Reinforcement Learning for Human Action Generation

1. Action as a game

Select videos of people (a) clapping (b) running (c) rub two hands (d) throwing etc. Each type of action can be thought of as a type of game, analogous to "chess", "pong", "go". In a game we can win or lose, and here we can perform such a type of action or not. If we are learning a policy for the "game" of clapping hands, then a video of a human "clapping" is considered a win. While a video of a human "rub two hands" represents a loss.

2. Skeleton 3D Coordinate System

A skeleton is defined by a set of N vertices $V = \{v_1,...,v_N\}$ and the edges $e_{u,v}$ between vertices. The vertices and edges form a tree-graph-skeleton $G_S(v,e)$. The 3D coordinates of each vertices are given by $coordinate(v) = (x,y,z)$.

The dataset used has 3D data for 25 joints - 1-base of the spine 2-middle of the spine 3-neck 4-head 5-left shoulder 6-left elbow 7-left wrist 8- left hand 9-right shoulder 10-right elbow 11-right wrist 12- right hand 13-left hip 14-left knee 15-left ankle 16-left foot 17- right hip 18-right knee 19-right ankle 20-right foot 21-spine 22- tip of the left hand 23-left thumb 24-tip of the right hand 25- right thumb

Given a video sequence the same physical skeletons will appear in different places, at different poses, and at different scales. Choose a coordinate system so that the N vertices coordinates can have a more intrinsic representation for action recognition. In order to eliminate global translation, global rotation and scaling create a new coordinate system. The following vertices define a 3D rigid body frame "left hip", "left shoulder", "right hip", "right shoulder", and "neck"

3. Data Preprocessing

Choose the "neck" vertex as the origin of each skeleton representation. So, every vertex coordinate is subtracted from $coordinate(v_{neck})$, or $coordinate(v) = coordinate(v) - coordinate(v_{neck})$. Such representation becomes translation invariant. In order to account for global rotations, we choose a pair of them to constitute together with the "neck" a coordinate system that also accounts for rotations and scale. Choose vertices "right hip", "left hip", "neck", since the three coordinates are far from each other to yield stable numerical coordinates.

Create the orthogonal coordinate system

$$u_1 = coordinate(v_{righthip}) - coordinate(v_{neck})$$

$$s = |\vec{u}_1|$$

$$\hat{u}_1 = \frac{\vec{u}_1}{s}$$

$$\vec{u}_2^{temp} = coordinate(v_{lefthip}) - coordinate(v_{neck})$$

$$\hat{u}_3 = \frac{\hat{u}_1 \times \vec{u}_2^{temp}}{|\hat{u}_1 \times \vec{u}_2^{temp}|}$$

$$\hat{u}_2 = \hat{u}_1 \times \hat{u}_3$$

where $\hat{u}_1$, $\hat{u}_2$ define the orthogonal body frame coordinate system while $\hat{u}_3$ is orthogonal to the body frame. We work with this orthonormal coordinate system, $\{\hat{u}_1, \hat{u}_2, \hat{u}_3\}$. Also, use a scale parameter *s*, computed above, to produce scale invariant skeleton coordinates.

$$coordinate(v_i) = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix}; i = 1, \ldots, N$$

More precisely, every coordinate of a skeleton is mapped to a coordinate where

$$XYZ(v_i) = \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix}; i = 1,...,N$$

$$XYZ(v_i) = \begin{pmatrix} X_i \\ Y_i \\ Z_i \end{pmatrix} = \frac{1}{s} \begin{pmatrix} (coordinate(v_i) - coordinate(v_{\text{neck}})) \cdot \hat{u}_1 \\ (coordinate(v_i) - coordinate(v_{\text{neck}})) \cdot \hat{u}_2 \\ (coordinate(v_i) - coordinate(v_{\text{neck}})) \cdot \hat{u}_3 \end{pmatrix}$$

where $N = 25$.

This representation is translation invariant, it is scale invariant, and the third coordinate represents how much a vertex is away from the body frame plane. Thus, the representation is also invariant to rotations of the body frame.

If two consecutive frames are extracted, tracking a graph overtime is a relatively easy problem. The tracking between $G^t_S(v,e) \equiv G_S(v^t,e^t)$ and $G^{t-1}_S(v,e) = G_S(v^{t-1},e^{t-1})$, is described by

$$v^t = v^{t-1} + d(v^t) \text{ where } \quad d(v^t) = (dx,dy,dz)$$

for all vertices v^t. We then describe the state s^t of a skeleton at time *t* by the set

$$s^t = \{(v^t_1, d_1(v^t_1)); (v^t_2, d_1(v^t_2)); \ldots; (v^t_N, d_N(v^t_N))\} \equiv \{(v^t_i, d^t_i); i = 1, \ldots, N = 13\}$$

4. Actions and Policies

In the context of humans, actions are the result of applying internal force to our skeletons. In the context of reinforcement learning, each type of action follows a policy $\pi(a|s)$. For the type of action "clapping hands" or for the type of action "pointing two hands" our arms and hands may follow different policies. Also, we clap hands in different ways and we grab objects in different ways, and we can visually distinguish the two actions. Each type of action has a probabilistic distribution over possible actions, given a state. We will use reinforcement learning to learn such policies. We can choose a few types of actions to recognize:

1. Clapping Hands (ClH)
2. Cross Hands (CrH)
3. Rub Two Hands (RTH)
4. Pointing to Something (PS)
5. Throw (T)

Collect a sequence of states and data from examples and samples of actions.The goal is to learn policies from all these examples.

Having all the data, states and actions, and model to learn the policies for each type of action. We learn the policies by considering positive and negative sequences of them. Thus, "ClH" and "CrH" are presented together as both require moving hands and so one can be the negative reward for the other.
For all training videos the states and actions are pre-computed. A method is needed to learn the policies.

5. Method

Simply train a neural network, say of few fully connected layers to learn the function $\pi_\theta(a|s)$, where θ describes all weights (parameters) of the network, s is represented by $150 = 50 \times 3$ input nodes and a represented by $75 = 25 \times 3$ output nodes.
We have positive examples and negative examples. Assign a weight to each example based on the return. The larger is the magnitude of the return (the closer to the last frames of a sequence) the more important is such an example. So, we use $\gamma = 0.95$ so that the first frame weight magnitude is $0.95^{60} \approx 0.045$ and the middle frame weight magnitude is $0.95^{30} = 0.214$. Negative examples will have negative weights. The final loss function for training is the mean square error.

$$E(\theta|s_{data}) = \text{weight}_{data}(a(\theta) - a_{data})^2$$

where $\text{weight}_{data} = \pm\gamma^{f-k}$, where the sign ± depends if the video used is positive example or negative example and f is total number of frames. The index k is the frame index. If the frame is the last one, $k = 60$, and the weight is simply ±1 with f=60, while if $k = 30$ the weight is ±0.214 with f=60. Because we have four times more negative videos than positive, we may apply an extra factor 0.25 to multiply the 163 negative weights. The negative weight push $a(\theta)$ away from a_{data} while the positive examples push $a(\theta)$ to be closer to a_{data}.
This method may be interpreted as supervised learning method, except we are using the return to weight the frames data and assigning negative examples.
Once we find a solution θ for the neural network, then we can send any state s and we can derive the action from it. Such an action will lead to a new state s that can be fed again to the neural network and so again, a new action is produced. Thus, someone clapping hands can be simulated from the initial state.