Widely Used DL Techniques

- Dead ReLUs
- Vanishing gradients and skip connections
- Batch normalization
- Transfer learning, data augmentation, and hyperparameter optimization
Dead ReLUs

- If input to last ReLU layer is 0,

\[ 0 \geq w_{(j_1,j_2),i} \times z^{4,(j_1,j_2),i} + b \]

- Then we have that

\[ 0 = \nabla \sigma (w_{(j_1,j_2),i} \times z^{4,(j_1,j_2),i} + b) \]

- So all back propagated gradients = 0!

- Solution: Leaky ReLU
Vanishing gradients

- Consider a pipelined DNN

- Associated back-propagation network
The Matrix Operator Norm

- Operator norm of a matrix is
  \[ \|A\| = \max_x \{\|Ax\| : \|x\| \leq 1\} \]
  or equivalently
  \[ \|A\| = \max_x \left\{ \frac{\|Ax\|}{\|x\|} \right\} \]

- Singular value decomposition (SVD)
  \[ A = U \Sigma V^t \]
  where \( \Sigma \) is diagonal, \( U \) and \( V \) have orthogonal columns.

- Relationship of SVD to matrix norm
  \[ \|A\| = \sigma_A^* \]
  where \( \sigma_A^* = \max_i \Sigma_{i,i} \)
Norm of product of matrices

- Assume $A$ is formed from the product of matrices

$$A = A_0 A_1 A_2 \cdots A_{M-1}$$

Then it’s easily shown that

$$\|A\| \leq \|A_0\| \|A_1\| \cdots \|A_{M-1}\|$$

$$= \sigma^*_A \sigma^*_A \cdots \sigma^*_A$$

- So we have that

$$\|Ax\| \leq \sigma^*_A \sigma^*_A \cdots \sigma^*_A \|x\|$$
Vanishing gradients: Analysis

- Associated back-propagation network

\[
g_0 = B_0^t A_1^t A_2^t A_3^t \cdots A_{M-1}^t \epsilon
\]

- Using the relationships from the previous slide, we can bound the gradients norm by

\[
\|g_0\| \leq \sigma_{B_0}^* \sigma_{A_1}^* \sigma_{A_2}^* \cdots \sigma_{A_{M-1}}^* \|\epsilon\|
\]

So if \(\sigma_{A_m}^* \leq 1\), then as the network becomes deeper, we have that

\[
\lim_{M \to \infty} \|g_0\| = 0
\]

- Partial Solutions: ADAM, ResNet, skip connection, LSTM
Skipped Connections

- Skip connection concept

\[ y \xrightarrow{+} \hat{x} \xleftarrow{f_\theta} \theta \]

- Adjoint gradient

\[ I + A^t \xrightarrow{\epsilon} B^t \xleftarrow{g_2} \]

Deep Skipped Connections

- Skip connection concept

\[
g + B + \cdots + I + A - 8 \epsilon \quad \text{if} \quad \sigma / 0, \
\text{we have that} \quad g \leq \sigma^* \epsilon \Rightarrow \text{fixes vanishing gradient}
\]

- Back propagation

- Adjoint gradient

\[
g_1 = B^t_1 (I + A^t_2) \cdots (I + A^t_M) \epsilon
\]

if \( \sigma^*_{(I+A)} \approx 1 \), we have that \( \|g_0\| \leq \sigma^*_B \|\epsilon\| \Rightarrow \text{fixes vanishing gradient} \)
Residual Learning and Skip Connections*

● Skip connections

Denoising

- The forward model is \( y = x + w \)

- The denoising inverse problem

- Simplest of all inverse problems
- Typically, \( w \sim N(0, \sigma^2 I) \)
- Loss function is \( L(\theta) = \frac{1}{K} \sum_{k=0}^{K-1} \| x_k - f_\theta(y_k) \|^2 \)
- Maximum likelihood estimate of \( \theta \)
  \[ \hat{\theta} = \arg \min_{\theta} L(\theta) \]
- MMSE estimate of \( x \)
  \[ \hat{x} = f_{\hat{\theta}}(y) \]
Predicting the Noise

- Since the forward model is $y = x + w$, denoised image is given by
  \[
  \hat{x} = E[x|y] = E[y-w|y] = y - E[w|y] = y - f_\theta(y)
  \]
  where $\hat{w} = f_\theta(y)$ is an estimate of the noise.

