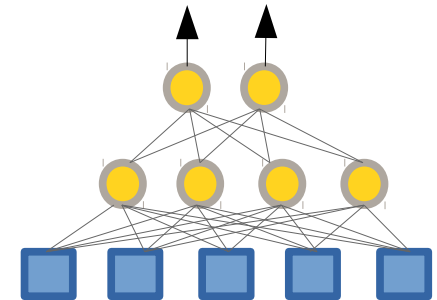


Dropout and Bayesian Neural Networks

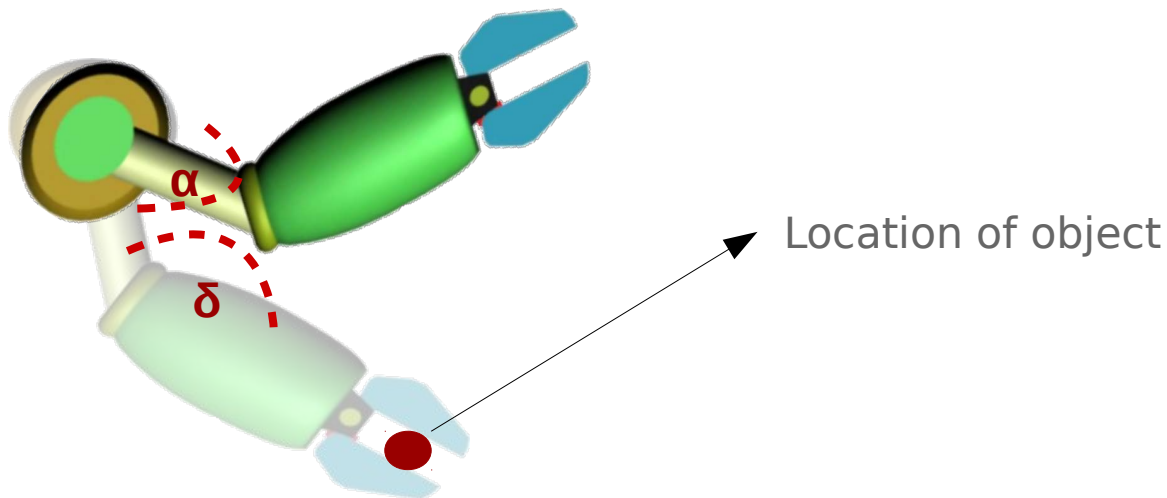
From Prediction to Control

- Neural network for predicting robot state
- Can we also **learn to control robot?**
- Example: reaching and grasping a cup
- What are the inputs and outputs?
- How to generalize to new situations, e.g., a mug?



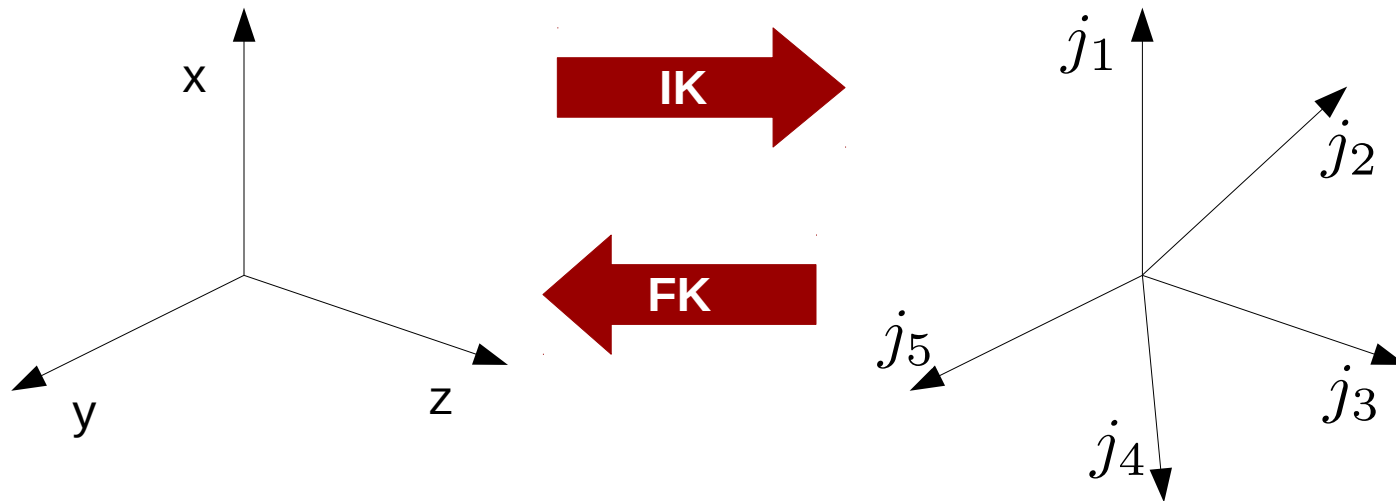
Example 2: Learning Inverse Kinematics

- Robots need to reach for objects
- Object position is in cartesian space $[x, y, z]$
- We need the corresponding robot joint angles
- Inverse kinematics calculates angles from position

