Neural Network Training: Old & New Tricks

Old: (80’s)
   Stochastic Gradient Descent, Momentum, “weight decay”

New: (last 5-6 years)
   Dropout
   ReLUs
   Batch Normalization
Reminder: Overfitting, in images

Classification

Underfitting ≫ just right ≫ Overfitting

Regression
Each sample is processed by a ‘decimated’ neural net
Each sample is processed by a ‘decimated’ neural net

Decimated nets: distinct classifiers

But: they should all do the same job
Dropout Performance

Figure 4: Test error for different architectures with and without dropout. The networks have 2 to 4 hidden layers each with 1024 to 2048 units.
Neural Network Training: Old & New Tricks

Old: (80’s)
  Stochastic Gradient Descent, Momentum, “weight decay”

New: (last 5-6 years)
  Dropout
  ReLUs
  Batch Normalization
‘Neuron’: Cascade of Linear and Nonlinear Function

Sigmoidal (‘logistic’)

\[ g(a) = \frac{1}{1 + \exp(-a)} \]

Rectified Linear Unit (RELU)

\[ g(a) = \max(0, a) \]
Reminder: a network in backward mode

\[ z_k = \frac{1}{1 + \exp(-a_k)} \]

\[ \frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k)) \]
Reminder: a network in backward mode

$$z_k = \frac{1}{1 + \exp(-a_k)}$$

Inputs

$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \begin{bmatrix} \frac{\partial l}{\partial z_k} g'(a_k) \end{bmatrix} = \begin{bmatrix} \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k)) \end{bmatrix}$$

Gradient signal from above

Outputs

$$y = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$
Reminder: a network in backward mode

Gradient signal from above

\[ \frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k)) \]

\[ z_k = \frac{1}{1 + \exp(-a_k)} \]
Reminder: a network in backward mode

\[ z_k = \frac{1}{1 + \exp(-a_k)} \]

Gradient signal from above

\[
\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k))
\]
Outputs

Gradient signal from above

scaling: <1 (actually <0.25)

$$z_k = \frac{1}{1 + \exp(-a_k)}$$

$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k))$$

$$\begin{bmatrix}
0 \\
1 \\
0
\end{bmatrix}$$
Vanishing Gradients Problem

Gradient signal from above

\[ \frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial z_k} g(a_k)(1 - g(a_k)) \]

scaling: <1 (actually <0.25)
Vanishing Gradients Problem

Gradient signal from above

\[ \frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k) = \frac{\partial l}{\partial a_k} g(a_k)(1 - g(a_k)) \]

scaling: <1 (actually <0.25)

Do this 10 times: updates in the first layers get minimal
Top layer knows what to do, lower layers “don’t get it”

Sigmoidal Unit: Signal is not getting through!
Vanishing Gradients Problem: ReLU Solves It

Gradient signal from above

$$\frac{\partial l}{\partial a_k} = \sum_m \frac{\partial l}{\partial z_m} \frac{\partial z_m}{\partial a_k} = \frac{\partial l}{\partial z_k} g'(a_k)$$

$$g(a) = \max(0, a)$$

$$g'(a) = \begin{cases} 
1 & a > 0 \\
0 & a < 0 
\end{cases}$$
Activation Functions: ReLU & Co

\[ g(a) = \max(0, a) \]

\[ g'(a) = \begin{cases} 
1 & a > 0 \\
0 & a < 0 
\end{cases} \]

Great! But... no gradient for negative half-space
Activation Functions: ReLU & Co

\[ g(a) = \max(0, a) \]

\[ g'(a) = \begin{cases} 
1 & a > 0 \\
0 & a < 0 
\end{cases} \]

Great! But... no gradient for negative half-space

Lots of follow up work: LeakyReLU, eLU, etc.

