Multi-Agent Tensor Fusion for Contextual Trajectory Prediction

Tianyang Zhao[2,3], Yifei Xu[1,3], Mathew Monfort[1,4], Wongun Choi[1], Chris Baker[1], Yibiao Zhao [1], Yizhou Wang[2], Ying Nian Wu[1,3]

[1]*ISEE.AI*, [2]*Peking University*, [3]*UCLA*, [4]*MIT CSAIL*

{zhaotianyang, yizhou.wang}@pku.edu.cn, {mmonfort, wchoi, chrisbaker, yz}@isee.ai, fei960922@ucla.edu, ywu@stat.ucla.edu

1. Supplementary Material: Implementation Details

This section presents learning details for the Convolutional Spatial Fusion architecture.

For the image encoder, we use shallow CNNs for the driving datasets, considering that static scene context images from these datasets have already been annotated semantically; while we use a U-Net-like architecture which includes a pretrained ResNet as image encoder for Stanford Drone dataset, considering that raw images are fed. Architecture of the shallow CNNs is presented as follows. Input images are resized into $3 \times 60 \times 60$ (channel $\times$ width $\times$ height) and then go through 3 Convolutional layers with kernel sizes of $3, 4, 5$, filters $16, 16, 32$, paddings $1, 2, 2$ respectively, all with ReLU activations and Batch Normalization. Max Pooling of kernel size 2, stride 2 is operated after the second Convolutional layer. Architecture for the ResNet encoder is presented as follows. A pretrained ResNet18 on ImageNet is included. Output feature maps from the 2nd Convolutional module is concatenated with up-scaled output feature maps from the 3-rd and the 4-th Convolutional modules, where the up-scaling layers are Transpose Convolutional layers of filters $128, 128$, kernel sizes $3, 7$, strides $2, 4$, paddings $1, 3$, output paddings $1, 3$ respectively, all with ReLU activations and Batch Normalization. The concatenated output is then upsampled to shape $384 \times 30 \times 30$, and go through a 1×1 Convolutional bottleneck to reduce its channels to 32. The output feature map of this scene encoding module, either the shallow encoder or the ResNet one, is of shape $32 \times 30 \times 30$.

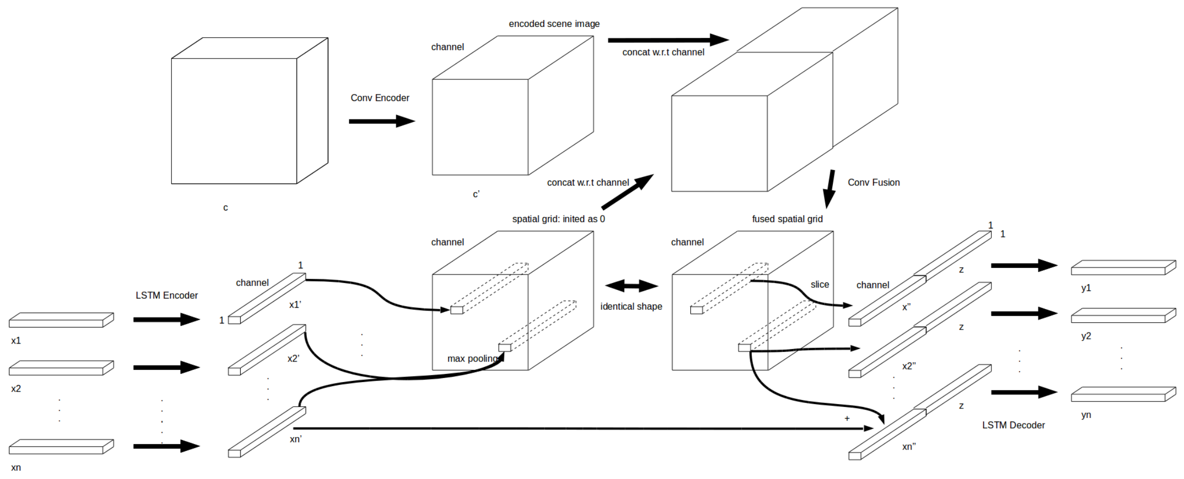
For the agent encoder, one-layer LSTMs of hidden state dimensions 32 are used. The input x, y coordinates are processed to be $\Delta x, \Delta y$, and are then embedded linearly to dimension 32 as inputs to the LSTMs. Dropout ratio in LTSMs are set to be 0.3 for training. The state tuples of the LSTMs are initialized as 0. Individual encoding of a given agent is its final state vector h of shape 32 of the LSTM.

The Multi-Agent Scene Spatial Fusion module operates at a resolution level of 30×30 with a U-Net-like architecture. The encoded agent grid and the encoded context grid are concatenated in their channel dimension to form a spatial grid of shape $64 \times 30 \times 30$. This input grid goes into 3 Convolutional layers sequentially of filters $32, 32, 32$, kernel sizes $3, 3, 4$, strides $1, 1, 1$, paddings $1, 1, 1$. After the first and the second layer, Max Pooling of kernel sizes $2, 2$ and strides $2, 2$ are performed respectively. The final output of this fusion module is the sum of 3 features maps: the output of the 1st Convolutional layer is added together with up-scaled feature maps of the 2nd and the 3rd feature maps. The up-scalings are performed with a Transpose Convolutional layer of kernel size 4, stride 2, padding 1, and an Upsampling layer of scale factor 5 respectively. All Convolutional and Transpose Convolutional layers operate ReLU activations and Batch Normalization. The output fused feature map is of shape $32 \times 30 \times 30$.

For the agent decoder, one-layer LSTMs of hidden state dimension 48 are used. Its state tuples of the LSTMs are initialized in the following way: h is initialized as 0, and c is initialized as $x'_i + x''_i$ (See Section 3.2) concatenated with a 16-dim z. z is 0 for all deterministic models and is sampled from Gaussian distribution for all stochastic models. The decoder decodes LSTM outputs h_t to $\Delta\hat{x}_t, \Delta\hat{y}_t$ linearly. These predicted $\Delta x_t, \Delta y_t$ are embedded linearly to dimension 48 as inputs to the LSTMs on future timestamps iteratively. $\Delta\hat{x}_t, \Delta\hat{y}_t$ are processed to form $\hat{x}_t, \hat{y}_t$ at last for calculating losses. Dropout ratio in LTSMs are set to be 0.3 for training.

The classifier of the discriminator is an MLP with a hidden layer size 512 and Leaky ReLU activation. Sigmoid activation is operated at last for $0 - 1$ classification.

We schedule the learning process as follows: only train *LSTM*, then train *Multi Agent* or *Single Agent Scene* from pretrained parameters of *LSTM* to learn residuals, then *Multi Agent Scene*, and finally *Individual D* or *Joint D*. Although all these architectures can be trained from scratch directly, this scheduled training process accelerates conver-



gence. While training *Multi Agent Scene*, the context scene image is randomly dropout of $p = 0.5$ to encourage the module not to be dependent too much on either scene or multi-agent information too much. We iteratively train the network with a batch size of 32 (note that 32 scenes per batch indicates much more than 32 agents given our multi-agent scenario) using Adam with an initial learning rate of 0.001. We use PyTorch for implementation.