# DNN

# Day1

## Section 1: 入力層・中間層

中間層のユニット数がひとつのネットワークに対して以下の設定を考える。

- 入力のベクトル $\mathbf{x}$
- 入力層と中間層の接続の重み $\mathbf{w}$
- 入力層と中間層の接続のバイアス $\mathbf{b}$
- 中間層のユニットの総入力 $u$
- 中間層の出力 $z$
- 活性化関数 $f$

このとき、中間層の総入力 $u$ は次の式で与えられる。

$$u = \mathbf{w}\mathbf{x} + \mathbf{b}$$

なお、上式を `numpy` を用いて実装すると次のようになる。

```
import numpy as np
u = np.dot(w, x) + b
```

## Section 2: 活性化関数

ニューラルネットワークにおいて、次の層への出力の大きさを決める非線形の関数のこと。入力値によって、次の層への信号のON/OFFや強弱を定める働きを持つ。

- 中間層用によく使われる活性化関数
  - ReLU関数
  - シグモイド（ロジスティック）関数
  - ステップ関数

- 出力層用によく使われる活性化関数
  - ソフトマックス関数
  - 恒等写像
  - シグモイド（ロジスティック）関数

## Section 3: 出力層

分類問題を解きたいとき事前に用意するデータとしては入力データと分類の正解ラベルがある。ニューラルネットワークから出力される値と正解との差を評価するために誤差関数が用いられる。誤差関数には例えば二乗誤差がある。

$$E_n(\mathbf{w}) = \frac{1}{2}\sum_{j=1}^{J}(y_j - d_j)^2 = \frac{1}{2}||(\mathbf{y} - \mathbf{d})||^2$$

それぞれの問題で用いられる出力層の活性化関数と誤差関数の組み合わせは以下の表の通り。

| | 回帰 | 二値分類 | 多クラス分類 |
|---|---|---|---|
| 活性化関数 | 恒等写像 | シグモイド関数 | ソフトマックス関数 |
| 誤差関数 | 二乗誤差 | 交差エントロピー | 交差エントロピー |

## Section 4: 勾配降下法

勾配降下法では以下の式に従ってパラメータを最適化する。

$$w^{(t+1)} = w^{(t)} - \varepsilon\nabla E$$

ただし、$\varepsilon$ は学習率、$E$ は誤差の値とする。学習率が多すぎると最小値にいつまでもたどり着かず発散してしまうことがある。学習率が小さいと発散することはないが、小さすぎると収束するまでに時間がかかってしまう。勾配降下法のアルゴリズムには以下のようなものがある。

- Momentum
- AdaGrad
- Adadelta
- Adam

勾配降下法は全サンプルの平均誤差を用いるのに対して、確率的勾配降下法（SGD）はランダムに抽出したサンプルの誤差を用いる。メリットは以下の通り。

- データが冗長な場合の計算コストの軽減
- 望まない局所極少解に収束するリスクの軽減
- オンライン学習ができる

これらの中間的な方法として、ランダムに分割したデータの集合に属するサンプルの平均誤差を用いるミニバッチ勾配降下法がある。ミニバッチ勾配降下法のメリットには次のようなものがある。

- 確率的勾配降下法のメリットを損なわず、計算機の資源を有効活用できる
    - CPUを利用したスレッド並列化やGPUを利用したSIMD並列化

### Section 5: 誤差逆伝播法

算出された誤差を、出力層側から順に微分し、前の層前の層へと伝播。最小限の計算で各パラメータでの微分値を解析的に計算する手法。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できる。

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial u}\frac{\partial u}{\partial w_{ji}}$$

出力層の活性化関数は恒等写像、誤差関数を二乗誤差とすると、それぞれの項は以下のように計算できる。

$$\frac{\partial E}{\partial y} = y - d$$

$$\frac{\partial y}{\partial u} = 1$$

$$\frac{\partial u}{\partial w_{ji}} = (y_j - d_j)z_i$$

# Day2

### Section 1: 勾配消失問題について

シグモイド関数は微分値が最大で0.25なため、層が深くなっていくと勾配が小さくなっていく。これを勾配消失問題という。ReLU関数では勾配消失問題を回避できる。現在最もよく使われている活性化関数はReLU関数である。

