Data Driven Computer Animation

HKU COMP 3360

Tutorial 4 - Data-Driven Character Animation

Prof. Taku Komura
TA: Shi Mingyi (myshi@cs.hku.hk)

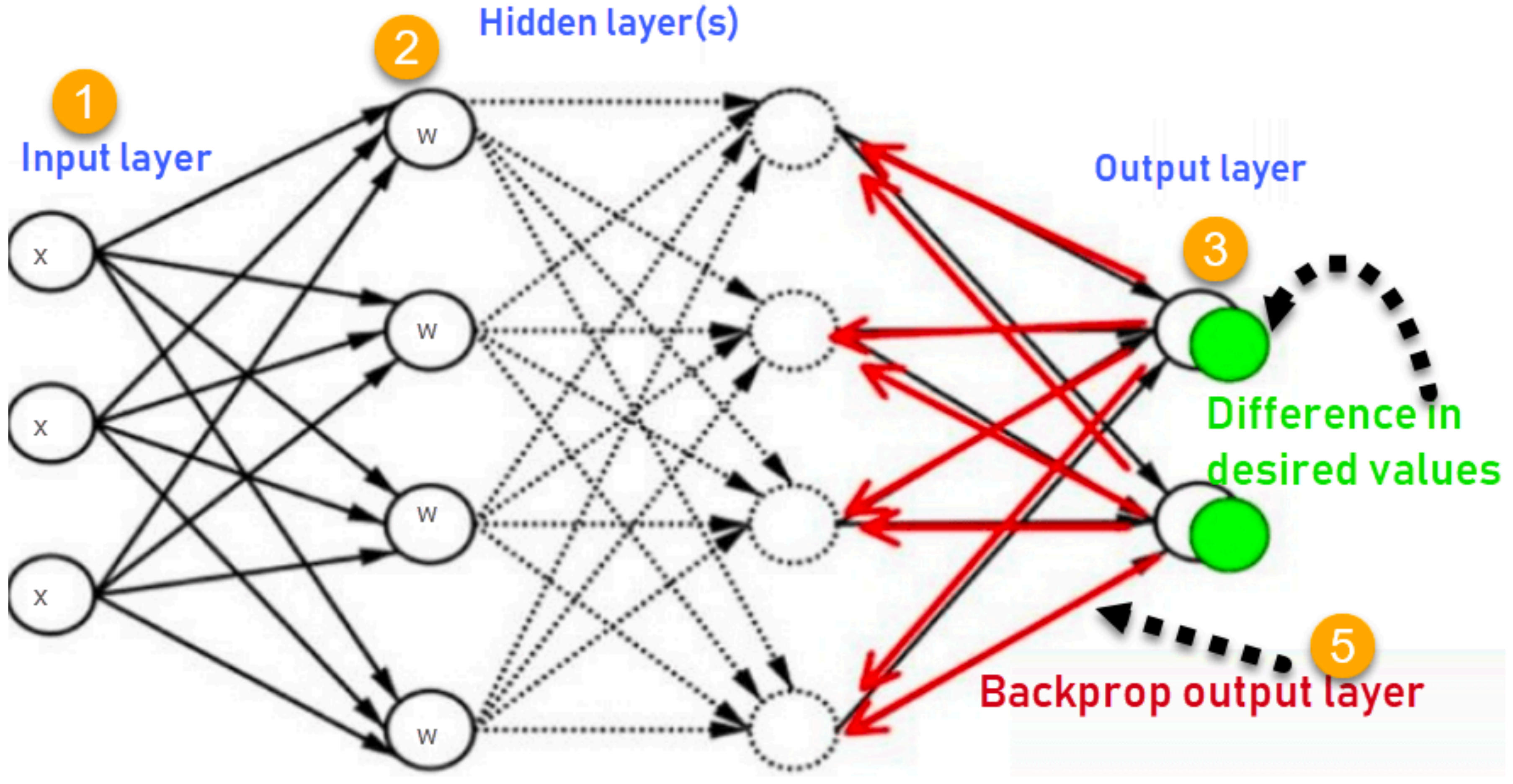
SECTION 2A, 2021

Tutorial 4 - Agenda

- Code Blocks in Deep Learning Project
- Deep Motion Structure and Tensor Operations
- Motion AutoEncoder with FC and Conv1D

(Under visual studio code programming environment)

Structure for Deep Learning Project



- Input, target_output, we have forwarding network
- The initial forwarding network cannot reach the target_output
- Calculate the difference in desired values, and backprop it and optimize our forwarding network, to make its output closer to the target

