Lecture 14

Advanced Neural Networks

Michael Picheny, Bhuvana Ramabhadran, Stanley F. Chen, Markus Nussbaum-Thom

Watson Group
IBM T.J. Watson Research Center
Yorktown Heights, New York, USA
{picheny,bhuvana,stanchen,nussbaum}@us.ibm.com

27 th April 2016

Variants of Neural Network Architectures

- Deep Neural Network (DNN),
- Convolutional Neural Network (CNN),
- Recurrent Neural Network (RNN),
  - unidirectional, bidirectional, Long Short-Term Memory (LSTM), Gated Recurrent Unit (GRU),
- Constraints and Regularization,
- Attention model,

Training

- Observations and labels $(x_n, a_n) \in \mathbb{R}^D \times A$ for $n = 1, \ldots, N$.
- Training criterion:

$$\mathcal{F}_{CE}(\theta) = -\frac{1}{N} \sum_{n=1}^{N} \log P(a_n | x_n, \theta)$$

$$\mathcal{F}_{\mathcal{L}}(\theta) = \frac{1}{N} \sum_{n=1}^{N} \sum_{\overline{\omega}} \sum_{a_1^{T_n} \in \overline{\omega}} P(a_1^{T_n} | x_1^{T_n}, \theta) \cdot \mathcal{L}(\overline{\omega}, \omega_n) \qquad \text{loss } \mathcal{L}$$

- Optimization:

$$\overline{\theta} = \arg\min_{\theta} \{\mathcal{F}(\theta)\}$$

- $\theta, \overline{\theta}$: Free parameters of the model (NN, GMM).
- $\omega, \overline{\omega}$: Word sequences.

Recap: Gaussian Mixture Model

- Recap Gaussian Mixture Model:

$$P(\omega|x_1^T) = \sum_{a_1^T \in \omega} \prod_{t=1}^{T} P(x_t|a_t)P(a_t|a_{t-1})$$

  - ω: word sequence
  - $x_1^T := x_1, \ldots, x_T$: feature sequence
  - $a_1^T := a_1, \ldots, a_T$: HMM state sequence
- Emission probability $P(x|a) \sim \mathcal{N}(\mu_a, \Sigma_a)$ Gaussian.
- Replace with a neural network $\Rightarrow$ hybrid model.
- Use neural network for feature extraction $\Rightarrow$ bottleneck features.

Hybrid Model

- Gaussian Mixture Model:

$$P(\omega|x_1^T) = \sum_{a_1^T \in \omega} \prod_{t=1}^{T} \underbrace{P(x_t|a_t)}_{\text{emission}} \underbrace{P(a_t|a_{t-1})}_{\text{transition}}$$

- Training: A neural network usually models $P(x|a)$.
- Recognition: Use as a hybrid model for speech recognition:

$$\frac{P(a|x)}{P(a)} = \frac{P(x,a)}{P(x)P(a)} = \frac{P(x|a)}{P(x)} \approx P(x|a)$$

$P(x|a)/P(x)$ and $P(x|a)$ are proportional.

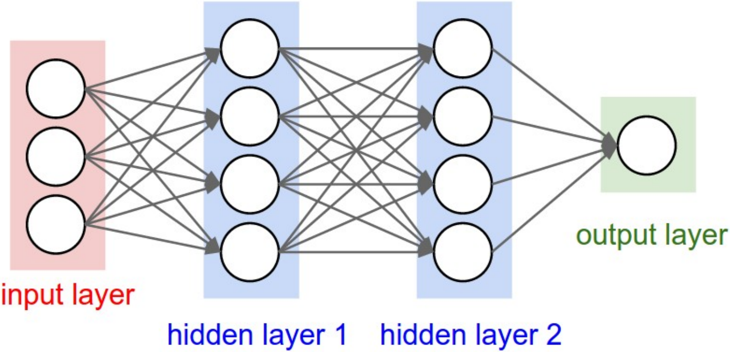
Hybrid Model and Bayes Decision Rule

$$
\begin{aligned}
\hat{\omega} &= \arg\max_{\omega} \left\{ P(\omega) P(x_1^T | \omega) \right\} \\
&= \arg\max_{\omega} \left\{ P(\omega) \sum_{a_1^T \in \omega} \prod_{t=1}^{T} \frac{P(x_t | a_t)}{P(x_t)} P(a_t | a_{t-1}) \right\} \\
&= \arg\max_{\omega} \left\{ P(\omega) \sum_{a_1^T \in \omega} \frac{\prod_{t=1}^{T} P(x_t | a_t) P(a_t | a_{t-1})}{\prod_{t=1}^{T} P(x_t)} \right\} \\
&= \arg\max_{\omega} \left\{ P(\omega) \sum_{a_1^T \in \omega} \prod_{t=1}^{T} P(x_t | a_t) P(a_t | a_{t-1}) \right\}
\end{aligned}
$$

Where Are We?

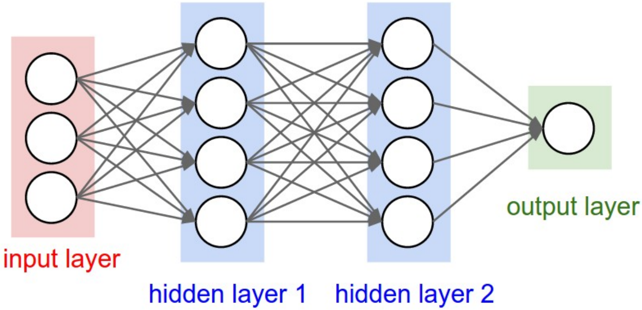
Recap: Deep Neural Network (DNN)

- First feed forward networks.
- Consists of input, multiple hidden and output layer.
- Each hidden and output layer consists of nodes.