Can improve results, but typically fine-tuning only
Neural Network Training: Old & New Tricks

Old: (80’s)
  Stochastic Gradient Descent, Momentum, “weight decay”

New: (last 5-6 years)
  Dropout
  ReLUs
  Batch Normalization
External Covariate Shift: your input changes

10 am  

2pm  

7pm
“Whitening”: Set Mean = 0, Variance = 1

Photometric transformation: $I \rightarrow aI + b$

Original Patch and Intensity Values

Brightness Decreased

Contrast increased,
“Whitening”: Set Mean = 0, Variance = 1

• Make each patch have zero mean:
  Photometric transformation: \( I \rightarrow aI + b \)
“Whitening”: Set Mean = 0, Variance = 1

• Make each patch have zero mean:
  Photometric transformation: \( I \rightarrow a I + b \)

\[
\mu = \frac{1}{N} \sum_{x,y} I(x, y)
\]

\[
Z(x, y) = I(x, y) - \mu
\]
“Whitening”: Set Mean = 0, Variance = 1

• Make each patch have zero mean:
  Photometric transformation: $I \rightarrow aI + b$

• The

- Original Patch and Intensity Values
- Brightness Decreased
- Contrast increased,

\[
\mu = \frac{1}{N} \sum_{x,y} I(x, y)
\]

\[
Z(x, y) = I(x, y) - \mu
\]
“Whitening”: Set Mean = 0, Variance = 1

- Make each patch have zero mean:
  Photometric transformation: \( I \rightarrow aI + b \)

- The
  - Original Patch and Intensity Values
  - Brightness Decreased
  - Contrast increased,

\[
\begin{align*}
\mu &= \frac{1}{N} \sum_{x,y} I(x, y) \\
Z(x, y) &= I(x, y) - \mu \\
\sigma^2 &= \frac{1}{N} \sum_{x,y} Z(x, y)^2 \\
ZN(x, y) &= \frac{Z(x, y)}{\sigma}
\end{align*}
\]
Batch Normalization

Whiten-as-you-go:

- Normalize the activations in each layer within a mini-batch.
- Learn the mean and variance ($\gamma, \beta$) of each layer as parameters.
Batch Normalization: Used in all current systems

- Multi-layer CNN’s train faster with fewer data samples (15x).
- Employ faster learning rates and less network regularizations.
- Achieves state of the art results on ImageNet.
Convolutional Neural Networks
Example: 200x200 image
40K hidden units
~1.6B parameters!!!

- Spatial correlation is local
- Waste of resources
- We don’t have enough training samples anyway…
Locally-connected Layer

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).
Locally-connected Layer

Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).
Convolutional Layer

Share the same parameters across different locations (assuming input is stationary):
Convolutions with learned kernels
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
Convolutional Layer
**Fully-connected layer**

\[
\begin{bmatrix}
  y_1 \\
  y_2 \\
  y_3 \\
  y_4 \\
  \vdots \\
  y_K 
\end{bmatrix}
= 
\begin{bmatrix}
  w_{1,1} & w_{1,2} & w_{1,3} & w_{1,4} & \ldots & w_{1,K} \\
  w_{2,1} & w_{2,2} & w_{2,3} & w_{2,4} & \ldots & w_{2,K} \\
  w_{3,1} & w_{3,2} & w_{3,3} & w_{3,4} & \ldots & w_{3,K} \\
  w_{4,1} & w_{4,2} & w_{4,3} & w_{4,4} & \ldots & w_{4,K} \\
  \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
  w_{K,1} & w_{K,2} & w_{K,3} & w_{K,4} & \ldots & w_{K,K}
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  \vdots \\
  x_K
\end{bmatrix}
\]
Convolutional layer

# of parameters: size of window

\[ \begin{bmatrix}
  y_1 \\
  y_2 \\
  y_3 \\
  y_4 \\
  \vdots \\
  y_K
\end{bmatrix}
= 
\begin{bmatrix}
  w_0 & w_1 & w_2 & 0 & \cdots & 0 \\
  0 & w_0 & w_1 & w_2 & \cdots & 0 \\
  0 & 0 & w_0 & w_1 & \cdots & 0 \\
  0 & 0 & 0 & w_0 & \cdots & 0 \\
  \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\
  0 & 0 & 0 & 0 & \cdots & w_0
\end{bmatrix}
\begin{bmatrix}
  x_1 \\
  x_2 \\
  x_3 \\
  x_4 \\
  \vdots \\
  x_K
\end{bmatrix} \]
Convolutional layer
Learning an edge filter
Learn multiple filters.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters
Convolutional layer with ReLU activation

