Support de cours
Apprentissage profond (Deep Learning)

Cycle Ingénieur
Filière INDIA (3ème année)

Pr. Abderrahim EL QADI

Département Mathématique appliquée et Génie Informatique
ENSAM-Université Mohammed V de Rabat

A.U. 2023/2024

Partie 2:

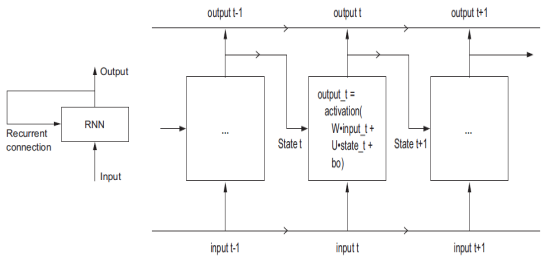
1. Réseaux de neurones recurrent RNN
2. RNN-LSTM

Réseau de neurones récurrent

Les réseaux de neurones récurrents (RNN) reposent sur deux principes:

- Le premier est l'astuce de la fenêtre glissante, qui permet de traiter des signaux de taille variable
- Le second est l'utilisation de connexions récurrentes qui permettent d'analyser la partie passée du signal.

Une connexion récurrente permet de réinjecter la sortie d'une couche en entrée de celle-ci. Comme toutes les autres connexions, on lui attribue un poids qui sera règle lors de la phase d'apprentissage.




```
state_t = 0
for input_t in input_sequence:
    output_t = f(input_t, state_t)
    state_t = output_t
```

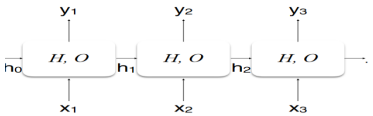
output_t = activation (dot(W, input_t) + dot(U, state_t) + b)

Applications de RNN

- Reconnaissance de la parole
- Traduction automatique
- Reconnaissance de l'écriture manuscrite/Reconnaissance optique des caractères
- Correction orthographique des phrases
- Synthèse, génération de dialogue, recherche d'informations, etc.
- Sous-titrage automatique des images
- RNN s'appuient sur les données passées pour prédire l'état futur, ils ont un intérêt pour le marché boursier

Déroulement d'un RNN

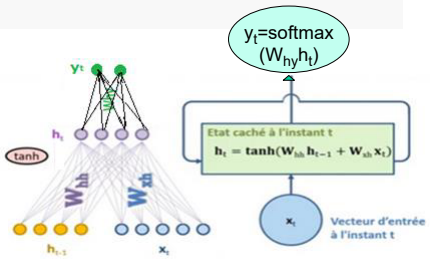
Un réseau récurrent parcourt successivement les entrées x_1 a x_T,
T étant la longueur de la **séquence**.
A l'instant t, la tème cellule combine l'entrée courante x_t avec l' état caché h_{t-1}, pour calculer l'état h_t



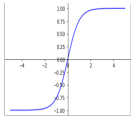
$h_t = f(x_t, h_{t-1})$

$h_t = \tanh(W^T * \text{concat}\ (x_t, h_{t-1}) + b)$

où **W** et **b** sont les poids appris par le modèle



La fonction d'activation classique utilisée est la tangente hyperbolique tanh (vu la convergence plus rapide que la sigmoïde



W_{xh} : matrice « input-hidden weights »
W_{hh} : matrice « hidden-hidden weights »
W_{hy} : matrice « hidden-output weights »

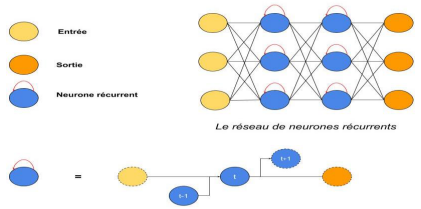
Pour t=0, on utilise un état caché initialisé aléatoirement.

