# MACHINE LEARNING

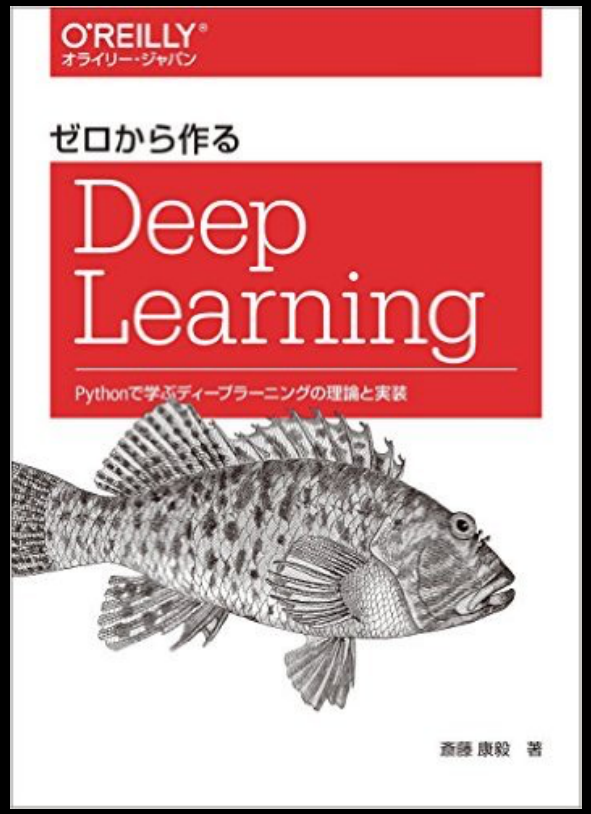
T314004　磯谷　将

# 今回の勉強会の目的と内容

- 機械学習の基礎の基礎、ニューラルネットワークについてどういう動きをしているか理解を深める
- ライブラリを上手にカスタマイズしてほしい
- 先輩、後輩同士の親睦を深める！

# 参考書

- ゼロから作るDeepLearning Pythonで学ぶディープラーニングの理論と実装
- O'REILLY Japan から出ています
- 時間はかかるけど理解しやすい



# 機械学習とは

- データから反復的に学習し、そこに潜むパターンを見つけること
- パターンに従って将来を予測することもできる
- 主要な技法に関しては、決定機学習、サポートベクターマシン(SVM)、クラスタリング、ベイジアンネットワーク、強化学習、ニューラルネットワーク、強化学習.....etc

