Chapter 2: Shallow Neural Networks to Multilayer Perceptrons

Mathematical Foundations of Deep Neural Networks

Spring 2024

Department of Mathematical Sciences

Ernest K. Ryu

Seoul National University

Supervised learning setup

We have data $X_1, \dots, X_N \in \mathcal{X}$ and corresponding labels $Y_1, \dots, Y_N \in \mathcal{Y}$.

Example) X_i is the ith email and $Y_i \in \{-1, +1\}$ denotes whether X_i is a spam email.

Example) X_i is the ith image and $Y_i \in \{0, \dots, 9\}$ denotes handwritten digit.

Assume there is a true unknown function

$$f_\star : \mathcal{X} \to \mathcal{Y}$$

mapping data to its label. In particular, $Y_i = f_\star(X_i)$ for $i = 1, \dots, N$.

The goal of supervised learning is to use $X_1, \dots, X_N$ and $Y_1, \dots, Y_N$ to find $f \approx f_\star$.

Formulating the right objective

The goal of "finding $f \approx f_\star$" must be further quantified.

Assume a loss function such that $\ell(y_1, y_2) = 0$ if $y_1 = y_2$ and $\ell(y_1, y_2) > 0$ if $y_1 \neq y_2$.

Attempt 1)

$$\underset{f}{\text{minimize}} \quad \sup_{x \in \mathcal{X}} \ell(f(x), f_\star(x))$$

Problems:

- There is a trivial solution $f = f_\star$.
- Minimization over all functions f is in general algorithmically intractable[1]. How would one represent a f on a computer?

[1] The space of all functions is an infinite-dimensional space. We want our optimization variable to be a finite-dimensional vector.

Formulating the right objective

Attempt 2) Restrict search to a class of parametrized functions $f_\theta(x)$ where $\theta \in \Theta \subseteq \mathbb{R}^p$, i.e., only consider $f \in \{f_\theta \mid \theta \in \Theta\}$ where $\Theta \subseteq \mathbb{R}^p$. Then solve

$$\underset{f \in \{f_\theta \mid \theta \in \Theta\}}{\text{minimize}} \quad \sup_{x \in \mathcal{X}} \ell(f(x), f_\star(x))$$

which is equivalent to

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \sup_{x \in \mathcal{X}} \ell(f_\theta(x), f_\star(x))$$

Problems:

- The supremum $\sup_{x \in \mathcal{X}}$ is computationally inconvenient to deal with.
- Objective is too pessimistic. We do not need to do well all the time, we just need to do well on average.

Formulating the right objective

Attempt 3) Take a finite sample* $X_1, \dots, X_N \in \mathcal{X}$ and corresponding labels $Y_1, \dots, Y_N \in \mathcal{Y}$. Then solve

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(X_i), f_\star(X_i))$$

which is equivalent to

$$\underset{\theta \in \Theta}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^{N} \ell(f_\theta(X_i), Y_i)$$

This is the standard form of the optimization problem (except regularizers) we consider in the supervised learning. We will talk about regularizers later.

*It is common to assume $X_1, \dots, X_N$ are IID samples from a certain probability distribution. Instead of this statistical view, we take the curve-fitting view.

Aside: Minimum vs. infimum

We clarify terminology.

- “Minimize”: Used to specify an optimization problem.
- “Minimizer”: A solution to a minimization problem.
- “Minimum”: Used to specify the smallest objective value and asserts a minimizer exists.
- “Infimum”: Used to specify the limiting smallest objective value, but a minimizer may not exist.

Analogous definitions with “maximize”, “maximizer”, “maximum”, and “supremum”

Training is optimization

In machine learning, the anthropomorphized word “training” refers to solving an optimization problem such as

$$\underset{\theta\in\Theta}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N}\ell(f_\theta(X_i),Y_i)$$

In most cases, SGD or variants of SGD are used.

We call f_θ the machine learning *model* or the *neural network*.

Least-squares regression

In LS, $\mathcal{X} = \mathbb{R}^p,\ \mathcal{Y} = \mathbb{R}$, $\Theta = \mathbb{R}^p$, $f_\theta(x) = x^\top \theta$, and $\ell(y_1, y_2) = \frac{1}{2}(y_1 - y_2)^2$.

So we solve

$$\underset{\theta \in \mathbb{R}^p}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}(f_\theta(X_i) - Y_i)^2 = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left(X_i^\top \theta - Y_i\right)^2 = \frac{1}{2N}\|X\theta - Y\|^2$$

where $X = \begin{bmatrix} X_1^\top \\ \vdots \\ X_N^\top \end{bmatrix}$ and $Y = \begin{bmatrix} Y_1 \\ \vdots \\ Y_N \end{bmatrix}$.

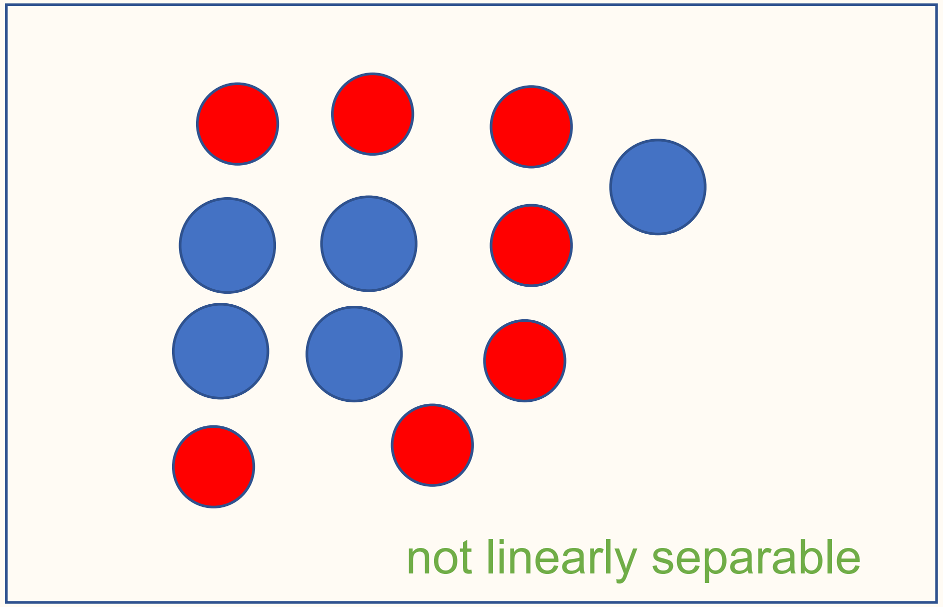
The model $f_\theta(x) = x^\top \theta$ is a *shallow* neural network. (The terminology will makes sense when contrasted with *deep* neural networks.)

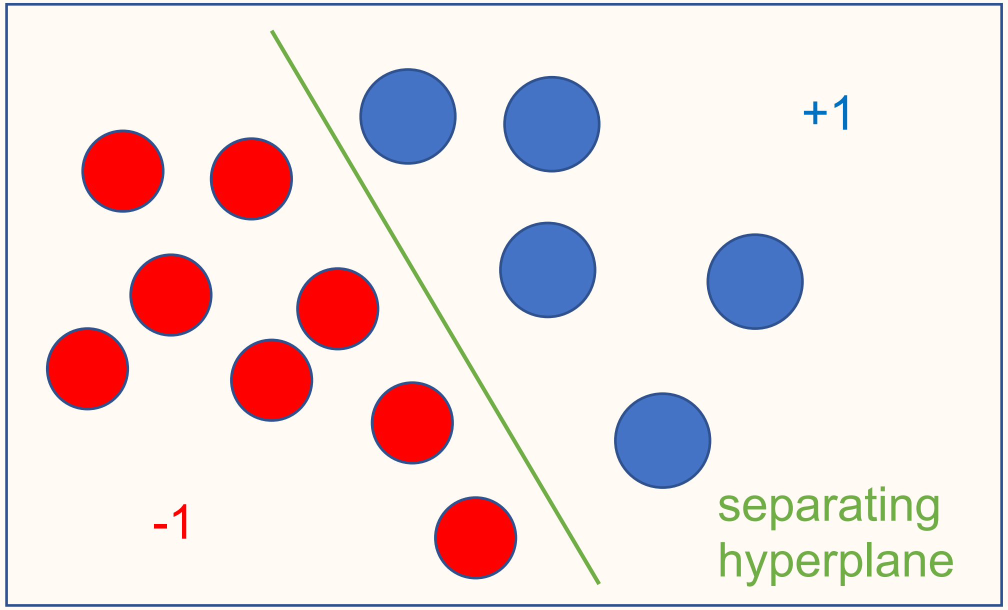
Binary classification and linear separability

In binary classification, we have $\mathcal{X} = \mathbb{R}^p$ and $\mathcal{Y} = \{-1, +1\}$.

The data is *linearly separable* if there is a hyperplane defined by $(a_{\text{true}}, b_{\text{true}})$ such that

$$y = \begin{cases} 1 & \text{if } a_{\text{true}}^{\top} x + b_{\text{true}} > 0 \\ -1 & \text{otherwise.} \end{cases}$$





Linear classification

Consider linear (affine) models

$$f_{a,b}(x) = \begin{cases} +1 & \text{if } a^\top x + b > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Consider the loss function

$$\ell(y_1, y_2) = \frac{1}{2}|1 - y_1 y_2| = \begin{cases} 0 & \text{if } y_1 = y_2 \\ 1 & \text{if } y_1 \neq y_2 \end{cases}$$

The optimization problem

$$\underset{a \in \mathbb{R}^p,\, b \in \mathbb{R}}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^{N} \ell(f_{a,b}(X_i), Y_i)$$

has a solution with optimal value 0 when the data is linearly separable.

Problem: Optimization problem is discontinuous and thus cannot be solved with SGD.

Relaxing into continuous formulation

Even if the underlying function or phenomenon to approximate is discontinuous, the model needs to be continuous* in its parameters. The loss function also needs to be continuous. (The prediction need not be continuous.)

We consider a *relaxation*, is a continuous proxy of the discontinuous thing. Specifically, consider

$$f_{a,b}(x) = a^\top x + b$$

Once trained, $f_{a,b}(x) > 0$ means the neural network is predicting $y = +1$ to be “more likely”, and $f_{a,b}(x) < 0$ means the neural network is predicting $y = -1$ to be “more likely”.

Therefore, we train the model to satisfy

$$Y_i f_{a,b}(X_i) > 0 \quad \text{for } i = 1, \dots, N.$$

*There are advanced deep learning techniques for learning discontinuous models, but we will not cover them in this course.

Relaxing into continuous formulation

Problem with strict inequality $Y_i f_{a,b}(X_i) > 0$:

- Strict inequality has numerical problems with round-off error.
- The magnitude $\left|f_{a,b}(x)\right|$ can be viewed as the confidence* of the prediction, but having a small positive value for $Y_i f_{a,b}(X_i)$ indicates small confidence of the neural network.

We modify our model's desired goal to be $Y_i f_{a,b}(X_i) \geq 1$.

*This "confidence" is related to the classifier's margin in the standard SVM derivation. The standard derivation is more principled, but it does not extend to the general setup of deep neural networks. We instead consider the presented heuristic argument as it is more general.

