Concepts Généraux

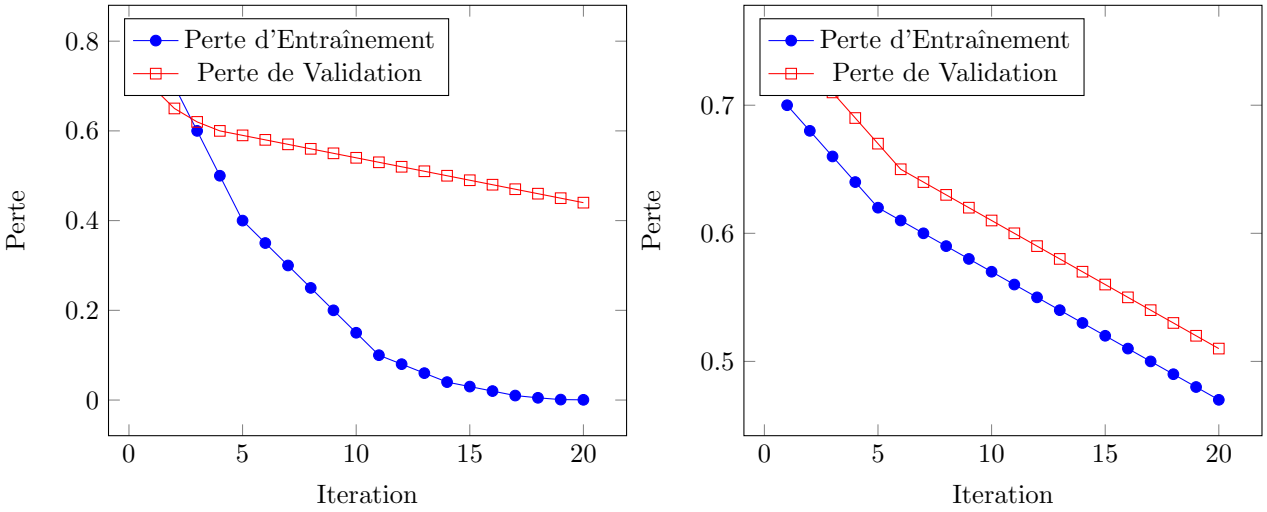
1. Comment les gradients qui disparaissent peuvent-ils affecter les performances des Réseaux de Neurones Récurrents (RNN), et quelles sont certaines stratégies pour atténuer ce problème ?
2. Discutez le concept de recadrage de gradient et son importance dans l'entraînement des RNN et LSTMs.
3. Expliquez la différence fondamentale entre les réseaux de neurones à propagation avant et les réseaux de neurones récurrents. Pourquoi les RNN sont-ils bien adaptés au traitement des données séquentielles ?
4. Décrivez les problèmes de gradient qui disparaît et qui explose dans les RNN. Comment l'architecture LSTM tente-t-elle d'atténuer ces problèmes ?
5. Expliquez le concept de "porte" dans le contexte des LSTMs et GRUs. Comment les portes aident-elles à capturer les dépendances à long terme ?
6. Discutez des avantages et inconvénients de l'utilisation des RNN par rapport à d'autres approches de modélisation de séquence, telles que les Modèles de Markov Cachés (HMM) et les Champs Aléatoires Conditionnels (CRFs).
7. Quel est le rôle de l'état caché initial dans un RNN ? Comment peut-il être initialisé, et comment cela affecte-t-il les performances du modèle ?
8. Expliquez le concept de RNN bidirectionnels (BRNNs) et leurs applications dans des tâches telles que la reconnaissance vocale et le traitement du langage naturel.
9. Discutez des défis de l'application des RNN à des séquences très longues et certaines solutions potentielles, telles que les mécanismes d'attention et les modèles hiérarchiques.

Concepts Mathématiques

10. Étant donné un RNN avec des fonctions d'activation sigmoïdes, montrez mathématiquement pourquoi les gradients ont tendance à disparaître lorsqu'ils sont propagés en arrière à travers chaque étape de temps. Utilisez la règle de la chaîne du calcul pour soutenir votre explication.
11. Déduisez les équations de mise à jour de l'état de cellule et de l'état caché dans un réseau LSTM. Expliquez le rôle de la porte d'oubli dans ces équations.
12. Comparez les mécanismes de portes des LSTMs et des GRUs en fournissant les équations de mise à jour pour les deux. Mettez en évidence comment les GRUs obtiennent un effet similaire aux LSTMs avec une structure simplifiée.
13. Déduisez l'algorithme de la rétropropagation à travers le temps (BPTT) pour les RNNs, et expliquez comment il est utilisé pour calculer les gradients pour la mise à jour des poids.
14. Montrez la formulation mathématique du mécanisme d'attention utilisé dans les modèles de séquence à séquence, et expliquez comment cela aide à traiter les longues séquences.
15. Déduisez les équations pour les connexions de guet dans les LSTMs, et discutez de leurs avantages et inconvénients potentiels.
16. Expliquez les fondements mathématiques de l'Unité Récurrente à Portes (GRU) et comment elle diffère du LSTM en termes de mécanismes de guet.
17. Déduisez les équations de mise à jour pour un RNN bidirectionnel, et discutez de comment les états cachés avant et arrière sont combinés pour capturer à la fois le contexte passé et futur.

