DAY3

Section1：再帰型ニューラルネットワークの概念

◇要点

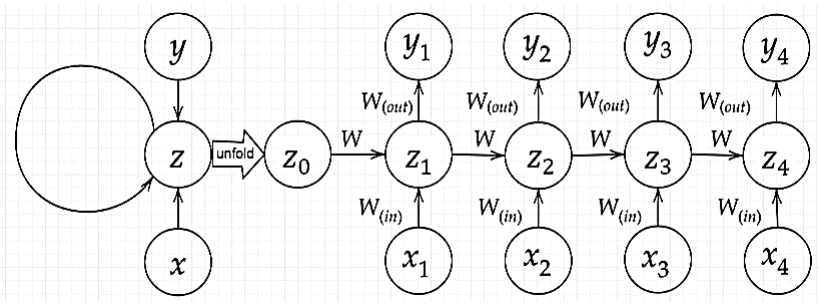
・RNN

時系列データに対応可能な、ニューラルネットワーク

・時系列データ

時間的順序を追って一定間隔ごとに観察され，しかも相互に統計的依存関係が認められるようなデータの系列

例）音声データ、テキストデータ



RNN の数式

$$u^t = W_{(in)}x^t + W\,z^{t-1} + b$$

$$z^t = f\left(W_{(in)}x^t + W\,z^{t-1} + b\right)$$

$$v^t = W_{(out)}z^t + c$$

$$y^t = g\left(W_{(out)}z^t + c\right)$$

```
u[:,t+1] = np.dot(X, W_in) + np.dot(z[:,t].reshape(1, -1), W)
z[:,t+1] = functions.sigmoid(u[:,t+1])
np.dot(z[:,t+1].reshape(1, -1), W_out)
y[:,t] = functions.sigmoid(np.dot(z[:,t+1].reshape(1, -1), W_out))
```

確認テスト

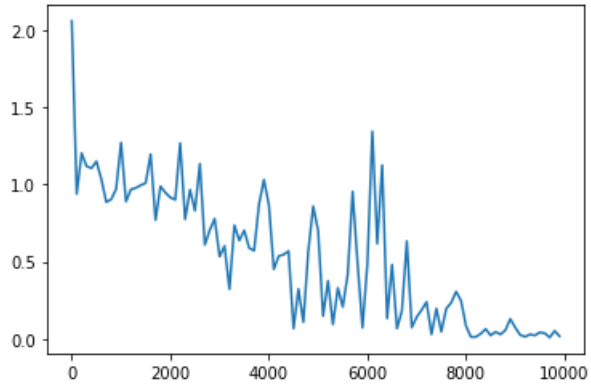
RNN のネットワークには大きくわけて 3 つの重みがある。1 つは入力から現在の中間層を定義する際にかけられる重み、1 つは中間層から出力を定義する際にかけられる重みである。残り 1 つの重みについて説明せよ。

A.中間層から中間層を定義する重み

・RNN の特徴

時系列モデルを扱うには、初期の状態と過去の時間 t-1 の状態を保持し、そこから次の時間での t を再帰的に求める再帰構造が必要

Simple RNN



3_1_simple_RNN.ipynb

準備

Googleドライブのマウント

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

sys.pathの設定

3_1_simple_RNN.ipynb

```
[5] sys.path.append('/content/drive/My Drive/Colab Notebooks/DNN_code_colab_lesson_3_4')
```

simple RNN

バイナリ加算

```
import numpy as np
from common import functions
import matplotlib.pyplot as plt


# def d_tanh(x):


# データを用意
# 2進数の桁数
binary_dim = 8
# 最大値 + 1
largest_number = pow(2, binary_dim)
# largest_numberまでの値を2進数に変換
binary = np.unpackbits(np.array([range(largest_number)],dtype=np.uint8).T,axis=1)
```

3_1_simple_RNN.ipynb

```
iters_num = 10000
plot_interval = 100

# ウェイト初期化 (バイアスは簡単のため省略)
W_in = weight_init_std * np.random.randn(input_layer_size, hidden_layer_size)
W_out = weight_init_std * np.random.randn(hidden_layer_size, output_layer_size)
W = weight_init_std * np.random.randn(hidden_layer_size, hidden_layer_size)

# Xavier


# He


# 勾配
W_in_grad = np.zeros_like(W_in)
W_out_grad = np.zeros_like(W_out)
W_grad = np.zeros_like(W)

u = np.zeros((hidden_layer_size, binary_dim + 1))
z = np.zeros((hidden_layer_size, binary_dim + 1))
y = np.zeros((output_layer_size, binary_dim))
```

3_1_simple_RNN.ipynb

```
    # A, B初期化 (a + b = d)
    a_int = np.random.randint(largest_number/2)
    a_bin = binary[a_int] # binary encoding
    b_int = np.random.randint(largest_number/2)
    b_bin = binary[b_int] # binary encoding

    # 正解データ
    d_int = a_int + b_int
    d_bin = binary[d_int]

    # 出力バイナリ
    out_bin = np.zeros_like(d_bin)

    # 時系列全体の誤差
    all_loss = 0

    # 時系列ループ
    for t in range(binary_dim):
        # 入力値
        X = np.array([a_bin[ - t - 1], b_bin[ - t - 1]]).reshape(1, -1)
        # 時刻tにおける正解データ
        dd = np.array([d_bin[binary_dim - t - 1]])

        u[:,t+1] = np.dot(X, W_in) + np.dot(z[:,t].reshape(1, -1), W)
```

3_1_simple_RNN.ipynb

```
            pass
            loss = functions.mean_squared_error(dd, y[:,t])

            delta_out[:,t] = functions.d_mean_squared_error(dd, y[:,t]) * functions.d_sigmoid(y[:,t])

            all_loss += loss

            out_bin[binary_dim - t - 1] = np.round(y[:,t])


        for t in range(binary_dim)[::-1]:
            X = np.array([a_bin[-t-1],b_bin[-t-1]]).reshape(1, -1)

            delta[:,t] = (np.dot(delta[:,t+1].T, W.T) + np.dot(delta_out[:,t].T, W_out.T)) * functions.d_sigmoid(u[:,t+1])

            # 勾配更新
            W_out_grad += np.dot(z[:,t+1].reshape(-1,1), delta_out[:,t].reshape(-1,1))
            W_grad += np.dot(z[:,t].reshape(1,-1), delta[:,t].reshape(1,-1))
            W_in_grad += np.dot(X.T, delta[:,t].reshape(1,-1))

        # 勾配適用
        W_in -= learning_rate * W_in_grad
        W_out -= learning_rate * W_out_grad
        W -= learning_rate * W_grad

        W_in_grad *= 0
```

3_1_simple_RNN.ipynb

```
        all_losses.append(all_loss)
        print("iters:" + str(i))
        print("Loss:" + str(all_loss))
        print("Pred:" + str(out_bin))
        print("True:" + str(d_bin))
        out_int = 0
        for index,x in enumerate(reversed(out_bin)):
            out_int += x * pow(2, index)
        print(str(a_int) + " + " + str(b_int) + " = " + str(out_int))
        print("------------")

lists = range(0, iters_num, plot_interval)
plt.plot(lists, all_losses, label="loss")
plt.show()
```

```
iters:0
Loss:2.0593381401289013
Pred:[0 0 0 0 0 0 0 0]
True:[0 1 1 0 1 1 0 0]
48 + 72 = 0
------------
iters:100
Loss:0.9397338857467855
Pred:[0 1 0 0 0 0 0 0]
True:[0 0 0 0 0 0 1 1]
36 + 29 = 64
------------
```

