Generative Artificial Intelligence

Lecture 1: GenAIs and VAEs

Seungjin Yang

Mon 3 – Wed 5, Aug 2026

Kyung Hee University

Outline

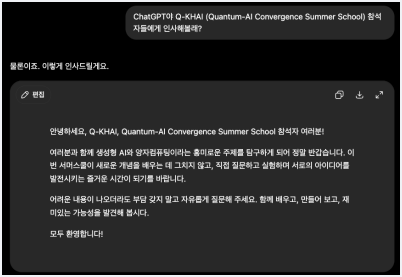
Introduction

What is a generative AI?

- A generative AI (GenAI) is a model that can generate new data samples that resemble the training data
- A discriminative model learns $p(y \mid x)$: given an input x, it predicts a label y
- A generative model learns $p(x)$: it draws new samples $x \sim p_\theta(x)$ that resemble the training data
- Same data, different question: *classify* or *create*

Figure 1: Not a photograph. A sample from a model trained on photographs.

Large Language Models



- Large language models (LLMs) are a type of generative AI that can generate text that resembles human writing
- An LLM samples the next token from $p_\theta(x_t \mid x_{<t})$, and repeating this produces a text
- The same mechanism underlies chat, code, and translation; we return to it in Lecture 2

LLMs for Vibe Coding

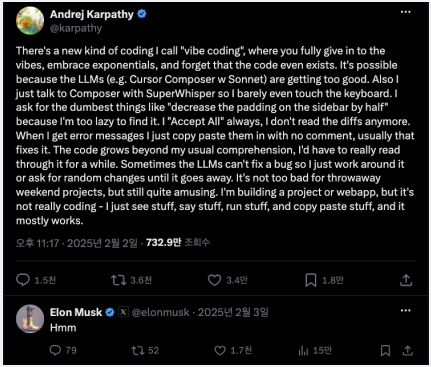
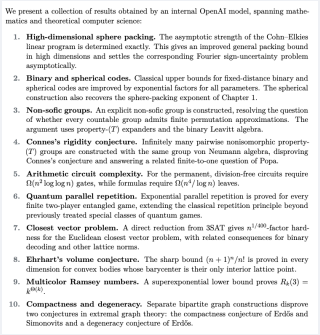


Figure 2: Andrej Karpathy's post coining "vibe coding"

Figure 3: Linus Torvalds' commit message

- Linus Torvalds says that vibe coding works better than manual coding for his non-kernel project

LLMs Solving Problems in Math and CS



We present a collection of results obtained by an internal OpenAI model, spanning mathematics and theoretical computer science:

1. **High-dimensional sphere packing.** The asymptotic strength of the Cohn–Elkies linear program is determined exactly. This gives an improved general packing bound in high dimensions and settles the corresponding Fourier sign-uncertainty problem asymptotically.
2. **Binary and spherical codes.** Classical upper bounds for fixed-distance binary and spherical codes are improved by exponential factors for all parameters. The spherical construction also recovers the sphere-packing exponent of Chapter 1.
3. **Non-sofic groups.** An explicit non-sofic group is constructed, resolving the question of whether every countable group admits finite permutation approximations. The argument uses property-(T) expanders and the binary Lexavitt algebra.
4. **Connes's rigidity conjecture.** Infinitely many pairwise nonisomorphic property-(T) groups are constructed with the same group von Neumann algebra, disproving Connes's conjecture and answering a related finite-to-one question of Popa.
5. **Arithmetic circuit complexity.** For the permanent, division-free circuits require $\Omega(n^2 \log n)$ gates, while formulas require $\Omega(n^2/\log n)$ leaves.
6. **Quantum parallel repetition.** Exponential parallel repetition is proved for every finite two-player entangled game, extending the classical repetition principle beyond previously treated special classes of quantum games.
7. **Closest vector problem.** A direct reduction from 3SAT gives $n^{1/400}$-factor hardness for the Euclidean closest vector problem, with related consequences for binary decoding and other lattice norms.
8. **Ehrhart's volume conjecture.** The sharp bound $(n+1)^n/n!$ is proved in every dimension for convex bodies whose barycenter is their only interior lattice point.
9. **Multicolor Ramsey numbers.** A superexponential lower bound proves $R_k(3) = k^{\Theta(k)}$.
10. **Compactness and degeneracy.** Separate bipartite graph constructions disprove two conjectures in extremal graph theory: the compactness conjecture of Erdős and Simonovits and a degeneracy conjecture of Erdős.

- LLMs can go beyond simply generating human-like text and solve problems in mathematics and theoretical computer science
- OpenAI announced that Astra, their next major model, has solved ten problems in mathematics and theoretical computer science [1]

[1] See the blogpost and paper for more details

Image Generation

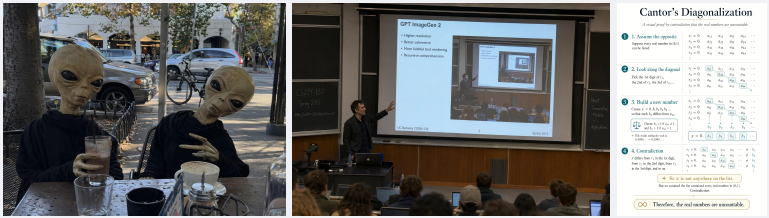


Figure 4: Source: Introducing ChatGPT Images 2.0

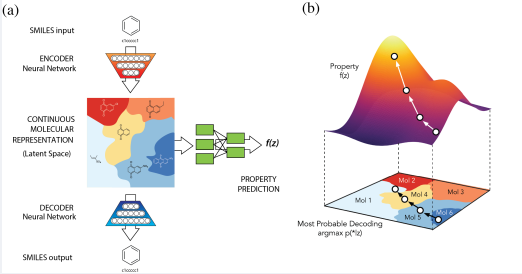
- Generative AI can also generate images from text prompts, such as ChatGPT Images and GEMINI Nano Banana
- Current image models render legible text and diagrams, not only photographs (right)

Video Generation

Figure 5: Screenshot from Seedance 2.5 demonstration video

- Seedance 2.5 can generate videos from text prompts
- A video is a sequence of images, so the dimension of one sample is multiplied by the number of frames
- The frames must also stay mutually consistent, so the model has to represent motion and object permanence, not only the appearance of a single frame

Generative AI for Drug Discovery



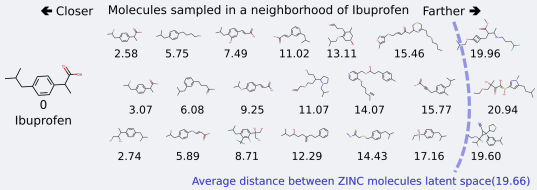


Figure 6: Molecules sampled near ibuprofen, with their latent-space distances shown below.

- GenAI enables the generation and optimization of novel drug candidates with desired chemical properties [1]

Physics

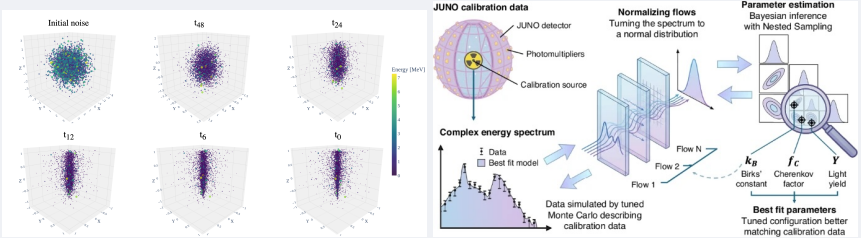


Figure 7: Left: Ref. [2] and Right: Ref. [3]

- Lattice field theory: normalizing flows propose field configurations directly, avoiding the critical slowing down of local-update Markov chains [4]
- Detector simulation: a generative model replaces the shower simulation of a calorimeter and runs orders of magnitude faster [5]
- Simulation-based inference: a generative model replaces the expensive simulator and learns to sample from the posterior distribution, enabling likelihood-free inference
- In each case the expensive object is a *sampler*, and the model learns to sample cheaply

Generative modelling

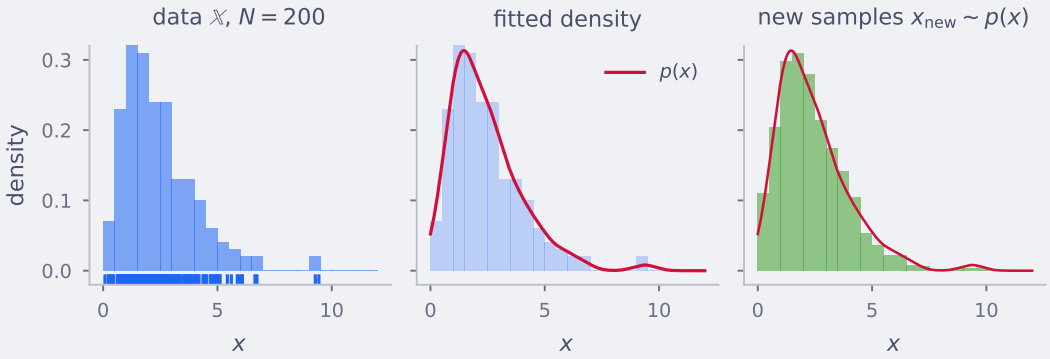
Goal of Generative Modelling

- We are given a dataset of N samples drawn from an unknown distribution

$$\mathbb{X} = \{x_1, \ldots, x_N\}, \qquad x_i \sim p_{\text{data}}(x) \tag{1}$$