重みの初期値設定手法には主にふたつある。

- Xavier
  - 重みの要素を、前の層のノード数の平方根で除算した値
  - ReLU, sigmoid, tanh関数に対して用いる

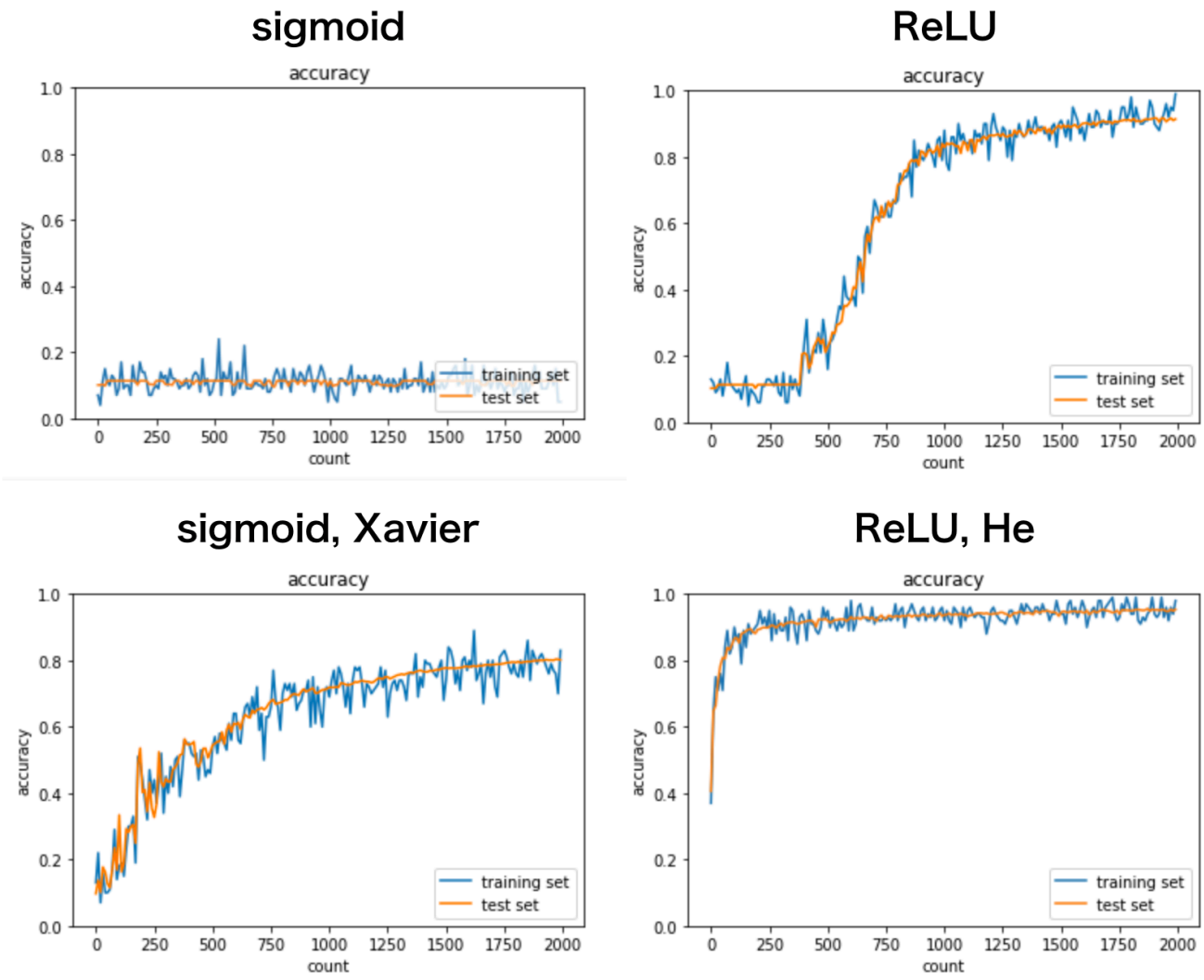
- He
  - 重みの要素を、前の層のノード数の平方根で除算した値に対し $\sqrt{2}$ を掛け合わせた値
  - ReLU関数に対して用いる