Schéma d'un RNN

Contrairement à un ANN, sur chaque neurone bleu, on a une boucle (développée sous le schéma du réseau) :

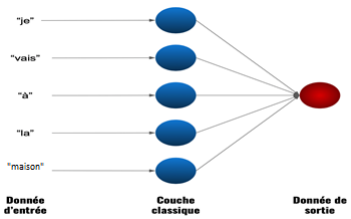
- on a une donnée t (par exemple 3 valeurs, une pour chaque neurone jaune) qui arrive dans le réseau
- celles-ci *se propagent* dans le réseau comme dans un ANN
- sauf que chaque neurone bleu, en plus de recevoir les sorties pondérées des neurones précédents, reçoit également *la valeur qui sortait de lui-même pour la donnée t-1*
- la valeur de sortie de chaque neurone bleu est conservée et servira pour la donnée t+1

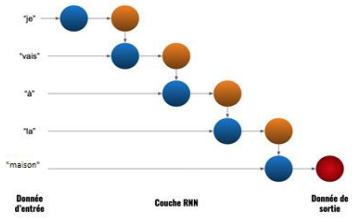
On a donc une **mémoire de 1 itération** pour les neurones bleus.



Exemple de schéma sur NLP

Ici, on calcul la signification de la phrase "je vais à la maison" :



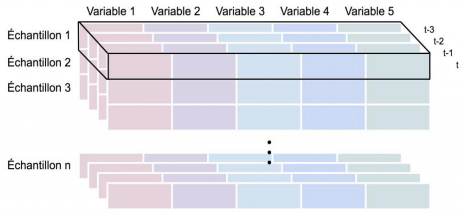


Dans une **couche non récurrente**, les neurones traitent bien **chaque donnée** (chaque mot) mais il n'y a **pas de partage d'information**. Il calcule **en bloc**.

Dans la **couche RNN**, chaque donnée a une **importance selon sa position**.
Le calcul de "je" a une importance dans le calcul de "vais" puis ces deux calculs ont une importance dans le calcul de "à".

Dimension du RNN

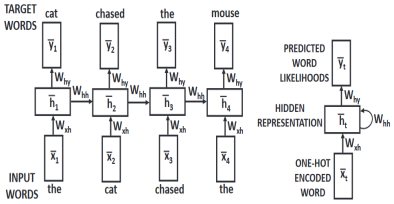
Un réseau de neurones récurrent attend en entrée une troisième dimension qui est la dimension temporelle **(nombre d'échantillons, longueur des séquences, nombre de variables)**.

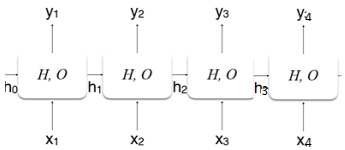


Exemple de schema : Prédiction du mot suivant

Un réseau récurrent peut par exemple génère une séquence de mots à partir d'un premier mot entré, « The cat » dans l'exemple suivant ("The cat chased the mouse")

La prédiction probabiliste du mot suivant pour chaque mot identifié temporellement de 1 à 4 est:

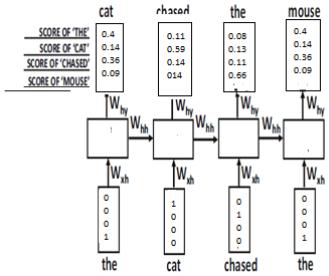




Les matrices de pondération W_{xh}, W_{hh} et W_{hy} sont partagées à travers les couches (architecture vectorielle). Une entrée x_t rencontre directement l'état caché construit de toutes les entrées avant

$$\overline{h_t} = f(\overline{h_{t-1}}, \overline{x_t})$$
$$= \tanh(w_{xh}\overline{x_t} + w_{hh}\overline{h_{t-1}})$$
$$\overline{y_t} = w_{hy}\overline{h_t}$$
$$\overline{h_1} = f(\overline{h_0}, x_1)$$

Exemple de schema : Prédiction du mot suivant

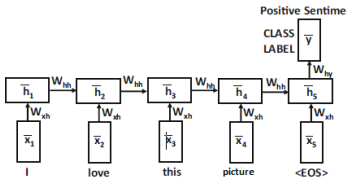


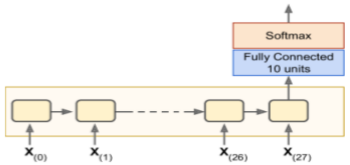

```
LENGTH_WORD=1
model = Sequential()
model.add(LSTM(16, input_shape=(LENGTH_WORD,
len(unique_words))))
model.add(Dense(len(unique_words)))
model.add(Activation('softmax'))

optimizer = RMSprop(lr=0.01)
model.compile(loss='categorical_crossentropy',
optimizer=optimizer, metrics=['accuracy'])
history = model.fit(X, y, batch_size=16, epochs=20,
shuffle=True).history
```

La dimension temporelle **(time_step, samples, features)=(4,1,4)**
input_shape=(1, 4)

Les sorties sont des valeurs continues (et non des probabilités) dans lesquelles des valeurs plus élevées indiquent une plus grande probabilité de présence.
Ces valeurs continues sont finalement converties en probabilités avec la fonction softmax.
Le mot « cat » est prédit dans le premier temps (t=0) avec une valeur de 0,4,
Cependant, le mot "chased semble être prédit correctement à t=1 avec une valeur de 0,59.

