Tips for Training Deep Network
Output

• Training Strategy: Batch Normalization
• Activation Function: SELU
• Network Structure: Highway Network
Batch Normalization
Feature Scaling

For each dimension $i$:

mean: $m_i$

standard deviation: $\sigma_i$

The means of all dimensions are 0, and the variances are all 1

In general, gradient descent converges much faster with feature scaling than without it.

\[ x_i^r \leftarrow \frac{x_i^r - m_i}{\sigma_i} \]
How about Hidden Layer?

- Feature Scaling
- Layer 1: $\alpha^1$
- Layer 2: $\alpha^2$
- Layer 3: $\cdots$

Feature Scaling

Difficulty: their statistics change during the training ...

Batch normalization

Smaller learning rate can be helpful, but the training would be slower.

Internal Covariate Shift
Batch

\[
x^1 \rightarrow W^1 \rightarrow z^1 \rightarrow \text{Sigmoid} \rightarrow a^1 \rightarrow W^2 \rightarrow \ldots
\]

\[
x^2 \rightarrow W^1 \rightarrow z^2 \rightarrow \text{Sigmoid} \rightarrow a^2 \rightarrow W^2 \rightarrow \ldots
\]

\[
x^3 \rightarrow W^1 \rightarrow z^3 \rightarrow \text{Sigmoid} \rightarrow a^3 \rightarrow W^2 \rightarrow \ldots
\]

\[
\begin{align*}
W^1 & = \begin{bmatrix} z^1 & z^2 & z^3 \end{bmatrix} \\
x^1 & = \begin{bmatrix} x^1 \end{bmatrix}, \quad x^2 = \begin{bmatrix} x^2 \end{bmatrix}, \quad x^3 = \begin{bmatrix} x^3 \end{bmatrix}
\end{align*}
\]
Batch normalization

\[ \mu = \frac{1}{3} \sum_{i=1}^{3} z^i \]
\[ \sigma = \sqrt{\frac{1}{3} \sum_{i=1}^{3} (z^i - \mu)^2} \]

Note: Batch normalization cannot be applied on small batch.

\( \mu \) and \( \sigma \) depends on \( z^i \)
Batch normalization

\[ \tilde{z}^i = \frac{z^i - \mu}{\sigma} \]

\[ \mu \text{ and } \sigma \text{ depends on } z^i \]

How to do backpropogation?
Batch normalization

\[ \tilde{z}^i = \frac{z^i - \mu}{\sigma} \]

\[ \hat{z}^i = \gamma \odot \tilde{z}^i + \beta \]

\[ \mu \text{ and } \sigma \text{ depends on } z^i \]
Batch normalization

- At testing stage:

\[ \tilde{z} = \frac{z - \mu}{\sigma} \]

\( \mu, \sigma \) are from batch

\[ \hat{z}^i = \gamma \odot \tilde{z}^i + \beta \]

\( \gamma, \beta \) are network parameters

We do not have batch at testing stage.

Ideal solution:

Computing \( \mu \) and \( \sigma \) using the whole training dataset.

Practical solution:

Computing the moving average of \( \mu \) and \( \sigma \) of the batches during training.
Batch normalization - Benefit

- BN reduces training times, and make very deep net trainable.
  - Because of less Covariate Shift, we can use larger learning rates.
  - Less exploding/vanishing gradients
    - Especially effective for sigmoid, tanh, etc.
  - Learning is less affected by initialization.

- BN reduces the demand for regularization.

\[
\hat{z}^i = \gamma \odot \tilde{z}^i + \beta
\]

\[
\tilde{z}^i = \frac{k z^i - k \mu}{k \sigma}
\]
Figure 2: Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.
To learn more ......

- Batch Renormalization
- Layer Normalization
- Instance Normalization
- Weight Normalization
- Spectrum Normalization
Activation Function: SELU
ReLU

- Rectified Linear Unit (ReLU)

\[ \sigma(z) \]

Reason:

1. Fast to compute
2. Biological reason
3. Infinite sigmoid with different biases
4. Vanishing gradient problem
ReLU - variant

**Leaky ReLU**

\[ a = z \]

\[ a = 0.01z \]

**Parametric ReLU**

\[ a = \alpha z \]

\[ a = z \]

\( \alpha \) also learned by gradient descent
(1) Definition of scaled exponential linear units (SELUs)

In [3]:
```python
    def selu(x):
        with ops.name_scope('elu') as scope:
            alpha = 1.6732632423543772848170429916717
            scale = 1.0507009873554804934193349852946
            return scale*tf.where(x>=0.0, x, alpha*tf.nn.elu(x))
```

https://github.com/bioinf-jku/SNNs

---

**Exponential Linear Unit (ELU)**

\[ a = \alpha (e^z - 1) \]

\[ a = z \text{ if } z \geq 0 \]

**Scaled ELU (SELU)**

\[ a = \alpha (e^z - 1) \times \lambda \]

\[ a = z \text{ if } z \geq 0 \]

\[ \alpha = 1.6732632423543772848170429916717 \]

\[ \lambda = 1.0507009873554804934193349852946 \]
The whole ReLU family has this property except the original ReLU.

