Lecture outline: neural nets

Machine learning, Summer 2023 (Geelon So)

Outline

(1) Why has deep learning been successful? (history, applications)

(2) What is a neural net? (multilayer perceptron)

(3) What can a neural net compute? (expressivity and universal approximation)

(4) How to train a neural net? (SGD, and backpropagation)

(5) How much data is required to learn? (generalization of neural nets, VC bound)

(6) What's left? (beyond MLPs, double descent/overparametrization, NTKs)

Summary A neural network architecture parametrizes a family of functions $f_\theta : \mathbb{R}^n \to \mathbb{R}^k$. As the number of parameters grow, the family becomes more *expressive* (i.e. more complicated functions can be computed). Empirically, we can find good settings of parameters by *fitting* the neural net to data (training on examples) via stochastic gradient descent (SGD), which can be performed efficiently using backpropagation (dynamic programming). It is an active area of research to understand why this works (i.e. why the trained network *generalizes* to unseen instances not in training data).

Resources

- https://www.cs.huji.ac.il/~shais/UnderstandingMachineLearning
- https://cseweb.ucsd.edu/~dasgupta/250B/neural-nets-handout.pdf
- https://www.deeplearningbook.org

1 Successes of deep learning

This section drawn from Goodfellow et al. (2016).

- Three waves of development
  - (i) 1940s–1960s cybernetics, Hebbian learning, the perceptron
  - (ii) 1980s–1990s connectionism/artificial neural networks, backpropagation for shallow nets
  - (iii) 2006–*present* deep learning with larger models, more data (cheap memory, default is to record data, crowdsourcing) and compute infrastructure (GPUs and specialized hardware)
- Upward trends in data, models, compute (Figures 1 to 4); accuracy and energy usage as well
- Applications (extremely non-exhaustive)
  - Computer vision/graphics: image recognition, autonomous vehicles, generative models
  - Natural language: machine translation, personal assistants, large language models
  - Commerce/finance: ads, fraud detection, personalization, time-series prediction
  - Recommendation systems: search engines, content-based recommendations
  - Science: drug discovery, climate models

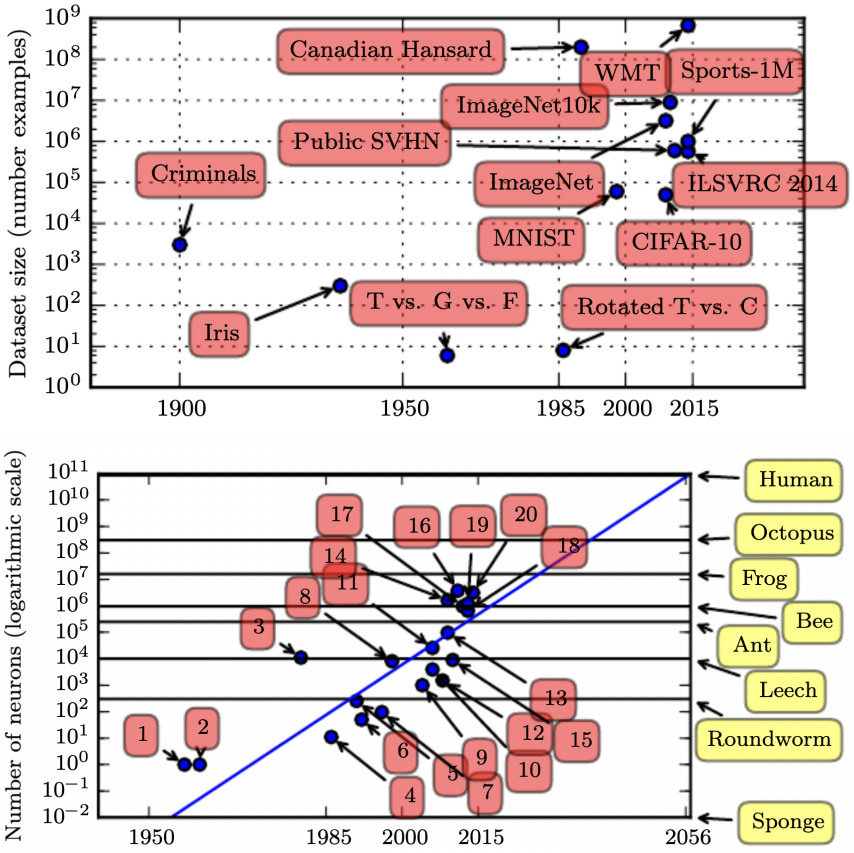


Figure 1: Growth of dataset and model size over the years. (Goodfellow et al., 2016)

