Highway Networks

(Srivastava, Greff, and Schmidhuber 2015)

Standard NN layer: $y = H(x, W_H)$ where H is a non-linear transformation parametrized by weights W_H

Highway network:

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot C(x, W_C)$$

where T is the *transform* gate and C is the *carry* gate, which define the ratio in which the output is defined by transforming the input in contrast to carrying it over. For simplicity, $C = 1 - T$, producing:

$$y = H(x, W_H) \cdot T(x, W_T) + x \cdot (1 - T(x, W_T))$$

Note: this formulation, where each layer can propagate its input x further requires that all of the elements have the same dimension (y, x, T, H). An option here is to use padding to upscale x or sub-sampling, in order to reduce the dimensionality. An option is also to use a regular layer (without the highway connections) to change the dimensionality, and then continue with the highway layers.

Recurrent Highway Networks

(Zilly et al. 2016)

The paper sketches out the proof of the vanishing/exploding gradient problem and isolates its relation to the largest singular value of the recurrent weight matrix:

$$|A| \leq \left|R^T\right| \left|diag[f'(Ry^{[t-1]})]\right| \leq \gamma\sigma_{max}$$

The gradient vanishes when $\gamma\sigma_{max} < 1$ and explodes when the expression is larger than one.

γ is the maximum value the gradient of the activation function.

Geršgorin circle theorem states:

$$spec(A) \subset \bigcup_{i \in \{1,\ldots,n\}} \left\{ \lambda \in \mathbb{C} \left\| \lambda - a_{ii} \right\|_{\mathbb{C}} \leq \sum_{j=1, j \neq i}^{n} |a_{ij}| \right\}$$

translated, the spectrum of eigenvelues of the square matrix $A \in \mathbb{R}^{n \times n}$ lies within the union of complex circles which are centered around the **diagonal values** of the matrix A, with a radius equal to the sum of the absolute values of the **non-diagonal** entries of each row.

Essentially, this means that shifting the diagonal values shifts the center of the circles, and therefore the possible location of the eigenvalues. Also, increasing the values of the remaining elements increases the radius in which the eigenvalues are contained.

TODO: Add viz of theorem w/TikZ

Initialization of recurrent weights as mentioned in (Le, Jaitly, and Hinton 2015), one way to circumvent this via initialization is to initialize the recurrent matrix as an identity matrix and the remainder as small random values. However, this method does nothing to mitigate the fact that the values of the matrix will change during training, resulting in the same exploding / vanishing gradient phenomenon.

Essentially, a reformulation of a RNN in the form of a vertical highway network is used (more or less equal to LSTM, where the previous cell state is propagated input).

Takeaways:

- most of the transform-processing is done in the first layer, and then to a lesser extent (first layer contextually transforms the input features? and the remaining layers use this contextual information).
- passing the input along in a resnet-like or highway-like fashion is useful.
- Geršgorin can help limit the range of singular values of a matrix.

Learning long term dependencies Dataset: pixel-by-pixel MNIST image classification, introduced in (Le, Jaitly, and Hinton 2015)

Learning long-term dependencies in RNNs with Auxilliary Losses

(Trinh et al. 2018)

- Randomly sample one or multiple anchor positions
- Use an unsupervised auxilliary loss
- Reconstruction auxilliary loss (reconstruct a subsequence given first symbol, enhances remembering)
- Prediction auxilliary loss (predict a future token in language-model fashion)
- Trained in two phases:
- Pure unsupervised pretraining (minimize auxilliary loss)
- Semi-supervised learning where $min_\theta L_{sup}(\theta) + L_{aux}(\theta')$

Hyperparameters: - How frequently to sample the reconstructon segments, and how long they are

The methods help with learning long-term dependencies, even when the backprop is truncated. Essentially, signal is needed for the network to remember things. That signal can't be achieved through very long backprop due to failure of credit assignment.

Additional ablation study & result analysis in paper.

Maxout networks

(Goodfellow et al. 2013)

Maxout networks use the *maxout* function as the activation. For an input $x \in \mathbb{R}^d$ the maxout is:

$$h_i(x) = max_{j \in [1,k]} z_{ij}$$

where $z_{ij} = x^T W_{...ij} + b_{ij}$, $W \in \mathbb{R}^{d \times m \times k}$ and $b \in \mathbb{R}^{m \times k}$ are the learned model parameters.

