Week 4


Stop using linear regression
You know what? I'm gonna start using linear regression even harder

Announcements

- Grades for HW1 are posted
- Next week is Exam 1
  - One double sided cheat sheet
  - Lecture slides weeks 1-4, HW1, all in-class labs

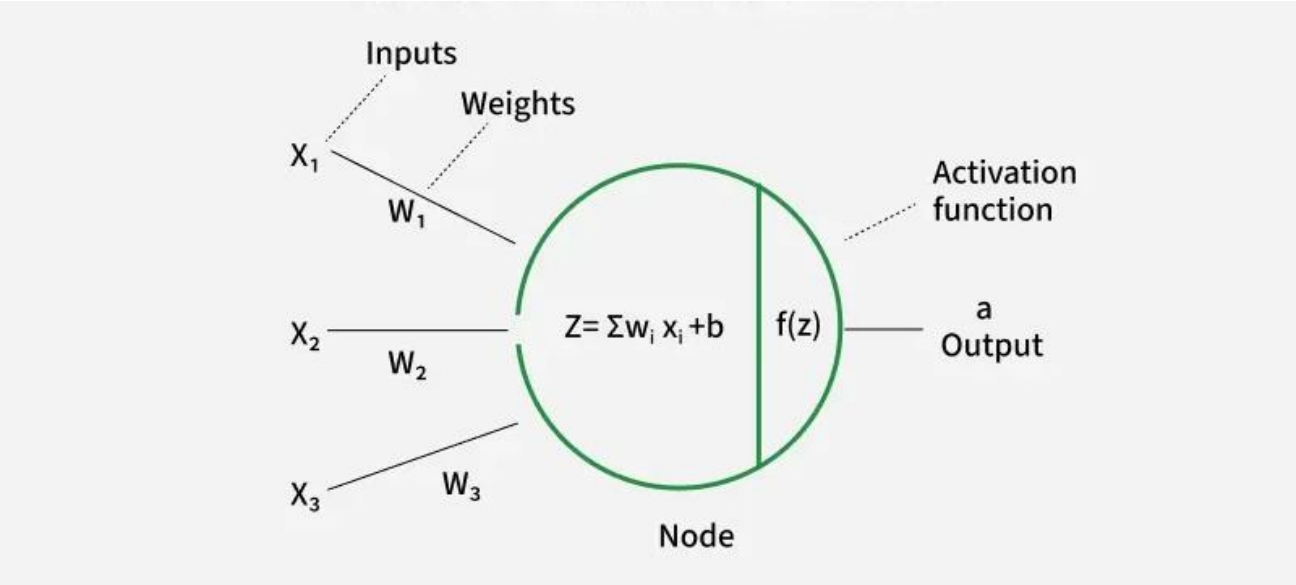
Recap: Gradient Descent

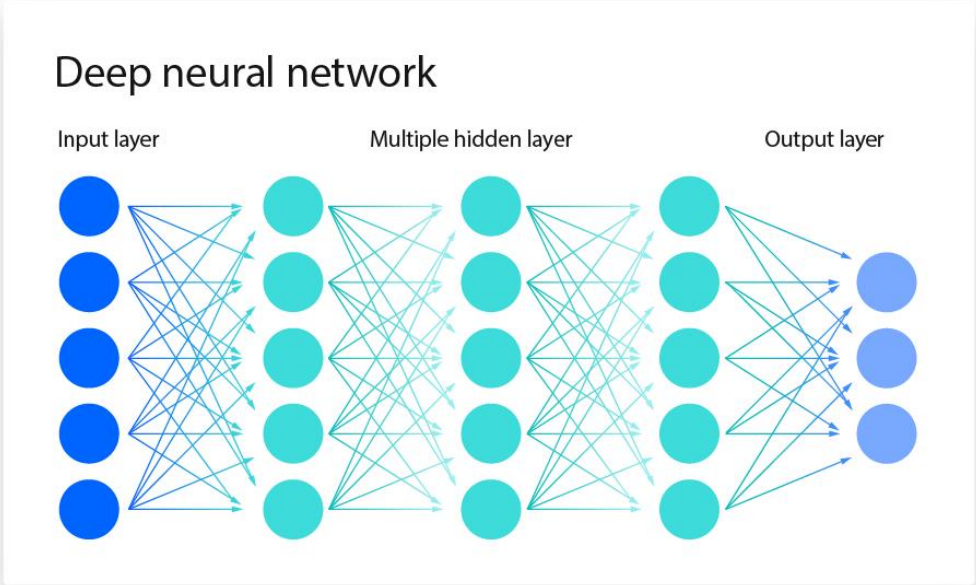
- Gradient Descent adjusts parameters w, b to minimize a cost function

$$w_j \leftarrow w_j - \alpha \frac{\partial J}{\partial w_j}, \quad b \leftarrow b - \alpha \frac{\partial J}{\partial b}$$

- But… for neural networks, the cost depends on many layers of weights!

Recap: NN

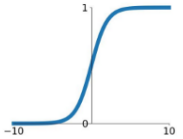




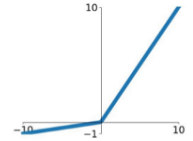
Neural Network (NN)
Broad term: any network of interconnected artificial neurons.
Includes many architectures:
Feedforward (MLP), CNNs, RNNs, Transformers, etc.

