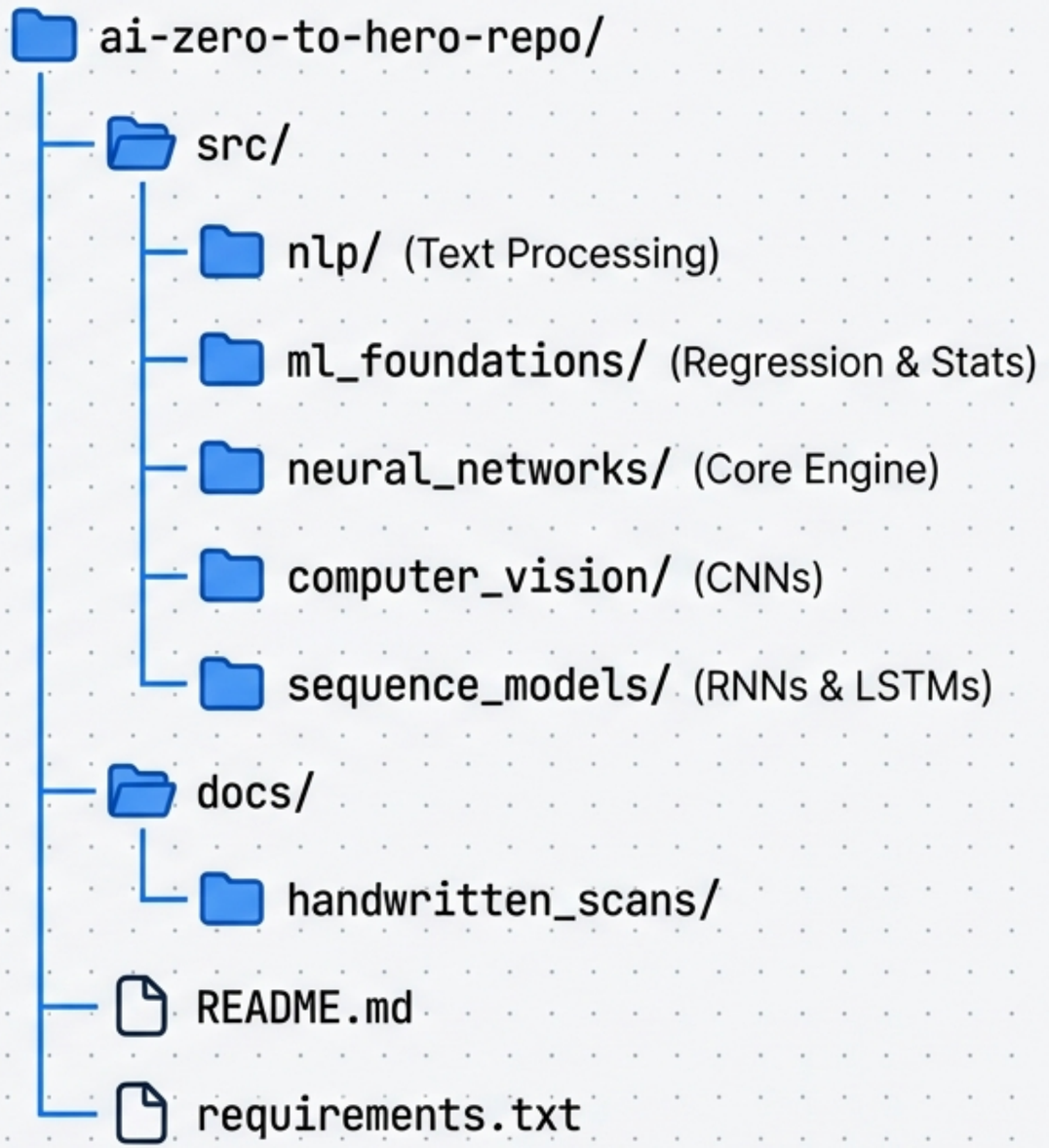
From Handwritten Notes to Code

A Zero-to-Hero Data Science Repository Architecture

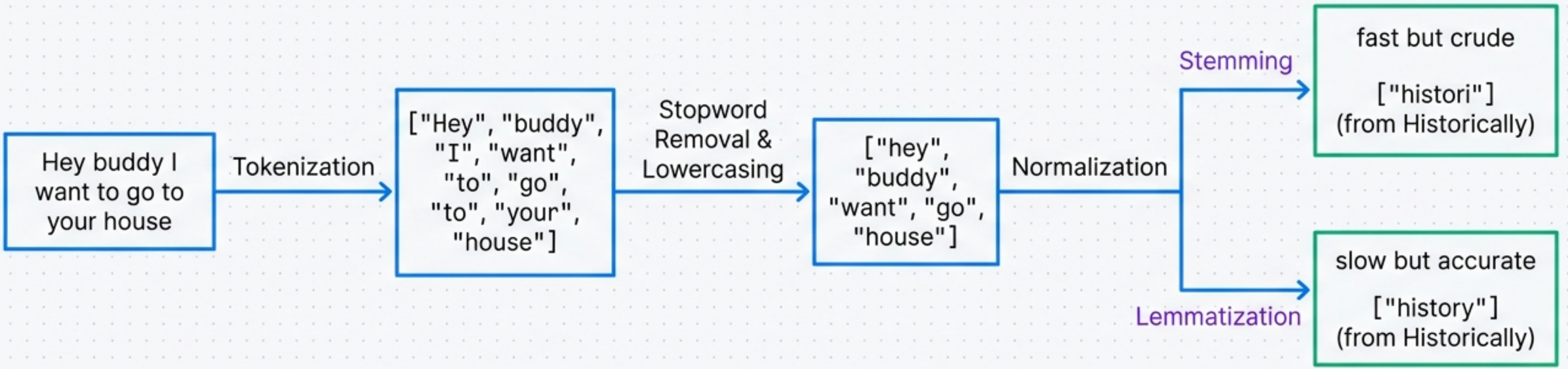
Objective: Digitizing 43 pages of fundamental AI theory—spanning NLP, Deep Learning, and Computer Vision—into a structured, production-ready codebase.

Stack: Python / TensorFlow / Scikit-learn



Module 1: NLP Preprocessing Pipeline

/src/nlp/preprocessing.py




```python
import nltk
from nltk.stem import PorterStemmer

text = "Historically, data science is interesting."
tokens = nltk.sent_tokenize(text)
# Stemming output: ['Histori', 'data', 'scienc', 'is', 'interest']
```