Support vector machine (SVM)

To train the neural network to satisfy

$$0 \geq 1 - Y_i f_{a,b}(X_i) \text{ for } i = 1, \dots, N.$$

we minimize the excess positive component of the RHS

$$\underset{a \in \mathbb{R}^p, b \in \mathbb{R}}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^{N} \max\{0, 1 - Y_i f_{a,b}(X_i)\}$$

which is equivalent to

$$\underset{a \in \mathbb{R}^p, b \in \mathbb{R}}{\text{minimize}} \quad \frac{1}{N} \sum_{i=1}^{N} \max\{0, 1 - Y_i(a^\top X_i + b)\}$$

If the optimal value is 0, then the data is linearly separable.

Support vector machine (SVM)

This formulation is called the *support vector machine* (SVM)[*]

$$\underset{a\in\mathbb{R}^p,b\in\mathbb{R}}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N}\max\{0,1-Y_i(a^\top X_i+b)\}$$

It is also common to add a regularizer

$$\underset{a\in\mathbb{R}^p,b\in\mathbb{R}}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N}\max\{0,1-Y_i(a^\top X_i+b)\}+\frac{\lambda}{2}\|a\|^2$$

We will talk about regularizers later.

[*] Cortes and Vapnik, Support-vector networks, *Machine Learning*, 1995.

Prediction with SVM

Once the SVM is trained, make predictions with

$$\operatorname{sign}\left(f_{a,b}(x)\right) = \operatorname{sign}(a^\top x + b)$$

when $f_{a,b}(x) = 0$, we assign $\operatorname{sign}\left(f_{a,b}(x)\right)$ arbitrarily.

Note that the prediction is discontinuous, but predictions are in $\{-1, +1\}$ so it must be discontinuous.

If $\sum_{i=1}^{N} \max\{0, 1 - Y_i f_{a,b}(X_i)\} = 0$, then $\operatorname{sign}\left(f_{a,b}(X_i)\right) = Y_i$ for $i = 1, \dots, N$, i.e., the neural network predicts the known labels perfectly. (Make sure you understand this.) Of course, it is a priori not clear how accurate the prediction will be for new unseen data.

SVM is a relaxation

Directly minimizing the prediction error on the data is

$$\underset{a\in\mathbb{R}^p,\, b\in\mathbb{R}}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N}\frac{1}{2}\left|1 - Y_i \operatorname{sign}\left(f_{a,b}(X_i)\right)\right|$$

The optimization we instead solve is

$$\underset{a\in\mathbb{R}^p, b\in\mathbb{R}}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N}\max\{0, 1 - Y_i f_{a,b}(X_i)\}$$

Let the optimal values be $p_1^\star$ and $p_2^\star$. Again, SVM is of as a relaxation of the first. The two are not equivalent. (An equivalent formulation is not referred to as a relaxation.)

- It is possible to show* that $p_1^\star = 0$ if and only if $p_2^\star = 0$.
- If $p_1^\star > 0$ and $p_2^\star > 0$, a solution to the first problem need not correspond to a solution to the second problem, i.e., there solutions may be completely different.

*The proof relies on the fact that $Y_i f_{a,b}(X_i)$ is linear in (a, b).

Relaxed supervised learning setup

We relax the supervised learning setup to predict probabilities, rather than make point predictions[*]. So, labels are generated based on data, *perhaps randomly*. Consider data $X_1, \ldots, X_N \in \mathcal{X}$ and labels $Y_1, \ldots, Y_N \in \mathcal{Y}$. Assume there exists a function

$$f_\star : \mathcal{X} \to \mathcal{P}(\mathcal{Y})$$

where $\mathcal{P}(\mathcal{Y})$ denotes the set of probability distributions on $\mathcal{Y}$.

Assume the generation of Y_i given X_i is independent of Y_j and X_j for $j \neq i$.

Example) $f(X) = \begin{bmatrix} 0.8 \\ 0.2 \end{bmatrix}$ in dog vs. cat classifier.

Example) An email saying "Buy this thing at our store!" may be spam to some people, but it may not be spam to others.

The relaxed SL setup is more general and further realistic.

[*] By point prediction, I mean predicting a single label, rather than a distribution of labels.

KL-divergence

Let $p, q \in \mathbb{R}^n$ represent probability masses, i.e., $p_i \geq 0 \text{ for } i = 1, \dots, n$ and $\sum_{i=1}^{n} p_i = 1$ and the same for q. The *Kullback-Leibler-divergence* (KL-divergence) from q to p is

$$D_{\mathrm{KL}}(p \| q) = \sum_{i=1}^{n} p_i \log\left(\frac{p_i}{q_i}\right) = \underbrace{-\sum_{i=1}^{n} p_i \log(q_i)}_{\substack{= H(p,q) \\ \text{cross entropy of } q \\ \text{relative to } p}} + \underbrace{\sum_{i=1}^{n} p_i \log(p_i)}_{\substack{= -H(p) \\ \text{entropy of } p}}$$

Properties:

- Not symmetric, i.e., $D_{\mathrm{KL}}(p \| q) \neq D_{\mathrm{KL}}(q \| p)$.
- $D_{\mathrm{KL}}(p \| q) > 0$ if $p \neq q$ and $D_{\mathrm{KL}}(p \| q) = 0$ if $p = q$.
- $D_{\mathrm{KL}}(p \| q) = \infty$ is possible. (Further detail on the next slide.)

Often used as a “distance” between p and q despite not being a metric.

KL-divergence

$$D_{\mathrm{KL}}(p\|q)=\sum_{i=1}^{n} p_i \log\left(\frac{p_i}{q_i}\right)$$

Clarification: Use the convention

- $0 \log\left(\frac{0}{0}\right) = 0$ (when $p_i = q_i = 0$)
- $0 \log\left(\frac{0}{q_i}\right) = 0$ if $q_i > 0$
- $p_i \log\left(\frac{p_i}{0}\right) = \infty$ if $p_i > 0$

Probabilistic interpretation:

$$D_{\mathrm{KL}}(p\|q)=\mathbb{E}_I\left[\log\left(\frac{p_I}{q_I}\right)\right]$$

with the random variable I such that $\mathbb{P}(I = i) = p_i$.

Empirical distribution for binary classification

In basic binary classification, define the *empirical distribution*

$$\mathcal{P}(y) = \begin{cases} \begin{bmatrix} 1 \\ 0 \end{bmatrix} & \text{if } y = -1 \\ \begin{bmatrix} 0 \\ 1 \end{bmatrix} & \text{if } y = +1 \end{cases}$$

More generally, the empirical distribution describes the data we have seen. In this context, we have only seen one label per datapoint, so our empirical distributions are *one-hot vectors*.

(If there are multiple annotations per data point x and they don’t agree, then the empirical distribution may not be one-hot vectors. For example, given the same email, some users may flag it as spam while others consider it useful information.)

Logistic regression

Logistic regression (LR), is another model for binary classification:

1. Use the model

$$f_{a,b}(x) = \begin{bmatrix} \dfrac{1}{1+e^{a^\top x+b}} \\ \dfrac{e^{a^\top x+b}}{1+e^{a^\top x+b}} \end{bmatrix} = \begin{bmatrix} \dfrac{1}{1+e^{a^\top x+b}} \\ \dfrac{1}{1+e^{-(a^\top x+b)}} \end{bmatrix} \begin{matrix} = \mathbb{P}(y=-1) \\ = \mathbb{P}(y=+1) \end{matrix}$$

2. Minimize KL-Divergence (or cross entropy) from the model $f_{a,b}(X_i)$ output probabilities to the empirical distribution $\mathcal{P}(Y_i)$.

$$\underset{a\in\mathbb{R}^p, b\in\mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{N} D_{\mathrm{KL}}(\mathcal{P}(Y_i) \| f_{a,b}(X_i))$$

Logistic regression

Note:

$$\underset{a\in\mathbb{R}^p, b\in\mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{N} D_{\mathrm{KL}}(\mathcal{P}(Y_i)\|f_{a,b}(X_i))$$

$$\Updownarrow$$

$$\underset{a\in\mathbb{R}^p, b\in\mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{N} H(\mathcal{P}(Y_i), f_{a,b}(X_i)) + \text{(terms independent of } a, b\text{)}$$

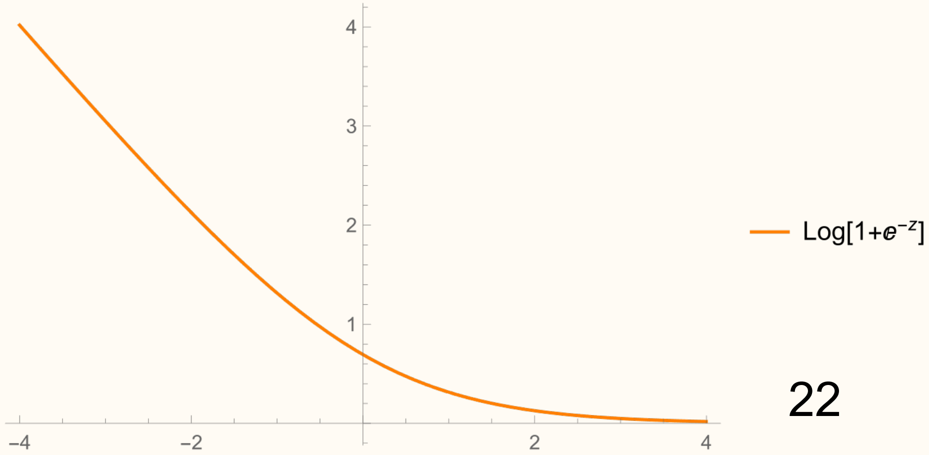
$$\Updownarrow$$

$$\underset{a\in\mathbb{R}^p, b\in\mathbb{R}}{\text{minimize}} \quad \sum_{i=1}^{N} \log\left(1+\exp\left(-Y_i(a^\top X_i + b)\right)\right)$$

$$\Updownarrow$$

$$\underset{a\in\mathbb{R}^p, b\in\mathbb{R}}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N} \ell\left(Y_i(a^\top X_i + b)\right)$$

where $\ell(z) = \log(1+e^{-z})$.



Point prediction with logistic regression

When performing point prediction with LR, $a^\top x + b > 0$ means $\mathbb{P}(y = +1) > 0.5$ and vice versa.

Once the LR is trained, make predictions with

$$\mathrm{sign}(a^\top x + b)$$

when $a^\top x + b = 0$, we assign $\mathrm{sign}(a^\top x + b)$ arbitrarily. This is the same as SVM.

