Deep Learning 輪読会 2017

第8章　深層モデルの訓練のための最適化

2017.11.27

経済学研究科 森下光之助

新領域創成科学研究科 藤野暢

Deep Learning 輪読会 2017

第8章　深層モデルの訓練のための最適化(前半)

2017.11.27

東京大学大学院経済学研究科
澤田康幸研究室
D1 森下光之助

構成

8.1 学習と純粋な最適化の差異

8.1.1 経験損失最小化

8.1.2 代理損失関数と早期終了

8.1.3 バッチアルゴリズムとミニバッチアルゴリズム

8.2 ニューラルネットワーク最適化の課題

8.2.1 悪条件

8.2.2 局所値

8.2.3 プラトー，鞍点，その他平坦な領域

8.2.4 崖と勾配爆発

8.2.5 長期依存性

8.2.6 不正確な勾配

8.2.7 局所構造と全体構造の不十分な対応関係

8.2.8 最適化の理論的限界

8.3 基本的なアルゴリズム

8.3.1 確率的勾配降下法

8.3.2 モメンタム

8.3.3 ネステロフのモメンタム

8.4 パラメータの初期化戦略

深層モデルの訓練のための最適化

- 深層学習のアルゴリズムでは最適化が頻繁に登場

- 最適化の目的：コスト関数を大幅に減少させるニューラルネットワークのパラメータを発見すること

8.1 学習と純粋な最適化の差異

- 通常の最適化とは違い，機械学習における最適化は間接的
  - コスト関数$J(\theta)$を最小化することで性能指標Pを最適化

- コスト関数は訓練集合における平均として書ける：

$$J(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim\hat{p}_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y), \tag{8.1}$$

- 通常は有限の訓練集合だけでなくデータの生成分布に渡って期待値を取る：

$$J^*(\boldsymbol{\theta}) = \mathbb{E}_{(\boldsymbol{x},\mathrm{y})\sim p_{\mathrm{data}}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y). \tag{8.2}$$

8.1.1 経験損失最小化

- 機械学習アルゴリズムは期待汎化誤差を最小化することが目的
- しかし，多くの場合真の分布 $p_{data}(x, y)$は未知

  → 経験損失最小化

$$\mathbb{E}_{\boldsymbol{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}(\boldsymbol{x},y)}[L(f(\boldsymbol{x};\boldsymbol{\theta}),y)] = \frac{1}{m}\sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}),y^{(i)}) \qquad (8.3)$$

- 問題点：
  - 過剰適合の可能性
  - 0/1損失などの場合，有用な導関数が存在しない

- → 別のアプローチが必要

8.1.2 代理損失関数と早期終了

- 経験損失の代わりに代理損失を最適化
  - Ex) 負の対数尤度関数　←　0/1損失の代理

- 経験損失を用いるよりも多くのことを学習できる場合がある
  - 訓練集合の 0/1損失が0に到達した後もテスト集合の0/1損失は減少し続ける場合がある　←　期待0/1損失が0になってもクラスを相互に分離するようにすることで分類器の頑健性を高め，さらに信頼性の高い分類器となるため

- アルゴリズムは早期終了に基づく収束基準が満たされた時点で停止

8.1.3 バッチアルゴリズムとミニバッチアルゴリズム

- 目的関数は訓練事例の和に分解できる
  - Ex) 最尤推定

$$\boldsymbol{\theta}_{\mathrm{ML}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{m} \log p_{\mathrm{model}}(\boldsymbol{x}^{(i)}, y^{(i)}; \boldsymbol{\theta}). \tag{8.4}$$

- 和を最大化することは経験分布の期待値を最大化することと等価

$$J(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathrm{y}\sim\hat{p}_{\mathrm{data}}} \log p_{\mathrm{model}}(\boldsymbol{x}, y; \boldsymbol{\theta}). \tag{8.5}$$

- 期待値を直接計算すると計算コストが高い

  → データ集合から少数の事例をランダムサンプリングして平均を取ることで期待値を計算

8.1.3 バッチアルゴリズムとミニバッチアルゴリズム

- 訓練事例すべてを用いる最適化アルゴリズムをバッチ/決定論的手法と呼ぶ
- 一度に１サンプルしか用いない最適化アルゴリズムは確率的/オンライン手法と呼ばれる
- 深層学習で用いられるアルゴリズムの多くは，この2つの手法の中間的なもの（使用する訓練事例は複数）
  - これらの手法は伝統的にミニ バッチ法/確率的ミニバッチ法と呼ばれる
  - 最近では単に確率的手法と呼ぶのが一般的

8.1.3 バッチアルゴリズムとミニバッチアルゴリズム

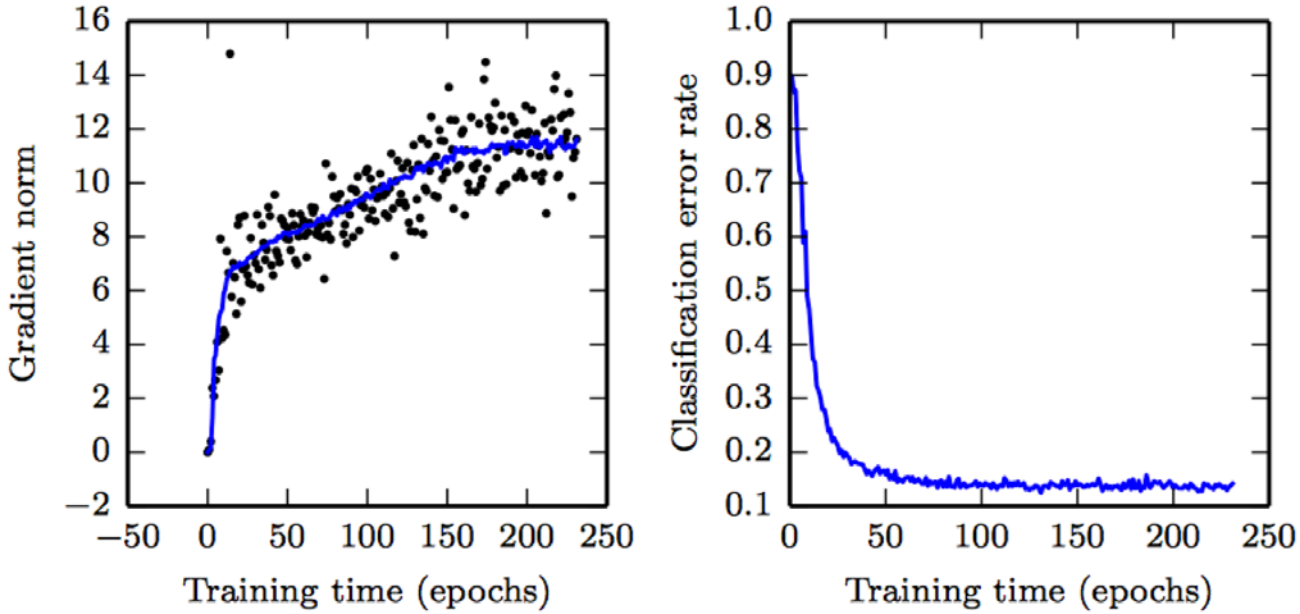
- バッチサイズが大きいほど勾配推定は正確になるが，その改善度はバッチサイズに対して線形以下
- マルチコア・アーキテクチャは非常に小さいバッチでは十分に活用されない
- バッチ内の事例がすべて並列に処理される場合，メモリ量はバッチサイズに応じて大きくなる
- GPUを用いる場合2のべき乗のバッチサイズで実行時間が短縮される
- 小さなバッチは正則化の効果をもたらすことがある
- ヘッセ行列を用いた更新などの二次手法では大きなバッチサイズが必要
- ミニバッチはランダムに選択されることが重要
  - 実用上はデータセットの順序を1回シャッフルするだけで十分

