Common Core of Statistical Learning

Understanding Machine Learning

October 2022

Felix Wick

General Recipe

statistical learning algorithm by combining:

- **model** (e.g., linear function)
- **objective function** (e.g., squared residuals)
- **optimization algorithm** (e.g., gradient descent)

Consider Supervised Learning Scenario

map input to output: $y = f(\boldsymbol{x})$ (estimated: $\hat{f}(\boldsymbol{x})$)

random variables Y and $\boldsymbol{X} = (X_1, X_2, \cdots, X_p)$ ← usually high-dimensional

curve fitting / parameter estimation:

fit train data set of (y_i, $\boldsymbol{x}_i$) pairs → minimization of cost function

consider discriminative models:

- predict conditional density function $p(y|\boldsymbol{x}_i)$

  (as opposed to generative models predicting $p(y, \boldsymbol{x})$)
- often just conditional moment of $p(y|\boldsymbol{x}_i)$, e.g., conditional mean $E[Y|\boldsymbol{X} = \boldsymbol{x}_i]$

Model

Model in Linear Regression

fit:

$$y_i = \overbrace{\hat{\alpha} + \sum_{j=1}^{p} \hat{\beta}_j \, x_{ij}}^{\hat{f}(\boldsymbol{x}_i)} + \varepsilon_i$$

to be estimated:

- $\hat{\alpha}, \widehat{\boldsymbol{\beta}}$

→ $\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \hat{f}(\boldsymbol{x}_i) \right)^2$

(approximating assumed true $\alpha, \boldsymbol{\beta}$, σ)

predict:

$$\hat{y}_i = E[Y | \boldsymbol{X} = \boldsymbol{x}_i] = \hat{f}(\boldsymbol{x}_i)$$

$$p(y | \boldsymbol{x}_i) = \mathcal{N}(y; \hat{y}_i, \hat{\sigma}^2)$$

Gaussian

mean

variance (reflected by ε_i in fit)

Model Ingredients

- underlying probability distribution of Y
- approximated functional form $\hat{f}(\boldsymbol{x})$ → depends on used ML algorithm

$$p(y|\boldsymbol{x}_i) = \mathrm{PDF}\left(y; \hat{f}(\boldsymbol{x}_i), \ldots\right)$$

assumed Probability Density Function of target

$\hat{f}(\boldsymbol{x})$ often estimation of location parameter

other parameters (e.g., scale parameter) often not predicted per sample (assume homoscedasticity)

Principle of Maximum Entropy

How to choose the right model, i.e., the assumed underlying probability distribution of Y generating the observed data?

Occam's razor: pick distribution with fewest assumptions

$\rightarrow$ distribution with maximum entropy under given range and moment constraints

discrete: $H(X) = -\sum p_k \log p_k$, continuous: $H(X) = -\int p(x) \log p(x) dx$

($\rightarrow$ bulk of entropy from tails of distributions)

examples for maximum entropy distributions:

- normal distribution: continuous on $(-\infty, \infty)$ with particular mean and variance
- Poisson distribution: discrete on $[0, \infty)$ with particular mean

Objective Function

Loss Function

loss function L: expressing deviation between prediction and target

$L\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{\theta}}\right)$

with $\widehat{\boldsymbol{\theta}}$ corresponding to parameters of model $\hat{f}(\boldsymbol{x})$

e.g., $\widehat{\alpha}, \widehat{\boldsymbol{\beta}}$ in linear regression

e.g., squared residuals (for regression problems):

$L\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{\theta}}\right) = \left(y_i - \hat{f}\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\theta}}\right)\right)^2$

Cost Function

averaging losses over (empirical) training data set:

$$J(\widehat{\boldsymbol{\theta}}) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{\theta}}\right)$$

cost function to be minimized according to model parameters $\widehat{\boldsymbol{\theta}}$

→ objective function

Optimization

Cost Minimization

minimize training costs $J\left(\widehat{\boldsymbol{\theta}}\right)$ according to model parameters $\widehat{\boldsymbol{\theta}}$:

$$\nabla_{\widehat{\boldsymbol{\theta}}}\, J\left(\widehat{\boldsymbol{\theta}}\right) = 0$$

for mean squared error:

$$\nabla_{\widehat{\boldsymbol{\theta}}}\, \frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}\left(\boldsymbol{x}_i; \widehat{\boldsymbol{\theta}}\right)\right)^2 = 0$$