Activation Functions

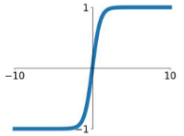
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



Leaky ReLU
$\max(0.1x, x)$

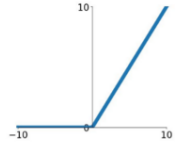


tanh
$\tanh(x)$

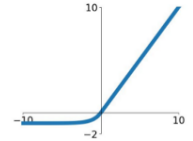


Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ReLU
$\max(0, x)$



ELU
$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$

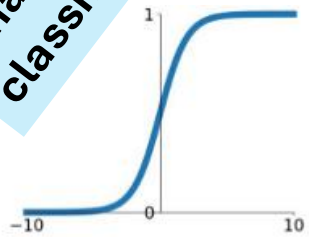


Activation Functions applications

Binary classification

Sigmoid

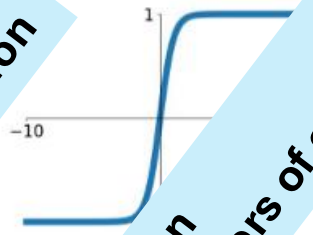
$$\sigma(x) = \frac{1}{1+e^{-x}}$$



Regression

tanh

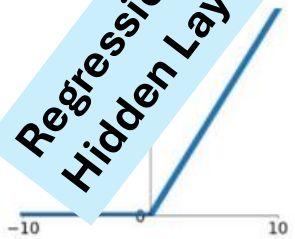
$$\tanh(x)$$



Regression
Hidden Layers of classification

ReLU

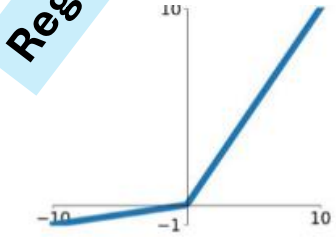
$$\max(0, x)$$



Regression

Leaky ReLU

$$\max(0.1x, x)$$

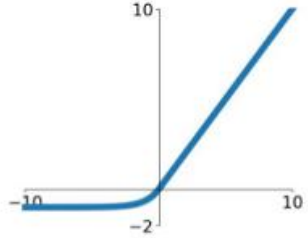


Maxout

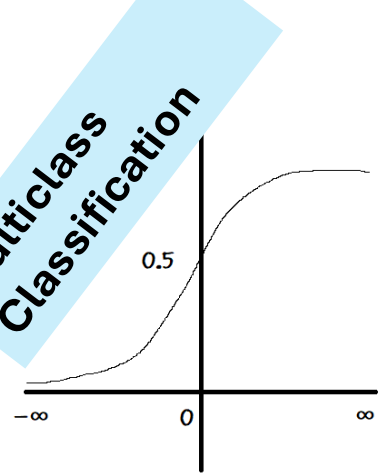
$$\max(w_1^T x + b_1, w_2^T x + b_2)$$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Multiclass Classification



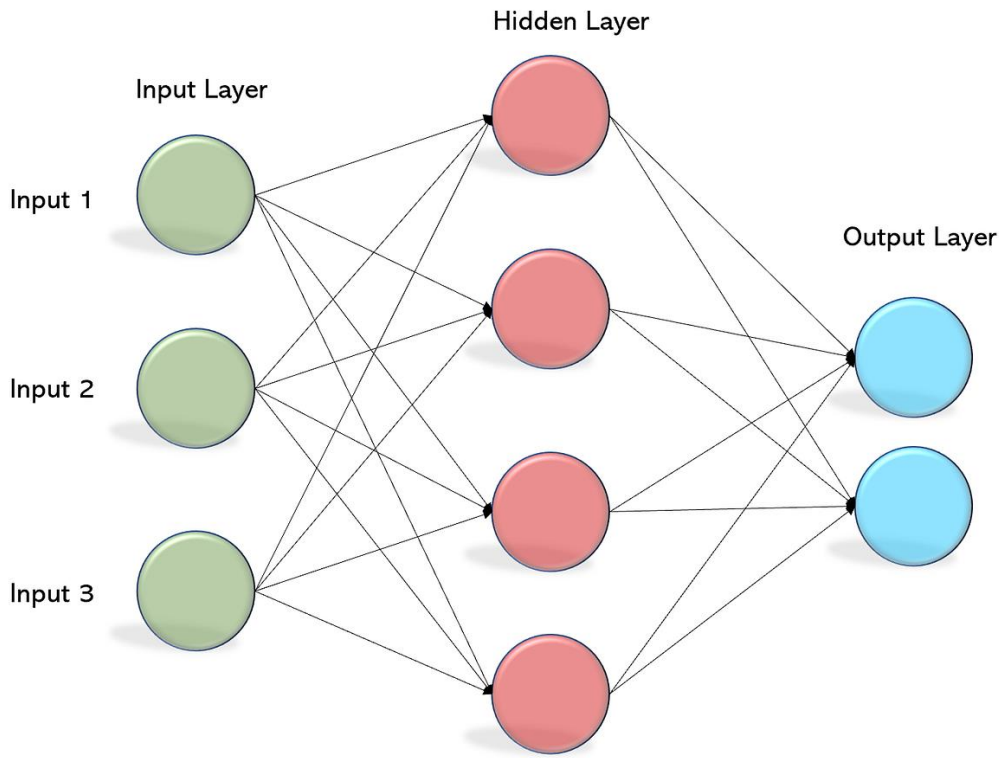
Softmax

Multilayer Perceptron (MLP)

A specific type of neural network.

