Chip-seq modeling

Preprocessing

DNA sequence:
We used one-hot encoding, like this:

ATCGGCTA

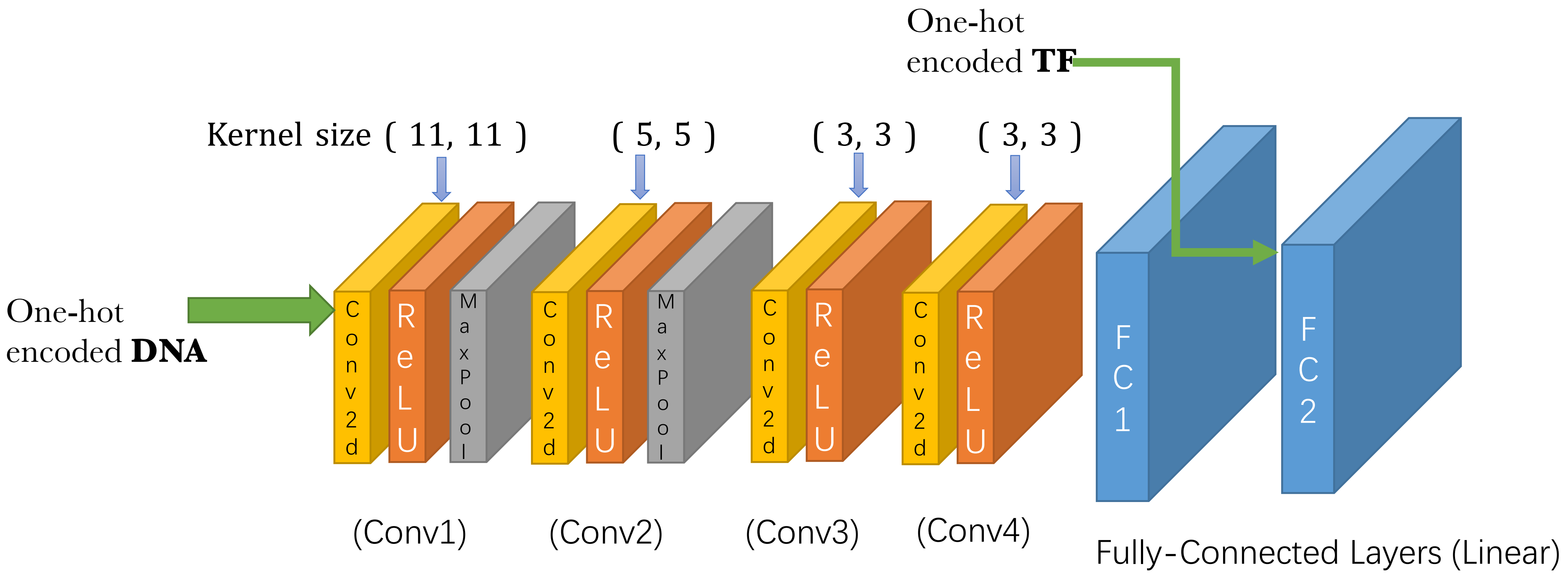
one-hot encoding

Position	0	1	2	3	4	5	6	7
A	🟧							🟧
C			🟧			🟧		
G				🟧	🟧			
T		🟧					🟧	

Transcription factor (TF) sequence:
We also use one-hot encoding, since each unique TF sequence represents/influences its biological functions.

Convolutional Neural Network, CNN

Why do we use this to accomplish the regression problem?
Most of the TF-DNA sequence has its score. Since enough amount of data are collected, we can use Convolutional Neural Network to predict the score of any given TF-DNA pair.



1. Data resource:
   TF
   DNA
   Score

2. Why we use this? P2
3. Training Platform : Pytorch
4. Data Preprocessing: P1

5. Function description:

  a. ReLU():
   Rectified Linear Unit. It is a kind of activation function commonly used in artificial neural network, represented by slope function(f(x) = max(0,x)), since its derivative = 1, it perfectly reduced gradient disappearance problem (GDP).

