Zoneout

Regularizing RNNs by randomly preserving hidden activations

Zone ut

Regularizing RNNs by randomly preserving hidden activations

Zone ut

Regularizing RNNs by randomly preserving hidden activations

Zoneout

Regularizing RNNs by randomly preserving hidden activations

Zoneout

Regularizing RNNs by randomly preserving hidden activations

Zoneout

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

* equal authors

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!

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

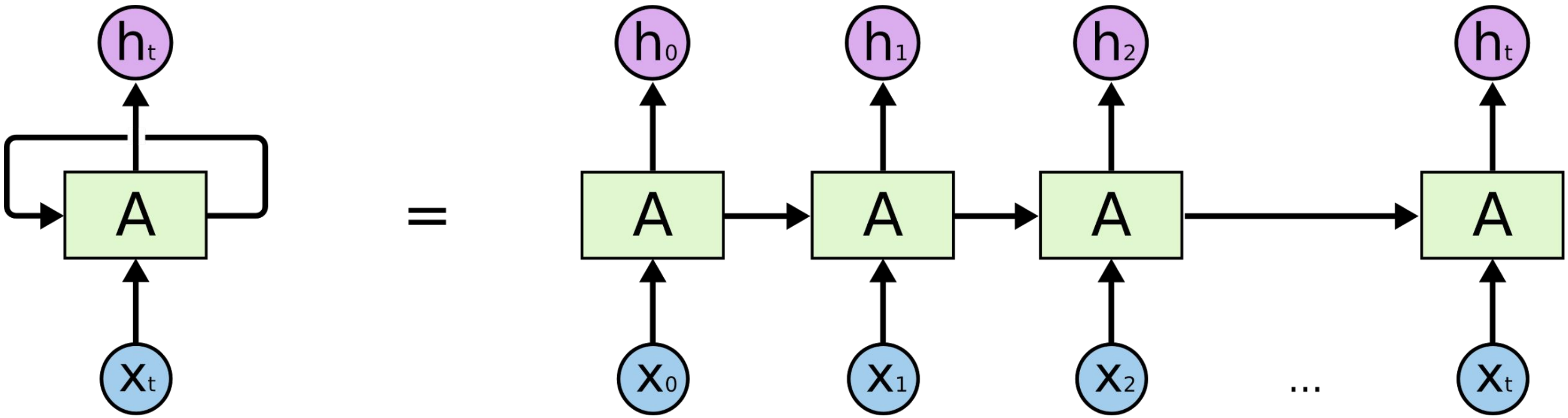


diagram from Chris Olah

1-layer RNN

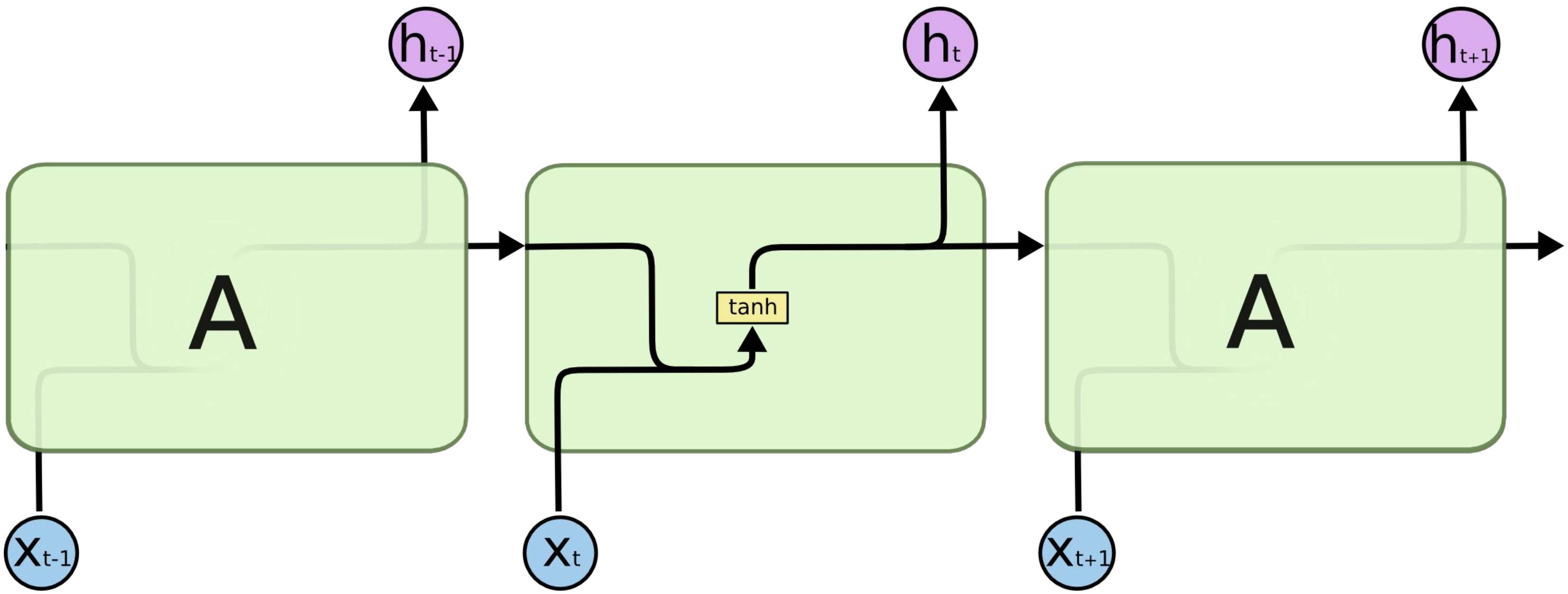
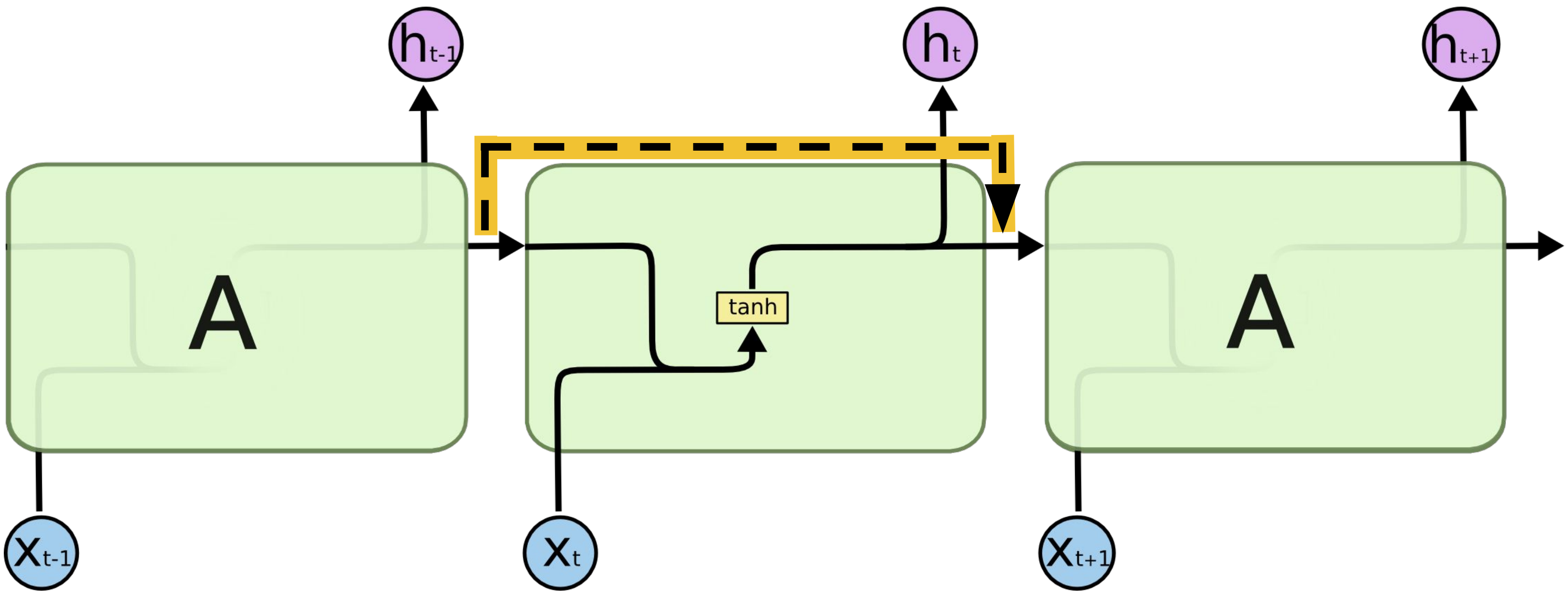


