Optimization for Machine Learning:

From Gradient Descent to Backpropagation

Dr. Lateef Jolaoso (PhD, FIMA, FHEA)

Big Data Technologies and Innovation Lab,
University of Hertfordshire, Hatfield, United Kingdom

Workshop Map: 60 Minutes

1. Optimization viewpoint
2. Convexity and optimality
3. Smoothness and strong convexity
4. Gradient descent and rates
5. Stochastic and adaptive methods
6. Backpropagation
7. Summary and questions

Optimization for Machine Learning

Optimization: The Engine of Machine Learning

Every machine learning model reduces, at its core, to an **optimization problem**:

$$\min_{\theta \in \Theta} \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) + \lambda R(\theta)$$

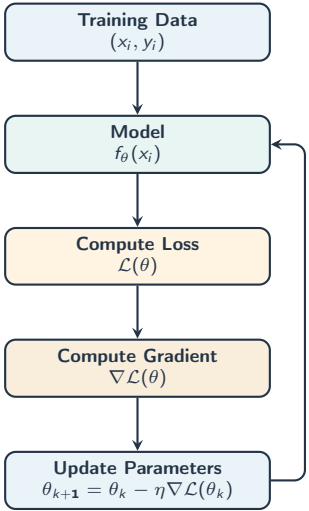
- θ — model parameters (millions to billions in deep learning)
- ℓ — loss function (measures prediction error)
- $R(\theta)$ — regularizer (controls model complexity)
- f_θ — prediction model (linear model, kernel method, neural network, etc.)

Optimization: The Engine of Machine Learning

Every machine learning model reduces, at its core, to an **optimization problem**:

$$\min_{\theta \in \Theta} \; \mathcal{L}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) + \lambda R(\theta)$$

- θ — model parameters (millions to billions in deep learning)
- ℓ — loss function (measures prediction error)
- $R(\theta)$ — regularizer (controls model complexity)
- f_θ — prediction model (linear model, kernel method, neural network, etc.)



A Taxonomy of Optimization Problems in AI

Convex: linear/logistic regression, SVM, LASSO, portfolio mean-variance

Nonconvex: deep neural networks, matrix factorization, clustering

Nonsmooth: ReLU nets, ℓ_1/nuclear norm, hinge loss, max-pooling

Multi-objective: fairness–accuracy, multi-task learning, risk–return

Bilevel: hyperparameter tuning, meta-learning, NAS, GANs

Mathematical Toolkit: Gradients and Hessians

Gradient = Direction

For a differentiable $f : \mathbb{R}^d \to \mathbb{R}$:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d} \right)^\top$$

- $\nabla f(x) = 0$ means **stationary point**.

Hessian = Curvature

$$\nabla^2 f(x) \in \mathbb{R}^{d \times d}, \quad [\nabla^2 f(x)]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

- $\nabla^2 f(x) \succeq 0 \Rightarrow$ local minimum
- $\nabla^2 f(x)$ indefinite $\Rightarrow$ saddle point.

Mathematical Toolkit: Gradients and Hessians

Gradient = Direction

For a differentiable $f : \mathbb{R}^d \to \mathbb{R}$:

$$\nabla f(x) = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_d} \right)^\top$$

- $\nabla f(x) = 0$ means **stationary point**.

Hessian = Curvature

$$\nabla^2 f(x) \in \mathbb{R}^{d \times d}, \quad [\nabla^2 f(x)]_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j}$$

- $\nabla^2 f(x) \succeq 0 \;\Rightarrow$ local minimum
- $\nabla^2 f(x)$ indefinite $\;\Rightarrow$ saddle point.

Example

Consider the quadratic bowl function

$$\min_{x \in \mathbb{R}^2} f(x) = \frac{1}{2} x_1^2 + 2x_2^2$$

The point $x^\star = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$ satisfies $\nabla f(x^*) = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$. Also, $\nabla^2 f(x) = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \succ 0$. Then x^* is a minimum point.