Module 2: Vectorization Strategies

`/src/nlp/vectorization.py`

One Hot Encoding (OHE)

	1	2	3	4	5	6	7	8	9
A	1	0	0	0	0	0	0	0	0
man	0	1	0	0	0	0	0	0	0
eat	0	0	1	0	0	0	0	0	0
food	0	0	0	1	0	0	0	0	0
cat	0	0	0	0	1	0	0	0	0
ppl	0	0	0	0	0	1	0	0	0
watch	0	0	0	0	0	0	1	0	0
KRISH	0	0	0	0	0	0	0	1	0
YT	0	0	0	0	0	0	0	0	1

Binary representation. No semantic meaning. Creates high sparsity.

Bag of Words (BOW)

	good	boy	girl
Doc 1	1	1	0
Doc 2	1	0	1

Frequency counting. Ignores context/order.

TF-IDF

$$\text{Weight} = \text{TF} * \text{IDF}$$

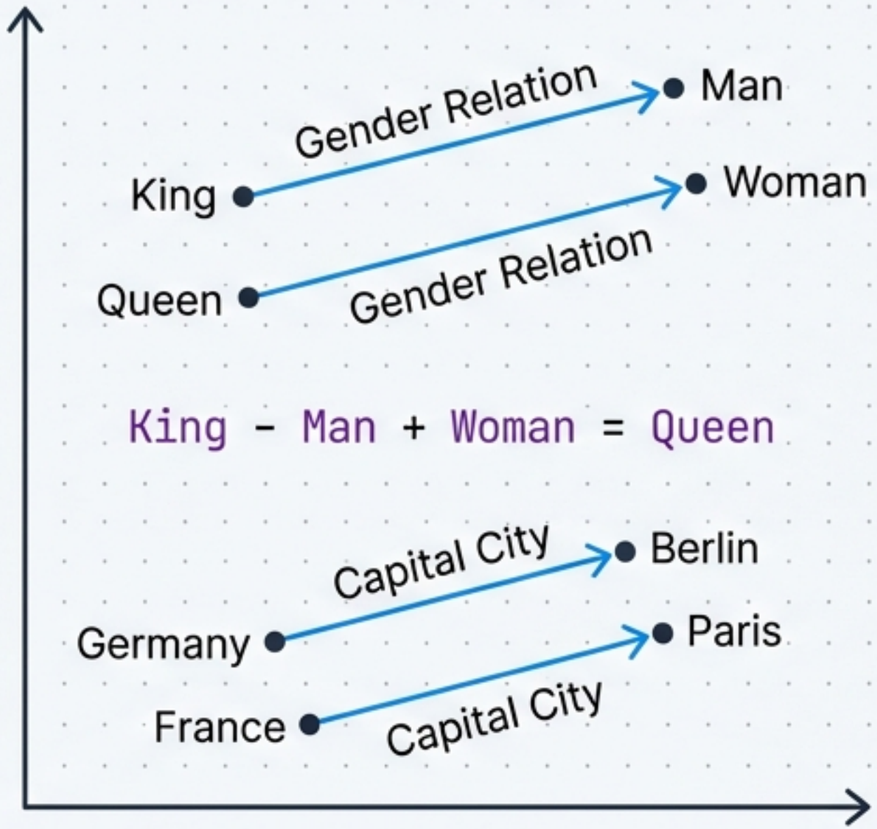
$$W_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