diagram from Chris Olah

1-layer RNN with zoneout



modified from Chris Olah

1-layer LSTM

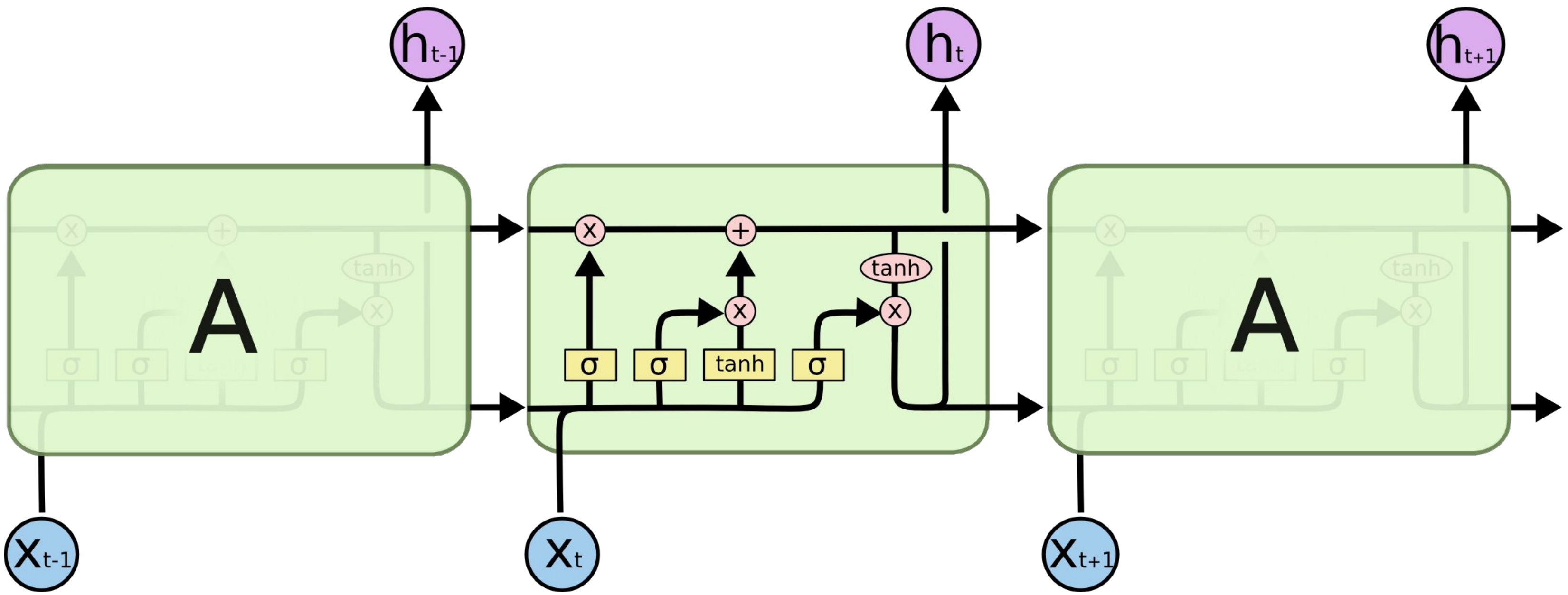
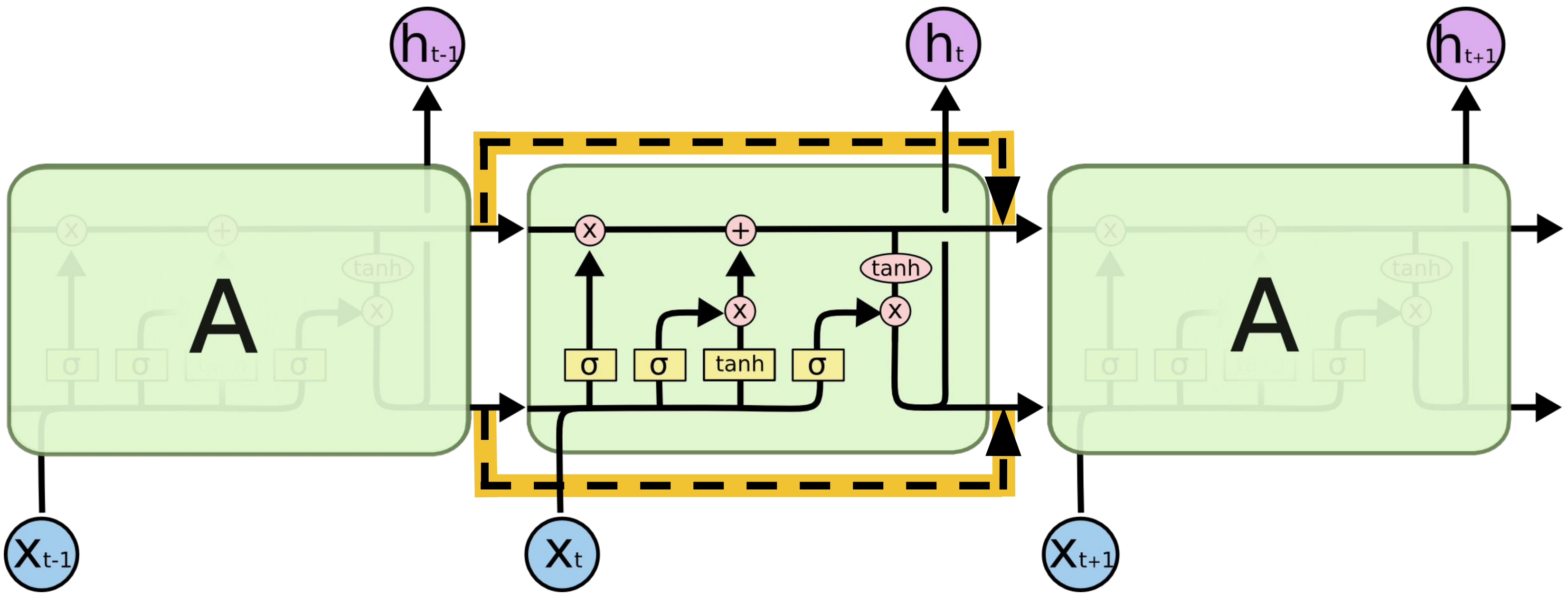


