

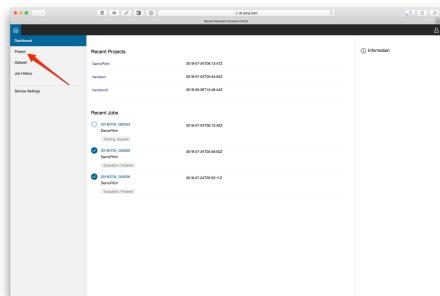
資料のURL <http://bit.ly/nnc-ho-1>

4と9を区別するディープラーニングを体験する

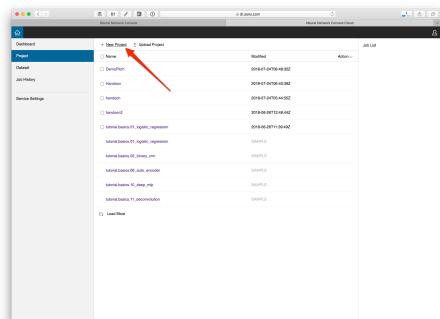
まず最初に28x28に書かれたモノクロの手書き画像を判別するディープラーニングを体験します。

プロジェクトを作成する

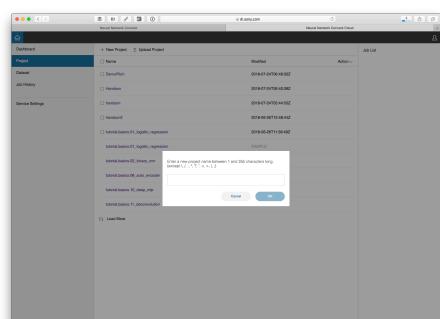
Neural Network Consoleにログインしたら、左側にあるプロジェクト（Project）を選択します。



New Projectを選択します。

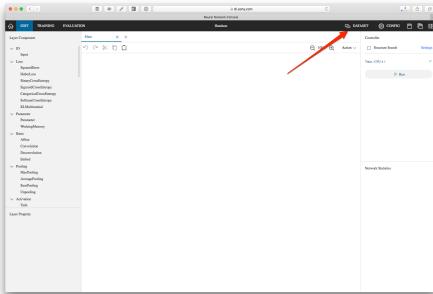


プロジェクト名はアルファベットや英数字が使えます。適当なプロジェクト名を（例えばHandsonなど）入力してOKボタンを押します。

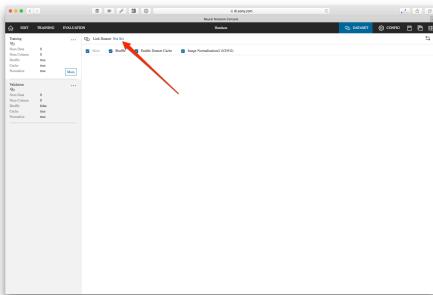


データを読み込む

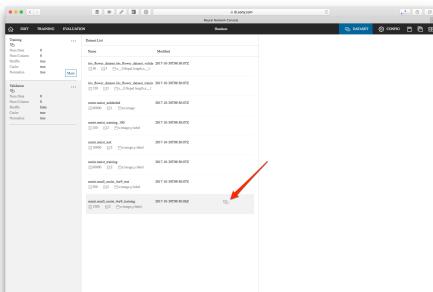
プロジェクトを作成すると、下のような画面が表示されます。まずデータを紐付けるために、右上にあるDATASETをクリックします。



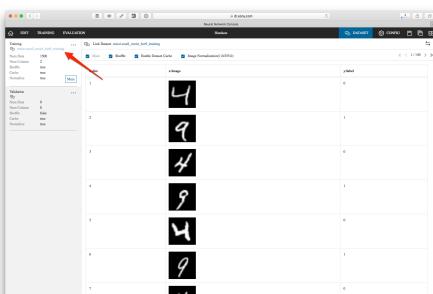
データはトレーニング (Training) と検証 (Validation) に分かれています。最初はトレーニングが選ばれている状態です。Not Setをクリックします。



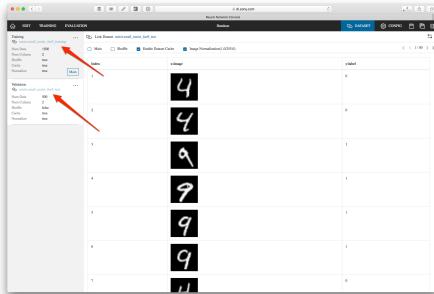
そうするとあらかじめ登録してあるデータが一覧表示されます。その中の mnist.small_mnist_4or9_training にマウスを当て、右側にあるリンクアイコンをクリックします。



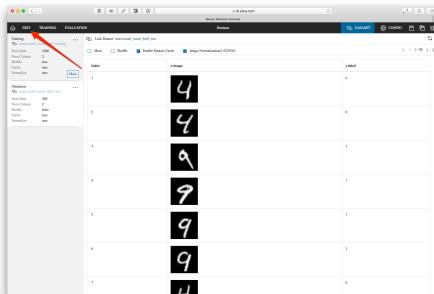
トレーニングのデータをリンクさせると、Trainingと書かれている欄の下にデータ名が表示されます。



同様に検証 (Validation) データとして mnist.small_mnist_4or9_test を紐付けてください。

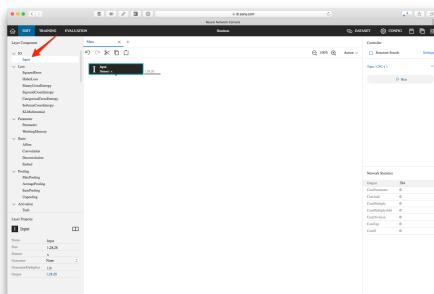


データの紐付けが終わったら、左上にあるEDITをクリックします。



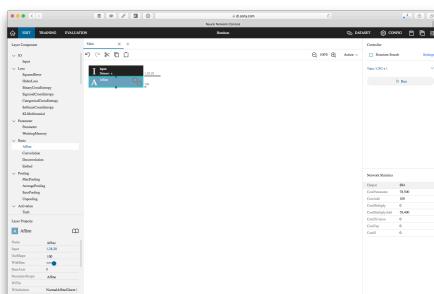
アルゴリズムの設計

ではいよいよ機械学習のアルゴリズムを作っていきます。まず IO カテゴリにある Input をダブルクリックします（またはドラッグ&ドロップ）。これで入力データが追加されました。



