Zoneout

Regularizing RNNs by randomly preserving hidden activations
Regularizing RNNs by randomly preserving hidden activations
Regularizing RNNs by randomly preserving hidden activations
Zone out

Regularizing RNNs by randomly preserving hidden activations
Zoneout

Regularizing RNNs by randomly preserving hidden activations
1. The basic idea
2. RNNs/LSTMs
3. How/why it works
4. It works!
Structure of the talk

1. The basic idea
2. RNNs/LSTMs
3. How/why it works
4. It works!
Structure of the talk

1. The basic idea
2. RNNs/LSTMs
3. How/why it works
4. It works!
Structure of the talk

1. The basic idea
2. RNNs/LSTMs
3. How/why it works
4. It works!
Basic idea

Have a random probability of keeping your hidden state (stochastically introduce identity connections between timesteps)
Basic idea

Have a **random** probability of keeping your hidden state (stochastically introduce identity connections between timesteps)
Basic idea

Have a random probability of keeping your hidden state (stochastically introduce identity connections between timesteps)
Basic idea

Have a random probability of keeping your hidden state (stochastically introduce identity connections between timesteps)
Recurrent neural networks

diagram from Chris Olah
1-layer RNN

diagram from Chris Olah
1-layer RNN with zoneout

modified from Chris Olah
1-layer LSTM

diagram from Chris Olah
1-layer LSTM with zoneout

modified from Chris Olah
1-layer LSTM with zoneout

modified from Chris Olah
Implementing zoneout

Dropout:

$$\mathcal{T}_t = d_t \odot \tilde{\mathcal{T}}_t + (1 - d_t) \odot 0$$

Zoneout:

$$\mathcal{T}_t = d_t \odot \tilde{\mathcal{T}}_t + (1 - d_t) \odot 1$$
Implementing zoneout

**Dropout:**

\[ T_t = d_t \odot \tilde{T}_t + (1 - d_t) \odot 0 \]

**Zoneout:**

\[ T_t = d_t \odot \tilde{T}_t + (1 - d_t) \odot 1 \]
Implementing zoneout

Dropout:
\[ T_t = d_t \odot \tilde{T}_t + (1 - d_t) \odot 0 \]

Zoneout:
\[ T_t = d_t \odot \tilde{T}_t + (1 - d_t) \odot 1 \]
Implementing zoneout

# sample masks, pass as inputs to network
zoneouts_states = np.random.binomial(n=1, p=(z_states),
                                      size=(T, B, N))
zoneouts_cells = np.random.binomial(n=1, p=(z_cells),
                                     size=(T, B, N))

# inside step function of LSTM after computing h and c
h = h_prev * zoneouts_states + (1 - zoneouts_states) * h

# inside step function of LSTM after computing h and c
h = h_prev * zoneouts_states + (1 - zoneouts_states) * h

Implementing zoneout

```python
# sample masks, pass as inputs to network
zoneouts_states = np.random.binomial(n=1, p=z_states),
                 size=(T, B, N))
zoneouts_cells  = np.random.binomial(n=1, p=z_cells),
                 size=(T, B, N))

# inside step function of LSTM after computing h and c
h = h_prev * zoneouts_states + (1 - zoneouts_states) * h
```
Implementing zoneout

# sample masks, pass as inputs to network
zoneouts_states = np.random.binomial(n=1, p=(z_states),
                                      size=(T, B, N))
zoneouts_cells = np.random.binomial(n=1, p=(z_cells),
                                     size=(T, B, N))

# inside step function of LSTM after computing h and c
h = h_prev * zoneouts_states + (1 - zoneouts_states) * h
h = c_prev * zoneouts_cells + (1 - zoneouts_cells) * c
Implementing zoneout

# sample masks, pass as inputs to network
zoneouts_states = np.random.binomial(n=1, p=(z_states),
        size=(T, B, N))
zoneouts_cells = np.random.binomial(n=1, p=(z_cells),
        size=(T, B, N))

