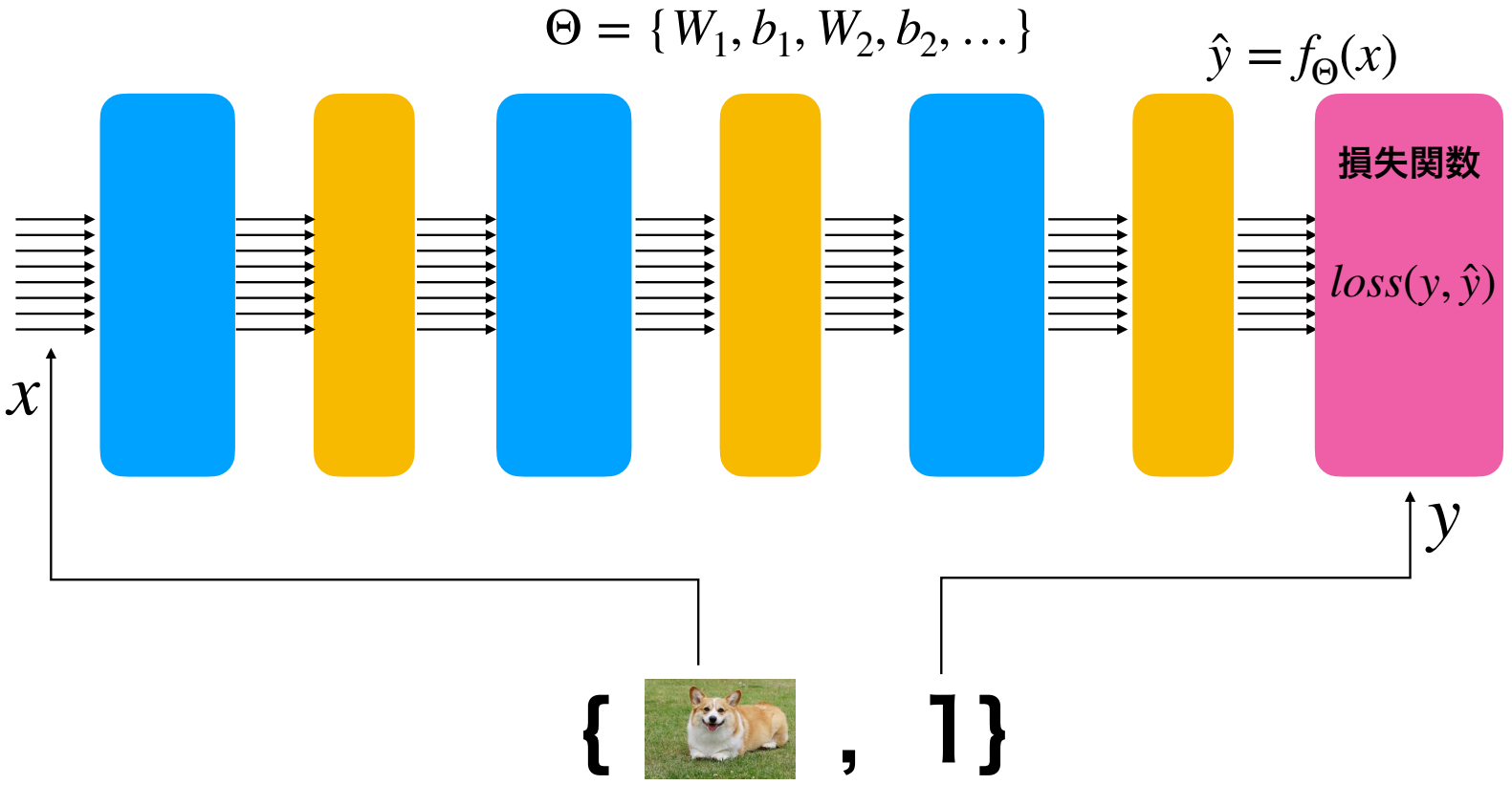
研究室ローテーション 第３回

担当：和田山 正・中井彩乃

本講義の内容

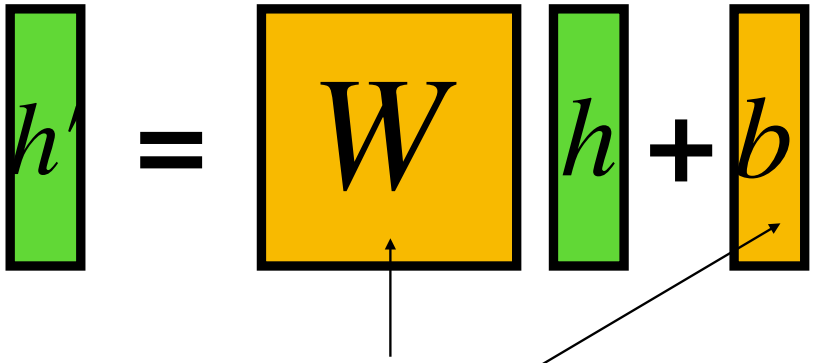
- 深層学習の概要(復習)
- 深層ニューラルネットワークの学習プロセス
- PyTorchを使ってみよう

深層ニューラルネットワークの訓練(学習)



