VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Cấu Trúc Dữ Liệu và Giải Thuật (DSA)

DSA2 - HK241

Developing List Data Structures and Artificial Neural Networks

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 9/2024

Mục lục

1 Hướng dẫn ban đầu

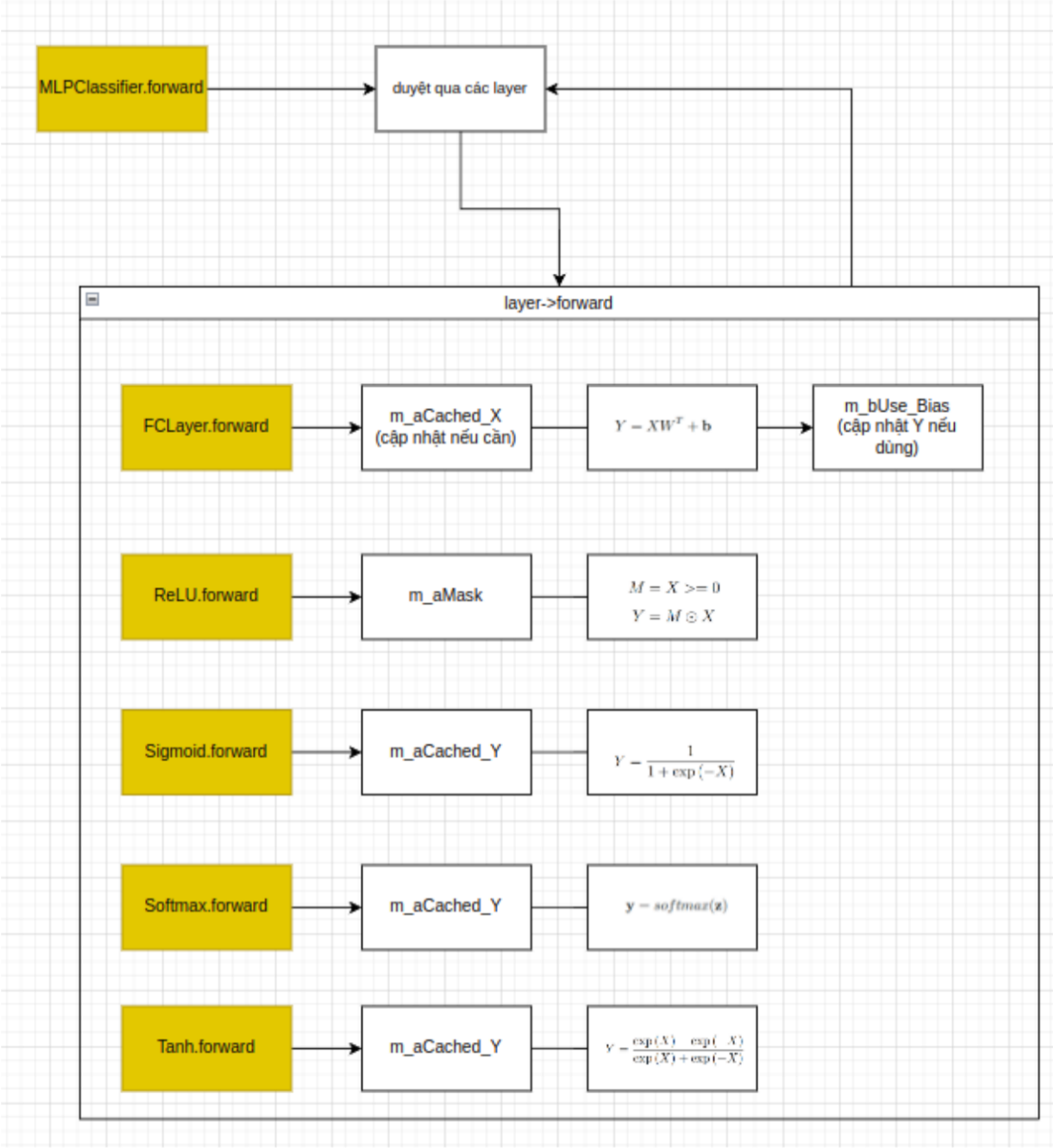
- Tải code thầy về
- đưa code BTL1 + BTL2 task hash và heap vào
- Hiện thực theo các flow sau
- chạy code theo thầy **./compilation-command.sh**

Chú Ý : BTL2 này thầy viết thêm 1 số hàm trong dataloader đưa code vào nhớ chú ý, m_eReduction phải là thuộc tính protected

Test case anh sẽ cập nhật sau và bổ sung thêm phần heap và hash

2 Flow MLPClassifier.forward

2.1 Lý thuyết Lan truyền thuận



Bước lan truyền thuận là một trong những bước chính trong quá trình huấn luyện mạng nơ-ron (neural network). Nó bao gồm việc tính toán đầu ra của mạng từ đầu vào thông qua các lớp nơ-ron. Dưới đây là một cái nhìn tổng quan về lý thuyết và cách thực hiện bước này.

