Deep Learning & Generative AI in Healthcare

Session 02

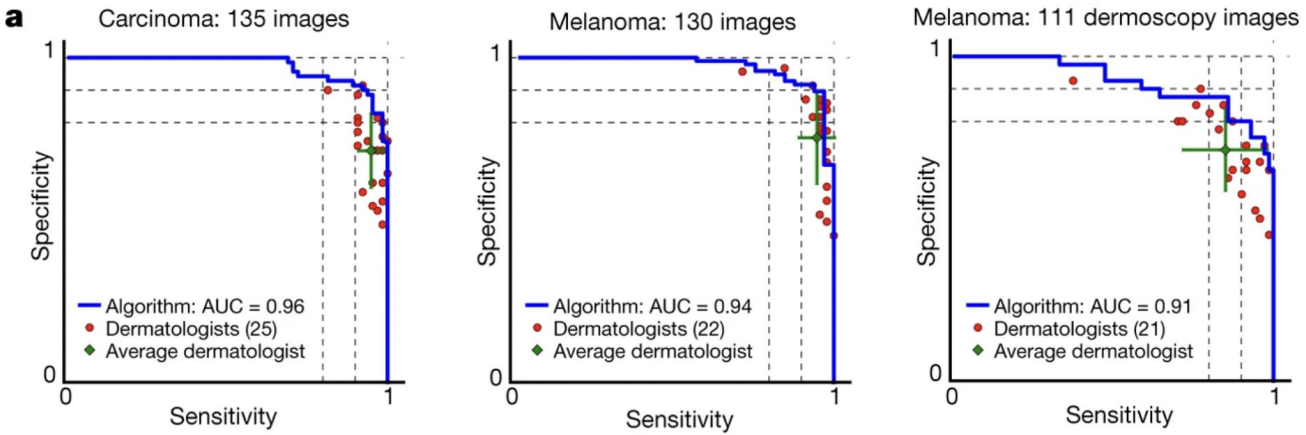
AI in Medical Diagnosis

Growing Need:

Address physician shortages, reduce medical errors, accelerate drug discovery, personalized medicine.

Multi-Cancer Type Performance (Esteva et al., Nature 2017)

Carcinoma	Melanoma	Melanoma
135 images	130 images	111 dermoscopy images
AUC = 0.96	AUC = 0.94	AUC = 0.91
vs. 25 Dermatologists	vs. 22 Dermatologists	vs. 21 Dermatologists



AI in Medicine: Drug Discovery & Molecular Dynamics

Geometric Learning

Input: Amino acid (AA) sequence

⬇

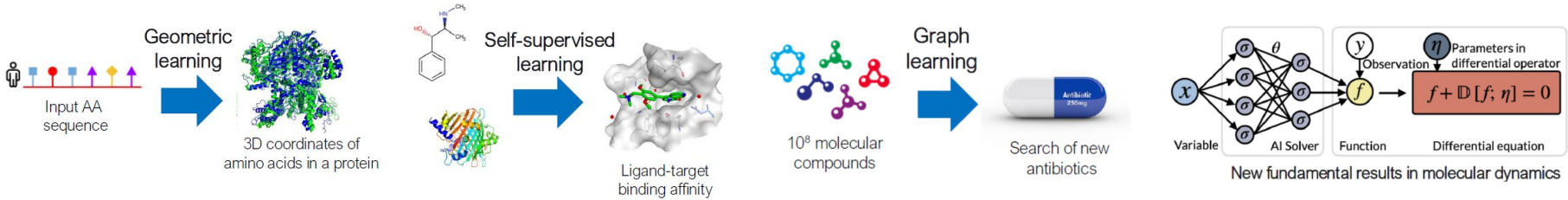
Output: 3D coordinates of amino acids in protein

