Sieci neuronowe

Krzysztof Halawa

Obliczenia neuronowe - sieci rekurencyjne (m.in. LSTM, GRU) i transformery

2024

Treść prezentacji

Sieci rekurencyjne (ang. Recurrent Neural Networks)

LSTM (ang. Long short-term memory)

GRU (ang. Gated Reccurent Unit)

Tokenizacja, embedding, NLP

RNN i generowanie tekstów w stylu słynnego pisarza

Transformery

Berty, GPT, itp.

Polecana literatura do samodzielnej nauki

[AGeron] Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Wyd. II, Helion, 2020

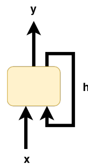
[FChollet] Francois Chollet, Deep Learning. Praca z językiem Python i biblioteką Keras, Helion, 2019

[DFoster] David Foster, Deep learning i modelowanie generatywne. Jak nauczyć komputer malowania, pisania, komponowania i grania, Helion, 2021

[Rotman] Denis Rothman Transformers for Natural Language Processing and Computer Vision - Third Edition: Explore Generative AI and Large Language Models with Hugging Face, ChatGPT, GPT-4V, and DALL-E 3 3rd ed., Pact Publishing Ltd. 2024 - transformery, berty, gpt, DALL-E itp.

[Transformer] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, Attention Is All You Need v7, https://arxiv.org/abs/1706.03762

Sieci rekurencyjne RNN (ang. Recurrent Neural Networks)



x,y,h są wektorami !

$$y = f\left(W_x^T x + W_h^T h + b\right),$$

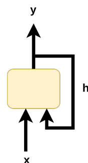
gdzie **x,y,h,b** są wektorami, W_x, W_h są macierzami wag, f oznacza funkcję aktywacji.

Sieci rekurencyjne

Występują sprzężenia zwrotne (więc nie są sieciami jednokierunkowymi)
Częstym problemem są niestabilne gradienty (zanikające lub eksplodujące)
RNN są stosowane do szeregów czasowych, przetwarzania tekstów, mowy, audio, tłumaczenia na inny język, tworzenia streszczeń, robienia chatbotów, generowania tekstów literackich w stylu słynnego pisarza, automatycznego wykonywania opisów...

Sieci rekurencyjne są szczególnie przydatne do przetwarzania sekwencji informacji. Sekwencje te mogą być różnej długości. Podczas czytania kolejnych wyrazów tekstu mózg ludzki przetwarza informacje w sposób przyrostowy dodając kolejne informacje do tych które, zostały przeczytane wcześniej.

Sieci rekurencyjne SimpleRNN



W SimpleRNN **h=y**

$$y = f\left(W_x^T x + W_h^T h + b\right),$$

gdzie **x,y,h,b** są wektorami, W_x, W_h są macierzami wag, f oznacza funkcję aktywacji.

SimpleRNN

SimpleRNN nie są praktycznie stosowane. Używane są sieci rekurencyjne takie jak LSTM i GRU, w których $h \neq y$

SimpleRNN

Z powodu zaniku gradientu SimpleRNN bardzo kiepsko uczy się dłuższych zależności.

SimpleRNN

```
model = Sequential()
model.add(SimpleRNN(units=32, input_shape=(1,step), activation="relu"))
model.add(Dense(8, activation="relu"))
model.add(Dense(1))
model.compile(loss='mean_squared_error', optimizer='rmsprop')
model.summary()
```

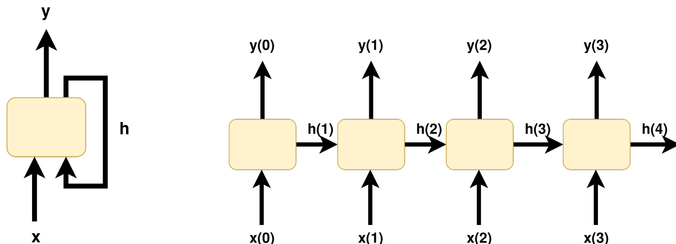
Dygresja

Do krótkich szeregów czasowych czasami są też stosowane wielowarstwowe perceptrony (np. modele NNFIR, NNARX), a do dłuższych bywają używane sieci konwolucyjne ze spektogramami https://en.wikipedia.org/wiki/Spectrogram, które są przetwarzane jak obrazy 2D.