1. Đầu vào: Dữ liệu được cung cấp cho mạng.
2. Các lớp ẩn: Nơi mà dữ liệu được xử lý qua các nơ-ron với trọng số (weights) và độ chệch (biases).
3. Đầu ra: Kết quả cuối cùng của mạng.

2.2 Code Lan truyền thuận

```cpp
// file Code/src/ann/model/MLPClassifier.cpp
double_tensor MLPClassifier::forward(double_tensor X) {
  // YOUR CODE IS HERE
}
```

- Duyệt qua tất cả các layer trong **m_layers** dung foreach thuận trong danh sách liên kết
- Gọi hàm **layer->forward** trả về **Y** sau đó truyền lại Y vào hàm để tiếp tục tính
- Trả về **Y**

```cpp
// file Code/src/ann/layer/FCLayer.cpp
xt::xarray<double> FCLayer::forward(xt::xarray<double> X) {
  // YOUR CODE IS HERE
}
```

- kiểm tra xem đang ở chế độ tranining không để cập nhật **m_aCached_X**
- tính Y như trong pdf thầy **5.2.2.a và 5.2.2.b**
- chọn công thức phù hợp https://xtensor-blas.readthedocs.io/en/stable/reference.html#_CPPv4I00EN2xt6linalg3dotEDaRK11xexpressionI1TERK11xexpressionI1OE
- kiểm tra xem đang ở chế độ tranining không để tính **cộng thêm bias**

```cpp
// file Code/src/ann/layer/ReLU.cpp
xt::xarray<double> ReLU::forward(xt::xarray<double> X) {
  // YOUR CODE IS HERE
}
```

- Cập nhật **m_aMask** và tính **Y** như trong pdf thầy **5.3**
- **Toán tử >=, xt::where**

```cpp
// file Code/src/ann/layer/Sigmoid.cpp
xt::xarray<double> Sigmoid::forward(xt::xarray<double> X) {
  // YOUR CODE IS HERE
}
```

- Cập nhật **m_aCached_Y** và tính **Y** như trong pdf thầy **5.4**
- **Toán tử +, -, *, /, xt::exp**

```cpp
// file Code/src/ann/layer/Tanh.cpp
xt::xarray<double> Tanh::forward(xt::xarray<double> X) {
  // YOUR CODE IS HERE
}
```