Graph Learning

Scale: 10^8 molecular compounds

⬇

Application: Search for new antibiotics



Self-Supervised Learning

Application: Ligand-target binding affinity prediction

AI for Molecular Dynamics

Breakthrough: New fundamental results in molecular dynamics

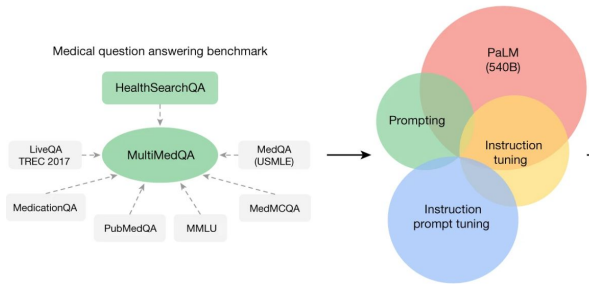
Variable → AI Solver → Function → Differential equation

Parameters embedded in differential operators

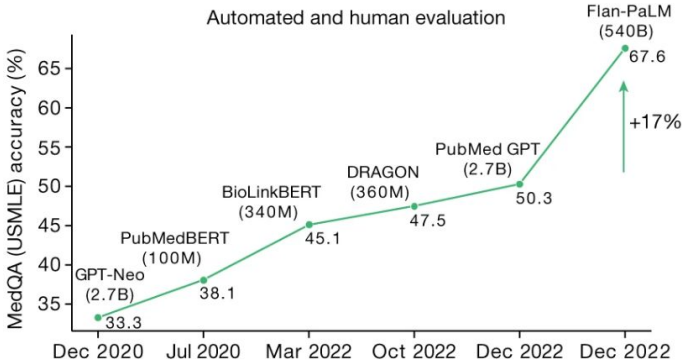
AI in Healthcare: Clinical Knowledge Encoding

Large language models encode clinical knowledge

Medical Question Answering Benchmark



Performance Evolution



Training Approaches

Prompting	**Instruction tuning**
Instruction prompt tuning	**PaLM (540B)**

Example Question

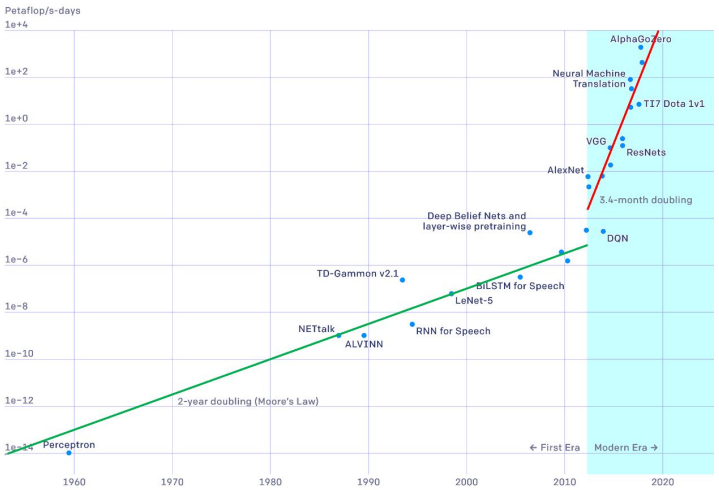
Q: How long does it take for newborn jaundice to go away?

Med-PaLM Answer: Newborn jaundice is caused by buildup of bilirubin. It is common and typically harmless, but can be a sign of a more serious condition. The jaundice typically goes away on its own within a few weeks...

Result: Med-PaLM performs encouragingly on consumer medical question answering

The Deep Learning Revolution

Computational Scaling



Period	Scaling Pattern
1960-2012	2-year doubling (Moore's Law)
2012-present	3.4-month doubling (10× per year!)

