Generarea de secvențe folosind rețele neuronale recurente

Rețelele neuronale recurente (RNN) sunt special proiectate pentru prelucrarea dateor care formează secvențe. Diferența esențială dintre rețelele de acest tip și rețelele neuronale clasice o constituie straturile recurente, unde legăturile dintre neuroni sunt ciclice. RNN se utilizează în special pentru prelucrarea de secvențe de text și imagini, unde fiecare element al secvenței de cuvinte/caractere/imagini poate depinde de contextul creat de elementele anterioare ale secvenței.

Principiul de funcționare al RNN se bazează pe faptul că stările straturilor recurente se actualizează după fiecare element din secvență furnizat la intrarea rețelei. Astfel, presupunem că pentru o secvență de intrare x_0, x_1, ..., x_t, ... , se generează secvența de ieșire o_0, o_1, ... o_t, ... Starea stratului ascuns, s, se actualizează pentru fiecare pereche (x, o), astfel încât starea s_t de la momentul t depinde de stările s_{t-1}, s_{t-2}, ... de la momentele t_{-1}, t_{-2}, ... Așadar, valoarea de ieșire o_t depinde prelucările făcute în vederea generării valorilor o_{t-1}, o_{t-2}, ... Ăn acest fel, la prelucrarea elementelor secvențelor se ține cont și de *contextul* creat de elementele anterioare ale secvenței. Spre exemplu, dacă se face prelucrarea unei secvențe de cuvinte, un anumit cuvânt dintr-o frază depinde de contextul creat de cuvintele anterioare din acea frază. Fraza însăși depinde textul anterior. Pentru se conforma acestui principiu, starea rețelei RNN depinde de o parte sau de toate elementele secvenței, nu doar de cel din momentul actual, cum este cazul rețelelor non-recurente.

Structura unui strat recurent este ilustrată în Fig.1. La momentul t, pentru elementul x_t al secvenței, starea s_t depinde atât de x_t, cât și de starea s_{t-1}, cea determinată pentru elementul anterior x_{t-1} al secvenței. U și V sunt ponderile straturilor de intrare și ieșire, similare celor dintr-o rețea neuronală clasică, în timp de W sunt ponderile ce asigură dependența între stările s de la momente diferite de timp.

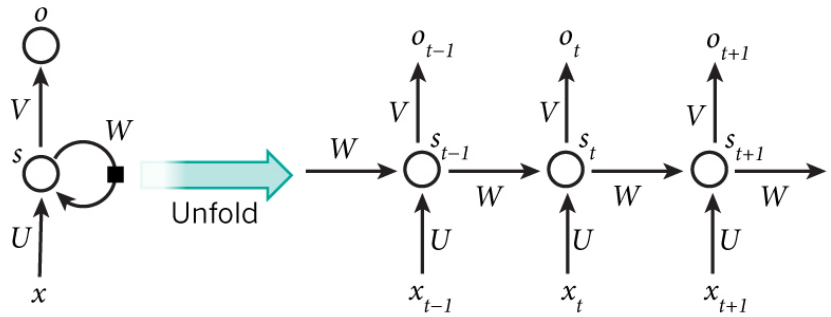

Fig. 1. Structura unei rețele neuronale recurente

O extensie a RNN sunt rețelele LSTM (*Long Short-Term Memory*), în cadrul cărora straturile recurente au o serie de componente suplimentare, numite *porți.* Acestea au rolul de a ameliora

problemele cauzate de gradienții cu valori prea mici sau prea mari (*vanishing gradients, exploding gradients*, vezi cursurile / documentația pentru detalii suplimentare).

Dorim să proiectăm o rețea recurentă pentru generarea de secvențe de text. Datele de antrenare provin dintr-un text oarecare, ideea fiind ca rețeaua să învețe limbajul folosit în text și să poată genera propriile secvențe folosind același limbaj.