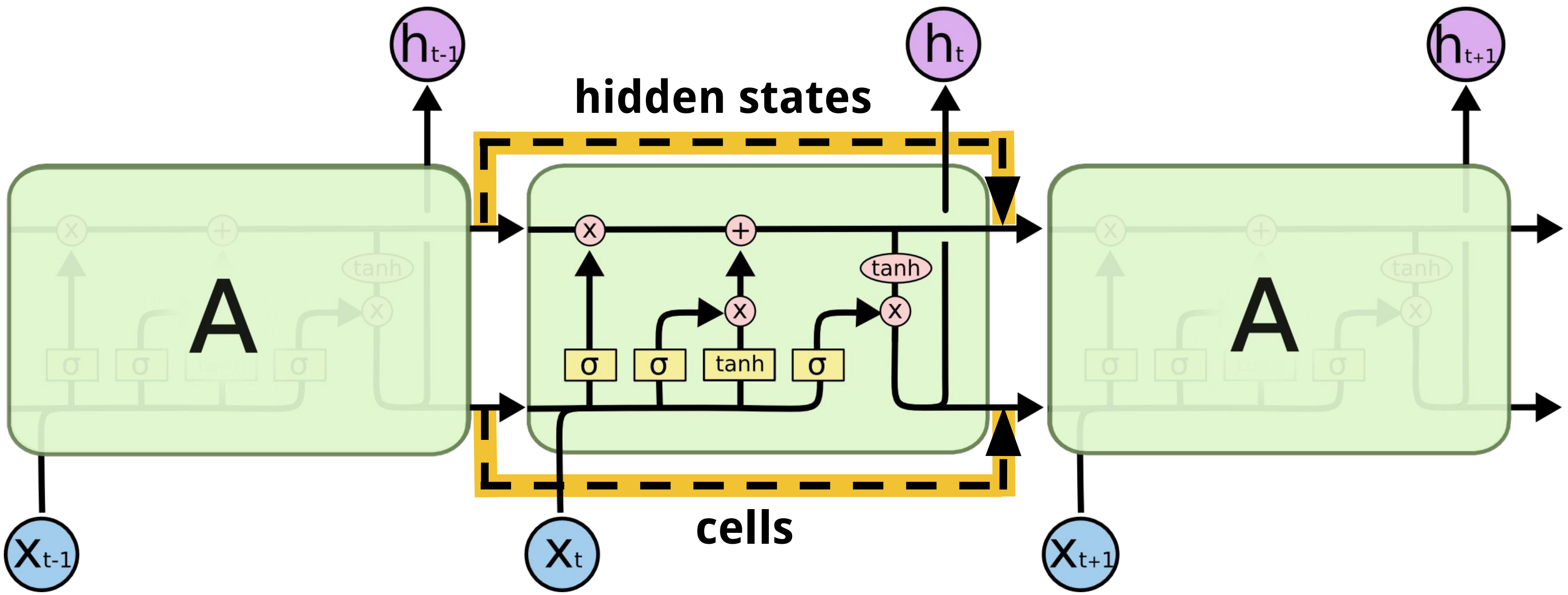
diagram from Chris Olah

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:

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

Zoneout:

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

Implementing zoneout

Dropout:

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

Zoneout:

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

Implementing zoneout

```
# 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
```

