An Introduction to Data Sciences

Gabriel Peyré
CNRS & DMA, École Normale Supérieure, PSL

CHAPTER 5

Neural networks

September 9, 2026

Neural networks learn transformations from examples. This chapter explains how layers are composed and trained, then shows how a generator can learn to reproduce a data distribution. The connection with optimal transport clarifies both the learning objective and its limitations.

5.1 Discriminative networks

5.1.1 A function built from layers

A neural network is a parameterized function f_w. Given an input x, such as an image, it computes an output $\hat{y} = f_w(x)$. For classification, the output can represent estimated probabilities of several classes. A large output probability expresses the model's confidence; it does not by itself ensure that this confidence is well calibrated or that the prediction is correct.

To begin, suppose an image belongs to exactly one of two classes, cat or dog. A classifier can return two nonnegative numbers summing to one. This differs from a multilabel task, where an image may contain both a cat and a dog: in that case, separate probabilities need not sum to one.

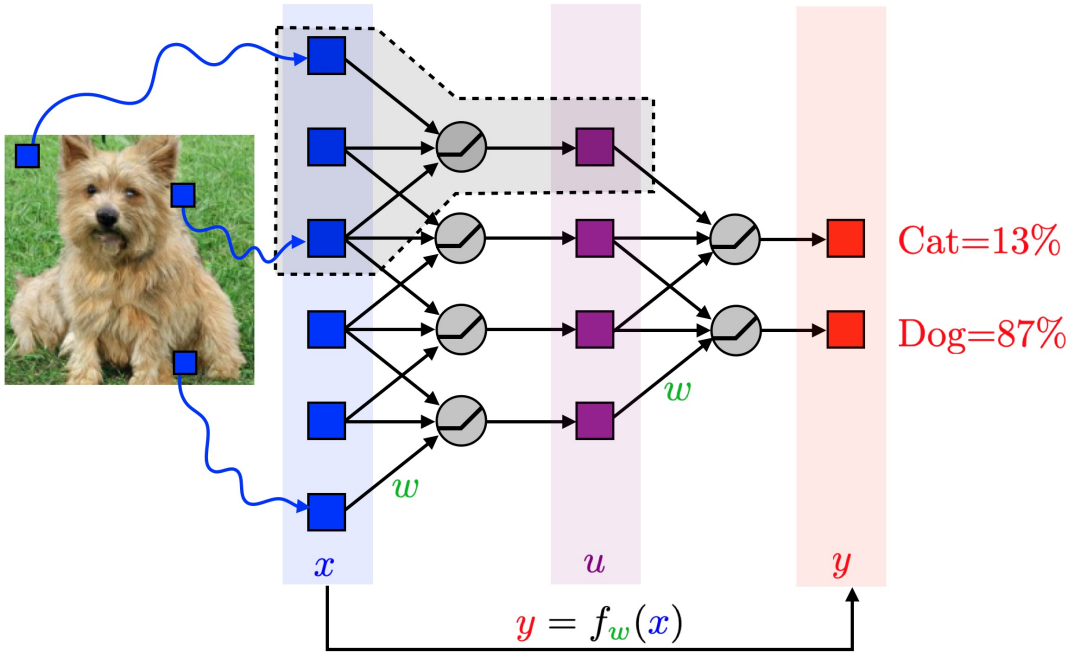

Figure 5.1. A schematic classifier with one hidden layer and one output layer. Only a few input pixels and connections are drawn. The displayed percentages are model predictions; a normalization such as softmax is needed to obtain class probabilities.

Each layer combines affine operations with an activation function. A rectified linear unit, or ReLU, computes

$$u_1 = \mathrm{ReLU}(w_1 x_1 + w_2 x_2 + w_3 x_3 + b), \qquad \mathrm{ReLU}(t) = \max(t, 0). \tag{5.1}$$

The weights determine the contribution of each input; the bias b shifts the activation threshold. A ReLU outputs zero for a nonpositive input and passes positive inputs unchanged. It differs from a binary threshold neuron, which has only two possible output values.

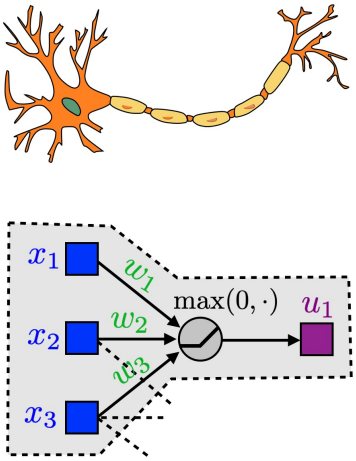

Figure 5.2. A biological neuron and a schematic artificial neuron. The mathematical model keeps only a weighted sum and an activation; the bias in (5.1) is omitted from the drawing.

