

## VIP チートシート: アドバイスやコツ



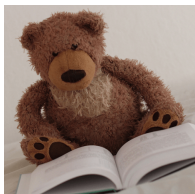

アフシンアミディ・シエルビンアミディ 著



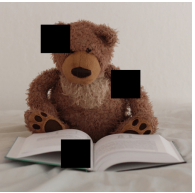

October 7, 2019

カムエララウ・中井喜之・森浩貴 訳

## データ処理

□ **Data augmentation (データ拡張)** – 大抵の場合は、深層学習のモデルを適切に訓練するには大量のデータが必要です。Data augmentation という技術を用いて既存のデータから、データを増やすことがよく役立ちます。以下、Data augmentation の主な手法はまとめています。より正確には、以下の入力画像に対して、下記の技術を適用できます。

元の画像	反転	回転	ランダムな切り抜き
			
- 何も変更されていない画像	- 画像の意味が変わらない軸における反転	- わずかな角度の回転 - 不正確な水平線の校正 (calibration) をシミュレートする	- 画像の一部へのランダムなフォーカス - 連続して数回のランダムな切り抜きが可能

カラーシフト	ノイズの付加	情報損失	コントラストの修正
			
- RGBのわずかな修正 - 照らされ方によるノイズを捉える	- ノイズの付加 - 入力画像の品質のばらつきへの耐性の強化	- 画像の一部を無視 - 画像の一部が欠ける可能性を再現する	- 明るさの変化 - 時刻による露出の違いをコントロールする

□ **Batch normalization** – ハイパーパラメータ  $\gamma, \beta$  によってバッチ  $\{x_i\}$  を正規化するステップです。修正を加えたいバッチの平均と分散を  $\mu_B, \sigma_B^2$  と表記すると、以下に行えます。

$$x_i \leftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

より高い学習率を利用可能にし初期化への強い依存を減らすことを目的として、基本的には全結合層・畳み込み層のあとで非線形層の前に行います。

## ニューラルネットワークの学習

□ **エポック** – モデル学習においてエポックとは学習の繰り返しの中の1回を指す用語で、1エポックの間にモデルは全学習データからその重みを更新します。

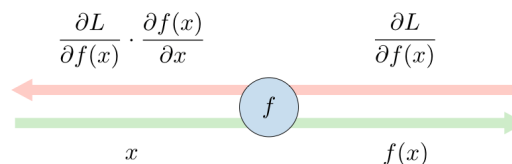
□ **ミニバッチ勾配降下法** – 学習段階では、計算が複雑になりすぎるため通常は全データを一度に使う重みを更新することはありません。またノイズが問題になるため1つのデータポイントだけを使って重みを更新することはありません。代わりに、更新はミニバッチごとに行われます。各バッチに含まれるデータポイントの数は調整可能なハイパーパラメータです。

□ **損失関数** – 得られたモデルの性能を数値化するために、モデルの出力  $z$  が実際の出力  $y$  をどの程度正確に予測できているかを評価する損失関数  $L$  が通常使われます。

□ **交差エントロピー誤差** – ニューラルネットワークにおける二項分類では、交差エントロピー誤差  $L(z, y)$  が一般的に使用されており、以下のように定義されています。

$$L(z, y) = - \left[ y \log(z) + (1 - y) \log(1 - z) \right]$$

□ **誤差逆伝播法** – 実際の出力と期待される出力の差に基づいてニューラルネットワークの重みを更新する手法です。各重み  $w$  に関する微分は連鎖律を用いて計算されます。

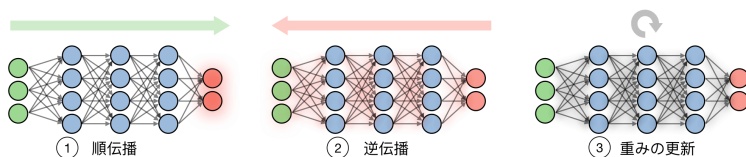


この方法を使用することで、それぞれの重みはそのルールにしたがって更新されます。

$$w \leftarrow w - \alpha \frac{\partial L(z, y)}{\partial w}$$

□ **重みの更新** – ニューラルネットワークでは、以下の方法にしたがって重みが更新されます。

- ステップ1: 訓練データのバッチを用いて順伝播で損失を計算します。
- ステップ2: 損失を逆伝播させて各重みに関する損失の勾配を求めます。
- ステップ3: 求めた勾配を用いてネットワークの重みを更新します。



## パラメータチューニング

□ **Xavier初期化** – 完全にランダムな方法で重みを初期化するのではなく、そのアーキテクチャのユニークな特徴を考慮に入れて重みを初期化する方法です。

□ **転移学習** – 深層学習のモデルを学習させるには大量のデータと何よりも時間が必要です。膨大なデータセットから数日・数週間をかけて構築した学習済みモデルを利用し、自身のユースケースに活かすことは有益であることが多いです。手元にあるデータ量次第ではありますが、これを利用する以下の方法があります。

学習サイズ	図	解説
小		全層を凍結し、softmaxの重みを学習させる
中		大半の層を凍結し、最終層とsoftmaxの重みを学習させる
大		学習済みの重みで初期化して各層とsoftmaxの重みを学習させる

