MATLABでディープラーニングってヘンですか？

新居浜工業高等専門学校
機械工学科
田中 大介

自己紹介

田中 大介（たなか だいすけ）

新居浜高専 機械工学科　准教授

専門　買ってきたロボットに知能を与えたり、
様々な機械に適用できるAIアルゴリズムを考えること
（知能ロボティクス、人工知能（AI）応用、機械学習、制御、システム同定など）

MATLAB歴：14年くらい

2007年頃　学生時代の制御工学の授業でSimulinkと出会う

2009年頃　卒業研究でMATLABのコードを書く

現在では　MATLABの書き方を教え、自分でもコードを書く
一応（自称）マトラバー

Robotics System Toolboxのユーザ事例

ところでみなさん、ディープラーニングといえば？

python™

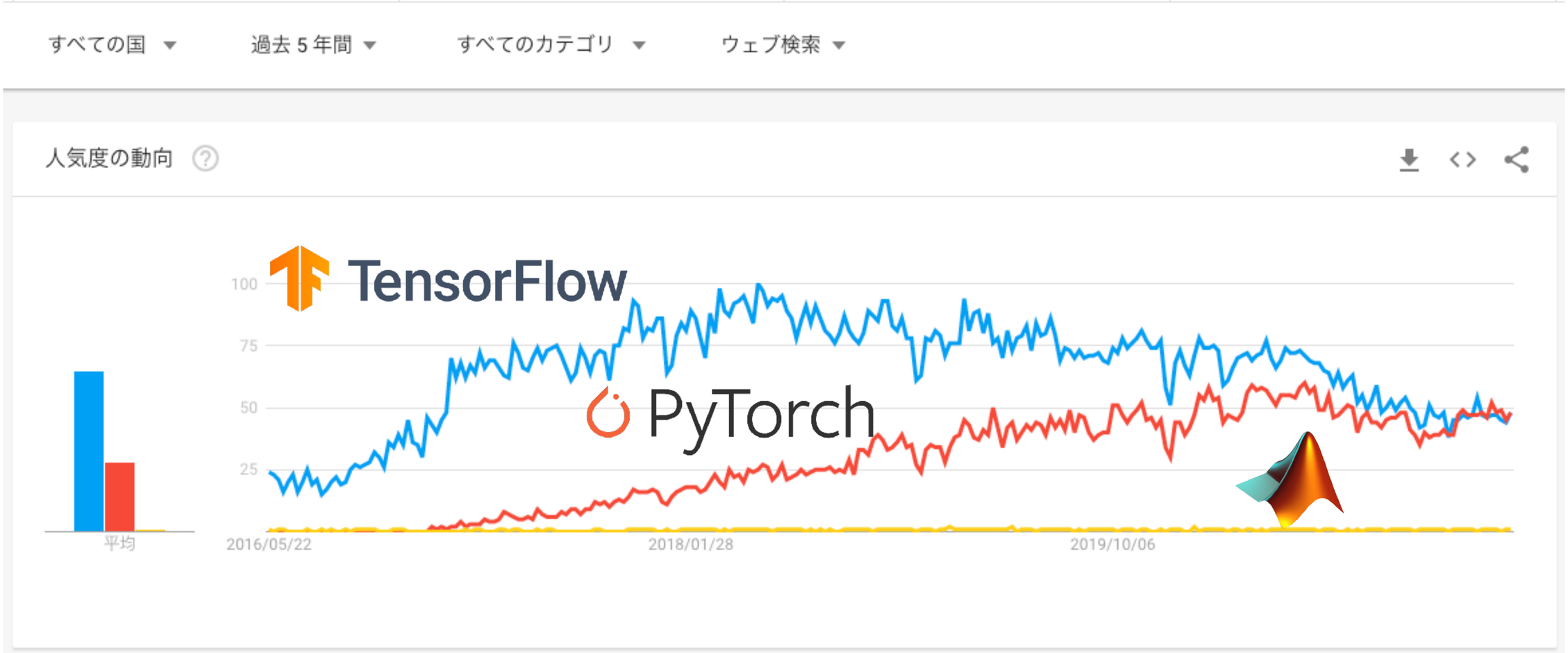
TensorFlow

PyTorch

Deep Learning Toolbox

Google Trend（全世界・過去5年間）で見てみると

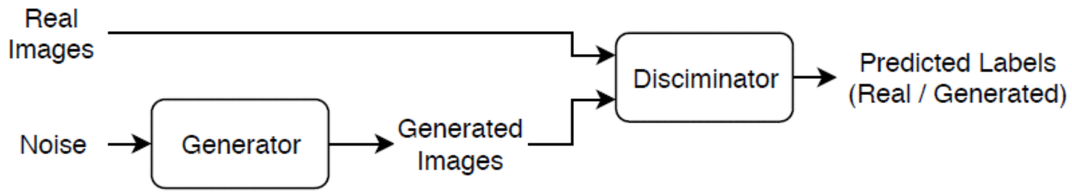
※[TensorFlow], [PyTorch], [MATLAB Deep Learning Toolbox]の結果



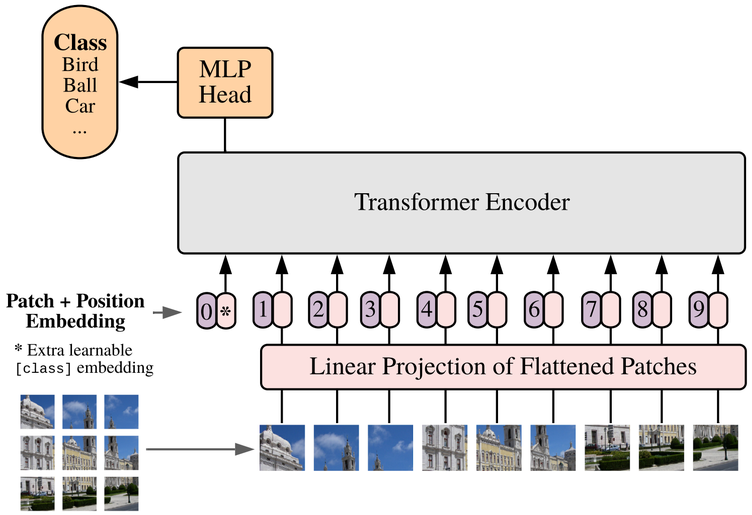
MATLover（MATLAB Lover）としては寂しい限り…

MATLABでディープラーニング、意外にここまでできた

Convolutional Neural Network (CNN)



Generative Adversarial Network (GAN)



Vision Transformer (ViT)

