

# Optimization (最適化) for Training Deep Models

Chapter 8  
楊 森 (Sen Yang) 2019/3/16

# 純粹最適化と違う所①Empirical (経験的な)

## Risk Estimation、Surrogate (代理、代用)

### Loss Functions and Early Stopping

本当目標  
(間接的な最適化) :

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

経験的な目標:

$$\begin{aligned} J(\theta) &= \mathbb{E}_{(x,y) \sim \hat{p}_{data}} L(f(x; \theta), y) \\ &= \frac{1}{m} \sum_{i=1}^m L(f(x^{(i)}; \theta), y^{(i)}) \end{aligned}$$

Deep Learningではあまり使わない。  
理由①: prone to overfitting (overfitting がち)  
理由②: many useful loss functions, such as 0-1 loss, have no useful derivatives.

0-1 lossなどの関数がgradient-based方法で最適化できない。

対策: 例えば、0-1 lossのsurrogate loss functionはnegative log-likelihood (log尤度) of the correct class.

メリット: learn more. 0-1 lossより、log-likelihoodの方が classを遠く押し分けられる。

$$L(i, j) = \begin{cases} 0 & i = j \\ 1 & i \neq j \end{cases} \quad i, j \in M$$

# 補足①: negative log-likelihoodの由来: Maximum Likelihood Estimation (最尤推定)

最尤推定器:

$$\theta_{ML} = \underset{\theta}{\operatorname{argmax}} p_{model}(\mathbb{X}; \theta)$$
$$= \underset{\theta}{\operatorname{argmax}} \prod_{i=1}^m p_{model}(x^{(i)}; \theta)$$

logarithmに変換:

$$= \underset{\theta}{\operatorname{argmax}} \sum_{i=1}^m \log p_{model}(x^{(i)}; \theta)$$

Maximization:

$$= \underset{\theta}{\operatorname{argmax}} \mathbb{E}_{x \sim \hat{p}_{data}} \log p_{model}(x; \theta)$$

KL divergence:  $D_{KL}(\hat{p}_{data} || p_{model}) = \mathbb{E}_{x \sim \hat{p}_{data}} [\log \hat{p}_{data}(x) - \log p_{model}(x)]$

モデルと関係ない

Minimization:

$$- \mathbb{E}_{x \sim \hat{p}_{data}} \log p_{model}(x)$$

# 純粹最適化と違う所②Batch and Minibatch Algorithms

求めたいgradient  
(log-likelihoodの場合) :  $\nabla_{\theta} J(\theta) = \mathbb{E}_{x \sim \hat{p}_{data}} \nabla_{\theta} \log p_{model}(x, y; \theta)$

$$SE(\hat{\mu}_m) = \sqrt{Var\left[\frac{1}{m} \sum_{i=1}^m x^{(i)}\right]} = \frac{\sigma}{\sqrt{m}}$$

minibatchでgradientを予測する理由  
①: less than linear returns to using more samples to estimate the gradient. 10000サンプルでは標準偏差が1/100まで下がるが、100サンプルでは標準偏差が1/10まで下がれる。  
②: redundancy (余分) in the training set.

minibatch方法について説明:

- ① 1回目トレーニングする時（重複サンプルなし）は、本当の汎化エラーのunbiased予測を計算する。2回目以降は、（重複サンプル）unbiasedではなくなる。
- ②しかし、超大きいデータでトレーニングする時、サンプルが一回だけ使われることが多くなるし、overfittingより、underfittingとcomputational efficiencyが目立った課題になる。

# 最適化挑戦①： III-Conditioning (悪条件)

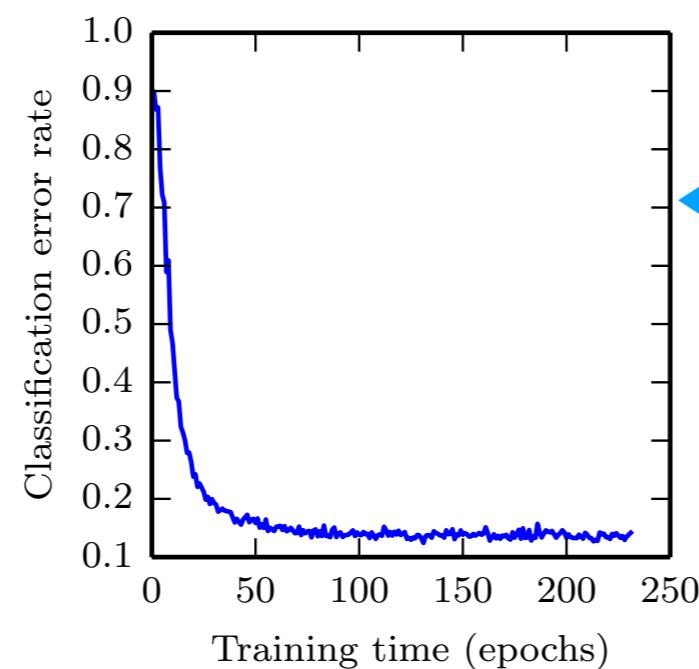
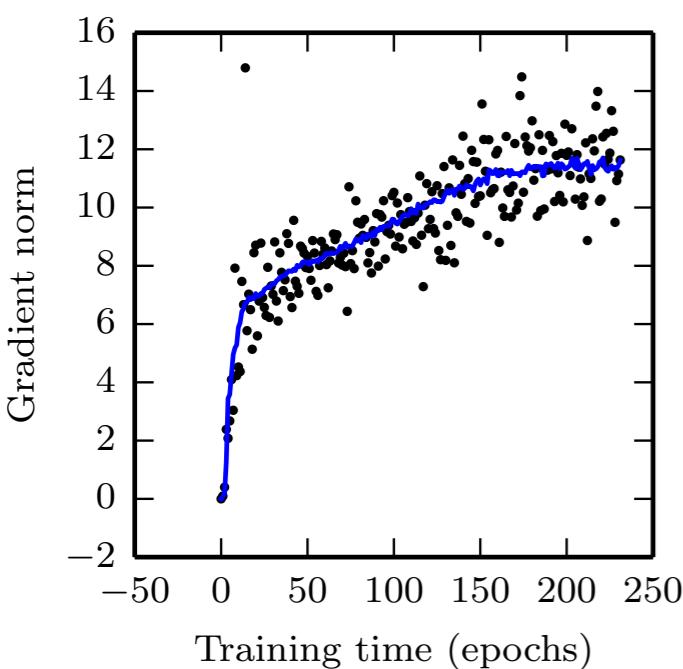
**Taylor expansion** (テイラー展開) :  $f(x) = f(x^{(0)}) + (x - x^{(0)})^T g + \frac{1}{2}(x - x^{(0)})^T H(x - x^{(0)})$