Figure 2: Growth of model size. (Villalobos et al., 2022)

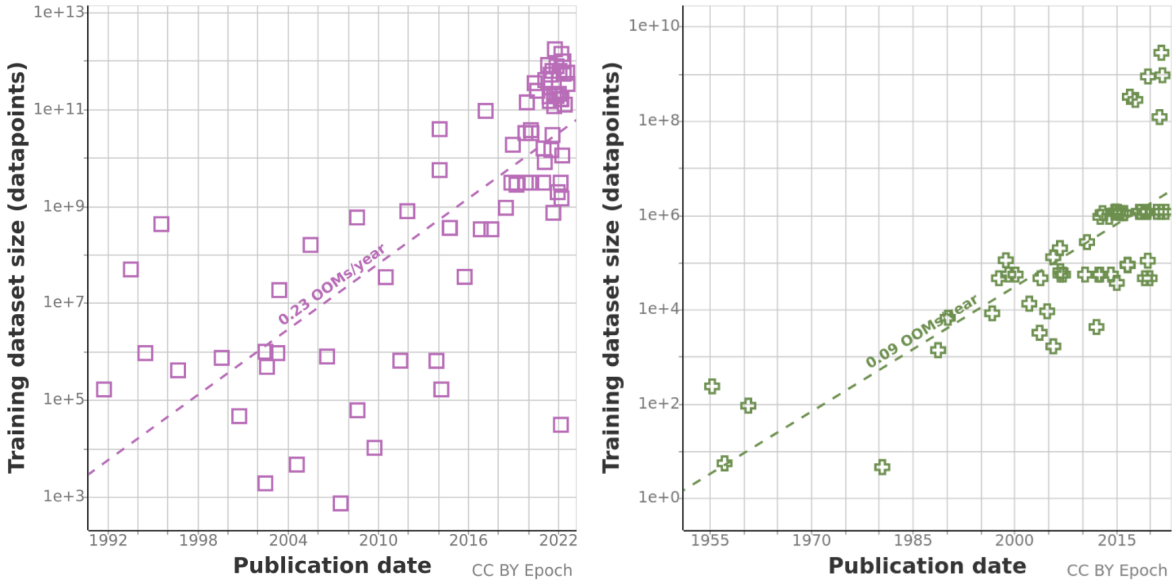


Figure 3: Growth of dataset size. (Villalobos and Ho, 2022)

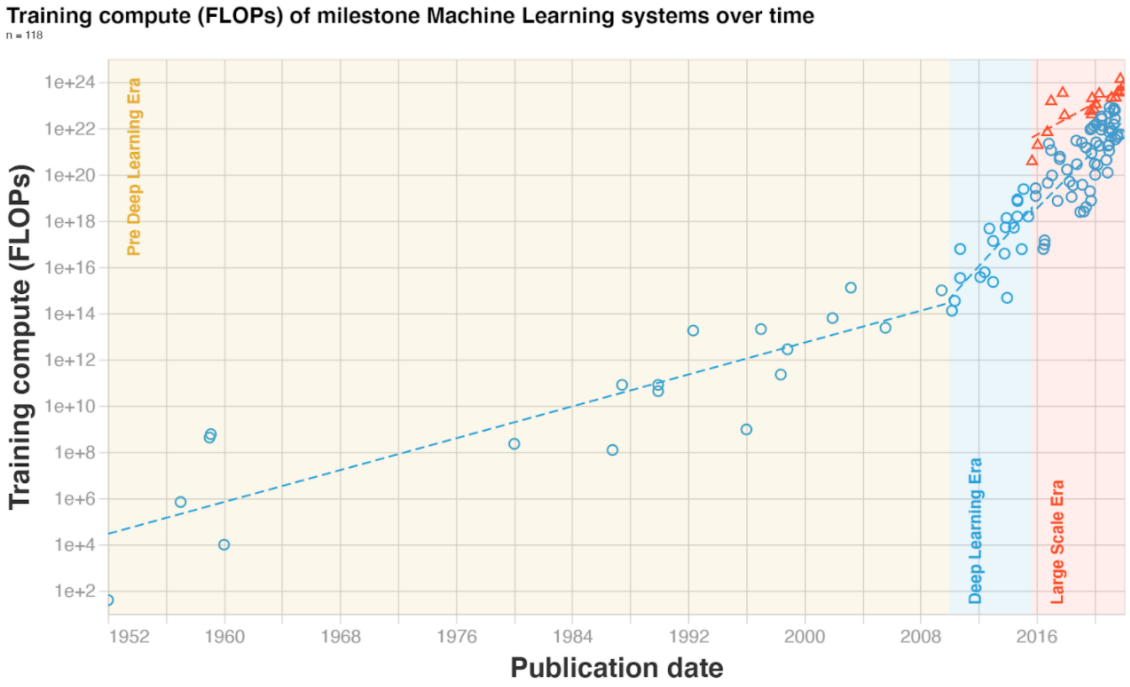
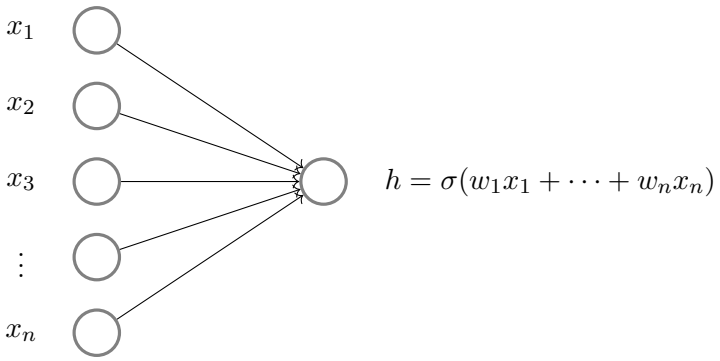