Textul de antrenare este următorul:

On a mango tree in a jungle, there lived many birds. They were happy in their small nests. Before the onset of the rainy season, all the animals of the jungle repaired their homes. The birds also made their homes more secure. Many birds brought twigs and leaves and others wove their nests. We should also store some food for our children, chirped one of the birds. And they collected food, until they had enough to see them through the rainy season. They kept themselves busy preparing for the tough times. Soon the rains came. It was followed by thunder and lighting. All the animals and birds stayed in their homes. It continued raining for many days. One day, a monkey went in the rain came into the forest. He sat on a branch, shivering with cold, Water dripping from its body. The poor monkey tried his best to get shelter, but in vain. The leaves were not enough to save him from the rains. It is so cold, said the monkey. The birds were watching all this. They felt sorry for the monkey but there was little they could do for him. One of them said, brother, Our small nests are not enough to give you shelter. Another bird said, all of us prepared for the rainy season. If you had, you would not be in this situation. How dare you tell me what to do, said the monkey, growling at the bird. The monkey angrily pounced on the birds nest, tore it and threw it on the ground. The bird and her chicks were helpless. The poor bird thought, fools never value good advice. It is better not to advise them.

Principul de antrenare a rețelei este ilustrat în Figura 2:

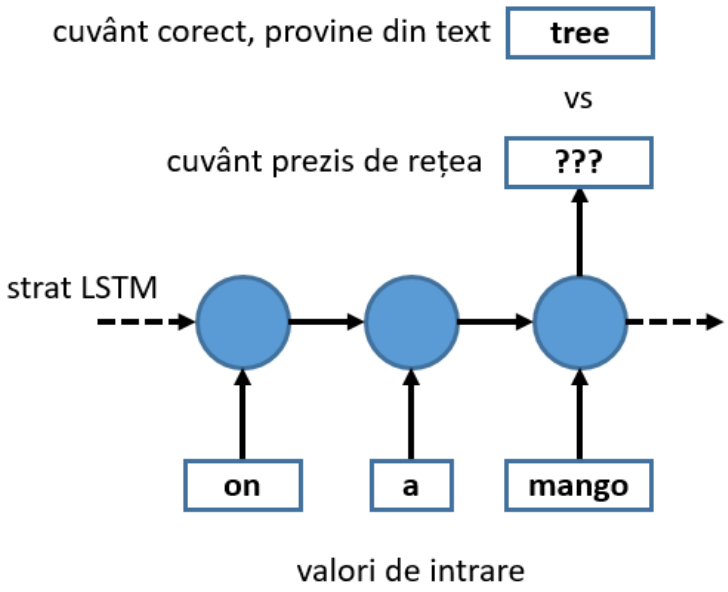

Fig. 2. Funcționarea rețelei LSTM în faza de antrenare. La intrare aceasta primește o secvență de trei cuvinte, iar la ieșire se compară cuvântul generat de rețea cu cel care urmează, preluat din textul folosit pentru antrenare.

Pentru antrenare trebuie generat **vocabularul** textului de antrenare - mulțimea cuvintelor/simbolurilor unice din text. Pentru textul de mai sus, vocabularul arată astfel:

```
[',', '.', 'a', 'advice', 'advise', 'all', 'also', 'and',
'angrily', 'animals', 'another', 'are', 'at', 'be', 'before', 'best',
'better', 'bird', 'birds', 'body', 'branch', 'brother', 'brought',
'busy', 'but', 'by', 'came', 'chicks', 'children', 'chirped', 'cold',
'collected', 'continued', 'could', 'dare', 'day', 'days', 'do',
'dripping', 'enough', 'felt', 'followed', 'food', 'fools', 'for',
'forest', 'from', 'get', 'give', 'good', 'ground', 'growling', 'had',
'happy', 'he', 'helpless', 'her', 'him', 'his', 'homes', 'how', 'if',
'in', 'into', 'is', 'it', 'its', 'jungle', 'kept', 'leaves',
'lighting', 'little', 'lived', 'made', 'mango', 'many', 'me',
'monkey', 'more', 'nest', 'nests', 'never', 'not', 'of', 'on', 'one',
'onset', 'others', 'our', 'poor', 'pounced', 'prepared', 'preparing',
'rain', 'raining', 'rains', 'rainy', 'repaired', 'said', 'sat',
'save', 'season', 'secure', 'see', 'shelter', 'shivering', 'should',
'situation', 'small', 'so', 'some', 'soon', 'sorry', 'stayed',
'store', 'tell', 'the', 'their', 'them', 'themselves', 'there',
'they', 'this', 'thought', 'threw', 'through', 'thunder', 'times',
```

```
'to', 'tore', 'tough', 'tree', 'tried', 'twigs', 'until', 'us',
'vain', 'value', 'was', 'watching', 'water', 'we', 'went', 'were',
'what', 'with', 'would', 'wove', 'you']
```

