

知能プログラミング演習 I 演習課題

1 準備

1.1 自前の環境の場合

- Moodle から課題を各自任意のフォルダにダウンロードし, 展開したフォルダの中に以下のものがすべて入っていることを確認
 - Tensor.py
 - Embedding.py
 - Trans.py
 - report.tex
 - report.pdf
 - task.pdf
- 前回から引き続き, **torch** と **torchvision** がインストールされている必要があるので注意.

```
$ pip install torch
```

```
$ pip install torchvision
```

1.2 CSE の場合

- まだ演習用のフォルダを作っていない人は DLL のフォルダを作成
 - ホームディレクトリに演習用のディレクトリを作成

```
step1: mkdir -p DLL
```
- 作業ディレクトリ DLL に移動

```
step1: cd ./DLL
```
- 今日の課題を DLL にダウンロードして展開
 - 展開したフォルダの中に, 以下のものがすべて入っていることを確認
 - * Tensor.py
 - * Embedding.py
 - * Trans.py
 - * report.tex
 - * report.pdf
 - * task.pdf

– Lec7 へ移動

step1: `cd ./Lec7`

2 課題

1. PyTorch を使った簡単な Transformer の作成に向けてまずは PyTorch における行列計算を確認する. `torch.tensor` は PyTorch で使われる行列を表現するクラスであり, NumPy の `ndarray` によく似ているが細かな違いも存在する. `torch.tensor` は自動微分に対応してるので, `torch.tensor` で作成した変数を使った演算に対して誤差逆伝播をさせることが可能であり, ネットワークを自分で定義する際には特に有用となる. `Tensor.py` でははじめに以下の変数

```
x = torch.tensor([1., 2., 3.])
y = torch.tensor([3., 2., 1.])
M = torch.tensor([[1., 2., 3.], [3., 2., 1.]])
N = torch.ones(2, 4, 3)
K = torch.arange(12).view(4,3)
A = torch.ones(2, 4, 5)
V = torch.ones(2, 5, 3)
V[1, :, :] = 2.
```

を定義したのち, 様々な演算を行っている. 変数の値や演算の結果を `print` するなどして, `torch.tensor` の計算について以下の問いに答えよ. この課題はコードの提出は不要とする.

注: `tensor` の 1 次元配列を `print` すると `tensor([1., 2., 3.])` のように横向きに表示されるが,

この資料の数学的な記法では縦ベクトルとして扱うこととする (つまり, $\begin{bmatrix} 1. \\ 2. \\ 3. \end{bmatrix}$). 一方, 2 次元配列を `print` すると `tensor([[1., 2., 3.], [3., 2., 1.]])` のように表示されるが, これは資料の数式上は $\begin{bmatrix} 1. & 2. & 3. \\ 3. & 2. & 1. \end{bmatrix}$ と解釈することとする.

- (a) `tensor` 配列 `x` 及び `y` の配列要素を以下のようにベクトル表記したとする.

$$\mathbf{x} : \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \mathbf{y} : \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

以下, 一つ目の例にならって `torch.tensor` に対する演算結果を表現するように, x_1, x_2, x_3 及び y_1, y_2, y_3 を使って空欄を埋めよ.

$$\mathbf{x} + \mathbf{y} : \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ x_3 + y_3 \end{bmatrix}, \mathbf{x} - \mathbf{y} : \begin{bmatrix} ??? \\ ??? \\ ??? \end{bmatrix}, \mathbf{x} * \mathbf{y} : \begin{bmatrix} ??? \\ ??? \\ ??? \end{bmatrix}, \mathbf{x} / \mathbf{y} : \begin{bmatrix} ??? \\ ??? \\ ??? \end{bmatrix}$$

- (b) `M` 及び `x` によって定義される配列要素を以下のようにベクトル表記したとする.

$$\mathbf{M} : \begin{bmatrix} \mathbf{m}_1^T \\ \mathbf{m}_2^T \end{bmatrix} \in \mathbb{R}^{2 \times 3}, \mathbf{x} : \mathbf{x} \in \mathbb{R}^3$$

$\mathbf{m}_1, \mathbf{m}_2 \in \mathbb{R}^3$ 及び \mathbf{x} を使って下の空欄を埋めよ. ただし, ベクトルの要素毎の積は \odot , 要素毎の

商は \odot とする (latex なら `\odot` と `\oslash`).

$$M + x : \begin{bmatrix} ??? \\ ??? \end{bmatrix}, M - x : \begin{bmatrix} ??? \\ ??? \end{bmatrix}, M * x : \begin{bmatrix} ??? \\ ??? \end{bmatrix}, M / x : \begin{bmatrix} ??? \\ ??? \end{bmatrix}$$

