LAMARR
INSTITUTE FOR MACHINE LEARNING AND ARTIFICIAL INTELLIGENCE

Deep Learning

An Introduction

Mirko Bunse, Quentin Führing, and Vukan Jevtic

Grad School on Astro-Particle Physics (Jan 18th–23rd, 2026)

Partner institutions:

technische universität dortmund
Fraunhofer IAIS
Universität Bonn
Fraunhofer IML

Institutionally funded by:

Bundesministerium für Forschung, Technologie und Raumfahrt
Ministerium für Kultur und Wissenschaft des Landes Nordrhein-Westfalen

Further Reading

Goodfellow, Bengio, and Courville, *Deep Learning*, 2016:

- principled and rigorous approach
- great technical coverage

Further Reading

Goodfellow, Bengio, and Courville, *Deep Learning*, 2016:

- principled and rigorous approach
- great technical coverage

Bishop and Bishop, *Deep Learning - Foundations and Concepts*, 2024:

- even more basics
- more advanced topics, like transformers

Further Reading

Goodfellow, Bengio, and Courville, *Deep Learning*, 2016:

- principled and rigorous approach
- great technical coverage

Bishop and Bishop, *Deep Learning - Foundations and Concepts*, 2024:

- even more basics
- more advanced topics, like transformers

Erdmann et al., *Deep Learning for Physics Research*, 2021:

- physics-oriented examples and exercises
- (some) coverage of uncertainties and custom loss functions

Further Reading

Goodfellow, Bengio, and Courville, *Deep Learning*, 2016:

- principled and rigorous approach
- great technical coverage

Bishop and Bishop, *Deep Learning - Foundations and Concepts*, 2024:

- even more basics
- more advanced topics, like transformers

Erdmann et al., *Deep Learning for Physics Research*, 2021:

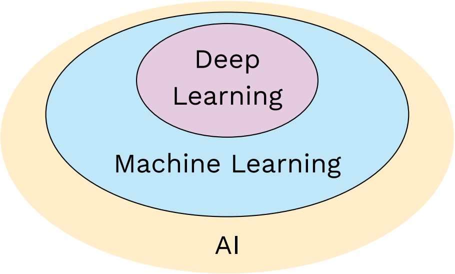
- physics-oriented examples and exercises
- (some) coverage of uncertainties and custom loss functions

Lippe, *UvA Deep Learning Tutorials*, 2023:

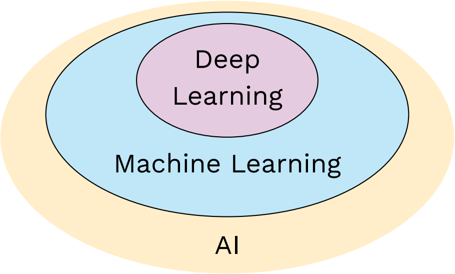
- `https://uvadlc-notebooks.readthedocs.io`

Introduction / Machine Learning

Machine Learning



Machine Learning



Machine learning = data ∘ model ∘ fit

Supervised Learning

Target: any quantity Y we want to predict (costly or impossible to measure)

Feature: any quantity X_i we compute from observable quantities

Training Data: $\mathrm{D} = \left\{ (x_i, y_i) \in \mathcal{X} \times \mathcal{Y} : \; 1 \leq i \leq m \right\}$

Supervised Learning

Target: any quantity Y we want to predict (costly or impossible to measure)

Feature: any quantity X_i we compute from observable quantities

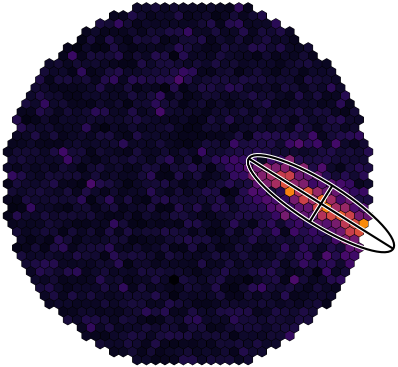
Training Data: $\mathrm{D} = \{ (x_i, y_i) \in \mathcal{X} \times \mathcal{Y} : 1 \leq i \leq m \}$

target	feature	feature	
Y	X_1	X_2	
+1	1.3	A	$\cdots$
-1	-0.2	B	
+1	0.8	A	
	$\vdots$		$\ddots$

Structured data:

- tabular representation
- X_i facilitate the prediction of Y, e.g., through well-designed preprocessing

Structured Data



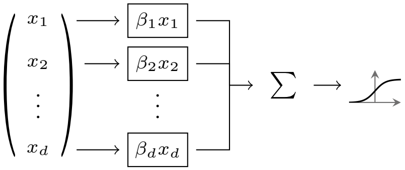
```python
df = fact.io.read_data( # pandas.DataFrame
    "gamma_simulations_facttools_dl2.hdf5",
    key = "events"
)

X = df[[ # select features
    "length", # -> shape (n_events, n_features)
    "width",
    "num_islands",
    "num_pixel_in_shower",
    # ...
]].to_numpy()

y = df["corsika_event_header_total_energy"]

clf = sklearn.ensemble.RandomForestClassifier()
clf.fit(X, y)
```

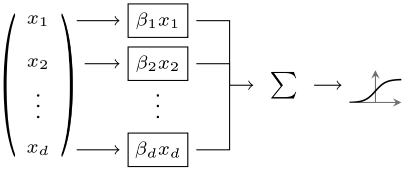
Structured Data



Logistic Regression:

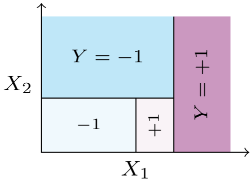
$$\widehat{\mathbb{P}}_\beta(Y = +1 \mid X = x) = \frac{e^{\langle \beta, x \rangle}}{1 + e^{\langle \beta, x \rangle}}$$

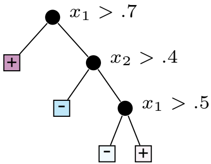
Structured Data



Logistic Regression:

$$\widehat{\mathbb{P}}_\beta(Y = +1 \mid X = x) \;=\; \frac{e^{\langle \beta, x \rangle}}{1 + e^{\langle \beta, x \rangle}}$$



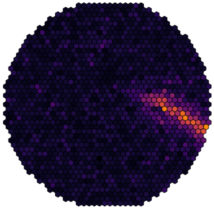


Decision Trees:

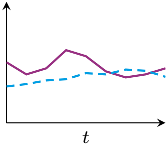
- recursively split $\mathcal{X}$
- boost performance through ensembling

These models perform very well 🎉 (if structure permits 😟)

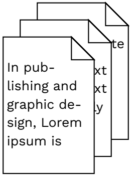
Unstructured Data



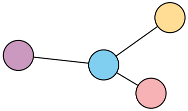
images



time series

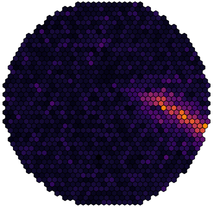


texts

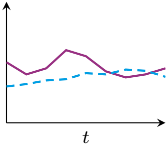


graphs

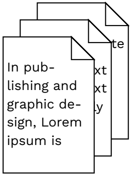
Unstructured Data



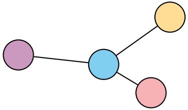
images



time series



texts

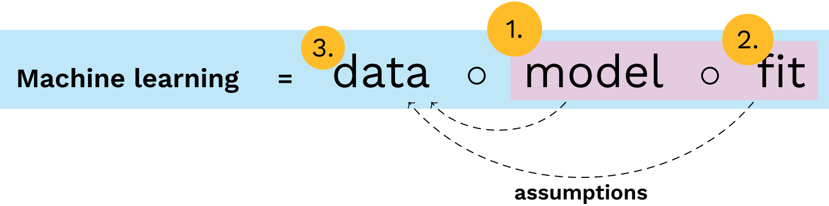


graphs

Deep Learning learns features as a part of the model 🚀

- no manual feature-engineering necessary 🙌
- instead, architecture optimization and more data are needed 😮‍💨

