Artificial Intelligence as Statistical Learning

- Before we talk about GNNs, we need to specify what we mean by learning and intelligence

  $\Rightarrow$ Statistical Learning and Empirical Learning

- An Artificial Intelligence (AI) extracts information from observations $\Rightarrow$ Sorry to disappoint you
- E.g., this image is an observation. The intelligence tells you this is your professor, Alejandro

- Names vary across communities. My own professional bias is to talk of signals and signal processing

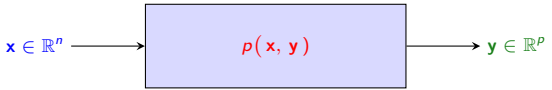
  $\Rightarrow$ And to call Observations = Input(s) and to call Information = Output(s)

- An Artificial Intelligence (AI) extracts information from observations $\Rightarrow$ Sorry to disappoint you
- E.g., this image is an observation. The intelligence tells you this is your professor, Alejandro

- Names vary across communities. My own professional bias is to talk of signals and signal processing

  $\Rightarrow$ And to call Observations = Input(s) and to call Information = Output(s)

- Observations (inputs) $\mathbf{x}$ and information (outputs) $\mathbf{y}$ are related by a statistical model $p(\mathbf{x}, \mathbf{y})$



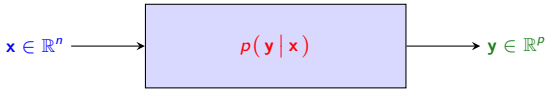
- Given that the universe (nature) associates inputs $\mathbf{x}$ and outputs $\mathbf{y}$ according to distribution $p(\mathbf{x}, \mathbf{y})$

  $\Rightarrow$ The AI should predict $\mathbf{y}$ from $\mathbf{x}$ with the conditional distribution $\Rightarrow \mathbf{y} \sim p(\mathbf{y} \,|\, \mathbf{x})$

  $\Rightarrow$ Or, if we want deterministic output, a conditional expectation $\Rightarrow \mathbf{y} = \mathbb{E}[\mathbf{y} \,|\, \mathbf{x}]$

- There is a lot to say about statistical estimation but this is beyond the scope of this course

- Observations (inputs) $\mathbf{x}$ and information (outputs) $\mathbf{y}$ are related by a statistical model $p(\mathbf{x}, \mathbf{y})$



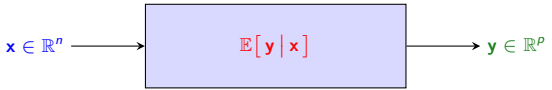
- Given that the universe (nature) associates inputs $\mathbf{x}$ and outputs $\mathbf{y}$ according to distribution $p(\mathbf{x}, \mathbf{y})$

  $\Rightarrow$ The AI should predict $\mathbf{y}$ from $\mathbf{x}$ with the conditional distribution $\Rightarrow \mathbf{y} \sim p(\mathbf{y} \mid \mathbf{x})$

  $\Rightarrow$ Or, if we want deterministic output, a conditional expectation $\Rightarrow \mathbf{y} = \mathbb{E}[\mathbf{y} \mid \mathbf{x}]$

- There is a lot to say about statistical estimation but this is beyond the scope of this course

- Observations (inputs) $\mathbf{x}$ and information (outputs) $\mathbf{y}$ are related by a statistical model $p(\mathbf{x}, \mathbf{y})$



- Given that the universe (nature) associates inputs $\mathbf{x}$ and outputs $\mathbf{y}$ according to distribution $p(\mathbf{x}, \mathbf{y})$

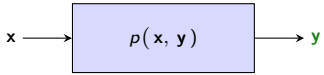
  $\Rightarrow$ The AI should predict $\mathbf{y}$ from $\mathbf{x}$ with the conditional distribution $\Rightarrow \mathbf{y} \sim p(\mathbf{y} \mid \mathbf{x})$

  $\Rightarrow$ Or, if we want deterministic output, a conditional expectation $\Rightarrow \mathbf{y} = \mathbb{E}[\mathbf{y} \mid \mathbf{x}]$

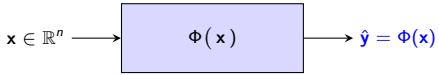
- There is a lot to say about statistical estimation but this is beyond the scope of this course

- AI is not perfect. Nature and AI may produce different outputs when presented with the same input

Nature relates $\mathbf{x}$ and $\mathbf{y}$ with distribution $p(\mathbf{x}, \mathbf{y})$



The AI relates $\mathbf{x}$ and $\hat{\mathbf{y}}$ with function $\Phi(\mathbf{x})$