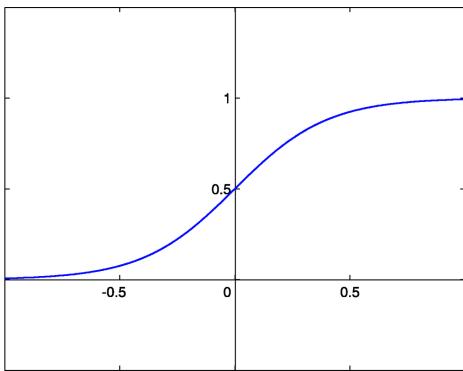
続いてBasicにあるAffineを追加します。Affineで画像データの表示位置を調整します。このAffineを選択して、OutShapeを1にします。

Affineはアフィン変換を行うノードです。アフィン変換は画像や数字の均質化を行います。バラバラな場所にある画像を回転、拡大縮尺を行うことで同じような位置に配置します。

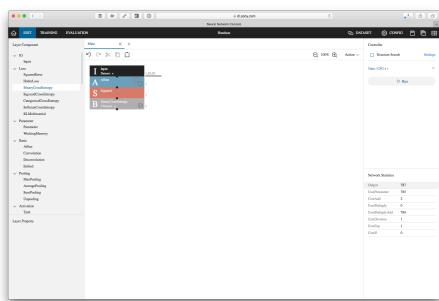


さらにActivation（活性化）としてSigmoidを追加します。これはシグモイド関数になります。シグモイド関数は以下のようなグラフを描きます。入力値（横軸）が-1～+1で変化する際に、その結果を0～1の範

囲で出力します。-0.5以下または+0.5以上の数字はそれぞれ0または1に丸められるのが分かります。

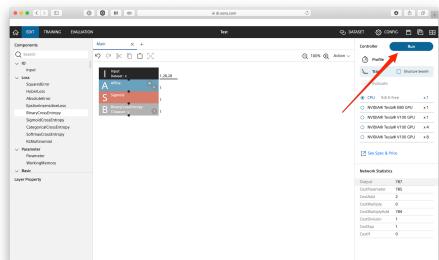


最後に出力としてLossにあるBinaryCrossEntropyを追加します。BinaryCrossEntropyは（バイナリと付いている通り）0または1という結果を出すのに用います。

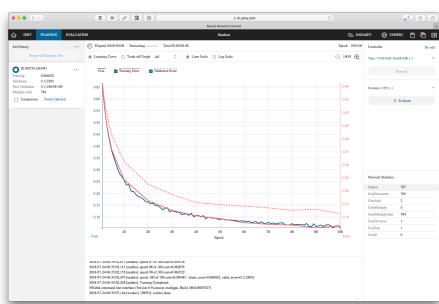


トレーニングの開始

アルゴリズムの設計が終わったら、右側にある Run ボタンを押します。



そうすると TRAINING タブに表示が移ってトレーニングが開始されます。CPUであったり、多人数で一気に行うとキューが詰まってしまうかも知れません。その場合には終わるまでお待ちください。トレーニングが終わるとグラフが表示されます。トレーニングが収束しているのを確認してください。



グラフの見方は以下の通りです。

- Cost

最適化段階でのロス関数の出力値

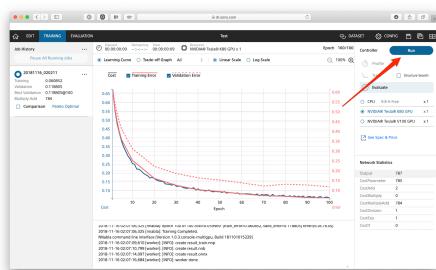
- Error

Epochの終了時点での学習データ、評価データそれぞれにおけるロス関数の出力値

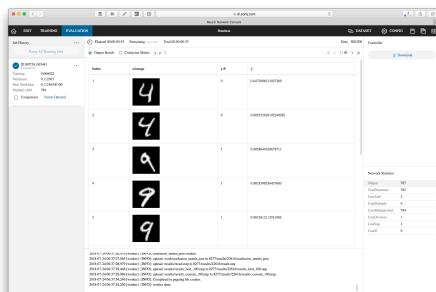
になります。右に進む（世代数が増える）に連れてCostとError両方の値が下がっていれば正しく最適化が行われています。