Ordinary Least Squares

linear regression: $\hat{f}(\boldsymbol{x}_i) = \hat{\alpha} + \sum_{j=1}^{p} \hat{\beta}_j \, x_{ij}$

matrix notation (drop intercept α for simplicity here): $\hat{\boldsymbol{y}} = \mathbf{X} \, \widehat{\boldsymbol{\beta}}$

solve normal equations for training data set:
$\nabla_{\widehat{\boldsymbol{\beta}}} \left(\boldsymbol{y} - \mathbf{X} \, \widehat{\boldsymbol{\beta}}\right)^{\mathrm{T}} \left(\boldsymbol{y} - \mathbf{X} \, \widehat{\boldsymbol{\beta}}\right) = 0$

$\rightarrow \widehat{\boldsymbol{\beta}} = \left(\mathbf{X}^{\mathrm{T}}\mathbf{X}\right)^{-1} \mathbf{X}^{\mathrm{T}} \boldsymbol{y}$

Maximum Likelihood Estimation

Likelihood

likelihood function $\mathcal{L}$:

- value of predicted probability density function (model) at observed target
- as function of model parameters $\widehat{\boldsymbol{\theta}}$

$$\mathcal{L}(\widehat{\boldsymbol{\theta}}; \boldsymbol{y}|\boldsymbol{x}) = p(\boldsymbol{y}|\boldsymbol{x}; \widehat{\boldsymbol{\theta}})$$

a little bit confusing:

$p(y|\boldsymbol{x}; \widehat{\boldsymbol{\theta}})$ is conditional density function if function of data with $\widehat{\boldsymbol{\theta}}$ fixed

$p(y|\boldsymbol{x}; \widehat{\boldsymbol{\theta}})$ is conditional likelihood function if function of $\widehat{\boldsymbol{\theta}}$ with data fixed

Independence Assumption

consider training data set (of size n) as one sample from unknown joint probability distribution of n independent (i.i.d.) sets of random variables $(Y_1, \boldsymbol{X}_1), (Y_2, \boldsymbol{X}_2), \dots, (Y_n, \boldsymbol{X}_n)$

(same thing as random sampling n times from $(Y, \boldsymbol{X})$)

→ likelihood as product of univariate probability density functions:

$$\mathcal{L}(\widehat{\boldsymbol{\theta}}; \boldsymbol{y}|\boldsymbol{x}) = \prod_{i=1}^{n} p(y_i|\boldsymbol{x}_i; \widehat{\boldsymbol{\theta}})$$

… to be maximized according to estimated model parameters $\widehat{\boldsymbol{\theta}}$

(known as maximum likelihood estimation)

Negative Log-Likelihood

model, e.g., for linear regression:
$p(y_i|\boldsymbol{x}_i) = \mathcal{N}\left(y; \hat{\alpha} + \boldsymbol{x}_i\, \widehat{\boldsymbol{\beta}}, \sigma^2\right)$

- with $\boldsymbol{y}, \mathbf{X}$ given in training
- $\widehat{\boldsymbol{\theta}} = (\hat{\alpha}, \widehat{\boldsymbol{\beta}}, \hat{\sigma}^2)$ to be estimated

logarithmic transformation of $\mathcal{L}$: log-likelihood function

$$\ell(\widehat{\boldsymbol{\theta}}; \boldsymbol{y}|\boldsymbol{x}) = \ln\left(\mathcal{L}(\widehat{\boldsymbol{\theta}}; \boldsymbol{y}|\boldsymbol{x})\right) = \sum_{i=1}^{n} \ln\left(p(y_i|\boldsymbol{x}_i; \widehat{\boldsymbol{\theta}})\right)$$

- logarithm monotonic function → maximum of ℓ and $\mathcal{L}$ at same $\widehat{\boldsymbol{\theta}}$ values
- most probability distributions (e.g., exponential family) only logarithmically concave (important for optimization)
- sum computationally more convenient than product

negative log-likelihood: minimization instead of maximization
$\widehat{\boldsymbol{\theta}} = \operatorname{argmax}_{\widehat{\boldsymbol{\theta}}}\, \ell(\widehat{\boldsymbol{\theta}}; \boldsymbol{y}\,|\boldsymbol{x}) = \operatorname{argmin}_{\widehat{\boldsymbol{\theta}}}\, \left(-\ell(\widehat{\boldsymbol{\theta}}; \boldsymbol{y}\,|\boldsymbol{x})\right)$