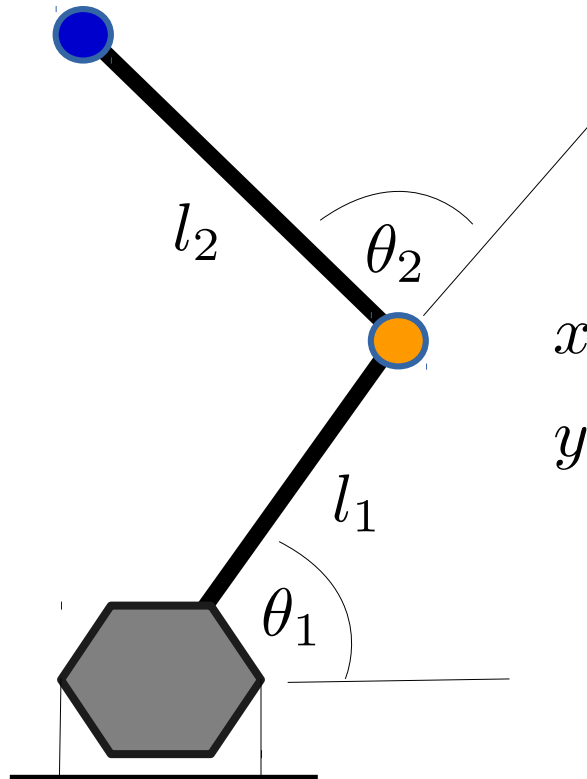


Task space vs Configuration Space

- Spaces for controlling robot
- Task space: Euclidean space of end-effector
- Configuration space: space of all joint angles
- Kinematics transforms between the two



Forward Kinematics in 2D

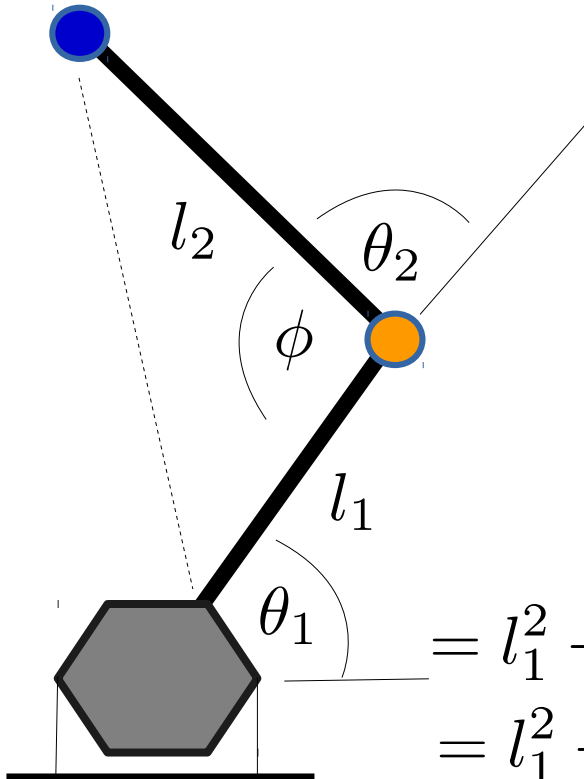


$$x = l_1 \cos(\theta_1) + l_2 \cos(\theta_1 + \theta_2)$$

$$y = l_1 \sin(\theta_1) + l_2 \sin(\theta_1 + \theta_2)$$

$$T = \begin{bmatrix} \cos_{12} & -\sin_{12} & l_1 \cos_1 + l_2 \cos_{12} \\ \sin_{12} & \cos_{12} & l_1 \sin_1 + l_2 \sin_{12} \\ 0 & 0 & 1 \end{bmatrix}$$

Inverse Kinematics in 2D



Use equality:

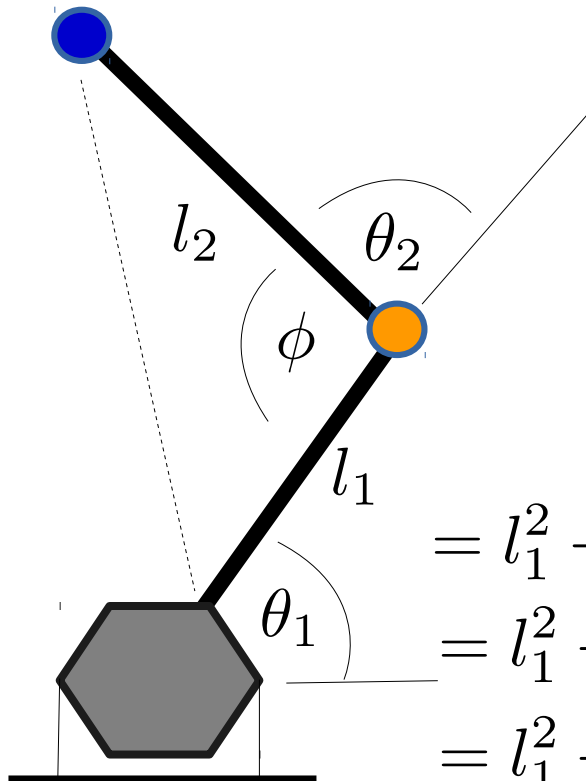
$$x^2 + y^2 = l_1^2 + l_2^2 - 2l_1l_2 \cos(\phi)$$