Agenda



1. Modeling
2. Fitting
3. Data and Assumptions
4. Concluding Remarks

+ Hands-On Exercises (Tue ~ 45 min, Thu ~ 90 min)

Modeling

Shallow Models

Polynomial Regression: $y = f_\beta(x) + \epsilon$, where $f_\beta(x) = \sum_{i=0}^{n} \langle \beta_i, x^i \rangle$

- $y \in \mathbb{R}$, $x \in \mathbb{R}^d$, and $\beta_i \in \mathbb{R}^d$
- $\langle a, b \rangle = \sum_{j=1}^{d} a_j \cdot b_j$ is the scalar product

Shallow Models

Polynomial Regression: $y = f_\beta(x) + \epsilon$, where $f_\beta(x) = \sum_{i=0}^{n} \langle \beta_i, x^i \rangle$

- $y \in \mathbb{R}$, $x \in \mathbb{R}^d$, and $\beta_i \in \mathbb{R}^d$
- $\langle a, b \rangle = \sum_{j=1}^{d} a_j \cdot b_j$ is the scalar product
- typical loss: $\mathcal{L}_{\mathrm{D}}(\beta) = \sum_{i=1}^{m} \big(y_i - f_\beta(x_i)\big)^2$ where $(x_i, y_i) \in \mathrm{D}$ and $D = \{(x_i, y_i) \in \mathcal{X} \times \mathcal{Y} : 1 \leq i \leq m\}$ is the training set

Shallow Models

Logistic Regression: $$\widehat{y} = \underset{i \in \{1,2,\ldots,C\}}{\arg\max} \; \widehat{\mathbb{P}}_\beta(Y = i \mid X = x)$$

Shallow Models

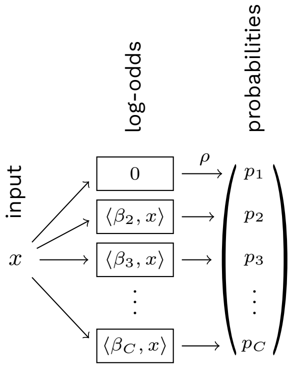
Logistic Regression: $$\widehat{y} = \underset{i \in \{1,2,\ldots,C\}}{\arg\max} \; \underbrace{\widehat{\mathbb{P}}_\beta(Y = i \mid X = x)}_{= \rho(\langle \beta_i, x \rangle)}$$

where $$\rho(v_i) = \begin{cases} \frac{1}{1+\sum_{j=2}^{k} e^{v_j}} & i = 1 \\ \frac{e^{v_i}}{1+\sum_{j=2}^{k} e^{v_j}} & i \in \{2, 3, \ldots, C\} \end{cases}$$

Shallow Models

Logistic Regression: $\widehat{y} = \underset{i \in \{1,2,\ldots,C\}}{\arg\max} \; \underbrace{\widehat{\mathbb{P}}_\beta(Y = i \mid X = x)}_{= \rho(\langle \beta_i, x \rangle)}$

where $\rho(v_i) = \begin{cases} \frac{1}{1+\sum_{j=2}^{k} e^{v_j}} & i = 1 \\ \frac{e^{v_i}}{1+\sum_{j=2}^{k} e^{v_j}} & i \in \{2, 3, \ldots, C\} \end{cases}$



The **soft-max** operation ρ projects to the unit simplex $\left\{p \in \mathbb{R}^C : \; p_i \geq 0, \; 1 = \sum_{i=1}^{C} p_i \right\}$

Shallow Models

Motivation: the Logistic Regression represents **linear models** of the **log-odds**.

$$\log \frac{\mathbb{P}(Y = 2 \mid X = x)}{\mathbb{P}(Y = 1 \mid X = x)} = \langle \beta_2, x \rangle + \epsilon \overset{?}{>} 0$$

$$\log \frac{\mathbb{P}(Y = 3 \mid X = x)}{\mathbb{P}(Y = 1 \mid X = x)} = \langle \beta_3, x \rangle + \epsilon \overset{?}{>} 0$$

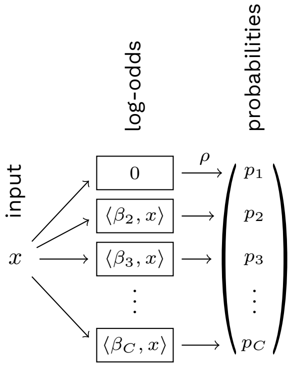
$$\dots$$

$$\log \frac{\mathbb{P}(Y = k \mid X = x)}{\mathbb{P}(Y = 1 \mid X = x)} = \langle \beta_C, x \rangle + \epsilon \overset{?}{>} 0$$

Shallow Models

Synopsis:

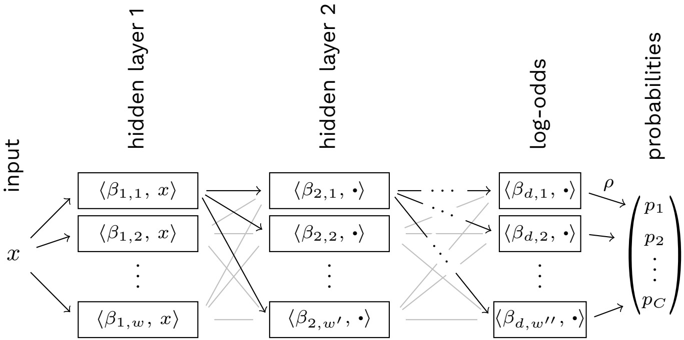
- **Polynomial Regression =**
  a linear model of exponentiated inputs x^i
- **Logistic Regression =**
  a linear model of the log-odds
- The **soft-max** operation maps these log-odds to (estimates of) class probabilities



Deep Networks

Deep Nets: use multiple (logistic regression-like) layers

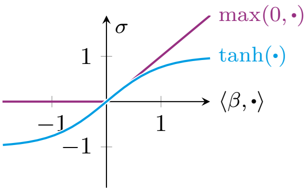
- learnable linear combinations $\langle \beta, \bullet \rangle$

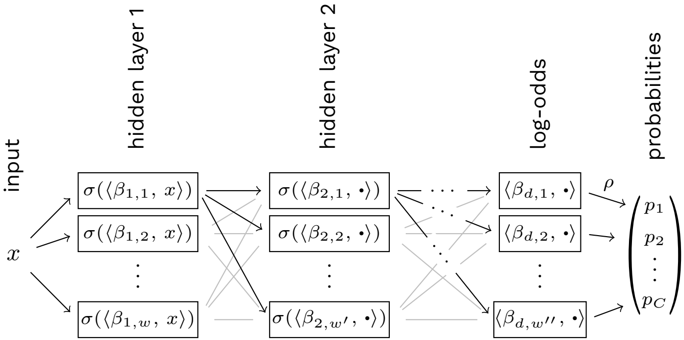


Deep Networks

Deep Nets: use multiple (logistic regression-like) layers

- learnable linear combinations $\langle \beta, \bullet \rangle$
- non-linear activations σ





Universal Approximation

Density: A family G of models can approximate any function $f \in C(\mathbb{R}^n)$, if $\forall\, \varepsilon > 0,$ compact $K \subseteq \mathbb{R}^n,$ $\exists\, g \in G,$ such that

$$\max_{x \in K} \left\| f(x) - g(x) \right\| \; < \; \varepsilon$$

[1] Pinkus, "Approximation theory of the MLP model in neural networks", 1999.
[2] Kidger and Lyons, "Universal approximation with deep narrow networks", 2020.