本日は**MATLABでもViTの実装ができた！**
ということをご紹介したいと思います

※ 時間が限られているので詳細はQiitaへの掲載を予定

[画像出典]
CNN https://jp.mathworks.com/discovery/convolutional-neural-network.html
GAN https://jp.mathworks.com/help/deeplearning/ug/train-generative-adversarial-network.html
ViT An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

（そもそも）私がMATLABを使う理由

- 理論通り（数式通り）にコードが書ける
- プログラムの共有をする敷居が低い（環境はほとんど同じ）
- 公式ドキュメントがそこそこ使える
- MATLAB Answers上ではコアなユーザが多い

それでもAIプログラミングはPythonに浮気

AIプログラミングでPythonを使う理由

- 無料で使える（Google Colab, Anaconda…）
- 最新のモデルがimportするだけで使える
- 参考書が多い（敷居が低い）
- ユーザが多い（ネットで調べれば情報が多い）

（全項目MATLABが逆の状態なのは気の所為だろうか…）

それでも私がAIプログラミングでMATLABを使う理由

- 過去の遺産（データ、前処理）が使える
- 進歩してきている（R2018a以降やっと本気を出しつつある）
  LSTMの実装, カスタム層の定義, 多入出力サポート…
- 新しいアルゴリズムを数学ベースでコーディングできる
- （個人的には）データ構造が直感的（次のスライドで説明）