Structure:

- Input layer -> hidden layer(s) -> output layer
- Fully connected (every neuron links to all in next layer)
- Learns via backpropagation + gradient descent



MLP is also referred as

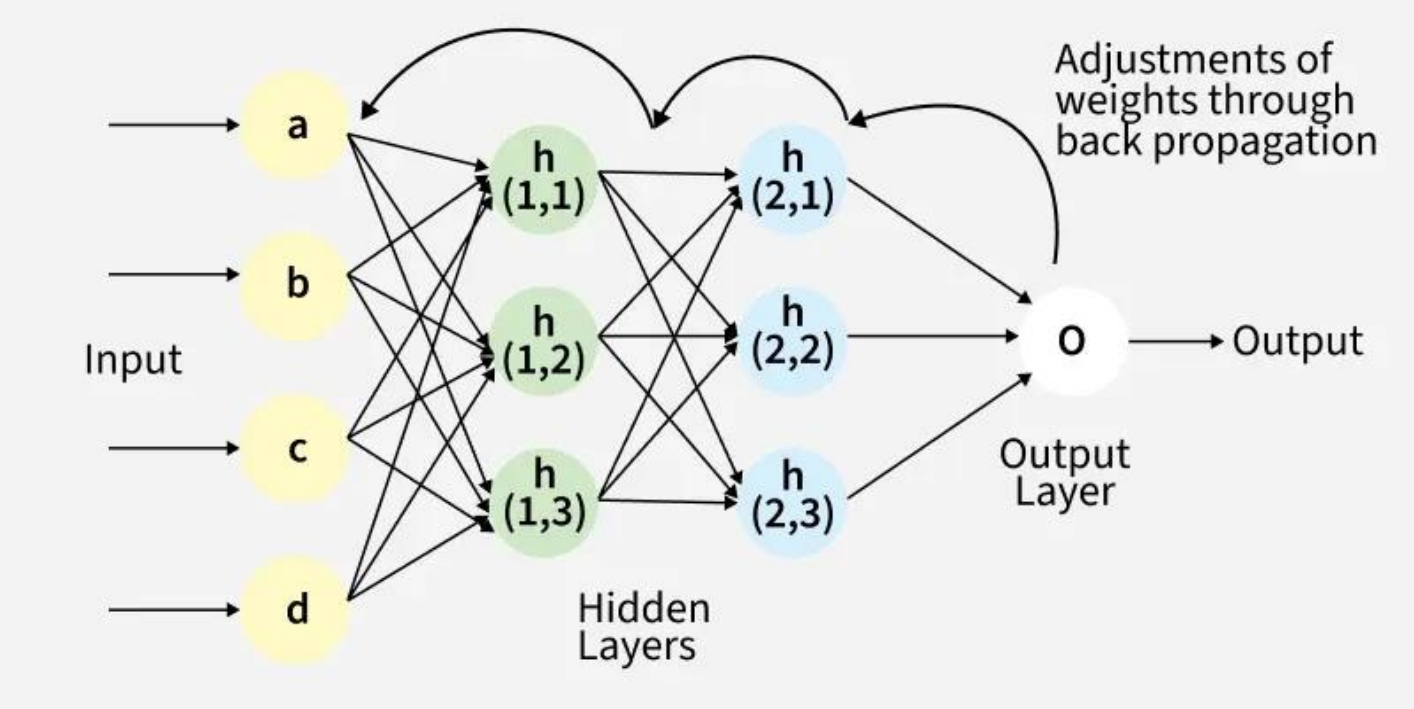
- **Feedforward Neural Network (FNN)**
  - Because information flows strictly forward (no loops).
- **Fully Connected Network (FCN)**
  - Emphasizes that each neuron in one layer connects to *all* neurons in the next.
- **Dense Network**
  - Term commonly used in libraries like Keras/TensorFlow (e.g., Dense layer).
- **Vanilla Neural Network**

The Challenge in Neural Networks

- Neural networks have layers of weights & activations
- Each weight contributes indirectly to the final output
- We need a systematic way to compute all gradients efficiently
- This is where Backpropagation comes in

What is Backpropagation?

- Algorithm to compute gradients for all weights in a neural network
- Uses chain rule of calculus to “propagate errors backwards”
- Foundation of training deep learning models.