- Cập nhật **m_aCached_Y** và tính **Y** như trong pdf thầy **5.5**

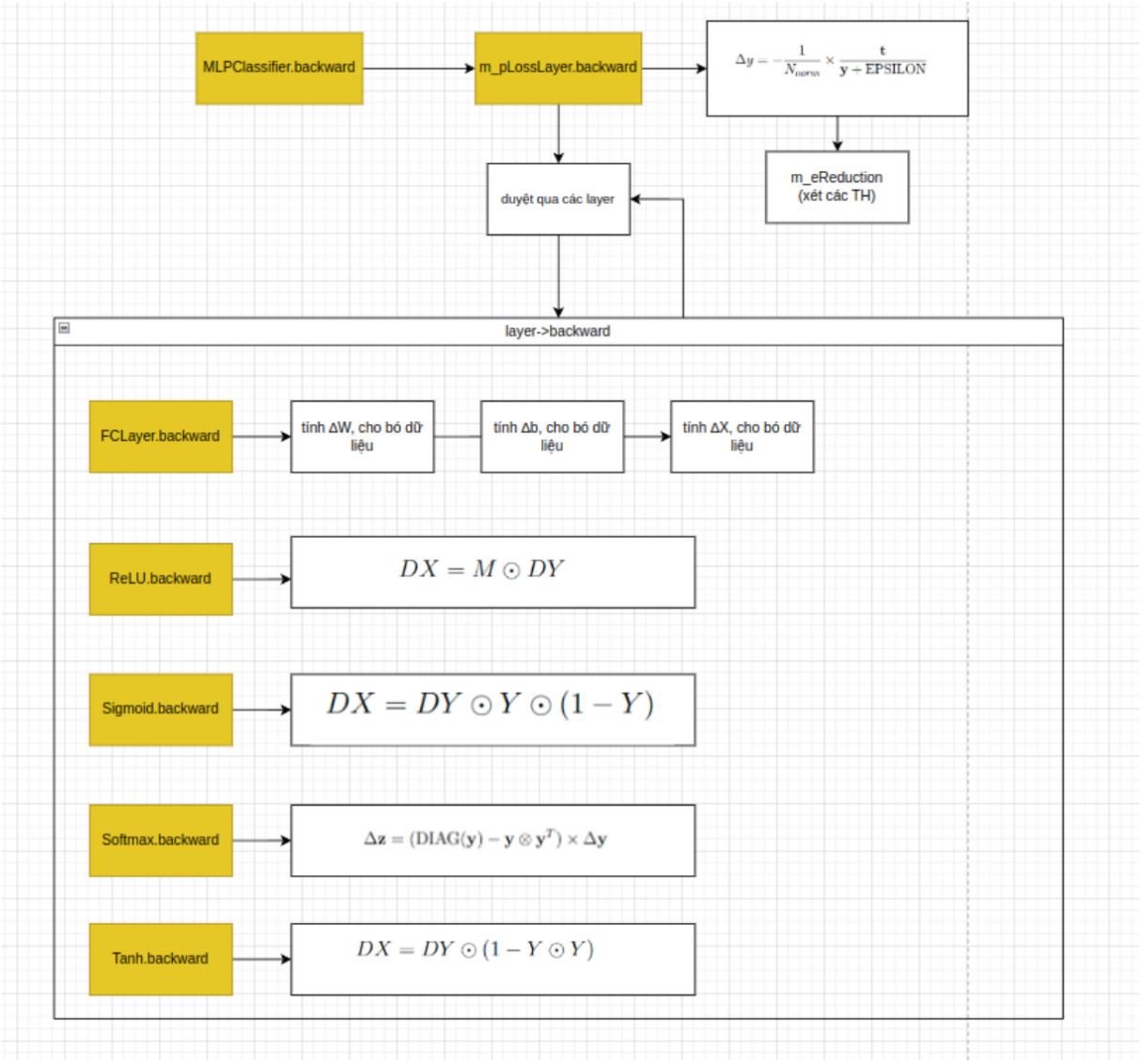
- **xt::tanh**

```cpp
// file Code/src/ann/layer/Softmax.cpp
xt::xarray<double> Softmax::forward(xt::xarray<double> X) {
  // YOUR CODE IS HERE
}
```

- Cập nhật **m_aCached_Y** và tính **Y** như trong pdf thầy **5.6**
- **hàm softmax thầy viết sẵn**

3 Flow MLPClassifier.backward

3.1 Lý thuyết Lan truyền ngược



Bước lan truyền ngược là một quy trình quan trọng trong việc huấn luyện mạng nơ-ron, giúp cập nhật trọng số (weights) và độ chệch (biases) để giảm thiểu lỗi dự đoán của mạng. Đây là bước thiết yếu để tối ưu hóa mạng nơ-ron thông qua thuật toán gradient descent. Dưới đây là một cái nhìn tổng quan về lý thuyết lan truyền ngược.

1. Tính Gradient của Hàm Mất Mát (tính đen ta y)
2. Tính Gradient cho Các Lớp Ẩn (tính đen ta w và đen ta b)
3. Cập Nhật Trọng Số và Độ Chệch (cập nhật w và b mới)

3.2 Code Lan truyền ngược

```cpp
// file Code/src/ann/model/MLPClassifier.cpp
void MLPClassifier::backward() {
  // YOUR CODE IS HERE
}
```

- tính ra được **DY** bằng cách gọi hàm **m_pLossLayer->backward**
- Duyệt qua tất cả các layer trong **m_layers** dung **foreach ngược** trong danh sách liên kết
- Gọi hàm **layer->backward** trả về **DY** sau đó truyền lại DY vào hàm để tiếp tục tính

```cpp
// file Code/src/ann/loss/CrossEntropy.cpp
enum LossReduction { REDUCE_NONE = 0, REDUCE_SUM, REDUCE_MEAN };
xt::xarray<double> CrossEntropy::backward()  {
  // YOUR CODE IS HERE
}
```