3_1_simple_RNN.ipynb

```
iters:5800
Loss:1.1185218117720645
Pred:[0 0 0 0 1 1 1 0]
True:[0 1 1 0 0 1 0 1]
14 + 87 = 14
------------
iters:5900
Loss:1.1705881124605787
Pred:[0 1 1 1 1 1 1 0]
True:[0 1 0 0 1 0 0 1]
45 + 30 = 126
------------
iters:6000
Loss:1.1504517482173076
Pred:[1 0 0 0 0 1 0 0]
True:[0 1 1 0 1 0 1 1]
104 + 3 = 132
------------
iters:6100
Loss:1.0347471215730055
Pred:[1 0 1 1 1 1 1 1]
True:[1 0 1 1 0 1 0 1]
181 + 80 = 255
------------
iters:6200
Loss:0.9985764465166213
Pred:[0 1 1 1 1 0 1 1]
True:[1 0 1 0 1 0 1 1]
85 + 170 = 123
```

3_1_simple_RNN.ipynb

```
iters:9800
Loss:0.9708911598935947
Pred:[1 0 0 1 1 0 0 0]
True:[0 1 1 0 1 1 0 0]
```

[try] weight_init_stdやlearning_rate, hidden_layer_sizeを変更してみよう

[try] 重みの初期化方法を変更してみよう

Xavier, He

[try] 中間層の活性化関数を変更してみよう

ReLU(勾配爆発を確認しよう)

tanh(numpyにtanhが用意されている。導関数をd_tanhとして作成しよう)

BPTT

RNN においてのパラメータ調整方法の一種⇒誤差逆伝播の一種

誤差逆伝播

計算結果（=誤差）から微分を逆算することで、不要な再帰的計算を避けて微分を算出

BPTT の数式

$$\frac{\partial E}{\partial W_{(in)}} = \frac{\partial E}{\partial u^t}\left[\frac{\partial u^t}{\partial W_{(in)}}\right]^T = \delta^t\left[x^t\right]^T$$

np.dot(X.T, delta[:,t].reshape(1,-1))

$$\frac{\partial E}{\partial W_{(out)}} = \frac{\partial E}{\partial v^t}\left[\frac{\partial v^t}{\partial W_{(out)}}\right]^T = \delta^{out,t}\left[z^t\right]^T$$

np.dot(z[:,t+1].reshape(-1,1), delta_out[:,t].reshape(-1,1))

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial u^t}\left[\frac{\partial u^t}{\partial W}\right]^T = \delta^t\left[z^{t-1}\right]^T$$

np.dot(z[:,t].reshape(-1,1), delta[:,t].reshape(1,-1))

$$\frac{\partial E}{\partial b} = \frac{\partial E}{\partial u^t}\frac{\partial u^t}{\partial b} = \delta^t$$

$$\frac{\partial E}{\partial c} = \frac{\partial E}{\partial v^t}\frac{\partial v^t}{\partial c} = \delta^{out,t}$$

$$u^t = W_{(in)}x^t + W\,z^{t-1} + b$$

u[:,t+1] = np.dot(X, W_in) + np.dot(z[:,t].reshape(1, -1), W)

$$z^t = f\left(W_{(in)}x^t + W\,z^{t-1} + b\right)$$

z[:,t+1] = functions.sigmoid(u[:,t+1])

$$v^t = W_{(out)}z^t + c$$

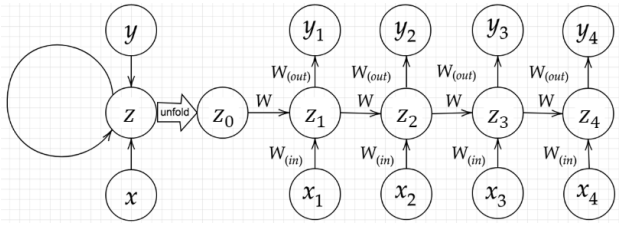
np.dot(z[:,t+1].reshape(1, -1), W_out)

$$y^t = g\left(W_{(out)}z^t + c\right)$$

y[:,t] = functions.sigmoid(np.dot(z[:,t+1].reshape(1, -1), W_out))

確認テスト

下図の y1 を x・s0・s1・win・w・wout を用いて数式で表せ。※バイアスは任意の文字で定義せよ。※また中間層の出力にシグモイド関数 g(x)を作用させよ。



z1=sigmoid(s0W+x1W(in)+b)

y1=sigmoid(z1W(out)+c)

$$\frac{\partial E}{\partial u^t} = \frac{\partial E}{\partial v^t}\frac{\partial v^t}{\partial u^t} = \frac{\partial E}{\partial v^t}\frac{\partial\{W_{(out)}f(u^t)+c\}}{\partial u^t} = f'(u^t)W_{(out)}^T\delta^{out,t} = \delta^t$$

delta[:,t]= (np.dot(delta[:,t+1].T, W.T) +np.dot(delta_out[:,t].T,W_out.T))* functions.d_sigmoid(u[:,t+1])

$$W_{(in)}^{t+1} = W_{(in)}^t - \epsilon\frac{\partial E}{\partial W_{(in)}} = W_{(in)}^t - \epsilon\sum_{z=0}^{T_t}\delta^{t-z}\left[\, x^{t-z}\,\right]^T$$

W_in -= learning_rate * W_in_grad

$$W^{t+1} = W^t - \epsilon\frac{\partial E}{\partial W} = W_{(in)}^t - \epsilon\sum_{z=0}^{T_t}\delta^{t-z}\left[\, z^{t-z-1}\,\right]^T$$

W_out -= learning_rate * W_out_grad

$$W^{t+1} = W^t - \epsilon\frac{\partial E}{\partial W} = W_{(in)}^t - \epsilon\sum_{z=0}^{T_t}\delta^{t-z}\left[\, z^{t-z-1}\,\right]^T$$

W -= learning_rate * W_grad

Section2：LSTM

◇要点

・RNN の課題

時系列を遡れば遡るほど、勾配が消失していく。長い時系列の学習が困難。

・解決策

構造自体を変えて解決したものが LSTM

・勾配消失問題

誤差逆伝播法が下位層に進んでいくに連れて、勾配がどんどん緩やかになっていく。そのため、勾配降下法による、更新では下位層のパラメータはほとんど変わらず、訓練は最適値に収束しなくなる。

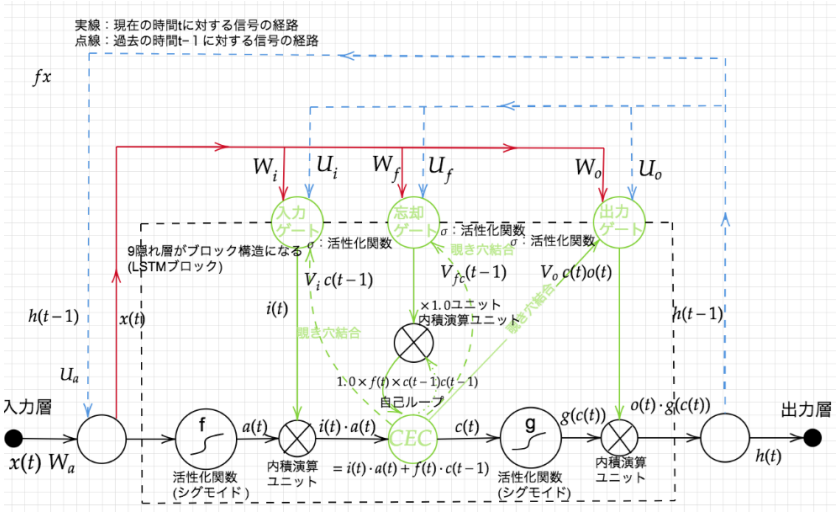
確認テスト

シグモイド関数を微分した時、入力値が 0 の時に最大値をとる。

F(1-f)=0.5(1-0.5)=0.25

勾配爆発

勾配が、層を逆伝播するごとに指数関数的に大きくなること



CEC

勾配消失および勾配爆発の解決方法として、勾配が、1 であれば解決できる。

$$\delta^{t-z-1} = \delta^{t-z}\left\{Wf'\left(u^{t-z-1}\right)\right\} = 1$$

$$\boxed{\frac{\partial E}{\partial c^{t-1}}} = \frac{\partial E}{\partial c^t}\frac{\partial c^t}{\partial c^{t-1}} = \frac{\partial E}{\partial c^t}\frac{\partial}{\partial c^{t-1}}\{a^t - c^{t-1}\} = \boxed{\frac{\partial E}{\partial c^t}}$$