\[
h^n_i = \max \left\{ 0, \sum_{j=1}^{\text{#input channels}} h^{n-1}_j \ast w^n_{ij} \right\}
\]
Convolutional layer with ReLU activation

\[ h_i^n = \max \left\{ 0, \sum_{j=1}^{\text{#input channels}} h_j^{n-1} * w_{ij}^n \right\} \]

- **output feature map**
- **ReLU**
- **input feature map**
- **kernel**

**Diagram:**
- \( h_1^{n-1} \)
- \( h_2^{n-1} \)
- \( h_3^{n-1} \)
- Conv. layer
- \( h_1^n \)
- \( h_2^n \)
Convolutional layer with ReLU activation

\[
    h_i^n = \max \left\{ \begin{array}{c}
    0, \\
    \sum_{j=1}^{\text{#input channels}} h_{ij}^{n-1} \ast w_{ij}^n
    \end{array} \right\}
\]
Convolutional layer with ReLU activation

\[ h^n_i = \max \left\{ 0, \sum_{j=1}^{\text{#input channels}} h^{n-1}_j \ast w^n_{ij} \right\} \]

output feature map

input feature map

kernel

Convolutional layer with ReLU activation

output feature map

input feature map

kernel
De-convolutional layer with ReLU activation

\[
h_i^n = \max \left\{ 0, \sum_{j=1}^{\text{#input channels}} h_j^{n-1} \ast w_{ij}^n \right\}
\]

Still holds, same structure
De-convolutional layer with ReLU activation

\[
h_i^n = \max\left\{ 0, \sum_{j=1}^{\text{#input channels}} h_j^{n-1} \ast w_{ij}^n \right\}
\]

Still holds, same structure

No real inverse - but convolutions can easily go the other way
De-convolutional layer with ReLU activation

\[
h_i^n = \max \left\{ 0, \sum_{j=1}^{\text{#input channels}} h_j^{n-1} \ast w_{ij}^n \right\}
\]

Still holds, same structure

No real inverse - but convolutions can easily go the other way

“De-convolution” or “Transposed convolution”
De-convolutional layer with ReLU activation

\[ h^n_i = \max \left\{ 0, \sum_{j=1}^{\text{#input channels}} h^{n-1}_j * w^n_{ij} \right\} \]

Still holds, same structure

No real inverse - but convolutions can easily go the other way

“De-convolution” or “Transposed convolution”

Also a convolution with transposed weight tensor
Pooling layer

Let us assume filter is an “eye” detector.

**Q.** how can we make the detection robust to the exact location of the eye?
Pooling layer

By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.
Pooling layer: receptive field size

If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: 

\((P+K-1) \times (P+K-1)\)
Pooling layer: receptive field size

If convolutional filters have size KxK and stride 1, and pooling layer has pools of size PxP, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: $(P+K-1) \times (P+K-1)$
Receptive field
Receptive field: layer 1
Receptive field: layer 2
Receptive field: layer 3
Receptive field: layer 4
Receptive field: layer 5
Receptive field: layer 6
Receptive field: layer 7
Receptive field: layer 8
Modern Architectures
CNNs, late 1980’s: LeNet

CNNs, late 1980’s: LeNet


https://www.youtube.com/watch?v=FwFduRA_L6Q
CNNs, late 1980’s: LeNet

https://www.youtube.com/watch?v=FwFduRA_L6Q

What happened in between?
What happened in between?

deep learning = neural networks (+ big data + GPUs)
deep learning = neural networks (+ big data + GPUs) + a few more recent tricks!
CNNs, 2012

**AlexNet**
CNNs, 2014: VGG

VGG
Karen Simonyan, Andrew Zisserman (=Visual Geometry Group)
Very Deep Convolutional Networks for Large-Scale Image Recognition, arxiv, 2014.
CNNs, 2015: ResNet