**gradient descent:**  $f(x^{(0)} - \epsilon g) \approx f(x^{(0)}) - \epsilon g^T g + \frac{1}{2}\epsilon^2 g^T H g$

**ill-conditioning** 発生:

$$\frac{1}{2}\epsilon^2 g^T H g > \epsilon g^T g$$

gradient descentの方向に進むと思うが、cost functionが上がっちゃう。結局、 $\epsilon$ がもっと小ちゃくなるしかない（学習がめっちゃ遅くなる）。

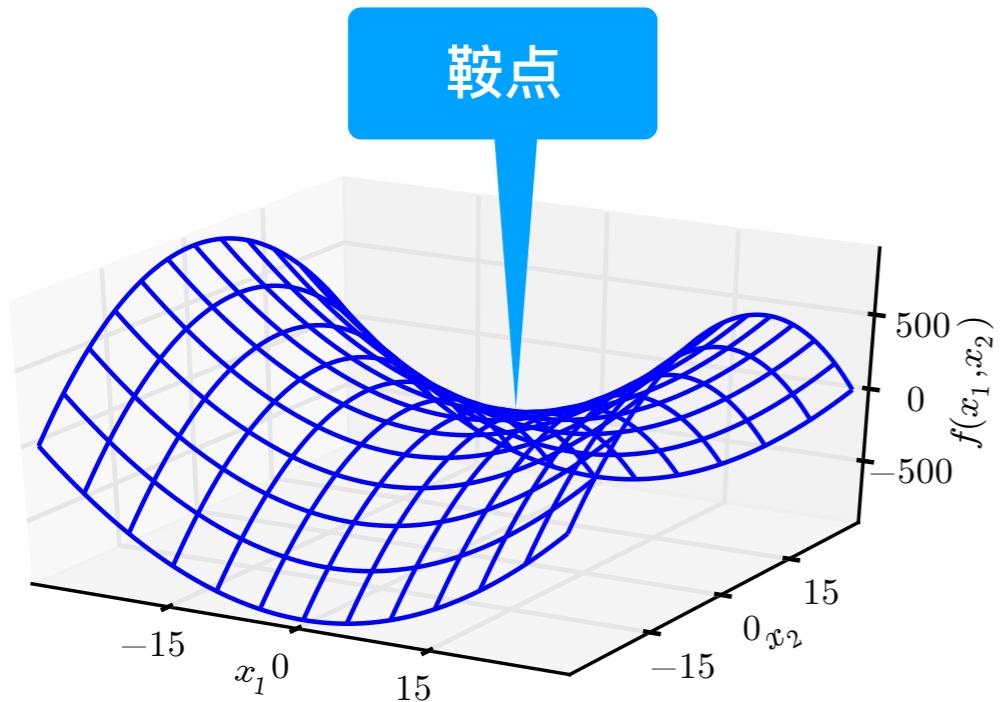


トレーニングプロセスが成功だが、gradient normが上がっている（僕ら想像した勾配の収束が発生していない）。つまり、学習が急崖にめっちゃ遅く動いている（僕の考え方）。

# 最適化挑戦②： Local Minima（局所最小値）

- Local Minimaが沢山存在する原因： model nonidentifiability（不可確認性、不可同定性）。つまり、モデルは唯一のparametersを決めるのができない。唯一に決められないパラメータは、latent variableという。
  - nonidentifiability一種目： weight space symmetry（重みスペース対称）： 1層目でunit iの入力重みとunit jの入力重みを交換しても、出力が変わらない。
  - nonidentifiability二種目： 重み減衰など（パラメータコントロールする）がないReLUもしくはmaxoutネットワークでは、入力の重みやバイアスを $a$ 倍、出力を $1/a$ 倍にscaleしても構わない。
- しかし、この種類のlocal minimaは問題にならない。costが同じだから。問題になるのが、costが高いlocal minima。
  - 十分大きいneural networkでは、local minimaが大体低いcostだと専門家が薄々気づいている。experts now suspect that, for sufficiently large neural networks, most local minima have a low cost function value, and that it is not important to find a true global minimum rather than to find a point in parameter space that has low but not minimal cost.