8.2 ニューラルネットワーク最適化の課題

- 一般的な機械学習では最適化問題が凸最適化となるようにして簡単化してきた
- ニューラルネットワークの学習では困難な非凸最適化に対処しなければならない
- この節では深層モデルの学習の最適化に関わる最も有名な課題をいくつかを紹介

8.2.1 悪条件

- ヘッセ行列Hの悪条件が問題となることがある



8.2.2 局所値

- 凸最適化では極小値＝最小値だが，非凸最適化ではその保証がない

- 重み空間の対称性から生じるモデルの非同定可能性は問題とはならない

  ←　極小値は等しくなるため

- 最小値よりも大きい極小値が存在する場合に問題となる

  →　最小ではないものの小さい極小値を見つけることも重要

8.2.3 プラトー，鞍点，その他平坦な領域

- 高次の非凸関数の多くでは鞍点と比較すると極小点は少ない
  - Dauphin et al. (2014)はニューラルネットワークがコストの高い鞍点を非常に多く含む損失関数を有することを実験的に示した

- 勾配降下法は経験的に鞍点を避けられることが多い

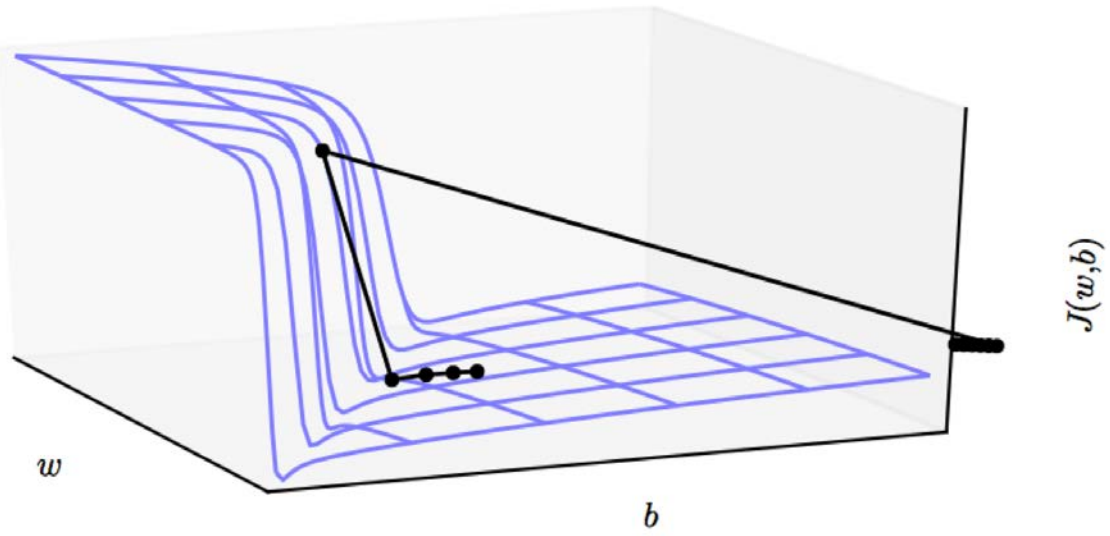
- ニュートン法では適切に修正を施さなければ鞍点に飛んでしまう場合がある

  → サドルフリーニュートン法 (Dauphin et al. (2014))

8.2.4 崖と勾配爆発

- 多層のニューラルネットワークは急峻な崖を持つことが多い

  → 非常に大きな微分が生じ，勾配降下法の更新はパラメータを非常に遠くまで移動させてしまう ← 勾配クリッピングで対処



8.2.5 長期依存性

- 非常に深い計算グラフでは勾配消失問題と勾配爆発問題が生じ得る
  - Ex) 行列Wを繰り返し掛けることで構成される経路が計算グラフに含まれる場合

$$\boldsymbol{W}^t = \left(\boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})\boldsymbol{V}^{-1}\right)^t = \boldsymbol{V}\mathrm{diag}(\boldsymbol{\lambda})^t\boldsymbol{V}^{-1} \tag{8.11}$$

  - 勾配消失があるとコスト関数を改善するためにパラメータをどの方向に動かすべきかを知ることが難しくなる
  - 勾配爆発があると学習は不安定になる

- 順伝播ネットワークは違う重みを利用するためこの問題は概ね回避できる（回帰結合型ネットワークは各時間ステップで同じ行列Wを使用するためこの問題が起きる(10.7節参照)）

8.2.6 不正確な勾配

- ほとんどの最適化アルゴリズムは，正確な勾配やヘッセ行列を利用できるという前提で設計されている　←　実際はこれにノイズが加わったりバイアスが含まれる推定しか得られないのが一般的