評価の開始

続いて右側にある Run ボタンを押します。

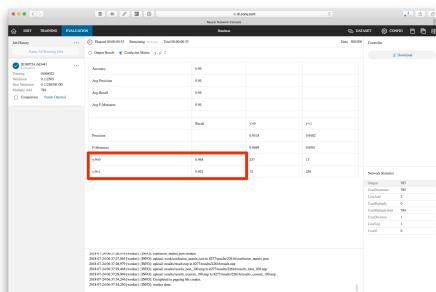


そうすると EVALUTE タブに表示が移って検証が行われます。処理が終わると画像の判定結果が表示されます。

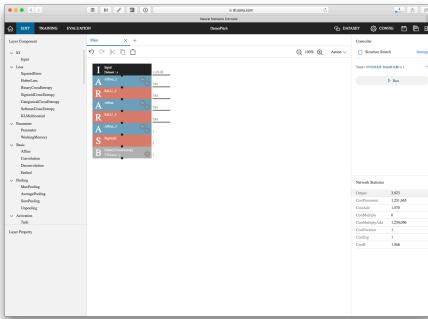


さらに上にあるConfusion Matrixを選ぶと処理全体の評価が確認できます。例えば今回は95%の精度で4と9を分類できたことが分かります。

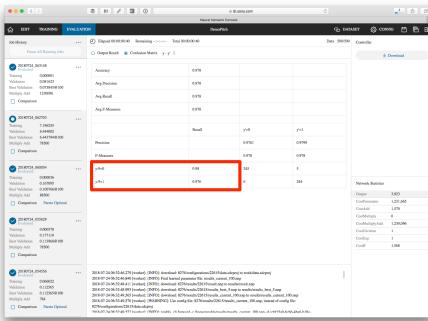
さらに詳細に書くと、今回は 9 ではない (9=0) つまり 4 の画像のうち、正しく 4 と判断できたものが 94.8%、9 の画像 (9=1) のうち、9 と正しく判断できたものが 95.2% であったと表示されています。



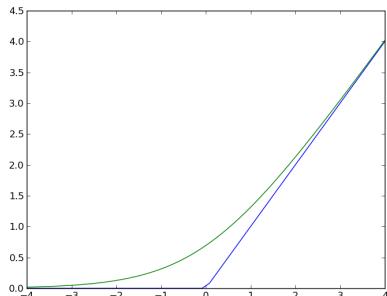
現状のアルゴリズムに手を加えて再度トレーニング、評価してどう結果が変わるのが確認してみましょう。例えば以下のようなアルゴリズムを組んだとします。



この結果は約98%の精度で4と9を分類できています。ReLUはランプ関数と呼びます。



ReLUは以下のようなグラフを描きます（青線）。0以下のは0、0以上は入力値をそのまま返す関数です。



複数のトレーニングを行うことで、それぞれのグラフを比較できるようになります。TRAINING タブで、比較したいトレーニングデータの Comparison をチェックしてください。

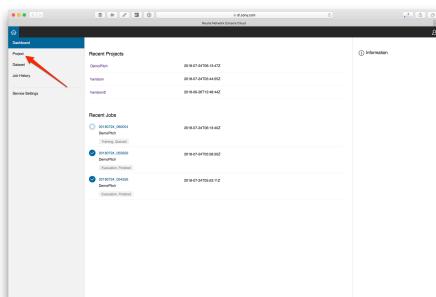
あやめの種類を見分けるディープラーニングを体験する

次にあやめ (iris) の種類を見分けるディープラーニングを体験します。あやめは花びらの幅と長さ、がくの幅と長さの4つの要素によってIris setosa、Iris Versicolour、 Iris Virginicaの3つの種類に分類できます。

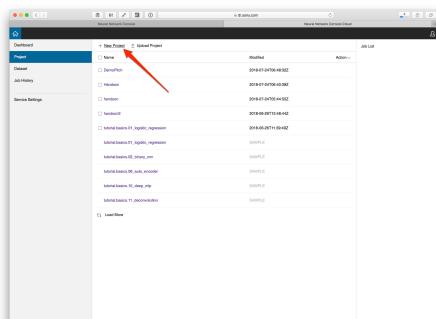
ぱっと見ただけでは殆ど区別がつきません。しかし上記のパラメータを用いることで、機械によって判別できるようになります。

プロジェクトを作成する

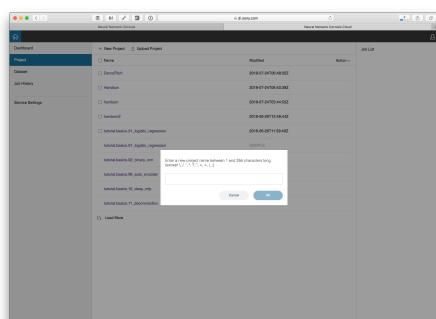
Neural Network Consoleにログインしたら、左側にあるプロジェクト (Project) を選択します。



New Projectを選択します。

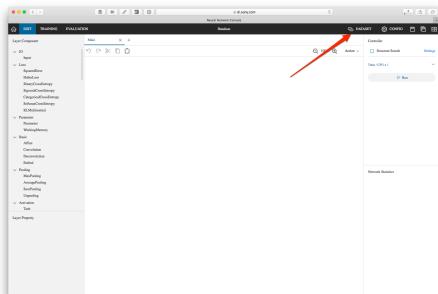


プロジェクト名はアルファベットや英数字が使えます。適当なプロジェクト名を (例えばHandsonなど) 入力してOKボタンを押します。

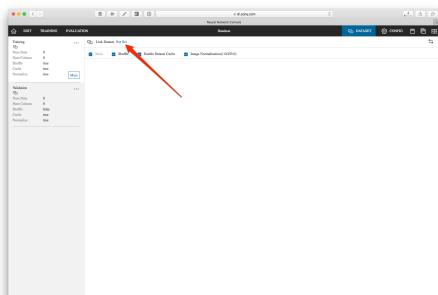


データを読み込む

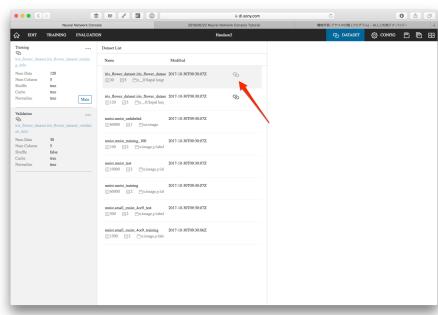
プロジェクトを作成すると、下のような画面が表示されます。まずデータを紐付けるために、右上にある DATASETをクリックします。



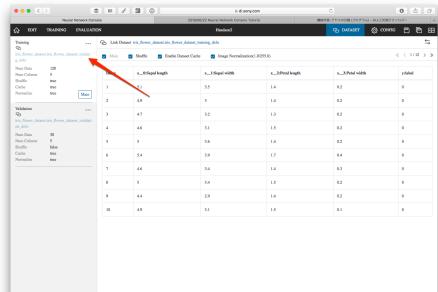
データはトレーニング（Training）と検証（Validation）に分かれています。最初はトレーニングが選ばれている状態です。Not Setをクリックします。



