# Statistical Learning
## *Common Core*

Understanding Machine Learning

# General Recipe

statistical learning algorithm by combining:

- **model** (e.g., linear function, Gaussian distribution)

- **objective function** (e.g., squared residuals)

- **optimization algorithm** (e.g., gradient descent)

# Consider Supervised Learning Scenario

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

random variables $Y$ and $X = (X_1, X_2, \cdots, X_p) \leftarrow$ usually high-dimensional

curve fitting / parameter estimation:

fit train data set of $(y_i, x_i)$ pairs $\rightarrow$ minimization of cost function

consider discriminative models:

- predict conditional density function $p(y|x_i)$

as opposed to generative models predicting $p(y, x)$ (or just $p(x)$) $\rightarrow x$ not given, more difficult

- often just conditional mean (depending on used loss) $E[Y|X = x_i]$ of $p(y|x_i)$

# Model

# Model in Linear Regression

fit:

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

to be estimated:
- $\hat{\alpha}, \widehat{\boldsymbol{\beta}}$

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

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

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 $\varepsilon_i$ in fit)

# Model Ingredients

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

$$p(y|\boldsymbol{x}_i) = \text{PDF}\big(y; \hat{f}(\boldsymbol{x}_i), \dots \big)$$

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
→ distribution with maximum entropy under given range and moment constraints
discrete: $H(X) = -\sum p_k \log p_k$, continuous: $H(X) = -\int p(x) \log p(x) dx$
(→ 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):
$$L\left(y_i, \hat{f}(\boldsymbol{x}_i); \widehat{\boldsymbol{\theta}}\right) = \left(y_i - \hat{f}(\boldsymbol{x}_i; \widehat{\boldsymbol{\theta}})\right)^2$$

# Cost Function

averaging losses over (empirical) training data set:

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

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

# Optimization

# Cost Minimization

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

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

for mean squared error (aka least squares method):

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

# 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 $\alpha$ for simplicity here): $\widehat{\boldsymbol{y}} = \mathbf{X}\,\widehat{\boldsymbol{\beta}}$

solve normal equations for training data set:

$$\nabla_{\widehat{\boldsymbol{\beta}}} \left(\boldsymbol{y} - \mathbf{X}\,\widehat{\boldsymbol{\beta}}\right)^{\mathrm{T}} \left(\boldsymbol{y} - \mathbf{X}\,\widehat{\boldsymbol{\beta}}\right) = 0$$

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

but usually no closed-form solution $\rightarrow$ need for numerical optimization

# Maximum Likelihood Estimation

# Likelihood

special objective function: likelihood function $\mathcal{L}$

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

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

a little bit confusing:

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

$p\big(y|\boldsymbol{x}; \widehat{\boldsymbol{\theta}}\big)$ 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 $(Y_1, \boldsymbol{X}_1), (Y_2, \boldsymbol{X}_2), \ldots, (Y_n, \boldsymbol{X}_n)$

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

$\rightarrow$ 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

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 $\rightarrow$ maximum of $\ell$ 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
$$\widehat{\boldsymbol{\theta}} = \text{argmax}_{\widehat{\boldsymbol{\theta}}}\, \ell(\widehat{\boldsymbol{\theta}}; \boldsymbol{y}\,|\boldsymbol{x}) = \text{argmin}_{\widehat{\boldsymbol{\theta}}}\left(-\ell(\widehat{\boldsymbol{\theta}}; \boldsymbol{y}\,|\boldsymbol{x})\right)$$

objective function

# Maximum Likelihood

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

$$\boldsymbol{\hat{\theta}} = \text{argmax}_{\boldsymbol{\hat{\theta}}}\ \ell\left(\boldsymbol{\hat{\theta}}; \boldsymbol{y} \mid \boldsymbol{x}\right)$$

mode of likelihood as point estimate

→ likelihood equations (if $\ell$ differentiable in $\boldsymbol{\hat{\theta}}$):

$$\frac{\partial \ell}{\partial \hat{\theta}_1} = 0,\ \frac{\partial \ell}{\partial \hat{\theta}_2} = 0,\ \cdots,\ \frac{\partial \ell}{\partial \hat{\theta}_k} = 0 \qquad \text{(in short: } \nabla_{\boldsymbol{\hat{\theta}}}\ \ell = 0 \text{)}$$

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: $D_{KL}(p||q) = H(p,q) - H(p)$) between true data-generating probability distribution (manifested by empirical distribution of training data) and probability distribution of model:

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

$$= \text{argmin}_{\widehat{\boldsymbol{\theta}}} \frac{1}{n} \sum_{i=1}^{n} \log \frac{p_{\text{data}}(y_i|\boldsymbol{x}_i)}{p_{\text{model}}(y_i|\boldsymbol{x}_i;\widehat{\boldsymbol{\theta}})} = \text{argmax}_{\widehat{\boldsymbol{\theta}}} \sum_{i=1}^{n} \ln\left(p_{\text{model}}(y_i|\boldsymbol{x}_i;\widehat{\boldsymbol{\theta}})\right) = \text{argmax}_{\widehat{\boldsymbol{\theta}}}\, \ell\left(\widehat{\boldsymbol{\theta}};\boldsymbol{y}\,|\boldsymbol{x}\right)$$

$E[\dots]$

no contribution from $p_{\text{data}}$

*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

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

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

Global minimum at $x = 0$.
Since $f'(x) = 0$, gradient descent halts here.