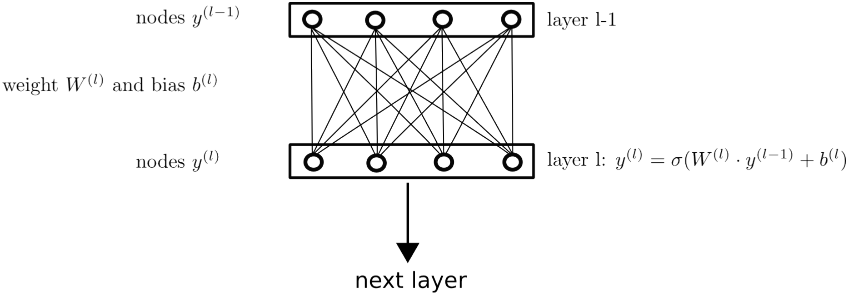
Recap: Deep Neural Network (DNN)

- Free parameters: weights W and bias b.
- Output of a layer is input to the next layer.
- Each node performs a linear followed by a non-linear activiation on the input.
- The output layer relates the output of the last hidden layer with the target states.

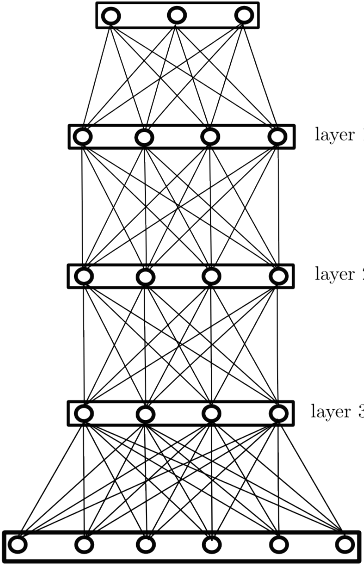


Neural Network Layer

- Number of nodes: n_l in layer l.
- Input from previous layer: $y^{(l-1)} \in \mathbb{R}^{n_{l-1}}$
- Weight and bias : $W^{(l)} \in \mathbb{R}^{n_{l-1} \times n_l}$, $b^{(l)} \in \mathbb{R}^{n_l}$.
- Activation: $y^{(l)} = \underbrace{\sigma(\underbrace{W^{(l)} \cdot y^{(l-1)} + b^{(l)}}_{\text{linear}})}_{\text{non-linear}}$



Deep Neural Network (DNN)



input $y^{(0)} = x_{t-\delta} \ldots x_t \ldots x_{t+\delta}$

weight $W^{(1)}$ and bias $b^{(1)}$

nodes $y^{(1)}$ — layer 1: $y^{(1)} = \sigma(W^{(1)} y^{(0)} + b^{(1)})$

weight $W^{(2)}$ and bias $b^{(2)}$

nodes $y^{(2)}$ — layer 2: $y^{(2)} = \sigma(W^{(2)} y^{(1)} + b^{(2)})$

weight $W^{(3)}$ and bias $b^{(3)}$

nodes $y^{(3)}$ — layer 3: $y^{(3)} = \sigma(W^{(3)} y^{(2)} + b^{(3)})$

weight $W^{(4)}$ and bias $b^{(4)}$

output nodes $y^{(4)}$ — layer 4: $y^{(4)} = \phi(W^{(4)} y^{(3)} + b^{(4)})$

Activation Function Zoo

- Sigmoid:

$$\sigma_{\mathrm{sigmoid}}(y) = \frac{1}{1 + exp(-y)}$$

- Hyperbolic tangent:

$$\sigma_{\mathrm{tanh}}(y) = \tanh(y) = 2\sigma_{\mathrm{sigmoid}}(2y)$$

- REctified Linear Unit (RELU):

$$\sigma_{\mathrm{relu}}(y) = \begin{cases} y, & y > 0 \\ 0, & y \leq 0 \end{cases}$$

Activation Function Zoo

- Parametric RELU (PRELU):

$$\sigma_{\text{prelu}}(y) = \begin{cases} y, & y > 0 \\ a \cdot y, & y \leq 0 \end{cases}$$

- Exponential Linear Unit (ELU):

$$\sigma_{\text{elu}}(y) = \begin{cases} y, & y > 0 \\ a \cdot (\exp(y) - 1), & y \leq 0 \end{cases}$$

- Maxout:

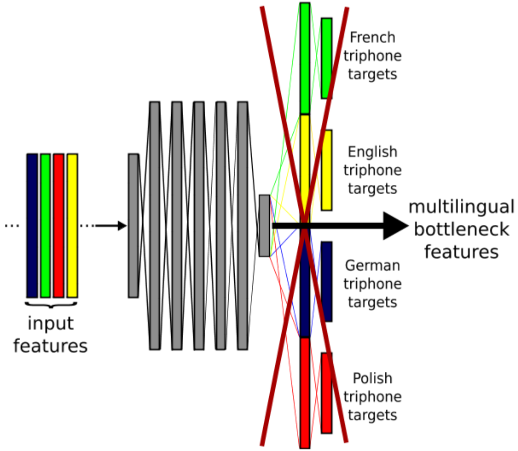
$$\sigma_{\text{maxout}}(y_1, \ldots, y_l) = \max_i \left\{ W_1 \cdot y^{(l-1)} + b_1, \ldots, W_l \cdot y^{(l-1)} + b_l \right\}$$

- Softmax:

$$\sigma_{\text{softmax}}(y) = \left(\frac{exp(y_1)}{Z(y)}, \ldots, \frac{exp(y_l)}{Z(y)} \right)^T \text{ with } Z(y) = \sum_j \exp(y_j)$$

Where Are We?

Multilingual Bottleneck



Multilingual Bottleneck

- Encoder-Decoder architecture: DNN with a bottleneck.
- Forces low-dimensional representation of speech across mutliple languages.
- Several languages are presented to the network randomly.
- Training: Labels from different languages.
- Recognition: Network is cut off after bottleneck.

Why are Multilingual Bottlenecks ?

- Train Multilingual Bottleneck features with lots of data.
- Future use: Bottleneck features on different tasks to train GMM system.
- No expensive DNN training, but WER gains similar to DNN.