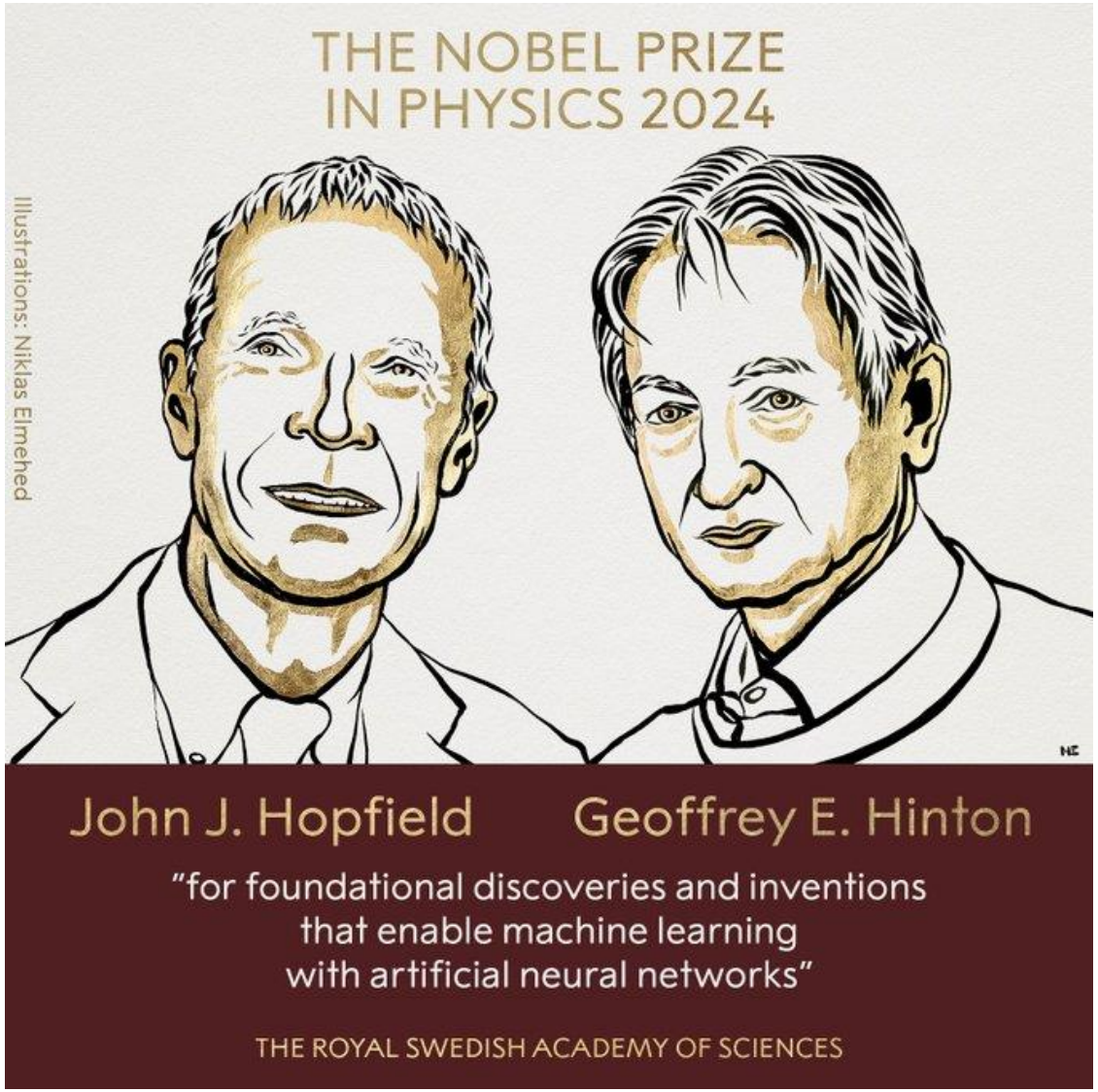


Letter | Published: 09 October 1986

Learning representations by back-propagating errors

David E. Rumelhart, Geoffrey E. Hinton & Ronald J. Williams

Nature **323**, 533–536 (1986) | Cite this article



Step-by-Step Flow

- Forward Pass:

  In this phase we feed the inputs through the network,
  make a prediction and measure its error with respect to the true label.
  Compute Loss (e.g. Mean Squared Error (MSE) or Cross-Entropy)

- Backward Pass:

  Calculate gradients of loss w.r.t. last layer weights
  Propagate gradients backwards using chain rule

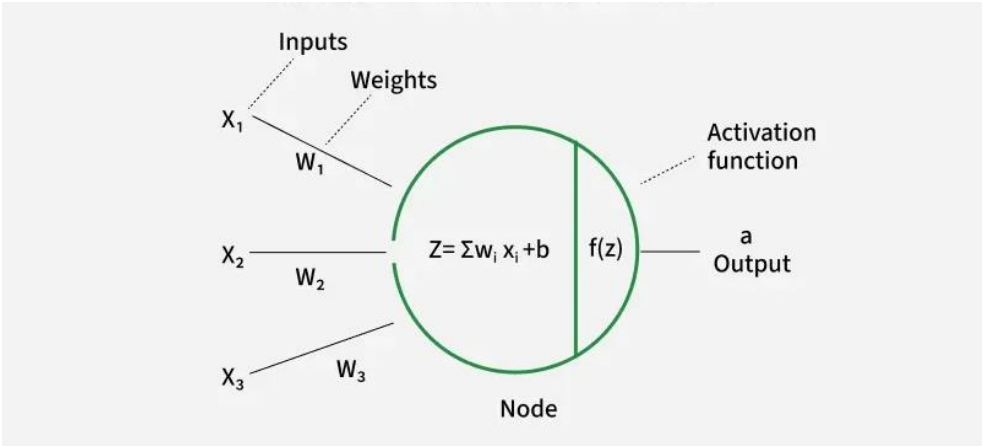
- Update Weights:

  Apply gradient descent updates to all weights.