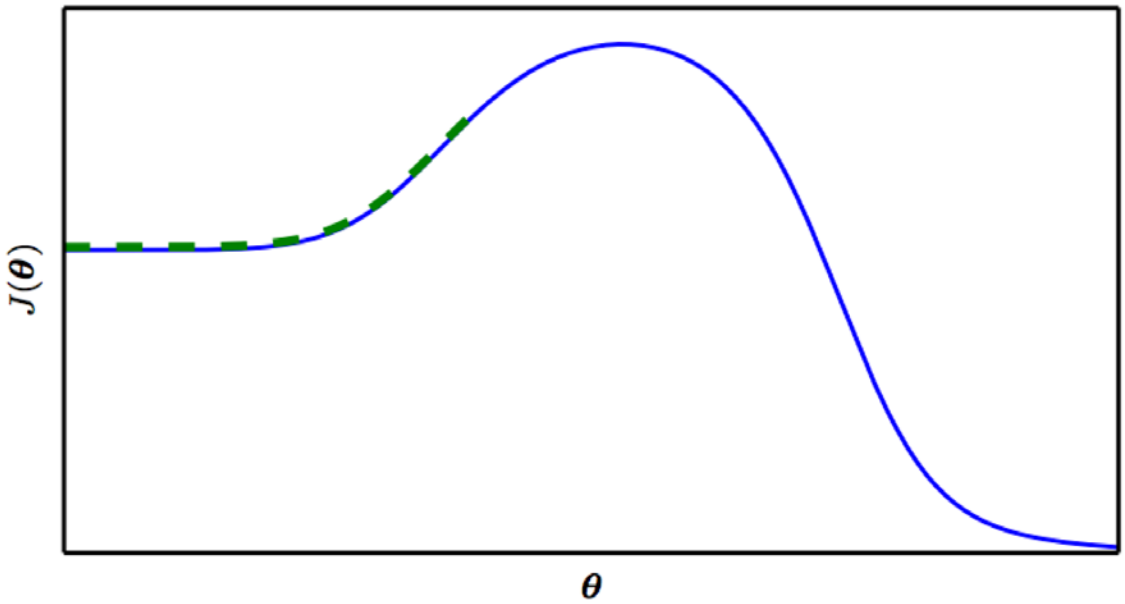
- 最小化したい目的関数もその勾配も扱いづらい場合がある
  →　近似しやすい代理損失で対処

8.2.7 局所構造と全体構造の不十分な対応関係

- ここまで議論してきた問題の多くは，ある点での損失関数の特性に対応

- 局所的に最大の改善をもたらす方向がより低コストの遠い領域を指していなければ性能が不十分なままの可能性

  → 難しい全体構造を持つ問題において良好な初期点を見つける



8.2.8 最適化の理論的限界

- いくつかの理論的研究結果で，ニューラルネットワークのために設計したいかなる最適化アルゴリズムにおいても，その性能に限界があることが示されている

- この結果は通常ニューラルネットワークを実際に使用する場合にはほとんど関係ない

8.3 基本的なアルゴリズム

- 基本的なアルゴリズムとして以下を紹介する：
  - 確率的勾配降下法
  - モメンタム
  - ネステロフのモメンタム

8.3.1 確率的勾配降下法

- 機械学習全般そして特に深層学習において最もよく用いられる最適化アルゴリズム
- ミニバッチの勾配の平均を取ることで勾配の不偏推定量を得る

Algorithm 8.1 k 回目の訓練の反復における確率的勾配降下法（SGD）の更新

Require: 学習率 ϵ_k.

Require: 初期パラメータ $\boldsymbol{\theta}$

while 終了条件を満たさない **do**

訓練集合 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ と対応する目標 $\boldsymbol{y}^{(i)}$ から m 個の事例のミニバッチをサンプリング

勾配の推定値を計算： $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

更新を適用： $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon_k \hat{\boldsymbol{g}}$

end while

8.3.1 確率的勾配降下法

- 重要なパラメータは学習率ϵ_k
- 実用上は$\alpha = \frac{k}{\tau}$として，τ 回までの反復で線形に学習率を減衰させる：

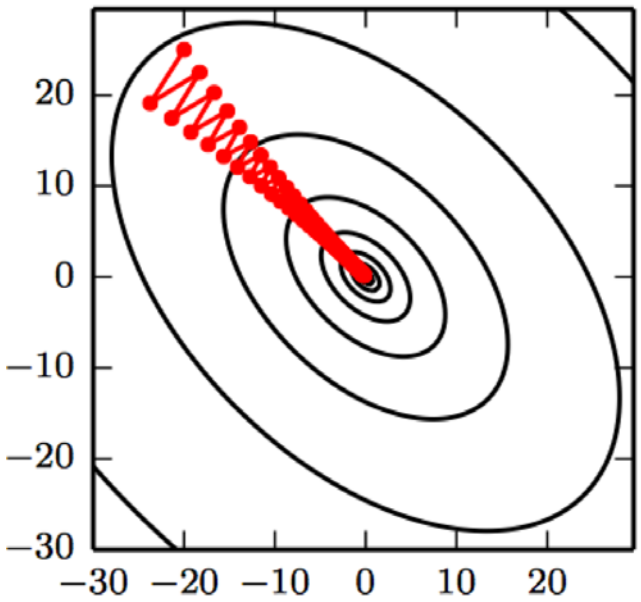
$$\epsilon_k = (1-\alpha)\epsilon_0 + \alpha\epsilon_\tau$$

- τ回の反復以降はϵ_τを定数にしておく
- ϵ_0の決め方について：
  - 大きすぎると学習曲線が激しく振動してしまう
  - 小さすぎると高いコスト値に留まってしまう
- → 最初の数回の反復を監視し，その時点で最も良い性能を出す学習率よりも高いが，深刻な不安定性を生じさせるほどは高くない学習率を用いる

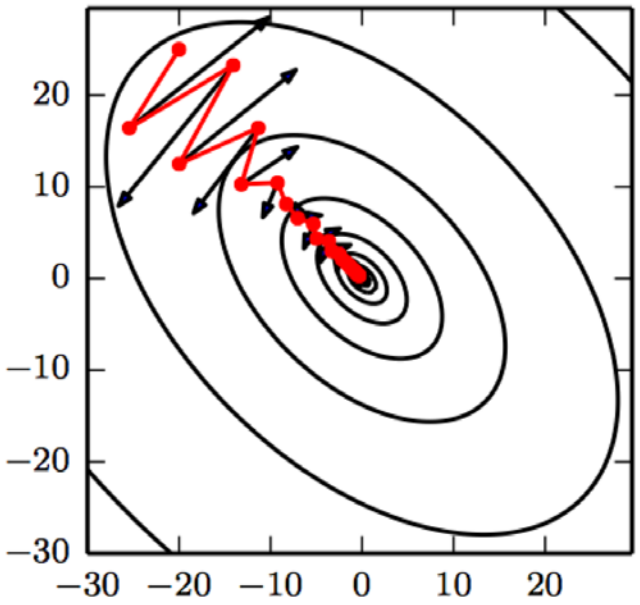
8.3.2 モメンタム

- 特に曲率が高い場合，小さく一定の勾配の場合，またはノイズが含まれる勾配に直面した場合に用いられる
- 指数関数的に減衰する過去の勾配の移動平均を蓄積し，継続的にその勾配の方向に進むようにする.