Weighted importance. Penalizes common words (e.g., 'the') and highlights unique terms.

Word Embeddings & Similarity

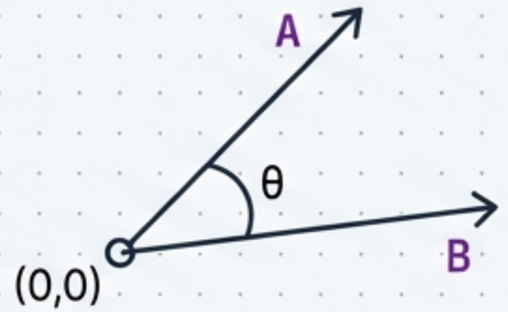
Mapping semantics to geometric space

Vector Space Representation



King - Man + Woman = Queen

Cosine Similarity



Cosine Similarity measures the angle, not magnitude.

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|}$$

Key:

θ = 0° → 1 (Identical)

θ = 90° → 0 (Unrelated)

θ = 180° → -1 (Opposite)

The Engine of Learning: Cost Functions

/src/ml_foundations/linear_regression.py

Hypothesis & Cost Function Definition

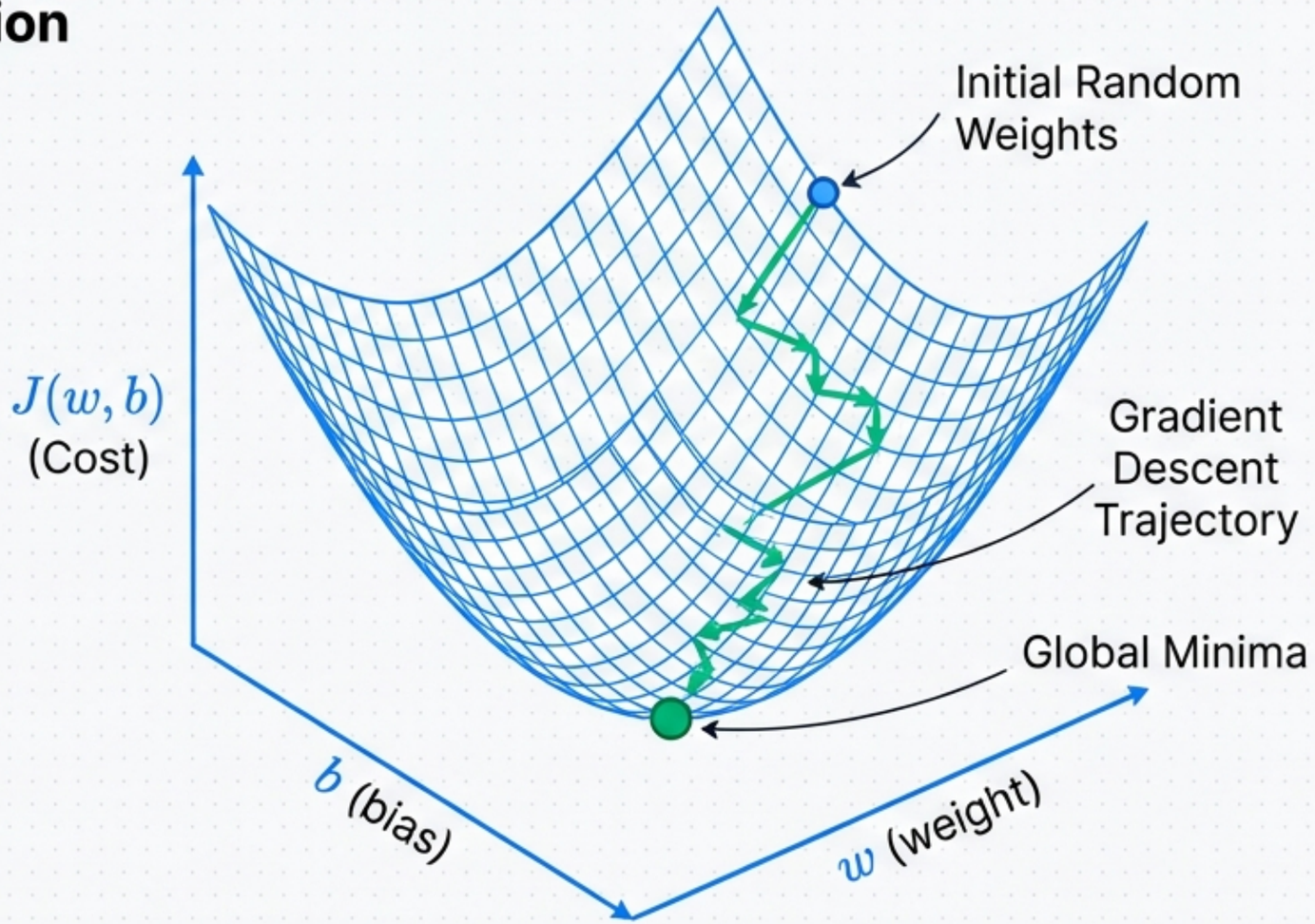
Hypothesis:

$$f_{w,b}(x) = wx + b$$

Cost (MSE):

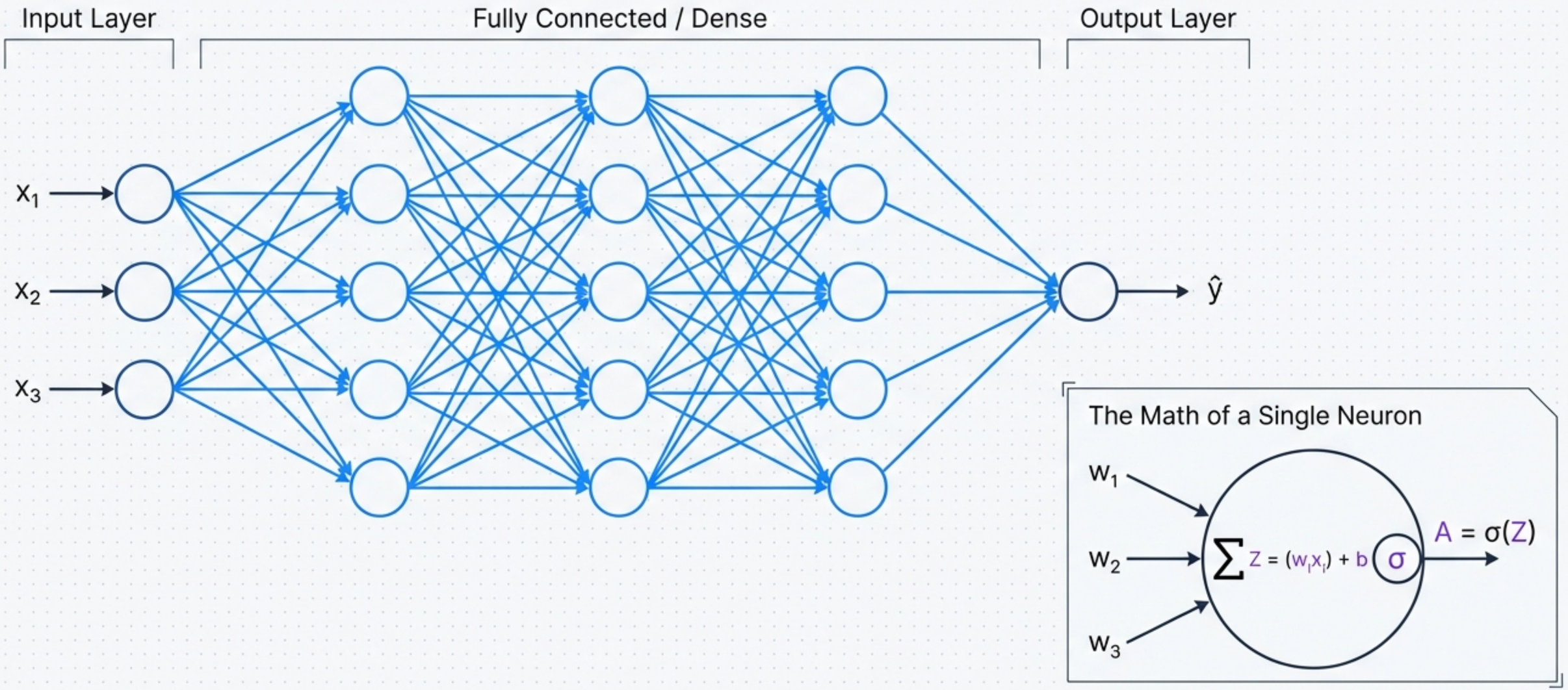
$$J(w,b) = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$

The 1/2m term simplifies the derivative calculation.