Structure for Deep Learning Project

- DataLoader
  - Implement __getitem(index)__ function
- Model Definition
  - Implement the model structure
  - Implement the forward function
- Trainer and Evaluator
  1. Take batches from dataloader
  2. Feed batches to network
  3. Calculate the losses
  4. Backward the losses and update network

Check ml_motion_autoencoder_fc.py

Check pytorch examples: https://github.com/pytorch/examples

Load Motion Data

return parameters:

- ***rotation*** (quaternion or euler): frame_number * joint_number * rotation_number
- ***position***: frame_number * joint_number * 3 (only the root position will use used)
- for build reset skeleton:
  - ***offset***: joint_number * rotation_number
  - ***parent***: joint_number
  - ***names***: joint_number

All the parameters can be observed by debugging mode

Tensor Operations - nn.Linear

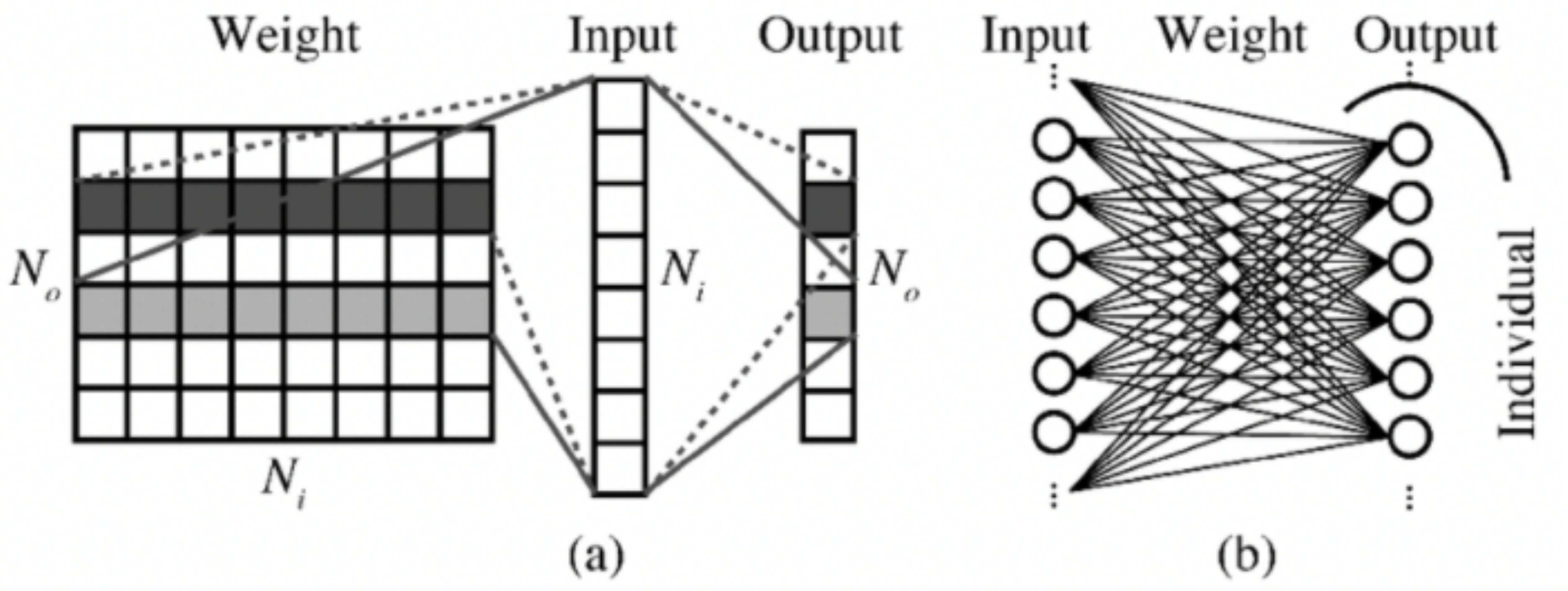
Fully connected layer

- can operate a tensor with shape (batch_size, channels, feature_number)

- fc_layer = nn.Linear(in_features=100, out_features=10)

- data = torch.zeros((128, 3, 100))

- output = fc_layer(data)   # (128, 3, 10)