KerasとMATLABでのCNNでのMNISTクラス分類コード

Build the model

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

Train the model

```
batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam",
metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs)
```

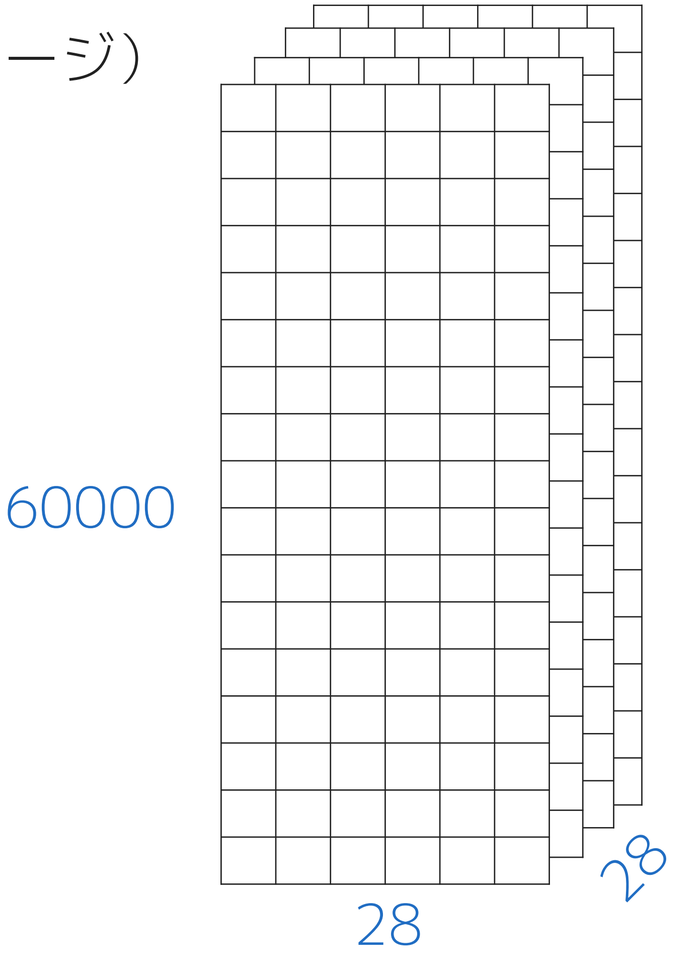
```
% Build the model
layers = [
    imageInputLayer([28 28 1])
    convolution2dLayer(3,32,'Padding','same')
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    convolution2dLayer(3,64,'Padding','same')
    reluLayer
    maxPooling2dLayer(2,'Stride',2)
    dropoutLayer(0.5)
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer];


% Train the model
options = trainingOptions('adam', ...
    'MaxEpochs',15, ...
    'ValidationData',imdsValidation, ...
    'ValidationFrequency',30, ...
    'MiniBatchSize',128, ...
    'Verbose',false, ...
    'Plots','training-progress');

net = trainNetwork(imdsTrain,layers,options);
```

コーディングの手間は実はそんなに変わらない

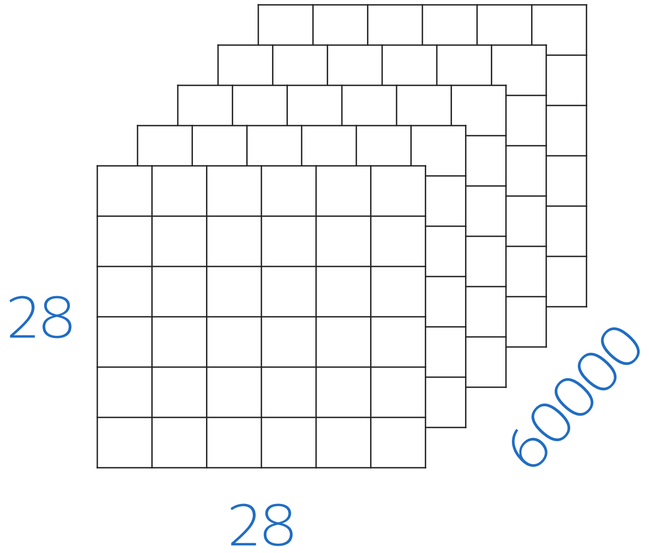
Kerasで実装するときに気に入らないこと

```
x_train shape: (60000, 28, 28, 1)
```