# パーセプトロン①

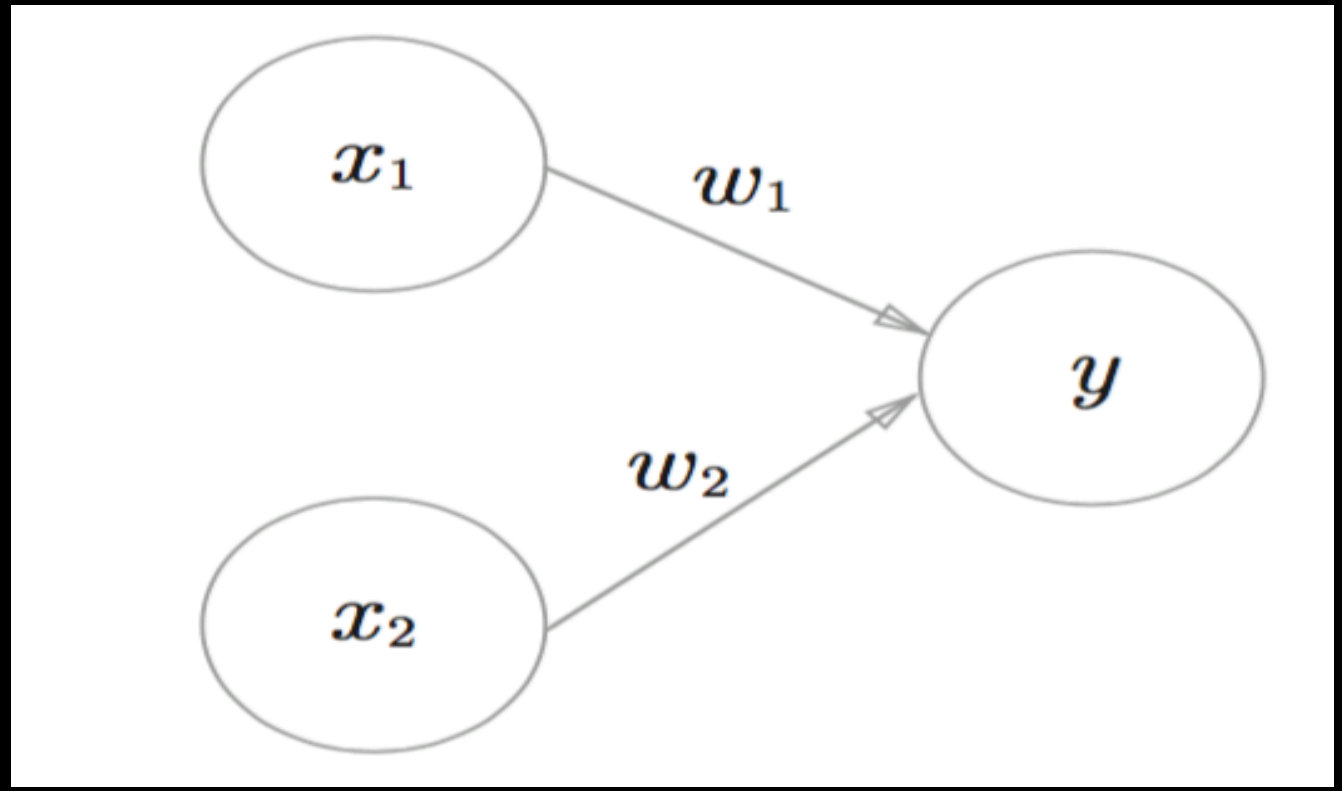
- ローゼンブラットというアメリカの研究者によって1957年に考案されたアルゴリズム
- ニューラルネットワークの起源となるアルゴリズム
- そのためこの仕組みを学ぶことはニューラルネットワーク、ディープラーニングへと進む上で重要な考え方を学ぶことになる

# パーセプトロン②

- 複数の信号を入力として受け取り、1つの信号(電流や川のような『流れ』を持つもののイメージ)を出力する。
- 電流のように導線を流し、電子を送り出すように、パーセプトロンの信号を流れを作り、情報を伝達する
- ただここの信号は0か1で表される。

# パーセプトロンの例

- 以下に2つの信号を入力として受け取るパーセプトロンの例を示す



# 先ほどのパーセプトロンの計算式

- $y = \begin{cases} 0, & (w1x1 + w2x2) \leq \theta \\ 1, & (w1x1 + w2x2) > \theta \end{cases}$

# パーセプトロンを使ってみよう①

- パーセプトロンを使って簡単な問題を考えてみる
- 論理回路を題材とする。
- AND、NAND、OR回路の真理値表わかりますか？
- ハンズオンでーす！まずはAND回路をPythonで実装しましょう！

# パーセプトロンを使ってみよう②

- $y=\begin{cases}0, & (b+w1x1+w2x2) \leq \theta \\ 1, & (b+w1x1+w2x2) > \theta\end{cases}$
- $b$をバイアスと呼び$w1$や$w2$を重みと言う。
- バイアスとは、出力信号を1にするための閾値の調整をする値
- ではこれを踏まえて、先ほどのプログラムを少し弄りましょう！

# パーセプトロンを使ってみよう③

- ではNAND回路、OR回路を先ほどの例に従って実装してみてください！

# なぜこの例を使ったか

- パーセプトロンの構造は全て同じなのだが、パラメータの値の違いだけで、出力される値が変わってくることを確かめるにはもってこいだから。

# パーセプトロンの限界①

- XORゲート(排他的論理和)がこれまでみてきた排他的論理和では実現が不可能だからである。
- ではなぜか？？
- 以下の式をグラフに書いて見ます。
- $y=\begin{cases}0, & (-0.5+x1+x2) \leq \theta \\ 1\ , & (-0.5+x1+x2) > \theta\end{cases}$