For an input $x \in \mathbb{R}^d$, a hidden layer of width m, and K classes, one possible network is

$$u = \mathrm{ReLU}(Wx + b), \qquad a = Vu + c, \qquad \hat{y}_k = \frac{e^{a_k}}{\sum_{j=1}^{K} e^{a_j}},$$

where $W \in \mathbb{R}^{m \times d}$, $b \in \mathbb{R}^m$, $V \in \mathbb{R}^{K \times m}$, and $c \in \mathbb{R}^K$. ReLU acts coordinatewise. The final operation is softmax; the intermediate values a_k are called logits. In computation, replacing every a_k by $a_k - \max_j a_j$ gives the same probabilities and avoids overflow. All entries of W, b, V, c form the parameter vector w.

Composing more layers gives a deep network. Nonlinear activations are essential: a composition of affine maps is still affine. Depth can represent certain functions more economically than a very wide single hidden layer, but the appropriate architecture depends on the task.

The biological terminology is an analogy, not a faithful model of the brain. McCulloch and Pitts studied logical threshold neurons [12], while Rosenblatt's perceptron introduced an early learning model [14]. These models should be distinguished from the ReLU networks described here. Biological studies of the visual cortex and of membrane dynamics concern much richer mechanisms [6, 7].

5.1.2 Learning from labeled examples

Supervised learning starts from pairs $(x_i, y_i)_{i=1}^n$. For a single class label k_i, the target is often a one-hot vector: $(y_i)_{k_i} = 1$, with all other entries zero. Such labels record annotations, rather than known conditional probabilities. The data may come from human annotation or another source of supervision, and may contain mistakes.

A training objective measures disagreement between predictions and labels. Two examples are the mean squared error and the cross-entropy loss:

$$E_{\mathrm{sq}}(w) = \frac{1}{n} \sum_{i=1}^{n} \| f_w(x_i) - y_i \|_2^2, \tag{5.2}$$

$$E_{\mathrm{CE}}(w) = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} (y_i)_k \log(f_w(x_i)_k). \tag{5.3}$$

For a one-hot target, the corresponding cross-entropy term is $-\log(f_w(x_i)_{k_i})$. The logarithm is natural, and softmax gives strictly positive probabilities for finite logits. A stable implementation evaluates the log-softmax directly. A penalty on the weights can be added to the objective.

Figure 5.3. Examples of labeled dog and cat images from ImageNet [4]. Training uses many examples; evaluation requires separate data.

Training seeks to reduce this empirical loss. Because the network depends nonlinearly on many parameters, the objective is generally nonconvex. Optimization methods can work well without providing a general guarantee of reaching its global minimum.

5.1.3 Backpropagation and weight updates

Backpropagation computes the gradient of a scalar loss by applying the chain rule through the network in reverse order [11, 15]. It is reverse-mode automatic differentiation. An optimizer then uses these derivatives to update the weights; for example, gradient descent takes the step

$$w^{(t+1)} = w^{(t)} - \eta_t \nabla E(w^{(t)}), \qquad \eta_t > 0.$$

The step size matters: an excessively large step can increase the loss. Stochastic gradient descent replaces the full gradient by a gradient estimated from a randomly chosen minibatch, reducing the cost of each update. An individual stochastic step need not decrease the full training loss.

For the two-layer classifier above and one example with cross-entropy ℓ, the chain rule is particularly transparent. Write $r = Wx + b$, $u = \mathrm{ReLU}(r)$, and $\delta = \hat{y} - y$. Away from $r_j = 0$,

$$\frac{\partial \ell}{\partial V} = \delta u^{\mathsf{T}}, \qquad \frac{\partial \ell}{\partial c} = \delta, \qquad q = (V^{\mathsf{T}}\delta) \odot \mathbf{1}_{r>0}, \qquad \frac{\partial \ell}{\partial W} = q x^{\mathsf{T}}, \qquad \frac{\partial \ell}{\partial b} = q.$$

Here $\odot$ is coordinatewise multiplication. At a ReLU kink, software chooses a convention, commonly zero, for the derivative of the activation. That convention does not make the network classically differentiable at the kink.