Vocabularul se poate genera folosind o listă / array / dicționar etc. Fiecărui cuvânt unic îi va corespunde un index numeric (numit și etichetă, *label*). Rețeaua neuronală prelucrează cuvintele folosind label-urile lor (primește la intrare o secvență de label-uri, la ieșire furnizează un label sau o altă secvență de label-uri).

Pentru detalii și exemple privind o astfel de rețea, consultați ultima parte din *Cursul 7*.

Implementarea rețelei descrise mai sus arată astfel (după codul sursă de mai jos urmează și explicațiile aferente):

```
textFile = open("rnn_text.txt")
text = textFile.read()
#separate punctuation by spaces
punctuation = [',', '.', ':', ';', '?', '!', '"', "'"]
tempCharList = [' '+c if c in punctuation else c for c in text]
text = ''.join(tempCharList)
text = text.lower()

#build vocabulary
words = text.split()
vocabulary = list(set(words))
vocabulary.sort()

#labels of words from training text:
wordLabels = [vocabulary.index(w) for w in words]

#build training data
sequenceLength = 3
noSequences = 100
#random indices from training text:
indices = random.sample(range(len(words)-sequenceLength-1), noSequences)

inputs = [wordLabels[i : i+sequenceLength] for i in indices]
targets = [wordLabels[i+sequenceLength] for i in indices]

inputs = tr.tensor(inputs, dtype=tr.float)
targets = tr.tensor(targets, dtype=tr.long)

class SimpleRNN(nn.Module):
    def __init__(self, inputSize, outputSize, lstmLayerSize, noLSTMLayers):
        super(SimpleRNN, self).__init__()
        self.inputSize = inputSize
        self.lstmLayerSize = lstmLayerSize
        self.outputSize = outputSize
        self.noLSTMLayers = noLSTMLayers

        self.lstmLayer = nn.LSTM(self.inputSize, self.lstmLayerSize,
self.noLSTMLayers)
        self.outLayer = nn.Linear(self.lstmLayerSize, self.outputSize)
```

```
    def forward(self, input):
        input = input.view(-1, 1, 1)
        lstmOut, hidden = self.lstmLayer(input)
        outLayerInput = lstmOut[-1, 0, :]
        predictedOut = self.outLayer(outLayerInput)
        return predictedOut

    def train(self, inputs, targets):
        noEpochs = 100
        learnRate = 0.001
        optimizer = tr.optim.Adam(self.parameters(), learnRate)
        lossFunc = nn.CrossEntropyLoss()

        for epoch in range(noEpochs):
            for input, target in zip(inputs, targets):
                optimizer.zero_grad()
                predicted = self.forward(input)
                loss = lossFunc(predicted.unsqueeze(0), target.unsqueeze(0))
                loss.backward()
                optimizer.step()

            print('Epoch', epoch, 'loss', loss.item())

myRNN = SimpleRNN(1, len(vocabulary), 16, 1)
myRNN.train(inputs, targets)

def generateText(desiredTextLength):
    while True:
        sentence = input('Introduceti %s cuvinte sau _quit: ' % sequenceLength)
        sentence.strip()
        if sentence == "_quit":
            break
        words = sentence.split()
        if len(words) != sequenceLength:
            continue

        try:
            inputLabels = [vocabulary.index(w) for w in words]
        except:
            print('Cuvintele introduse trebuie sa faca parte din vocabular.')
            continue

        sentence += ' '
        for i in range(desiredTextLength - sequenceLength):
            rnnInput = tr.tensor(inputLabels, dtype=tr.float)
            rnnOut = myRNN(rnnInput)
            outputLabel = rnnOut.argmax().item()
            outputWord = vocabulary[outputLabel]
            sentence += outputWord
            sentence += ' '
        print(sentence)
```

