0.11 |
| shoe | 0.05 |

Illustrative class probabilities

Teaching sequence inspired by the IP Paris Deep Learning lecture slides (CC BY 4.0).

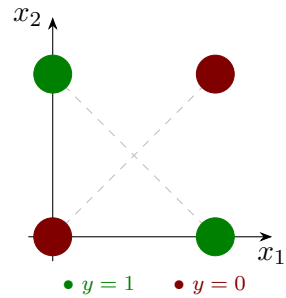
# Feedforward networks

# Where does a linear rule fail?

The positive and negative XOR vertices have intersecting convex hulls, so a linear classifier cannot separate them.

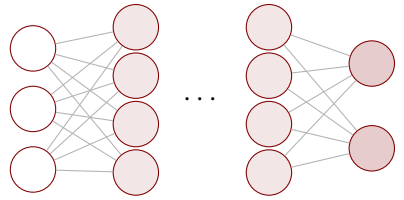
For binary inputs, a two-unit ReLU network computes XOR exactly:

$$f(x_1, x_2) = \rho(x_1 - x_2) + \rho(x_2 - x_1) = |x_1 - x_2|.$$



# Stack nonlinear features to build a classifier

$$
\begin{aligned}
h^0 &= x, \\
z^\ell &= W^\ell h^{\ell-1} + b^\ell, \\
h^\ell &= \phi_\ell(z^\ell), \quad \ell < L, \\
a_\theta(x) &= z^L.
\end{aligned}
$$




- Hidden widths: $d_1, \ldots, d_{L-1}$
- Output scores: $a_\theta(x) \in \mathbb{R}^K$
- Parameters: $P = \sum_{\ell=1}^{L} d_\ell(d_{\ell-1} + 1)$

Goodfellow, Bengio, and Courville (2016), Ch. 6.

# Why use nonlinear activations?

Without hidden activations, equivalently $\phi(t) = t$, two layers collapse:

$$\begin{aligned} a_\theta(x) &= W^2(W^1 x + b^1) + b^2 \\ &= \underbrace{W^2 W^1}_{\widetilde{W}} x + \underbrace{W^2 b^1 + b^2}_{\widetilde{b}}. \end{aligned}$$

At any depth, affine layers plus a linear classifier still cannot separate XOR.

**Nonlinear activations allow nonlinear hidden features:**

$$h^1 = \phi(W^1 x + b^1), \qquad a_\theta(x) = W^2 h^1 + b^2.$$

ReLU, $\rho(t) = \max\{0, t\}$, has slope $0$ for $t < 0$ and $1$ for $t > 0$. Combining these units can represent XOR, as shown earlier.

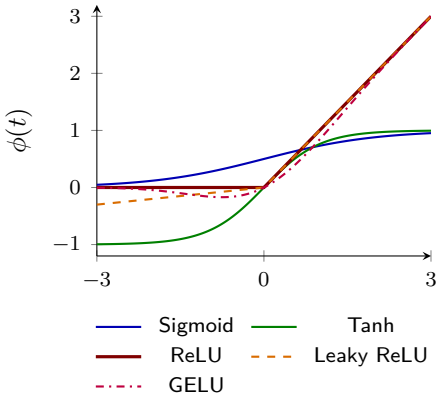
Goodfellow, Bengio, and Courville (2016), Secs. 6.1, 6.3.

# Activation functions

The hidden layer applies an activation coordinatewise: $h_j = \phi(z_j)$.

| Activation | $\phi(t)$ |
|---|---|
| Sigmoid | $(1 + e^{-t})^{-1}$ |
| Tanh | $\tanh(t)$ |
| ReLU | $\max\{0, t\}$ |
| Leaky ReLU | $\max\{\alpha t, t\}$ |
| GELU | $t\Phi(t)$ |

$0 < \alpha < 1$; the plot uses $\alpha = 0.1$.
$\Phi$ is the standard normal CDF.




Goodfellow, Bengio, and Courville (2016), Sec. 6.3; GELU: Hendrycks and Gimpel (2016).

# Approximation theory

# Universal approximation

## Continuous nonpolynomial activations

Let $\phi : \mathbb{R} \to \mathbb{R}$ be continuous and not a polynomial. For every compact $K \subset \mathbb{R}^d$, $f \in C(K)$, and $\varepsilon > 0$, there exist a finite width $m$ and parameters $c, a_j, b_j \in \mathbb{R}$, $w_j \in \mathbb{R}^d$ such that

$$\sup_{x \in K} \left| f(x) - c - \sum_{j=1}^{m} a_j \phi(w_j^\top x + b_j) \right| < \varepsilon.$$

Leshno et al. (1993), Theorem 1 (continuous activations) and Definition 5, p. 863 (PDF p. 3).

# Constructive ReLU approximation in one dimension

Let $f : [0,1] \to \mathbb{R}$ be $M$-Lipschitz. For an integer $m \geq 1$, set

$$x_j = \frac{j}{m}, \qquad s_j = m(f(x_{j+1}) - f(x_j)), \quad 0 \leq j < m.$$

A one-hidden-layer ReLU network represents the linear interpolant:

$$\boxed{I_m f(x) = f(0) + s_0 \rho(x) + \sum_{j=1}^{m-1} (s_j - s_{j-1}) \rho(x - x_j).}$$

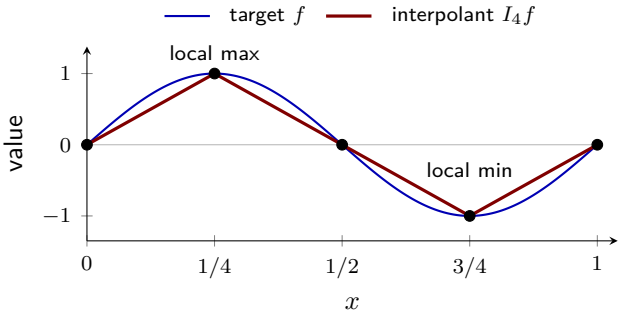
On $(x_j, x_{j+1})$, its slope is

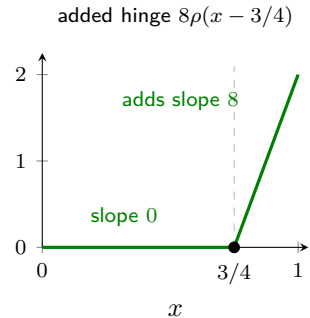
$$s_0 + \sum_{r=1}^{j} (s_r - s_{r-1}) = s_j.$$

Its value at zero is $f(0)$, so it passes through every knot $(x_j, f(x_j))$.

# ReLU hinges build the interpolant

For $f(x) = \sin(2\pi x)$ and $m = 4$, use the knots $x_j = j/4$ on $[0, 1]$.




$$I_4 f(x) = \underbrace{4\rho(x)}_{s_0=4} \underbrace{-8\rho(x-\tfrac{1}{4})}_{s_1-s_0=-8} \underbrace{+8\rho(x-\tfrac{3}{4})}_{s_3-s_2=8}, \qquad 0 \le x \le 1.$$

The segment slopes are $(4, -4, -4, 4)$. At $x = 1/2$, the slope change is zero.

# Interpolation error bound

Suppose $|f(x) - f(y)| \le M|x - y|$ on $[0, 1]$. On an interval $[x_j, x_{j+1}]$ of length $h = 1/m$, write $x = x_j + t$.

$$I_m f(x) = \left(1 - \frac{t}{h}\right) f(x_j) + \frac{t}{h} f(x_{j+1}).$$

$$\begin{aligned} |f(x) - I_m f(x)| &\le M \left[\left(1 - \frac{t}{h}\right) t + \frac{t}{h}(h - t)\right] \\ &= \frac{2Mt(h - t)}{h} \le \boxed{\frac{M}{2m}}. \end{aligned}$$

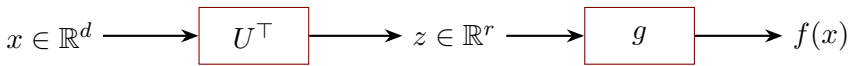
In dimension $d$, a grid with $m$ cells per coordinate has $m^d$ cells. Useful high-dimensional rates require structure beyond continuity.

# Low effective dimension

Assume the target depends on $r$ linear combinations of its $d$ inputs:

$$f(x) = g(U^\top x), \qquad U \in \mathbb{R}^{d\times r}, \quad U^\top U = I_r, \quad r \ll d.$$

linear projection — nonlinear function




$z_j = u_j^\top x$: all inputs may contribute

**Low-rank projection structure**

$$P = UU^\top, \qquad \operatorname{rank}(P) = r.$$

Nonlinear approximation: $r$ variables.
Projection storage: $dr$ coefficients.

**Coordinate selection is a special case**

$$U = [e_1, e_7] \quad \Longrightarrow \quad z = (x_1, x_7).$$

This recovers $f(x) = g(x_1, x_7)$.
$U$ must be known or learned.

On a compact input domain $K$, let $\mathcal{Z} = U^\top K$ and $\widehat{f}(x) = \widehat{g}(U^\top x)$. Then

$$\|f - \widehat{f}\|_{\infty,K} = \|g - \widehat{g}\|_{\infty,\mathcal{Z}}.$$

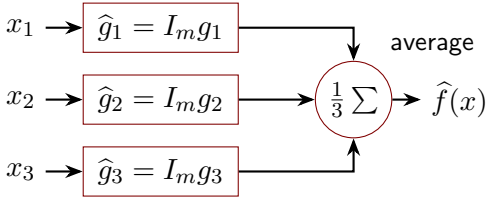
Constantine, Eftekhari, Hokanson, and Ward (2017), *A near-stationary subspace for ridge approximation.*

# Additive structure

All coordinates may matter, with separate contributions and no interaction terms:

$$f(x) = \frac{1}{d}\sum_{j=1}^{d} g_j(x_j), \qquad \widehat{g}_j = I_m g_j, \qquad x \in [0,1]^d.$$

Three 1D ReLU interpolants ($d = 3$)




**Example components**

$$g_1(t) = \sin(2\pi t),$$
$$g_2(t) = t^2,$$
$$g_3(t) = |t - \tfrac{1}{2}|.$$

Storage: $dm$ one-dimensional segments.

Full grid: $m^d$ cells at the same resolution.

If every branch has uniform error at most $\varepsilon$, averaging keeps that bound:

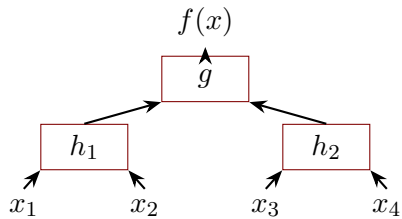
$$\|f - \widehat{f}\|_\infty \le \frac{1}{d}\sum_{j=1}^{d} \|g_j - I_m g_j\|_\infty \le \varepsilon.$$

Schmidt-Hieber (2020), *Nonparametric regression using deep neural networks with ReLU activation function.*

# Hierarchical composition

Interactions can be built from functions with only two inputs:

$$f(x) = g\big(h_1(x_1, x_2), h_2(x_3, x_4)\big), \qquad x \in [0,1]^4.$$




A deep network follows this computation tree.

**Example: multiply four inputs**

$$h_1 = x_1 x_2, \qquad h_2 = x_3 x_4,$$
$$g(u, v) = uv, \qquad f(x) = x_1 x_2 x_3 x_4.$$

At mesh width $1/m$:

Component grids: $3m^2$ cells.
Full grid: $m^4$ cells.

Let $g$ be $L$-Lipschitz in the max norm. For inner errors $\leq \delta$ and outer error $\leq \eta$,

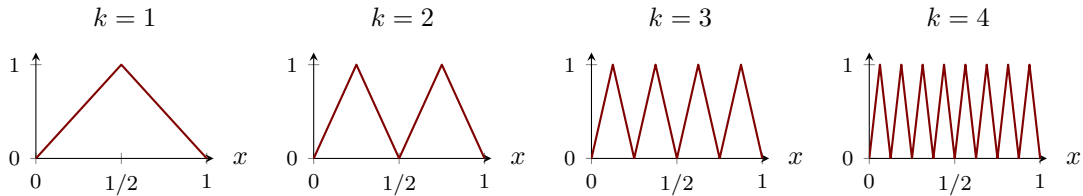
$$\widehat{f} = \widehat{g}(\widehat{h}_1, \widehat{h}_2), \qquad \|f - \widehat{f}\|_\infty \leq L\delta + \eta.$$

Exact and approximate inner outputs must stay in the outer approximation domain.

Schmidt-Hieber (2020), *Nonparametric regression using deep neural networks with ReLU activation function.*

# Depth and the tent map

$$\tau(x) = 2\rho(x) - 4\rho(x - \tfrac{1}{2}) + 2\rho(x - 1), \qquad x \in [0, 1].$$




1 peak, 2 pieces | 2 peaks, 4 pieces | 4 peaks, 8 pieces | 8 peaks, 16 pieces

Each composition doubles the pieces: $2^k$ linear pieces and $2^{k-1}$ peaks.

Network: $k$ hidden layers of width 3, then a linear output ($L = k + 1$).

How many units would a single hidden layer need?

Telgarsky (2016); the one-dimensional construction is detailed in the lecture notes.