# 最適化挑戦③: Plateaus (平坦域), Saddle Points (鞍点) and Other Flat Regions

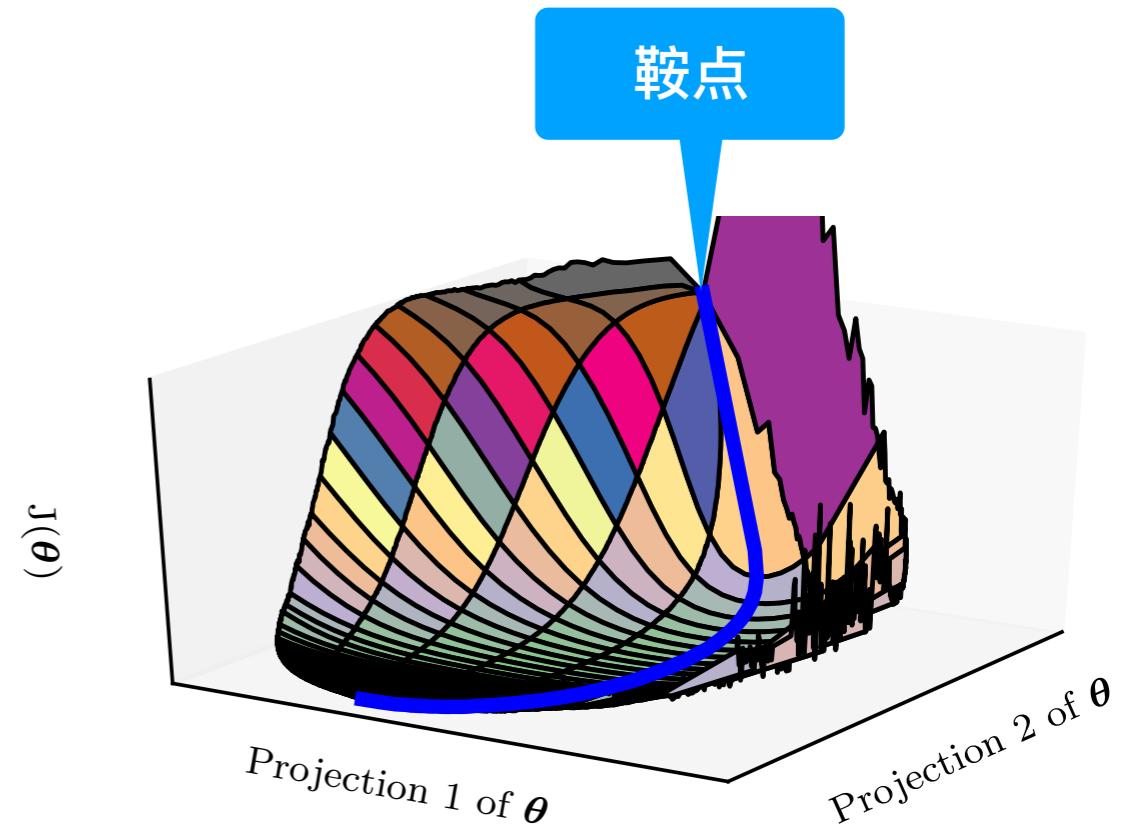


鞍点

鞍点に関する説明:

- ①鞍点で、Hessian matrix has both positive and negative eigenvalues (固有値)
- ②high-dimensional non convex関数では、鞍点の数と比べると、local minimaが非常に少ない。  
(直感に言うと、local minimaでのHの固有値が全部positiveだから)

