RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

METEO FRANCE
À VOS CÔTÉS, DANS UN CLIMAT QUI CHANGE

Retour sur le TP n°4 (entraînement d'un Transformer)
Présentation du TP n°5 (FCNs)

Pierre Lepetit
ENM, le 28/11/2025

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?

```
class MultiHeadSelfAttention(nn.Module):

    def __init__(self, d_model: int = 64, n_heads: int = 4):
        super().__init__()
        assert d_model % n_heads == 0,
        self.d_model = d_model
        self.n_heads = n_heads
        self.d_head = d_model // n_heads
        self.verbose = verbose

        self.Wq = nn.Linear(d_model, d_model, bias=False)
        self.Wk = nn.Linear(d_model, d_model, bias=False)
        self.Wv = nn.Linear(d_model, d_model, bias=False)
        self.Wo = nn.Linear(d_model, d_model, bias=False)
```

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?

```
class MultiHeadSelfAttention(nn.Module):

    def __init__(self, d_model: int = 64, n_heads: int = 4):
        super().__init__()
        assert d_model % n_heads == 0,
        self.d_model = d_model
        self.n_heads = n_heads
        self.d_head = d_model // n_heads
        self.verbose = verbose

        self.Wq = nn.Linear(d_model, d_model, bias=False)
        self.Wk = nn.Linear(d_model, d_model, bias=False)
        self.Wv = nn.Linear(d_model, d_model, bias=False)
        self.Wo = nn.Linear(d_model, d_model, bias=False)
```

En sortie des Wq, Wk, Wv, les tenseurs sont « reshapés ».

Q.view(B, L, n_heads, d_head)

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?
- Calculer le nombre de poids dans une instance de `MultiHeadSelfAttention` ?

```
class MultiHeadSelfAttention(nn.Module):

    def __init__(self, d_model: int = 64, n_heads: int = 4):
        super().__init__()
        assert d_model % n_heads == 0,
        self.d_model = d_model
        self.n_heads = n_heads
        self.d_head = d_model // n_heads
        self.verbose = verbose

        self.Wq = nn.Linear(d_model, d_model, bias=False)
        self.Wk = nn.Linear(d_model, d_model, bias=False)
        self.Wv = nn.Linear(d_model, d_model, bias=False)
        self.Wo = nn.Linear(d_model, d_model, bias=False)
```

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?
- Calculer le nombre de poids dans une instance de `MultiHeadSelfAttention` ?

```
class MultiHeadSelfAttention(nn.Module):

    def __init__(self, d_model: int = 64, n_heads: int = 4):
        super().__init__()
        assert d_model % n_heads == 0,
        self.d_model = d_model
        self.n_heads = n_heads
        self.d_head = d_model // n_heads
        self.verbose = verbose

        self.Wq = nn.Linear(d_model, d_model, bias=False)
        self.Wk = nn.Linear(d_model, d_model, bias=False)
        self.Wv = nn.Linear(d_model, d_model, bias=False)
        self.Wo = nn.Linear(d_model, d_model, bias=False)
```

= 4 * d_model**2

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?
- Calculer le nombre de poids dans une instance de `MultiHeadSelfAttention` ?
- Calculer le nombre de poids dans une instance de `TransformerBlock` ?

```
class TransformerBlock(nn.Module):
    def __init__(self, d_model=64, n_heads=4, mlp_ratio=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.attn = MultiHeadSelfAttention(d_model,n_heads)
        self.ln2 = nn.LayerNorm(d_model)
        self.mlp = nn.Sequential(
            nn.Linear(d_model, d_model * mlp_ratio),
            nn.GELU(),
            nn.Linear(d_model * mlp_ratio, d_model),
        )
    def forward(self, x):
        x = x + self.attn(self.ln1(x))
        x = x + self.mlp(self.ln2(x))
        return x
```

= 4 * d_model**2
+ ...

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?
- Calculer le nombre de poids dans une instance de `MultiHeadSelfAttention` ?
- Calculer le nombre de poids dans une instance de `TransformerBlock` ?

```
class TransformerBlock(nn.Module):
    def __init__(self, d_model=64, n_heads=4, mlp_ratio=4):
        super().__init__()
        self.ln1 = nn.LayerNorm(d_model)
        self.attn = MultiHeadSelfAttention(d_model,n_heads)
        self.ln2 = nn.LayerNorm(d_model)
        self.mlp = nn.Sequential(
            nn.Linear(d_model, d_model * mlp_ratio),
            nn.GELU(),
            nn.Linear(d_model * mlp_ratio, d_model),
        )
    def forward(self, x):
        x = x + self.attn(self.ln1(x))
        x = x + self.mlp(self.ln2(x))
        return x
```

= 4 * d_model**2
+ 2 * d_model**2 * mlp_ratio

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?
- Calculer le nombre de poids dans une instance de `MultiHeadSelfAttention` ?
- Calculer le nombre de poids dans une instance de `TransformerBlock` ?
- Ce que sont `nn.GELU(), nn.LayerNorm` et pourquoi les utilise-t-on ?

```
...
        self.ln1 = nn.LayerNorm(d_model)
...
        nn.GELU(),
...
    def forward(self, x):
        x = x + self.attn(self.ln1(x))
        x = x + self.mlp(self.ln2(x))
...
```