Multilingual Bottleneck: Performance

	WER [%]			
Model	FR	EN	DE	PL
MFCC	23.6	28.6	23.3	18.1
MLP BN targets	19.3	23.1	19.0	14.5
MLP BN multi	18.7	21.3	17.9	14.0
deep BN targets	17.4	20.3	17.3	13.0
deep BN multi	17.1	19.7	16.4	12.6
+lang.dep. hidden layer	16.8	19.7	16.2	12.4

More Fancy Models

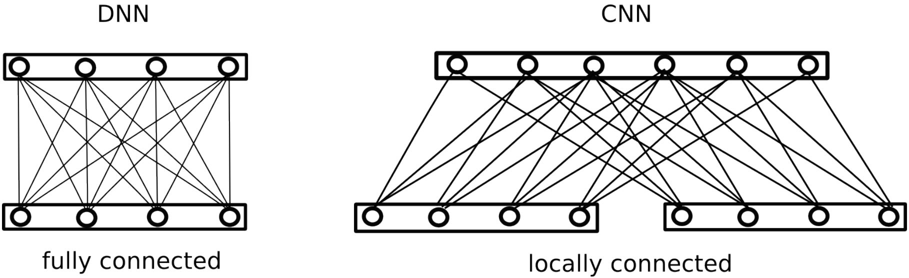
- Convolutional Neural Networks.
- Recurrent Neural Networks:
  - Long Short-Term Memory (LSTM) RNNs,
  - Gated Recurrent Unit (GRU) RNNs.
- Unstable Gradient Problem.

Where Are We?

Convolutional Neural Networks (CNNs)

- Convolution (remember signal analysis ?):

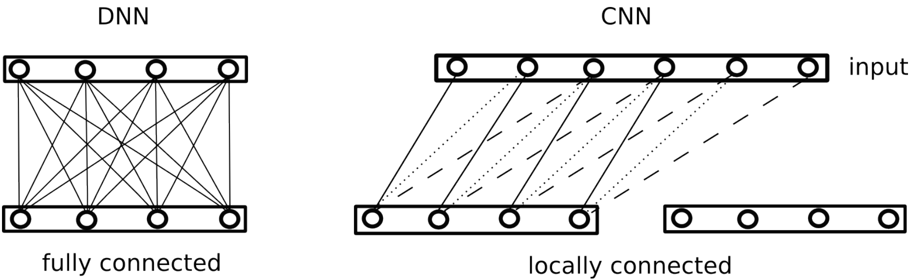
$$(x_1 * x_2)[k] = \sum_i x_1[k-i] \cdot x_2[i]$$



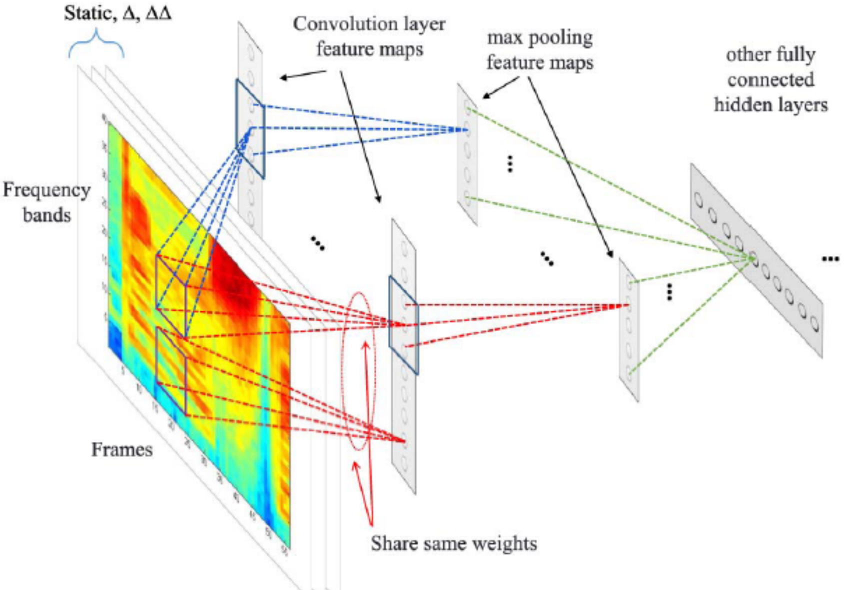
Convolutional Neural Networks (CNNs)

- Convolution (remember signal analysis ?):

$$(x_1 * x_2)[k] = \sum_i x_1[k-i] \cdot x_2[i]$$



Convolutional Neural Networks (CNNs)



CNNs

- Consists of multiple local maps with channels and kernels.
- Kernels are convolved across the input.
- Multidimensional input:
  - 1D (frequency),
  - 2D (time-frequency),
  - 3D (time-frequency-?).
- Neurons are connected to a local receptive fields of input.
- Weights are shared across multiple receptive fields.

Formal Definition: Convolutional Neural Networks

- Free parameters: Feature maps $W_n \in \mathbb{R}^{C \times k}$ bias $b_n \in \mathbb{R}^k$ for $n = 1, \ldots, N$
  - $c = 1, \ldots, C$ channels,
  - $k \in \mathbb{N}$ kernel size
- Activation function:

$$
\begin{aligned}
y_{n,i} &= \sigma(W_{n,i} * x_i + b) \\
&= \sigma\left(\sum_{c=1}^{C} \sum_{j=i-k}^{i+k} W_{n,c,i-j} x_{c,j} + b_f\right)
\end{aligned}
$$

Pooling

- Max-Pooling:

$$\text{pool}(y_{n,c,i}) = \max_{j=i-k,\dots,i+k} \{y_{n,c,j}\}$$

- Average-Pooling:

$$\text{average}(y_{n,c,i}) = \frac{1}{2 \cdot k + 1} \sum_{j=i-k}^{i+k} y_{n,c,j}$$

CNN vs. DNN: Performance

- GMM, DNN use fMLLR features.
- CNN use log-Mel features which have local structure,
- opposed to speaker normalized features.

Table: Broadcast News 50 h.