Document: nn.Linear

Tensor Operations - nn.Linear

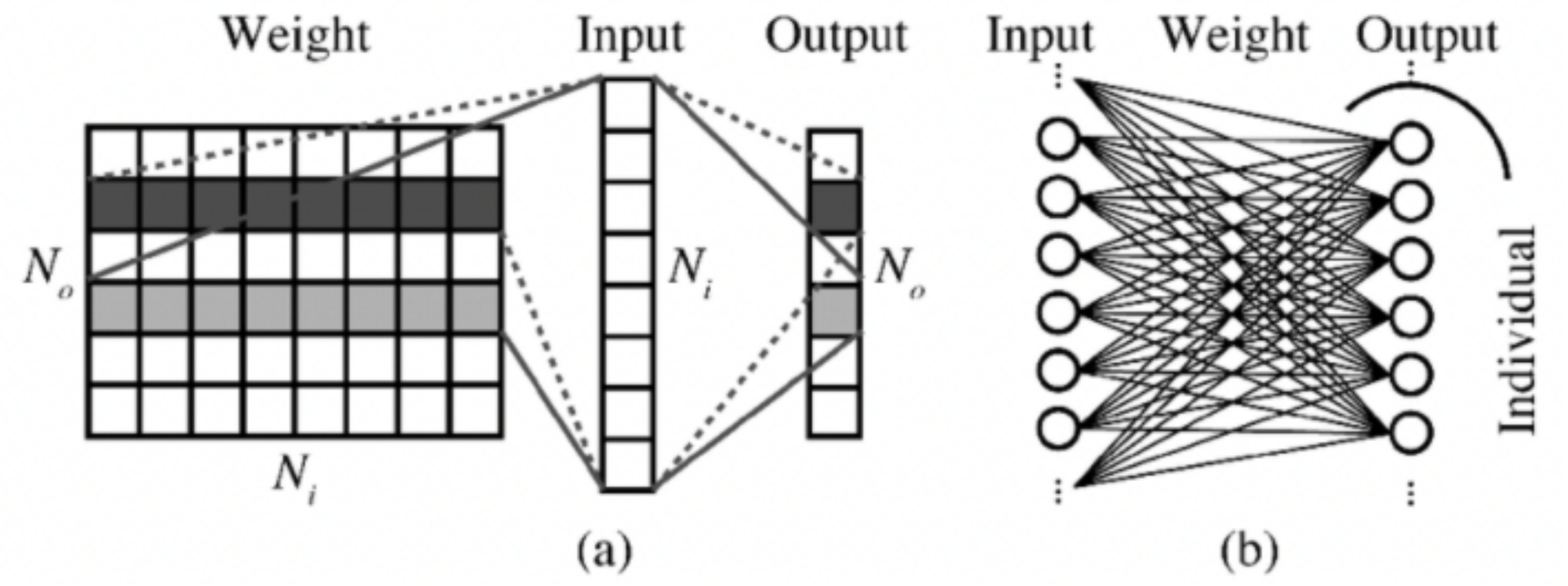
Fully connected layer

given a rotation tensor which is shaped by (2000, 31, 4)

- fc_layer = nn.Linear(in_features=4, out_features=10)

Work? output = fc_layer(rotation)

No



Document: nn.Linear

Tensor Operations for Motion Data

Fully connected layer on motion data

Rotation: (2000, 31, 4)

fc_layer: in_features=100, out_features=10

new_fc_layer: in_features=4, out_features=10

output = new_fc_layer(rotations)   #   (2000, 31, 10)

Document: nn.Linear

Tensor Operations - nn.Conv1D

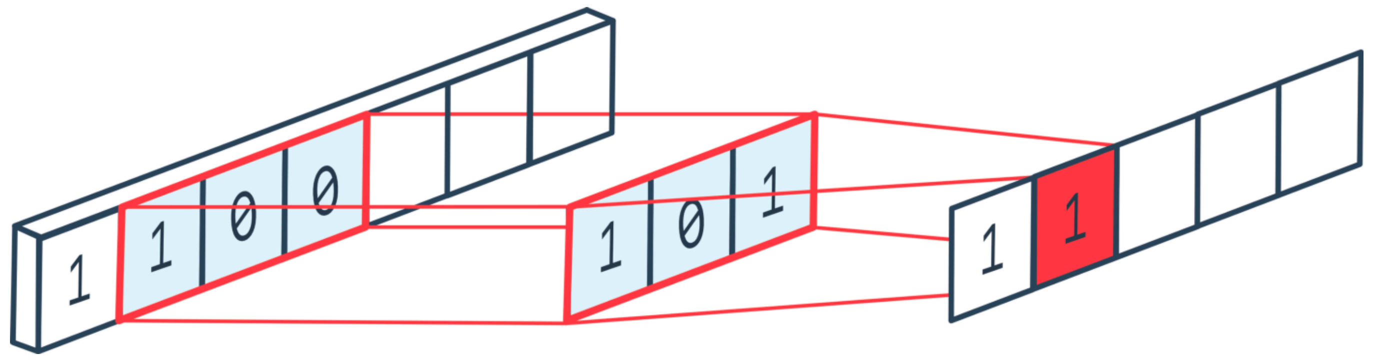
1D Convolutional layer

- can operate a tensor with shape (batch_size, channels, width)