Back Propagation: One Neuron

$$w := w - \alpha \frac{\partial L}{\partial w}$$

$$b := b - \alpha \frac{\partial L}{\partial b}$$



- Pre-activation: $z = w^\top x + b = \sum_{i=1}^{d} w_i x_i + b$
- Activation: $a = \phi(z)$
- Loss: $L = \ell(a, y)$

- $\frac{\partial L}{\partial a} = \frac{\partial \ell(a,y)}{\partial a}$ (depends on the loss)
- $\frac{\partial a}{\partial z} = \phi'(z)$ (depends on the activation)
- $\frac{\partial z}{\partial w_i} = x_i$ because $z = \sum_j w_j x_j + b$
- $\frac{\partial z}{\partial b} = 1$

$$\frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial w_i}; \qquad \frac{\partial L}{\partial b} = \frac{\partial L}{\partial a} \cdot \frac{\partial a}{\partial z} \cdot \frac{\partial z}{\partial b}.$$

Back Propagation: generally

$$w := w - \alpha \frac{\partial L}{\partial w}$$

$$b := b - \alpha \frac{\partial L}{\partial b}$$

$$\frac{\partial L}{\partial w_{ij}^{(l)}} = \frac{\partial L}{\partial y_j^{(l)}} \cdot \frac{\partial y_j^{(l)}}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial w_{ij}^{(l)}}$$

Where:

- i corresponds to the previous layer (input layer for this transformation)
- j corresponds to the next layer (output layer of the transformation)
- any weight $w_{ij}^{(l)}$,which connects neuron i in layer $(l-1)$ to neuron j in layer l.

Regularization

Making models more generalizable & preventing overfitting

Custom Loss Functions in ML

- In **machine learning**, the loss function defines *what the model should optimize*
- While standard losses exist (MSE, Cross-Entropy, Hinge, etc.), it is **common to design custom loss functions** for specific problems

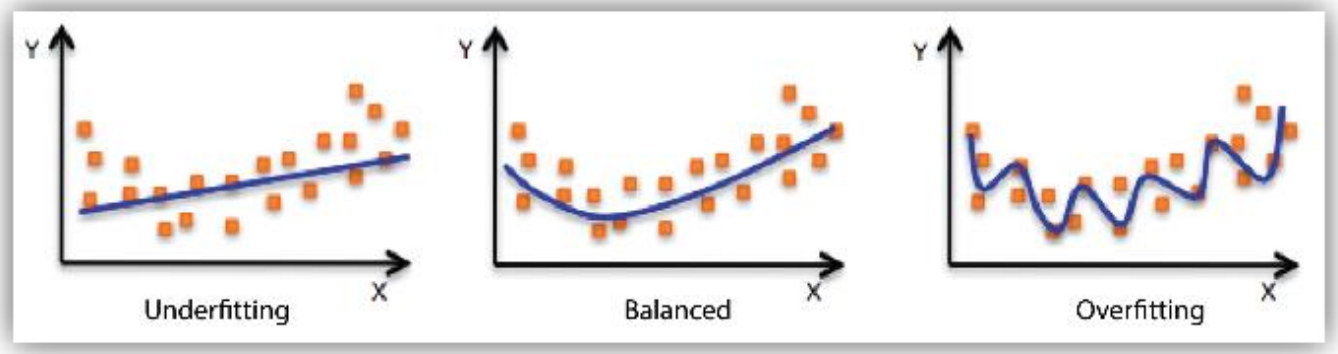
- Standard losses may not fully capture business or domain needs.
- Customization allows aligning training with the *real evaluation metric*.

Example: Credit Risk Scoring

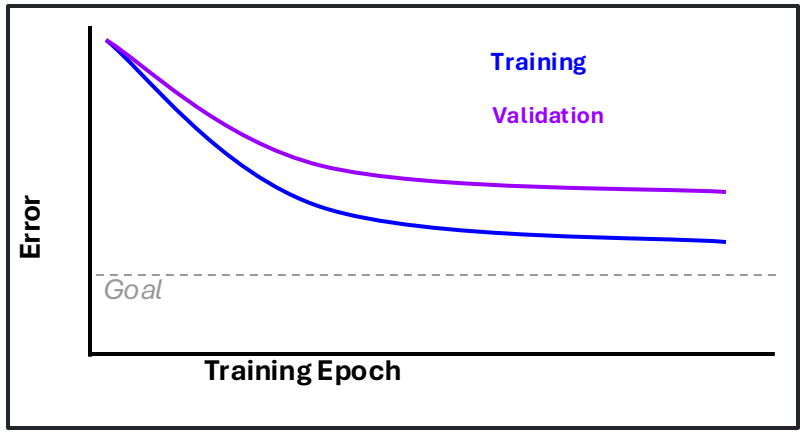
Add a penalty term to the loss function that would penalize model for approving risky borrowers more than non-risky borrowers.