Universal Approximation

Density: A family G of models can approximate any function $f \in C(\mathbb{R}^n)$, if $\forall\, \varepsilon > 0,$ compact $K \subseteq \mathbb{R}^n,\ \exists\, g \in G,$ such that

$$\max_{x \in K} \left\| f(x) - g(x) \right\| \;<\; \varepsilon$$

- **One hidden layer of arbitrary width** is dense iff σ is non-polynomial.[1]
- **Arbitrarily deep nets with minimum width** $d + C + 2$ are dense.[2]

[1] Pinkus, “Approximation theory of the MLP model in neural networks”, 1999.
[2] Kidger and Lyons, “Universal approximation with deep narrow networks”, 2020.

Universal Approximation

Density: A family G of models can approximate any function $f \in C(\mathbb{R}^n)$, if $\forall\, \varepsilon > 0,$ compact $K \subseteq \mathbb{R}^n,\ \exists\, g \in G,$ such that

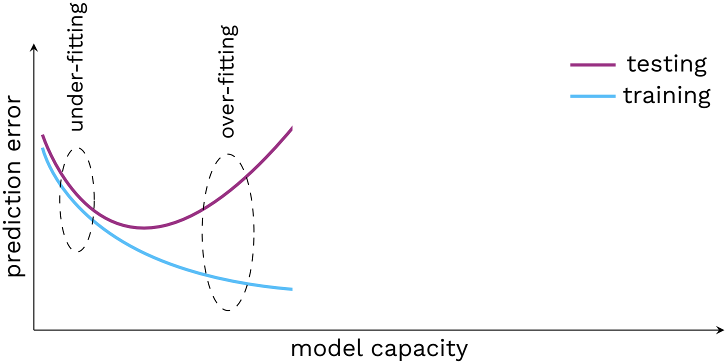
$$\max_{x \in K} \left\| f(x) - g(x) \right\| < \varepsilon$$

- **One hidden layer of arbitrary width** is dense iff σ is non-polynomial.[1]
- **Arbitrarily deep nets with minimum width** $d + C + 2$ are dense.[2]

- Deep nets are often more *efficient* approximators than wide shallow nets.
- Density does not imply the existence of a learning algorithm to select g from G

[1] Pinkus, "Approximation theory of the MLP model in neural networks", 1999.
[2] Kidger and Lyons, "Universal approximation with deep narrow networks", 2020.

Over- and Underfitting



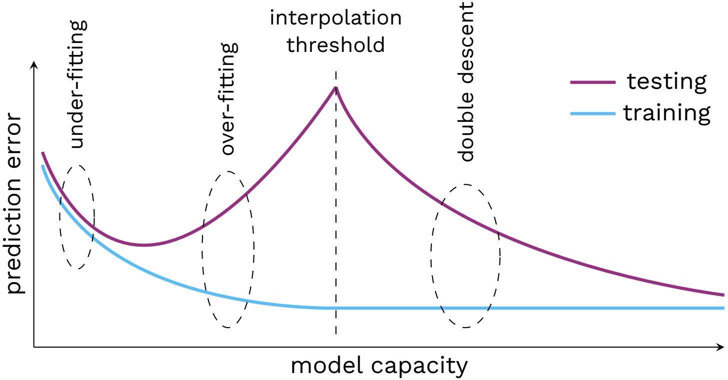
Under-Fitting:

- approximation
- high bias, low variance

Over-Fitting:

- memorization
- low bias, high variance

Over- and Underfitting



Under-Fitting:

- approximation
- high bias, low variance

Over-Fitting:

- memorization
- low bias, high variance

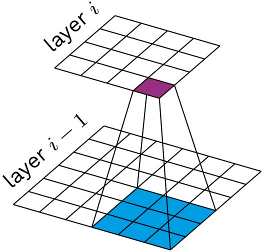
Double Descent:

- interpolation[3]

[3] Belkin et al., "Reconciling modern machine-learning practice and the classical bias-variance trade-off", 2019

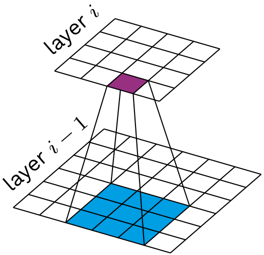
Inductive Biases

Convolution: $S(i,j) = (K * I)(i,j) = \sum_{m,n} I(i-m, j-n) \cdot K(m,n)$



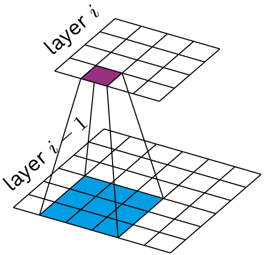
Inductive Biases

Convolution: $S(i, j) = (K * I)(i, j) = \sum_{m,\, n} I(i - m, j - n) \cdot K(m, n)$



Inductive Biases

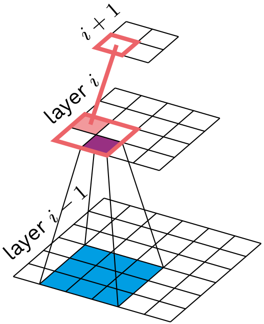
Convolution: $S(i,j) = (K * I)(i,j) = \sum_{m,n} I(i-m, j-n) \cdot K(m,n)$



Inductive Biases

Convolution: $$S(i,j) = (K * I)(i,j) = \sum_{m,\,n} I(i-m, j-n) \cdot K(m,n)$$

Pooling: only maintain the **maximum** of each neighborhood.

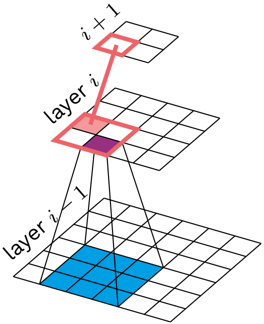


Inductive Biases

Convolution: $S(i,j) = (K * I)(i,j) = \sum_{m,n} I(i-m, j-n) \cdot K(m,n)$

Pooling: only maintain the **maximum** of each neighborhood.

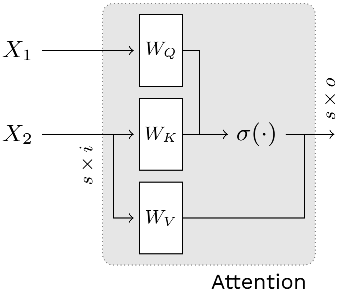
- translation invariance
- sparse interactions
- parameter sharing



In general, specialized layers are used to introduce **biases** that suit the data.

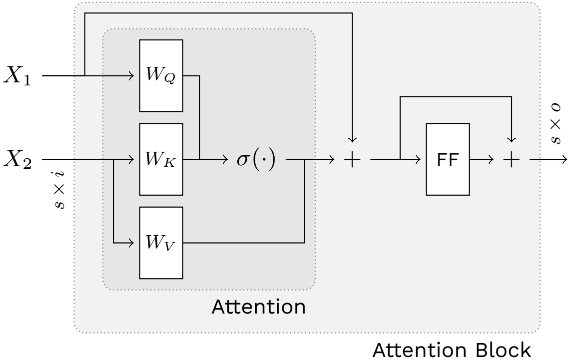
Inductive Biases

$\text{Attention}(X_1, X_2) = \sigma\Big((X_1 W_Q)(X_2 W_K)^\top\Big) X_2 W_V$ explicitly models interactions.