訓練・学習過程では，損失関数値を最小化するように
学習パラメータを変更する

学習プロセス



中身は動かしてよい・うまく
チューンアップしたい！

二乗誤差関数を最小化するようにパラメータを動かせばよい

$$loss(y, \hat{y}) = \frac{1}{2} \sum_{i=1}^{n} |y_i - \hat{y}_i|^2$$

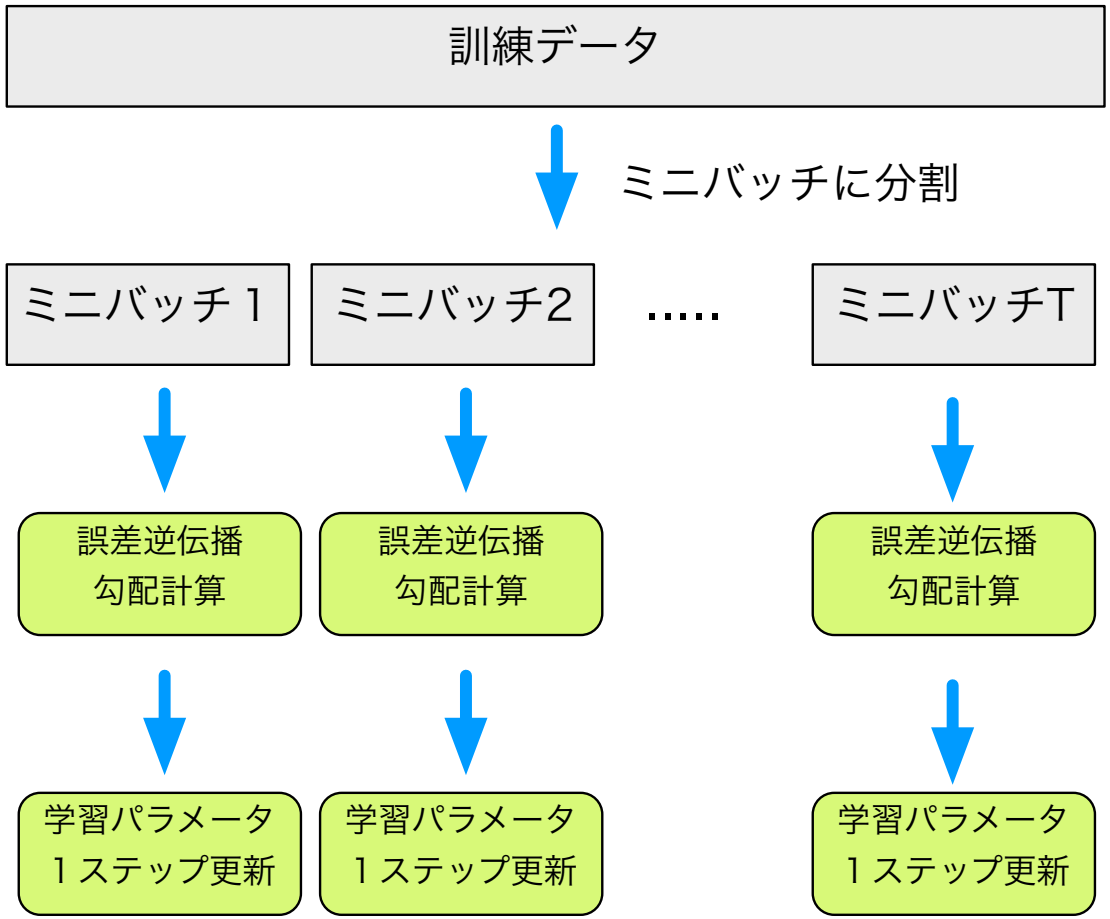
ニューラルネットワークの特徴