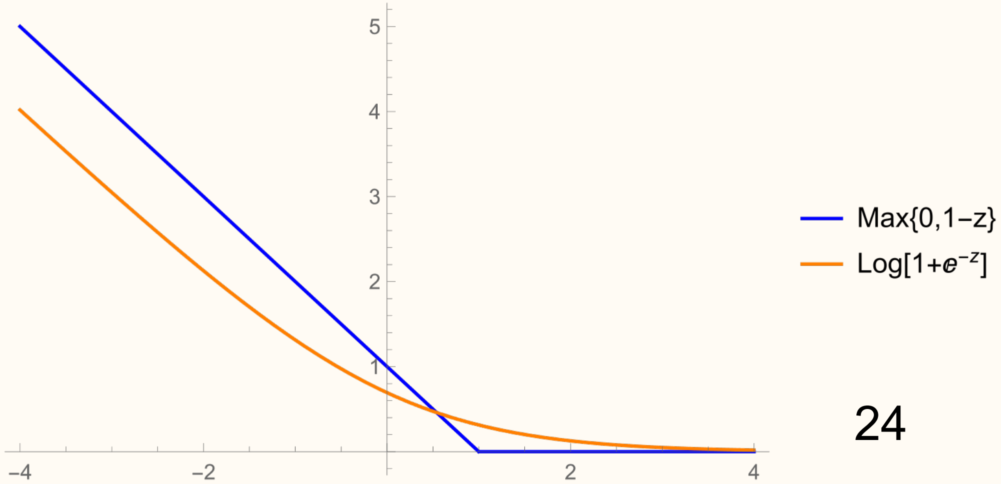
Again, it is a priori not clear how accurate the prediction will be for new unseen data.

SVM vs. LR

Both support vector machine and logistic regression can be written as

$$\underset{a\in\mathbb{R}^p, b\in\mathbb{R}}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N} \ell\big(Y_i(a^\top X_i + b)\big)$$

- SVM uses $\ell(z) = \max\{0, 1 - z\}$. Obtained from relaxing the discontinuous prediction loss.
- LR uses $\ell(z) = \log(1 + e^{-z})$. Obtained from relaxing the supervised learning setup from predicting the label to predicting the label probabilities.



SVM vs. LR

SVM and LR are both “linear” classifiers:

- Decision boundary $a^\top x + b = 0$ is linear.
- Model completely ignores information perpendicular to a.

LR naturally generalizes to multi-class classification via softmax regression. Generalizing SVM to multi-class classification is trickier and less common.

Estimation vs. Prediction

Finding $f \approx f_\star$ for unknown

$$f_\star\colon \mathcal{X} \to \mathcal{P}(\mathcal{Y})$$

is called *estimation*[*]. When we consider a parameterized model f_θ, finding θ is the estimation. However, estimation is usually not the end goal.

The end goal is *prediction*. It is to use $f_\theta \approx f_\star$ on *new data* $X_1', \dots, X_M' \in \mathcal{X}$ to find labels $Y_1', \dots, Y_M' \in \mathcal{Y}$.

[*] The word *inference* is sometimes, but not always, used as a synonym of estimation. In machine learning and statistics, the words estimation, inference, and prediction are used wildly inconsistently to the point that one must always ask for the definition to be clarified. In any case, what is most important is that you understand the distinction between the two concepts, regardless of which two of the three words are used to describe them.

Is prediction possible?

In the worst hypotheticals, prediction is impossible.

- Even though smoking is harmful for every other human being, how can we be 100% sure that this one person is not a mutant who benefits from the chemicals of a cigarette?
- Water freezes at 0°, but will the same be true tomorrow? How can we be 100% sure that the laws of physics will not suddenly change tomorrow?

Of course, prediction is possible in practice.

Theoretically, prediction requires assumptions on the distribution of X and the model of $f_{\star}$ is needed. This is in the realm of statistics of statistical learning theory.

For now, we will take the view that if we predict known labels of the training data, we can reasonably hope to do well on the new data. (We will discuss the issue of generalization and overfitting later.)

Training data vs. test data

When testing a machine learning model, it is essential that one separates the training data with the test data.

In other classical disciplines using data, one performs a statistical hypothesis test to obtain a p-value. In ML, people do not do that.

The only sure way to ensure that the model is doing well is to assess its performance on new data.

Usually, training data and test data is collected together. This ensures that they have the same statistical properties. The assumption is that this test data will be representative of the actual data one intends to use machine learning on.

Aside: Maximum likelihood estimation ≅ minimizing KL divergence

Consider the setup where you have IID discrete random variables $X_1, \dots, X_N$ that can take values $1, \dots, k$. We model the probability masses with $\mathbb{P}_\theta(X=1), \dots, \mathbb{P}_\theta(X=k)$. The maximum likelihood estimation (MLE) is obtained by solving

$$\underset{\theta}{\text{maximize}} \quad \frac{1}{N}\sum_{i=1}^{N} \log\big(\mathbb{P}_\theta(X_i)\big)$$

Next, define

$$f_\theta = \begin{bmatrix} \mathbb{P}_\theta(X=1) \\ \vdots \\ \mathbb{P}_\theta(X=k) \end{bmatrix}, \qquad \mathcal{P}(X_1, \dots, X_N) = \frac{1}{N}\begin{bmatrix} \#(X_i = 1) \\ \vdots \\ \#(X_i = k) \end{bmatrix}.$$

Then MLE is equivalent to minimizing the KL divergence from the model to the empirical distribution.

$$\begin{array}{c} \text{MLE} \\ \Updownarrow \\ \underset{\theta}{\text{minimize}} \quad H(\mathcal{P}(X_1, \dots, X_N), f_\theta) \\ \Updownarrow \\ \underset{\theta}{\text{minimize}} \quad D_{\text{KL}}(\mathcal{P}(X_1, \dots, X_N) \| f_\theta) \end{array}$$

Aside: Maximum likelihood estimation ≅ minimizing KL divergence

One can also derive LR equivalently as the MLE.

Generally, one can view the MLE as minimizing the KL divergence between the model and the empirical distribution. (For continuous random variables like the Gaussian, this requires extra work, since we haven’t defined the KL divergence for continuous random variables.)

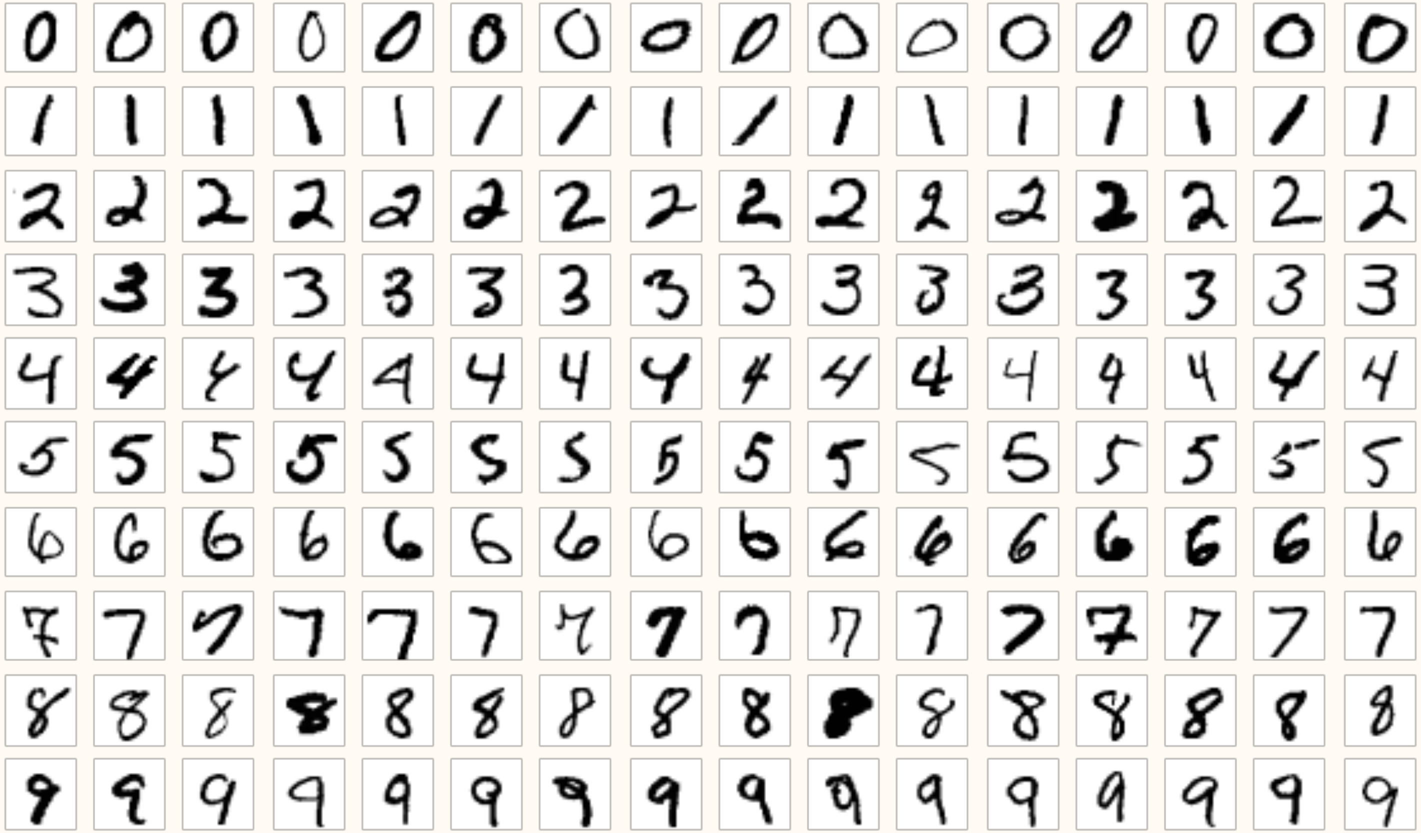
In deep learning, the distance measure need not be KL divergence.