# A breakpoint argument for depth

## A breakpoint argument

If a one-hidden-layer ReLU network with $m$ units satisfies $\sup_{x\in[0,1]} |g(x) - \tau^{\circ k}(x)| < 1/2$, then

$$m \geq 2^{k-1}.$$

Exact representation needs $m \geq 2^k - 1$.

1. $\tau^{\circ k}$ has $2^{k-1}$ tent peaks with disjoint interiors.
2. An affine function cannot be within $1/2$ at both ends and the peak.
3. Each peak therefore needs an interior breakpoint; one ReLU gives at most one.

Telgarsky (2016); the stated one-dimensional bound is proved in the lecture notes.

# Backpropagation

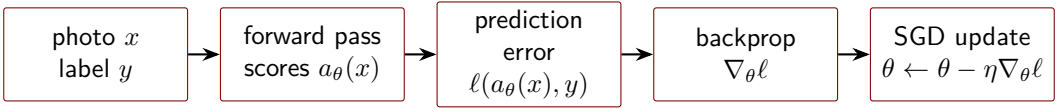
# How do scores become training feedback?

| Task | Prediction from score $a$ | Training loss |
|---|---|---|
| Regression | $\widehat{y} = a$ | $\frac{1}{2}(a - y)^2$ |
| Binary classification | $p = \sigma(a)$ | $-y \log p - (1 - y) \log(1 - p)$ |
| $K$ classes | $p_k = e^{a_k} / \sum_j e^{a_j}$ | $-\log p_y$ |

- Hidden features may be reused across tasks; the output interpretation changes.
- Cross-entropy gives smooth training gradients even when the reported metric is zero-one error.

# How one labeled item changes the network

For a catalog example $(x, y)$, the same parameters $\theta$ serve every item.




Backpropagation computes the gradient; the update changes the predictor.

Pedagogical ordering inspired by the IP Paris Deep Learning backpropagation lecture.

# Output error signal for quadratic loss

For one training example with target $y \in \mathbb{R}^q$, use the identity output $\widehat{y} = a = z^L$:

$$\ell(a, y) = \frac{1}{2}\|a - y\|_2^2 = \frac{1}{2}\sum_{k=1}^{q}(a_k - y_k)^2.$$

Differentiating one coordinate gives

$$\frac{\partial \ell}{\partial a_k} = a_k - y_k, \qquad \boxed{\nabla_a \ell = a - y}.$$

Since $a = z^L$, the output layer sends the prediction residual into backpropagation:

$$\delta^L := \nabla_{z^L} \ell = a - y.$$

Example: $a = (2, -1)$ and $y = (1, 1)$ give $\ell = 5/2$ and $\delta^L = (1, -2)$.

# What error signal leaves the output layer?

For a class label $y \in \{1, \dots, K\}$,

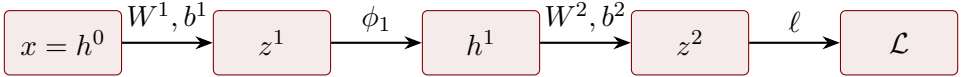
$$\ell(a, y) = -a_y + \log \sum_{j=1}^{K} e^{a_j}, \qquad p_k = \frac{e^{a_k}}{\sum_j e^{a_j}}.$$

Differentiating one coordinate gives

$$\frac{\partial \ell}{\partial a_k} = -\mathbf{1}_{\{k=y\}} + p_k, \qquad \boxed{\nabla_a \ell = p - e_y}.$$

Example: $a = (\log 2, 0, 0)$, $y = 1$ gives $p = (1/2, 1/4, 1/4)$ and $\nabla_a \ell = (-1/2, 1/4, 1/4)$.

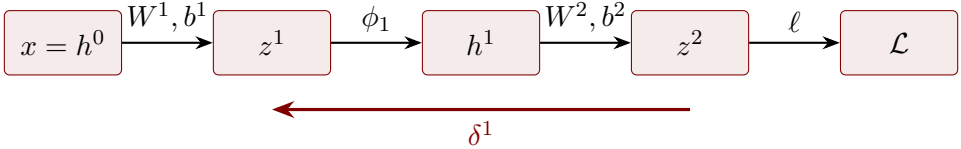
# How does error reach earlier layers?




- Forward pass stores $z^\ell$ and $h^\ell$.

Goodfellow, Bengio, and Courville (2016), Ch. 6.

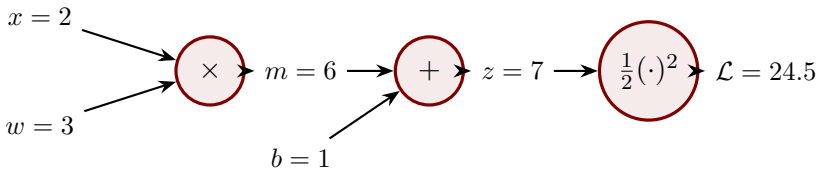
# How does error reach earlier layers?




- Forward pass stores $z^\ell$ and $h^\ell$.
- Reverse pass applies the chain rule once at each graph node.
- $\delta^\ell := \nabla_{z^\ell} \mathcal{L}$ is the gradient at a layer's preactivation.

Goodfellow, Bengio, and Courville (2016), Ch. 6.

# Backpropagation on a small computation graph




$$m = xw, \quad z = m + b, \quad \mathcal{L} = \tfrac{1}{2}z^2, \qquad \underbrace{\frac{\partial \mathcal{L}}{\partial z}}_{7}\underbrace{\frac{\partial z}{\partial m}}_{1}\underbrace{\frac{\partial m}{\partial w}}_{x=2} = \boxed{\frac{\partial \mathcal{L}}{\partial w} = 14}.$$

The same reverse pass gives $\partial \mathcal{L}/\partial b = 7$ and $\partial \mathcal{L}/\partial x = 7w = 21$.

Original numerical example; see Stanford CS231n, Lecture 4.

# Activations control local gradient flow

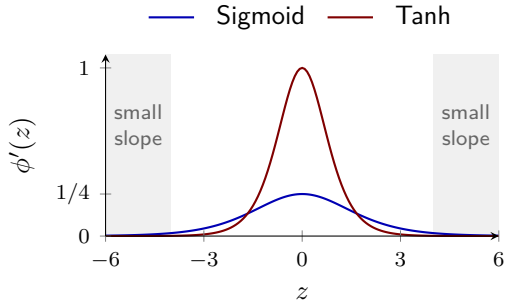
| Activation | $\phi(u)$ | Derivative |
|---|---|---|
| ReLU | $\max\{u, 0\}$ | $\mathbf{1}_{\{u>0\}}$ |
| Sigmoid | $(1 + e^{-u})^{-1}$ | $\phi(u)(1 - \phi(u))$ |
| Tanh | $\tanh(u)$ | $1 - \phi(u)^2$ |

- Sigmoid and tanh derivatives become small in saturated regions.
- ReLU has zero derivative on its negative half-line.
- At $u = 0$, ReLU has no ordinary derivative. Software chooses a local convention.

# Small derivatives multiply across layers

For a scalar chain, let $z^j = w_j h^{j-1} + b_j$ and $h^j = \phi(z^j)$. Then

$$\frac{\partial \mathcal{L}}{\partial h^0} = \frac{\partial \mathcal{L}}{\partial h^k} \prod_{j=1}^{k} [w_j \phi'(z^j)].$$




**Ten-layer illustration**

Assume unit weights: $w_j = 1$. For sigmoid, $0 < \sigma'(z) \leq 1/4$, so

$$\left| \frac{\partial \mathcal{L}}{\partial h^0} \right| \leq 4^{-10} \left| \frac{\partial \mathcal{L}}{\partial h^{10}} \right|.$$

Multiplier: $4^{-10} \approx 9.5 \times 10^{-7}$.

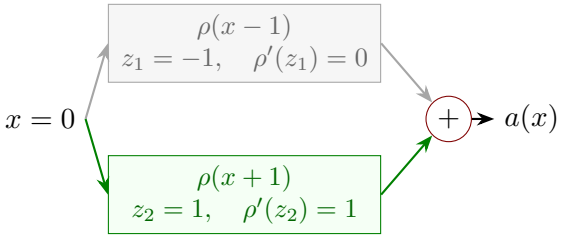
Saturation reduces the factors. Weights can shrink or amplify gradients.

Glorot and Bengio (2010), *Understanding the difficulty of training deep feedforward neural networks*. Scalar chain illustrates the activation effect.

# ReLU can block a gradient in one step

For one input, an inactive ReLU has local derivative $0$:

$$h_j = \rho(z_j), \qquad \frac{\partial \mathcal{L}}{\partial z_j} = \frac{\partial \mathcal{L}}{\partial h_j} \mathbf{1}_{\{z_j > 0\}}, \qquad z_j \neq 0.$$




**Another path stays active**

$$a(x) = \rho(x-1) + \rho(x+1).$$

At $x = 0$,

$$a'(0) = \underbrace{0}_{\text{first path}} + \underbrace{1}_{\text{second path}} = 1.$$

One zero gate blocks its own branch.

A **dying ReLU** is persistently inactive across training inputs.

Active ReLUs have slope $1$; weight products can still shrink or amplify gradients.

Lu et al., *Dying ReLU and Initialization: Theory and Numerical Examples* (arXiv:1903.06733).

# The chain rule at each layer

For $z^{\ell+1} = W^{\ell+1}\phi_\ell(z^\ell) + b^{\ell+1}$,

$$D_{z^\ell} z^{\ell+1} = W^{\ell+1} \operatorname{diag}(\phi_\ell'(z^\ell)).$$

Thus the reverse pass is

$$\boxed{\delta^\ell = ((W^{\ell+1})^\top \delta^{\ell+1}) \odot \phi_\ell'(z^\ell)}.$$

For each affine layer, the parameter gradients are

$$\boxed{\nabla_{W^\ell} \mathcal{L} = \delta^\ell (h^{\ell-1})^\top}, \qquad \nabla_{b^\ell} \mathcal{L} = \delta^\ell.$$

The outer product has shape $d_\ell \times d_{\ell-1}$, matching $W^\ell$.

# Worked example: one hidden layer and quadratic loss

For one sample, let $z = Wx + b$, $h = \rho(z)$, and $a = v^\top h + c$. The output is $a$ itself.

$$\ell = \frac{1}{2}(a - y)^2, \qquad \boxed{r = \frac{\partial \ell}{\partial a} = a - y}.$$

**Backward pass**

$$\nabla_v \ell = rh, \qquad \frac{\partial \ell}{\partial c} = r,$$

$$\delta^1 = (rv) \odot \mathbf{1}_{\{z>0\}},$$

$$\boxed{\nabla_W \ell = \delta^1 x^\top}, \qquad \nabla_b \ell = \delta^1.$$

**Numerical check**

Scalar case ($W = w$): set $x = w = 1$, $b = c = 0$, $v = 2$, and $y = 1$.

$$z = h = 1, \quad a = 2,$$

$$\ell = \tfrac{1}{2}, \quad r = 1,$$

$$\nabla_v \ell = \nabla_c \ell = 1,$$

$$\delta^1 = \nabla_w \ell = \nabla_b \ell = 2.$$

# Worked example: one hidden layer and binary loss

Let $z = Wx + b$, $h = \rho(z)$, $a = v^\top h + c$, and $p = \sigma(a)$. For binary cross-entropy,

$$\frac{\partial \ell}{\partial a} = p - y.$$

The backward pass gives

$$\nabla_v \ell = (p - y)h, \qquad \delta^1 = (p - y)v \odot \mathbf{1}_{\{z>0\}},$$

$$\boxed{\nabla_W \ell = \delta^1 x^\top}, \qquad \nabla_b \ell = \delta^1.$$

# Worked example: one hidden layer and multiclass loss

Let $z = Wx + b$, $h = \rho(z) \in \mathbb{R}^m$, $a = Vh + c$, and $p = \mathrm{softmax}(a)$, where $V \in \mathbb{R}^{K \times m}$. For class label $y \in \{1, \dots, K\}$, let $e_y$ be its one-hot vector.

$$\ell = -\log p_y, \qquad \boxed{r = \nabla_a \ell = p - e_y}.$$

**Backward pass**

$$\nabla_V \ell = r h^\top, \qquad \nabla_c \ell = r,$$

$$\delta^1 = (V^\top r) \odot \mathbf{1}_{\{z>0\}},$$

$$\boxed{\nabla_W \ell = \delta^1 x^\top}, \qquad \nabla_b \ell = \delta^1.$$

**Numerical check ($K = 3$)**

Set $x = w = h = 1$, $b = 0$, $c = 0 \in \mathbb{R}^3$, and $y = 1$.

$$V = (\log 2, 0, 0)^\top,$$

$$p = (\tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{4})^\top, \quad \ell = \log 2,$$

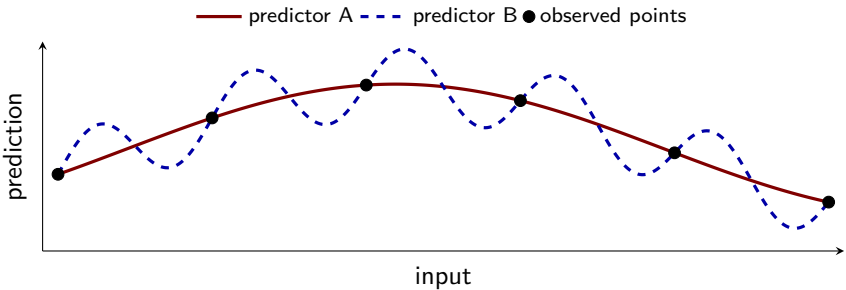
$$r = (-\tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{4})^\top,$$

$$\nabla_V \ell = \nabla_c \ell = r,$$

$$\delta^1 = \nabla_w \ell = \nabla_b \ell = -\tfrac{\log 2}{2}.$$

# Generalization

# Training points leave many predictions possible




Illustrative functions and observations; this is not an empirical result.