Maximum Likelihood

least squares method corresponds to maximum likelihood estimation with **Gaussian model**: $\sim e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$

estimate parameters $\widehat{\boldsymbol{\theta}} = (\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_k)$ solving

$$\widehat{\boldsymbol{\theta}} = \mathrm{argmax}_{\widehat{\boldsymbol{\theta}}}\ \ell(\widehat{\boldsymbol{\theta}}; \boldsymbol{y}\ | \boldsymbol{x})$$

→ likelihood equations (if ℓ differentiable in $\widehat{\boldsymbol{\theta}}$):

$$\frac{\partial \ell}{\partial \hat{\theta}_1} = 0,\ \frac{\partial \ell}{\partial \hat{\theta}_2} = 0,\ \cdots,\ \frac{\partial \ell}{\partial \hat{\theta}_k} = 0$$

(in short: $\nabla_{\widehat{\boldsymbol{\theta}}}\ \ell = 0$)

can be solved explicitly for some cases (e.g., ordinary least squares for linear regression)

but usually no closed-form solution
→ need for numerical optimization (e.g., gradient descent)

Interpretation of Maximum Likelihood

quantification of difference between two probability distributions

maximum likelihood estimation corresponds to minimization of Kullback-Leibler divergence (as well as cross entropy) between true data-generating probability distribution (manifested by empirical distribution of training data) and probability distribution of model:

$$\operatorname{argmin}_{\hat{\boldsymbol{\theta}}} D_{KL}\left(p_{\text{model}}(y|\boldsymbol{x};\hat{\boldsymbol{\theta}}) \,||\, p_{\text{data}}(y|\boldsymbol{x})\right) = \operatorname{argmin}_{\hat{\boldsymbol{\theta}}} \int p_{\text{data}}(y|\boldsymbol{x}) \, \log \frac{p_{\text{data}}(y|\boldsymbol{x})}{p_{\text{model}}(y|\boldsymbol{x};\hat{\boldsymbol{\theta}})} \, dy$$

$$\underset{E[\dots]}{=} \operatorname{argmin}_{\hat{\boldsymbol{\theta}}} \frac{1}{n} \sum_{i=1}^{n} \log \frac{p_{\text{data}}(y_i|\boldsymbol{x}_i)}{p_{\text{model}}(y_i|\boldsymbol{x}_i;\hat{\boldsymbol{\theta}})} \underset{\text{no contribution from } p_{\text{data}}}{=} \operatorname{argmax}_{\hat{\boldsymbol{\theta}}} \sum_{i=1}^{n} \ln\left(p_{\text{model}}(y_i|\boldsymbol{x}_i;\hat{\boldsymbol{\theta}})\right) = \operatorname{argmax}_{\hat{\boldsymbol{\theta}}} \ell(\hat{\boldsymbol{\theta}};\boldsymbol{y}\,|\boldsymbol{x})$$

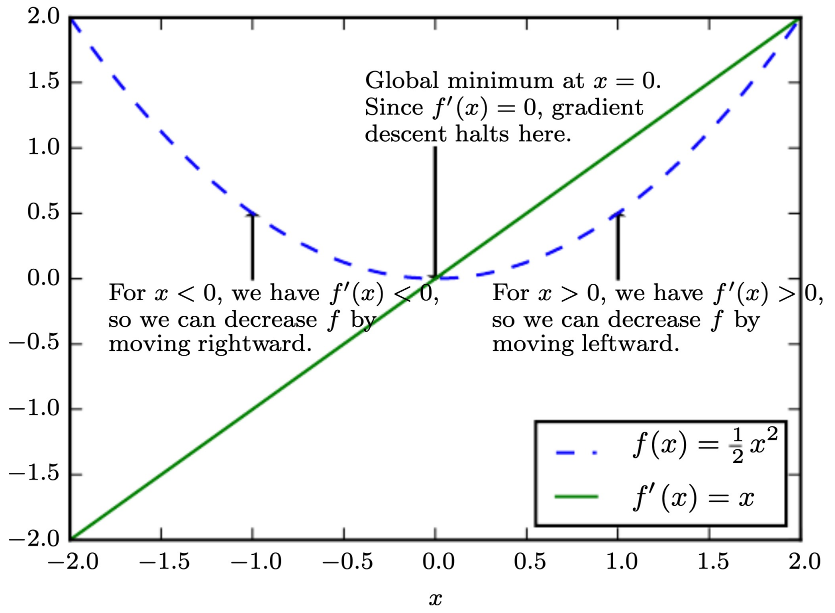
make the model distribution match the empirical distribution

