名古屋大学特別講義 2016年6月29日(水)

ディープラーニングによる自然言語処理 (技術編)

日本アイ・ビー・エム株式会社
東京基礎研究所
坪井 祐太
yutat@jp.ibm.com

ニューラルネットワーク技術詳細

- 目的関数
  - 誤差関数
- 目的関数の最小化
  - 勾配法
- 目的関数の微分計算
  - 誤差逆伝搬法
- 誤差の分解と対処手法
  - 推定誤差に効く手法
  - 最適化誤差に効く手法
- RNNの話題

目的関数: 誤差

本当の目的は、訓練データの誤差を減らすことではなく、真の分布の元での誤差を減らしたいことがポイント

- 教師あり学習の目的関数
  - $x \in X$: 入力, $y \in Y$: 出力
    - 入力xからyを予測したい問題設定
  - 真の目的関数: $L^*(\theta) = \int p(x,y)\ell_\theta(x,y)\,dx\,dy$
  - ℓ_θは事例ごとの損失関数（後述）
- 訓練データでの誤差
  - データ分布p(x,y)は普通わからないので訓練データN個: $D=\{(x^{(i)},y^{(i)})\}_{i=1}^{N}$を使って近似
  - $L^*(\theta) \approx L(\theta) = \frac{1}{N}\sum_{i=1}^{N}\ell_\theta\left(x^{(i)},y^{(i)}\right)$
- 学習 ≈ 訓練データでの誤差最小パラメータを得る
  - $\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, L(\theta)$

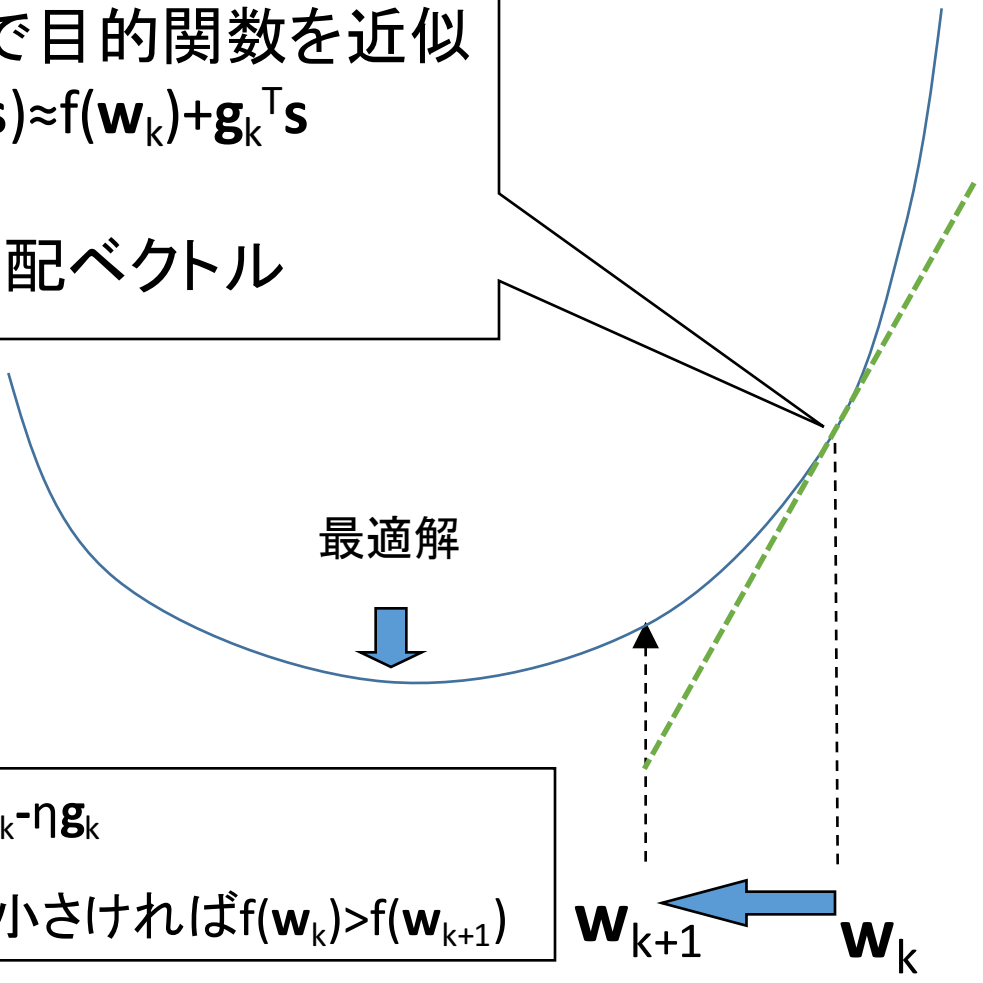
よく使われる損失関数

- $f_\theta(x, y)$を予測yに対するスコア関数（この講義では特にニューラルネットワーク）とする
- ソフトマックス損失
  - $\ell_\theta\left(x^{(i)}, y^{(i)}\right) = -\log \frac{\exp\left(f_\theta\left(x^{(i)}, y^{(i)}\right)\right)}{\sum_{\tilde{y}\in Y} \exp\left(f_\theta\left(x^{(i)}, \tilde{y}\right)\right)}$
    - 確率モデル$\mathrm{P}\left(y^{(i)} | x^{(i)}\right) = \frac{\exp\left(f_\theta\left(x^{(i)}, y^{(i)}\right)\right)}{\sum_{\tilde{y}\in Y} \exp\left(f_\theta\left(x^{(i)}, \tilde{y}\right)\right)}$と思うと負の対数尤度に相当
- ヒンジ損失
  - $\ell_\theta\left(x^{(i)}, y^{(i)}\right) = \max\left(0, 1 - f_\theta\left(x^{(i)}, y^{(i)}\right) + \max_{\tilde{y}\in Y\setminus y^{(i)}} f_\theta\left(x^{(i)}, \tilde{y}\right)\right)$
    - $\max_{\tilde{y}\in Y\setminus y^{(i)}} f_\theta\left(x^{(i)}, \tilde{y}\right)$は正解$y^{(i)}$を除いた中で最も高いスコア

関数の最小化：勾配法

直線で目的関数を近似
$f(\mathbf{w}_k+\mathbf{s})\approx f(\mathbf{w}_k)+\mathbf{g}_k^T\mathbf{s}$