- Loss function $\ell(\mathbf{y}, \hat{\mathbf{y}}) = \ell(\mathbf{y}, \Phi(\mathbf{x}))$ measures cost of predicting $\hat{\mathbf{y}} = \Phi(\mathbf{x})$ when actual output is $\mathbf{y}$

  $\Rightarrow$ In estimation problems we often use quadratic loss $\Rightarrow \ell(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$

  $\Rightarrow$ In classification problems we often use hit loss $\Rightarrow \ell(\mathbf{y}, \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|_0 = \#(\mathbf{y} \neq \hat{\mathbf{y}})$

- Average the loss $\ell(\mathbf{y}, \Phi(\mathbf{x}))$ over nature's distribution $p(\mathbf{x}, \mathbf{y})$ and choose best estimator/classifier

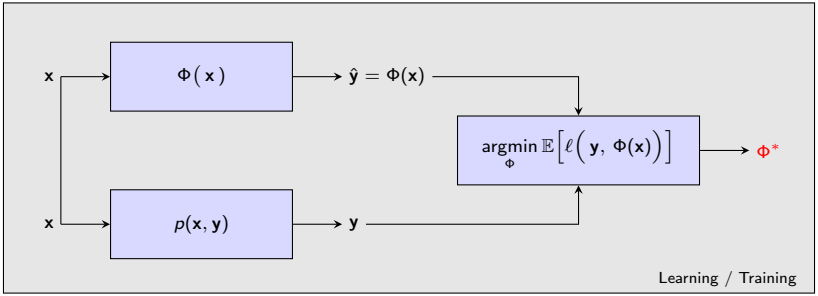
$$\Phi^* = \underset{\Phi}{\operatorname{argmin}}\, \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\Big[\, \ell\Big(\mathbf{y}, \Phi(\mathbf{x})\Big)\,\Big]$$

- Predict $\Phi(\mathbf{x})$. Nature draws $\mathbf{y}$. Evaluate loss ℓ. Take loss expectation over distribution $p(x, y)$

  $\Rightarrow$ Optimal estimator is the function with minimum average cost over all possible estimators.

- This optimization program is called the statistical risk minimization (SRM) problem

- Learning, or Training, is the process of solving the statistical risk minimization problem



- Outcome of learning is function Φ^* with minimum average statistical loss $\Rightarrow$ We learn to estimate $\mathbf{y}$

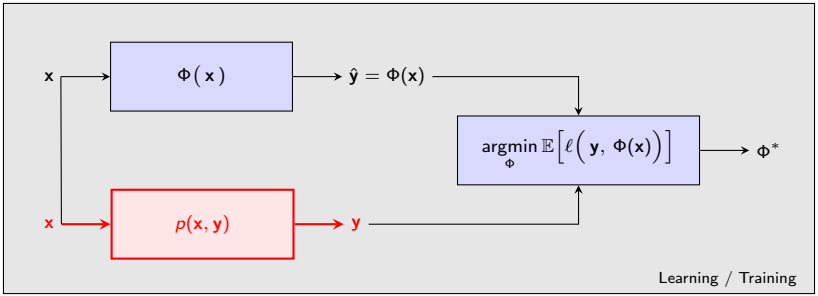
  $\Rightarrow$ During execution time, we just evaluate $\Phi(\mathbf{x}^*)$ to predict output associated with input $\mathbf{x}$

A Word on Models

- We have seen how to formulate learning as a mathematical program
- Our formulation requires access to a model (probability distribution). This is easier said than done

- We have reduced learning to the solution of a statistical risk minimization (SRM) problem

  $\Rightarrow$ Requires access to the distribution $p(\mathbf{x}, \mathbf{y})$ $\Rightarrow$ A model of how $\mathbf{x}$ and $\mathbf{y}$ are jointly generated

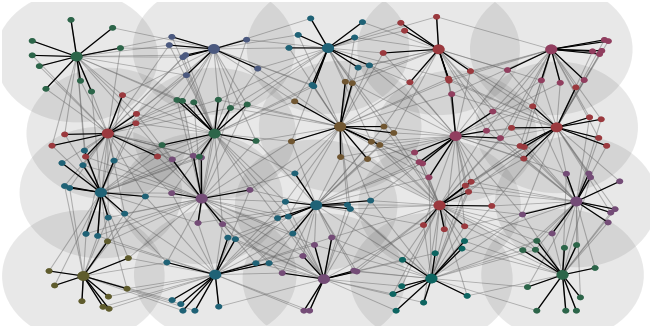


- This block diagram does not work without a model $\Rightarrow$ Where is the model coming from?