- tính **DY** như trong pdf thầy **6.1**
- **EPSILON = 1e-7**
- kiểm tra **m_eReduction**
- **Toán tử +, -, *, /, hàm shape(0)**

```cpp
// file Code/src/ann/layer/FCLayer.cpp
xt::xarray<double> FCLayer::backward(xt::xarray<double> DY) {
  // YOUR CODE IS HERE
}
```

- **Cập nhật m_unSample_Counter**
- tính các đenta như trong pdf thầy **5.3**
- chọn công thức phù hợp https://xtensor-blas.readthedocs.io/en/stable/reference.html#_CPPv4I00EN2xt6linalg3dotEDaRK11xexpressionI1TERK11xexpressionI1OE

```cpp
// file Code/src/ann/layer/ReLU.cpp
xt::xarray<double> ReLU::backward(xt::xarray<double> DY) {
  // YOUR CODE IS HERE
}
```

- tính **DY** như trong pdf thầy **5.3**
- **Toán tử ***

```cpp
// file Code/src/ann/layer/Sigmoid.cpp
xt::xarray<double> Sigmoid::backward(xt::xarray<double> DY) {
  // YOUR CODE IS HERE
}
```

- tính **DY** như trong pdf thầy **5.4**
- **Toán tử +, -, *, /**

```cpp
// file Code/src/ann/layer/Tanh.cpp
xt::xarray<double> Tanh::backward(xt::xarray<double> DY) {
  // YOUR CODE IS HERE
}
```

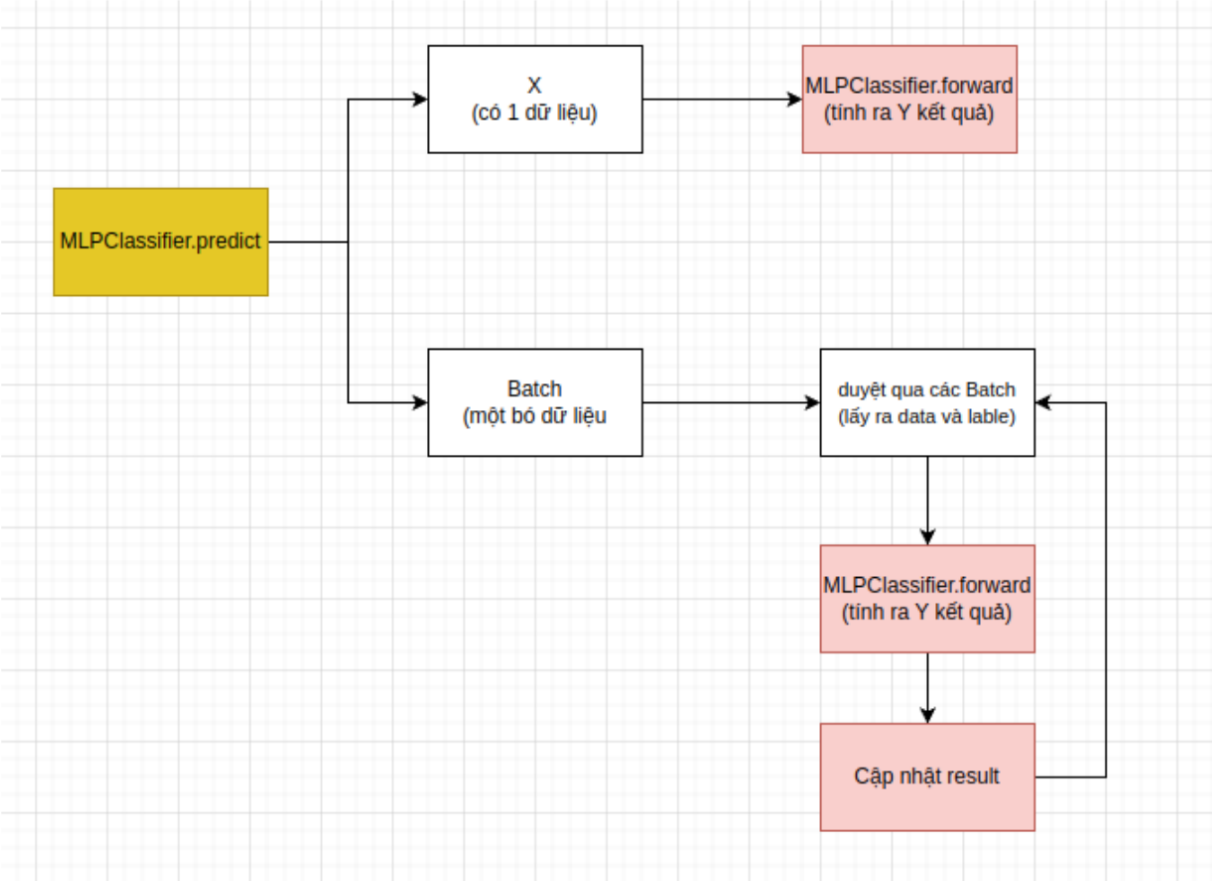
- tính **DY** như trong pdf thầy **5.5**
- **Toán tử +, -, *, /**

```cpp
// file Code/src/ann/layer/Softmax.cpp
xt::xarray<double> Softmax::backward(xt::xarray<double> DY) {
  // YOUR CODE IS HERE
}
```