SGD方法では、鞍点から容易に逃げられる。鞍点はあまり問題にならない。代わりに、平坦地域を通るのが一番時間がかかる。costがまだ高い平坦地域は問題になる。



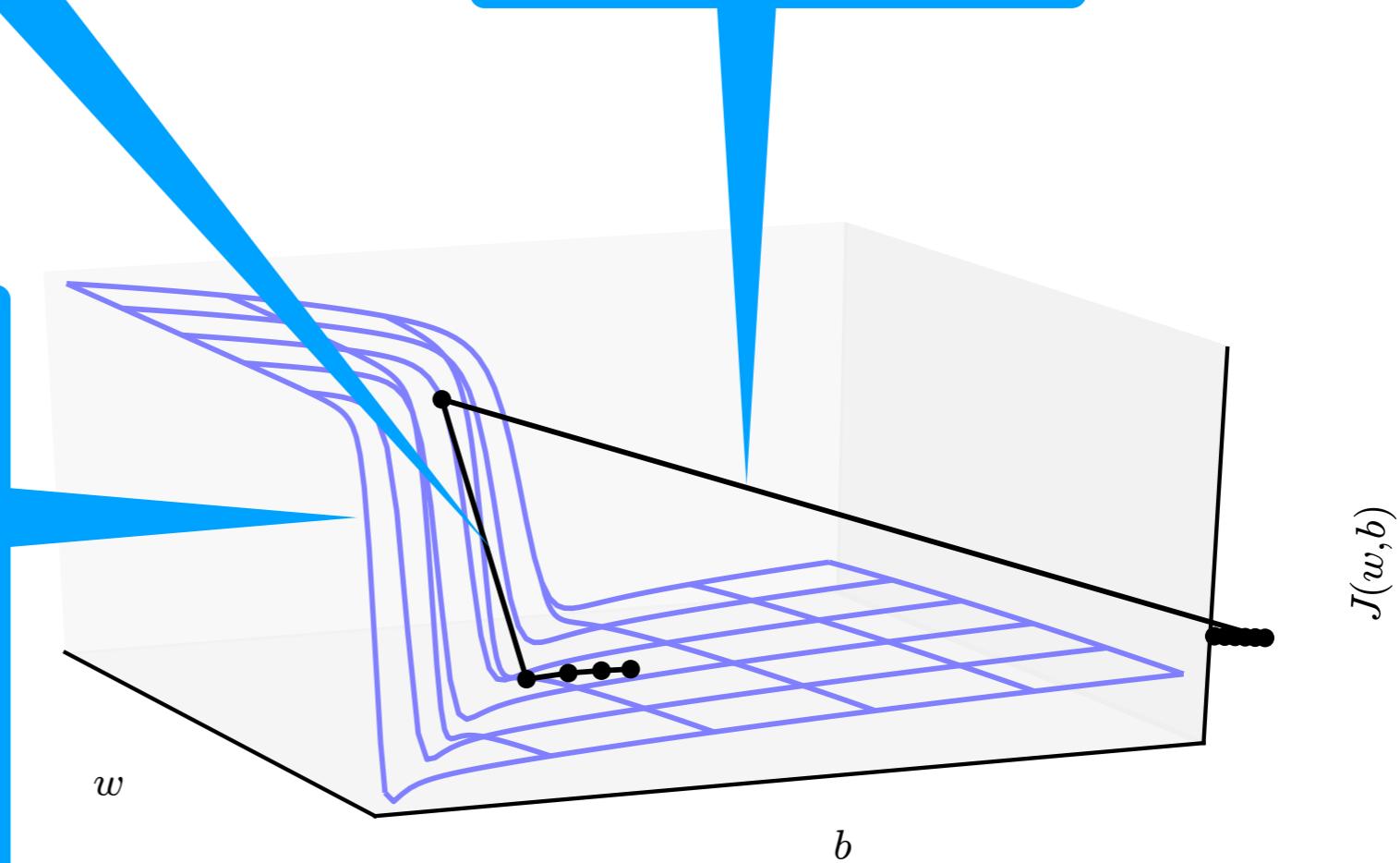
鞍点

# 最適化挑戦④: Cliffs (崖) and Exploding Gradients (爆発する勾配)

gradient clipping (刈り込み) : 大きいstepを検知したら、stepを制限する。

急勾配でtraditional gradient descentアルゴは大きいstepをする。

崖: recurrent neural networkによくある。沢山大きい重みの掛け算があるからです。崖での勾配が大きいので、stepを制限しないと、次のstepは飛んちゃって、既に学習したパラメータが大きく変更されて、この前の学習は全部無駄になる。



# 最適化挑戦⑤: Long-Term Dependencies (長期依頼)

RNNによくある状況:  
(construct very deep computational graphs by repeatedly applying the same operation at each time step of a long temporal sequence)

$$W^t = (V \text{diag}(\lambda) V^{-1})^t = V \text{diag}(\lambda)^t V^{-1}$$

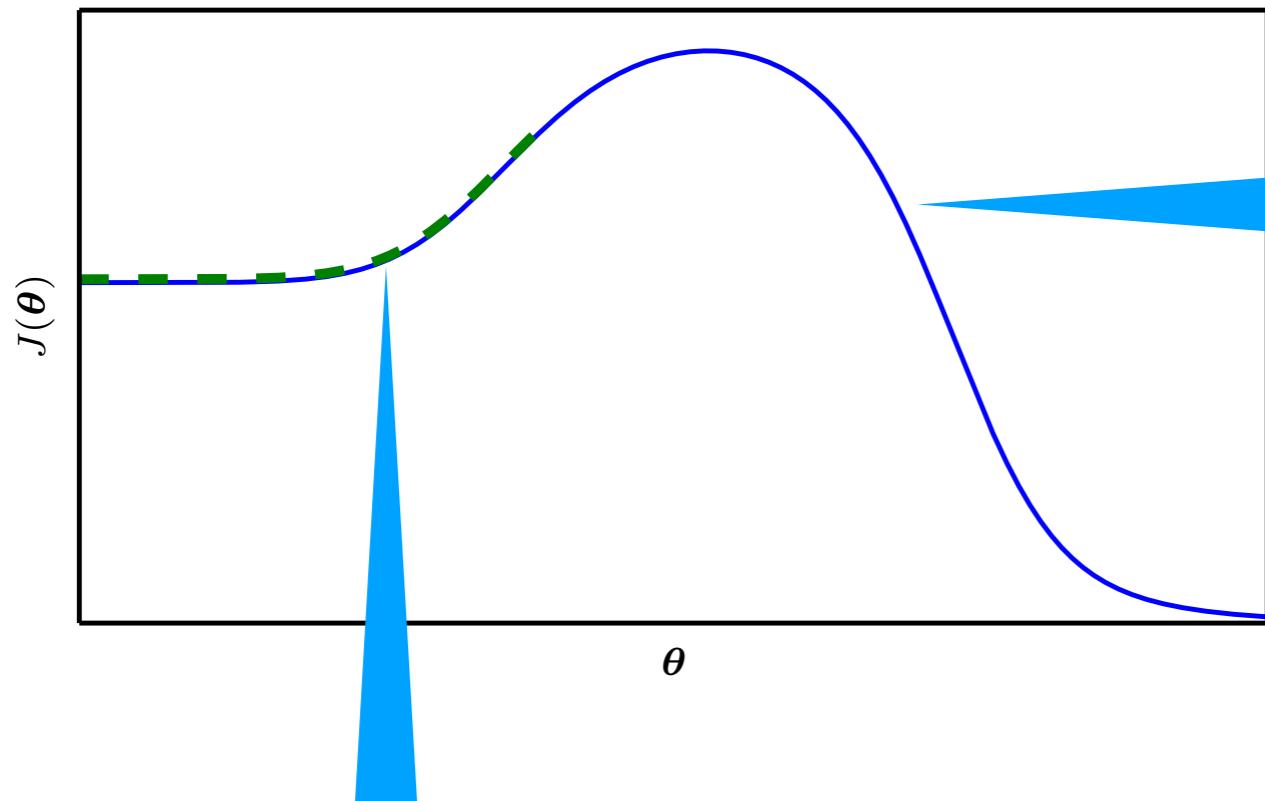
$\lambda < 1 \rightarrow$  vanishing gradient problem (消失)  
 $\lambda > 1 \rightarrow$  exploding gradient problem (爆発)

$x^T W^t$  ← xの中に、Wの主要な固有ベクトルと直交する成分は全部捨てられる。

Long-term dependenciesは Recurrent networkによくある課題。 feedforward networkでは大丈夫。

最適化挑戦⑥: Inexact Gradients  
(厳密でない勾配) (略)

