画像認識 _ 課題 4・発展課題

村山健太
2018
1029300689

2020 年 11 月 25 日

課題内容

MNIST のテスト画像 1 枚を入力とし，3 層ニューラルネットワークを用いて，0〜9 の値のうち 1 つを出力するプログラムを作成した．また発展課題として発展課題 A1〜A7 を実装し, cifar-10 を推論するニューラルネットワークを構築した.

作成したプログラムの説明

以下の 5 つのプログラムを作成した.

- layer.py･･･layer クラスを管理するプログラム
- optimizer.py･･･ 最適化手法のクラスを管理するプログラム
- network.py･･･ 学習, 推論, 学習データの読み込み, 保存をするクラスのプログラム
- cifar10.py･･･cifar10 の学習, 推論するプログラム
- main.py･･･MNIST の画像を学習, 推論するプログラム．ただし課題 4 のための簡単なニューラルネットワークである.

layer.py の説明

はじめに layer.py の概要を説明したのちに詳細について説明する. 実装した層は以下の 8 種類のクラスである.

- Sigmoid 層 ･･･ 活性化関数であるシグモイド関数に基づく層
- Affine 層 ･･･ 線形関数に基づく全結合層
- SoftmaxWithLoss 層 ･･･ 損失関数とソフトマックス関数に基づく層
- ReLU 層 ･･･ 活性化関数に基づく層
- Dropout 層 ･･･ 学習時にランダムに一部の重みだけを使って計算を行う層
- Batch_Normalization 層 ･･･ 出力を正規化する層
- Convolution 層 ･･･ 二次元データをそのまま計算する層
- MaxPooling 層 ･･･ 二次元データの特徴を掴んでサイズを小さくする層

それぞれのクラスに以下の 7 つの関数を実装することで統一的に学習, 推論ができるようにした.

- forward 関数 ･･･ 層の順伝搬の計算を行う関数
- backward 関数 ･･･ 層の逆伝搬の計算を行う関数
- set_optimizer 関数 ･･･ 層の学習に使用する最適化手法を設定する関数
- update 関数 ･･･ 層の重みを更新する関数
- getW 関数 ･･･ 層の重みを返す関数
- numW 関数 ･･･ 層の重みの種類の数を返す関数
- loadW 関数 ･･･ 層の重みを読み込む関数

この関数以外にそれぞれの層に必要な関数も定義した.

Sigmoid 層の説明

コンストラクタ

必要なインスタンス変数を定義しておく.

foward 関数

戻り値 out(ndarray) は出力データである. また引数は, 層の入力データである x(ndarray), 学習時か推論時かを区別する flag(bool) をとる. ただし学習と推論で計算方法は同じであるから flag は用いない. 出力結果 out は逆伝搬の計算に用いるのでインスタンス変数として保存する.

backward 関数

戻り値 dx(ndarray) は 1 つ前の層に渡す逆伝搬である. また引数は 1 つ次の層の逆伝搬である dout(ndarray) をとる. 計算はテキストにしたがって行う.

set_optimizer 関数

Sigmoid 層では学習する重みが存在しないため, 何も行わない.

update 関数

Sigmoid 層では学習する重みが存在しないため, 何も行わない.

getW 関数

Sigmoid 層では学習する重みが存在しないため, 空リストを返す.

numW 関数

Sigmoid 層では学習する重みが存在しないため, 0 を返す.

loadW 関数

Sigmoid 層では学習する重みが存在しないため, 何も行わない.

Affine 層の説明

コンストラクタ

引数は, 層の入力データのノード数である input_node_num(int), 層の出力データのノード数である output_node_num(int) をとる. input_node_num, output_node_num に応じて重みとバイアスの初期値を __initial_weight 関数によって定める. また必要なインスタンス変数を定義しておく.

foward 関数

戻り値 out(ndarray) は出力データである. また引数は, 層の入力データである x(ndarray), 学習時か推論時かを区別する flag(bool) をとる. ただし学習と推論で計算方法は同じであるから flag は用いない. 入力データの行数を batch 数にして行列として計算を行う. そのため逆伝搬の計算のときに元のサイズに戻すために元の入力 shape を original_x_shape として保存する. また入力データ x も逆伝搬の計算に使用するのでインスタ