Parameter Scale Evolution:
1980s: Hundreds to thousands
→ Millions → Billions → Trillions (10^{12})

Key Enabling Technologies

GPUs (Graphics Processing Units)
Massive parallelism enables layer-wise computation. Training now uses thousands of GPUs linked by high-speed interconnections

Residual Connections (He et al., 2015a)
Address vanishing gradients, enable training of networks with hundreds of layers

Automatic Differentiation
Backpropagation code generated automatically from forward propagation specification. Enables rapid architecture experimentation

Open Source Ecosystem
Researchers build on others' work, accelerating progress

Machine Learning Fundamentals: Curve Fitting Example

Synthetic Data Setup

Component	Description
Input variable	x (continuous, real axis)
Target variable	t (continuous, real axis)
Training set	N observations: $\{x_1,...,x_n\} \rightarrow \{t_1,...,t_n\}$
Goal	Predict t for new value of x

Key Concept: Generalization
Ability to make accurate predictions on previously unseen inputs

Data Generation Process

Example: N = 10 data points

- Input: x uniformly spaced in [0,1]
- Target: t = sin(2πx) + Gaussian noise
- Captures real-world property: underlying regularity corrupted by random noise

Linear Model: Polynomial Fitting

Polynomial Function:

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + ... + w_m x^M$$

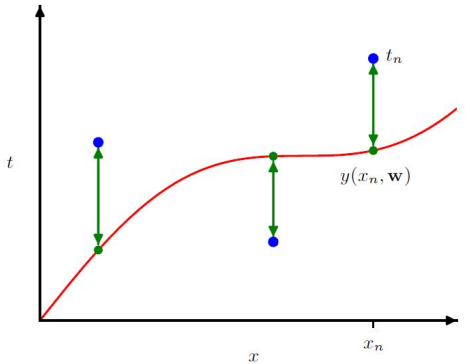
- M = order of polynomial
- $\mathbf{w}$ = coefficient vector $\{w_0,...,w_m\}$
- Linear in coefficients $\mathbf{w}$ (despite nonlinear in x)

Error Function

Sum of Squares Error:

$$E(\mathbf{w}) = \frac{1}{2} \sum \{y(x_n, \mathbf{w}) - t_n\}^2$$

Measures misfit between predictions and training data

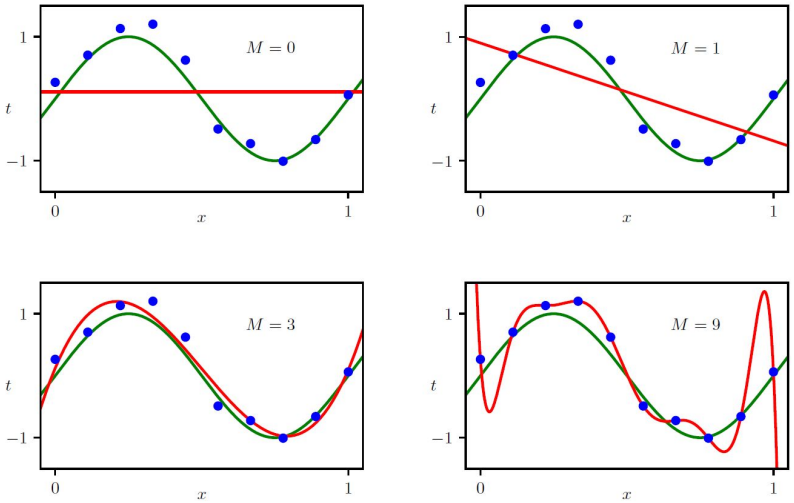


Solution: Minimize $E(\mathbf{w})$ by finding optimal coefficients $\mathbf{w}^*$. Since E is quadratic in $\mathbf{w}$, has unique closed-form solution.

Model Complexity and Overfitting

Model Selection: Choosing Polynomial Order M

Model (M)	Behavior	Issue
M = 0 (constant)	Horizontal line	Underfitting
M = 1 (linear)	Straight line	Underfitting
M = 3 (cubic)	Best fit to sin(2πx)	Optimal
M = 9 (9th order)	Passes through all points E(w*) = 0	Overfitting