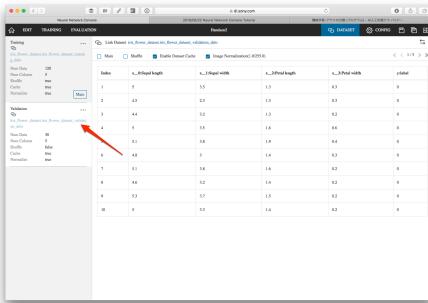
そうするとあらかじめ登録してあるデータが一覧表示されます。その中の `iris_flower_dataset.iris_flower_dataset_training_dele` にマウスを当て、右側にあるリンクアイコンをクリックします。



トレーニングのデータをリンクさせると、Trainingと書かれている欄の下にデータ名が表示されます。



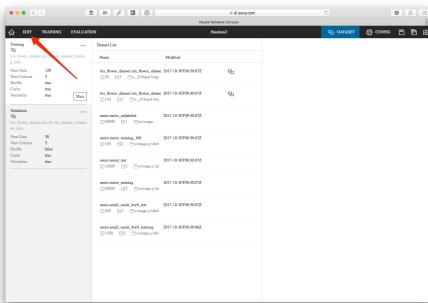
同様に検証 (Validation) データとして iris_flower_dataset.iris_flower_dataset_validation_del0 を紐付けてください。



The screenshot shows the 'DATA' tab of the Neural Network Designer. A red arrow points to the 'EDIT' button in the top-left corner of the main workspace. The workspace displays a table with columns: Index, sepal length, sepal width, petal length, petal width, and class. The data consists of 15 entries from the Iris dataset, with the last row being the validation set.

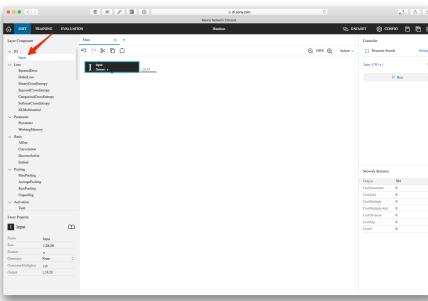
Index	sepal length	sepal width	petal length	petal width	class
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.4	1.4	0.3	Iris-setosa
6	5.4	3.9	1.7	0.4	Iris-setosa
7	4.6	3.4	1.4	0.2	Iris-setosa
8	5.0	3.4	1.5	0.2	Iris-setosa
9	5.5	4.5	1.4	0.2	Iris-setosa
10	4.9	3.1	1.5	0.1	Iris-versicolor
11	4.7	3.2	1.3	0.2	Iris-versicolor
12	4.6	3.1	1.4	0.2	Iris-versicolor
13	5.0	3.6	1.4	0.2	Iris-versicolor
14	5.4	4.5	1.7	0.4	Iris-versicolor
15	5.1	3.3	1.7	0.2	Iris-versicolor
16	5.7	4.4	1.5	0.4	Iris-virginica
17	5.0	3.4	1.6	0.4	Iris-virginica
18	5.4	3.9	1.7	0.4	Iris-virginica
19	5.1	3.5	1.4	0.3	Iris-virginica
20	5.7	4.4	1.5	0.4	Iris-virginica

データの紐付けが終わったら、左上にあるEDITをクリックします。

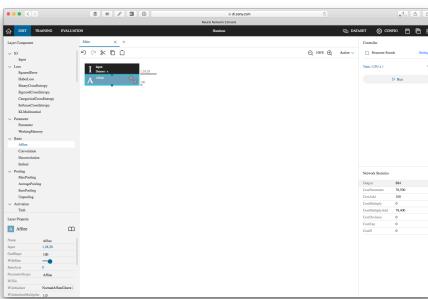


アルゴリズムの設計

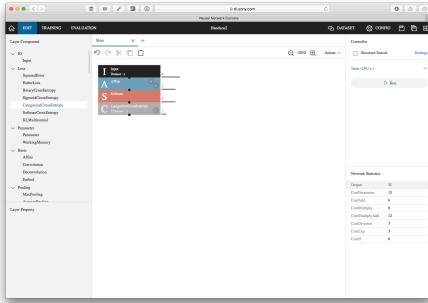
ではいよいよ機械学習のアルゴリズムを作っていきます。まず IO カテゴリにある Input をダブルクリックします（またはドラッグ&ドロップ）。これで入力データが追加されました。サイズ (Size) を4とします（パラメータが4要素のため）。



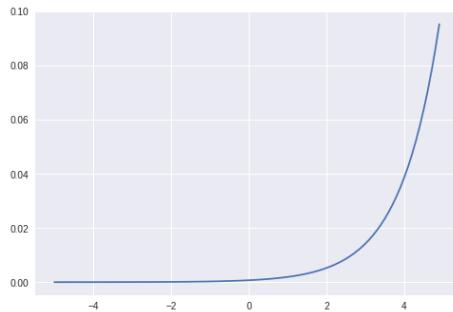
続いてBasicにあるAffineを追加します。このAffineを選択して、OutShapeを3にします（分類が3パターンのため）。



さらにActivation（活性化）としてSoftmaxを追加します。これはソフトマックス関数になります。ソフトマックス関数によって、結果を確立に変換します。最後に出力としてLossにあるCategoricalCrossEntropyを追加します。