（データのイメージ）



28x28の画像が60000枚ならこうでは？



MATLABだったら

(28, 28, 60000) で処理を書ける

→個人的には（数学的に）直感的

MATLABでディープラーニング

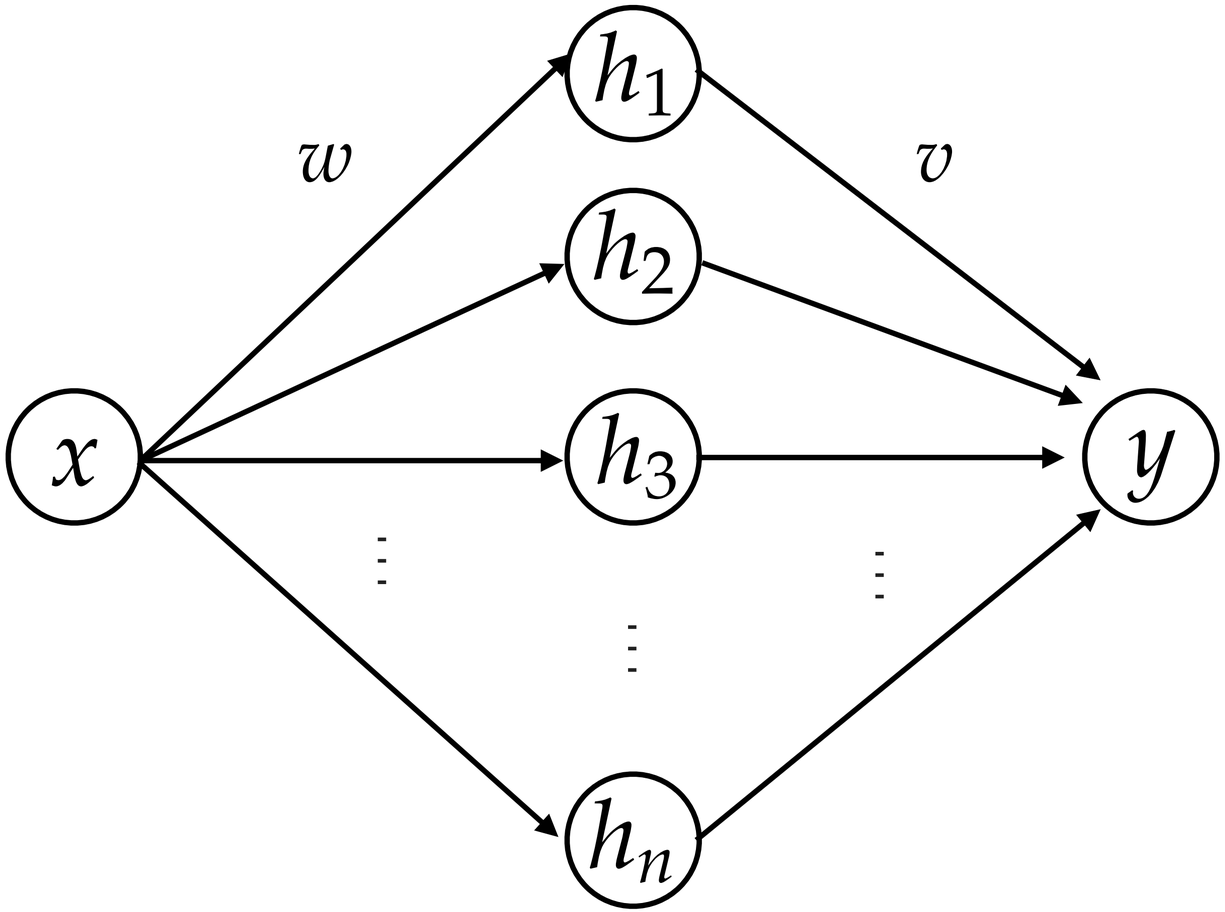
自由度低　　自由度高

① layer ＋ trainNetwork	② dlnetwork ＋ カスタム学習ループ	③ モデル関数 ＋ カスタム学習ループ
CNNを使った分類	DCGAN, VAE	VRAE, ViT
誤差評価は二乗誤差かクロスエントロピー誤差に限られる	誤差関数は自分で定義できるが対応していないLayerもある	dlarray型に対応した計算であればなんでも可能