- 層構造を持つパラメトリック非線形関数モデル: 学習可能であり、高い表現能力を持つ
- 各層は行列ベクトル積計算（アフィン変換）と非線形関数の要素ごとの適用からなる
- 適切な学習（訓練）プロセスが必要
- 学習プロセスでは、大量の訓練データが必要
- 学習プロセスでは、確率的勾配法を利用してパラメータを調節する

深層ニューラル
ネットワークの学習プロセス

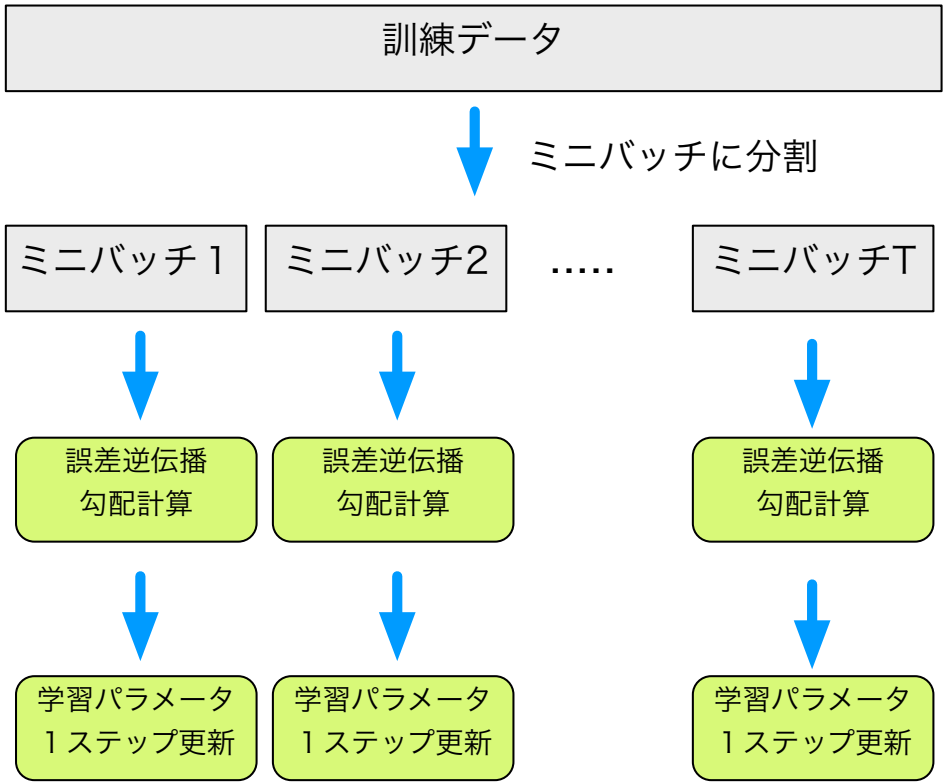
ミニバッチ学習法(1)

データは小分けにして扱うことが一般的



ミニバッチ学習法(2)