Ponadto opracowano specjalne modyfikacje architektur sieci konwolucyjnych dla przetwarzania "surowych" sygnałów audio np. WaveNet https://arxiv.org/pdf/1609.03499.pdf (na płycie CD próbkowanie 44,1kHz, więc w ciągu jednej sekundy dla stereo jest 88200 próbek)

Sieci rekurencyjne RNN (ang. Recurrent Neural Networks)

Rozwijanie sieci w czasie - w kolejnych chwilach zwanych taktami lub ramkami(ang. time step, frame)



Wsteczna propagacja w czasie BPTT (ang. backpropagation through time)

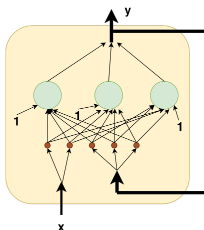
Po rozwinięciu w czasie widać, że można zastosować algorytm wstecznej propagacji błędów podobnie jak w jednokierunkowych sieciach składających się z szeregowo połączonych warstw.

Keras - Liczba neuronów i SimpleRNN

Keras - SimpleRNN

W SimpleRNN **h=y**

Załóżmy, że wektory **x** mają dwa elementy i są trzy neurony (liczba elementów **y** w SimpleRNN jest równa liczbie neuronów).



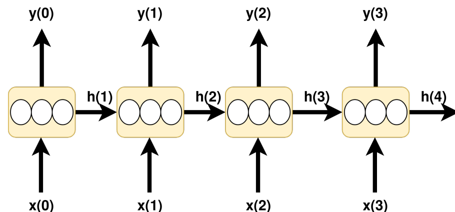
Neurony są zaznaczone na zielono, a ich wejścia na brązowo.

Wartość wyjścia neuronu (McCullocha-Pittsa) jest równa $f(\text{sum})$, gdzie f oznacza f. aktywacji, sum jest sumą iloczynów wartości wejść neuronu i jego wag.
Domyślną funkcją aktywacji dla SimpleRNN jest tanh. (RELU - brak ograniczenia wartości funkcji aktywacji może spowodować eksplozję gradientów w sieciach rekurencyjnych).

```
model.add(SimpleRNN(liczba_neuronow))  #Dodanie do modelu warstwy SimpleRNN
```

Keras - Liczba neuronów

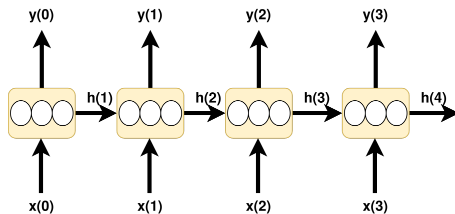
W bardziej skomplikowanych sieciach rekurencyjnych $h \neq y$



Keras - sygnały wyjściowe z warstwy rekurencyjnej

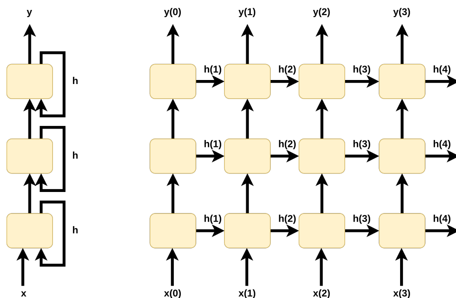
Jakie wartości otrzymujemy z warstwy rekurencyjnej?

Dla return_sequence=False ostatni **y**.