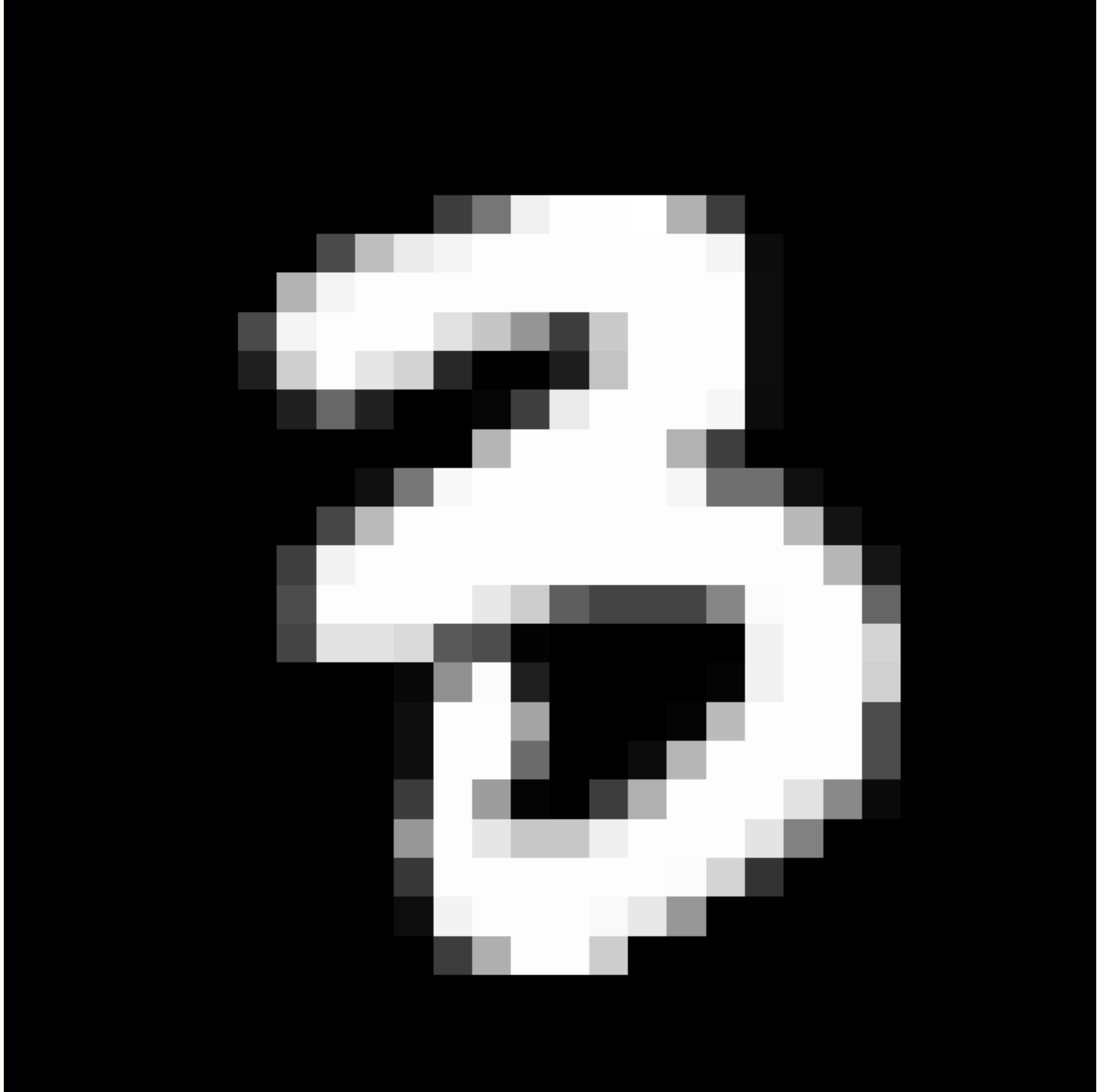
Dataset: MNIST

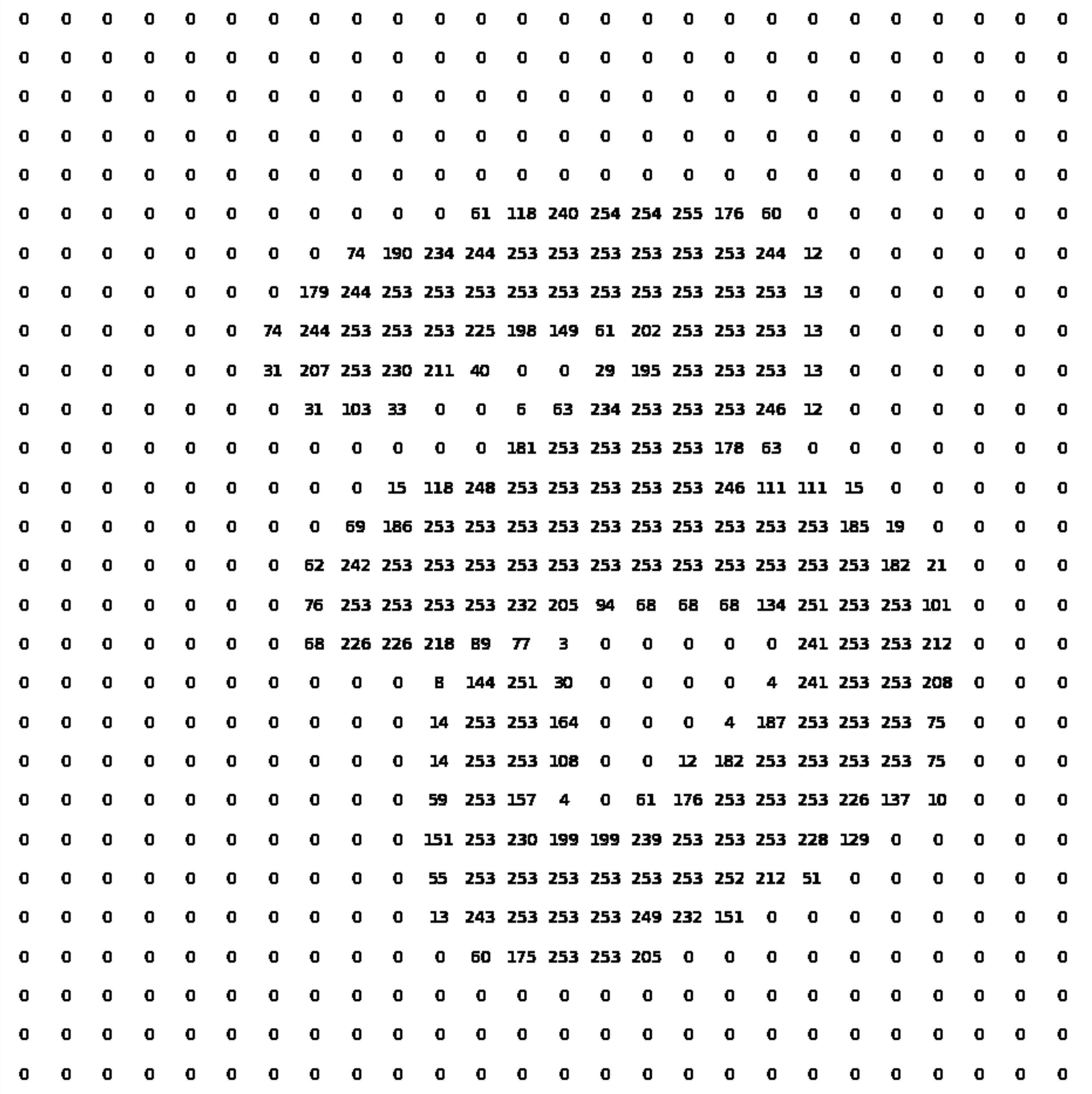
Images of hand-written digits with $28{\times}28 = 784$ pixels and integer-valued intensity between 0 and 255. Every digit has a label in $\{0,1,\dots,8,9\}$.

70,000 images (60,000 for training 10,000 testing) of 10 almost balanced classes.

One of the simplest data set used in machine learning.





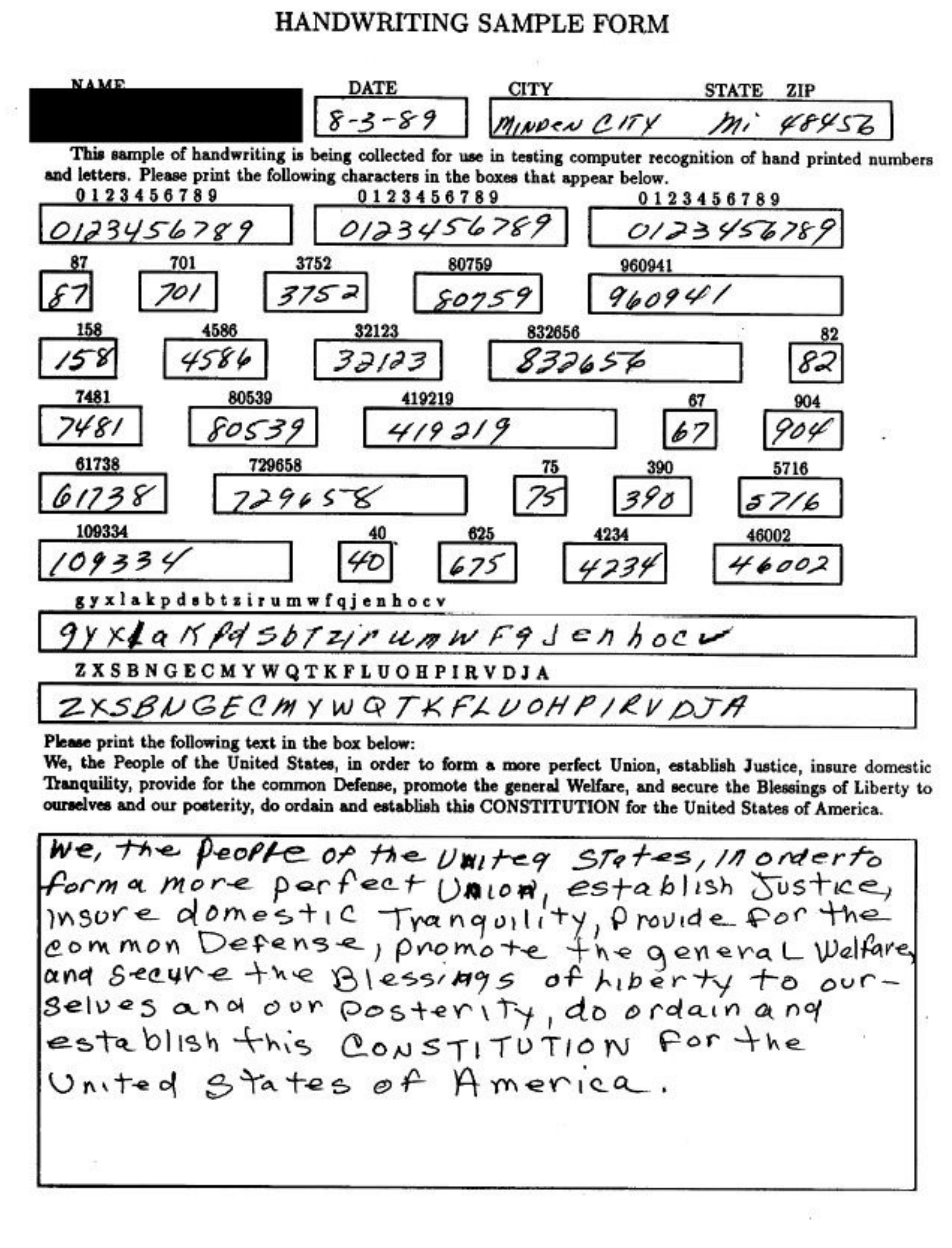


0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	61	118	240	254	254	255	176	60	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	74	190	234	244	253	253	253	253	253	253	244	12	0	0	0	0	0	0
0	0	0	0	0	0	0	179	244	253	253	253	253	253	253	253	253	253	253	13	0	0	0	0	0	0
0	0	0	0	0	0	74	244	253	253	253	225	198	149	61	202	253	253	253	13	0	0	0	0	0	0
0	0	0	0	0	0	31	207	253	230	211	40	0	0	29	195	253	253	253	13	0	0	0	0	0	0
0	0	0	0	0	0	0	31	103	33	0	0	6	63	234	253	253	253	246	12	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	181	253	253	253	253	178	63	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	15	118	248	253	253	253	253	253	246	111	111	15	0	0	0	0	0
0	0	0	0	0	0	0	0	69	186	253	253	253	253	253	253	253	253	253	253	185	19	0	0	0	0
0	0	0	0	0	0	0	62	242	253	253	253	253	253	253	253	253	253	253	253	253	182	21	0	0	0
0	0	0	0	0	0	0	76	253	253	253	253	232	205	94	68	68	68	134	251	253	253	101	0	0	0
0	0	0	0	0	0	0	68	226	226	218	89	77	3	0	0	0	0	0	241	253	253	212	0	0	0
0	0	0	0	0	0	0	0	0	0	8	144	251	30	0	0	0	0	4	241	253	253	208	0	0	0
0	0	0	0	0	0	0	0	0	0	14	253	253	164	0	0	0	4	187	253	253	253	75	0	0	0
0	0	0	0	0	0	0	0	0	0	14	253	253	108	0	0	12	182	253	253	253	253	75	0	0	0
0	0	0	0	0	0	0	0	0	0	59	253	157	4	0	61	176	253	253	253	226	137	10	0	0	0
0	0	0	0	0	0	0	0	0	0	151	253	230	199	199	239	253	253	253	228	129	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	55	253	253	253	253	253	253	252	212	51	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	13	243	253	253	253	249	232	151	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	60	175	253	253	205	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Dataset: MNIST

The USA government needed a standardized test to assess handwriting recognition software being sold to the government. So the NIST (National Institute of Standards and Technology) created the dataset in the 1990s. In 1990, NIST Special Database 1 distributed on CD-ROMs by mail. NIST SD 3 (1992) and SD 19 (1995) were improvements.