自動微分（dlarray）に基づく実装（ R2019b 以降対応）

特にViTは R2021a で実装しやすくなった（後述）

3つの違いをMLPコードで見てみる



1個の中間層 (n=10)を持つ
浅いニューラルネットワーク
(tanh活性化)

regressionモデルを作る

$$h_i = \tanh(w_i x + b_i)$$

$$y = \sum_{i=1}^{n} v_i h_i + c$$

① layer + trainNetwork (自由度低)

```
layers = [ ...
    featureInputLayer(1, 'Name', 'in')
    fullyConnectedLayer(10, 'Name', 'fc1')
    tanhLayer('Name', 'tanh1')
    fullyConnectedLayer(1, 'Name', 'fc2')
    regressionLayer];

options = trainingOptions('sgdm', ...
    'MaxEpochs', 5000, ...
    'Verbose', false, ...
    'Plots','training-progress');

net = trainNetwork(X, T, layers, options);
```

① layer
＋
trainNetwork

簡単だが二乗誤差最小化か
クロスエントロピー最小化だけ取扱

② dlnetwork + カスタム学習ループ (自由度中)

```
layers = [ ...
    featureInputLayer(1, 'Name', 'in')
    fullyConnectedLayer(10, 'Name', 'fc1')
    tanhLayer('Name', 'tanh1')
    fullyConnectedLayer(1, 'Name', 'fc2')];


lgraph = layerGraph(layers);
dlnet = dlnetwork(lgraph)

% Loop over epochs.
for epoch = 1:numEpochs
    % Evaluate the model gradients, state, and loss using dlfeval and the
    % modelGradients function and update the network state.
    [gradients,state,loss] = dlfeval(@modelGradients,dlnet,dlX,dlY);
    dlnet.State = state;
    % Update the network parameters using the Adam optimizer.
    [dlnet,averageGrad,averageSqGrad]
            = adamupdate(dlnet,gradients,averageGrad,averageSqGrad,iteration);
end
```

② dlnetwork
＋
カスタム学習ループ

誤差関数はなんでも良いが、
dlnetwork非対応なものも
(例: sequenceInputLayer + lstm)

③ モデル関数 ＋ カスタム学習ループ（自由度高）

```
function [dlY,state]
            = model(parameters,dlX,doTraining,state)
weights = parameters.fc1.Weights;
bias = parameters.fc1.Bias;
dlY = fullyconnect(dlX,weights,bias);

dlY = tanh(dlY);

weights = parameters.fc2.Weights;
bias = parameters.fc2.Bias;
dlY = fullyconnect(dlY,weights,bias);
end

params.fc1.Weights = initializeGlorot([10, 1], 10, 1);
params.fc1.Bias = initializeZeros([10 1]);
params.fc2.Weights = initializeGlorot([1, 10], 10, 1);
params.fc2.Bias = initializeZeros([1 1]);
```