Large labeled datasets, efficient GPU computation, and improvements in architectures and training contributed to the success of deep image classifiers, notably the 2012 ImageNet result [8]. Regularization techniques such as dropout can help limit overfitting [16]. The work of Yoshua Bengio, Geoffrey Hinton, and Yann LeCun was recognized by the 2018 ACM A. M. Turing Award, announced in 2019.[1]

5.1.4 Approximation, optimization, and generalization

A universal approximation theorem concerns what a network can represent. A classical result of Cybenko [3] states that, for a continuous function $F : [0,1]^d \to \mathbb{R}$ and any $\varepsilon > 0$, there exist a finite m and coefficients such that

$$G(x) = \sum_{j=1}^{m} \alpha_j \sigma(v_j \cdot x + \beta_j), \qquad \sup_{x \in [0,1]^d} |F(x) - G(x)| < \varepsilon,$$

where σ is a fixed continuous sigmoidal activation, such as $\sigma(t) = 1/(1 + e^{-t})$. The output layer is linear, and the width and weights are unrestricted. This is the sigmoidal theorem; approximation results for ReLU networks require their own statements.

The theorem does not involve training data. It neither requires infinitely many examples nor guarantees that a training algorithm can find the approximating network from finitely many observations. Three questions

[1] https://awards.acm.org/about/2018-turing

must be separated: whether a good representation exists, whether optimization can find it, and whether the learned predictor performs well on new data.

Generalization is evaluated on data excluded from training. Validation data guide choices such as architecture and stopping time; a separate test set estimates the performance of the final procedure. Good performance also depends on whether these data represent the conditions in which the network will be used.

Convolutional networks build in assumptions about spatial structure [9]. Local connections restrict the receptive field, and weight sharing applies the same filter at different positions. A stride-one convolution commutes with translations on an infinite or periodic grid; boundary handling and subsampling can change this property. Translation equivariance of a feature map should also be distinguished from translation invariance of a final classification.

Architecture, regularization, data, and optimization all influence performance [10]. Mathematical results explain particular mechanisms and regimes, while a complete account of every successful architecture remains beyond a single theorem.

5.2 Generative networks

5.2.1 From random inputs to a distribution

A generator g_w transforms a random input $Y \in \mathbb{R}^r$ into an image $X = g_w(Y) \in \mathbb{R}^d$. The input law η, for instance a standard Gaussian, is chosen in advance. The output law is the pushforward $\mu_w = (g_w)_{\#}\eta$. This notation describes how probability mass is redistributed by the network. A generator is a learned transformation, not generally the inverse of a classifier.

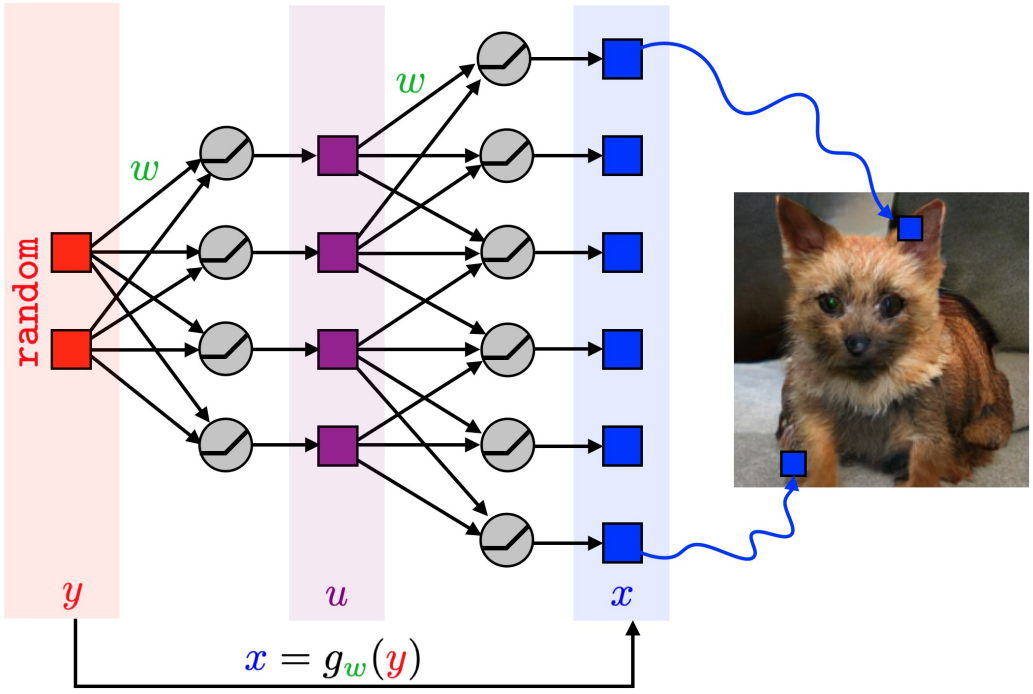

Figure 5.4. A schematic generator maps a random input y through a hidden layer to an image $x = g_w(y)$. Practical image generators have more layers and a much larger output dimension.