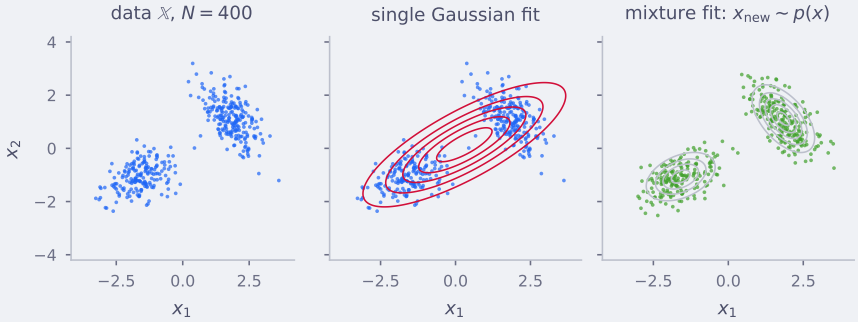
- p_{data} is never written down; the samples are all we ever see
- Goal: learn a model $p_\theta(x) \approx p_{\text{data}}(x)$ from $\mathbb{X}$ alone, where θ are the parameters
- Two things we want from p_θ
  - **density estimation**: evaluate $p_\theta(x)$ for a given x
  - **sampling**: draw new $x_{\text{new}} \sim p_\theta(x)$, cheaply
- Model families differ in which of the two they give
- Copying the training set gives a perfect fit and is useless: the new samples must resemble the data without being members of it

1D example



- $\mathbb{X} \subset \mathbb{R}$, one number per sample, $N = 200$
- Fit: count the samples in bins, or smooth them into a continuous density $p(x)$
- Sample: $x_{\text{new}} \sim p(x)$, by drawing $u \sim \mathcal{U}[0, 1]$ and inverting the cumulative distribution
- In one dimension the histogram already *is* a generative model: no network is needed

2D example



- $\mathbb{X} \subset \mathbb{R}^2$, and the data has two modes
- A single Gaussian is fitted correctly and is still wrong: it puts its mass *between* the modes, where no data was ever observed
- A mixture of two Gaussians has enough freedom, and its samples reproduce both modes
- The choice of model family matters as much as the fitting procedure

ND example

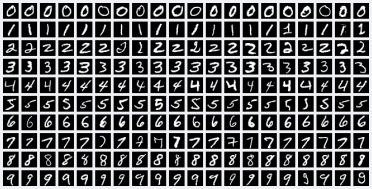


Figure 8: The MNIST dataset of handwritten digits, 28×28 pixels, $D = 784$.

- Binning the data space costs b^D cells for b bins per axis: the count explodes with the dimension D
- MNIST is a 28×28 image, so $D = 784$; even in black and white it has $2^{784} \approx 10^{236}$ possible images
- Against this: 6×10^4 training images, and $\approx 10^{80}$ atoms in the observable universe
- Almost every cell stays empty, so counting gives $p(x) = 0$ nearly everywhere: this is the curse of dimensionality
- Way out: replace the table of counts by a parametric $p_\theta(x)$, a neural network that shares structure between different x instead of treating each cell separately

Simulation as a generative model

- Physicsists already has a generative model: the simulator
- Given parameters θ of a theory, it draws samples $x \sim p_\theta(x)$ by sampling every step of a chain
- Randomness enters at every step, so the output is a sample and nothing else
- The density is an integral over all unobserved intermediate states z, and is intractable

$$p_\theta(x) = \int p_\theta(x, z)\,\mathrm{d}z \tag{2}$$

- So a simulator samples but cannot score: it cannot evaluate $p_\theta(x)$, be conditioned on an observation, or be inverted
- It is also slow:
- Plan for the rest of the lecture: learn p_θ from samples and let a neural network do the sampling

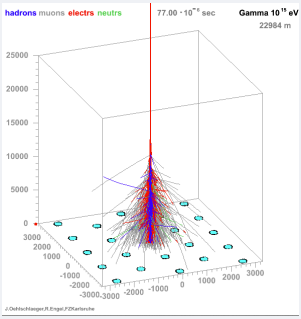


Figure 9: CORSIKA simulation of an extensive air shower initiated by a 1 PeV gamma ray. See CORSIKA Shower Movies for more examples.

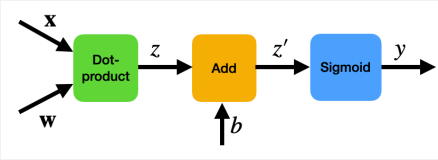
Deep Learning Refresher

Data

- Every data point is a vector, or a collection of vectors
  - image: a 2D array of pixels
  - text: a sequence of token indices, each mapped to a learned vector
  - physics event: positions and momenta of particles
- The network never sees an “image”; it sees numbers, and the structure has to be put back in by the model
- Notation: x is one data point, $\mathbb{X} = \{x_1, \ldots, x_N\}$ the dataset, θ the parameters of the model

Perceptron

- In 1958, Frank Rosenblatt introduced the perceptron, a binary classifier built from one artificial neuron [6]
- Perceptron: $y = f(\mathbf{x}; \theta) = \sigma\left(\mathbf{w}^T\mathbf{x} + b\right)$
  - $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n$, and $y, b \in \mathbb{R}$
  - $\theta = \{\mathbf{w}, b\}$ are the trainable parameters
  - $\mathbf{w}^T\mathbf{x} + b$ is called the logit
- σ is a non-linear function called the activation
  - e.g. the sigmoid, $\sigma(z) = 1/(1 + e^{-z})$
- Without σ the perceptron is a linear model, and a stack of linear models is again linear



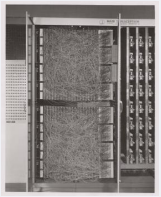


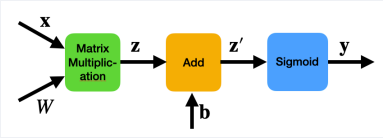
Figure 10: The Mark I Perceptron machine, in which the weights were motor-driven potentiometers

Perceptron: Many-to-Many

- A perceptron is extended to return a vector instead of a scalar

$$\mathbf{y} = \sigma\left(W\mathbf{x} + \mathbf{b}\right) = (\sigma(z_1), \ldots, \sigma(z_m)) \tag{3}$$

  - $\mathbf{x} \in \mathbb{R}^n$, $W \in \mathbb{R}^{m \times n}$, and $\mathbf{y}, \mathbf{b} \in \mathbb{R}^m$
- The activation is applied element-wise, so the layer is one matrix product followed by one non-linearity
- This is a linear or dense layer, the basic building block of a network
- The sigmoid is rarely used inside modern networks; ReLU, $\max(0, z)$, is the default, since its gradient does not vanish for large z



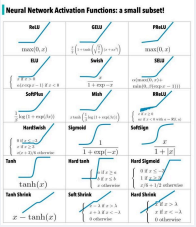
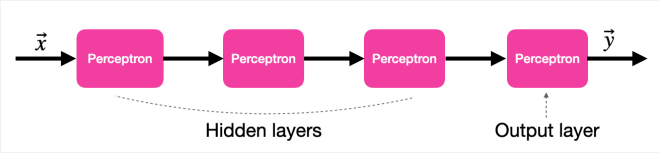


Figure 11: A small subset of the activation functions in use

Multi-Layer Perceptron



- A multi-layer perceptron (MLP), also called a feedforward neural network, is a composition of such layers

$$f_\theta(\mathbf{x}) = \sigma_L \circ W_L \circ \cdots \circ \sigma_1 \circ W_1(\mathbf{x}) \tag{4}$$

- Universal approximation theorem: an MLP with one hidden layer and enough hidden units approximates any continuous function on a compact domain to arbitrary accuracy [7]
- The theorem says the function exists; it does not say that gradient descent will find it, nor how many units are needed
- Depth is what makes this practical: "representation-learning methods with multiple levels of representation, obtained by composing simple but non-linear modules" [8]
- Try it: https://playground.tensorflow.org

Loss Function

- A loss function $J(\theta)$ measures how far the predictions are from the targets, averaged over the dataset
- Regression: mean squared error

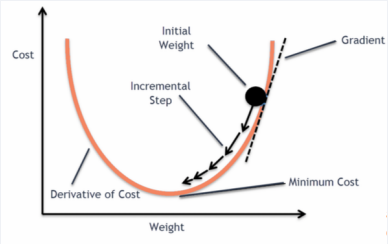
$$J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{2} \left(f_\theta(x_i) - t_i \right)^2 \tag{5}$$

- Classification: cross entropy, where $f_\theta(x_i)_k$ is the predicted probability of class k

$$J(\theta) = -\frac{1}{N} \sum_{i=1}^{N} \sum_{k=1}^{K} t_{ik} \log f_\theta(x_i)_k \tag{6}$$

- Training means finding $\theta^* = \arg\min_\theta J(\theta)$
- The loss is the only place where the goal of the task is stated, so the choice of loss defines what the model learns
- Both of these losses are derived later in the lecture as negative log-likelihoods of a probabilistic model

