現場で使えるディープラーニング基礎講座

DAY3

SkillUP AI

前回の復習

- 前回は何を学びましたか？

学習の最適化

目次

勾配法の学習を最適化させる方法

確率的勾配降下法

- 確率的勾配降下法(Stochastic Gradient Descent, 以下SGD)とは、無作為に選びだしたデータを用いてパラメータを更新していく勾配降下法のこと。
- ミニバッチ学習を行うと、SGDで解いていることになる。
- ミニバッチ学習の特徴については、DAY2の資料を参照。

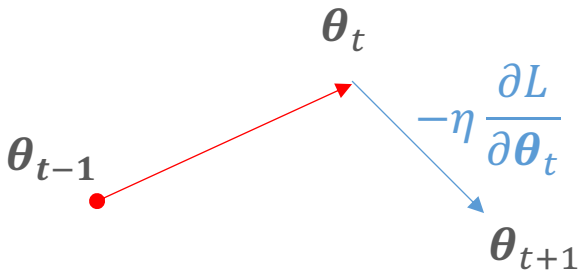
SGDの更新式

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\partial L}{\partial \boldsymbol{\theta}_t}$$

$\boldsymbol{\theta}$: パラメータ
L : 損失
η : 学習率

ここでのパラメータ$\boldsymbol{\theta}$は、ネットワーク全体の重みwバイアスbなどを並べたベクトルを表す。
また、tは時点を表す。

SGD



勾配法の学習を最適化させる方法

- SGDの学習をより効率的に行う方法があり、それらは1次の方法と2次の方法に分類される。
- 一般的に、DNNでは1次の方法が用いられるが、2次の方法を用いている事例もある。
  - 1次の方法
    - 1階の微分のみを用いる。
    - 学習率を事前に決める必要がある。
  - 2次の方法
    - 1階の微分と2階の微分の両方を用いる。
    - 1次の方法のように学習率を事前に決める必要がない。
    - 2階の微分が学習率と同様の働きをする。
    - 2階の微分の行列(ヘッセ行列)の逆行列を計算する必要があるため、計算負荷が大きい。

勾配法の学習を最適化させる方法

1次	Momentum	物理法則に準じる動きをする。進む方向と勾配方向が同じであれば、移動量が大きくなる。
	Nesterov AG	Momentumの改良版。次の位置を予測し、その位置の勾配も加味する。
	AdaGrad	勾配の2乗を積み上げていくことで、見かけの学習係数を小さくしていく。
	RMSProp	AdaGradの改良版。勾配の2乗の移動平均を用いることで、見かけの学習率を変化させる。
	AdaDelta	AdaGradの改良版。勾配の2乗の移動平均と更新量の2乗の移動平均を用いることで、見かけの学習率を変化させる。
	Adam	RMSPropとMomentumを組み合わせたような方法
2次	Newton法	ヘッセ行列を使用
	共役勾配法	共役方向によって逆ヘッセ行列の計算を効率化する
	BFGS	各ステップでのヘッセ行列の近似を改良したもの
	L-BFGS	省メモリ化したBFGS

Momentum

- 物理法則に準じる動きをする。
- 真っ直ぐに進んでいるときは移動量が大きくなり、曲がりながら進んでいるときは緩やかに曲がる。

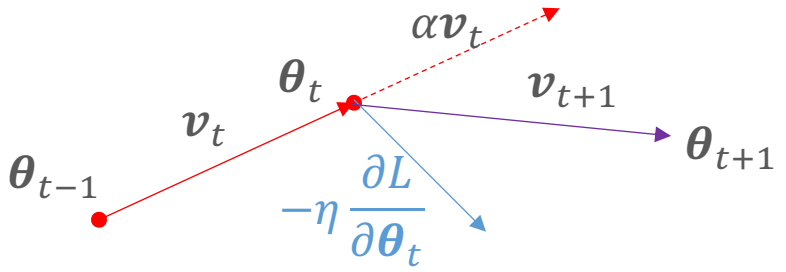
Momentumの更新式

$$\boldsymbol{v}_{t+1} = \alpha \boldsymbol{v}_t - \eta \frac{\partial L}{\partial \boldsymbol{\theta}_t}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1}$$

$\boldsymbol{\theta}$: パラメータ
L : 損失
η : 学習率
α: モーメンタム係数(0以上1未満)

Momentum



1期前の更新量$\boldsymbol{v}_t$を利用する

Nesterov Accelerated Gradient

- Momentumの改良版。
- 仮の1期先の位置を求め、その位置の勾配を利用する。

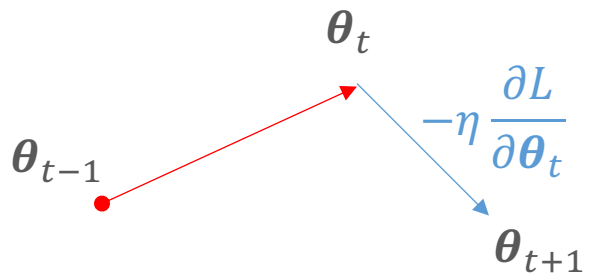
Nesterov Accelerated Gradientの更新式

$$\boldsymbol{v}_{t+1} = \alpha \boldsymbol{v}_t - \eta \frac{\partial L}{\partial(\boldsymbol{\theta}_t + \alpha \boldsymbol{v}_t)}$$

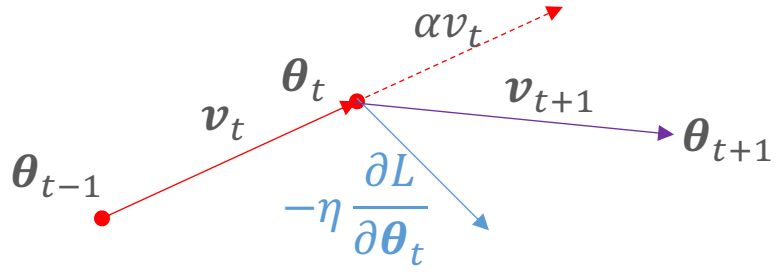
$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{v}_{t+1}$$

$\boldsymbol{\theta}$: パラメータ
L : 損失
η : 学習率
α: モーメンタム係数(0以上1未満)

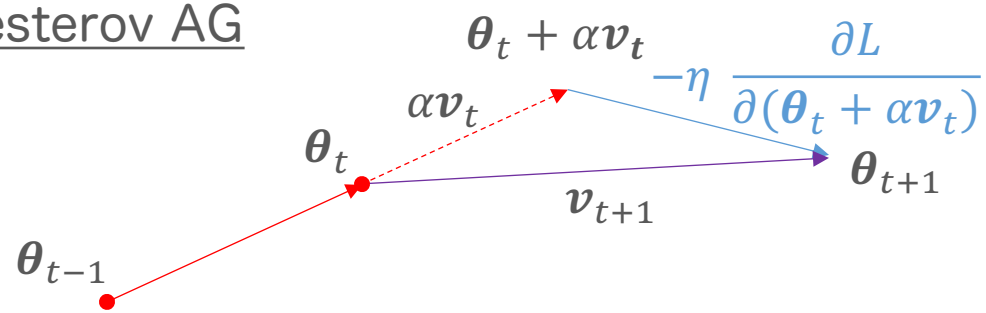
SGD



Momentum



Nesterov AG



暫定的な1期先の勾配を利用することにより、学習を加速させようというアイデア

Nesterov Accelerated Gradient

- Nesterov AGは、定義式通りに計算しようとすると、暫定的1期先の勾配が必要になり、計算が煩雑になる。
- Yoshua Bengioらが効率よく計算する方法を提案しており、kerasやchainerはこの方法を採用している。
  - https://arxiv.org/pdf/1212.0901v2.pdf

Nesterov Accelerated Gradient

Nesterov Accelerated Gradientの更新式(実装版)

$\Theta_{t+1} = \boldsymbol{\theta}_{t+1} + \alpha \boldsymbol{v}_{t+1}$とおくと、
$\Theta_t = \boldsymbol{\theta}_t + \alpha \boldsymbol{v}_t$なので、定義式は以下のように変形できる

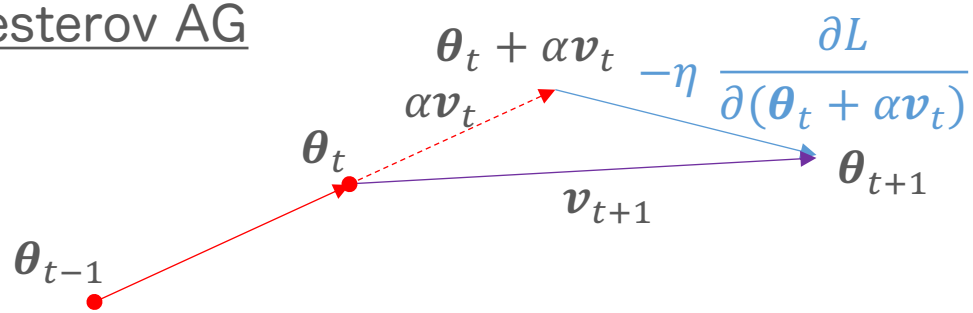
$$\boldsymbol{v}_{t+1} = \alpha \boldsymbol{v}_t - \eta \frac{\partial L}{\partial \Theta_t}$$

$$\begin{aligned}\Theta_{t+1} &= \Theta_t - \alpha \boldsymbol{v}_t + \alpha \boldsymbol{v}_{t+1} + \boldsymbol{v}_{t+1} \\ &= \Theta_t - \alpha \boldsymbol{v}_t + (\alpha + 1)\boldsymbol{v}_{t+1} \\ &= \Theta_t - \alpha \boldsymbol{v}_t + (\alpha + 1)\left(\alpha \boldsymbol{v}_t - \eta \frac{\partial L}{\partial \Theta_t}\right) \\ &= \Theta_t + \alpha\alpha \boldsymbol{v}_t - (\alpha + 1)\left(\eta \frac{\partial L}{\partial \Theta_t}\right) \\ &= \Theta_t + \alpha\left(\alpha \boldsymbol{v}_t - \eta \frac{\partial L}{\partial \Theta_t}\right) - \eta \frac{\partial L}{\partial \Theta_t}\end{aligned}$$