- Maybe we know the laws that relate inputs and outputs $\Rightarrow$ Indeed, we very often do

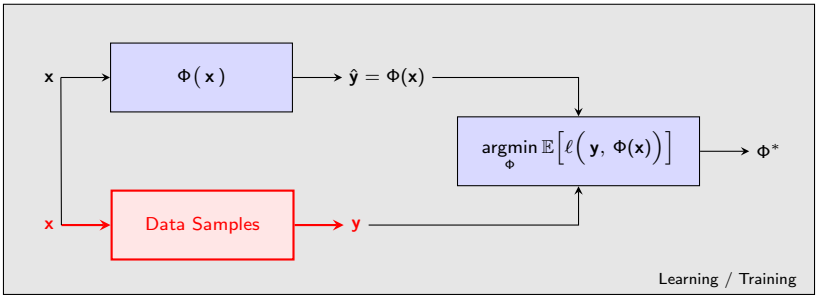
- Do not underestimate models $\Rightarrow$ This is how we design the vast majority of marvels around you

- Or, we acquire data pairs $(\mathbf{x}_q, \mathbf{y}_q) \sim p(\mathbf{x}, \mathbf{y})$ to estimate the model $\Rightarrow$ We learn the distribution



- Very powerful too $\Rightarrow$ What we do not design with models, we design with systems identification

- Bypass the learning of the distribution $\Rightarrow$ Go straight to the learning of the estimation map $\Phi(\mathbf{x})$

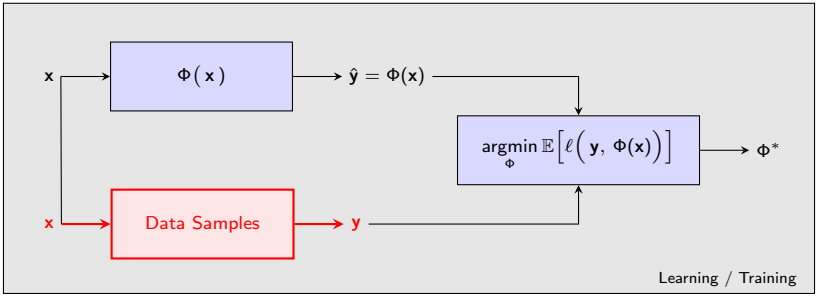


- Very powerful $\Rightarrow$ Recent impressive transformations in speech processing and computer vision

Empirical Risk Minimization

- Learning bypasses models. It tries to imitate observations. Let us formulate mathematically.

- AI and ML in this course refer to the pipeline where we learn from data samples. Not distributions



- AI learns to imitate input-output pairs observed in nature.

- Statistical Risk Minimization works on the cost averaged over the distribution of inputs and outputs

$$\Phi^* = \underset{\Phi}{\operatorname{argmin}}\, \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\Big[\ell\Big(\mathbf{y},\, \Phi(\mathbf{x})\Big)\Big]$$

- This expectation can be approximated with data

  $\Rightarrow$ Acquire training set with Q pairs $(\mathbf{x}_q, \mathbf{y}_q) \in \mathcal{T}$ drawn independently from distribution $p(\mathbf{x}, \mathbf{y})$

  $\Rightarrow$ For sufficiently large Q we can approximate $\Rightarrow \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\Big[\ell\Big(\mathbf{y},\, \Phi(\mathbf{x})\Big)\Big] \approx \frac{1}{Q}\sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q,\, \Phi(\mathbf{x}_q)\Big)$

  $\Rightarrow$ This is just the law of large numbers. True under very mild conditions

- Replace statistical risk minimization (SRM) with empirical risk minimization (ERM)

$$\Phi_S^* = \underset{\Phi}{\operatorname{argmin}}\, \mathbb{E}_p(\mathbf{x}, \mathbf{y})\Big[\ell\Big(\mathbf{y},\, \Phi(\mathbf{x})\Big)\Big] \quad \Rightarrow \quad \Phi_E^* = \underset{\Phi}{\operatorname{argmin}}\, \frac{1}{Q}\sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q,\, \Phi(\mathbf{x}_q)\Big)$$

- Since the objectives are close, one would think the optima are close $\Rightarrow \Phi_S^* \approx \Phi_E^*$

  $\Rightarrow$ Alas, this it not true $\Rightarrow \Phi_S^* \not\approx \Phi_E^*$ $\Rightarrow$ Statistical and empirical risk minimizers need not be close

- In fact, the solution of ERM is trivial $\Rightarrow$ Make $\Phi(\mathbf{x}_q) = \mathbf{y}_q$ for all pairs in the training set