課題　入力データについて、時間依存度に関係なく重みが一律である。ニューラルネットワークの学習特性が無いということ。

入力ゲート・出力ゲート

入力・出力ゲートを追加することで、それぞれのゲートへの入力値の重みを、重み行列 W,U で可変可能とする。（CEC の問題を解決）

・LSTM の現状

　CEC は、過去の情報が全て保管されている。

・課題

　過去の情報が要らなくなった場合、削除することはできず、保管され続ける。

・解決策

　過去の情報が要らなくなった場合、そのタイミングで情報を忘却する機能が必要⇒忘却ゲートの誕生

確認テスト

以下の文章を LSTM に入力し空欄に当てはまる単語を予測したいとする。文中の「とても」という言葉は空欄の予測においてなくなっても影響を及ぼさないと考えられる。このような場合、どのゲートが作用すると考えられるか。「映画おもしろかったね。ところで、とてもお腹が空いたから何か_____。」

A.忘却ゲート

覗き穴結合

課題

CEC の保存されている過去の情報を、任意のタイミングで他のノードに伝播させたり、あるいは任意のタイミングで忘却させたい。

CEC 自身の値は、ゲート制御に影響を与えていない。覗き穴結合とは？CEC 自身の値に、重み行列を介して伝播可能にした構造。

Section3：GRU

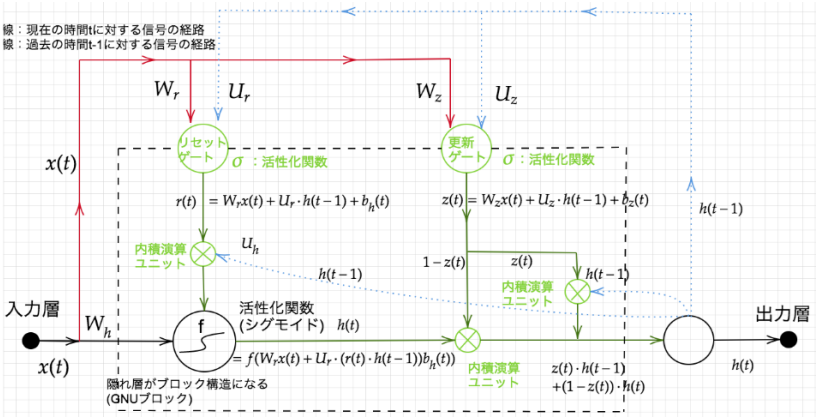
◇要点

LSTM では、パラメータ数が多く、計算負荷が高くなる問題があった。

GRU

従来の LSTM では、パラメータが多数存在していたため、計算負荷が大きかった。しかし、GRU では、そのパラメータを大幅に削減し、精度は同等またはそれ以上が望める様になった構造。

メリット　計算負荷が低い。



確認テスト

LSTM と CEC が抱える課題について、それぞれ簡潔に述べよ。

A.

LSTM の課題　パラメータ数が多く、計算負荷が高い。

CEC の課題　重みが一律になってしまい、NN の学習特性が無い。

LSTM と GRU の違いを簡潔に述べよ。

A.

　LSTM　パラメータ数が多く計算負荷が高い。

　GRU　パラメータ数が少なく計算負荷が低い。

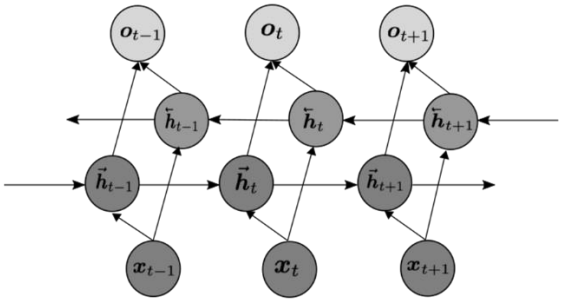
Section4：双方向 RNN

◇要点

過去の情報だけでなく、未来の情報を加味することで、精度を向上させるためのモデル

実用例　文章の推敲や、機械翻訳等

系列の最初のステップから繰り返して順方向に予想することに加えて，系列の最後のステップからの逆方向の予測も行う，RNN を双方向形に拡張したモデルである.



Section5：Seq2Seq

◇要点

Encoder-Decoder モデルの一種を指す。機械対話や、機械翻訳などに使用されている。

・Encoder RNN

ユーザーがインプットしたテキストデータを、単語等のトークンに区切って渡す構造。

Taking :文章を単語等のトークン毎に分割し、トークンごとの ID に分割する。

Embedding :ID から、そのトークンを表す分散表現ベクトルに変換。

Encoder RNN:ベクトルを順番に RNN に入力していく。

・Encoder RNN 処理手順

vec1 を RNN に入力し、hidden state を出力。この hiddenstate と次の入力 vec2 をまた RNN に入力してきた hidden state を出力という流れを繰り返す。

最後の vec を入れたときの hiddenstate を finalstate としてとっておく。この finalstate が thoughtvector と呼ばれ、入力した文の意味を表すベクトルとなる。

・Decoder RNN

システムがアウトプットデータを、単語等のトークンごとに生成する構造。

・Decoder RNN の処理

1.Decoder RNN: Encoder RNN の final state (thought vector) から、各 token の生成確率を出力していきます final state を Decoder RNN の initial state ととして設定し、Embedding を入力。

2.Sampling:生成確率にもとづいて token をランダムに選びます。

3.Embedding:2 で選ばれた token を Embedding して Decoder RNN への次の入力とします。

4.Detokenize:1 -3 を繰り返し、2 で得られた token を文字列に直します。

確認テスト

下記の選択肢から、seq2seq について説明しているものを選べ。

（1）時刻に関して順方向と逆方向の RNN を構成し、それら 2 つの中間層表現を特徴量として利用するものである。

（2）RNN を用いた Encoder-Decoder モデルの一種であり、機械翻訳などのモデルに使われる。

（3）構文木などの木構造に対して、隣接単語から表現ベクトル（フレーズ）を作るという演算を再帰的に行い（重みは共通）、文全体の表現ベクトルを得るニューラルネットワークである。

（4）RNN の一種であり、単純な RNN において問題となる勾配消失問題を CEC とゲートの概念を導入することで解決したものである。

Seq2seq の課題⇒一問一答しかできない⇒問に対して文脈も何もなく、ただ応答が行われる続ける。

・HRED

過去 n-1 個の発話から次の発話を生成する。Seq2seq では、会話の文脈無視で、応答がなされたが、HRED では、前の単語の流れに即して応答されるため、より人間らしい文章が生成される。

HRED の構造　Seq2Seq+ Context RNN

Context RNN: Encoder のまとめた各文章の系列をまとめて、これまでの会話コンテキスト全体を表すベクトルに変換する構造。⇒過去の発話の履歴を加味した返答をできる。

