

CS 182 / EECS 282A Official Course Notes

Fall 2026

Note 1: Function Approximation

Based on lectures by Prof. Gireeja Ranade and Prof. Anant Sahai

Prerequisites

Comfort with vectors and matrices, affine functions, elementary calculus, and the idea of a probability distribution. No previous neural-network notation is assumed.

Reading and coverage map

Prince correspondence (direct: Chapters 1–5 and 8–9). Prince develops supervised learning, loss minimization, feed-forward networks, and the role of nonlinear activations. This lecture touches on various aspects of these chapters.

JRS prerequisite review (Chapters 1, 12, and 16–17). JRS reviews supervised learning, empirical risk, generalization, and neural-network basics. It is useful prerequisite review rather than a substitute for the construction below.

1.1 Learning a function from samples

Begin with the smallest nontrivial example, a scalar target function

$$f : \mathbb{R} \rightarrow \mathbb{R}, \quad x \mapsto y = f(x). \quad (1.1)$$

In supervised learning, the function is not available in closed form. Instead, the training set contains sample pairs

$$\mathcal{D}_{\text{train}} = \{(x_1, y_1), \dots, (x_n, y_n)\}, \quad y_i \approx f(x_i). \quad (1.2)$$

We seek a parameterized function N_θ such that

$$N_\theta : \mathbb{R} \rightarrow \mathbb{R}, \quad x \mapsto \hat{y} = N_\theta(x), \quad (1.3)$$

fits the observed pairs and behaves sensibly between and beyond them.

Remark

Two very different piecewise-linear curves can interpolate a set of samples. The samples alone do not select between a jagged and a smooth interpolant; preferring smoothness is an inductive bias, and we have to check if this is appropriate on unseen data.

In introductory calculus, one approximates a function by a piecewise-constant step function. (See Figure 1.1.) Refining the intervals improves approximation.

However, piecewise-constant functions are hard to learn using gradient-based methods. Why? Hard unit-step hidden activations have derivative zero at every point except the jump itself. Ordinary

backpropagation therefore cannot deliver useful gradients through those nonlinearities to train their locations or earlier layers. A final linear layer over fixed steps can still learn their heights, if the locations are fixed, but cannot simultaneously learn the locations.

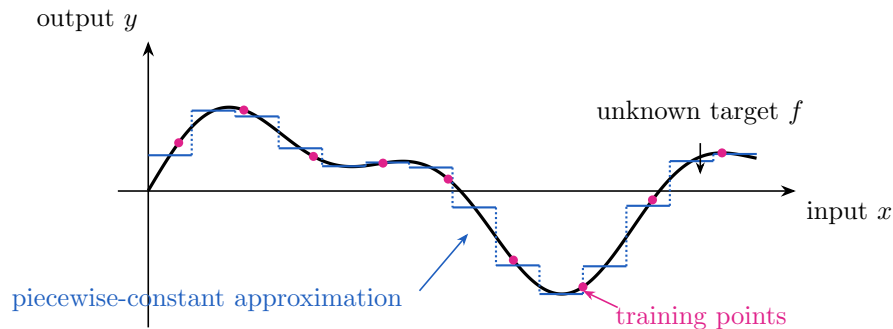


Figure 1.1: The starting point: samples from an unknown scalar function and a calculus-style piecewise-constant approximation.

The natural upgrade is a continuous piecewise-linear function. On a compact (closed, bounded) interval, every continuous function can be approximated arbitrarily well by such functions. A continuous piecewise-linear function is determined by an intercept, its kink or “elbow” locations, an initial slope, and the change in slope at every elbow.

1.2 ReLU ramps are a spline basis

The rectified linear unit is

$$\text{ReLU}(z) = \max(0, z) = \begin{cases} z, & z \geq 0, \\ 0, & z < 0. \end{cases} \quad (1.4)$$

Composing an affine function with ReLU creates one ramp (see Figure 1.2):

$$g_i(x) = \text{ReLU}(w_i^{(1)}x + b_i^{(1)}). \quad (1.5)$$

When $w_i^{(1)} \neq 0$, its elbow is at $-b_i^{(1)}/w_i^{(1)}$. The sign and magnitude of $w_i^{(1)}$ determine the active side and inner slope; an outer coefficient controls how much this ramp changes the output slope.