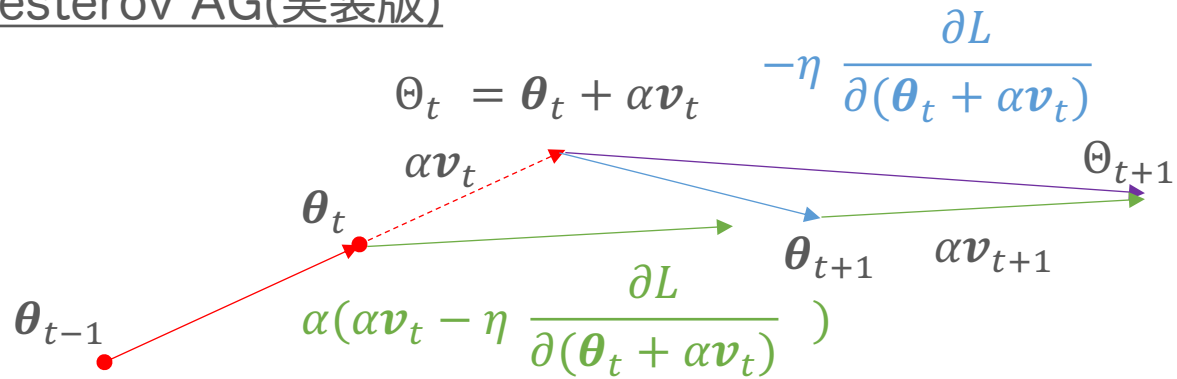
暫定的1期先のパラメータを更新するように変形することで、定義式と等価な計算を実現している

$\boldsymbol{\theta}$：パラメータ
L：損失
η：学習率
α: モーメンタム係数(0以上1未満)

Nesterov AG



Nesterov AG(実装版)



※Θもベクトル

$\boldsymbol{\theta}$を更新するのではなく、暫定的1期先のパラメータΘを更新していく

AdaGrad

- パラメータ毎に学習率を適応させる(adaptive)方法。
- 学習が進むにつれ、見かけの学習率が減衰していく。

AdaGradの更新式

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \frac{\partial L}{\partial \boldsymbol{\theta}_t} \odot \frac{\partial L}{\partial \boldsymbol{\theta}_t}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{1}{\varepsilon + \sqrt{\boldsymbol{h}_{t+1}}} \odot \frac{\partial L}{\partial \boldsymbol{\theta}_t}$$

見かけの学習率

$\boldsymbol{\theta}$: パラメータ
L : 損失
η : 学習率
ε : 計算を安定化させるための定数
$\odot$: アダマール積記号(同じ要素同士の掛け算)

$\boldsymbol{h}$はベクトルであることに注意
計算が進むにつれ、パラメータ毎の見かけの学習率が小さくなっていく

RMSProp

- AdaGradの改良版。
- 勾配の2乗の移動平均を用いて、見かけの学習率を変化させていく。
- ここでの移動平均とは、指数平滑化移動平均のこと。減衰率ρの割合で足していくことによって、過去の情報が指数関数的に薄れていく

RMSPropの更新式

$$\boldsymbol{h}_{t+1} = \rho \boldsymbol{h}_t + (1-\rho)\frac{\partial L}{\partial \boldsymbol{\theta}_t} \odot \frac{\partial L}{\partial \boldsymbol{\theta}_t}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{1}{\sqrt{\varepsilon + \boldsymbol{h}_{t+1}}} \odot \frac{\partial L}{\partial \boldsymbol{\theta}_t}$$

見かけの学習率

$\boldsymbol{\theta}$：パラメータ
L：損失
ρ：減衰率, 0.9など
η：学習率
ε：計算を安定化させるための定数
$\odot$：アダマール積記号(同じ要素同士の掛け算)

勾配の小さな領域にとどまると、$\boldsymbol{h}_{t+1}$が小さくなっていき、見かけの学習率が大きくなっていく。これにより、プラトーを抜け出しやすくなる。

AdaDelta

- AdaGradの改良版。
- 勾配の2乗の移動平均と、更新量の2乗の移動平均を用いて、見かけの学習率を変化させていく。
- 学習率ηを更新量の移動平均と置き換えることで、単位を揃えている。

AdaDeltaの更新式

$$\boldsymbol{h}_{t+1} = \rho \boldsymbol{h}_t + (1-\rho)\frac{\partial L}{\partial \boldsymbol{\theta}_t} \odot \frac{\partial L}{\partial \boldsymbol{\theta}_t}$$ 勾配の2乗の移動平均

$$\Delta \boldsymbol{\theta}_t = -\frac{\sqrt{\varepsilon + \boldsymbol{r}_t}}{\sqrt{\varepsilon + \boldsymbol{h}_{t+1}}} \odot \frac{\partial L}{\partial \boldsymbol{\theta}_t}$$ 更新量

見かけの学習率

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \Delta \boldsymbol{\theta}_t$$

$$\boldsymbol{r}_{t+1} = \rho \boldsymbol{r}_t + (1-\rho)\Delta \boldsymbol{\theta}_t \odot \Delta \boldsymbol{\theta}_t$$ 更新量の2乗の移動平均

$\boldsymbol{\theta}$：パラメータ
L：損失
ρ：減衰率, 0.95など
ε：計算を安定化させるための定数
$\odot$：アダマール積記号(同じ要素同士の掛け算)

更新量の2乗の移動平均が小さいほど見かけの学習率が小さくなるので、極小値付近では動きがゆっくりになる。

Adam

- RMSPropとMomentumを組み合わせたような方法。

Adamの更新式

$$\boldsymbol{m}_{t+1} = \rho_1 \boldsymbol{m}_t + (1-\rho_1)\frac{\partial L}{\partial \boldsymbol{\theta}_t}$$

1次のモーメント
(Momentumに似た部分)

$$\boldsymbol{v}_{t+1} = \rho_2 \boldsymbol{v}_t + (1-\rho_2)\frac{\partial L}{\partial \boldsymbol{\theta}_t} \odot \frac{\partial L}{\partial \boldsymbol{\theta}_t}$$

2次のモーメトン
(RMSPropに似た部分)

$$\hat{\boldsymbol{m}}_{t+1} = \frac{\boldsymbol{m}_{t+1}}{1-\rho_1^t}$$

$$\hat{\boldsymbol{v}}_{t+1} = \frac{\boldsymbol{v}_{t+1}}{1-\rho_2^t}$$

バイアス修正式(学習初期の計算を安定化させるのが目的)

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{1}{\sqrt{\hat{\boldsymbol{v}}_{t+1}} + \varepsilon} \odot \hat{\boldsymbol{m}}_{t+1}$$

見かけの学習率

$\boldsymbol{\theta}$：パラメータ
L：損失
ρ_1：減衰率, 0.9など
ρ_2：減衰率, 0.999など
η：学習率
ε：計算を安定化させるための定数
$\odot$：アダマール積記号(同じ要素同士の掛け算)

<バイアス修正式に関する補足, 2020.03.23追記>
バイアス修正式の目的は、「パラメータの更新量を大きくすることではなく、0で初期化されている$\boldsymbol{m}$と$\boldsymbol{v}$を大きく評価すること」である。
$\boldsymbol{m}$のバイアス修正は、学習初期のパラメータ更新量を大きくすることにつながる。
$\boldsymbol{v}$のバイアス修正は、学習初期において見かけの学習率が過剰に大きくなることを防ぐ。

[演習] 勾配法の学習を最適化させる手法の実装

- 3_1_SGD_trainee.ipynb
  - SGDおよびその学習を最適化させる手法について、NNでの利用を想定したクラスを実装しましょう。

[演習] 確率的勾配降下法などを用いたNNの実装

- 3_2_SGD_withNN_trainee.ipynb
  - 確率的勾配降下法などを用いたNNを実装しましょう。

SGD更新式の別の表現

- SGD更新式の表現は、書籍によって異なるので注意すること。
- 例えば、Goodfellow著の深層学習では以下のように表現されている。

学習率ε

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \frac{\epsilon}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(f\left(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right), \boldsymbol{y}^{(i)}\right)$$

勾配　損失関数　予測結果　正解データ

データ数mで割って損失を平均化

Any Questions?