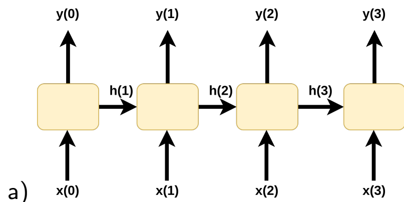
Jeżeli wszystkie poprzednie wyniki $y(0), y(1), y(2)$ oprócz najnowszego $y(3)$ są ignorowane (w Kerasie domyślnie dla warstw rekurencyjnych argument return_sequence=False), to otrzymuje się sieć sekwencyjno wektorową (ang. sequence-to-vector-network). Następnie można utworzyć drugą sieć rekurencyjną generującą sekwencję z wektora. Poprzez połączenie sieci sekwencyjno-wektorowej (zwanej koderem) i wektorowo-sekwencyjnej (dekoder), można uzyskać lepszy wynik np. tłumaczenia niż w przypadku zwykłej sieci rekurencyjnej, ponieważ ostatnie słowa w zdaniu mogą istotnie zmieniać treść, więc przed rozpoczęciem tłumaczenia lepiej dysponować informacjami o wszystkich wyrazach. UWAGA: obecnie istnieją znacznie lepsze architektury do tłumaczenia na inne języki.

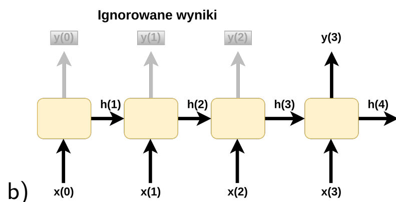
Keras - kilka warstw rekurencyjnych

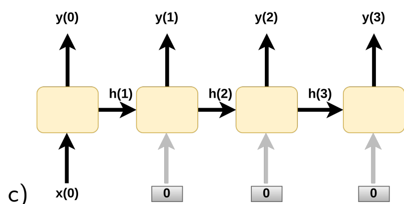


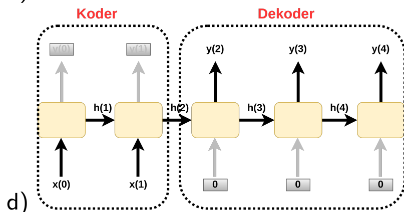

```
model.add(LSTM(liczba_neuronow), return_sequence=True)
model.add(LSTM(liczba_neuronow), return_sequence=True)
...
```

RNN return_sequence=?
a) sequence to sequence, b) sequence to vector, c)vector to sequence, d) coder-decoder









Sieć sekwencyjno-wektorowa - na wejścia słowa recenzji i prognozowany jest pozytywność recenzji.
Sieć wektorowo sekwencyjna - na wejścia np. obraz a na wyjściu generowany jest opis.
Koder-dekoder - np. tłumaczenie zdań na inny język (obecnie lepsze wyniki mają transformery).
Sieć sekwencyjna - prognozowanie szeregów czasowych np. akcji - podawane są notowania z N dni i prognozy są przesunięte o 1 naprzód (od N-1 do jutra). Można też sekwencyjno-wektorowa.

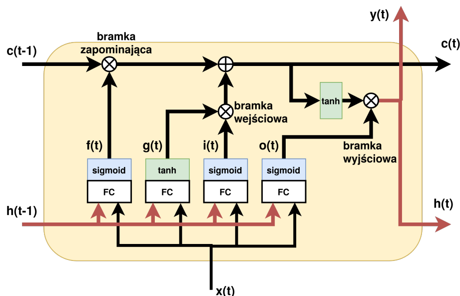
Long short-term memory LSTM

Sieć rekurencyjną można po rozwinięciu przedstawić jako sekwencję warstw, przez które przepływają informacje. Takie rozwinięcie może być uczone za pomocą algorytmów gradientowych podobnie jak wielowarstwowe sieci jednokierunkowe.
W sieciach składających się z wielu warstw bardzo często występuje problem zanikającego gradientu. Sieć LSTM została zaprojektowana w celu redukcji tego problemu, co osiągnięto głównie dzięki linii stanu **c** (redukuje ona problem zanikającego gradientu w sposób podobny jak połączenia szczątkowe w sieciach konwolucyjnych). Prymitywne

SimpleRNN nie są stosowane w praktyce z wyjątkiem bardzo trywialnych zastosowań.