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?
- Calculer le nombre de poids dans une instance de `MultiHeadSelfAttention` ?
- Calculer le nombre de poids dans une instance de `TransformerBlock` ?
- Ce que sont `nn.GELU(), nn.LayerNorm` et pourquoi les utilise-t-on ?

```
...
        self.ln1 = nn.LayerNorm(d_model)
...
        nn.GELU(),
...
    def forward(self, x):
        x = x + self.attn(self.ln1(x))
        x = x + self.mlp(self.ln2(x))
...
```

GELU =x Φ(x) (fonction C^∞)

Justification empirique : les transformers sont plus performants comme ça

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?
- Calculer le nombre de poids dans une instance de `MultiHeadSelfAttention` ?
- Calculer le nombre de poids dans une instance de `TransformerBlock` ?
- Ce que sont `nn.GELU()`, `nn.LayerNorm` et pourquoi les utilise-t-on ?

```
...
        self.ln1 = nn.LayerNorm(d_model)
...
        nn.GELU(),
...
    def forward(self, x):
        x = x + self.attn(self.ln1(x))
        x = x + self.mlp(self.ln2(x))
...
```

- LayerNorm = normalisation
- pour chaque token indépendamment des autres
- le long de la dernière dim. (d_model)

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?
- Calculer le nombre de poids dans une instance de `MultiHeadSelfAttention` ?
- Calculer le nombre de poids dans une instance de `TransformerBlock` ?
- Ce que sont `nn.GELU()`, `nn.LayerNorm` et pourquoi les utilise-t-on ?

```
...
        self.ln1 = nn.LayerNorm(d_model)
...
        nn.GELU(),
...
    def forward(self, x):
        x = x + self.attn(self.ln1(x))
        x = x + self.mlp(self.ln2(x))
...
```

Justifications :

- Petite bs pour les transformers → BatchNorm instable
- La BatchNorm briserait l'équivariance de l'attention
- ...

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?
- Calculer le nombre de poids dans une instance de `MultiHeadSelfAttention` ?
- Calculer le nombre de poids dans une instance de `TransformerBlock` ?
- Ce que sont `nn.GELU(), nn.LayerNorm` et pourquoi les utilise-t-on ?
- Pourquoi le mode résiduel dans la fonction `forward` ?

```
...
        self.ln1 = nn.LayerNorm(d_model)
...
        nn.GELU(),
...
    def forward(self, x):
        x = x + self.attn(self.ln1(x))
        x = x + self.mlp(self.ln2(x))
...
```

Retour sur le TP 4 :

Nous avons vu comment entraîner un transformer. Mais savez-vous :

- Pourquoi le nombre de self.Wq, self.Wk, self.Wv n'est pas proportionnel au nombre de têtes ?
- Calculer le nombre de poids dans une instance de `MultiHeadSelfAttention` ?
- Calculer le nombre de poids dans une instance de `TransformerBlock` ?
- Ce que sont `nn.GELU()`, `nn.LayerNorm` et pourquoi les utilise-t-on ?
- Pourquoi le mode résiduel dans la fonction `forward` ?

```
...
        self.ln1 = nn.LayerNorm(d_model)
...
        nn.GELU(),
...
    def forward(self, x):
        x = x + self.attn(self.ln1(x))
        x = x + self.mlp(self.ln2(x))
...
```

$\partial x_{k+1} / \partial x_k = \partial \text{Attn} / \partial x_k$
→ vanishing gradients

Residual connections :
$\partial x_{k+1} / \partial x_k = I + \partial \text{Attn} / \partial x_k$

(même idée que dans ResNet)

Présentation du TP5 :

TP ayant pour but :

- De revenir sur les couches de convolutions
  → notion de FCN

- D'introduire des tâches de prédiction « dense » sur des images (Segmentation, Débruitage)

- De sortir un peu du cadre supervisé standard
  → weak supervision

- De présenter le UNet

Projets : au moins 4 groupes de plus de 3 étudiants

- **Suivi des mouvements d'une caméra** :
  - Input : 500 groupes de 20 images prises par caméra « fixe »
  - Objectif : apprendre à suivre les mouvement de l'axe de visée au cours du temps (dus aux var. de température, à la maintenance, aux changements de paramétrisation)
- **Détection des inondations** :
  - Input : archives de quelques centaines de caméras qui montrent des variations du niveau d'eau visible (bord de rivière, prés inondables, bord de mer)
  - Objectif : parvenir à annoter les séries d'images convenablement, entraîner un modèle à suivre les variations sur :
    1/ les caméras vues à l'entrainement
    2/ de nouvelles caméras
- **Typage des nuage** :
  - Input : codes pour une annotation crowdsourcée, images des caméras ENM + sat.
  - Objectif : prédiction des **types de nuage** + **hauteur** + **nébul** sur les mêmes caméras