General DL Techniques

- Dead ReLUs
- Vanishing gradients and skip connections
- Batch normalization
- Transfer learning
- Data Augmentation
- Hyperparameter optimization
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!
Vanishing gradients

- Consider a pipelined DNN

\[
\begin{align*}
y &\rightarrow f_{0,\theta_0} \rightarrow f_{1,\theta_1} \rightarrow \cdots \rightarrow f_{M-1,\theta_M} \rightarrow \hat{x} \\
z_0 &\uparrow \quad \theta_0 &\uparrow \quad \theta_1 &\uparrow \quad \theta_2 &\uparrow \\
&\downarrow \quad \theta_0 &\uparrow \quad \theta_1 &\uparrow \quad \theta_2 &\uparrow \\
\end{align*}
\]

- Associated back-propagation network

\[
\begin{align*}
A_0^t &\leftrightarrow A_1^t \leftrightarrow \cdots \leftrightarrow A_{M-1}^t &\rightarrow \epsilon \\
B_0^t &\downarrow g_0 &\downarrow g_1 &\downarrow g_2 &\downarrow \\
&\downarrow \quad B_0^t &\downarrow \quad B_1^t &\downarrow \quad B_{M-1}^t &\downarrow \\
\end{align*}
\]
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 $\sigma_A^* = \max_i \Sigma_{i,i}$
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

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

$$ = \sigma^*_{A_0} \sigma^*_{A_1} \cdots \sigma^*_{A_{M-1}} $$

So we have that

$$ \|Ax\| \leq \sigma^*_{A_0} \sigma^*_{A_1} \cdots \sigma^*_{A_{M-1}} \|x\| $$
Vanishing gradients: Analysis

- Associated back-propagation network

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

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

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: ResNet, skip connection, LSTM
Skipped Connections

- Skip connection concept

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

- Adjoint gradient

\[ \epsilon \xrightarrow{I + A^t} \]
\[ B^t \]
\[ g_2 \]
Deep Skipped Connections

- Skip connection concept

- Back propagation

- Adjoint gradient

\[
g_1 = B_1^t (I + A_2^t) \cdots (I + A_M^t) \epsilon
\]

if \( \sigma_{(I+A)}^* \approx 1 \), we have that \( \|g_0\| \leq \sigma_{B_0}^* \|\epsilon\| \Rightarrow \text{fixes vanishing gradient} \)
The forward model is $y = x + w$

unknown image $\xrightarrow{+} y$

noisy image $\xleftarrow{w \text{ noise}}$

The denoising inverse problem

$y \xrightarrow{f_\theta(y)} \hat{x}$

noisy image $\xleftarrow{\theta}$

denoised image

- 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

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

Batch Normalization: Training

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

Initialize $\mu_I \leftarrow 0; \sigma_I^2 \leftarrow 0$

For each $b$ (batch) {

$$\mu_B \leftarrow \frac{1}{K_b} \sum_{k \in S_b} z_k$$

$$\sigma_B^2 \leftarrow \frac{1}{K_b} \sum_{k \in S_b} (z_k - \mu_B)^2$$

For each $k \in S_b$ {

$$\hat{z}_k \leftarrow \frac{z_k - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

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

$$\mu_I \leftarrow \alpha \mu_I + (1 - \alpha) \mu_B$$

$$\sigma_I^2 \leftarrow \alpha \sigma_I^2 + (1 - \alpha) \sigma_B^2$$

}

Input parameters:

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

Output parameters:

- $\mu_I, \sigma_I^2$ – 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

Block Normalization: Training

Subtle questions: Does the back propagation account for the variation of $\mu_B$ and $\sigma_B^2$? I don’t think so, but I’m not totally sure. I think these are treated in the software like fixed program variables that are precomputed for each 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:

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

Usually only 1 input for inference
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 $\mu_I \leftarrow 0; \sigma_I^2 \leftarrow 0$

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 $k \in S_b, j_1, j_2$ {

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

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

} }

$$\mu_{I,i} \leftarrow \alpha \mu_{I,i} + (1 - \alpha)\mu_{B,i}$$

$$\sigma_{I,i}^2 \leftarrow \alpha \sigma_{I,i}^2 + (1 - \alpha)\sigma_{B,i}^2$$

**Input parameters:**

$\gamma, \beta$ – scaling and offset parameters

$\epsilon$ – small number to regularize division

$\alpha$ – momentum parameter

**Output parameters:**

$\mu_{I,i}, \sigma_{I,i}^2$ – learned mean and variance parameters

$\gamma, \beta, \epsilon$ – same as above

**Comments:**

$\mu_{I,i}, \sigma_{I,i}^2$ is estimated for each channel $i$
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 channel $i, j_1, j_2$:

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

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

Batch Normalization: Inference
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.
• 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/

Weight initialization and regularization
Data Augmentation

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

- Can use:
  - Translation
  - Rotation
  - Stretching
  - Shearing
  - Whatever…
Transfer Learning
Hyperparameter optimization
Recurrent Neural Networks*

- Basic concept of RNNs
- LSTM networks
- GRU networks

*Slides “barrowed” from Prof. Greg Buzzard
Basic Concept of RNN

- **State machine viewpoint**
  - $x_t, y_t$ - input and output
  - $s_t$ - state
  - $\theta$ – parameter

- **“Unrolling the loop”**
  - Parameters $\theta$ are shared
  - Time dependent processes such as speech

\[
[y_t, s_t] = f_\theta(x_t, s_{t-1})
\]
RNN Problems and Solutions

Problem:
- Back propagation now iterates in time
- Gradient tends to vanish over long time scales
- Difficult to model long time dependencies in data

Solution:
- Use skip connection, batch normalization (BM) and tricky methods for gating information from the past.
- Results in long-short time memory (LSTM) RNN
Long-Short Term Memory (LSTM)

- LSTM architecture
  - State has two components \( s_t = [C_t, h_t] \)
  - \( C_t \) - cell state
    - Store information that flows from one time to the next
    - Reduces vanishing gradient problem much like the skipped connection
  - \( h_t \) - hidden state
    - This is usually the output of the LSTM
    - It typically needs to be further processed to produce the desired output
A Look Inside an LSTM block

Forget old information that is not useful

Add new information that is of value

\[ C_t = f_t \cdot C_{t-1} + a_t \]
A Look Inside an LSTM block

\[ f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \]
\[ i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \]
\[ \tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \]
\[ C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \]
A Look Inside an LSTM block

\[ f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \]
\[ i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \]
\[ \tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \]
\[ C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t \]
\[ o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \]
\[ h_t = o_t \cdot \tanh(C_t) \]
Basic Structure of LSTM

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
LSTM Cell State

- Two key ideas of LSTM:
  - A backbone to carry state forward and gradients backward.
  - Gating (pointwise multiplication) to modulate information flow. Sigmoid makes $0 < \text{gate} < 1$.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
The \( f \) gate is ‘forgetting.’ Use previous state, \( C \), previous output, \( h \), and current input, \( x \), to determine how much to suppress previous state.

- E.g., \( C \) might encode the fact that we have a subject and need a verb. Forget that when verb found.

\[
f_t = \sigma \left( W_f \cdot [h_{t-1}, x_t] + b_f \right)
\]
Input gate $i$ determines which values of $C$ to update.
Separate tanh layer produces new state to add to $C$.

\[
i_t = \sigma (W_i \cdot [h_{t-1}, x_t] + b_i)
\]
\[
\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)
\]
LSTM gating: update to $C$

- Forget gate does pointwise modulation of $C$.
- Input gate modulates the tanh layer – this is added to $C$.

$$C_t = f_t \times C_{t-1} + i_t \times \tilde{C}_t$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- $o$ is the output gate: modulates what part of the state $C$ gets passed (via tanh) to current output $h$.

- E.g., could encode whether a noun is singular or plural to prepare for a verb.

- But the real features are learned, not engineered.

$$o_t = \sigma (W_o [h_{t-1}, x_t] + b_o)$$
$$h_t = o_t \times \tanh (C_t)$$

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
- Combine $C$ and $h$ into a single state/output.
- Combine forget and input gates into update gate, $z$.

\[
\begin{align*}
z_t &= \sigma (W_z \cdot [h_{t-1}, x_t]) \\
r_t &= \sigma (W_r \cdot [h_{t-1}, x_t]) \\
\tilde{h}_t &= \tanh (W \cdot [r_t \cdot h_{t-1}, x_t]) \\
h_t &= (1 - z_t) \cdot h_{t-1} + z_t \cdot \tilde{h}_t
\end{align*}
\]
• Dropout is used to regularize weights and prevent co-adaptation.
• Dropout for RNNs must respect the time invariance of weights and outputs.
• In Keras GRU, dropout applies vertically, recurrent_dropout applied horizontally.

(a) Naive dropout RNN
(b) Variational RNN
ConvLSTM2D for SBP

\[ X'(16*256*256*1) \]
Implementation details of ConvLSTM2D
Why use ConvLSTM to SBP

Since

In ConvLSTM, cell state can accumulate past information by minor linear interactions. And forget gate and input gate will decide what kind of information will be stored by comparing past cell state and current new information.

\[
c_t = f_t \circ c_{t-1} + i_t \circ \tanh(W_{xc} \cdot x_t + W_{hc} \cdot h_{t-1} + b_c)
\]
Autoencoders
Generative Adversarial Networks