Inductive Biases

Attention$(X_1, X_2) \; = \; \sigma\Big((X_1 W_Q)(X_2 W_K)^\top\Big) X_2 W_V$ explicitly models interactions.



Modeling

Synopsis:

- **Deep Nets** use layers of increasingly abstract representations
- **Layers** consist of linear parameters and non-linear activations

Modeling

Synopsis:

- **Deep Nets** use layers of increasingly abstract representations
- **Layers** consist of linear parameters and non-linear activations
- **Model Capacity** should consider sample sizes (over-/under-fitting)
- **Inductive Biases** facilitate learning

Modeling

Synopsis:

- **Deep Nets** use layers of increasingly abstract representations
- **Layers** consist of linear parameters and non-linear activations
- **Model Capacity** should consider sample sizes (over-/under-fitting)
- **Inductive Biases** facilitate learning

Practical Recommendations:

- **Build on Existing Solutions** for similar problems

Modeling

Synopsis:

- **Deep Nets** use layers of increasingly abstract representations
- **Layers** consist of linear parameters and non-linear activations
- **Model Capacity** should consider sample sizes (over-/under-fitting)
- **Inductive Biases** facilitate learning

Practical Recommendations:

- **Build on Existing Solutions** for similar problems
- **Extensively Tune** the hyper-parameters (# layers, # features per layer, ...)

Modeling

Synopsis:

- **Deep Nets** use layers of increasingly abstract representations
- **Layers** consist of linear parameters and non-linear activations
- **Model Capacity** should consider sample sizes (over-/under-fitting)
- **Inductive Biases** facilitate learning

Practical Recommendations:

- **Build on Existing Solutions** for similar problems
- **Extensively Tune** the hyper-parameters (# layers, # features per layer, ...)
- **Assumptions > Depth** hence, prioritize baseline methods

Fitting

Empirical Risk Minimization

Notation:

- $h_\beta : \mathcal{X} \rightarrow \mathbb{R}^C$ is our model, parametrized by $\beta \in \mathbb{R}^B$ (fixed architecture)
- $\ell(h_\beta(x), y)$ measures the deviation between $h_\beta(x)$ and y

Empirical Risk Minimization

Notation:

- $h_\beta : \mathcal{X} \to \mathbb{R}^C$ is our model, parametrized by $\beta \in \mathbb{R}^B$ (fixed architecture)
- $\ell(h_\beta(x), y)$ measures the deviation between $h_\beta(x)$ and y

Ultimate Goal: minimize the *expected* risk:

$$R(h_\beta, \ell) \;=\; \mathbb{E}_{(x,y)\sim\mathbb{P}}\big(\ell(h_\beta(x), y)\big) \;=\; \int_{\mathcal{X}\times\mathcal{Y}} \mathbb{P}\big(X = x,\, Y = y\big) \cdot \ell\big(h_\beta(x), y\big)\; \mathrm{d}x\, \mathrm{d}y$$

Empirical Risk Minimization

Notation:

- $h_\beta : \mathcal{X} \to \mathbb{R}^C$ is our model, parametrized by $\beta \in \mathbb{R}^B$ (fixed architecture)
- $\ell(h_\beta(x), y)$ measures the deviation between $h_\beta(x)$ and y

Ultimate Goal: minimize the *expected* risk:

$$R(h_\beta, \ell) \;=\; \mathbb{E}_{(x,y)\sim\mathbb{P}}\big(\ell(h_\beta(x), y)\big) \;=\; \int_{\mathcal{X}\times\mathcal{Y}} \mathbb{P}\big(X = x,\, Y = y\big) \cdot \ell\big(h_\beta(x), y\big)\; \mathrm{d}x\, \mathrm{d}y$$

Approach: approximate $R(h_\beta, \ell)$ empirically with the training data D:

$$\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \;=\; \frac{1}{m} \sum_{i=1}^{m} \ell\big(h_\beta(x_i), y_i\big) \;\xrightarrow[m\to\infty]{}\; R(h_\beta, \ell)$$

and choose $\beta^* \;=\; \arg\min_{\beta\in\mathbb{R}^B} \widehat{R}_{\mathrm{D}}(h_\beta, \ell)$.

Loss Functions

Mean Squared Error: $\ell\big(h(x), y\big) \;=\; \Big\| h(x) - y \Big\|_2^2$

Cross Entropy / Logistic Loss: $\ell'\big(h(x), y\big) \;=\; -\sum_{i=1}^{C} \delta_{y=i} \log\big([h(x)]_i\big)$

Loss Functions

Mean Squared Error: $\ell\big(h(x), y\big) \;=\; \big\|h(x) - y\big\|_2^2$

Cross Entropy / Logistic Loss: $\ell'\big(h(x), y\big) \;=\; -\sum_{i=1}^{C} \delta_{y=i} \log\big([h(x)]_i\big)$

Proper Scoring Rule: any $\ell : \mathcal{Z} \times \mathcal{Y} \to \mathbb{R}$ for which $\arg\min_{h \in \mathcal{H}} R(h; \ell) \;=\; \mathbb{P}(Y \mid X)$.

- cross entropy is proven to be such a loss function
- hence, ERM with cross entropy readily learns $\mathbb{P}(Y \mid X)$ 🚀

Empirical Risk Minimization (Revisited)

Ultimate Goal: minimize the *expected* risk:

$$R(h_\beta, \ell) \;=\; \mathbb{E}_{(x,y)\sim\mathbb{P}}\big(\ell(h_\beta(x), y)\big)$$

ERM: approximate $R(h_\beta, \ell)$ empirically with the training data D:

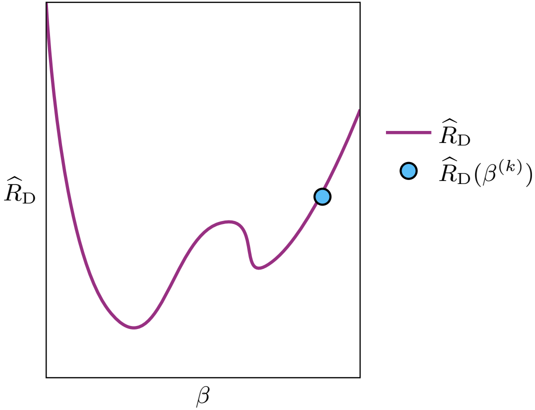
$$\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \;=\; \frac{1}{m}\sum_{i=1}^{m} \ell\big(h_\beta(x_i), y_i\big) \;\xrightarrow[m\to\infty]{}\; R(h_\beta, \ell)$$

and choose $\beta^* \;=\; \arg\min_{\beta\in\mathbb{R}^B} \widehat{R}_{\mathrm{D}}(h_\beta, \ell)$.