LSTM

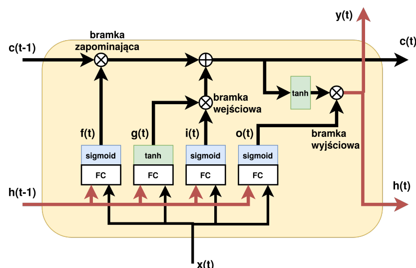
LSTM zostały zaproponowane w 1997. Obecnie dwie najbardziej popularne architektury sieci rekurencyjnych to LSTM i GRU (ang. Gated Reccurent Unit).



c, h, x, y, f, g, i, o to wektory !
c(t) - informacje długotrwałe,
h(t) - informacje krótkotrwałe,
x(t) - wejście,
y(t) - wyjście,
$\otimes$ - iloczyn Hadamarda wektorów (mnożenie element po elemencie dwóch wektorów)
$\oplus$ - sumator.

```
model.add(LSTM(liczba_neuronow))  #Dodanie warstwy LSTM
```

Long short-term memory LSTM



$$
\begin{aligned}
\mathrm{f}(t) &= \sigma\left(\mathrm{W}_{xf}^{T}\mathrm{x}(t) + \mathrm{W}_{hf}^{T}\mathrm{h}(t-1) + \mathrm{b}_f\right), \\
\mathrm{i}(t) &= \sigma\left(\mathrm{W}_{xi}^{T}\mathrm{x}(t) + \mathrm{W}_{hi}^{T}\mathrm{h}(t-1) + \mathrm{b}_i\right), \\
\mathrm{o}(t) &= \sigma\left(\mathrm{W}_{xo}^{T}\mathrm{x}(t) + \mathrm{W}_{ho}^{T}\mathrm{h}(t-1) + \mathrm{b}_o\right), \\
\mathrm{g}(t) &= \tanh\left(\mathrm{W}_{\mathrm{xg}}^{\mathrm{T}}\mathrm{x}(\mathrm{t}) + \mathrm{W}_{\mathrm{hg}}^{\mathrm{T}}\mathrm{h}(\mathrm{t}-1) + \mathrm{b}_{\mathrm{g}}\right), \\
\mathrm{c}(t) &= \mathrm{f}(t) \otimes \mathrm{c}(t-1) + \mathrm{i}(t) \otimes \mathrm{g}(t), \\
\mathrm{y}(t) &= \mathrm{h}(t) = \mathrm{o}(t) \otimes \tanh\left(\mathrm{c}(\mathrm{t})\right),
\end{aligned}
$$

gdzie σ oznacza logistyczna funkcję aktywacji (keras.activations.sigmoid), która może przyjmować wartości od 0 do 1, $\otimes$ oznacza mnożenie element po elemencie, t - czas.

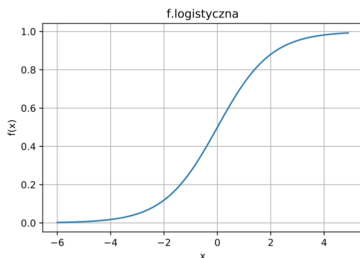
Przypomnienie f. aktywacji

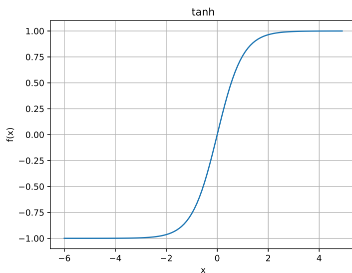
funkcja logistyczna - keras.activations.sigmoid:

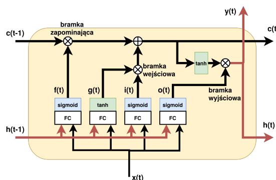
$$\sigma(u) = \frac{1}{1 + e^{-u}}, \tag{1}$$

tangens hiperboliczny - keras.activations.tanh:

$$tanh(u) = \frac{e^{u} - e^{-u}}{e^{u} + e^{-u}}, \tag{2}$$







Bramka zapominająca (indeks f - forget) określa, które składniki pamięci długotrwałej mają zostać usunięte. Jest ona sterowana przez wektor

$$f(t) = \sigma\left(W_{xf}^T x(t) + W_{hf}^T h(t-1) + b_f\right).$$

Niewielkie wartości elementów wektora **f(t)** powodują tłumienie. Elementy **f(t)** równe 1, powodują przekazanie informacji bez tłumienia.

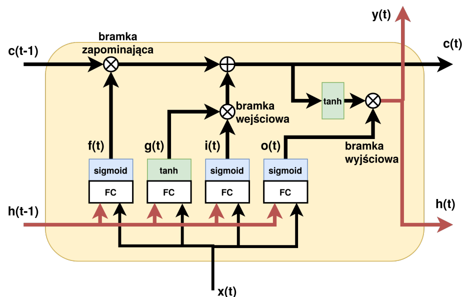
Bramka wejściowa (indeks i - input) określa, które składniki wektor $\mathbf{g}(t)$ będą dodane do pamięci długotrwałej powinny zostać usunięte. Jest ona sterowana przez wektor

$$i(t) = \sigma\left(W_{xi}^T x(t) + W_{hi}^T h(t-1) + b_i\right).$$

Bramka wyjściowa (indeks o - output) określa, co ma być wysłane na $\mathbf{y}(t)$ oraz $\mathbf{h}(t)$

$$o(t) = \sigma\left(W_{xo}^T x(t) + W_{ho}^T h(t-1) + b_o\right).$$

Long short-term memory LSTM



Do sterowania bramek wykorzystywane są wartości z przedziału [0,1] - dlatego logistyczna funkcja aktywacji.

Long short-term memory LSTM

Szersze spojrzenie na interpretację poszczególnych składników LSTM

Cytat autora biblioteki Kerasa z książki Francois Chollet, „Deep Learning Praca z językiem Python i biblioteką Keras” Helion, 2019:
„Tak naprawdę interpretacje te nie mają większego sensu, ponieważ rzeczywisty cel wykonywania operacji zależy od wag określających ich parametry, a wagi są uczone od końca do końca — proces ten rozpoczyna się od początku w każdej operacji trenowania, co uniemożliwia określenie celu poszczególnych operacji. Specyfikacja komórki RNN określa przestrzeń hipotez — przestrzeń, w której podczas trenowania ustalana jest poprawna konfiguracja modelu. Specyfikacja ta nie określa funkcji komórki. Cel pracy komórki zależy od jej wag. Ta sama komórka po przypisaniu różnych wag może wykonywać zupełnie inne operacje.”

Long short-term memory LSTM

W Kerasie można łatwo utworzyc sieć do klasyfikacji lub regresji zawierającą LSTM poprzez dodanie do modelu sekwencyjnego warstwy rekurencyjnej, a następnie warstwy gęstej. Wyjścia warstwy LSTM są wejściami warstwy gęstej, która jest odpowiednio dobrana do zadania regresji lub klasyfikacji.

```
#sieć dla problemu regresji
keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
model = Sequential()
model.add(LSTM(20, input_shape=(lahead, 1))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
```

Do skomplikowanych zadań można użyć kilka warstw rekurencyjnych w jednej sieci (stos warstw rekurencyjnych):

```
model = Sequential();
model.add(LSTM(liczba_neuronow), return_sequence=True)
model.add(LSTM(liczba_neuronow), return_sequence=True)
model.add(TimeDistributed(Dense(10))
```

GRU (ang. Gated Reccurent Unit)

Zostały przedstawione w 2014 w https://arxiv.org/abs/1406.1078. Mimo iż mają znacznie prostszą budowę od LSTM, to często osiągają zbliżoną wydajność. Ponadto wymagają wykonania znacznie mniejszej liczby operacji, więc szybciej można trenować.