(mean squared error corresponds to cross-entropy between empirical distribution and Gaussian model)

Iterative Optimization

Gradient Descent

$$f(x+\varepsilon) \approx f(x) + \varepsilon \frac{df(x)}{dx}$$



source

usually (except for special cases like ordinary least squares) no closed-form solution to ML optimization problems like minimization of a cost function or maximization of a likelihood function:

$$\nabla_{\widehat{\boldsymbol{\theta}}} J(\widehat{\boldsymbol{\theta}}) = 0$$

→ need for numerical methods

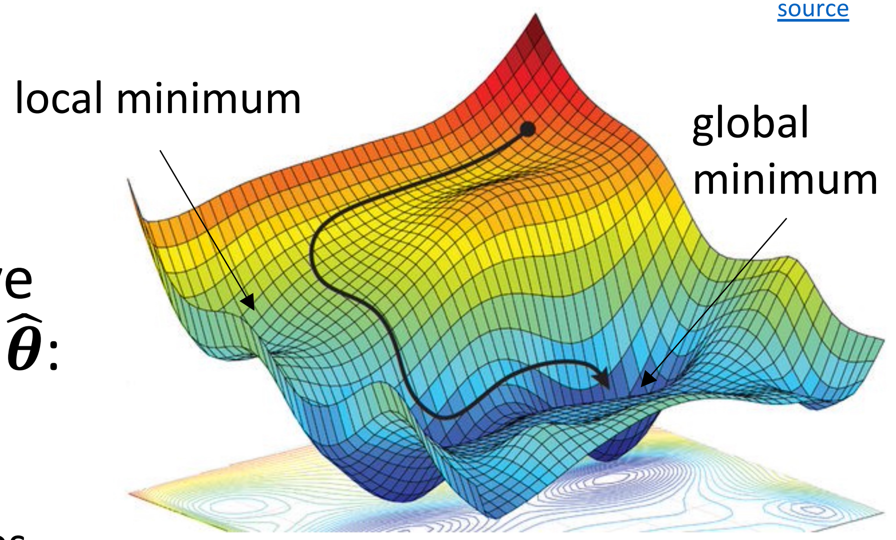
most popular choice: gradient descent

decreasing J by iteratively moving in direction of negative gradient (steepest descent) with respect to input vector $\widehat{\boldsymbol{\theta}}$:

$$\widehat{\boldsymbol{\theta}} \leftarrow \widehat{\boldsymbol{\theta}} - \eta \nabla_{\widehat{\boldsymbol{\theta}}} J(\widehat{\boldsymbol{\theta}})$$

step size (learning rate)