$\mathbf{g}$は勾配ベクトル



$\mathbf{w}$を反復更新：$\mathbf{w}_{k+1}=\mathbf{w}_k-\eta\mathbf{g}_k$

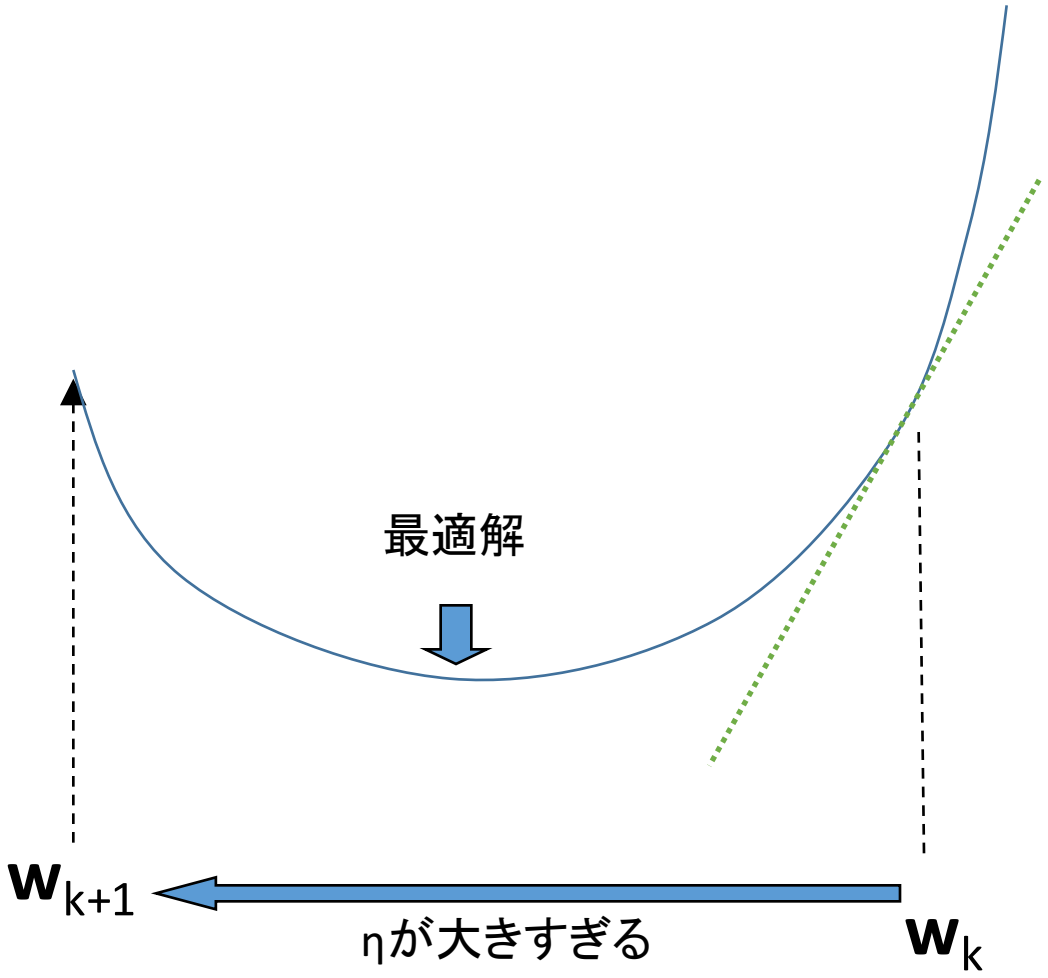
ηは更新率→ηが十分小さければ$f(\mathbf{w}_k)>f(\mathbf{w}_{k+1})$

関数の最小化：勾配法

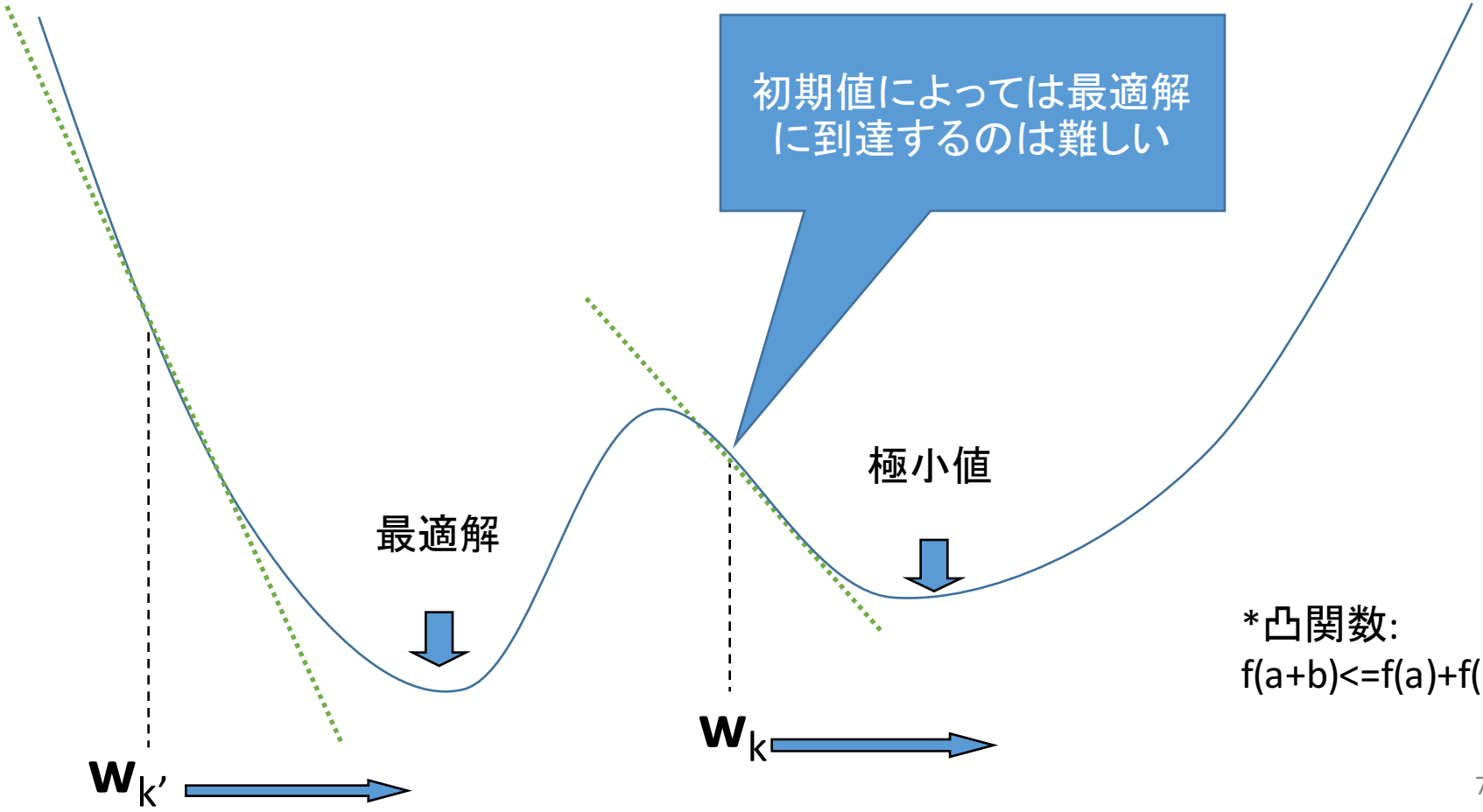
wを反復更新

$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \mathbf{g}_k$

ηは更新率



関数の最小化：非凸関数の場合



*凸関数:
f(a+b)<=f(a)+f(b)

ミニバッチ化による確率的勾配法（SGD）

- 訓練データ全てを使って勾配計算・パラメータ更新するのは時間がかかりすぎる
  - $g_{full} = \frac{1}{N}\sum_{i=1}^{N}\frac{\partial \ell_\theta\left(x^{(i)},y^{(i)}\right)}{\partial \theta}$
- 1事例ごとの勾配計算ではGPU計算資源を使い切れない
  - $g_{online} = \frac{\partial \ell_\theta\left(x^{(i)},y^{(i)}\right)}{\partial \theta}$
- ランダムサンプルM個をまとめて計算（Mは通常16個～256個）
  - $g_{minibatch} = \frac{1}{M}\sum_{j=1}^{M}\frac{\partial \ell_\theta\left(x^{(j)},y^{(j)}\right)}{\partial \theta}$

誤差逆伝搬法(Back Propagation)

- ニューラルネットワーク：合成関数
  - $f(g(x))$

例：活性化関数
$f(h) = \max(0, h)$

例：行列・ベクトル積
$g(x) = \boldsymbol{W}x$

- ニューラルネットワークの微分：導関数の積
  - 連鎖律
  - $\dfrac{\partial f(g(x))}{\partial x} = \dfrac{\partial f}{\partial g}\dfrac{\partial g}{\partial x}$

xが微小変化した時のfの変化

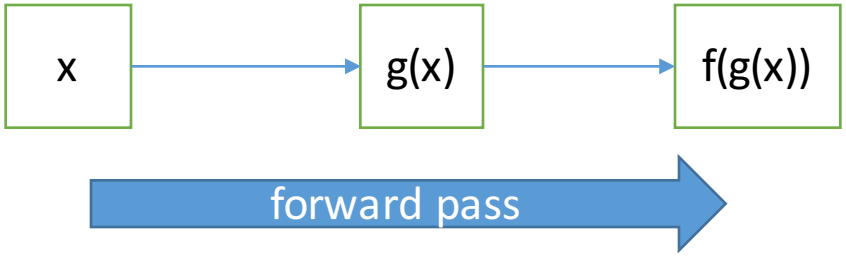
例：活性化関数*
$$\frac{\partial f}{\partial h} = \begin{cases} 0, & h \leq 0 \\ 1, & h > 0 \end{cases}$$