(c) N 及び K によって定義される配列要素を以下のように行列表記したとする.

$$N : [N_1, N_2] \in \mathbb{R}^{2 \times 4 \times 3}, K : K \in \mathbb{R}^{4 \times 3}$$

N の表記はあまり一般的ではないがここでは, $N_1 \in \mathbb{R}^{4 \times 3}$ が $N[0, :, :]$ であり, $N_2 \in \mathbb{R}^{4 \times 3}$ が $N[1, :, :]$ に相当するとする. N_1, N_2 及び K を使って下の空欄を埋めよ. またこの tensor 配列のサイズを答えよ.

$$N + K : [???, ???] \in \mathbb{R}^{??? \times ??? \times ???}$$

(d) $M: M \in \mathbb{R}^{2 \times 3}$ と $x: x \in \mathbb{R}^3$ とすると以下はそれぞれ

$$M @ x : Mx$$

$$M.transpose(0,1) : M^\top$$

$$M @ M.transpose(0,1) : MM^\top$$

という行列演算に対応する. では, N 及び M によって定義される配列要素を以下のようにベクトル表記したとする.

$$N : [N_1, N_2] \in \mathbb{R}^{2 \times 4 \times 3}, M : M \in \mathbb{R}^{3 \times 2}$$

$N_1, N_2 \in \mathbb{R}^{4 \times 3}$ 及び M を使って下の空欄を埋めよ. また配列サイズを答えよ.

$$N @ M.transpose(0,1) : [???, ???] \in \mathbb{R}^{??? \times ??? \times ???}$$

また,

`linear = nn.Linear(in_features=3, out_features=2, bias=False)`

として線形層を定義し, この層の重みパラメータ (`linear.weight` で参照可能) を

$$linear.weight : W \in \mathbb{R}^{??? \times ???}$$

とする. N_1, N_2, M, W を使って下の空欄を埋めよ. また, 上の W 及び下の二つそれぞれの配列サイズを答えよ.

$$linear(M) : ??? \in \mathbb{R}^{??? \times ???}$$

$$linear(N) : [???, ???] \in \mathbb{R}^{??? \times ??? \times ???}$$

(e) A 及び V によって定義される配列要素を以下のようにベクトル表記したとする.

$$A : [A_1, A_2] \in \mathbb{R}^{2 \times 4 \times 5}, V : [V_1, V_2] \in \mathbb{R}^{2 \times 5 \times 3}$$

$A_1, A_2 \in \mathbb{R}^{4 \times 5}$ 及び $V_1, V_2 \in \mathbb{R}^{5 \times 3}$ を使って下の空欄を埋めよ. また配列サイズを答えよ.

$$A @ V : [???, ???] \in \mathbb{R}^{??? \times ??? \times ???}$$

(f) N 及び V によって定義される配列要素を以下のようにベクトル表記したとする.

$$N : [N_1, N_2] \in \mathbb{R}^{2 \times 4 \times 3}, V : [V_1, V_2] \in \mathbb{R}^{2 \times 5 \times 3}$$

$N_1, N_2 \in \mathbb{R}^{4 \times 3}$ 及び $V_1, V_2 \in \mathbb{R}^{5 \times 3}$ を使って下の空欄を埋めよ. また配列サイズを答えよ.

$$V.\text{transpose}(1,2) : [???, ???]$$

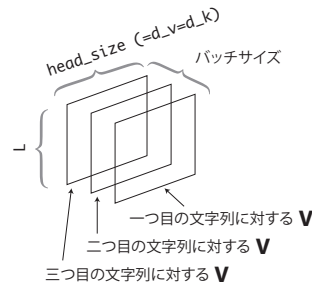
$$N @ V.\text{transpose}(1,2) : [???, ???] \in \mathbb{R}^{??? \times ??? \times ???}$$

(g) $F.\text{softmax}(N, \text{dim}=0)$, $F.\text{softmax}(N, \text{dim}=1)$, $F.\text{softmax}(N, \text{dim}=2)$ の結果を観察してなぜそうになるか数行程度で説明せよ.

2. `Embedding.py` はアルファベット一文字を Token として, 3 つの簡単な文字列を 2 次元空間に埋め込み, Positional Encoding と Self-Attention を実行する検証用のコードである (学習はせず初期重みで順伝播するだけ). 以下, 説明を求めるものについては 1~数行程度で述べること.

注: スライドの数式は全て, ある一つの入力文が与えられた時のものである. 一方, `Embedding.py` 含め通常の実装では複数の入力と同時に与えられた場合を想定する (ミニバッチに対する確率勾配法など) ため, プログラム上の配列と数式上の行列のサイズの違いに注意すること (具体的にはバッチサイズが最初の次元に入る).