- As trivial as nonsensical $\Rightarrow$ Yields no information about observations outside the training set

ERM with Learning Parametrizations

- Our first attempt at learning from data led to an ERM problem that does not make sense
- The search for a problem that makes sense brings us to the notion of learning parametrizations

- A sensical ERM problem, requires the introduction of a function class $\mathcal{C}$

$$\Phi^* = \underset{\Phi \in \mathcal{C}}{\operatorname{argmin}} \frac{1}{Q} \sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q) \Big)$$

- For example, we can select the class of linear functions $\Phi(\mathbf{x}) = \mathbf{H}\mathbf{x}$ and solve for

$$\mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \frac{1}{Q} \sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q, \mathbf{H}\,\mathbf{x}_q \Big)$$

- This choice of parametrization may be good or bad. But at least is sensical

  $\Rightarrow$ Good or bad, having $\mathbf{H}^*$ allows estimates $\hat{\mathbf{y}} = \mathbf{H}^*\mathbf{x}$ for observations $\mathbf{x}$ outside the training set

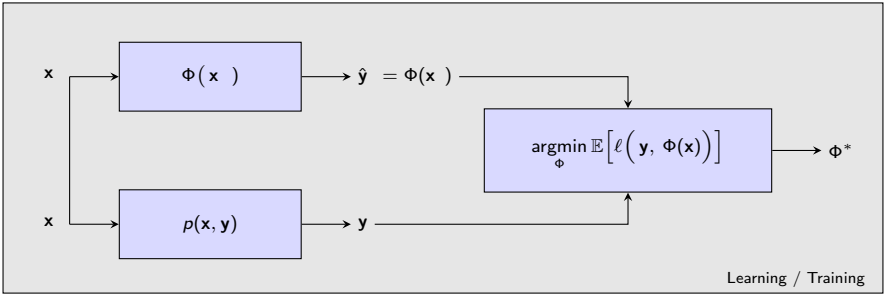
- Selecting $\mathcal{C}$ to contain sufficiently smooth functions makes SRM and ERM close

$$\underset{\Phi \in \mathcal{C}}{\operatorname{argmin}}\, \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\Big[\ell\Big(\mathbf{y},\, \Phi(\mathbf{x})\Big)\Big] \quad \approx \quad \underset{\Phi \in \mathcal{C}}{\operatorname{argmin}}\, \frac{1}{Q}\sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q,\, \Phi(\mathbf{x}_q)\Big)$$

- Fundamental theorem of statistical learning $\Rightarrow$ ERM is a valid approximation of SRM

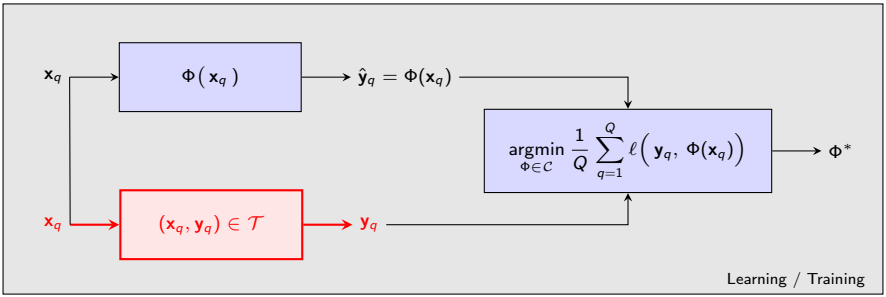
- Need to identify the appropriate function class $\mathcal{C}$ $\Rightarrow$ But this problem is unavoidable

- SRM learns from model $\Rightarrow$ Parametrized ERM learns from data $\Rightarrow$ Three differences:



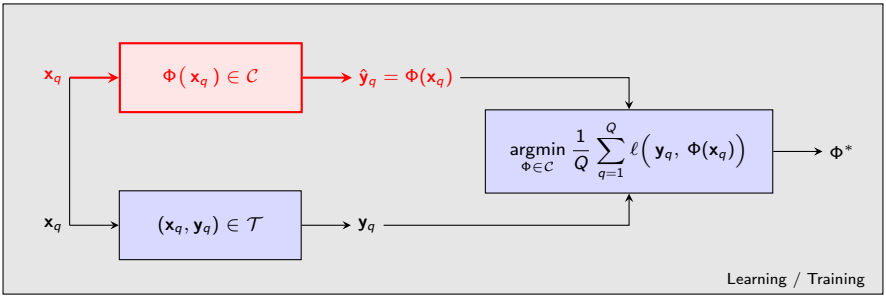
- SRM learns from model $\Rightarrow$ Parametrized ERM learns from data $\Rightarrow$ Three differences:

  $\Rightarrow$ The distribution is unknown $\Rightarrow$ We have access to a training set of data samples



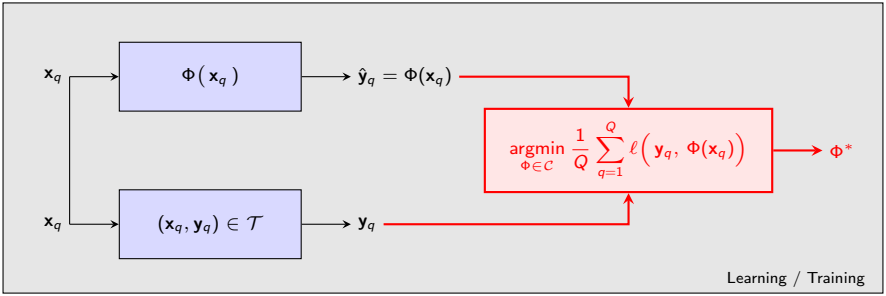
- SRM learns from model $\Rightarrow$ Parametrized ERM learns from data $\Rightarrow$ Three differences:

  $\Rightarrow$ The nonparametric ERM problem is nonsensical $\Rightarrow$ We restrict the function class



- SRM learns from model $\Rightarrow$ Parametrized ERM learns from data $\Rightarrow$ Three differences:

  $\Rightarrow$ The statistical risk $\Rightarrow$ Is replaced by the empirical risk



- Here, Machine learning (ML) $\equiv$ Artificial Intelligence (AI) $\equiv$ Empirical Risk Minimization (ERM)

$$\boldsymbol{\Phi}^* = \underset{\boldsymbol{\Phi}\in\mathcal{C}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{y}, \boldsymbol{\Phi}(\mathbf{x})\Big) = \underset{\Phi\in\mathcal{C}}{\operatorname{argmin}} \frac{1}{Q}\sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q, \boldsymbol{\Phi}(\mathbf{x}_q)\Big)$$

- The components of ERM are a dataset, a loss function and, most importantly, a function class
- Make parametrization more explicit $\Rightarrow$ Parameter $\mathbf{H} \in \mathbb{R}^p$ to span function class $\Phi(\mathbf{x}; \hat{\mathbf{H}})$

$$\mathbf{H}^* = \underset{\mathbf{H}}{\operatorname{argmin}} \sum_{(\mathbf{x},\mathbf{y})\in\mathcal{T}} \ell\Big(\mathbf{y}, \boldsymbol{\Phi}(\mathbf{x}; \mathbf{H})\Big)$$

- Designing an ML / AI system means selecting the appropriate function class $\mathcal{C}$ $\Rightarrow$ What else?

$\Rightarrow$ The function class determines generalization from inputs in training set to unseen inputs

Stochastic Gradient Descent (SGD)

- SGD is the customary method for the minimization of the empirical risk

- We have seen that the training of an estimator requires minimization of the empirical risk

$$\mathbf{H}^* = \underset{\mathbf{H}\in\mathbb{R}^P}{\operatorname{argmin}} \frac{1}{Q}\sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q;\mathbf{H})\Big) = \underset{\mathbf{H}\in\mathbb{R}^P}{\operatorname{argmin}}\, L(\mathbf{H})$$

- Minimization of the average loss function defined as the average of pointwise loss function

$$L(\mathbf{H}) := \frac{1}{Q}\sum_{q=1}^{Q} \ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q;\mathbf{H})\Big)$$

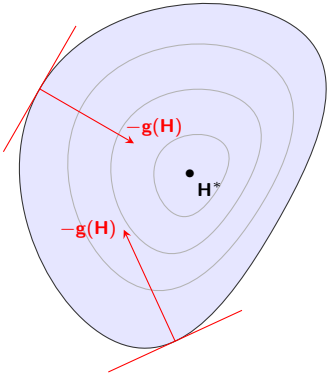
- It's particular form notwithstanding, just a minimization $\Rightarrow$ Use gradient descent algorithm

- Gradient $\mathbf{g}(\mathbf{H}) = \nabla L(\mathbf{H})$ is perpendicular to level set of loss $L(\mathbf{H})$
- Thus, they point towards the minimum. Not directly. Towards