例：行列・ベクトル積
$$\frac{\partial g}{\partial x} = \boldsymbol{W}$$

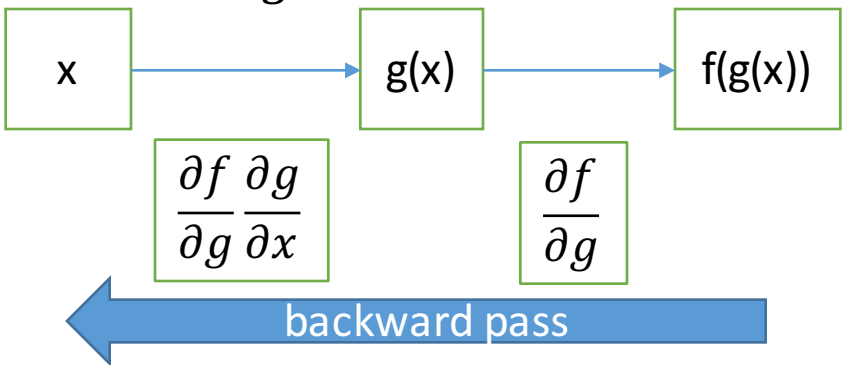
*正確には劣微分(subgradient)

誤差逆伝搬法(Back Propagation)

- ニューラルネットワーク：合成関数
  - $f(g(x))$



- ニューラルネットワークの微分：導関数の積
  - $\frac{\partial f(g(x))}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$



誤差逆伝搬法を学ぶ意義

- ディープラーニングライブラリを利用すれば、よく使われる関数の微分計算は用意されている
- 最近は自動微分（アルゴリズム微分）も利用可能（Threano, etc）
  - 定義された関数（プログラム）を解析し、遷移律を適用して自動的に偏導関数を計算するプログラムを導出してくれる
- 実装する必要性は少ないが、ニューラルネットワークのモデルの理解に役立つ
  - 誤差の伝播→各層の学習がどう進むか

誤差逆伝搬を通じたNN理解例

- 例1: $f\left(g(g(g(x)))\right), g(x) = \boldsymbol{W}x, \frac{\partial g}{\partial x} = \boldsymbol{W}$
  - $\frac{\partial f\left(g(g(g(x)))\right)}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial g}\frac{\partial g}{\partial g}\frac{\partial g}{\partial x} = \frac{\partial f}{\partial g}\boldsymbol{W}\boldsymbol{W}\boldsymbol{W} = \frac{\partial f}{\partial g}\boldsymbol{W}^3$
  - w=0.01の場合 w^3=0.000001
    - 誤差が伝わるうちに極端に減衰する(Vanishing Gradient)
  - w=100の場合 w^3= 1000000
    - 誤差が伝わるうちに極端に増幅される (Exploding Gradient)

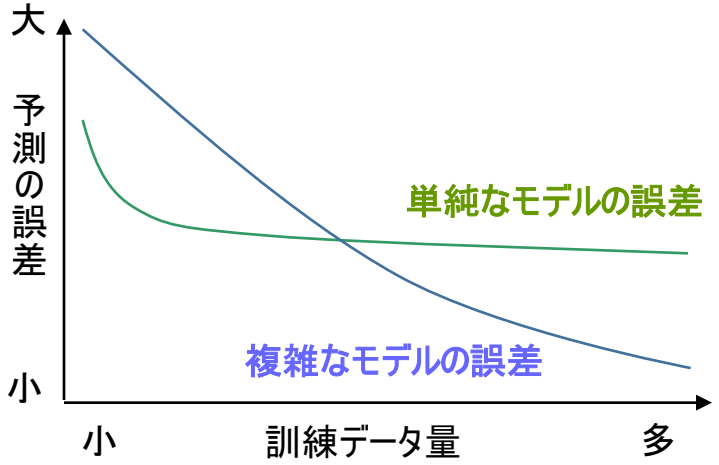
  →深いニューラルネットワークの学習は難しい
- 例2: $f(g(x) + x)$
  - $\frac{\partial f(g(x)+x)}{\partial x} = \frac{\partial f}{\partial g}\left(\frac{\partial g}{\partial x} + \frac{\partial x}{\partial x}\right) = \frac{\partial f}{\partial g}(\boldsymbol{W} + 1)$
  - wが小さくても誤差は伝わる(Vanishing Gradientの回避)
    - w=10^{-6}の場合でも$\frac{\partial f}{\partial g}$($10^{-6}$+1)
  - ただし、Exploding Gradientは回避できない

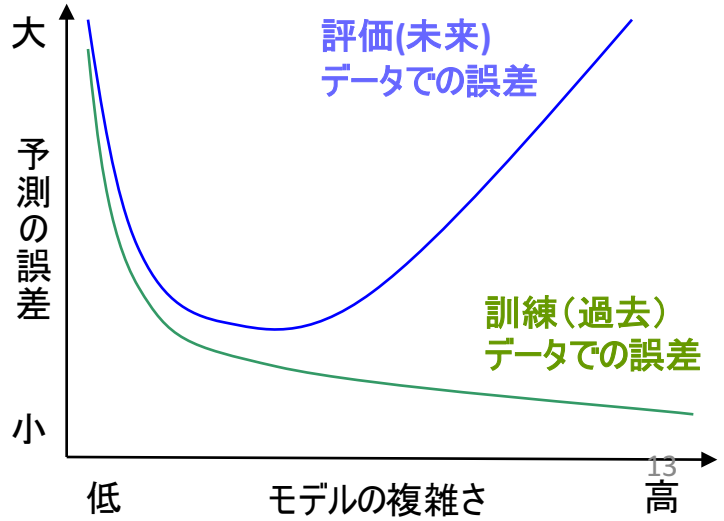
3層 NNのイメージ

Skip Connection

機械学習の基礎：モデルの表現力とバラツキのトレードオフ（Bias Variance Tradeoff）

- 表現力の高いモデル（複雑なモデル）は訓練データの誤差を減らせる
- 限られた訓練データ量では表現力の高いモデルは学習結果のバラツキが大きい
- 極端に複雑なモデル例：個々のデータを丸覚え
  - 訓練データ誤差は0になるが、訓練データと同じデータがこないと回答できない