# Which fitted model should we keep?

For training examples $(X_i, Y_i)$, write

$$\Phi_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(a_\theta(X_i), Y_i) + R(\theta).$$

- **Explicit regularization** changes the training objective or the training-data distribution: weight penalties, dropout, augmentation.
- **Implicit regularization** comes from the training procedure: initialization, update geometry, noise, or stopping time.

Choose penalties and stopping time on a held-out **validation set**; reserve the **test set** for the final evaluation.

Goodfellow, Bengio, and Courville (2016), Chs. 7–8.

# Explicit regularization

# Should a shifted photo keep its label?

For a distribution $\mathcal{T}$ of transformations,

$$\Phi_{\text{aug}}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{T\sim\mathcal{T}}\big[\ell(a_\theta(TX_i), Y_i)\big].$$

- A small image translation may preserve an object label.
- A crop can remove the object; a rotation can change a digit.
- The assumption that $Y_i$ stays correct after $T$ must be checked for the task.

Augmentation encourages similar predictions for examples related by the chosen transformations.

# Parameter symmetries and regularization

Positive homogeneity of ReLU gives, for every $c > 0$,

$$a\,\rho(w^\top x + b) = \frac{a}{c}\,\rho((cw)^\top x + cb).$$

- Different parameter vectors can represent the *same* function.
- Permuting hidden units with their output weights also preserves the function.
- A parameter penalty can distinguish these equivalent representations.

# What does a weight penalty prefer?

$$\Phi_{n,\lambda}(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(a_\theta(X_i), Y_i) + \frac{\lambda}{2}\sum_{\ell=1}^{L} \left\| W^\ell \right\|_F^2 .$$

For a single ReLU unit, suppose both incoming weights and bias are penalized. Write $v = (w, b)$. The reparameterization $(a, v) \mapsto (a/c, cv)$ preserves the function, but

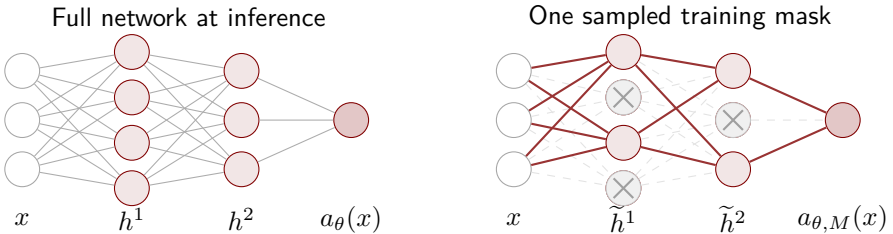
$$\frac{1}{2}\left(\frac{a^2}{c^2} + c^2 \left\|v\right\|_2^2\right) \geq |a| \left\|v\right\|_2 .$$

Thus a parameter penalty selects a particular representation of an unchanged function. Biases are often omitted in the standard layer-wise weight-decay objective above.

# Dropout: randomly masking hidden units

Inverted dropout keeps each hidden unit independently with probability $q$ $(0 < q \leq 1)$:

$$\widetilde{h}_j = \frac{M_j}{q} h_j, \qquad M_j \overset{\text{iid}}{\sim} \text{Bernoulli}(q), \qquad \mathbb{E}_M[\widetilde{h}_j \mid h_j] = h_j.$$




A crossed unit is inactive for this pass. Scale surviving activations by $1/q$.

Training resamples masks. Inference uses the full network without masking or scaling.

Inverted-dropout convention. Srivastava et al. (2014). Dashed connections are inactive in this pass.

# The exact dropout penalty

For a linear predictor $w^\top x$ and square loss, mask *input* coordinates independently: $\widetilde{x}_j = M_j x_j / q$, with $M_j \sim \text{Bernoulli}(q)$.

$$\mathbb{E}_M[w^\top \widetilde{x}] = w^\top x, \qquad \text{Var}_M(w^\top \widetilde{x}) = \frac{1-q}{q} \sum_j w_j^2 x_j^2.$$

The bias–variance identity gives

$$\mathbb{E}_M[(y - w^\top \widetilde{x})^2] = (y - w^\top x)^2 + \text{Var}_M(w^\top \widetilde{x}).$$

Thus the expected training loss is

$$\mathbb{E}_M\left[\frac{1}{2}(y - w^\top \widetilde{x})^2\right] = \frac{1}{2}(y - w^\top x)^2 + \boxed{\frac{1-q}{2q} \sum_{j=1}^{d} w_j^2 x_j^2}.$$

The extra term is half the prediction variance and acts as a feature-dependent weight penalty. For nonlinear networks, there is generally no identical quadratic formula.

Linear-predictor calculation from the lecture notes. Dropout: Srivastava et al. (2014).

# Implicit regularization

# Can the optimizer choose among perfect fits?

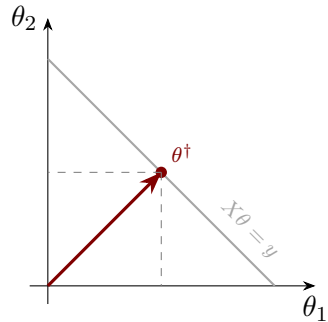
Let $X \in \mathbb{R}^{n \times d}$ have full row rank with $d > n$. For $F(\theta) = \|X\theta - y\|_2^2 / (2n)$ and $\theta_0 = 0$,

$$\theta_{t+1} = \theta_t - \frac{\eta}{n} X^\top (X\theta_t - y).$$

With $0 < \eta \leq n / \|X\|_{\mathrm{op}}^2$,

$$\boxed{\theta_t \to X^\top (XX^\top)^{-1} y.}$$

This is the unique interpolant with the smallest Euclidean norm.



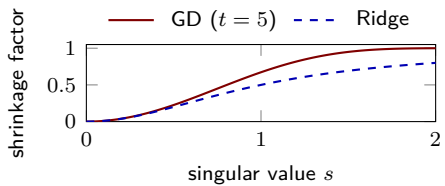
# When does stopping early help?

With $X = U \operatorname{diag}(s_j) V^\top$ and $F(\theta) = \|X\theta - y\|_2^2 / (2n)$:
**Early stopping**, with $\theta_0 = 0$:

$$\theta_t = \sum_{j=1}^{n} q_t(s_j) \frac{u_j^\top y}{s_j} v_j.$$

**Ridge regression**: minimize $F(\theta) + \frac{\lambda}{2} \|\theta\|_2^2$, $\lambda > 0$.

$$\widehat{\theta}_\lambda = (X^\top X + n\lambda I_d)^{-1} X^\top y = \sum_{j=1}^{n} r_\lambda(s_j) \frac{u_j^\top y}{s_j} v_j.$$




$$q_t(s) = 1 - \left(1 - \frac{\eta s^2}{n}\right)^t,$$

$$r_\lambda(s) = \frac{s^2}{s^2 + n\lambda}.$$

Plot: $\eta/n = 0.2$, $n\lambda = 1$.

Goodfellow et al. (2016), Sec. 7.8, Eqs. (7.40)–(7.42), p. 248; Raskutti, Wainwright, and Yu (2014), Sec. 4.1, p. 351.

# SGD implicit regularization: a stability constraint

Let $\theta \in \mathbb{R}$, let $\ell_i \geq 0$ be smooth, and fix $\eta > 0$.

$$F(\theta) = \frac{1}{n}\sum_i \ell_i(\theta), \qquad \theta_{t+1} = \theta_t - \eta \ell_{i_t}'(\theta_t), \qquad i_t \overset{\text{iid}}{\sim} \text{Unif}\{1, \ldots, n\}.$$

**The training loss accepts every perfect fit**

$$\mathcal{M} = \{\theta^* : \ell_i(\theta^*) = 0 \ \forall i\}, \qquad F(\theta^*) = 0 \quad \text{for every } \theta^* \in \mathcal{M}.$$

**SGD dynamics impose a further requirement**

For a perturbation $\delta_t = \theta_t - \theta^*$, define the local linearized multiplier:

$$\mathbb{E}[\delta_{t+1}^2 \mid \delta_t] = \gamma_\eta(\theta^*)\delta_t^2.$$

A multiplier below $1$ contracts nearby deviations in mean square; above $1$ amplifies them.

$$\boxed{\mathcal{M}_\eta^{\text{contract}} = \{\theta^* \in \mathcal{M} : \gamma_\eta(\theta^*) < 1\}}.$$

**Implicit regularization:** SGD can restrict attractive fits while $F$ stays unchanged.

Scalar stability mechanism. Wu and Su (2023), Sec. 3.1. Exact zero-loss fits remain fixed. Contraction describes nearby perturbations.

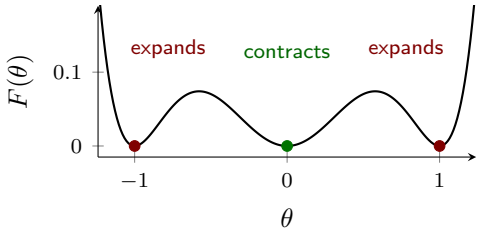
# Example: SGD distinguishes equal-loss minima

Sample either loss independently with probability $1/2$. Set $\eta = 1/3$:

$$\ell_1(\theta) = 0, \qquad \ell_2(\theta) = (\theta^3 - \theta)^2, \qquad F(\theta) = \tfrac{1}{2}(\theta^3 - \theta)^2.$$

At a fit, $a = F''(\theta^*) = (3(\theta^*)^2 - 1)^2$. Local linearized multipliers:

$$\gamma_{\mathrm{GD}} = (1 - \eta a)^2, \qquad \gamma_{\mathrm{SGD}} = \tfrac{1}{2}[1 + (1 - 2\eta a)^2].$$




| Candidate fit | $\theta^* = 0$ | $\theta^* = \pm 1$ |
|---|---|---|
| Training loss | 0 | 0 |
| Curvature $a$ | 1 | 4 |
| GD multiplier | $4/9$ | $1/9$ |
| SGD multiplier | $5/9$ | $17/9$ |

**SGD stability constraint:** $a < 1/\eta = 3$. Only the central zero-loss fit satisfies it.

Original example illustrating Wu and Su (2023), Sec. 3.1. Local linearized stability. Exact minima remain fixed.

# Other sources of implicit regularization

- **SGD and averaging.** In one-hot least squares, SGD on fresh samples with a tuned constant step size and averaging of late iterates is competitive with ridge in excess risk at the same sample budget, when ridge generalizes. [1]
- **Loss function.** For separable logistic regression, GD with a small step size approaches the $\ell_2$ maximum-margin direction while its weight norm diverges. [2]
- **Update geometry.** For separable data with exponential loss, steepest descent with a small fixed step size approaches the maximum normalized margin for its chosen norm. [3]
- **Initialization scale.** For interpolating diagonal two-layer linear regression under gradient flow, the bias approaches minimum $\ell_2$ norm as the balanced initialization scale grows, and minimum $\ell_1$ norm as it vanishes. [4]

[1] Zou et al. (2021), *The Benefits of Implicit Regularization from SGD in Least Squares Problems*, NeurIPS, Thm. 4.1.
[2] Soudry et al. (2018), *The Implicit Bias of Gradient Descent on Separable Data*, JMLR, Thm. 3.
[3] Gunasekar et al. (2018), *Characterizing Implicit Bias in Terms of Optimization Geometry*, ICML, Thm. 5.
[4] Woodworth et al. (2020), *Kernel and Rich Regimes in Overparametrized Models*, COLT, Thms. 1–2.

# The logistic margin direction

For separable data with labels $\widetilde{y}_i \in \{-1, +1\}$, fix a separating unit direction $u$: $m_i = \widetilde{y}_i u^\top x_i > 0$ and $\|u\|_2 = 1$.

**Loss keeps falling as the norm grows along this direction**

$$F(cu) = \frac{1}{n} \sum_{i=1}^{n} \log(1 + e^{-cm_i}) \searrow 0, \qquad \|cu\|_2 = c \to \infty.$$

For $c > 0$, the decision boundary stays fixed as $c$ increases.