最急降下法



モメンタム



8.3.2 モメンタム

- モメンタムの更新規則：

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left(\frac{1}{m} \sum_{i=1}^{m} L(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)}) \right), \tag{8.15}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}. \tag{8.16}$$

- vが勾配要素を蓄積しαが大きいほど過去の勾配が現在の方向に影響する

Algorithm 8.2 モメンタムを使った確率的勾配降下法（SGD）

Require: 学習率 ϵ, モメンタムパラメータ α.

Require: 初期パラメータ $\boldsymbol{\theta}$, 初期速度 $\boldsymbol{v}$.

while 終了条件を満たさない **do**

訓練集合 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ と対応する目標 $\boldsymbol{y}^{(i)}$ から m 個の事例のミニバッチをサンプリングする.

勾配の推定値を計算する: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

速度の更新を計算する: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

更新を適用する: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

8.3.3 ネステロフのモメンタム

- 更新規則：

$$\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \nabla_{\boldsymbol{\theta}} \left[\frac{1}{m} \sum_{i=1}^{m} L\left(\boldsymbol{f}(\boldsymbol{x}^{(i)}; \boldsymbol{\theta} + \alpha \boldsymbol{v}), \boldsymbol{y}^{(i)} \right) \right], \tag{8.22}$$

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}, \tag{8.23}$$

Algorithm 8.3 ネステロフのモメンタムを使った確率的勾配降下法（SGD）

Require: 学習率 ϵ, モメンタムパラメータ α.

Require: 初期パラメータ $\boldsymbol{\theta}$, 初期速度 $\boldsymbol{v}$.

while 終了条件を満たさない **do**

訓練集合 $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ と対応する目標 $\boldsymbol{y}^{(i)}$ から m 個の事例のミニバッチをサンプリングする.

暫定的な更新を適用: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

（暫定点での）勾配を計算する: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$

速度の更新を計算する: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$

更新を適用する: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

8.4 パラメータの初期化戦略

- 深層学習においてほとんどのアルゴリズムは初期設定の選択に大きく影響を受ける

- 現代の初期化戦略は試行錯誤に基づく

- 確実にわかっている唯一の性質は，おそらく初期パラメータが異なるユニット間での「対称性を破る」必要があるということ
  - 同じ活性化関数を持つ隠れユニットが同じ入力に接続されている場合，異なる初期パラメータを持つことが必要

8.4 パラメータの初期化戦略

- 最適化の観点からは情報をうまく伝播させるために重みを大きくすることが推奨されるが，正則化の懸念からは重みを小さくすることが推奨される

- m個の入力とn個の出力がある全結合層の重みを初期化する場合に，Glorot and Bengio (2010) では正規化された初期化が提案されている：

$$\mathrm{W}_{i,j} \sim U\left(-\sqrt{\frac{6}{m+n}}, \sqrt{\frac{6}{m+n}}\right).$$

- 計算資源が十分にあれば，各層の重みの初期の大きさをハイパーパラメータとして扱い探索するのも良いアイデア

8.4 パラメータの初期化戦略

- バイアスの設定は重みの設定と連係させる必要がある
  ←　多くの場合は0で初期化する

  – バイアスを出力ユニットで使うのであれば，出力に関する正しい周辺統計量を出力する ようにバイアスを初期化する

  – 過度の飽和を防ぐようにバイアスを選択したい場合，たとえばReLU の隠れ層のバイアスを0.1に設定して，初期化時に飽和するのを防ぐ
    ←　ランダムウォーク初期化と一緒に利用することは非推奨

  – LSTM モデルの忘却ゲートのバイアスを1に初期化する

Deep Learning 輪読会 2017

第8章　深層モデルの訓練のための最適化(後半)

2017.11.27

新領域 陳研究室

D2 藤野暢

構成

深層モデルの訓練のための最適化（後半）

- 8.5 適応的な学習率を持つアルゴリズム
  - 8.5.1 Adagrad
  - 8.5.2 RMSProp
  - 8.5.3 Adam

  - 8.5.4 適切な最適化アルゴリズムの選択
- 8.6 二次手法の近似
  - 8.6.1 ニュートン法
  - 8.6.2 共役勾配
  - 8.6.3 BFGS

- 8.7 最適化戦略とメタアルゴリズム
  - 8.7.1 バッチ正規化
  - 8.7.2 座標降下法
  - 8.7.3 ポルヤック平均化
  - 8.7.4 教師あり事前学習
  - 8.7.5 最適化を支援するモデルの設計
  - 8.7.6 継続法とカリキュラム学習

8.5 適応的な学習率を持つアルゴリズム

- 学習率 (learning rate) はNNの性能に大きく影響するハイパラメータ
  - 適当な値を見つけるのが難しい

- 誤差関数の各パラメータに対する影響は均一的ではない
  - パラメータごとにいい感じに設定したい

- パラメータ別に, かつ自動で設定できると嬉しい

8.5.1 AdaGrad

- パラメータごとに自動で学習率を小さくしていく
  - 学習率を過去の勾配の二乗の累積和で割る
  - 大きな勾配を経験したパラメータは急速に, 小さな勾配を経験したパラメータは穏やかに学習率が減少していく
  - 初期段階で大きな勾配をとると, その後の学習率が急激に小さくなってしまう

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $\boldsymbol{r} = \mathbf{0}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
Compute update: $\Delta\boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.
end while