Essentially: instead of projecting into the output dimension m, project into $m \times k$ and max over the k additional dimensions. Pytorch impl: link

Grid LSTM

(Kalchbrenner, Danihelka, and Graves 2015)

Similar to Multi-dimensional Recurrent Neural Networks (Graves and Schmidhuber 2009)

LSTM along each dimension of network (depth, T). The vertical LSTM hidden / cell states initialized by the inputs.

N-dimensional Grid LSTM accepts N hidden vectors $h_1, \ldots, h_N$ and N memory vectors $m_1, \ldots, m_N$, which are all distinct for each dimension.

All of the hidden states are then concatenated:

$$H = \begin{bmatrix} \hat{h}_1 \\ \vdots \\ \hat{h}_N \end{bmatrix} \tag{1}$$

The N-dimensional block then computes N LSTM tranfsorms, one for each dimension. Each LSTM transform has its individual weight matrices. Each block accepts input hidden and memory vectors from N dimensions, and outputs them into N dimensions.

$$\begin{aligned}(\hat{h}_1, \hat{m}_1) &= LSTM(H, m_1, W_1) \\ &\dots \\ (\hat{h}_N, \hat{m}_N) &= LSTM(H, m_N, W_N)\end{aligned} \tag{2}$$

CONT (postponed)

Learning to Skim Text

(Yu, Lee, and Le 2017) : LSTM-Jump

- Uses a policy gradient method to make decisions to skip a number of tokens in the input
- Hyperparam: max jump size K, number of tokens to read before jumping R
- Processing stops if
  - Jumping softmax predicts 0
  - Jump exceeds sequence length N
  - The network processed all tokens
- The last hidden state is used for prediction in downstream tasks

REINFORCE procedure (+Baselines):

Objective:

- Minimize cross-entropy error (classification loss)
- Maximize expected reward under the current jumping policy (R = -1 for misclassification, +1 for correct)
- Baselines regularization term: minimize difference between actual reward and predicted baseline

Neural Speed Reading via Skim-RNN

(Seo et al. 2017)

OpenReview discussion: link

Update just a part of the hidden state for irrelevant words (uses a smaller RNN)

The hard decision (whether the word is important or not) isn't differentiable – the gradient is approximated by Gumbel-Softmax instead of REINFORCE

(policy gradient). The method results in a reduced number of FLOPs (floating point operations).

"skimming achieves higher accuracy compared to skipping the tokens, implying that paying some attention to unimportant tokens is better than completely ignoring (skipping) them"

RNNs with hard decisions – last paragraph of chapter 2, references

Model:

- Two RNNs, default (big), skim (small)
- Binary hard decision of skimming is a stochastic multinomial variable over the probabilities of a single layer NN which accepts the next token and current hidden state
- During inference, instead of sampling, the greedy argmax of the multinomial parameters is used

$$h_t = \begin{cases} f(x_t, h_{t-1}), & \text{if } Q_t = 1 \\ [f'(x_t, h_{t-1}); h_{t-1}[d'+1:d]], & \text{if } Q_t = 2 \end{cases}$$

where $Q_t = 1$ means the network chose to fully read and $Q_t = 2$ means the network has decided to skim.The dimension k of the multinomial distribution is 2.

Gumbel-softmax:

Expected loss over the sequence of skim-read decisions is:

$$\mathbb{E}_{Q_t \sim Multinomial(p_t)}\left[L(\sigma)\right] = \sum_Q L(\sigma; Q)P(Q) = \sum_Q L(\sigma; Q) \prod_j p_j^{Q_j}$$

to approximate the expected loss, all of the hard decisions Q_t need to be enumerated and evaluated, which is intractable. One approximation of the gradient is by using REINFORCE which while unbiased has high variance. A replacemnt is to use the gumbel-softmax distribution (Jang, Gu, and Poole 2016).

The reparametrization constructs a softmax over the probabilities of the multinomial distribution with an added Gumbel noise:

$$r_t^i = \frac{exp((log(p_t^i) + g_t^i)/\tau)}{\sum_j exp((log(p_t^j) + g_t^j)/\tau)}$$

where τ is a temperature hyperparameter, and g_t^i is an independent sample from $Gumbel(0,1) = -log(-log(Uniform(0,1)))$. There are two distinct r^i's, one for skim and one for fully read.

To encourage the model to *skim when possible*, a regularization term is added which minimizes the mean of the negative log probability of skimming $\frac{1}{T}\sum log(p_t^2)$:

$$L'(\sigma) = L(\sigma) + \gamma \frac{1}{T} \sum -log(p_t^2)$$

Discussion on openreview:

Similar to: (Jernite et al. 2016)

Variable Computation in Recurrent Neural Networks

(Jernite et al. 2016)

Variable Computation GRU and Variable Computation RNN (VCGRU, VCRNN)

At each timestep t, the *scheduler* takes the current hidden and input vectors and decides on the number of dimensions to use for the update (d). The first d dimensions of the **hidden state and input vector (!)** are then used to compute the first d elements of the new hidden state, while the rest is carried over from the previus state.

