# Protein Classes Library

September 10, 2018

## 1 Introduction

Protein folding is a problem of predicting the atomic structure of a protein given its amino-acid sequence. The challenging part of this problem is the complexity of the manifold to which belongs a protein structure. The key idea of the library is the local conversion of this manifold to the euclidian space, where deep learning techniques can be applied. After some machine learning algorithm provides prediction of the folding step in this local space, we have to convert this step back to the changes of the coordinates on the manifold.

## 2 Angles2Backbone

This layer takes a tensor of dihedral angles $\phi_i, \psi_i$ (batch size, 2, number of amino-acids) and outputs coordinates of N, CA, C atoms $\mathbf{x}_i$ (number of atoms * 3):

```
from ProteinClassesLibrary import Angles2Backbone

input_angles = torch.FloatTensor(batch_size, 2, num_amino_acids)
num_aa = torch.IntTensor(batch_size).fill_(num_amino_acids)
a2b = Angles2Backbone()
coords = a2b(Variable(input_angles.cuda()), Variable(num_aa.cuda()))
```

### 2.1 Forward pass

The position of the $i$-th atom in the chain is obtained by:

$$x_i = B_0 \cdot \cdots \cdot B_i x_0$$

where $B_i$ is the transformation matrix, parametrized by dihedral angles. During the forward pass, we save the products of transformation matrixes for each atom:

$$A_i = B_0 \cdot \cdots \cdot B_i$$

### 2.2 Backward pass

Notice, that the atom $3j$, where $j$ is integer is always N, $3j+1$ is CA and $3j+2$ is C. Thus transformation matrixes $B_{3j+1}$ and $B_{3j+2}$ depend on the angles $\phi_j$

and $\psi_j$ correspondingly. During the backward pass, we first compute gradients of each atomic coordinate w.r.t each angle:

$$\frac{\partial \mathbf{x}_i}{\partial \phi_j} = B_0 \dots \frac{\partial B_{3j+1}}{\partial \phi_j} \dots B_i \cdot \mathbf{x}_0$$

$$\frac{\partial \mathbf{x}_i}{\partial \psi_j} = B_0 \dots \frac{\partial B_{3j+2}}{\partial \psi_j} \dots B_i \cdot \mathbf{x}_0$$

We can rewrite these expressions using the matrixes $A$, that we saved during the forward pass:

$$\frac{\partial \mathbf{x}_i}{\partial \phi_j} = A_{3j} \frac{\partial B_{3j+1}}{\partial \phi_j} \cdot A3j + 1^{-1} A_i \cdot \mathbf{x}_0$$

$$\frac{\partial \mathbf{x}_i}{\partial \psi_j} = A_{3j+1} \frac{\partial B_{3j+2}}{\partial \psi_j} \cdot A3j + 2^{-1} A_i \cdot \mathbf{x}_0$$

We also notice that transformation matrixes have two properties:

1. Product of transformation matrixes is a transformation matrix, therefore matrixes $A$ are transformation matrixes
2. Inverse of transformation matrix can be computed as follows:
   If $T = \begin{bmatrix} R & \mathbf{x} \\ 0 & 1 \end{bmatrix}$ then $T^{-1} = \begin{bmatrix} R^T & -R^T \cdot \mathbf{x} \\ 0 & 1 \end{bmatrix}$

Therefore we can compute all the derivatives simultaneously on GPU. To compute the derivatives of input of the layer with respect to the output we just have to calculate the following sums:

$$\sum_i \frac{\partial L_k}{\mathbf{x}_i} \frac{\partial \mathbf{x}_i}{\partial \phi_j}$$

and

$$\sum_i \frac{\partial L_k}{\mathbf{x}_i} \frac{\partial \mathbf{x}_i}{\partial \psi_j}$$