Humans were instructed to fill out handwriting sample forms.



HANDWRITING SAMPLE FORM

NAME	DATE	CITY	STATE	ZIP
[redacted]	8-3-89	MINDEN CITY	MI	48456

This sample of handwriting is being collected for use in testing computer recognition of hand printed numbers and letters. Please print the following characters in the boxes that appear below.

0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9	0 1 2 3 4 5 6 7 8 9
0123456789	0123456789	0123456789

87	701	3752	80759	960941
87	701	3752	80759	960941
158	4586	32123	832656	82
158	4586	32123	832656	82
7481	80539	419219	67	904
7481	80539	419219	67	904
61738	729658	75	390	5716
61738	729658	75	390	5716
109334	40	625	4234	46002
109334	40	675	4234	46002

g y x l a k p d s b t z i r u m w f q j e n h o c v

gyxlakpdsbtzirumwfqjenhocv

Z X S B N G E C M Y W Q T K F L U O H P I R V D J A

ZXSBNGECMYWQTKFLUOHPIRVDJA

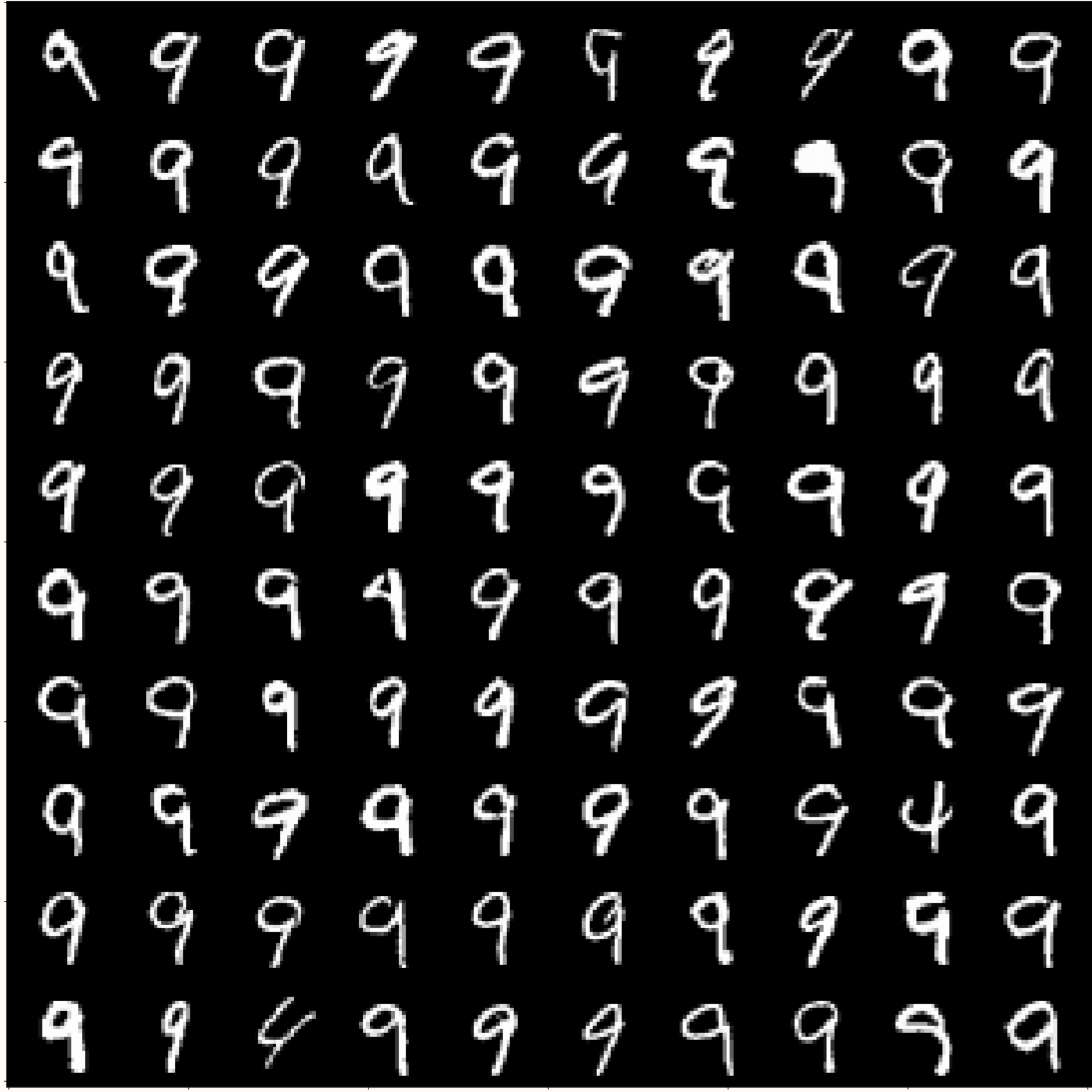
Please print the following text in the box below:
We, the People of the United States, in order to form a more perfect Union, establish Justice, insure domestic Tranquility, provide for the common Defense, promote the general Welfare, and secure the Blessings of Liberty to ourselves and our posterity, do ordain and establish this CONSTITUTION for the United States of America.

We, the People of the United States, in order to
form a more perfect Union, establish Justice,
insure domestic Tranquility, provide for the
common Defense, promote the general Welfare,
and secure the Blessings of Liberty to our-
selves and our posterity, do ordain and
establish this CONSTITUTION for the
United States of America.

Dataset: MNIST

However, humans cannot be trusted to follow instructions, so a lab technician performed “human ground truth adjudication”.

In 1998, Yann LeCun, Corinna Cortes, Christopher J. C. Burges took the NIST dataset and modified it to create the MNIST dataset.



Role of Datasets in ML Research

An often underappreciated contribution.

Good datasets play a crucial role in driving progress in ML research.

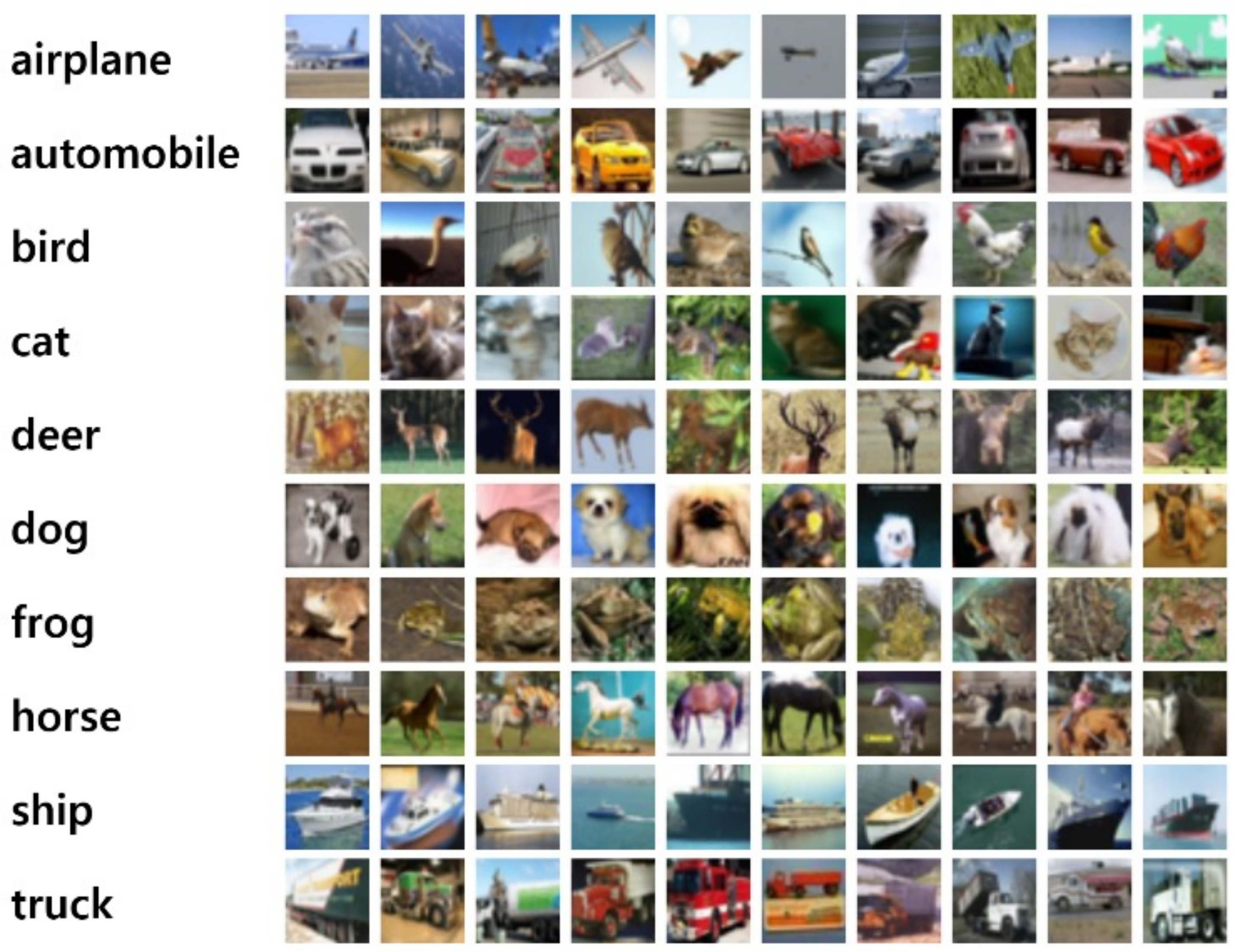
Thinking about the dataset is the essential first step of understanding the feasibility of a ML task.

Accounting for the cost of producing datasets and leveraging freely available data as much as possible (semi-supervised learning) is a recent trend in machine learning.

Dataset: CIFAR10

60,000 32×32 color images in 10 (perfectly) balanced classes.

(There is no overlap between automobiles and trucks. “Automobile” includes sedans, SUVs, things of that sort. “Truck” includes only big trucks. Neither includes pickup trucks.)