Neural Network Architecture: The Perceptron

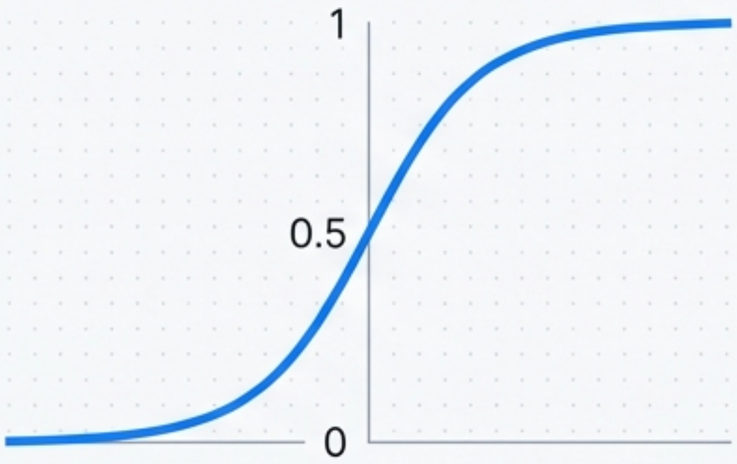
From biological inspiration to digital logic



Activation Functions

introducing non-linearity

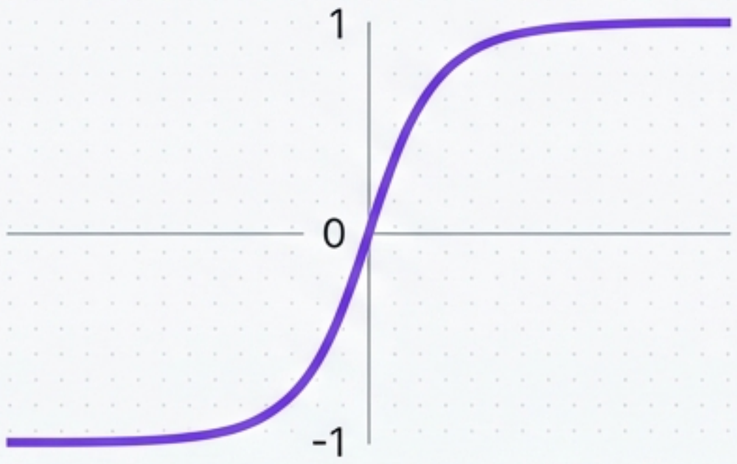
Sigmoid



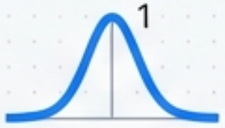
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Used for binary output.
Problem: Vanishing Gradient.

Tanh

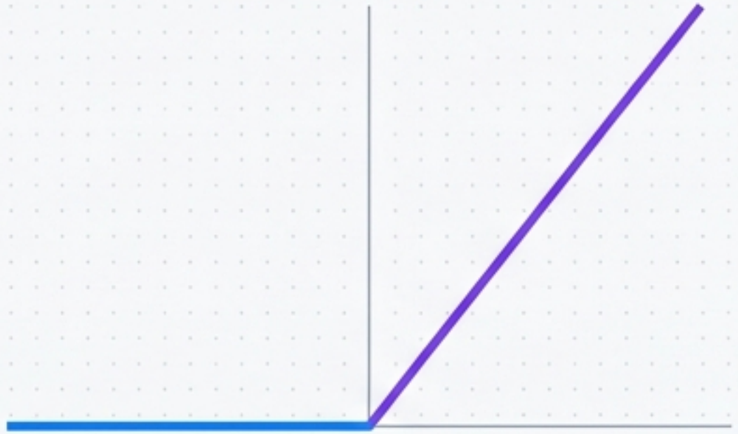


$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

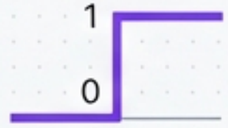


Zero-centered.

ReLU (Rectified Linear Unit)



$$f(x) = \max(0, x)$$



Standard for hidden layers.
Solves vanishing gradient.
Efficient.