- tính **DY** như trong pdf thầy **5.6**
- **xt::diag, xt::linalg::outer, xt::linalg::dot**

4 Flow MLPClassifier.predict

4.1 Lý thuyết dự đoán



Dự đoán (predict) trong ngữ cảnh học máy và mạng nơ-ron đề cập đến quá trình sử dụng một mô hình đã được huấn luyện để ước lượng đầu ra cho một tập hợp dữ liệu mới. Quá trình này rất quan trọng trong việc áp dụng mô hình vào các tình huống thực tế, nơi mà mô hình cần đưa ra quyết định hoặc dự đoán kết quả dựa trên đầu vào mới.

1. Chuẩn bị Dữ Liệu
2. Tải Mô Hình
3. Dự Đoán
4. Đánh Giá Kết Quả

4.2 Code Dự đoán

```cpp
// file Code/src/ann/model/MLPClassifier.cpp
double_tensor MLPClassifier::predict(double_tensor X, bool make_decision) {
  // bật chế độ dự đoán
  bool old_mode = this->m_trainable;
  this->set_working_mode(false);

  // YOUR CODE IS HERE

  // quay lại chế độ cũ
  this->set_working_mode(old_mode);

  // muốn trả về phân loại hay xác xuất
  if (make_decision)
    return Y;
  else
    return xt::argmax(Y, -1);
}
```

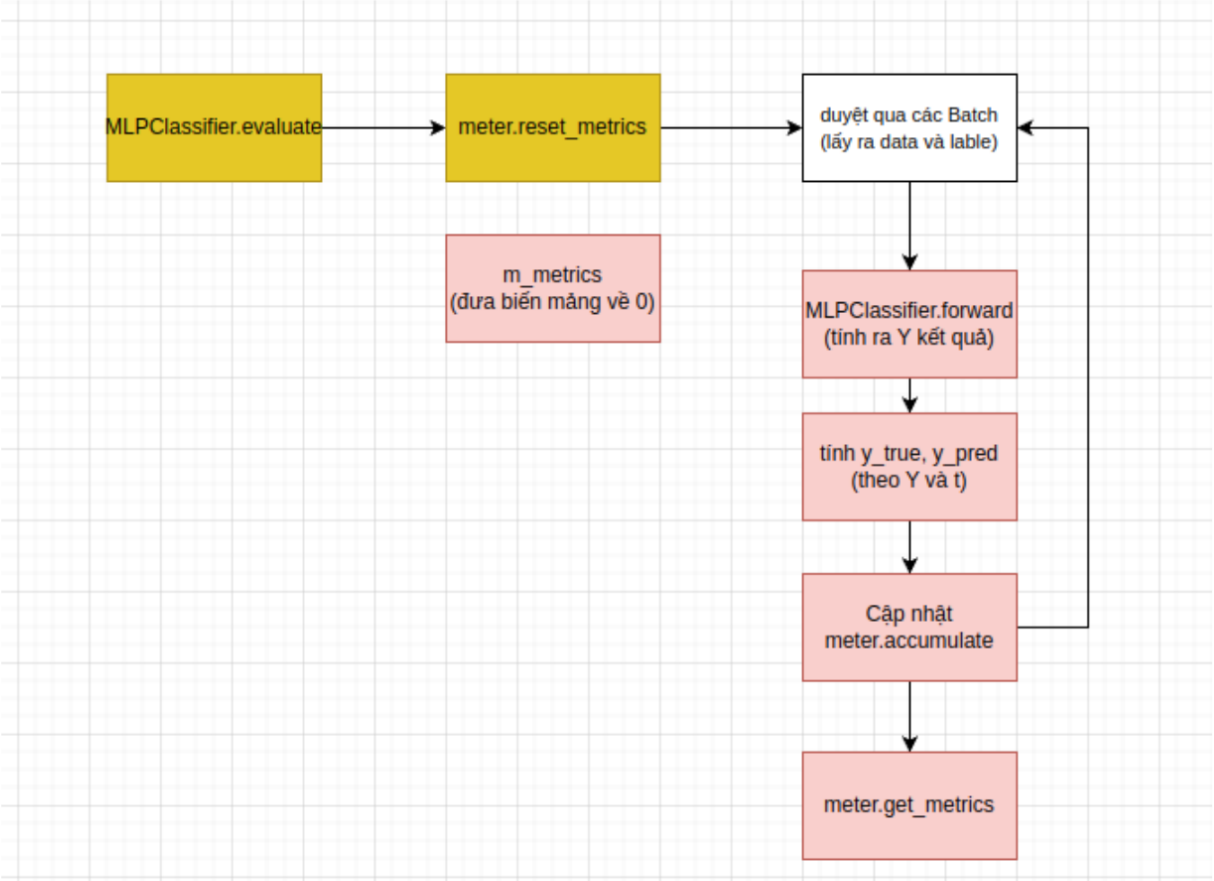
- tính ra được **Y** bằng cách gọi hàm **forward**

```cpp
// file Code/src/ann/model/MLPClassifier.cpp
double_tensor MLPClassifier::predict(DataLoader<double, double>* pLoader,
                                     bool make_decision) {
  bool old_mode = this->m_trainable;
  this->set_working_mode(false);

  double_tensor results;
  bool first_batch = true;

  cout << "Prediction: Started" << endl;
  string info = fmt::format("{:<6s}/{:<12s}|{:<50s}\n", "Batch", "Total Batch",
                            "Num. of samples processed");
  cout << info;

  int total_batch = pLoader->get_total_batch();
  int batch_idx = 1;
  unsigned long long nsamples = 0;
  for (auto batch : *pLoader) {
    // YOUR CODE IS HERE
  }
  cout << "Prediction: End" << endl;

  // restore the old mode
  this->set_working_mode(old_mode);

  if (make_decision)
    return results;
  else
    return xt::argmax(results, -1);
}
```