Stochastic First-Order Optimization

Ideas:

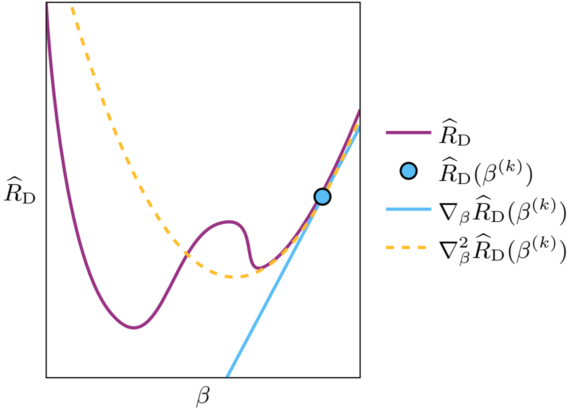
- $\widehat{R}_{\mathrm{D}}(h_\beta, \ell)$ is just a function to be minimized



Stochastic First-Order Optimization

Ideas:

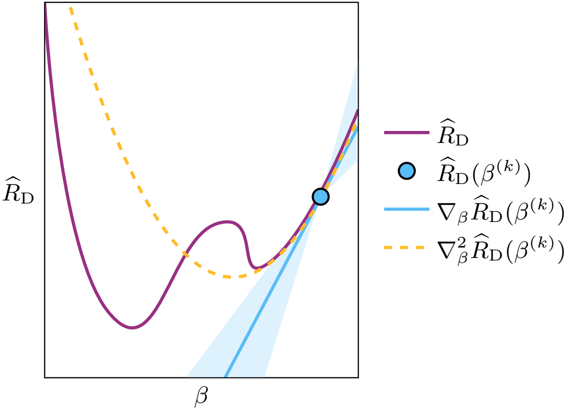
- $\widehat{R}_{\mathrm{D}}(h_\beta, \ell)$ is just a function to be minimized
- use gradient information to reduce $\widehat{R}_{\mathrm{D}}(h_\beta, \ell)$ until β^* is found.
- ignore higher-order derivatives to safe computation time.



Stochastic First-Order Optimization

Ideas:

- $\widehat{R}_{\mathrm{D}}(h_\beta, \ell)$ is just a function to be minimized
- use gradient information to reduce $\widehat{R}_{\mathrm{D}}(h_\beta, \ell)$ until β^* is found.
- ignore higher-order derivatives to safe computation time.
- introduce randomness into the gradients to improve convergence.



Stochastic First-Order Optimization

Stochastic Gradient Descent (SGD): in each step k, reduce the risk $\widehat{R}_{\mathrm{D}}(h_\beta, \ell)$ w.r.t. a *single, random* example.

$$\beta^{(k+1)} \leftarrow \beta^{(k)} - \alpha^{(k)} \nabla_\beta \, \ell\Big(h\big(x_{i^{(k)}}, \beta^{(k)}\big), y_{i^{(k)}}\Big) \quad \text{where} \quad \begin{cases} \beta^{(k)} & \text{the parameter vector of } h \\ \alpha^{(k)} & \text{the step size} \\ (x_{i^{(k)}}, y_{i^{(k)}}) & \text{the example} \end{cases}$$

Stochastic First-Order Optimization

Stochastic Gradient Descent (SGD): in each step k, reduce the risk $\widehat{R}_{\mathrm{D}}(h_\beta, \ell)$ w.r.t. a *single, random* example.

$$\beta^{(k+1)} \leftarrow \beta^{(k)} - \alpha^{(k)} \nabla_\beta \, \ell\Big(h\big(x_{i^{(k)}}, \beta^{(k)}\big), y_{i^{(k)}}\Big) \quad \text{where} \quad \begin{cases} \beta^{(k)} & \text{the parameter vector of } h \\ \alpha^{(k)} & \text{the step size} \\ (x_{i^{(k)}}, y_{i^{(k)}}) & \text{the example} \end{cases}$$

Full Gradient Descent (GD): in each step k, reduce $\widehat{R}_{\mathrm{D}}(h_\beta, \ell)$ w.r.t. *all examples*.

$$\beta^{(k+1)} \leftarrow \beta^{(k)} - \alpha^{(k)} \nabla_\beta \, \widehat{R}_{\mathrm{D}}(h_\beta, \ell) = \beta^{(k)} - \alpha^{(k)} \frac{1}{m} \sum_{i=1}^{m} \nabla_\beta \ell\Big(h\big(x_i, \beta^{(k)}\big), y_i\Big)$$

Stochastic First-Order Optimization

Convergence Rate[4]**:** worst-case # iterations, in which $\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \leq \widehat{R}_{\mathrm{D}}(h_{\beta^*}, \ell) + \epsilon$

4 Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning", 2018.

Stochastic First-Order Optimization

Convergence Rate[4]**:** worst-case # iterations, in which $\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \leq \widehat{R}_{\mathrm{D}}(h_{\beta^*}, \ell) + \epsilon$

- **GD:** $\propto m \cdot \log(\frac{1}{\epsilon})$
- **SGD:** $\propto \frac{1}{\epsilon}$ (independent of m)

[4] Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning", 2018.

Stochastic First-Order Optimization

Convergence Rate[4]**:** worst-case # iterations, in which $\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \leq \widehat{R}_{\mathrm{D}}(h_{\beta^*}, \ell) + \epsilon$

- **GD:** $\propto m \cdot \log(\frac{1}{\epsilon})$
- **SGD:** $\propto \frac{1}{\epsilon}$ (independent of m)
- For SGD, the same rate applies to $R(h_\beta, \ell)$ (independent of D if $m \gg k$) 🚀

Hence, SGD has an amazing performance for large data sets.

[4] Bottou, Curtis, and Nocedal, "Optimization Methods for Large-Scale Machine Learning", 2018.

Stochastic First-Order Optimization

Noise Reduction: use mini-batches instead of single examples,

$$\beta^{(k+1)} \leftarrow \beta^{(k)} - \alpha^{(k)} \frac{1}{b} \sum_{i=1}^{b} \nabla_\beta \ell\Big(h\big(x_{b_i}, \beta^{(k)}\big), y_{b_i}\Big). \quad \text{where} \quad b \ll m.$$

- smaller variance of update steps
- stepsize $\{\alpha^{(k)}\}$ is easier to tune
- most common approach for deep nets

Stochastic First-Order Optimization

Noise Reduction: use mini-batches instead of single examples,

$$\beta^{(k+1)} \leftarrow \beta^{(k)} - \alpha^{(k)} \frac{1}{b} \sum_{i=1}^{b} \nabla_\beta \ell\Big(h\big(x_{b_i}, \beta^{(k)}\big), y_{b_i}\Big). \quad \text{where} \quad b \ll m.$$

- smaller variance of update steps
- stepsize $\{\alpha^{(k)}\}$ is easier to tune
- most common approach for deep nets

Learning Rate Scheduling:

- even with mini-batches, noise can eventually prevent the reduction of $\widehat{R}_\mathrm{D}(h_\beta, \ell)$
- hence, decrease step sizes $\{\alpha^{(k)}\}$ over time

Stochastic First-Order Optimization

Momentum:

$$\beta^{(k+1)} \leftarrow \beta^{(k)} - g\left(\beta^{(k)}\right) + \gamma^{(k)} \cdot \left(\beta^{(k)} - \beta^{(k-1)}\right) \text{ where } \begin{cases} g(\beta^{(k)}) & \text{SGD, GD, or mini-batch gradient} \\ \gamma^{(k)} & \text{a weighting parameter} \end{cases}$$

Stochastic First-Order Optimization

Momentum:

$$\beta^{(k+1)} \leftarrow \beta^{(k)} - g\big(\beta^{(k)}\big) + \gamma^{(k)} \cdot \big(\beta^{(k)} - \beta^{(k-1)}\big) \text{ where } \begin{cases} g(\beta^{(k)}) & \text{SGD, GD, or mini-batch gradient} \\ \gamma^{(k)} & \text{a weighting parameter} \end{cases}$$

Accelerated Gradient a.k.a. Nesterov Momentum:

$$\beta^{(k+1)} \;\leftarrow\; \beta^{(k)} \;-\; g\Big(\beta^{(k)} + \gamma^{(k)} \cdot \big(\beta^{(k)} - \beta^{(k-1)}\big)\Big) \;+\; \gamma^{(k)} \cdot \big(\beta^{(k)} - \beta^{(k-1)}\big)$$

- momentum is applied before $g(\bullet)$
- GD: optimal convergence rate $\propto \frac{1}{\epsilon^2}$
- SGD: good practical performance but (theoretical) convergence rate is not improved
- even better: if combined with adaptive gradients $\rightarrow$ Adam