Figure 4: Growth of compute power. (Sevilla et al., 2022)

2 Defining the neural net

This section drawn from Sanjoy Dasgupta's CSE 250B lecture on neural nets.

The **neuron** is the basic unit of computation in a **neural net**:



- $x_1, \ldots, x_n \in \mathbb{R}$ are **inputs** to the neuron (right vertex in figure)
- $w_1, \ldots, w_n \in \mathbb{R}$ are **parameters** (also called **weights**, one for each edge)
- $\sigma : \mathbb{R} \to \mathbb{R}$ is a (non-linear) **activation function**
- h is the **output** of the neuron

For short, let $\mathbf{x}, \mathbf{w} \in \mathbb{R}^n$ be the input and weight vectors. The neuron performs a simple computation:

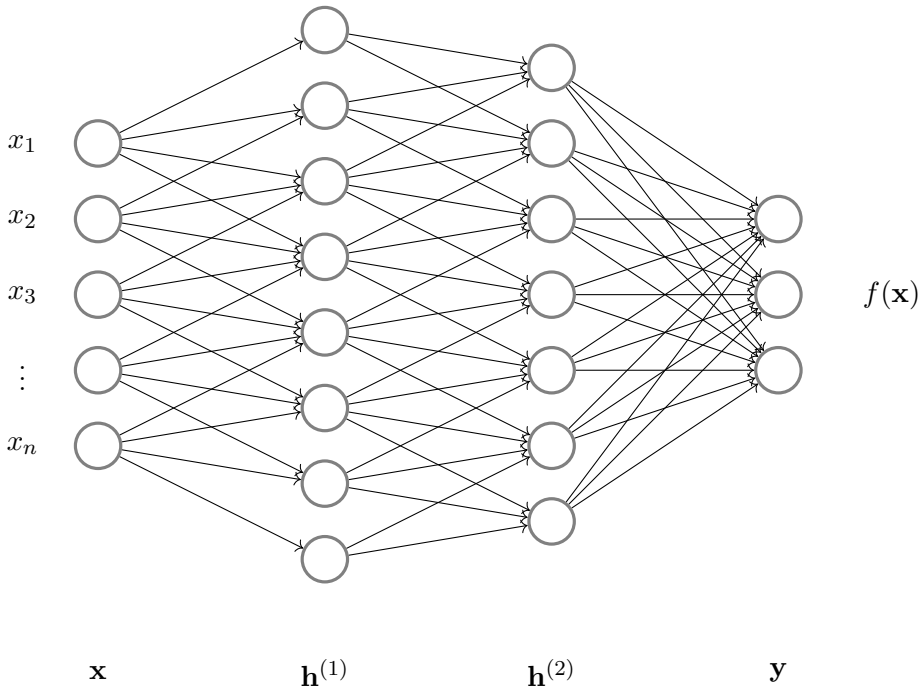
$$h = \sigma(\mathbf{w} \cdot \mathbf{x}).$$

2.1 Examples of activation functions

- threshold function: $\sigma(z) = \begin{cases} 1 & z \geq 0 \\ 0 & z < 0 \end{cases}$
- rectified linear unit (ReLU): $\sigma(z) = \max(0, z)$
- sigmoid: $\sigma(z) = \dfrac{1}{1 + e^{-z}}$
- hyperbolic tangent: $\sigma(z) = \tanh(z)$