さらに、バッチ正規化という、ミニバッチ単位で入力値のデータの偏りを抑制する手法を用いると良いとされている。勾配消失が起きづらくなるのとともに計算の高速化の効果を孕む。

$$\mu_t = \frac{1}{N_t}\sum_{i=1}^{N_t} x_{ni} \quad {\sigma_t}^2 = \frac{1}{N_t}\sum_{i=1}^{N_t}(x_{ni} - \mu_t)^2 \quad \hat{x_{ni}} = \frac{x_{ni} - \mu_t}{\sqrt{{\sigma_t}^2 + \theta}}$$

$$y_{ni} = \gamma \hat{x_{ni}} + \beta$$

以下、中間層に用いる活性化関数と初期化手法を変化させた場合のMNISTに対する分類精度を比較したプロットである。




## Section 2: 学習率最適化手法について

- Momentum
  - $$V_t = \mu V_{t-1} - \epsilon \nabla E$$
  - $$w^{(t+1)} = w^{(t)} + V_t$$
  - 誤差をパラメータで微分したものと学習率の積を減算した後、現在の重みに前回の重みを減算した値と慣性の積を加算する
- AdaGrad
  - $$h_0 = \theta$$
  - $$h_t = h_{t-1} + (\nabla E)^2$$

- $$w^{(t+1)} = w^{(t)} - \epsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E$$
    - 誤差をパラメータで微分したものと再定義した学習率の積を減算する

- RMSProp
    - $$h_t = \alpha h_{t-1} + (1 - \alpha)(\nabla E)^2$$
    - $$w^{(t+1)} = w^{(t)} - \epsilon \frac{1}{\sqrt{h_t} + \theta} \nabla E$$
    - 誤差をパラメータで微分したものと再定義した学習率の積を減算する

- Adam
    - モメンタムの、過去の勾配の指数関数的減衰平均
    - RMSPropの、過去の勾配の2乗の指数関数的減衰平均
    - 上記をそれぞれ孕んだ最適化アルゴリズムである

## Section 3: 過学習について

過学習とはテスト誤差と訓練誤差とで学習曲線が乖離することである。原因としては、パラメータの数が多い、値が適切でない、ノードが多いなどと言ったものが挙げられる。ネットワークの自由度を制約する正則化という手法を用いて過学習を抑制することが求められる。

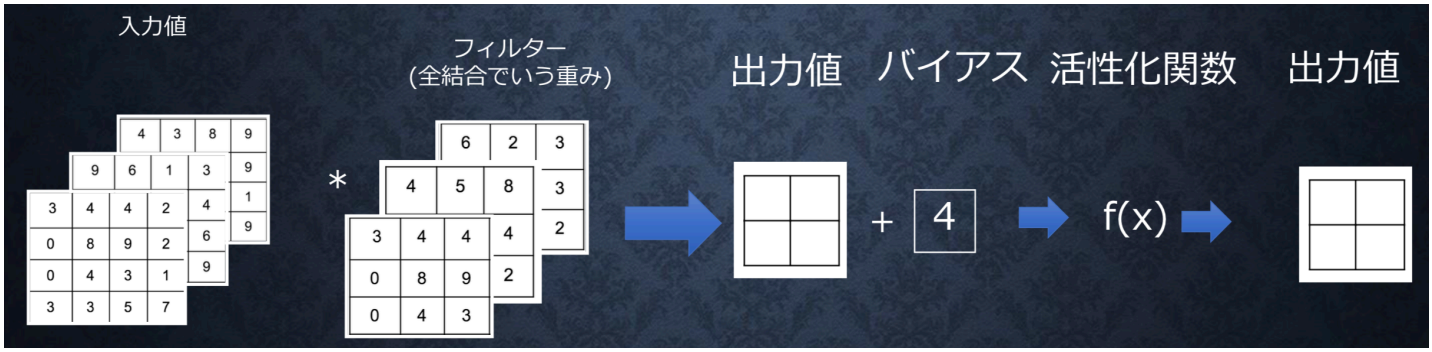
$$E_n(\mathbf{w}) + \frac{1}{p}\lambda ||x||_p$$

$$||x||_p = (|x_1|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$$

以上のように誤差関数に $p$ ノルムを加えることをLp正則化と呼ぶ。$p = 1$ の場合をLasso、$p = 2$ の場合をRidgeとも呼ぶ。 また、ドロップアウトという、ランダムにノードを削除して学習させる手法も多く用いられる。