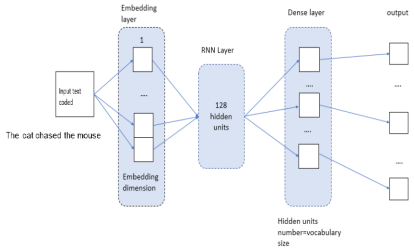
Exemple de schema : Analyse de sentiment





Ici, on reporte la sortie de la classe à la fin de la phrase.
Une seule étiquette de classe est prédite, et elle est utilisée pour rétropropager les erreurs de prédiction de classe.

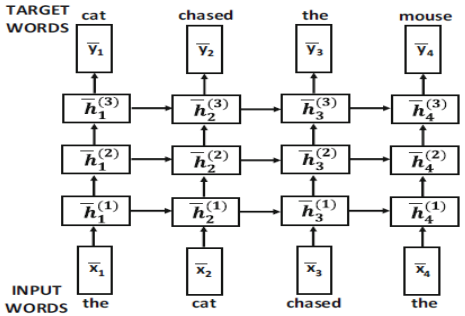
RNN-LSTM : codage avec TensorFlow-Keras



- Embedding layer: est utilisée pour obtenir les vecteurs de mots
- RNN layer: est considérée comme la couche centrale où elle est responsable du traitement des données,
- Dense layer: est responsable du traitement de la sortie, où elle prend comme paramètre les unités qui sont dans notre cas la taille du vocabulaire et la fonction d'activation où nous utilisons le Softmax.

```
# define model
model = Sequential()
model.add(LSTM(...))
# compile model
model.compile(...)
# fit model
model.fit(...)
# save model to single file
model.save( lstm_model.h5 )

from keras.models import load_model
# load model from single file
model = load_model( lstm_model.h5 )
# make predictions
yhat = model.predict(X, verbose=0)
print(yhat)
```

RNN multicouches



$$\overline{h}_t = \tanh(W_{xh}\overline{x}_t + W_{hh}\overline{h}_{t-1})$$
$$= \tanh W \begin{bmatrix} \overline{x}_t \\ \overline{h}_{t-1} \end{bmatrix}$$

Apprentissage de RNN

Pour une séquence d'entrée donnée :

- créer le réseau RNN
- ajouter un nœud de fonction de perte au réseau
- ensuite, utiliser la rétro propagation pour calculer les gradients. Cet algorithme est connu sous le nom de rétro propagation dans le temps (BPTT)

Algorithme BPTT

(i) Exécution séquentiellement les entrées et calcul les erreurs (et la perte de log négatif de la couche softmax) à chaque étape.

(ii) Calcul les gradients des poids des bords vers l'arrière sur le réseau déployé sans tenir compte du fait que les poids dans différentes couches temporelles sont partagés.

(iii) Addition tous les poids (partagés) correspondant aux différentes instanciations d'une arête dans le temps.

$$[\hat{p}_t^1 \dots \hat{p}_t^d] = \text{Softmax}([\hat{y}_t^1 \dots \hat{y}_t^d])$$
$$L = -\sum_{t=1}^{T} \log(\hat{p}_t^{j_t})$$
$$\frac{\partial L}{\partial \hat{y}_t^k} = \hat{p}_t^k - I(k, j_t)$$

$$\frac{\partial L}{\partial W_{xh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial W_{xh}^{(t)}} \qquad \frac{\partial L}{\partial W_{hh}} = \sum_{t=1}^{T} \frac{\partial L}{\partial W_{hh}^{(t)}} \qquad \frac{\partial L}{\partial W_{hy}} = \sum_{t=1}^{T} \frac{\partial L}{\partial W_{hy}^{(t)}}$$

Problème d'apprentissage et de mémoire

Les RNN souffrent de mémoire à court terme qui apparaît lorsqu'on travaille avec des séquences de données plus longues.