# 最適化挑戦⑦: Poor Correspondence (悪い対応) between Local and Global Structure



局所最小値や鞍点がなくても、gradient descent最適化が失敗することも可能。

この課題についての対策は、新しいアルゴ(ローカルじゃない動きを目指す)を開発するではなくて、良い開始点を見つけること(パラメータ初期化)です。

因みに、最小値が存在しないこともある。例えば、漸近的にアプローチする(asymptotically approach)こと。

最適化挑戦⑧: Theoretical Limits of Optimization (最適化の理論的な限界) (略)

# 基本アルゴ①Stochastic Gradient Descent (確率的勾配降下法)

gradient予測を求める:

$$\hat{g} \leftarrow -\frac{1}{m} \nabla_{\theta} \sum_i L(f(x^{(i)}; \theta), y^{(i)})$$

更新する:

$$\theta \leftarrow \theta - \epsilon \hat{g}$$

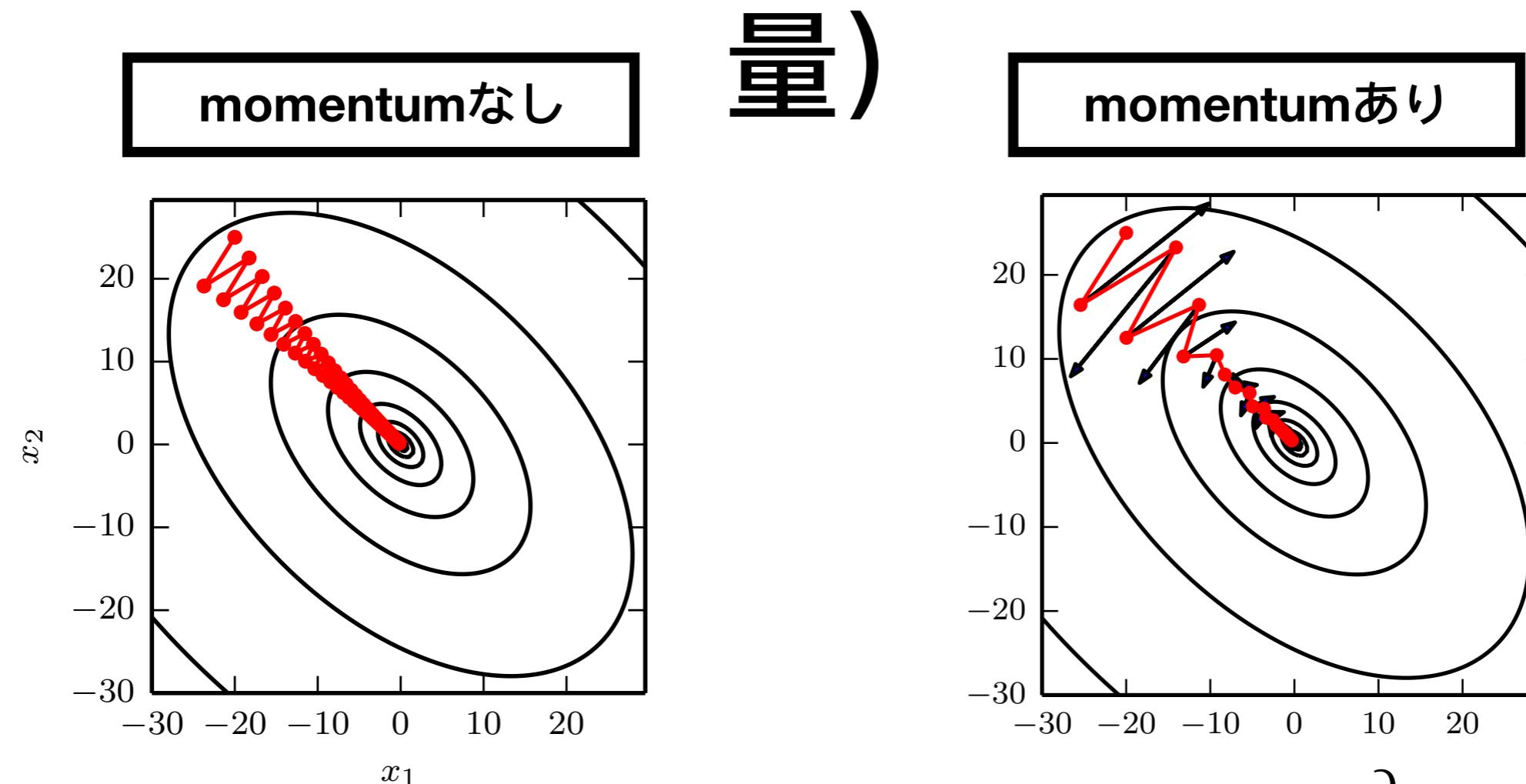
学習率を減衰する  
のがよくある:

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_{\tau}; \alpha = \frac{k}{\tau}$$

- ① $\tau$ は、100回くらいtraining setを使い切るiterationの数。 $\tau$ 回以降は学習率が固定。
- ② $\epsilon_{\tau}$ は、 $\epsilon_0$ の1%。
- ③ $\epsilon_0$ は、最初の何回iterationを見て、最優 $\epsilon$ より若干高い値にする。

- ①SGDや他のminibatch or online gradient-based最適化方法の一番重要なプロパティは、訓練サンプルが増えても、step毎の計算時間は上がらない。
- ②だから、十分大きいデータセットで学習すると、全部のデータを使い切ってなくても、学習終了は可能。

# 基本アルゴ②Momentum (運動量)



速度更新:

$$v \leftarrow \alpha v - \epsilon g$$



$$f(t) = \frac{\partial}{\partial t} v(t)$$

基本アルゴ  
③Nesterov  
Momentum  
(略)