## Section 4: 畳み込みニューラルネットワークの概念

CNNでは以下の図のようにフィルターを入力のテンソルに対して走査して要素ごとの積の和を取る。それにバイアスを足し、活性化関数に入力すると畳み込み層の出力値が得られる。




さらに、CNNに固有の概念としてパディングとストライドがある。パディングは入力のテンソルの周りに0などの固定値を追加することであり、ストライドはフィルターを走査する際のステップ幅のことである。

CNNではプーリング層という対象領域のMax値または平均値を取得するような層が用いられることもある。

$$\text{output image size} = \frac{n + 2p - f}{s} + 1$$

ただし、$n$ は入力画像のサイズ、$f$ はフィルターサイズ、$p$ はパッディングの大きさ、$s$ はストライド幅である。

### Section 5: 最新のCNN

AlexNetとは2012年に開かれた画像認識コンペティションImageNetにて2位に大差をつけて優勝したモデルである。5層の畳み込み層およびプーリング層など、それに続く3層の全結合層から構成される。サイズ4,096の全結合層の出力にドロップアウトを用いている。

## Day3

### Section 1: 再帰型ニューラルネットワークの概念

RNNとは、時系列データに対応可能な、ニューラルネットワークである。ここで、時系列データとは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のことである。

$$u^t = W_{(in)}x^t + Wz^{t-1} + b \; z^t = f(W_{(in)}x^t + Wz^{t-1} + b) \; v^t = W_{(out)}z^t + c$$

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

BPTT (Back Propagation Through Time) とは、RNNにおいてのパラメータ調整方法の一種であり、誤差逆伝播法の一種である。

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

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

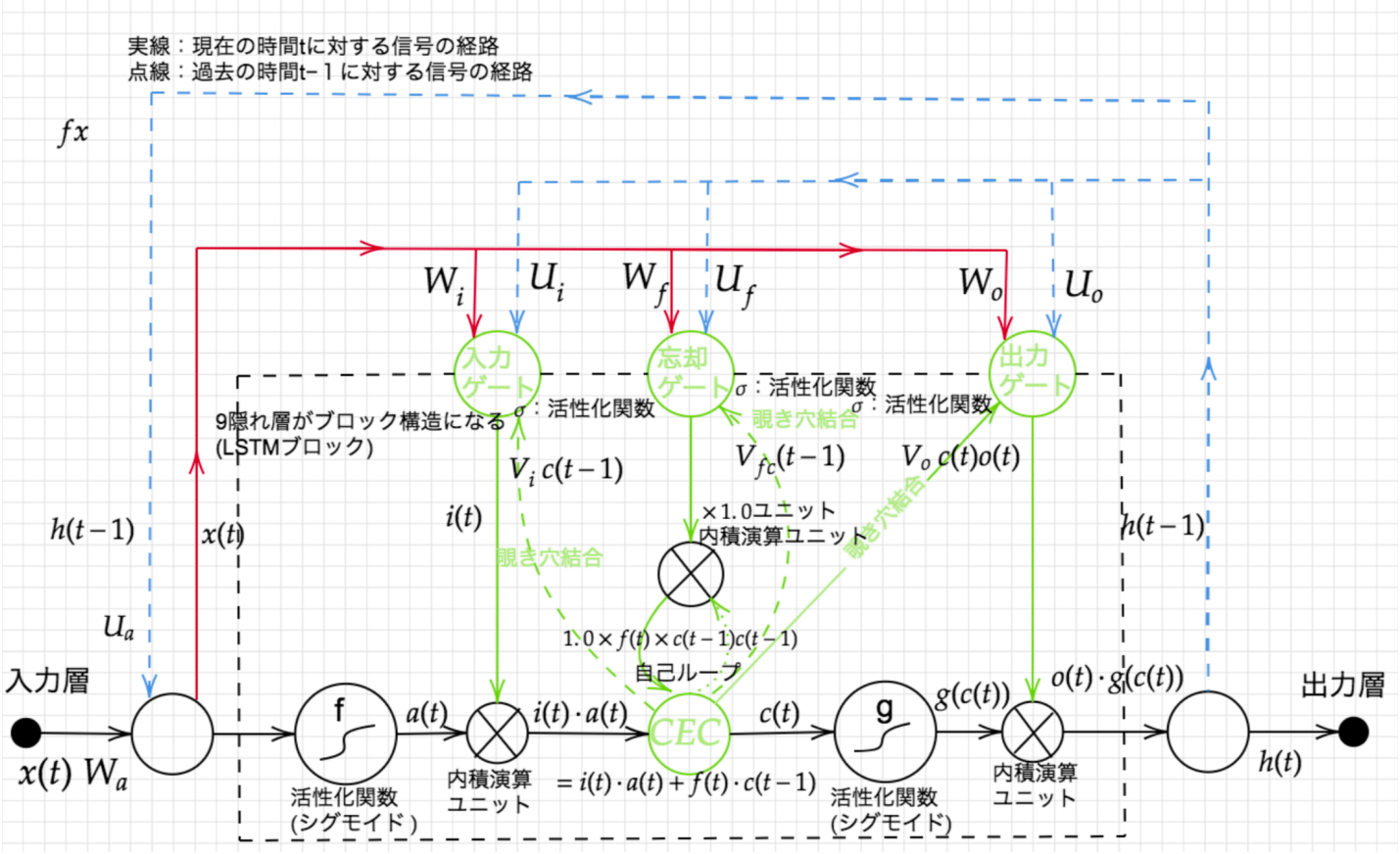
$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial u^t}\left[\frac{\partial u^t}{\partial W}\right]^{\mathrm{T}} = \delta^t[z^{t-1}]^{\mathrm{T}}$$