Pour « apprendre », un RNN utilise la méthode de la descente du gradient afin de mettre à jour les poids entre ses neurones. Cela repose sur la formule suivante :

$w = w - \alpha \cdot Fw$

avec :

w un poids du réseau
α la vitesse d'apprentissage du réseau
Fw le gradient du réseau par rapport au poids *w*

$h_1 = f_W(x_1, h_0)$
$h_2 = f_W(x_2, h_1) = f_W(x_2, f_W(x_1, h_0))$
$h_3 = f_W(x_3, h_2) = f_W(x_3, f_W(x_2, f_W(x_1, h_0)))$

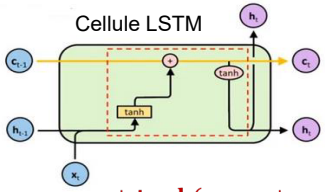
Problème : la mise à jour des poids se fait de droite à gauche.

A mesure que l'on avance vers la gauche, le produit *α·Fw* devient **très petit** et les poids des premières couches de neurones ne sont quasiment pas modifiés
Ainsi, ces couches n'apprennent strictement rien…

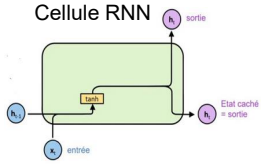
Et par conséquent, le RNN peut *facilement oublier* des données un petit peu anciennes lors de la phase d'apprentissage : **sa mémoire est courte**.

Vers une meilleure architecture : LSTM

LSTM (*Long Short-Term Memory)* ce sont des RNN qui **possèdent une mémoire interne appelée *cellule*** (ou *cell*).
La cellule permet de **maintenir un état aussi longtemps que nécessaire**.
Cette cellule consiste en une valeur numérique que le réseau peut piloter en fonction des situations.
C'est une nouvelle variable c, qui permettra de se souvenir des informations pendant de longues périodes. c_t est appelée état de la cellule



$$c_t = c_{t-1} + \tanh(w_{xh}x_t + w_{hh}h_{t-1})$$
$$h_t = \tanh(c_t)$$



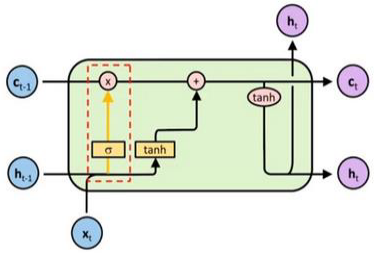
$$h_t = \tanh(w_{xh}x_t + w_{hh}h_{t-1})$$

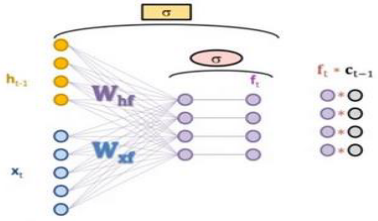
L' état caché h_t, calculé à l'instant t tient compte de tous les états cachés précédents.

Les portes de la cellule LSTM

la cellule mémoire peut être pilotée par **trois portes** de contrôle qu'on peut voir comme des vannes.

La porte d'oubli (**forget gate**) décide s'il faut **remettre à 0** le contenu de la cellule ;
Le réseau va apprendre **à oublier les informations non pertinentes**.
La porte oubli permet au réseau de garder (multiplication par 1) ou d'oublier (multiplication par 0) les informations du passé selon qu'elles soient déterminantes ou pas.



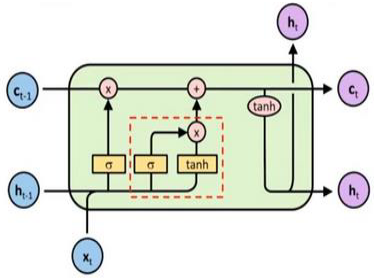


$$c_t = f_t \times c_{t\_1} + \tanh(w_{xh}x_t + w_{hh}h_{t\_1}) \qquad h_t = \tanh(c_t)$$

$$\mathbf{f_t = \sigma(w_{xf}x_t + w_{hf}h_{t\_1})} \qquad \sigma(x) = sigmoid(x) = 1/(1 + exp - ^x)$$

Les portes de la cellule LSTM

La porte d'entrée (Input gate) décide si l'entrée doit **modifier le contenu** de la cellule.
Elle détermine quelles informations produites par la couche tanh doivent entrer dans l'état de la cellule (sauvegarder dans la mémoire à long terme).



