General DL Techniques

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

- Consider a pipelined DNN
  
  ![Diagram of a pipelined DNN with functions $f_0, f_1, \ldots, f_{M-1}$ and parameters $\theta_0, \theta_1, \ldots, \theta_{M-1}$, and input $y$ and output $\hat{x}$]

- Associated back-propagation network

  ![Diagram of the associated back-propagation network with variables $A_0^t, A_1^t, \ldots, A_{M-1}^t$, $B_0^t, B_1^t, \ldots, B_{M-1}^t$, and error $\epsilon$]
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

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

\[ = \sigma^*_0 \sigma^*_1 \cdots \sigma^*_M \]

- So we have that

\[ \|Ax\| \leq \sigma^*_0 \sigma^*_1 \cdots \sigma^*_M \|x\| \]
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: ResNet, skip connection, LSTM
Skipped Connections

- Skip connection concept

\[ f_{\theta} \]

- Adjoint gradient

\[ (I + A^t)B^t \]
Deep Skipped Connections

- Skip connection concept
  \[ y \xrightarrow{f_1,\theta} + \xrightarrow{f_2,\theta} + \cdots \xrightarrow{f_M,\theta} \hat{x} \]

- Back propagation
  \[ I + A_1^t \xrightarrow{B_1^t} g_1 \quad I + A_2^t \xrightarrow{B_2^t} g_2 \quad \cdots \quad I + A_M^t \xrightarrow{B_M^t} g_M \quad \epsilon \]

- 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 \( \|g_0\| \leq \sigma_B^* \|\epsilon\| \Rightarrow \) fixes vanishing gradient
Denoising

- The forward model is $y = x + w$

\[ x \xrightarrow{+} y \]

unknown image noisy image

$w$ noise

- The denoising inverse problem

\[ f_\theta (y) \rightarrow \hat{x} \]

noisy image denoised image

$\theta$

- 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 $x$
  \[ \hat{x} = f_\theta (y) \]
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

\[ y \quad \rightarrow \quad - \quad \rightarrow \quad \hat{x} \quad \text{- denoised image} \]

\[ f_\theta(y) \quad \rightarrow \quad \hat{w} \quad \text{- noise estimate} \]

\[ \theta \]
Denoising ResNet (DN-ResNet)*

- Typical DN-ResNet denoising network

![Diagram of DN-ResNet network]

15 Layers

- Limitation: Dependencies are local

**Problem: Slow Training**

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

- **Solution: Block Normalization**
**Block 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

Block Normalization: Training

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

For each batch {

$$\mu_B \leftarrow \frac{1}{K_b} \sum_{k=0}^{K_b-1} z_k$$

$$\sigma_B^2 \leftarrow \frac{1}{K_b} \sum_{k=0}^{K_b-1} (z_k - \mu_B)^2$$

For each sample in the batch {

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

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

}\}

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

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

\}

Block Normalization Inference

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.
**Block Normalization: Inference**

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

Read stored values of $\mu_I, \sigma_I^2, \epsilon, \gamma$, and $\beta$

For each input \{ 
\[
\hat{z} \leftarrow \frac{z - \mu_I}{\sqrt{\sigma_I^2 + \epsilon}}
\]

\[
y \leftarrow \gamma \hat{z} + \beta
\]

\}
**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
The basic structure of LSTM and some symbols to aid understanding

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
LSTM core ideas

- 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/
LSTM gating: forget

- 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.

\[ f_t = \sigma (W_f \cdot [h_{t-1}, x_t] + b_f) \]
LSTM gating: input gate

- Input gate $i$ determines which values of $C$ to update.
- Separate tanh layer produces new state to add to $C$.

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

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

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

- $o$ 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.

\[ o_t = \sigma (W_o [h_{t-1}, x_t] + b_o) \]
\[ h_t = o_t \times \tanh (C_t) \]
- Combine $C$ and $h$ into a single state/output.
- Combine forget and input gates into update gate, $z$.

\[
\begin{align*}
    z_t &= \sigma \left( W_z \cdot [h_{t-1}, x_t] \right) \\
    r_t &= \sigma \left( W_r \cdot [h_{t-1}, x_t] \right) \\
    \tilde{h}_t &= \tanh \left( W \cdot [r_t * h_{t-1}, x_t] \right) \\
    h_t &= (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t
\end{align*}
\]
RNN dropout

- 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.
Autoencoders
Generative Adversarial Networks