重みの初期値

重みの初期値

- ここでは、重み(パラメータW)の初期値について、どのように設定するのが適正かを考える
- 良い初期値の条件とはなにか？

重みの初期値

- 例えば、重みの初期値をある値(０とか１)に固定するとどうなるか？
  - ノードを沢山設定することの意義は、それぞれのノードに別々の役割を担わせることができること。これにより、複雑な関係性を表現することができる。
  - もし、ノードの出力値が同じになるのであれば、それは、ノードを1つだけ配置していることと同じ意味になる。
  - つまり、初期値がバラけることによって、ノードを沢山設定する意味が出てくる。
- 逆に、重みの初期値を適当に大きくしたり小さくしたりするとどうなるか？
  - 重みの初期値が0から離れると、計算が収束するまでにより多くの時間がかかってしまうことが考えられる。重みの初期値は、最終的に求まる値に近い方が計算の収束が早いはず。 NNの計算ではデータを標準化することが多く、標準化すると入力層に入力される値は0付近に分布することになる。この場合、最終的に求まる重みの値も0付近に分布することが多い。
  - 重みの初期値を0に近づけすぎると、ノードの多様性がなくなる。
- これより、「重みの初期値は、0付近で適度なバラつきをもっていると良さそうである」と言える。
- では、適度にバラけさせるにはどうすればいいか？

重みの初期値

n_1 は前の層のノード数
n_2 は後ろの層のノード数

- Xavierの初期値
  - 初期値を適度にバラつかせる方法として、Xavier Glorotらが提案する方法がある。
  - Xavierの方法では、$\sqrt{\frac{2}{n_1+n_2}}$を標準偏差とする。
  - sigmoid関数やtanh関数のように点対称で中央付近で線形関数としてみなせる活性化関数に向いている。
  - http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf
- Heの初期値
  - 初期値を適度にバラつかせる方法として、Kaiming Heらが提案する方法がある。
  - Heの方法では、$\sqrt{\frac{2}{n_1}}$を標準偏差とする。
  - Xavierの初期値にくらべ、広がりを持った初期値になる。
  - ReLU関数を用いる場合に向いている。
  - https://arxiv.org/pdf/1502.01852.pdf

[演習] 初期値の影響の確認

- 3_3_initial_value_fixed.ipynb
  - 重みの初期値(固定値)の影響を確認しましょう。
- 3_4_initial_value_random.ipynb
  - 重みの初期値(ランダム値)の影響を確認しましょう。

[演習] 初期値の実装

- 3_5_initial_value_Xavier_trainee.ipynb
  - Xavierの初期値を利用するための関数を実装しましょう。
- 3_6_initial_value_He_trainee.ipynb
  - Heの初期値を利用するための関数を実装しましょう。

[演習] ニューラルネットワークにおける初期値の影響

- 3_7_two_layer_NeuralNetwork_regression_trainee.ipynb
  - パラメータの初期値を変えながら、2層ニューラルネットワークで単純な回帰問題を解いてみましょう。

Any Questions?

機械学習と純粋な最適化問題の差異

機械学習と純粋な最適化問題の差異

- 機械学習における学習とは、本質的には最適化問題を解くことである。
- では、機械学習における最適化は、純粋な最適化問題とどのように異なるのか？
- 参考：Ian Goodfelllow, 深層学習, 8.1節

純粋な最適化問題の例

- 資源配分問題
  - 限られた資源で最大の収益を得るにはどうすればいいか？
- 車体の形状最適化問題
  - 風の抵抗を最小にするにはどうすればいいか？
- 最短ルート探索問題
  - 目的地まで最も早く辿り着くにはどの道を通ればよいか？

機械学習における最適化の特徴

- 機械学習では極小点もしくは最小点が最善な結果とは限らない。
  - 純粋な最適化問題は、勾配が小さくなり極小点もしくは最小点が見つかったら計算が終了。
  - 機械学習では、極小点もしくは最小点が最善の結果とは限らない。
    - 過学習を防ぐために、早期終了や正則化を行うことが多い。
- 機械学習は経験損失を最小化する。
  - 純粋な最適化問題は、あらかじめ決められた不変な目標に近づけることが目的になる。
    - 例、最短ルート探索問題は、距離を0に近づけることが目的。
  - 機械学習では、観測されたデータに近づけることが目的になる。よって、観測されたデータとの誤差(経験損失)を最小化する問題になる。

機械学習における最適化の特徴

- 機械学習では代理損失関数を設定する。
  - 機械学習では、最適化したい対象(目的関数) を直接最適化しないことがある。この場合、学習が進行しやすくなる損失関数を考え、その損失関数の最小化を図る。
  - この損失関数は、代理損失関数とも呼ばれる。
    - 例えば、最適化の対象(目的関数)を誤判定率にすると、 学習中の目的関数の値は6/10、5/10、4/10など離散的な値になり、重みの収束が不安定になる。この場合、クロスエントロピー誤差関数を損失関数に設定することで、目的関数を連続的な数値に置き換えることができ、目的関数の変化が滑らかになる。

機械学習における最適化の特徴

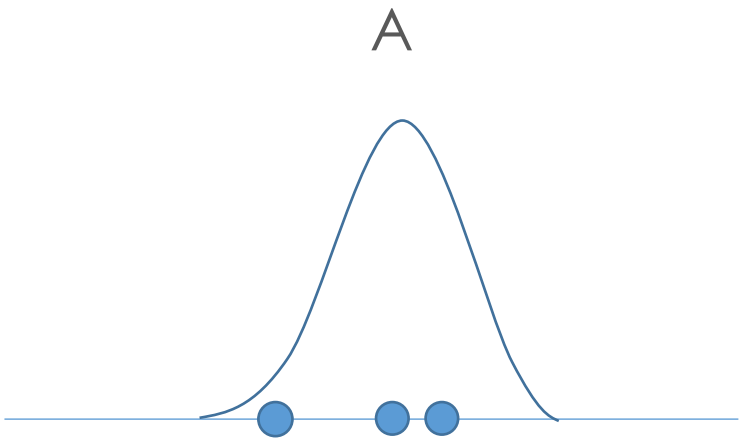
- 機械学習では目的関数を訓練事例の和に分解できる。
  - 機械学習での最適化問題を最尤推定問題として考えてみると、以下のように表現できる。
    - $\boldsymbol{\theta}_{ML} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p\left(\mathrm{y}^{(\mathrm{i})} \middle| \boldsymbol{x}^{(i)}; \boldsymbol{\theta}\right)$
      あるθの時の, $\boldsymbol{x}^{(i)}$のもとで$y^{(i)}$が観測される確率
      - $\boldsymbol{\theta}_{ML}$：対数尤度の和が最大になる時のパラメータ
      - $\boldsymbol{\theta}$：パラメータ
      - $\boldsymbol{x}^{(i)}$：i番目のデータの説明変数, $y^{(i)}$：i番目のデータの目的変数
  - この式が意味しているのは、「訓練事例(学習データ)に対する対数尤度の和が最大になるときのパラメータ$\boldsymbol{\theta}$が学習結果として採用される」ということである。なお、ここでは過学習のことは考えていない。
  - 目的関数を訓練事例の和に分解できるという考え方は、ミニバッチ学習でもバッチ学習と同等の結果(期待値が一致)が得られていることと整合がとれる。
  - 純粋な最適化問題では、目的関数を分解できないことが多い。