どの誤差に効く手法かを意識すると、取り組んでいる課題に手法を取り入れるべきかの判断材料に(多少は)なる

- 機械学習における誤差=
  近似(モデル)誤差＋推定(サンプル)誤差＋最適化誤差

対処法:
モデルの表現力を増す

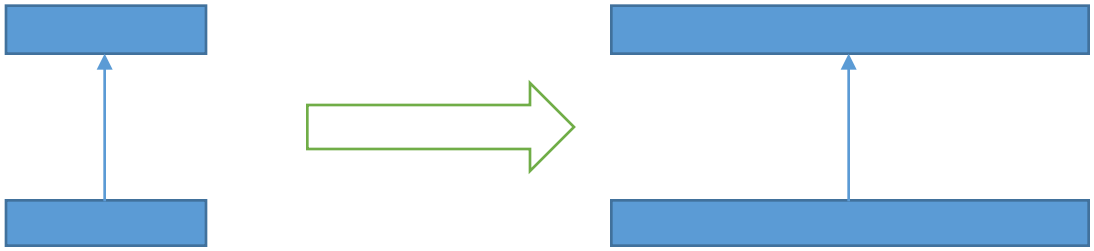
対処法:
訓練データを増やす・モデルのバラツキを抑える

対処法:
洗練された最適化方法を採用する

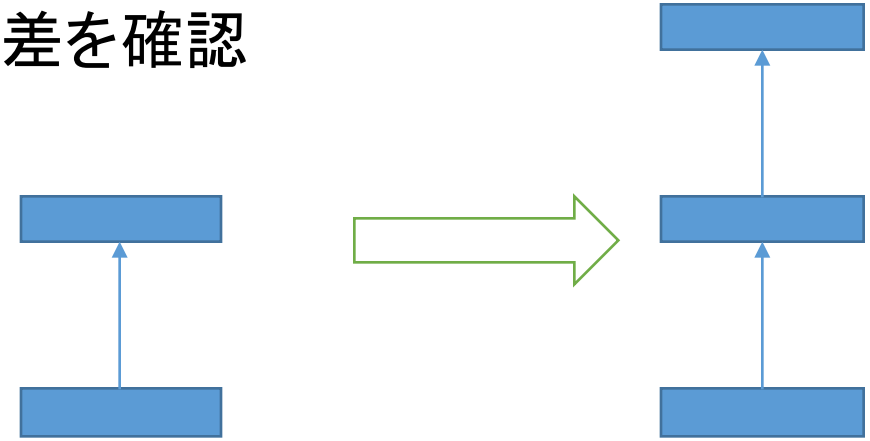
- 例
  - 近似誤差を減らすためのモデル: 隠れ層を深くする・隠れ変数を増やす, etc
  - 推定誤差を減らすためのモデル:L2正則化, パラメータ共有, ドロップアウト, etc
  - 最適化誤差を減らすためのモデル:LSTM, Batch Normalization, etc.

近似(モデル)誤差に効く手法

- 隠れ変数の数(幅)を増やして訓練データの誤差を確認



- 隠れ層を深さ増やして訓練データの誤差を確認



ディープにすると最適化誤差が増えるのでまずは幅を広げるのがお勧め

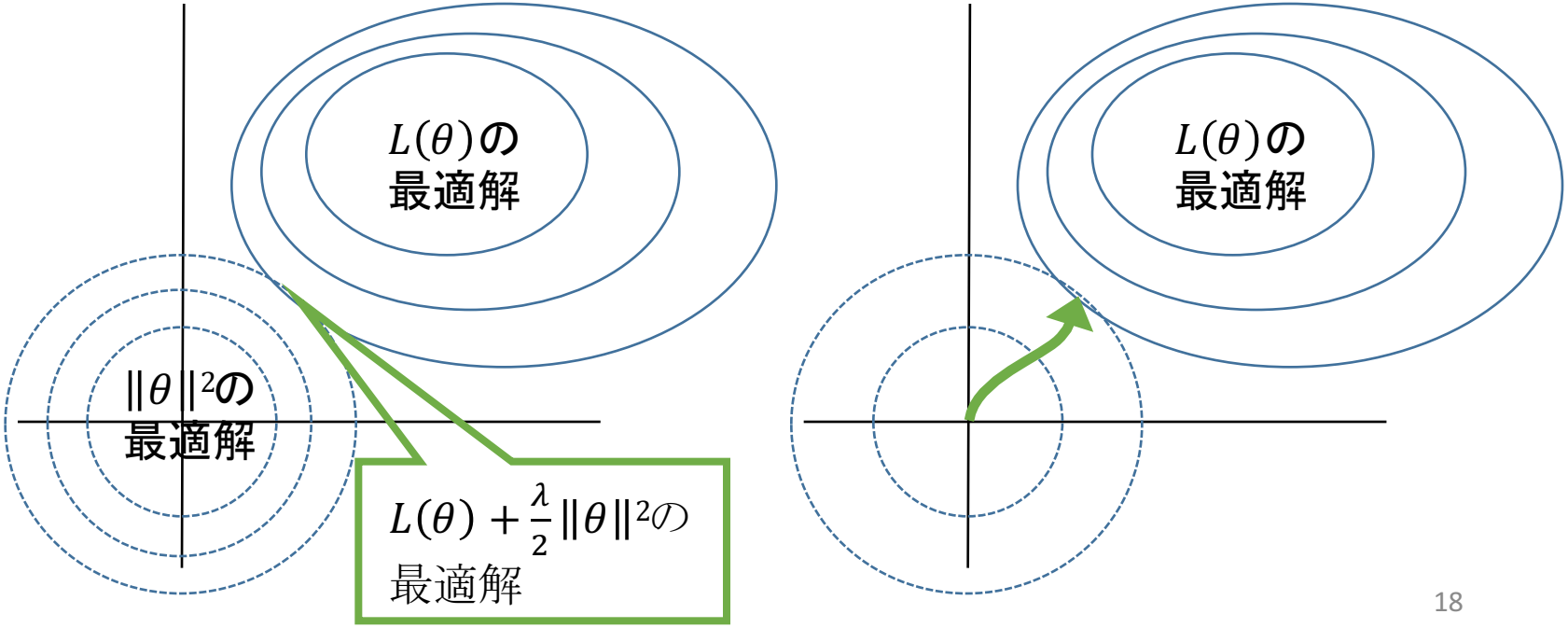
推定誤差に効く手法

L2正則化

- パラメータの値が大きすぎるとExploding Gradientが発生
- パラメータが大きくなりすぎることにペナルティを与える項を目的関数に追加
- $L(\theta) + \frac{\lambda}{2}\|\theta\|^2$
  - λは正則化の強さを決めるパラメータ
    - 次に紹介するDropoutと併用する場合は10^{-6}などかなり小さな値にすることが多い
  - Weight decayとも呼ばれる
- L2正則化付き勾配法での更新式
  - $\theta_{k+1}=\theta_k - \eta\left(\frac{\partial L(\theta_k)}{\partial \theta_{\partial L(\theta_k)}} + \frac{\lambda}{2}\frac{\partial\|\theta_k\|^2}{\partial\theta}\right)=\theta_k - \eta\left(\frac{\partial L(\theta_k)}{\partial\theta} + \lambda\theta_k\right)$
    $= (1-\eta\lambda)\theta_k - \eta\frac{\partial L(\theta_k)}{\partial\theta}$
  - $(1-\eta\lambda)$倍パラメータを小さくする効果

Early Stop

- 検証データ(訓練データとしては使わない学習用データ)を使って学習中のNNを評価し、性能が上がらなくなったら早めに停止する
- 正則化と似た効果



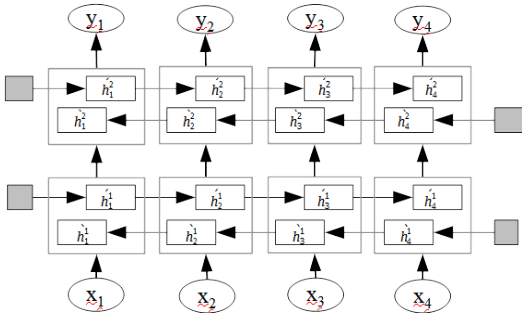
アンサンブル法

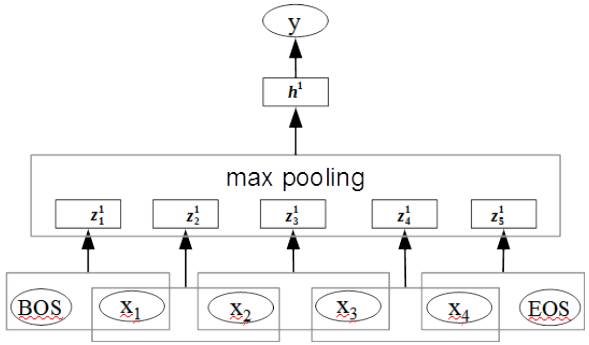
- モデルのバラツキを抑える直接的な方法
- 異なる設定で学習したNNの結果を統合
  - 投票式
  - （スケールが同じなら）スコアを平均する
  - （同じ形式のモデルなら）パラメータを平均する
- 最高性能を出しているディープラーニング論文の結果はほとんどアンサンブル法を使用
  - 翻訳精度(BLUEスコア)の例[Sutskever et al. 2014]
    - 探索よりもアンサンブル数の方が性能向上に効果的