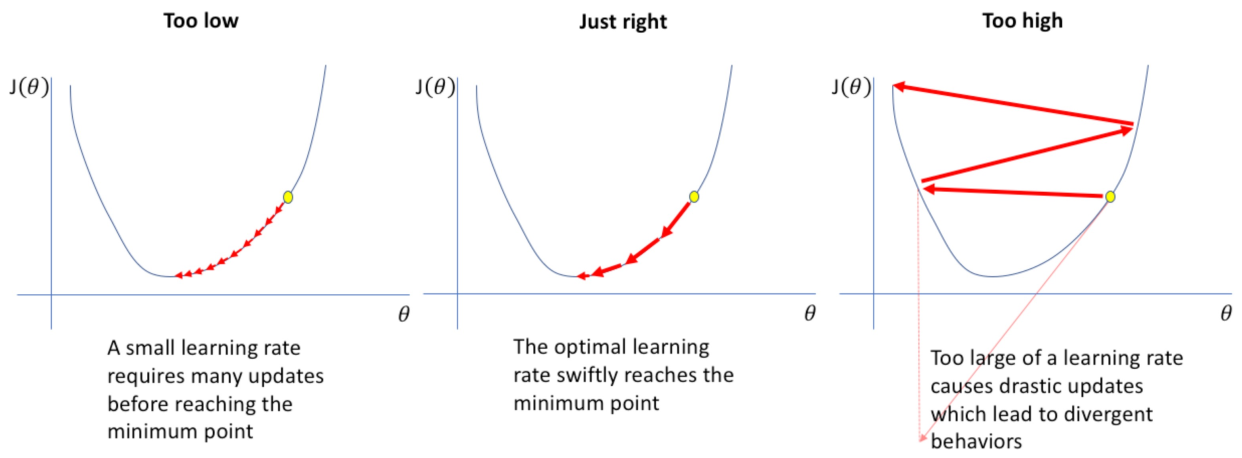
vector containing all partial derivatives



Learning Rate

learning rate η corresponds to positive scalar value (hyperparameter)

- often set to small constant
- can be optimized via line search: evaluate $J\left(\hat{\boldsymbol{\theta}} - \eta\nabla_{\hat{\boldsymbol{\theta}}}J\left(\hat{\boldsymbol{\theta}}\right)\right)$ for several values of η and choose η resulting in smallest value of objective function
- can be varied from iteration to iteration (e.g., via some heuristic): smaller steps closer to the minimum (learning rate schedule/decay)


Too low
J(θ)
θ
A small learning rate requires many updates before reaching the minimum point
Just right
J(θ)
θ
The optimal learning rate swiftly reaches the minimum point
Too high
J(θ)
θ
Too large of a learning rate causes drastic updates which lead to divergent behaviors

source

Batch and Stochastic Gradient Descent

in usual gradient descent: parameters $\widehat{\boldsymbol{\theta}}$ (and in turn objective function J) updated after full training epoch (one sweep through entire training data set) → batch learning

consider: $J(\widehat{\boldsymbol{\theta}}) = \frac{1}{n}\sum_{i=1}^{n} J_i(\widehat{\boldsymbol{\theta}})$ (J_i corresponds to loss function)

stochastic gradient descent updates after each training example (gradient of $J(\widehat{\boldsymbol{\theta}})$ approximated with gradient of single loss): $\widehat{\boldsymbol{\theta}} \leftarrow \widehat{\boldsymbol{\theta}} - \eta \nabla_{\widehat{\boldsymbol{\theta}}} J_i(\widehat{\boldsymbol{\theta}})$

→ online learning

- shuffling of samples after training epoch to prevent cycles
- adaptive learning rate to improve convergence

Mini Batches

Compared to its batch mode, stochastic gradient descent helps to avoid local minima (more robust), but is computationally expensive (less efficient).

→ mini-batch stochastic gradient descent as compromise:

splitting training data set into small batches used to calculate model error and update parameters

at the cost of one additional hyperparameter specifying mini batch size

Gradient Descent with Momentum

goal: improve optimization process by avoiding gradients bouncing around the search space (dampening oscillations), accelerate in direction of minima

hyperparameter specifying exponential decay (like learning rate η, α may be adapted over course of optimization)

algorithm:

- estimate gradient $\boldsymbol{g}$
- compute velocity update: $\boldsymbol{v} \leftarrow \alpha\, \boldsymbol{v} - \eta\, \boldsymbol{g}$
- apply parameter update: $\widehat{\boldsymbol{\theta}} \leftarrow \widehat{\boldsymbol{\theta}} + \boldsymbol{v}$

velocity: direction and speed at which the parameters move through parameter space, set to an exponentially decaying average of the past negative gradients (continues to move in its direction)

Bayesian Methods

Bayesian Statistics

frequentist perspective:

- true model parameters $\boldsymbol{\theta}$ fixed (but unknown → uncertain point estimates $\widehat{\boldsymbol{\theta}}$)
- data set from random sampling

Bayesian perspective:

- true model parameters $\boldsymbol{\theta}$ random (estimation of full distributions over parameters)
- data not random (observed)