Backpropagation & The Chain Rule

The mathematical derivation of learning

Change in Loss w.r.t Prediction

$$\frac{\partial L}{\partial w} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w}$$

Input value (x)

Derivative of Activation (e.g., Sigmoid')

Weight Update Rule

$$w_{new} = w_{old} - \eta \cdot \frac{\partial L}{\partial w}$$

(Where η = Learning Rate)

Loss Functions

Measuring the error for different tasks

Regression (Predicting Numbers)

JetBrains Mono

Mean Squared Error (MSE)

$$L = \frac{1}{N} * \sum (y - \hat{y})^2$$

Penalizes outliers heavily due to squaring.

JetBrains Mono

Huber Loss

Best of both worlds (MSE + MAE).

Classification (Predicting Categories)

JetBrains Mono

Binary Cross Entropy (Log Loss)

$$L = -[y * \log(\hat{y}) + (1-y) * \log(1-\hat{y})]$$

JetBrains Mono

Categorical Cross Entropy

$$L = -\sum y_i * \log(\hat{y}_i)$$

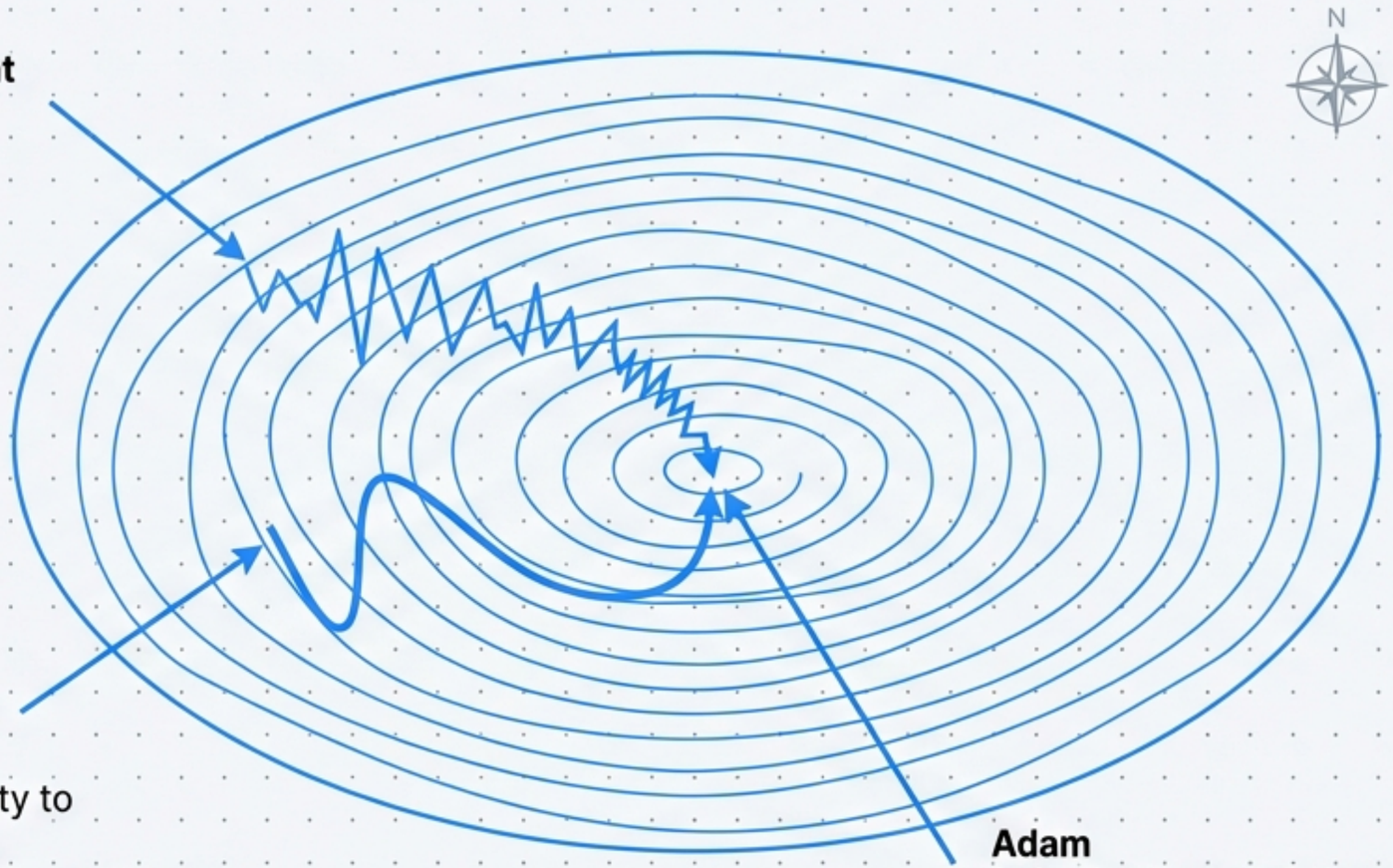
Used with Softmax output for multi-class problems.