8.5.2 RMSProp

- AdaGradの累積和を重み付き指数移動平均で置き換えることで問題を軽減

$$\boldsymbol{r}^{(t+1)} \leftarrow \rho \boldsymbol{r}^{(t)} + (1-\rho)\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)}$$

$$= (1-\rho)\sum_{i=1}^{t} \rho^{t-i} \boldsymbol{g}^{(i)} \odot \boldsymbol{g}^{(i)}$$

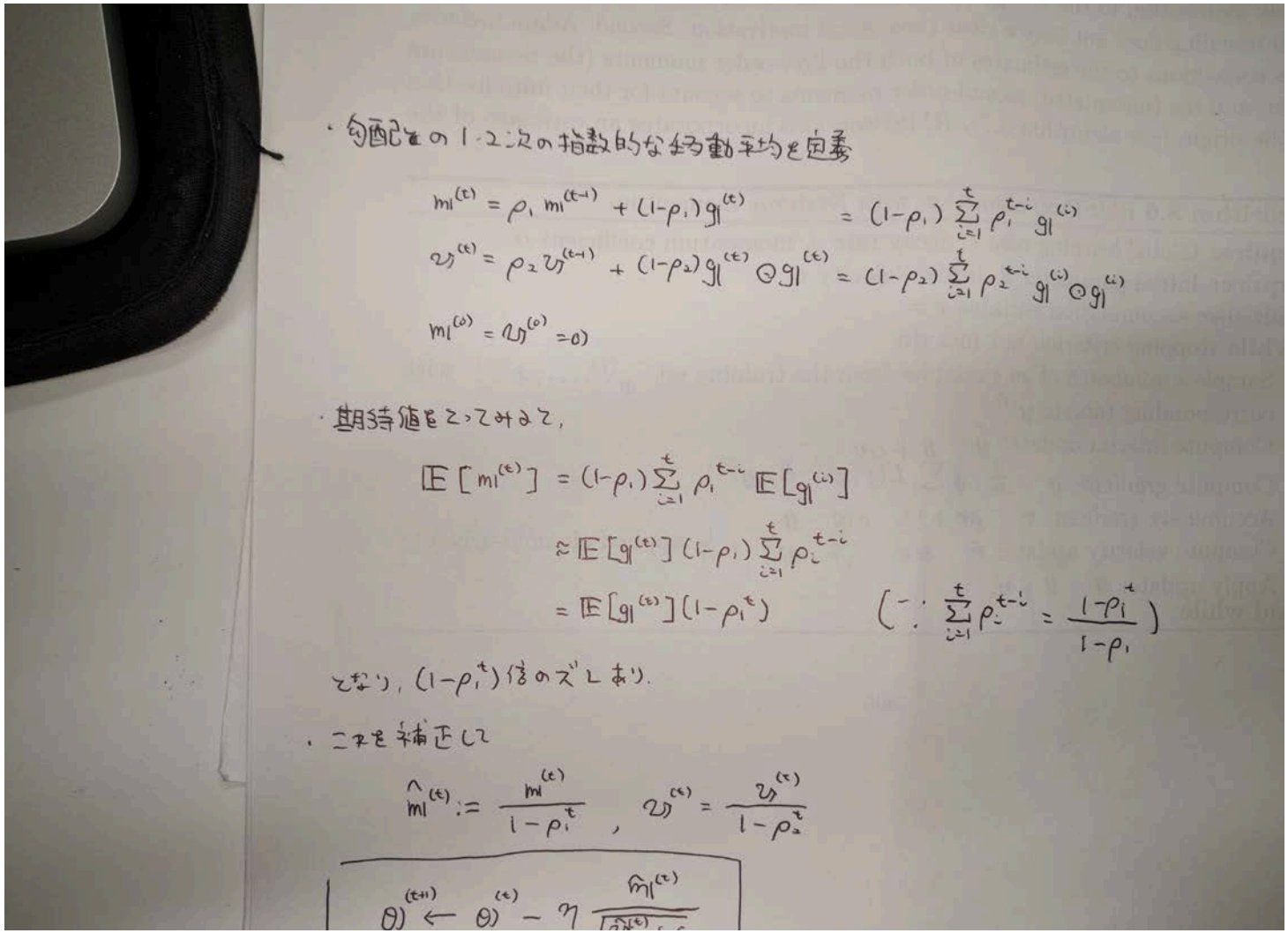
  - 古い勾配には小さい重み, あたらしい勾配には大きな重みを付ける

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
  Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
  Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
  Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$.
  Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta+\boldsymbol{r}}}$ applied element-wise)
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.

8.5.3 Adam

- 補正した勾配の1・2次モーメントを用いた最適化手法
  - RMSPropからの進化と考えるとわかりやすいかもしれない



8.5.4 適切な最適化アルゴリズムの選択

- これが良いという1つを決めるのは難しい

- 現在実際によく使われているのは
  - SGD (with momentum)
  - RMSProp (with momentum)
  - AdaDelta
  - Adam

8.6.1 ニュートン法 (Newton's Method)

- 二次勾配を利用した最適化手法
  - コスト関数の2次までのテイラー展開

$$J(\boldsymbol{\theta}) \approx J(\boldsymbol{\theta}_0) + (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0) + \frac{1}{2}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)^T \boldsymbol{H}(\boldsymbol{\theta} - \boldsymbol{\theta}_0)$$

- 更新式

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \boldsymbol{H}^{-1}\boldsymbol{g}$$

  - $\boldsymbol{g} = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$: 一次微分
  - $\boldsymbol{H} = \nabla^2_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$: 二次微分

8.6.1 ニュートン法 (Newton's Method)

- ニュートン法はヘッセ行列が正定値であるときのみ適切

- 深層学習においては
  - コスト関数は非凸. 鞍点が沢山
  - 鞍点近傍でヘッセ行列の固有値すべてが正でない場合, ニュートン法は間違った方向に移動するように更新してしまう
  - この問題はヘッセ行列の対角成分に定数を加えて正則化することで回避可能 (らしい)

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - [\boldsymbol{H}(f(\boldsymbol{\theta}_0) + \alpha \boldsymbol{I}]^{-1} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_0)$$