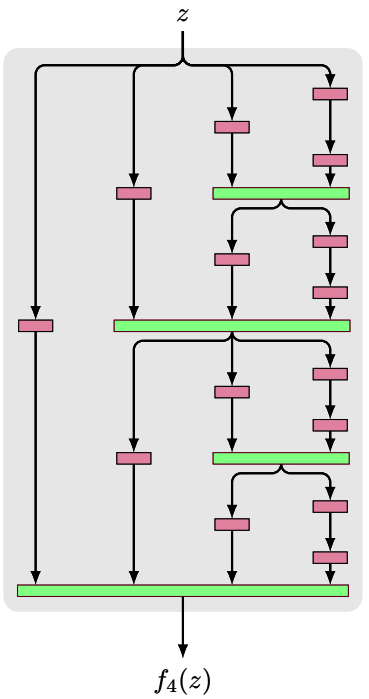
アンサンブル	ビーム探索幅	BLUE
N/A	12	30.59
5	N/A	33.00
2	12	33.27
5	2	34.50

パラメータ共有

- RNN, CNNとも場所によらない共通のパラメータを持つ
  - 一つの場所に特化しないようにパラメータを制限







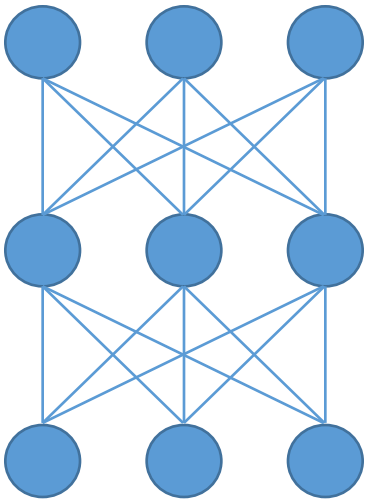
- Larsson et al., FractalNet: Ultra-Deep Neural Networks without Residuals, 2016
  - 再帰的に同じネットワーク構造を共有

ドロップアウト [Srivastava et al., 2014]

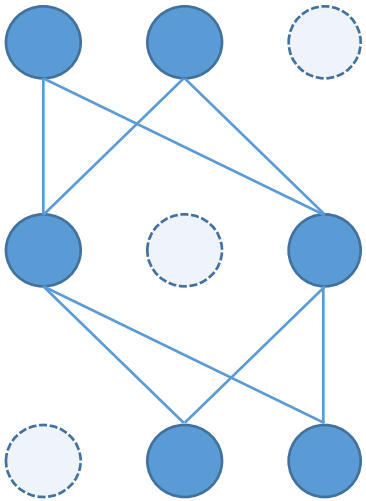
- 訓練時にドロップアウト確率(1-p)で隠れ変数hを0に置換
  - 事例毎に異なるネットワーク構造を評価・更新していることに相当
  - テスト時には学習結果パラメータをp倍することで、擬似的に複数のネットワークの幾何平均で予測していることに相当

→ アンサンブル法

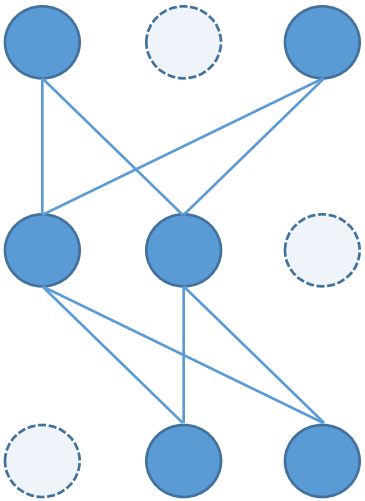
ドロップアウトなし



ドロップアウト例1



ドロップアウト例1



最適化誤差に効く手法

初期値

- パラメータ行列は正規分布または一様分布からサンプリングすることが一般的
  - パラメータのスケール（分散）が重要
    - 小さすぎると誤差が伝わらない（Vanishing Gradient）
    - 大きすぎると誤差が発散(Exploding Gradient)
- 全ての層で活性化関数の分散と勾配の分散が等しくなるようにするヒューリスティックス
  - Xavier initialization [Glorot and Bengio, 2010]
    - $W_{ij} \sim \mathrm{Uniform}\left(-\frac{6}{\sqrt{a}}, \frac{6}{\sqrt{a}}\right)$
    - a=#input + #output
  - ReLU用初期値 [He et al., 2015]
    - $W_{ij} \sim N\left(0, \sqrt{\frac{2}{\#input}}\right)$

Batch Normalization [Ioffe and Szegedy, 2015]

- 隠れ変数を平均0, 分散1に変換する層を追加
  - $h^{l+1} = \frac{h^l - \mu}{\sigma}$
- 平均μ、分散σ^2はミニバッチM個内で推定
  - $\mu = \frac{1}{M}\sum_j h^{l(j)}$
  - $\sigma = \sqrt{\varepsilon + \frac{1}{M}\sum_j (h^{l(j)} - \mu)^2}$
  - εは$\sqrt{0}$を防ぐための微小な値
- すべての層の隠れ変数が同じ範囲だと微分も同じ範囲になりやすい
  - 再掲: $\frac{\partial f(g(x))}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$

例：活性化関数

$$\frac{\partial f}{\partial h} = \begin{cases} 0, & h \leq 0 \\ 1, & h > 0 \end{cases}$$

例：行列・ベクトル積

$$\frac{\partial g}{\partial x} = \boldsymbol{W}$$

Batch Normalization and beyond

- BNによって表現力が落ちないようにスケールとバイアスを加える
  - $\alpha \frac{h^l - \mu}{\sigma} + \beta$
  - αとβはスカラーでW行列より学習が容易
- ミニバッチが不要なBNの拡張：NormProp [Arpit et al. 2016]
  - RNNなどは系列の長さが可変長だとミニバッチサイズもばらついてしまいBNが使いづらい
  - 入力を平均0, 分散1の正規分布を仮定して、出力も平均0, 分散1の正規分布になるように関数を解析的に変更

Long Short-Term Memory (LSTM)

- $$\begin{pmatrix} \boldsymbol{i}_t \\ \boldsymbol{f}_t \\ \boldsymbol{o}_t \\ \boldsymbol{g}_t \end{pmatrix} = \begin{pmatrix} sigmoid(\boldsymbol{W}_i[\boldsymbol{x}_t;\boldsymbol{h}_{t-1}]+\boldsymbol{b}_i) \\ sigmoid(\boldsymbol{W}_f[\boldsymbol{x}_t;\boldsymbol{h}_{t-1}]+\boldsymbol{b}_f) \\ sigmoid(\boldsymbol{W}_o[\boldsymbol{x}_t;\boldsymbol{h}_{t-1}]+\boldsymbol{b}_o) \\ tanh(\boldsymbol{W}_g[\boldsymbol{x}_t;\boldsymbol{h}_{t-1}]+\boldsymbol{b}_g) \end{pmatrix}$$
- $\boldsymbol{c}_t = \boldsymbol{f}_t * \boldsymbol{c}_{t-1} + \boldsymbol{i}_t * \boldsymbol{g}_t$
- $\boldsymbol{h}_t = \boldsymbol{o}_t * \tanh(\boldsymbol{c}_t)$
- LSTMではVanishing Gradientが起こりにくい理由
  - 簡単のためf=1の場合:
    $\boldsymbol{h}_t = \boldsymbol{h}_0 + \boldsymbol{o}_t * \tanh(\boldsymbol{i}_1 * \boldsymbol{g}_1 + \boldsymbol{i}_2 * \boldsymbol{g}_2 + \cdots + \boldsymbol{i}_t * \boldsymbol{g}_t)$
  - 時刻1へのshort cutがある
    $$\frac{\partial \boldsymbol{h}_t}{\partial x_1} = \frac{\partial \boldsymbol{h}_0}{\partial x_1} + \frac{\partial \boldsymbol{o}_t}{\partial x_1} * \frac{\partial tanh}{\partial(\boldsymbol{i}_1*\boldsymbol{g}_1+\boldsymbol{i}_2*\boldsymbol{g}_2+\cdots+\boldsymbol{i}_t*\boldsymbol{g}_t)} \left(\frac{\partial \boldsymbol{i}_1 * \boldsymbol{g}_1}{\partial x_1} + \cdots \right)$$