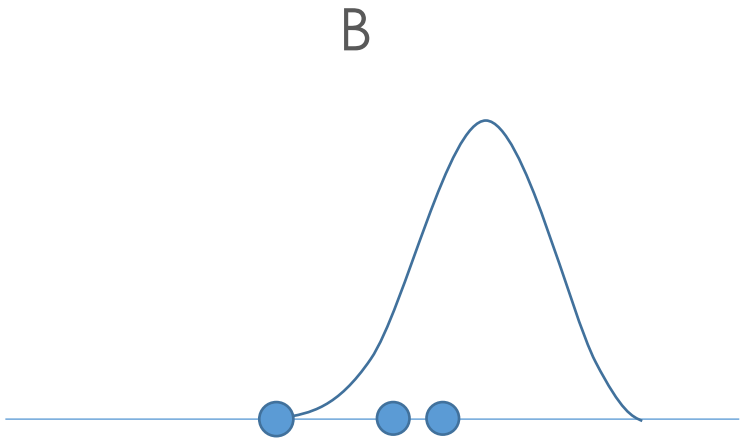
最尤法の確認

- 以下のデータが観測されたとする。
- このデータに最もよく当てはまる正規分布を求めたい。

最尤法の確認

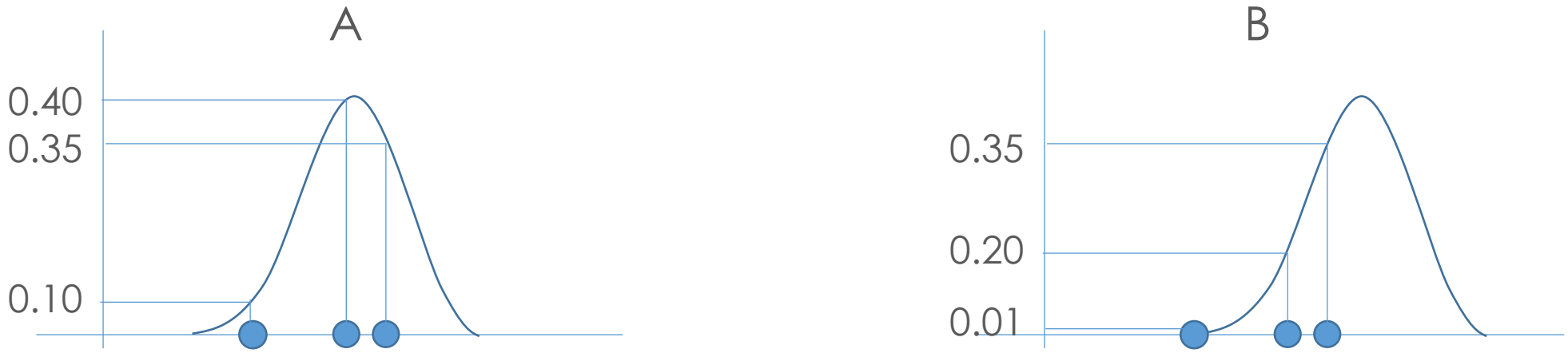
- 以下のどちらの正規分布の方がデータによく当てはまっているか？





最尤法の確認

- 正規分布A、正規分布Bのそれぞれについて、尤度を算出する。
- 尤度とは、確率(密度)を掛け算した値。



尤度=0.40×0.35×0.10=0.014 > 尤度= 0.35×0.20×0.01=0.0007

Aの正規分布の方が尤度が大きいので、Aの方がよく当てはまっていると判断する。
これが最尤法。実際には、解析的に最もよく当てはまる正規分布を求める。

ニューラルネットワーク最適化の課題

ニューラルネットワーク最適化の課題

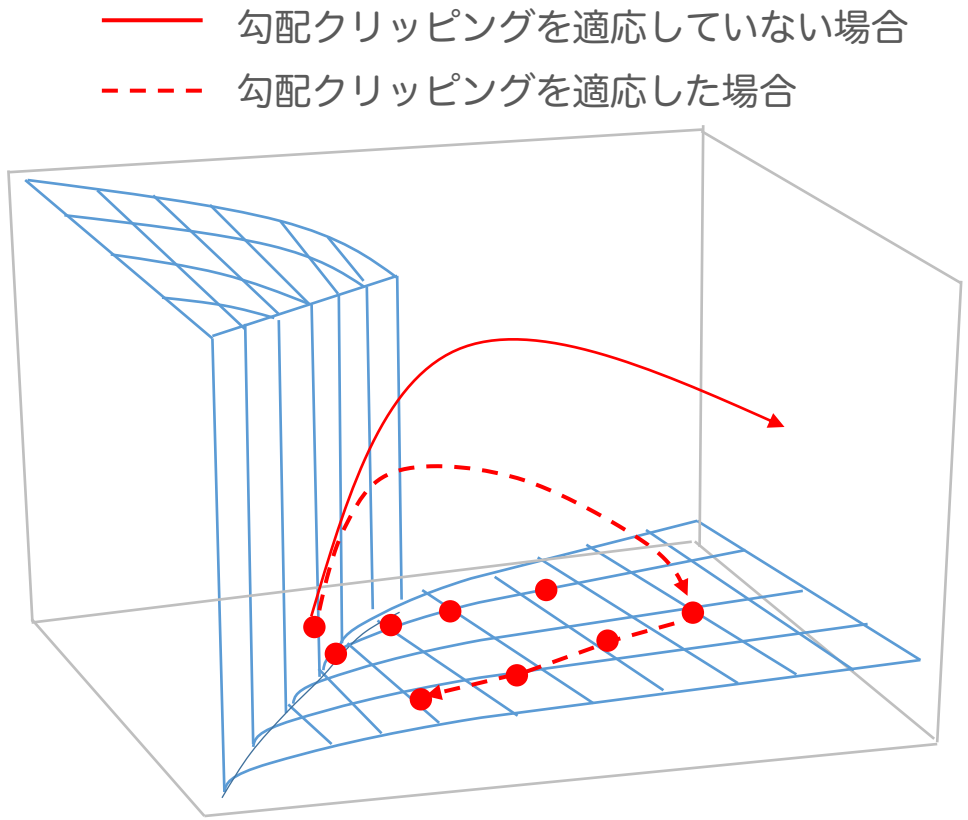
- 局所解
  - ニューラルネットワークは、一般に非凸最適化問題になる。
  - 最小点ではなく、極小点で計算が終了してしまうことがある。極小点でも最小点と同じくらいに損失が小さくなっていれば大きな問題にはならない。
  - 大きなニューラルネットワークでは、極小点の損失が最小点の損失と同程度に小さい値になるため、極小点に陥ることは大きな問題にならない、という報告もある。(Ian Goodfellow, 深層学習, 8.2.2節)
  - ミニバッチ学習を行うと、局所解に陥りにくくなる傾向がある。(DAY2で説明済み)
- 鞍点(あんてん)
  - 極小点の一種。ある方向でみると極小値、他の方向でみると極大値になっている場所のこと。最小値でないにも関わらず勾配が0になるため、探索が終了してしまう。(DAY2で説明済み)
- プラトー
  - ほとんど平らな場所。プラトーに入ると、学習が停滞する。(DAY2で説明済み)

ニューラルネットワーク最適化の課題

- 崖と勾配爆発
  - 探索領域に急峻な崖があり、そこを登ってしまうと、次のステップの勾配が非常に大きくなり(勾配爆発)、更新時の移動距離が大きくなる（遠くに飛ばされてしまう）ことがある。
  - RNNで発生することが多い問題。
  - この問題を回避する方法として、勾配クリッピングという方法がある。
- 長期依存性
  - 計算グラフが深くなる場合、同じ操作を何度も繰り返し適用することになる。これにより、誤差逆伝播時の勾配がどんどん大きく(勾配爆発)なったり、どんどん小さく(勾配消失)なったりすることがある。
  - RNNで発生することが多い問題。

勾配クリッピング

- 崖のイメージと勾配クリッピングの効果を以下に示す。



崖のイメージ

勾配クリッピングとは、勾配の値が大きくなりすぎないように調整する手法。通常は、パラメータ更新の直前に行う。勾配クリッピングには、要素ごとにクリッピングする方法や全ての要素をクリッピングする方法などがある。全ての要素をクリッピングする方法の例を以下に示す。

$$if\ \|\boldsymbol{g}\| > v$$
$$\boldsymbol{g} = \frac{\boldsymbol{g}v}{\|\boldsymbol{g}\|}$$

$\boldsymbol{g}$: 勾配ベクトル(ここでは、NN上の全ての勾配がベクトル状に並んでいるとしている)
v : 閾値(一度計算して決めたりする)
http://proceedings.mlr.press/v28/pascanu13.pdf
https://arxiv.org/pdf/1211.5063.pdf

ニューラルネットワーク最適化の課題

- 不正確な勾配
  - 例えば、目的関数が非連続であると、勾配が不正確になる。この場合は、代理損失関数を設定すれば回避できる。
- 局所構造と全体構造の不十分な対応関係
  - 難しい全体構造をもつ問題の場合、良い初期点を見つけなければ、局所解に陥る可能性が高くなってしまう。
  - 良い初期点を確実に見つける手法はまだ確立されていない。

最適化戦略とメタアルゴリズム

最適化戦略とメタアルゴリズム

- バッチ正規化(batch normalization)
  - 各層のアクティベーション分布を適度な広がりを持つように調整する手法。
  - 学習を速く進行させることができる。
  - 初期値に神経質にならなくてもよくなる。
  - 過学習を抑制する。
- 座標降下法(coordinate descent)
  - 重みを1つずつ更新していく方法。
  - 重みが2つの場合、w1を更新し、w2を更新し、w1を更新、、、これを繰り返す。
- ポルヤック平均化(Polyak–Ruppert averaging)
  - 通過した点を平均化したものを最適な点とする方法。
  - どうしても谷底にたどり着かない場合に、谷底まわりを行ったり来たりしている点を平均化すると、谷底に極めて近い点になるはずであるという発想。

最適化戦略とメタアルゴリズム

- 教師あり事前学習
  - 単純なタスクを解くモデルを訓練した後で、目的とするタスクを解くという戦略。
  - 目的とするタスクが複雑な場合に効果的。
- 再学習(fine tuning)
  - 学習済みモデルの重みを初期値にして、目的とする訓練データで学習させていく方法。
  - 目的とする訓練データで学習を始めるよりも収束が早くなることが多い。
- 転移学習(transfer learning)
  - ある問題設定で学んだことを別の問題設定の汎化性能向上に役立てること。
  - 異なる問題設定間において、共通の特徴量を仮定できるとき、転移学習は有効である。
  - 例えば、囲碁が強いモデルに将棋を覚えさせるなど。