Remark

Why ReLU rather than the sigmoid or tanh activations used in earlier networks? A sigmoid squashes its input into $(0, 1)$, so for large positive or negative inputs the derivative is nearly zero. Gradients that pass through many such saturated units shrink exponentially, starving earlier layers of learning signal—the *vanishing gradient* problem. ReLU has derivative one whenever the input is positive, so it transmits gradients without shrinking them. This simple property made deep training practical and is why ReLU became the default activation.

The construction is easiest to see directly. Suppose a continuous piecewise-linear function has knots $\tau_1 < \dots < \tau_K$, leftmost slope s_0 , slope s_i after knot τ_i , and baseline intercept c , so $c + s_0x$ is the affine piece before the first knot. Then

$$g(x) = c + s_0x + \sum_{i=1}^K (s_i - s_{i-1}) \text{ReLU}(x - \tau_i). \quad (1.6)$$

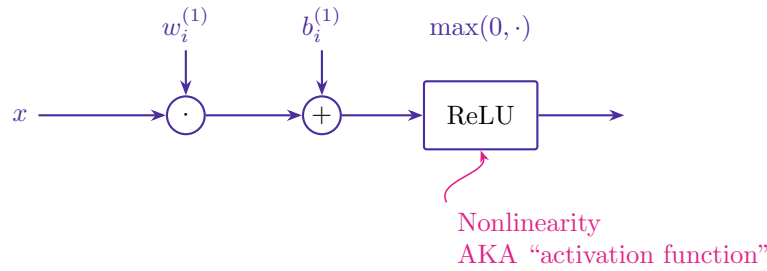


Figure 1.2: One ReLU unit as a computation graph: multiply by $w_i^{(1)}$, add $b_i^{(1)}$, and apply $\max(0, \cdot)$.

Each activated ramp adds exactly the required change in slope. The linear term itself can be written as $x = \text{ReLU}(x) - \text{ReLU}(-x)$, so the whole expression has the standard one-hidden-layer form

$$N_{\theta}(x) = b^{(2)} + \sum_{i=1}^d w_i^{(2)} \text{ReLU}\left(w_i^{(1)}x + b_i^{(1)}\right). \quad (1.7)$$

Thus any continuous univariate piecewise-linear function can be represented exactly by a sufficiently wide one-hidden-layer ReLU network, and continuous targets on compact intervals can be approximated by refining the piecewise-linear approximation (Figure 1.3).

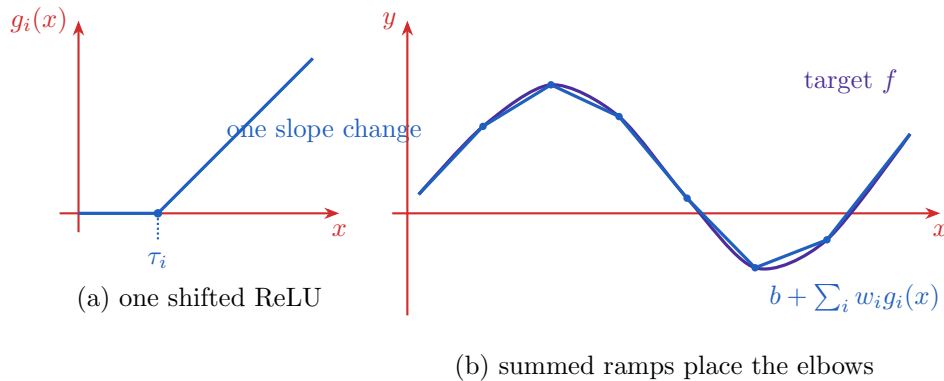


Figure 1.3: Each shifted ReLU supplies one controllable slope change. Their sum places the elbows in Equation (1.7) and steers the piecewise-linear output toward the target.

Remark

The approximation statement concerns a compact domain and a sufficiently wide network. It does not say that a particular finite network approximates every function equally well, nor that gradient descent will automatically find the constructive parameters in Equation (1.6).