Gradient Descent



- Gradient descent is an iterative method: start from a random θ and repeatedly step against the gradient

$$\theta \leftarrow \theta - \eta \nabla_\theta J(\theta) \tag{7}$$

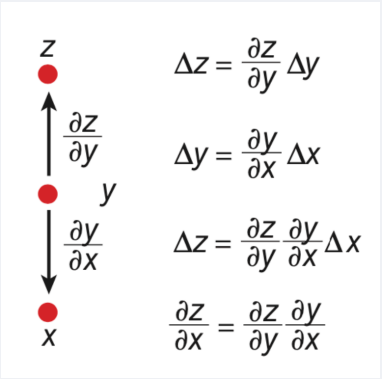
  where η is the learning rate
- η is a hyperparameter: it is not adjusted by gradient descent itself, and is tuned by hand, typically between 10^{-4} and 10^{-2}
- Computing $\nabla_\theta J$ over all N samples is too expensive, so the gradient is estimated on a random mini-batch of a few hundred samples: stochastic gradient descent
- In practice one uses an adaptive variant such as Adam, and the gradient is computed automatically by PyTorch

Backpropagation

- A network has millions of parameters; how is $\partial J / \partial \theta_j$ obtained for each of them?
- Backpropagation is the chain rule applied to the composition of layers

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \tag{8}$$

- A small change in an early parameter propagates forward through the layers into a change in the loss; the chain rule multiplies the per-layer factors
- Key point: the cost of one gradient is of the same order as one forward pass, independent of the number of parameters
- This is why deep networks are trainable at all
- One training step is: forward, backward, update. Repeat over mini-batches until the loss stops improving



Deterministic

- Everything so far is a deterministic map $f_\theta : x \mapsto y$: the same input always returns the same output
- For the supervised tasks this is what is wanted – one image has one label
- Generation is a one-to-many problem: “a photo of a cat” has no single correct answer, but a whole set of plausible ones
- Write the generator as $g_\theta : y \mapsto x$, from a condition y (a label, a prompt) to a data point x
- Trained under MSE, its optimum is the *mean* of the valid answers

$$g_\theta(y) \to \mathbb{E}[x \mid y] \tag{9}$$

  the average over all cats, which is a blur and not a cat
- Averaging over valid answers is exactly the failure mode that returns later as blurry VAE samples
- Two consequences
  - the output must be a distribution over x, not a single point
  - the randomness must live inside the model, so that the same input can be run twice and give two different samples

Probabilistic Modelling

- In the probailistic modelling framework, a neural network can parametrises a distribution over data x[2] instead of a point prediction
- Unconditional model $p_\theta(x)$
  - what it means: how plausible the data point x is under the model
  - example: $p_\theta(x)$ trained on MNIST. A sample is some handwritten digit, but which digit is not under our control
- Conditional model $p_\theta(x \mid y)$
  - what it means: the distribution of x once a condition y is fixed
  - y is a class label, a text prompt, a galaxy morphology, or a set of physics parameters
  - example: $p_\theta(x \mid y = 7)$ trained on MNIST. Every sample is a seven, and they differ in stroke, slant and thickness
- Conditioning is how generation is controlled in practice, and it returns in the conditional VAE and, in Lecture 2, as prompting
- Remaining problem: x lives in $\mathbb{R}^{784}$, and writing down $p_\theta(x)$ there directly is hopeless – the next section introduces the structure that makes it possible

[2] In this lecture, x denotes a data point, which can be a vector, an image, a sequence, etc.

Manifold Hypothesis

Manifold Hypothesis

- High-dimensional data often has low-dimensional structure
- An image may have many pixels, but its variation is controlled by far fewer factors:
  - object identity, shape, pose, lighting, background
- Manifold hypothesis
  - Natural data points lie near a low-dimensional manifold embedded in a high-dimensional observation space
  - Coordinates on this manifold form a latent space: a compact set of variables that parametrize the data's true degrees of freedom
- Two analogies:
  - In particle physics, a few degrees of freedom, such as position, momentum, energy, and spin, are turned by quantum processes and detector noise into a high-dimensional vector of channels or hits
  - In biology, DNA provides the generative information, growth and environment add stochastic variation, and a photograph records it as many pixels

What Is a Manifold?

- A manifold is a space that is locally simple, even if it is globally curved
- A generative model learns this structured region, not the whole ambient space

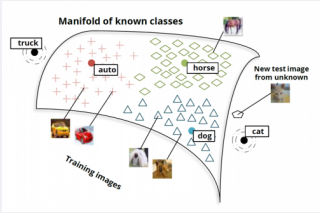


Figure 12: Source: Ref. [9]

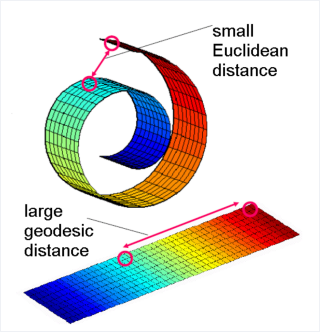
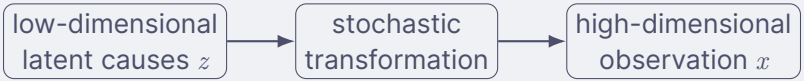


Figure 13: Ambient dimension: (x, y, z) in $\mathbb{R}^3$. Manifold dimension: (s, t) in $\mathbb{R}^2$, where $(t, s) \mapsto (x, y, z) = (t\cos t, s, t\sin t)$. Source: Ref. [10]

Latent Causes and Observations

- The manifold hypothesis suggests a causal structure:



- latent variables z represent hidden generative factors;
- observations x are measured data, such as pixels or detector signals;
- stochasticity represents noise, unresolved variables, or measurement uncertainty
- Supervised models learn only the subset of z relevant to a task, whereas generative models aim to learn the full latent space
- This fuller latent representation is why generative pretraining now also improves supervised performance, e.g. GPT

Autoencoders

Autoencoder

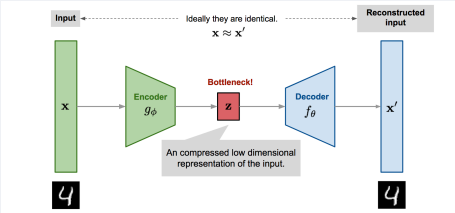


Figure 14: Source: Ref. [11].

- An autoencoder is a neural network trained to reproduce its own input through a narrow intermediate layer [12, 13]
- Data: $x \in \mathbb{R}^d$; latent code: $z \in \mathbb{R}^k$ with $k \ll d$
- The encoder $z = g_\phi(x)$ compresses the input into a low-dimensional code
- The decoder $\hat{x} = f_\theta(z)$ reconstructs the input from that code
- Training needs no labels, only the data itself
- The narrow layer forces the network to keep the informative structure of the data and discard the rest
- This is nonlinear dimensionality reduction
- Autoencoders were introduced to learn useful representations without labels by reconstructing their inputs

Architecture

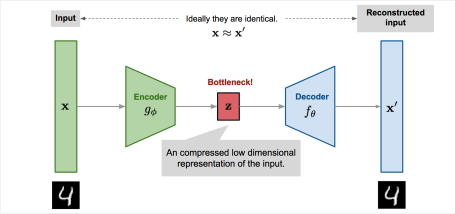


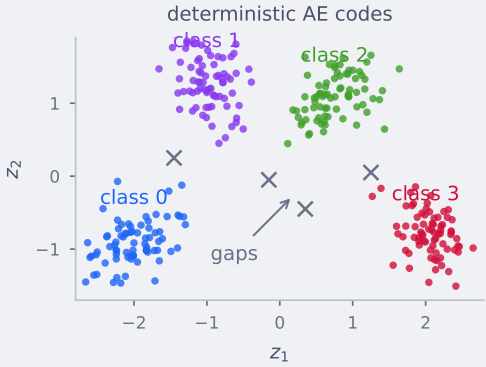
Figure 15: Source: Ref. [11].

- The output is not copied pixel by pixel; it is rebuilt from the code z.
- The layer of size k is the bottleneck
- If k is too small, fine details are lost first.
- If k is too large, the network might collapse to the identity function and learn nothing about the data
- Good reconstruction means the code keeps the main factors of variation.
- The two networks are trained together to minimize the mean squared error (MSE)

$$L_{\mathrm{AE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \left(x^{(i)} - f_\theta \left(g_\phi(x^{(i)}) \right) \right)^2 . \tag{10}$$

Why This Is Not Yet a Generative Model

- An autoencoder can compress and reconstruct, but it cannot generate
- The encoder maps each data point to a single point z.
- The model never defines a distribution over z.
- The occupied region of latent space is irregular and contains gaps.
- A code taken at random usually decodes to something meaningless.
- A prior over the latent variable, and a training objective that makes the encoded data agree with that prior



Variational Autoencoders

Latent Variable Model

- Assume the data are produced in two steps:

$$z \sim p(z), \qquad x \sim p_\theta(x \mid z) \tag{11}$$

  - $p(z)$: simple prior, usually $\mathcal{N}(0, I)$.
  - $p_\theta(x \mid z)$: decoder, a distribution rather than a single output
- The likelihood of one observation is

$$p_\theta(x) = \int_z p_\theta(x \mid z)\, p(z)\, dz. \tag{12}$$

- The optimal parameters maximize the likelihood of the training data:

$$\theta^* = \arg\max_\theta \sum_{i=1}^{N} \log p_\theta(x^{(i)}). \tag{13}$$

- The integral runs over all of latent space and cannot be computed exactly, so maximum likelihood is not directly available