$$c_t = f_t \times c_{t\_1} + \mathbf{i_t} \times \tanh(w_{xh}x_t + w_{hh}h_{t\_1})$$

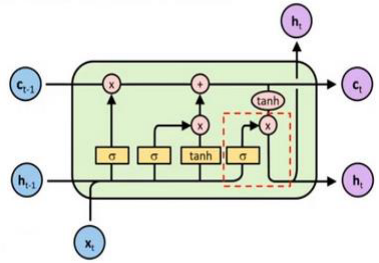
$$f_t = \sigma(w_{xf}x_t + w_{hf}h_{t\_1})$$

$$\mathbf{i_t = \sigma(w_{xi}x_t + w_{hi}h_{t\_1})}$$

$$h_t = \tanh(c_t)$$

Les portes de la cellule LSTM

La porte de sortie (Output gate) décide si le contenu de la cellule doit **influer sur la sortie** du neurone.
Elle indique les informations qui doivent passer au prochain état caché h_t.



$$f_t = \sigma(w_{xf}x_t + w_{hf}h_{t\_1})$$

$$i_t = \sigma(w_{xi}x_t + w_{hi}h_{t\_1})$$

$$\mathbf{o_t = \sigma(w_{xo}x_t + w_{ho}h_{t\_1})}$$

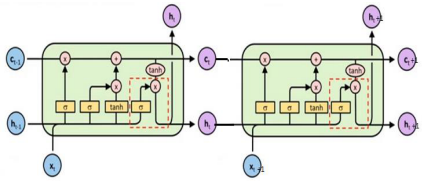
$$c_t = f_t \times c_{t\_1} + i_t \times \tanh(w_{xh}x_t + w_{hh}h_{t\_1})$$

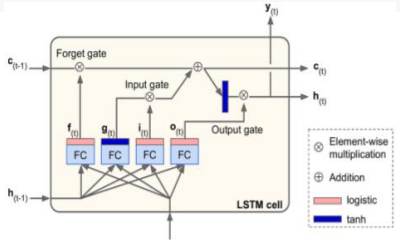
$$h_t = \mathbf{o_t} \times \tanh(c_t)$$

Les portes de la cellule LSTM

Etapes de LSTM:

1. Dans la première étape, nous avons découvert ce qui devait être supprimé.
2. La deuxième étape consistait à déterminer quelles nouvelles entrées sont ajoutées au réseau.
3. La troisième étape consistait à combiner les entrées précédemment obtenues pour générer les nouveaux états de cellule.
4. Enfin, nous sommes arrivés à la sortie selon l'exigence.





$$f_t = \sigma(w_{xf}x_t + w_{hf}h_{t\_1})$$

$$\mathbf{i_t = \sigma(w_{xi}x_t + w_{hi}h_{t\_1})}$$

$$\mathbf{o_t = \sigma(w_{xo}x_t + w_{ho}h_{t\_1})}$$

$$\mathbf{c_t = f_t \times c_{t\_1} + i_t \times \tanh(w_{xh}x_t + w_{hh}h_{t\_1})}$$

$$\mathbf{h_t = o_t \times \tanh(c_t)}$$

Apprentissage des LSTM

Les LSTM sont une version améliorée des réseaux de neurones récurrents simples, capables de modéliser des dépendances a très long terme.

Les LSTM sont constitués de sommes pondérées, de fonctions d'activations différentiables et de connexions récurrentes.

Ils peuvent donc s'apprendre avec l'algorithme **BPTT** (Backpropagation Through Time) comme pour les réseaux récurrents classiques, en dépliant le réseau récurrent à travers le temps.

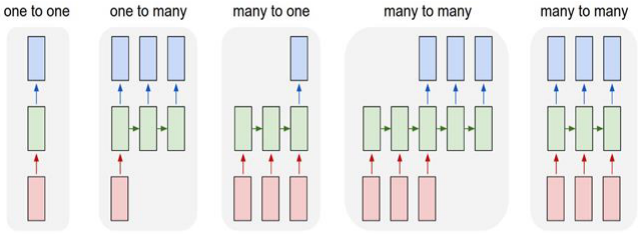
Utilisation RNN-LSTM

Nous pourrions utiliser les LSTM de quatre manières différentes :

- Un à un : théoriquement possible, mais étant donné qu'un élément n'est pas une séquence, aucun avantage offert par les LSTM.

