Alon Rosen

Model Extraction

Never ask a model her weights.[*]

The basic goal of *model extraction* is to reproduce a model's behavior on typical inputs, given only input–output access. Usually the model's training procedure and internal representation are hidden. The interface may return only labels, or richer outputs such as probabilities or logits, which can reveal information about parameters and hidden structure.

Numerical outputs can provide *equations* for unknown parameters. In ReLU networks, changes in the input–output function can reveal *hidden neurons*. Queried outputs can also be used to train a smaller model to imitate the original one, a procedure known as *model distillation*.

Extraction can nevertheless be computationally hard, and exact behavior need not determine unique parameters.

The main points of this chapter are:

- Extraction asks to reconstruct a model's internal representation or reproduce its behavior.
- Numerical outputs yield equations; ReLU boundaries reveal hidden neurons.
- Model distillation trains a surrogate by gradient descent on queried outputs.

[*] @poteto, X.

1 Model extraction

Training a model can require far more computation than evaluating it. Once a trained model is available through an interface, it is natural to ask whether its behavior can be reproduced more cheaply by querying it. The answer can depend on what is known about the model's structure and parameterization, and on what the deployed system returns as output.

Understanding when extraction is possible is useful both for limiting unintended extraction and for deliberately transferring a model's behavior to a smaller surrogate.

1.1 Functional extraction

Suppose a trained classifier $f : X \to Y$ has been deployed. We would like to construct another classifier g that agrees with f on most inputs drawn from a distribution D.

Definition 1 (Functional extraction)**.** *Let $f : X \to Y$ be a deterministic model and let D be a distribution on X. An extractor* ε-extracts *f over D if it outputs $g : X \to Y$ such that*

$$\Pr_{x \leftarrow D}[g(x) \neq f(x)] \leq \varepsilon.$$

If the extractor receives independent examples $(x, f(x))$ with $x \leftarrow D$, this is ordinary PAC learning with target f; if the learner instead chooses inputs adaptively and receives $f(x)$ in reply, it has membership-query access. The interface may also reveal more than labels, for example *probabilities* or *logits*.

Functional extraction asks only to reproduce input–output behavior. If $f = f_\theta$ is parameterized by θ, one may ask for the stronger task of recovering θ itself; reproducing the function need not uniquely determine its parameters.

Oracle access alone does not make extraction easy. We already saw that pseudorandom functions can remain hard to learn even with membership queries, and that fresh samples can sometimes be generated without recovering the hidden representation.

1.2 Distributional extraction

In generative models, extraction should reproduce a distribution rather than a particular reply. A randomized model may return different outputs on repeated queries, so the definition of functional extraction should be adapted.

Definition 2 (Distributional extraction)**.** *Let $F : X \to Y$ be a randomized model and let D be a distribution on X. An extractor* ε-extracts *F over D if it outputs a randomized model $G : X \to Y$ such that, for $X \leftarrow D$,*

$$d_{\mathrm{TV}}\big((X, F(X)), (X, G(X))\big) \leq \varepsilon.$$

If $F = f$ and $G = g$ are deterministic, then the two joint distributions differ exactly on inputs x for which $f(x) \neq g(x)$. Hence their total variation distance is $\Pr_{x \leftarrow D}[f(x) \neq g(x)]$, exactly the criterion in Definition 1.

An autoregressive generator is a randomized model: on input x it repeatedly samples the next symbol conditioned on x and the prefix already generated. Thus $F(x)$ is the distribution over complete sequences produced by this process. If the next-symbol distributions of F and G are within δ after every common prefix, then their length-T output distributions are within $T\delta$: couple each step and union-bound.

2 Outputs as equations

Some interfaces return numerical information rather than only a prediction. When the form of the model is known, each such output can impose an equation on the unknown parameters. Extraction can then become a problem of solving equations. We begin with linear equations, and then see how probabilities and logits can be transformed into linear equations as well.