- lấy ra data trong **batch**
- đưa **batch** vào hàm **forward**
- cập nhật **result** cộng dồn lên **toán tử +, +=, xt::add**

5 Flow MLPClassifier.evaluate

5.1 Lý thuyết đánh giá



Đánh giá (evaluate) là quy trình đánh giá mô hình để đo lường hiệu quả của nó trên dữ liệu chưa từng thấy trong quá trình huấn luyện, như tập kiểm tra (test set) hoặc tập đánh giá (validation set). Việc này là rất quan trọng để xác định khả năng tổng quát hóa của mô hình và hiểu được mô hình có thể áp dụng được trong thực tế đến mức độ nào.

1. Chuẩn bị Dữ Liệu
2. Tải Mô Hình
3. Đo Lường Độ Chính Xác (Accuracy)
4. Tính Toán Hàm Mất Mát (Loss Function)
5. So Sánh Giữa Các Mô Hình

5.2 Code Đánh giá

```cpp
// file Code/src/ann/model/MLPClassifier.cpp
double_tensor MLPClassifier::evaluate(DataLoader<double, double>* pLoader) {
  bool old_mode = this->m_trainable;
  this->set_working_mode(false);

  ClassMetrics meter(this->get_num_classes());
  meter.reset_metrics();

  // YOUR CODE IS HERE
  double_tensor metrics = meter.get_metrics();

  this->set_working_mode(old_mode);
  return metrics;
}
```