**Implicit regularization selects a direction**

For this unregularized homogeneous linear model, full-batch GD with a sufficiently small fixed positive step size satisfies

$$\|w_t\|_2 \to \infty, \qquad \frac{w_t}{\|w_t\|_2} \longrightarrow \frac{w_{\mathrm{mm}}}{\|w_{\mathrm{mm}}\|_2},$$

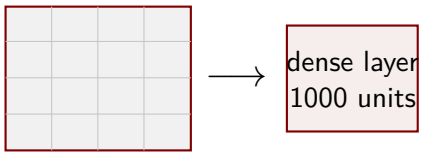
$$w_{\mathrm{mm}} = \operatorname*{argmin}_w \frac{1}{2} \|w\|_2^2 \quad \text{subject to} \quad \widetilde{y}_i w^\top x_i \geq 1 \quad \forall i.$$

Any separating ray drives loss toward zero. GD selects the maximum-margin direction.

Soudry et al. (2018), *The Implicit Bias of Gradient Descent on Separable Data*, Lemma 1 and Theorem 3.

# Convolutional networks

# A dense layer on a photo is expensive




RGB input: $640 \times 480 \times 3$

$$(640 \cdot 480 \cdot 3) \cdot 1000 + 1000$$

921,601,000 parameters

The dense layer learns a separate weight for each pixel-to-unit connection. It does not encode nearby pixels as neighbors.

Can one small filter be reused across positions instead?

Numerical example and teaching motivation adapted from the IP Paris Deep Learning CNN lecture.

# A sliding filter: the one-dimensional case

For a signal $x = (0, 0, 4, 0, 0)$, use the three weights

$$K = \tfrac{1}{4}(1, 2, 1), \qquad y_t = \tfrac{1}{4}x_t + \tfrac{1}{2}x_{t+1} + \tfrac{1}{4}x_{t+2}.$$

| Position $t$ | Input window | Output $y_t$ |
|---|---|---|
| 1 | $(0, 0, 4)$ | $\frac{1}{4}(0 + 2 \cdot 0 + 4) = 1$ |
| 2 | $(0, 4, 0)$ | $\frac{1}{4}(0 + 2 \cdot 4 + 0) = 2$ |
| 3 | $(4, 0, 0)$ | $\frac{1}{4}(4 + 2 \cdot 0 + 0) = 1$ |

The same weights are used in overlapping windows. This fixed filter smooths the signal; a CNN **learns the filter weights** from training data.

Weighted-filter motivation adapted from W. Xia, MATH 548, Lecture 8 (2026), p. 4.

# Two-dimensional convolution: a numerical example

One input channel, one $3 \times 3$ filter, stride $1$, no padding, bias $0$.

$X \in \mathbb{R}^{4\times 4}$

| | | | |
|---|---|---|---|
| 1 | 0 | 0 | 2 |
| 0 | 0 | 0 | 3 |
| 2 | 0 | 1 | 0 |
| 0 | 2 | 1 | 0 |

multiply and sum

$K \in \mathbb{R}^{3\times 3}$

| | | |
|---|---|---|
| 0 | −1 | 1 |
| 0 | 1 | 0 |
| 1 | −1 | 0 |

$\longrightarrow$

$Y \in \mathbb{R}^{2\times 2}$

| | |
|---|---|
| 2 | 1 |
| −2 | 5 |

$$y_{1,1} = (1\cdot 0 + 0\cdot(-1) + 0\cdot 1) + (0\cdot 0 + 0\cdot 1 + 0\cdot 0) + (2\cdot 1 + 0\cdot(-1) + 1\cdot 0) = 2.$$

Move the same filter to the other three positions to obtain the **feature map** $Y$.

Matrices adapted from W. Xia, MATH 548, Lecture 8 (2026), p. 8.

# Convolution and nonlinear feature maps

For an input $X \in \mathbb{R}^{H \times W}$ and a $k_h \times k_w$ filter,

$$z_{u,v} = b + \sum_{r=0}^{k_h-1} \sum_{t=0}^{k_w-1} K_{r+1,t+1} X_{u+r,v+t}.$$

With stride $1$ and no padding,

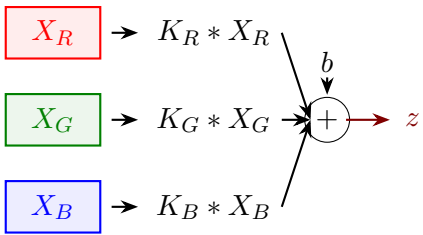
$$u = 1, \ldots, H - k_h + 1, \qquad v = 1, \ldots, W - k_w + 1.$$

The layer applies the activation at every output position:

$$h_{u,v} = \rho(z_{u,v}), \qquad \begin{pmatrix} 2 & 1 \\ -2 & 5 \end{pmatrix} \xrightarrow{\text{ReLU}} \begin{pmatrix} 2 & 1 \\ 0 & 5 \end{pmatrix}.$$

The operation above is cross-correlation (the kernel is not flipped), conventionally called convolution in CNNs. Goodfellow et al. (2016), Secs. 9.1 and 9.3.

# Input channels, filters, and output channels




One RGB filter has three $k_h \times k_w$ slices.

One filter combines **all input channels**:

$$z = K_R * X_R + K_G * X_G + K_B * X_B + b.$$

It produces **one output channel**.

For $C_{\text{out}}$ filters,

$$K \in \mathbb{R}^{k_h \times k_w \times C_{\text{in}} \times C_{\text{out}}}, \qquad b \in \mathbb{R}^{C_{\text{out}}}.$$

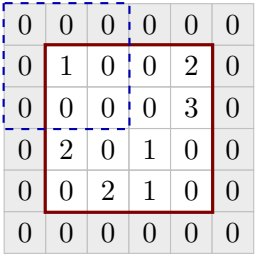
The output has shape

$$H' \times W' \times C_{\text{out}}.$$

RGB input has $C_{\text{in}} = 3$; a bank of $16$ filters produces $16$ feature maps, each combining information from all three colors.

Channel construction adapted from W. Xia, MATH 548, Lecture 8 (2026), p. 9; Goodfellow et al. (2016), Sec. 9.5.

# Padding: filters at the boundary




Original $4 \times 4$ input in red
First $3 \times 3$ window in blue

**Zero padding** $p = 1$: add one row or column of zeros on every side.

$$4 \times 4 \longrightarrow 6 \times 6.$$

For a $3 \times 3$ filter and stride $1$,

$$H' = W' = 6 - 3 + 1 = 4.$$

Thus the output retains the original spatial dimensions.

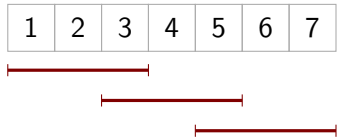
For odd kernel width $k$ and stride $1$, padding $p = (k-1)/2$ preserves size. Other boundary choices include reflection and periodic padding.

Padding motivation adapted from W. Xia, MATH 548, Lecture 8 (2026), p. 11.

# Stride and output dimensions

**Stride** $s$ is the distance between successive filter positions.

$$H' = \left\lfloor \frac{H + 2p - k_h}{s} \right\rfloor + 1, \qquad W' = \left\lfloor \frac{W + 2p - k_w}{s} \right\rfloor + 1$$




Width $7$, kernel width $3$, stride $2$
Windows start at positions $1, 3, 5$

$5 \times 5$ input, $3 \times 3$ filters:

| Padding $p$ | Stride $s$ | Output |
|---|---|---|
| $0$ | $1$ | $3 \times 3$ |
| $1$ | $1$ | $5 \times 5$ |
| $0$ | $2$ | $2 \times 2$ |
| $1$ | $2$ | $3 \times 3$ |

Stride and padding set spatial size; the filter count sets output channels.

Stride motivation adapted from W. Xia, MATH 548, Lecture 8 (2026), p. 10. Formula assumes no dilation and the same stride/padding along both axes.

# Local connections and shared parameters

- **Local connections**: each output sees a $k_h \times k_w$ patch across all $C_{\text{in}}$ channels.
- **Parameter sharing**: each filter and its bias are reused at every spatial position.

$$\#\text{parameters} = k_h k_w C_{\text{in}} C_{\text{out}} + C_{\text{out}}.$$

For a $640 \times 480 \times 3$ input, $16$ filters of size $3 \times 3$, stride $1$, and padding $1$:

$$\underbrace{16(3 \cdot 3 \cdot 3 + 1) = \boxed{448}}_{\text{learned parameters}} \qquad \underbrace{640 \times 480 \times 16}_{\text{output shape}}.$$

For input features $h^{\ell-1}$, sum every use of the shared weights:

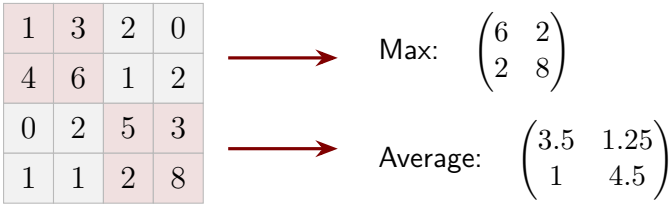
$$\frac{\partial \mathcal{L}}{\partial K^{\ell}_{r,t,c,j}} = \sum_{u,v} \delta^{\ell}_{u,v,j} \, h^{\ell-1,\text{pad}}_{u+r-1,v+t-1,c}, \qquad \delta^{\ell}_{u,v,j} = \frac{\partial \mathcal{L}}{\partial z^{\ell}_{u,v,j}}.$$

$h^{\ell-1,\text{pad}}$ is the padded incoming feature map. Stride 1, one-based kernel indices. Goodfellow et al. (2016), Secs. 9.2 and 9.7.

# Pooling: a numerical example

Apply a $2 \times 2$ window with stride $2$ to each channel separately.

Input feature map




$$\max\{1,3,4,6\} = 6, \qquad \frac{1+3+4+6}{4} = 3.5.$$

Pooling reduces spatial size and preserves channels, with no learned parameters.

Pooling operations adapted from W. Xia, MATH 548, Lecture 8 (2026), pp. 12–13; Goodfellow et al. (2016), Sec. 9.3. Numerical example is original.

# Backpropagation through a convolutional layer

At layer $\ell$, $h^{\ell-1}$ is the input feature map. Raw data are $h^0 = x$. Use one channel, stride 1, no padding, and incoming gradient $G^\ell = \partial \mathcal{L} / \partial h^\ell$.

$$z^\ell_{u,v} = b^\ell + \sum_{r=1}^{k_h} \sum_{t=1}^{k_w} K^\ell_{r,t} h^{\ell-1}_{u+r-1,v+t-1}, \qquad h^\ell = \rho(z^\ell).$$

ReLU gate: $\delta^\ell_{u,v} = G^\ell_{u,v} \mathbf{1}_{\{z^\ell_{u,v} > 0\}}$,

Parameters: $\dfrac{\partial \mathcal{L}}{\partial K^\ell_{r,t}} = \sum_{u,v} \delta^\ell_{u,v} h^{\ell-1}_{u+r-1,v+t-1}, \qquad \dfrac{\partial \mathcal{L}}{\partial b^\ell} = \sum_{u,v} \delta^\ell_{u,v},$

To previous layer: $G^{\ell-1}_{i,j} = \dfrac{\partial \mathcal{L}}{\partial h^{\ell-1}_{i,j}} = \sum_{r=1}^{k_h} \sum_{t=1}^{k_w} K^\ell_{r,t} \delta^\ell_{i-r+1,j-t+1}.$

Set $\delta^\ell = 0$ outside the output grid. Pass $G^{\ell-1}$ back to compute earlier parameter gradients. The raw input $x$ stays fixed during training.

CNN cross-correlation: Stanford CS231n; D2L, Sec. 14.10.3. Multiple channels and batch examples add gradient contributions.

# CNN backpropagation: a complete numerical example

Take an intermediate feature map $h^{\ell-1} = (0, 0, 4, 0, 0)$, $K^\ell = (\frac{1}{4}, \frac{1}{2}, \frac{1}{4})$, and $b^\ell = 0$ for $\ell > 1$. Apply valid stride-1 convolution and ReLU, then max-pool all three outputs.

$$h^{\ell-1} = (0,0,4,0,0) \xrightarrow{\text{Conv + ReLU}} z^\ell = h^\ell = (1,2,1) \xrightarrow{\text{Max pool}} a = 2 \xrightarrow{y=1} \mathcal{L} = \tfrac{1}{2}(a-1)^2 = \tfrac{1}{2}$$

$$G^{\ell-1} = (0, \tfrac{1}{4}, \tfrac{1}{2}, \tfrac{1}{4}, 0) \longleftarrow \delta^\ell = (0,1,0) \longleftarrow \partial\mathcal{L}/\partial a = 1 \longleftarrow 1$$

The maximum is $h_2^\ell = 2$, so $G^\ell = (0, 1, 0)$. All $z_u^\ell > 0$, hence $\delta^\ell = G^\ell$.

$$\nabla_{K^\ell}\mathcal{L} = \sum_{u=1}^{3} \delta_u^\ell (h_u^{\ell-1}, h_{u+1}^{\ell-1}, h_{u+2}^{\ell-1}) = (0, 4, 0), \qquad \frac{\partial \mathcal{L}}{\partial b^\ell} = \sum_u \delta_u^\ell = 1.$$

For average pool, each of $q$ inputs receives $g/q$ from an incoming pool gradient $g$. Overlapping windows add contributions.

Original calculation reusing the earlier 1D filter. Pool-gradient routing: Stanford CS231n. At max-pool ties, use a chosen argmax/subgradient convention.

