Nome: Raphael Baptista da Silva Fontenelle

Relatório de RNN - VIDEO CLASSIFICATION USING RECURRENT NEURAL NETWORKS

Objetivo

Avalie o desempenho das redes neurais recorrentes para classificação de vídeos. Dado um conjunto de vídeos com humanos em atividade, foi feita a implementação de modelos de RNN para identificar a atividade sendo realizada. O programa atribui cada vídeo a uma das seguintes dez categorias: arremesso de basquete, andar de bicicleta, mergulho, balanço de golfe, montaria, embaixadinhas de futebol, balanço, balanço de tênis, pulo em trampolim, cortada de vôlei e caminhada com cachorro.

Arquitetura do Modelo

LSTM, ou **Long Short-Term Memory**, é um tipo especial de rede neural recorrente (RNN) projetada para lidar com o problema do desaparecimento do gradiente, que afeta as RNNs convencionais. As LSTMs são especialmente adequadas para aprender dependências de longo prazo em sequências de dados, tornando-as eficazes para tarefas como processamento de linguagem natural, previsão de séries temporais e reconhecimento de fala.

A arquitetura de uma LSTM é composta por unidades chamadas células de memória, cada uma contendo três portas reguladoras que controlam o fluxo de informações:

1. **Estado da célula (Cell state)**: É a linha central que passa pela célula e carrega as informações ao longo de toda a sequência de tempo, com pequenas modificações lineares. É a "memória" da rede, que transporta informações relevantes ao longo do processamento da sequência.
2. **Porta de esquecimento (Forget gate)**: Decide quais informações serão descartadas do estado da célula. Ela olha para a entrada atual $xtxt$ e o estado oculto anterior ht−1ht−1, e usa uma função sigmoid para retornar valores entre 0 e 1 para cada número no estado da célula. Um valor próximo a 0 significa "esquecer completamente", enquanto um valor próximo a 1 significa "manter completamente".

3. **Porta de entrada (Input gate)**: Decide quais novas informações serão adicionadas ao estado da célula. Primeiro, uma função sigmoid decide quais valores serão atualizados, e depois uma função tanh cria um vetor de novos valores candidatos, $C\sim tC\sim t$, que poderiam ser adicionados ao estado.
4. **Atualização do estado da célula**: O estado antigo da célula, $Ct-1Ct-1$, é atualizado para o novo estado $CtCt$. As informações antigas são esquecidas (conforme determinado pela porta de esquecimento) e as novas informações são adicionadas (conforme determinado pela porta de entrada).
5. **Porta de saída (Output gate)**: Decide qual a próxima saída oculta, $htht$, baseada no estado atual da célula, filtrando as informações através de uma função sigmoid. O estado da célula passa por uma função tanh e é multiplicado pelo resultado da função sigmoid para determinar quais informações o estado oculto deve passar como saída.
6. **Saída oculta (Hidden output)**: O resultado da porta de saída é o estado oculto $htht$, que é passado para a próxima etapa de tempo e pode ser usado para a saída da sequência.

Essa estrutura complexa é o que permite que as LSTMs lidem eficazmente com problemas de sequência onde a dependência de longo prazo é uma preocupação chave.

Deep LSTM: A técnica conhecida como "deep LSTM" refere-se à prática de empilhar múltiplas camadas de células LSTM em uma rede neural. Essa abordagem é uma forma de deep learning em redes neurais recorrentes, onde cada camada de LSTM passa suas saídas (estados ocultos) para a camada subsequente como entrada. Quando você tem "two stacked LSTM layers" (duas camadas de LSTM empilhadas), isso significa que a saída de cada célula LSTM na primeira camada serve como entrada para cada célula na segunda camada.

Vantagens de Deep LSTM

1. **Capacidade de abstração mais alta**: Com múltiplas camadas, a rede pode aprender representações mais complexas e abstratas dos dados. A primeira camada pode capturar padrões básicos, enquanto camadas subsequentes podem aprender a interpretar esses padrões em um contexto mais amplo ou mais abstrato.
2. **Melhor desempenho em tarefas complexas**: Redes mais profundas são geralmente mais eficazes para tarefas complexas que envolvem sequências de longa duração ou dependências complicadas, como tradução automática, modelagem de linguagem e geração de texto.
3. **Capacidade de memória melhorada**: Camadas adicionais podem ajudar a rede a manter informações úteis por períodos mais longos, o que é crucial em tarefas onde o contexto histórico distante influencia as decisões atuais.

Desafios de Deep LSTM

1. **Risco de overfitting**: Com mais parâmetros para aprender, as redes profundas correm maior risco de overfitting, especialmente se o conjunto de dados de treinamento for limitado.