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
```

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

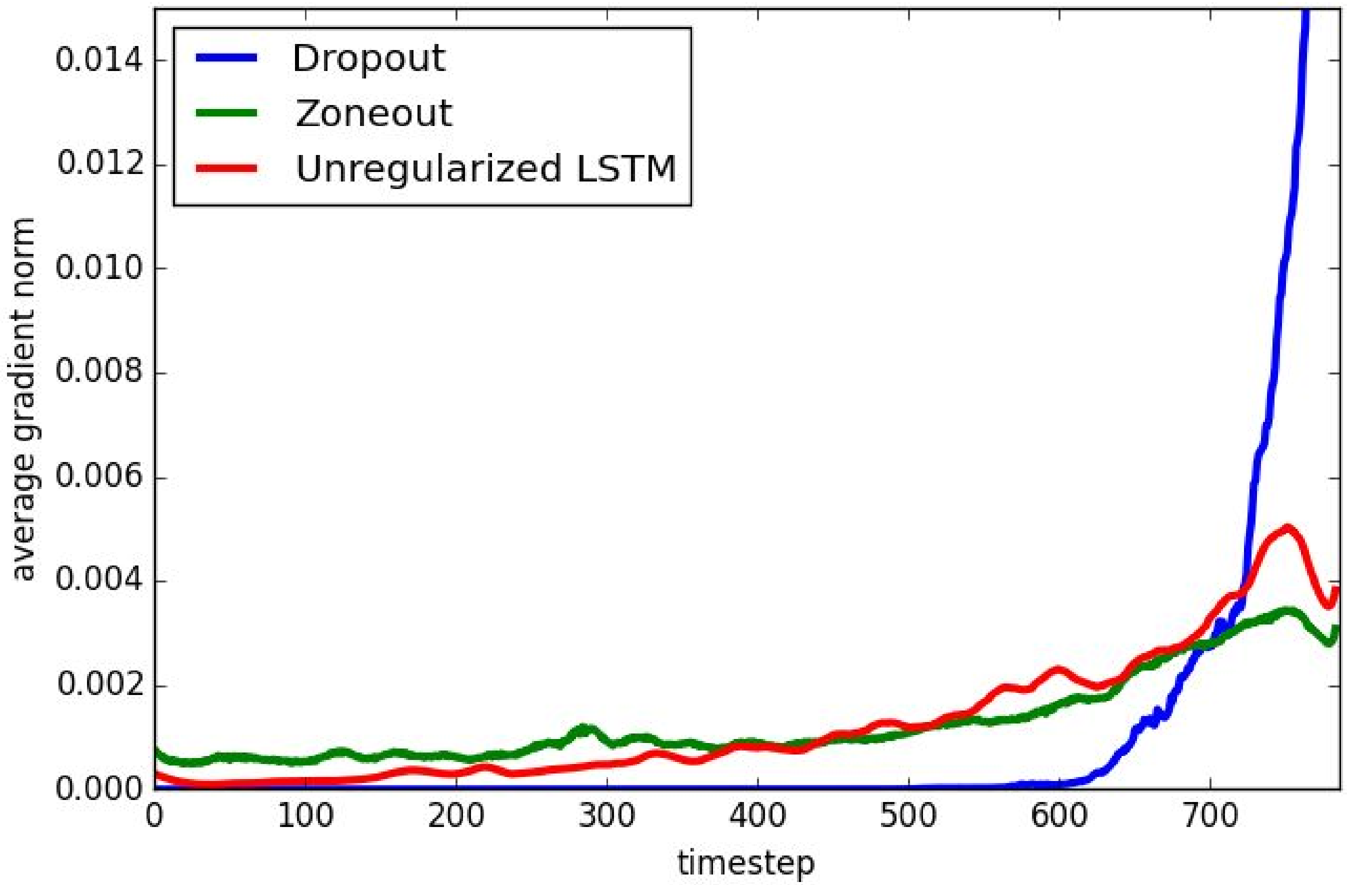
Gao Huang*, Yu Sun*, Zhuang Liu, Daniel Sedra, Kilian Weinberger. CVPR 2016

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

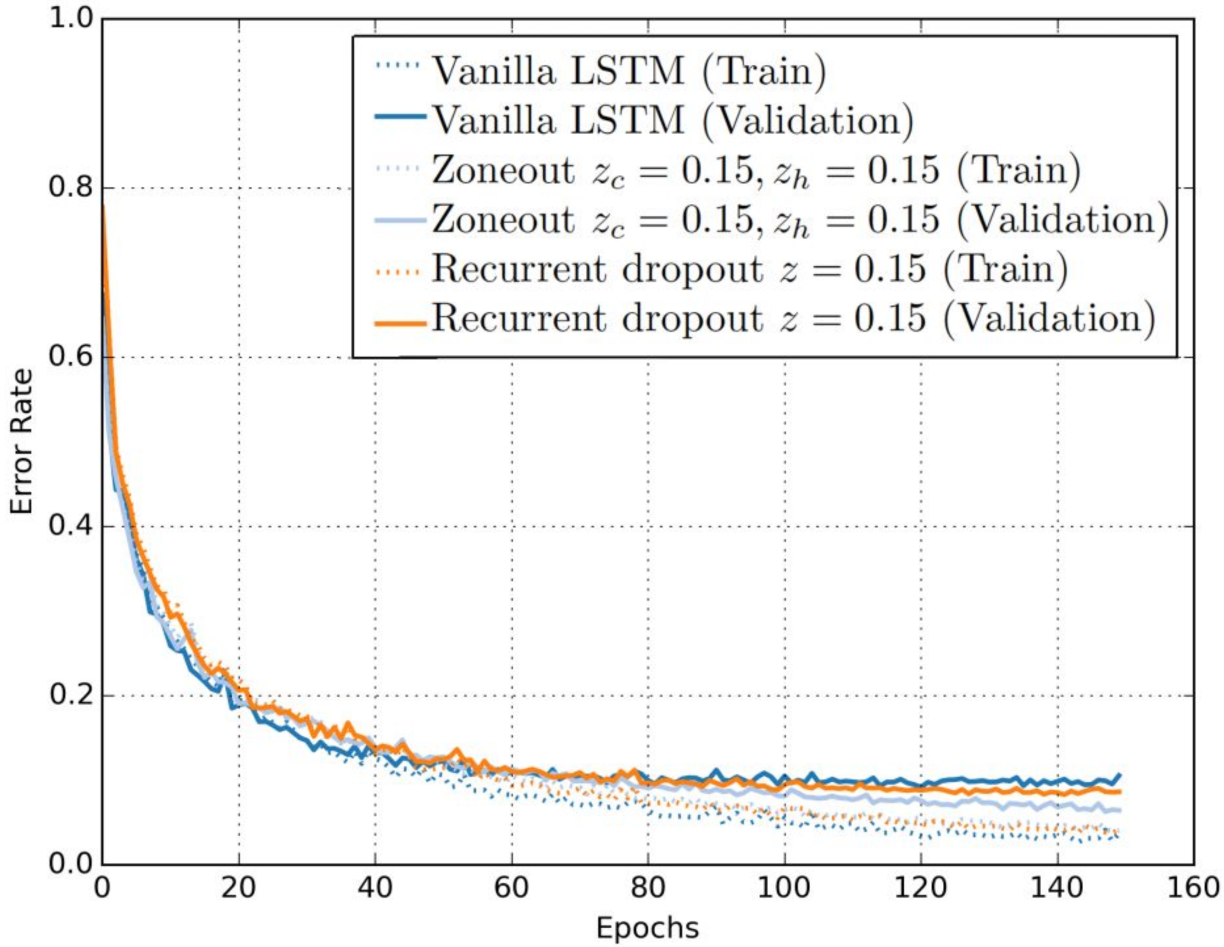
Other related work

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



Permuted sequential MNIST

Model	% Error rate
Unregularized LSTM	10
Recurrent batch normalization*	4.6