1.3 From scalar ramps to a layer

Parallel multiply-accumulate operations are exactly what modern accelerators perform well. Collect the d hidden ramps into the vector

$$\vec{h} = \text{ReLU}\left(W^{(1)}x + \vec{b}^{(1)}\right) \in \mathbb{R}^d, \quad (1.8)$$

where $W^{(1)} \in \mathbb{R}^{d \times 1}$ and $\vec{b}^{(1)} \in \mathbb{R}^d$. A linear readout, shown componentwise in Figure 1.4, gives

$$N_\theta(x) = W^{(2)}\vec{h} + b^{(2)}, \quad W^{(2)} \in \mathbb{R}^{1 \times d}. \quad (1.9)$$

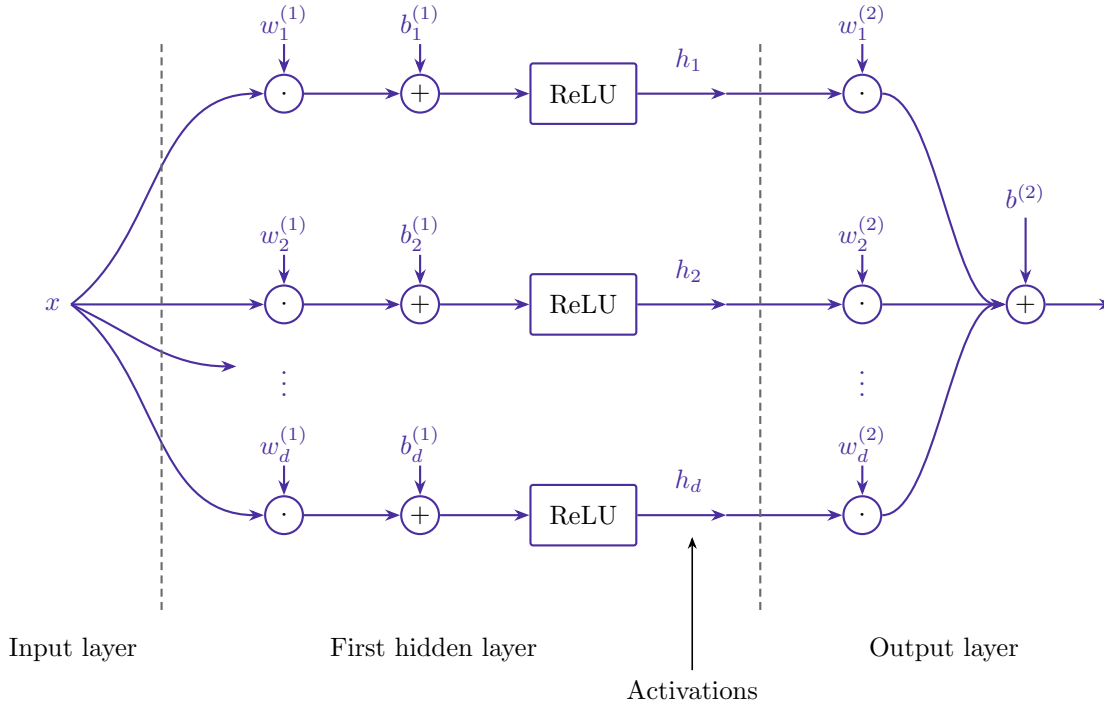


Figure 1.4: Vectorized computation graph for a single-hidden-layer network. The hidden units are evaluated in parallel and then combined by a linear readout.

The same pattern works when the input and output are vectors instead of scalars: multiply by a weight matrix, add a bias, apply ReLU, then multiply by another weight matrix and add another bias. A deeper network repeats this pattern several times. The nonlinearities are essential—without them, stacking multiple affine maps would collapse into a single affine map, and depth would add nothing.

1.4 Empirical risk and generalization

For paired data (x_i, y_i) and a model f_θ , empirical risk minimization solves

$$\hat{\theta} = \arg \min_{\theta} \frac{1}{n} \sum_{i=1}^n \ell(y_i, f_\theta(x_i)). \quad (1.10)$$

The loss must express prediction quality and also permit the chosen optimizer to make progress. Squared error and cross-entropy are standard examples; a nondifferentiable task metric is often replaced by a differentiable surrogate during training.

It helps to separate four ideas, shown across the top row of Figure 1.5: the intended deployment outcome, the metric used to evaluate it, the surrogate optimized during training, and the route used to estimate an update. A useful surrogate expresses the relevant preference, provides usable local

information, behaves numerically, and is affordable to optimize. These requirements need not select the same mathematical object as the deployment metric.