2. **Custo computacional**: Redes mais profundas requerem mais cálculos, o que pode aumentar significativamente os tempos de treinamento e exigir hardware mais potente ou mais tempo para processar os dados.
3. **Dificuldades de treinamento**: Assim como outras redes profundas, as deep LSTMs podem ser difíceis de treinar devido a problemas como o desvanecimento ou explosão de gradientes, embora técnicas como gradient clipping e o uso de inicializações cuidadosas possam ajudar.

Arquitetura Típica

Na prática, uma rede com duas camadas de LSTM pode ser configurada da seguinte forma:

1. **Primeira Camada LSTM**: Recebe a entrada da sequência e passa seu estado oculto para a segunda camada. Normalmente, essa camada mantém todo o seu estado oculto (ou seja, não usa dropout ou regularização entre as células LSTM).
2. **Segunda Camada LSTM**: Recebe a saída (estado oculto) da primeira camada como sua entrada e continua o processamento. Esta camada pode, por sua vez, passar sua saída para uma camada de densidade (fully connected) para tarefas de classificação ou regressão, ou pode continuar passando para outras camadas LSTM se mais profundidade for necessária.
3. **Camada de Saída**: Após a última camada LSTM, normalmente há uma ou mais camadas densas (fully connected) que transformam a saída final da LSTM em previsões para a tarefa específica (por exemplo, categorias para classificação).

Empilhar camadas de LSTM dessa maneira permite que a rede capture nuances e relações complexas nos dados de entrada, oferecendo um poderoso mecanismo para tarefas desafiadoras de sequência temporal.

LSTM Bidirecional, ou **BiLSTM**, é uma variação da arquitetura tradicional de Long Short-Term Memory (LSTM) que pode melhorar significativamente o desempenho do modelo em tarefas que se beneficiam do conhecimento do contexto tanto anterior quanto posterior ao ponto atual na sequência de dados. A BiLSTM é especialmente útil em aplicações de processamento de linguagem natural e outras tarefas de sequência onde o contexto em ambas as direções é informativo para fazer previsões.

Como Funciona

A BiLSTM consiste em duas LSTMs separadas que são treinadas simultaneamente:

- Uma LSTM processa a sequência de dados na ordem direta (do início ao fim).
- A outra LSTM processa a sequência de dados na ordem inversa (do fim ao início).

As saídas dessas duas LSTMs são combinadas em cada etapa de tempo, geralmente por concatenação ou soma, para formar a saída final. Esta abordagem permite que o modelo capture informações tanto do passado quanto do futuro em relação ao ponto atual na sequência.

Vantagens

1. **Melhor Contextualização**: Ao acessar informações de antes e depois do ponto atual, a BiLSTM pode fazer previsões mais informadas e precisas. Isso é particularmente útil em tarefas como etiquetagem de partes da fala, reconhecimento de entidade nomeada, e tradução automática, onde o contexto completo é crucial.
2. **Maior Poder de Captura de Dependência**: A arquitetura bidirecional permite que o modelo capte dependências de longo alcance que podem ser perdidas em uma LSTM unidirecional.
3. **Aplicável a Diversos Tipos de Dados**: Embora frequentemente usada em texto, a BiLSTM também pode ser aplicada a outros tipos de dados sequenciais como séries temporais, dados de áudio, e até mesmo em imagens, quando estruturadas como sequências.

Desafios

1. **Complexidade Computacional**: Por envolver duas LSTMs, a BiLSTM geralmente exige aproximadamente o dobro dos recursos computacionais para treinamento e inferência em comparação com uma LSTM unidirecional.
2. **Gerenciamento de Dados**: A necessidade de processar a sequência em ambas as direções pode complicar o pré-processamento dos dados e o gerenciamento das entradas e saídas do modelo.
3. **Risco de Overfitting**: Com mais parâmetros para treinar, pode haver um risco aumentado de overfitting, especialmente em conjuntos de dados menores ou menos variados.

Em resumo, a LSTM bidirecional é uma ferramenta poderosa que amplia as capacidades das LSTMs tradicionais, oferecendo melhorias significativas na modelagem de sequências quando o contexto de ambos os lados é relevante para a tarefa.