訓練データ　$D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_T, y_T)\}$



$$B = \{(x_{b1}, y_{b1}), (x_{b2}, y_{b2}), \ldots, (x_{bK}, y_{bK})\}$$

$$G_B(\Theta) = \frac{1}{K}\sum_{k=1}^{K} loss(y_{bk} - f_\Theta(x_{bk}))$$

目的関数がミニバッチに依存している点に注意

確率的勾配法(SGD)の手続き

Step 1 (初期点設定)　$\Theta := \Theta_0$

Step 2 (ミニバッチ取得)　B をランダムに生成

Step 3 (勾配ベクトルの計算)　$g := \nabla G_B(\Theta)$

Step 4 (探索点更新)　$\Theta := \Theta - \alpha g$

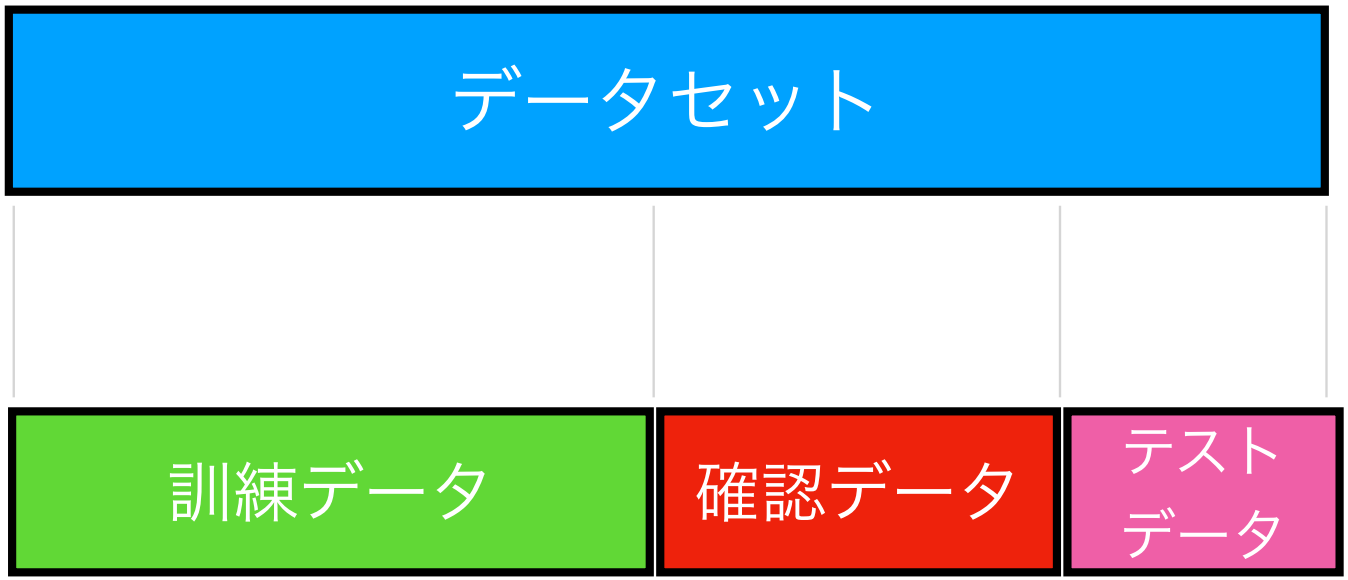
Step 5 (反復)　Step 2 に戻る

ここが確率的

- Momentum (慣性法)
- Adagrad
- Adadelta
- RMSprop
- Adam

- ←に挙げたいろんな亜種があるが，どれがよいかは状況によりけり
- SGD, Momentum, Adam が比較的よく利用されている

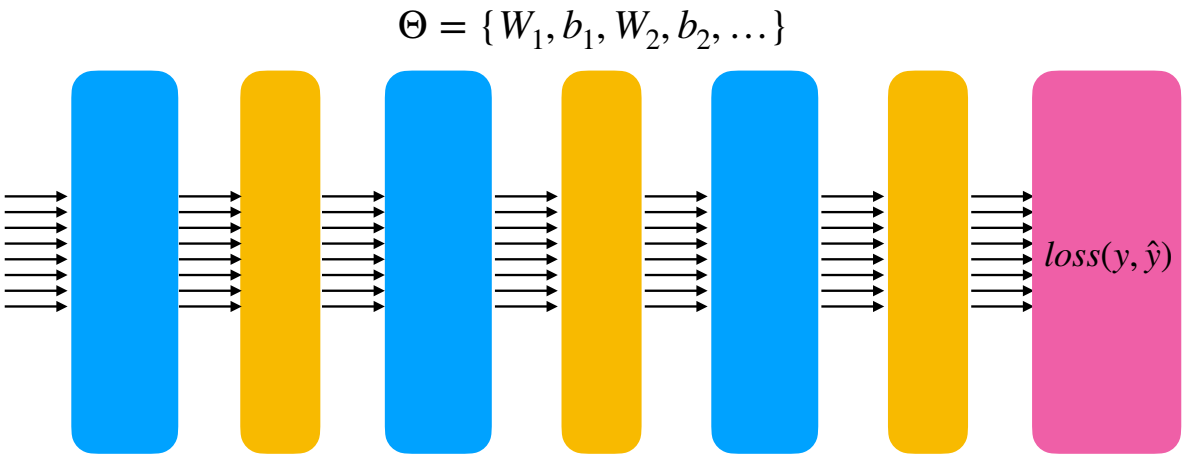
データセットの区分



訓練データを使って学習したモデルの性能を確かめるために，
テストデータを残しておく．
データセットが小規模の場合，データセットの使い回し技法
として，クロスバリデーションなどの技法がしばしば利用される．