For unconditional generation, training can use images without class labels. Their collection, selection, and preprocessing still require care. Conditional generation additionally supplies information such as a class label, giving a model $g_w(y, c)$. Thus not every generative model is trained without supervision.

The aim is to reproduce a distribution, including its diversity. If the reference distribution assigns probability $1/2$ to cats and $1/2$ to dogs, an ideal generator has those probabilities too. A finite random batch need not contain exactly half of each. A model that produces the same plausible image repeatedly has not learned the full distribution; this is an extreme form of mode collapse.

5.2.2 Comparing empirical distributions

Let $z_1, \dots, z_n$ be training images, and let $x_i = g_w(y_i)$ be generated images from independent latent samples. Their empirical distributions are

$$\mu_{w,n} = \frac{1}{n} \sum_{i=1}^{n} \delta_{x_i}, \qquad \nu_n = \frac{1}{n} \sum_{j=1}^{n} \delta_{z_j}.$$

Using measures retains frequencies and repeated samples, which ordinary set notation would discard. One possible objective is a transport distance between these measures.

For Euclidean ground cost, the empirical Wasserstein distance of order one is

$$W_1(\mu_{w,n}, \nu_n) = \frac{1}{n} \min_{s \in \Sigma_n} \sum_{i=1}^{n} \| g_w(y_i) - z_{s(i)} \|_2. \tag{5.4}$$

The factor $1/n$ normalizes the total probability mass. This distance is different from the squared quadratic cost introduced in the transport chapter. For unequal sample counts or weights, replace the permutation by a coupling with the prescribed marginals.

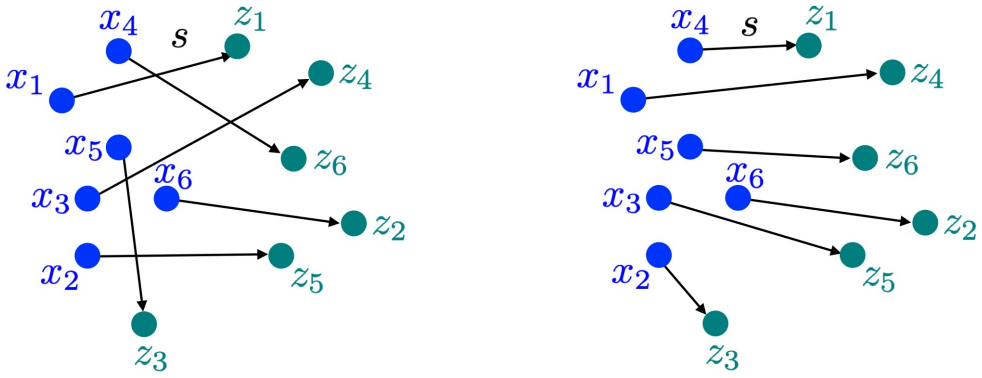

Figure 5.5. Two assignments of six source points to six target points. The right-hand assignment has smaller Euclidean transport cost and is the optimal matching in this example.

In Figure 5.5, the left assignment is $(1, 5, 4, 6, 3, 2)$, while the optimal one on the right is $(4, 3, 5, 1, 6, 2)$. The notation lists $(s(1), \dots, s(6))$. Although there are $6! = 720$ possibilities, efficient assignment algorithms avoid enumerating all $n!$ permutations and can solve a dense problem in $O(n^3)$ operations. The transport chapter explains the convex relaxation behind this computation [13].

The distributional learning objective is $\min_w W_1(\mu_w, \nu)$ for a target law ν. Finite-sample transport provides a numerical approximation, but brings statistical and optimization difficulties. Transport between small batches is generally a biased estimate of the population distance. The optimal assignment may change as w changes, so the empirical objective is not everywhere differentiable.