過学習と正則化

過学習と正則化

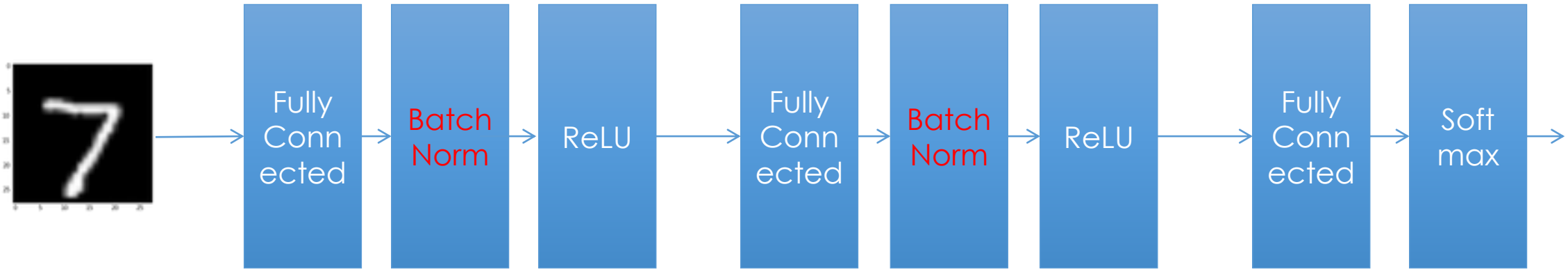
- 訓練誤差に比べ汎化誤差が大きくなる現象を過学習という。
    - 過学習が起きやすい条件
        - パラメータが多い。
        - 訓練データが少ない。
- パラメータに制限をつけることで過学習を抑制する方法を正則化(regularization)という。
- 機械学習でよく用いられる正則化の方法として、L1正則化やL2正則化がある。

バッチ正規化とその類似手法

バッチ正規化(batch normalization)

- バッチ正規化とは各層のアクティベーション分布を適度な広がりを持つように調整する手法。活性化関数の手前に設置されることが多い。
- 学習の進行が速くなることが期待できる。
- 初期値に神経質にならなくてもよくなる。
- 過学習を抑制する効果がある。(ドロップアウトを使わなくてもよくなる)
- 原著論文：https://arxiv.org/pdf/1502.03167.pdf

バッチ正規化を含む全結合層NNの例

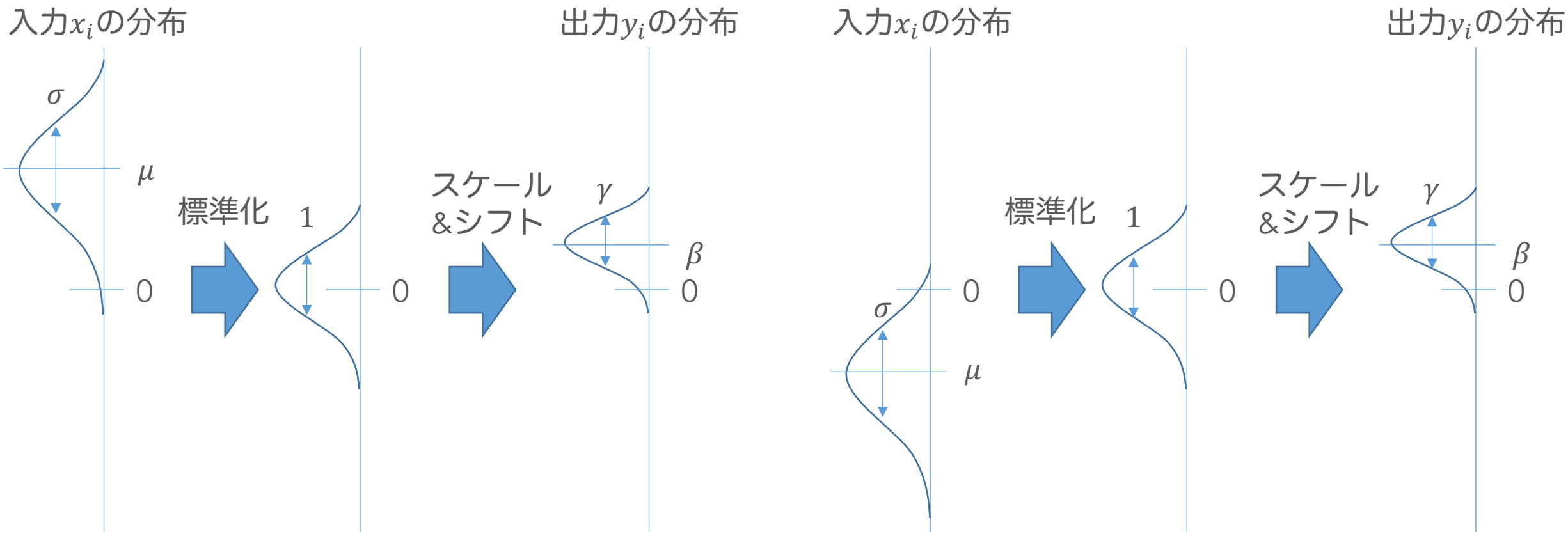


内部共変量シフト

- 共変量とは、なんらかのモデルに入力されるデータのこと。いわゆる説明変数のこと。
- 共変量シフトとは、与えられた入力に対する出力の生成規則は訓練時とテスト時で変わらないが、入力 (共変量) の分布が訓練時とテスト時で異なるという状況のこと。非定常データでよく観察される現象。
- ニューラルネットにおいては層間でもこれが起こる。あるL層の出力はL+1層の入力になるが、この入力は学習の過程において、重みが更新されるたびに全体的に移動する。これを内部共変量シフトという。
- バッチ正規化を用いると、この内部共変量シフトを抑えることができる。

バッチ正規化の概念図

- バッチ正規化の概念図を以下に示す。



入力x_iの分布がミニバッチ毎に異なっていても、同じ分布になるように変換される。

バッチ正規化の計算手順

バッチ正規化(batch normalization)レイヤの計算手順

ここでの説明は、バッチ正規化レイヤの入力層側に設置されている層の出力値のうちの1ノード分を対象にする。

[計算手順(学習時の順伝播計算)]

(1) 計算の対象をxとする

入力：$\mathbf{x} = \{x_1, x_2, \ldots, x_n\}$
n：データ数=バッチサイズ

(2) 入力の平均値を求める

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

(3) 入力の分散を求める

$$\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)^2$$

(4) 入力を標準化する

各入力値について以下の処理を行う。numpyで計算する場合はベクトルとスカラーの演算が可能。

$$\hat{x}_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}}$$

ϵ : $1e-8$ (深層学習, Goodfellow, p.229)

(5) スケールし、平行移動させる

各入力値について以下の処理を行う。numpyで計算する場合はベクトルとスカラーの演算が可能。

$y_i = \gamma\hat{x}_i + \beta$

y_iが返り値になる。

γとβは、標準化されたxの分布を最適な分布に変換するための係数であり、学習の過程で最適化されていくパラメータ。

1つのミニバッチ内で計算される平均μと分散σ^2とは値が異なる。

バッチ正規化の計算

[計算手順(予測時の順伝播計算)]

基本的には、学習時の順伝播計算と同じだが、μとσ^2は、学習時に求めた移動平均値を使う

[計算手順(学習時の逆伝播計算)]

スライドの計算グラフを参照

[参考]

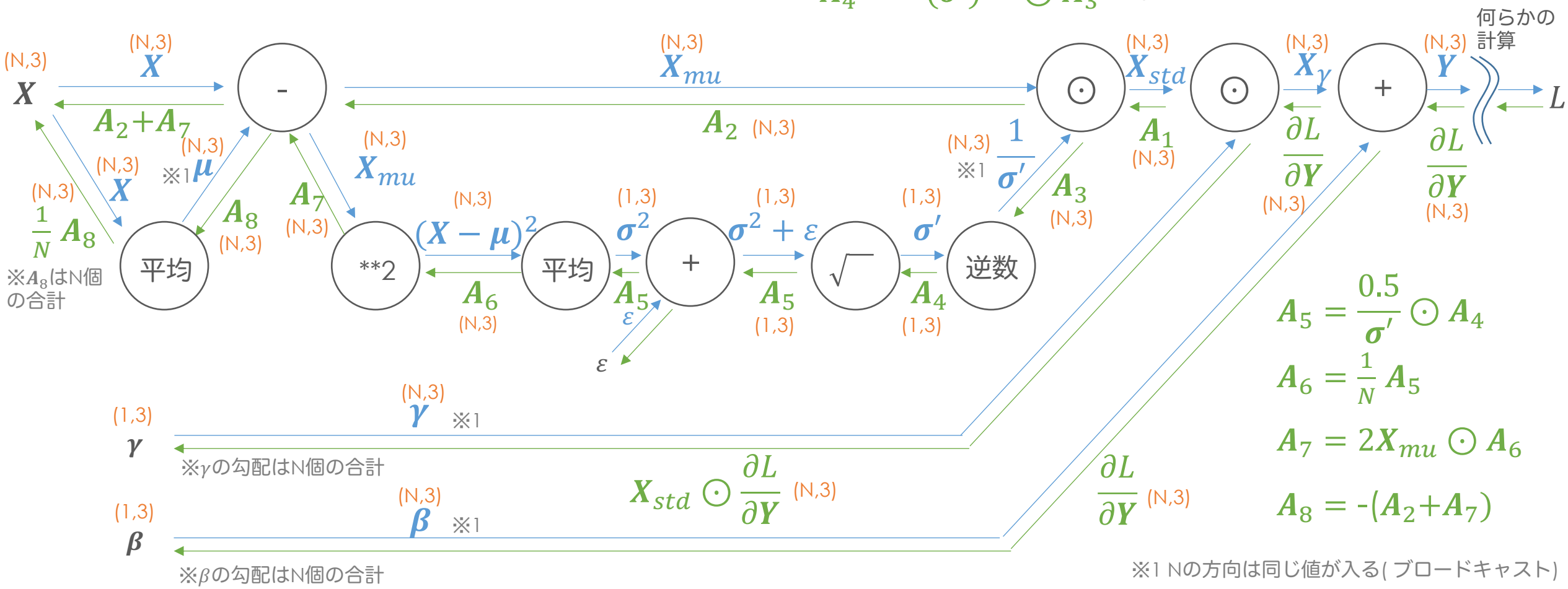
- 原著論文
  - https://arxiv.org/pdf/1502.03167.pdf
- ブログ
  - https://kratzert.github.io/2016/02/12/understanding-the-gradient-flow-through-the-batch-normalization-layer.html