Explicații:

```
textFile = open("rnn_text.txt")
text = textFile.read()
#separate punctuation by spaces
punctuation = [',', '.', ':', ';', '?', '!', '"', "'"]
tempCharList = [' '+c if c in punctuation else c for c in text]
text = ''.join(tempCharList)
text = text.lower()

#build vocabulary
words = text.split()
vocabulary = list(set(words))
vocabulary.sort()

#labels of words from training text:
wordLabels = [vocabulary.index(w) for w in words]
```

Se citește fișierul care conține textul de antrenare. Se separă cuvintele și semnele de punctuație și se generează un vocabular sub forma unei liste. Fiecare cuvânt sau semn de punctuație din text va avea asociat un index în vocabular (poziția cuvântului în vocabular). Termenul utilizat pentru acești indecși este *label*. De exemplu cuvântul *'advice'* se află în vocabular la indexul 3, prin urmare label-ul său este 3. Rețeaua neuronală nu prelucrează]n mod direct șiruri de caractere, ci secvențe formate din label-urile cuvintelor corespunzătoare.

```
#labels of words from training text:
wordLabels = [vocabulary.index(w) for w in words]

#build training data
sequenceLength = 3
noSequences = 100
#random indices from training text:
indices = random.sample(range(len(words)-sequenceLength-1), noSequences)

inputs = [wordLabels[i : i+sequenceLength] for i in indices]
targets = [wordLabels[i+sequenceLength] for i in indices]

inputs = tr.tensor(inputs, dtype=tr.float)
targets = tr.tensor(targets, dtype=tr.long)
```

Datele de antrenare constau în perechi formate din secvențe de 3 cuvinte – al patrulea cuvânt care urmează. De exemplu, în textul de antrenare, secvenței de 3 cuvinte din inputs[i] îi urmează cuvântul din targets[i]. Se generează 100 de astfel de secvențe. Rețeaua se va antrena folosind aceste secvențe, ideea fiind ca aceasta să asocieze oricare trei cuvinte consecutive cu al patrulea imediat următor.

```
class SimpleRNN(nn.Module):
    def __init__(self, inputSize, outputSize, lstmLayerSize, noLSTMLayers):
        super(SimpleRNN, self).__init__()
        self.inputSize = inputSize
        self.lstmLayerSize = lstmLayerSize
        self.outputSize = outputSize
        self.noLSTMLayers = noLSTMLayers

        self.lstmLayer = nn.LSTM(self.inputSize, self.lstmLayerSize,
self.noLSTMLayers)
        self.outLayer = nn.Linear(self.lstmLayerSize, self.outputSize)
```

Rețeaua neuronală este implementată prin clasa SimpleRNN. Semnificația diverșilor parametri este următoarea:

inputSize = dimensiunea unui element din secvență. În cazul de față, un element este un simplu număr întreg (label-ul unui cuvânt), așadar în exemplu nostru acest parametru va avea valoarea 1. În caz general, un element al unei secvențe poate avea mai multe caracteristici (de ex. pixelii unei imagini dintr-o secvență video).

lstmLayerSize și **noLSTMLayers** – caracteristici ale straturilor recurente din rețeaua neuronală. Pentru detalii, consultați cursurile / documentația aferentă.

outputSize – dimensiunea output-ului rețelei neuronale este dimensiunea vocabularului. Pentru fiecare cuvânt/semn de punctuație, rețeaua neuronală generează o valoare reală care indică probabilitatea ca în secvență să urmeze acel cuvânt.

lstmLayer – obiect are implementează unul sau mai multe straturi de tip *Long Short-Term Memory* (i.e. straturile recurente ale rețelei).

outLayer – stratul de ieșire al rețelei, este un strat “clasic”, similar celor din rețelele neuronale non-recurente.

```
    def forward(self, input):
        input = input.view(-1, 1, 1)
        lstmOut, hidden = self.lstmLayer(input)
        outLayerInput = lstmOut[-1, 0, :]
        predictedOut = self.outLayer(outLayerInput)
        return predictedOut
```