The Posterior Is Also Intractable

- Inferring the latent cause of an observation requires

$$p_\theta(z \mid x) = \frac{p_\theta(x \mid z)\, p(z)}{p_\theta(x)}, \tag{14}$$

- $p_\theta(z \mid x)$ contains the same intractable integral in its denominator
- The variational autoencoder introduces a second network that approximates it [14]:

$$q_\phi(z \mid x) \approx p_\theta(z \mid x). \tag{15}$$

- Inference is replaced by regression: instead of solving for the posterior of each data point, one network predicts it for every input

Probabilistic Encoder and Decoder

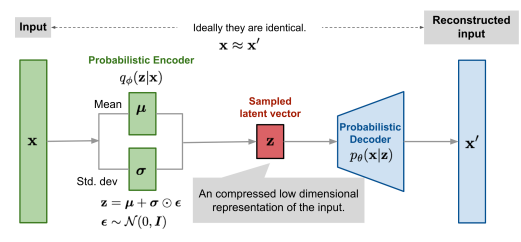


Figure 16: Source: Ref. [11]

$$q_\phi(z \mid x) = \mathcal{N}\Big(\mu_\phi(x),\, \operatorname{diag}\sigma_\phi^2(x)\Big), \qquad p(z) = \mathcal{N}(0, I). \tag{16}$$

- Each input is mapped to a small cloud in latent space, not to a single point
- Neighbouring inputs give overlapping clouds, which fills the gaps left by an ordinary autoencoder

Evidence Lower Bound (1/2)

$$\log p_\theta(x) = \log \int p_\theta(x, z)\, dz \tag{17}$$

$$= \log \int q_\phi(z \mid x)\, \frac{p_\theta(x, z)}{q_\phi(z \mid x)}\, dz \tag{18}$$

$$= \log \mathbb{E}_{q_\phi(z|x)} \left[\frac{p_\theta(x, z)}{q_\phi(z \mid x)} \right] \tag{19}$$

$$\geq \mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x, z)}{q_\phi(z \mid x)} \right] \quad \text{(Jensen's inequality)} \tag{20}$$

- Multiplying and dividing by $q_\phi(z \mid x)$ turns the intractable integral into an expectation over the encoder, which can be estimated by sampling
- Jensen's inequality moves the concave $\log$ inside the expectation, which can only decrease the value: the result is a lower bound on $\log p_\theta(x)$

Evidence Lower Bound (2/2)

$$\log p_\theta(x) \geq \mathbb{E}_{q_\phi(z|x)}\left[\log \frac{p_\theta(x,z)}{q_\phi(z \mid x)}\right] \tag{21}$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x \mid z) + \log p(z) - \log q_\phi(z \mid x)\right] \tag{22}$$

$$= \mathbb{E}_{q_\phi(z|x)}\left[\log p_\theta(x \mid z)\right] - D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p(z)) \tag{23}$$

$$\equiv \mathcal{L}_{\mathrm{ELBO}}(x). \tag{24}$$

- Expanding $p_\theta(x, z) = p_\theta(x \mid z)\, p(z)$ splits the bound into two terms
- The first term rewards reconstructing x from its code; the second keeps the encoder close to the prior
- Both terms are computable, so the bound can be maximized by gradient ascent; it becomes the training objective on the next frames

Kullback-Leibler Divergence

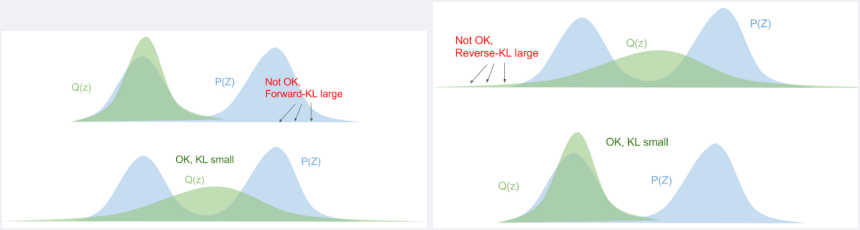


Figure 17: Source: Ref. [15]

- The **Kullback–Leibler (KL) divergence** measures how different a distribution $q(x)$ is from a reference distribution $p(x)$:

$$D_{\mathrm{KL}}(p \parallel q) = \mathbb{E}_{x\sim p}\left[\log \frac{p(x)}{q(x)}\right] \tag{25}$$

- $D_{\mathrm{KL}}(p\|q) \geq 0$
- $D_{\mathrm{KL}}(p\|q) = 0$ only when $p = q$ almost everywhere
- It is **asymmetric**:

$$D_{\mathrm{KL}}(p\|q) \neq D_{\mathrm{KL}}(q\|p). \tag{26}$$

- It can be interpreted as the information lost when q is used to approximate p

The VAE Loss

- Training minimizes the negative bound, averaged over the data:

$$L_{\mathrm{VAE}}(\theta, \phi) = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] + D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p(z)) \,. \tag{27}$$

- Reconstruction loss

$$-\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] \tag{28}$$

  - The code must keep enough information to rebuild x
  - Alone, this term pushes the clouds apart and shrinks their width to zero
- Regularization: $D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p(z))$
  - The codes must stay close to the prior
  - Alone, this term collapses every input to the same distribution and destroys the information in z
- The two terms pull in opposite directions
- Their balance decides how usable the latent space is

Gaussian Decoder

- The bound holds for any decoder; a concrete choice is needed before it can be evaluated
- For continuous data, take a Gaussian with fixed isotropic variance:

$$p_\theta(x \mid z) = \mathcal{N}\Big(\mu_\theta(z),\, \sigma^2 I\Big)\,, \qquad \mu_\theta(z) = f_\theta(z). \tag{29}$$

- The network predicts only the mean; σ is a hyperparameter, not learned
- The reconstruction is the mean itself:

$$\hat{x} = \mu_\theta(z) = \mathbb{E}[x \mid z]\,. \tag{30}$$

  - The remaining noise $\sigma\epsilon$ is isotropic and carries no structure, so it is dropped at generation time
- σ sets the weight between the two loss terms, as the next frames show

Gaussian Decoder – Reconstruction Loss

- The log-likelihood of a Gaussian is

$$\log p_\theta(x \mid z) = \log\left[\left(2\pi\sigma^2\right)^{-D/2} \exp\left(-\frac{\|x-\mu_\theta(z)\|^2}{2\sigma^2}\right)\right] \tag{31}$$

$$= -\frac{1}{2\sigma^2}\|x-\mu_\theta(z)\|^2 - \frac{D}{2}\log\left(2\pi\sigma^2\right) \tag{32}$$

$$\propto -\|x-\mu_\theta(z)\|^2 \tag{33}$$

  - D is the dimension of x
- Taking the negative expectation,

$$L_{\text{recon}} = -\mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x \mid z)] = \frac{1}{2\sigma^2}\,\mathbb{E}_{q_\phi(z|x)}\left[\|x-\hat{x}\|^2\right] + \text{const.} = \|x-\mu_\theta(z)\|^2 \tag{34}$$

- A fixed-variance Gaussian decoder turns the reconstruction term into a mean squared error
- The constant depends on neither θ nor ϕ, so it does not affect the gradient
- The expectation is estimated with one sample $z \sim q_\phi(z \mid x)$ per data point
- $1/(2\sigma^2)$ is the weight against the KL term: small σ favours reconstruction, large σ favours the prior

Reconstruction Loss for Other Decoders

- The reconstruction loss depends on the decoder distribution $p_\theta(x \mid z)$
- Gaussian decoder $\Rightarrow$ L2 loss
- Laplace decoder $\Rightarrow$ L1 loss
- Bernoulli decoder $\Rightarrow$ Binary cross-entropy reconstruction loss

KL Term

- Both distributions are diagonal, so the divergence factorizes over the k latent dimensions
- Writing μ_j, σ_j for the encoder output,

$$D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p(z)) = \sum_{j=1}^{k} \mathbb{E}_q[\log q(z_j) - \log p(z_j)] \tag{35}$$

$$\mathbb{E}_q[\log q(z_j)] = -\tfrac{1}{2} \log\left(2\pi\sigma_j^2\right) - \tfrac{1}{2} \tag{36}$$

$$\mathbb{E}_q[\log p(z_j)] = -\tfrac{1}{2} \log(2\pi) - \tfrac{1}{2}\left(\mu_j^2 + \sigma_j^2\right) \qquad \text{using } \mathbb{E}_q[z_j^2] = \mu_j^2 + \sigma_j^2 \tag{37}$$

$$\Rightarrow \quad D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p(z)) = \frac{1}{2} \sum_{j=1}^{k} \left(\mu_{\phi,j}^2 + \sigma_{\phi,j}^2 - \log \sigma_{\phi,j}^2 - 1\right). \tag{38}$$

- Closed form: no sampling and exact gradients, unlike the reconstruction term
- Zero only at $\mu_{\phi,j} = 0$ and $\sigma_{\phi,j} = 1$, where the encoder matches the prior
- The network predicts $\log \sigma_\phi^2$ rather than σ_ϕ, which keeps the variance positive and the logarithm stable

Reparameterization Trick

- The loss is an expectation over $q_\phi(z \mid x)$, whose distribution depends on ϕ
- Backpropagation cannot pass through such a random draw
- Write the sample instead as a deterministic function of the parameters and of an independent noise variable [14]:

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I). \tag{39}$$

- The noise is an input, not a parameter, so the gradient flows along the deterministic path into ϕ.