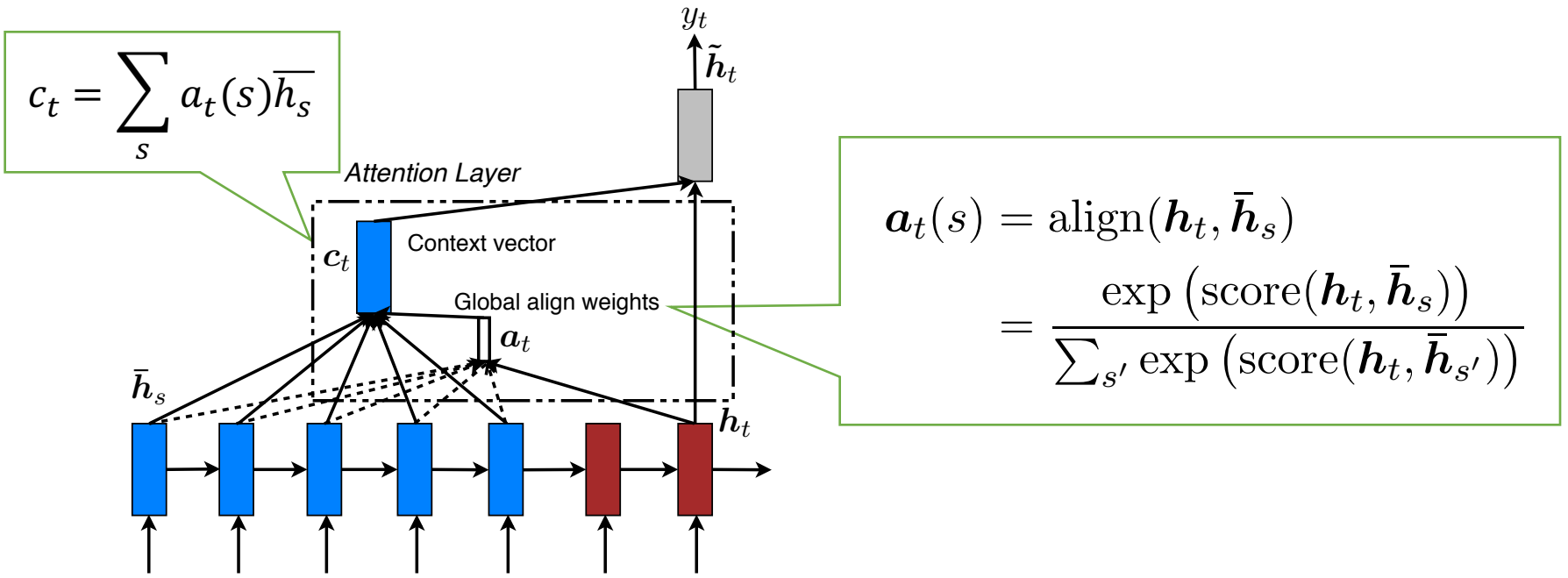
x_1を変化させた時の時刻tのh_tへの影響が直接的

LSTMに関連した重要な手法

- forget gate fのバイアスbは1に初期化する
  - Jozefowics et al. “An Empirical Exploration of Recurrent Network Architectures”, 2015
  - $\boldsymbol{f}_t = \frac{1}{\left(1+exp\left(-\boldsymbol{W}_f[\boldsymbol{x}_t;\boldsymbol{h}_{t-1}]-\boldsymbol{b}_f\right)\right)}$
  - b=1に初期化→fが1になりやすい→初期にvanishing gradientが起きにくい
- メモリセルcへのdropoutは更新分のみに適用する
  - Semeniuta et al., “Recurrent Dropout without Memory Loss”, 2016
  - $\boldsymbol{c}_t = \boldsymbol{f}_t * \boldsymbol{c}_{t-1} + dropout(\boldsymbol{i}_t * \boldsymbol{g}_t)$
  - 勧められないdropoutの適用
    - $\boldsymbol{c}_t = dropout(\boldsymbol{f}_t * \boldsymbol{c}_{t-1} + \boldsymbol{i}_t * \boldsymbol{g}_t)$
    - $\boldsymbol{c}_t = dropout(\boldsymbol{f}_t * dropout(\boldsymbol{f}_{t-1} * \boldsymbol{c}_{t-2} + \boldsymbol{i}_{t-1} * \boldsymbol{g}_{t-1}) + \boldsymbol{i}_t * \boldsymbol{g}_t)$
    - $\cdots$
    - t回dropoutを適用していることになり、0になる可能性が高い

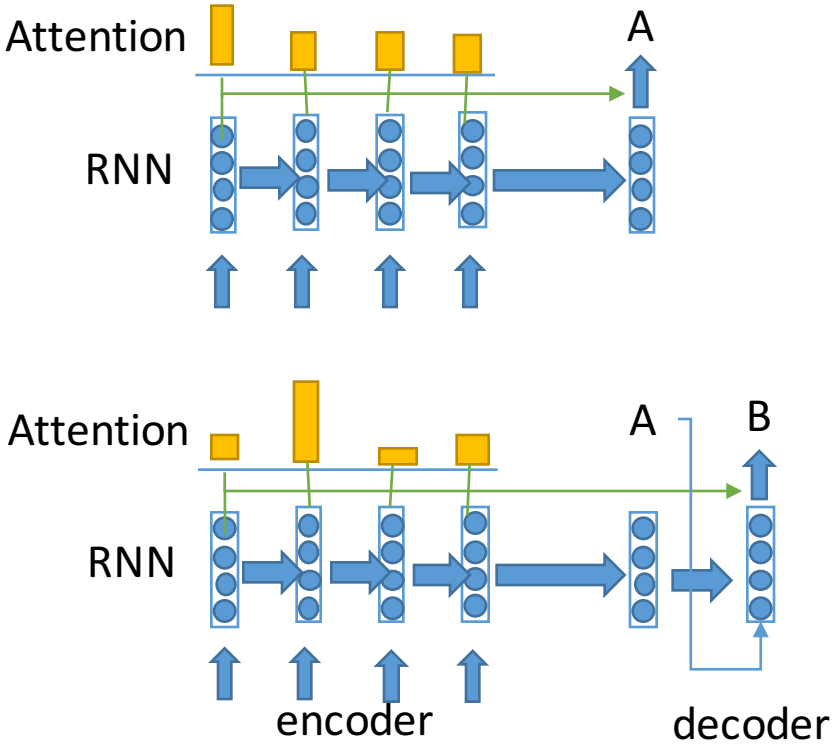
ソフトアテンションモデル

- 図・式はLuong et al., “Effective Approaches to Attention based Neural Machine Translation”, 2015 より
- 入力列の中で注目の仕方を学習