GRU, sigla para **Gated Recurrent Unit**, é uma arquitetura de rede neural recorrente projetada para capturar dependências de longo prazo sem os problemas de desaparecimento do gradiente frequentemente encontrados nas RNNs convencionais. A GRU simplifica o modelo de uma LSTM ao combinar as portas de entrada e esquecimento em uma única "porta de atualização". Ela tende a ser mais fácil de modelar e pode executar tarefas em um nível comparável às LSTMs com menos parâmetros e complexidade computacional.
Uma célula GRU típica tem a seguinte estrutura:

1. **Porta de atualização (Update gate)**: Essa porta decide quanto da informação passada (do estado anterior) será mantida e quanto do novo candidato a estado será adicionado. Isso é similar à combinação da porta de esquecimento e da porta de entrada em uma LSTM.
2. **Porta de reinicialização (Reset gate)**: Decide quanto do estado anterior será esquecido. Isso permite que a GRU descarte as informações que não

são mais úteis para a previsão atual. A saída dessa porta é usada para modificar a entrada antes de criar o novo candidato a estado.

3. **Candidato a estado (Candidate state)**: Calculado usando uma combinação da entrada atual e a versão modificada do estado anterior (modificado pela porta de reinicialização). Isso determina os novos valores a serem adicionados ao estado.
4. **Estado oculto (Hidden state)**: O estado oculto para cada passo no tempo é atualizado usando uma interpolação entre o estado anterior e o novo candidato a estado, ponderada pela porta de atualização. Isso determina o grau de mudança em relação ao estado anterior.

Esse modelo simplificado, com menos portas que uma LSTM, oferece vantagens em termos de eficiência computacional, enquanto ainda mantém uma forte capacidade de modelar dependências complexas em dados sequenciais.

ConvLSTM, ou **Convolutional LSTM**, é uma variante da arquitetura LSTM tradicional, projetada especificamente para lidar melhor com dados sequenciais que têm uma estrutura espacial, como vídeos ou imagens em séries temporais. A principal inovação da ConvLSTM é a integração de operações de convolução diretamente dentro das portas da célula LSTM, permitindo que a rede capture dependências tanto temporais quanto espaciais nos dados.

Estrutura da ConvLSTM

A ConvLSTM substitui as operações de multiplicação de matrizes das LSTMs tradicionais por operações de convolução. Isso significa que as entradas, os estados ocultos e as saídas são todos tensores tridimensionais (com dimensões para as coordenadas espaciais e os canais), e as transformações aplicadas a esses dados são realizadas através de filtros convolucionais.
Aqui está como os componentes principais da LSTM são modificados na ConvLSTM:

1. **Porta de Esquecimento**: Decide quais informações do estado da célula anterior serão mantidas ou descartadas, usando operações de convolução para avaliar a importância espacial das informações.
2. **Porta de Entrada**: Determina quais novas informações serão armazenadas no estado da célula, também utilizando operações de convolução para integrar informações espaciais e temporais.
3. **Candidato a Estado da Célula**: Forma um novo candidato a estado da célula que é uma combinação das entradas atuais e do estado anterior, novamente processado por convoluções.
4. **Porta de Saída**: Usa convoluções para determinar quais partes do estado da célula devem ser passadas para o estado oculto, que serve como saída da célula para o próximo passo de tempo.

Aplicações da ConvLSTM

As ConvLSTMs são particularmente úteis em áreas como:

- **Previsão de séries temporais espaciais**: Por exemplo, previsão de precipitação usando mapas de radar ao longo do tempo, onde é importante entender não apenas quando e onde vai chover, mas também os padrões de movimento das nuvens.
- **Análise de vídeo**: Detecção e reconhecimento de atividades em vídeos, onde a rede precisa entender e prever movimentos baseados tanto no aspecto temporal quanto no conteúdo espacial dos frames.
- **Processamento de dados de sensores espaciais e temporais**: Por exemplo, dados de fluxos de tráfego ou de multidões, onde é crucial capturar padrões espaciais e temporais simultaneamente.

Vantagens

A integração de convoluções permite que a ConvLSTM capture eficazmente as correlações espaciais nos dados, o que é uma grande vantagem para dados visuais e outras formas de dados multidimensionais. Além disso, mantém a capacidade das LSTMs de lidar com dependências de longo prazo, tornando-a uma ferramenta poderosa para análise de dados sequenciais complexos que também possuem uma dimensão espacial significativa.

Desafios

Apesar de suas vantagens, a ConvLSTM pode ser mais computacionalmente intensiva devido à complexidade adicional das operações de convolução. Isso pode torná-la mais lenta para treinar e exigir mais recursos computacionais, especialmente com grandes volumes de dados ou em tarefas que exigem muitas camadas de ConvLSTM.