ResNet
Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun,
Going Deeper - The Deeper, the Better

- Deeper networks can cover more complex problems
  - Increasingly large receptive field size & rich patterns

ImageNet Classification top-5 error (%)
Going Deeper

• From 20 to 100/1000
  • Residual networks
Residual Network

Naïve solution

• If extra layers are an identity mapping, the training errors do not increase
Residual Modelling: Basic idea in image processing

• Goal: estimate update between an original image and a changed image

Preserving base information

Some Network \rightarrow \text{residual} \rightarrow \text{can treat perturbation}
Residual Network

• Plain block
  • Difficult to make identity mapping because of multiple non-linear layers
Residual Network

- Residual block
  - If identity were optimal, easy to set weights as 0
  - If optimal mapping is closer to identity, easier to find small fluctuations

Appropriate for treating perturbation as keeping a base information
Residual Network: Deeper is better

- Deeper ResNets have lower training error
Residual Network: Deeper is better
CNNs, 2017: DenseNet

Densely Connected Convolutional Networks, CVPR 2017
Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger

Recently proposed, better performance/parameter ratio
Image-to-Image
Graphics: Multiresolution
Image-to-image

• So far we mapped an image image to a number or label
• In graphics, output often is “richer”:
  • An image
  • A volume
  • A 3D mesh
  • ...
• Note: “image” just placeholder name here for any Eulerian data
• Architectures
  • Fully convolutional
  • Encoder-Decoder
  • Skip connections
Fully-convolutional Neural Networks
Fully-convolutional Neural Networks
Fully-convolutional Neural Networks
Fully-convolutional Neural Networks
Fully-convolutional Neural Networks

Flexible - works with varying input sizes
Fully Convolutional Neural Networks in Practice

Flexible - works with varying input sizes
Typically reduces input by fixed factor

32-fold decimation
224x224 to 7x7

FCNN
Encoder-Decoder
Interpretation

• Encoder: turns data set (e.g. image) into vector
• This vector is a very compact and abstract “code”
• Lives in the “latent space” of the neural network
• Decoder: turns code back into image
Encoder-decoder + Skip connections

- 1\textsuperscript{st}: \textbf{Reduce} resolutions as before
- 2\textsuperscript{nd}: \textbf{Increase} resolution
- Transposed convolutions
- \textbf{Preserves} information
- But cannot be split into en- and decoder anymore
Thank you!

http://geometry.cs.ucl.ac.uk/creativeai/
Recurrent Neural Networks
Recurrent Neural Networks

- Time dependent problems: repeated evaluations with internal “state”
- State $x_t$ at time $t$, depends on previous times
- Recurrent Neural Networks (RNNs)
- Specialized back-prop possible: Back-propagation through time (BPTT)
- Unrolled:
Common Building Block: LSTM Units

- *Long short term memory (LSTM)* networks
- Three internal states: input, output, forget
Common Building Block: LSTM Units

- Long short term memory (LSTM) networks
- Three internal states: input, output, forget
Common Building Block: LSTM Units

- Long short term memory (LSTM) networks
- Three internal states: input, output, forget
Common Building Block: LSTM Units

- *Long short term memory (LSTM)* networks
- Three internal states: input, output, forget
Common Building Block: LSTM Units

- *Long short term memory (LSTM)* networks
- Three internal states: input, output, forget

![Diagram of LSTM Unit](image)
Common Building Block: LSTM Units

- **Long short term memory (LSTM) networks**
- Three internal states: input, output, forget
Common Building Block: LSTM Units

- Long short term memory (LSTM) networks
- Three internal states: input, output, forget
Common Building Block: LSTM Units