And: $\phi = \pi - \theta_2$

Algebraic rewriting:

$$\begin{aligned} x^2 + y^2 &= l_1^2 + l_2^2 - 2l_1l_2 \cos(\pi - \theta_2) \\ &= l_1^2 + l_2^2 - 2l_1l_2(\cos \pi \cos(\theta_2) - \sin(\pi)\sin(\theta_2)) \\ &= l_1^2 + l_2^2 - 2l_1l_2(-\cos(\theta_2)) \\ &= l_1^2 + l_2^2 + 2l_1l_2 \cos(\theta_2) \end{aligned}$$

Inverse Kinematics in 2D



Use equality:

$$x^2 + y^2 = l_1^2 + l_2^2 - 2l_1l_2 \cos(\phi)$$

And: $\phi = \pi - \theta_2$

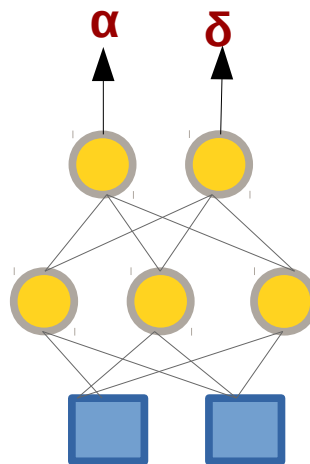
Algebraic rewriting:

$$\begin{aligned} x^2 + y^2 &= l_1^2 + l_2^2 - 2l_1l_2 \cos(\pi - \theta_2) \\ &= l_1^2 + l_2^2 - 2l_1l_2(\cos \pi \cos(\theta_2) - \sin(\pi)\sin(\theta_2)) \\ &= l_1^2 + l_2^2 - 2l_1l_2(-\cos(\theta_2)) \\ &= l_1^2 + l_2^2 + 2l_1l_2 \cos(\theta_2) \end{aligned}$$

Solve for theta: $\theta = \pm \cos^{-1} \left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2l_1l_2} \right)$

Inverse Kinematics via Neural Network

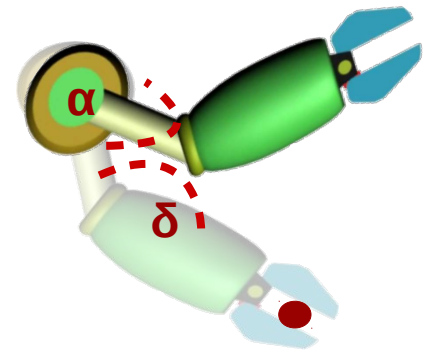
- Now let's learn inverse kinematics via ANN
- Randomly set the arm to joint angle configuration
- Measure the position of gripper
- **Input:** position → **Output:** joint angle



Output units
(joint angles)

Hidden units

Input units
(position)



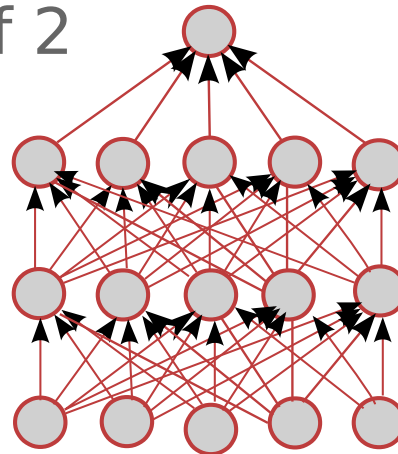
Regularization

- Regularization:
methods that attempt to minimize the error of our loss function with respect to a set of inputs that are not used for training
- Dropout, simple but powerful regularization
- **During training**, individual neurons are either kept with probability p or dropped with $1 - p$
- The approach emulates learning an ensemble of different variants (network structures) of an ANN

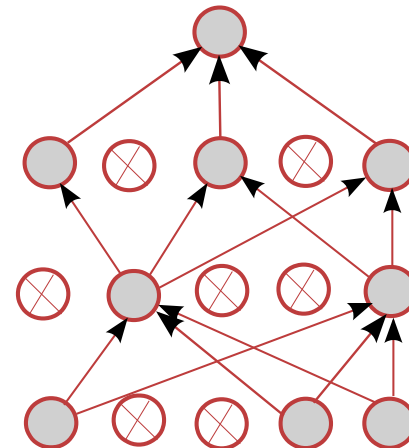
Dropout

- Prevent overfitting
- Disactivate neurons throughout learning
- Drop with probability $1 - p$
- Weight Scaling Inference Rule (Hinton et al.)