The ground distance also matters. Euclidean distance between pixel arrays is sensitive to small translations and may poorly reflect semantic similarity. A feature representation can provide a more relevant cost, but its choice changes the geometry being learned.

5.2.3 Generative adversarial networks

Generative adversarial networks (GANs), introduced by Goodfellow and collaborators in 2014 [5], train a generator against a discriminator. The discriminator $D_v(x) \in (0, 1)$ tries to distinguish data from generated samples. The original minimax objective is

$$\min_w \max_v \left\{ \mathbb{E}_{Z \sim \nu}[\log D_v(Z)] + \mathbb{E}_{Y \sim \eta}[\log(1 - D_v(g_w(Y)))] \right\}. \tag{5.5}$$

The discriminator increases the objective; the generator decreases it. Alternating gradient updates approximate this game, without a general guarantee of convergence for arbitrary networks.

For an unrestricted discriminator optimized exactly, the inner supremum equals $-\log 4 + 2\,\mathrm{JS}(\nu, \mu_w)$, where the Jensen–Shannon divergence is

$$\mathrm{JS}(\nu, \mu) = \tfrac{1}{2}\,\mathrm{KL}(\nu \mid m) + \tfrac{1}{2}\,\mathrm{KL}(\mu \mid m), \qquad m = \tfrac{1}{2}(\nu + \mu).$$

Here KL uses natural logarithms and is defined between measures by their Radon–Nikodym densities. Both measures are absolutely continuous with respect to m. This identity concerns the ideal discriminator optimum; finite networks and incomplete optimization change the objective actually achieved. In practice, a commonly used generator loss is $-\mathbb{E}\log D_v(g_w(Y))$, which has the same desired distributional optimum but different gradients.

5.2.4 The Wasserstein connection

Wasserstein GANs [1] use a different objective, based on the Kantorovich–Rubinstein dual formula. For probability measures on $\mathbb{R}^d$ with finite first moments,

$$W_1(\nu, \mu_w) = \sup_{\mathrm{Lip}(h)\le 1} \left\{\mathbb{E}_{Z\sim\nu} h(Z) - \mathbb{E}_{Y\sim\eta} h(g_w(Y))\right\}. \tag{5.6}$$

The constraint means $|h(x) - h(x')| \le \|x - x'\|_2$ for all x, x'. A critic network approximates this class of functions, and its values are real scores, not class probabilities. The generator minimizes the discrepancy estimated by the critic.

The critic is a dual potential for a *fixed* ground distance; it does not automatically learn a new transport cost between pairs of images. Enforcing or approximating the Lipschitz constraint is essential. Restricting the critic class and optimizing it only approximately also means that the computed score need not equal the true Wasserstein distance.

Ordinary GANs and Wasserstein GANs thus express related adversarial principles with different mathematical objectives. Neither training procedure guarantees that every generated image is meaningful or that all modes are represented.

Figure 5.6. Two sequences of synthetic images interpolating between cats and dogs, using the BigGAN model [2]. BigGAN is class-conditional; its class information is part of the conditioning. These examples illustrate generation and interpolation rather than a claim of an optimal transport path.

Interpolating latent or conditioning variables can produce smooth visual transitions, as in Figure 5.6. Such paths need not be transport geodesics or remain typical under the training distribution. Visual plausibility, diversity, fidelity to the target distribution, and performance on held-out data are separate aspects of a generative model's quality.

Acknowledgements. I thank Gwenn Guichaoua for her careful reading, and Sébastien Racanière and Vincent Barra for their corrections.

References

[1] Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein generative adversarial networks. In *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 214–223, 2017. URL https://proceedings.mlr.press/v70/arjovsky17a.html.

[2] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019. URL https://arxiv.org/abs/1809.11096.

[3] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, 1989.

[4] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[6] Alan L Hodgkin and Andrew F Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *The Journal of physiology*, 117(4):500–544, 1952.

[7] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

[11] Seppo Linnainmaa. Taylor expansion of the accumulated rounding error. *BIT Numerical Mathematics*, 16 (2):146–160, 1976.

[12] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133, 1943.

[13] Gabriel Peyré and Marco Cuturi. Computational optimal transport. *Foundations and Trends in Machine Learning*, 11(5–6):355–607, 2019. URL https://arxiv.org/abs/1803.00567.

[14] Frank Rosenblatt. The perceptron: A perceiving and recognizing automaton. Technical Report 85-460-1, Cornell Aeronautical Laboratory, 1957.

[15] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1): 1929–1958, 2014.