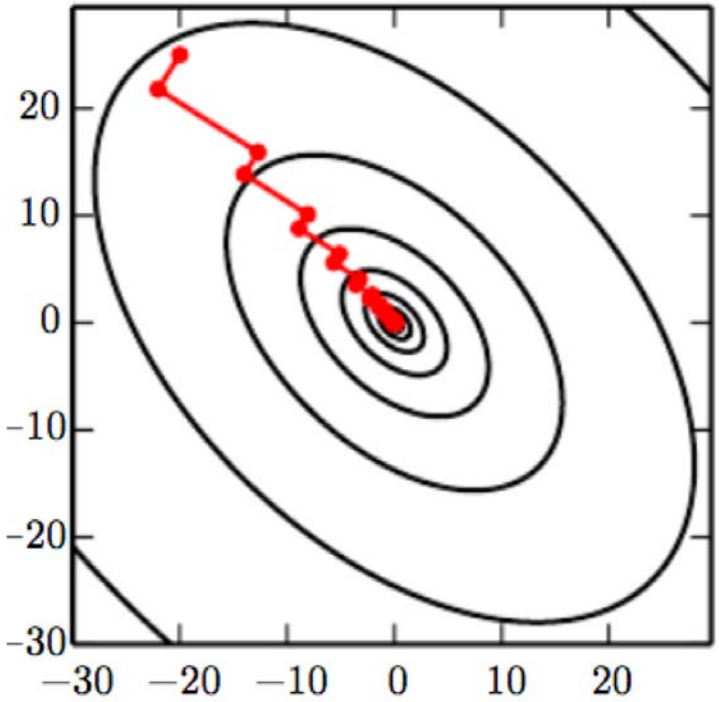
  - ヘッセ行列の負の固有値がゼロに近い値である限りよく機能する

8.6.1 ニュートン法 (Newton's Method)

- ニュートン法は k x k のヘッセ行列を毎回計算する必要がある
  - $O(k^3)$. kは数百万 ~
  - 現実的でない
  - 代替案がほしい

8.6.2 共役勾配 (Conjugate Gradients)

- 最急降下法は勾配が最も急な方向に更新
  - コスト関数や初期値によっては非効率な探索になる
  - 更新方向が直前のものと直交するため



8.6.2 共役勾配 (Conjugate Gradients)

- 直前の探索方向に対して共役な探索方向を求める

Algorithm 8.9 共役勾配法

Require: 初期パラメータ $\boldsymbol{\theta}_0$

Require: m 個の事例から成る訓練集合

初期化 $\boldsymbol{\rho}_0 = \mathbf{0}$

初期化 $g_0 = 0$

初期化 $t = 1$

while 終了条件を満たさない **do**

勾配を初期化する $\boldsymbol{g}_t = \mathbf{0}$

勾配を計算する: $\boldsymbol{g}_t \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$

以下を計算する $\beta_t = \frac{(\boldsymbol{g}_t - \boldsymbol{g}_{t-1})^\top \boldsymbol{g}_t}{\boldsymbol{g}_{t-1}^\top \boldsymbol{g}_{t-1}}$ (Polak-Ribière)

(非線形の共役勾配: β_t を必要に応じてゼロにリセットする．たとえば $k = 5$ など，t がある定数 k の倍数のとき．)

探索方向を計算する: $\boldsymbol{\rho}_t = -\boldsymbol{g}_t + \beta_t \boldsymbol{\rho}_{t-1}$

直線探索を実行する: $\epsilon^* = \text{argmin}_\epsilon \frac{1}{m} \sum_{i=1}^{m} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}_t + \epsilon \boldsymbol{\rho}_t), \boldsymbol{y}^{(i)})$

（真の二次コスト関数では，明示的に探索するのではなく，解析的に ϵ^* を求める）

更新を適用する: $\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{\rho}_t$

$t \leftarrow t + 1$

end while

8.6.3 BFGS

- ニュートン法の更新式

$$\boldsymbol{\theta}^* = \boldsymbol{\theta}_0 - \boldsymbol{H}^{-1}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_0)$$

  - $\boldsymbol{H}^{-1}$の計算コスト（？）は $O(k^3)$. 高い回避したい

- Broyden-Fletcher-Goldfarb-Shanno（GFGS）アルゴリズム
  - ヘッセ逆行列を低ランク行列 $\boldsymbol{M}_t$ で近似して更新する

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \epsilon^* \boldsymbol{M}_t \boldsymbol{g}_t$$

  - Limited Memory BFGS（L-BFGS）アルゴリズム

8.7.1 バッチ正規化（Batch Normalization）

- 各層のユニット数1のL層MLPを考える

$$\hat{y} = xw_1w_2w_2 \cdots w_L$$

- 重み $w_1, w_2, w_3, \ldots, w_l$ の勾配をそれぞれ $g_1, g_2, g_3, \ldots, g_l$ とすると, 単純な勾配降下法適用後の新しい出力は

$$\hat{y} = x(w_1 - \epsilon g_1)(w_2 - \epsilon g_2)(w_3 - \epsilon g_3) \cdots (w_L - \epsilon g_L)$$

- 各パラメータの勾配 g_l は他のパラメータを固定した下で計算されているので, 大局的には間違った更新を引き起こす可能性がある
  - たとえば上式の左辺に現れる $\epsilon^2 g_1 g_2 \prod_{i=3}^{L} w_i$ という項は $w_3, w_4, w_5, \ldots, w_L$ の絶対値がすべて1より大きいときに爆発的に大きくなってしまい. 適切な更新を邪魔する(?)
  - 2回微分による最適化手法で軽減可能(?)だが計算コストが大きい

8.7.1 バッチ正規化（Batch Normalization）

- 各層の出力を平均0, 分散1に正規化する

$$\boldsymbol{H}' = \frac{\boldsymbol{H} - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

$$\boldsymbol{\mu} = \frac{1}{m}\sum_i \boldsymbol{H}_{i,\cdot}, \qquad \boldsymbol{\sigma} = \sqrt{\delta + \frac{1}{m}\sum_i (\boldsymbol{H} - \boldsymbol{\mu})_i^2}$$

- 前の例（ $\hat{y} = xw_1w_2w_3 \ldots w\_L$ ）で $x \sim \mathcal{N}(0,1)$ とすると, 正規化を各層の出力にかけることでL-1層の出力は

$$h_{L-1} \sim \mathcal{N}(0,1)$$

となり, ネットワーク全体の計算はパラメータ w_{L-1} のみをもつ線形関数

$$\hat{y} = w_{L-1}\hat{h}_{L-1}$$

となり, 学習が容易になる（同時に下層のパラメータは効果を失う）

- 正規化がない場合は前述の $\prod_{i=2}^{L} w_i$ などの問題が残る

8.7.1 バッチ正規化（Batch Normalization）

- 前の例では, 正規化をすることで最終層以外のパラメータは意味を失う
  - ただし非線形関数をかけることで・・・
- 各層を正規化することで表現力が落ちてしまうので, 力を維持するために平均と標準偏差を制御する新たな学習パラメータを導入

$$\boldsymbol{\gamma H' + \beta}$$

- 「一度正規化->平均・標準偏差の制御」をすることで,
  - バッチ正規化以前は平均は下層の複雑な合成関数によって決まっていたものが
  - バッチ正規化後は単一のパラメータ $\boldsymbol{\gamma}$ によって決まる