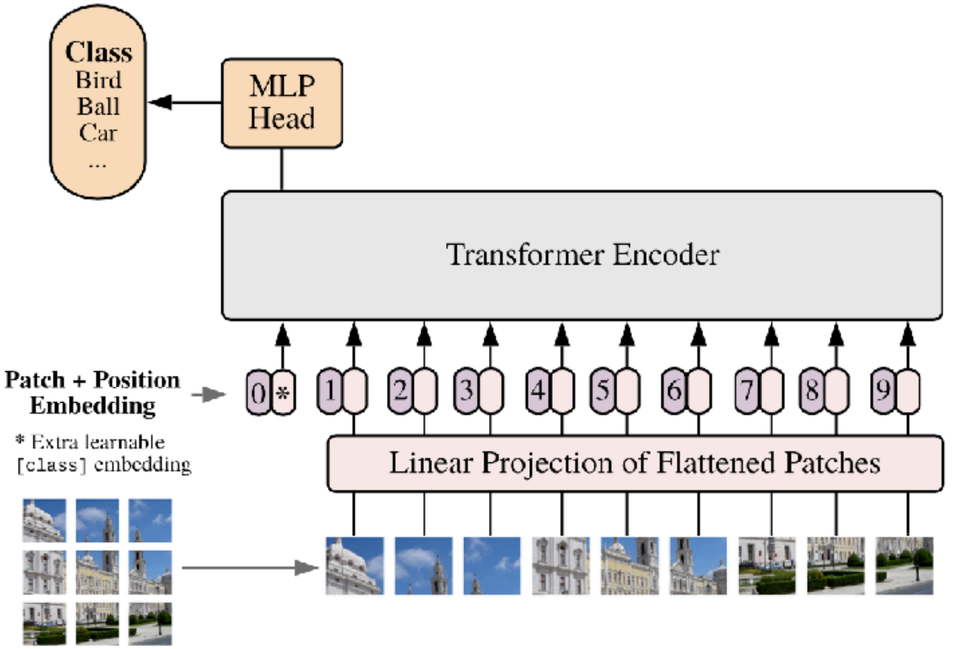
③ モデル関数
＋
カスタム学習ループ

自由度は高いが、自分で数学的に
記述していく必要がある
（メリット&デメリット）

（あとは②の学習ループ）

Vision Transformer (ViT)

- 自然言語処理分野で用いられている Transformerモデルをもとにした 画像認識モデル
- たたみこみ処理なしでSoTAモデルと 同程度もしくはより高い性能



- 実装上必要なのは、MLP, Layer Normalization（R2021aで実装）
  Multi-head Attention, GELU (Gaussian Error Linear Units)

→ **赤字**の実装がMATLABにないので③で実現

ViT原論文・図出典：An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (arXiv:2010.11929)

Multi-head Attention

- もとになるAttentionは次の実装

$$\mathbf{Attention}(Q, K, V) = \mathbf{softmax}\left(\frac{QK^{\mathrm{T}}}{\sqrt{d_k}}\right)V$$

各パッチ（シーケンス）内での重要さを表す

- Multi-head Attentionは（誤解を恐れずに言えば）
  各パッチに対して複数のAttentionを適用してよせ集める

→ 行列の計算だけできれば実装できる
(③にとっても適している)

解説：https://medium.com/the-dl/transformers-from-scratch-in-pytorch-8777e346ca51

GELU Activation

- ReLUやELUより性能が良くなると言われる活性化関数
- 厳密な実装は

$$\mathrm{GELU}(x) = \frac{x}{2}\left[1 + \mathrm{erf}\left(\frac{\sqrt{x}}{2}\right)\right]$$

(erfはdlarrayに非対応)