If we set $p = 0.5$ then we need to scale outputs by a factor of 2

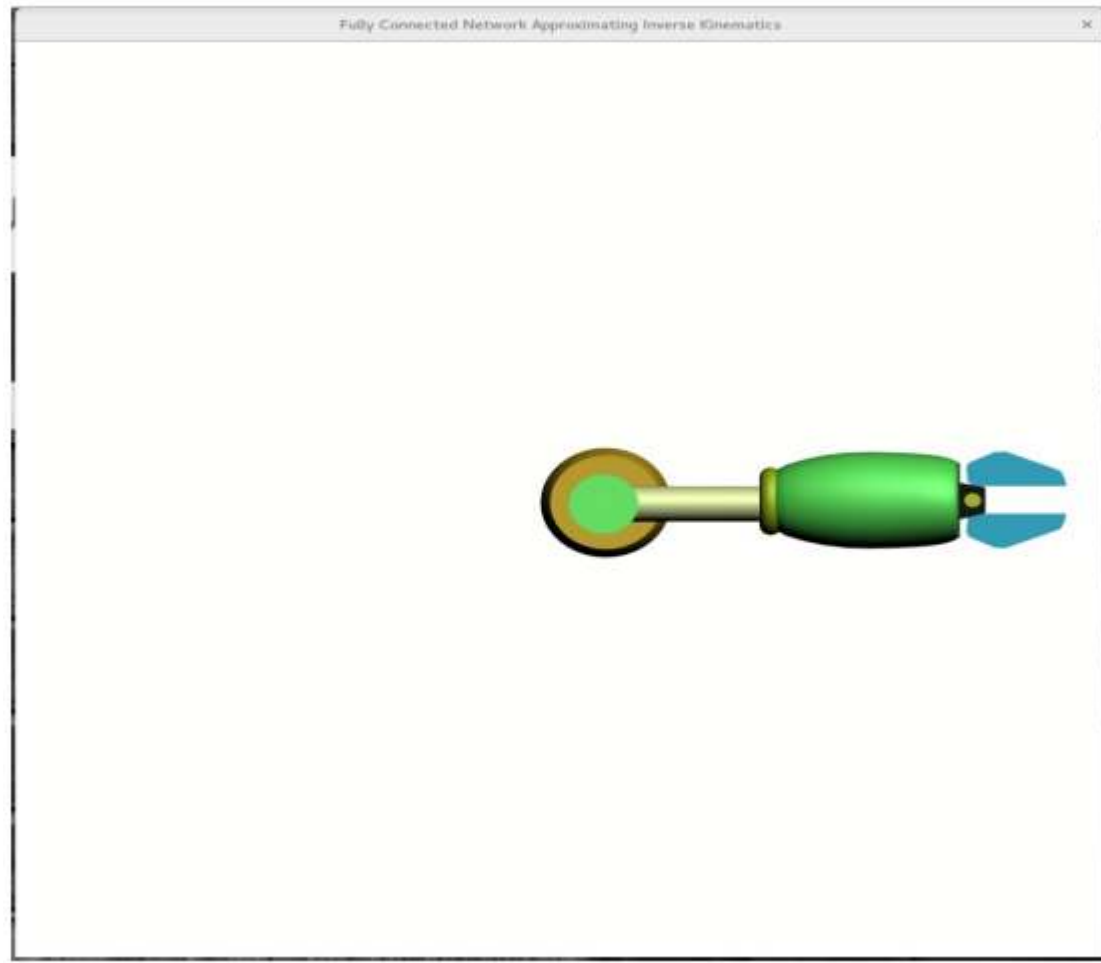


Standard Neural Net



After Applying Dropout

Inverse Kinematics via Neural Network



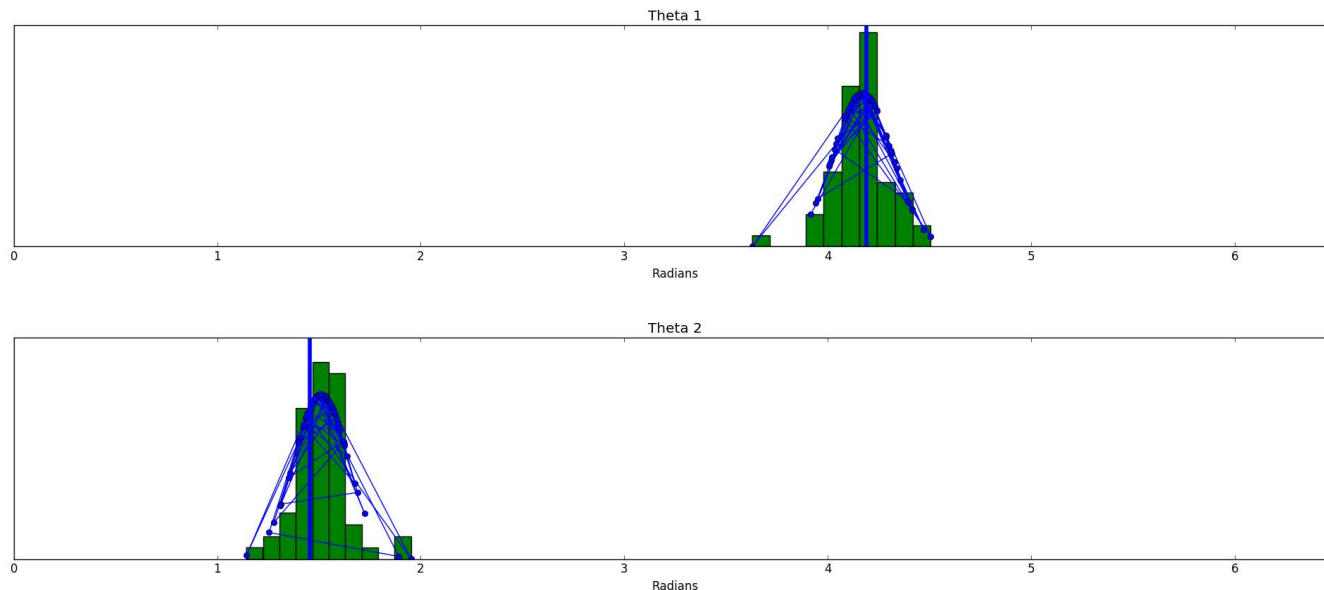
[Play Video](#)

Uncertainty in Neural Networks

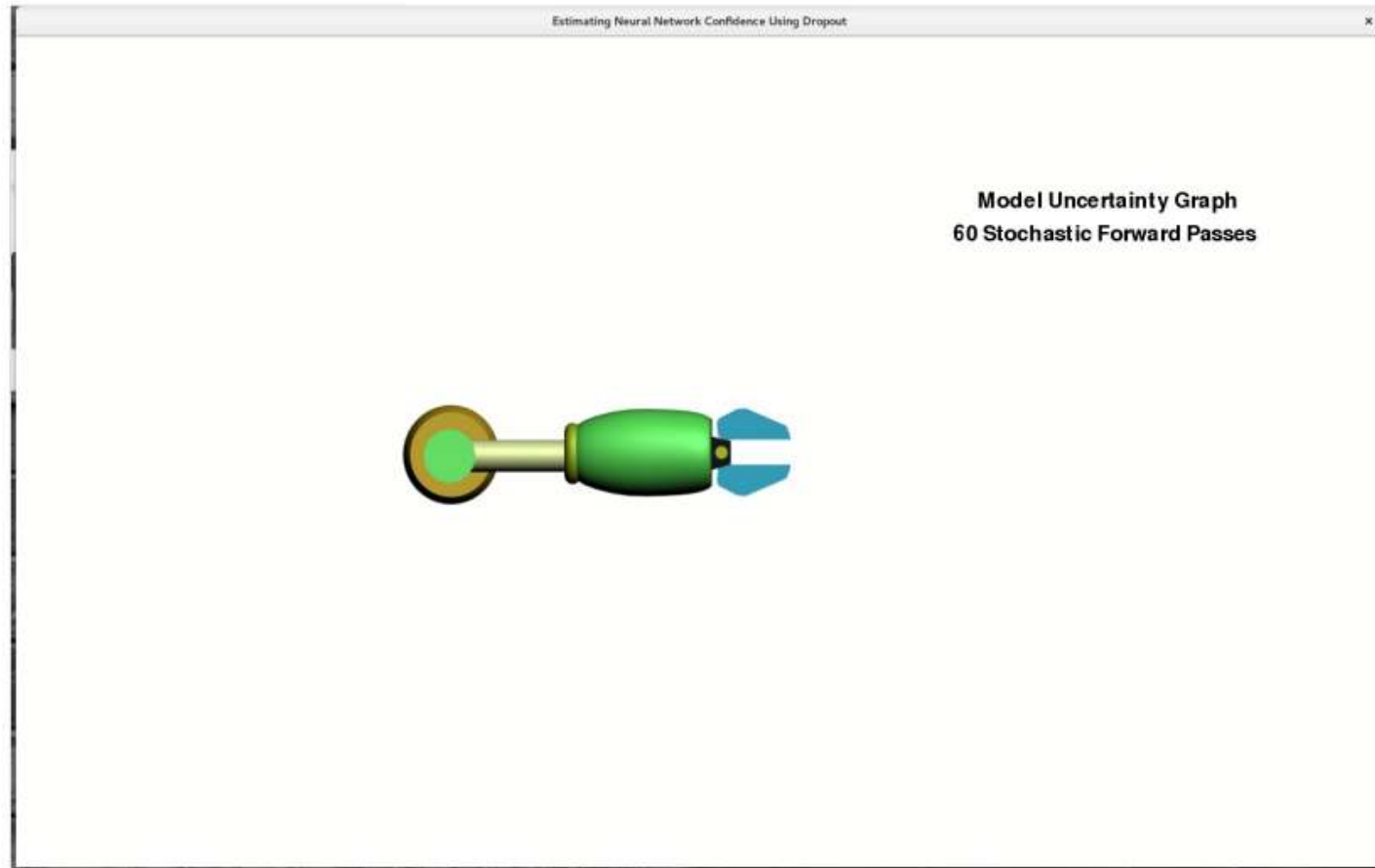
- The output of an ANN is not guaranteed to be accurate
- Depends on task complexity, non-deterministic environment, noise, training set, etc.
- We need a way to assess the confidence / uncertainty of an ANN in its outputs
- Ideally, **probabilistic** or **Bayesian** outputs
- Efficient approximation via Dropout