Generating New Data

- After training, the encoder is no longer needed:

$$z \sim \mathcal{N}(0, I), \qquad x \sim p_\theta(x \mid z). \tag{40}$$

- This works because the KL term forced the encoded data to cover the prior, so a prior sample lands in a region the decoder has seen
- Sampling is a single forward pass, which makes generation cheap
- The price is an approximation: training maximizes a lower bound, not the likelihood itself, and samples are often blurred

The Learned Latent Space: MNIST

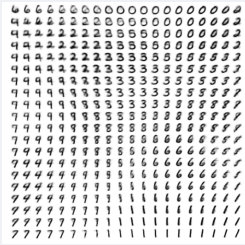


Figure 18: Source: Ref. [14]

- The figure shows a 2D latent space learned by a VAE on MNIST dataset

The Learned Latent Space: Frey Faces

Figure 19: Source: Ref. [14]

- The figure shows a 2D latent space learned by a VAE on Frey Faces dataset

Why Are VAE Images Blurry?

- With a fixed-variance Gaussian decoder,

$$-\log p_\theta(x|z) \propto \|x - \mu_\theta(z)\|^2, \tag{41}$$

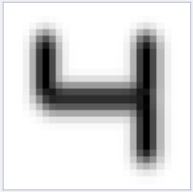
  so the decoder is trained with MSE
- The optimal prediction under MSE is the *pixel-wise average*

$$\mu_\theta^*(z) = \mathbb{E}[x|z]. \tag{42}$$

- If many sharp images are compatible with the same code z, the decoder outputs their average; averaging different edges and positions produces blur

one real image

VAE sample



- The KL term also limits how much detail can be stored in z

Summary: Training Step

1. $x \sim p(x)$ *(sample data from the training set)*
2. $\mu_\phi, \log \sigma_\phi^2 = g_\phi(x)$ *(encode)*
3. $z = \mu_\phi + \sigma_\phi \odot \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$ *(reparameterize)*
4. $\hat{x} = \mu_\theta = f_\theta(z)$ *(decode)*
5. $L_{\text{VAE}}(\theta, \phi) = L_{\text{recon}} + L_{\text{KL}}$ *(compute loss)*
   - $L_{recon}(\theta, \phi) = -\log p_\theta(x|z) = \frac{1}{2\sigma^2}\|x - \hat{x}\|^2$
   - $L_{KL}(\phi) = D_{KL}(q_\phi(z|x)||p(z)) = \frac{1}{2}\sum_{j=1}^{k}(\mu_{\phi,j}^2 + \sigma_{\phi,j}^2 - \log \sigma_{\phi,j}^2 - 1)$, where k denotes the dimension of the latent space
6. $w \leftarrow w - \eta \nabla_w L_{\text{VAE}}(\theta, \phi)$, where $w = \{\theta, \phi\}$ *(update parameters)*

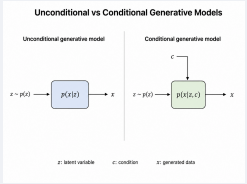
Summary: Generation Step

1. $z \sim p(z) = \mathcal{N}(0, I)$ *(sample code from the prior)*
2. $\hat{x} = \mu_\theta = f_\theta(z)$ *(decode)*
3. $x \sim p_\theta(x|z) = \mathcal{N}(\mu_\theta, \sigma^2 I)$, or simply take $x = \hat{x}$ *(sample or take the mean)*

- The encoder g_ϕ is discarded: no input image is needed
- The prior replaces $q_\phi(z|x)$ as the source of codes, which the KL term made possible during training
- Generation is one forward pass of the decoder

Conditional Variational Autoencoders

Adding a Condition



- A VAE models $p_\theta(x)$; often the target is a conditional distribution

$$p_\theta(x \mid c), \tag{43}$$

  with c some external information: a class label, a physical parameter, a measurement
- Every distribution of the VAE simply gains c [16]:

$$z \sim p(z \mid c), \qquad x \sim p_\theta(x \mid z, c), \qquad q_\phi(z \mid x, c) \approx p_\theta(z \mid x, c). \tag{44}$$

- Division of labour: c carries the controlled information, z carries the remaining variation
- The simplest choice keeps the prior independent of the condition, $p(z \mid c) = \mathcal{N}(0, I)$; a learned conditional prior $p_\psi(z \mid c)$ is more flexible

Conditional ELBO

- The derivation of the previous section goes through unchanged with c held fixed everywhere:

$$\log p_\theta(x \mid c) \geq \mathbb{E}_{q_\phi(z \mid x, c)}[\log p_\theta(x \mid z, c)] - D_{\mathrm{KL}}(q_\phi(z \mid x, c) \,\|\, p(z \mid c)) \,. \tag{45}$$

- The loss is again reconstruction plus regularization:

$$L_{\mathrm{CVAE}}(\theta, \phi) = -\mathbb{E}_{q_\phi}[\log p_\theta(x \mid z, c)] + D_{\mathrm{KL}}(q_\phi(z \mid x, c) \,\|\, p(z \mid c)) \,. \tag{46}$$

- With Gaussian q_ϕ and $p(z \mid c) = \mathcal{N}(0, I)$, the KL term stays analytic and the reparameterization trick is unchanged:

$$z = \mu_\phi(x, c) + \sigma_\phi(x, c) \odot \epsilon, \qquad \epsilon \sim \mathcal{N}(0, I). \tag{47}$$

- Training needs paired data (x, c)

Injecting the Condition

- A discrete label becomes a one-hot vector or a learned embedding h_c; a continuous parameter can be fed directly
- Encoder: concatenate before the head that outputs the Gaussian parameters

$$h = \mathrm{concat}(\mathrm{Encoder}(x),\ h_c)\,, \qquad \mu_\phi, \log \sigma^2_\phi = \mathrm{MLP}(h). \tag{48}$$

- Decoder: concatenate with the latent code

$$\hat{x} = \mathrm{Decoder}(\mathrm{concat}(z, h_c))\,. \tag{49}$$

- In convolutional decoders c is more often injected at every layer, by broadcasting it as extra channels or by feature-wise modulation
- If the decoder can ignore c, the model degenerates back to a plain VAE

Generation, and an Example

- As in a VAE the encoder is discarded after training, but now the condition is chosen by the user:

$$z \sim p(z \mid c), \qquad \hat{x} = f_\theta(z, c). \tag{50}$$

- Fixing c and drawing several z gives diverse samples that all satisfy the same requirement
- **GalaxyMNIST:** let c be the morphology,

$$c \in \{\text{smooth round},\ \text{smooth cigar},\ \text{edge-on disk},\ \text{spiral}\}. \tag{51}$$

  For fixed c, the latent z then controls orientation, brightness profile, axis ratio and fine structure
- $\boxed{\text{CVAE} = \text{VAE} + \text{conditional control}}$

From Conditioning to Prompting

- In a CVAE the condition c is explicit: a class label, a physical parameter, or a measurement
- Modern generative AI uses a richer condition: a text prompt

$$p_\theta(x \mid c) \quad \longrightarrow \quad p_\theta(\text{output} \mid \text{prompt}). \tag{52}$$

- The role is the same: user-provided information controls what is generated
- The remaining variation comes from latent variables, sampling, or internal model randomness
- Lecture 2 studies this conditioning mechanism in language models

Questions?

Goals for Today

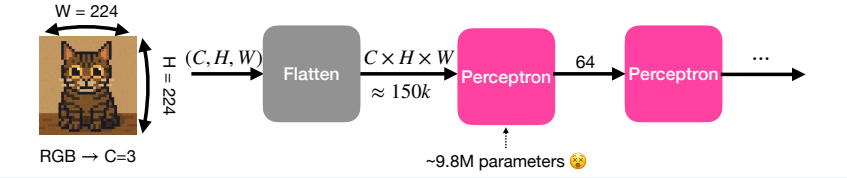
- Understand the limitations of fully connected neural networks in computer vision
- Learn the core principles behind convolutional neural networks, including local connectivity and parameter sharing
- Explore the building blocks of CNNs: convolution, activation, pooling, normalization, and dropout
- Introduce representative CNN architectures
- Familiarize with key computer vision tasks

Computer Vision

- Computer vision (CV) is a field of AI that enables machines to interpret and understand visual information from the world
- Traditional CV methods rely on hand-crafted features: Scale-invariant feature transform (SIFT), Histogram of oriented gradients (HOG), and etc.
- These features often failed under real-world variability

Figure 20: A pedestrian image (left) and its gradient (right). Source: [17]

DL for CV?