2.2 Composition of neurons

In a **feed-forward network** $f : \mathbb{R}^n \to \mathbb{R}^k$, we *compose* neurons with a directed graph $G = (V, E)$:



- When the graph is acyclic (no loops), we can arrange the neurons into **layers**
  - **input layer** (left-most vertices) corresponds to the input $\mathbf{x} \in \mathbb{R}^n$
  - **hidden layers** (inner vertices) are intermediate computations
  - **output layer** (right-most vertices) corresponds to the output $f(\mathbf{x}) \in \mathbb{R}^k$
- In this case, the network is called a **multi-layer perceptron** (MLP).

- Each neuron $h_i^{(\ell)}$ in the **hidden layer** receives the outputs to which it is connected in the previous layer, and it computes $h_i^{(\ell)} = \sigma\big(\mathbf{w}_i^{(\ell)} \cdot \mathbf{h}^{(\ell-1)}\big)$. For short, we write:

$$\mathbf{h}' = \sigma\big(W\mathbf{h}\big),$$

where $\mathbf{h}'$ corresponds to the output of the current layer, W the matrix of weights, and $\mathbf{h}$ the output of the previous layer. Here, σ acts coordinate-wise.

- Each final neurons in the **output layer** usually do not have activations. For regression:

$$\mathbf{y} = W\mathbf{h}^{(\ell)},$$

where $\mathbf{h}^{(\ell)}$ is the output of the last hidden layer. To perform classification, we often want the output of the neural net $f(\mathbf{x})$ to be a probability:

$$f(\mathbf{x})_i = \text{probability that } \mathbf{x} \text{ belongs to class } i.$$

To do this, we can apply a **softmax**:

$$\mathbf{y} = \text{softmax}(W\mathbf{h}^{(\ell)}),$$

where $\text{softmax}(\mathbf{z})_i = \dfrac{e^{z_i}}{e^{z_1} + \cdots + e^{z_k}}$.

- A family $\mathcal{F}$ of neural nets f can be specified by the graph G and the choice of activation σ. We call the pair (G, σ) the **architecture** of the neural net.
  - We **parametrize** this family of neural nets by their weights, $\theta = (W^{(1)}, \ldots, W^{(\ell)}, W^{(o)})$,

$$f_\theta(\mathbf{x}) := W^{(o)}\sigma\big(W^{(\ell)}\sigma(\cdots\sigma(W^{(1)}\mathbf{x})\cdots\big).$$

3 Expressivity of neural nets

This section drawn from Chapter 20 of Shalev-Shwartz and Ben-David (2014).

3.1 Boolean functions

Proposition 1. *There is a family of neural networks whose architecture has a single hidden layer with the sign activation function containing all functions:*

$$f : \{-1, +1\}^n \to \{-1, +1\}.$$

Proof. Consider the neural network where the layers have size:

$$|V_0| = n + 1 \qquad |V_1| = 2^n + 1 \qquad |V_2| = 1,$$

all edges between adjacent layers, and activation function $\sigma(z) = \text{sign}(z) = \begin{cases} +1 & z \geq 0 \\ -1 & z < 0. \end{cases}$

- Let $\mathbf{x} \in \{-1, +1\}^n$. Given another $\mathbf{z} \in \{-1, +1\}^n$, notice that:

$$\begin{aligned} \mathbf{z}^\top \mathbf{x} &= n \qquad \text{if } \mathbf{z} = x \\ \mathbf{z}^\top \mathbf{x} &< n \qquad \text{if } \mathbf{z} \neq x. \end{aligned}$$

  Therefore, we can implement the indicator using a one-layer neural net:

$$\mathbb{1}\{\mathbf{z} = \mathbf{x}\} \equiv \sigma\left(\mathbf{z}^\top \mathbf{x} - n + 1\right) = \begin{cases} +1 & \mathbf{z} = \mathbf{x} \\ -1 & \mathbf{z} \neq \mathbf{x}. \end{cases}.$$

  Set weights of the first layer so that each node in the hidden layer corresponds to a unique $\mathbf{x}$, turning on if and only if the input is $\mathbf{x}$.