ンス変数として保存する.

backward 関数

戻り値 dx(ndarray) は 1 つ前の層に渡す逆伝搬である． また引数は 1 つ次の層の逆伝搬である dout(ndarray) をとる．計算はテキストにしたがって行う．また dx は計算のちに shape を original_x_shape に戻す．損失関数の重みとバイアスの偏微分をそれぞれ dW, db として保存する.

set_optimizer 関数

引数は最適化手法を管理するクラスのインスタンス optimizer をとる.

update 関数

更新すべき重み W, b を optimizer の update メソッドを適用して更新する.

getW 関数

学習すべき重み W, b を空 list に append して返す.

numW 関数

学習すべき重み W, b の 2 種類存在するので 2 を返す.

loadW 関数

引数は層に設定したい重みを並べたものである list_W(list) をとる．list_W から重みを取り出して W, b に代入する.

__initial_weight 関数

重みとバイアスの初期値をランダムに決める関数である．ただし手前の層のノード数を N として 1/N を分散とする平均 0 の正規分布で与えることとする．引数 curnode_num は現在の層のノード数、prenode_num は手間の層のノード数、bias=False は bias=True の時バイアス, bias=False の時は重みの初期値を定める．戻り値は重み, バイアスの初期値である.

SoftmaxWithLoss 層

コンストラクタ

必要なインスタンス変数を定義しておく.

foward 関数

戻り値 loss(int) はクロスエントロピー誤差である．また引数は, 層の入力データである x(ndarray), ラベルを one-hot vector 表記した t(ndarray) をとる．y=softmax(x), t をインスタンス変数として保存する．またテキストに従って計算を行う.

backward 関数

戻り値 dx(ndarray) は 1 つ前の層に渡す逆伝搬である．計算はテキストにしたがって行う.

__softmax 関数

softmax 関数を担う関数である．オーバーフローを防ぐために入力データ x の行ごとに最大値をとって, それぞれの行からその最大値を減じる．また次元数を 1 次元にしないために keepdims を True にする．

__cross_entropy_error 関数

クロスエントロピーを計算する関数である．バッチサイズが 1 のときは y,t の次元数が 1 となってしまい計算できなくなってしまうために次元数を 2 にする．また出力値が発散しないように log 関数に適用する際に微少量 delta を加算して計算する．

ReLU 層の説明

コンストラクタ

必要なインスタンス変数を定義しておく．

foward 関数

戻り値 out(ndarray) は出力データである．また引数は, 層の入力データである x(ndarray), 学習時か推論時かを区別する flag(bool) をとる．ただし学習と推論で計算方法は同じであるから flag は用いない．numpy の where 関数を用いることで要素の値が 0 より大きければそのまま, 小さければ 0 に置き換え, 計算を行っている．出力結果 out は逆伝搬の計算に用いるのでインスタンス変数として保存する．

backward 関数

戻り値 dx(ndarray) は 1 つ前の層に渡す逆伝搬である． また引数は 1 つ次の層の逆伝搬である dout(ndarray) をとる．計算はテキストにしたがって行う．

set_optimizer 関数

ReLU 層では学習する重みが存在しないため, 何も行わない.

update 関数

ReLU 層では学習する重みが存在しないため, 何も行わない.

getW 関数

ReLU 層では学習する重みが存在しないため, 空リストを返す.

numW 関数

ReLU 層では学習する重みが存在しないため, 0 を返す.

loadW 関数

ReLU 層では学習する重みが存在しないため, 何も行わない.