Metoda primește ca parametru o secvență de trei cuvinte și returnează un vector de dimensiunea vocabularului – câte o valoare reală pentru fiecare cuvânt posibil. Astfel, pentru fiecare secvență de trei cuvinte, rețeaua îl prezice pe al patrulea, ca fiind cel de pe poziția ce corespunde valorii maxime a vectorului returnat de această metodă.

```
def train(self, inputs, targets):
    noEpochs = 100
    learnRate = 0.001
    optimizer = tr.optim.Adam(self.parameters(), learnRate)
    lossFunc = nn.CrossEntropyLoss()

    for epoch in range(noEpochs):
        for input, target in zip(inputs, targets):
            optimizer.zero_grad()
            predicted = self.forward(input)
            loss = lossFunc(predicted.unsqueeze(0), target.unsqueeze(0))
            loss.backward()
            optimizer.step()

        print('Epoch', epoch, 'loss', loss.item())
```

Antrenarea este foarte asemănătoare cu cea a rețelelor studiate la laboratoarele precedente. În cadrul unei iterații, rețeaua primește o secvență de trei cuvinte consecutive la intrare (*input*), și pe al patrulea care urmează, la ieșire (*target*). Al patrulea cuvând prezis de rețea se regăsește în *predicted* , iar diferența dintre cuvântul prezis de rețea (predicted) și cel corect (target) se determină folosind funcția lossFunc. Scopul antrenării este ajustarea ponderilor rețelei astfel încât să se minimizeze această diferență. Minimizarea se realizează printr-o metodă de optimizare, implementată de către obiectul *optimizer*. Secvențele utilizate pentru antrenare sunt prelucrate de către rețea de multiple ori. Ori de câte ori în cadrul procesului de antrenare s-a parcurs întregul set de date, se spune ca mai trecut o *epocă*.

```
def generateText(desiredTextLength):
        . . .
```

Funcția generează o secvență de text folosind rețeaua neuronală descrisă anterior. Parametrul metodei este numărul de cuvinte al secvenței dorite. Pentru detalii privind modul de funcționare și exemple de execuție, consultați materialele de la cursul corespunzător.

Sarcini

1. Determinați acuratețea rețelei la sfârșitul fiecărei epoci de antrenare (asemănător cu modul de calcul al acurateții de la laboratoarele anterioare)

2. Pentru textul furnizat ca exemplu și/sau propriile texte, ajustați parametrii rețelei astfel încât să obțineți o acuratețe cât mai mare (preferabil > 95%). Cu cât acuratețea de la sfârșitul antrenării este mai mare, cu atât textele generate de rețea ar trebui să fie mai corecte și mai coerente. De exemplu, se poate ajusta:

- numărul de epoci
- numărul datelor de antrenare (numărul secvențelor generate pornind de la textul de antrenare)
- dimensiunea secvențelor de cuvinte de la intrare (în exemplul nostru lungimea unei astfel de secvențe este 3)
- numărul de straturi LSTM și dimensiunea lor
- rata de învățare
- metoda de optimizare (clasa obiectului *optimizer*). Printre cei mai frecvent utilizați algoritmi menționăm SGD, Adam, RMSprop. Mai multe detalii se găsesc la următorul link: https://pytorch.org/docs/stable/optim.html

3. Antrenați și testați rețeaua folosind și alte secvențe de text (nu foarte lungi, cât mai simple, cu un vocabular de dimensiuni reduse)

***Observație*:** Exemplul prezentat în cadrul laboratorului este simplu, iar textul folosit pentru antrenare este de mici dimensiuni, pentru ca antrenarea rețelei să se poată realiza într-un timp decent. Din acest motiv, chiar și pentru o rețea de mare acuratețe, este puțin probabil ca secvențele de text generate să fie foarte coerente și corecte dpdv gramatical. Pentru a genera text corect și coerent în limba engleză, este nevoie de date de antrenare mult mai mari, și de o rețea cu mult mai mulți parametri, ceea ce înseamnă timpi de antrenare de ordinul orelor, chiar zilelor. Pornind de la exemplul furnizat în laborator, se pot crea modele generative mult mai complexe și convingătoare. La următorul link se prezintă o aplicație “din lumea reală” cu rețele LSTM:
https://arstechnica.com/gaming/2016/06/an-ai-wrote-this-movie-and-its-strangely-moving/