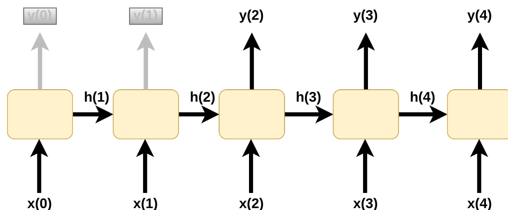
$$
\begin{aligned}
z(t) &= \sigma\left(W_{xz}^{T} x(t) + W_{hz}^{T} h(t-1) + b_z\right),\\
r(t) &= \sigma\left(W_{xr}^{T} x(t) + W_{hr}^{T} h(t-1) + b_r\right),\\
g(t) &= \tanh\left(W_{\mathrm{xg}}^{\mathrm{T}} x(\mathrm{t}) + W_{\mathrm{hg}}^{\mathrm{T}} r(\mathrm{t}) \otimes h(\mathrm{t}-1) + b_{\mathrm{g}}\right),\\
y(t) &= h(t) = z(t) \otimes h(t-1) + (1 - z(t) \otimes g(t)).
\end{aligned}
$$

UWAGA Istnieją różnorodne modyfikacje GRU
https://en.wikipedia.org/wiki/Gated_recurrent_unit

Funkcja straty (celu)

Propagacja wsteczna w czasie BPTT (ang. Backpropagation through time

Po rozwinięciu w czasie widać, że sieć można trenować algorytmem propagacji wstecznej.



Rysunek: Rozwinięcie w czasie

Wartość funkcja straty może zależeć tylko od niektórych wyjść

Funkcja straty w może zależeć tylko od trzech oststnich wyjść $E(Y(4), Y(3), Y(2))$.
W sieciach sekwencyjno-wektorowych istotne jest tylko ostatnie wyjście $Y(4)$.

Dygresja - nieliniowe obiekty dynamiczne

Można użyć sieci LSTM lub GRU do modelowania nieliniowych obiektów dynamicznych.

```
#sieć dla problemu regresji
keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
model = Sequential()
model.add(LSTM(20, input_shape=(lahead, 4))
model.add(Dense(1))
model.compile(loss='mse', optimizer='adam')
```

Komórki dwukierunkowe

Jeżeli cały tekst jest dostępny dla problemów predykcji, to można od razu wykorzystywać możliwość przesuwania w dwóch kierunkach - w przód i w tył. Są przechowywane dwa zestawy stanów ukrytych - w przód oraz wstecz (dla GRU 2 linie stanów **h**, dla LSTM dwie **h** i dwie **c**).

```
layer = Bidirectional (LSTM(100))
```

Zamiast LSTM można wewnątrz wzorca `Bidrectional` użyć `GRU` a nawet `SimpleRNN`. Prymitywne SimpleRNN nie są stosowane w praktyce nawet w jednym kierunku z wyjątkiem bardzo trywialnych zastosowań !

Tokenizacja - przetwarzanie języka naturalnego NLP (ang. Natural Language Processing)

Tokenizacja - proces podziału tekstu na pojedyncze jednostki, takie jak słowa lub znaki. Niekiedy najrzadziej występujące wyrazy są usuwane lub zastępowane wybranym ciągiem znaków (np. słowem "nieznany") , w celu użycia sieci z mniejszą ilością wag (sieć ma mniej do nauczenia się).
W przypadku wyrazów często przeprowadza się usuwanie końcówek (ang stemming).
Czasami zamienia się wszystkie litery na małe - wyrazy mają przyporządkowane takie same numery niezależnie czy zaczynają się dużą czy małą literą
Znaki interpunkcyjne można normalnie tokenizować, zamienić jednym numerem lub nawet całkowicie usunąć

```
tokenizator = keras.preprocessing.text.Tokenizer (char_level=True)  #False dla toker
tokenizator.fit_on_texts ([nasz_tekst])
```