Dropout 層の説明

コンストラクタ

引数は出力するノード数の無視する割合である rate(int) をとる．必要なインスタンス変数を定義しておく．

foward 関数

戻り値 out(ndarray) は出力データである．また引数は, 層の入力データである x(ndarray), 学習時か推論時かを区別する flag(bool) をとる．flag が False の時, 学習時を意味し, True の時, 推論時を意味する．学習の時は変数 random に shape が x の shape と同じで要素が 0 以上 1 未満となるようにランダムに値を生成し, 代入する．その後変数 random の要素の値が rate より大きければ 1, そうでなければ 0 を格納するような変数 mask を生成することで擬似的に無視するノードを決めている．最後に mask と x をアダマール積で計算し, out として返す．推論時はノードを全て使うため x に (1-rate) をかけて出力する．

backward 関数

戻り値 dx(ndarray) は 1 つ前の層に渡す逆伝搬である． また引数は 1 つ次の層の逆伝搬である dout(ndarray) をとる．計算はテキストにしたがって行う．

set_optimizer 関数

Dropout 層では学習する重みが存在しないため, 何も行わない．

update 関数

Dropout 層では学習する重みが存在しないため, 何も行わない．

getW 関数

Dropout 層では学習する重みが存在しないため, 空リストを返す．

numW 関数

Dropout 層では学習する重みが存在しないため, 0 を返す．

loadW 関数

Dropout 層では学習する重みが存在しないため, 何も行わない．

Batch_Normalization 層の説明

コンストラクタ

引数は, γ である gamma(float), β である beta(float) をとる．計算の値が発散しないように epsilon を 1e-7 と定めておく．また必要なインスタンス変数を定義しておく．

foward 関数

戻り値 y(ndarray) は出力データである．また引数は, 層の入力データである x(ndarray), 学習時か推論時かを区別する flag(bool) をとる．exp_mean, exp_var はそれぞれ入力データ x の平均と分散の期待値を意味

する．はじめに exp_mean が定義されていなければ初期値を定める．学習時はテキストに従って計算を行い, mean, var, x_conv はそれぞれ入力データの平均, 分散, 正規化した値を意味する．また推論時に平均と分散の期待値を使う必要があるので期待値を近似して随時更新する．推論時は期待値を使って計算を行う．

backward 関数

戻り値 dx(ndarray) は 1 つ前の層に渡す逆伝搬である． また引数は 1 つ次の層の逆伝搬である dout(ndarray) をとる．計算はテキストにしたがって行う．また dx_conv, dsigma, dmean, dx, dgamma, dbeta はそれぞれ $\frac{\partial E_n}{\partial \hat{x_i}}$, $\frac{\partial E_n}{\partial \sigma_B^2}$, $\frac{\partial E_n}{\partial \mu_B}$, $\frac{\partial E_n}{\partial x_i}$, $\frac{\partial E_n}{\partial \gamma}$, $\frac{\partial E_n}{\partial \beta}$ に相当する．

set_optimizer 関数

引数は最適化手法を管理するクラスのインスタンス optimizer をとる．

update 関数

更新すべき重み gamma, beta を optimizer の update メソッドを適用して更新する．

getW 関数

保存すべきパラメータ gamma, beta, exp_mean, exp_var を空 list に append して返す．

numW 関数

保存すべきパラメータ gamma, beta, exp_mean, exp_var の 4 種類存在するので 4 を返す．

loadW 関数

引数は層に設定したい重みを並べたものである list_W(list) をとる．list_W から重みを取り出して gamma, beta, exp_mean, exp_var に代入する．

Convolution 層の説明

コンストラクタ

引数は, filter の枚数である num_filter(int), filter の大きさである size_filter(int), 入力データの shape である input_shape(tuple), パディングの数である padding(int), ストライドの大きさである stride(int) をとる．重みの初期値を __initial_weigth 関数によって定める．また必要なインスタンス変数を定義しておく．

foward 関数

戻り値 out(ndarray) は出力データである．また引数は, 層の入力データである x(ndarray), 学習時か推論時かを区別する flag(bool) をとる．ただし学習と推論で計算方法は同じであるから flag は用いない．num_f, channel_f, height_f, width_f, batch, channel, height, width, OH, OW, col, col_filter はそれぞれフィルターの枚数, チャンネル数, 高さ, 幅, 入力データのバッチ数, チャンネル数, 高さ, 幅, 畳み込み層に通したあとのデータの高さ, 幅, 入力データを二次元に展開したもの, フィルターを二次元に展開したものである．計算はテキストにしたがって行列の計算として実装した．