Convex Optimization Foundations

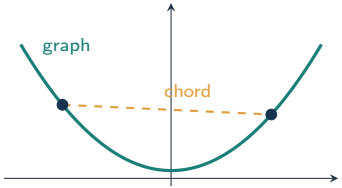
Convex Functions

A function f is convex if

$$f(tx + (1-t)y) \leq tf(x) + (1-t)f(y), \qquad t \in [0,1].$$

Geometric meaning

A function f is convex if $\nabla^2 f \succ 0$.



Running example

$$\nabla^2 f = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix} \succ 0.$$

Therefore $f(x_1, x_2) = \frac{1}{2}x_1^2 + 2x_2^2$ is convex.

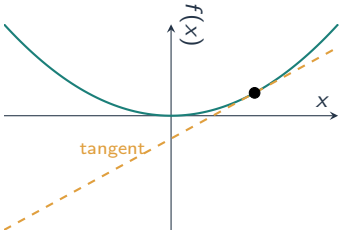
First-Order Characterization of Convexity

For differentiable f, convexity is equivalent to

$$\boxed{f(y) \geq f(x) + \nabla f(x)^\top (y - x) \quad \forall x, y.}$$

Powerful consequence

If f is convex and $\nabla f(x^\star) = 0$, then $x^\star$ is a global minimizer.



Smoothness and Strong Convexity

L-smoothness (Lipschitz gradient)

A differentiable function is L-smooth if

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|.$$

For twice-differentiable convex functions, this means

$$\nabla^2 f(x) \preceq LI.$$

Smoothness and Strong Convexity

L-smoothness (Lipschitz gradient)

A differentiable function is L-smooth if

$$\|\nabla f(x) - \nabla f(y)\| \leq L\|x - y\|.$$

For twice-differentiable convex functions, this means

$$\nabla^2 f(x) \preceq LI.$$

μ-strong convexity:

A function is μ-strongly convex if it has at least μ amount of upward curvature:

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{\mu}{2}\|y - x\|^2.$$

For twice-differentiable functions,

$$\nabla^2 f(x) \succeq \mu I.$$

Running example

Running example

$$\nabla^2 f = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}.$$

Largest eigenvalue:

$$\boxed{L = 4.}$$

Smallest eigenvalue:

$$\boxed{\mu = 1.}$$

Therefore

$$\boxed{I \preceq \nabla^2 f \preceq 4I.}$$

Running example

Running example

$$\nabla^2 f = \begin{pmatrix} 1 & 0 \\ 0 & 4 \end{pmatrix}.$$

Largest eigenvalue:

$$\boxed{L = 4.}$$

Smallest eigenvalue:

$$\boxed{\mu = 1.}$$

Therefore

$$\boxed{I \preceq \nabla^2 f \preceq 4I.}$$

Interpretation

- L controls the largest curvature and helps choose a safe learning rate.

  $$\eta = \frac{1}{L}$$

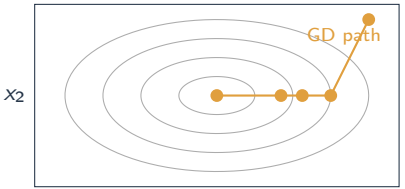
  is the standard step size for gradient descent.
- Strong convexity gives a unique minimizer.
- The function cannot be too flat near the solution.
- It leads to faster convergence rates.

Condition Number: Why Some Problems Are Slow

If f is both L-smooth and μ-strongly convex, define

$$\boxed{\kappa = \frac{L}{\mu}}.$$

- L: maximum curvature.
- μ: minimum curvature.
- κ: unevenness of curvature.
- Large κ gives elongated contours and slower GD.



Elliptical contours reflect unequal curvature.