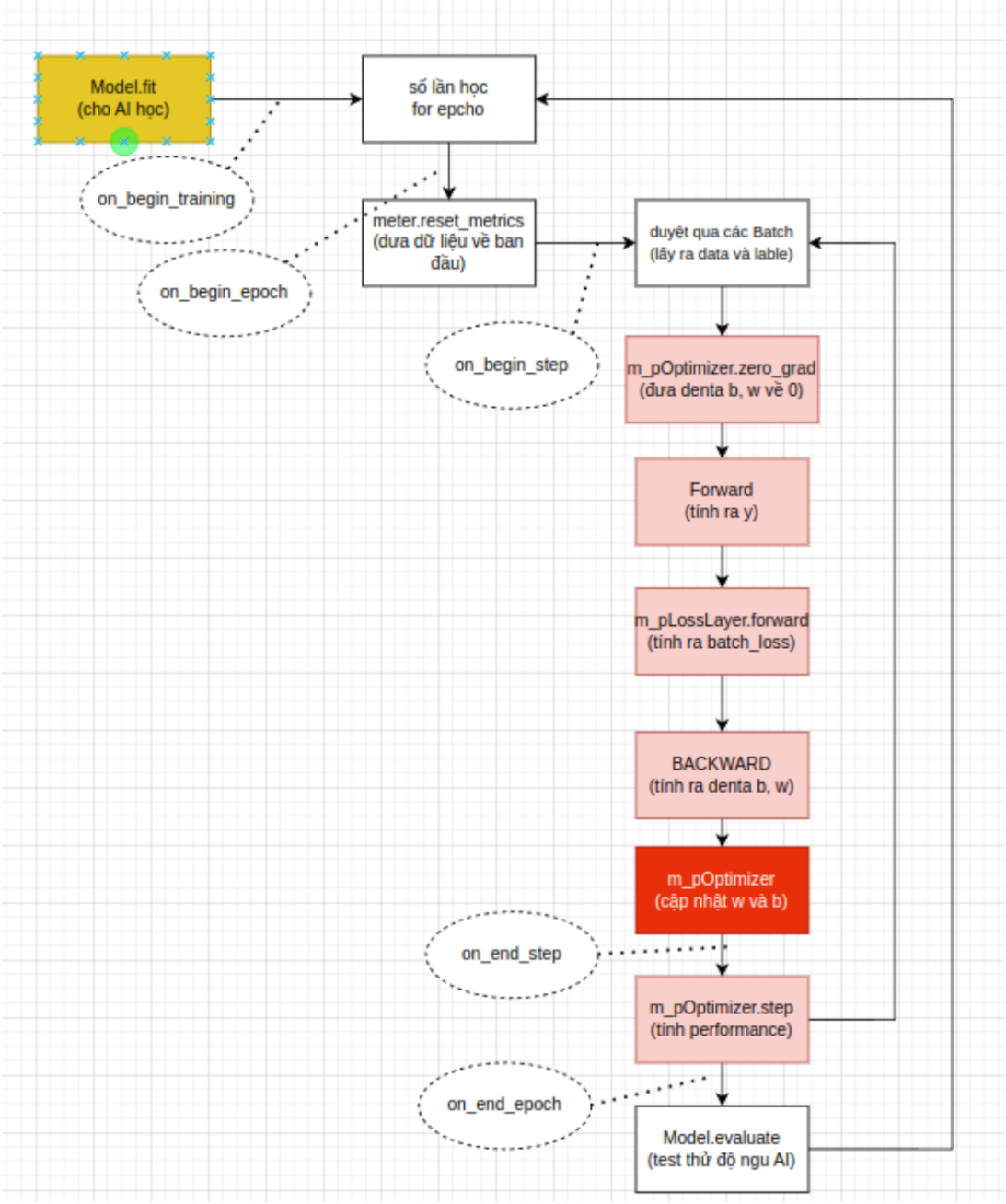
- lấy ra từng data trong **batch** bằng foreach
- tính ra được **Y** bằng cách gọi hàm **forward**
- tính **y_true và y_pred theo t và Y xt::argmax**
- cập nhật **meter.accumulate**

```cpp
// file Code/src/ann/metrics/ClassMetrics.cpp
void ClassMetrics::reset_metrics() {
  // YOUR CODE IS HERE
}
```

- Đặt toàn bộ phần tử của **m_metrics về 0**

6 Flow Model.fit

6.1 Lý thuyết huấn luyện



Huấn luyện(fit) là một phương thức quan trọng để huấn luyện mô hình. Nó thực hiện toàn bộ quy trình điều chỉnh trọng số của mô hình dựa trên tập dữ liệu huấn luyện và nhãn tương ứng. Hàm fit cho phép mô hình "học" từ dữ liệu, tức là tối ưu hóa các tham số (như trọng số và độ chệch) sao cho mô hình có thể đưa ra dự đoán chính xác nhất.