backward 関数

戻り値 dx(ndarray) は 1 つ前の層に渡す逆伝搬である． また引数は 1 つ次の層の逆伝搬である dout(ndarray) をとる．num_f, channel_f, height_f, width_f, dout, db, dW, dx はそれぞれフィルルターの枚数, チャンネル数, 高さ, 幅, $\frac{\partial E_n}{\partial Y}$, $\frac{\partial E_n}{\partial \boldsymbol{b}}$, $\frac{\partial E_n}{\partial W}$, $\frac{\partial E_n}{\partial X}$ に相当する．計算は Affine 層と基本的に同じだが dout の形を二次元データとして計算した結果を使うことなどに注意して行った．また勾配の計算後に元の形に戻すことも必要になる．

set_optimizer 関数

引数は最適化手法を管理するクラスのインスタンス optimizer をとる．

update 関数

更新すべき重み W, b を optimizer の update メソッドを適用して更新する．

getW 関数

保存すべきパラメータ W, b を空 list に append して返す．

numW 関数

保存すべきパラメータ W, b の 2 種類存在するので 4 を返す．

loadW 関数

引数は層に設定したい重みを並べたものである list_W(list) をとる．list_W から重みを取り出して W, b に代入する．

__initial_weight 関数

重みの初期値をランダムに決め, バイアスの初期時を 0 にする関数である．ただし 0.01 を分散とする平均 0 の正規分布で与えることとする．引数 num_filter はフィルターの枚数、channel_f はチャンネル数、bias=False は bias=True の時バイアス, bias=False の時は重みの初期値を定める．戻り値は重み, バイアスの初期値である．

img2col 関数

4 次元データ (batch 数 ×channel 数 × 高さ × 幅) を 2 次元データ (output_height*output_width*batch 数 ×filter_height*filter_width*channel 数) に変換する関数である．実装の基本的な流れは入力データを filter のサイズだけ batch, channel をまとめて取り出し, 1 列に変換し, 並べるというものである．ただし axis=0 が 1batch のかたまりごとに並ぶように工夫を施している．

col2img 関数

2 次元データ (output_height*output_width*batch 数 ×filter_height*filter_width*channel 数) を 4 次元データ (batch 数 ×channel 数 × 高さ × 幅) に変換する関数である．実装は img2col の逆変換であるから基本的には同じである．

MaxPooling 層の説明

コンストラクタ

引数は, filter の高さである height(int), 幅である width(int), パディングの数である padding(int), ストライドの大きさである stride(int) をとる. また必要なインスタンス変数を定義しておく.

foward 関数

戻り値 out(ndarray) は出力データである. また引数は, 層の入力データである x(ndarray), 学習時か推論時かを区別する flag(bool) をとる. ただし学習と推論で計算方法は同じであるから flag は用いない. batch, channel, height, width, out_h, out_w, col, arg_max はそれぞれ入力データの batch 数, channel 数, 高さ, 幅, 出力データの高さ, 幅, 入力データを 2 次元に展開したもの, 入力データにフィルターをかけたときにどこが最大の値を取ったのかを保存するものである. 実装の基本的な流れは 2 次元データに展開するときにフィルターにかける部分を 1 行に並べて, 順番に行列に積むことで, max 関数で最大値をとることができるようにすることである. argmax は逆伝搬の計算に必要であるから計算し, 出力時に二次元に展開し, max 関数をとったものを元の形に戻す.

backward 関数

戻り値 dx(ndarray) は 1 つ前の層に渡す逆伝搬である. また引数は 1 つ次の層の逆伝搬である dout(ndarray) をとる. pool_size, col, a, b, batch, channel, height, width, out_h, out_w はそれぞれプールフィルターの要素数, dx を二次元に展開したもの, dout の 0 から要素数までの数を 1 列に並べたもの, argmax を 1 列に並べたもの, 入力データの batch 数, channel 数, 高さ, 幅, 出力データの高さ, 幅である. col の要素を全て 0 にしたのち argmax に対応するする部分に dout を代入することで計算を行っている. また勾配の計算後に元の形に戻すことも必要になる.

set_optimizer 関数

MaxPooling 層では学習する重みが存在しないため, 何も行わない.

update 関数

MaxPooling 層では学習する重みが存在しないため, 何も行わない.

getW 関数

MaxPooling 層では学習する重みが存在しないため, 空リストを返す.

numW 関数

MaxPooling 層では学習する重みが存在しないため, 0 を返す.

loadW 関数

MaxPooling 層では学習する重みが存在しないため, 何も行わない.

