1. Data

1.1 Augmentations

1.1.1 `aug2`

Copied and changed from Frog's public discussion. It includes `random_affine` and `random_interpolate`. `random_affine` will to scale, shift and global rotate on spatial coordinates. `random_interpolate` will scale and shift (only a little) the time dimenstion.

1.1.2 `random_hand_op_h4`

Rotate hand fingers joint by joint. First, I defined five joint routes as five fingers (`HAND_ROUTES`). E.g. `[0, *range(1, 5)]` is the thumb. And for each route, I created trees from one finger (`HAND_TREES`), in which the root is one joint and the children are joints following this joint.

For example, the thumb finger will generate trees like:

```
root, children
[0], [1 2 3 4]
[1], [2 3 4]
[2], [3 4]
[3], [4]
```

In `random_hand_op_h4`, I random selected trees by `joint_prob = 0.15`, and rotated all children around root with random degree in (-4, 4).

For example, rotate `[1 2 3 4]` around `[0]` and rotate `[3 4]` around `[2]`.

This augmentation gives small disturb to hand shapes, but it is more natural than noise, because it will not affect the bone length.

PS: `random_hand_op_h4` accept `Lx21x3` input. In my code, it is `random_hand_op_h4(pos[:,-21:])`, and in yours, it should be `random_hand_op_h4(pos[:,20:41])`.

1.1.3 mirror flip

Flip lip.

1.2 Hand craft feature

Hand is selected by `hand.isnan().mean()`, less nan hand will be kept. Flip R-hand to L-hand if it is selected.

1.2.1 Position and Motion

> NOTE: Normalization is applied in these features.

1. xyz of lip and hand (`pos`).
2. history motion of `pos`
3. future motion of `pos`

$$dpos = pos_t - pos_{t-1}$$
$$rdpos = pos_{t+1} - pos_t$$

1.2.1 Distance

> NOTE: xy are used in distance, Normalization is **not** applied in these features.

Full pairwise distance of hand. For hand, `self.dis_idx0` and `self.dis_idx1` is the indices of upper triangle of `torch.ones((21, 21))`. N = (21 * 20) / 2 = 210 non-repeated pairs in total. `torch.linalg.vector_norm` is applied to get Euclidean-distance between joints.

```
self.dis_idx0, self.dis_idx1 = torch.where(torch.triu(torch.ones((21, 21)), 1)
== 1)
# self.dis_idx0 <-- tensor([ 0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0, ...
# self.dis_idx1 <-- tensor([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, ...

ld = torch.linalg.vector_norm(
    lhand[:,self.dis_idx0,:2] - lhand[:,self.dis_idx1,:2],
    dim = -1) # <--- distance between (0, 1), (0, 2), (0, 3), ...
```

For lip, only a patial pairwise distance is generated. It should have N = (40 * 39) / 2 = 780 pairs in total. But I made a mistake in lip's `dis_idx`, which gives less pairs. `self.dis_idx2, self.dis_idx3` only used pairs between first 20 points of lip. However, in my experiments, this partial pairs could give a good result.

1.2.2 Angle

> NOTE: xyz are used in angle, Normalization is **not** applied in these features.

As mentioned before, `HAND_ROUTES` are routes of five fingers. For each finger, we can get 3 angles:

```
thumb = 0-1-2-3-4

angle_1 = cos_sim(0-1, 1-2)
angle_2 = cos_sim(1-2, 2-3)
angle_3 = cos_sim(2-3, 3-4)
```

1.3 Postprocessing

Use `linspace` to subsample seq longer than 96.

2. Model

Huggingface BERT and DeBerta-v2, the only difference between them is disentangled attention. Conditional Position Embedding (CPE) from Twins is added before all encoder layer except the first one.

3. Training

3.1 Hyperparams

Ranger optimizer, 45-epochs flat learning rate + 25-epochs cos-anneal learning rate. Label-smooth is 0.5.

3.2 Distilation

1. Stage 1: Train a 4-layer model from scratch (model#1)
2. Stage 2: Train a 4-layer model from scratch, with additional knowledge distillation loss from model#1 (model#2)
3. Stage 3: Train a 3-layer model, load weights from first 3 layers of model#2, with additional knowledge distillation loss from model#2 (model#3)