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

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

## Section 2: LSTM

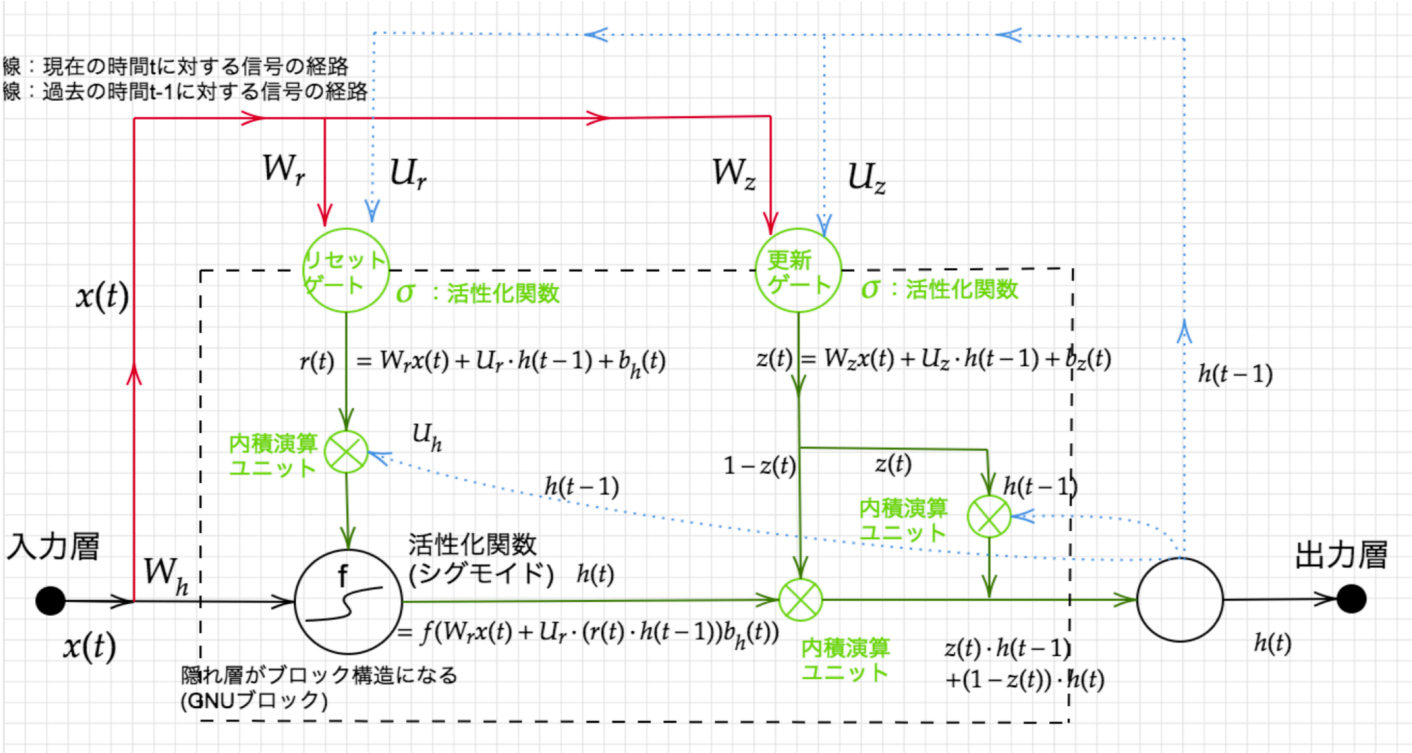
単純なRNNでは時系列を遡るほど勾配が消失してしまい、長い時系列の学習が困難になる。構造自体を変えて勾配消失問題を解決したのがLSTMである。