	WER [%]	
Model	CE	ST
GMM	18.8	n/a
DNN	16.2	14.9
CNN	15.8	13.9
CNN+DNN	15.1	13.2

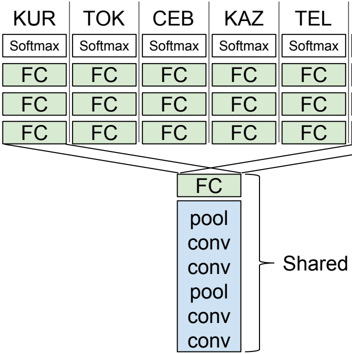
Table: Broadcast conversation 2k h.

	WER [%]	
Model	CE	ST
DNN	11.7	10.3
CNN	12.6	10.4
DNN+CNN	11.3	9.6

VGG

# Fmaps	Classic [16, 17, 18]	VB(X)	VC(X)	VD(X)	WD(X)
64		conv(3,64) conv(64,64) pool 1x3	conv(3,64) conv(64,64) pool 1x2	conv(3,64) conv(64,64) pool 1x2	conv(3,64) conv(64,64) pool 1x2
128		conv(64, 128) conv(128, 128) pool 2x2	conv(64, 128) conv(128, 128) pool 2x2	conv(64, 128) conv(128, 128) pool 1x2	conv(64, 128) conv(128, 128) pool 1x2
256			conv(128, 256) conv(256, 256) pool 1x2	conv(128, 256) conv(256, 256) pool 2x2	conv(128, 256) conv(256, 256) conv(256, 256) pool 2x2
512	conv9x9(3,512) pool 1x3 conv3x4(512,512)			conv(256, 512) conv(512, 512) pool 2x2	conv(256, 512) conv(512, 512) conv(512, 512) pool 2x2
	FC 2048 FC 2048 (FC 2048) FC output size				
	Softmax				

VGG



KUR
TOK
CEB
KAZ
TEL
LIT
Softmax
FC
pool
conv
conv
pool
conv
conv
Shared

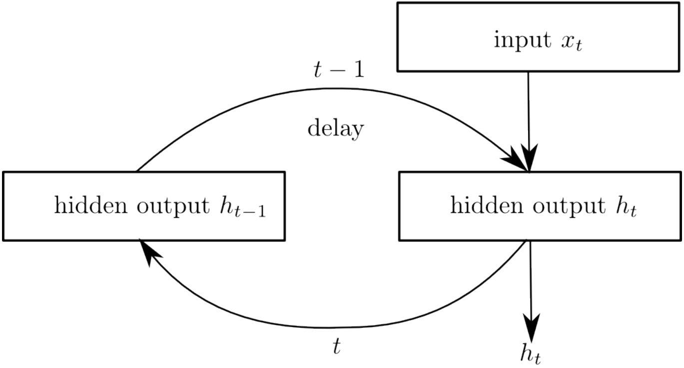
VGG Performance

	WER	# params (M)	#M frames
Classic 512 [17]	13.2	41.2	1200
Classic 256 ReLU (A+S)	13.8	58.7	290
VCX (6 conv) (A+S)	13.1	36.9	290
VDX (8 conv) (A+S)	12.3	38.4	170
WDX (10 conv) (A+S)	12.2	41.3	140
VDX (8 conv) (S)	11.9	38.4	340
WDX (10 conv) (S)	**11.8**	41.3	320

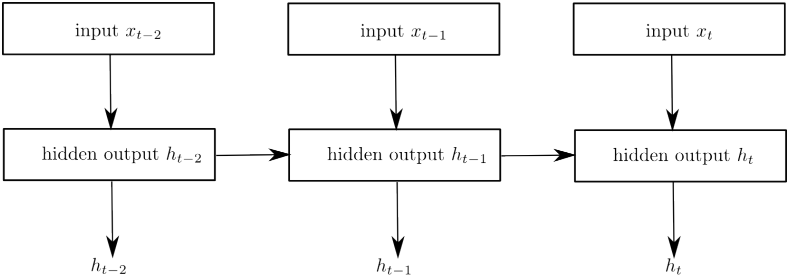
Where Are We?

Recurrent Neural Networks (RNNs)

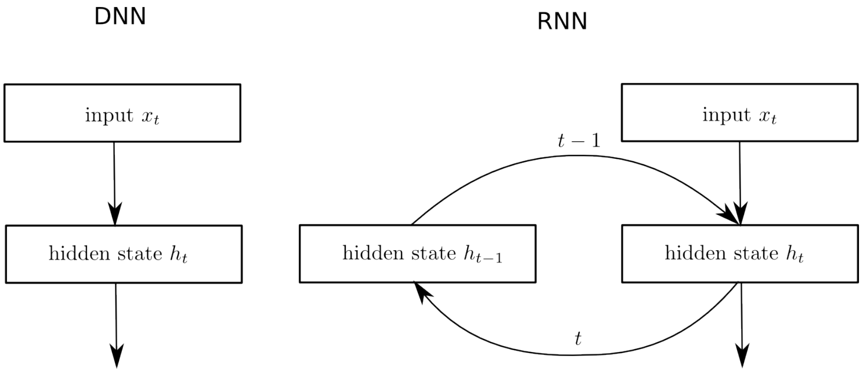
- DNNs are deep in layers.
- RNNs are deep in time (in addition).
- Shared weights and biases across time steps.