Running example

$$L = 4, \qquad \mu = 1, \qquad \boxed{\kappa = 4.}$$

Optimality Conditions for Constrained Problem

Constrained Problem

$$
\begin{array}{lll}
\min\limits_{x} & f(x) & \\
\text{s.t.} & g_i(x) \leq 0, & i = 1, \ldots, m, \\
& h_j(x) = 0, & j = 1, \ldots, p.
\end{array}
$$

Optimality Conditions for Constrained Problem

Constrained Problem

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0, \qquad i = 1, \ldots, m, \\ & h_j(x) = 0, \qquad j = 1, \ldots, p. \end{aligned}$$

Karush–Kuhn–Tucker (KKT) Conditions

At a candidate solution $x^\star$, there exist multipliers μ_i and λ_j such that:

$$\begin{aligned} \text{Stationarity:} \quad & \nabla f(x^\star) + \sum_{i=1}^{m} \mu_i \nabla g_i(x^\star) + \sum_{j=1}^{p} \lambda_j \nabla h_j(x^\star) = 0, \\ \text{Primal feasibility:} \quad & g_i(x^\star) \leq 0, \qquad h_j(x^\star) = 0, \\ \text{Dual feasibility:} \quad & \mu_i \geq 0, \\ \text{Complementary slackness:} \quad & \mu_i g_i(x^\star) = 0, \qquad i = 1, \ldots, m. \end{aligned}$$

Linear Regression: The Model and the MSE Loss

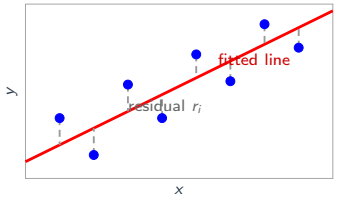
Model. Given training data

$$(x_i, y_i)_{i=1}^n, \qquad x_i \in \mathbb{R}^d, \quad y_i \in \mathbb{R},$$

linear regression predicts

$$\boxed{\hat{y}_i = w^\top x_i + b},$$

where $w \in \mathbb{R}^d$ contains the feature weights and $b \in \mathbb{R}$ is the intercept.



Goal: choose w and b so that the squared vertical residuals are as small as possible.

Mean Squared Error (MSE):

$$\boxed{\mathcal{L}(w, b) = \frac{1}{n} \sum_{i=1}^{n} \left(w^\top x_i + b - y_i \right)^2}$$

$$\boxed{\min_{w,b} \ \mathcal{L}(w, b)}$$

- $\mathcal{L}$ is a **convex, smooth quadratic** function of (w, b).
- Closed form (normal equations) $w^\star = (X^\top X)^{-1} X^\top y$, or solve by GD.

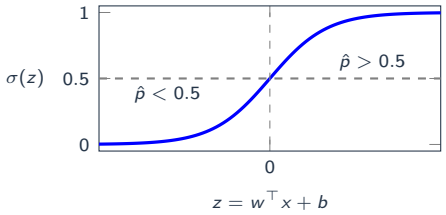
Logistic Regression: The Model and the Log-Loss

Binary classification. Given

$$(x_i, y_i)_{i=1}^n, \qquad x_i \in \mathbb{R}^d, \quad y_i \in \{0, 1\}.$$

The sigmoid converts this score into a probability:

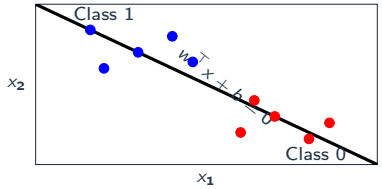
$$\hat{p}_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}} = P(y_i = 1 \mid x_i)$$



The sigmoid maps any score $z \in \mathbb{R}$ into a probability $\hat{p} \in (0, 1)$.