Backpropagation

Goal: compute $\nabla_\beta \ell(h(x_i, \beta), y_i)$ where

$$h(x_i, \beta) \;=\; \rho\Big(\big\langle \beta_d,\, \phi\big(\langle \beta_{d-1},\, \dots\, \phi(\langle \beta_1, x_i \rangle)\rangle\big)\big\rangle\Big)$$

$$x \longrightarrow \boxed{\sigma(\langle \beta_1,\, x\rangle)} \longrightarrow \boxed{\sigma(\langle \beta_2,\, \bullet\rangle)} \;\text{——}\; \cdots \longrightarrow \boxed{\rho(\langle \beta_d,\, \bullet\rangle)} \longrightarrow h(x)$$

Backpropagation

Goal: compute $\nabla_\beta \ell(h(x_i, \beta), y_i)$ where

$$h(x_i, \beta) \;=\; \rho\Big(\big\langle \beta_d,\, \phi\big(\langle \beta_{d-1},\, \ldots\, \phi(\langle \beta_1, x_i \rangle)\,\rangle\big)\,\big\rangle\Big)$$

$$x \longrightarrow \boxed{\sigma(\langle \beta_1,\, x \rangle)} \longrightarrow \boxed{\sigma(\langle \beta_2,\, \bullet \rangle)} \;—\; \cdots \longrightarrow \boxed{\rho(\langle \beta_d,\, \bullet \rangle)} \longrightarrow h(x)$$

Chain rule of calculus: $\dfrac{\partial f(g(x))}{\partial x} = \dfrac{\partial f(g)}{\partial g} \dfrac{\partial g(x)}{\partial x}$

Backpropagation

Goal: compute $\nabla_\beta \ell(h(x_i, \beta), y_i)$ where

$$h(x_i, \beta) \;=\; \rho\Big(\big\langle \beta_d,\, \phi\big(\langle \beta_{d-1},\, \dots\, \phi(\langle \beta_1, x_i \rangle)\rangle\big)\big\rangle\Big)$$

$$x \longrightarrow \boxed{\sigma(\langle \beta_1,\, x \rangle)} \longrightarrow \boxed{\sigma(\langle \beta_2,\, \bullet \rangle)} \longrightarrow \cdots \longrightarrow \boxed{\rho(\langle \beta_d,\, \bullet \rangle)} \longrightarrow h(x)$$

Chain rule of calculus: $$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(g)}{\partial g} \frac{\partial g(x)}{\partial x}$$

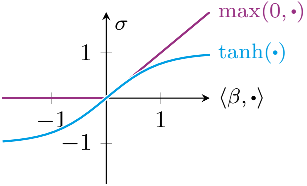
Automatic Differentiation: each function $f(x)$ also implements its gradient

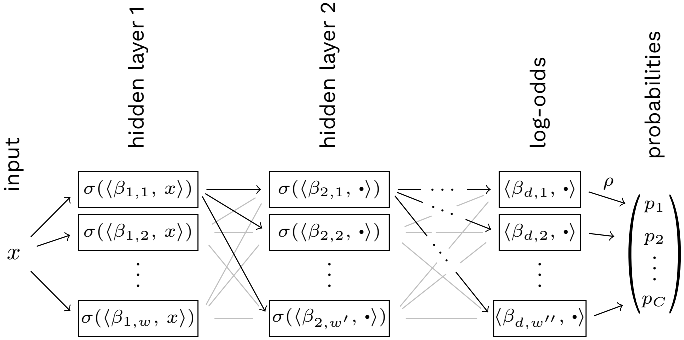
$$\nabla_x f(x) = (\frac{\partial f(x)}{\partial x_1}, \dots \frac{\partial f(x)}{\partial x_n})^\top$$

Deep Networks

Deep Nets: use multiple (logistic regression-like) layers

- learnable linear combinations $\langle \beta, \bullet \rangle$
- non-linear activations σ





Stochastic First-Order Optimization

Synopsis:

- **ERM:** we minimize $\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \xrightarrow[m \to \infty]{} R(h_\beta, \ell)$
- **SGD:** gradients randomized through sampling converge quickly for large m

Stochastic First-Order Optimization

Synopsis:

- **ERM:** we minimize $\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \xrightarrow[m \to \infty]{} R(h_\beta, \ell)$
- **SGD:** gradients randomized through sampling converge quickly for large m
- **Mini-Batching**: common practice to reduce SGD gradient noise
- **LR Scheduling**: common practice to balance the noise
- **Nesterov Momentum**: can improve convergence

Stochastic First-Order Optimization

Synopsis:

- **ERM:** we minimize $\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \xrightarrow[m \to \infty]{} R(h_\beta, \ell)$
- **SGD:** gradients randomized through sampling converge quickly for large m
- **Mini-Batching**: common practice to reduce SGD gradient noise
- **LR Scheduling**: common practice to balance the noise
- **Nesterov Momentum**: can improve convergence

Practical Recommendations:

- **Carefully Design Loss Functions** to reflect your goals

Stochastic First-Order Optimization

Synopsis:

- **ERM:** we minimize $\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \xrightarrow[m \to \infty]{} R(h_\beta, \ell)$
- **SGD:** gradients randomized through sampling converge quickly for large m
- **Mini-Batching**: common practice to reduce SGD gradient noise
- **LR Scheduling**: common practice to balance the noise
- **Nesterov Momentum**: can improve convergence

Practical Recommendations:

- **Carefully Design Loss Functions** to reflect your goals
- **Use Popular First-Order Methods** like Adam or SGD with Nesterov Momentum

Data and Assumptions

A Premature Conclusion

Machine learning = data ∘ model ∘ fit

What we have learned:

- Deep Nets are universal function approximators
- Customized loss functions let them learn what we need
- We know effective ways of optimizing them

A Premature Conclusion

Machine learning = data ∘ model ∘ fit

What we have learned:

- Deep Nets are universal function approximators
- Customized loss functions let them learn what we need
- We know effective ways of optimizing them

What could possibly go wrong? 💣

Learning Assumptions

Recall that we approximate

$$R(h_\beta, \ell) \;=\; \mathbb{E}_{(x,y)\sim\mathbb{P}}\Big(\ell(h_\beta(x), y)\Big)$$

through

$$\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \;=\; \frac{1}{m}\sum_{i=1}^{m} \ell\big(h_\beta(x_i), y_i\big)$$

Learning Assumptions

Recall that we approximate

$$R(h_\beta, \ell) \;=\; \mathbb{E}_{(x,y)\sim\mathbb{P}}\Big(\ell(h_\beta(x), y)\Big)$$

through

$$\widehat{R}_{\mathrm{D}}(h_\beta, \ell) \;=\; \frac{1}{m}\sum_{i=1}^{m} \ell\big(h_\beta(x_i), y_i\big)$$

Independent and Identical Distribution (IID) Assumption:

$$(x, y) \sim \mathbb{P} \quad \forall \;\; (x, y) \in \mathrm{D} \cup \mathrm{D}_{\mathrm{test}}$$

Learning Assumptions

Recall that we approximate

$$R(h_\beta, \ell) \;=\; \mathbb{E}_{(x,y)\sim\mathbb{P}}\big(\ell(h_\beta(x), y)\big)$$

through

Data Set Shift breaks the IID assumption 💣

- $\mathrm{D} \sim \mathbb{P}_\mathcal{S}$ (e.g., a *simulation*)
- $\mathrm{D}_\mathrm{test} \sim \mathbb{P}_\mathcal{T}$ (e.g., a *real detector*)
- $\mathbb{P}_\mathcal{S} \neq \mathbb{P}_\mathcal{T}$

$$(x, y) \sim \mathbb{P} \quad \forall \; (x, y) \in \mathrm{D} \cup \mathrm{D}_\mathrm{test}$$