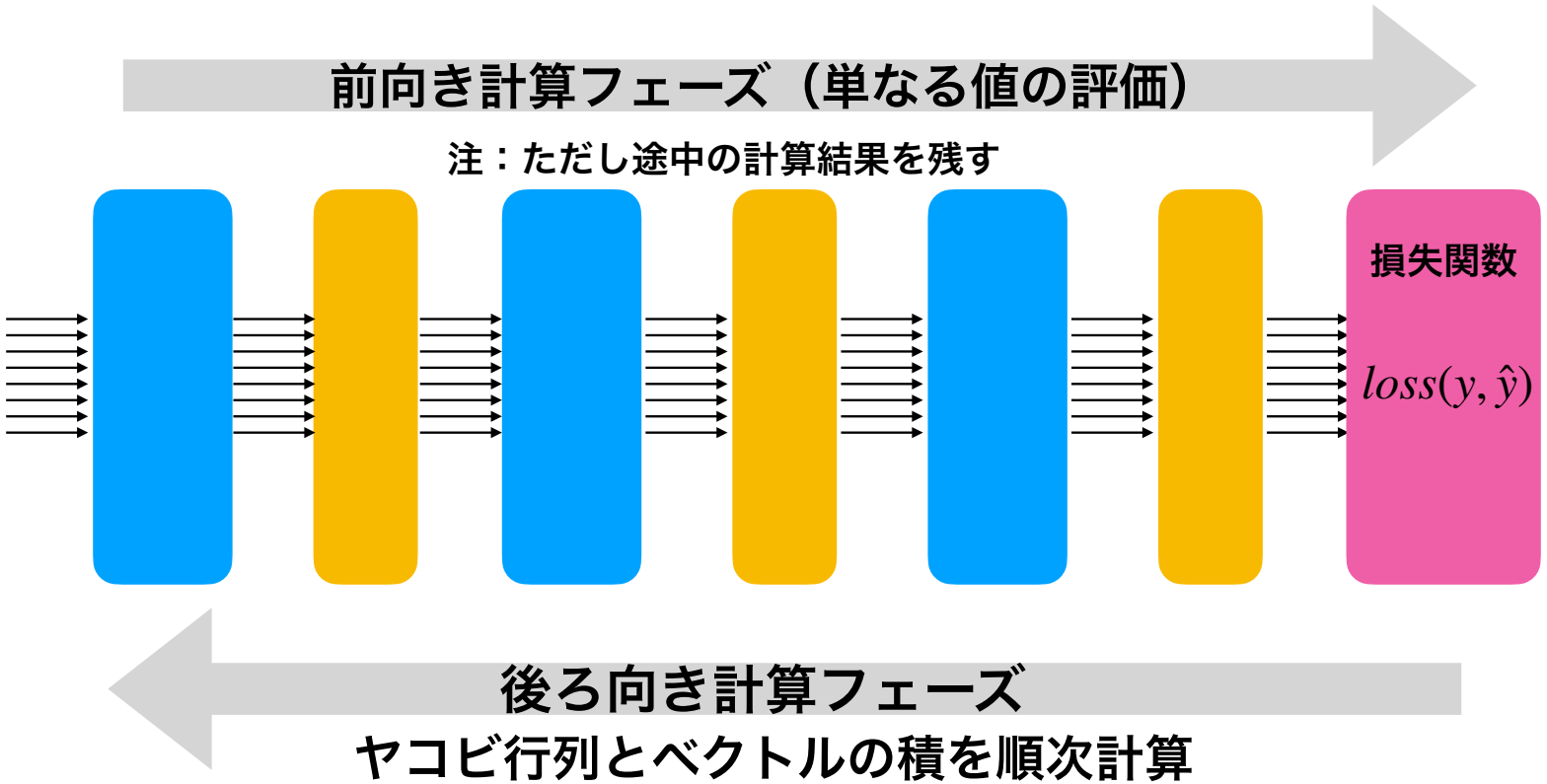
勾配ベクトルの計算

- 確率的勾配法を利用するためには、学習パラメータΘに関する勾配ベクトル(gradient)の計算が必要
- 学習における計算量は勾配計算が支配する
- 層構造のネットワークに対して、効率のよい計算方法とは?



誤差逆伝播法(詳細は次回)

・パラメータの勾配ベクトルを効率良く求めることが目的



中間まとめ

- 最適化技法として確率的勾配法を利用
- 学習パラメータの勾配ベクトルの効率的な計算には誤差逆伝播法(back propagation)を利用
- よい汎化性能(十分に小さい汎化誤差)を得るためには，大量の訓練データが必要
- 一般には，非凸最適化となる

PyTorchを使う

Deep-Learning(DL)フレームワーク

- 深層学習のコードをフルスクラッチで(ゼロから)作るのは辛い！
- DLフレームワークにより簡単にプログラミング可能
- 最も書くのが面倒な誤差逆伝播法の逆方向計算を自動で計算してくれる（自動微分）
- SGDなどの最適化関数やミニバッチ学習の仕組みも内蔵されている
- 現状、TensorFlow、もしくはPyTorchがメジャー

PyTorchのコード例(1)

・PyTorchによるプログラム例

これだけで
ネットワークの
作成が完了

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(2, 2)
        self.fc2 = nn.Linear(2, 2)
    def forward(self, x):
        x = torch.sigmoid(self.fc1(x))
        x = torch.sigmoid(self.fc2(x))
        return x
```