$$\text{Loss} = \text{MSE}(\text{TrueLoan}, \text{PredictedLoan}) + \text{Penalty}(\text{Risky Borrower})$$

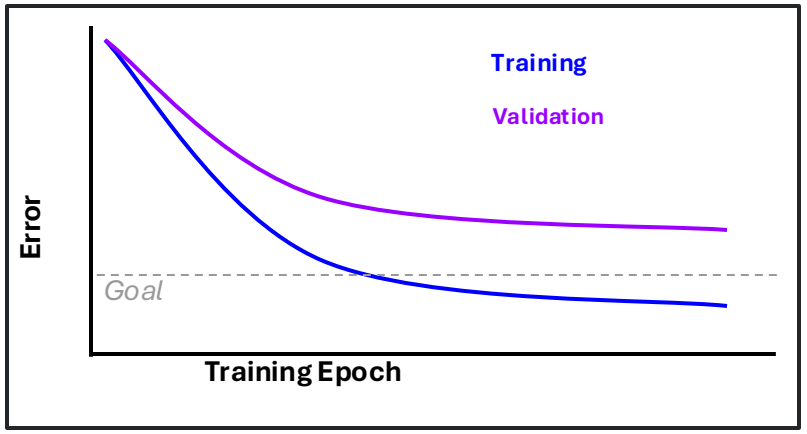
Why Regularization?



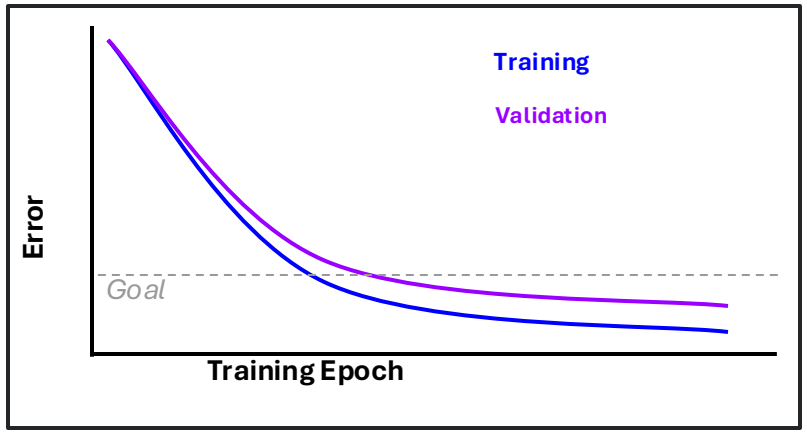
- Overfitting: model learns noise in training data
- Leads to poor performance on unseen data
- Goal: Encourage simpler models that generalize better



Underfitting



Overfitting



Good Fit

L2 Regularization (Ridge)

- Adds penalty proportional to sum of squared weights
- Encourages smaller weights, spreads out importance
- Helps with multicollinearity

L2 Regularization

$$\text{Cost} = \underbrace{\sum_{i=0}^{N}(y_i - \sum_{j=0}^{M} x_{ij}W_j)^2}_{\text{Loss function}} + \underbrace{\lambda \sum_{j=0}^{M} W_j^2}_{\text{Regularization Term}}$$

Example: Predicting house prices with many correlated features

Ridge spreads weights evenly across correlated features

L1 Regularization (Lasso)

- Adds penalty proportional to sum of absolute weights
- Encourages sparsity (some weights = 0)
- Useful for feature selection

L1 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

- Example: Predicting customer churn with 100+ features
- Lasso eliminates irrelevant predictors, keeps only key ones

Comparison: L1 vs L2

$$w := w - \alpha \frac{\partial L}{\partial w}$$

$$b := b - \alpha \frac{\partial L}{\partial b}$$

- L1 (Lasso): sparse weights, feature selection
- L2 (Ridge): small weights, stability, no sparsity
- Elastic Net: combination of both penalties

L1 Regularization

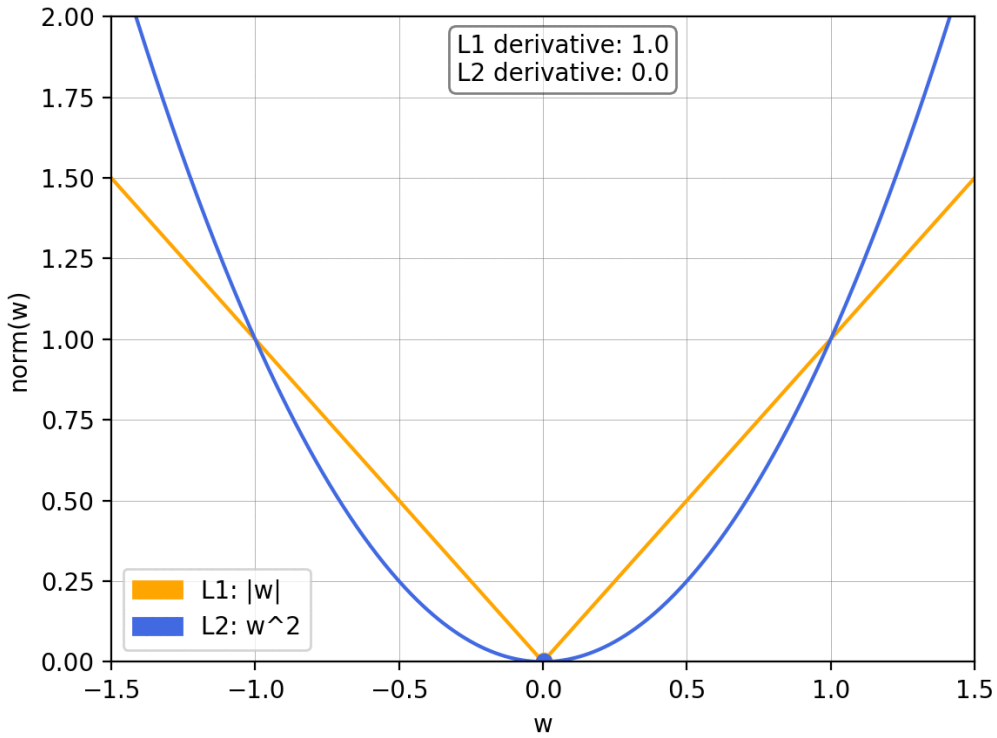
$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} |W_j|$$