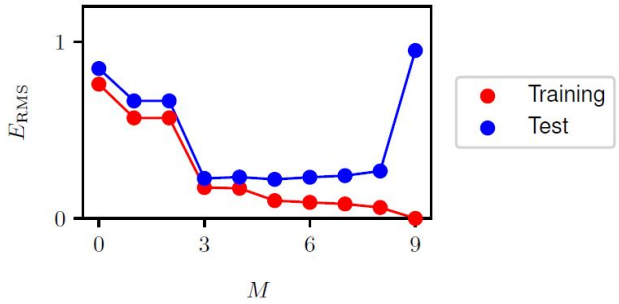


Evaluation: Training vs Test Error

Root Mean Square (RMS) Error:

$$E_{RMS} = \sqrt{[1/N \Sigma [y(x_n, w) - t_n]^2]}$$

Allows comparison across different dataset sizes



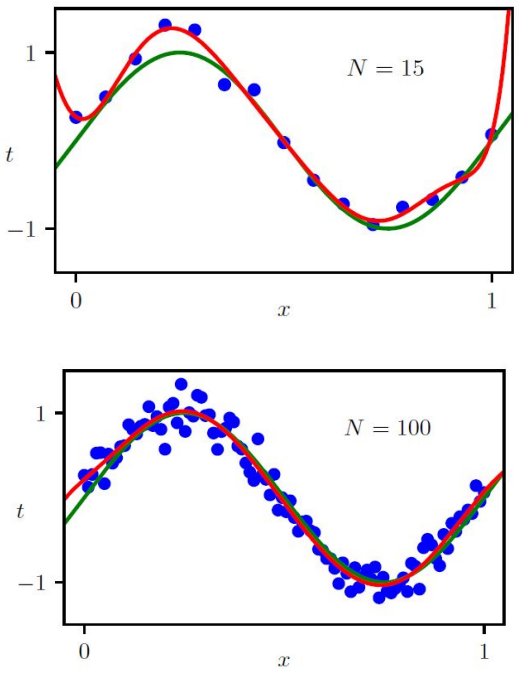
M too small
High test error
(Inflexible)

M optimal
Low test error
(Good generalization)

M too large
High test error
(Overfitting)

Impact of Data Size & Deep Learning Paradigm

Overfitting vs Dataset Size



Classical vs Deep Learning Heuristics

Paradigm	Data-to-Parameter Ratio
Classical Statistics	Data points ≥ 5-10× parameters (N ≥ 5M to 10M)
Deep Learning	Excellent results even when parameters >> data points

Classical Approach

- Constrained by data availability
- Risk of overfitting with complex models
- Must balance model complexity carefully

Deep Learning Approach

- Models with millions of parameters
- Regularization techniques prevent overfitting
- Architecture design and training methods crucial

Example: Melanoma detection used ~25M parameters with only ~129K training images—violating classical heuristics but achieving expert-level performance

Regularization: Controlling Overfitting

The Challenge

> Limiting model parameters based on dataset size is unsatisfying. Better approach: choose model complexity based on **problem complexity**

Regularization Approach

Modified Error Function:

$$\tilde{E}(w) = \frac{1}{2} \sum \{y(x_n, w) - t_n\}^2 + (\lambda/2)\|w\|^2$$

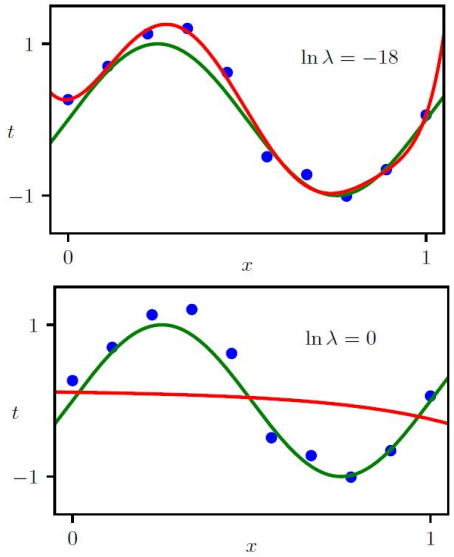
Penalty term: $\lambda/2 \, \|w\|^2$ discourages large coefficient magnitudes

Where:

- $\|w\|^2 = w_0^2 + w_1^2 + \ldots + w_m^2$
- λ controls relative importance of regularization
- w_0 often omitted from regularizer