アテンションとRNNによる生成の組み合わせ

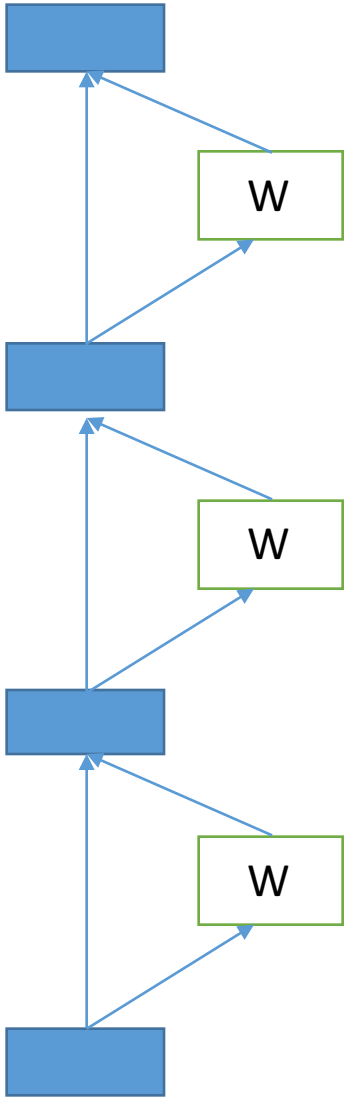
- 入力列から出力層へのshortcutを作っているとも考えられる(Vanishing Gradientが起こりにくい）



その他のshortcut (skip connection)を用いる手法

- Residual Networks [He et al., 2016]
  - $f(g(x)+x)$
- Highway Networks [Srivastava et al., 2015]
  - $f(T(x)g(x)+(1-T(x))x)$
    - $T(x)=\mathrm{sigmoid}(\boldsymbol{W}x+b)$
    - 重み付きのshortcut

- ただし、shortcutはアンサンブル効果も指摘されている [Veit et al., 2016]
  - 複数の深さの混ぜ合わせと見ることもできる

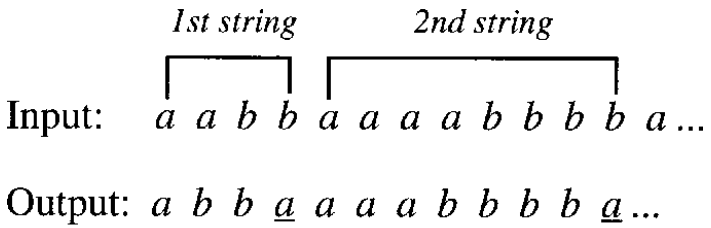


RNNの話題

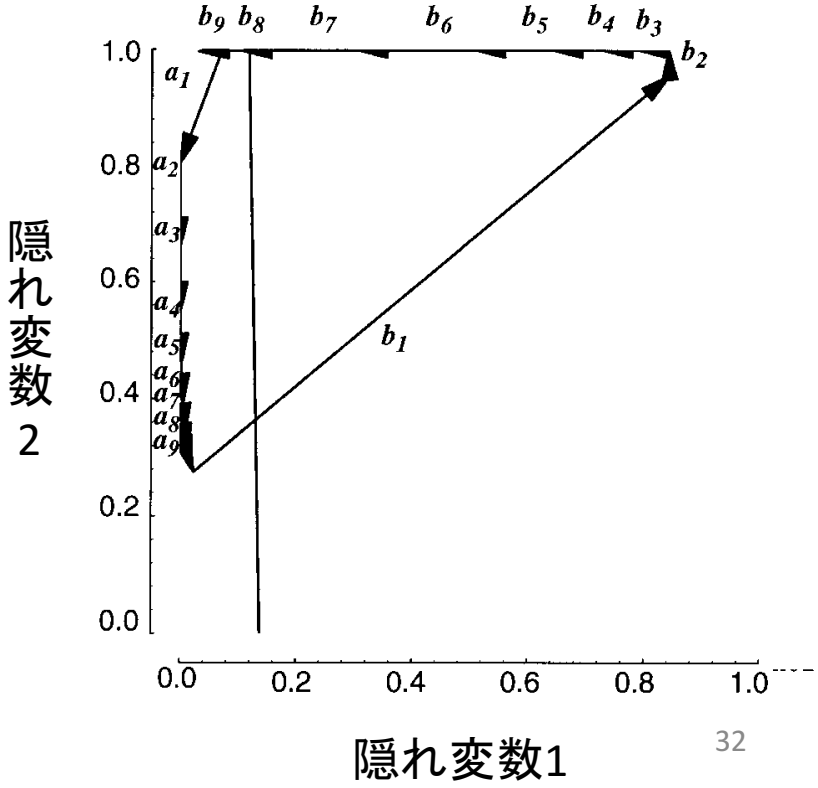
- RNNの可視化
- RNN学習時に使われるヒューリスティックス
- RNNはパラメータより状態保持にメモリが必要
- 言葉を生成する手法は出力層の計算・空間量が重い

RNNは数を数えられるか

- Rodriguez et al., “A Recurrent Neural Network that Learns to Count”, 1999（図は論文から引用）
  - aとbの数が同じ（$a^n b^n$）データをRNNで学習できるか

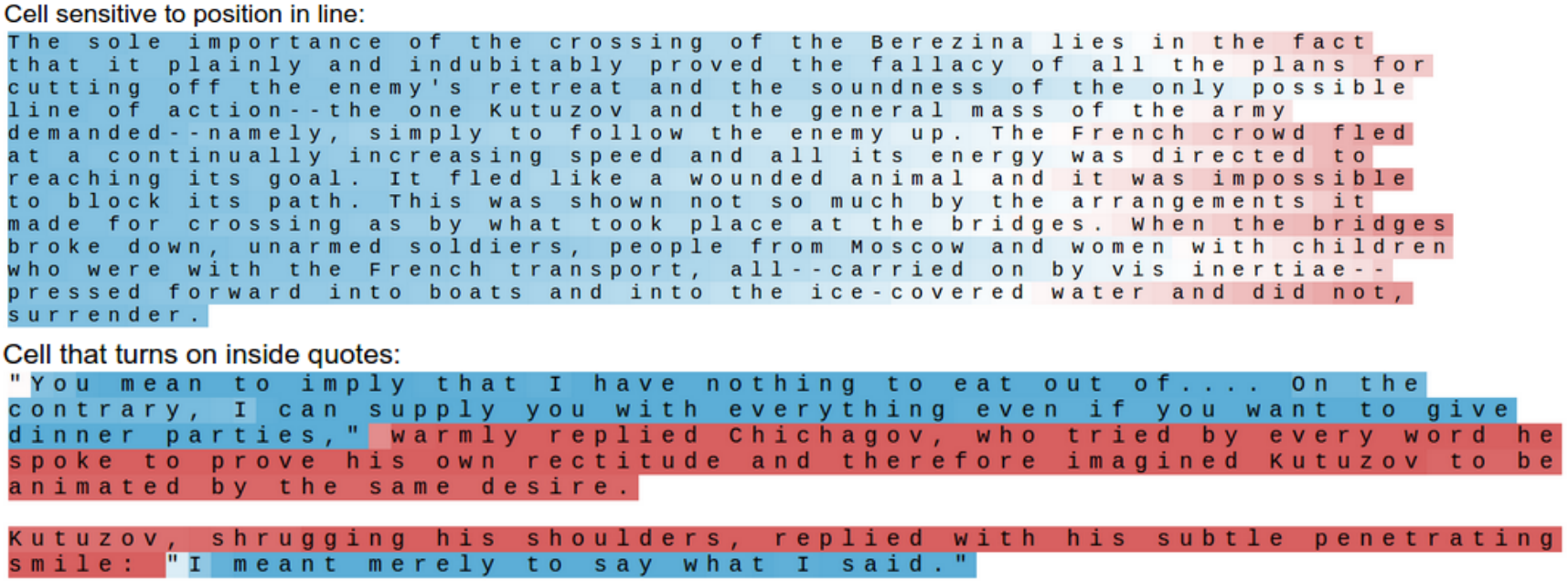


  - 2つの隠れ変数を使って数えたケースを実験的に確認　n=9の場合→
    - うまく解釈可能な結果を選んだ点に注意



RNN文字言語モデルの可視化

- Karpathy et al., “Visualizing and Understanding Recurrent Networks”, 2016. (図は論文より引用)
- LSTMのセルの値を可視化（うまく解釈可能な結果を選んだ点に注意）
  - 文末・引用符に反応するセル