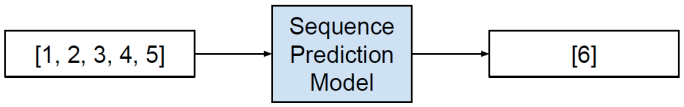
Il est préférable d'utiliser plutôt un réseau neuronal Feed-Forward dans un tel scénario.

- Un à plusieurs : utilisation d'une valeur pour prédire une séquence de valeurs.
- Plusieurs à un : utilisation d'une séquence de valeurs pour prédire la valeur suivante.
- Plusieurs à plusieurs : utilisation d'une séquence de valeurs pour prédire la prochaine séquence de valeurs.



RNN-LSTM : Prédiction de séquence

consiste à prédire la valeur suivante pour une séquence d'entrée donnée.

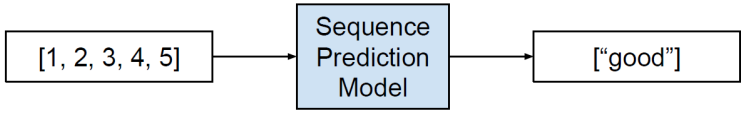


Exemples de problèmes de prédiction de séquence

- Prévision météo: Compte tenu d'une séquence d'observations sur le temps au fil du temps, prédire le temps prévu demain.
- Prédiction du marché boursier: Étant donné une séquence de mouvements d'un objet dans le temps, prédire le prochain mouvement de l'objet.
- Recommandation de produit: Compte tenu d'une séquence d'achats passés pour un client, prédire le prochain achat pour un client.

RNN-LSTM : Classification de séquence

implique de prédire une étiquette de classe pour une séquence d'entrée donnée.

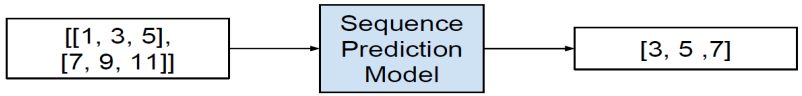


Exemples de problèmes de classification de séquence

- Classification des séquences d'ADN: Étant donné une séquence d'ADN de valeurs A, C, G et T, prédire si la séquence est correcte ou non.
- Détection d'une anomalie: Étant donné une séquence d'observations, prédire si la séquence est anormale ou non.
- Analyse des sentiments: Étant donné une séquence de texte (ex. tweet), prédire si le sentiment du texte est positif ou négatif.

RNN-LSTM : Génération de Séquence

consiste à générer une nouvelle séquence de sortie qui a le même caractéristiques que les autres séquences du corpus.

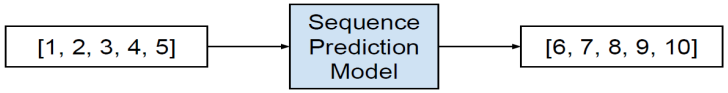


Exemples de problèmes de génération de séquence

- Génération de texte: Étant donné un corpus de texte, générer de nouvelles phrases ou paragraphes de texte qu'ils auraient pu être tirés du corpus.
- Prédiction d'écriture manuscrite: Étant donné un corpus d'exemples d'écriture manuscrite, générer une écriture manuscrite pour de nouvelles phrases qui ont les propriétés de l'écriture manuscrite dans le corpus.
- Génération musicale. Étant donné un corpus d'exemples de musique, générer de nouvelles pièces musicales qui ont les propriétés du corpus.
- Génération de légendes d'image (Automatic Image Caption Generation): Étant donné une image en entrée, générer une séquence de mots qui décrivent une image.

RNN-LSTM : Sequence-to-Sequence Prédiction

implique de prédire une séquence de sortie étant donné une séquence d'entrée



Exemples de problèmes sequence-sequence

- Prévision de séries chronologiques à plusieurs étapes: Étant donné une série chronologique d'observations, prédire une séquence d'observations pour une gamme d'étapes de temps futures.
- Résumé de texte: Étant donné un document de texte, prédire une séquence de texte plus courte qui décrit les parties saillantes du document source.
- Exécution du programme: Étant donné le programme de description textuelle ou l'équation mathématique, prédire la séquence de caractères qui décrit la sortie correcte.