2.1 Linear equations

A basic example is *linear equations*. For unknown $\mathbf{s} \in \mathbb{F}^n$, let

$$b = f_{\mathbf{s}}(\mathbf{a}) = \mathbf{a}^\top \mathbf{s}.$$

Proposition 1 (Linear equations)**.** *The parameters consistent with $A\mathbf{s} = \mathbf{b}$ are exactly $\mathbf{s} + \ker A$.*

Proof. If $\mathbf{s}' = \mathbf{s} + \mathbf{z}$ with $\mathbf{z} \in \ker A$, then $A\mathbf{s}' = A\mathbf{s} = \mathbf{b}$, so $\mathbf{s}'$ gives exactly the same answers. Conversely, if $A\mathbf{s}' = \mathbf{b} = A\mathbf{s}$, then $A(\mathbf{s}' - \mathbf{s}) = 0$, hence $\mathbf{s}' - \mathbf{s} \in \ker A$. Thus any consistent $\mathbf{s}'$ exactly matches all observed answers, even when $\mathbf{s}$ is not determined. □

The observations need not determine the original parameters in order to support a model consistent with everything seen so far. Any representative of $\mathbf{s} + \ker A$ reproduces all queried outputs exactly; parameter recovery may require more information than behavioral emulation.

Over the reals, noisy linear equations are *linear regression*: from $\mathbf{b} = A\mathbf{s} + \mathbf{e}$, we estimate $\mathbf{s}$.

Proposition 2 (Least-squares stability)**.** *If A has full column rank, then the least-squares estimate satisfies $\|\widehat{\mathbf{s}} - \mathbf{s}\|_2 \le \|\mathbf{e}\|_2 / \sigma_{\min}(A)$.*

Proof. Let $A^\dagger = (A^\top A)^{-1} A^\top$. The least-squares solution is

$$\widehat{\mathbf{s}} = A^\dagger \mathbf{b} = (A^\top A)^{-1} A^\top \mathbf{b}.$$

Since $A^\dagger A = I$, we have $\widehat{\mathbf{s}} - \mathbf{s} = A^\dagger \mathbf{e}$. Hence $\|\widehat{\mathbf{s}} - \mathbf{s}\|_2 \le \|A^\dagger\|_2 \|\mathbf{e}\|_2 = \|\mathbf{e}\|_2 / \sigma_{\min}(A)$. □

Thus approximate outputs can still recover the parameters accurately when the chosen queries give a *well conditioned* system. The quantity $\sigma_{\min}(A)$ measures how close the measurements are to losing information about some direction in parameter space. If it is small, a small output error can produce a much larger parameter error; if it is bounded away from zero, recovery is stable. Query choice therefore affects not only whether the parameters are determined, but also how robustly they can be recovered from finite-precision outputs.

When the equations are over $\mathbb{F}_2$ and noisy, extraction does not appear to be easy. With uniform rows $\mathbf{a}_i$, consider

$$\mathbf{b} = A\mathbf{s} + \mathbf{e} \pmod 2, \qquad e_i \leftarrow \mathrm{Ber}(\eta).$$

This is LPN, which is conjectured hard even when the samples contain enough information to determine $\mathbf{s}$ uniquely; the difficulty is *computational* rather than *information-theoretic*.

Proposition 3 (LPN with chosen queries)**.** *LPN can be learned efficiently with chosen queries.*

Proof. Fix $\eta < 1/2$. Query each basis vector $\mathbf{e}_j$ repeatedly and take a majority vote. To recover s_j with error probability δ, a Chernoff bound shows that $O\big(\log(1/\delta)/(1-2\eta)^2\big)$ repetitions suffice. □

2.2 Linear equations in the exponent

Labels reveal only the predicted class, while probabilities and logits retain numerical information. In logistic and softmax models, simple logarithms yield linear equations in the unknown parameters.

For binary *logistic regression*, the model predicts by thresholding at 1/2 the probability

$$p(x) = \frac{1}{1 + e^{-(\mathbf{w}^\top x + b)}},$$

with unknown $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$; the map $z \mapsto 1/(1 + e^{-z})$ is the *sigmoid*.

With labels alone, a query reveals only the sign of $\mathbf{w}^\top x + b$. Chosen queries can recover the separating hyperplane arbitrarily well, but never its scale: $(\mathbf{w}, b)$ and $c(\mathbf{w}, b)$, $c > 0$, give identical labels. Exact probabilities instead give linear equations in $\mathbf{w}$ and b.

Proposition 4 (Extracting logistic regression)**.** *If exact probabilities are returned, the logistic-regression parameters can be extracted efficiently.*

Proof. Invert the sigmoid to obtain, for every query x,

$$\log \frac{p(x)}{1 - p(x)} = \mathbf{w}^\top x + b.$$

The query at 0 gives b. The query at $\mathbf{e}_i$ gives $w_i + b$, and hence w_i. □

Multiclass classification assigns each class $j \in \{1, \dots, K\}$ the score

$$\ell_j(x) = \mathbf{w}_j^\top x + b_j.$$

The predicted label is the one with largest score. The values $\ell_j(x)$ are the same *logits* we saw in Chapter 6. There, softmax turned logits into probabilities; here we ask what the information returned by the model reveals about the logits and their parameters.

