Applying Deep Learning to Inverse Kinematics

Deep Learning Mini Project

December 9, 2025

Section: Universal Approximation Theorem

Universal Approximation Theorem

The foundation for learning inverse kinematics with neural networks

Universal Approximation Theorem

Core Concept

Universal approximation theorem proves that for any continuous function, there exists a network that can approximate this function to any specified precision.

Key Implications for Inverse Kinematics:

- IK is a continuous mapping (given constraints)
- Neural networks have sufficient capacity to learn this mapping
- With proper architecture and training, we can achieve arbitrary accuracy
- This justifies using NNs instead of classical solvers

The theorem provides theoretical grounding for our practical approach.

robot1.png

Kinematics Fundamentals

Understanding the geometry of robot motion

Two-Link Planar Manipulator

Problem Setup

For the 2-link planar robot in the figure, let:

- Link lengths: a_1, a_2
- Joint angles: θ_1, θ_2
- End-effector position: (x, y) in the base frame

This simple case demonstrates the core IK concepts that scale to complex robots.

Forward Kinematics (2-Link)

From geometry of the two revolute links in the plane:

$$x = a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) \tag{1}$$

$$y = a_1 \sin\theta_1 + a_2 \sin(\theta_1 + \theta_2) \tag{2}$$

Interpretation:

- Given joint angles θ_1, θ_2, compute end-effector position (x, y)
- Direct and unambiguous
- Forms basis for forward kinematics matrix $T = T_1 T_2$

Inverse Kinematics (2-Link) — Step 1

Step 1: Solve for θ_2 (elbow angle)

Define: $r^2 = x^2 + y^2$

Apply law of cosines:

$$\cos\theta_2 = \frac{r^2 - a_1^2 - a_2^2}{2a_1 a_2} \tag{3}$$

Solve for angle:

$$\theta_2 = \operatorname{atan2}\left(\pm\sqrt{1 - \cos^2\theta_2}, \cos\theta_2\right) \tag{4}$$

Note:

- The $\pm$ gives two solutions: "elbow-down" and "elbow-up"
- This is the *multiple solutions problem*

Inverse Kinematics (2-Link) — Step 2

Step 2: Solve for θ_1 (shoulder angle)

Using the shoulder and elbow geometry:

$$\theta_1 = \operatorname{atan2}(y, x) - \operatorname{atan2}\left(a_2 \sin\theta_2, a_1 + a_2 \cos\theta_2\right) \tag{5}$$

Result:

- We now have the joint angles that place end-effector at (x, y)
- Two possible configurations exist (elbow-down, elbow-up)
- This is the classic geometric IK solution for planar 2-DOF robots

This approach doesn't generalize well to 3+ DOF or complex geometries

Section: Denavit-Hartenberg Parameters

Denavit-Hartenberg Parameters

Generalizing robot kinematics to arbitrary configurations

Denavit-Hartenberg (DH) Parameterization

DH Parameter Convention

A standard method to describe robot geometry using 4 parameters per joint:

- a_i: link length (distance along $\hat{x}$-axis)
- d_i: link offset (distance along $\hat{z}$-axis)
- α_i: link twist (rotation around $\hat{x}$-axis)
- θ_i: joint angle (rotation around $\hat{z}$-axis) — **variable**

The homogeneous transformation matrix:

$$T_i = \begin{pmatrix} \cos\theta_i & -\sin\theta_i\cos\alpha_i & \sin\theta_i\sin\alpha_i & a_i\cos\theta_i \\ \sin\theta_i & \cos\theta_i\cos\alpha_i & -\cos\theta_i\sin\alpha_i & a_i\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

RRR Robot Architecture (3-DOF)

RRR Configuration: Three revolute joints arranged vertically

DH Parameters:

Joint	a_i (mm)	d_i (mm)	α_i	θ_i (variable)
1	0	0	$-90°$	θ_1
2	0	0	$+90°$	θ_2
3	0	50	$0°$	θ_3

Workspace:

- Spherical region with radius 50 mm
- 3 DOF allows positioning in 3D space
- Multiple configurations (redundancy)

RRRRRR Robot Architecture (6-DOF)

RRRRRR Configuration: Six revolute joints for position and orientation

DH Parameters (simplified):