Zoneout (cells=states=0.15)	6.9
Zoneout + recurrent batch normalization*	4.1

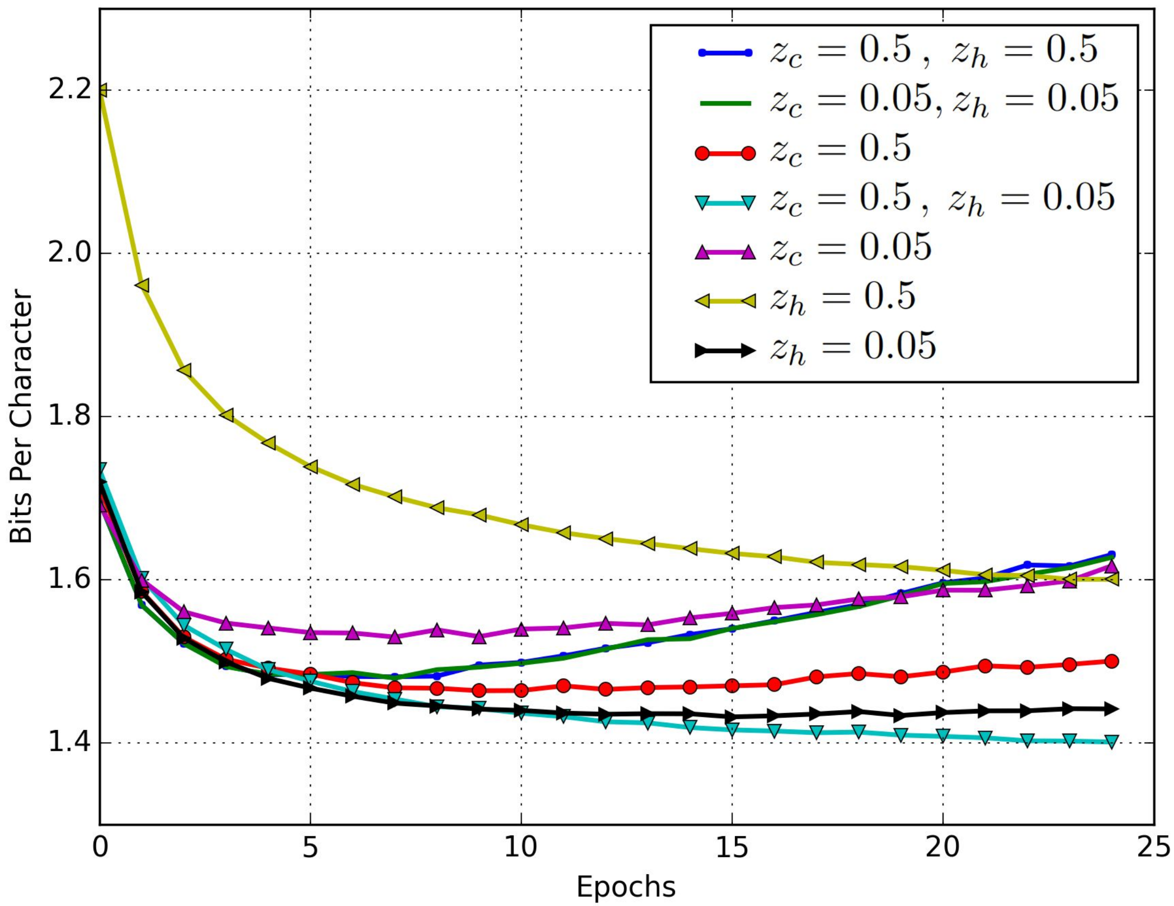
*Cooijmans et al. 2016

Permuted sequential MNIST

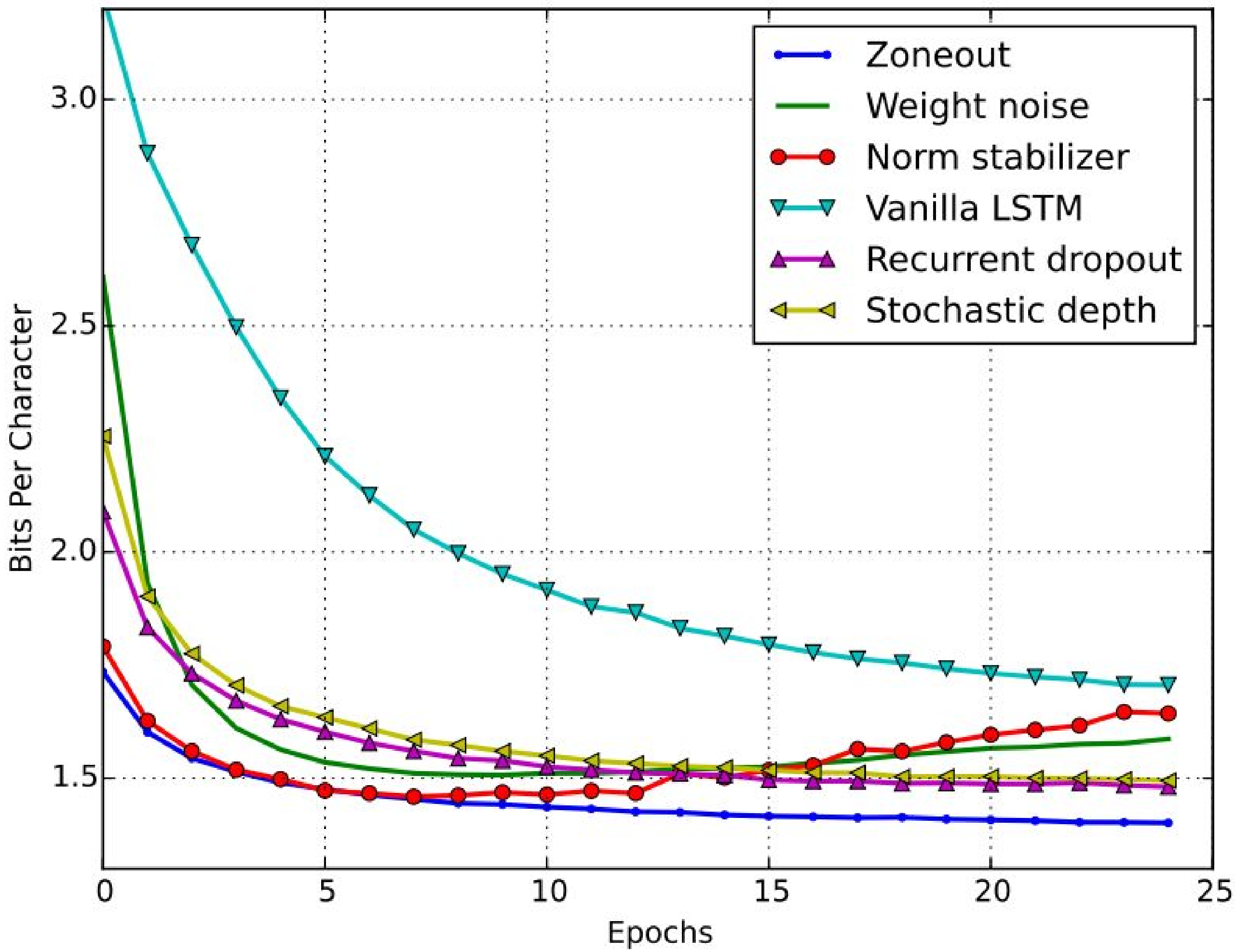
Model	% Error rate
Unregularized LSTM	10
Recurrent batch normalization*	4.6
Zoneout (cells=states=0.15)	6.9
Zoneout + recurrent batch normalization*	4.1

*Cooijmans et al. 2016

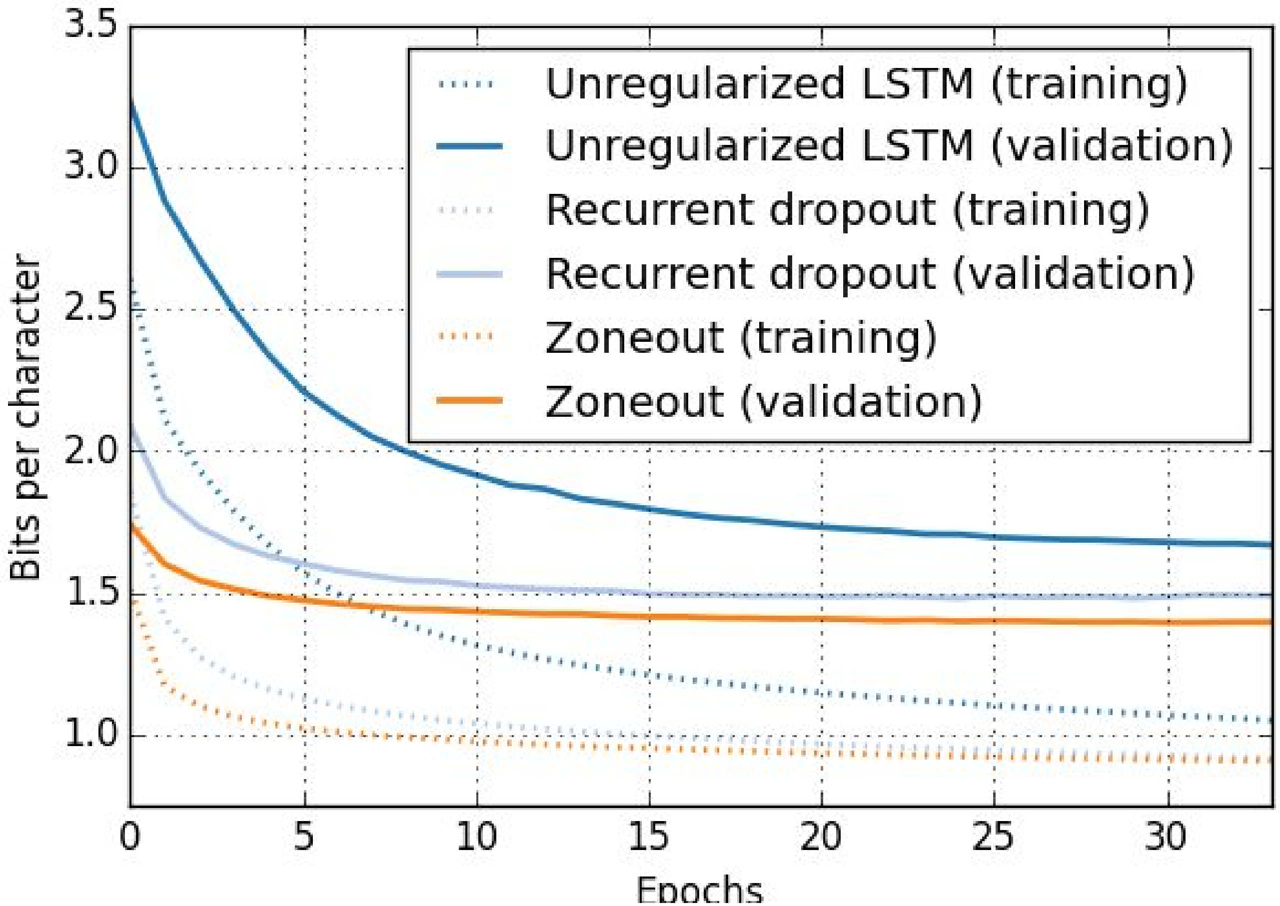
Character-level Penn Treebank



Character-level Penn Treebank



Character-level Penn Treebank