# パーセプトロンの限界②

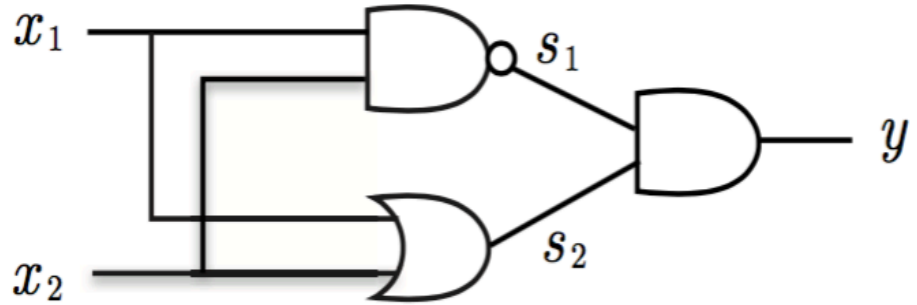
- XORになると直線によって出力を分けることができない
- パーセプトロンの限界はこの直線で分けれないところにある。
- このように直線で分けれる領域を線形な領域といい、分けれない曲線による領域を非線形な領域といいます。

# 多層パーセプトロン

- パーセプトロンは層を重ねることができるという特徴があります。
- 層を重ねることでパーセプトロンでXORを表現できるのです！！
- 一旦層を重ねるということはどういうことかという説明は後にしてXORゲートの問題を別の視点から考えていこう！

XOR回路作ってください！！

# ちなみに、、こんな感じ




| $x_1$ | $x_2$ | $s_1$ | $s_2$ | $y$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 0 |

# パーセプトロン　まとめ

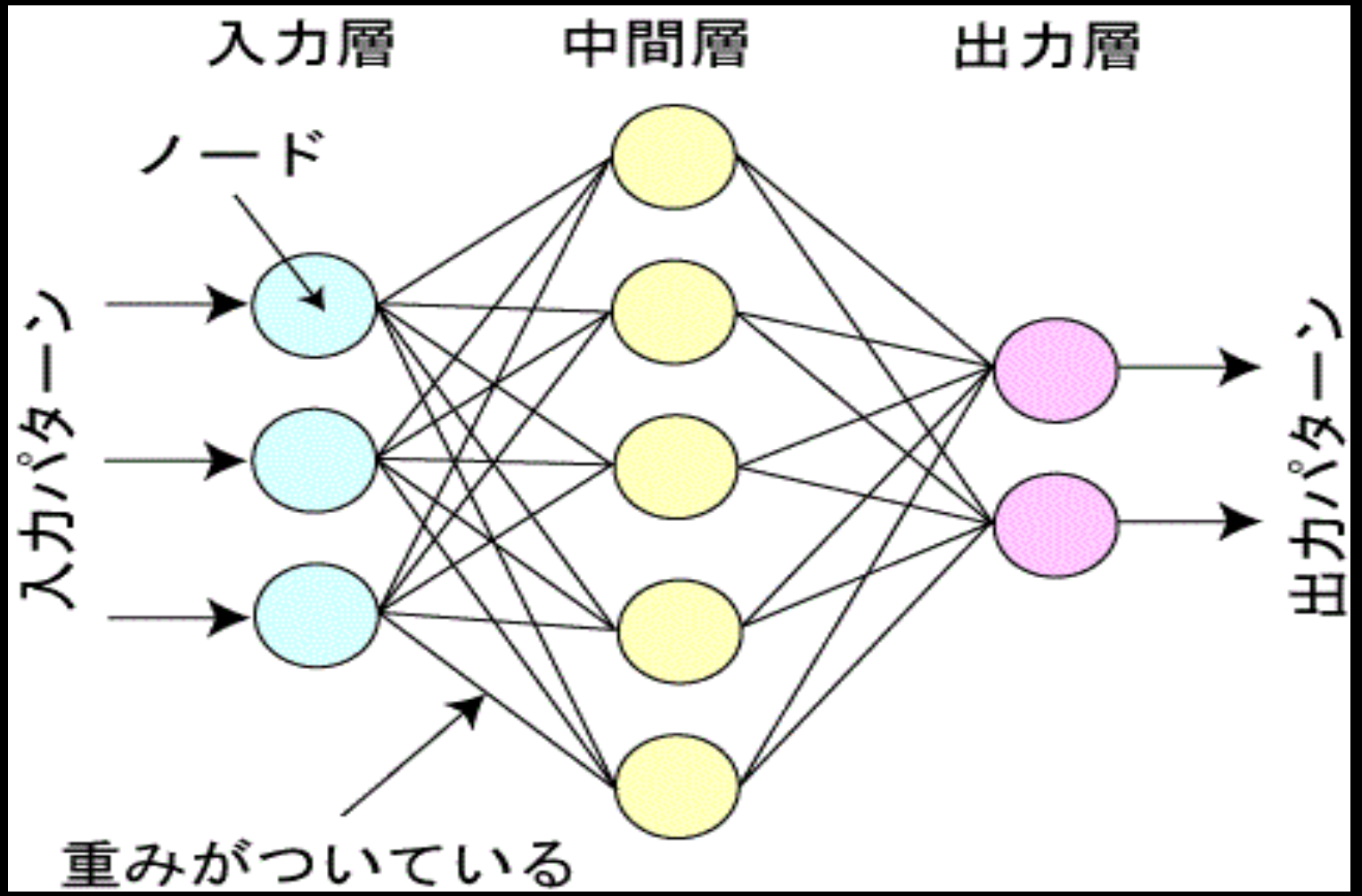
- 入力を与えたら、決まった値が出るシンプルなアルゴリズム。
- 層を重ねることで、線形だけでなく、非線形な領域も表現できる。
- 理論上、コンピュータを表現できる。