Binary cross-entropy (log-loss):

$$\mathcal{L}(w, b) = -\frac{1}{n} \sum_{i=1}^{n} [y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i)]$$



Optimization problem:

$$\min_{w,b} \mathcal{L}(w, b)$$

Support Vector Machines: Maximum-Margin Classification

Binary classification. Given training data

$$(x_i, y_i)_{i=1}^n, \qquad x_i \in \mathbb{R}^d, \quad y_i \in \{-1, +1\},$$

SVM uses the linear score

$$z_i = w^\top x_i + b$$

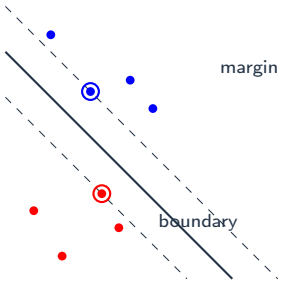
and predicts

$$\boxed{\hat{y}_i = \operatorname{sign}(w^\top x_i + b).}$$

Hard-margin SVM:

$$\boxed{\begin{aligned} \min_{w,b} \quad & \frac{1}{2}\|w\|_2^2 \\ \text{s.t.} \quad & y_i(w^\top x_i + b) \geq 1, \qquad i = 1, \ldots, n. \end{aligned}}$$

Used when the classes are linearly separable. The constraints require every point to be correctly classified and outside (or on) the margin.

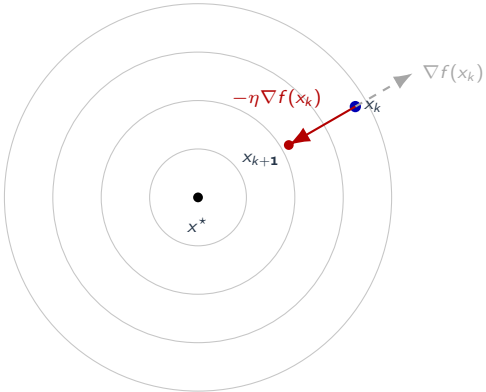


Solid line: decision boundary $w^\top x + b = 0$. Dashed lines: margins (± 1). Circled points: support vectors touching the margin.

Gradient Descent and its Variants

Gradient Descent: The Basic Algorithm

Input: $x_0 \in \mathbb{R}^d$, step size $\eta > 0$
for $k = 0, 1, 2, \ldots, K-1$ **do**
  Compute $\nabla f(x_k)$;
  $x_{k+1} = x_k - \eta \nabla f(x_k)$;
end
Output: x_K



Running Example

Recall

$$f(x_1, x_2) = \frac{1}{2}x_1^2 + 2x_2^2, \qquad \nabla f(x) = \begin{pmatrix} x_1 \\ 4x_2 \end{pmatrix}.$$

Since $L = 4$, choose $\eta = 1/L = 1/4$. At $x_0 = (4, 2)^\top$

$$\nabla f(x_0) = \begin{pmatrix} 4 \\ 8 \end{pmatrix}.$$

Iteration 1

$$x_1 = x_0 - \frac{1}{4}\nabla f(x_0) = \begin{pmatrix} 4 \\ 2 \end{pmatrix} - \frac{1}{4}\begin{pmatrix} 4 \\ 8 \end{pmatrix} = \boxed{\begin{pmatrix} 3 \\ 0 \end{pmatrix}}.$$

Iteration 2

$$x_2 = \begin{pmatrix} 3 \\ 0 \end{pmatrix} - \frac{1}{4}\begin{pmatrix} 3 \\ 0 \end{pmatrix} = \boxed{\begin{pmatrix} 2.25 \\ 0 \end{pmatrix}}.$$

Learning Rate: Too Small, Good, Too Large

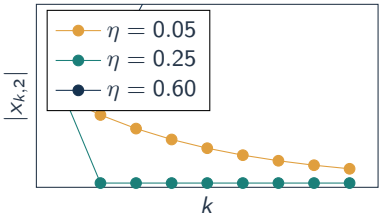
What is happening?

$$(4,2) \to (3,0) \to (2.25,0) \to \cdots \to (0,0).$$

For the running example,

$$x_{k+1,2} = x_{k,2} - 4\eta x_{k,2} = (1-4\eta)x_{k,2}.$$

- If $\eta = 0.05$: $1 - 4\eta = 0.8$; stable but slow.
- If $\eta = 0.25$: $1 - 4\eta = 0$; fast for this coordinate.
- If $\eta = 0.60$: $1 - 4\eta = -1.4$; oscillates and diverges.