1. Khởi động tham số của mô hình.
   - **trainloader, validloader**: Các *DataLoader* cho các tập huấn luyện và kiểm thử.
   - **optimizer, lossLayer, metricLayer**: Bộ tối ưu, hàm tổn thất và lớp đánh giá hiệu quả.
   - **Các siêu tham số**: *nepochs*, *learning-rate* (α).
   - **Kết quả mong muốn**: (W^*, b^*)
2. **for epoch in range(nepochs) do**
   (a) Bật $train\_mode = true$.
   (b) **for batch in trainloader do**
      i. X, t = batch.data, batch.label.
      ii. Lan truyền thuận (xuyên qua toàn bộ *lossLayer*).
      iii. Lan truyền ngược (từ đầu ra của *lossLayer*).
      iv. Cập nhật tham số của mô hình (sử dụng *optimizer*, α).
      v. Ghi nhận giá trị tổn thất và hiệu quả của mô hình (cần *metricLayer*).
   (c) Đánh giá mô hình với tập kiểm thử (*validloader*).
   (d) In ra thông tin.

6.2 Code huấn luyện

```cpp
// file Code/src/ann/model/IModel.cpp
void IModel::fit(DataLoader<double, double>* pTrainLoader,
                 DataLoader<double, double>* pValidLoader, unsigned int nepoch,
                 unsigned int verbose) {
  //
  on_begin_training(pTrainLoader, pValidLoader, nepoch, verbose);

  for (int epoch = 1; epoch <= nepoch; epoch++) {
    on_begin_epoch();
    m_pMetricLayer->reset_metrics();

    for (auto batch : *pTrainLoader) {
      double_tensor X = batch.getData();
      double_tensor t = batch.getLabel();
      on_begin_step(X.shape()[0]);

      //(0) Set gradient buffer to zeros
      // TODO CODE YOUR

      //(1) FORWARD-Pass
      // TODO CODE YOUR

      //(2) BACKWARD-Pass
      // TODO CODE YOUR

      //(3) UPDATE learnable parameters
      // TODO CODE YOUR

      // Record the performance for each batch
      ulong_tensor y_true = xt::argmax(t, 1);
      ulong_tensor y_pred = xt::argmax(Y, 1);
      m_pMetricLayer->accumulate(y_true, y_pred);

      on_end_step(batch_loss);
    }  // for-each batch: end
    on_end_epoch();
  }  // for-epoch: end
  on_end_training();
}
```

- đưa tất cả đen ta trong **m_pOptimizer** về 0
- tính Y bằng **forward**
- tính **batch_loss là đen ta Y** bằng gọi hàm **m_pLossLayer->forward**
- gọi **backward** để tính đen ta w và b
- **m_pOptimizer->step** cập nhật W và B theo đen ta w và b

```cpp
// file Code/src/ann/loss/CrossEntropy.cpp
double CrossEntropy::forward(xt::xarray<double> X, xt::xarray<double> t) {
  //YOUR CODE IS HERE
}
```

- Cập nhật **m_aCached_Y và m_aYtarget** và tính **Y** như trong pdf thầy **6.1**
- **hàm cross_entropy thầy viết sẵn**

6.3 code các class Optimizer

```cpp
// file Code/src/ann/optim/Adagrad.cpp
IParamGroup* Adagrad::create_group(string name) {
  //YOUR CODE IS HERE
  return pGroup;
}
```

- khởi tạo **new AdaParamGroup** có truyền tham số
- gọi **hash.put** để cập nhật vào hash

```cpp
// file Code/src/ann/optim/Adam.cpp
IParamGroup* Adam::create_group(string name){
    //YOUR CODE IS HERE
}
```

- khởi tạo **new AdamParamGroup** có truyền tham số
- gọi **hash.put** để cập nhật vào hash

```cpp
// file Code/src/ann/optim/AdamParamGroup.cpp
void AdamParamGroup::register_param(string param_name,
        xt::xarray<double>* ptr_param,
        xt::xarray<double>* ptr_grad){
    //YOUR CODE IS HERE
}
```

- gọi **hash.put** để cập nhật vào hash với các value và name được truyền vào
- gọi **hash.put** để cập nhật vào hash với các khởi tạo value và name được truyền vào

```cpp
// file Code/src/ann/optim/AdamParamGroup.cpp
void AdamParamGroup::zero_grad(){
    //YOUR CODE IS HERE
}
```

- cập nhật **m_pGrads** về không để quá thực hiện các quá trình tiếp theo

```cpp
// file Code/src/ann/optim/AdamParamGroup.cpp
void AdamParamGroup::step(double lr){
    //YOUR CODE IS HERE

    //UPDATE step_idx:
    m_step_idx += 1;
    m_beta1_t *= m_beta1;
    m_beta2_t *= m_beta2;
```

```
9  }
10
```

- Tính theo công thức hãy tìm trên mạng về phương pháp này