- (a) tensor 配列 `input_idx` が何をどのように表現しているか説明せよ. 特に, 配列のサイズを明記し, 各次元が何に対応しているか述べること (例: サイズは $??? \times ???$ であり, 1 次元目が $???$, 2 次元目が $???$ に対応している).
- (b) `token_embedding = nn.Embedding(char_size, dim_embed)` により生成された `token_embedding` が何をを行うか説明せよ. 特に, `char_size`, `dim_embed` の意味することを明記すること. また, `x = token_embedding(input_idx)` によって作られた `x` のサイズを明記し, 各次元が何に対応しているか述べよ (例: `token_embedding` は行うのは $????$. `x` のサイズは $???$ であり, 1 次元目が....).
- (c) `embed-1.pdf` の図を説明せよ. 特に, 重複して同じ位置にある点がなぜそうなるのか明記すること (プロットされているテキストが被ってしまっているため視認しにくい, 処理を理解すれば必然的に何が重複するかはわかるはずなので考えてみる).
- (d) `PositionalEncoding` クラスの `pe` に適切な値を入れて Positional Encoding を完成させよ (配列 `pe` のどの要素に Positional Encoding のどの式が入るのべきか考えること).
ヒント: 三角関数には `np.sin`, `np.cos` または `torch.sin`, `torch.cos` を使うことができる (どちらで作成してもよい). ただし, `torch.sin`, `torch.cos` は入力が `torch.tensor` である必要があるので注意 (例: `torch.cos(torch.tensor(0))`)
結果として作成される `embed-2.pdf` の図を説明せよ. 特に, `embed-1.pdf` との重複の変化を明記すること.
- (e) `SelfAttention` クラスの `forward` 関数に (single-head の)Self-attention の順伝播計算を実装せよ. コンストラクタの引数 `dim_embed` が最初の Token 埋め込みの空間の次元, `head_size` はキー・クエリー・バリュー全て共通でこの次元とする (つまり, `head_size = d_v = d_k`). キー・クエリー・バリューのそれぞれについて, バッチ (一度に処理するデータ集合) の存在により, 3 次元配列となっていることに注意し, プログラム中の `K`, `Q`, `V` のサイズを確認して各軸の意味を理解して進めること (参考: 下図は `V` の場合). ヒント: バッチの存在を考慮した上で, 課題 1 の (e), (f), (g) を参考に演算を考える.



結果として作成される embed-3.pdf の図を説明せよ。特に、embed-2.pdf との重複の変化を明記すること。

3. Trans.py はトランスフォーマー (厳密には Encoder のみの Transformer) によってひらがなの文章生成を行うプログラムである。学習データとしてひらがなだけで書かれた昔話を用いる (mukashibanashi.txt^{*1})。ひらがな一文字を Token として扱い、ユーザーから 8 文字ひらがなの入力を受け取りその続きを生成する (厳密には訓練データに存在する文字を受け取る。それ以外の文字が入力されるとエラーで止まる)。まず、2. で作成した PositionalEncoding クラスと SelfAttention クラスの内容を Trans.py に転記せよ (これらは正しく作られていればそのまま用いることができる)。以下、説明を求めるものについては 1~数行程度で述べること。

- (a) 学習は `loss = loss_fn(logits, y)` が定める損失関数の最小化として定義されている。`loss_fn(logits, y)` が何を計算しているのか、コードを読んだり、変数を print するなどして理解し説明せよ。
- (b) Transformer クラスの `__init__` と `forward` を完成させよ。アーキテクチャは「Token 埋め込み → Positional encoding → Self-attention (single head) → 全結合層」とする。全結合層は「線形層 (output: 100 次元) → ReLU → 線形層 (output: char_size 次元)」とする。初期状態では埋め込みから直接線形層が実行されてるので書き換えること。実行後、変数 `input_str` に 8 文字のひらがな文字列を代入して `generate_text(model, input_str)` とすると文章生成が行われる (例: `generate_text(model, "ももからうまれた")`)。損失の推移 loss.pdf の図を示し、また、学習後に実行して生成された文章の例を 5 つ記載すること (同じ prompt から生成されたものを含めても全て異なる prompt からでもどちらでも構わない)。

^{*1} データの収集元サイト: http://hukumusume.com/douwa/0_6/index.html

3 課題の提出

Moodle を使ってファイルを提出してください。提出方法は以下の通りです。

- Moodle にログインし, 知能プログラミング演習のページへ移動。
- Lec7 の項目に, Embedding.py, Trans.py, レポートをアップロードする。

7/26(金) の 17:00 を提出期限とします。