$\alpha$ : viscous drag (粘性抵抗)。  
勾配がcost functionの唯一の力  
だと、 $\theta$ がずっと止まらないかも

パラメータ更新:

$$\theta \leftarrow \theta + v$$



$$v(t) = \frac{\partial}{\partial t} \theta(t)$$

他にturbulent drag (乱流抵抗、 $v^2$ と比例する) やdry friction (乾燥摩擦、定数) モデルもあるが、力が弱すぎるともしくは強すぎるので、使われてない。

# Parameter Initialization Strategies

## (パラメータ初期化戦略)

- 唯一確かな標準: 初期パラメータは、違うユニットの間に対称性を壊すべき。 Perhaps the only property known with complete certainty is that the initial parameters need to “break symmetry” between different units. p293.
- 大分の初期化戦略の目標は: 初期化後のネットワークがいくつかの良いプロパティを持つ。しかし
  - 学習開始後、どのプロパティが保持されるか分からない。
  - 最適化の観点でメリットだが、一般化の観点ではデメリットになるかも。例えば、最適化は情報をうまく伝えるため、大きい重みが欲しいが、正則化 (weight decayとか) は小さい重みが欲しい。
- 通常では、正規分布もしくは一様分布からランダム的に重みを初期化する。選択の理由はまだ徹底的に研究されていない。分布のscaleは最適化と一般化に影響大きい。
- scaleはいくつかの競争因子から決める。 large weights may yield a stronger symmetry-breaking effect, avoid losing signal, result in exploding values, cause the activation function to saturate. p294.

# 適応 (adaptive) 学習率アルゴ

## ①AdaGrad

### Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate  $\epsilon$

Require: Initial parameter  $\theta$

Require: Small constant  $\delta$ , perhaps  $10^{-7}$ , for numerical stability

Initialize gradient accumulation variable  $r =$

while stopping criterion not met do

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$ .

    Accumulate squared gradient:  $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g} \odot \mathbf{g}$ .

    Compute update:  $\Delta \theta \leftarrow -\frac{\epsilon}{\delta + \sqrt{\mathbf{r}}} \odot \mathbf{g}$ . (Division and square root applied element-wise)

    Apply update:  $\theta \leftarrow \theta + \Delta \theta$ .

end while

AdaGradの課題: 学習開始から  
gradient平方を累積すると、有効学習  
率の早すぎる低減や過大な低減になる

rにgradient平方を蓄積する

割り算や平方根は要素毎です。つまり、偏微分が大きいパラメータの学習率は遅く降下する。この方向は急、遅く進もう。偏微分が小さいパラメータの学習率は早く降下する。この方向は緩やか、早く進もう。

# 適応 (adaptive) 学習率アルゴ

## ② RMSProp

---

### Algorithm 8.5 The RMSProp algorithm

---

Require: Global learning rate  $\epsilon$ , decay rate  $\rho$

Require: Initial parameter  $\theta$

Require: Small constant  $\delta$ , usually  $10^{-6}$ , used to stabilize division by small numbers

Initialize accumulation variables  $r = 0$

while stopping criterion not met do

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$ .

    Accumulate squared gradient:  $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g} \odot \mathbf{g}$ .

    Compute parameter update:  $\Delta \theta = -\frac{\epsilon}{\sqrt{\delta + \mathbf{r}}} \odot \mathbf{g}$ . ( $\frac{1}{\sqrt{\delta + \mathbf{r}}}$  applied element-wise)

    Apply update:  $\theta \leftarrow \theta + \Delta \theta$ .

end while

---

Momentumの $\alpha$ と同じように、 $\rho$ で過去の累積を減衰する。

経験的に、RMSPropは有効で現実的な最適化方法です。  
deep learning実践者に日常使われている最適化方法です。

# 適応 (adaptive) 学習率アルゴ

## ③Adam

Algorithm 8.7 The Adam algorithm

Require: Step size  $\epsilon$  (Suggested default: 0.001)

Require: Exponential decay rates for moment estimates,  $\rho_1$  and  $\rho_2$  in  $[0, 1]$ .  
(Suggested defaults: 0.9 and 0.999 respectively)

Require: Small constant  $\delta$  used for numerical stabilization (Suggested default:  
 $10^{-8}$ )

Require: Initial parameters  $\theta$

Initialize 1st and 2nd moment variables  $s = \mathbf{0}$ ,  $r = \mathbf{0}$

Initialize time step  $t = 0$

while stopping criterion not met do

    Sample a minibatch of  $m$  examples from the training set  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  with  
    corresponding targets  $\mathbf{y}^{(i)}$ .

    Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

$t \leftarrow t + 1$

    Update biased first moment estimate:  $s \leftarrow \rho_1 s + (1 - \rho_1)\mathbf{g}$

    Update biased second moment estimate:  $r \leftarrow \rho_2 r + (1 - \rho_2)\mathbf{g} \odot \mathbf{g}$

    Correct bias in first moment:  $\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$

    Correct bias in second moment:  $\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$

    Compute update:  $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}} + \delta}$  (operations applied element-wise)

    Apply update:  $\theta \leftarrow \theta + \Delta\theta$

end while

Adamは、  
hyperparameterの選  
択にrobust

RMSPropの上、  
Momentumを入れる

$$\hat{s} \leftarrow \frac{s}{1 - \rho_1^t}$$
$$\hat{r} \leftarrow \frac{r}{1 - \rho_2^t}$$

bias修正

# 二次近似方法 (Approximate Second-Order Methods)

## ① ニュートン方法 (Newton's Method)

$$J(\theta) \approx J(\theta_0) + (\theta - \theta_0)^T \nabla_{\theta} J(\theta_0) + \frac{1}{2} (\theta - \theta_0)^T H(\theta - \theta_0)$$