バッチ正規化の計算グラフ

- バッチ正規化の計算グラフを以下に示す。
  (入力が3次元、データ数がN次元の場合)

$$A_1 = \gamma \odot \frac{\partial L}{\partial Y}$$
$$A_2 = \frac{1}{\sigma'} \odot A_1$$
$$A_3 = X_{mu} \odot A_1$$
$$A_4 = -(\sigma')^{-2} \odot A_3 \quad ※A_3はN個の合計$$

$$X_{mu} = X - \mu$$
$$X_{std} = X_{mu} \odot \frac{1}{\sigma'}$$
$$X_\gamma = \gamma \odot X_{std}$$
$$Y = X_\gamma + \beta$$



$$A_5 = \frac{0.5}{\sigma'} \odot A_4$$
$$A_6 = \frac{1}{N} A_5$$
$$A_7 = 2X_{mu} \odot A_6$$
$$A_8 = -(A_2 + A_7)$$

※1 Nの方向は同じ値が入る(ブロードキャスト)

[演習] バッチ正規化(batch normalization)レイヤの実装

- 3_8_batch_normalization_layer_trainee.ipynb
  - バッチ正規化レイヤを実装しましょう。
- 3_9_batch_norm_test.ipynb
  - MNISTデータを用いて、バッチ正規化の効果を確認しましょう。

バッチ正規化とその類似手法

- バッチ正規化が登場して以降、その類似手法がいくつか提案されている。
- ここでは、以下の3つを紹介する。
- レイヤー正規化(layer normalization)
  - Jimmy Lei Ba , Jamie Ryan Kiros , Geoffrey E. Hinton. Layer Normalization. https://arxiv.org/pdf/1607.06450.pdf
- インスタンス正規化(instance normalization)
  - Dmitry Ulyanov, Andrea Vedaldi, Victor Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. https://arxiv.org/pdf/1607.08022.pdf
- グループ正規化(group normalization)
  - Yuxin Wu, Kaiming He. Group Normalization. https://arxiv.org/pdf/1803.08494.pdf

バッチ正規化とその類似手法

- レイヤー正規化
  - レイヤー内で正規化する。
  - 原著論文では、RNNへの適応方法も提案されている。
- インスタンス正規化
  - インスタンス(1チャンネル分の画像)毎に正規化する。
  - 1チャンネル分の画像で正規化することにより、画像のコントラストを取り除くことができる。
- グループ正規化
  - チャンネルをグループ化し、正規化する。
  - バッチ正規化は、バッチサイズを小さくすると、バッチ統計量(平均と分散) の推定精度が落ち、学習がうまくいかなかった。これに対し、グループ正規化は、バッチサイズの影響をほとんど受けない。

バッチ正規化とその類似手法

- 各手法は、正規化の対象が異なる。

全てCNNを想定

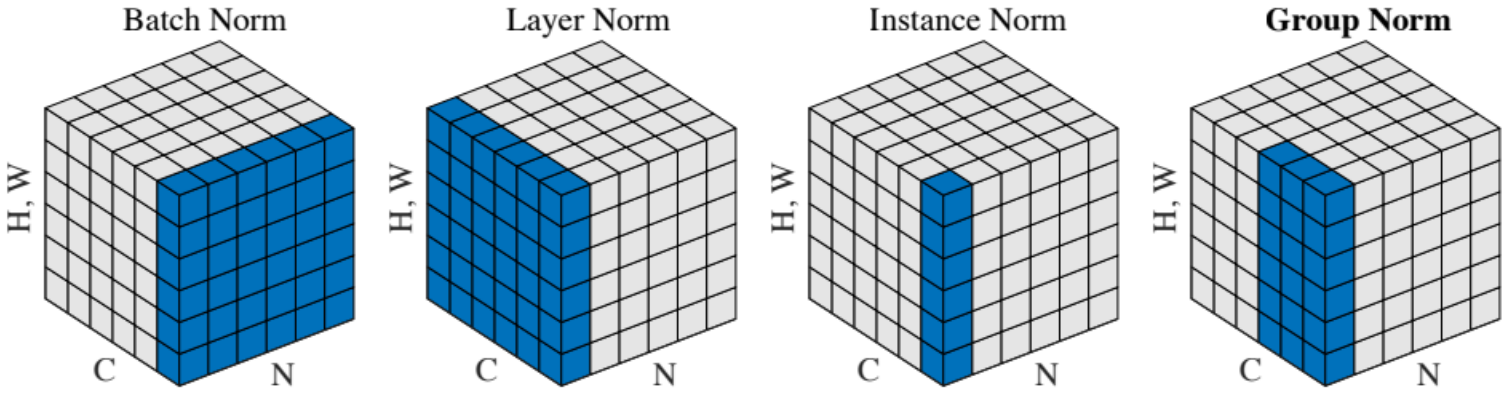


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

引用元： https://arxiv.org/pdf/1803.08494.pdf

バッチ正規化とその類似手法

- 性能比較結果の例を以下に示す。

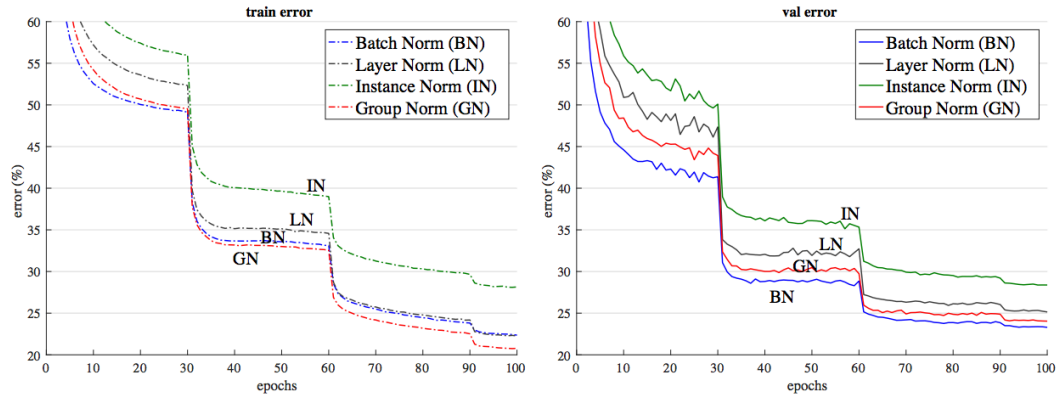


Figure 4. **Comparison of error curves** with a batch size of **32 images**/GPU. We show the ImageNet training error (left) and validation error (right) *vs*. numbers of training epochs. The model is ResNet-50.

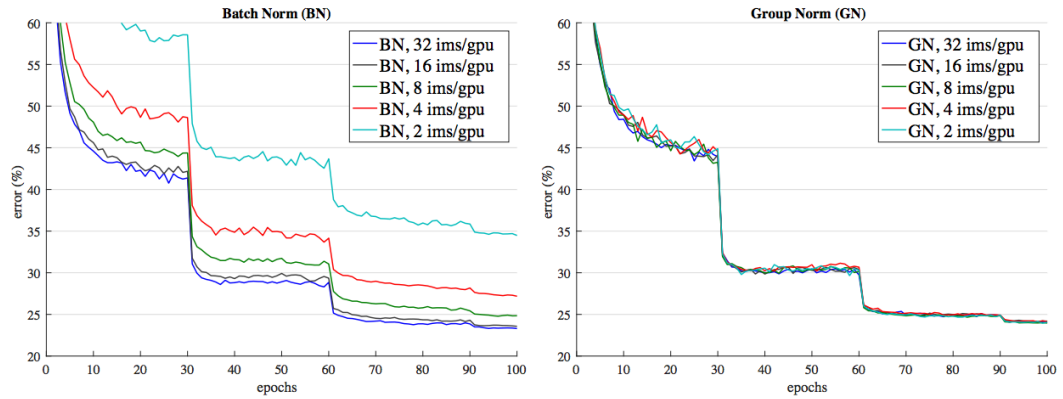


Figure 5. **Sensitivity to batch sizes**: ResNet-50's validation error of BN (left) and GN (right) trained with 32, 16, 8, 4, and 2 images/GPU.