# A complete CNN classifier

Example: ten-class classification of $32 \times 32$ RGB images.

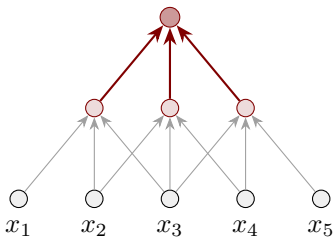
| Layer | Settings | Output shape | Parameters |
|---|---|---|---|
| Input | | $32 \times 32 \times 3$ | $0$ |
| Conv + ReLU | $3 \times 3$, $p = 1$, $s = 1$, 16 filters | $32 \times 32 \times 16$ | $448$ |
| Max pool | $2 \times 2$, $s = 2$ | $16 \times 16 \times 16$ | $0$ |
| Conv + ReLU | $3 \times 3$, $p = 1$, $s = 1$, 32 filters | $16 \times 16 \times 32$ | $4{,}640$ |
| Global average pool | Average each channel over space | $32$ | $0$ |
| Dense + softmax | 32 features, 10 class scores | $10$ | $330$ |

$$3 \cdot 3 \cdot 16 \cdot 32 + 32 = 4{,}640, \qquad 32 \cdot 10 + 10 = 330.$$

Original architecture and calculations; layer organization follows Goodfellow et al. (2016), Sec. 9.3. Total learned parameters: $5{,}418$.

# Receptive fields grow with depth

The **receptive field** is the region of the original input that can affect an output unit.




Two filters of width $3$, both with stride $1$:

$$r_1 = 3, \qquad r_2 = 3 + (3 - 1) = 5.$$

For $N$ layers of width $k$,
with stride $1$ and no dilation,

$$\boxed{r_N = 1 + N(k - 1)}.$$

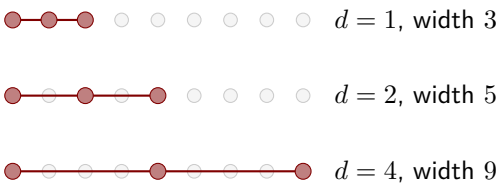
In two dimensions, two $3 \times 3$ layers cover a $5 \times 5$ input region; three cover $7 \times 7$.

Diagram adapted from W. Xia, MATH 548, Lecture 8 (2026), p. 15. Receptive-field widths are along one input axis.

# Dilated convolution: spacing within a filter

**Dilation** $d$ spaces the kernel's sample locations apart. It does not set the distance between successive output positions.

$$z_t = b + \sum_{r=0}^{k-1} K_{r+1} x_{t+dr}, \qquad k_{\text{effective}} = 1 + d(k-1).$$




One filter, three sampled locations

For a stack with stride 1, kernel width 3, and dilations $1, 2, 4$:

$$r_3 = 1 + 2(1 + 2 + 4) = \boxed{15}.$$

The spacing increases the input span without increasing the number of weights in each filter.

Dilation motivation: W. Xia, MATH 548, Lecture 8 (2026), pp. 14–15. Construction: Yu and Koltun (2016), Sec. 2 and Fig. 1.

# CNN architecture milestones

# CNN architecture milestones: 1998–2022

The core operations stay familiar. The organization of layers changes.

| Introduced | Representative network | Structural idea |
|---|---|---|
| 1998 | LeNet-5 | Local filters and learned subsampling |
| 2012 | AlexNet | ReLU and large-scale GPU training |
| 2014 | VGG | Deep stacks of small $3 \times 3$ filters |
| 2014 | GoogLeNet | Parallel branches at several spatial scales |
| 2015 | ResNet | Residual shortcuts between layers |
| 2017 | MobileNet-v1 | Depthwise filtering and pointwise mixing |
| 2019 | EfficientNet | Joint depth, width, and resolution scaling |
| 2022 | ConvNeXt | Large depthwise kernels and modern blocks |

LeNet uses MNIST digits; later milestones mainly use ImageNet classification.

Years refer to model introduction or preprint. Primary papers and evaluation settings appear on the following slides.

# Reading the benchmark results

## Datasets

**MNIST**: ten handwritten digit classes. $60{,}000$ training and $10{,}000$ test images.

**ImageNet-1K / ILSVRC**: $1{,}000$ object classes.
About $1.28$ million training images and $50{,}000$ validation images.

Competition test labels are held out. Validation and competition-test scores are identified separately.

## Metrics and evaluation settings

**Top-1 error**: the highest-scoring class is wrong.

**Top-5 error**: the correct class is absent from the five highest-scoring predictions.

For the same $k$,

top-$k$ accuracy $= 100\% -$ top-$k$ error.

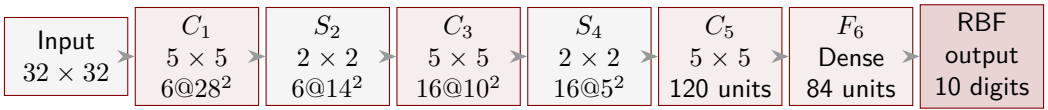
A single model, an ensemble, and multi-crop evaluation can give different scores for the same architecture.

**Comparison rule:** Read the dataset, split, crop setting, and training data beside each score.

LeCun et al. (1998), Sec. III; He et al. (2016), Sec. 4.1. Later papers also change training recipes.

# LeNet-5 (1998): learning digit features

A grayscale digit passes through convolution and subsampling stages before classification.




$C$: convolution; $S$: learned subsampling; $6@28^2$: six $28 \times 28$ feature maps. The original uses scaled tanh, partial $C_3$ connections, and padded $32 \times 32$ inputs.

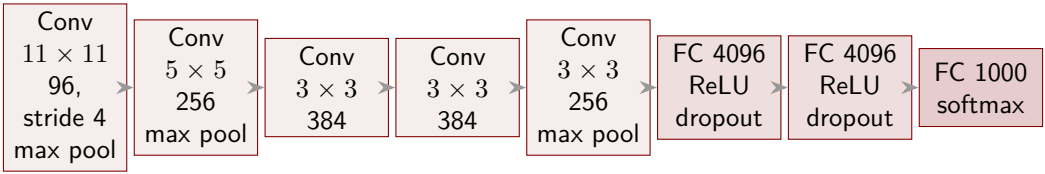
**MNIST test results: one LeNet-5, $10{,}000$ unchanged test images**

| Training data | Test classification error |
|---|---|
| Original training images | $0.95\%$ |
| Added affine-distorted training images | $0.8\%$ |

LeCun et al. (1998), Fig. 2, Secs. II.B and III.B, pp. 2284–2288. Original architecture, including the RBF classifier.

# AlexNet (2012): CNNs at ImageNet scale

Five convolutional layers feed three FC layers; the two hidden FC layers use dropout.




Convolution labels give output channels. Hidden layers use ReLU; training used two GPUs.

**ILSVRC-2012 classification: reported error rates**

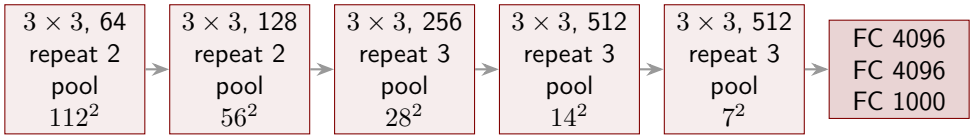
| Evaluation | Top-1 error | Top-5 error |
|---|---|---|
| Single CNN, validation, 10 crops | 40.7% | 18.2% |
| 7-CNN ensemble, competition test | — | 15.3% |

Two ensemble members used extra ImageNet-2011 pretraining data.

Krizhevsky et al. (2012), Fig. 2, Table 2. Stage schematic; both results average ten crops.

# VGG-16 (2014): depth with small filters

Each block repeats $3 \times 3$ convolutions, then uses $2 \times 2$ max pooling.




Spatial sizes above assume a $224 \times 224$ input. ReLU follows hidden layers. There are $13$ convolutional and $3$ fully connected layers, with about $138$ million parameters.

**ImageNet / ILSVRC validation: one VGG-16 (configuration D)**

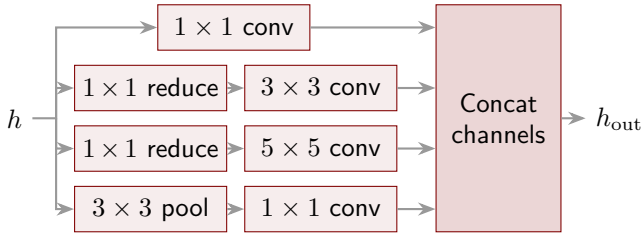
| Evaluation | Top-1 error | Top-5 error |
|---|---|---|
| Dense, three scales and horizontal flips | $24.8\%$ | $7.5\%$ |

Training used scale jitter $S \in [256, 512]$; evaluation used $Q \in \{256, 384, 512\}$.

Simonyan and Zisserman (2014 preprint; ICLR 2015), Tables 1–2 and 4. Labels: kernel, channels, repeats, pool, and spatial size.

# GoogLeNet (2014): parallel Inception branches

A stem feeds nine Inception modules, global average pooling, and a classifier.



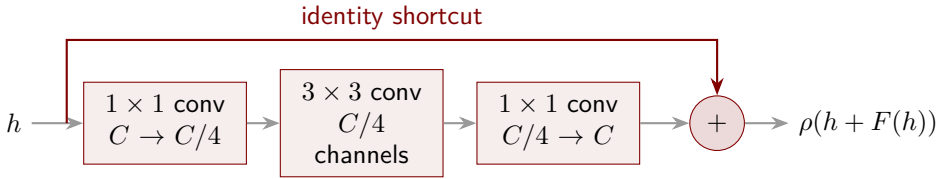

**ILSVRC-2014 top-5 classification error**

| Model and views | Split | Top-5 error |
|---|---|---|
| 1 model, 1 crop | Validation | $10.07\%$ |
| 1 model, 144 crops | Validation | $7.89\%$ |
| 7 models, 144 crops each | Competition test | $6.67\%$ |

Szegedy et al. (2014 preprint; CVPR 2015), Fig. 2, Table 3. $1 \times 1$ reductions control branch cost. Auxiliary heads are used during training.

# ResNet (2015): residual shortcuts

A bottleneck block learns a correction $F(h)$ and adds the input through a shortcut.




ResNet-152 repeats bottlenecks $(3, 8, 36, 3)$ across four stages, then uses GAP and a classifier. Projections handle shape changes. Internal BatchNorm and ReLU are omitted.

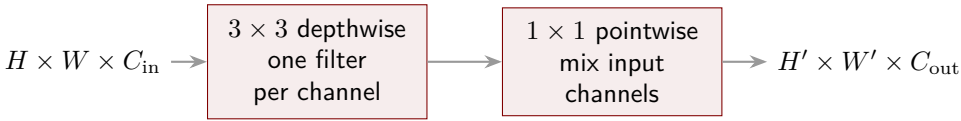
**ImageNet classification: original ResNet results**

| Evaluation | Top-1 error | Top-5 error |
|---|---|---|
| ResNet-152, validation, single model, 10 crops | 21.43% | 5.71% |
| 6-model ensemble, competition test | — | 3.57% |

He et al. (2015 preprint; CVPR 2016), Fig. 5, Tables 1, 3 and 5. Identity block shown for matching input/output shapes.

# MobileNet-v1 (2017): separate space and channels

Depthwise convolution filters each channel; pointwise convolution mixes channels.




For a $k \times k$ convolution, the weight counts are

$$\underbrace{k^2 C_{\text{in}} C_{\text{out}}}_{\text{standard}} \quad \text{versus} \quad \underbrace{k^2 C_{\text{in}} + C_{\text{in}} C_{\text{out}}}_{\text{depthwise + pointwise}} .$$

A convolutional stem precedes these blocks; global average pooling and a classifier follow.

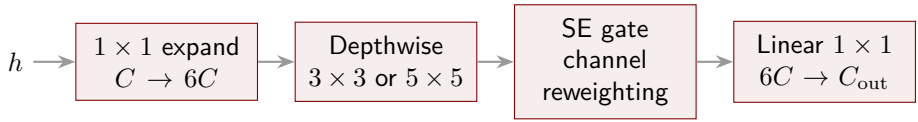
**ImageNet: width multiplier $1.0$, input $224 \times 224$**

| Model | Top-1 accuracy | Parameters | Mult-adds |
|---|---|---|---|
| MobileNet-v1 | $70.6\%$ | $4.2$ million | $569$ million |

Howard et al. (2017), Figs. 2–3, Tables 1 and 6. Each convolution uses BatchNorm and ReLU. The original result table does not specify the crop protocol.

# EfficientNet (2019): scale depth, width, and resolution

EfficientNet uses mobile inverted bottlenecks. A typical MBConv6 branch is:




Joint scaling uses a common budget parameter $\phi$:

$$d = \alpha^{\phi}, \qquad w = \beta^{\phi}, \qquad r = \gamma^{\phi}, \qquad \alpha\beta^2\gamma^2 \approx 2.$$

$d, w, r$ scale depth, width, and resolution. Shortcuts require matching shapes.