Dataset: CIFAR10

In 2008, a MIT and NYU team created the *80 million tiny images* data set by searching on Google, Flickr, and Altavista for every non-abstract English noun and downscaled the images to 32×32. The search term provided an unreliable label for the image. This dataset was not very easy to use since the classes were too numerous.

In 2009, Alex Krizhevsky published the CIFAR10, by distilling just a few classes and cleaning up the labels. Students were paid to verify the labels.

The dataset was named CIFAR-10 after the funding agency Canadian Institute For Advanced Research. There is also a CIFAR-100 with 100 classes.

Shallow learning with PyTorch

PyTorch demo

We follow the following steps

1. Load data
2. Define model
3. Miscellaneous setup
   - Instantiate model
   - Choose loss function
   - Choose optimizer
4. Train with SGD
   - Clear previously computed gradients
   - Compute forward pass
   - Compute gradient via backprop
   - SGD update
5. Evaluate trained model
6. Visualize results of trained model

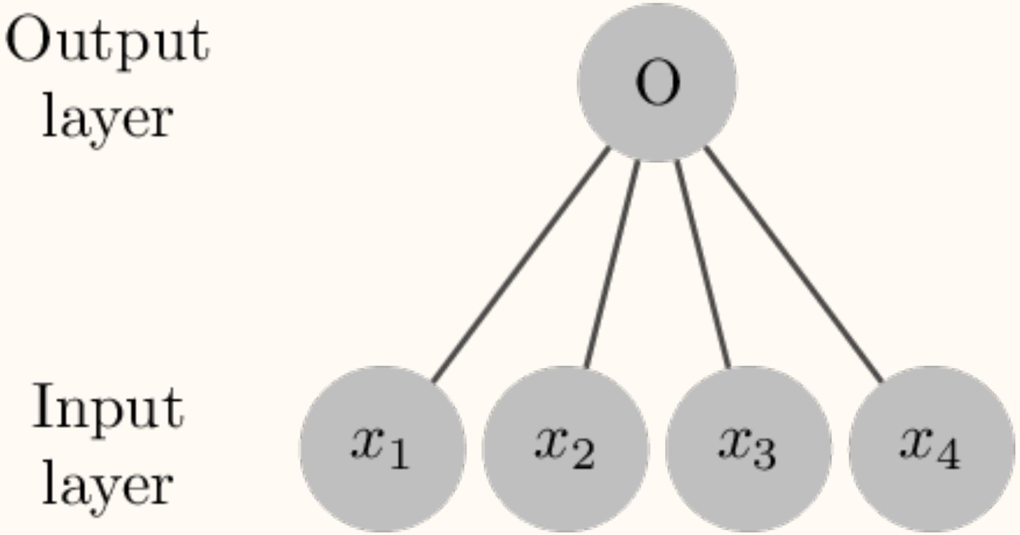
LR as a 1-layer neural network

In LR, we solve

$$\underset{a\in\mathbb{R}^p, b\in\mathbb{R}}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N} \ell(f_\theta(X_i), Y_i)$$

where $\ell(y_1, y_2) = \log(1 + e^{-y_1 y_2})$ and f_θ is linear.

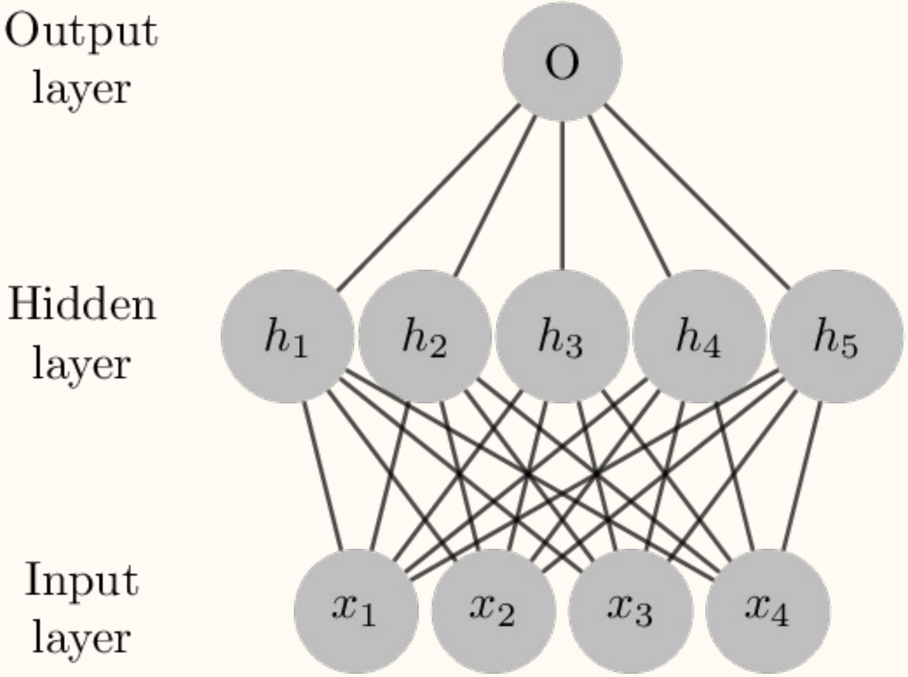
We can view $f_\theta(x) = O = a^\top x + b$ as a 1-layer (shallow) neural network.



Linear deep networks make no sense

What happens if we stack multiple linear layers?

Problem: This is pointless because composition of linear functions is linear.



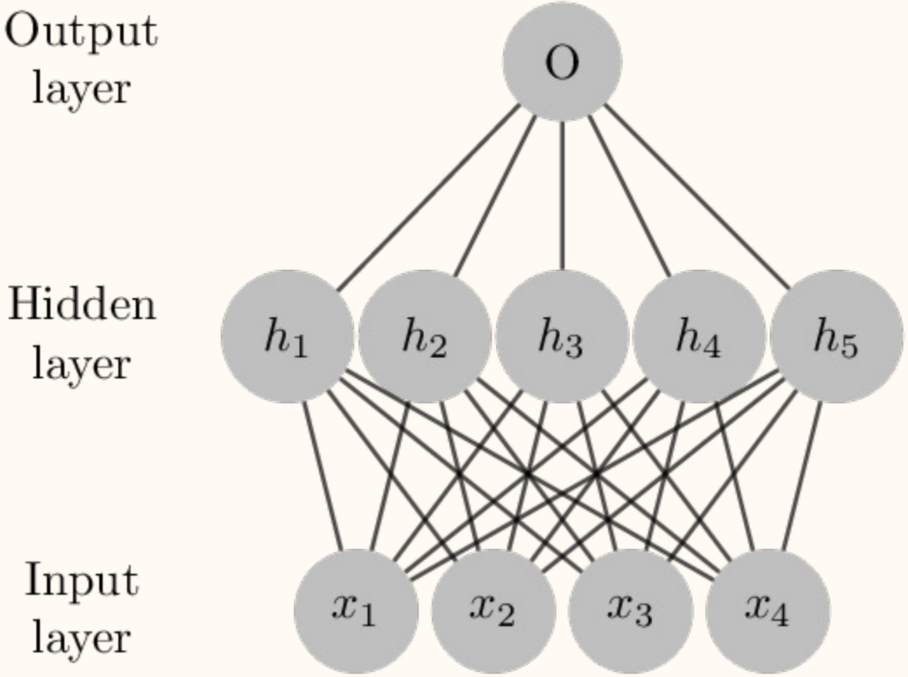
$\underset{1\times5}{O = W_2}h = W_2(W_1x) = (W_2W_1)x$ ← linear in x!

$h = \underset{5\times4}{W_1}x$

$h_i = \underset{1\times4}{(W_1)_i}x \quad i = 1, \ldots, 5$

Deep neural networks with nonlinearities

Solution: use a nonlinear activation function σ to inject nonlinearities.



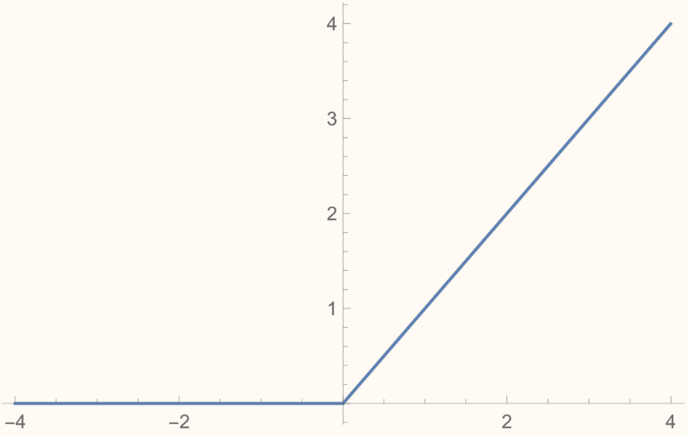
$O = \underset{1\times5}{W_2} h = W_2 \sigma(W_1 x)$ ← nonlinear in x

$h = \sigma(\underset{5\times4}{W_1} x)$ ← nonlinear function applied elementwise

Common activation functions

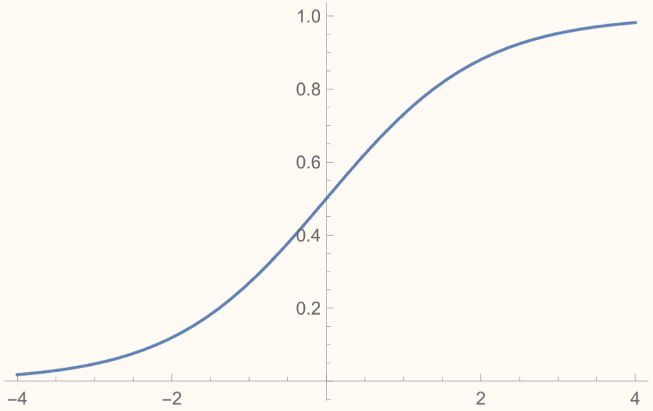
Rectified Linear Unit (ReLU)

$$\text{ReLU}(z) = \max(z, 0)$$



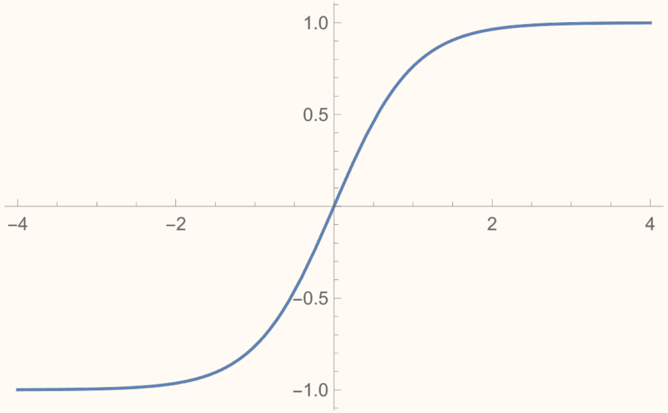
Sigmoid

$$\text{Sigmoid}(z) = \frac{1}{1 + e^{-z}}$$



Hyperbolic tangent

$$\tanh(z) = \frac{1 - e^{-2z}}{1 + e^{-2z}}$$



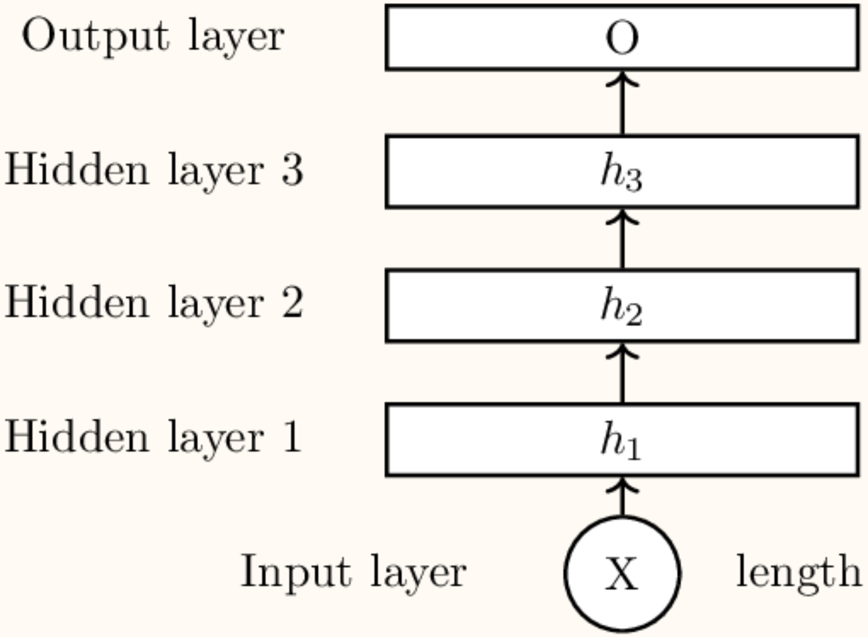
Multilayer perceptron (MLP)