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.11	0.90	0.12	0.26
$w_1^\star$		−1.58	11.20	−66.13
$w_2^\star$			−33.67	1,665.69
$w_3^\star$			22.43	−15,566.61
$w_4^\star$				76,321.23
$w_5^\star$				−217,389.15
$w_6^\star$				370,626.48
$w_7^\star$				−372,051.47
$w_8^\star$				202,540.70
$w_9^\star$				−46,080.94

Effect of Regularization



λ Value	Effect
ln λ = -18	Optimal: suppresses overfitting, good fit to sin(2πx)
ln λ = 0	Too large: poor fit (underfitting)
ln λ = -∞	No regularization: overfitting (M=9 case)

Also called: Weight decay (in neural networks) or shrinkage methods (in statistics)

Model Selection & Cross-Validation

Hyperparameters

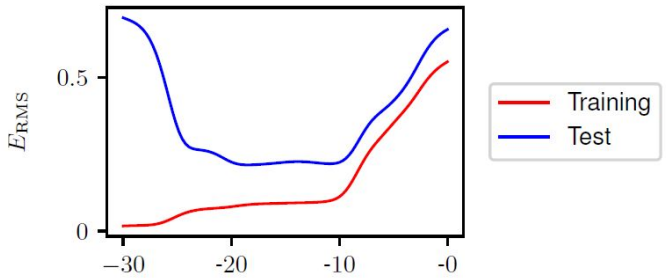
> **Definition:** Parameters fixed during error minimization (e.g., λ, polynomial order M)

Why not optimize jointly?

- Minimizing w.r.t. both w and λ leads to $\lambda \to 0$ (overfitting)
- Optimizing M via training error leads to large M (overfitting)

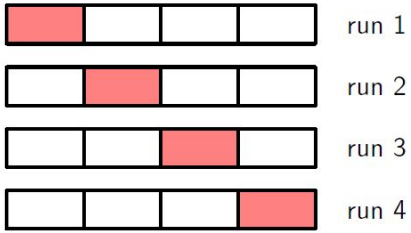
Data Partitioning Strategy

Dataset	Purpose
Training Set	Determine coefficients w
Validation Set (Hold-out/Development)	Select hyperparameters (lowest error)
Test Set	Final model evaluation



Cross-Validation

> **Problem:** Limited data means small validation set → noisy performance estimates

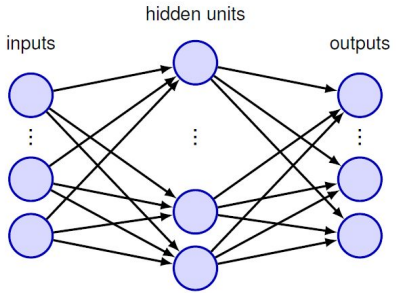


S-fold Cross-Validation:

- Partition data into S equal groups
- Use S-1 groups for training, 1 for validation
- Repeat S times, each group as validation once
- Average performance scores
- Uses (S-1)/S of data for training

Modern Deep Learning Paradigms

Representation Learning



> **Key Concept:** Hidden layers learn to transform input data into semantically meaningful representations, creating easier problems for final layers

Feed-forward neural networks: Information flows forward through network via repeated application of equations (1.5) and (1.6)

Training: Random initialization → iterative gradient-based updates via error backpropagation

Foundation Models

Paradigm Shift:

- Large models adaptable/fine-tuned to multiple downstream tasks
- Leverage large, heterogeneous datasets
- Broad applicability (Bommasani et al., 2021)

Visual Cortex Similarity

Neural networks for image processing learn internal representations remarkably similar to those in mammalian visual cortex

Scaling Benefits

Aspect	Benefit
Performance	Superior on specific tasks
Generality	Solve broader range of problems
Example	Large language models outperform specialist networks

> **Scaling beats innovation:** Improvements from architectural innovations or inductive biases often superseded by simply scaling up data, model size, and compute (Sutton, 2019)

Deep Networks & Hierarchical Representations

From Two-Layer to Deep Networks

Multi-Layer Architecture:

$$z^{(l)} = h^{(l)}(W^{(l)}z^{(l-1)})$$

Layer $l = 1,\ldots,L$
$z^{(0)} = x$ (input), $z^{(L)} = y$ (output)
$W^{(l)}$: weight matrix for layer l

Why Go Deep?