ネットワークの定義部
順方向計算のみを記述すればよい。
誤差逆伝播法の後ろ向き計算フェーズは
明示的にユーザが書く必要ない

これだけで
損失関数と
勾配法の
設定が完了

ネットワークのインスタンス化

```
model = Net()
loss_func = nn.MSELoss()
optimizer = optim.Adam(model.parameters(), lr=0.1)
```

損失関数の指定

オプティマイザの指定

PyTorchのコード例(2)

この5行だけで
勾配法の
更新が完了

```
for i in range(1000):
    inputs = torch.bernoulli(0.5 * torch.ones(mb_size, 2))
    labels = torch.Tensor(mb_size, 2)
    for j in range(mb_size):
        if (inputs[j, 0] == 1.0) and (inputs[j, 1] == 1.0):
            labels[j, 0] = 1.0
            labels[j, 1] = 0.0
        else:
            labels[j, 0] = 0.0
            labels[j, 1] = 1.0
    optimizer.zero_grad()
    outputs = model(inputs)
    loss = loss_func(outputs, labels)
    loss.backward()
    optimizer.step()
```

訓練データのフィード

前向き計算

後ろ向き計算

パラメータ更新

本日のまとめ

- 深層ニューラルネットワークの学習プロセス
- PyTorchを使う