# ニューラルネットワーク

- 人間の脳の神経回路の仕組みを模したモデル
- 基本的にはパーセプトロンの考え方と一緒
- 適切な重みパラメータをデータから自動で学習できないのがパーセプトロンの弱点

ニューラルネットワーク

# ニューラルネットワークの例



# 活性化関数

- 先ほどのXOR回路のように非線形な領域なもんだらけ。
- 非線形な領域を無理に線形な領域と同じように計算しては、出力が変わらないという問題が発生してしまう

**活性化関数**

# 活性化関数

- ステップ関数
- シグモイド関数
- ReLU関数

# ステップ関数

- 入力が0を超えたら1を出力し、それ以外は0を出力すると言うさっき見たパーセプトロンと考え方は一緒

# シグモイド関数

- ニューラルネットワークでよく用いられる活性化関数の一つ
- 計算式は以下の通り
- $f(x) = \frac{1}{1+\exp(-x)}$

# 比較

- 滑らかさの違い
- 両者とも入力が大きくなるにつれて1に近く

# RELU関数

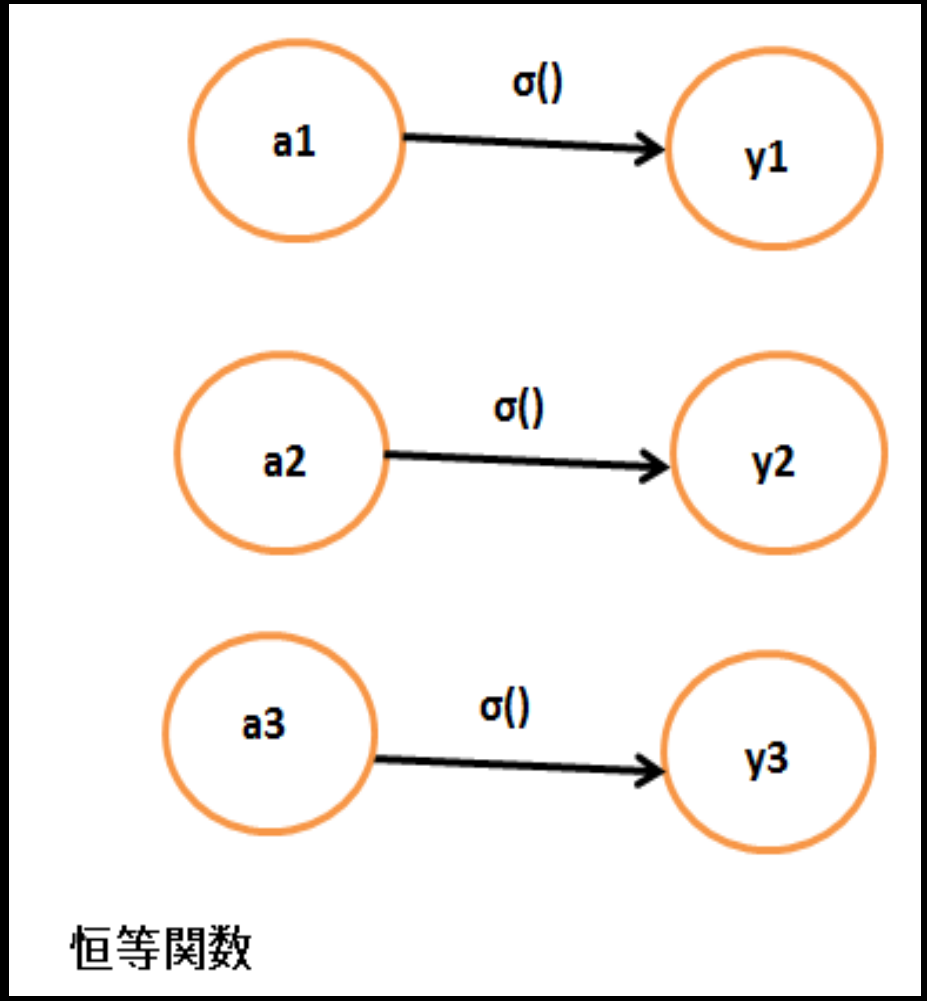
- ReLU関数ですww
- 最近はこっちが主流ですww
- 入力値が0を超えていればその入力をそのまま出力し、0以下なら0を出力する

# 出力層の設計

- 機械学習の問題によって出力層の活性化関数を分けなくてはいけない
- 機械学習の問題には分類問題と回帰問題がある
- 分類問題ではソフトマックス関数
- 回帰問題では恒等関数

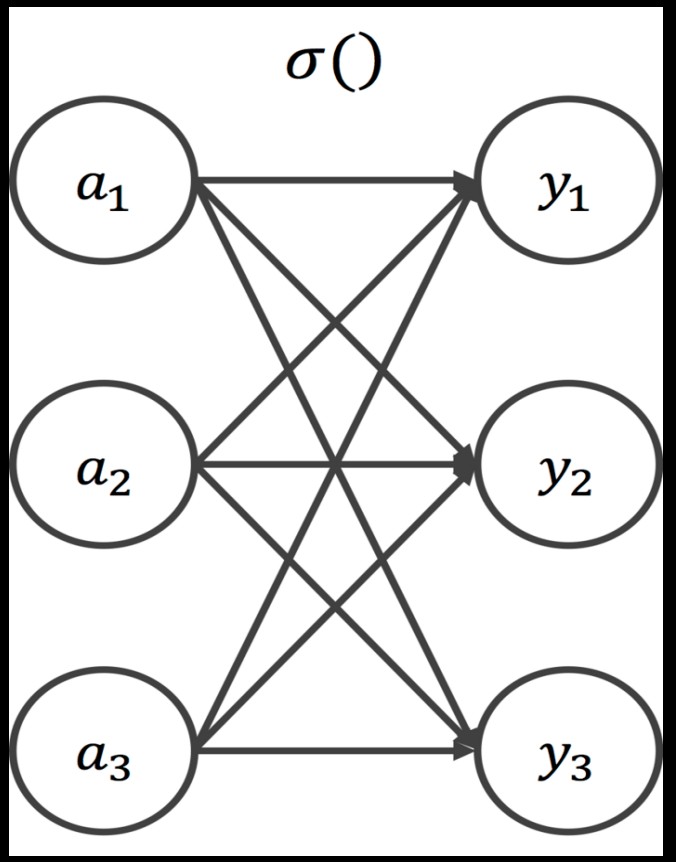
# 恒等関数

- 入力をそのまま出力する関数



# ソフトマックス関数

- 全ての入力信号からの矢印の結びつきがある関数
- $y_{\mathrm{k}} = \dfrac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$