Exponential Efficiency:

Network function divides input space into regions exponential in depth, but only polynomial in width (Montúfar et al., 2014)

Universal Approximation:

Two-layer networks can approximate any function, but may require exponential hidden units

Hierarchical Feature Learning

Compositional Inductive Bias:

Higher-level objects composed of lower-level objects
Example: Image Recognition

- Early layers: detect edges
- Middle layers: combine edges into shapes (eyes, whiskers)
- Later layers: combine shapes into objects (cats)

Distributed Representations

Exponential Expressiveness:

M units can represent 2^M different features through **combinations** of activations

Example: Face recognition with 3 binary features (glasses, hat, beard) = 8 combinations vs. 8 separate units

Key Insight: Deep networks learn semantically meaningful internal representations at multiple levels of abstraction

Error Functions for Deep Networks

Regression Error Functions

Gaussian Assumption:

$$p(t|x, w) = N(t|y(x, w), \sigma^2)$$

Sum-of-Squares Error:

$$E(w) = \frac{1}{2} \Sigma\{y(x_n, w) - t_n\}^2$$

Equivalent to maximum likelihood with Gaussian noise

Multiple Outputs:

$$E(w) = \frac{1}{2} \Sigma ||y(x_n, w) - t_n||^2$$

Assumes independent outputs with shared noise variance σ^2

Variance Estimation:

$$\sigma^{2\star} = (1/N) \Sigma\{y(x_n, w\star) - t_n\}^2$$

Binary Classification

Bernoulli Distribution:

$$p(t|x, w) = y(x, w)^t\{1 - y(x, w)\}^{1-t}$$

Output activation: Logistic sigmoid

Cross-Entropy Error:

$$E(w) = -\Sigma\{t_n \ln y_n + (1-t_n) \ln(1-y_n)\}$$

where $y_n = y(x_n, w)$

Advantage: Faster training and better generalization than sum-of-squares for classification (Simard et al., 2003)

Multiple Binary Classifications

K Independent Binary Outputs:

$$E(w) = -\Sigma\Sigma\{t_{nk} \ln y_{nk} + (1-t_{nk}) \ln(1-y_{nk})\}$$

Each output has logistic sigmoid activation

Gradient-Based Optimization & Error Surfaces

Optimization Challenge

Goal: Find network parameters (weights and biases) that minimize error function E(w) for good test set generalization

Why Gradients?

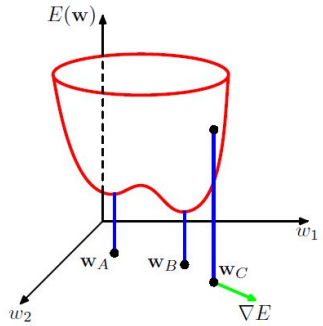
Direct error evaluation is very inefficient. Gradient information via backpropagation enables efficient optimization.

Computational Complexity

Method	Complexity
Without gradients	$O(W^3)$ function evaluations
With gradients	$O(W^2)$ steps
Each gradient eval.	W pieces of information

Backpropagation: Efficient technique to evaluate error function derivatives by flowing computations backward through the network

Error Surface Geometry



Stationary Points

Where ∇E(w) = 0:

- **Global minimum:** Smallest value across all w-space
- **Local minima:** Higher error values
- **Saddle points:** Mixed curvature

Symmetries: Two-layer network with M hidden units has M!2^M equivalent minima (permutations + sign flips)

Gradient Descent Algorithms

Iterative Weight Update

General Form:

$$w^{(\tau)} = w^{(\tau-1)} + \Delta w^{(\tau-1)}$$

Different algorithms use different choices for $\Delta w^{(\tau)}$

Batch Gradient Descent

Update Rule:

$$w^{(\tau)} = w^{(\tau-1)} - \eta \nabla E(w^{(\tau-1)})$$

- η: learning rate (controls step size)
- Move in direction of steepest descent
- Process entire training set each iteration
- Also called "batch methods"