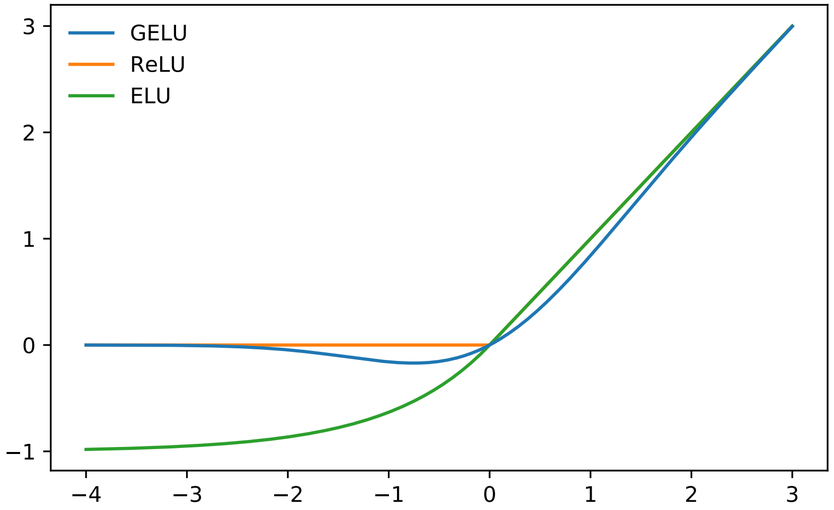


Figure 1: The GELU ($\mu = 0, \sigma = 1$), ReLU, and ELU ($\alpha = 1$).

- 原論文には初等関数による近似

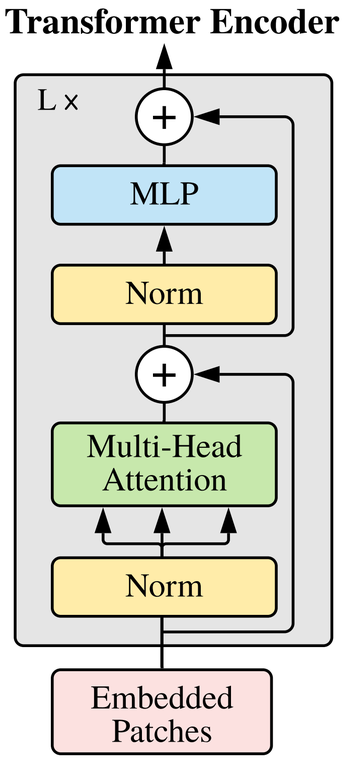
$$\mathrm{GELU}(x) \approx 0.5x\left(1 + \tanh\left[\sqrt{\frac{2}{\pi}}\left(x + 0.044715x^3\right)\right]\right)$$

(dlarrayで実装可能)

原論文・図出典：D.Hendrycks, K.Gimpel “Gaussian Error Linear Units (GELUs)” arXiv:1606.08415 (2016)

実装の概略（詳細は後日Qiitaに記事を書く予定です！）

```
function [dlY, state] = VisionTransformer(parameters, dlX, doTraining, state)
dlX = patchEncoder(dlX, clsToken, patchprojWeight, positionEmbedding);
for i = 1:numTransformerLayers
    % Layer Normalization 1
    dlX1 = LayerNormalization(dlX, 1.0e-6);
    % Attention
    dlAttentionOutput = MultiheadAttention(dlX1, dlX1, numHeads, attentiondropoutRate, …
                          qDenseWeight(:,:,i), kDenseWeight(:,:,i), vDenseWeight(:,:,i), …
                          outDenseWeight(:,:,i), outDenseBias(:,i), doTraining);
    % SkipConnection 1
    dlX2 = dlX + dlAttentionOutput;
    % MLP
    dlX3 = LayerNormalization(dlX2, 1.0e-6);
    dlX3 = pagemtimes(MLPWeight1(:,:,i), dlX3) + MLPBias1(:,i);
    dlX3 = geluApprox(dlX3);
    dlX3 = pagemtimes(MLPWeight2(:,:,i), dlX3) + MLPBias2(:,i);
   % SkipConnection 2
    dlX = dlX2 + dlX3;
end
dlX = LayerNormalization(dlX, 1.0e-6);
dlY = fullyconnect(dlX(:,1,:), classWeight, classBias, 'DataFormat', 'CSB');
dlY = softmax(dlY, 'DataFormat', 'CB');
end
```



ViT原論文・図出典：An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (arXiv:2010.11929)