Bayes Theorem

H : hypothesis (model)
E : evidence (data)

likelihood function (derived from statistical model)

$$\boxed{P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}}$$

posterior probability (belief updated with observed evidence)

new data (same for all hypotheses → uninteresting)

prior probability (belief in H before evidence)

Bayes Theorem for Parameter Estimation

data likelihood:
model (function of $\boldsymbol{\theta}$ with data fixed), e.g.,
for linear regression: $\mathcal{N}(y; \alpha + \boldsymbol{x}\,\boldsymbol{\beta}, \sigma^2)$

$$\boxed{p(\boldsymbol{\theta} | y, \boldsymbol{x}) \propto p(y | \boldsymbol{x}, \boldsymbol{\theta}) \ \cdot p(\boldsymbol{\theta})}$$

posterior:
probability distribution reflecting the effect of data on prior belief about parameters (increasing certainty with new evidence)

to be compared to frequentist point estimates $\widehat{\boldsymbol{\theta}}$

prior (e.g., broad Gaussian):
probability distribution reflecting initial belief about uncertain parameters (helps to reduce variance with few data)

Bayesian Prediction

$$p\big(y_{n+1}|\boldsymbol{x}_{n+1},(y_1,\boldsymbol{x}_1),\cdots,(y_n,\boldsymbol{x}_n)\big) = \int \underbrace{p(y_{n+1}|\boldsymbol{x}_{n+1},\boldsymbol{\theta})}_{\text{likelihood}}\; \underbrace{p\big(\boldsymbol{\theta}|(y_1,\boldsymbol{x}_1),\cdots,(y_n,\boldsymbol{x}_n)\big)}_{\text{latest posterior}}\, d\boldsymbol{\theta}$$

prediction of full probability distribution instead of frequentist point estimate(s) plugged into model distribution $(p(y|\boldsymbol{x}) = \mathrm{PDF}\big(y;\hat{f}(\boldsymbol{x}),\ldots\big))$

drawback: high computational costs due to integral over parameter distribution (especially for large training data sets)

Common Core of Statistical Learning

Most ML algorithms can be described by the general recipe of combining models, costs, and optimization methods.

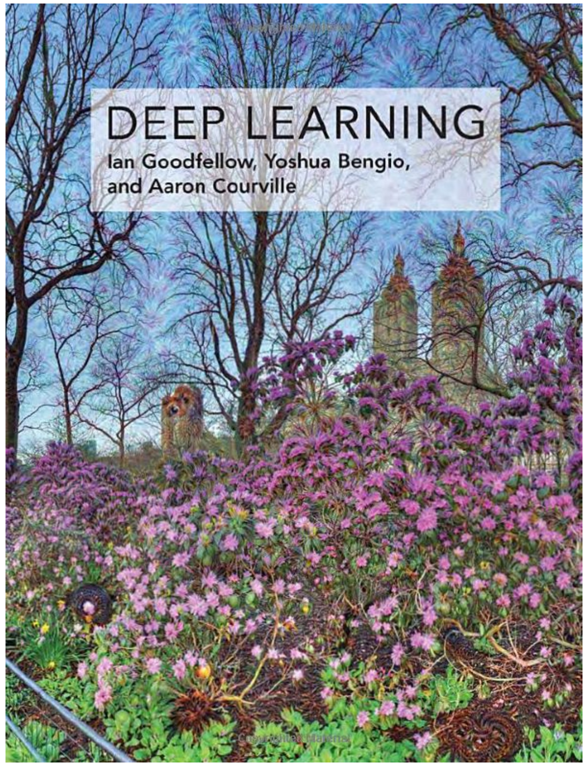
including non-linear models:

- neural networks: backpropagation
- support-vector machines: minimizing hinge loss (soft-margin SVM)
- decision trees: minimizing Kullback-Leibler divergence (classification)

and even unsupervised learning (e.g., principal component analysis)

Literature

not only deep learning, but also a nice introduction: https://www.deeplearningbook.org/



another general overview:

Machine Learning, Tom Mitchell

Overcome our Mathematical Limitations

evolution provided us with moderate math skills

AI/ML to the rescue:

- recognition of mathematical structures and patterns
- algorithm discovery: AlphaTensor
- symbolic regression (e.g., for physics)