- **Loss of spatial structure**
  - 2D images are 3D arrays, but fully connected neural networks (FCNNs) require 1D input
  - Flattening destroys the spatial relationships between pixels
  - FCNNs treat every pixel independently, ignoring local correlations like edges
  - If a pattern (like an edge) shifts slightly in position, the FCNN sees it as a completely different input
- **Parameter explosion**
  - For high-resolution images, the number of parameters becomes massive
  - A single perceptron layer applied to a 3×224×224 image requires about 9.8 million parameters to produce a 64-dim hidden vector

Core Ideas

- **Convolutional Neural Networks (CNNs)** are a type of neural network designed to process data with a grid-like structure, such as 1D time-series or 2D images
- CNNs are particularly effective at capturing local patterns using **convolution** operations
- **Local Connectivity**: convolution extracts local features from small subregions of the image
- **Parameter sharing**: convolution uses shared parameters to detect local features throughout the entire image

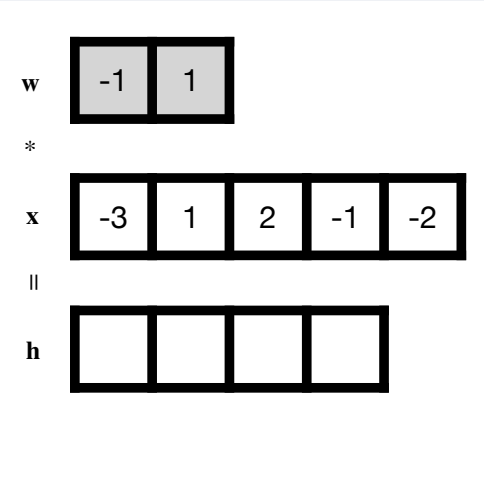
1D Discrete Convolution

1D discrete convolution[3]:

- **Input**, $\mathbf{x} \in \mathbb{R}^M$
  - e.g. $\mathbf{x} = [x_1, x_2, x_3, x_4, x_5]$
- **Kernel** or **filter**, $\mathbf{w} \in \mathbb{R}^N$
  - e.g. $\mathbf{w} = [w_1, w_2]$)
- **Output (feature map)**, $\mathbf{w} * \mathbf{x}$ or $(\mathbf{w} * \mathbf{x})_i = \sum_{n=1}^{N} w_n x_{i+n}$
  - e.g. $\mathbf{w} * \mathbf{x} = (w_0 x_0 + w_1 x_1, w_0 x_1 + w_1 x_2, w_0 x_2 + w_1 x_3, w_0 x_3 + w_1 x_4)$

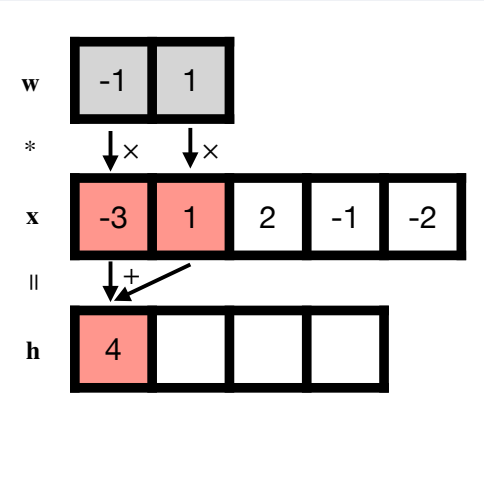
[3] Convolutional neural networks typically use *cross-correlation* instead of convolution, but since the kernels are learnable and the distinction is minor, we refer to cross-correlation as "convolution" throughout this lecture for simplicity.

Visualization of 1D Discrete Convolution



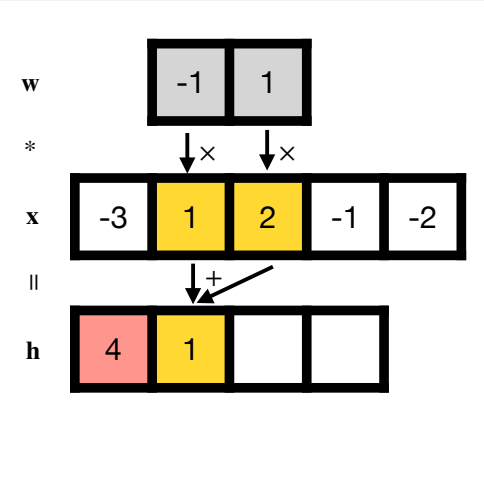
- Kernel: $\mathbf{w}$ / Input: $\mathbf{x}$ / Output feature map: $\mathbf{h}$

Visualization of 1D Discrete Convolution (cont.)



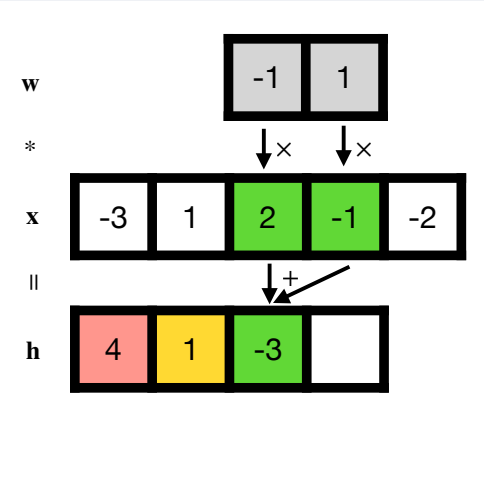
- Kernel: $\mathbf{w}$ / Input: $\mathbf{x}$ / Output feature map: $\mathbf{h}$

Visualization of 1D Discrete Convolution (cont.)



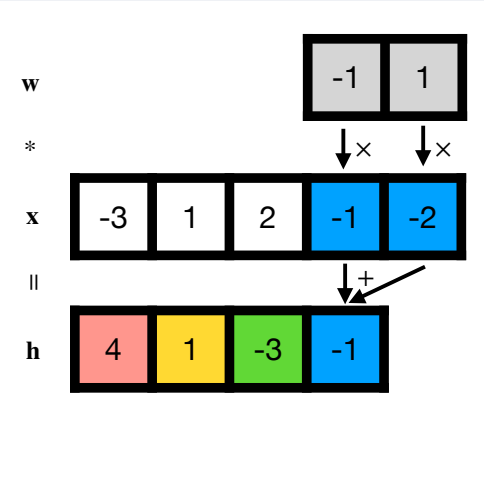
- Kernel: $\mathbf{w}$ / Input: $\mathbf{x}$ / Output feature map: $\mathbf{h}$

Visualization of 1D Discrete Convolution (cont.)



- Kernel: $\mathbf{w}$ / Input: $\mathbf{x}$ / Output feature map: $\mathbf{h}$

1D Discrete Convolution (cont.)



- Kernel: $\mathbf{w}$ / Input: $\mathbf{x}$ / Output feature map: $\mathbf{h}$

2D Discrete Convolution (cont.)

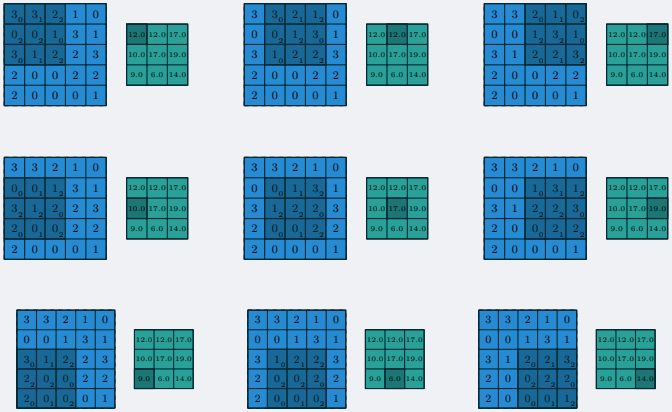


Figure 21: Source: [18]

- 2D discrete convolution: $(W * X)_{ij} = \sum_h \sum_w W_{h,w} X_{i+h,j+w}$

Multi-Channel Convolution

- Input images can have multiple channels, such as RGBA images with four channels: red, green, blue, and alpha (transparency)
- A convolution takes a multi-channel input (leftmost stack of purple-blue squares), and applies multiple kernels (small purple-blue squares)
- In each path, each kernel processes the corresponding input channel, and the results are summed element-wise to produce one channel of the output feature map
- Combining multiple such outputs results in a multi-channel output (rightmost stack of green squares)
- This enables the construction of deeper and more expressive models

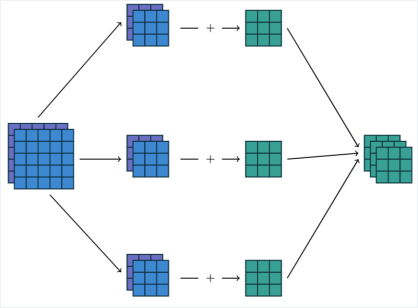
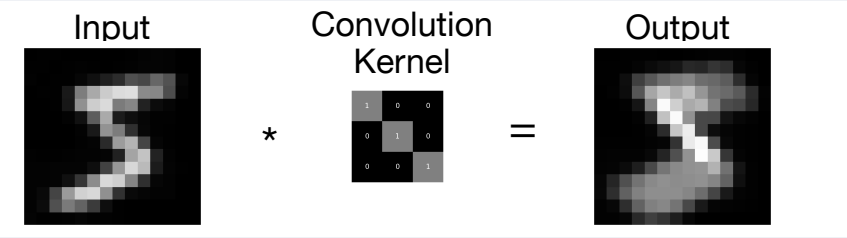


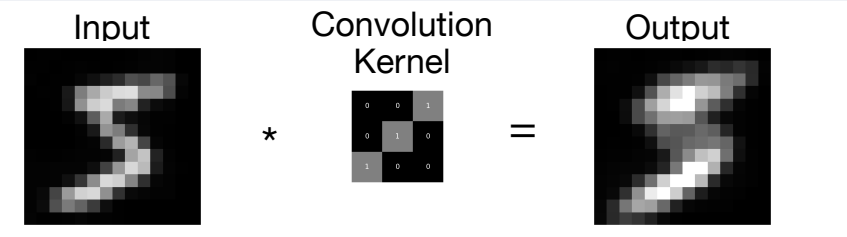
Figure 22: Source: [18]

Convolution as Edge Detection



- **Input** is a grayscale image of the digit 5, represented as a 2D array of pixel intensities, where brighter pixels indicate higher intensity, and darker pixels indicate lower intensity
- **Convolution kernel** is a small 3×3 matrix that slides over the input image to detect edges
- **Output feature map** is the result of applying the kernel to the input image, highlighting edges where intensity changes align with the kernel's pattern

Convolution as Edge Detection (cont.)