# inside step function of LSTM after computing h and c
h = h_prev * zoneouts_states + (1 - zoneouts_states) * h  
c = c_prev * zoneouts_cells + (1 - zoneouts_cells) * c
Implementing zoneout

# sample masks, pass as inputs to network
zoneouts_states = np.random.binomial(n=1, p=(z_states),
                                     size=(T, B, N))
zoneouts_cells = np.random.binomial(n=1, p=(z_cells),
                                     size=(T, B, N))

# inside step function of LSTM after computing h and c
h = h_prev * zoneouts_states + (1 - zoneouts_states) * h
C = c_prev * zoneouts_cells + (1 - zoneouts_cells) * c
Zoneout trains a pseudo-ensemble

**Pseudo-ensemble:** a (possibly infinite) collection of *child models* spawned from a *parent model* by perturbing it according to some noise process.

Philip Bachman, Ouais Alsharif, Doina Precup. NIPS 2014
Zoneout as per-unit stochastic depth

Stochastic depth: per minibatch, randomly drop a subset of layers and replace with identity

Gao Huang*, Yu Sun*, Zhuang Liu, Daniel Sedra, Kilian Weinberger. CVPR 2016
Zoneout as per-unit stochastic depth

Stochastic depth: per minibatch, randomly drop a subset of layers and replace with identity

Gao Huang*, Yu Sun*, Zhuang Liu, Daniel Sedra, Kilian Weinberger. CVPR 2016

Zoneout: in RNNs, layer = whole timestep. Per-unit works better.
Other related work

**Dropout** - Hinton et al. 2013
**Fast dropout** in RNNs - Bayer et al. 2013; Wang & Manning 2013
**Dropout on non-recurrent** connections in RNNs - Pham et al. 2013; Zaremba et al. 2014

**Variational RNN** (drop columns of weights) - Gal 2015
**rnnDrop** (same mask at every timestep) - Moon et al. 2015
**Recurrent dropout** (on input gate) - Semeniuta et al. 2016
**Residual networks** (add identity skip connections in feedforward nets) - He et al. 2015
Zoneout helps propagate gradients
Permuted sequential MNIST

Error Rate vs Epochs for different models:
- Vanilla LSTM (Train)
- Vanilla LSTM (Validation)
- Zoneout $z_c = 0.15, z_h = 0.15$ (Train)
- Zoneout $z_c = 0.15, z_h = 0.15$ (Validation)
- Recurrent dropout $z = 0.15$ (Train)
- Recurrent dropout $z = 0.15$ (Validation)
### Permuted sequential MNIST

<table>
<thead>
<tr>
<th>Model</th>
<th>% Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unregularized LSTM</td>
<td>10</td>
</tr>
<tr>
<td>Recurrent batch normalization*</td>
<td>4.6</td>
</tr>
<tr>
<td>Zoneout (cells=states=0.15)</td>
<td>6.9</td>
</tr>
<tr>
<td>Zoneout + recurrent batch normalization*</td>
<td>4.1</td>
</tr>
</tbody>
</table>

*Cooijmans et al. 2016*
## Permutated sequential MNIST

<table>
<thead>
<tr>
<th>Model</th>
<th>% Error rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unregularized LSTM</td>
<td>10</td>
</tr>
<tr>
<td>Recurrent batch normalization*</td>
<td>4.6</td>
</tr>
<tr>
<td>Zoneout (cells=states=0.15)</td>
<td>6.9</td>
</tr>
<tr>
<td>Zoneout + recurrent batch normalization*</td>
<td>4.1</td>
</tr>
</tbody>
</table>

*Cooijmans et al. 2016*
Character-level Penn Treebank

![Graph showing the relationship between bits per character and epochs for different values of \(z_c\) and \(z_h\).]
Character-level Penn Treebank

![Graph showing the performance of various techniques over epochs]

- Zoneout
- Weight noise
- Norm stabilizer
- Vanilla LSTM
- Recurrent dropout
- Stochastic depth
Character-level Penn Treebank

![Graph showing the performance of different models over epochs.]