Unfolded RNN



DNN vs. RNN


DNN
input x_t
hidden state h_t
RNN
input x_t
$t-1$
hidden state h_{t-1}
hidden state h_t
t

Formal Definition: RNN

- Input vector sequence: $x_t \in \mathbb{R}^D, t = 1, \dots, T$
- Hidden outputs: $h_t, t = 1, \dots, T$
- Free parameters:
  - Input to hidden weight: $W \in \mathbb{R}^{n_{l-1} \times n_l}$
  - Hidden to hidden weight: $R \in \mathbb{R}^{n_l \times n_l}$
  - Bias: $b \in \mathbb{R}^{n_l}$
- Output: Iterate the equation for $t = 1, \dots, T$

$$h_t = \sigma(W \cdot x_t + R \cdot h_{t-1} + b)$$

- Compare with DNN:

$$h_t = \sigma(W \cdot x_t + b)$$

BackPropagation Through Time (BPTT)

- Chain rule through time:

$$\frac{d\,\mathcal{F}(\theta)}{d\,h_t} = \sum_{\tau=1}^{t-1} \frac{d\,\mathcal{F}(\theta)}{d\,h_\tau} \frac{d\,h_\tau}{d\,h_t}$$

$$\frac{\mathrm{d}\,\mathcal{F}(\theta)}{\mathrm{d}\,h_1} \longrightarrow \frac{\mathrm{d}\,\mathcal{F}(\theta)}{\mathrm{d}\,h_1}\frac{\mathrm{d}\,h_1}{\mathrm{d}\,h_2} \longrightarrow \sum_{t=1}^{2} \frac{\mathrm{d}\,\mathcal{F}(\theta)}{\mathrm{d}\,h_t}\frac{\mathrm{d}\,h_t}{\mathrm{d}\,h_3} \longrightarrow \cdots \longrightarrow \sum_{t=1}^{T-1} \frac{\mathrm{d}\,\mathcal{F}(\theta)}{\mathrm{d}\,h_t}\frac{\mathrm{d}\,h_t}{\mathrm{d}\,h_T}$$

BackPropagation Through Time (BPTT)

- Implementation:
  - Unfold RNN over time through $t = 1, \ldots, T$.
  - Forward propagate RNN.
  - Backpropagate error through unfolded network.
- Faster than other optimization methods (e.g. evolutionary search).
- Difficulty with local optima.

Bidirectional RNN (BRNN)

- Forward RNN processes data forward left to right.
- Backward RNN processes data backward right to left.
- Output joins the output of forward and backward RNN.

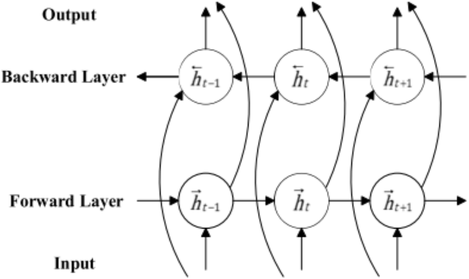


Fig.2. Bidirectional RNN

Formal Definition: BRNN

- Input vector sequence: $x_t \in \mathbb{R}^D, t = 1, \ldots, T$
- Forward and backward hidden outputs: $\overrightarrow{h}_t, \overleftarrow{h}_t, t = 1, \ldots, T$
- Forward and backward free parameters:
  - Input to hidden weight: $\overrightarrow{W}, \overleftarrow{W} \in \mathbb{R}^{n_{l-1} \times n_l}$
  - Hidden to hidden weight: $\overrightarrow{R}, \overleftarrow{R} \in \mathbb{R}^{n_l \times n_l}$
  - Bias: $\overrightarrow{b}, \overleftarrow{b} \in \mathbb{R}^{n_l}$
- Output: Iterate the equation for $t = 1, \ldots, T$

$$\overrightarrow{h}_t = \sigma(\overrightarrow{W} \cdot x_t + \overrightarrow{R} \cdot \overrightarrow{h}_{t-1} + \overrightarrow{b})$$

- Output: Iterate the equation for $t = T, \ldots, 1$

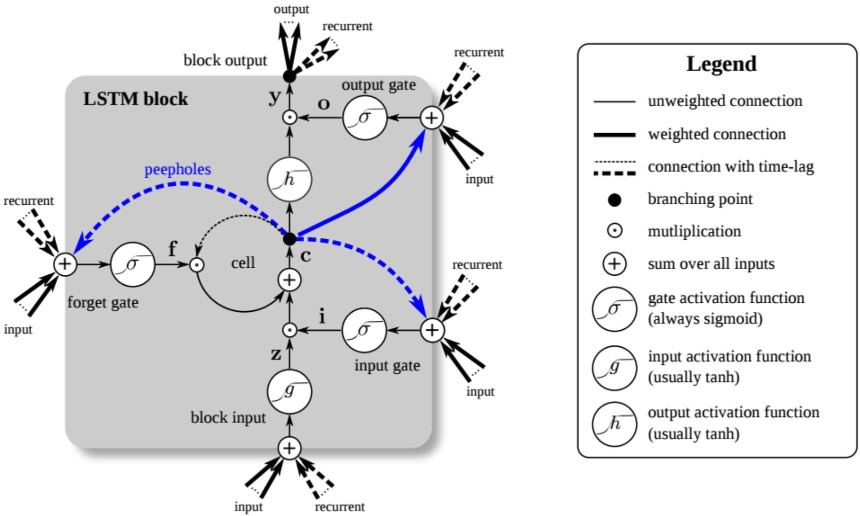
$$\overleftarrow{h}_t = \sigma(\overleftarrow{W} \cdot x_t + \overleftarrow{R} \cdot \overleftarrow{h}_{t+1} + \overleftarrow{b})$$

$\rightarrow \quad \leftarrow$

RNN using Memory Cells

- Equip an RNN with a memory cell.
- Can store information for a long time.
- Introduce gating units to:
  - activations going in,
  - activations going out,
  - saving activations,
  - forgetting activations.