Joint	a_i	d_i	α_i
1	0	0	-90°
2	0	0	$+90^\circ$
3	0	50	0°
4	0	100	-90°
5	0	0	$+90^\circ$
6	0	50	0°

Characteristics:

- Significantly more complex workspace
- 6 degrees of freedom (full position and orientation control)
- Even higher dimensionality of solutions
- More challenging to learn

Classical Solutions

Traditional approaches to inverse kinematics

Classical Geometric Solution

Approach: Decompose problem into geometric subproblems

For a 3-DOF RRR robot:

1. Use joint 3 offset to simplify to 2D positioning problem
2. Solve position using inverse law of cosines
3. Solve orientation from desired end-effector orientation

Advantages:

- Algebraically exact solutions
- Computationally fast ($\sim 1\,\mu$s)
- Deterministic

Disadvantages:

- Problem-specific (doesn't generalize)
- Requires expert knowledge of robot geometry
- Difficult for 6+ DOF systems
- May miss some solutions

Damped Least Squares (DLS) Method

Approach: Iteratively refine joint angles to minimize pose error

DLS Update Rule:

$$\Delta\theta = (J^T J + \lambda^2 I)^{-1} J^T \mathbf{e}$$

Where:

- J: Jacobian matrix (derivatives of FK w.r.t. joint angles)
- $\mathbf{e}$: pose error vector
- λ: damping factor (prevents singular matrices)

Algorithm:

1. Start with initial guess $\theta_0 = [0, 0, 0]$
2. Compute forward kinematics and error
3. Update $\theta \leftarrow \theta + \Delta\theta$
4. Repeat until $|\mathbf{e}| < \epsilon$ (convergence)

Precision Criterion: $\epsilon = 10^{-6}$ radians ($\approx 0.0000573°$)

DLS vs Geometric Solutions

Geometric:

- Fast ($\sim 1\,\mu$s)
- Problem-specific
- Exact
- Doesn't scale to 6+ DOF

DLS:

- Slower ($\sim$ 1–10 ms)
- General method
- Approximate but converges
- Works for any DOF

Both are limited to:

- Computing one solution
- Slow real-time control
- Problem-specific tuning

The Problem: Multiple IK Solutions

Why classical approaches fail

The Challenge: Multiple Solutions

Problem: Many end-effector poses have multiple joint configurations

Example (RRR Robot):

- Position: $[1.8, 4.2, 49.8]$ mm
- Solution 1: $\theta = [-113^\circ, -5.3^\circ, 118.9^\circ]$
- Solution 2: $\theta = [67.1^\circ, 5.2^\circ, -63.4^\circ]$ (different angles, same position!)

Training Network on $\pm 180^\circ$ Range:

- Same input (end-effector pose) has contradictory targets
- Network receives conflicting training signals
- Cannot learn a well-defined function
- **Result: 0% Accuracy**

This is not a network failure—it's an ill-posed problem!

First Attempt: Random $\pm 180^\circ$ Dataset

Training Configuration:

- Joint range: $[-180^\circ, +180^\circ]$ (full range)
- Dataset: 50,000 random samples
- Network: Simple 4-layer fully connected
- Metric: MSE loss with 0.5 rad threshold

Accuracy: 0%

Why? Multiple IK solutions create contradictory training data

The Solution: Domain Knowledge

Applying constraints to make the problem well-posed

Solution: Constrain to Unique Solutions

Key Insight: Restrict joint range to ensure one-to-one mapping

Apply Domain Knowledge:

- Physically realistic robots operate in limited ranges
- Not all $\pm 180^\circ$ configurations are used in practice
- Constrain to $\pm 90^\circ$ per joint

Result of $\pm 90^\circ$ Constraint:

- Eliminates redundant solutions
- Creates well-defined inverse function
- Each pose has unique joint configuration
- Network can learn the mapping

This demonstrates the importance of:

- Problem understanding
- Domain knowledge application
- Proper problem formulation

After Solution: Consistent $\pm 90^\circ$ Dataset

Updated Configuration:

- Joint range: $[-90^\circ, +90^\circ]$ (constrained, realistic)
- Dataset: 50,000 samples from random angles
- Network: Simple 4-layer fully connected
- Metric: MSE loss with 0.5 rad threshold