工夫点

img2col, col2img 関数において計算量を出力サイズの高さ × 幅に抑えたことである．はじめに作成した関数ではこの計算量に加え, さらに batch 数だけ for 文を回す実装となっており, 計算がとても遅くなってしまっていた．しかし batch も channel と同様の考え方で取り出すことで計算量を抑えることに成功した．

問題点

同じクラス内で同じ計算をしてしまっている部分があり, 計算, 記憶資源を削減できる点である．

optimizer.py の説明

はじめに optimizer.py の概要を説明したのちに詳細について説明する. 実装した最適化手法は以下の 6 種類のクラスである．

- SGD(確率的勾配降下法)
- Momentum(慣性項付き SGD)
- AdaGrad
- RMSProp
- AdaDelta
- Adam

それぞれのクラスに update 関数を実装することで統一的に学習できるようにした．

SGD クラスの説明

コンストラクタ

引数は, 学習率 μ である mu をとり, インスタンス変数として代入する．

update 関数

戻り値 param(ndarray) は重みの更新値である．また引数は, 更新する重みである param(ndarray), 勾配である grad(ndarray) をとる．計算はテキストにしたがって行う．

Momentum クラスの説明

コンストラクタ

引数は, 学習率 η である eta, 学習率 α である alpha をとり, インスタンス変数として代入する．また layer ごとに $\alpha, \Delta W$ を学習する必要があるため, それぞれの値を list として alphas, deltaWs に保存する． それぞれ空リストとして初期化する．インスタンス変数 count はそれぞれの layer での学習率の初期値を決めるための flag のようなものである．

update 関数

戻り値 param(ndarray) は重みの更新値である．また引数は, 更新する重みである param(ndarray), 勾配である grad(ndarray), どの層の学習を行っているかを区別するための index(int) をとる．計算はテキストにしたがって行う．

AdaGrad クラスの説明

コンストラクタ

引数は, 学習率 η である eta, 学習率 h である h をとり, インスタンス変数として代入する．また layer ごとに η, h を学習する必要があるため, それぞれの値を list として etas, hs に保存する． それぞれ空リストとして初期化する．インスタンス変数 count はそれぞれの layer での学習率の初期値を決めるための flag のようなものである．

update 関数

戻り値 param(ndarray) は重みの更新値である．また引数は, 更新する重みである param(ndarray), 勾配である grad(ndarray), どの層の学習を行っているかを区別するための index(int) をとる．計算はテキストにしたがって行う．

RMSProp クラスの説明

コンストラクタ

引数は, 学習率 h である h, 学習率 η である eta, 学習率 ρ である rho, 学習率 ϵ である epsilon をとり, インスタンス変数として代入する．また layer ごとに η, h, ρ, ϵ を学習する必要があるため, それぞれの値を list として etas, hs, rhos, epsilons に保存する． それぞれ空リストとして初期化する．インスタンス変数 count はそれぞれの layer での学習率の初期値を決めるための flag のようなものである．

update 関数

戻り値 param(ndarray) は重みの更新値である．また引数は, 更新する重みである param(ndarray), 勾配である grad(ndarray), どの層の学習を行っているかを区別するための index(int) をとる．計算はテキストにしたがって行う．

AdaDelta クラスの説明

コンストラクタ

引数は, 学習率 ρ である rho, 学習率 ϵ である epsilon をとり, インスタンス変数として代入する．また layer ごとに h, s, ρ, ϵ を学習する必要があるため, それぞれの値を list として hs, ss, rhos, epsilons に保存する．
それぞれ空リストとして初期化する．インスタンス変数 count はそれぞれの layer での学習率の初期値を決めるための flag のようなものである．

update 関数

戻り値 param(ndarray) は重みの更新値である．また引数は, 更新する重みである param(ndarray), 勾配である grad(ndarray), どの層の学習を行っているかを区別するための index(int) をとる．計算はテキストにしたがって行う．