Optimizers: Navigating the Loss Landscape

SGD, Momentum, and Adam

Stochastic Gradient Descent (Noisy)

High variance updates cause zigzag path.



SGD + Momentum (Velocity)

Accumulates velocity to dampen oscillation.

$v_t = \gamma v_{t-1} + \eta \nabla L$; $w_t = w_{t-1} - v_t$

Adam

(Adaptive Moment Estimation)

`Momentum + RMSProp`

Analogy: Ball on a Hill

Imagine a ball rolling down a hill.

SGD is like a ball being kicked in random directions at each step.

Momentum adds velocity, so the ball builds up speed and keeps moving in the general downhill direction, dampening sideways movements.

Adam combines momentum with adaptive learning rates for each parameter, enabling fast and efficient convergence.

Computer Vision: CNN Architecture

/src/computer_vision/cnn.py

Convolution Operation

1	0	-1	0	1	0
2	0	-2	2	0	1
1	0	-1	1	2	0
0	1	0	2	0	1
0	0	3	4	1	4
1	0	2	0	1	0

Input Image (6x6)

-4	-4	0	2
-2	-4	6	0
-5	2	0	2
-4	0	2	1

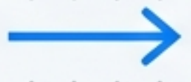
Feature Map (4x4)

Input * Filter = Feature Map

Max Pooling

4	2	4	5
6	1	8	2
1	0	2	1
3	5	7	3

Max Pooling Input (4x4)



6	8
5	7

Max Pooling Output (2x2)

Reduces dimensionality.
Translation invariant.

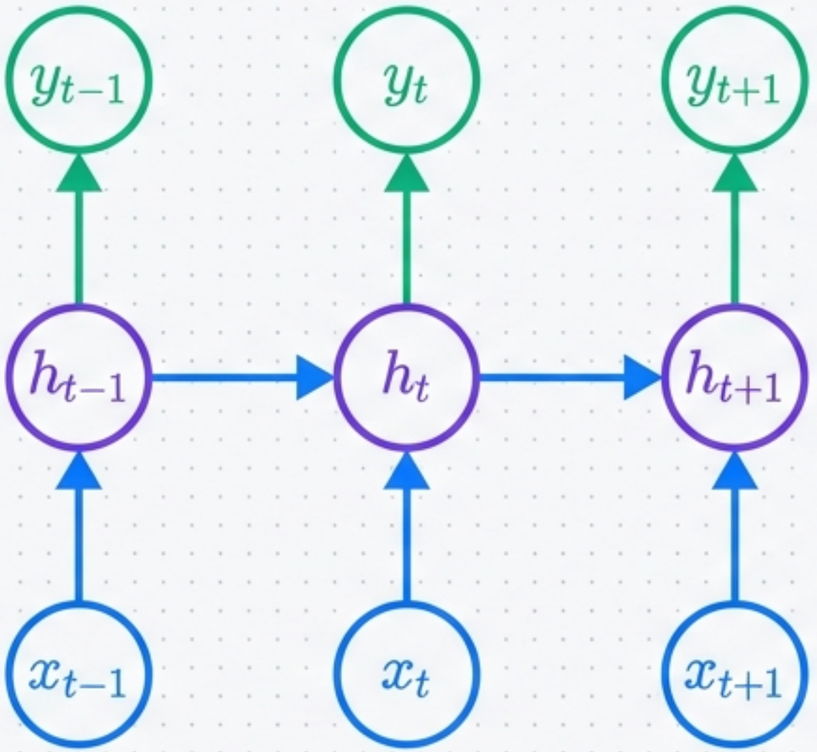
$$n_{out} = \left\lfloor \frac{n_{in} + 2p - f}{s} \right\rfloor + 1$$

(Label variables: Padding p, Filter f, Stride s)