  $\Rightarrow$ Angle is less than $\pi/2 \Rightarrow -\mathbf{g}^T(\mathbf{H})\,(\mathbf{H} - \mathbf{H}^*) \geq 0$
- We can then use gradients in a gradient descent algorithm

$$\mathbf{H}_{t+1} = \mathbf{H}_t - \epsilon\, \mathbf{g}(\mathbf{H}_t)$$

- Converges to the optimum $\mathbf{H}^*$ if the stepsize ϵ is sufficiently small



- The gradient of the average loss function is the average of the gradients of the pointwise losses

$$\mathbf{g}(\mathbf{H}) \;=\; \nabla L(\mathbf{H}) \;=\; \frac{1}{Q}\sum_{q=1}^{Q} \nabla_{\mathbf{H}} \ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q; \mathbf{H})\Big)$$

- Equipped with gradients, we write the gradient descent method as the recursion given by

$$\mathbf{H}_{t+1} \;=\; \mathbf{H}_t - \epsilon\, \mathbf{g}(\mathbf{H}_t) \;=\; \mathbf{H}_t - \frac{\epsilon}{Q}\sum_{q=1}^{Q} \nabla_{\mathbf{H}} \ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q; \mathbf{H}_t)\Big)$$

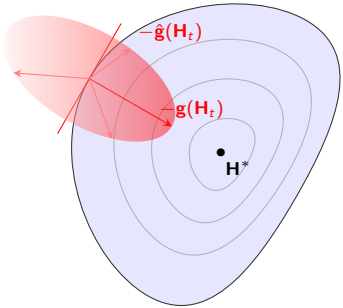
- This is all good, but those gradients are costly to compute $\Rightarrow$ An average of Q pointwise gradients

- At iteration t, select a batch of $Q_t \ll Q$ samples $\mathcal{T}_t$. Randomly chosen from dataset $\mathcal{T}$
- Define stochastic gradient as sum over batch $\hat{\mathbf{g}}(\mathbf{H}_t) = \frac{1}{Q_t} \sum_{(\mathbf{x}_q, \mathbf{y}_q) \in \mathcal{T}_t} \nabla_{\mathbf{H}} \ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q; \mathbf{H}_t)\Big)$
- SGD $\equiv$ Replace gradients with stochastic gradients in gradient descent

$$\mathbf{H}_{t+1} = \mathbf{H}_t - \epsilon\, \hat{\mathbf{g}}(\mathbf{H}_t) = \mathbf{H}_t - \frac{1}{Q_t} \sum_{(\mathbf{x}_q, \mathbf{y}_q) \in \mathcal{T}_t} \nabla_{\mathbf{H}} \ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q; \mathbf{H}_t)\Big)$$

- This is cheaper to implement because the sum is over a smaller number of pointwise gradients.

- If samples are chosen independently and with equal probability $\Rightarrow \mathbb{E}(\hat{\mathbf{g}}(\mathbf{H}_t)) = \mathbf{g}(\mathbf{H}_t)$
- Stochastic gradients point in the right direction on average
- We move towards the optimum more often than not
- Expected angle is accute $\Rightarrow \mathbb{E}\Big[-\hat{\mathbf{g}}^T(\mathbf{H})\,(\mathbf{H} - \mathbf{H}^*)\Big] \geq 0$
- Can build a submartingale and prove convergence



Stochastic Gradient Descent Memorabilia

- I covered SGD briefly because there are some things I wanted you to know.

- GD converges because negative gradients point towards the optimum $\Rightarrow -\mathbf{g}^T(\mathbf{H})\,(\mathbf{H}-\mathbf{H}^*) \geq 0$
- SGD converges because stochastic gradients do so on expectation $\Rightarrow \mathbb{E}\Big[-\hat{\mathbf{g}}^T(\mathbf{H})\,(\mathbf{H}-\mathbf{H}^*)\Big] \geq 0$
- Computing stochastic gradients is much cheaper ($Q_t \ll Q$) than the cost of computing gradients

$$\hat{\mathbf{g}}(\mathbf{H}_t) \;=\; \frac{1}{Q_t}\sum_{(\mathbf{x}_q,\mathbf{y}_q)\in\mathcal{T}_t} \nabla_{\mathbf{H}}\ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q;\mathbf{H}_t)\Big) \qquad \text{vs} \qquad \hat{\mathbf{g}}(\mathbf{H}_t) \;=\; \frac{1}{Q}\sum_{(\mathbf{x}_q,\mathbf{y}_q)\in\mathcal{T}} \nabla_{\mathbf{H}}\ell\Big(\mathbf{y}_q, \Phi(\mathbf{x}_q;\mathbf{H}_t)\Big)$$

- The difference between a method that works and a method that does not work.