Cifar10を学習（バッチサイズ以外の原論文のパラメータを遵守）

- なかなか学習が大変（現在も途中）

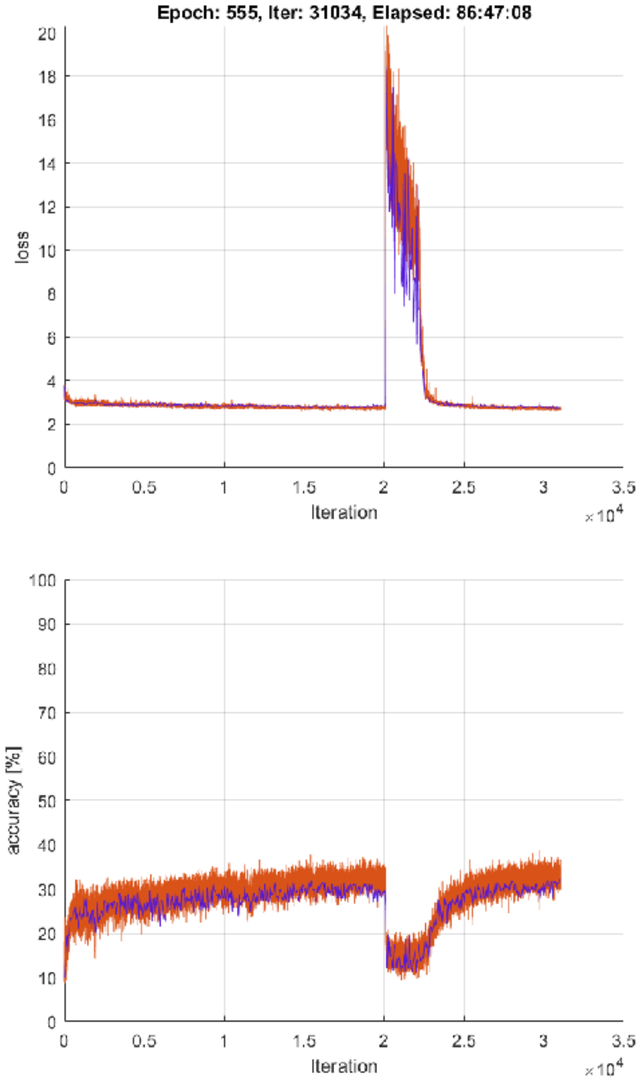
（RTX 8000 (VRAM 48G) でもバッチ数を上げるとメモリに乗らない）

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

	Ours-JFT (ViT-H/14)	Ours-JFT (ViT-L/16)	Ours-I21K (ViT-L/16)	BiT-L (ResNet152x4)	Noisy Student (EfficientNet-L2)
ImageNet	**88.55** ± 0.04	87.76 ± 0.03	85.30 ± 0.02	87.54 ± 0.02	88.4/88.5*
ImageNet ReaL	**90.72** ± 0.05	90.54 ± 0.03	88.62 ± 0.05	90.54	90.55
CIFAR-10	**99.50** ± 0.06	99.42 ± 0.03	99.15 ± 0.03	99.37 ± 0.06	–
CIFAR-100	**94.55** ± 0.04	93.90 ± 0.05	93.25 ± 0.05	93.51 ± 0.08	–
Oxford-IIIT Pets	**97.56** ± 0.03	97.32 ± 0.11	94.67 ± 0.15	96.62 ± 0.23	–
Oxford Flowers-102	99.68 ± 0.02	**99.74** ± 0.00	99.61 ± 0.02	99.63 ± 0.03	–
VTAB (19 tasks)	**77.63** ± 0.23	76.28 ± 0.46	72.72 ± 0.21	76.29 ± 1.70	–
TPUv3-core-days	2.5k	0.68k	0.23k	9.9k	12.3k

Imagenet21kだと原論文のバッチサイズにすると
ViT-Baseでもメモリ1TB以上(!)必要



ViT原論文・表出典：An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale (arXiv:2010.11929)

まとめ

- MATLABでディープラーニングをしてみました（from scratch）
- モデル関数を使ってVision Transformerを実装しました
- MATLABだからこそできる数学的に直感的な実装ができました
- 自動微分のメモリ量で死んでいます（MATLABの問題？）が、
  実装はとても楽しかったです

もう少し詳しい説明は、Qiitaに載せようと思っています。
掲載次第、発表者のHP（www.daisuket.net）等でご案内いたします！