L2 Regularization

$$\text{Cost} = \sum_{i=0}^{N} (y_i - \sum_{j=0}^{M} x_{ij} W_j)^2 + \lambda \sum_{j=0}^{M} W_j^2$$

Loss function

Regularization Term



Regularization in Neural Networks

- Neural Networks are prone to overfitting
- L1/L2 weight penalties

Other strategies:

- Dropout: randomly drop neurons during training
- Early stopping: stop training before overfitting
- Data augmentation: increase training diversity

Dropout Example

- Each epoch: randomly ignore some neurons
- Prevents co-adaptation
- Improves robustness

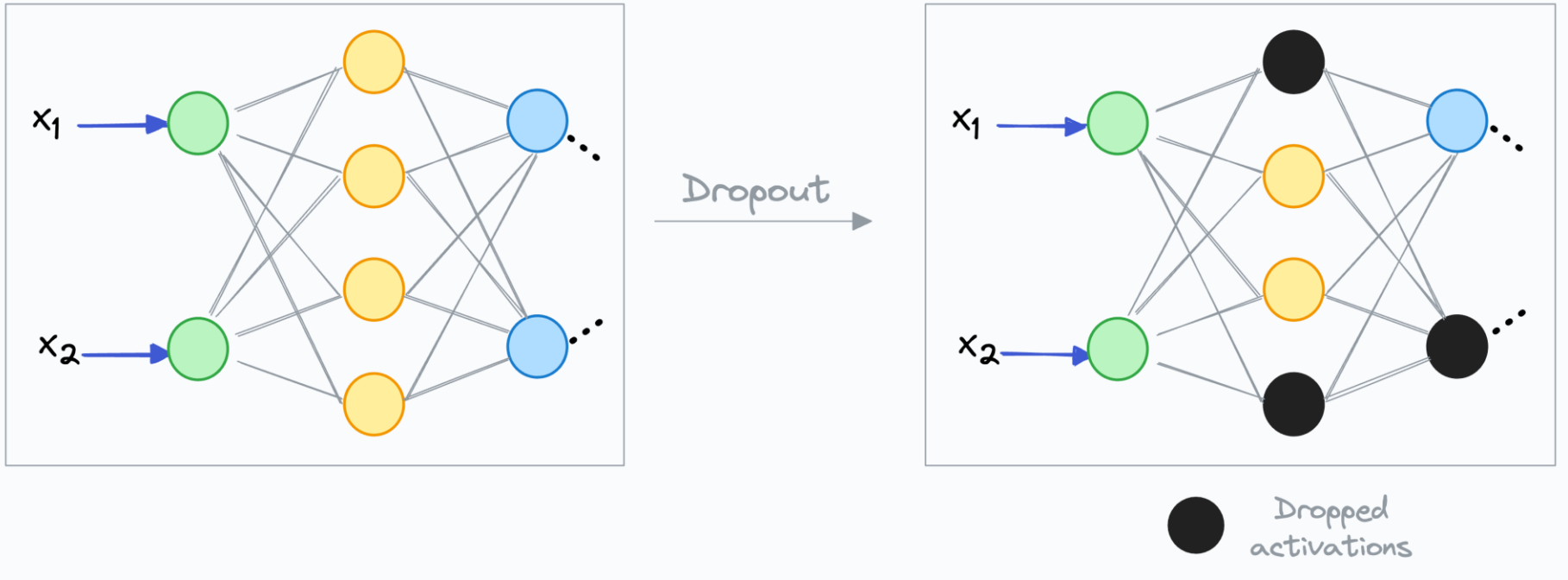
Dropout

class **torch.nn.Dropout**(***p***=*0.5*, ***inplace***=*False*)

During training, randomly zeroes some of the elements of the input tensor with probability `p`.