Proposition 5 (Extracting multiclass linear classifiers)**.** *For the multiclass linear model above:*

1. *if the interface returns the exact logits, then all parameters can be extracted efficiently;*
2. *if softmax probabilities are returned, parameters are extractable up to an affine function.*

Proof. Exact logits give the parameters directly: query 0 to obtain every b_j, then query $\mathbf{e}_i$ to obtain $w_{j,i} + b_j$ and hence every weight.

For softmax probabilities,

$$p_j(x) = \frac{e^{\ell_j(x)}}{\sum_{r=1}^{K} e^{\ell_r(x)}}.$$

Taking ratios eliminates the normalizing denominator:

$$\log \frac{p_j(x)}{p_r(x)} = \ell_j(x) - \ell_r(x).$$

Fixing one label r, the same queries therefore recover $\mathbf{w}_j - \mathbf{w}_r$ and $b_j - b_r$ for every j. This determines an equivalent model, but not the original parameters: adding the same affine function $\mathbf{u}^\top x + c$ to every logit leaves all softmax probabilities unchanged. □

Finite-precision probabilities and logits give only approximate equations. As in linear regression, recovery accuracy depends on output precision and conditioning.

3 Piecewise-linear networks

Even function values can reveal ReLU parameters: small input changes recover local affine behavior, whose changes give equations for hidden neurons. Consider a one-hidden-layer network on $x \in \mathbb{R}^d$,

$$f(x) = \sum_{j=1}^{m} a_j \,\mathrm{ReLU}(\mathbf{w}_j^\top x + b_j), \qquad \mathrm{ReLU}(z) = \max\{0, z\}.$$

For neuron j, let $z_j(x) = \mathbf{w}_j^\top x + b_j$; it is active if $z_j(x) > 0$. The hyperplane $z_j(x) = 0$ is its *switching boundary*: crossing it switches neuron j on or off. On any region where the active set $S \subseteq [m]$ is fixed, f is affine with $\nabla f = \sum_{j \in S} a_j \mathbf{w}_j$. Its d linear coefficients and constant term are determined by values at $d+1$ nearby affinely independent points.

Many neurons may be active at once, but their switching boundaries are distinct hyperplanes. A simultaneous switch occurs only if the probing line meets the intersection of two boundaries.

Claim 1. *For every $\varepsilon > 0$, there is $\delta > 0$ such that, with probability at least $1 - \varepsilon$, all crossings of a random line with a finite set of distinct hyperplanes are separated by at least δ.*

Proof. For each pair of hyperplanes, the probability that their crossing locations differ by less than δ tends to 0 as $\delta \to 0$, since exact coincidence requires the line to meet their intersection. There are finitely many pairs, so choose δ small enough that the sum of these probabilities is at most ε; the union bound gives the claim. □

The extraction procedure is:

- Probe a random line; slope changes of $f(x(t))$ locate isolated boundary crossings.
- Compare the local gradients on the two sides to obtain the switching neuron's contribution.
- Shift the line and match equal gradient jumps; the matched points recover its hyperplane.

Claim 2 (A boundary reveals one neuron)**.** *At an isolated crossing, the change in local gradient is exactly the contribution of the neuron that switches.*

Proof. Suppose only neuron j changes. If S is active on one side, $\nabla f_- = \sum_{i \in S} a_i \mathbf{w}_i$; if j turns on, $\nabla f_+ = \nabla f_- + a_j \mathbf{w}_j$, with the roles reversed if it turns off. Subtraction cancels every unchanged neuron, leaving $\nabla f_+ - \nabla f_- = \pm a_j \mathbf{w}_j$. □

One crossing gives $a_j \mathbf{w}_j$; matched crossings give points on $\mathbf{w}_j^\top x + b_j = 0$. Together they recover the neuron's contribution and boundary up to scale. Repeating this reconstructs the hidden layer under suitable nondegeneracy and precision assumptions.

For a multi-layer network, only the final output is observed. Claim 2 gives the same local mechanism: at a generic crossing, one neuron in one layer switches and changes the final-output gradient. Nearby queries therefore reveal a local piece of its switching boundary.

First-layer boundaries are easiest to identify: each pre-activation is affine in the input, so all pieces lie on one hyperplane. In later layers the affine formula depends on earlier activation states; when one changes, the next piece may lie on a different hyperplane. Once the first layer is recovered, its activations can be computed and later layers treated in turn. At finite precision, errors in earlier layers propagate and may be amplified with depth, so later layers are less stable to recover.