The *multilayer perceptron*, also called *fully connected neural network*, has the form

$$\begin{aligned} y_L &= W_L y_{L-1} + b_L \\ y_{L-1} &= \sigma(W_{L-1} y_{L-2} + b_{L-1}) \\ &\vdots \\ y_2 &= \sigma(W_2 y_1 + b_2) \\ y_1 &= \sigma(W_1 x + b_1), \end{aligned}$$

where $x \in \mathbb{R}^{n_0}$, $W_\ell \in \mathbb{R}^{n_\ell \times n_{\ell-1}}$, $b_\ell \in \mathbb{R}^{n_\ell}$, and $n_L = 1.$ To clarify, σ is applied element-wise.

MLP for CIFAR10 binary classification



$$O = \underset{1\times 384}{A_4} h_3 + b_4 \in \mathbb{R}$$

$$h_3 = \sigma(\underset{384\times 768}{A_3} h_2 + b_3) \in \mathbb{R}^{384}$$

$$h_2 = \sigma(\underset{768\times 1536}{A_2} h_1 + b_2) \in \mathbb{R}^{768}$$

$$h_1 = \sigma(\underset{1536\times 3072}{A_1} x + b_1) \in \mathbb{R}^{1536}$$

length $32 \times 32 \times 3 = 3072$

activation function $\sigma = \mathrm{ReLU}$

PyTorch demo

Linear layer: Formal definition

Input tensor: $X \in \mathbb{R}^{B \times n}$, B batch size, n number of indices.

Output tensor: $Y \in \mathbb{R}^{B \times m}$, B batch size, m number of indices.

With weight $A \in \mathbb{R}^{m \times n}$, bias $b \in \mathbb{R}^{m}$, $k = 1, \dots B$, and $i = 1, \dots, m$:

$$Y_{k,i} = \sum_{j=1}^{n} A_{i,j} X_{k,j} + b_i$$

Operation is independent across elements of the batch.

If `bias=False`, then $b = 0$.

Weight initialization

Remember, SGD is

$$\theta^{k+1} = \theta^k - \alpha g^k$$

where $\theta^0 \in \mathbb{R}^p$ is an initial point.

In nice (convex) optimization problems, the initial point θ^0 is not important; you converge to the global solution no matter how you initialize.

In deep learning, it is very important to initialize θ^0 well. In fact, $\theta^0 = 0$ is a terrible idea.

Example) With an MLP with ReLU activations functions, if all weights and biases are initialized to be zero, then only the output layer's bias is trained and all other parameters do not move. So the training is stuck at a trivial network setting with $f_\theta(x) = \text{constant}$.

Weight initialization

PyTorch layers have default initialization schemes. (The default is *not* to initialize everything to 0.) Sometimes this default initialization scheme is sufficient (eg. Chapter 2 code.ipynb) sometimes it is not sufficient (eg. Hw3 problem 1).

How to initialize weights is tricky. More on this later.

Gradient computation via backprop

PyTorch and other deep learning libraries allows users to specify how to evaluate a function then compute derivatives (gradients) automatically.

No need to work out gradient computation by hand (even though I make you do it in homework assignments).

This feature is called, *automatic differentiation*, *back propagation*, or just the *chain rule*. This is implemented in the `torch.autograd` module.

More on this later.

Multi-class classification problem

Consider supervised learning with data $X_1, \dots, X_N \in \mathbb{R}^n$ and labels $Y_1, \dots, Y_N \in \{1, \dots, k\}$. (A k-class classification problem.) Assume there exists a function $f_\star : \mathbb{R}^n \to \Delta^k$ mapping from data to label probabilities. Here, $\Delta^k \subset \mathbb{R}^k$ denotes the set of probability mass functions on $\{1, \dots, k\}$.

Define the empirical distribution $\mathcal{P}(y) \in \mathbb{R}^k$ as the *one-hot vector*:

$$\big(\mathcal{P}(y)\big)_i = \begin{cases} 1 & \text{if } y = i \\ 0 & \text{otherwise} \end{cases}$$

for $i = 1, \dots, k$.

Softmax function

Examples:

$$\mu\left(\begin{bmatrix}1\\2\\3\end{bmatrix}\right)=\begin{bmatrix}0.09\\0.24\\0.67\end{bmatrix}$$

$$\mu\left(\begin{bmatrix}999\\0\\-2\end{bmatrix}\right)\approx\begin{bmatrix}1\\0\\0\end{bmatrix}$$

$$\mu\left(\begin{bmatrix}-2\\-2\\-99\end{bmatrix}\right)\approx\begin{bmatrix}0.5\\0.5\\0\end{bmatrix}$$

Softmax function $\mu : \mathbb{R}^k \to \Delta^k$ is defined by

$$\mu_i(z) = \big(\mu(z)\big)_i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}$$

for $i = 1, \dots, k$, where $z = (z_1, \dots, z_k) \in \mathbb{R}^k$. Since

$$\sum_{i=1}^{k} \mu_i(z) = 1, \qquad \mu > 0$$

Name “softmax” is a misnomer. “Softargmax” would be more accurate

- $\mu(z) \not\approx \max(z)$
- $\mu(z) \approx \operatorname{argmax}(z)$

Softmax regression

In *softmax regression* (SR):

1. Choose the model

$$\mu\left(f_{A,b}(x)\right) = \frac{1}{\sum_{i=1}^{k} e^{a_i^\top x + b_i}} \begin{bmatrix} e^{a_1^\top x + b_1} \\ e^{a_2^\top x + b_2} \\ \vdots \\ e^{a_k^\top x + b_k} \end{bmatrix}, \quad f_{A,b}(x) = Ax + b,\ A = \begin{bmatrix} a_1^\top \\ a_2^\top \\ \vdots \\ a_k^\top \end{bmatrix} \in \mathbb{R}^{k\times n},\ \ b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_k \end{bmatrix} \in \mathbb{R}^k.$$

2. Minimize KL-Divergence (or cross entropy) from the model $\mu\left(f_{A,b}(X_i)\right)$ output probabilities to the empirical distribution $\mathcal{P}(Y_i)$.

$$\underset{A\in\mathbb{R}^{k\times n}, b\in\mathbb{R}^k}{\text{minimize}} \quad \sum_{i=1}^{N} D_{\text{KL}}\left(\mathcal{P}(Y_i) \| \mu\left(f_{A,b}(X_i)\right)\right) \quad \Leftrightarrow \quad \underset{A\in\mathbb{R}^{k\times n}, b\in\mathbb{R}^k}{\text{minimize}} \quad \sum_{i=1}^{N} H\left(\mathcal{P}(Y_i), \mu\left(f_{A,b}(X_i)\right)\right)$$

Softmax regression

$$\underset{A\in\mathbb{R}^{k\times n}, b\in\mathbb{R}^k}{\text{minimize}} \quad \sum_{i=1}^{N} H\left(\mathcal{P}(Y_i), \mu\left(f_{A,b}(X_i)\right)\right)$$

$$\Updownarrow$$

$$\underset{A\in\mathbb{R}^{k\times n}, b\in\mathbb{R}^k}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N} -\log\left(\mu_{Y_i}\left(f_{A,b}(X_i)\right)\right)$$

$$\Updownarrow$$

$$\underset{A\in\mathbb{R}^{k\times n}, b\in\mathbb{R}^k}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{\exp\left(a_{Y_i}^\top X_i + b_{Y_i}\right)}{\sum_{j=1}^{k}\exp\left(a_j^\top X_i + b_j\right)}\right)$$

$$\Updownarrow$$

$$\underset{A\in\mathbb{R}^{k\times n}, b\in\mathbb{R}^k}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N}\left(-\left(a_{Y_i}^\top X_i + b_{Y_i}\right) + \log\left(\sum_{j=1}^{k}\exp\left(a_j^\top X_i + b_j\right)\right)\right)$$

Cross entropy loss

So

$$\underset{A\in\mathbb{R}^{k\times n},b\in\mathbb{R}^k}{\text{minimize}} \quad \sum_{i=1}^{N} H\left(\mathcal{P}(Y_i), \mu\left(f_{A,b}(X_i)\right)\right)$$

$$\Updownarrow$$

$$\underset{A\in\mathbb{R}^{k\times n},b\in\mathbb{R}^k}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^{N} \ell^{\mathrm{CE}}\left(f_{A,b}(X_i), Y_i\right)$$

where

$$\ell^{\mathrm{CE}}(f, y) = -\log\left(\frac{\exp(f_y)}{\sum_{j=1}^{k}\exp(f_j)}\right)$$

is the *cross entropy loss*.

Classification with deep networks

SR = linear model $f_{A,b}$ with cross entropy loss:

$$\underset{A\in\mathbb{R}^{k\times n},b\in\mathbb{R}^k}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^N \ell^{\mathrm{CE}}\big(f_{A,b}(X_i), Y_i\big) \quad \Leftrightarrow \quad \underset{A\in\mathbb{R}^{k\times n},b\in\mathbb{R}^k}{\text{minimize}} \quad \sum_{i=1}^N D_{\mathrm{KL}}\left(\mathcal{P}(Y_i)\|\mu\left(f_{A,b}(X_i)\right)\right)$$

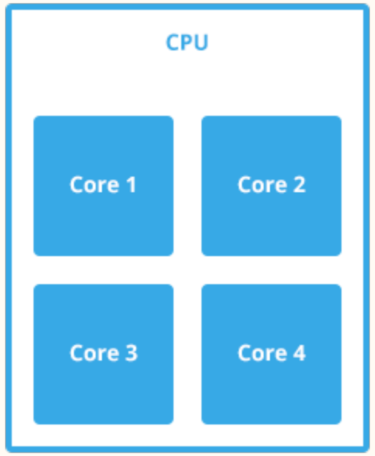
(Note $\ell^{\mathrm{CE}}(f,y) > 0$. More on homework 3.)