臨界点を求める:

$$\theta^* = \theta_0 - H^{-1} \nabla_{\theta} J(\theta_0)$$

二次関数だと、one-step  
で臨界点に飛べる。

ヘッセ行列が正定  
じゃないと、正定にする:

$$\theta^* = \theta_0 - [H(J(\theta_0)) + \alpha I]^{-1} \nabla_{\theta} J(\theta_0)$$

①  $H$ の負固有値が0に近ければ、OK。遠ければ、 $\alpha$ が大きくなるしかない。結局  $\alpha I$  が主要になって、 $H$ を使わないと同じ。

②  $H^{-1}$ の計算が大変。  
 $O(k^3)$  計算複雑さ。結局 パラメータ数が少ないネットワークしかニュートン方法を使えない。

二次関数以上の  
convex関数、  
ニュートン方法を  
反復する

---

Algorithm 8.8 Newton's method with objective  $J(\theta) = \frac{1}{m} \sum_{i=1}^m L(f(x^{(i)}; \theta), y^{(i)})$

Require: Initial parameter  $\theta_0$

Require: Training set of  $m$  examples

while stopping criterion not met do

Compute gradient:  $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(x^{(i)}; \theta), y^{(i)})$

Compute Hessian:  $\mathbf{H} \leftarrow \frac{1}{m} \nabla_{\theta}^2 \sum_i L(f(x^{(i)}; \theta), y^{(i)})$

Compute Hessian inverse:  $\mathbf{H}^{-1}$

Compute update:  $\Delta \theta = -\mathbf{H}^{-1} \mathbf{g}$

Apply update:  $\theta = \theta + \Delta \theta$

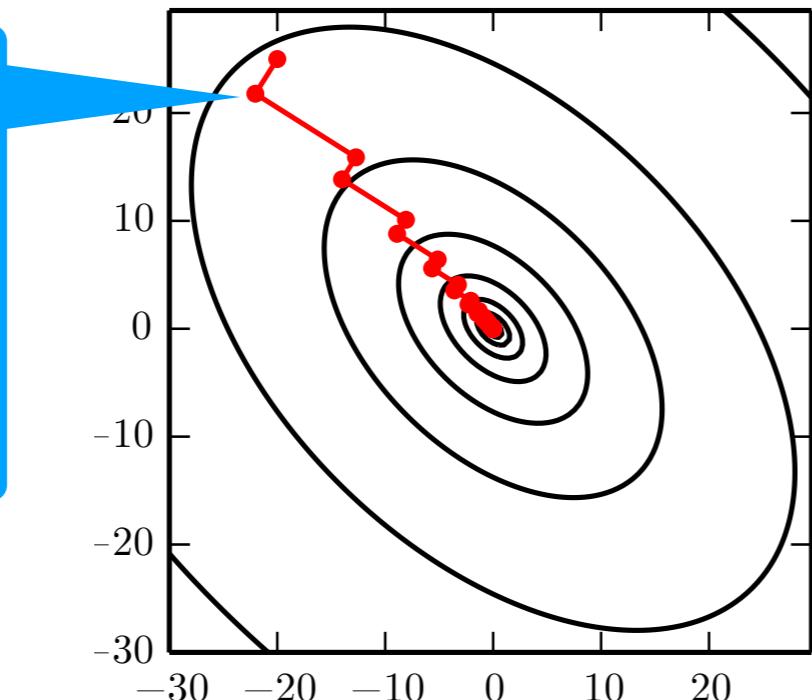
end while

# 二次近似方法 (Approximate Second-Order Methods)

## ② 共役勾配法 (Conjugate Gradients)

### 最急降下法

直線探索で最急方向の最低点に行く。  
前探索方向と現探索方向は直交する。  
効率が低い。



直線探索方向。現在の gradientによって決まる

$$\nabla_{\theta} J(\theta) \cdot d_{t-1} = 0$$
$$d_t = \nabla_{\theta} J(\theta)$$

低効率原因：次のステップで直交方向に進むので、前回の進捗が取り消された。

過去の進捗を保留するため、  
直交方向じゃなくて、次ス  
テップ共役方向で進む：

メリット： Hの逆を使わない

Hの逆を計算せず、Fletcher-ReevesやPolak-Ribiere方法  
でβを計算できる。

$$\text{共役の定義: } d_t^T H d_{t-1} = 0$$

二次近似方法  
③BFGS (略)

# 最適化策とメタアルゴ

(Optimization Strategies and  
Meta-Algorithms)

## ① バッチ正規化 (Batch Normalization)

課題の例 (悪条件の課題と似てると思う) :  $\hat{y} = x\omega_1\omega_2\omega_3 \dots \omega_l$

ステップ更新:  $x(\omega_1 - \epsilon g_1)(\omega_2 - \epsilon g_2) \dots (\omega_l - \epsilon g_l)$  項の一例:  $\epsilon^2 g_1 g_2 \prod_{i=3}^l \omega_i$

対策 (全て層の活性化  $h$  を正規化する) :

$$H' = \frac{H - \mu}{\sigma} \quad (H'_{i,j} = \frac{H_{i,j} - \mu_j}{\sigma_j})$$

$H'$  がいつも zero mean や unit variance (unit Gaussian) を保つ

$w_i < 1$  だと無視していい。  
 $w_i > 1$  だとでかいすぎる。 $\epsilon$  は選びにくい。

$$\mu = \frac{1}{m} \sum_i H_{i,:}$$

$$\sigma = \sqrt{\delta + \frac{1}{m} \sum_i (H - \mu)_i^2}$$

- ① 正規化でリニア低層が無駄になるが、非線形低層は大丈夫。
- ② 正規化でその層の表現力が下がる。だから、 $H'$  の代わりに  $\gamma H' + \beta$  を使うのがよくある。 $\gamma H' + \beta$  は、学習しやすい。