Long Short-Term Memory RNN



Formal Definition: LSTM

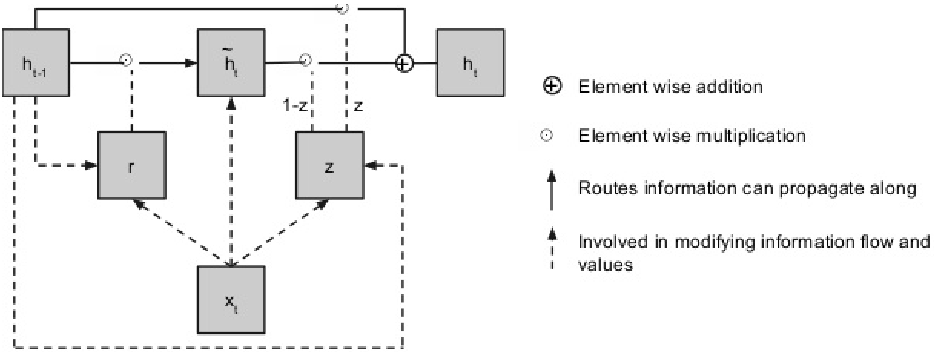
- Input vector sequence: $x_t \in \mathbb{R}^D, t = 1, \ldots, T$
- Hidden outputs: $h_t, t = 1, \ldots, T$
- Iterate the equation for $t = 1, \ldots, T$:

$$
\begin{aligned}
z_t &= \sigma(W_z \cdot x_t + R_z \cdot h_{t-1} + b_z) && \text{(block input)} \\
i_t &= \sigma(W_i \cdot x_t + R_i \cdot h_{t-1} + P_i \odot c_{t-1} + b_i) && \text{(input gate)} \\
f_t &= \sigma(W_f \cdot x_t + R_f \cdot h_{t-1} + P_f \odot c_{t-1} + b_f) && \text{(forget gate)} \\
c_t &= i_t \odot z_t + f_t \odot c_{t-1} && \text{(cell state)} \\
o_t &= \sigma(W_o \cdot x_t + R_o \cdot h_{t-1} + P_o \odot c_t + b_i) && \text{(output gate)} \\
h_t &= o_t \odot \tanh(c_t) && \text{(block output)}
\end{aligned}
$$

LSTM: Too many connections ?

- Some of the connections in the LSTM are not necessary [1].
- Peepholes do not seem to be necessary.
- Coupled input and forget gates.
- Simplified LSTM $\Rightarrow$ Gated Recurrent Unit (GRU).

Gated Recurrent Unit (GRU)



References: [2, 3, 4]

Formal Definition: GRU

- Input vector sequence: $x_t \in \mathbb{R}^D, t = 1, \ldots, T$
- Hidden outputs: $h_t, t = 1, \ldots, T$
- Iterate the equation for $t = 1, \ldots, T$:

$$
\begin{aligned}
r_t &= \sigma(W_r \cdot x_t + R_r \cdot h_{t-1} + b_r) && \text{(reset gate)} \\
z_t &= \sigma(W_z \cdot x_t + R_z \cdot h_{t-1} + b_z) && \text{(update gate)} \\
\overline{h}_t &= \sigma(W_h \cdot x_t + R_h \cdot (r_t \odot h_{t-1}) + b_h) && \text{(candidate gate)} \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \overline{h}_t && \text{(output gate)}
\end{aligned}
$$

CNN vs. DNN vs. RNN: Performance

- GMM, DNN use fMLLR features.
- CNN use log-Mel features which have local structure,
- opposed to speaker normalized features.

Table: Broadcast News 50 h.

	WER [%]	
Model	CE	ST
GMM	18.8	n/a
DNN	16.2	14.9
CNN	15.8	13.9
BGRU (fMLLR)	14.9	n/a
BLSTM (fMLLR)	14.8	n/a
BGRU (Log-Mel)	14.1	n/a

DNN vs. CNN vs. RNN: Performance

- GMM, DNN use fMLLR features.
- CNN use log-Mel features which have local structure,
- opposed to speaker normalized features.

Table: Broadcast Conversation 2000 h.

	WER [%]	
Model	CE	ST
DNN	11.7	10.3
CNN	12.6	10.4
RNN	11.5	9.9
DNN+CNN	11.3	9.6
RNN+CNN	11.2	9.4
DNN+RNN+CNN	11.1	9.4

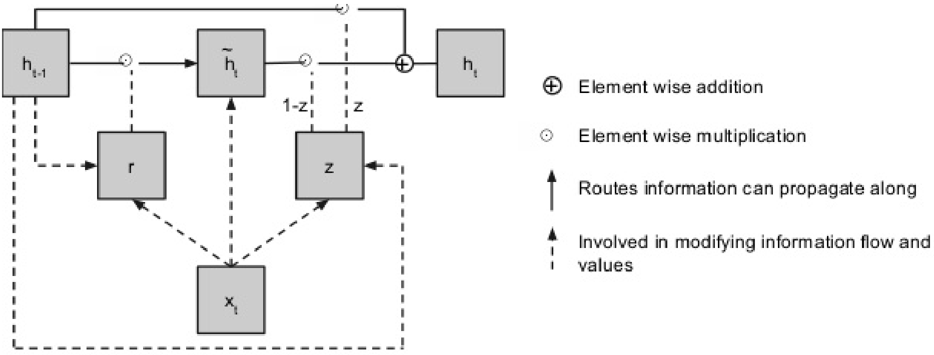
RNN Black Magic

- Unrolling the RNN in training:
  - whole utterance [5],
  - vs. truncated BPTT with carryover [6]:
    - Split utterance into subsequences of e.g. 21 frames.
    - Carry over last cell from previous subsequence to new subsequence.
    - Compose minibatch from subsequences.
  - vs. truncated BPTT with overlap:
    - Split utterance in subsequences of e.g. 21 frames.
    - Overlap subsequences by 10.
    - Compose minibatch of subsequences from different utterances.
- Gradient clipping of the LSTM cell.

RNN Black Magic

- Recognition: Unrolling RNN
  - whole utterance,
  - vs. unrolling subsequences
    - Split utterance in subsequences of e.g. 21 frames.
    - Carry over last cell from previous subsequence to new subsequence.
  - vs. unrolling on spectral window [7]
    - For each frame unroll on the spectral window
    - Last RNN layer only returns center/last frame.