Key idea

The learning rate must respect curvature. Smoothness gives the safe scale $\eta \approx 1/L$.

Convergence Result I: Convex and Smooth

Assume f is convex and L-smooth. With step size $\eta = 1/L$,

$$\boxed{f(x_k) - f^\star \le \frac{L\|x_0 - x^\star\|^2}{2k}.}$$

Meaning

The objective gap decreases like $1/k$.

- This is **sublinear convergence**.
- Accurate solutions may require many iterations.

Running example bound

Here $L = 4$ and $\|x_0 - x^\star\|^2 = 20$, so

$$f(x_k) - f^\star \le \frac{40}{k}.$$

This bound is general but often conservative.

Convergence Result II: Strongly Convex and Smooth

Assume f is μ-strongly convex and L-smooth. With $\eta = 1/L$,

$$\|x_k - x^\star\|^2 \leq \left(1 - \frac{\mu}{L}\right)^k \|x_0 - x^\star\|^2.$$

Meaning

The error shrinks by a constant factor each iteration.

- This is **linear convergence**.
- Since $\kappa = L/\mu$, the factor is $1 - 1/\kappa$.

Running example

$$L = 4, \qquad \mu = 1.$$

Thus

$$\left(1 - \frac{\mu}{L}\right) = \frac{3}{4}.$$

So the distance error is bounded by

$$\|x_k - x^\star\|^2 \leq \left(\frac{3}{4}\right)^k 20.$$

Why Stochastic Gradient Methods?

Many ML objectives have finite-sum form:

$$f(x) = \frac{1}{n}\sum_{i=1}^{n} \ell_i(x), \qquad \nabla f(x) = \frac{1}{n}\sum_{i=1}^{n} \nabla \ell_i(x).$$

- Full gradient descent uses all n data points before one update.
- If n is millions, one full gradient step can be expensive.
- Stochastic methods use a cheap random gradient estimate instead.

Stochastic Gradient Descent (SGD)

At iteration k, randomly choose one data index:

$$i_k \sim \mathrm{Uniform}\{1, \ldots, n\}.$$

Use one sample gradient:

$$g_k = \nabla \ell_{i_k}(x_k).$$

SGD update:

$$\boxed{x_{k+1} = x_k - \eta_k \nabla \ell_{i_k}(x_k).}$$

- Cheap per iteration.
- Noisy path, because each sample gives a different gradient.
- Unbiased: $\mathbb{E}[g_k \mid x_k] = \nabla f(x_k)$.
- Classical exact-convergence theory usually uses diminishing step sizes $\eta_k \to 0$.

Mini-Batch SGD

Instead of one data point, choose a small batch B_k with $|B_k| = b$:

$$g_{B_k}(x_k) = \frac{1}{b} \sum_{i \in B_k} \nabla \ell_i(x_k).$$

Then update:

$$x_{k+1} = x_k - \eta_k g_{B_k}(x_k).$$

Small batch

Cheap but noisy.

Large batch

More stable but more expensive.

Practical reason

Mini-batches work well with GPUs and are standard in deep learning.

Momentum: Remembering Previous Directions

Gradient descent uses only the current gradient. Momentum keeps a memory of previous gradients:

$$v_{k+1} = \beta v_k + \nabla f(x_k), \qquad x_{k+1} = x_k - \eta v_{k+1}.$$

- Accumulates directions that remain useful.
- Reduces zig-zagging in narrow valleys.
- Often speeds up training in practice.

Adam: Adaptive Learning Rates

Adam keeps two moving averages from the gradient $g_k = \nabla f(x_k)$:

$$\underbrace{m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k}_{\text{average direction}}, \qquad \underbrace{v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^{\odot 2}}_{\text{average squared size}}.$$

Adam update

$$\boxed{x_{k+1} = x_k - \eta \frac{\hat{m}_k}{\sqrt{\hat{v}_k} + \varepsilon}}$$

- m_k smooths noisy gradients and remembers direction.
- v_k estimates how large gradients are in each coordinate.
- Large historical gradients give smaller effective steps.
- Adam combines momentum-like direction with coordinate-wise scaling.