# 最適化策とメタアルゴ

(Optimization Strategies and Meta-Algorithms)

## ②座標降下 (Coordinate Descent) ③Polyak平均

考え方:  $f(x)$ を最小化する時、まず $x_1$ に対して最小化して、また $x_2$ に対して最小化して…

sparse codingの例:

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

Jはconvexじゃない。HもしくはWに対する最小化はconvexです。Hを固定してWを最適化して、Wを固定してHを最適化する。

しかし、もし1つ変数の値が別の変数の最適値に強い影響があれば、座標降下はいい方法じゃない。例えば：

$$f(x) = (x_1 - x_2)^2 + \alpha(x_1^2 + x_2^2)$$

左半分は $x_1=x_2$ にしたい。右半分は $x_1=x_2=0$ にしたい。 $x_2$ を固定すると $x_1$ は $x_2$ から離れないで、0には行きにくい。でも、ニュートン方法だと、one-stepで行ける。

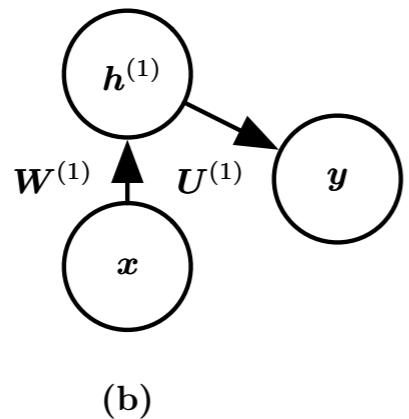
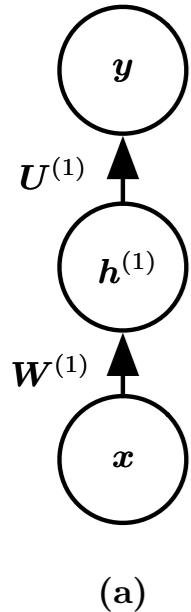
$$\hat{\theta}^{(t)} = \alpha \hat{\theta}^{(t-1)} + (1 - \alpha) \theta^{(t)}$$

Polyak平均の考え方：谷の崖に行ったり戻ったりするが、谷の底に行かないので、平均化して谷の底に導く。

# 最適化策とメタアルゴ

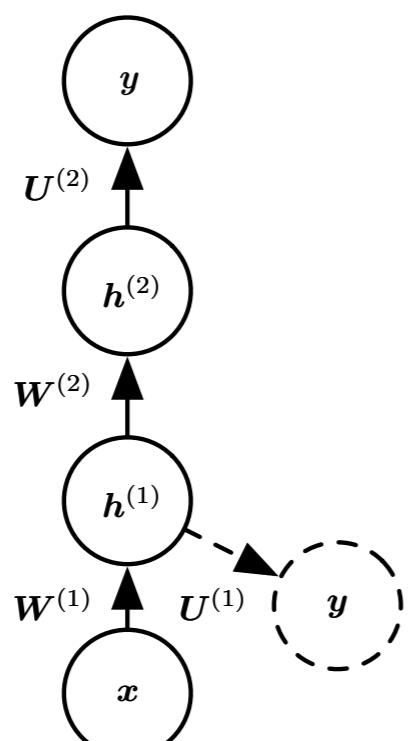
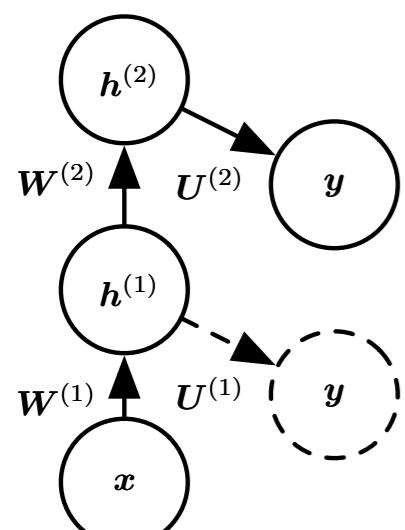
(Optimization Strategies and  
Meta-Algorithms)

## ④ Supervised Pretraining



考え方：簡単モデルを訓練して、訓練した簡単モデルの一部パラメータを使ってさらに複雑モデルを訓練する。

なぜgreedy（貪欲どんよく）supervised pretrainingが効果ある？深い層の良い指導になるから。通常、pretrainingは最適化にも一般化にも良い。転移学習も近い考え方。



最適化策とメタアルゴ⑤最適化しやすいモデルを設計する  
(Designing Models to Aid Optimization) : 強い最適化アルゴを使うより、最適化しやすいモデルを選ぶのは大事。  
①use more linear functions (LSTM, ReLU, maxoutなど)  
②skip connection

# 最適化策とメタアルゴ

## (Optimization Strategies and Meta-Algorithms) ⑥ Continuation (継続) Methods and Curriculum Learning

**Continuation Methods**の考え方：簡単cost関数を最適化して、最適化したcost関数の最優点を複雑cost関数の開始点として使って複雑cost関数を最適化する。

cost関数から簡単なcost関数を作る方法：ぼかし（暈し、blurring）

説明①：Continuation Methodsは失敗する場合：costが高い（baggingと似てる原因だと思う）、ぼかしてもconvexになれない、ぼかしたcost関数のminimaはlocal minimaになるかも。

説明②：Continuation Methodsはlocal minimaを解決するため提出された方法。しかし、Neural Network最適化では、local minimaはもう問題ではなくなってくる。

**Curriculum Learning**の考え方：簡単なサンプルから訓練して、複雑なサンプルをどんどん増やして訓練する。