- Any boolean function f can be written as a *disjunction*. Let $\mathcal{S} = \{\mathbf{x} : f(\mathbf{x}) = +1\}$. Then:

$$\bigvee_{\mathbf{x} \in \mathcal{S}} \mathbb{1}\{\mathbf{z} = \mathbf{x}\} \equiv \sigma\left(\sum_{\mathbf{x} \in \mathcal{S}} \mathbb{1}\{\mathbf{z} = \mathbf{x}\} + |\mathcal{S}|\right) = \begin{cases} +1 & \mathbf{z} \in \mathcal{S} \\ -1 & \mathbf{z} \notin \mathcal{S}. \end{cases}$$

  We can set weights in the second layer so that it computes disjunction over $\mathcal{S}$.

□

Theorem 1. *Let $\mathcal{H}$ be a family of neural nets with architecture $G = (V, E)$ with the sign activation. If $\mathcal{H}$ contains all functions $\{-1, +1\}^n \to \{-1, +1\}$, then $|V|$ must be exponential in n.*

Proof. If $\mathcal{H}$ contains all functions $\{-1, +1\}^n \to \{-1, +1\}$, then the VC dimension of $\mathcal{H}$ must be at least 2^n. But, the VC dimension is bounded by $O(|E| \log |E|) \leq O(|V|^2 \log |V|) \leq O(|V|^3)$. Thus:

$$|V| = \Omega(2^{n/3}).$$

□

While not all Boolean functions can be implemented in a small neural network, the following shows that those that can be computed efficiently under the Turing model of computation can also be efficiently computed by a neural net:

Theorem 2. *Let $T : \mathbb{N} \to \mathbb{N}$. Let $\mathcal{F}_n$ be the set of Boolean functions $\{-1, +1\}^n \to \{-1, +1\}$ that can be computed in $T(n)$ time. There exists constants $a, b > 0$ such that for any n, there is a family of neural nets with architecture $G_n = (V_n, E_n)$ that contains $\mathcal{F}_n$ where:*

$$|E_n| \leq aT(n)^2 + b.$$

Proof ideas. We'll just give the main ideas here:

- Make use of the correspondence between computational complexity and circuit complexity.
- Implement *conjunction*, *disjunction*, *negation* in neurons. We already saw disjunction above. *Exercise:* implement conjunction and negation.

□

3.2 Universal approximation

This section drawn from Hornik et al. (1989) and an unpublished lecture by Sanjoy Dasgupta.

Recall that the **Weierstrass approximation theorem** states that any continuous $f : [a, b] \to \mathbb{R}$ can be approximated arbitrarily well by a polynomial. One proof technique is to convolve f with a sufficiently small Gaussian kernel (so as to not change f too much), then consider the Taylor expansion of the convolution (Taylor expansion exists because convolution is smoothed).

The **Stone-Weierstrass theorem** generalizes this result from the polynomial ring to any algebra of real continuous functions. Before describing the approximation theorem, we'll describe the setting:

- Let $K \subset \mathbb{R}^d$ be compact. Let $\mathcal{F}$ be a class of functions $f : K \to \mathbb{R}$. The **L_∞-distance** is:
$$\rho_K(f, g) = \sup_{x \in K} |f(x) - g(x)|.$$
- Let $C(K; \mathbb{R})$ be the **family of continuous functions** $f : K \to \mathbb{R}$.
- We say that $\mathcal{F}$ is **dense** in $C(K; \mathbb{R})$ if for all $g : K \to \mathbb{R}$ continuous and $\varepsilon > 0$, there exists $f \in \mathcal{F}$ such that:
$$\rho_K(f, g) < \varepsilon.$$

Definition 1 (Algebra of functions)**.** *A family $\mathcal{F} \subset C(K; \mathbb{R})$ of functions $f : K \to \mathbb{R}$ is an **algebra** if it is closed under pointwise addition and multiplication, and scalar multiplication.*

Definition 2 (Separating points)**.** *We say that $\mathcal{F}$ **separates points** in K if for all distinct $x, y \in K$, there exists $f \in \mathcal{F}$ such that $f(x) \neq f(y)$.*

Definition 3 (Vanishing nowhere)**.** *We say that $\mathcal{F}$ **vanishes nowhere** in K if for all $x \in K$, there exists $f \in \mathcal{F}$ such that $f(x) \neq 0$.*

Theorem 3 (Stone-Weierstrass theorem)**.** *Let $\mathcal{F}$ be an algebra of real continuous functions on a compact set $K \subset \mathbb{R}^d$. If $\mathcal{F}$ separates points and vanishes nowhere in K, then $\mathcal{F}$ is dense in $C(K; \mathbb{R})$.*

Definition 4 (Squashing function)**.** *A nondecreasing map $\sigma : \mathbb{R} \to \mathbb{R}$ is a **squashing function** if:*
$$\lim_{z \to -\infty} \sigma(z) = 0 \quad \text{and} \quad \lim_{z \to \infty} \sigma(z) = 1.$$