- Convergence means that as iteration index t grows $\Rightarrow$ $\liminf_{t \to \infty} \left\| \mathbf{H}_t - \mathbf{H}^* \right\|^2 \leq \mathcal{O}\left(\epsilon \,/\, \sqrt{Q_t} \right)$

  $\Rightarrow$ We do not converge exactly $\Rightarrow$ We approach the optimum and hover around it

  $\Rightarrow$ Size of hover region is proportional to stepsize

  $\Rightarrow$ Size of hover region is inversely proportional to square root of batch size

- For large batch size Q_t we have $\hat{\mathbf{g}}(\mathbf{H}_t) \approx \mathbf{g}(\mathbf{H}_t)$ $\Rightarrow$ Not needed $\Rightarrow$ Mistakes corrected in next step

- Plots illustrates, comments apply, and results hold for convex functions

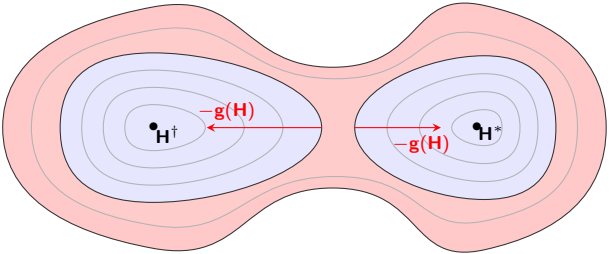
  $\Rightarrow$ Not always (rarely!) true $\Rightarrow$ Notably, not true for neural networks. Convolutional or not

- Gradients may move iterates towards

  $\Rightarrow$ Global minimum $\mathbf{H}^*$

  $\Rightarrow$ Local minimum $\mathbf{H}^\dagger$

- Depending on initial condition



- We may converge to local minima but we won't care $\Rightarrow$ Implicitly assume that $\mathbf{H}^\dagger$ is optimal

Stochastic Gradient Descent is Finicky

- Stochastic gradient descent is not a great algorithm. Just the one we have.
- Convergence speed and convergence itself is very sensitive to choice of parameters
- Requires trying different stepsizes and different batch sizes. Maybe different initial conditions

  $\Rightarrow$ Small changes in any of these parameters may have large effects on convergence

The Importance of Learning Parametrizations

- AI reduces to ERM. And in ERM all we have to do is choose a parametrization.
- Not an easy choice $\Rightarrow$ The parametrization controls generalization. Make or break.
- The parametrization is a model of how outputs are related to inputs $\Rightarrow$ It has to be accurate

- To illustrate effect of learning parametrizations generate fake data following models we specify
- A linear model with inputs $\mathbf{x} \in \mathbb{R}^n$ and outputs $\mathbf{y} \in \mathbb{R}^m$ related by $\Rightarrow \mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{w}$

  $\Rightarrow$ For some matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$. Noise $\mathbf{w} \in \mathbb{R}^p$ Gaussian white, independent with mean $\mathbb{E}(\mathbf{w}) = \mathbf{0}$
- A non-linear model postprocessing the linear model with a sign function $\Rightarrow \mathbf{y} = \text{sign}\big(\mathbf{A}\mathbf{x} + \mathbf{w}\big)$

- Given that we know the models we can compute the Statistical Risk Minimizer (SRM). ""The AI"
- For instance, if we use the squared 2-norm loss to penalize AI estimation errors

$$\Phi_S^*(\mathbf{x}) = \underset{\Phi}{\operatorname{argmin}}\, \mathbb{E}_{p(\mathbf{x},\mathbf{y})}\left[\frac{1}{2}\|\mathbf{y} - \Phi(\mathbf{x})\|_2^2\right]$$

- Using the given model $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{w}$ and taking derivatives, the AI is $\Rightarrow \Phi_S^*(\mathbf{x}) = \mathbf{A}\mathbf{x}$
- Literally, the AI mimics nature.

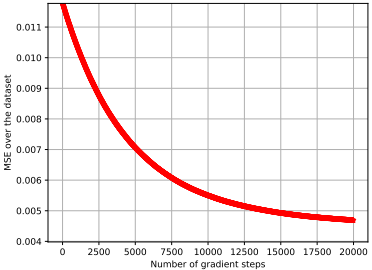
- Suppose model is unknown $\Rightarrow$ Instead, we have access to Q data pairs $(\mathbf{x}_q, \mathbf{y}_q)$ in training set $\mathcal{T}$
- Hypothesize a linear parametrization $\Phi(\mathbf{x}) = \mathbf{H}\mathbf{x}$ Formulate parametrized ERM problem