□ **学習率** – 多くの場合 $\alpha$ や時々 $\eta$ と表記される学習率とは、重みの更新速度を表しています。学習率は固定することもできる上に、適応的に変更することもできます。現在もっとも使用される手法は、学習率を適切に調整するAdamと呼ばれる手法です。

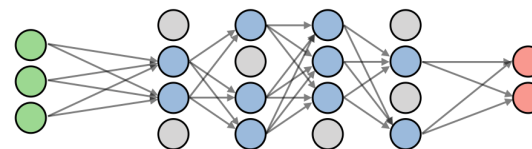
□ **適応学習率法** – モデルを学習させる際に学習率を変動させると、学習時間の短縮や精度の向上につながります。Adamがもっとも一般的に使用されている手法ですが、他の手法も役立つことがあります。それらの手法を下記の表にまとめました。

手法	解説	$w$ の更新	$b$ の更新
Momentum (運動量)	- 振動を抑制する - SGDの改良 - チューニングするパラメータは2つ	$w - \alpha v_{dw}$	$b - \alpha v_{db}$
RMSprop	- 二乗平均平方根のプロパゲーション - 振動をコントロールすることで学習アルゴリズムを高速化する	$w - \alpha \frac{dw}{\sqrt{s_{dw}}}$	$b \leftarrow b - \alpha \frac{db}{\sqrt{s_{db}}}$
Adam	- Adaptive Moment estimation - もっとも人気のある手法 - チューニングするパラメータは4つ	$w - \alpha \frac{v_{dw}}{\sqrt{s_{dw}} + \epsilon}$	$b \leftarrow b - \alpha \frac{v_{db}}{\sqrt{s_{db}} + \epsilon}$

備考：他に *Adadelata*, *Adagrad*, *SGD* などの手法があります。

## 正則化

□ **ドロップアウト** – ドロップアウトとは、ニューラルネットワークで過学習を避けるために $p > 0$ の確率でノードをドロップアウト（無効化）する手法です。モデルが特定の特徴量に依存しすぎることを避けるよう強制します。

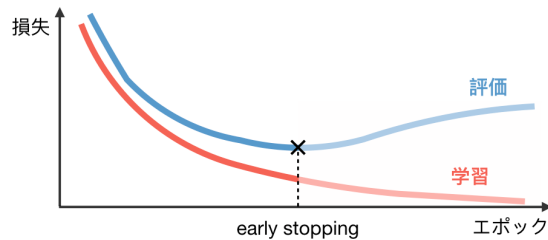


備考：ほとんどの深層学習のフレームワークでは、ドロップアウトを 'keep' というパラメータ ( $1 - p$ ) でパラメータ化します。

□ **重みの正則化** – 重みが大きくなりすぎず、モデルが過学習しないようにするため、モデルの重みに対して正則化を行います。主な正則化手法は以下の表にまとめられています。

LASSO	Ridge	Elastic Net
- 係数を0へ小さくする - 変数選択に適している	係数を小さくする	変数選択と小さい係数のトレードオフ
$\dots + \lambda \ \theta\ _1$ $\lambda \in \mathbb{R}$	$\dots + \lambda \ \theta\ _2^2$ $\lambda \in \mathbb{R}$	$\dots + \lambda \left[ (1 - \alpha) \ \theta\ _1 + \alpha \ \theta\ _2^2 \right]$ $\lambda \in \mathbb{R}, \alpha \in [0, 1]$

□ **Early stopping** – バリデーションの損失が変化しなくなるか、あるいは増加し始めたときに学習を早々に止める正則化方法



### おすすめの技法

□ **小さいバッチの過学習** – モデルをデバッグするとき、モデル自体の構造に大きな問題がないか確認するため簡易的なテストが役に立つことが多いです。特に、モデルを正しく学習できることを確認するため、ミニバッチをネットワークに渡してそれを過学習できるかを見ます。もしできなければ、モデルは複雑すぎるか単純すぎるかのいずれかであることを意味し、普通サイズの学習データセットはもちろん、小さいバッチですら過学習できないのです。

□ **Gradient checking (勾配チェック)** – Gradient checking とは、ニューラルネットワークの逆伝播を実装する際に用いられる手法です。特定の点で解析的勾配と数値的勾配とを比較する手法で、逆伝播の実装が正しいことを確認できます。

	数値的勾配	解析的勾配
公式	$\frac{df}{dx}(x) \approx \frac{f(x+h) - f(x-h)}{2h}$	$\frac{df}{dx}(x) = f'(x)$
コメント	<ul style="list-style-type: none"> <li>- 計算コストが高い、損失を次元ごとに2回計算する必要がある</li> <li>- 解析的実装が正しいかのチェックに用いられる</li> <li>- <math>h</math>を選ぶ時に小さすぎると数値不安定になり、大きすぎると勾配近似が不正確になるというトレードオフがある</li> </ul>	<ul style="list-style-type: none"> <li>- 「正しい」結果</li> <li>- 直接的な計算</li> <li>- 最終的な実装で使われる</li> </ul>

★ ★ ★