Sequence Models: Recurrent Neural Networks

Processing time-series data and temporal dependencies in Inter

Unrolled RNN Diagram



Weights are shared across time steps

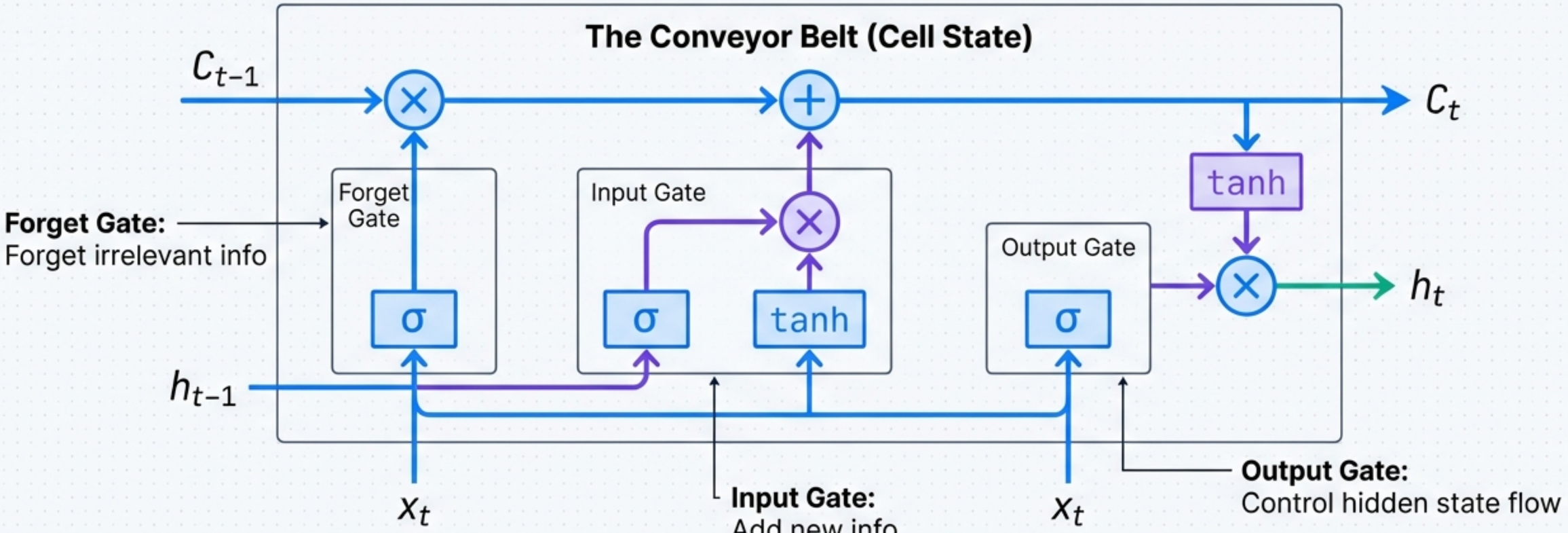
The Vanishing Gradient Problem

Gradients shrink exponentially during **Backpropagation Through Time (BPTT)**, causing the network to forget early inputs.

Gradient Flow (BPTT)

Long Short-Term Memory (LSTM)

The solution to short-term memory



Core Mechanism: The Cell State acts as a superhighway for gradients, preventing them from vanishing.

Implementation: The Keras Builder

Synthesizing theory into runnable code

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout

# Initialize the Model
classifier = Sequential()

# Input Layer & First Hidden Layer
# Units: 6 neurons, Activation: ReLU (for efficiency)
classifier.add(Dense(units=6, activation='relu', input_dim=11))
classifier.add(Dropout(0.3)) # Regularization to prevent overfitting

# Output Layer
# Units: 1 neuron, Activation: Sigmoid (for binary probability)
classifier.add(Dense(units=1, activation='sigmoid'))

# Compilation
# Optimizer: Adam (Adaptive Learning Rate)
# Loss: Binary Crossentropy
classifier.compile(optimizer='adam',
                   loss='binary_crossentropy',
                   metrics=['accuracy'])
```

Dropout: Prevents overfitting by randomly dropping neurons during training, forcing the network to learn more robust features (Referencing Vanishing Gradient discussion).

Sigmoid Activation: Compresses output between 0 and 1, suitable for binary probability (Referencing LSTM Output Gate).

Adam Optimizer: Adaptive Learning Rate method that efficiently adjusts the gradient descent steps (Referencing Gradient Flow and BPTT).

Repository Roadmap & Next Steps

Status: Documentation Complete

The Artifact

ai-zero-to-hero

build passing | coverage 100%

- requirements.txt
- README.md
- /src modules

Future Roadmap: Next Release

Future Roadmap: Next Release

- [x] **Bi-Directional LSTMs**: Processing sequence data in both directions.
- [x] **Encoder-Decoder Models**: Seq2Seq architectures for translation. (Reference: Advanced Notes)
- [x] **Transformers & BERT**: The foundation of modern LLMs. (Reference: Transformers & BERT notes)
- [ ] Fine-tuning Pre-trained Models.
- [ ] Deployment Strategies.

Ready for Commit.