Types of Data Set Shift[5]

Recognize that $\mathbb{P}(X, Y) \quad = \quad \mathbb{P}(X \mid Y) \cdot \mathbb{P}(Y)$

[5] Kull and Flach, “Patterns of dataset shift”, 2014

Types of Data Set Shift[5]

Recognize that $\mathbb{P}(X, Y) = \mathbb{P}(X \mid Y) \cdot \mathbb{P}(Y)$

Label Shift:

$\mathbb{P}_{\mathcal{S}}(X \mid Y) = \mathbb{P}_{\mathcal{T}}(X \mid Y)$

$\mathbb{P}_{\mathcal{S}}(Y) \neq \mathbb{P}_{\mathcal{T}}(Y)$

[5] Kull and Flach, “Patterns of dataset shift”, 2014

Types of Data Set Shift[5]

Recognize that $\mathbb{P}(X, Y) = \mathbb{P}(X \mid Y) \cdot \mathbb{P}(Y)$

Label Shift:

$$\mathbb{P}_{\mathcal{S}}(X \mid Y) = \mathbb{P}_{\mathcal{T}}(X \mid Y)$$

$$\mathbb{P}_{\mathcal{S}}(Y) \neq \mathbb{P}_{\mathcal{T}}(Y)$$

Concept Shift:

$$\mathbb{P}_{\mathcal{S}}(Y) = \mathbb{P}_{\mathcal{T}}(Y)$$

$$\mathbb{P}_{\mathcal{S}}(X \mid Y) \neq \mathbb{P}_{\mathcal{T}}(X \mid Y)$$

[5] Kull and Flach, “Patterns of dataset shift”, 2014

Types of Data Set Shift[5]

Recognize that

$$\mathbb{P}(X, Y) = \mathbb{P}(X \mid Y) \cdot \mathbb{P}(Y)$$
$$= \mathbb{P}(X) \cdot \mathbb{P}(Y \mid X)$$

Label Shift:

$$\mathbb{P}_{\mathcal{S}}(X \mid Y) = \mathbb{P}_{\mathcal{T}}(X \mid Y)$$

$$\mathbb{P}_{\mathcal{S}}(Y) \neq \mathbb{P}_{\mathcal{T}}(Y)$$

Concept Shift:

$$\mathbb{P}_{\mathcal{S}}(Y) = \mathbb{P}_{\mathcal{T}}(Y)$$

$$\mathbb{P}_{\mathcal{S}}(X \mid Y) \neq \mathbb{P}_{\mathcal{T}}(X \mid Y)$$

[5] Kull and Flach, “Patterns of dataset shift”, 2014

Types of Data Set Shift[5]

Recognize that
$$\mathbb{P}(X, Y) = \mathbb{P}(X \mid Y) \cdot \mathbb{P}(Y)$$
$$= \mathbb{P}(X) \cdot \mathbb{P}(Y \mid X)$$

Label Shift:

$\mathbb{P}_{\mathcal{S}}(X \mid Y) = \mathbb{P}_{\mathcal{T}}(X \mid Y)$

$\mathbb{P}_{\mathcal{S}}(Y) \neq \mathbb{P}_{\mathcal{T}}(Y)$

Concept Shift:

$\mathbb{P}_{\mathcal{S}}(Y) = \mathbb{P}_{\mathcal{T}}(Y)$

$\mathbb{P}_{\mathcal{S}}(X \mid Y) \neq \mathbb{P}_{\mathcal{T}}(X \mid Y)$

(Also) Concept Shift:

$\mathbb{P}_{\mathcal{S}}(X) = \mathbb{P}_{\mathcal{T}}(X)$

$\mathbb{P}_{\mathcal{S}}(Y \mid X) \neq \mathbb{P}_{\mathcal{T}}(Y \mid X)$

[5] Kull and Flach, “Patterns of dataset shift”, 2014

Types of Data Set Shift[5]

Recognize that
$$\mathbb{P}(X,Y) = \mathbb{P}(X \mid Y) \cdot \mathbb{P}(Y)$$
$$= \mathbb{P}(X) \cdot \mathbb{P}(Y \mid X)$$

Label Shift:

$\mathbb{P}_{\mathcal{S}}(X \mid Y) = \mathbb{P}_{\mathcal{T}}(X \mid Y)$

$\mathbb{P}_{\mathcal{S}}(Y) \neq \mathbb{P}_{\mathcal{T}}(Y)$

Concept Shift:

$\mathbb{P}_{\mathcal{S}}(Y) = \mathbb{P}_{\mathcal{T}}(Y)$

$\mathbb{P}_{\mathcal{S}}(X \mid Y) \neq \mathbb{P}_{\mathcal{T}}(X \mid Y)$

(Also) Concept Shift:

$\mathbb{P}_{\mathcal{S}}(X) = \mathbb{P}_{\mathcal{T}}(X)$

$\mathbb{P}_{\mathcal{S}}(Y \mid X) \neq \mathbb{P}_{\mathcal{T}}(Y \mid X)$

Covariate Shift:

$\mathbb{P}_{\mathcal{S}}(Y \mid X) = \mathbb{P}_{\mathcal{T}}(Y \mid X)$

$\mathbb{P}_{\mathcal{S}}(X) \neq \mathbb{P}_{\mathcal{T}}(X)$

Correction Methods are available for each type, but require extra information (additional data, more assumptions, ...) 🔧

[5] Kull and Flach, “Patterns of dataset shift”, 2014

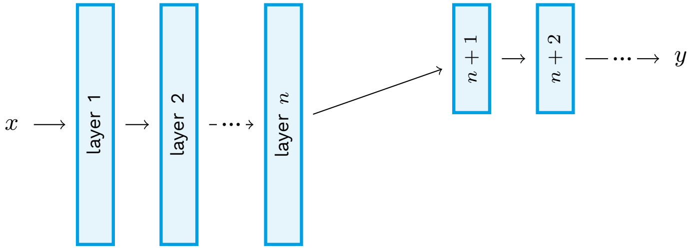
Domain-Adversarial Unsupervised Domain Adaptation[6]

- **Assume Concept Shift** $\mathbb{P}_{\mathcal{S}}(X \mid Y) \neq \mathbb{P}_{\mathcal{T}}(X \mid Y)$ and $\mathbb{P}_{\mathcal{S}}(Y) = \mathbb{P}_{\mathcal{T}}(Y)$
- **Employ Unlabeled Data** $\mathrm{D}_{\mathcal{T}} = \{x \sim \mathbb{P}_{\mathcal{T}}(X)\}$

[6] Ganin et al., "Domain-Adversarial Training of Neural Networks", 2016

Domain-Adversarial Unsupervised Domain Adaptation[6]

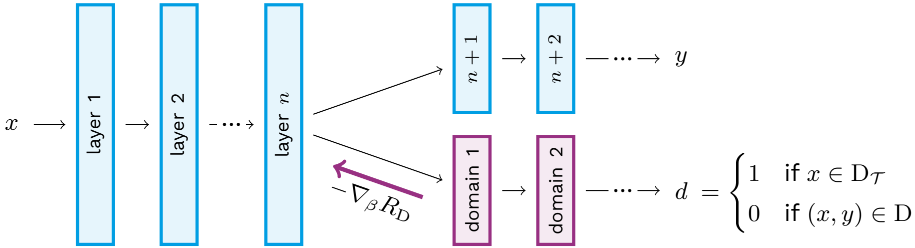
- **Assume Concept Shift** $\mathbb{P}_{\mathcal{S}}(X \mid Y) \neq \mathbb{P}_{\mathcal{T}}(X \mid Y)$ and $\mathbb{P}_{\mathcal{S}}(Y) = \mathbb{P}_{\mathcal{T}}(Y)$
- **Employ Unlabeled Data** $\mathrm{D}_{\mathcal{T}} = \left\{x \sim \mathbb{P}_{\mathcal{T}}(X)\right\}$