Stochastic Gradient Descent (SGD)

Update Rule:

$$w^{(\tau)} = w^{(\tau-1)} - \eta \nabla E_n(w^{(\tau-1)})$$

where $E(w) = \sum E_n(w)$

- Update based on single data point
- One complete pass = training epoch
- Also called "online gradient descent"
- Handles data redundancy efficiently
- Can escape local minima

Mini-Batch Gradient Descent

Hybrid Approach:

Use small subset of data points per iteration

Algorithm 7.1: Stochastic Gradient Descent

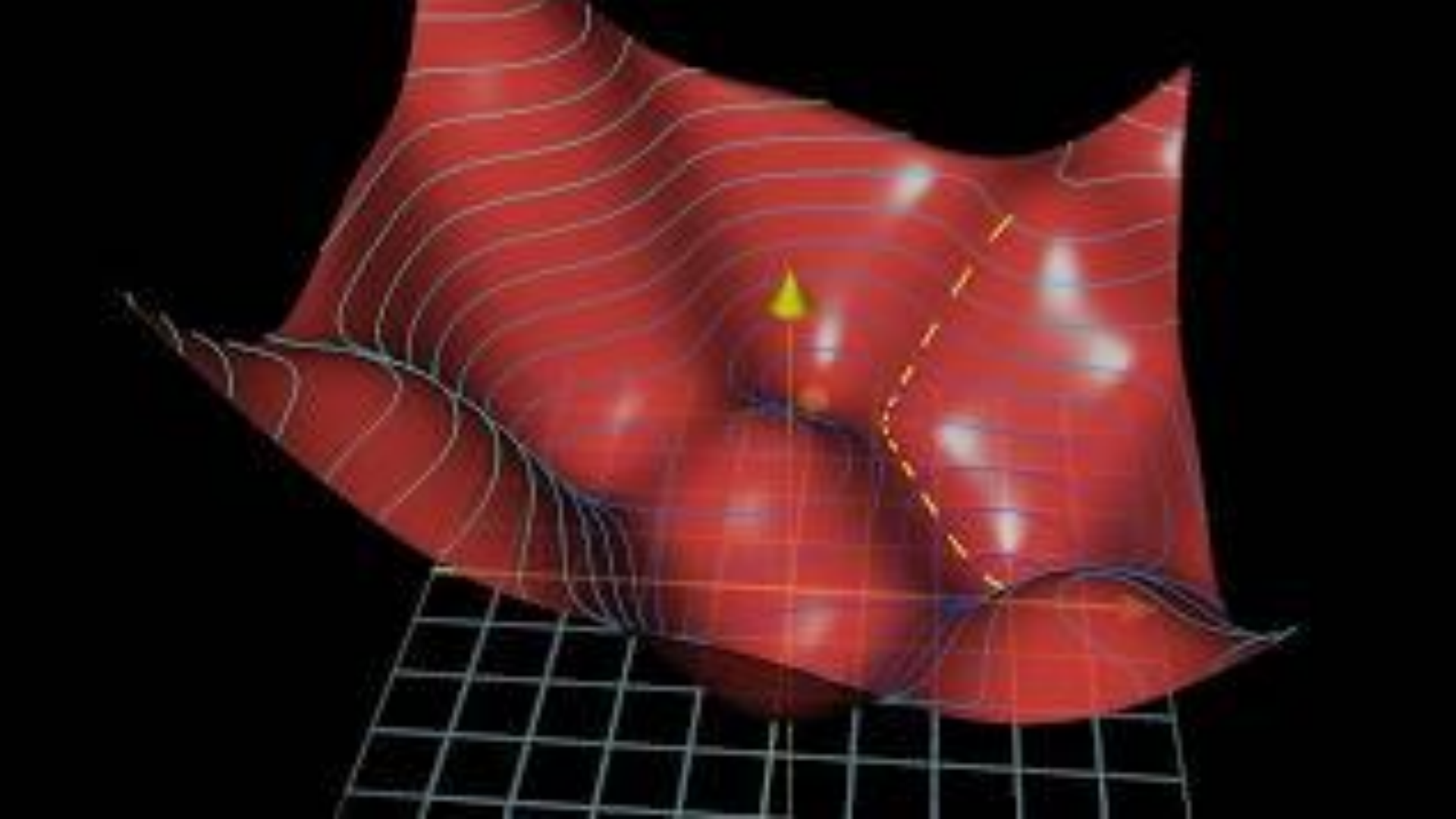
```
Input: Training set {1,...,N}, E_n(w), η, initial w
Output: Final weight vector w

n ← 1
repeat
  w ← w − η∇E_n(w)
  n ← n + 1 (mod N)
until convergence
return w
```