- **Long short term memory (LSTM) networks**
- In equation form:

\[
\begin{align*}
    i_t &= \sigma \left( W_{xi} x_t + W_{hi} h_{t-1} + W_{ci} c_{t-1} + b_i \right) \\
    f_t &= \sigma \left( W_{xf} x_t + W_{hf} h_{t-1} + W_{cf} c_{t-1} + b_f \right) \\
    c_t &= f_t c_{t-1} + i_t \tanh \left( W_{xc} x_t + W_{hc} h_{t-1} + b_c \right) \\
    o_t &= \sigma \left( W_{xo} x_t + W_{ho} h_{t-1} + W_{co} c_t + b_o \right) \\
    h_t &= o_t \tanh (c_t)
\end{align*}
\]

[Sutskever et al., “Sequence to Sequence Learning with Neural Networks”, 2014]
Recurrent Neural Networks

- LSTM networks powerful tool for sequences over time
- Alternatives:
  - Gated Recurrent Units (GRUs)
  - Time convolutional networks (TCNs)
  - ...

[Chung et al., "Empirical evaluation of gated recurrent neural networks on sequence modeling", 2014]
[Bai et al., "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling", 2018]
Deep Learning Frameworks
Main frameworks

- TensorFlow™: (Python, C++, Java)
- Keras: (Python, backends support other languages)
- PyTorch: (Python)
- Caffe: (C++, Python, Matlab)

Currently less frequently used

- Chainer: (Python)
- theano: (Python)
- Caffe2: (Python, C++)
- CNTK: (Python, C++, C#)
- MATLAB: (Matlab)
- DL4J: (Python, Java, Scala)
- mxnet: (Python, C++, and others)
Popularity

Google Trends for search terms: “[name] github”

Google Trends for search terms: “[name] tutorial”
Typical Training Steps

```
for i = 1 .. max_iterations

    input, ground_truth = load_minibatch(data, i)

    output = network_evaluate(input, parameters)

    loss = compute_loss(output, ground_truth)

    # gradients of loss with respect to parameters
    gradients = network_backpropagate(loss, parameters)

    parameters = optimizer_step(parameters, gradients)
```
Tensors

• Frameworks typically represent data as tensors
• Examples:

4D input data: \( B \times C \times H \times W \)

4D convolution kernel: \( OC \times IC \times KH \times KW \)

- \( B \): batches
- \( C \): feature channels
- \( H \): spatial height
- \( W \): spatial width
- \( IC \): input channels
- \( IC \): input channels
- \( IC \): input channels
- \( KH \): kernel height
- \( KW \): kernel width
- \( OC \): output channels
What Does a Deep Learning Framework Do?

• Tensor math
• Common network operations/layers
• Gradients of common operations
• Backpropagation
• Optimizers
• GPU implementations of the above
• usually: data loading, network parameter saving/loading
• sometimes: distributed computing
Automatic Differentiation & the Computation Graph

parameters = (weight, bias)
output = \sigma(weight \times input + bias)
loss = (output - ground_truth)^2

# gradients of loss with respect to parameters
gradients = backpropagate(loss, parameters)

Since loss is a scalar, the gradients are the same size as the parameters
Automatic Differentiation & the Computation Graph

outputs = forward(inputs, )

,  = backward()
Static vs Dynamic Computation Graphs

- Static analysis allows optimizations and distributing workload
- Dynamic graphs make data-driven control flow easier
- In static graphs, the graph is usually defined in a separate ‘language’
- Static graphs have less support for debugging

**Static**

```
x = Variable()
loss = if_node(x < parameter[0],
               x + parameter[0],
               x - parameter[1])
```

```
for i = 1 .. max_iterations
  x = data()
  run(loss)
  backpropagate(loss, parameters)
```

**Dynamic**

```
define implicitly by running operations,
   a new graph is created in each evaluation
```

```
for i = 1 .. max_iterations
  x = data()
  if x < parameter[0]
    loss = x + parameter[0]
  else
    loss = x – parameter[1]
  backpropagate(loss, parameters)
```
Tensorflow

• Currently the largest community
• Static graphs (dynamic graphs are in development: Eager Execution)
• Good support for deployment
• Good support for distributed computing
• Typically slower than the other three main frameworks on a single GPU
PyTorch

- Fast growing community
- Dynamic graphs
- Distributed computing is in development (some support is already available)
- Intuitive code, easy to debug and good for experimenting with less traditional architectures due to dynamic graphs
- Very Fast
Keras

- A high-level interface for various backends (Tensorflow, CNTK, Theano)
- Intuitive high-level code
- Focus on optimizing time from idea to code
- Static graphs
Caffe

• Created earlier than Tensorflow, PyTorch or Keras
• Less flexible and less general than the other three frameworks
• Static graphs
• Legacy - to be replaced by Caffe2: focus is on performance and deployment
  • Facebook’s platform for Detectron (Mask-RCNN, DensePose, ...)