Character-level Penn Treebank

Model	BPC (entropy)
Unregularized LSTM	1.36
Stochastic depth	1.343
Weight noise	1.344
Norm stabilizer	1.352
Recurrent dropout	1.334
Recurrent batch norm	1.32
Zoneout	1.29

Character-level Penn Treebank

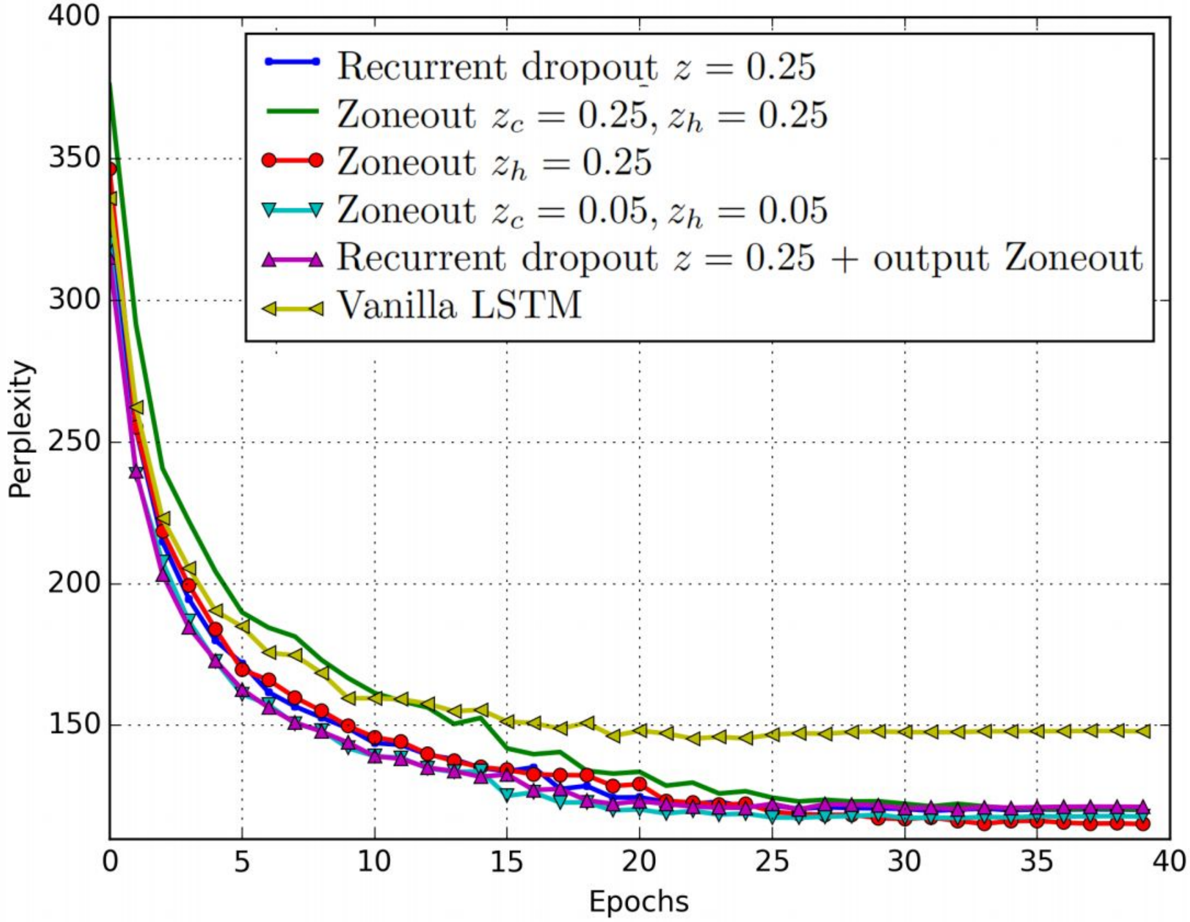
Model	BPC (entropy)
Unregularized LSTM	1.36
Stochastic depth	1.343
Weight noise	1.344
Norm stabilizer	1.352
Recurrent dropout	1.334
Recurrent batch norm	1.32
Zoneout	1.29

Character-level Penn Treebank

Model	BPC (entropy)
Unregularized LSTM	1.36
Stochastic depth	1.343
Weight noise	1.344
Norm stabilizer	1.352
Recurrent dropout	1.334
Recurrent batch norm	1.32
Zoneout	1.27

Trained on overlapping input data (after Cooijmans et al. 2016)

Word-level Penn Treebank



Word-level Penn Treebank

Model	Validation Perplexity
Unregularized LSTM	145.4
Stochastic depth	129.9
Weight noise	172.0
Norm stabilizer	141.8
Recurrent dropout	119.9
Zoneout	115.2

Thank you!

Questions?

Zoneout

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