課題

・HRED は確率的な多様性が字面にしかなく、会話の「流れ」のような多様性が無い。

　同じコンテキスト（発話リスト）を与えられても、答えの内容が毎回会話の流れとしては同じものしか出せない

・HRED は短く情報量に乏しい答えをしがちである。

　短いよくある答えを学ぶ傾向がある。

・VHRED

　HRED に、VAE の潜在変数の概念を追加したもの。

　HRED の課題を、VAE の潜在変数の概念を追加することで解決した構造。

確認テスト

seq2seq と HRED、HRED と VHRED の違いを簡潔に述べよ。

A.

　seq2seq　一問一答しかできない

　HRED　会話の「流れ」のような多様性が無い

　VHRED　HRED に、VAE の潜在変数の概念を追加したもの

・オートエンコーダ

　教師なし学習の一つ。そのため学習時の入力データは訓練データのみで教師データは利用しない。

　具体例）MNIST の場合、28x28 の数字の画像を入れて、同じ画像を出力するニューラルネットワーク

　オートエンコーダ構造

　入力データから潜在変数 z に変換するニューラルネットワークを Encoder 逆に潜在変数 z をインプットとして元画像を復元するニューラルネットワークを Decoder

　メリット　次元削減が行える

・VAE

通常のオートエンコーダーの場合、何かしら潜在変数 z にデータを押し込めているものの、その構造がどのような状態かわからない。

VAE はこの潜在変数 z に確率分布 z~N(0,1)を仮定したもの。

VAE は、データを潜在変数 z の確率分布という構造に押し込めることを可能にする。

確認テスト

VAE に関する下記の説明文中の空欄に当てはまる言葉を答えよ。

A.自己符号化器の潜在変数に確率分布を導入したもの。

Section6：Word2vec

◇要点

課題:RNN では、単語のような可変長の文字列を NN に与えることはできない。

固定長形式で単語を表す必要がある。

学習データからボキャブラリを作成

※わかりやすく 7 語のボキャブラリを作成したら本来は、辞書の単語数だけできあがる。

I want to eat apples. I like apples.

↓

{apples,eat,I,like,to,want}

・one-hot ベクトル

Apple s を入力する場合は、入力層には以下のベクトルが入力される。

※本来は、辞書の単語数だけ one-hot ベクトルができあがる。

1…apples

0…eat

0…I

0…like

0…to

⋮

メリット大規模データの分散表現の学習が、現実的な計算速度とメモリ量で実現可能にした。

以前の方法：ボキャブラリ数 × ボキャブラリ数

word2vec：ボキャブラリ数 × 単語ベクトル

重みの数を減らせるようになった。

Section7：Attention Mechanism

◇要点

課題:seq2seq の問題は長い文章への対応が難しい。

seq2seq では、2 単語でも、100 単語でも、固定次元ベクトルの中に入力しなければならない。

「入力と出力のどの単語が関連しているのか」の関連度を学習する仕組み。

解決策:文章が長くなるほどそのシーケンスの内部表現の次元も大きくなっていく、仕組みが必要になる。

具体例

「私」「は」「ペン」「を」「持って」「いる」

「I」「have」「a」「pen」

「a」の関連度は低い。

「I」は「私」との関連度が高い。

確認テスト

RNNとword2vec、seq2seqとAttentionの違いを簡潔に述べよ。

A.

seq2seqは固定次元ベクトルの学習

Attention Mechanismは重要度、関連度の概念あり。

長い文章での翻訳が成り立つ。

DAY4

Section1：強化学習

◇要点

長期的に報酬を最大化できるように環境のなかで行動を選択できるエージェントを作ることを目標とする機械学習の一分野

行動の結果として与えられる利益(報酬)をもとに、行動を決定する原理を改善していく仕組み。

応用例

マーケティングの場合

環境:会社の販売促進部

エージェント:プロフィールと購入履歴に基づいて、キャンペーンメールを送る顧客を決めるソフトウェアである。

行動:顧客ごとに送信、非送信のふたつの行動を選ぶことになる。

報酬:キャンペーンのコストという負の報酬とキャンペーンで生み出されると推測される売上という正の報酬を受けるマーケティングの場合

探索と利用のトレードオフ

環境について事前に完璧な知識があれば、最適な行動を予測し決定することは可能。

どのような顧客にキャンペーンメールを送信すると、どのような行動を行うのかが既知である状況。

強化学習の場合、上記仮定は成り立たないとする。

不完全な知識を元に行動しながら、データを収集。最適な行動を見つけていく。

・探索が足りない状態

過去のデータで、ベストとされる行動のみを常に取り続ければ他にもっとベストな行動を見つけることはできない。

↑

トレードオフの関係性

↓

・利用が足りない状態

未知の行動のみを常に取り続ければ、過去の経験が活かせない。

強化学習と通常の教師あり、教師なし学習との違い

結論:目標が違う・教師なし、あり学習では、データに含まれるパターンを見つけ出すおよびそのデータから予測することが目標・

強化学習では、優れた方策を見つけることが目標

強化学習の歴史

強化学習について・冬の時代があったが、計算速度の進展により大規模な状態をもつ場合の、強化学習を可能としつつある。

・関数近似法と、Q 学習を組み合わせる手法の登場

Q 学習

・行動価値関数を、行動する毎に更新することにより学習を進める方法

関数近似法

・価値関数や方策関数を関数近似する手法のこと

価値関数

・価値を表す関数としては、状態価値関数と行動価値関数の 2 種類がある

ある状態の価値に注目する場合は、状態価値関数

状態と価値を組み合わせた価値に注目する場合は、行動価値関数

方策関数

方策ベースの強化学習手法において、ある状態でどのような行動を採るのかの確率を与える関数のことです。

方策勾配法について

$$\theta^{(t+1)} = \theta^{(t)} + \epsilon \nabla J(\theta)$$

定義方法

・平均報酬

・割引報酬和上記の定義に対応して、行動価値関数:Q(s,a)の定義を行い。

方策勾配定理が成り立つ。

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[(\nabla_\theta log \pi_\theta(a \mid s) Q^\pi(s,a))]$$

Section2：AlphaGo

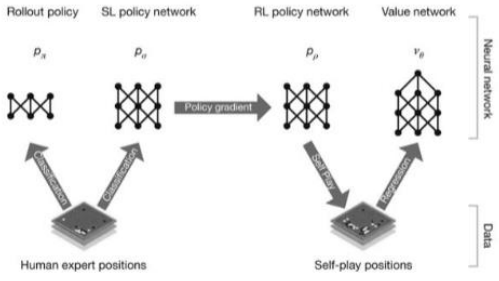
◇要点

AlphaGo の学習は以下のステップで行われる

１.教師あり学習による RollOutPolicy と PolicyNet の学習

２.強化学習による PolicyNet の学習

３.強化学習による ValueNet の学習



AlphaGoの学習フロー

出典:David Silver Mastering the game of Go with deep neural networks and tree search nature 27 January 2016 https://www.nature.com/articles/nature16961

・PolicyNet の教師あり学習

KGS Go Server（ネット囲碁対局サイト）の棋譜データから 3000 万局面分の教師を用意し、教師と同じ着手を予測できるよう学習を行った。具体的には、教師が着手した手を 1 とし残りを 0 とした 19×19 次元の配列を教師とし、それを分類問題として学習した。この学習で作成した PolicyNet は 57%ほどの精度である。

・PolicyNet の強化学習

現状の PolicyNet と PolicyPool からランダムに選択された PolicyNet と対局シミュレーションを行い、その結果を用いて方策勾配法で学習を行った。PolicyPool とは、PolicyNet の強化学習の過程を 500Iteraion ごとに記録し保存しておいたものである。現状の PolicyNet 同士の対局ではなく、PolicyPool に保存されているものとの対局を使用する理由は、対局に幅を持たせて過学習を防ごうというのが主である。この学習を minibatch size 128 で 1 万回行った。