- Intuition:
  - Noise is easier to estimate because it is “smaller”
  - Estimate the noise, and subtract it from the noisy image
  - Same as skipped connection
Denoising ResNet (DN-ResNet)*

- Typical DN-ResNet denoising network

![Diagram](image)

\[
y - N \times N \times 1 \rightarrow (3 \times 3) \text{ ReLU} \\
N \times N \times 64 \rightarrow (3 \times 3) \text{ ReLU} \\
N \times N \times 64 \rightarrow \ldots \\
N \times N \times 64 \rightarrow (3 \times 3) \text{ ReLU} \\
N \times N \times 64 \rightarrow (3 \times 3) \text{ ReLU} \\
N \times N \times 1 \\
\]

15 Layers

- Limitation: Dependencies are local

Problem: Slow Training

- Example

  - Training can be slow in deep layers
  - Internal features can vary rapidly

- Solution: Batch Normalization
Batch Normalization

- **Example**

  - During training:
    - Normalizes sample mean and variance of features for each batch
    - Does not contain trainable parameters
    - Remembers mean and variance estimates learned using “momentum”

  - During inference:
    - Uses mean and variance estimates to normalize features

- **Issues:**
  - Can be very effective, particularly with ResNet training
  - Can add multiplicative “noise” to estimate
  - Best to use when output is small or does not require high relative accuracy
  - It is sometimes useful to turn off parameter adaptation for final training.

Batch Normalization: Training

Example for layer with input feature tensor $z$ and output tensor $y$

Initialize $\mu \leftarrow 0; \sigma^2 \leftarrow 0$

For each $b$ (batch) { 
$$
\mu_B \leftarrow \frac{1}{K_b} \sum_{k \in S_b} z_k,
$$
$$
\sigma^2_B \leftarrow \frac{1}{K_b - 1} \sum_{k \in S_b} (z_k - \mu_B)^2,
$$

For each $k \in S_b$ { 
$$
\hat{z}_k \leftarrow \frac{z_k - \mu_B}{\sqrt{\sigma^2_B + \epsilon}},
$$

$$
y_k \leftarrow \gamma \hat{z}_k + \beta
$$

}

$$
\mu \leftarrow \alpha \mu + (1 - \alpha)\mu_B
$$
$$
\sigma^2 \leftarrow \alpha \sigma^2 + (1 - \alpha)\sigma^2_B
$$

}

Input parameters:

$\gamma, \beta$ – scaling and offset parameters
$\epsilon$ – small number to regularize division
$\alpha$ – momentum parameter

Output parameters:

$\mu, \sigma^2$ – learned mean and variance parameters
$\gamma, \beta, \epsilon$ – same as above

Comments:

Time average over $\approx \frac{-1}{\log \alpha}$

Typically, $\alpha \approx 0.99$

$\beta$ can be used to ensure that ReLUs are not off

Subtle questions: Does back propagation/SGD update the values of $\mu_B$ and $\sigma^2_B$? Answer: No, these are treated as fixed program variables that are precomputed for each component feature of each batch. But this means that there is essentially a random variation with each batch.
Batch Normalization: Inference

- Example for layer with input feature tensor $z$ and output tensor $y$

Get Input:

- $\epsilon, \gamma, \beta$ – same as used in training
- $\mu_I, \sigma^2_I$ – estimated during training

For each input {

- $\hat{z} \leftarrow \frac{z - \mu_I}{\sqrt{\sigma^2_I + \epsilon}}$
- $y \leftarrow \gamma \hat{z} + \beta$

}

Comments:
- Typically, only one input for testing
- The noisy values of $\mu_B$ and $\sigma^2_B$ used in training will typically cause variation in the output of the CNN.
- BN works well in CNNs used for residual estimation.