- conv1d_layer = nn.Conv1d(in_channels=3, out_channels=16, kernel_size=2, stride=2)

- data = torch.zeros((128, 3, 100))

- output = conv1d_layer(data)   # (128, 16, 50)



Document: nn.Conv1d

Tensor Operations - nn.Conv1D

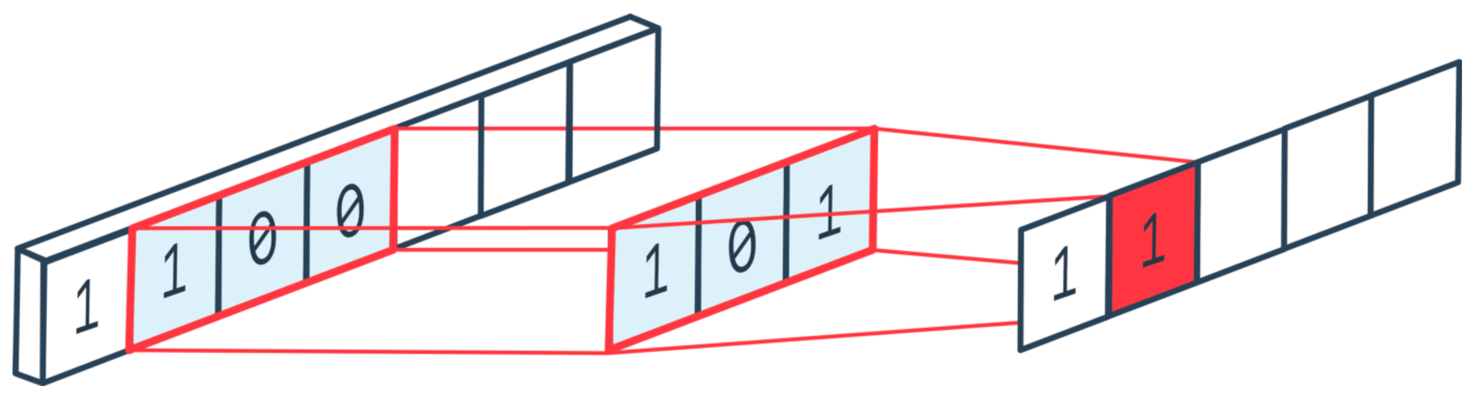
1D Convolutional layer - what does it do?

1. Channel level -> information exchange

- data = torch.zeros((128, $\boxed{3,}$ 100))
- output = conv1d_layer(data)   # (128, $\boxed{16,}$ 50)

2. Width level -> compress information

- data = torch.zeros((128, 3, $\boxed{100)}$)
- output = conv1d_layer(data)   # (128, 16, $\boxed{50)}$



Document: nn.Conv1d

Tensor Operations - nn.Conv1D

1D Convolutional layer

Given: Rotation: (2000, 31, 4)

Goal: We hope it can be [batch_size, channel, width]

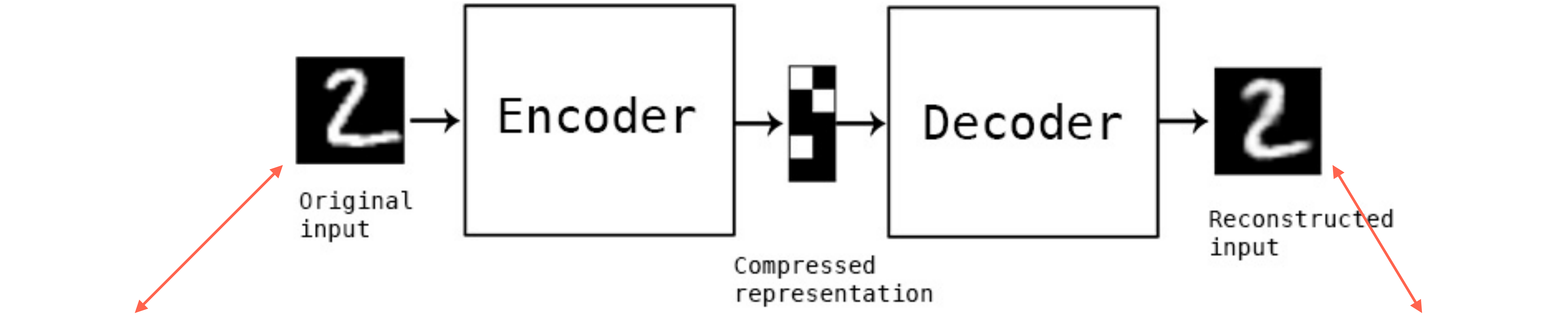
- Make motion clips -> take 60 frames as one clip