・ValueNet の学習

PolicyNet を使用して対局シミュレーションを行い、その結果の勝敗を教師として学習した。教師データ作成の手順は 1 、まず SL PolicyNet(教師あり学習で作成した PolicyNet)で N 手まで打つ。2 、N+1 手目の手をランダムに選択し、その手で進めた局面を S（N+1）とする。3 、S（N+1）から RLPolicyNet（強化学習で作成した PolicyNet）で終局まで打ち、その勝敗報酬を R とする。S(N+1）と R を教師データ対とし、損失関数を平均二乗誤差とし、回帰問題として学習した。この学習を minibatch size 32 で 5000 万回行った N 手までと N+1 手からの PolicyNet を別々にしてある理由は、過学習を防ぐためであると論文では説明されている

Section3：軽量化・高速化技術

◇要点

分散深層学習とは

・深層学習は多くのデータを使用したり、パラメータ調整のために多くの時間を使用したりするため、高速な計算が求められる。

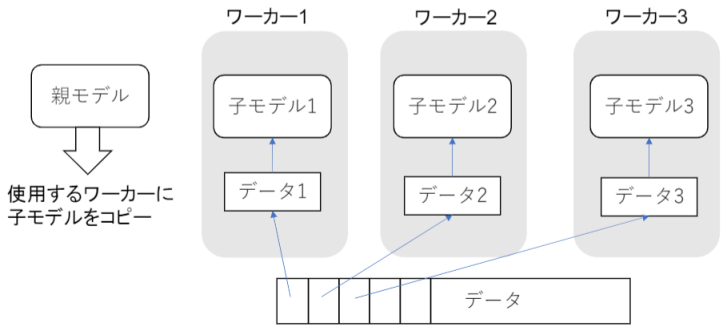
・複数の計算資源(ワーカー)を使用し、並列的にニューラルネットを構成することで、効率の良い学習を行いたい。

・データ並列化、モデル並列化、GPU による高速技術は不可欠である。

データ並列化

・親モデルを各ワーカーに子モデルとしてコピー

・データを分割し、各ワーカーごとに計算させる



同期型と非同期型の比較

・処理のスピードは、お互いのワーカーの計算を待たない非同期型の方が早い。

・非同期型は最新のモデルのパラメータを利用できないので、学習が不安定になりやすい。-> Stale Gradient Problem・

・現在は同期型の方が精度が良いことが多いので、主流となっている。

モデル並列化

・親モデルを各ワーカーに分割し、それぞれのモデルを学習させる。全てのデータで学習が終わった後で、一つのモデルに復元。

・モデルが大きい時はモデル並列化を、データが大きい時はデータ並列化をすると良い。

GPU による高速化
・GPGPU (General-purpose on GPU)
元々の使用目的であるグラフィック以外の用途で使用される GPU の総称
・CPU
高性能なコアが少数
複雑で連続的な処理が得意
・GPU
比較的低性能なコアが多数
簡単な並列処理が得意
ニューラルネットの学習は単純な行列演算が多いので、高速化が可能

モデルの軽量化
モデルの精度を維持しつつパラメータや演算回数を低減する手法の総称
高メモリ負荷高い演算性能が求められる通常は低メモリ低演算性能での利用が必要とされる Iot など

モデルの軽量化の利用
モデルの軽量化はモバイル, IoT 機器において有用な手法
モバイル端末や IoT はパソコンに比べ性能が大きく劣る
主に計算速度と搭載されているメモリ
モデルの軽量化は計算の高速化と省メモリ化を行うためモバイル,IoT 機器と相性が良い手法になる。

軽量化の手法
量子化：重みの精度を下げることにより計算の高速化と省メモリ化を行う技術
蒸留：複雑で精度の良い教師モデルから軽量な生徒モデルを効率よく学習を行う技術
プルーニング：寄与の少ないニューロンをモデルから削減し高速化と省メモリ化を行う技術

Section4：応用モデル
◇要点
MobileNet
Depthwise Convolution と Pointwise Convolution の組み合わせで軽量化を実現
一般的な畳み込みレイヤー
・入力特徴マップ(チャネル数)：H×W×C
・畳込みカーネルのサイズ：K×K×C
・出力チャネル数(フィルタ数)：M
・ストライド 1 でパディングを適用した場合の畳み込み計算の計算量
この点を計算するための計算量は K×K×C×M

Depthwise Convolution
仕組み
・入力マップのチャネルごとに畳み込みを実施

・出力マップをそれらと結合(入力マップのチャネル数と同じになる)

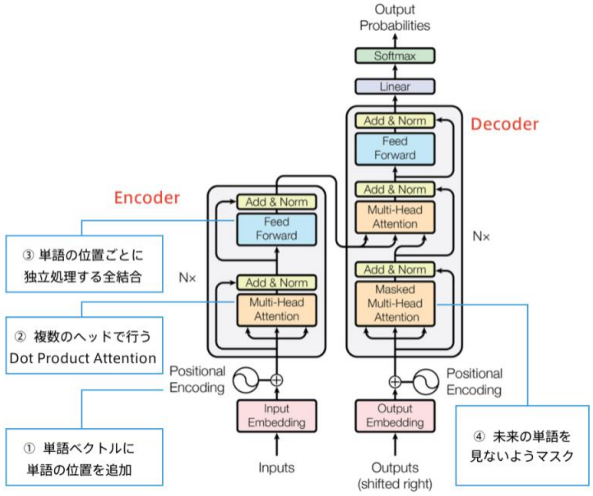
Pointwise Convolution
仕組み
・1 x 1 conv とも呼ばれる(正確には 1 x 1 x c)
・入力マップのポイントごとに畳み込みを実施
・出力マップ(チャネル数)はフィルタ数分だけ作成可能(任意のサイズが指定可能)

DenseNet
Dense Convolutional Network（以下、DenseNet）は、畳込みニューラルネットワーク（以下、CNN）アーキテクチャの一種である。ニューラルネットワークでは層が深くなるにつれて、学習が難しくなるという問題があったが、Residual Network（以下、ResNet）などの CNN アーキテクチャでは前方の層から後方の層へアイデンティティ接続を介してパスを作ることで問題を対処した。DenseBlock と呼ばれるモジュールを用いた、DenseNet もそのようなアーキテクチャの一つである。

Section5：Transformer
◇要点
Encoder-Decoder モデル
- Encoder RNN
- 翻訳元の文を読み込み、実数値ベクトルに変換
- Decoder RNN
- 実数値ベクトルから、翻訳先の言語の文を生成
Attention
翻訳先の各単語を選択する際に、翻訳元の文中の各単語の隠れ状態を利用
Attention は辞書オブジェクト
query(検索クエリ)に一致する key を索引し、対応する value を取り出す操作であると見做すことができる。これは辞書オブジェクトの機能と同じである

Transformer
2017 年 6 月に登場
- RNN を使わない
- 必要なのは Attention だけ
- 当時の SOTA をはるかに少ない計算量で実現
- 英仏（3600 万文）の学習を 8GPU で 3.5 日で完了
構造
・Encoder
Input Embedding
Positional Encoding
Encoder
Multi-Head Attention
Add & Norm
Feed Forward

Add & Norm
・Decoder
Output Embedding
Positional Encoding
Decoder
Masked Multi-Head Attention
Add & Norm
Multi-Head Attention
Add & Norm
Feed Forward
Add & Norm
Linear
Softmax



Section6：物体検知・セグメンテーション

◇要点
物体検知
物体検出は画像を取り込み、画像の中から定められた物体の位置とカテゴリー(クラス)を検出

技術の種類
・Faster R-CNN
R-CNN や Fast R-CNN を進化させた技術。Fast R-CNN では Selective Search という技術を使っていたが処理速度がネックになっていた。Faster R-CNN では、RPN(Region Proposal Network) と言われる小さな畳み込みネットワークで問題点を解決。