Embedding - przetwarzanie języka naturalnego

Prosta warstwa, która zamienia tokeny w wektory np. 100 elementowe z wartościami rzeczywistymi. Uzyskiwana jest reprezentacja, w której słowa o podobnych znaczeniach są blisko siebie (jeżeli mamy porządnie wytrenowaną warstwę osadzeń (ang. embedding). Dodatkowo wektor między np. man i women jest podobny do wektora między słowami ram i sheep.

Warstwa Embedding jest dostępna w Kerasie.

```
from keras.layers import Embedding
embedding_layer = Embedding (liczba_tokenow, wymiar_osadzen)
```

Wymiar osadzeń oznacza liczbę elementów wektorów z liczbami rzeczywistymi. Można pobrać gotowe wagi dla języka angielskiego.

Generowanie tekstu w stylu wybranego autora

1) Pobranie dzieł(a) autora

2) Tokenizacja na poziomie słów lub znaków + Embedding

3) Utworzenie zbioru uczącego - sieć trenowana do przewidywania kolejnego znak/słowa

4) Trening sieci

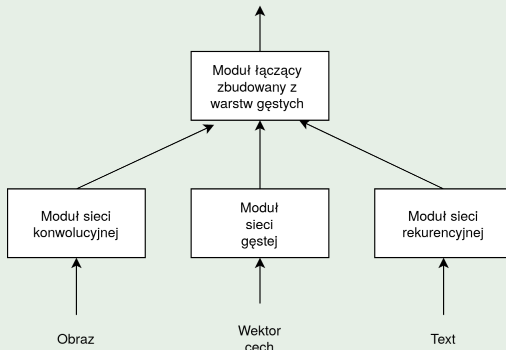
5) Rekurencyjne przewidywanie kolejnych znaków lub słów (najlepiej na podstawie prawdopodobieństw z tzw. temperaturą im mniejsza temperatura tym mniej stochastyczne próbkowanie - generowany tekst ma bardziej spójne słownictwo ale jest mniej barwny - mniej zaskakująca akcja)

Po warstwie rekurencyjnej np. LSTM może być jedna warstwa gęsta z funkcja aktywacji softmax - przewidywane prawdopodobieństwa kolejnych znaków lub słów. Przed warstwą rekurencyjną warstwa Embedding

Najłatwiej język angielski - dużo gotowych narzędzi, gotowych od razu do zastosowania z językiem agielskim.
Są gotowe wagi dla warstw Embedding wytrenowanych na zbiorze Stanford GloVe (Global Vectors) - 6 miliardów słów `https://nlp.stanford.edu/projects/glove/`.
Dla Pythona jest m.in. pakiet NLTK `https://www.nltk.org/` (można go też zastosować do innych języków).

Sieć przetwarzająca równocześnie różnorodne dane wejściowe (np. obrazy, tekst i dane numeryczne)

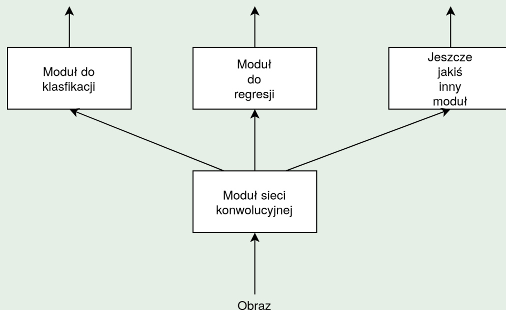
W kerasie można utworzyć sieci przetwarzające bardzo zróżnicowane dane



Rysunek: Przykład sieci do klasyfikacji lub regresji na podstawie obrazów, tekstu i danych numerycznych

Taką sieć można stworzyć w Kerasie z `functional API`.