CEC (Constant Error Carousel) 構造を用いて勾配が常に1になるようにして勾配消失問題を解決した。ただし、それでは時間依存度に関係なく重みが一律になり、学習特性がなくなる。入力・出力重み衝突が問題となる。そこで、入力ゲートと出力ゲートを用意した。さらに、CECは過去の情報が全て補完されており、過去の情報が入らなくなっても補完され続けるため、忘却ゲートという機構を導入した。CEC自身の値をゲート制御に影響させるために覗き穴結合を追加したネットワークを用いることもある。

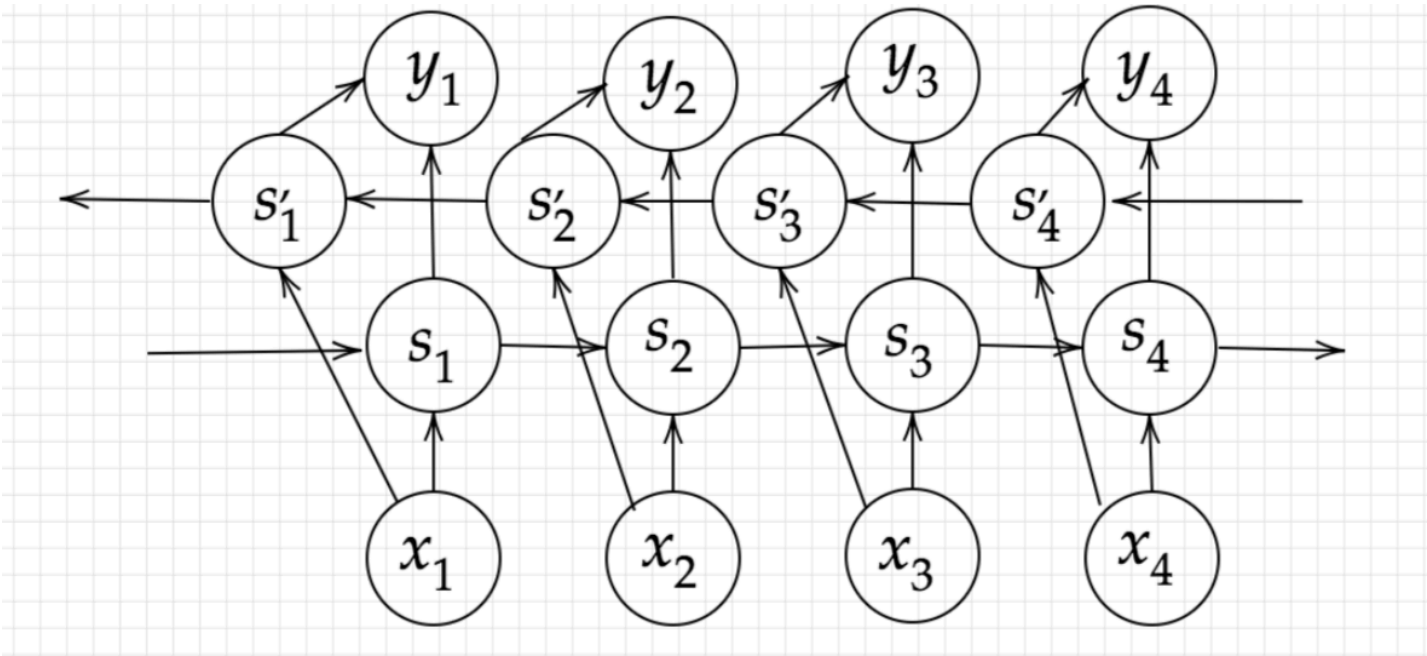
## Section 3: GRU

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。




## Section 4: 双方向RNN

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。文章の推敲や、機械翻訳などのタスクで用いられる。双方向RNNでは、順方向と逆方向に伝播したときの中間層表現をあわせたものが特徴量となる。