Mini-Batch Considerations

Gradient Noise: Error estimate $\sigma/\sqrt{N}$ (diminishing returns)
100× batch increase → only 10× error reduction

Hardware Efficiency: Powers of 2 work well (64, 128, 256)



Backpropagation Algorithm

Forward Propagation

General Feed-Forward Network:

$a_j = \Sigma_i w_{ji} z_i$ (pre-activation)

$z_j = h(a_j)$ (activation)

- z_i: activation from previous layer or input
- w_{ji}: weight connecting unit i to j
- $h(\cdot)$: nonlinear activation function
- Biases included via fixed +1 input

Error Signal δ

Definition:

$\delta_j \equiv \partial E_n / \partial a_j$

Error signal for unit j (often called "errors")

Output Units:

$\delta_k = y_k - t_k$

(for canonical link with sum-of-squares)

Backward Propagation

Chain Rule Application:

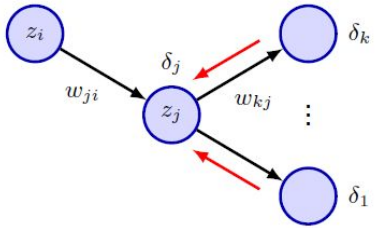
$\partial E_n / \partial w_{ji} = (\partial E_n / \partial a_j)(\partial a_j / \partial w_{ji})$

$= \delta_j z_i$

Backpropagation Formula:

$\delta_j = h'(a_j) \Sigma_k w_{kj} \delta_k$

Propagate δ's backwards from units k to unit j



Key Insight: Calculate δ for each unit, then apply $\partial E_n / \partial w_{ji} = \delta_j z_i$

Backpropagation Algorithm

Algorithm 8.1: Backpropagation

```
Input: Input vector x_n, Network parameters w,
       Error function E_n(w), Activation h(a)
Output: Derivatives {∂E_n/∂w_ji}

// Forward propagation
for j ∈ all hidden and output units do
  a_j ← Σ_i w_ji z_i
  z_j ← h(a_j)
end for

// Error evaluation
for k ∈ all output units do
  δ_k ← ∂E_n/∂a_k
end for

// Backward propagation
for j ∈ all hidden units do
  δ_j ← h'(a_j) Σ_k w_kj δ_k
  ∂E_n/∂w_ji ← δ_j z_i
end for

return {∂E_n/∂w_ji}
```

Algorithm Characteristics

Efficiency: Can be applied to general feed-forward networks with arbitrary topology, differentiable activations, and broad error functions

Key Steps

Phase	Operation
1. Forward	Compute all activations from input to output
2. Error	Calculate δ_k for output units
3. Backward	Propagate δ's recursively to hidden units
4. Derivatives	$\partial E_n/\partial w_{ji} = \delta_j z_i$ for all weights

Computational Complexity

O(W) per evaluation

Each gradient evaluation takes O(W) steps (where W is number of weights)

Batch Processing

For batch/mini-batch methods:

$$\partial E/\partial w_{ji} = \Sigma_n \partial E_n/\partial w_{ji}$$

Sum gradients over batch or mini-batch