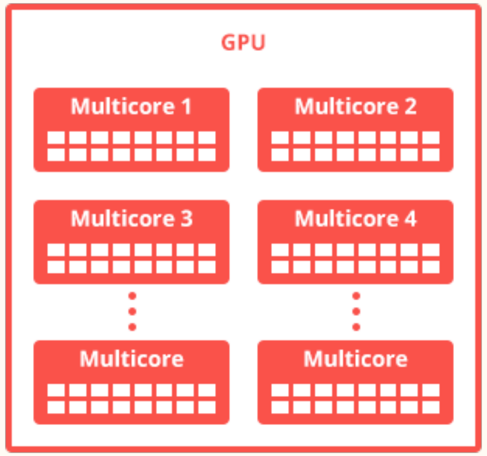
The natural extension of SR is to consider

$$\underset{\theta\in\mathbb{R}^p}{\text{minimize}} \quad \frac{1}{N}\sum_{i=1}^N \ell^{\mathrm{CE}}(f_\theta(X_i), Y_i) \quad \Leftrightarrow \quad \underset{\theta\in\mathbb{R}^p}{\text{minimize}} \quad \sum_{i=1}^N D_{\mathrm{KL}}\left(\mathcal{P}(Y_i)\|\mu\big(f_\theta(X_i)\big)\right)$$

where f_θ is a deep neural network.

History of GPU Computing





Rendering graphics involves computing many small tasks in parallel. Graphics cards provide many small processors to render graphics.

In 1999, Nvidia released GeForce 256 and introduced programmability in the form of vertex and pixel shaders. Marketed as the first ‘Graphical Processing Unit (GPU)’.

Researchers quickly learned how to implement linear algebra by mapping matrix data into textures and applying shaders.

To be precise, the GPU is the chip that goes inside the graphics card. The graphics card is the complete unit with the physical encasing, monitor port, and other supporting electronic circuits.

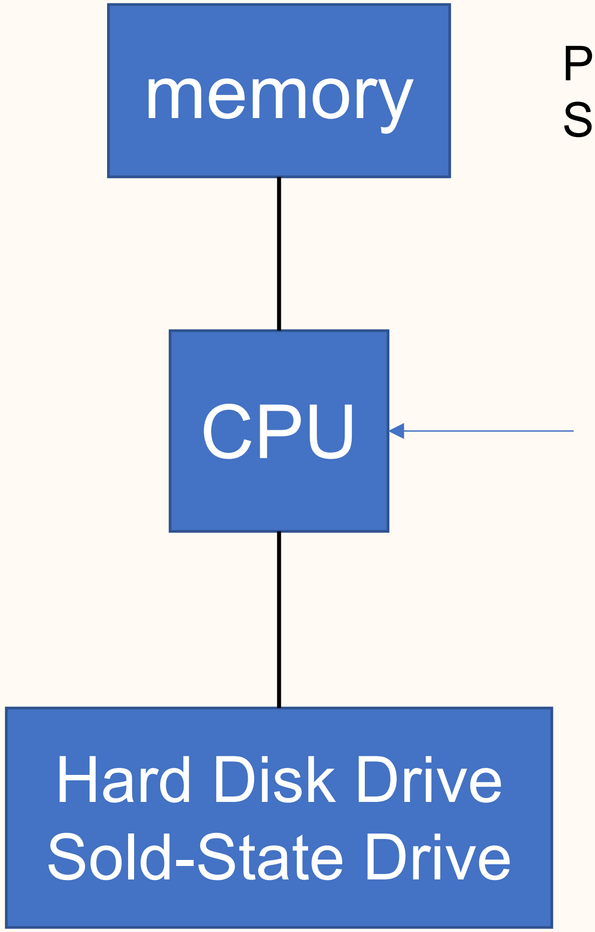
General Purpose GPUs (GPGPU)

In 2007, Nvidia released ‘Compute Unified Device Architecture (CUDA)’, which enabled general purpose computing on a CUDA-enabled GPUs.

Unlike CPUs which provide fast serial processing, GPUs provide massive parallel computing with its numerous slower processors.

The 2008 financial crisis hit Nvidia very hard as GPUs were luxury items used for games. This encouraged Nvidia to invest further in GPGPUs and create a more stable consumer base.

CPU computing model



memory

Primary storage
Stores variables (arrays) in programs

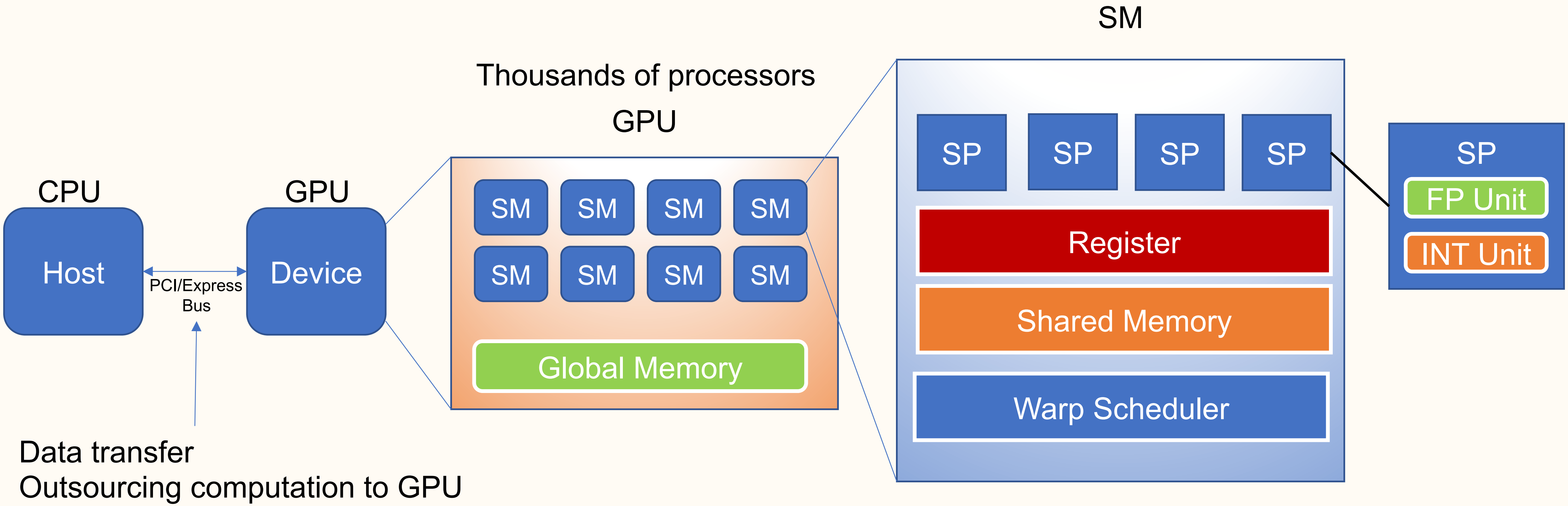
CPU

Several ≈ 12 processors

Hard Disk Drive
Sold-State Drive

Secondary Storage
Long-term storage

GPU computing model



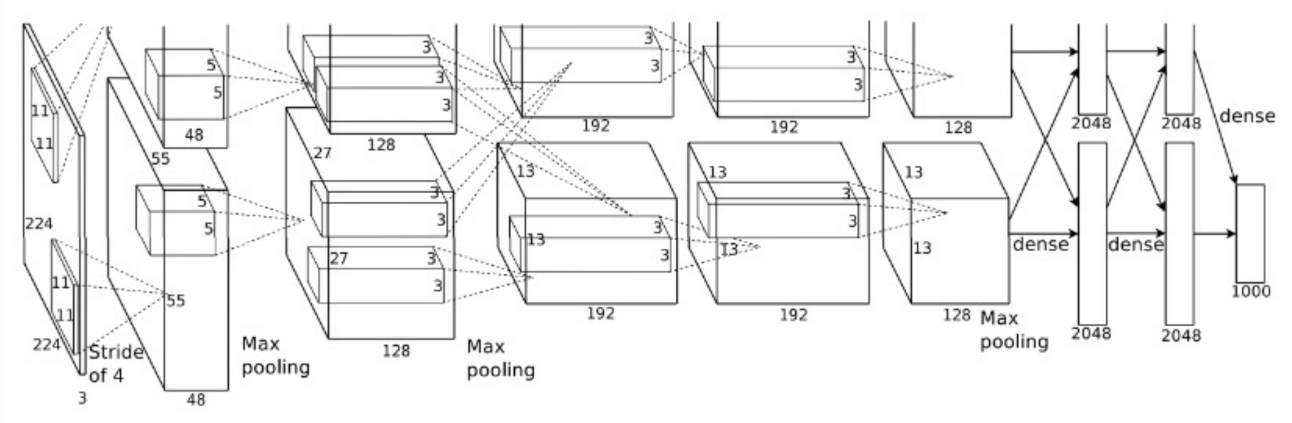
Data transfer
Outsourcing computation to GPU

GPUs for machine learning

Raina et al.’s 2009[*] paper demonstrated that GPUs can be used to train large neural networks. (This was not the first to use of GPUs in machine learning, but it was one of the most influential.)

Modern deep learning is driven by big data and big compute, respectively provided by the internet and GPUs.

Krizhevsky et al.’s 2012[*] landmark paper introduced AlexNet trained on GPUs and kickstarted the modern deep learning boom.



[*] R. Raina, A. Madhavan, and A. Y. Ng , Large-scale Deep Unsupervised Learning using Graphics Processors, *ICML*, 2009.
A. Krizhevsky, I. Sutskever, G. E. Hinton, ImageNet Classification with Deep Convolutional Neural Networks, *NeurIPS*, 2012.

Example: Power iteration with GPUs

Computing $x^{100} = A^{100} x^{0}$ with a GPU:

```
send A from host (CPU) to device (GPU)
send x=x0 from host (CPU) to device (GPU)
for _ in range(100):
  tell GPU to compute x=A*x
send x from device (GPU) to host (CPU)
```

In this example and deep learning, GPU accelerates computation since:
Amount of computation $\gg$ data communication.

Large information resides in the GPU, and CPU issues commands to perform computation on the data. (A in this example, neural network architecture in deep learning.)

PyTorch demo

Deep learning on GPUs

Steps for training neural network on GPU:

1. Create the neural network on CPU and send it to GPU. Neural network parameters stay on GPU.
    - Sometimes you load parameters from CPU to GPU.
2. Select data batch (image, label) and send it to GPU every iteration
    - Data for real-world setups is large, so keeping all data on GPU is infeasible.
3. On GPU, compute network output (forward pass)
4. On GPU, compute gradients (backward pass)
5. On GPU, perform gradient update
6. Once trained, perform prediction on GPU.
    - Send test data to GPU.
    - Compute network output.
    - Retrieve output on CPU.
    - Alternatively, neural network can be loaded on CPU and prediction can be done on CPU.

PyTorch demo