Stochastic Forward Passes

- Approximate the confidence of our model
- Sampling multiple predictions at inference time
- Each sample may result in different net output
- The samples form a **probability distribution**



Inverse Kinematics with Uncertainty



[Play Video](#)

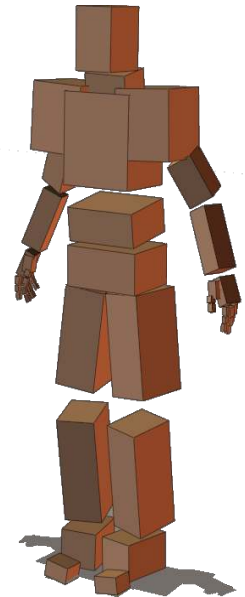
Dimensionality Reduction

The Curse of Dimensionality

- First mentioned by Bellman [Bellman, 1957]
- Exponential growth in data and computation
- High-dimensional spaces challenging for machine learning

Human movement inherently low dimensional

Muscle Synergies [Bernstein 1967, Santello et al. 1998]



Muscle Synergies in Grasping

- Finger muscles are co-activated
- First 2 principal components account for $> 80\%$
- 2 DOF are sufficient to achieve 80% of all grasps

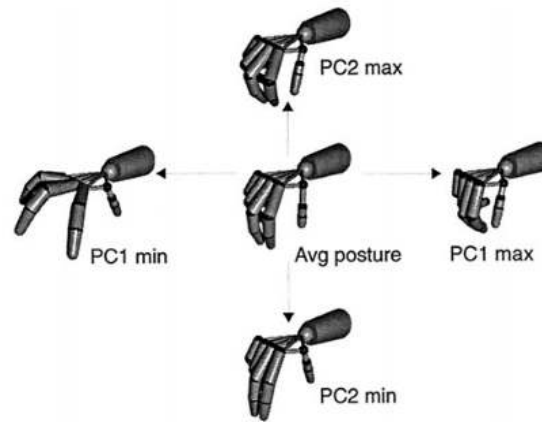


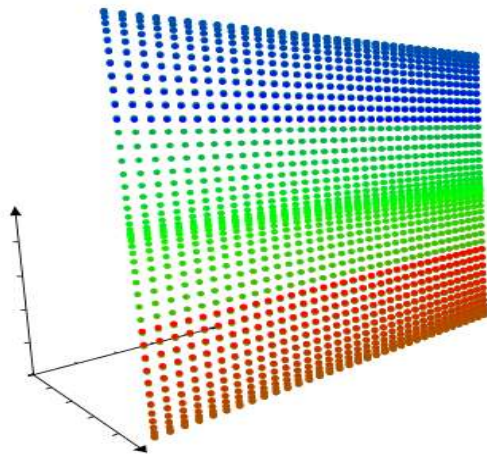
Figure 6. Postural synergies defined by the first two principal components. The hand posture at the center of the PC axes is the average of 57 hand postures for one subject (U.H.). The postures to the *right* and *left* are for the postures for the maximum (*max*) and minimum (*min*) values of the first principal component (*PC1*), coefficients for the other principal components having been set to zero. The postures at the *top* and *bottom* are for the maximum and minimum values of the second principal component (*PC2*).