グループ正規化(Group Norm)は、バッチ正規化(Batch Norm)と同等の精度がでており、また、バッチサイズの影響をほとんど受けていないことがわかる

引用元：
https://arxiv.org/pdf/1803.08494.pdf

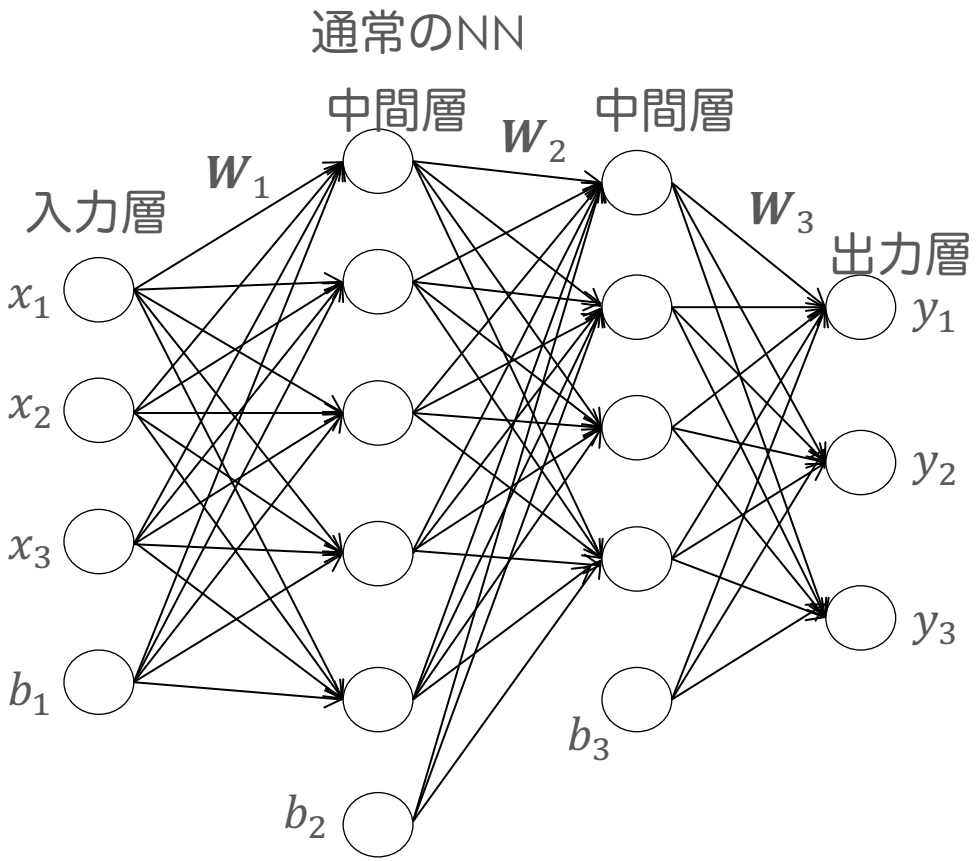
Any Questions?

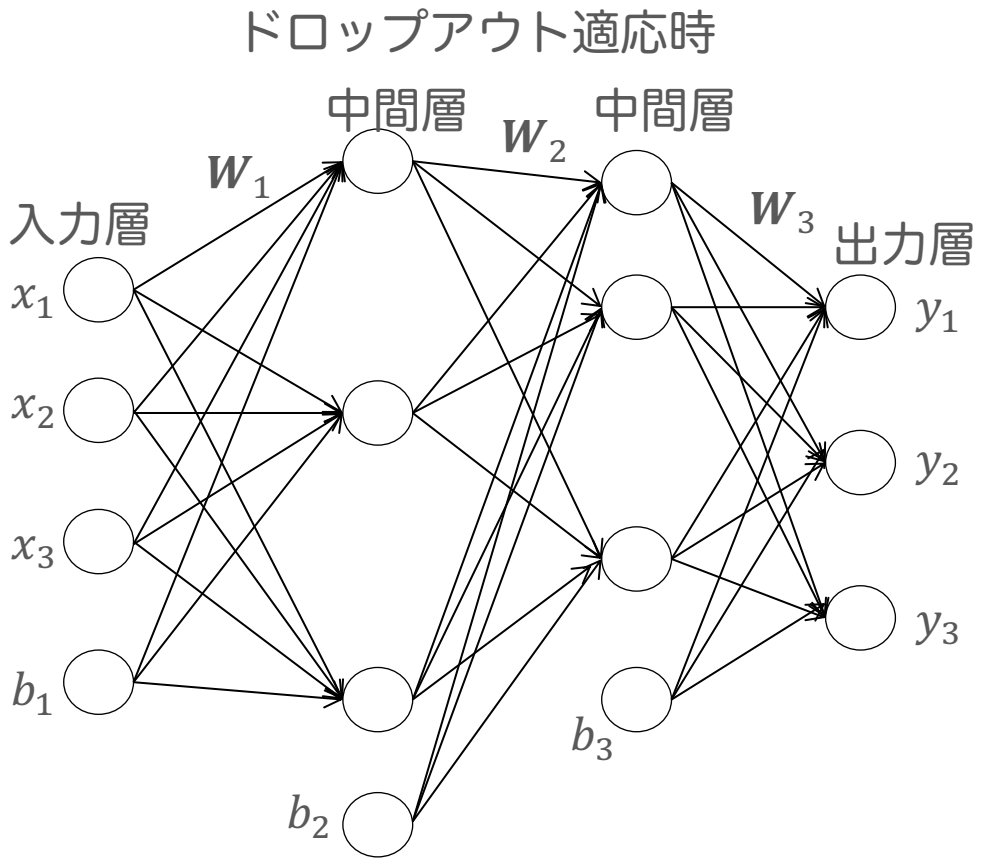
ドロップアウト

ドロップアウト

原著論文：Nitish Srivastava, et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting.
http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf

- ドロップアウト(dropout)とは、ノードをランダムに消去しながら学習する方法。
- 過学習を抑制する効果がある。





ドロップアウト(dropout)の計算

- 学習時
  - ミニバッチ ごとに、消去するノードをランダムに選ぶ。
  - 順伝播計算と逆伝播計算を行う。
  - 順伝播計算で信号を通さなかったノードは、逆伝播でも信号を通さない。
- 予測時
  - すべてのノードを採用し、すべての重みにドロップアウトしなかった割合(1 – dropout_ratio)を一律に掛ける。
  - すべてのノードを採用すると、学習時に比べ結合後の値が大きくなるので、消去した割合を掛けることにより、値を調整している。

ドロップアウトとバギング

- ドロップアウトは、複数のネットワークをアンサンブル学習していると解釈できることから、バギングの近似であると表現されることがある。 (Ian Goodfellow,深層学習, 7.12節)
- バギングとは、いくつかのモデルを組み合わせる(アンサンブル学習)ことによって、汎化誤差を減少させる方法。
    - 代表的な手法として、ランダムフォレストがある。

[演習] ドロップアウト(dropout)の実装

- 3_10_dropout_trainee.ipynb
  - ドロップアウト(dropout)を実装しましょう。

Any Questions?

荷重減衰

荷重減衰

- 荷重減衰(weight decay)は、過学習を抑制する方法の一つ。
- 重みの値が大きくなることにペナルティを与える。
- 重みの値が大きくなることによって過学習が起きることが多いので、その重みの値が大きくならないように制限する。
- 荷重減衰では、損失関数に全ての層の$\frac{1}{2}\lambda w_{ij}^2$を加える。いわゆるL2ノルム(の2乗)のこと。
- これにより、重みは、自身の大きさに比例したペナルティが加わるため、重みの値が小さくなっていく(減衰していく)。
- λはハイパーパラメータで、一般に、0.01~0.00001程度の範囲から選ぶ。
- この荷重減衰は、 wにだけ適応し、バイアスbには適応しないことが多い。バイアスbは総数が少ないため過適合を起こしにくいことと、ときに大きな値をとる必要があるため。

荷重減衰の計算式

損失関数

$$L = L' + \frac{1}{2}\lambda \sum_{l=1}^{layers} \sum_{i,\,j} (w_{ij}^{l})^2$$

$w_{i,j}^{l}$: l層目の重み行列$\mathbf{W}$のi,j成分
L : 正則化項を加えた後の損失
L' : 正則化項を加える前の損失
$layers$: 層番号
λ : 係数

重みの更新(SGDの場合)

$$\mathbf{W}_{t+1} = \mathbf{W}_t - \eta(\frac{\partial L'}{\partial \mathbf{W}_t} + \lambda \mathbf{W}_t)$$

$\mathbf{W}$: 重み行列
L' : 正則化項を加える前の損失
η : 学習率
λ : 係数

$\boldsymbol{W}_t$が正の場合、 $\boldsymbol{\lambda W}_t$が引かれるため
$\boldsymbol{W}_{t+1}$はより0に近づく。

$\boldsymbol{W}_t$が負の場合、 $\boldsymbol{\lambda W}_t$が加算されるため
$\boldsymbol{W}_{t+1}$はより0に近づく。

実装する際は、上記の更新を勾配の補正によって考慮するのが簡単でよい

勾配の補正

$$\frac{\partial L}{\partial \mathbf{W}_t} = \frac{\partial L'}{\partial \mathbf{W}_t} + \lambda \mathbf{W}_t$$