## Section 5: Seq2Seq

Seq2Seqはencoder-decoderモデルの一種。機械対話や、機械翻訳などに使用されている。 encoder RNNではまずtokenizerで文章を単語等のトークンごとに分割し、IDに変換する。次にIDからそのトークンを表す分散表現へembeddingし、ベクトルを順番にRNNに入力していく。vec1をRNNに入力し、hidden stateを出力。このhidden stateと次の入力vec2をまたRNNに入力し、hidden stateを出力するという流れを繰り返す。最後のvecを入れたときのhidden stateをfinal stateとしてとっておく。このfinal stateがthought vectorと呼ばれ、入力した文の意味を表すベクトルとなる。Decoder RNNでは、Encoder RNNのfinal state (thought vector) から、各tokenの生成確率を出力していく。

Seq2Seqの課題は一問一答しかできない点である。問いに対して文脈も何もなく、ただ応答が行われ続ける。それに対し、HRED (Hierarchical Recurrent Encoder-Decoder) では過去 $n-1$ 個の発話から次の発話を生成する。ただし、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。

VHRED (Latent Variable Hierarchical Recurrent Encoder-Decoder) はHREDに、VAE (Variational Auto Encoder) の潜在変数の概念を追加したもの。

### Section 6: Word2vec

学習データからボキャブラリを作成するとき、単純には各単語をone-hotベクトルにすればよい。しかし、これではボキャブラリ×ボキャブラリだけの重み行列が必要になる。Word2Vecでは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となり、ボキャブラリ×任意の単語ベクトル次元で重み行列を作ることができる。