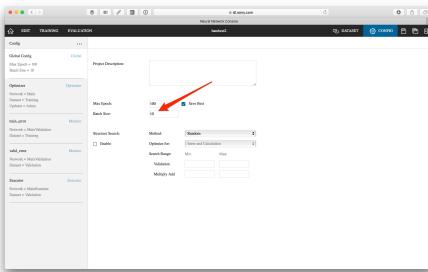


ソフトマックス関数は以下のようなグラフを描きます。入力値を確率値に変換するのに用いる関数です。今回であれば、それぞれのアヤメである可能性を算出します。



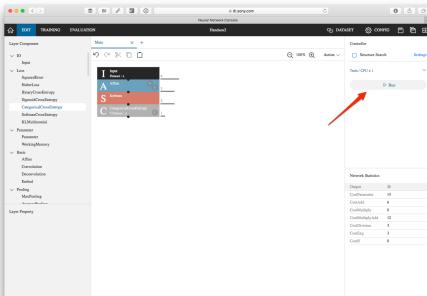
CONFIGの変更

右上にあるCONFIGをクリックしてCONFIGを表示します。Batch Sizeを10とします。

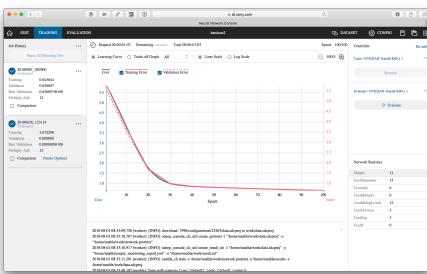


トレーニングの開始

アルゴリズムの設計やCONFIGの変更が終わったら、EDITタブに戻って右側にあるRunボタンを押します。

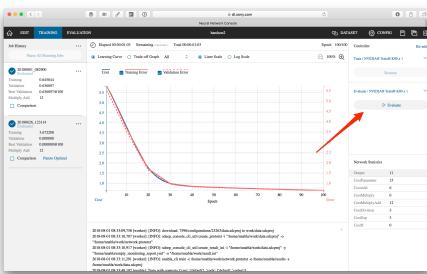


そうすると TRAINING タブに表示が移ってトレーニングが開始されます。CPUであったり、多人数で一気に行うとキューが詰まってしまうかも知れません。その場合には終わるまでお待ちください。トレーニングが終わるとグラフが表示されます。トレーニングが収束しているのを確認してください。



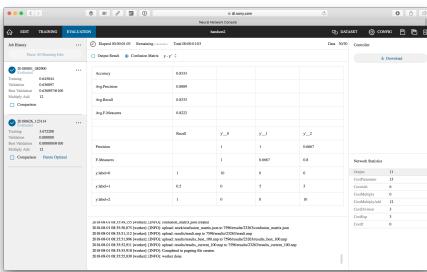
評価の開始

続いて右側にある Run ボタンを押します。



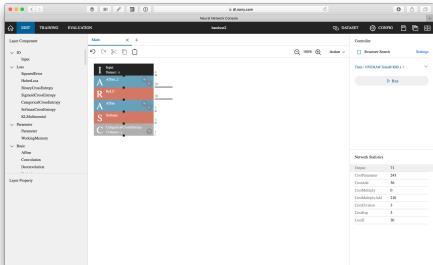
そうすると EVALUATE タブに表示が移って検証が行われます。処理が終わると評価結果が表示されます。評価結果を見ると、次のようなことが分かります。

- Iris Setosa (y=0) → 10/10正しく評価
- Iris Versicolour (y=1) → 10あるデータの内、5つしか正しく判定できていない
- Iris Virginica (y=2) → 10あるデータは10正しく判定、ただし y=1のデータが5つ誤判定

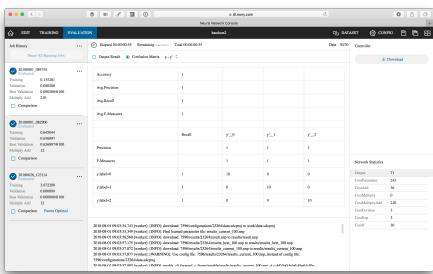


アルゴリズムの更新

現状のアルゴリズムに手を加えて再度トレーニング、評価してどう結果が変わるのが確認してみましょう。例えば以下のようなアルゴリズムを組んだとします。



この結果は100%正確に分類分けできます。



複数のトレーニングを行うことで、それぞれのグラフを比較できるようになります。TRAINING タブで、比較したいトレーニングデータの Comparison をチェックしてください。

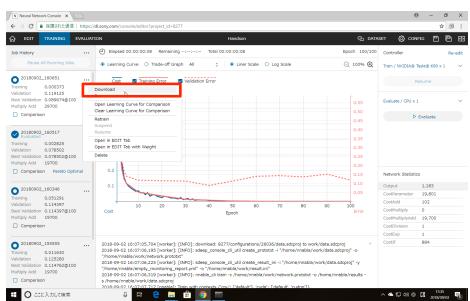
作成したモデル、ネットワークをローカルコンピュータで体験する

では作成した4か9かを見分けるディープラーニングをローカルコンピュータで使ってみましょう。CLI（コマンドライン プログラミング インタフェース）であれば今回はプログラミングを書かずに試す方法を紹介します。なお、今回はクラウド版を対象に紹介します。

ネットワークと学習済みのパラメータをダウンロードする

まず作成したネットワークと学習済みのパラメータをダウンロードします。これはトレーニング (TRAINING) または検証 (EVALUTION) タブに移動して、ジョブ履歴の中からダウンロードしたい学習結果の三点リーダー (...) をクリックします。そして出てきたメニューのDownload > NNP(Neural Network Libraries file format)をクリックします。

最初の課題である4/9を判別するディープラーニングのネットワークをダウンロードしてください。アヤメではありません。



そうすると result_train.nnp というファイルがダウンロードされます。これがディープラーニングのネットワークと学習済みのパラメータが入った内容になります。

ここからPython環境を整えますが、Google Colaboratoryを使う場合、Pythonをインストールする必要はありません。ただし、Webアプリケーションを試すことはできませんので注意してください。