Adam クラスの説明

コンストラクタ

引数は, 学習率 α である alpha, 学習率 β_1 である beta1, 学習率 β_2 である beta2, 学習率 ϵ である epsilon をとり, インスタンス変数として代入する．また layer ごとに $\alpha, \beta_1, \beta_2, \epsilon$, t, m, s を学習する必要があるため, それぞれの値を list として alphas, beta1s, beta2s, epsilons, ts, ms, ss に保存する．それぞれ空リストとして初期化する．インスタンス変数 count はそれぞれの layer での学習率の初期値を決めるための flag のようなものである．

update 関数

戻り値 param(ndarray) は重みの更新値である．また引数は, 更新する重みである param(ndarray), 勾配である grad(ndarray), どの層の学習を行っているかを区別するための index(int) をとる．計算はテキストにしたがって行う．

工夫点

1 つの layer に複数の重みの種類が存在するときに 1 つのインスタンスで一括して管理できるようにした点である．

問題点

Network クラスで optimizer を設定するときにインスタンスとして渡してしまうと, 1 つのインスタンスで全ての layer の重みを学習するための学習率を保存しようとしてしまい, 上手くいかなくなる点である．そのため今回はデフォルト引数のみでの指定となり, Network クラス側からパラメータの調整ができなくなっている．

network.py の説明

はじめに network.py の概要を説明したのちに詳細について説明する．Network クラスが実装されており, このクラスには以下の 9 個の関数が実装されている．

- fit 関数 $\cdots$ モデルの学習を行う関数
- save 関数 $\cdots$ モデルの重みを保存する関数
- load 関数 $\cdots$ モデルの重みを読み込む関数
- evaluate 関数 $\cdots$ モデルの accuracy を計算する関数
- __set_optimizer 関数 $\cdots$ optimizer を設定する関数
- __forward 関数 $\cdots$ 順方向の計算を行う関数

- __backward 関数 · · · 逆伝搬の計算を行う関数
- __update 関数 · · · 重みの更新を行う関数
- __one_hot 関数 · · · one_hot vector 形式に変換する関数

Network クラス

コンストラクタ

引数は, model の layer をリストとして積んだものである lists(list), 損失関数をもつ layer である loss_layer(layer), データの batch 数である batch_size(int) をとる. それぞれをインスタンス変数として代入する.

fit 関数

引数は, model の入力データの shape である input_shape(tuple), 学習データ data_x(ndarray), 学習データのラベル data_y(ndarray), エポック数である epoch(int), 設定する optimizer の名前である optimizer_name(string), 1epoch ごとに重みを保存するかどうかの flag である save_flag(bool), 重みの保存先である savedata_path(pathlib.PosixPath) をとる. はじめに optimizer を設定し, epoch 数だけ学習を行う. 学習は順伝搬の計算, 損失関数の計算, 誤差逆伝搬の計算, 重みの更新を繰り返すことで実現する. 1epoch ごとにその最後の loss 値を出力する. また save_flag が True ならば重みを保存する.

save 関数

引数は, 重みの保存先のパスである savedata_path(pathlib.PosixPath) をとる. savedata_path が存在しなければディレクトリを作成する. 積まれた層を順番に取り, 学習すべき重みをリストとして list_W に代入し, 重みを順番に index をつけて path のディレクトリに保存する.

load 関数

引数は, 重みの保存先のパスである savedata_path(pathlib.PosixPath) をとる. savedata_path が存在しなければディレクトリを作成する. 積まれた層を順番に取り, 学習すべき重みをリストとして list_W に path 先のデータから代入し, 重みを index にしたがって layer に load する.

evaluate 関数

戻り値は test データの正答率である accuracy(float) である. 引数は, モデルの入力データの shape である input_shape(tuple), テストデータである test_x, テストデータのラベルである _y をとる. データの数だけ推論を行い, 推論結果とラベル比較し一致すればインクリメントし, 最後にデータサイズで割ることで accuracy を計算する.