$\mathbf{W}$: 重み行列
L' : 正則化項を加える前の損失
λ : 係数

参考：荷重減衰(weight decay)の仕組み

- 荷重減衰では、通常の損失関数に比べ、全ての重みが0に近づくように計算が進む。この時、仮に全ての重みが0になってしまうと、どんな入力を入れても出力が常にゼロになる意味のないモデルができあがってしまう。よって、荷重減衰で求めたい重みは、荷重減衰を適応しなかった場合と0との間になるはず。これは損失関数の式で考慮される。
- 荷重減衰の損失関数をみてみる。右辺第一項は通常の損失関数であり、これを基準に考える。右辺第二項は、重みの値がより小さくなるように働く。この時、重みが小さくなりすぎると、逆に右辺第一項の損失（例えば、2乗和誤差）が大きくなる。つまり、右辺第一項と右辺第二項のバランスによって、最終的な重みが決定されることになる。
- 入力データが標準化されている場合、最終的に決定された重みのうち、絶対値の大きな重みは出力に対する影響度が大きいものであり、逆に絶対値の小さな重みは出力に対する影響度が小さいものであると解釈することができる。

[演習] 荷重減衰に対応したNNの実装

- 3_11_weight_decay_trainee.ipynb
  - 荷重減衰に対応したNNを実装しましょう。

その他の話題

ラベル平滑化

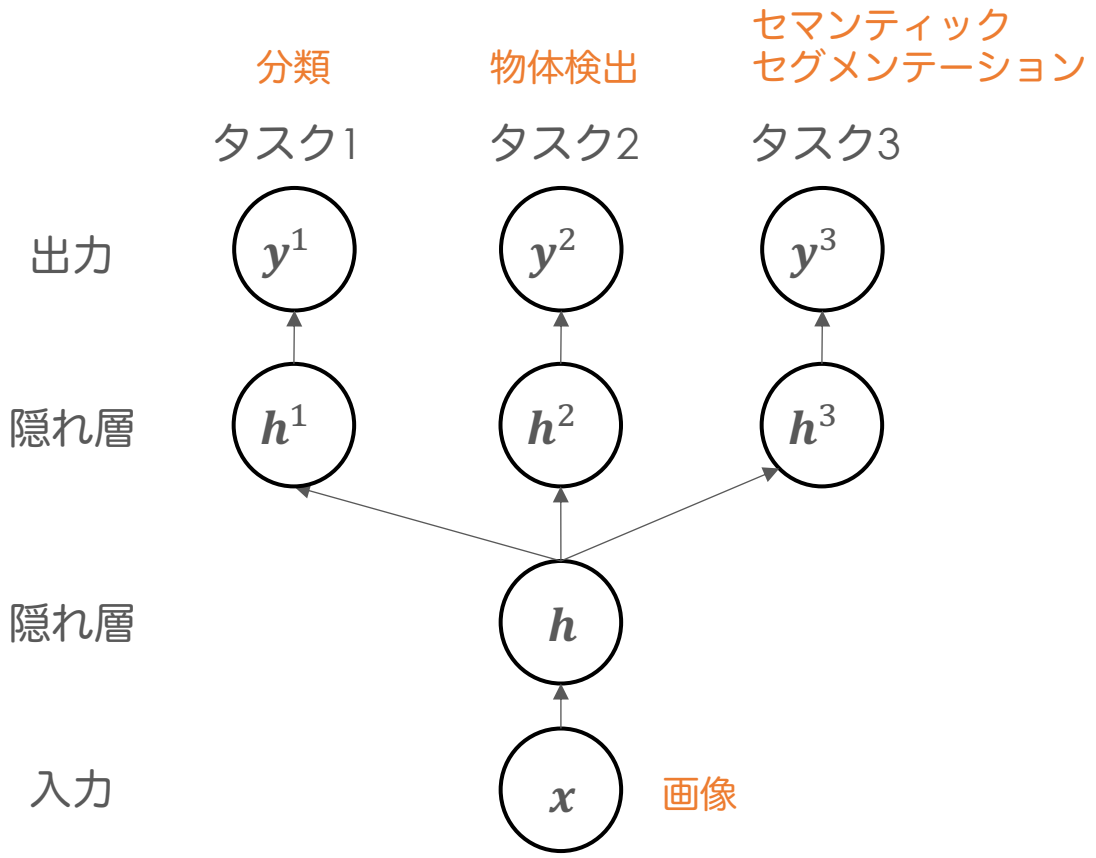
- ラベル平滑化(label smoothing)とは、ラベルにある程度の間違いがあると想定される場合に、明示的にラベルのノイズをモデル化する方法 (Ian Goodfellow,深層学習, 7.5.1節)
- 例えば、訓練集合のラベルが確率ϵで誤っていると仮定し、正解ラベルを$1-\epsilon$、残りのラベルを$\epsilon/(\mathrm{k}-1)$に置き換えるという方法がある。ここで、ϵはある小さな値、 kはクラス数。
  - 5クラス分類問題の場合の例
    - 元のラベル：$\{0,0,1,0,0\}$
    - ラベル平滑化適応時のラベル：$\{\epsilon/4, \epsilon/4, 1-\epsilon, \epsilon/4, \epsilon/4\}$
- ラベルに間違いが想定されない場合でも、正則化を目的として利用されることがある。
  - Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. CoRR, abs/1512.00567, 2015.

早期終了

- 早期終了(early stopping)とは、一定のルールを満たせば学習を終了させるという方法。
  (Ian Goodfellow,深層学習, 7.8節)
- 過学習を抑える効果がある。
- 早期終了の計算手順例
  - 学習の各ステップにおいて、テスト用データを用いて検証誤差(test_loss)を算出する。
  - 検証誤差が前ステップの検証誤差よりも大きければ、1カウントする。
  - このカウントが指定回数以上になれば、学習を終了させる。
  - ここでの指定回数はハイパーパラメータ。

マルチタスク学習

- マルチタスク学習は、ニューラルネットワークの一部を複数のタスク間で共有する方法。(Ian Goodfellow, 深層学習, 7.7節)
- 正則化の効果が期待できる。
- マルチタスク学習の例を以下に示す。



Any Questions?

[グループワーク] 確率的勾配降下法の確認

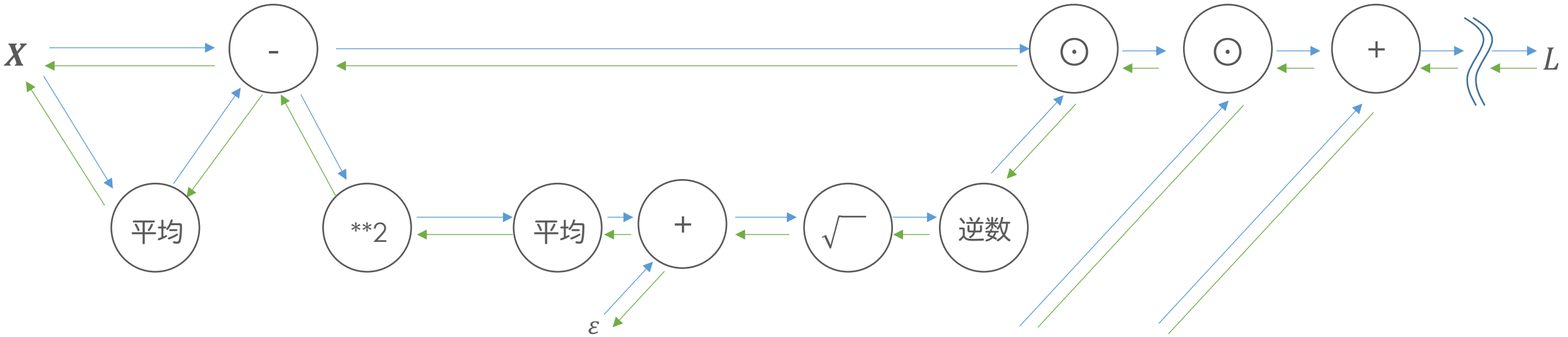
- 3_1_SGD_trainee.ipynbの最下部に記載している演習にとりくみます。
  - 学習率やモーメンタム係数を変更して、結果がどのように変わるか確認しましょう。
  - 手法、学習率、係数をいろいろ変化させて、収束までのステップ回数を比較してみましょう。一覧表をつくると比較しやすいでしょう。
- 3_2_SGD_withNN_trainee.ipynbの最下部に記載している演習にとりくみます。
  - 最適化手法を変更し、結果がどのように変わるかを確認しましょう。
- 上記を予習段階で各自取り組みましょう。
- 予習段階の取り組み結果をグループで共有します。(15分)
- 最後に、 1グループだけ発表していただきます。 (5分)

[グループワーク] 荷重減衰適応時の計算グラフ

- 荷重減衰を考慮したニューラルネットワークの計算グラフを描いてみましょう。
- 上記を予習段階で各自取り組みましょう。
- 予習段階の取り組み結果をグループで共有します。(15分)
- 最後に、グループごとに発表していただきます。 (15分)

[グループワーク] バッチ正規化の計算グラフ

- 以下のバッチ正規化の計算グラフを完成させましょう。
- 「バッチ正規化の計算グラフ」のページが解答例になりますが、なるべくこのページを見ないで取り組みましょう。
- 上記を予習段階で各自取り組みましょう。
- 予習段階の取り組み結果をグループで共有します。(15分)



講座の時間が余ったら

- 今回の復習をします。
- 次回の予習をします。