Data lies on a
low-dimensional manifold!
[Santello et al., 1998]

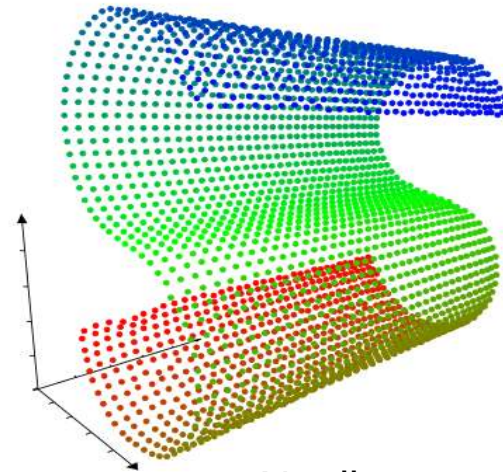
**We can extract the manifold
via dimensionality
reduction!**

Dimensionality Reduction

- Assume data lies on a **manifold**
- Manifold has lower dimensionality than space
- Goal: identify Manifold



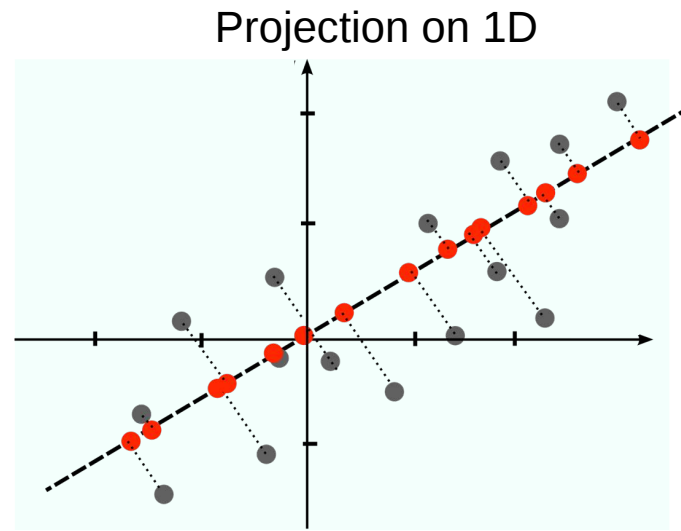
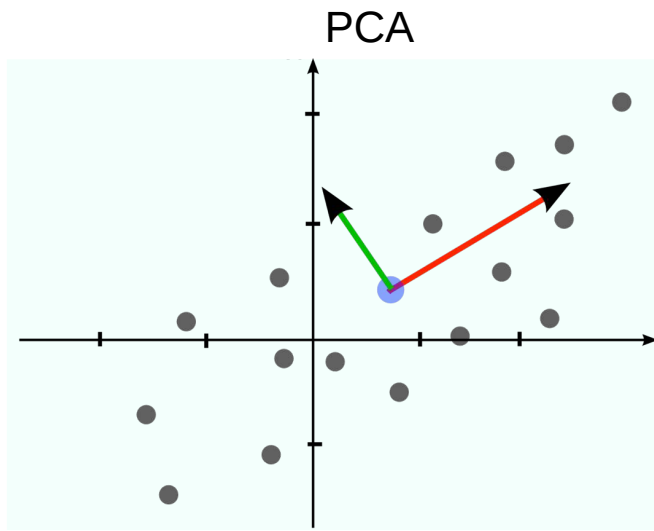
Linear



Nonlinear

Principal Component Analysis

- Extracts a set of principal components
- Eigenvectors: principal components
- Eigenvalues: the variance of data along direction



Computing PCA

Approach

- Remove mean
- Calculate covariance
- Eigen-Decomposition
- Fast computation of PCA via Intel MKL

$$\mathbf{X}_c = (\mathbf{I} - \mathbf{1}\mathbf{1}^T)\mathbf{X}$$

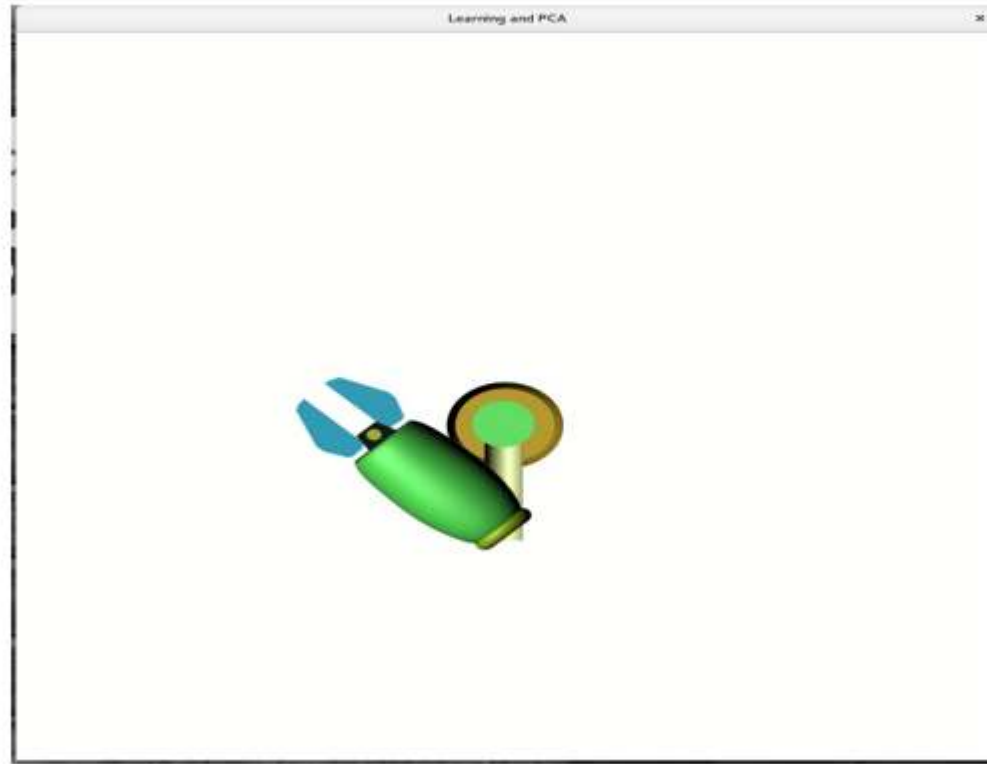
$$\mathbf{\Sigma} = \mathbf{X}_c\mathbf{X}_c^T$$