Accuracy: 95.79%

Why? Well-defined mapping with unique solutions

Neural Network Approach

Learning inverse kinematics end-to-end

Simple 4-Layer Fully Connected Network (3-DOF)

Architecture:

- Input: 3 dimensions (end-effector position: x, y, z)
- Hidden Layer 1: 128 neurons, ReLU activation
- Hidden Layer 2: 64 neurons, ReLU activation
- Hidden Layer 3: 32 neurons, ReLU activation
- Output: 3 dimensions (joint angles: $\theta_1, \theta_2, \theta_3$)

Training Details:

- Loss: Mean Squared Error (MSE)
- Optimizer: Adam (learning rate $= 0.001$)
- Scheduler: ReduceLROnPlateau (patience $= 20$)
- Epochs: Up to 1000 with early stopping
- Batch size: 32

Results:

- Accuracy threshold: 0.5 radians
- Achieved: **95.79%**

SimpleCNN Network (3-DOF)

Convolutional Architecture:

- Conv Layer 1: 32 filters, kernel size 3, ReLU
- MaxPooling: kernel size 2
- Conv Layer 2: 64 filters, kernel size 3, ReLU
- MaxPooling: kernel size 2
- Flatten and Dense layers for output

Why CNN?

- Can capture spatial relationships in input
- Parameter sharing reduces overfitting
- Often achieves higher accuracy for similar architectures

Results at Higher Precision:

- Accuracy threshold: 0.01 radians (more strict)
- Achieved: **99.71%**
- Demonstrates superior performance on fine-grained accuracy

Extending to 6-DOF: RRRRRR Robot

Scaling to 6 Degrees of Freedom:

- Input: 6 dimensions (position x, y, z + orientation angles)
- Hidden layers: 256, 128, 64 neurons (scaled up for complexity)
- Output: 6 dimensions (all joint angles)

Challenges at 6-DOF:

- Higher dimensional input/output space
- More complex workspace geometry
- Longer training time required
- Potentially more multiple solutions

CNN Variant for 6-DOF:

- 4 convolutional layers (progressively deeper)
- More filters at each layer
- Better captures complex patterns

Section: Results and Comparison

Results and Comparison

Evaluating neural network performance

Accuracy Definition

For a prediction to be "correct":

- Network predicts joint angles $\hat{\theta}$
- Compute forward kinematics: pose $\hat{p} = FK(\hat{\theta})$
- Compare to target pose p
- Accept if $||\hat{p} - p|| <$ threshold

Different Thresholds:

Threshold	Degrees	Purpose
0.5 rad	28.6°	Training (loose)
0.01 rad	0.57°	Evaluation (tight)
10^{-6} rad	0.0000573°	DLS comparison (very precise)

Multiple thresholds allow evaluating network at different precision levels

Performance Comparison

Method	**Inference Time**	**Accuracy (10^{-6} rad)**
DLS Solver	$\sim$ 1–10 ms	$>$ 99%
Simple4Layer 3-DOF	$\sim$ 0.1 ms	95.79%
SimpleCNN 3-DOF	$\sim$ 0.2 ms	99.71%
Simple4Layer 6-DOF	TBD	TBD
SimpleCNN 6-DOF	TBD	TBD

Key Observations:

- Neural networks: **50–100$\times$ faster** than iterative solvers
- Accuracy: Comparable to classical methods
- Real-time: NNs enable fast robot control
- Scalability: Same approach works for higher DOF

Section: Conclusion

Conclusion

Bringing it all together

Key Takeaways

Universal Approximation in Practice

1. Neural networks successfully learned inverse kinematics
2. Achieved 95.79% accuracy on 3-DOF, 99.71% on CNN variant
3. Demonstrated 50–100× speedup vs classical iterative methods

The Challenge of Multiple Solutions

1. Without domain knowledge: 0% accuracy
2. With proper constraints: 95.79%+ accuracy
3. Problem formulation matters as much as the algorithm

Practical Implications

1. Pre-trained networks enable real-time robot control
2. Domain knowledge + ML combines best of both worlds
3. Scalable to higher DOF robots (6-DOF demonstrated)

Future Work

- Measure and optimize 6-DOF inference times
- Integrate solution selection (multiple IK solutions)
- Handle singularities and unreachable poses
- Train on task-specific subspaces
- Real robot deployment and validation
- Compare with other architectures (RNNs, transformers)

Questions?