Adam on the Running Example (1/2)

Recall, the first gradient is

$$g_1 = \nabla f(x_0) = \boxed{\begin{pmatrix} 4 \\ 8 \end{pmatrix}}.$$

Initialize

$$m_0 = v_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \qquad \beta_1 = 0.9, \qquad \beta_2 = 0.999.$$

First moment: average direction

$$m_1 = 0.9 m_0 + 0.1 g_1 = \boxed{\begin{pmatrix} 0.4 \\ 0.8 \end{pmatrix}}.$$

Second moment: average squared size

$$v_1 = 0.999 v_0 + 0.001 g_1^{\odot 2} = 0.001 \begin{pmatrix} 16 \\ 64 \end{pmatrix} = \boxed{\begin{pmatrix} 0.016 \\ 0.064 \end{pmatrix}}.$$

Adam on the Running Example (2/2)

Bias correction gives

$$\hat{m}_1 = \frac{m_1}{1-0.9} = \begin{pmatrix} 4 \\ 8 \end{pmatrix}, \qquad \hat{v}_1 = \frac{v_1}{1-0.999} = \begin{pmatrix} 16 \\ 64 \end{pmatrix}.$$

Therefore

$$\frac{\hat{m}_1}{\sqrt{\hat{v}_1} + \varepsilon} \approx \begin{pmatrix} 4/4 \\ 8/8 \end{pmatrix} = \boxed{\begin{pmatrix} 1 \\ 1 \end{pmatrix}}.$$

With $\eta = 0.1$,

$$x_1 = x_0 - 0.1 \begin{pmatrix} 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \\ 2 \end{pmatrix} - \begin{pmatrix} 0.1 \\ 0.1 \end{pmatrix} = \boxed{\begin{pmatrix} 3.9 \\ 1.9 \end{pmatrix}}.$$

Key idea

The raw gradient was $(4, 8)^\top$, but Adam scaled each coordinate using its gradient history.

Backpropagation and Neural Network Training

Neural Network Training as Optimization

A neural network is a function f_θ built from layers:

$$f_\theta(x) = W_L\sigma(W_{L-1}\sigma(\cdots\sigma(W_1x + b_1)\cdots) + b_{L-1}) + b_L.$$

Training solves

$$\min_\theta \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i).$$

Backpropagation

Computes the gradient $\nabla_\theta \mathcal{L}(\theta)$ efficiently.

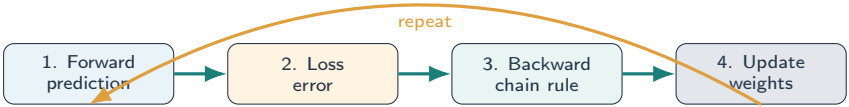
Optimizer

Uses the gradient to update parameters: SGD, momentum, Adam, etc.

Backpropagation

The question

If I change one weight slightly, how much does the final loss change?



Backpropagation

Efficiently computes $\partial L/\partial w$ for every weight by reusing local derivatives.

Optimizer

Uses those gradients to change the weights: GD, SGD, momentum, Adam, etc.

Tiny Network: Forward Pass

One input, one hidden unit, one output:

$$x = 1, \quad w_1 = 0.5, \quad w_2 = 2, \quad \bar{y} = 1, \quad \sigma(z) = \frac{1}{1 + e^{-z}}.$$

$$z_1 = w_1 x, \qquad h = \sigma(z_1), \qquad y = w_2 h, \qquad L = \frac{1}{2}(y - \bar{y})^2.$$



$$z_1 = 0.5 \cdot 1 = 0.5,$$
$$h = \sigma(0.5) \approx 0.622,$$
$$y = 2(0.622) \approx 1.245,$$
$$L = \tfrac{1}{2}(1.245 - 1)^2 \approx 0.030.$$

Tiny Network: Backward Pass

Now compute how the loss changes with the weights.