When an ordinary differentiable surrogate is unavailable, other update routes—randomized estimators, relaxations, or structured optimization—can still supply useful local information.

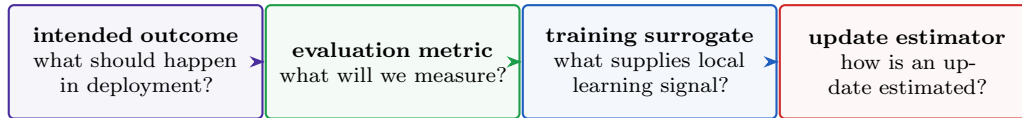


Figure 1.5: Separating four objects: the intended deployment outcome, the evaluation metric, the training surrogate, and the update estimator.

Training loss is not the final objective. If samples are drawn from a joint distribution $P(x, y)$, the ideal target is the population risk

$$\theta^* = \arg \min_{\theta} \mathbb{E}_{(x,y) \sim P} [\ell(y, f_{\theta}(x))] . \quad (1.11)$$

The distribution is unknown, so a held-out validation set guides model and hyperparameter choices, while a test set estimates performance only after those choices are made. This reasoning assumes that the data used for evaluation represent the population of interest; distribution shift is precisely a failure of that assumption.

Pitfall

Empirical risk is observable and optimizable; population performance and the real-world objective are not. Training is therefore another form of “looking where the light is.” The proxy is useful only if its mismatch with the real goal remains explicit, and assumptions that make analysis possible must be paired with engineering work that tests the bridge to reality.

Remark

All this uncertainty might make you feel uneasy. So much of your education up to this point has taught you clean stories about things humanity understood only after centuries and millennia of effort. Keeping that historical perspective in mind brings a bit of comfort—at least you are not alone. Craft can precede theory—ceramics and cathedrals preceded modern material models—but “alchemy” licenses careful experiments, not ritual. Ask why methods work, how their inductive biases affect generalization, and where their optimization geometry permits learning. In this rapidly evolving area, that kind of curiosity will make questions stick in the back of your mind. Then, when you hear about what might otherwise seem like an unrelated advance, you will be in a position to make the connection and stay a bit ahead of the crowd.

Remark

How can we tell whether deployment data are close enough to the training distribution? A classifier trained on cats and dogs but asked to classify dinosaurs makes the mismatch clear. Density estimation, representative synthetic or acquired samples, and additional labeling may help detect or repair shift, but none guarantees that every shift has been found or fixed. Subtler shifts are more dangerous. Zech et al. (2018) trained a CNN to detect pneumonia from chest X-rays using images from three hospitals. Internally the model looked strong

(AUC 0.93 on held-out data from the same hospitals), but when tested on X-rays from a new hospital it had never seen, performance dropped significantly (AUC 0.82). Why? The CNN could identify which hospital took an image with over 99.9% accuracy—it was picking up on scanner-specific signatures, patient positioning conventions, and text overlays rather than the lung pathology itself. Because pneumonia prevalence differed sharply across hospitals (34% at one site versus 1% at another), a trivial model that simply guessed based on hospital identity—ignoring the image entirely—already achieved an AUC of 0.86 on the pooled internal test set. The CNN had learned a shortcut that happened to work internally but was useless at a new site.

Hold out data depends on the deployment. The unit that must be held out depends on deployment. It may be a person, patient, source document, time period, molecule scaffold, or near-duplicate cluster rather than one row. Splitting rows at random can leak the same entity into training and evaluation. But making the test set maximally different is not the goal either. The split should answer the deployment question without introducing an unrelated distribution shift (Kapoor and Narayanan 2023) (Figure 1.6).