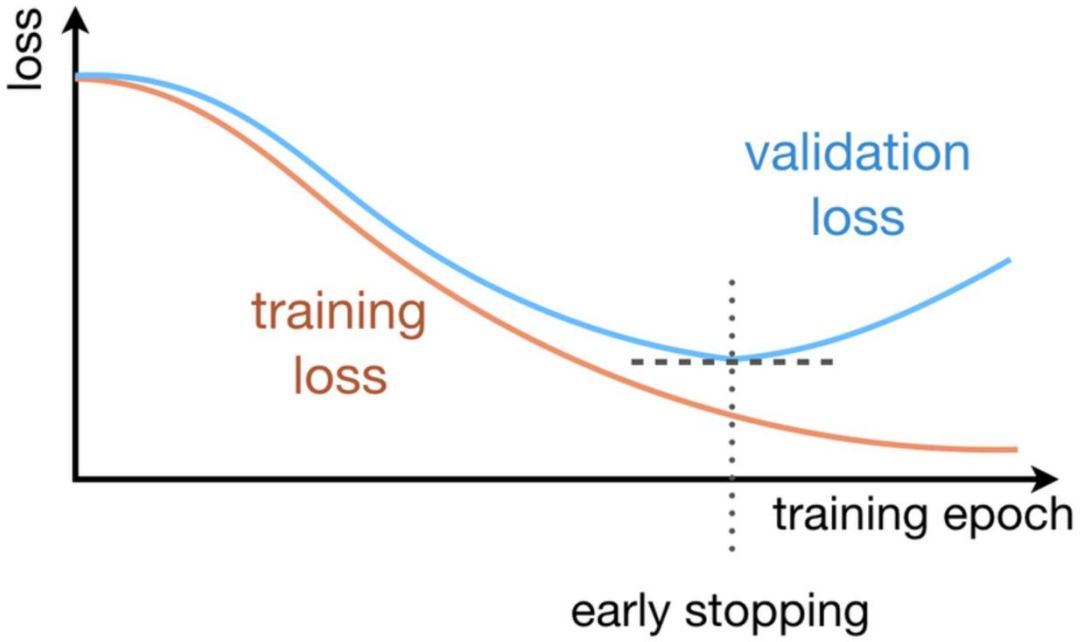
Examples:

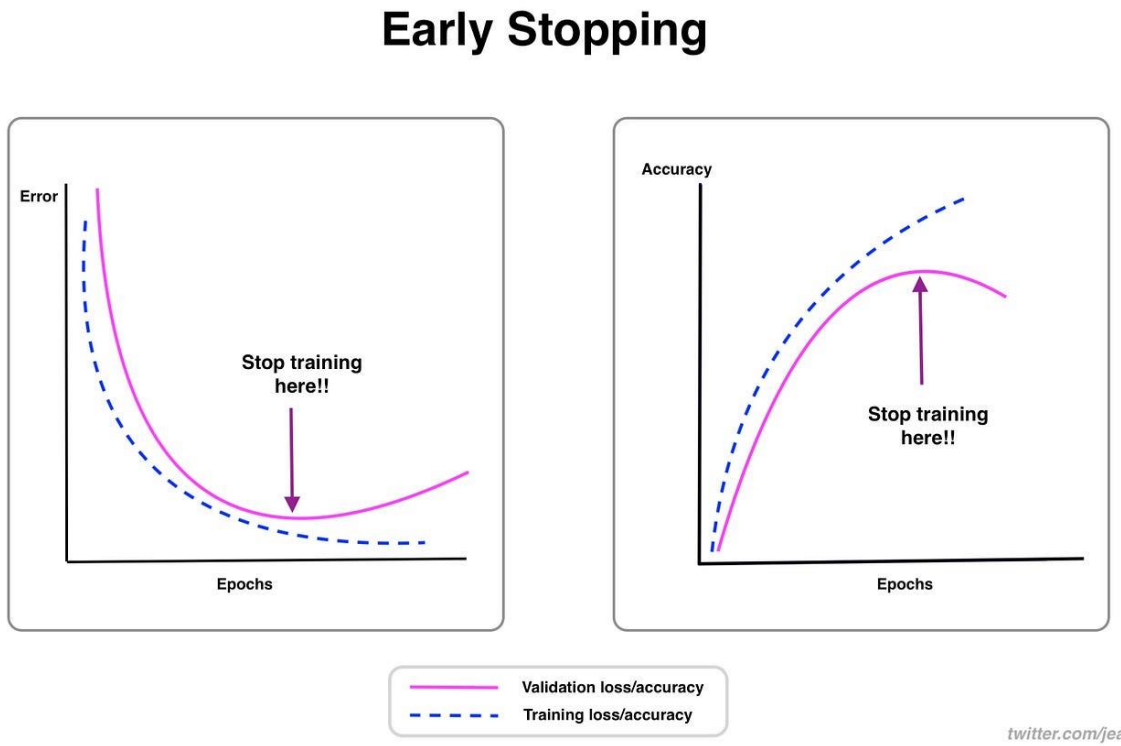
```
>>> m = nn.Dropout(p=0.2)
>>> input = torch.randn(20, 16)
>>> output = m(input)
```



Early Stopping

- Monitor validation error
- Stop training when validation error starts increasing
- Prevents overfitting without changing loss function
- Saves compute





Summary

- Regularization = techniques to reduce overfitting
- L1: sparsity, feature selection
- L2: small weights, stability
- Dropout, early stopping, augmentation
- Key idea: balance model complexity and generalization