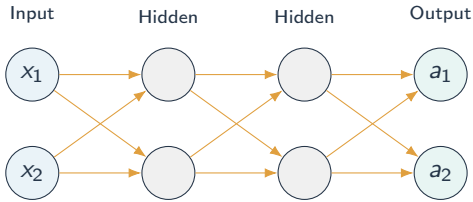
$$\frac{\partial L}{\partial y} = y - \bar{y} \approx 0.245,$$
$$\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w_2} = (0.245)h \approx 0.152,$$
$$\frac{\partial L}{\partial h} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial h} = (0.245)w_2 \approx 0.490,$$
$$\frac{\partial h}{\partial z_1} = h(1-h) \approx 0.235,$$
$$\frac{\partial L}{\partial w_1} = 0.490(0.235)(1) \approx 0.115.$$

Interpretation

The loss is too high because $y > \bar{y}$. Both gradients are positive, so gradient descent reduces w_1 and w_2.

$$w_j \leftarrow w_j - \eta \frac{\partial L}{\partial w_j}.$$

General Network Notation



For layer l:

$$z^{(l)} = W^{(l)} a^{(l-1)} + b^{(l)}, \qquad a^{(l)} = \sigma(z^{(l)}).$$

Backpropagation Recursion

Define the error signal

$$\delta^{(l)} = \frac{\partial L}{\partial z^{(l)}}.$$

For squared loss and differentiable activation:

$$\boxed{\delta^{(L)} = (a^{(L)} - y) \odot \sigma'(z^{(L)}).}$$

For hidden layers:

$$\boxed{\delta^{(l)} = \left((W^{(l+1)})^\top \delta^{(l+1)}\right) \odot \sigma'(z^{(l)}), \qquad l = L-1, \ldots, 1.}$$

Gradients:

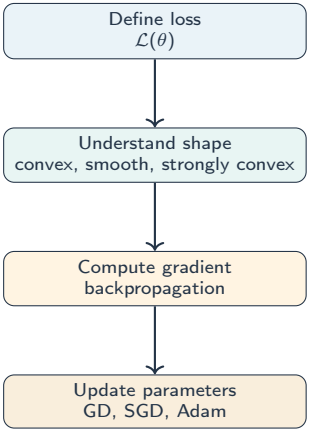
$$\boxed{\frac{\partial L}{\partial W^{(l)}} = \delta^{(l)} (a^{(l-1)})^\top, \qquad \frac{\partial L}{\partial b^{(l)}} = \delta^{(l)}.}$$

Training Loop: Backpropagation + Optimizer

Input: Training data, network parameters θ, learning rate η
for *each mini-batch* B_k **do**
    Forward pass: compute predictions and loss;
    Backward pass: compute gradients using backpropagation;
    Update parameters, e.g. $\theta_{k+1} = \theta_k - \eta \nabla \mathcal{L}_{B_k}(\theta_k)$;
end
Output: Trained parameters θ

Backpropagation	Optimizer
Computes gradients.	Uses gradients to improve parameters.

Big Picture Summary



Final Takeaways

1. Formulate

- ML training is an optimization problem:

$$\min_{\theta} \mathcal{L}(\theta).$$

- The loss measures how well the model fits the data.

2. Understand

- **Gradient**: direction of change.
- **Smoothness**: controls safe step sizes.
- **Convexity**: provides global guarantees.

3. Optimize

- GD updates parameters using gradients.
- SGD/mini-batches scale to large data.
- Momentum and Adam improve practical training.

Selected References

- S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.
- Y. Nesterov, *Lectures on Convex Optimization*, Springer, 2018.
- L. Bottou, F. Curtis and J. Nocedal, "Optimization Methods for Large-Scale Machine Learning," *SIAM Review*, 2018.
- A. Beck, *First-Order Methods in Optimization*, SIAM, 2017.
- D. E. Rumelhart, G. E. Hinton and R. J. Williams, "Learning Representations by Back-Propagating Errors," *Nature*, 1986.

Thank you!

Questions & Discussion