=====

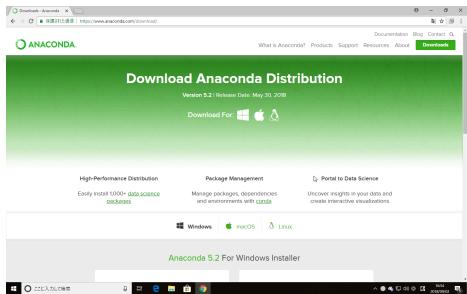
ここからはPythonをローカルにインストールする方向け

Google Colaboratoryは後述しますのでスキップしてください

Python環境を整える

このファイルを扱うためにPython 3.6をインストールします。ディープラーニングを行っていくのであれ

ば素のPythonよりも[Anaconda](#)をインストールする方が良いでしょう。Anacondaはディープラーニングを行うのに便利な、必須とも言えるライブラリがあらかじめインストールされています。macOSやLinuxについてもデフォルトでは3.7系のPythonになってしまい、nnablaがインストールできなかったり、算術系のライブラリが不足しているのでAnacondaの利用をお勧めします。



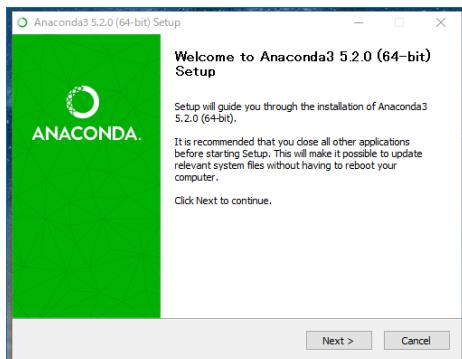
AnacondaはWindows/macOS/Linux向けにそれぞれリリースされています。自分の環境に合わせたものをダウンロード、インストールしてください。

[Downloads - Anaconda](#)

Anacondaを実行する

Windowsの場合は、Anacondaをスタートメニューから実行します。そうするとパスにPythonなどが入った状態のコマンドプロンプトが立ち上ります。macOSやLinuxの場合はターミナルを立ち上げて、以下のコマンドを入力するとAnacondaのPythonが優先実行されます。

```
export PATH="/anaconda3/bin:$PATH"
```



まずPythonのライブラリ管理であるpipをアップデートします。

```
python -m pip install --upgrade pip
```

次にディープラーニング用ライブラリのnnablaをインストールします。

```
pip install nnabla
```

nnablaをインストールすると、以下のコマンドが使えるようになります。

```
nnabla_cli
```

これがNNCを試す際のコマンドになります。これで準備完了です。

=====

ここまでPythonをローカルにインストールする方向け

「デモをダウンロードする」までスキップしてください。

Google Colaboratoryで環境を整える

[Google Colaboratory](#)にアクセスして「新しいPython3のノートブック」を作成します。

ノートブックが開いたら、まずディープラーニング用ライブラリのnnablaをインストールします。!はGoogle Colaboratoryでコマンドを実行する際に付与するプリフィックスです。

```
!pip install nnabla
```

記述した後、Shift + Enterで実行されます。

nnablaをインストールすると、以下のコマンドが使えるようになります。

```
!nnabla_cli
```

これがNNCを試す際のコマンドになります。これで準備完了です。

デモをダウンロードする

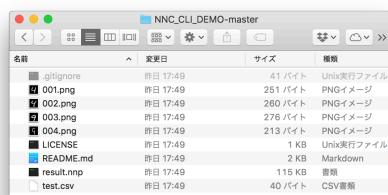
デモ用のリポジトリを用意しています。こちらをダウンロードして、Zipを展開してください。

ローカル実行の場合

```
http://bit.ly/nnc_cli_demo
```

Google Colaboratoryの場合

```
!wget https://github.com/goofmint/NNC_CLI_DEMO/archive/master.zip  
!unzip master.zip  
!cd NNC_CLI_DEMO-master
```



中には以下のようなファイルがあります。

- 001~004.png
テスト用の画像
- result.nnp
あらかじめダウンロードしておいたNNCのモデル/ネットワーク
- test.csv
テスト用のCSVファイル
- web/
[ハンズオン4回目](#)の内容です

このディレクトリに、あなたがダウンロードした (result_train.nnp) を置いてください。

Google Colaboratoryの場合

Google Colaboratoryの場合は、以下を実行するとファイル選択ダイアログが表示されます。時間制限があるようですが、表示した後すぐにresult_train.nnpを選択します。

```
from google.colab import files
uploaded = files.upload()
```

test.csvについて

test.csvは以下のような内容になっています。今回は検証目的に使うので、yカラムはありません。

```
x:image
001.png
002.png
003.png
004.png
```

指定されている画像は、同じフォルダ内にある画像ファイルを指定しています。

実行する

では実際に試してみましょう。上記のファイルをダウンロード、解凍したディレクトリでコマンドを以下のように実行します。

□一カルの場合

```
nnabla_cli forward -c result_train.nnp -d test.csv -o ./
```

Google Colaboratoryの場合

```
!nnabla_cli forward -c result_train.nnp -d test.csv -o ./
```

-c で学習した内容を、 -d で評価するCSVファイルを指定します。最後の -o は実行結果の出力先です。

実行すると `output_result.csv` と `progress.txt` が出力されます。大事なのは `output_result.csv` です。内容を見ると、以下のようになっています（実行環境によって異なるかも知れません）。

```
x:image,y'  
/path/to/001.png,0.00019709873595274985  
/path/to/002.png,1.0769620750750164e-08  
/path/to/003.png,0.9999932050704956  
/path/to/004.png,0.9113416075706482
```

今回は 001~002.pngまでが4を書いたもの、003~004.pngが9を書いたものになります。ただしく判定されているのが分かります。

Google Colaboratoryの場合は以下で結果を確認できます。

```
!cat output_result.csv
```

別な画像ファイルで試す

画像は[myleott/mnist_png](#)にあるものをお借りしています。他にもたくさんの4、9画像がありますので、ダウンロードして試してみてください。ファイルを追加したらtest.csvを更新するのを忘れないでください。

http://bit.ly/mnist_png

このように自分で作ったモデル、ネットワークをコンピュータ上で実行できます。コマンドを実行して、結果のCSVファイルを読むという方法であればPython以外のプログラムからでもNNCが利用できます。