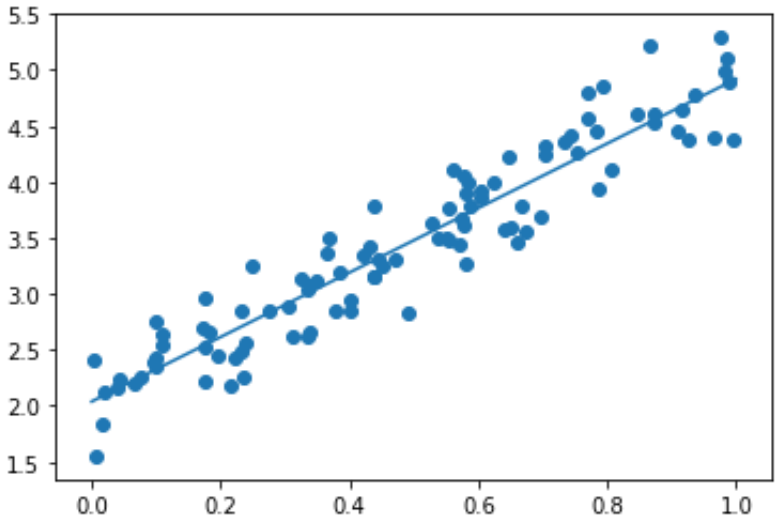
### Section 7: Attention Mechanism

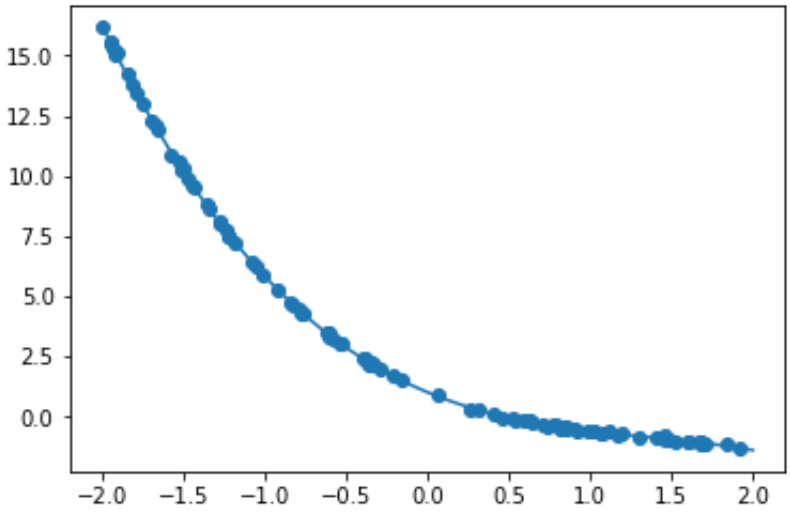
Seq2Seqでは長い文章への対応が難しい。何単語の系列に対しても固定次元のベクトルの中にエンコードしなくてはいけない。そこで、「入力と出力のどの単語が関連しているのか」の関連どを学習する仕組みとしてattention mechanismを導入。

## Day4

### Section 1: TensorFlowの実装演習

`tf.constant()` で定数、 `tf.placeholder()` でプレースホルダー、 `tf.Variable()` で変数を定義する。 誤差関数、SGDは `tf.reduce_mean()` 、 `tf.train.GradientDescentOptimizer()` を用いる。Adamを使いたいときは `tf.train.AdamOptimzer()` を用いる。これらを用いて線形回帰、非線形回帰を行なった結果は以下の通り。




分類問題を解きたいときは `tf.nn.softmax()` を用いる。交差エントロピーは `-tf.reduce_sum(d * tf.log(y), reduction_indices=[1])` のようにして求める。CNNを構成するときは `tf.nn.conv2d()` や `tf.nn.relu()`, `tf.nn.max_pool` などを組み合わせる。ドロップアウトには `tf.nn.dropout()` を用いる。

Kerasを用いるとさらに単純にネットワークを構成することができる。例えば、`keras.models.Sequential` に対してそれぞれのレイヤーを `.add` していけばよい。

```
model = Sequential()

model.add(Conv2D(32, (3, 3), padding='same',input_shape=x_train.shape[1:]))
model.add(Activation('relu'))
model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(10))
model.add(Activation('softmax'))
```

その後、誤差関数、最適化手法、評価指標を指定してモデルをコンパイルする。訓練は `.fit` を実行すれば良い。

```
# コンパイル
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'
])

#訓練
history = model.fit(x_train, d_train, epochs=20)
```

単純なRNNも同様な手続きで構築できる。

```
model = Sequential()

model.add(SimpleRNN(units=16,
                return_sequences=True,
                input_shape=[8, 2],
                go_backwards=False,
                activation='relu',
                # dropout=0.5,
                # recurrent_dropout=0.3,
                # unroll = True,
            ))
# 出力層
model.add(Dense(1, activation='sigmoid', input_shape=(-1,2)))
model.summary()
model.compile(loss='mean_squared_error', optimizer=SGD(lr=0.1), metrics=['accuracy
'])
# model.compile(loss='mse', optimizer='adam', metrics=['accuracy'])

history = model.fit(x_bin, d_bin.reshape(-1, 8, 1), epochs=5, batch_size=2)
```

## Section 2: 強化学習

強化学習とは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である。

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であるが、この過程は成り立たないとする。不完全な知識をもとに行動しながら、データを収集し、最適な行動を見つけていく。価値ある行動の探索と過去の経験の利用はトレードオフの関係にある。

強化学習と通常の教師あり、教師なし学習との違いは、目標の違いにある。通常の学習ではデータに含まれるパターンを見つけ出したり、そのデータから予測することが目標だが、一方強化学習では、優れた方策を見つけることが目標となる。

強化学習で用いられる手法として以下のふたつに着目する。

- Q学習 - 行動価値関数を、行動するごと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、状態と行動を組み合わせた価値に注目する場合は行動価値関数を用いる。

方策関数とは、方策ベースの強化学習手法において、ある状態でどの行動をとるのかの確率を与える関数のこと。学習は方策反復法を用いて行われる。

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

ここで $J$ とは方策の良さを表す量であり、平均報酬や割引報酬和として定義される。行動価値関数 $Q(s,a)$ との関係性から方策勾配定理が成り立つ。

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