- 各層間のパラメータの依存度を下げることで最適化しやすくしてるイメージ

8.7.2 座標降下法（Coordinate Descent）

- 座標降下法
  - 一つ（or 一部）のパラメータずつ更新を行う
  - local minimumにはたどり着く
  - 学習パラメータがきれいに分割できる場合に特に有効
- 例1）

$$\min_{\boldsymbol{H},\boldsymbol{W}} J(\boldsymbol{H},\boldsymbol{W}) = \sum_{i,j} \left|H_{i,j}\right| + \sum_{i,j} (\boldsymbol{X} - \boldsymbol{W}^T \boldsymbol{H})_{i,j}^2$$

  - このままでは非凸だが, H, Wそれぞれに対する関数とみれば凸
  - Wについて最適化を行ったのち, Hについて最適化をおこなう
- 例2）

$$\min_{x_1,x_2} f(x_1,x_2) = (x_1 - x_2)^2 + \alpha(x_1^2 + x_2^2), \alpha > 0$$

8.7.3 ポルヤック平均化（Polyak Averaging）

- 最適化の過程で得られたパラメータを平均化して最終的なパラメータとする

$$\widehat{\boldsymbol{\theta}}^{(t)} = \frac{1}{t}\sum_{i} \boldsymbol{\theta}^{(i)}$$

$$\widehat{\boldsymbol{\theta}}^{(t)} = \widehat{\boldsymbol{\theta}}^{(t-1)} - \frac{1}{t}(\widehat{\boldsymbol{\theta}}^{(t-1)} - \boldsymbol{\theta}^{(t)})$$

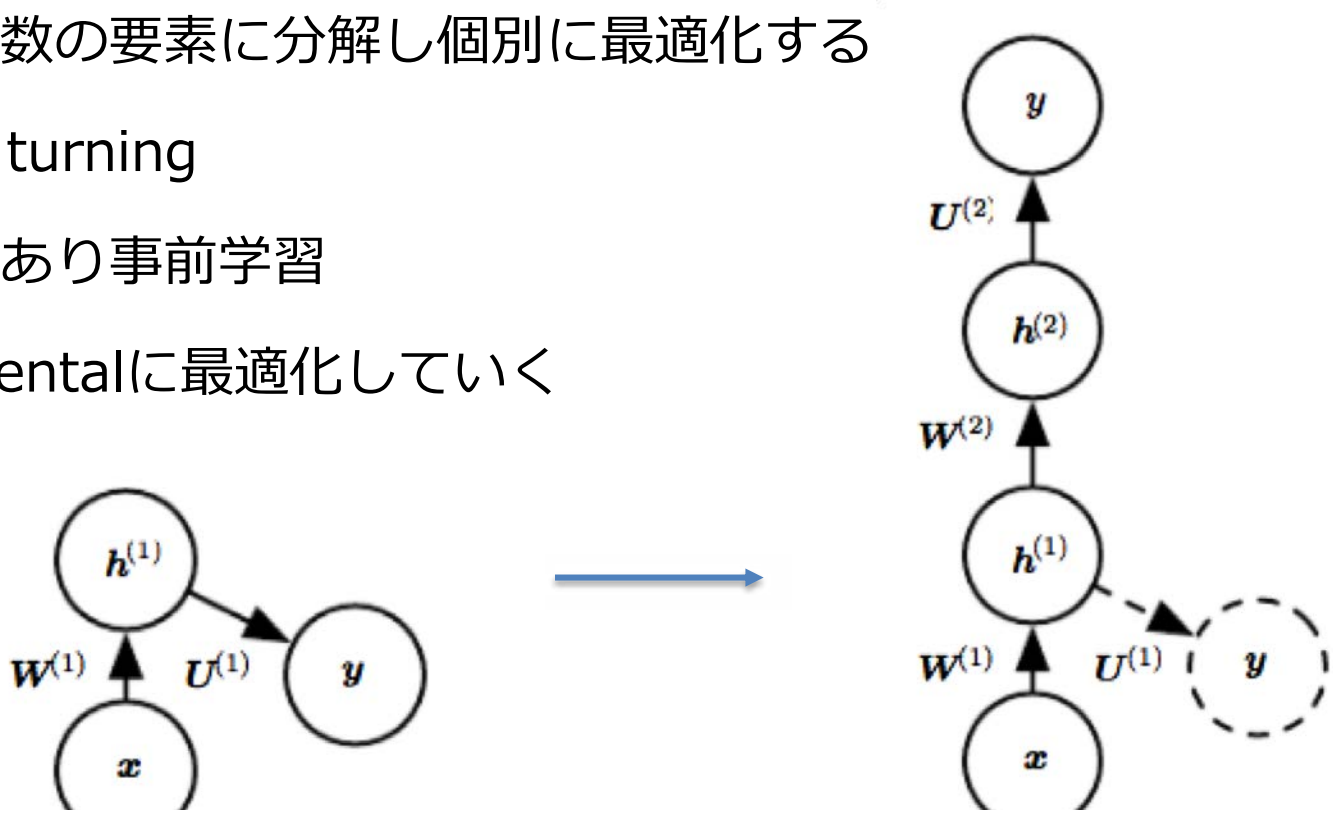
  - 谷で行き来しているときに平均化することでうまいこと谷底にたどり着くと期待
  - 凸関数に対する最適化では強い収束の保証あり
  - ニューラルで用いられるような非凸関数は凸凹なので，直近の値を重視するために移動平均を利用

$$\widehat{\boldsymbol{\theta}}^{(t)} = \alpha\widehat{\boldsymbol{\theta}}^{(t-1)} + (1-\alpha)\boldsymbol{\theta}^{(t)}$$