作成したモデル、ネットワークをWebアプリケーションで体験する

3回目のCLIで作った環境をそのまま使っていきます。もしAnacondaやnnablaをインストールしていなかったら、[ハンズオン3回目](#)の内容を実施してください。

今回はPythonを使って、Webアプリケーションの中にどう組み込めば良いかを解説します。

Webフレームワークをインストールする

今回はPython製の簡易的なWebフレームワーク `bottle` を使います。これはコマンドプロンプトやターミナルで実行します。

```
pip install bottle
```

HTMLの解説

まずWebアプリケーションの画面を紹介します。今回はCanvasタグを使ってマウスで文字を書き、それをサーバに送る仕組みとします。JavaScriptのコードはコメントを参照してください。コードは[endoyuta/mnist_test](#)のHTMLを参考にさせてもらっています。ファイルは前回ダウンロードしたGitリポジトリのwebフォルダの中になります。

app.pyの編集

編集前は `web/app.py` は以下のようになっています。

```
def image():
    img_str = request.params['img']
    if img_str == False:
        return "{}"

    # 画像を28x28、グレースケールに変換
    b64_str = img_str.split(',')[1]
    img = Image.open(BytesIO(a2b_base64(b64_str)))
    img = img.resize((28, 28))
    img = img.convert('L')

    # Neural Network Consoleのファイルを読み込む

    # 実行するネットワークを指定する

    # 入力変数を取得

    # 出力変数を取得

    # 画像を配列に変換、数値を0～1.0にする

    # 判定する

    # 結果を出力する
```

コード中のコメントアウトされている部分を編集していきます。

Neural Network Consoleのファイルを読み込む

ダウンロードした result.nnp を読み込みます。 `web` フォルダの一つ上の階層にあるはずなので、以下のように記述します。大文字、小文字も区別されるので注意してください。

```
# Neural Network Consoleのファイルを読み込む
nnp = nnp_graph.NnpLoader('../result_train.nnp')
```

実行するネットワークを指定する

ネットワーク/モデルを読み込んだら、実行するネットワークを取得します。`MainRuntime` という名前は Neural Network ConsoleのCONFIGタブで確認できます。

```
# 実行するネットワークを指定する
graph = nnp.get_network('MainRuntime', batch_size=1)
```

入力変数を取得

入力変数（今回は`x`）を取得します。入力が複数ある場合は、複数指定しなければいけません。今回は一つ

なので、入力値の最初の値（一つしかないので）を指定します。

```
# 入力変数の名前を取得  
input = list(graph.inputs.keys())[0]  
x = graph.inputs[input]
```

出力変数の名前を取得

出力も同じように取得します。今回は一つしかないので入力と同じように取得できます。

```
# 出力変数の名前を取得  
output = list(graph.outputs.keys())[0]  
y = graph.outputs[output]
```

画像を配列に変換、数値を0～1.0にする

HTMLから受け取った画像を配列に変換する際には numpy というライブラリが使えます。ディープラーニングを行う際にはよく使うライブラリです。そして、NNCでは-1.0～1.0の値のみ扱えますので、配列を255で割ります。

```
# 画像を配列に変換、数値を0～1.0にする  
x.d = np.array(img) * (1.0 / 255.0)
```

判定する

入力変数の準備ができたら、後は判定するだけです。判定は以下のようにします。

```
# 判定する  
y.forward(clear_buffer=True)
```

結果を出力する

結果をHTMLに返します。この時、HTMLでも判別しやすいようにJSON形式で返します。結果は `y.d` の中に入っています、今回の場合は `y.d[0][0]` で取得できます。

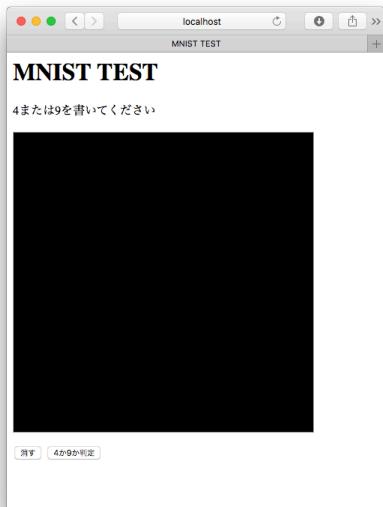
```
# 結果を出力する  
response.content_type = 'application/json'  
return '{{"result": {}}}'.format(y.d[0][0])
```

実行する

ではPythonを実行します。

```
cd web  
python app.py
```

その後、Webブラウザで <http://localhost:8080/> にアクセスします。HTML画面が表示されれば成功です。



手書きで4または9を書いて、サーバに送信します。ディープラーニングによって判定され、その結果が返ってきます。



今回のコードで、ディープラーニングのネットワークを作るコードは一切書いていないのが分かります。ダウンロードした result.nnp ファイルからディープラーニングのネットワーク情報や重みを取得することで、プログラミングはとてもシンプルになります。

より複雑なネットワークをWebアプリケーションで試す

今回はより複雑なネットワークを体験します。4か9だけでなく、0～9までの数値を判別できるようにします。

作成するネットワーク

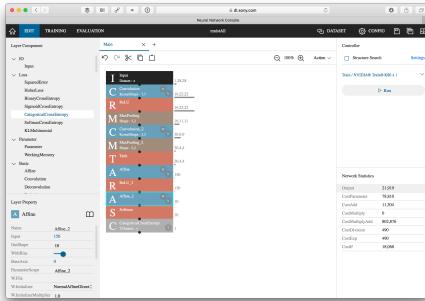
データセット

新しいプロジェクトを作成して、`mnist.mnist_training` をトレーニング、`mnist.mnist_test` をテストデータとして指定してください。