- **Input** is a grayscale image of the digit 5, represented as a 2D array of pixel intensities, where brighter pixels indicate higher intensity, and darker pixels indicate lower intensity
- **Convolution kernel** is a small 3×3 matrix that slides over the input image to detect edges
- **Output feature map** is the result of applying the kernel to the input image, highlighting edges where intensity changes align with the kernel's pattern

Convolution as Edge Detection (cont.)

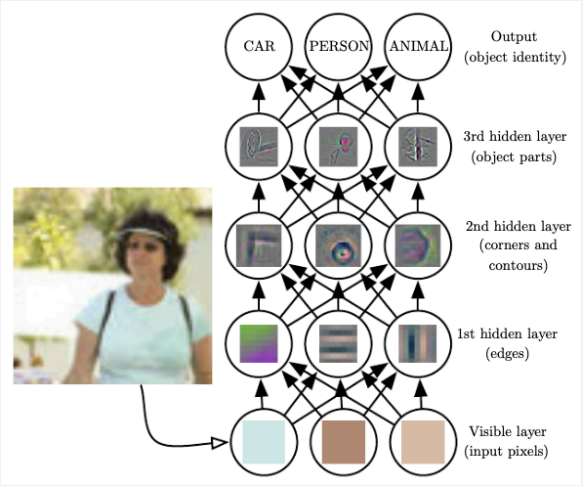
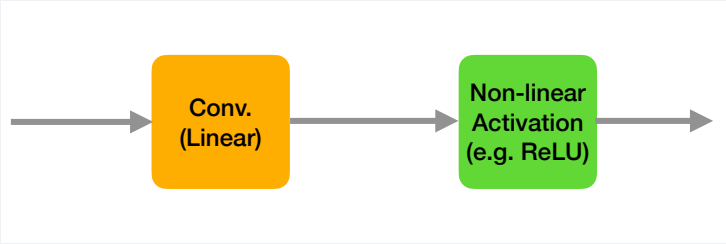


Figure 23: Source: [19, 7]

Convolution Arithmetic

To gain a deeper understanding of convolution, refer to the following link:
https://github.com/vdumoulin/conv_arithmetic

Linearity of Convolution

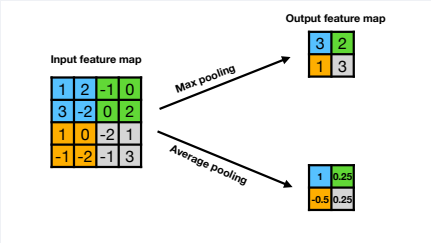


- Convolution is a linear operation

$$\mathbf{w} * (a\mathbf{u} + b\mathbf{v}) = a\mathbf{w} * \mathbf{u} + b\mathbf{w} * \mathbf{v} \tag{53}$$

- In convolutional neural networks (CNNs), a convolution layer is typically followed by a non-linear activation function, just like in fully connected neural networks

Pooling



- **Pooling** is a downsampling operation that reduces the spatial dimensions of feature maps, making the model more translation invariant and reduces computational cost
- A pooling window (e.g., 2×2) slides across the input feature map
- For each window, it applies an aggregation function, such as max and average
- PyTorch provides a bunch of pooling methods (link)

Dropout

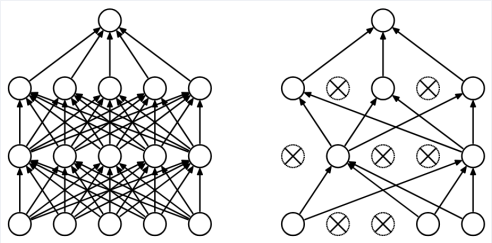


Figure 24: Left: Standard Neural Net. Right: After applying dropout. Source: [20]

- Dropout is a regularization technique where, during training, a random subset of neurons is ”dropped” (set to zero) in each layer, preventing overfitting by encouraging the network to learn redundant representations
- Undropped nodes are scaled up by $\frac{1}{1-p_{\text{dropout}}}$ during training to ensure the network’s output remains consistent in scale
- See torch.nn.Dropout

Normalization

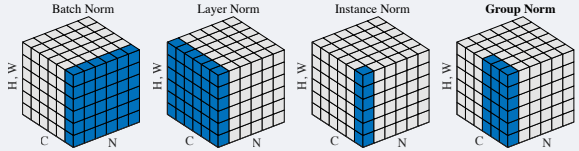


Figure 25: Source: [21]

- **Normalization** is a technique used to improve the training of DL models by adjusting the inputs of each layer to have a mean of zero and a standard deviation of one, thereby stabilizing the learning process
- During training, a normalization layer uses the mean and standard deviation of each batch for backpropagation
- In the inference phase, a normalization layer uses a running average of the means and standard deviation from the training phase, maintaining stable outputs
- Batch normalization is widely used in DL models for image data
- See Normalization Layers in PyTorch

CNN Demo

ConvNetJS CIFAR-10 demo

Image Generation: PatchMatch

- PatchMatch is a non-learning-based algorithm that quickly finds similar patches and combines them for image restoration and synthesis [22]
- It requires suitable patches and may produce repeated or unnatural artifacts

Text Generation: Rule-based Chatbots

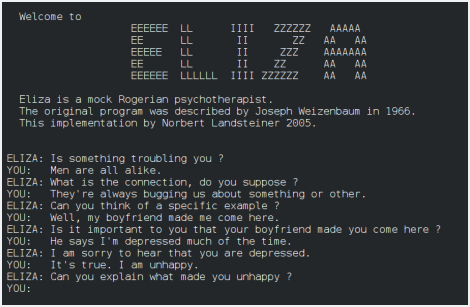


Figure 26: A conversation with the ELIZA chatbot

- ELIZA is an early chatbot developed by Joseph Weizenbaum in the 1960s
- It generates responses using keyword detection and predefined pattern-matching rules

Simulation

- There is one more way out: a simulator produces samples x without ever writing down $p(x)$
- Examples: physics event generators, detector simulation, game engines
- Building one takes years of domain knowledge, and running one is slow – a single LHC event takes minutes of CPU time
- It also gives no tractable density, so it cannot score an observation, cannot be conditioned on one, and cannot be inverted
- Hence the plan for the rest of the lecture: learn p_θ from data, and let a neural network do the sampling

Task

- **Classification**: $f : \mathbf{x} \to y$, where $\mathbf{x} \in \mathbb{R}^d$ and $y \in \{1, 2, \ldots, K\}$
- **Regression**: $f : \mathbf{x} \to y$, where $\mathbf{x} \in \mathbb{R}^d$ and $y \in \mathbb{R}^{d'}$
- **Semantic segmentation**: classification of every pixel
- **Object detection**: bounding box regression, plus classification of each box
- In all of them the target y is given with the input, and one input has one correct answer
- This is supervised learning: the data is a set of pairs (x, y)

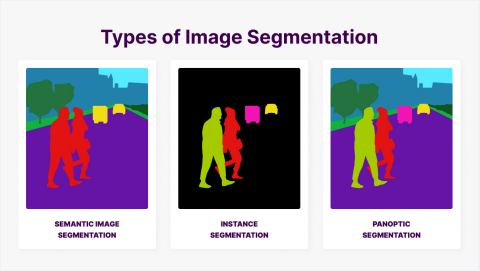


Figure 27: Pixel-level tasks: the label is an image of the same size as the input

Backpropagation (cont.)

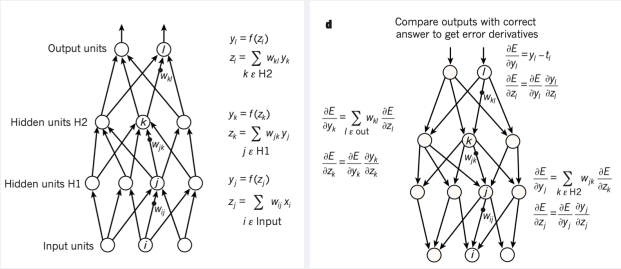


Figure 28: Forward pass, the prediction (left), and backward pass, the gradient (right)

- **Forward**: the input flows through the layers, each intermediate value is kept, and the loss is evaluated
- **Backward**: starting from the loss, the gradient flows back through the same graph, reusing the stored values
- One training step is: forward, backward, update. Repeat over mini-batches until the loss stops improving

Underfitting and Overfitting

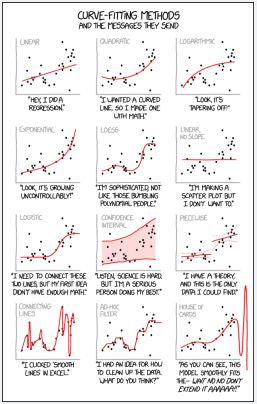


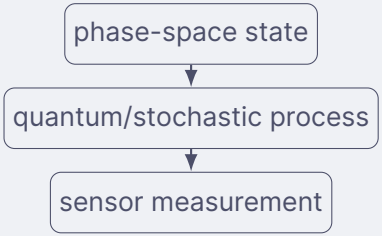
Figure 29: Source: [23]