[6] Ganin et al., “Domain-Adversarial Training of Neural Networks”, 2016

Domain-Adversarial Unsupervised Domain Adaptation[6]

- **Assume Concept Shift** $\mathbb{P}_{\mathcal{S}}(X \mid Y) \neq \mathbb{P}_{\mathcal{T}}(X \mid Y)$ and $\mathbb{P}_{\mathcal{S}}(Y) = \mathbb{P}_{\mathcal{T}}(Y)$
- **Employ Unlabeled Data** $\mathrm{D}_{\mathcal{T}} = \{x \sim \mathbb{P}_{\mathcal{T}}(X)\}$



[6] Ganin et al., “Domain-Adversarial Training of Neural Networks”, 2016

Class-Conditional Label Noise[7]

Label Noise:

- **Training Labels** $\widehat{y}$ are randomly flipped versions of the ground-truth y
- **Assumptions** about the flipping process $y \to \widehat{y}$ are required

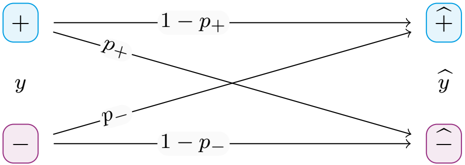
[7] Menon et al., “Learning from Corrupted Binary Labels via Class-Probability Estimation”, 2015

Class-Conditional Label Noise[7]

Label Noise:

- **Training Labels** $\widehat{y}$ are randomly flipped versions of the ground-truth y
- **Assumptions** about the flipping process $y \to \widehat{y}$ are required

Class-Conditional Noise: $\mathbb{P}(Y = +1 \mid X = x) \;=\; a \cdot \mathbb{P}(\widehat{Y} = +1 \mid X = x) + b$



[7] Menon et al., "Learning from Corrupted Binary Labels via Class-Probability Estimation", 2015

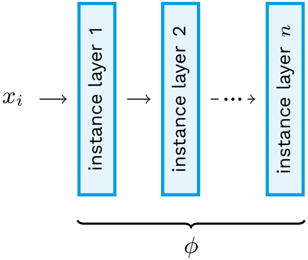
Deep Sets[8]

- **Each instance is a set** $\{x_i \in \mathcal{X} : 1 \leq i \leq m\}$ of variable size m
- $\mathcal{Y}$ are properties of such sets

[8] Zaheer et al., “Deep sets”, 2017

Deep Sets[8]

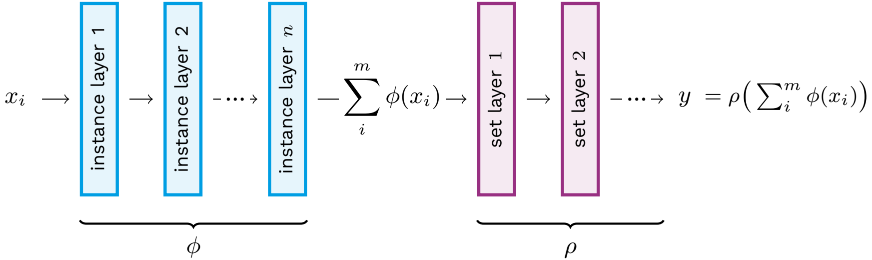
- **Each instance is a set** $\{x_i \in \mathcal{X} : 1 \leq i \leq m\}$ of variable size m
- $\mathcal{Y}$ are properties of such sets



[8] Zaheer et al., “Deep sets”, 2017

Deep Sets[8]

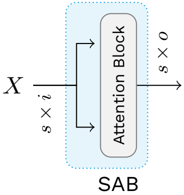
- **Each instance is a set** $\{x_i \in \mathcal{X} : 1 \leq i \leq m\}$ of variable size m
- $\mathcal{Y}$ are properties of such sets

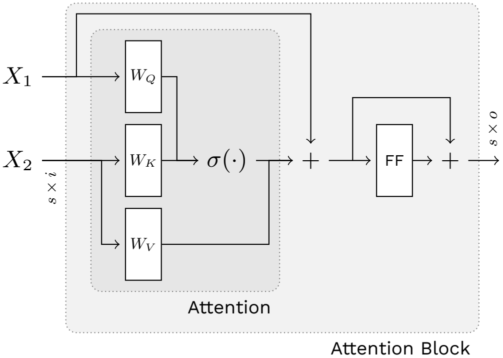


[8] Zaheer et al., “Deep sets”, 2017

Set Transformers[9]

Self-Attention Block:

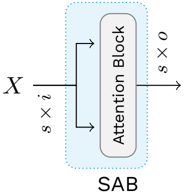




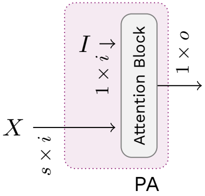
[9] Lee et al., "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks", 2019

Set Transformers[9]

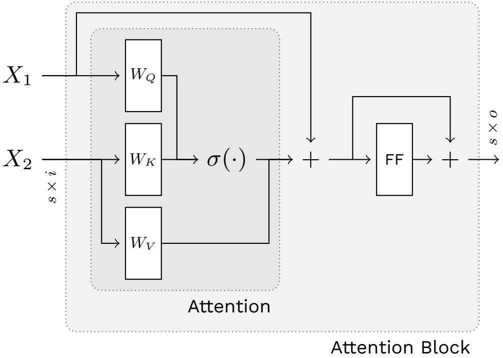
Self-Attention Block:



Pooling by Attention:



Set Encoder: SAB ∘ ... ∘ SAB ∘ PA



[9] Lee et al., "Set Transformer: A Framework for Attention-based Permutation-Invariant Neural Networks", 2019

Concluding Remarks

Should I Use Neural Networks?

Architecture Search vs feature engineering

Scale great for big data (but not for small data)

GPUs required as well as computation time for fitting

Implementing Neural Networks

JAX, PyTorch, Tensorflow, or Keras?

- Keras, Tensorflow: established solutions
- PyTorch, JAX: maximum flexibility

Implementing Neural Networks

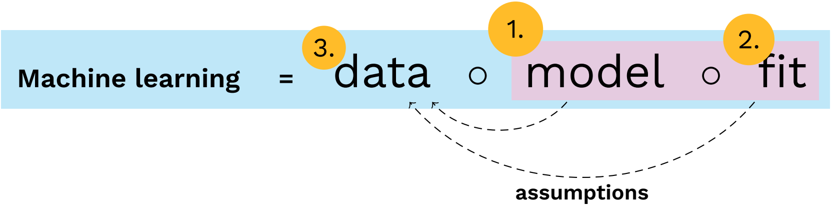
JAX, PyTorch, Tensorflow, or Keras?

- Keras, Tensorflow: established solutions
- PyTorch, JAX: maximum flexibility

JAX:

- JIT compilation speedups
- API identical to Numpy/Scipy
- Clean functional programming style (clarity, separation of concerns)
- Evolving eco-system and fewer solutions

Agenda



1. Modeling
2. Fitting
3. Data and Assumptions
4. Concluding Remarks

+ Hands-On Exercises (Tue ~ 45 min, Thu ~ 90 min)

Hands-On Exercises

https://git.e5.physik.tu-dortmund.de/qfuehring/ml_intro_handson