Resultados obtidos

Foram realizados os seguintes experimentos:

1. Use diferentes unidades internas de LSTM: 64, 128, 256, 512.
2. Substitua a LSTM por GRU. Quais foram as vantagens ou desvantagens? Use diferentes unidades internas: 64, 128, 256, 512. (Dica: Use https://pytorch.org/docs/stable/generated/torch.nn.GRU.html)
3. Use uma LSTM profunda. Duas camadas de LSTM empilhadas: A saída da primeira LSTM vai para a entrada da segunda LSTM. Use diferentes unidades internas: 64, 128, 256, 512 (Dica: Use num_layers=2 na definição da LSTM).
4. Use uma ConvLSTM. Use diferentes unidades internas: 64, 128, 256, 512 (Dica: Use como em https://github.com/ndrplz/ConvLSTM_pytorch/tree/master)

RNN type	hidden_size	Parameter count	Training time (s)	Overall Accuracy	Avg. F1-Score
LSTM	64	148,683	113,1	0,64	0,56
LSTM	128	330,123	111,45	0,76	0,75
LSTM	256	791,307	139,71	0,74	0,73
LSTM	512	2106,891	158,76	0,76	0,75
GRU	64	111,691	103,17	0,77	0,76
GRU	128	247,947	88,28	0,77	0,77
GRU	256	594,187	50,26	0,78	0,77
GRU	512	1581,579	29,94	0,76	0,75
Deep LSTM (num_layers=2)	64	181,963	112,87	0,65	0,63
Deep LSTM (num_layers=2)	128	462,219	117,41	0,71	0,7
Deep LSTM (num_layers=2)	256	1317,643	114,97	0,77	0,76
Deep LSTM (num_layers=2)	512	4208,139	100,33	0,77	0,76
ConvLSTM	64	1328,075	1778,58	0,76	0,74
ConvLSTM	128	2951,051	1292,99	0,8	0,78
ConvLSTM	256	7081,739	4197,88	0,8	0,79
ConvLSTM	512	18882,059	1346,28	0,79	0,78

A tabela apresentada mostra o desempenho de diferentes tipos de redes neurais recorrentes (RNNs) com variações em tamanho da camada oculta ("hidden_size"), quantidade de parâmetros, tempo de treinamento, acurácia geral e média de pontuação F1 para uma tarefa de classificação. Aqui estão algumas observações e interpretações dos dados:

Comparação por Tipo de RNN

1. **LSTM**: A LSTM com tamanho de camada oculta de 128 e 512 apresentou melhor desempenho em termos de acurácia e pontuação F1, ambos em 0,76 e 0,75, respectivamente. Isso indica que a LSTM beneficia de um tamanho de camada oculta maior, mas depois de certo ponto (256 para 512), o aumento não traz melhorias significativas em acurácia.
2. **GRU**: A GRU com tamanho de camada oculta de 256 teve o melhor desempenho em acurácia (0,78) e F1-score (0,77) comparado às outras configurações de GRU. As GRUs mostraram-se consistentemente mais rápidas e com menos parâmetros do que as LSTMs correspondentes, sugerindo que GRUs são mais eficientes para tarefas semelhantes.
3. **Deep LSTM**: A LSTM de duas camadas com 256 e 512 de tamanho de camada oculta teve desempenhos semelhantes em acurácia e F1-score, mas com uma diminuição significativa no tempo de treinamento para o maior tamanho de camada oculta. Isso sugere uma eficiência computacional melhor à medida que o tamanho da camada aumenta.
4. **ConvLSTM**: A ConvLSTM com tamanho de camada oculta de 256 mostrou a melhor acurácia (0,8) e F1-score (0,79) entre todas as configurações testadas. No entanto, a quantidade de parâmetros e o tempo de treinamento

são significativamente maiores, especialmente para o tamanho de camada oculta de 256, indicando um alto custo computacional.

Considerações Gerais

- **Tamanho da Camada Oculta**: De modo geral, um aumento no tamanho da camada oculta leva a um maior número de parâmetros e, frequentemente, a melhorias na acurácia e no F1-score. No entanto, isso também pode resultar em tempos de treinamento mais longos e risco aumentado de overfitting.
- **Eficiência**: GRUs parecem ser mais eficientes em termos de tempo de treinamento e quantidade de parâmetros comparadas às LSTMs e ConvLSTMs, mantendo uma performance competitiva em acurácia e F1-score.
- **Complexidade vs. Desempenho**: ConvLSTMs e Deep LSTMs tendem a ter tempos de treinamento mais longos e mais parâmetros, mas podem alcançar maior acurácia em tarefas que requerem captura de dependências espaciais e temporais complexas.

Conclusão

A escolha entre esses diferentes tipos de RNN depende do problema específico, da disponibilidade de recursos computacionais, do tempo disponível para treinamento e da necessidade de capturar dependências complexas nos dados. Cada configuração oferece um equilíbrio entre eficiência e capacidade de modelagem.