__set_optimizer 関数

引数は, optimizer の名前である optimizer_name(string) をとる. optimizer_name にしたがって積んだ層に optimizer を設定する.

__forward 関数

戻り値は model の最後の layer の 1 つ前の layer の出力 x(ndarray) である．引数は, model の入力データである input(ndarray), 学習か推論であるかを判別するための flag(bool) をとる．積まれた layer を順番に取り出し, 計算を行う．

__backward 関数

積まれた layer を逆順に取り出し, 逆伝搬の計算を行う．

__update 関数

積まれた layer を順番に取り出し, 重みの更新を行う．

__one_hot 関数

戻り値は one-hot vector 形式に変換されたラベル (ndarray) である．引数は正解ラベルのデータである Y(ndarray), 取り出したデータの index の集合である data_idxs(ndarray) である．

工夫点

順方向, 逆方向の計算において, model を layer の list として表すことで for 文によって回すことができるようにした点である．また 1epoch ごとに重みを保存できるようにしているため, 好きなタイミングで学習を止めても再び同じ重みで学習を再開できるようにした点である．

問題点

accuracy の計算において batch を 1 として計算しているため計算速度が下がってしまっている点である．

cifar10.py の説明

model を Network クラスとして作成し, 重みデータの読み込み, 学習, 保存, 評価をする．図 1 に model を可視化した．また学習には 3 万枚の画像を使い, 評価には 1 万枚の画像を使用した．

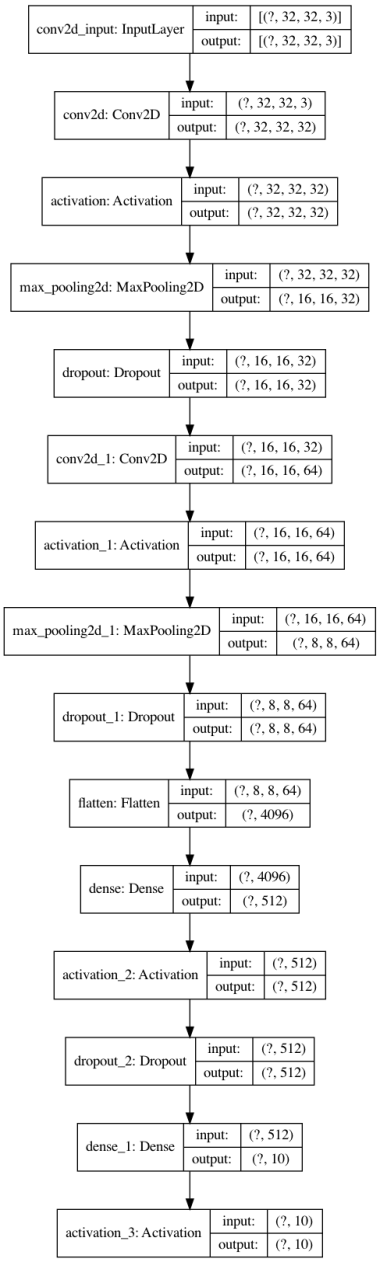

図 1　model

実行結果

100epoch の学習で accuracy は 0.6755 となった.

工夫点

学習モデルを Network クラスとして作成することによって可読性を上げ, keras でのモデル作成, 学習, 評価に近づけた点である.

問題点

モデルを生成するときに batch 数を指定しまうために学習時に batch 数を変更しようとすると再度モデルを作り直す必要がある点である.

main.py の説明

model を Network クラスとして作成し, 重みデータの読み込み, 学習, 保存, 評価をする. model は第 1 層のノード数を 28*28=784, 第 2 層のノード数を 100, 第 3 層のノード数を 10 とし, 順番に Affine, Sigmoid, Affine, SoftmaxWithLoss 層で繋いだものである. また学習には 6 万枚の画像を使い, 評価には 1 万枚の画像を使用した.

実行結果

100epoch の学習で accuracy は 0.9776 となった.

工夫点

最も簡単な 3 層のニューラルネットワークを組んだだけであるから特に工夫点はない.

問題点

畳み込みなどの画像の特徴量を潰して, 列データとして画像を見ているためそれほど accuracy が伸びない点である.