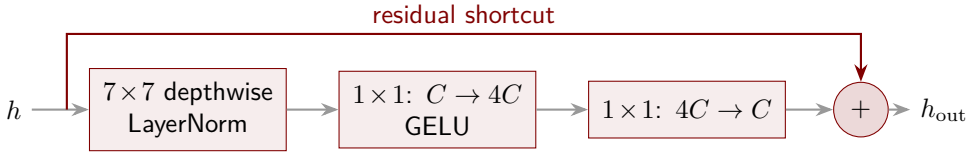
**ImageNet: single-model, single-crop results in the ICML paper**

| Model | Top-1 accuracy | Top-5 accuracy | Parameters |
|---|---|---|---|
| B0 | 76.3% | 93.2% | 5.3 million |
| B7 | 84.4% | 97.1% | 66 million |

Tan and Le (ICML 2019), Fig. 1, Tables 1–2. Conference-version results. SE: squeeze-and-excitation. B0's first MBConv stage uses expansion 1.

# ConvNeXt (2022): modern convolutional blocks

ConvNeXt-T uses a stride-4 stem and four stages with widths $(96, 192, 384, 768)$ and block counts $(3, 3, 9, 3)$, followed by global average pooling and a classifier.




The block combines large spatial kernels, an expanded channel representation, and residual learning. Training also uses a modern optimizer and augmentation recipe.

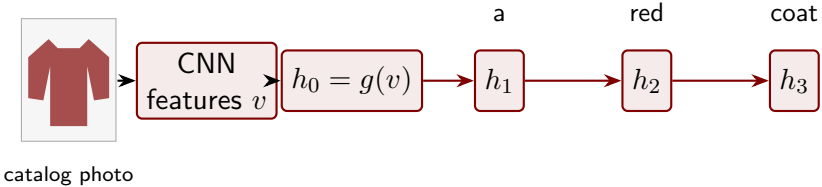
**ImageNet-1K validation: trained on ImageNet-1K only**

| Model | Input size | Top-1 accuracy | Parameters |
|---|---|---|---|
| ConvNeXt-T | $224 \times 224$ | $82.1\%$ | $28.6$ million |

Liu et al. (CVPR 2022), Fig. 4, Tables 1 and 9. Block schematic omits layer scale and stochastic depth.

# Recurrent networks

# One photo can require an ordered output




$$\Pr(y_1, y_2, y_3 \mid x) = \Pr(y_1 \mid x)\,\Pr(y_2 \mid y_1, x)\,\Pr(y_3 \mid y_1, y_2, x).$$

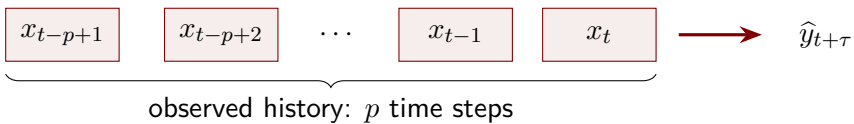
Classification produces one label. A caption produces successive words, each conditioned on the photo and earlier words.

Original catalog example; image-captioning motivation from Stanford CS231n, Lecture 7.

# Sequential prediction

At time $t$, use a lookback window of $p$ observations to predict a target $\tau$ steps ahead:

$$\widehat{y}_{t+\tau} = F_\theta(x_{t-p+1}, \ldots, x_t), \qquad x_s \in \mathbb{R}^d.$$




A recurrent model reads the observations in order and updates a state:

$$h_s = F_{\text{cell}}(h_{s-1}, x_s), \qquad \widehat{y}_{t+\tau} = g_\theta(h_t).$$

The input can be a multivariate time series or a sequence of word embeddings.

Forecasting and lookback formulation adapted from W. Xia, MATH 548, Lecture 7 (2026), pp. 3–5.

# The RNN cell and its parameters

$$x_t \in \mathbb{R}^d, \qquad h_t \in \mathbb{R}^H, \qquad a_t \in \mathbb{R}^K,$$
$$u_t = Ah_{t-1} + Bx_t + b, \qquad h_t = \phi(u_t), \qquad a_t = Ch_t + c.$$

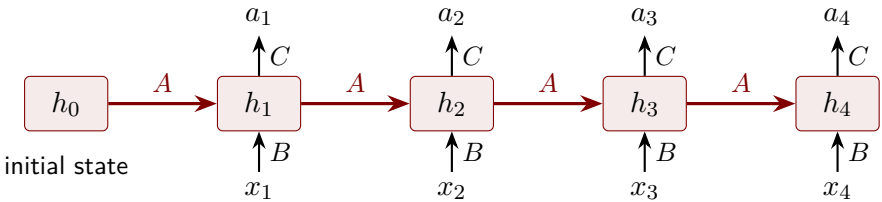
| Parameter | Shape | Role |
|---|---|---|
| $A$ | $H \times H$ | Transform the previous hidden state |
| $B$ | $H \times d$ | Transform the current input |
| $C$ | $K \times H$ | Map the hidden state to output scores |
| $b, c$ | $H,\ K$ | Hidden and output biases |

$$\boxed{\#\text{parameters} = H^2 + Hd + H + KH + K}.$$

Use $\widehat{y}_t = a_t$ for regression, or $\widehat{p}_t = \mathrm{softmax}(a_t)$ for classification.

Structure and dimensions adapted from W. Xia, MATH 548, Lecture 7 (2026), pp. 5–6; Goodfellow et al. (2016), Sec. 10.2.

# Unrolling the same cell through time




The same $A, B, C, b, c$ are reused at every time step. Sequence length changes the computation depth, but not the parameter count.

- Use the final state for one forecast or one sequence label.
- Use outputs at successive times for an ordered output sequence.

Unrolling adapted from W. Xia, MATH 548, Lecture 7 (2026), p. 7; Goodfellow et al. (2016), Secs. 10.1–10.2. Diagram is editable.

# A linear RNN is a lag filter

Use one hidden unit, identity activation, and zero hidden bias. For each length-$p$ window, reset $h_{t-p} = 0$:

$$h_s = \gamma h_{s-1} + \beta x_s, \qquad \widehat{x}_{t+1} = \mu + h_t.$$

Repeated substitution gives

$$h_t = \beta x_t + \gamma\beta x_{t-1} + \gamma^2 \beta x_{t-2} + \cdots + \gamma^{p-1}\beta x_{t-p+1},$$

$$\boxed{\widehat{x}_{t+1} = \mu + \sum_{j=1}^{p} \underbrace{\beta\gamma^{j-1}}_{\varphi_j} x_{t+1-j}}.$$

This is an AR($p$) predictor with **geometrically constrained lag coefficients**. A general AR($p$) has $p$ freely chosen lag coefficients.

Linear special case adapted from W. Xia, MATH 548, Lecture 7 (2026), p. 8; window initialization and coefficient restriction made explicit.

# Memory decay in a plain RNN

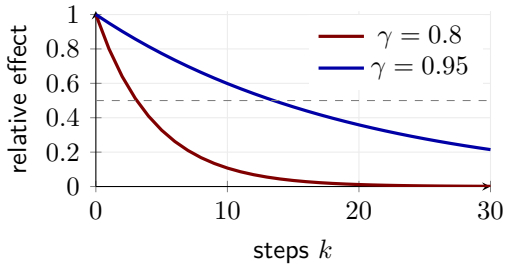
Compare two initial states driven by the *same* input sequence. If $\phi$ is $L_\phi$-Lipschitz, then

$$\|h_t - h'_t\|_2 \le \underbrace{L_\phi \, \|A\|_{\mathrm{op}}}_{q} \, \|h_{t-1} - h'_{t-1}\|_2 \le q^t \, \|h_0 - h'_0\|_2 .$$

If $q < 1$, the effect of the initial state decays geometrically.

In a scalar linear recurrence, a perturbation has relative effect $\gamma^k$ after $k$ steps.

$$k_{1/2} = \left\lceil \frac{\log(1/2)}{\log \gamma} \right\rceil, \qquad 0 < \gamma < 1.$$




Memory-decay motivation: W. Xia, MATH 548, Lecture 7 (2026), pp. 9–12. Plot: $\gamma^k$; half-lives are 4 and 14 steps.

# Backpropagation through time

Train on the unrolled graph. For losses at each output, write

$$\mathcal{L} = \sum_{t=1}^{T} \ell_t(a_t, y_t), \qquad e_t = \nabla_{a_t} \ell_t, \qquad \delta_t = \nabla_{u_t} \mathcal{L}.$$

The backward recursion is

$$\boxed{\delta_t = \operatorname{diag}(\phi'(u_t))(C^\top e_t + A^\top \delta_{t+1})}, \qquad \delta_{T+1} = 0.$$

- $C^\top e_t$ carries the error from the current output.
- $A^\top \delta_{t+1}$ carries the error from future time steps.

Every use of a shared parameter contributes to its gradient:

$$\nabla_A \mathcal{L} = \sum_{t=1}^{T} \delta_t h_{t-1}^\top, \qquad \nabla_B \mathcal{L} = \sum_{t=1}^{T} \delta_t x_t^\top, \qquad \nabla_C \mathcal{L} = \sum_{t=1}^{T} e_t h_t^\top.$$

Goodfellow, Bengio, and Courville (2016), Sec. 10.2. Derivatives assume a fixed initial state; the same chain rule applies with only a final-time loss.

# Vanishing and exploding gradients

For a loss only at time $T$, set $J_s = \operatorname{diag}(\phi'(u_s))A$. Then

$$\nabla_{h_t} \ell_T = J_{t+1}^\top \cdots J_T^\top \nabla_{h_T} \ell_T.$$

$$\|\nabla_{h_t} \ell_T\|_2 \leq \left( \prod_{s=t+1}^{T} \|J_s\|_{\mathrm{op}} \right) \|\nabla_{h_T} \ell_T\|_2 .$$

| Scalar local derivative | Multiplier over 20 steps |
|---|---|
| $J_s = 0.5$ | $0.5^{20} \approx 9.5 \times 10^{-7}$ |
| $J_s = 1.5$ | $1.5^{20} \approx 3.3 \times 10^{3}$ |

Gradient clipping can limit exploding updates. Gates provide a way to control how state information is carried across time.

Goodfellow et al. (2016), Sec. 10.7; Bengio et al. (1994), Learning long-term dependencies with gradient descent is difficult.

# Exponential smoothing of the hidden state

Compute a candidate, then blend it with the previous state:

$$\widetilde{h}_t = \phi(Ah_{t-1} + Bx_t + b), \qquad \boxed{h_t = (1-\alpha)h_{t-1} + \alpha\widetilde{h}_t}, \quad \alpha \in [0,1].$$

Expanding the updates gives

$$h_t = (1-\alpha)^t h_0 + \sum_{s=1}^{t} \alpha(1-\alpha)^{t-s}\widetilde{h}_s.$$

- $\alpha = 0$: keep the previous state.
- $\alpha = 1$: use the candidate, recovering the plain RNN.
- $0 < \alpha < 1$: spread updates across time through geometric weights.

For fixed $\alpha$, the full state Jacobian is

$$\frac{\partial h_t}{\partial h_{t-1}} = (1-\alpha)I + \alpha\,\mathrm{diag}(\phi'(u_t))A.$$

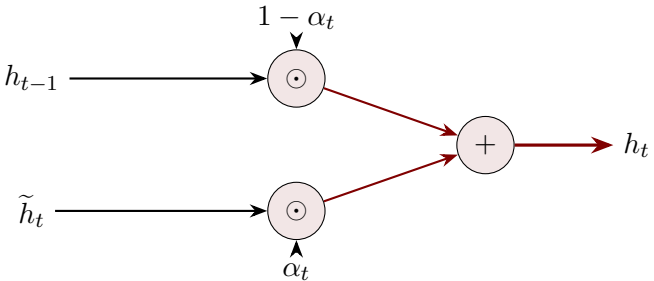
Fixed-smoothing construction adapted from W. Xia, MATH 548, Lecture 7 (2026), pp. 16–19.

# A learned update gate

Let the input and previous state choose an update weight for each coordinate:

$$\alpha_t = \sigma(A_\alpha h_{t-1} + B_\alpha x_t + b_\alpha) \in (0,1)^H,$$

$$\boxed{h_t = (1 - \alpha_t) \odot h_{t-1} + \alpha_t \odot \widetilde{h}_t}.$$




Weights near $0$ retain a coordinate; weights near $1$ replace it with the candidate.

Learned smoothing adapted from W. Xia, MATH 548, Lecture 7 (2026), pp. 19–20. Here $\alpha_t$ weights the new candidate.

# GRU: update and reset gates

$$
\begin{aligned}
\alpha_t &= \sigma(A_\alpha h_{t-1} + B_\alpha x_t + b_\alpha), && \text{update gate,} \\
r_t &= \sigma(A_r h_{t-1} + B_r x_t + b_r), && \text{reset gate,} \\
\widetilde{h}_t &= \tanh\big(A_h(r_t \odot h_{t-1}) + B_h x_t + b_h\big), && \text{candidate,} \\
h_t &= (1-\alpha_t) \odot h_{t-1} + \alpha_t \odot \widetilde{h}_t, && \text{new state.}
\end{aligned}
$$

- $r_t \approx 0$: the candidate is formed mainly from the current input.
- $\alpha_t \approx 0$: the final state still retains the previous state, even if the reset gate is small.
- $\alpha_t \approx 1$: the candidate replaces the previous state.