Scheduler: function $m : \mathbb{R}^{2D} \rightarrow [0, 1]$ decides which portion of the hidden state to change. For each timestep t:

$$m_t = \sigma(u \cdot h_{t-1} + v \cdot x_t + b)$$

The first $\lceil m_t D \rceil$ dimensions are then the ones updated. In the lower-dimensional recurrent unit, the upper left sub-square matrices of shape $d \times d$ are used.

Soft masking: the decision to update only a subset of a state is essentially a hard chouce and makes the model non-differentianle. The hard choice is approximated by using a gating function which applies a soft mask.

The gating vector e_t is defined by:

$$\forall i \in 1, \ldots, D, (e_t)_i = \text{Thres}_\epsilon(\sigma(\lambda(m_t D - i)))$$

Where λ is a *sharpness parameter*, and Thres maps all values greater than $1 - \epsilon$ and smaller than ϵ to 1 and 0.

Learning when to skim and when to read

(Johansen and Socher 2017)

- Use a simple BOW model to predict first
- Use a LSTM model if the BOW model is either not secure (1) or a decision network decides to use it (2)

1. **Confidence based strategy**: if the probability of prediction of the BOW model is not larger than a threshold τ, use the larger LSTM model.
2. **Decision network**: train a network to predict, given an input sample, whether the BOW misclassified it and the LSTM classified it correctly. Labelled via the confusion matrix from the trained BOW and LSTM models.

Saves computation time, beter accuracy than baseline.

Skip RNN: Learning to Skip State Updates in Recurrent Neural Networks

(Campos et al. 2017)

OpenReview discussion: link

Official blogpost: link

Pytorch implementation: link

Add a *binary state update gate* $u_t \in \{0, 1\}$ to a RNN network, which selects whether the state of the RNN is updated ($u_t = 1$) or copied from the previous timestep ($u_t = 0$). (Highway-network like updates)

$$u_t = f_{\text{binarize}}(\hat{u}_t)$$

Where $\hat{u}$ is the probability of each outcome, and f_{binarize} is a function that maps $[0, 1] \rightarrow \{0, 1\}$

$$s_t = u_t \cdot S(s_{t-1}, x_t) + (1 - u_t) \cdot s_{t-1}$$

Where S is the recurrent transition model (cell).

$$\Delta\hat{u}_t = \sigma(W_p s_t + b_p)$$

$\Delta\hat{u}$ is an intermediate value of $\hat{u}$.

$$\hat{u}_{t+1} = u_t \cdot \Delta\hat{u}_t + (1 - u_t) \cdot (\hat{u}_t + min(\Delta\hat{u}_t, 1 - \hat{u}_t))$$

"The model formulation encodes the observation that the likelihood of requesting a new input to update the state increases with the number of consecutively skipped samples." The longer a state is not updated, the larger the probability that it will be updated ($1 - u_t$ part). On state update ($u_t = 1$) the accumulated value is flushed.

Gradient of binarize fn: Straight-through (biased) estimator:

$$\frac{\partial f_{\text{binarize}}(x)}{\partial x} = 1$$

Annotation Artifacts in Natural Language Inference Data

(Gururangan et al. 2018)

Classification model on just the *hypothesis* of NLI achieves 67% on SNLI and 53% on MultiNLI.

"Negation and vagueness are highly correlated with ceratin inference classes. Our findings suggest that the success of natural language inference models to date has been overestimated, and that the task remains a hard open problem." ->

- entailed hypotheses contain gender-neutral references to people
- purpose clauses are a sign of neutral hypotheses
- negation correlates with contradiction

Annotation artifacts: patterns in the data that occur as a result of the framing of the annotation task influencing the language used by the crowd workers.