Highway Network



References: [2, 3, 4]

Formal Definition: Highway Network

- Input vector sequence: $x_t \in \mathbb{R}^D, t = 1, \ldots, T$
- Hidden outputs: $h_t, t = 1, \ldots, T$
- Iterate the equation for $t = 1, \ldots, T$:

$$
\begin{aligned}
z_t &= \sigma(W_z \cdot x_t + b_z) && \text{(highway gate)} \\
\overline{h}_t &= \sigma(W_h \cdot x_t + b_h) && \text{(candidate gate)} \\
h_t &= z_t \odot x_t + (1 - z_t) \odot \overline{h}_t && \text{(output gate)}
\end{aligned}
$$

Formal Definition: Highway GRU

- Input vector sequence: $x_t \in \mathbb{R}^D, t = 1, \ldots, T$
- Hidden outputs: $h_t, t = 1, \ldots, T$
- Iterate the equation for $t = 1, \ldots, T$:

$$
\begin{aligned}
r_t &= \sigma(W_r \cdot x_t + R_r \cdot h_{t-1} + b_r) && \text{(reset gate)} \\
z_t &= \sigma(W_z \cdot x_t + R_z \cdot h_{t-1} + b_z) && \text{(update gate)} \\
d_t &= \sigma(W_d \cdot x_t + R_d \cdot h_{t-1} + b_d) && \text{(highway gate)} \\
\overline{h}_t &= \sigma(W_h \cdot x_t + R_h \cdot (r_t \odot h_{t-1}) + b_h) && \text{(candidate gate)} \\
h_t &= d_t \odot x_t + (1 - d_t) \odot (z_t \odot h_{t-1} + (1 - z_t) \odot \overline{h}_t) \\
& && \text{(output gate)}
\end{aligned}
$$

Where Are We?

Unstable Gradient Problem

- Happens in deep as well in recurrent neural networks.
- If gradient becomes very small $\Rightarrow$ vanishing gradient.
- If gradient becomes very large $\Rightarrow$ exploding gradient.
- Simplified Neural Network (w_i are just scalars):

$$
\begin{aligned}
\mathcal{F}(w_1, \ldots, w_N) &= \mathcal{L}(\sigma(y_N)) \\
&= \mathcal{L}(\sigma(w_N \cdot \sigma(y_{N-1})) \\
&= \mathcal{L}(\sigma(w_N \cdot \sigma(w_{N-1} \cdot \ldots \sigma(w_1 \cdot x_t) \ldots)))
\end{aligned}
$$

Unstable Gradient Problem, Constraints and Regularization

- Gradient:

$$\frac{\mathrm{d}\,\mathcal{F}(w_1,\ldots,w_N)}{\mathrm{d}\,w_1} = \frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,\theta}\cdot\frac{\mathrm{d}\,\sigma(w_N\cdot\sigma(w_{N-1}\cdot\ldots\sigma(w_1\cdot x_t)\ldots))}{\mathrm{d}\,w_1} = \frac{\mathrm{d}\,\mathcal{L}}{\mathrm{d}\,w_1}\cdot\sigma'(y_N)\cdot w_N\cdot\sigma'(y_{N-1})\cdot w_{N-1}\cdot\ldots\sigma'(w_1)\cdot x_t$$

- If $|w_i\sigma'(y_i)| < 1, i = 2,\ldots,N \Rightarrow$ gradient vanishes.
- If $|w_i\sigma'(y_i)| >> 1, i = 2,\ldots,N \Rightarrow$ gradient explodes.

Solution: Unstable Gradient Problem

- Gradient Clipping.
- Weight constraints.
- Let the network save activations over layers/time steps:

$$y_{\text{new}} = \alpha y_{\text{previous}} + (1 - \alpha) y_{\text{common}}$$

  - Long Short-Term Memory RNN
  - Highway Neural Network (>100 layers)

Gradient Clipping

- Keeps gradient weights in range.
- One approach to deal with the exploding gradient problem.
- Ensure gradient is in the range $[-c, c]$ for a constant c:

$$\mathrm{clip}\left(\frac{\mathrm{d}\,\mathcal{F}}{\mathrm{d}\,\theta}, c\right) = \min\left(c, \max\left(-c, \frac{\mathrm{d}\,\mathcal{F}}{\mathrm{d}\,\theta}\right)\right)$$

Constraints (I)

- Keeps weights in range (for e.g. Relu, Maxout).
- Ignored for gradient backpropagation.
- Constraints are forced after gradient update.

Constraints (II)

- Max-Norm: force $\|W\|_2 \leq c$ for constant c

$$\|W\|_{\max} = W \cdot \frac{\max(\min(\|W\|_2, 0), c)}{\|W\|_2}$$

- Unity-Norm: force $\|W\|_2 \leq 1$

$$\|W\|_{\text{unity}} = \frac{W}{\|W\|_2}$$

- Positivity-Norm: force $W > 0$

$$\|W\|_{+} = W \cdot \max(0, W)$$

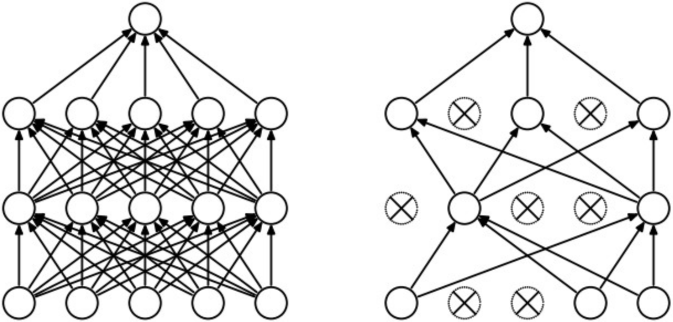
Regularization: Dropout

- Dropout:

- Prevents getting stuck in local optimum $\Rightarrow$ avoids overfitting.

Regularization: Dropout

- Dropout:



(a) Standard Neural Net (b) After applying dropout.

- Prevents getting stuck in local optimum $\Rightarrow$ avoids overfitting.

Regularization: Dropout

- Input vector sequence: $x_t \in \mathbb{R}^D$
- Choose $z_t \in \{0, 1\}^D$ for $t = 1, \dots, T$
- According to Bernoulli distribution $P(z_{t,d} = i) = p^{1-i}(1 - p)^i$ with dropout probability with $p \in [0, 1]$:
- Training: $x_t := x_t \odot \frac{z_t}{1-p}$ for $t = 1, \dots, T$.
- Recognition: $x_t := x_t$ for $t = 1, \dots, T$.