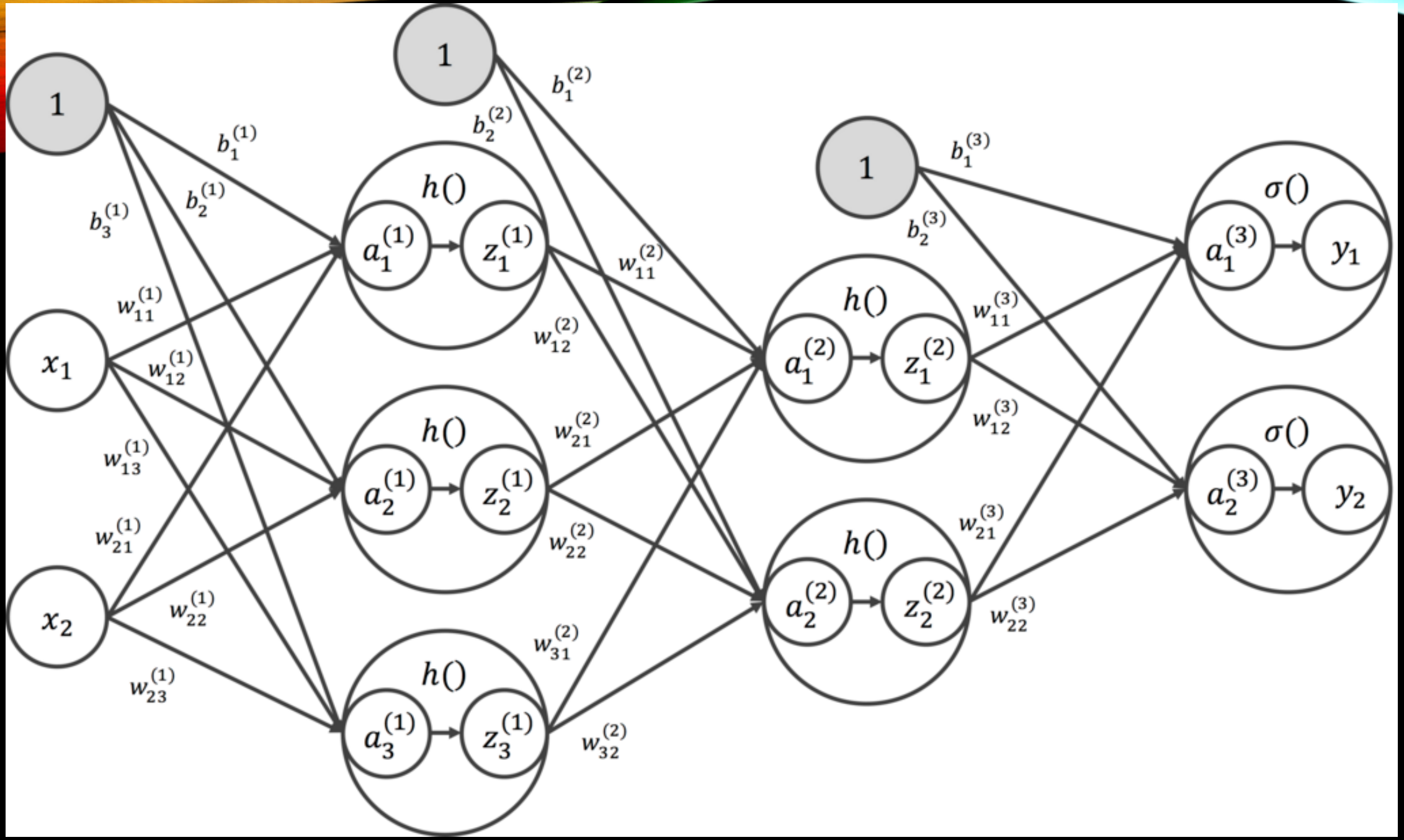


# ソフトマックス関数の実装

- ソフトマックス関数を計算する際は、オーバーフローに注意しなくてはならない
- 大きな値通しを計算してしまうと不安定な計算になりエラーが起きてしまう
- 入力信号の最大値から入力される値を入れると解決する