For $x < 0$, we have $f'(x) < 0$, so we can decrease $f$ by moving rightward.

For $x > 0$, we have $f'(x) > 0$, so we can decrease $f$ by moving leftward.
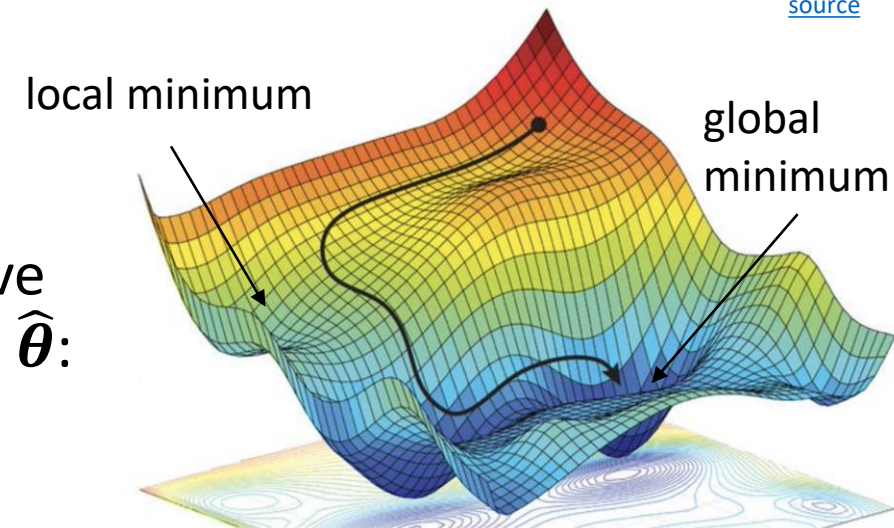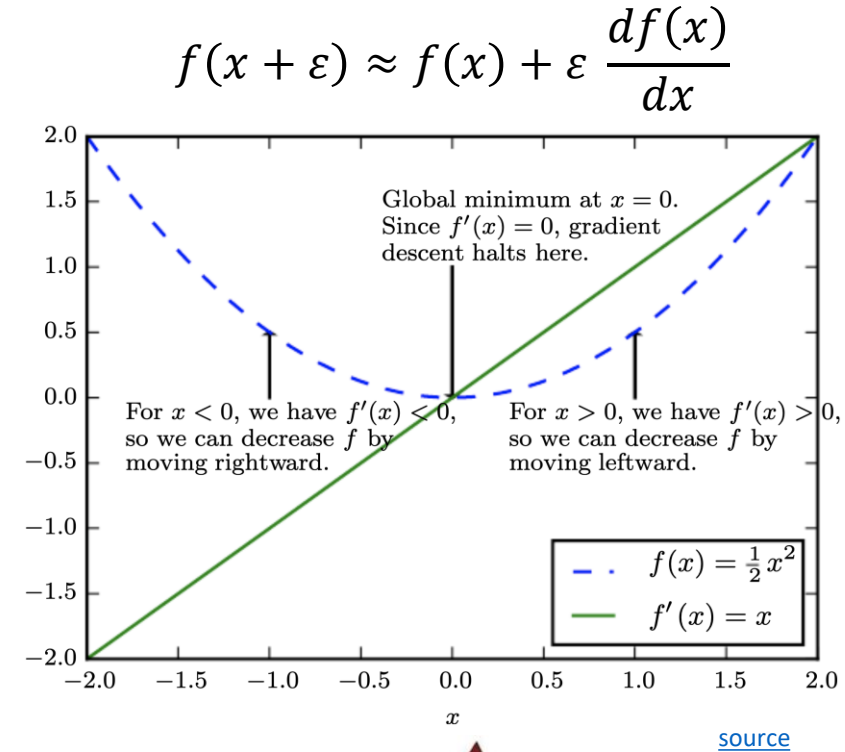
$f(x) = \frac{1}{2} x^2$
$f'(x) = x$

source

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

local minimum

global minimum

21

# Learning Rate

learning rate $\eta$ corresponds to positive scalar value (hyperparameter)

- often set to small constant

- can be optimized via line search: evaluate $J\left(\widehat{\boldsymbol{\theta}} - \eta \nabla_{\widehat{\boldsymbol{\theta}}} J(\widehat{\boldsymbol{\theta}})\right)$ for several values of $\eta$ and choose $\eta$ 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**