- Unregularized LSTM (training)
- Unregularized LSTM (validation)
- Recurrent dropout (training)
- Recurrent dropout (validation)
- Zoneout (training)
- Zoneout (validation)
## Character-level Penn Treebank

<table>
<thead>
<tr>
<th>Model</th>
<th>BPC (entropy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unregularized LSTM</td>
<td>1.36</td>
</tr>
<tr>
<td>Stochastic depth</td>
<td>1.343</td>
</tr>
<tr>
<td>Weight noise</td>
<td>1.344</td>
</tr>
<tr>
<td>Norm stabilizer</td>
<td>1.352</td>
</tr>
<tr>
<td>Recurrent dropout</td>
<td>1.334</td>
</tr>
<tr>
<td>Recurrent batch norm</td>
<td>1.32</td>
</tr>
<tr>
<td>Zoneout</td>
<td>1.29</td>
</tr>
</tbody>
</table>
## Character-level Penn Treebank

<table>
<thead>
<tr>
<th>Model</th>
<th>BPC (entropy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unregularized LSTM</td>
<td>1.36</td>
</tr>
<tr>
<td>Stochastic depth</td>
<td>1.343</td>
</tr>
<tr>
<td>Weight noise</td>
<td>1.344</td>
</tr>
<tr>
<td>Norm stabilizer</td>
<td>1.352</td>
</tr>
<tr>
<td>Recurrent dropout</td>
<td>1.334</td>
</tr>
<tr>
<td>Recurrent batch norm</td>
<td>1.32</td>
</tr>
<tr>
<td>Zoneout</td>
<td><strong>1.29</strong></td>
</tr>
</tbody>
</table>
# Character-level Penn Treebank

<table>
<thead>
<tr>
<th>Model</th>
<th>BPC (entropy)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unregularized LSTM</td>
<td>1.36</td>
</tr>
<tr>
<td>Stochastic depth</td>
<td>1.343</td>
</tr>
<tr>
<td>Weight noise</td>
<td>1.344</td>
</tr>
<tr>
<td>Norm stabilizer</td>
<td>1.352</td>
</tr>
<tr>
<td>Recurrent dropout</td>
<td>1.334</td>
</tr>
<tr>
<td>Recurrent batch norm</td>
<td>1.32</td>
</tr>
<tr>
<td>Zoneout</td>
<td>1.27</td>
</tr>
</tbody>
</table>

Trained on overlapping input data (after Cooijmans et al. 2016)
Word-level Penn Treebank

![Perplexity vs Epochs graph showing different models and their performance. The graph compares Recurrent dropout, Zoneout with different parameters, and Vanilla LSTM.]
Word-level Penn Treebank

<table>
<thead>
<tr>
<th>Model</th>
<th>Validation Perplexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unregularized LSTM</td>
<td>145.4</td>
</tr>
<tr>
<td>Stochastic depth</td>
<td>129.9</td>
</tr>
<tr>
<td>Weight noise</td>
<td>172.0</td>
</tr>
<tr>
<td>Norm stabilizer</td>
<td>141.8</td>
</tr>
<tr>
<td>Recurrent dropout</td>
<td>119.9</td>
</tr>
<tr>
<td>Zoneout</td>
<td>115.2</td>
</tr>
</tbody>
</table>
Thank you!

Questions?
Regularizing RNNs by randomly preserving hidden activations

David Krueger*, Tegan Maharaj*, Janos Kramar*, Mohammad Pezeshki, Nicolas Ballas, Rosemary Nan Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, Chris Pal

arxiv.org/pdf/1606.01305v2.pdf  github.com/teganmaharaj/zoneout