Regularization (II)

- L_p Norm:

$$\|\theta\|_p = \left(\sum_{j=0}^{|\theta|} |\theta|^p \right)^{\frac{1}{p}}$$

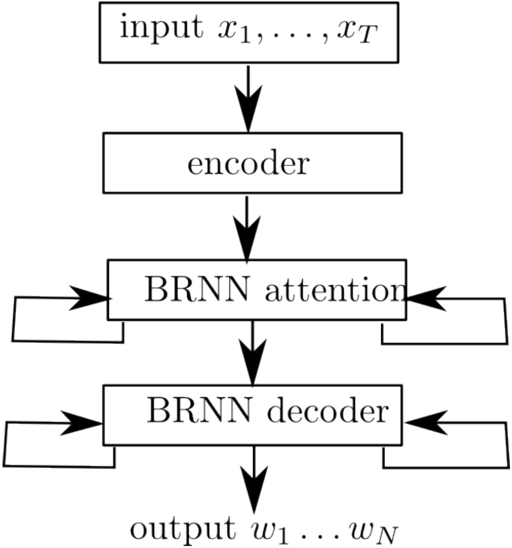
- Training criterion regularization:

$$\mathcal{F}_p(\theta) = \mathcal{F}(\theta) + \lambda \|\theta\|_p \qquad \text{with scalar } \lambda$$

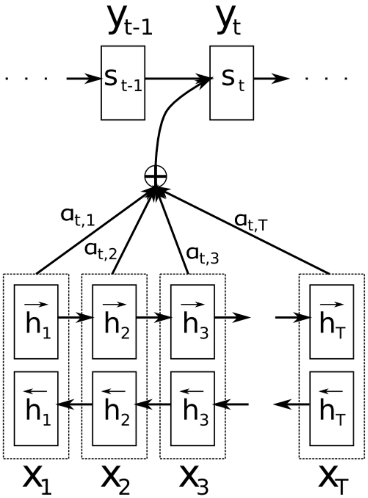
- Smoothes the training criterion.
- Pushes the free parameter weights closer to zero.

Where Are We?

Attention-based End-to-End Architecture



Attention model



Formal Definition: Content Focus

- Input vector sequence: $x_t \in \mathbb{R}^D, t = 1, \ldots, T$
- Hidden outputs: $h_t, t = 1, \ldots, T$
- Scorer:

$$\epsilon_{m,t} = \tanh(V_{m,\epsilon} \cdot x_t + b_\epsilon) \text{ for } t = 1, \ldots, T, m = 1, \ldots, M$$

- Generator:

$$\alpha_{m,t} = \frac{\sigma(W_\alpha \cdot \epsilon_{m,t})}{\sum_{\tau=1}^{T} \sigma(W_\alpha \cdot \epsilon_{m,\tau})} \text{ for } t = 1, \ldots, T, m = 1, \ldots, M$$

- Glimpse:

$$g_m = \sum_{t=1}^{T} \alpha_{m,t} x_t \qquad \text{for } m = 1, \ldots, M$$

- Output:

$$h_m = \sigma(W_h \cdot g_m + b_h) \qquad \text{for } m = 1, \ldots, M$$

Formal Definition: Recurrent Attention

- Scorer:

$$\epsilon_{m,t} = \tanh(W_\epsilon \cdot x_t + R_\epsilon \cdot s_{m-1} + U_\epsilon \cdot (F_\epsilon * \alpha_{m-1}) + b_\epsilon)$$

- Generator:

$$\alpha_{m,t} = \frac{\sigma(W_\alpha \cdot \epsilon_{m,t})}{\sum_{\tau=1}^{T} \sigma(W_\epsilon \cdot \epsilon_{m,\tau})} \text{ for } t = 1, \ldots, T, m = 1, \ldots, M$$

- Glimpse:

$$g_m = \sum_{t=1}^{T} \alpha_{m,t} x_t \text{ for } m = 1, \ldots, M$$

- GRU state:

$$s_m = GRU(g_m, h_m, s_{m-1}) \text{ for } m = 1, \ldots, M$$

- Output:

$$h_m = \sigma(W_h \cdot g_m + R_h \cdot s_{m-1} + b_h) \text{ for } m = 1, \ldots, M$$

End-to-End Performance

Table: TIMIT

	WER [%]	
Model	dev	eval
HMM	13.9	16.7
End-to-end	15.8	17.6
RNN Transducer	n/a	17.7

- K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, "LSTM: A search space odyssey," *CoRR*, vol. abs/1503.04069, 2015. [Online]. Available: http://arxiv.org/abs/1503.04069

- K. Cho, B. Van Merriënboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder–decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734. [Online]. Available: http://www.aclweb.org/anthology/D14-1179

- J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: http://arxiv.org/abs/1412.3555

- R. Józefowicz, W. Zaremba, and I. Sutskever, “An empirical exploration of recurrent network architectures,” in *ICML*, ser. JMLR Proceedings, vol. 37. JMLR.org, 2015, pp. 2342–2350.
- A. Graves, N. Jaitly, and A. Mohamed, “Hybrid speech recognition with deep bidirectional LSTM,” in *ASRU*. IEEE, 2013, pp. 273–278.
- H. Sak, A. W. Senior, and F. Beaufays, “Long short-term memory recurrent neural network architectures for large scale acoustic modeling,” in *INTERSPEECH*. ISCA, 2014, pp. 338–342.
- A.-R. Mohamed, F. Seide, D. Yu, J. Droppo, A. Stolcke, G. Zweig, and G. Penn, “Deep bi-directional recurrent networks over spectral windows,” in *Proc. IEEE Automatic Speech Recognition and Understanding Workshop*. IEEE Institute of Electrical and Electronics Engineers, December

2015, pp. 78–83. [Online]. Available:
http://research.microsoft.com/apps/pubs/default.aspx?id=259236