$J(\theta)$

$\theta$

A small learning rate requires many updates before reaching the minimum point

**Just right**

$J(\theta)$

$\theta$

The optimal learning rate swiftly reaches the minimum point

**Too high**

$J(\theta)$

$\theta$

Too large of a learning rate causes drastic updates which lead to divergent behaviors
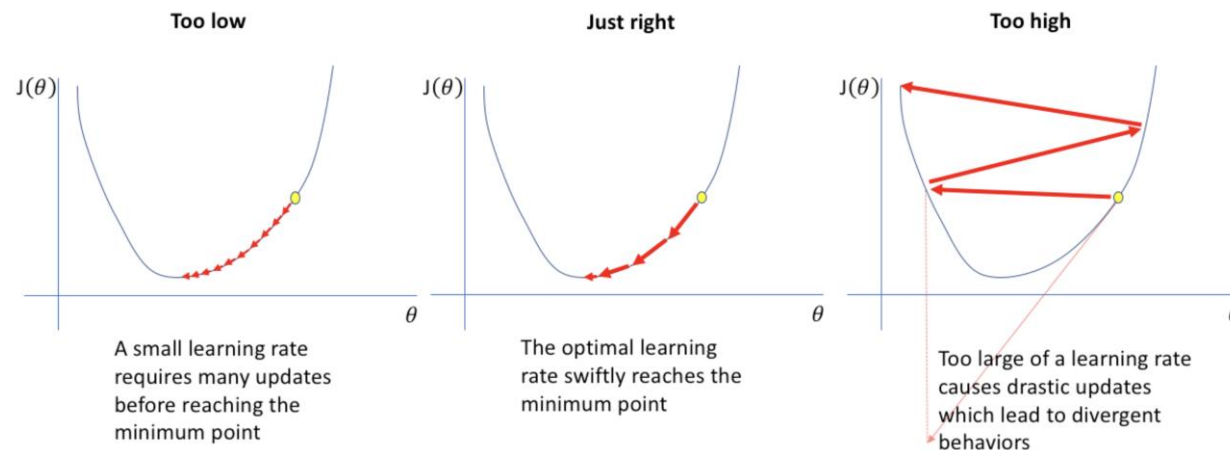
# Batch and Stochastic Gradient Descent

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

$\rightarrow$ 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 SGD 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

algorithm:

hyperparameter ($0 < \alpha < 1$) specifying exponential decay (like learning rate $\eta$, $\alpha$ may be adapted over course of optimization)

- 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 $\rightarrow$ 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)

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

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: $\mathcal{N}(y; \alpha + \boldsymbol{x}\,\boldsymbol{\beta}, \sigma^2)$

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

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

$$p(y_{n+1}|\boldsymbol{x}_{n+1}, (y_1, \boldsymbol{x}_1), \cdots, (y_n, \boldsymbol{x}_n)) = \int p(y_{n+1}|\boldsymbol{x}_{n+1}, \boldsymbol{\theta})\, p(\boldsymbol{\theta}|(y_1, \boldsymbol{x}_1), \cdots, (y_n, \boldsymbol{x}_n))\, d\boldsymbol{\theta}$$

likelihood    latest posterior

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

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

# Alternatives for PDF Predictions

**quantile regression**: estimate quantile $\tau$ of distribution instead of conditional mean by minimizing pinball loss

$$(1 - \tau) \sum_{y_i < \hat{q}_i} (\hat{q}_i - y_i) + \tau \sum_{y_i \geq \hat{q}_i} (y_i - \hat{q}_i)$$

instead of squared error loss (choice of loss function defines point estimate)

- possible with various ML methods, including neural networks and tree-based methods (like random forests or gradient boosting)

- subsequently approximate full probability distribution from a few predicted quantiles by fitting (assumed distribution or spline) or quantile-parameterized distributions

**PDF assumption for model and estimation of its parameters**: assume, e.g., Gaussian or negative binomial and estimate mean and variance (by means of different ML models or corresponding likelihood)

# 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 impurity functions (mean squared error for regression, (kind of) Kullback-Leibler divergence for classification)

and even

- unsupervised learning: self-supervised, PCA by maximum variance directions
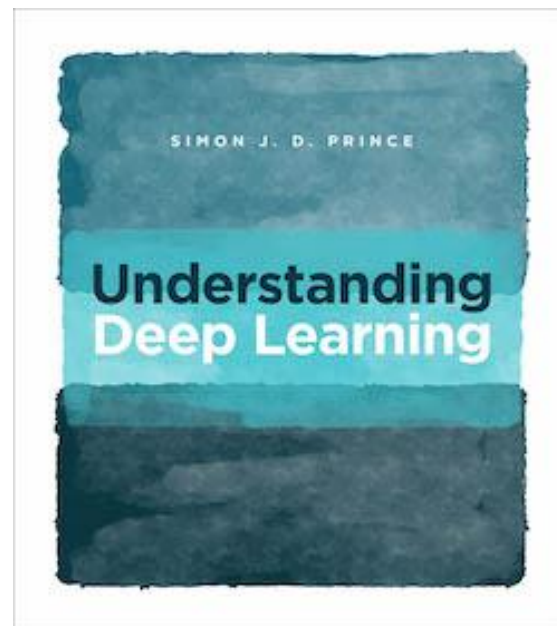- reinforcement learning: express rewards in loss functions

# Literature



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



following on from the above, also covering newer topics: udlbook

# 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)