・YOLO
一枚の画像を特定サイズのグリッドに分割。その分割されたグリッドごとにクラス推定とバウンディングボックスの回帰を実施
・SSD
OLO に類似した物体検出タスク用モデル。回帰であることが特徴

通常の CNN
SSD は各層の大きさの異なる特徴マップを利用して物体を検出

セグメンテーション
セグメンテーションでは物体の位置をピクセル単位で特定
全畳み込みネットワーク（Fully Convolutional Network, FCN）
セマンティックセグメンテーション（Semantic Segmentation）の代表的なモデル

セマンティックセグメンテーション
・ピクセルごとにクラス分類を解く
・いくつかのピクセルが孤立して異なるクラスに分類される問題あり
・この問題は条件付き確率場（Conditional random field, CRF）による後処理で解決

TensorflowFlow

base

```
[ ] a = code base
```

線形回帰

[try]

- notes3の作成者しよう
- AIの勉強会参考しよう

```
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# データ生成
x = np.random.rand(100)
y = 3 * x + 0.5

# パラメータ
a = tf.Variable(0.0)
b = tf.Variable(0.0)

# 予測値
y_pred = a * x + b

# 損失関数（平均二乗誤差）
loss = tf.reduce_mean(tf.square(y_pred - y))

# 最適化
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# 学習
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for step in range(201):
        sess.run(train)
        if step % 20 == 0:
            print(step, sess.run(a), sess.run(b), sess.run(loss))
```

```
import numpy as np
import matplotlib.pyplot as plt

# 真の関数：y = 3x + 0.5 + ノイズ
np.random.seed(0)
x = np.random.rand(100)
y = 3 * x + 0.5 + np.random.randn(100) * 0.1

# 学習率
lr = 0.1

# パラメータ初期化
a = 0.0
b = 0.0

# 学習ループ
for step in range(201):
    y_pred = a * x + b
    loss = np.mean((y_pred - y) ** 2)

    # 勾配計算
    grad_a = np.mean(2 * (y_pred - y) * x)
    grad_b = np.mean(2 * (y_pred - y))

    # 更新
    a = a - lr * grad_a
    b = b - lr * grad_b

    if step % 20 == 0:
        print(step, a, b, loss)

# 可視化
plt.scatter(x, y, label="data")
plt.plot(x, a * x + b, color="red", label="fit")
plt.legend()
plt.show()
```

単純形回帰

[try]

- notes3の作成者しよう
- AIの勉強会参考しよう

```
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
np.random.seed(0)
x = np.random.rand(100).astype(np.float32)
y = 3 * x + 0.5 + np.random.randn(100) * 0.1

# パラメータ
a = tf.Variable(0.0)
b = tf.Variable(0.0)

# 予測
y_pred = a * x + b

# 損失（平均二乗誤差）
loss = tf.reduce_mean(tf.square(y_pred - y))

# 最適化
optimizer = tf.train.GradientDescentOptimizer(0.5)
train = optimizer.minimize(loss)

# 学習
init = tf.global_variables_initializer()
with tf.Session() as sess:
    sess.run(init)
    for step in range(201):
        sess.run(train)
        if step % 20 == 0:
            print(step, sess.run(a), sess.run(b), sess.run(loss))
```

```
# 可視化
plt.scatter(x, y, label="data")
plt.plot(x, sess.run(a) * x + sess.run(b), color="red", label="fit")
plt.legend()
plt.show()
```

```
# ハイパーパラメータ
lr = 0.1
for method in ["GradientDescent", "Momentum", "Adam"]:
    # 初期化
    a = tf.Variable(0.0)
    b = tf.Variable(0.0)
    y_pred = a * x + b
    loss = tf.reduce_mean(tf.square(y_pred - y))

    if method == "GradientDescent":
        optimizer = tf.train.GradientDescentOptimizer(lr)
    elif method == "Momentum":
        optimizer = tf.train.MomentumOptimizer(lr, momentum=0.9)
    elif method == "Adam":
        optimizer = tf.train.AdamOptimizer(lr)

    train = optimizer.minimize(loss)

    losses = []
    with tf.Session() as sess:
        sess.run(init)
        for step in range(201):
            sess.run(train)
            losses.append(sess.run(loss))

    plt.plot(losses, label=method)

plt.yscale("log")
plt.legend()
plt.show()
```

$$y = 3x^2 - 0.5x + 0.2$$

- 別数が収束するよう3class_numの調整しよう

```
[ ]
```

分類3層 (resize)

[try]

- x, y, 入力値 x・教師データ y を、重み a、バイアス b それぞれ定義しよう

```
import tensorflow as tf
import matplotlib as matp
```

```
# tensorflow 初期化
tf.keras.set_memory_growth(gpu[0], True)
```

```
from tensorflow.examples.tutorials.mnist import input_data
```

```
# ハイパーパラメータ
learning_rate = 0.1
training_epochs = 15
batch_size = 100
display_step = 1

# TensorFlow グラフの入力
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])

# 重み W とバイアス b の定義
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# 活性化関数（ソフトマックス回帰）
activation = tf.nn.softmax(tf.matmul(x, W) + b)

# 損失関数（クロスエントロピー）
cost = tf.reduce_mean(-tf.reduce_sum(y * tf.log(activation), reduction_indices=1))

# 勾配降下法によるオプティマイザ
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

# 変数の初期化
init = tf.global_variables_initializer()

# セッションを開始
with tf.Session() as sess:
    sess.run(init)

    # 学習サイクル
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # すべてのバッチを表示
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # 最適化処理（逆伝播）と損失関数の計算
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
            # 平均損失を計算
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})/total_batch
        # 指定したエポックごとに学習状況を表示
        if (epoch+1) % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))

    print("Optimization Finished!")

    # テストモデル
    correct_prediction = tf.equal(tf.argmax(activation, 1), tf.argmax(y, 1))
    # 精度を計算
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print("Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels}))
```

分類3層 (resize)

[try]

- 隠れ層のサイズを変更してみよう
- optimizerを変更しよう

```
from tensorflow.keras.optimizers import Adam

# ハイパーパラメータ
learning_rate = 0.001
training_epochs = 15
batch_size = 100
display_step = 1

# TensorFlow グラフの入力
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])

# 重み W とバイアス b の定義
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# 活性化関数（ソフトマックス回帰）
activation = tf.nn.softmax(tf.matmul(x, W) + b)

# 損失関数（クロスエントロピー）
cost = tf.reduce_mean(-tf.reduce_sum(y * tf.log(activation), reduction_indices=1))

# 勾配降下法によるオプティマイザ
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

# 変数の初期化
init = tf.global_variables_initializer()

# セッションを開始
with tf.Session() as sess:
    sess.run(init)

    # 学習サイクル
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # すべてのバッチを表示
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # 最適化処理（逆伝播）と損失関数の計算
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
            # 平均損失を計算
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})/total_batch
        # 指定したエポックごとに学習状況を表示
        if (epoch+1) % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))

    print("Optimization Finished!")

    # テストモデル
    correct_prediction = tf.equal(tf.argmax(activation, 1), tf.argmax(y, 1))
    # 精度を計算
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print("Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels}))
```

分類CNN (resize)

conv - elu - pool - conv - elu - pool -
affin - elu - dropout - affin - softmax

[try]

- ドロップアウト率を0.5に変更しよう

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
```

```
# tensorflow 初期化
tf.keras.set_memory_growth(gpu[0], True)
```

```
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

# ハイパーパラメータ
learning_rate = 0.001
training_epochs = 15
batch_size = 100
display_step = 1

# TensorFlow グラフの入力
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])

# 重み W とバイアス b の定義
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# 活性化関数（ソフトマックス回帰）
activation = tf.nn.softmax(tf.matmul(x, W) + b)

# 損失関数（クロスエントロピー）
cost = tf.reduce_mean(-tf.reduce_sum(y * tf.log(activation), reduction_indices=1))

# 勾配降下法によるオプティマイザ
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

# 変数の初期化
init = tf.global_variables_initializer()

# セッションを開始
with tf.Session() as sess:
    sess.run(init)

    # 学習サイクル
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # すべてのバッチを表示
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # 最適化処理（逆伝播）と損失関数の計算
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
            # 平均損失を計算
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})/total_batch
        # 指定したエポックごとに学習状況を表示
        if (epoch+1) % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))

    print("Optimization Finished!")