8.7.4 教師あり事前学習

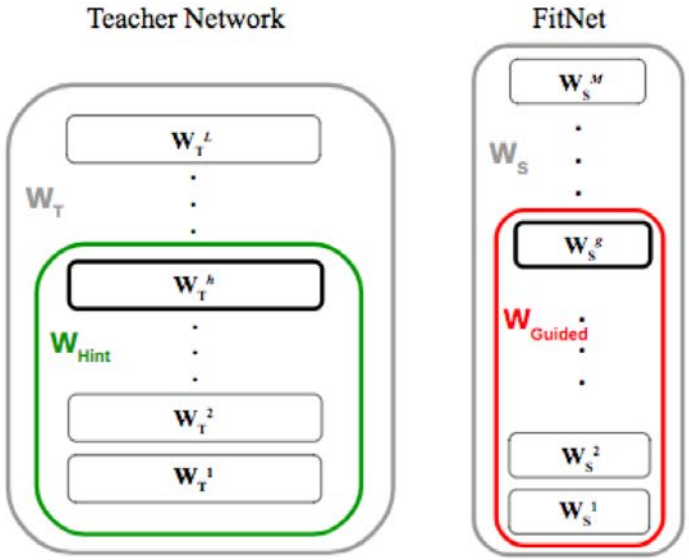
- モデルが複雑で最適化がむずかしい or タスク自体が難しい場合に効果的
  - モデルを徐々に複雑にしながら順に最適化していく

- 貪欲的（Greedy）アルゴリズム
  - モデルを複数の要素に分解し個別に最適化する
  - 最後にfine turning
  - 貪欲的教師あり事前学習
    - Incrementalに最適化していく



8.7.4 教師あり事前学習

- FitNets
  - 学習が比較的容易な(?)浅くて広いネットワークを学習（Teacher）
  - 学習が比較的難しい(?)深くて狭いネットワークをTeacherをguideに学習
    - Teacherの各層の重みを予測するように学習させる



$$\mathbf{W}^*_{\text{Guided}} = \underset{\mathbf{W}_{\text{Guided}}}{\text{argmin}}\ \mathcal{L}_{HT}(\mathbf{W}_{\text{Guided}}, \mathbf{W}_r)$$

Input: $\mathbf{W}_S, \mathbf{W}_T, g, h$
Output: $\mathbf{W}^*_S$

1: $\mathbf{W}_{\text{Hint}} \leftarrow \{\mathbf{W}_T{}^1, \ldots, \mathbf{W}_T{}^h\}$
2: $\mathbf{W}_{\text{Guided}} \leftarrow \{\mathbf{W}_S{}^1, \ldots, \mathbf{W}_S{}^g\}$
3: Intialize $\mathbf{W}_r$ to small random values
4: $\mathbf{W}^*_{\text{Guided}} \leftarrow \underset{\mathbf{W}_{\text{Guided}}}{\text{argmin}}\ \mathcal{L}_{HT}(\mathbf{W}_{\text{Guided}}, \mathbf{W}_r)$
5: $\{\mathbf{W}_S{}^1, \ldots, \mathbf{W}_S{}^g\} \leftarrow \{\mathbf{W}_{\text{Guided}}{}^{*1}, \ldots, \mathbf{W}_{\text{Guided}}{}^{*g}\}$
6: $\mathbf{W}^*_S \leftarrow \text{argmin}\, \mathcal{L}_{KD}(\mathbf{W}_S)$

8.7.5 最適化を支援するモデルの設計

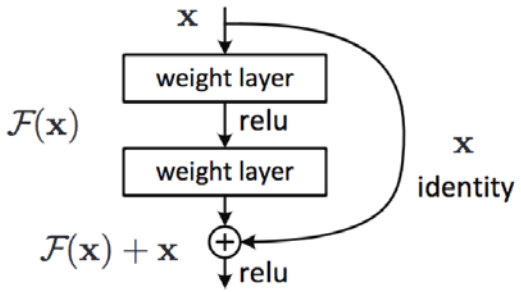
- 最適化をうまくおこなうためには, **最適化手法を向上させる**だけではなく**モデルを最適化しやすいように設計する**ことが重要
  - 実際に近年のNNの精度向上はモデル設計の工夫による部分が大きい
  - たとえばモメンタム法は1980年代からありいまだに最先端で使われている

8.7.5 最適化を支援するモデルの設計

- より線形に近い関数を使う
  - LSTM, ReLU, Maxout

8.7.5 最適化を支援するモデルの設計

- Skip connectionを使う
  - 勾配消失問題を軽減する
    - 重み・活性化関数をかけることなく伝播させることで勾配が伝わり(?)やすくなる
  - なし：$\frac{\partial}{\partial x}F(x)$
  - あり：$\frac{\partial}{\partial x}(F(x)+x)=\frac{\partial}{\partial x}F(x)+1$



- 中間層付近に追加で出力層を設ける
  - 中間層付近も出力に近くなるので勾配が大きくなる
  - マルチタスク学習も場合によっては同じような効果？

8.7.6 継続法（Continuation Methods）とカリキュラム学習

- 継続法（Continuation Methods）
  - パラメータの初期値をいい感じの領域におく
  - コスト関数を徐々に最適化が難しいものにしていく

$$J^{(i)}(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{\theta}' \sim \mathcal{N}(\boldsymbol{\theta}', \boldsymbol{\theta}, \sigma^{(i)2})} J(\boldsymbol{\theta}')$$

  - 最適化 -> ぼやけさす -> 最適化 -> ぼやけさす -> ・・・
  - 分散は徐々に小さくしていく（less-blurred）

8.7.6 継続法（Continuation Methods）とカリキュラム学習

- カリキュラム学習（Curriculum learning）
  - 簡単な概念の学習からスタートし，　徐々に難しい概念を学習させていく
  - Continuation methodsの一種
  - 人間がカリキュラムに沿って簡単なものから学習していくのと同じ

- RNNにおけるカリキュラム学習
  - Stochastic curriculum learning (”Learning to execute”, Zaremba et al. 2014)
  - 難しいタスクを含む学習データ（時間依存の長さ）の割合を徐々に上げていく

参考文献

- Deep Learning
  - Ian Goodfellow, Yoshua Bengio, Aaron Courville
  - 日本語版
    https://www.amazon.co.jp/%E6%B7%B1%E5%B1%A4%E5%AD%A6%E7%BF%92-Ian-Goodfellow/dp/4048930621