Exact query access still leaves intrinsic ambiguity: hidden neurons may be permuted, and for $c > 0$, $a_j \,\mathrm{ReLU}(\mathbf{w}_j^\top x + b_j) = (a_j/c)\,\mathrm{ReLU}((c\mathbf{w}_j)^\top x + c b_j)$. Thus recovery is only up to permutation and positive rescaling.

4 Distillation

Parameter recovery is not always the goal. For functional extraction, queried outputs can instead be used as training targets for a surrogate model. This procedure is called *distillation*.

For a classifier, choose inputs $x_1, \dots, x_m$, query the original model, and train the surrogate on the returned outputs. If query x_i returns class probabilities $p_i(y)$, a natural objective is

$$L(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{y=1}^{K} p_i(y) \log q_\theta(y \mid x_i),$$

where q_θ is the surrogate. With labels alone, p_i is concentrated on the returned label, giving ordinary training on examples labeled by the original model. Probabilities or logits provide richer targets because they also describe how the model compares the alternatives.

For a generative model the same idea is applied one symbol at a time. Given a prompt x and a prefix $y_{<t}$, let $p_t(a)$ be the original model's next-symbol distribution. The surrogate can be trained to minimize

$$L(\theta) = -\sum_{t} \sum_{a} p_t(a) \log q_\theta(a \mid x, y_{<t}).$$

If only sampled outputs are available, one instead trains on sequences generated by the original model using the usual next-symbol loss. In either case, training can be carried out by ordinary gradient descent.

Distillation is not synonymous with faithful extraction. A student may achieve similar task accuracy while disagreeing with the teacher on many inputs; faithful extraction instead asks to reproduce the behavior of the particular deployed model.

The surrogate need not share the original architecture or parameters and may be substantially smaller. Success depends on whether the queried inputs cover the distribution of interest and whether the surrogate can represent the relevant behavior. Distillation can therefore be used deliberately for model compression or, from query access alone, to copy a deployed model's behavior.

5 Historical notes

Knowledge distillation was introduced as a way to transfer predictive behavior from a teacher model to a smaller student [1]. Tramèr et al. studied model extraction through prediction APIs and showed how chosen queries and confidence values can be used to recover several model classes [2]. Jagielski et al. later distinguished extraction aimed at task accuracy from extraction aimed at reproducing the particular target model [3].

Rolnick and Kording used the geometry of linear regions to reverse engineer ReLU networks [4]. Carlini, Jagielski and Mironov developed a cryptanalytic attack based on critical points and high-precision outputs [5]. Canales-Martínez et al. gave polynomial-time cryptanalytic extraction algorithms under suitable assumptions [6].

A related line of work asks when a network's input–output function determines its parameters, possibly only up to intrinsic symmetries. Vlačić and Bölcskei studied affine symmetries of neural networks [7]. Bona-Pellissier, Bachoc and Malgouyres gave conditions under which deep ReLU networks are determined up to permutation and positive rescaling [8].

References

[1] G. Hinton, O. Vinyals, and J. Dean. *Distilling the Knowledge in a Neural Network*. arXiv:1503.02531, 2015.

[2] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart. *Stealing Machine Learning Models via Prediction APIs*. USENIX Security, 2016.

[3] M. Jagielski, N. Carlini, D. Berthelot, A. Kurakin, and N. Papernot. *High Accuracy and High Fidelity Extraction of Neural Networks*. USENIX Security, 2020.

[4] D. Rolnick and K. Kording. *Reverse-Engineering Deep ReLU Networks*. ICML, 2020.

[5] N. Carlini, M. Jagielski, and I. Mironov. *Cryptanalytic Extraction of Neural Network Models*. CRYPTO, 2020.

[6] I. A. Canales-Martínez, J. Chávez-Saab, A. Hambitzer, F. Rodríguez-Henríquez, N. Satpute, and A. Shamir. *Polynomial Time Cryptanalytic Extraction of Neural Network Models*. EUROCRYPT, 2024.

[7] V. Vlačić and H. Bölcskei. *Affine Symmetries and Neural Network Identifiability*. Advances in Mathematics 376, 107485, 2021.

[8] J. Bona-Pellissier, F. Bachoc, and F. Malgouyres. *Parameter Identifiability of a Deep Feedforward ReLU Neural Network*. Machine Learning 112(11), 4431–4493, 2023.