$$\mathbf{\Sigma} = \mathbf{U}\mathbf{W}\mathbf{V}^T$$

Application to Robot Learning

- We can use PCA to reduce number of control parameters needed to control robot
- In grasping we can control robot hand using only 2 degrees-of-freedom instead of 15+
- Approach: collect training data and perform PCA
- Then control robot using only first 2-3 principal components

Example 3: PCA for Arm Kinematics



[Play Video](#)

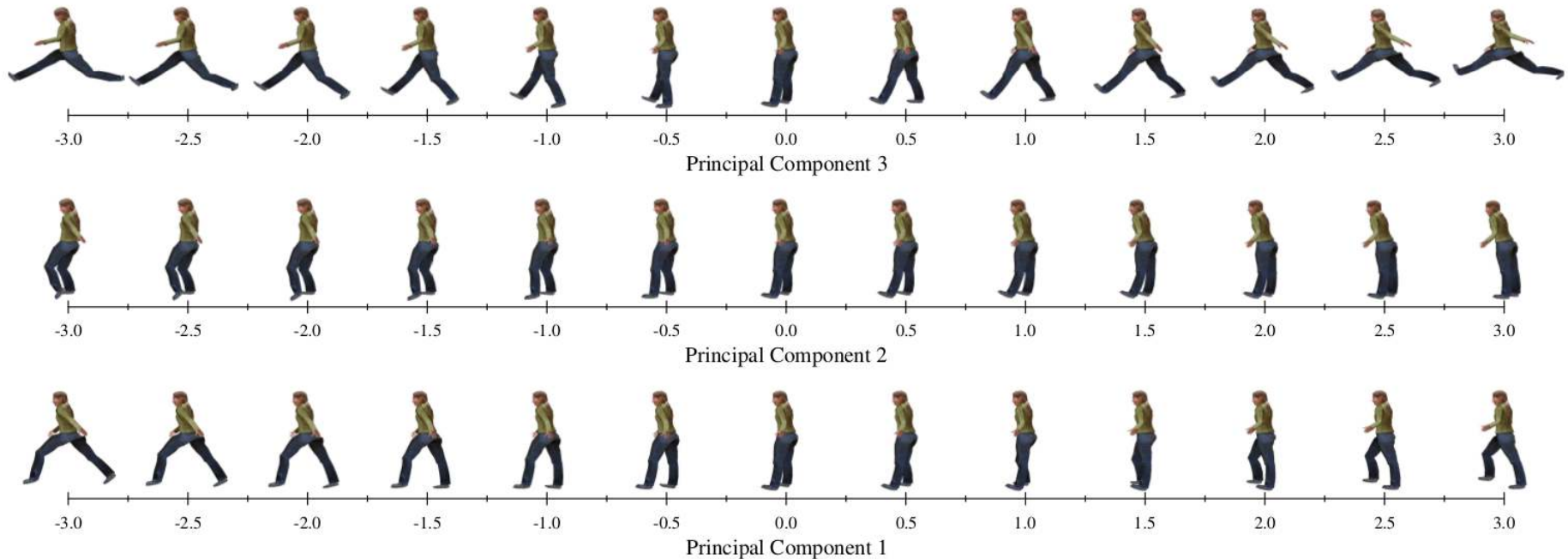
Example Application

- Python code implementing the above examples can be found in folder “ProbabilitiesDropout”
- The README includes instructions on learning and testing a model



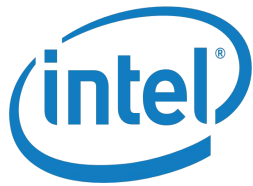
Example 4: Projecting a Walking Gait

- We can use PCA on demonstrations from a human expert and analyze the individual components



Summary

- Neural network for robot control
- Output of the network are controls
- Training with Dropout to achieve better performance, i.e., regularization
- Tasks may involve high-dimensional variables
- We can reduce dimensions using PCA
- However, so far all training is supervised



The development of this course was supported by an Intel AI Academy grant. We thank the sponsor for the continuing support of open-source efforts in research and education.