    # テストモデル
    correct_prediction = tf.equal(tf.argmax(activation, 1), tf.argmax(y, 1))
    # 精度を計算
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print("Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels}))
```

```
[ ]
```

```
# ハイパーパラメータ
learning_rate = 0.001
training_epochs = 15
batch_size = 100
display_step = 1

# TensorFlow グラフの入力
x = tf.placeholder(tf.float32, [None, 784])
y = tf.placeholder(tf.float32, [None, 10])

# 重み W とバイアス b の定義
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# 活性化関数（ソフトマックス回帰）
activation = tf.nn.softmax(tf.matmul(x, W) + b)

# 損失関数（クロスエントロピー）
cost = tf.reduce_mean(-tf.reduce_sum(y * tf.log(activation), reduction_indices=1))

# 勾配降下法によるオプティマイザ
optimizer = tf.train.AdamOptimizer(learning_rate).minimize(cost)

# 変数の初期化
init = tf.global_variables_initializer()

# セッションを開始
with tf.Session() as sess:
    sess.run(init)

    # 学習サイクル
    for epoch in range(training_epochs):
        avg_cost = 0.
        total_batch = int(mnist.train.num_examples/batch_size)
        # すべてのバッチを表示
        for i in range(total_batch):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)
            # 最適化処理（逆伝播）と損失関数の計算
            sess.run(optimizer, feed_dict={x: batch_xs, y: batch_ys})
            # 平均損失を計算
            avg_cost += sess.run(cost, feed_dict={x: batch_xs, y: batch_ys})/total_batch
        # 指定したエポックごとに学習状況を表示
        if (epoch+1) % display_step == 0:
            print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))

    print("Optimization Finished!")

    # テストモデル
    correct_prediction = tf.equal(tf.argmax(activation, 1), tf.argmax(y, 1))
    # 精度を計算
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
    print("Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels}))
```

準備

Googleドライブのマウント

```
from google.colab import drive
drive.mount('/content/drive')
```

sys.pathの設定

以下では、Googleドライブのマイドライブ下にDNN_codeフォルダ作成のこととを仮定しています。必要に応じて、パスを変更してください。

keras

線形回帰

```
import tensorflow as tf

# tpuの設定
if 'COLAB_TPU_ADDR' in os.environ:
  cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver()
  tf.config.experimental_connect_to_cluster(cluster_resolver)
  tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
  strategy = tf.distribute.experimental.TPUStrategy(cluster_resolver)
  print('Running on TPU ', cluster_resolver.cluster_spec().as_dict()['worker'])
else:
  strategy = tf.distribute.get_strategy()
  print('Running on CPU')
print("Number of accelerators: ", strategy.num_replicas_in_sync)

# 乱数のシード設定
tf.random.set_seed(0)

# モデルの構築
with strategy.scope():
  model = tf.keras.models.Sequential()
  model.add(tf.keras.layers.Dense(1, input_shape=(1,)))

# モデルのコンパイル
model.compile(optimizer='sgd', loss='mse')

# データの生成
import numpy as np
x = np.linspace(-1, 1, 101)
y = 2 * x + np.random.randn(*x.shape) * 0.33

# モデルの訓練
model.fit(x, y, epochs=10)

# モデルの評価
model.evaluate(x, y)

# 重みの確認
W, b = model.get_weights()
print('W = ', W, 'b = ', b)
```

線形(パーセプトロン)

- np.random.seed(0)を np.random.seed(1)に変更
- エポック回数を10に変更
- 学習率を0.1に変更
- 活性関数をシグモイド関数に変更
- エポック回数を1000に変更しよう

```
import tensorflow as tf

# tpuの設定
if 'COLAB_TPU_ADDR' in os.environ:
  cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver()
  tf.config.experimental_connect_to_cluster(cluster_resolver)
  tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
  strategy = tf.distribute.experimental.TPUStrategy(cluster_resolver)
  print('Running on TPU ', cluster_resolver.cluster_spec().as_dict()['worker'])
else:
  strategy = tf.distribute.get_strategy()
  print('Running on CPU')
print("Number of accelerators: ", strategy.num_replicas_in_sync)

# 乱数のシード設定
tf.random.set_seed(0)

# モデルの構築
with strategy.scope():
  model = tf.keras.models.Sequential()
  model.add(tf.keras.layers.Dense(1, input_shape=(1,)))

# モデルのコンパイル
model.compile(optimizer='sgd', loss='mse')

# データの生成
import numpy as np
x = np.linspace(-1, 1, 101)
y = 2 * x + np.random.randn(*x.shape) * 0.33

# モデルの訓練
model.fit(x, y, epochs=10)

# モデルの評価
model.evaluate(x, y)

# 重みの確認
W, b = model.get_weights()
print('W = ', W, 'b = ', b)
```

分類 (iris)

Iris

- 中間層の活性関数をsigmoidに変更しよう
- 損失関数をcategorical_crossentropyに変更しよう

```
import tensorflow as tf

# tpuの設定
if 'COLAB_TPU_ADDR' in os.environ:
  cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver()
  tf.config.experimental_connect_to_cluster(cluster_resolver)
  tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
  strategy = tf.distribute.experimental.TPUStrategy(cluster_resolver)
  print('Running on TPU ', cluster_resolver.cluster_spec().as_dict()['worker'])
else:
  strategy = tf.distribute.get_strategy()
  print('Running on CPU')
print("Number of accelerators: ", strategy.num_replicas_in_sync)

# 乱数のシード設定
tf.random.set_seed(0)

# モデルの構築
with strategy.scope():
  model = tf.keras.models.Sequential()
  model.add(tf.keras.layers.Dense(3, input_shape=(4,), activation='sigmoid'))
  model.add(tf.keras.layers.Dense(3, activation='softmax'))

# モデルのコンパイル
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

# データの生成
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

iris = load_iris()
x = iris.data
y = to_categorical(iris.target)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# モデルの訓練
model.fit(x_train, y_train, epochs=100)

# モデルの評価
model.evaluate(x_test, y_test)
```

分類 (mnist)

Iris

- hidden_sizes=[128,128]を hidden_sizes=[16,16]に変更しよう
- 訓練関数train_categorical_crossentropyに変更しよう
- Adamの引数の値を変更しよう

```
import tensorflow as tf

# tpuの設定
if 'COLAB_TPU_ADDR' in os.environ:
  cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver()
  tf.config.experimental_connect_to_cluster(cluster_resolver)
  tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
  strategy = tf.distribute.experimental.TPUStrategy(cluster_resolver)
  print('Running on TPU ', cluster_resolver.cluster_spec().as_dict()['worker'])
else:
  strategy = tf.distribute.get_strategy()
  print('Running on CPU')
print("Number of accelerators: ", strategy.num_replicas_in_sync)