Implémentation

18. Analysez les courbes de perte d'entraînement et de validation fournies d'un modèle LSTM entraîné sur une tâche de classification de séquences. Identifiez les problèmes potentiels tels que le sur-apprentissage ou le sous-apprentissage et suggérez des modifications spécifiques pour y remédier.



19. Proposez un guide de dépannage étape par étape pour un modèle LSTM qui échoue à converger sur une tâche de génération de texte. Incluez des considérations pour l'ajustement des hyperparamètres, le recadrage des gradients et les programmes de taux d'apprentissage.

20. Identifiez et corrigez les erreurs dans la mise en œuvre LSTM fournie dans la Figure 1.

21. Implémentez un modèle LSTM pour l'analyse de sentiment des critiques de films en utilisant PyTorch ou TensorFlow. Décrivez les étapes de prétraitement, l'architecture du modèle et la procédure d'entraînement.

22. Expliquez la technique de "forcing de l'enseignant" utilisée dans les modèles de séquence à séquence, et discutez de ses avantages et inconvénients par rapport à d'autres stratégies d'entraînement.

23. Proposez une méthode pour visualiser les poids d'attention dans un modèle de traduction automatique basé sur LSTM, et discutez de comment de telles visualisations peuvent aider dans l'interprétabilité et le débogage du modèle.

24. Décrivez comment vous appliqueriez des techniques d'apprentissage par transfert pour affiner un modèle LSTM pré-entraîné pour une tâche différente d'étiquetage de séquence, telle que la reconnaissance d'entités nommées ou l'étiquetage des parties du discours.

25. Implémentez un modèle de langue au niveau des caractères en utilisant un LSTM ou GRU dans PyTorch ou TensorFlow. Discutez des défis liés au travail avec des modèles au niveau des caractères et des techniques potentielles pour améliorer leurs performances.

26. Décrivez comment vous utiliseriez un modèle basé sur RNN pour la prévision de séries temporelles, telles que la prédiction des prix des actions ou de la demande en énergie. Discutez des choix architecturaux spécifiques et des considérations pour cette tâche.

27. Proposez une méthode pour assembler plusieurs modèles LSTM pour une tâche de classification de séquences, et discutez des avantages et défis potentiels d'une telle approche.

```python
import tensorflow as tf

# Import the TensorFlow library

# Define the RNN model
class RNN(tf.keras.Model):
    def __init__(self, input_size, hidden_size, output_size):
        # Initialize the RNN model by inheriting from tf.keras.Model
        # input_size: size of the input vector
        # hidden_size: size of the hidden state in the RNN
        # output_size: size of the output vector
        super(RNN, self).__init__()
        # Call the constructor of the parent class (tf.keras.Model)
        self.rnn = tf.keras.layers.SimpleRNN(hidden_size, return_sequences=True)
        # Create a SimpleRNN layer with the specified hidden_size
        # return_sequences=True returns the full sequence of hidden states
        self.fc = tf.keras.layers.Dense(output_size)
        # Create a Dense layer with the specified output_size

    def call(self, x):
        # Define the forward pass of the model
        # x: input tensor of shape (batch_size, sequence_length, input_size)
        out = self.rnn(x)
        # Pass the input through the RNN layer
        # out: tensor of shape (batch_size, sequence_length, hidden_size)
        out = self.fc(out[:, -1, :])
        # Pass the last hidden state through the Dense layer
        # out: tensor of shape (batch_size, output_size)
        return out

# Hyperparameters
input_size = 10
hidden_size = 20
output_size = 5
batch_size = 32
num_epochs = 100
learning_rate = 0.01

# Create the model, loss function, and optimizer
model = RNN(input_size, hidden_size, output_size)
# Create an instance of the RNN model
loss_fn = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
# Create a loss function for classification (SparseCategoricalCrossentropy)
# from_logits=True means the input to the loss is unnormalized logits
optimizer = tf.keras.optimizers.SGD(learning_rate=learning_rate)
# Create an SGD optimizer with the specified learning rate

# Training Loop
for epoch in range(num_epochs):
    # Generate some dummy input and target data
    inputs = tf.random.normal((batch_size, 10, input_size))
    # Generate random input data with shape (batch_size, 10, input_size)
    targets = tf.random.uniform((batch_size,), maxval=output_size, dtype=tf.int32)
    # Generate random target data with shape (batch_size,) and values between 0 and output_size-1

    with tf.GradientTape() as tape:
        # Create a GradientTape to record operations for computing gradients
        outputs = model(inputs)
        # Forward pass through the model to get the output
        loss = loss_fn(targets, outputs)
        # Compute the loss between the targets and outputs

    gradients = tape.gradient(loss, model.trainable_variables)
    # Compute the gradients of the loss with respect to the trainable variables
    optimizer.apply_gradients(zip(gradients, model.trainable_variables))
    # Update the trainable variables using the computed gradients

    if epoch % 10 == 0:
        print(f'Epoch [{epoch+1}/{num_epochs}], Loss: {loss.numpy():.4f}')
        # Print the epoch number and loss every 10 epochs
```

Figure 1: Extrait de code d'entraînement LSTM