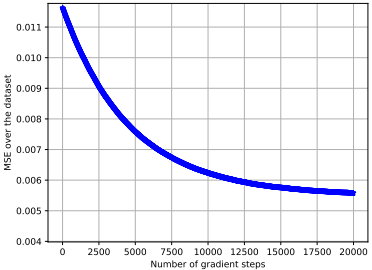
$$\mathbf{H}^* = \underset{\mathbf{H}\in\mathbb{R}^{m\times n}}{\operatorname{argmin}} \frac{1}{Q}\sum_{q=1}^{Q}\left[\frac{1}{2}\left\|\mathbf{y}_q - \mathbf{H}\mathbf{x}_q\right\|_2^2\right] = \underset{\mathbf{H}\in\mathbb{R}^{m\times n}}{\operatorname{argmin}} L(\mathbf{H})$$

- Solve with SGD $\Rightarrow \mathbf{H}_{t+1} = \mathbf{H}_t - \epsilon\,\hat{\mathbf{g}}(\mathbf{H}_t) = \mathbf{H}_t - \frac{\epsilon}{Q_t}\sum_{q=1}^{Q_t}\left(\mathbf{y}_q - \mathbf{H}_t\mathbf{x}_q\right)\mathbf{x}_q^T$
- Can use linear parametrization irrespectively of the actual model relating inputs $\mathbf{x}_q$ to outputs $\mathbf{y}_q$.

  $\Rightarrow$ But it will work well only if the parametrization matches the unknown model.

- Data generated by linear model with dimensions $m = n = 10^2$. Number of samples $Q = 10^3$.
- ERM learning with linear parametrization. $\Rightarrow$ SGD trajectory iterates reduce loss (left)
- Live operation tested outside of training set $\Rightarrow$ loss is also reduced in test set

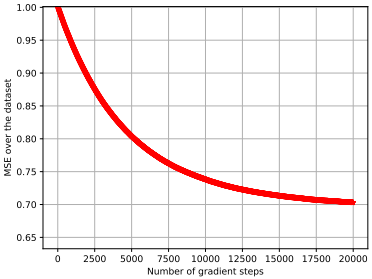


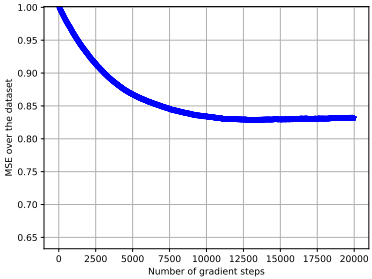


- The model is linear. The parametrization is linear. The parametrization learn the model.

Parametrization and Model are Mismatched

- Data generated by sign model with dimensions $m = n = 10^2$. Number of samples $Q = 10^3$.
- ERM learning with linear parametrization. $\Rightarrow$ SGD trajectory iterates reduce loss (left)
- But we converge to a high loss. We do not learn. $\Rightarrow$ Situation is just as bad in the test set

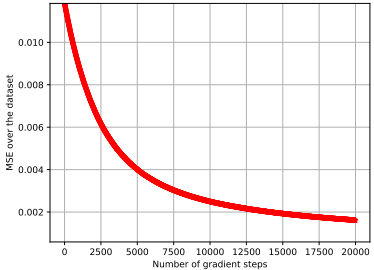


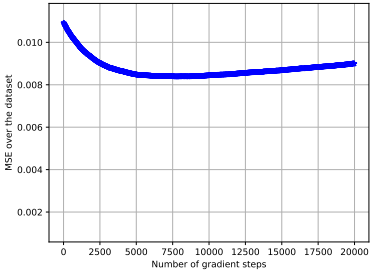


- Model is **NOT** linear. Parametrization is linear. The parametrization **DOES NOT** learn the model.

Parametrization and Model are Matched but There is Insufficient Data

- Data generated by linear model with dimensions $m = n = 10^2$. Number of samples $Q = 10^2$.
- ERM learning with linear parametrization. $\Rightarrow$ SGD trajectory iterates reduce loss (left)
- Live operation tested outside of training set $\Rightarrow$ loss is **NOT** reduced in test set





- Model is linear. Parametrization is linear. Not enough data to learn model. $\Rightarrow$ There never is

Machine Learning is Model Free but Not Model Free

- Machine learning does not require a model relating inputs **x** to outputs **y**

  $\Rightarrow$ For example, we don't need to know the matrix **A**

- But we need to know a class of functions to which the model belongs

  $\Rightarrow$ For example, we need to know the model relating inputs to outputs is linear

- Model also needs to be sufficiently simple to operate with insufficient data

  $\Rightarrow$ This is where we leverage structure using convolutional architectures such as CNNs and GNNs