Theorem 4 (Universal approximation theorem, Hornik et al. (1989))**.** *Let $\mathcal{F}$ be the collection of all neural networks $f : K \to \mathbb{R}$ whose architecture contains a single hidden layer and with activation σ. If σ is a squashing function, then $\mathcal{F}$ is dense in $C(K; \mathbb{R})$.*

- The proof of this is beyond the scope of the course.
- **Main takeaway.** The Stone-Weierstrass theorem can be read this way:
  - Suppose we have access to a collection of computational primitives (with the separating points and nowhere vanishing properites).
  - If we can combine these together (multiplication, addition, scaling, apply non-linearity), then we can compute any function with arbitrary precision.
  - The complexity of performing such computation may be extremely large.

3.3 Geometric intuition

Consider a neural net whose architecture has a single hidden layer and whose activation is the sign function. Each node in the hidden layer computes a half-space:

$$h(\mathbf{x}) = \sigma(W\mathbf{x}).$$

This discretizes the space into regions that are intersection of half-spaces, where the 'signature' of each region is given by a sequence of ± 1, depending on which side of the hyperplane it falls on.

4 Training neural nets

This section drawn from Sanjoy Dasgupta's CSE 250B lecture on neural nets.

While the previous section tells us that we can define a family of neural nets $\mathcal{F} = \{f_\theta : \theta \in \Theta\}$ that can express many functions. The hope is that one of the neural nets computes what we want. But, how do we find which one? By fitting it to data.

Consider the **classification problem**: learn some classifier $\Phi : \mathbb{R}^n \to \{1, \ldots, k\}$.

- Constructing the classifier from a neural network $f : \mathbb{R}^n \to \mathbb{R}^k$:

$$\Phi(\mathbf{x}) = \arg\max_i f(\mathbf{x})_i.$$

- Given a dataset $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$, we ideally find $f_\theta \in \mathcal{F}$ making the fewest mistakes:

$$f_{\theta^*} = \arg\min_{\theta \in \Theta} \sum_{i=1}^{N} \mathbb{1}\{y_i \neq \Phi_\theta(\mathbf{x}_i)\}. \tag{1}$$

- The process of finding f_{θ^*} is called **training** or **fitting** the neural net.

Problem: how do we solve this *optimization problem*? Gradient descent and its variants.

- Non-smooth/non-convex objectives are hard to optimize. We usually use *surrogate losses*.

4.1 Cross-entropy loss

The **likelihood** that the data $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ was generated by the model f_θ is:

$$\text{likelihood}(\theta) := \prod_{i=1}^{n} f_\theta(\mathbf{x}_i)_{y_i},$$

where we can think of $f_\theta(\mathbf{x}_i)$ as a probability vector over the k classes. That is, $f_\theta(\mathbf{x}_i)_{y_i}$ is the probability we see the data point $\mathbf{x}_i$ is labeled y_i:

$$\Pr_\theta(y_i | \mathbf{x}_i) = f_\theta(\mathbf{x}_i)_{y_i}.$$

Intuitively, we would like to fine the **maximum likelihood estimator** f_θ, which is the model in $\mathcal{F}$ that most likely generated the data. Equivalently, we can minimize the **cross-entropy loss**:

$$\mathcal{L}(\theta) = -\sum_{i=1}^{N} \ln f_\theta(\mathbf{x}_i)_{y_i}.$$

In comparison to Equation (1), this is a *differentiable loss*, which allows us to apply gradient descent.

4.2 Gradient descent, stochastic gradients, and mini-batches

We've now formulated a *learning problem* as the *optimization problem*:

$$\arg\min_{\theta \in \Theta} \mathcal{L}(\theta),$$

recalling that $\theta = \big(W^(1), \ldots, W^{(\ell)}, W^{(o)}\big)$ are the weights/parameters of the neural net.

- Ideally, make incremental improvements to the model θ so that each iteration improves:

  $$\mathcal{L}(\theta_{t+1}) \leq \mathcal{L}(\theta_t).$$

- The negative gradient points in the direction of *steepest descent*. In gradient descent:

  $$\theta_{t+1} \leftarrow \theta_t - \eta \nabla \mathcal{L}(\theta_t),$$

  where η is a **learning rate**.