- (batch_size, 60, 31, 4) -> reshape -> (batch_size, 60, 124) -> transpose -> (batch_size, 124, 60)

- Apply conv1d

Document: nn.Conv1d

Motion AutoEncoder



AutoEncoder for Image

[Wiki](#)

An autoencoder is a type of artificial neural network used to learn efficient codings of unlabeled data (unsupervised learning).[1] The encoding is validated and refined by attempting to regenerate the input from the encoding. The autoencoder learns a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore insignificant data (“noise”).

Encoder - Linear

Map the input to code

```
## Encoder
rotations_tensor = torch.from_numpy(np.array(rotations.qs, dtype=np.float32))  # (2752, 31, 4)
rotations_tensor_item = rotations_tensor[0].reshape((1, 124))
fc_layer1 = nn.Linear(in_features=124, out_features=96)
fc_layer2 = nn.Linear(in_features=96, out_features=72)
fc_layer3 = nn.Linear(in_features=72, out_features=54)

output1 = fc_layer1(rotations_tensor_item)
output2 = fc_layer2(output1)
output3 = fc_layer3(output2)
```

Decoder - Linear

Map the code to input

```
## Decoder
fc_layer4 = nn.Linear(in_features=54, out_features=72)
fc_layer5 = nn.Linear(in_features=72, out_features=96)
fc_layer6 = nn.Linear(in_features=96, out_features=124)

output4 = fc_layer4(output3)
output5 = fc_layer5(output4)
output6 = fc_layer6(output5)
```

Encoder - Conv1D

Map the input to code

```
self.encoder = nn.Sequential(
    nn.Conv1d(in_channels=input_feature_size, out_channels=128, kernel_size=5, stride=1),
    nn.BatchNorm1d(128), nn.ReLU(True),
    nn.Conv1d(in_channels=128, out_channels=256, kernel_size=2, stride=1),
    nn.BatchNorm1d(256), nn.ReLU(True),
    nn.Conv1d(in_channels=256, out_channels=512, kernel_size=2, stride=1),
    nn.BatchNorm1d(512), nn.ReLU(True),
)
```

Decoder - Conv1D

Map the code to input

```
self.decoder = nn.Sequential(
    nn.ConvTranspose1d(in_channels=512, out_channels=256, kernel_size=2, stride=1),
    nn.BatchNorm1d(256), nn.ReLU(True),
    nn.ConvTranspose1d(in_channels=256, out_channels=128, kernel_size=2, stride=1),
    nn.BatchNorm1d(128), nn.ReLU(True),
    nn.ConvTranspose1d(in_channels=128, out_channels=input_feature_size, kernel_size=5, stride=1),
    nn.ReLU(True),
)
```

Refer to the document: nn.ConvTranspose1d

Task 2: Data-Driven Motion Processing

- ml_motion_denoising.py (25%) and ml_motion_interpolation.py (25%)
    - Implement model class and training code
    - Try to make the results to better
    - Reports
        - Visual performance
        - Network performance (a comparison table, including the losses and errors in different epoch/network structure/hyper parameters/conditions)
        - Conditions: noised_facter, betweening_frame_number

Tips

1. Ensure the AutoEncoder scripts can work, and then check the shape of each variable in the bugging mode.
2. Understand how network manipulates the input tensor
3. Try to take codes from an example file to meet different requirements.
    1. Denoising: Learning a mapping from noised data to clean data
    2. Interpolation: Learning a mapping from a masked motion to complete motion
4. Due to the computational limitation of the no-GPU environment, we only provide 47 motion clips to train the model so that the performance won't be so good (shaking, non-smooth). The motion quality from exampled AutoEncoder script can be seen as the baseline.