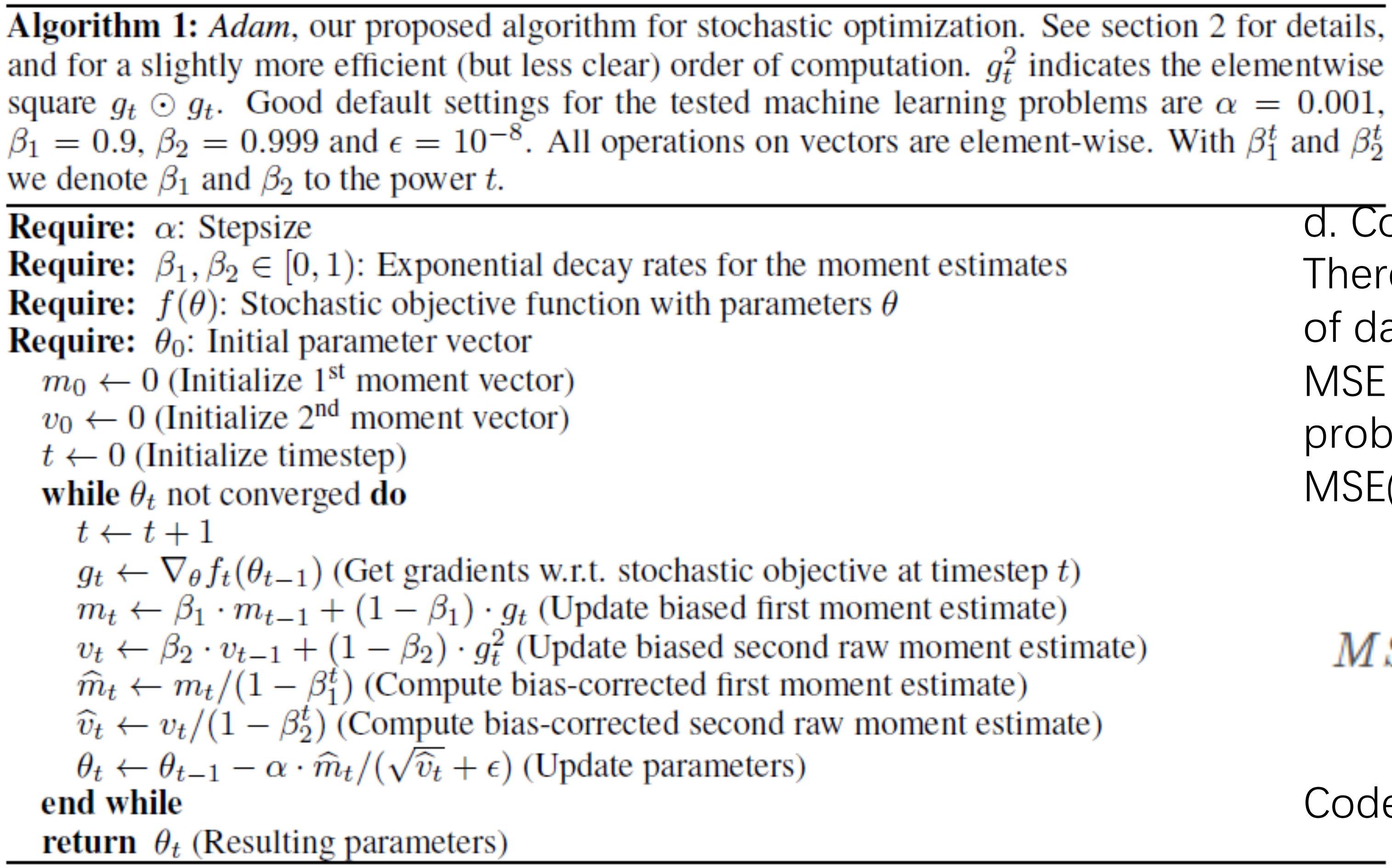
b. MaxPool2d:
   which applies a 2D max pooling over an input signal composed of several input planes.

Paramters used in our model:

kernel_size – the size of the window to take a max over
stride – the stride of the window. Default value is kernel_size
padding – implicit zero padding to be added on both sides
dilation – a parameter that controls the stride of elements in the window
return_indices – if True, will return the max indices along with the outputs.
ceil_mode – when True, will use *ceil* instead of *floor* to compute the output shape

c. Gradient Descent: use Adam

code : optimizer = torch.optim.Adam(···)



Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

d. Cost Function:

There are many loss functions, with a large amount of data, we may get a large loss in the beginning, MSE is a smart choice, it is ideal for regression problem.

MSE(Mean Square Error):

$$MSE(y, y') = \frac{\sum_{i=1}^{n} (y_i - y_i')^2}{n}$$

Code : loss_fn = torch.nn.MSELoss()

6. CNN (visualized) model : P2