ELU also has this property

Only SELU also has this property
The inputs are i.i.d random variables with mean $\mu$ and variance $\sigma^2 = 1$.

$$z = a_1 w_1 + \Lambda + a_k w_k + \Lambda + a_K w_K$$

Do not have to be Gaussian.
The inputs are i.i.d random variables with mean $\mu$ and variance $\sigma^2$. 

$$
\mu_z = 0 \quad \mu_w = 0 \\
\sigma_z^2 = E[(z - \mu_z)^2] = E[z^2] \\
= E[(a_1w_1 + a_2w_2 + \cdots)^2] \\
= \sum_{k=1}^{K} (w_k)^2 \sigma^2 = \sigma^2 \cdot K \sigma_w^2 = 1 \\
= 1 \quad = 1
$$

$$
E[(a_kw_k)^2] = (w_k)^2 E[(a_k)^2] = (w_k)^2 \sigma^2
$$

$$
E[a_i a_j w_i w_j] = w_i w_j E[a_i] E[a_j] = 0
$$

Assume Gaussian

$$
\mu = 0, \sigma = 1
$$

$$
z = a_1w_1 + \Lambda + a_kw_k + \Lambda + a_K w_K
$$
Demo
SELU is actually more general.
**MNIST**

- **BatchNorm Depth 8**
- **BatchNorm Depth 16**
- **BatchNorm Depth 32**
- **SNN Depth 8**
- **SNN Depth 16**
- **SNN Depth 32**

**CIFAR-10**

- **BatchNorm Depth 8**
- **BatchNorm Depth 16**
- **BatchNorm Depth 32**
- **SNN Depth 8**
- **SNN Depth 16**
- **SNN Depth 32**

---

### FNN Method Comparison

<table>
<thead>
<tr>
<th>Method</th>
<th>avg. rank diff.</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNN</td>
<td>-0.756</td>
<td></td>
</tr>
<tr>
<td>MSRAinit</td>
<td>-0.240*</td>
<td>2.7e-02</td>
</tr>
<tr>
<td>LayerNorm</td>
<td>-0.198*</td>
<td>1.5e-02</td>
</tr>
<tr>
<td>Highway</td>
<td>0.021*</td>
<td>1.9e-03</td>
</tr>
<tr>
<td>ResNet</td>
<td>0.273*</td>
<td>5.4e-04</td>
</tr>
<tr>
<td>WeightNorm</td>
<td>0.397*</td>
<td>7.8e-07</td>
</tr>
<tr>
<td>BatchNorm</td>
<td>0.504*</td>
<td>3.5e-06</td>
</tr>
</tbody>
</table>

### ML Method Comparison

<table>
<thead>
<tr>
<th>Method</th>
<th>avg. rank diff.</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SNN</td>
<td>-6.7</td>
<td></td>
</tr>
<tr>
<td>SVM</td>
<td>-6.4</td>
<td>5.8e-01</td>
</tr>
<tr>
<td>RandomForest</td>
<td>-5.9</td>
<td>2.1e-01</td>
</tr>
<tr>
<td>MSRAinit</td>
<td>-5.4*</td>
<td>4.5e-03</td>
</tr>
<tr>
<td>LayerNorm</td>
<td>-5.3</td>
<td>7.1e-02</td>
</tr>
<tr>
<td>Highway</td>
<td>-4.6*</td>
<td>1.7e-03</td>
</tr>
</tbody>
</table>

...
Demo
Highway Network & Grid LSTM
Feedforward v.s. Recurrent

1. Feedforward network does not have input at each step
2. Feedforward network has different parameters for each layer

\[ a_t = f_l(a_{t-1}) = \sigma(W^t a_{t-1} + b^t) \]

\[ h_t = f(h_{t-1}, x^t) = \sigma(W^h h_{t-1} + W^i x^t + b^i) \]

Applying gated structure in feedforward network
GRU → Highway Network

No input $x^t$ at each step
No output $y^t$ at each step
$a^{t-1}$ is the output of the (t-1)-th layer
$a^t$ is the output of the t-th layer
No reset gate
Highway Network

- Highway Network

\[
h' = \sigma(Wa^{t-1}) \\
z = \sigma(W'a^{t-1}) \\
a^t = z \odot a^{t-1} + (1 - z) \odot h
\]

- Residual Network


Highway Network automatically determines the layers needed!
Highway Network

MNIST

CIFAR-100

Mean Cross Entropy Error vs. Lesioned Highway Layer

non-lesioned performance
Grid LSTM

Memory for both time and depth

The figure illustrates the Grid LSTM architecture, which combines LSTM for time and depth memory. The LSTM unit processes the input sequence $x$ and passes it through the cell state $c$ and hidden state $h$. The updated state variables $c'$ and $h^t$ represent the memory for time, while the values $a'$ and $b'$ with $a$ and $b$ represent the depth memory. The diagram shows the flow of information through the network, with arrows indicating the direction of data propagation.
Grid LSTM

Grid LSTM

Grid LSTM

h' b'

c'
c

a'
a

h

b

z^f

z^i

z

z^o

h

b

⨀

⨀

⨀

tanh

⨀
3D Grid LSTM
3D Grid LSTM

- Images are composed of pixels

3 x 3 images