Deep Learning Made Easier
by Linear Transformations in Perceptrons
Tapani Raiko, Harri Valpola, Yann LeCun
Aalto University, New York University
AISTATS 2012
• Learning deep networks (many hidden layers) used to be difficult
• Layerwise pretraining by RBMs or denoising autoencoders helps
• Could similar performance be achieved with back-propagation?
Proposed method

- **Standard MLP** (only shallow shown)
- Include shortcut connections $C$

$$y_t = Af \left( Bx_t \right) + Cx_t + \epsilon_t$$

- Add linear transformations to nonlinearities

$$f_i(b_i x_t) = \tanh(b_i x_t) + \alpha_i b_i x_t + \beta_i$$

- Alphas and betas are not learned, but set to make learning the weights $A,B,C$ easier
\[ y_t = \text{Af} \left( Bx_t \right) + Cx_t + \epsilon_t \]

- **Separate** the **nonlinear** and **linear** problems by disabling linear dependencies from \( f \)

\[
\sum_{t=1}^{T} f_i(b_i x_t) = 0 \quad \sum_{t=1}^{T} f'_i(b_i x_t) = 0
\]

by setting

\[
\alpha_i = -\frac{1}{T} \sum_{t=1}^{T} \tanh'(b_i x_t) \quad \beta_i = -\frac{1}{T} \sum_{t=1}^{T} [\tanh(b_i x_t) + \alpha_i b_i x_t]
\]

- **Compensate** by changing \( C \) accordingly

\[
C_{\text{new}} = C_{\text{old}} - A(\alpha_{\text{new}} - \alpha_{\text{old}})B - A(\beta_{\text{new}} - \beta_{\text{old}}) [0 \ 0 \ldots 1]
\]
Theoretical Motivation

- Fisher information matrix becomes more diagonal
- Standard gradient becomes closer to natural gradient

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
</table>
| **A** | \( \begin{cases} 
0 \\
-\frac{1}{\sigma_i^2} \sum_t f_j(b_jx_t)f_{j'}(b_{j'}x_t) 
\end{cases} \) | \( -\frac{1}{\sigma_i^2} a_{ij'} \sum_t f_j(b_jx_t)f_{j'}(b_{j'}x_t)x_{kt} \) | \( \begin{cases} 
0 \\
-\frac{1}{\sigma_i^2} \sum_t f_j(b_jx_t)x_{kt} 
\end{cases} \) |

| **B** | \( -\frac{1}{\sigma_i^2} a_{ij'} \sum_t f_j(b_jx_t)f_{j'}(b_{j'}x_t)x_{kt} \) | \( -\frac{1}{\sigma_i^2} a_{ij'} \sum_t f_j'(b_jx_t)f_{j'}(b_{j'}x_t)x_{kt}x_{k't} \) | \( -\frac{1}{\sigma_i^2} a_{ij} \sum_t f_j'(b_jx_t)x_{kt}x_{k't} \) |

| **C** | \( \begin{cases} 
0 \\
-\frac{1}{\sigma_i^2} \sum_t f_j(b_jx_t)x_{kt} 
\end{cases} \) | \( -\frac{1}{\sigma_i^2} a_{ij} \sum_t f_j'(b_jx_t)x_{kt}x_{k't} \) | \( \begin{cases} 
0 \\
-\frac{1}{\sigma_i^2} \sum_t x_{kt}x_{k't} 
\end{cases} \) |
Implementation Details

- Learning algorithm: Stochastic gradient
- Mini-batch size 1000, momentum 0.9
- Transformations done initially and after every 1000 iterations
- Soft-max for discrete outputs
- Normalized random initialization, shortcut weights to zero
- Learning rate decreased linearly in the second half of learning time
- Regularization: PCA in classification, weight decay, added noise to inputs
Experiments

- MNIST Classification
- CIFAR-10 Classification
- MNIST Autoencoder

- Image data, but nothing image-specific
MNIST Classification

- Data
- PCA
- Noise

Input layer 1
- Hidden layer 1
- Hidden layer 2
- Output class
MNIST Classification

Error against learning rate
Training (lower) and test errors (higher)

Error against learning time
MNIST Classification

- Test errors after 15 minutes as regularization methods are included:

<table>
<thead>
<tr>
<th>regularization</th>
<th>none</th>
<th>weight decay</th>
<th>PCA</th>
<th>noise</th>
<th>(150 minutes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>original</td>
<td>1.87</td>
<td>1.85</td>
<td>1.62</td>
<td>1.15</td>
<td>1.03</td>
</tr>
<tr>
<td>shortcuts</td>
<td>2.02</td>
<td>1.77</td>
<td>1.59</td>
<td>1.23</td>
<td>1.17</td>
</tr>
<tr>
<td>transform.</td>
<td>1.63</td>
<td>1.56</td>
<td>1.56</td>
<td>1.10</td>
<td><strong>1.02</strong></td>
</tr>
</tbody>
</table>

Histograms of $\alpha_i$ and $\beta_i$ in the first hidden layer. Examples of $f_i(\cdot)$. 
MNIST Classification

- Visualization of learned weights to randomly chosen hidden units on layers 1 and 2, and to the class outputs 0, 1, ..., 9
CIFAR-10 Classification

- 500-500-500-10 network

- original data
- after PCA to 500
- with noise
- with noise
CIFAR-10 Classification

Classification error against learning time
# CIFAR-10 Classification

<table>
<thead>
<tr>
<th>Classification %</th>
<th>linear</th>
<th>original</th>
<th>shortcuts</th>
<th>transf.</th>
<th>Krizhevsky (2009)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Training error</td>
<td>58.07</td>
<td>23.21</td>
<td>22.46</td>
<td>4.56</td>
<td>48.47</td>
</tr>
<tr>
<td>Test error</td>
<td>59.09</td>
<td>44.42</td>
<td>44.99</td>
<td>43.70</td>
<td></td>
</tr>
</tbody>
</table>
Deep Learning Made Easier by Linear Transformations in Perceptrons

Tapani Raiko1, Harri Valpola1, Yann LeCun2

1 Aalto ... unsupervised pre-
training for further improvement

• How about doing classification and autoencoder as a multitask?
MNIST Autoencoder

Reconstruction error against learning time
## MNIST Autoencoder

<table>
<thead>
<tr>
<th></th>
<th>linear</th>
<th>original</th>
<th>shortcuts</th>
<th>transf.</th>
<th>Martens (2010)</th>
</tr>
</thead>
<tbody>
<tr>
<td>training error</td>
<td>8.11</td>
<td>2.37</td>
<td>2.11</td>
<td>1.94</td>
<td>1.75</td>
</tr>
<tr>
<td>test error</td>
<td>7.85</td>
<td>2.76</td>
<td>2.61</td>
<td><strong>2.44</strong></td>
<td>2.55</td>
</tr>
<tr>
<td># of iterations</td>
<td>92k</td>
<td>49k</td>
<td>38k</td>
<td>37k</td>
<td>?</td>
</tr>
</tbody>
</table>

- **x-h1**
- **x-h2**
- **x-h3**
- **h5-y**
- **h4-y**
- **h3-y**
Discussion

• Simple transformations make basic gradient competitive with state-of-the-art
• Making parameters more independent will also help variational Bayes and MCMC
• Could be initialized with unsupervised pre-training for further improvement
• How about doing classification and autoencoder as a multitask?