SIGGRAPH ASIA 2018 TOKYO
# Converting Between Frameworks

- **Example:** develop in one framework, deploy in another
- **Currently:** a large range of converters, but no clear standard
- **Standardized model formats are in development**

## Converters

<table>
<thead>
<tr>
<th>convertor</th>
<th>tensorflow</th>
<th>pytorch</th>
<th>keras</th>
<th>caffe</th>
<th>caffe2</th>
<th>CNTK</th>
<th>chainer</th>
<th>mxnet</th>
</tr>
</thead>
<tbody>
<tr>
<td>tensorflow</td>
<td>-</td>
<td>pytorch-tf/MMdnn</td>
<td>model-converters/nn_toolsconvert-to-tensorflow/MMdnn</td>
<td>MMdnn/nn_tools</td>
<td>None</td>
<td>crosstalk/MMdnn</td>
<td>None</td>
<td>MMdnn</td>
</tr>
<tr>
<td>pytorch</td>
<td>pytorch2keras (over Keras)</td>
<td>-</td>
<td>Pytorch2keras/nn-transfer</td>
<td>Pytorch2caffe/pytorch-caffe-darknet-convert</td>
<td>onnx-caffe2</td>
<td>ONNX</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>keras</td>
<td>nn_tools/convert-to-tensorflow/keras_to_tensorflow/keras_to_tensorflow/MMdnn</td>
<td>MMdnn/nn-transfer</td>
<td>-</td>
<td>MMdnn/nn-transfer</td>
<td>None</td>
<td>MMdnn</td>
<td>None</td>
<td>MMdnn</td>
</tr>
<tr>
<td>caffe</td>
<td>MMdnn/nn_tools/caffe-tensorflow</td>
<td>MMdnn/pytorch-caffe-darknet-convet/pytorch-resnet</td>
<td>caffe_weight_converter/caffe2keras/nn_tools/kerascaffe2keras/Deep Learning Model_Converter/MMdnn</td>
<td>-</td>
<td>CaffeToCaffe2</td>
<td>crosstalkcaffe/CaffeConverter/MMdnn</td>
<td>None</td>
<td>MMdnn</td>
</tr>
<tr>
<td>caffe2</td>
<td>None</td>
<td>ONNX</td>
<td>None</td>
<td>None</td>
<td>-</td>
<td>ONNX</td>
<td>None</td>
<td>None</td>
</tr>
<tr>
<td>CNTK</td>
<td>None</td>
<td>ONNX</td>
<td>mmdnn</td>
<td>MMdnn</td>
<td>ONNX</td>
<td>-</td>
<td>None</td>
<td>MMdnn</td>
</tr>
<tr>
<td>chainer</td>
<td>None</td>
<td>chainer2pytorch</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>None</td>
<td>-</td>
<td>None</td>
</tr>
<tr>
<td>mxnet</td>
<td>MMdnn</td>
<td>MMdnn</td>
<td>MMdnn</td>
<td>MMdnn/MMNet2Caffe/MMNet2Caffe</td>
<td>None</td>
<td>MMdnn</td>
<td>None</td>
<td>MMdnn</td>
</tr>
</tbody>
</table>

*From https://github.com/ysh329/deep-learning-model-convertor*
ONNX

- Standard format for models
- Native support in development for Pytorch, Caffe2, Chainer, CNTK, and MxNet
- Converter in development for Tensorflow

MMDnn

- Converters available for several frameworks
- Common intermediate representation, but no clear standard
Thank you!

http://geometry.cs.ucl.ac.uk/creativeai/