# 乱数のシード設定
tf.random.set_seed(0)

# モデルの構築
with strategy.scope():
  model = tf.keras.models.Sequential()
  model.add(tf.keras.layers.Flatten(input_shape=(28, 28)))
  model.add(tf.keras.layers.Dense(128, activation='relu'))
  model.add(tf.keras.layers.Dense(128, activation='relu'))
  model.add(tf.keras.layers.Dense(10, activation='softmax'))

# モデルのコンパイル
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# データの準備
from tensorflow.keras.datasets import mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], -1) / 255.0
x_test = x_test.reshape(x_test.shape[0], -1) / 255.0

# 訓練
model.fit(x_train, y_train, epochs=5)

# 評価
model.evaluate(x_test, y_test)
```

CNN(手書) (mnist)

畳込み層追加で99%超え目標

```
import tensorflow as tf

# tpuの設定
if 'COLAB_TPU_ADDR' in os.environ:
  cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver()
  tf.config.experimental_connect_to_cluster(cluster_resolver)
  tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
  strategy = tf.distribute.experimental.TPUStrategy(cluster_resolver)
  print('Running on TPU ', cluster_resolver.cluster_spec().as_dict()['worker'])
else:
  strategy = tf.distribute.get_strategy()
  print('Running on CPU')
print("Number of accelerators: ", strategy.num_replicas_in_sync)

# 乱数のシード設定
tf.random.set_seed(0)

# モデルの構築
with strategy.scope():
  model = tf.keras.models.Sequential()
  model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
  model.add(tf.keras.layers.MaxPooling2D((2, 2)))
  model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
  model.add(tf.keras.layers.MaxPooling2D((2, 2)))
  model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
  model.add(tf.keras.layers.Flatten())
  model.add(tf.keras.layers.Dense(64, activation='relu'))
  model.add(tf.keras.layers.Dense(10, activation='softmax'))
  model.summary()

# データの準備
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train = x_train.reshape(x_train.shape[0], 28, 28, 1) / 255.0
x_test = x_test.reshape(x_test.shape[0], 28, 28, 1) / 255.0
y_train = to_categorical(y_train)
y_test = to_categorical(y_test)

# 訓練
model.fit(x_train, y_train, epochs=5)

# 評価
model.evaluate(x_test, y_test)
```

alllay10

畳込み層増加で99%超え目標

データセット:alllay10
Alpha10フリーな文字画像 画像データ一式
10種のフリーな 手書き 文字 画像 素 材 集 画 像 。 (Pixiv)
https://www.pixiv.net/artworks/88519888
https://www.pixiv.net/artworks/88519888

```
import tensorflow as tf

# tpuの設定
if 'COLAB_TPU_ADDR' in os.environ:
  cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver()
  tf.config.experimental_connect_to_cluster(cluster_resolver)
  tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
  strategy = tf.distribute.experimental.TPUStrategy(cluster_resolver)
  print('Running on TPU ', cluster_resolver.cluster_spec().as_dict()['worker'])
else:
  strategy = tf.distribute.get_strategy()
  print('Running on CPU')
print("Number of accelerators: ", strategy.num_replicas_in_sync)

# 乱数のシード設定
tf.random.set_seed(0)

# モデルの構築
with strategy.scope():
  model = tf.keras.models.Sequential()
  model.add(tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
  model.add(tf.keras.layers.MaxPooling2D((2, 2)))
  model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
  model.add(tf.keras.layers.MaxPooling2D((2, 2)))
  model.add(tf.keras.layers.Conv2D(64, (3, 3), activation='relu'))
  model.add(tf.keras.layers.Flatten())
  model.add(tf.keras.layers.Dense(64, activation='relu'))
  model.add(tf.keras.layers.Dense(10, activation='softmax'))
  model.summary()

# データの準備
from tensorflow.keras.utils import to_categorical
import numpy as np
import glob
from PIL import Image
x = []
y = []
for i in range(10):
  files = glob.glob('/content/drive/MyDrive/DNN_code/alllay10/' + str(i) + '/*.png')
  for file in files:
    image = Image.open(file).convert('L')
    image = image.resize((28, 28))
    data = np.asarray(image)
    x.append(data)
    y.append(i)
x = np.array(x)
y = np.array(y)
x = x.reshape(x.shape[0], 28, 28, 1) / 255.0
y = to_categorical(y)
x_train, x_test = x[:1000], x[1000:]
y_train, y_test = y[:1000], y[1000:]

# 訓練
model.fit(x_train, y_train, epochs=5)

# 評価
model.evaluate(x_test, y_test)
```

回帰

Wine(白)データセット https://archive.ics.uci.edu/ml/datasets/Wine+quality

Iris

- 中間層のユニット数を128に変更
- 中間層の活性関数をsigmoidに変更
- 損失関数を確率的勾配降下法(sgd)に変更
- 最適化の反復回数(epoch)を変更
- 訓練データとテストデータの分割比率を変更
- 中間層数をRegular Net (3層)に変更
- 中間層数をDeep Net (5層)に変更

```
import tensorflow as tf

# tpuの設定
if 'COLAB_TPU_ADDR' in os.environ:
  cluster_resolver = tf.distribute.cluster_resolver.TPUClusterResolver()
  tf.config.experimental_connect_to_cluster(cluster_resolver)
  tf.tpu.experimental.initialize_tpu_system(cluster_resolver)
  strategy = tf.distribute.experimental.TPUStrategy(cluster_resolver)
  print('Running on TPU ', cluster_resolver.cluster_spec().as_dict()['worker'])
else:
  strategy = tf.distribute.get_strategy()
  print('Running on CPU')
print("Number of accelerators: ", strategy.num_replicas_in_sync)

# 乱数のシード設定
tf.random.set_seed(0)

# モデルの構築
with strategy.scope():
  model = tf.keras.models.Sequential()
  model.add(tf.keras.layers.Dense(128, input_shape=(4,), activation='sigmoid'))
  model.add(tf.keras.layers.Dense(128, activation='sigmoid'))
  model.add(tf.keras.layers.Dense(128, activation='sigmoid'))
  model.add(tf.keras.layers.Dense(128, activation='sigmoid'))
  model.add(tf.keras.layers.Dense(3, activation='softmax'))

# モデルのコンパイル
model.compile(optimizer='sgd', loss='categorical_crossentropy', metrics=['accuracy'])

# データの準備
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical

iris = load_iris()
x = iris.data
y = to_categorical(iris.target)
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2)

# 訓練
model.fit(x_train, y_train, epochs=100)

# 評価
model.evaluate(x_test, y_test)
```

predict_word.ipynb

File Edit View Insert Runtime Tools Help

準備

Googleドライブのマウント

```
from google.colab import drive
drive.mount('[illegible]')
```

ディレクトリを移動

以下では、Googleドライブのマイドライブ直下にDNN_codeフォルダを置くことを仮定しています。必要に応じて、パスを変更してくださ

predict_word.ipynb

File Edit View Insert Runtime Tools Help

predict_word

```
import tensorflow as tf
import numpy as np
import re
import glob
import collections
import random
import pickle
import time
import datetime
import os

# logging levelを変更
[illegible]

class Corpus:
    def __init__(self):
        [illegible]
```

[illegible]

Table of contents
TensorFlow
Setup
constant
Variable
placeholder
session
定数の計算
行列の計算
行列の積 (matmul)
行列の逆行列 (matrix_inverse)
行列式 (matrix_determinant)
固有値と固有ベクトル (self_adjoint_eig)
連立一次方程式の解 (matrix_solve)
最小二乗法の解 (matrix_solve_ls)
特異値分解 (svd)
コレスキー分解 (cholesky)
QR分解 (qr)
multiply
reduce_sum
reduce_mean
argmax
equal
cast
reduce_mean
Variable
optimizer
GradientDescentOptimizer
minimize
placeholder
session
run
print
Session
run
close
with
定数