RNNの学習時によく使われるヒューリスティックス

- Truncated Backpropagation Through Time (BPTT) (Elman (1990) , Mikolov et al., 2010)
  - 誤差逆伝播をFステップ毎にB時刻分行う
  - for t in 1…T
    - forwardprop: $h_t = RNN(h_{t-1}, xt)$
    - if t % F == 0 then
      - for s in t … t – B; backprop
    - end
  - exploding / vanishing gradientに有効
- gradient norm clipping (Pascanu et al., 2013)
  - 勾配ベクトルgのノルムに閾値を設けて、超えたらスケールする
    - if $\|\boldsymbol{g}\| \geq \text{threshold}$ then
      - $\boldsymbol{g} = \frac{\text{threshold}}{\|\mathbf{g}\|}\boldsymbol{g}$
  - exploding gradientに有効

RNN実装の課題: メモリ使用量

- 入力・隠れ変数の数H=256, 出力サイズ|Y|=10K
- ミニバッチサイズB=32, 長さT=64
- RNN: $\mathrm{y_t} = \mathrm{o}(\mathbf{W}_\mathrm{o}\mathrm{f}(\mathbf{W}_\mathrm{r}[\boldsymbol{x}_t; \boldsymbol{h}_{t-1}]))$
- パラメータ数:
  - $|W_r|$= H * 2H = 128K
  - $|W_o|$=H * |Y| = 2500K
- Backprop用状態変数
  - H * B * T = 512K
  - |Y| * B * T = 20000K

RNNはパラメータよりBackprop用の状態保持にメモリが必要

単語を出力するモデルの場合 出力層のメモリ使用量が問題となる

- 出力サイズ|Y|=800K (頻度3以上の単語のみ)
  - 1 billion word language modeling benchmark [Chelba et al., 2013] https://github.com/ciprian-chelba/1-billion-word-language-modeling-benchmark
  - 出力層パラメータ数: $|W_o|$=H * |Y| = 194M
  - 出力層状態変数 |Y| * B * T = 1.464G
- 最新GPUでもメモリ搭載量は12-16GB程度
  - 出力層の状態変数 を抑える手法が必要

出力層の状態変数 を抑える手法

- 階層ソフトマックス（Hierarchical Softmax）
  Goodman 2001, Mikolov et al. 2011
  - yをクラスタリングし、クラスタc(y)を定義
  - 階層化: p(y|x) = p(c(y)|x) p(y|c(y))
  - クラスタ数を$\sqrt{|Y|}$とすれば$2\sqrt{|Y|} \ll$ |Y|に抑えられる
  - クラスタは頻度などで決定
- サンプリング法 Jozefowicz et al. 2016, Ji et al. 2016
  - $\ell_\theta\left(x^{(i)}, y^{(i)}\right) = -\log \frac{\exp\left(f_\theta\left(x^{(i)}, y^{(i)}\right)\right)}{\exp\left(f_\theta\left(x^{(i)}, y^{(i)}\right)\right) + \sum_{\tilde{y} \in S} \exp\left(f_\theta\left(x^{(i)}, \tilde{y}\right)\right)}$
  - $S \in Y \setminus y^{(i)}$: 全出力を使わずに部分集合を使用
    - 部分集合は頻度に基づきサンプリングすることが多い
  - 学習時のメモリ・計算量を減らす手法

文字単位で予測する手法

- 単語単位: |Y| = 単語異なり数 1万以上
  - 未知語の問題（通常は低頻度語を未知語として学習）
  - 語形変化を扱えない（wordとwordsは別々のシンボル）
- 文字単位: |Y| = 文字異なり数
  - 訓練データにでてこない文字は稀、語形変化を学習できる可能性
- Chung et al. “A Character-Level Decoder without Explicit Segmentation for Neural Machine Translation”, 2016.
  - En-Cs, En-De, En-Fi で最先端の性能を達成

まとめ

- RNNの可視化
- RNN学習時に使われるヒューリスティックス
- RNNはパラメータより状態保持にメモリが必要
- 言葉を生成する手法は出力層の計算・空間量が重い

オススメの教科書

- Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016.
  - online version (free): http://www.deeplearningbook.org/

参考文献

- Ilya Sutskever, Oriol Vinyals, and Quoc V. V Le. “Sequence to sequence learning with neural networks”. NIPS 2014.
- Gustav Larsson, Michael Maire, and Gregory Shakhnarovich. “FractalNet: Ultra-Deep Neural Networks without Residuals”. arXiv:1605.07648, 2016.
- Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. JMLR, 15(1), 2014.

参考文献

- Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”, In Proc. of AISTATS 2010.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification”, arXiv:1502.01852, 2015.
- Sergey Ioffe and Christian Szegedy. “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”. In Proc. of ICML 2015.
- Devansh Arpit, Yingbo Zhou, Bhargava U. Kota, Venu Govindaraju. “Normalization Propagation: A Parametric Technique for Removing Internal Covariate Shift in Deep Networks”. In Proc. of ICML 2016.
- Rafal Jozefowicz, Wojciceh Zaremba, and Ilya Sutskever. “An Empirical Exploration of Recurrent Network Architectures”, In Proc. of ICML 2015.
- Stanislau Semeniuta, Aliaksei Severyn, Erhardt Barth. “Recurrent Dropout without Memory Loss”. arXiv:1603.05118, 2016.
- Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. “Effective Approaches to Attention-based Neural Machine Translation”, In Proc. of EMNLP 2015.
- Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. “Identity Mappings in Deep Residual Networks”. arXiv:1603.05027, 2016.
- Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. “Training Very Deep Networks”. NIPS 2015.
- Andreas Veit, Michael Wilber, Serge Belongie, “Residual Networks are Exponential Ensembles of Relatively Shallow Networks”, arXiv:1605.06431, 2016.

参考文献

- Paul Rodriguez , Janet Wiles, and Jeffrey L. Elman. “A Recurrent Neural Network that Learns to Count”. Connection Science 11 (1), 1999.
- Andrej Karpathy, Justin Johnson, and Li Fei-Fei. “Visualizing and Understanding Recurrent Networks”. In Proc. of ICLR2016 Workshop.
- Jeffrey L. Elman. “Finding structure in time”. Cognitive science, 14(2), 1990.
- Tomas Mikolov, Martin Karafiat, Kukas Burget, Jan “Honza” Cernocky, Sanjeev Khudanpur : Recurrent Neural Network based Language” In Proc. of INTERSPEECH 2010.
- Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. “On the difficulty of training Recurrent Neural Networks”, In Proc. of ICML 2013.
- Ciprian Chelba, Tomas Mikolov, Mike Schuster, Qi Ge, Thorsten Brants, Phillipp Koehn, and Tony Robinson. “One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling”, arXiv:1312.3005, 2013.
- Joshua Goodman. “Classes for Fast Maximum Entropy Training”. In Proc. of ICASSP 2001.
- Tomas Mikolov, Stefan Kombrink, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur. “Extensions of Recurrent Neural Network Language Model”. In Proc. of ICASSP 2011.
- Rafal Jozefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. “Exploring the Limits of Language Modeling”. arXiv: 1602.02410. 2016.
- Shihao Ji, S. V. N. Vishwanathan, Nadathur Satish, Michael J. Anderson, and Pradeep Dubey. “BlackOut: Speeding up Recurrent Neural Network Language Models With Very Large Vocabularies”. In Proc. of ICLR 2016.
- Junyoung Chung, Kyunghyun Cho, and Yoshua Bengio. “A Character-Level Decoder without Explicit Segmentation for Neural Machine Translation”, 2016.