4.3 Backpropagation

Quick review of multivariate calculus

- The **derivative** or **gradient** of a smooth map $f : \mathbb{R}^m \to \mathbb{R}^k$ at $\mathbf{w}$ corresponds to the *best local linear approximation*. In particular if:

  $$f(\mathbf{z}) = c + L(\mathbf{z} - \mathbf{w}) + \text{remainder}(\mathbf{z}),$$

  where $L \in \mathbb{R}^{k \times m}$ and the remainder goes to zero quickly:

  $$\lim_{\|\mathbf{z}\| \to 0} \frac{\|\text{remainder}(\mathbf{z})\|}{\|\mathbf{z}\|} = 0,$$

  then $L \equiv \dfrac{df(\mathbf{w})}{d\mathbf{w}}$.

- The **chain rule**: if $f : \mathbb{R}^m \to \mathbb{R}^k$ and $g : \mathbb{R}^n \to \mathbb{R}^m$ are smooth functions, then the map:

  $$\mathbf{w} \overset{g}{\mapsto} \mathbf{v} \overset{f}{\mapsto} \mathbf{z}$$

  has derivative:

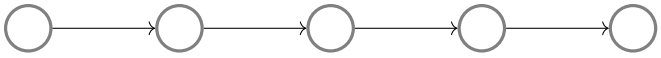
  $$\frac{d\mathbf{z}}{d\mathbf{v}} \frac{d\mathbf{v}}{d\mathbf{w}}$$

  where $\mathbf{v} = g(\mathbf{w})$ and $\mathbf{z} = f(\mathbf{v})$, $\dfrac{d\mathbf{z}}{d\mathbf{v}} \in \mathbb{R}^{k \times m}$ and $\dfrac{d\mathbf{v}}{d\mathbf{w}} \in \mathbb{R}^{m \times n}$.

Gradient of loss of neural net Consider the cross-entropy loss of θ for a single data point $(\mathbf{x}, y)$:

$$\mathcal{L}(\theta; \mathbf{x}, y) = -\ln \left(\frac{\exp\big(W_y^{(o)} \sigma\big(W^{(\ell)} \sigma(\cdots \sigma(W^{(1)}\mathbf{x}) \cdots)\big)\big)}{\sum_{i \in k]} \exp\big(W_i^{(o)} \sigma\big(W^{(\ell)} \sigma(\cdots \sigma(W^{(1)}\mathbf{x}) \cdots)\big)\big)} \right)$$

Computation of gradient Suppose we have a neural net with one node per layer:



$$\mathbf{x} = \mathbf{h}^{(0)} \qquad \mathbf{h}^{(1)} \qquad \mathbf{h}^{(2)} \qquad \mathbf{h}^{(3)} \qquad \mathbf{h}^{(4)} = \mathbf{y}$$

- Each layer can be computed: $\mathbf{h}^{(j)} = \sigma\big(W^{(j)}\mathbf{h}^{(j-1)}\big)$
- The gradient of the loss $\mathcal{L}$ with respect to $W^{(j)}$ is:

$$\frac{d\mathcal{L}}{dW^{(j)}} = \frac{d\mathcal{L}}{d\mathbf{h}^{(j)}}\frac{d\mathbf{h}^{(j)}}{dW^{(j)}} = \frac{d\mathcal{L}}{d\mathbf{h}^{(j)}} \cdot \underbrace{\sigma'\big(W^{(j)}\mathbf{h}^{(j-1)}\big)\mathbf{h}^{(j-1)}}_{d\mathbf{h}^{(j)}/dW^{(j)}}.$$

  - We just need to be able to compute $d\mathcal{L}/d\mathbf{h}^{(j)}$. Using $\mathbf{h}^{(j+1)} = \sigma\big(W^{(j+1)}\mathbf{h}^{(j)}\big)$:

$$\frac{d\mathcal{L}}{d\mathbf{h}^{(j)}} = \frac{d\mathcal{L}}{d\mathbf{h}^{(j+1)}}\frac{d\mathbf{h}^{(j+1)}}{d\mathbf{h}^{(j)}} = \frac{d\mathcal{L}}{d\mathbf{h}^{(j+1)}} \cdot \sigma'\big(W^{(j+1)}\mathbf{h}^{(j)}\big)W^{(j+1)}.$$

- **Backpropagation algorithm** (Rumelhart et al., 1986):
  - In the **forward pass**, save all the computation of $\mathbf{h}^{(j)}$ in order $j = 1, \ldots, \ell$
  - In the **backward pass**, compute $d\mathcal{L}/d\mathbf{h}^{(j)}$ in order $j = \ell, \ldots, 1$

5 Generalization

5.1 Statistical learning framework

A **learner** would like to answer a question about the world.

1. The learner selects a **model**—a family of possible explanations/hypotheses.
2. The learner collects **data** from the world.
3. The learner then **fits** the model to the data.