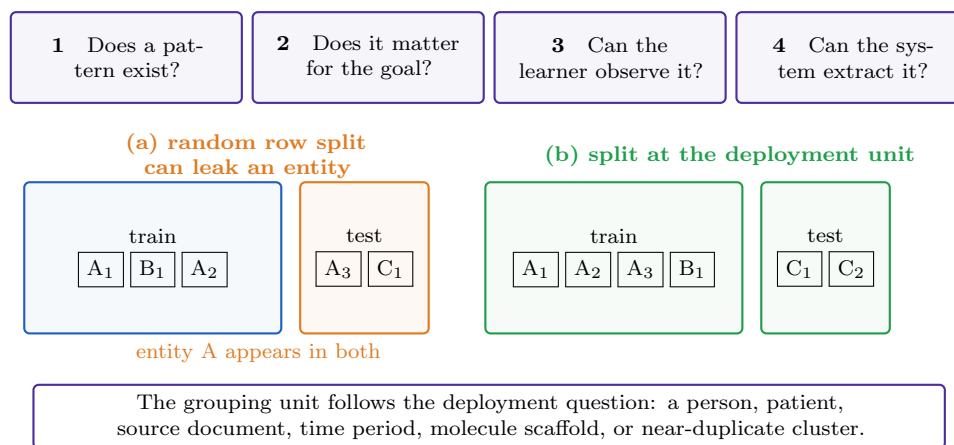


Figure 1.6: Problem formulation begins with four questions, while evaluation must split at the unit that will be new in deployment. Randomly choosing certain rows of the training data to hold out or test risks having similar examples show up in the train and test datasets and therefore not give a true sense of the model generalization performance.

When a model fits the training data but performs poorly on new samples, it overfits. Regularization adds another pressure or source of information. Ridge regression, for example, balances fit and parameter size:

$$\arg \min_{\vec{w}} (\|X\vec{w} - \vec{y}\|_2^2 + \lambda \|\vec{w}\|_2^2). \quad (1.12)$$

That’s why traditional machine learning says *Love the training data—but not that much*. Keep enough distance that a finite sample does not dictate every weakly supported detail. Regularization encodes that second pressure, but still does not guarantee improved generalization.

Remark

A distinction that will recur throughout the course: parameters (such as the weights $\vec{w}$) are learned by the optimizer from training data, while hyperparameters (such as the regularization

strength λ , the learning rate, or the number of hidden units) are chosen by the engineer. Hyperparameters are selected using validation data rather than by repeatedly inspecting the final test set. Typically, we search over orders of magnitude for these.

1.5 One function-learning vocabulary for many tasks

Deep Learning

Supervised

Regression

Classification

Localized Annotation

→ semantic segmentation

Unsupervised

Learned embeddings

→ like PCA (dimensionality reduction)

Generative models

→ one route: density estimation

Foundation models

broad pretraining → prompting or adaptation

Figure 1.7: A working vocabulary: supervised regression, classification, and localized annotation; unlabeled-data representation learning and generation; and foundation models built by broad pretraining followed by prompting or adaptation.

Thinking about generation specifically — density estimation is *one* mathematical route to generation, not its definition. Some generative systems are easier to describe by how they produce samples than by a tractable normalized density, and some questions about generation fit density-estimation language only awkwardly. Figure 1.5 keeps the task separate from one convenient modeling route. It's something to keep in mind going forward.

Before those modeling choices, ask whether a relevant pattern exists, whether it matters for the intended use, whether the learner can observe it in the available data, and whether the learning system can extract it. These questions and the split diagnostic in Figure 1.6 do not replace the function, data, loss, and generalization checklist. They ask whether the learning problem is coherent in the first place.

Recap

- Shifted ReLU ramps provide controllable slope changes; their sum represents any continuous univariate piecewise-linear function.
- Vectorization turns parallel ramps into the familiar affine-ReLU-affine layer computation.
- Empirical risk is observable, population risk is the real target, and held-out data mediate between them.

- The same function-learning language covers supervised, representation-learning, generative, and foundation-model settings.

References and Further Reading

- Kapoor, S. and A. Narayanan (2023). “Leakage and the Reproducibility Crisis in Machine-Learning-Based Science”. In: *Patterns* 4.9, p. 100804. DOI: [10.1016/j.patter.2023.100804](https://doi.org/10.1016/j.patter.2023.100804). URL: <https://doi.org/10.1016/j.patter.2023.100804>.
- Zech, J. R., M. A. Badgeley, M. Liu, A. B. Costa, J. J. Titano, and E. K. Oermann (2018). “Variable Generalization Performance of a Deep Learning Model to Detect Pneumonia in Chest Radiographs: A Cross-Sectional Study”. In: *PLOS Medicine* 15.11, e1002683. DOI: [10.1371/journal.pmed.1002683](https://doi.org/10.1371/journal.pmed.1002683).