W. Xia, MATH 548, Lecture 7 (2026), pp. 21–22; Cho et al. (2014), Sec. 2.3, Eqs. (5)–(8). Candidate weight: $\alpha_t = 1 - z_t$ in Cho et al.

# LSTM

# LSTM: a separate cell state

An LSTM maintains stored memory $c_t$ and a visible hidden state $h_t$:

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

The forget and input gates are controlled independently; their weights need not sum to one.

| Limiting gate values | Cell or hidden state | Effect |
|---|---|---|
| $f_t = 1,\ i_t = 0$ | $c_t = c_{t-1}$ | Retain stored memory |
| $f_t = 0,\ i_t = 1$ | $c_t = \widetilde{c}_t$ | Replace stored memory |
| $f_t = 1,\ i_t = 1$ | $c_t = c_{t-1} + \widetilde{c}_t$ | Accumulate information |
| $o_t = 0$ | $h_t = 0$ | Hide the cell from the output |

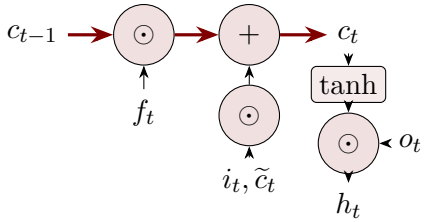
The output gate can hide stored information without erasing the cell state.

W. Xia, MATH 548, Lecture 7 (2026), pp. 23–24; Hochreiter and Schmidhuber (1997); Gers et al. (2000).

# LSTM gate computations

$$
\begin{aligned}
i_t &= \sigma(W_i x_t + U_i h_{t-1} + b_i), \\
f_t &= \sigma(W_f x_t + U_f h_{t-1} + b_f), \\
o_t &= \sigma(W_o x_t + U_o h_{t-1} + b_o), \\
\widetilde{c}_t &= \tanh(W_g x_t + U_g h_{t-1} + b_g).
\end{aligned}
$$

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




Hochreiter and Schmidhuber (1997); forget gate: Gers et al. (2000).

# LSTM memory over time

Repeated substitution gives, with coordinatewise products,

$$c_T = \left(\prod_{s=1}^{T} f_s\right) \odot c_0 + \sum_{t=1}^{T} \left(\prod_{s=t+1}^{T} f_s\right) \odot i_t \odot \widetilde{c}_t.$$

Holding the gates fixed, the direct path through the cell state has Jacobian

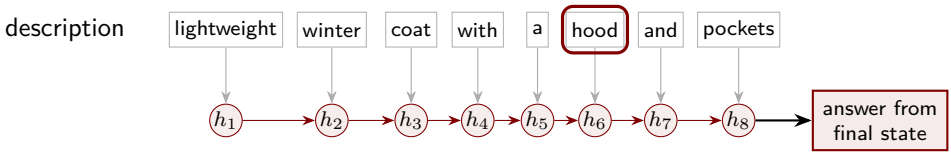
$$\left.\frac{\partial c_T}{\partial c_t}\right|_{\text{direct}} = \operatorname{diag}\left(\prod_{s=t+1}^{T} f_s\right).$$

Forget gates near one can preserve a contribution. Gates near zero erase it. The *total* gradient also contains paths through the gates.

# Transformers

# A single state can become a bottleneck

Imagine answering: *Does this coat have a hood?*




If a decoder receives only $h_T$, that vector must preserve the earlier word *hood*.

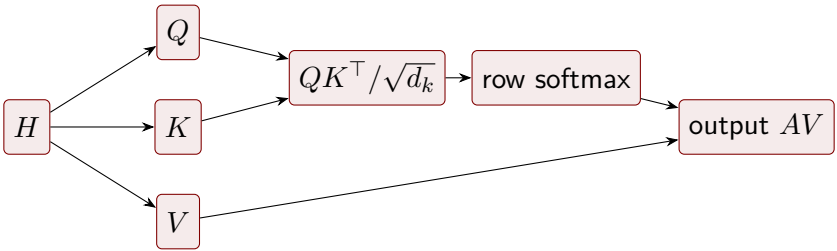
Attention lets the current prediction read all token states: $c_t = \sum_{j=1}^{T} \alpha_{tj} h_j$.

Fixed-context motivation adapted from the IP Paris Deep Learning attention lecture.

# Can every word look at every other word?

Cross-attention reads encoder states. In the self-attention below, $Q, K, V$ all come from $H \in \mathbb{R}^{T \times d_{\text{model}}}$:

$$Q = HW_Q, \qquad K = HW_K, \qquad V = HW_V.$$




$$\boxed{\text{Attn}(Q, K, V; M) = \text{softmax}_{\text{row}}\left(QK^\top/\sqrt{d_k} + M\right)V.}$$

Each output row is a weighted combination of value vectors.

Vaswani et al. (2017).

# How do we scale scores and block the future?

If coordinates of $q_i$ and $k_j$ are independent, centered, and have unit variance, then

$$\mathrm{Var}(q_i^\top k_j) = d_k.$$

Dividing by $\sqrt{d_k}$ keeps score variance of order one under this simple calculation. For causal prediction, the mask for four positions is

$$M = \begin{pmatrix} 0 & -\infty & -\infty & -\infty \\ 0 & 0 & -\infty & -\infty \\ 0 & 0 & 0 & -\infty \\ 0 & 0 & 0 & 0 \end{pmatrix}.$$

The $i$th output can combine values only from positions $j \leq i$. Every row still has at least one allowed key.

Vaswani et al. (2017).

# How does attention know token order?

Without positions or a mask, shared-projection self-attention is permutation equivariant. For a permutation matrix $P$,

$$A(PQ, PK, PV) = PA(Q, K, V), \quad A(Q, K, V) = \mathrm{Attn}(Q, K, V; 0).$$

The score matrix transforms as $QK^\top \mapsto PQK^\top P^\top$; row softmax simply permutes rows and columns. To encode order, add position vectors $p_t$ to token embeddings. The original sinusoidal choice includes

$$p_{t,2j} = \sin\left(t/10000^{2j/d_{\text{model}}}\right), \quad p_{t,2j+1} = \cos\left(t/10000^{2j/d_{\text{model}}}\right).$$

A causal mask also supplies an order relation.

Vaswani et al. (2017).

# Attention permutation proof

Let $S = QK^\top/\sqrt{d_k}$ and let $P$ permute token positions. Then

$$(PQ)(PK)^\top/\sqrt{d_k} = PSP^\top.$$

Because permuting rows and the entries within each row commutes with row softmax,

$$\mathrm{softmax}_{\mathrm{row}}(PSP^\top) = P\,\mathrm{softmax}_{\mathrm{row}}(S)P^\top.$$

Multiplying by $PV$ proves $A(PQ, PK, PV) = PA(Q, K, V)$. This argument assumes no position information and no causal mask.

# How do attention heads form a layer?

Different heads learn separate query, key, and value projections:

$$\mathrm{MHA}(H) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_r) W_O.$$

The original post-normalization encoder block uses

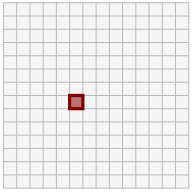
$$U = \mathrm{LN}\big(H + \mathrm{MHA}(H)\big),$$

$$H^+ = \mathrm{LN}\big(U + \mathrm{FFN}(U)\big).$$

$\mathrm{FFN}$ applies the same two-layer feedforward map to each position. The original decoder also has masked self-attention and cross-attention to encoder states.

Vaswani et al. (2017), original encoder–decoder architecture.

# The product photo can also become a token sequence



Resized product photo:
$224 \times 224 \times 3$, split into $14 \times 14$ patches of $16 \times 16$ pixels.

$$\boxed{(224/16)^2 = 196 \text{ patch tokens}}$$

Each patch has $16 \cdot 16 \cdot 3 = 768$ raw values. A shared linear map embeds it in $\mathbb{R}^D$; position embeddings mark where patches came from.

$$\{e_j + p_j\}_{j=1}^{196} \xrightarrow{\text{full self-attention}} \{z_j\}_{j=1}^{196} \xrightarrow{\text{pool + linear head}} s(x).$$

A $16 \times 16$ convolution with stride $16$ implements the shared patch projection. Attention then mixes information between any two patches.

Original diagram; ViT construction from Stanford CS231n, Lecture 8.

# Large language models

# A language model for the product description

Our product example now provides text:
*A red cotton coat with a hood and two pockets.*

A prompt can ask the model to summarize it, extract its attributes, or answer *Does this coat have a hood?* A language model assigns probabilities to token sequences and can generate a continuation one token at a time. Here we study **autoregressive, decoder-only LLMs**: large neural language models trained on broad text corpora, then often adapted to follow instructions. Other language-model architectures also exist.

Brown et al. (2020), Sec. 2. Product description and prompts are illustrative.

# Tokenization and the vocabulary

A tokenizer maps text to IDs $x_1, \ldots, x_T$ in a fixed vocabulary $\mathcal{V}$, with $V = |\mathcal{V}|$.

| Text | Illustrative pieces | Illustrative IDs |
|---|---|---|
| `waterproof` | `water`, `proof` | $[7, 29]$ |
| `coat` | `coat` | $[106]$ |

- A token can be a word, part of a word, or a byte sequence.
- BPE builds a vocabulary by repeatedly merging frequent adjacent units.
- The embedding table maps ID $x_t$ to a learned vector $E_{x_t} \in \mathbb{R}^d$.

Sennrich et al. (2016), Sec. 3; byte-level BPE: Radford et al. (2019), Sec. 2.2. Table is a toy segmentation, not a particular tokenizer's output.

# Autoregressive language modeling

Let $x_0$ denote a beginning-of-sequence token. The chain rule gives

$$\boxed{p_\theta(x_{1:T}) = \prod_{t=1}^{T} p_\theta(x_t \mid x_0, \ldots, x_{t-1}).}$$

An end-of-sequence token can be included in $x_{1:T}$ to model stopping. For a prompt $c$ and response $y = (y_1, \ldots, y_m)$,

$$\pi_\theta(y \mid c) = \prod_{t=1}^{m} p_\theta(y_t \mid c, y_{<t}).$$

The same conditional model assigns likelihoods during training and supplies the next-token distribution during generation.

Radford et al. (2019), Sec. 2, Eq. (1).

# A decoder-only Transformer

For $T$ input positions, let $H^0 \in \mathbb{R}^{T \times d}$ contain token embeddings plus position vectors. A pre-normalization block computes

$$\widetilde{H}^\ell = H^{\ell-1} + \mathrm{MHA}_{\mathrm{causal}}\big(\mathrm{LN}(H^{\ell-1})\big),$$

$$H^\ell = \widetilde{H}^\ell + \mathrm{FFN}\big(\mathrm{LN}(\widetilde{H}^\ell)\big).$$

- Causal attention at position $t$ reads only positions $j \le t$.
- Residual paths carry the representation through $L$ blocks.
- A final layer normalization supplies hidden vectors $h_t \in \mathbb{R}^d$.

Radford et al. (2019), Sec. 2.3. This is a representative GPT-style block; position encodings and block details vary across models.

# Vocabulary logits and shifted targets

The hidden vector at position $t$ predicts the *next* token:

$$z_t = W_{\text{vocab}} h_t + b \in \mathbb{R}^V, \qquad p_\theta(x_{t+1} = v \mid x_{0:t}) = \text{softmax}(z_t)_v.$$

| | | | | | |
|---|---|---|---|---|---|
| Input | <bos> | The | coat | is | red |
| Target | The | coat | is | red | <eos> |

During **teacher forcing**, the input uses the known training tokens. The causal mask prevents a hidden vector from reading its target. All known positions can be processed in parallel within a layer.

Vaswani et al. (2017), Secs. 3.1, 3.4; illustrative word-level sequence.

# Language-model pretraining

# The pretraining objective and data

For a corpus $\mathcal{D}$, let $\mathcal{I}_s$ be the valid prediction positions in document $s$, and $N_{\text{tok}} = \sum_s |\mathcal{I}_s|$.

$$\mathcal{L}_{\text{pre}}(\theta) = -\frac{1}{N_{\text{tok}}} \sum_{s \in \mathcal{D}} \sum_{t \in \mathcal{I}_s} \log p_\theta(x_{t+1}^{(s)} \mid x_{0:t}^{(s)}).$$

- The text supplies its own next-token labels.
- Minibatch gradient updates fit the model across many documents.
- Corpus filtering, deduplication, and the mixture of sources affect the training distribution.
- Held-out documents measure generalization. Padding positions are excluded from this loss.

Brown et al. (2020), Secs. 2.2–2.3, 4. Loss written as a token average for this lecture.

# Model size, data, and training compute

For dense Transformer training, a common compute approximation is

$$C \approx 6ND,$$

where $N$ is the number of parameters and $D$ the number of training tokens. An empirical loss model is

$$\widehat{\mathcal{L}}(N, D) = L_\infty + \frac{A}{N^\alpha} + \frac{B}{D^\gamma}, \qquad A, B, \alpha, \gamma > 0.$$

At fixed $C$, increasing $N$ leaves fewer tokens $D$ to train on. Compute allocation therefore trades model size against data exposure. These are fitted relations for particular architectures, data, and training procedures. Inference cost is a separate part of the budget.