The model's ability to **generalize** is how well it accounts for out-of-sample data.

The standard approach to learning

1. Select a **model** $\mathcal{F}$.
2. Define the **risk** of a hypothesis $f \in \mathcal{F}$ as:

$$R(f) = \text{a measure of how poorly } f \text{ explains the world.}$$

   - The goal is to find the *best-in-class explanation* $f^* = \arg\min_{f \in \mathcal{F}} R(f)$.
   - The *model bias* measures the risk $R(f^*)$ of the best explanation.
   - We generally cannot compute the risk directly, but we can estimate it.

3. Construct a **risk estimation** procedure that finds an estimate $\hat{R}(f)$ of $R(f)$.
4. **Minimize** the risk estimator:
$$\hat{f} := \arg\min_{f \in \mathcal{F}} \hat{R}(f).$$

We can decompose the risk $R(f)$ as:

$$R(f) = \underbrace{\hat{R}(f) - \hat{R}(f^*)}_{\text{estimated gap}} + \underbrace{\big(R(f) - \hat{R}(f)\big)}_{\text{estimation error for } f} + \underbrace{\big(\hat{R}(f^*) - R(f^*)\big)}_{\text{estimation error for } f^*} + \underbrace{R(f^*)}_{\text{model bias}}$$

- For the empirical risk minimizer $\hat{f}$, the *estimated gap* term is non-positive, so:
$$R(\hat{f}) \leq \text{estimation error terms} + \text{model bias term}.$$

Generalization theory tends to give us bounds on the estimation error term so that:

$$R(\hat{f}) \leq \sqrt{\frac{\text{capacity of model}}{\text{amount of training data}}} + \text{model bias}.$$

- The **capacity** of $\mathcal{F}$ measures how many worlds $\mathcal{F}$ can explain.
- Generally, the bias of a model increases as its capacity shrinks.
  - This leads to the **bias-variance tradeoff**.

Intuition: how the bias of $\mathcal{F}$ relate to the capacity of $\mathcal{F}$.

- **Small capacity:** if $\mathcal{F}$ cannot explain many worlds, it may poorly explain the one in which the learner lives. This leads to a large bias term.
- **Large capacity:** if many (very different) explanations account for what the learner sees, how to pick among these explanations? This leads to a large variance term.

Theorem 5 (Shalev-Shwartz and Ben-David (2014))**.** *The VC dimension of a neural network with K weight parameters is:*
$$O\big(K \log K\big)$$

5.2 Interpolating regime

The VC dimension bounds (and other such capacity bounds) fail to explain why neural nets seem to have empirically good generalization: there are way more parameters than data.

6 Research questions

- Generalization: trajectory length, algorithmic stability, overparametrization/double descent
- neural tangent kernels (NTKs; random initialization close to optimal), lottery ticket
- Other architectures (e.g. transformers, graph neural nets, recurrent neural nets, etc.)
- Constructing and training neural nets (many 'tricks' and a lot of engineering)
- Implementations (e.g. pytorch)

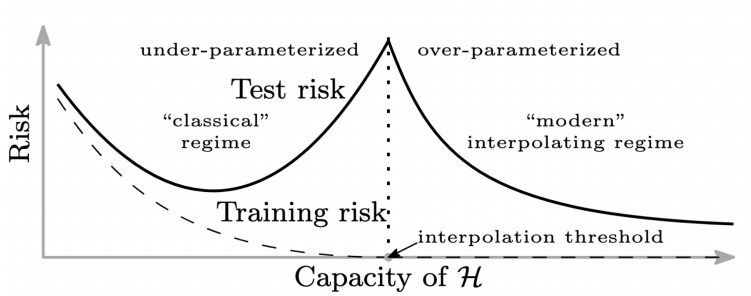


Figure 5: Double descent curve (Belkin et al., 2019)

References

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

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

Jaime Sevilla, Lennart Heim, Anson Ho, Tamay Besiroglu, Marius Hobbhahn, and Pablo Villalobos. Compute trends across three eras of machine learning. In *2022 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2022.

Shai Shalev-Shwartz and Shai Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge University Press, 2014.

Pablo Villalobos and Anson Ho. Trends in training dataset sizes, 2022. URL https://epochai.org/blog/trends-in-training-dataset-sizes. Accessed: 2023-7-24.

Pablo Villalobos, Jaime Sevilla, Tamay Besiroglu, Lennart Heim, Anson Ho, and Marius Hobbhahn. Machine learning model sizes and the parameter gap, 2022.