Można też utworzyć sieć wykonującą równocześnie różne zadania



Rysunek: Przykład sieci wykonującej równocześnie różne zadania (z różnorodnymi wyjściami)

Taką sieć można stworzyć w Kerasie z `functional API`.

WaveNet https://arxiv.org/pdf/1609.03499.pdf

Opracowano specjalnie zmodyfikowaną architekturę sieci konwolucyjnych dla przetwarzania "surowych" sygnałów audio np. WaveNet https://arxiv.org/pdf/1609.03499.pdf (na płycie CD próbkowanie 44,1kHz, więc w ciągu jednej sekundy dla stereo jest 88200 próbek)

Transformer 2017r.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, Attention Is All You Need v7, https://arxiv.org/abs/1706.03762

UWAGA - najpierw uwagę stosowano w sieciach rekurencyjnych, ale kiepsko się zrównoleglało trenowanie rekurencyjnych

BERT itp.

BERT - Bidirectional Encoder Representations from Transformers (2018r) - oparty na powiększonym enkoderze z transformera
https://en.wikipedia.org/wiki/BERT_(language_model)

RoBERTa: A Robustly Optimized BERT Pretraining Approach
https://huggingface.co/docs/transformers/model_doc/roberta ,
https://arxiv.org/pdf/1907.11692

DistilBERT - mniejsza sieć https://arxiv.org/pdf/1910.01108

BERT, DistilBERT, RoBERTa to modele językowe oparte na transformatorach, używane do różnych zadań przetwarzania języka naturalnego. BERT to model dwukierunkowy, który uczy się informacji kontekstowych zarówno z kontekstu lewego, jak i prawego. DistilBERT to mniejsza i szybsza wersja BERT, zaprojektowana tak, aby była bardziej wydajna. RoBERTa to odmiana BERTa, która jest trenowana z dodatkowym pretreningiem i dłuższym czasem treningu, co skutkuje lepszą wydajnością. Modele te zastosowano do różnych zadań, takich jak analiza nastrojów, wykrywanie fałszywych wiadomości, rozpoznawanie emocji itd.

Hugging Face

https://huggingface.co/
Przykładowe kody z BERT:
https://huggingface.co/docs/transformers/model_doc/bert

GPT - Generative pre-trained transformers

GPT 1 - oparty na architekturze dekodera (powiększonej) z transformera

Później kolejne wersje GPT

https://en.wikipedia.org/wiki/Generative_pre-trained_transformer

OpenAI API

https://community.openai.com/t/the-system-role-how-it-influences-the-chat-behavior/87353
https://arize.com/blog-course/mastering-openai-api-tips-and-tricks/

Benchmark GLUE i SuperGLUE

GLUE - General Language Understanding Evaluation
https://arxiv.org/abs/1804.07461
SuperGLUE https://arxiv.org/pdf/1905.00537

Literatura

[AGeron] Aurélien Géron, Uczenie maszynowe z użyciem Scikit-Learn i TensorFlow, Wyd. II, Helion, 2020

[FChollet] Francois Chollet, Deep Learning. Praca z językiem Python i biblioteką Keras, Helion, 2019

[DFoster] David Foster, Deep learning i modelowanie generatywne. Jak nauczyć komputer malowania, pisania, komponowania i grania, Helion, 2021

[Rotman] Denis Rothman Transformers for Natural Language Processing and Computer Vision - Third Edition: Explore Generative AI and Large Language Models with Hugging Face, ChatGPT, GPT-4V, and DALL-E 3 3rd ed., Pact Publishing Ltd. 2024 - transformery, berty, gpt, DALL-E itp.

[Transformer] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin, Attention Is All You Need v7, https://arxiv.org/abs/1706.03762

Dziękuję za uwagę

Dziękuję za uwagę