Warning:
- Don’t use BN in CNNs that must have large and precise outputs.
- Or turn off BN adaptation in final training.
Batch Normalization: Important Practical Issues

- Typically, $\mu_i, \sigma_i^2$ are estimated for each channel
  - “axis: Integer, the axis that should be normalized (typically the features axis). For instance, after a Conv2D layer with data_format="channels_first", set axis=1 in BatchNormalization.”
  - Channels typically contain different features for each pixel
Practical Batch Normalization: Training

- Example for layer with input feature tensor $z$ and output tensor $y$

Initialize $\mu_l \leftarrow 0; \sigma_l^2 \leftarrow 0$

For each $b$ (batch) {
  For each channel $i$ {
    $\mu_{B,i} \leftarrow \frac{1}{K_b N_1 N_2} \sum_{k \in S_b} \sum_{j_1,j_2} z_{i,j_1,j_2,k}$
    $\sigma_{B,i}^2 \leftarrow \frac{1}{K_b N_1 N_2} \sum_{k \in S_b} \sum_{j_1,j_2} (z_k - \mu_B)^2$
  }
  For each $k \in S_b, j_1, j_2$ {
    $\hat{z}_{i,j_1,j_2,k} \leftarrow \frac{z_{i,j_1,j_2,k} - \mu_{B,i}}{\sqrt{\sigma_{B,i}^2 + \epsilon}}$
    $y_{i,j_1,j_2,k} \leftarrow \gamma \hat{z}_{i,j_1,j_2,k} + \beta$
  }
  $\mu_{l,i} \leftarrow \alpha \mu_{l,i} + (1 - \alpha) \mu_{B,i}$
  $\sigma_{l,i}^2 \leftarrow \alpha \sigma_{l,i}^2 + (1 - \alpha) \sigma_{B,i}^2$
}

Block Normalization: Training

Input parameters:
- $\gamma, \beta$ – scaling and offset parameters
- $\epsilon$ – small number to regularize division
- $\alpha$ – momentum parameter

Output parameters:
- $\mu_{l,i}, \sigma_{l,i}^2$ – learned mean and variance parameters
- $\gamma, \beta, \epsilon$ – same as above

Comments:
- $\mu_{l,i}, \sigma_{l,i}^2$ is estimated for each channel $i$
Practical Batch Normalization: Inference

- Example for layer with input feature tensor $z$ and output tensor $y$

Get Input:

$\epsilon, \gamma, \beta$ – same as used in training

$\mu_i, \sigma_i^2$ – estimated during training

For each channel $i, j_1, j_2$

$$\hat{z}_{i,j_1,j_2} \leftarrow \frac{z_{i,j_1,j_2} - \mu_{i,i}}{\sqrt{\sigma_{i,i}^2 + \epsilon}}$$

$$y_{i,j_1,j_2} \leftarrow \gamma \hat{z}_{i,j_1,j_2} + \beta$$

Batch Normalization: Inference
Batch Renormalization

- **New and Improved Batch Normalization**

- **Problem with Batch Normalization**
  - Batch Normalization parameters are not learned through gradient descent.
  - There may be differences between training and inference.
  - Normalization is done differently between training and inference.

- **Solution:**
  - Tries to learn some batch normalization parameter through back projection.
  - Still retains some parameters used through moving averages updating.

Merge Connections*

- Problem with skip connection
  - Some information may be lost by simply adding $y$ and $f_\theta(y)$
  - Example, $f_\theta(y) = -y$

- Merge Connection*
  - Concatenate $y$ and $f_\theta(y)$

*As far as I know, I made up this term.
U-Net*

- Can model long distance dependencies
- Much like a wavelet transform, but nonlinear

Figure taken from https://lmb.informatik.uni-freiburg.de/people/ronneber/u-net/

Data Augmentation

- Problem: Not enough training data
- Solution: Transform the data in different ways

- Can use:
  - Translation
  - Rotation
  - Stretching
  - Shearing
  - Whatever…