Hoffmann et al. (2022), Sec. 3, Eqs. (2), (4). $C$ counts approximate training FLOPs.

# Held-out loss and perplexity

For $K$ held-out target tokens, using natural logarithms,

$$\mathcal{L}_{\text{test}} = -\frac{1}{K}\sum_{t=1}^{K} \log p_\theta(x_t \mid x_{<t}), \qquad \text{PPL} = \exp(\mathcal{L}_{\text{test}}).$$

Suppose the probabilities assigned to three observed targets are $1/2$, $1/4$, and $1/8$. Then

$$\mathcal{L}_{\text{test}} = \frac{\log 2 + \log 4 + \log 8}{3} = \log 4, \qquad \boxed{\text{PPL} = 4.}$$

Perplexity is the inverse geometric mean of the observed-token probabilities. Comparisons require the same tokenization and evaluation data.

Radford et al. (2019), Sec. 3.1. Numerical example is illustrative.

# Instruction tuning and preference alignment

# Pretraining and post-training

A representative assistant-training recipe uses different signals:

| Stage | Training examples | Learning signal |
|---|---|---|
| Pretraining | Text documents | Next-token prediction |
| Supervised fine-tuning | Prompt and desired response | Response likelihood |
| Preference alignment | Prompt and ranked responses | Preference objective |

Pretraining learns continuations across a broad corpus. Post-training uses demonstrations and preferences to shape responses to user requests. The architecture can stay the same while the data and objective change.

Ouyang et al. (2022), Sec. 3.1, Fig. 2. A representative recipe, not a requirement for every LLM.

# Supervised instruction fine-tuning

For a prompt $c$ and demonstrated response $y$, minimize

$$\mathcal{L}_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{(c,y)\sim\mathcal{D}_{\mathrm{SFT}}} \sum_{t=1}^{|y|} \log p_\theta(y_t \mid c, y_{<t}).$$

**Prompt:** Summarize this listing in one sentence: red cotton coat, hood, two pockets.

**Demonstrated response:** A red cotton coat with a hood and two pockets.

- The response loss mask selects which target tokens contribute.
- The causal attention mask controls which input positions are visible.

  In this recipe, prompt tokens condition the response but receive no direct loss.

Ouyang et al. (2022), Sec. 3.1. Example and response-only notation are illustrative.

# Preference pairs and a reward model

**Prompt $c$:** Is this cotton coat waterproof? The listing does not say.

**Preferred $y^+$:** Waterproofing is not specified in the listing.

**Rejected $y^-$:** Yes, it is fully waterproof. A scalar reward model $r_\phi(c, y)$ can fit these comparisons:

$$P_\phi(y^+ \succ y^- \mid c) = \sigma\big(r_\phi(c, y^+) - r_\phi(c, y^-)\big),$$

$$\mathcal{L}_{\mathrm{RM}}(\phi) = -\mathbb{E}_{(c,y^+,y^-)} \log \sigma\big(r_\phi(c, y^+) - r_\phi(c, y^-)\big).$$

Ouyang et al. (2022), Sec. 3.5, Eq. (1). Bradley–Terry preference model; pair above is illustrative.

# RLHF as a sequential decision problem

With fixed reward model $r_\phi$, reference policy $\pi_{\text{ref}}$, and $\beta > 0$, optimize

$$\max_\theta \; \mathbb{E}_{\substack{c \sim \mathcal{D} \\ y \sim \pi_\theta(\cdot|c)}} \left[ r_\phi(c, y) - \beta \log \frac{\pi_\theta(y \mid c)}{\pi_{\text{ref}}(y \mid c)} \right].$$

The expected log ratio penalizes KL divergence from the reference, often the SFT model.

| | |
|---|---|
| State | Prompt plus generated prefix $(c, y_{<t})$ |
| Action | Next token $y_t$ |
| Terminal reward | Learned score $r_\phi(c, y)$ |

InstructGPT uses PPO for policy optimization. A higher preference reward alone does not establish factual correctness.

Ouyang et al. (2022), Secs. 3.5, 5, Eq. (2). Core KL-regularized term shown; their PPO-ptx variant also mixes in pretraining gradients.

# Direct preference optimization

Define a response's log-likelihood ratio to the fixed reference:

$$\Delta_\theta(c, y) = \log \frac{\pi_\theta(y \mid c)}{\pi_{\text{ref}}(y \mid c)}.$$

DPO fits preferred and rejected responses directly:

$$\boxed{\mathcal{L}_{\text{DPO}}(\theta) = -\mathbb{E}_{(c,y^+,y^-)} \log \sigma\Big(\beta[\Delta_\theta(c, y^+) - \Delta_\theta(c, y^-)]\Big).}$$

- The loss favors a larger gap between preferred and rejected log-likelihood ratios.
- Optimization uses preference pairs without a separate reward network or online RL rollouts during DPO training.

Rafailov et al. (2023), Secs. 3–4, Eq. (7). Uses the Bradley–Terry preference model and $\beta > 0$; each $\pi$ is a full-response likelihood.

# Generation and applications

# Autoregressive generation

For a prompt $c$, generation repeatedly predicts and appends a token:

$$y_t \sim p_\theta(\cdot \mid c, y_{<t}), \qquad t = 1, 2, \ldots$$

A stopping token or length limit ends the loop.

- **Training:** the target sequence is known. Causal masking lets each transformer layer compute all token positions in parallel. Layers still run in order.
- **Generation:** later tokens depend on the generated prefix. A **KV cache** stores earlier keys and values at each layer and reuses them at later steps.

Brown et al. (2020); Shazeer (2019), incremental decoding and cached keys/values.

# Temperature and nucleus sampling

For logits $z_i$ and temperature $T > 0$, sample from

$$q_i(T) = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)} = \frac{p_i^{1/T}}{\sum_j p_j^{1/T}}, \qquad p = \mathrm{softmax}(z).$$

For the toy distribution $p = (0.6, 0.3, 0.1)$:

- At $T = 0.5$, $q \approx (0.783, 0.196, 0.022)$, rounded. Lower temperature concentrates probability on likely tokens.
- At $T = 1$, **top-$p$** with threshold $0.8$ keeps the smallest sorted prefix reaching that mass: the first two tokens. Renormalization gives $(2/3, 1/3, 0)$.

Greedy decoding selects the largest logit. Sampling changes output variation without updating the model's parameters.

Holtzman et al. (2020), nucleus sampling. Toy probabilities are illustrative.

# In-context learning

Examples in the prompt can specify a task and its output format:

| Product description | Category |
|---|---|
| “leather ankle boot” | shoe |
| “cotton button shirt” | shirt |
| “winter coat with a hood” | ? |

The model conditions its answer on these examples:

$$p_\theta(y \mid \text{instruction, examples, new description}).$$

**The weights $\theta$ stay fixed.** The examples change the context used for prediction. The same interface supports classification, extraction, and catalog writing. Evaluate it with the prompts and examples used for the task.

Brown et al. (2020), few-shot learning without gradient updates. Catalog examples are illustrative.

# Retrieval for product facts

An assistant answering "Does this coat have a hood?" can consult the product catalog. A simple **retrieve-then-prompt workflow** is:

1. Retrieve entries relevant to the question and product identifier.
2. Put the retrieved text and question into the model's context.
3. Generate an answer with references to the supporting entries.

$$p_\theta(y \mid \text{question, retrieved catalog text}).$$

Updating the catalog can change the available evidence while model weights stay fixed. Retrieval can miss the right entry, and generation can misread it. Check the answer against the retrieved evidence.

Lewis et al. (2020), retrieval-augmented generation. The workflow here is simplified retrieve-then-prompt.

# Tool use and external execution

For “Is product SKU42 available now?”, an assistant can request an inventory lookup:

```
get_inventory(product_id="SKU42").
```

1. The model produces a tool request with arguments.
2. An external runtime executes the request and returns a result.
3. The model conditions its answer on that returned result.

The API supplies the inventory value. Generating a request alone does not establish availability. If the lookup fails, the assistant should report that it could not check. Tools can also perform calculations or retrieve records that the model needs for an answer.

Schick et al. (2023), Toolformer. Inventory lookup is an illustrative application.

# Prompting, retrieval, and fine-tuning

The choice depends on what the product assistant needs to learn or access.

| Method | Weights | Information supplied | Catalog application |
|---|---|---|---|
| Prompting | Fixed | Prompt and examples | Choose labels or formatting |
| Retrieval | Fixed here | Retrieved records | Look up current product facts |
| Fine-tuning | Updated | Training examples | Learn a recurring response style |

These methods can work together. Fine-tuning can teach a consistent format, while retrieval supplies the facts for the current product. Facts stored through training still need checks when records change.

Brown et al. (2020); Lewis et al. (2020); Ouyang et al. (2022). Retrieval row describes the workflow above.

# Evaluation and limitations

Held-out loss and perplexity measure next-token prediction. They leave other questions about a product assistant to be tested:

- **Task success:** category accuracy or correct extraction.
- **Factual support:** whether catalog evidence supports each claim.
- **Instruction following:** format checks and response preferences.

Keep test examples out of training and prompt demonstrations. Test unfamiliar products and missing evidence. Fluent, likely text can contain false claims. Retrieved passages and tool results provide evidence, but the final answer still needs evaluation.

Brown et al. (2020), evaluation and contamination; Lin et al. (2022), TruthfulQA; Ouyang et al. (2022), human preferences.

# Summary

# Which architecture fits the relation we need?

| Model | Parameters shared over | Main relation in one layer |
|---|---|---|
| Dense feedforward | None by default | Learned feature composition |
| Convolutional | Image locations | Local neighborhoods |
| RNN / LSTM | Time steps | Ordered history |
| Transformer / ViT | Tokens or image patches | Content-based interactions |

Dense attention forms a $T \times T$ score matrix and therefore uses quadratic memory in sequence length for the scores. Its positions can be processed in parallel within one layer, while RNN states depend sequentially on previous states.

A decoder-only language model adds a vocabulary head and next-token training to a causal Transformer.

# Return to the product example

1. A dense ReLU model can learn nonlinear class scores; backpropagation supplies its gradients.
2. Augmentation and regularization help us choose a predictor for new photos.
3. A CNN reuses a local visual detector across image locations.
4. For the product caption, an RNN/LSTM tracks word order; attention relates words or image patches directly.
5. An LLM predicts text tokens. Instruction tuning shapes its responses; retrieval and tools supply product evidence.

**Class check:** Which changes the model's weights: adding prompt examples, retrieving a catalog entry, or fine-tuning?

# Selected sources


- Goodfellow, Bengio, and Courville (2016), *Deep Learning*, Chs. 6–10.
- Grisel and Ollion, *Deep Learning* lecture slides, Institut Polytechnique de Paris, course page, CC BY 4.0. Narrative sequence and motivating examples adapted; diagrams here are original.
- Stanford CS231n, *Deep Learning for Computer Vision*, course slides, Lectures 4, 5, 7, and 8. Worked examples and teaching sequence adapted; diagrams redrawn.
- Leshno et al. (1993), *Neural Networks* 6:861–867; Telgarsky (2016), COLT.
- Srivastava et al. (2014), *Dropout*, JMLR 15:1929–1958; Soudry et al. (2018), JMLR 19(70).
- Hochreiter and Schmidhuber (1997), *Neural Computation* 9(8); Gers et al. (2000), *Neural Computation* 12(10).
- Vaswani et al. (2017), *Attention Is All You Need*, NeurIPS 30; Dosovitskiy et al. (2021), *An Image is Worth 16x16 Words*, ICLR.
- Application examples: Ren et al. (2015), NeurIPS; Amodei et al. (2016), ICML; Xu et al. (2015), ICML.
- Ramesh et al. (2021), arXiv:2102.12092; Jumper et al. (2021), *Nature* 596; Silver et al. (2016), *Nature* 529.

# LLM sources


- Sennrich, Haddow, and Birch (2016), *Neural Machine Translation of Rare Words with Subword Units*, ACL.
- Radford et al. (2019), *Language Models are Unsupervised Multitask Learners*.
- Brown et al. (2020), *Language Models are Few-Shot Learners*, NeurIPS.
- Hoffmann et al. (2022), *Training Compute-Optimal Large Language Models*, NeurIPS.
- Ouyang et al. (2022), *Training Language Models to Follow Instructions with Human Feedback*, NeurIPS.
- Rafailov et al. (2023), *Direct Preference Optimization*, NeurIPS.
- Shazeer (2019), *Fast Transformer Decoding: One Write-Head is All You Need*.
- Holtzman et al. (2020), *The Curious Case of Neural Text Degeneration*, ICLR.
- Lewis et al. (2020), *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, NeurIPS.
- Schick et al. (2023), *Toolformer: Language Models Can Teach Themselves to Use Tools*, NeurIPS.
- Lin, Hilton, and Evans (2022), *TruthfulQA: Measuring How Models Mimic Human Falsehoods*, ACL.