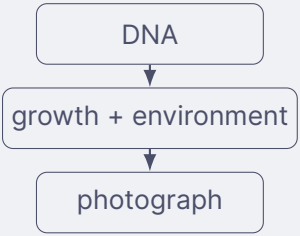
- A low training loss is not the goal; the goal is a low error on data the model has never seen
- Model capacity is how flexible the model is; data complexity is how structured the true function is
- capacity $<$ complexity $\rightarrow$ underfitting: the model is too simple to capture the structure
- capacity $>$ complexity $\rightarrow$ overfitting: the model fits the noise, and does not generalise
- An overfitted model has effectively memorised the training set
- This matters twice over for generative models: a model that memorises the data can reproduce it perfectly and still generate nothing new
- Overfitting is invisible if the model is evaluated on its own training data, so the dataset is split into a **training** set (fits θ), a **validation** set (chooses hyperparameters and the stopping point) and a **test** set (measured once, at the end)

Training, Validation and Test Sets

- Overfitting is invisible if the model is evaluated on the data it was trained on, so the dataset is split into three non-overlapping parts
  - **training set**: the parameters θ are fitted on it
  - **validation set**: the hyperparameters and the stopping point are chosen on it
  - **test set**: the final performance is measured on it, once
- The validation and test sets are never used for gradient updates
- Nothing may be changed in response to test set performance; otherwise the test set becomes a validation set and the estimate is optimistic
- There is no universal ratio; $64 : 16 : 20$ is a reasonable default

Two Analogies





- The true state is a few degrees of freedom: position, momentum, energy, spin, or decay configuration
- Quantum mechanics gives probabilistic outcomes; detectors add resolution and noise
- The output is a high-dimensional vector of channels, hits, or pixels, constrained by physical law

- DNA provides the generative information
- Growth and environment add stochastic variation
- Photography adds pose, viewpoint, lighting, and sensor noise

- Both: few causes, many measured numbers, and realistic data fills only a small region of the observation space

From Manifolds to Latent Variable Models

- A latent variable model formalizes the idea:

$$z \sim p(z), \qquad x \sim p_\theta(x \mid z). \tag{54}$$

- z: low-dimensional latent coordinate;
- $p(z)$: simple prior distribution;
- $p_\theta(x \mid z)$: decoder likelihood, or probabilistic observation model.
- The decoder maps latent coordinates into the data manifold or its neighborhood.

Denoising Autoencoder

Corrupt the input, then ask for the clean input back[24]:

$$\tilde{x}^{(i)} \sim \mathcal{M}_{\mathcal{D}}\left(\tilde{x} \mid x^{(i)}\right), \qquad L_{\mathrm{DAE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} \left(x^{(i)} - f_{\theta}\left(g_{\phi}(\tilde{x}^{(i)})\right)\right)^2.$$

- The corruption $\mathcal{M}_{\mathcal{D}}$ masks a random fraction of the components or adds Gaussian noise.
- Copying the input is no longer a solution, so the network must use the dependence between components.
- The model learns to map a noisy point back onto the data surface, which is a first hint of the generative models that follow.

How Tight Is the Bound?

- The same quantity can be written as an exact identity:

$$\log p_\theta(x) = \mathcal{L}_{\mathrm{ELBO}}(x) + D_{\mathrm{KL}}(q_\phi(z \mid x) \,\|\, p_\theta(z \mid x)) \,. \tag{55}$$

- The gap is exactly the error of the approximate posterior
- It is non-negative, which proves the bound
- Raising the bound therefore does two things at once: it fits the data through θ and improves the posterior approximation through ϕ.

Training Loop

- For each minibatch:
  1. Encode: compute $\mu_\phi(x)$ and $\sigma_\phi(x)$
  2. Sample $\epsilon \sim \mathcal{N}(0, I)$ and set $z = \mu_\phi(x) + \sigma_\phi(x) \odot \epsilon$
  3. Decode z and evaluate the reconstruction term; one sample per data point is enough in practice
  4. Add the analytic KL term
  5. Backpropagate through θ and ϕ, then update the parameters
- The only difference from an ordinary autoencoder is the noise injection and the extra KL term.

Summary: Model

- $p(x) = ?$
- $p(z) = \mathcal{N}(0, I)$
- Probabilistic encoder: $q_\phi(z|x) = \mathcal{N}(z; \mu_\phi(x), \operatorname{diag} \sigma^2_\phi(x))$
- Probabilistic decoder: $p_\theta(x|z) = \mathcal{N}(x; \mu_\theta(z), \sigma^2 I)$
- Encoding network: $g_\phi : x \mapsto (\mu_\phi(x), \log \sigma^2_\phi(x))$
- Decoding network: $f_\theta : z \mapsto \mu_\theta(z)$

Summary: Key Points

- An autoencoder compresses data through a bottleneck and reconstructs it, without labels.
- A denoising autoencoder corrupts the input first, so the network must learn the structure of the data instead of copying it.
- Neither defines a distribution over the latent code, so neither can generate.
- A VAE adds a prior $p(z)$ and an approximate posterior $q_\phi(z \mid x)$, and maximizes the evidence lower bound.
- The bound splits into reconstruction and KL regularization; the reparameterization trick makes it trainable by backpropagation.
- Generation is then one draw from the prior and one decoder pass.

References i

[1] Rafael Gómez-Bombarelli, David Duvenaud, José Miguel Hernández-Lobato, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik.
Automatic chemical design using a data-driven continuous representation of molecules.
CoRR, abs/1610.02415, 2016.

[2] Erik Buhmann, Sascha Diefenbacher, Engin Eren, Frank Gaede, Gregor Kasieczka, Anatolii Korol, William Korcari, Katja Krüger, and Peter McKeown.
CaloClouds: fast geometry-independent highly-granular calorimeter simulation.
JINST, 18(11):P11025, 2023.

[3] Arsenii Gavrikov et al.
Simulation-based inference for precision neutrino physics through neural Monte Carlo tuning.
Commun. Phys., 9(1):63, 2026.

[4] M. S. Albergo, G. Kanwar, and P. E. Shanahan.
Flow-based generative models for markov chain monte carlo in lattice field theory.
Physical Review D, 100(3):034515, 2019.

[5] Michela Paganini, Luke de Oliveira, and Benjamin Nachman.
CaloGAN: Simulating 3D high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks.
Physical Review D, 97(1):014021, 2018.

References iii

[6] Frank Rosenblatt.
The perceptron: A probabilistic model for information storage and organization in the brain.
Psychological Review, 65(6):386–408, 1958.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
***Deep Learning*.**
MIT Press, 2016.
http://www.deeplearningbook.org.

[8] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton.
Deep learning.
Nature, 521(7553):436–444, 2015.

References iv

[9] Min hyuk Kim.
Manifold hypothesis.
https://velog.io/@kim_min_hyuk/Manifold-hypothesis, 2023.

[10] Barnabás Póczos.
Advanced introduction to machine learning.
https://www.cs.cmu.edu/~epxing/Class/10715/lectures/ManifoldLearning.pdf, 2014.

[11] Lilian Weng.
From autoencoder to beta-vae.
https://lilianweng.github.io/posts/2018-08-12-vae/, 2018.
Lil'Log.

[12] David E Rumelhart, Geoffrey E Hinton, James L McClelland, et al.
A general framework for parallel distributed processing.
Parallel distributed processing: Explorations in the microstructure of cognition, 1(45-76):26, 1986.

[13] Geoffrey E Hinton and Ruslan R Salakhutdinov.
Reducing the dimensionality of data with neural networks.
science, 313(5786):504–507, 2006.

[14] Diederik P. Kingma and Max Welling.
Auto-encoding variational bayes.
In Yoshua Bengio and Yann LeCun, editors, *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.

[15] Eric Jang.
A beginner's guide to variational methods: Mean-field approximation.
https://lilianweng.github.io/posts/2018-08-12-vae/, 2016.

[16] Kihyuk Sohn, Honglak Lee, and Xinchen Yan.
Learning structured output representation using deep conditional generative models.
In *Advances in Neural Information Processing Systems 28, NIPS 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3483–3491, 2015.

[17] Wikipedia.
Histogram of oriented gradients — Wikipedia, the free encyclopedia.
http://en.wikipedia.org/w/index.php?title=Histogram%20of%20oriented%20gradients&oldid=1280028145, 2025.
[Online; accessed 27-May-2025].

[18] Vincent Dumoulin and Francesco Visin.
A guide to convolution arithmetic for deep learning, 2018.

[19] Matthew D. Zeiler and Rob Fergus.
Visualizing and understanding convolutional networks.
CoRR, abs/1311.2901, 2013.

[20] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov.
Dropout: a simple way to prevent neural networks from overfitting.
The journal of machine learning research, 15(1):1929–1958, 2014.

[21] Yuxin Wu and Kaiming He.
Group normalization.
CoRR, abs/1803.08494, 2018.

[22] Connelly Barnes, Eli Shechtman, Adam Finkelstein, and Dan B Goldman.
Patchmatch: A randomized correspondence algorithm for structural image editing.
ACM Trans. Graph., 28(3):24, 2009.

[23] Randall Munroe.
Curve fitting.
xkcd 2048, https://xkcd.com/2048/, 2018.
Accessed 2026-08-03.

[24] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol.
Extracting and composing robust features with denoising autoencoders.
In *Proceedings of the 25th International Conference on Machine Learning*, pages 1096–1103, 2008.