ネットワーク

作成するネットワークは以下のようになります。パラメータを変更しているものについては下に設定を記述しています。

- **Input**
- **Convolution**
KernelShape : 7,7
WithBias : True
- **ReLU**
- **MaxPooling**
IgnoreBorder : True
- **Convolution**
OutMaps : 30
KernelShape : 3,3
- **MaxPooling**
IgnoreBorder : True
- **Tanh**
- **Affine**
OutShape : 150
- **ReLU**
- **Affine**
OutShape : 10
- **Softmax**
- **CategoricalCrossEntropy**

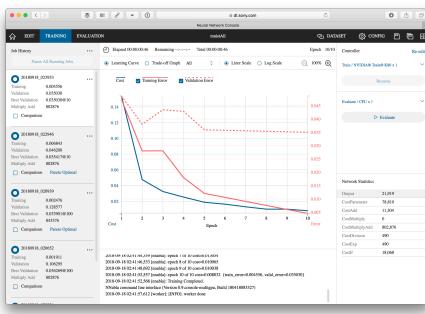


設定

CONFIGで、Max Epochを10にします。

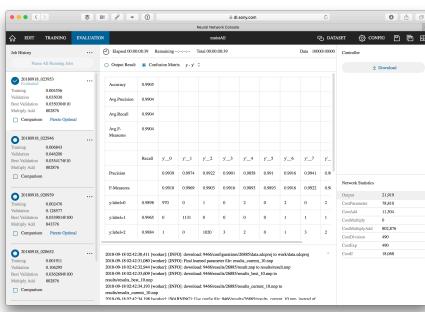
学習の実施

トレーニングを実施します。10エポックなのですぐに終わるでしょう。



検証の実施

続けて検証を実施します。98%くらいの精度になるでしょう。



ネットワークのダウンロード

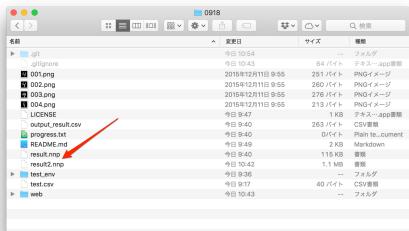
検証が終わったら、そのネットワーク情報をダウンロードします。

The screenshot shows the NNC interface with the 'EVALUATION' tab selected. It displays a list of jobs:

- 20180918_023953** (Evaluated)
 - Training: 0.004556
 - Validation: 0.035030
 - Best Validation: 0.035030@10
 - Multiply Add: 802876
 - Comparison
 - Pareto Optimal
- 20180918_022946**
 - Training: 0.006843
 - Validation: 0.046200
 - Best Validation: 0.035417@10
 - Multiply Add: 802876

A red arrow points from the text 'Download' in the context menu to the 'Download' option in the menu for the second job.

ダウンロードしたら result2.nnp としてCLIで実行したresult.nnpと同じフォルダにいれます。あらかじめ result2.nnp は存在しますが、上書きして構いません。



web/app.pyの修正

今回は 0または1ではなく、0~9それぞれの数字において可能性が返ってきます。イメージとしては書きのようになります。最初が0の可能性、次が1の可能性...となります。この中で一番1に近いものが対象の数値になります。今回であれば8です。

```
[ 3.5272753e-03 1.0275582e-07 3.2790832e-03 3.3631185e-03 3.5058970e-06
 1.4182813e-03 2.3267221e-05 1.2304156e-06 9.7950816e-01 8.8759838e-03 ]
```

e-01は浮動小数点表記になります。9.7e-01は0.97、8.8e-03は0.0088になります。通常は一番eが少ないものが対象になるでしょう。

これらは可能性なので、100で割るとパーセントになるものです。つまり最も1.0近いものが手書きされた文字になります。

読み込むネットワークの変更

先ほどダウンロード、配置したnnpファイルに変更します。

```
# Neural Network Consoleのファイルを読み込む
# 修正前
nnp = nnp_graph.NnpLoader('../result_train.nnp')

# 修正後
nnp = nnp_graph.NnpLoader('../result_train-2.nnp')
```

関数の追加

0～9の可能性について、最も1に近いものを算出する関数を追加します。

`run(host='localhost', port=8080)` の上に追加します。

```
# この関数を追加
def getNearestValue(list, num):
    # リストの差分を絶対値で算出し、差分が一番小さいインデックスを取得
    idx = np.abs(np.asarray(list) - num).argmin()
    return idx

run(host='localhost', port=8080)
```

via [Pythonのリスト要素からある値と最も近い値を取り出す - Qiita](#)

データを返す部分を修正

画像を判定した後、先ほど追加した関数を実行して最も1に近い可能性を持った数値を取得します。そして、それをHTMLに返します。

```
# HTMLに返す
number = getNearestValue(y.d[0], 1.0)      # ← 追加
return '{{"result": {}}}'.format(number)    # ← 修正
```

web/index.html を修正

前はPythonからの返却値に応じて処理を判別していましたが、それが不要になります。一番下の方にある表示処理を修正します。

```
// 修正前
document.querySelector("#answer").innerText = (json.result > 0.2 ? '9' : '4') + 'です'

// 修正後
document.querySelector("#answer").innerText = json.result + 'です';
```

これで修正は完了です。修正後のファイル（web/app.py）を以下にアップロードしておりますので、問題があったら確認してください。

[NNCデモ（0～9の数字判定）](#)

実行する

ではPythonを実行します。

```
cd web  
python app.py
```

その後、Webブラウザで <http://localhost:8080/> にアクセスします。HTML画面が表示されれば成功です。



手書きで0～9の数字を書いて判定させてみましょう。正しい判定が行われていれば、ネットワークやモデルが正しく組めているということになります。



今回のデモで分かる通り、NNCの処理自体は殆ど変更していません。nnpファイルにはネットワーク構造や重みに関する情報が入っていますので、それらを用いることで再利用性の高いコーディングが実現できます。