# ソフトマックス関数の特徴

- 出てくる値見たく、問題を確率的に対処することができる
- しかし、実際の問題では指数関数の計算はコンピュータにそれなりの負荷を与えるので、推論フェーズでは省略させることは多いが学習フェーズに移る際には必要となってくる(損失関数やら、交差エントロピーとか。。。）


1
1
1
$x_1$
$x_2$
$b_1^{(1)}$
$b_2^{(1)}$
$b_3^{(1)}$
$w_{11}^{(1)}$
$w_{12}^{(1)}$
$w_{13}^{(1)}$
$w_{21}^{(1)}$
$w_{22}^{(1)}$
$w_{23}^{(1)}$
$b_1^{(2)}$
$b_2^{(2)}$
$w_{11}^{(2)}$
$w_{12}^{(2)}$
$w_{21}^{(2)}$
$w_{22}^{(2)}$
$w_{31}^{(2)}$
$w_{32}^{(2)}$
$b_1^{(3)}$
$b_2^{(3)}$
$w_{11}^{(3)}$
$w_{12}^{(3)}$
$w_{21}^{(3)}$
$w_{22}^{(3)}$
h()
$a_1^{(1)}$
$z_1^{(1)}$
h()
$a_2^{(1)}$
$z_2^{(1)}$
h()
$a_3^{(1)}$
$z_3^{(1)}$
h()
$a_1^{(2)}$
$z_1^{(2)}$
h()
$a_2^{(2)}$
$z_2^{(2)}$
σ()
$a_1^{(3)}$
$y_1$
σ()
$a_2^{(3)}$
$y_2$

# バッチ処理

- 一定量のデータを集め、一括処理するための方法
- 画像１枚あたりの処理時間を短縮するために使われる

# 機械学習　学習手法①

- 訓練データとテストデータに分けられる
- 訓練データ→学習を行い、最適なパラメータを探索
- テストデータ→その訓練したモデルの実力の評価

# 機械学習　学習手法②

なぜ、二つにデータを分けるのか？

汎化能力の獲得のため

# ニューラルネットワーク　学習手法①

- あなたは今どれだけ幸せですか？
- ニューラルネットワークでは、ある『一つの指標』によって現在の状態を表しています
- そしてこの指標を手掛かりに最適な人生を探索するように、最適なパラメータを探索します

**損失関数**

# ニューラルネットワーク　学習手法②

- 損失関数を求める際には2乗和誤差や交差エントロピー誤差などが用いられます
- 損失関数はニューラルネットワークの性能の悪さを示す指標
- 損失関数にマイナスをかけた場合は性能の良さを指標とすることができる
- 損失関数の値が小さくなるように重みパラメータを更新していくのがニューラルネットワークの学習

# ２乗和誤差

- ニューラルネットワークの出力と正解となる教師データの各要素の差の2乗を計算し、その総和を計算する
- $E = \frac{1}{2}\sum_k (y_k - t_k)^2$

# 交差エントロピー誤差

- $E = -\sum_k t_k \, log y_k$
- 正解ラベルが1に対応する出力の自然対数を計算する

# ミニバッチ学習

- バッチ処理をするとはいえ、ビッグデータとなった時いくら全てのデータを対象として計算することは現実的に厳しい
- そこでデータの中から、一部を選び出し、そのデータを全体の『近似』として利用し、学習させていく

# 勾配

- 全ての変数の偏微分をベクトルとしてまとめたもの
- 重みパラメータを更新する際に、この勾配を利用して、勾配方向に重みの値を更新する作業を繰り返す。

**勾配法**

# 勾配法

- 勾配降下法と勾配上昇法がある
- 微小な値を与えた時の差分によって微分を求めることを数字微分といい、これにより重みパラメータの勾配を求めることができる。

# 勾配法

- 重みのパラメータのっこうは意を求める際に誤差逆伝播法というものがある。
- これについて今日は解説しないが、これを用いるともっと処理が早くなる

# 紹介

- https://book.mynavi.jp/manatee/series/detail/id=65670
- http://qiita.com/jintaka1989/items/bfcf9cc9b0c2f597d419