Discussion: "Many datasets contain annotation artifacts..." – references to other examples of this phenomenon

Controlling Decoding for More Abstractive Summaries with Copy-Based Networks

Reference not yet on Google Scholar: https://arxiv.org/pdf/1803.07038.pdf

Cites (See, Liu, and Manning 2017) as reporting that "at test time, the [copy/generate] distribution is heavily skewed towards copying" (while this does not hold for train time) – without teacher forcing, the algorithm mostly copies

Copy-controlled decoding: add a penalty to beam search n order to push the mixture coefficient (copy vs generate) towards the intended ratio.

$$s(y_{\leq t}, X) = \sum_{t'=1}^{t} \underbrace{logp(y_{t'}|y_t, X)}_{\text{beam search}} - \underbrace{\eta_t max(0, m^* - \hat{m}_v)}_{\text{new penalty}}$$

Where m^{star} is a target coefficient, η_t a time-varying penalty strength and $\hat{m}_t = \frac{1}{t'} \sum_{t''=1} t' m_{t''}$ the average mixure component (higher means generate more, lower means sample more).

On the Importance of Single Directions for Generalization

(Morcos et al. 2018)

- Networks that overfit and memorize data are more dependent on single directions (activation of a single unit / feature map).
- Interpretable neurons / units are not more or less important than uninterpretable.
- Batchnorm effectively decreases class selectivity of feature maps and makes them more robust towards overfitting / underfitting.

All experiments use ReLU nonlinearities and are trained on MNIST, CIFAR-10 and ImageNet.

Class-conditional mean activity = mean value when an image from a certain class is input

$$selectivity = \frac{\mu_{max} - \mu_{-max}}{\mu_{max} + \mu_{-max}}$$

Selectivity of 0 means that the average activity of a unit is the same across classes, while 1 means that the unit only fires for a single class.

On the State of the Art of Evaluation in Neural Language Models

(Melis, Dyer, and Blunsom 2017)

OpenReview discussion: link

"Standard LSTM architectures, when properly regularized, outperform more recent models."

Comparisons on Penn Treebank and Wikitext-2 with Recurrent Highway Networks (Zilly et al. 2016) and NAS (Zoph and Le 2016) as baselines with a vanilla LSTM (Hochreiter and Schmidhuber 1997).

"For the recurrent states, all architectures use either variational dropout (Gal & Ghahramani, 2016, state dropout) or recurrent dropout (Semeniuta et al., 2016), unless explicitly noted otherwise."

Hyperparameters optimized with Google Vizier (Golovin et al. 2017): learning rate, input embedding ratio, input dropout, state dropout, output dropout, weight decay (L2) and for deep LSTMs: intra-layer dropout.

Effect of individual featues:

- **Down-projection** reduces ppl by 2-5 points
- **Tied embeddings** improve ppl by 6 points
- **Variational dropout** helps a lot
- **Recurrent dropout** performs on par with variational dropout

Figure 2.: input dropout, output dropout and state dropout help when relatively high ([0.6 ~ 0.8]), intra layer dropout helps when medium ([0.2 ~ 0.4])

Regularizing and Optimizing LSTM Language Models

(Merity, Keskar, and Socher 2017)

OpenReview discussion link

"Given the over-parameterization of neural networks, generalization performance crucially relies on the ability to regularize the models sufficiently. Strategies such as dropout (Srivastava et al., 2014) and batch normalization (Ioffe & Szegedy, 2015) have found great success and are now ubiquitous in feed-forward and convolutional neural networks."

Awesome overview of regularization in RNNs in introduction.

Regularization strategies:

- Weight-dropped LSTM: DropConnect mask on H2H weights
- Randomized length BPTT
- Embedding dropout
- Activation regularization
- Temporal activation regularization

Optimizer choice: Adam (generally), SGD (word-level LM), avSGD (Averaged SGD)

AvSGD: *"AvSGD carries out iterations similar to SGD, but instead of returning the last iterate as the solution, returns an average of the iterates past a certain, tuned, threshold T. This threshold T is typically tuned and has a direct impact on the performance of the method. We propose a variant of AvSGD where T is determined on the fly through a non-monotonic criterion and show that it achieves better training outcomes compared to SGD."* (Polyak and Juditsky 1992)

Classic SGD:

$$w_{k+1} = w_k - \gamma_k \hat{\nabla} f(w_k)$$

where $\hat{\nabla}$ is a stochastic gradient computed over a minibatch, and k the current iteration. References to various useful SGD analysis papers.

Averaged SGD:

params: learning rate, λ - decay term, *alpha* - power for η (lr) update, t_0 - point (iteration) at which to start averaging (default pytorch: $1e6$) and weight decay - L2 reg strength.

Take standard SGD steps, after some time point T start remembering weights multiplied by μ. The final weight configuration is not the last iterate but the (weighted) average of the last $T_total - T$ iterates.

Regularizers

- Variational dropout: (sample one dropout mask, use it everywheres) for inputs and outputs of LSTMs
- DropConnect: (sample dropout mask for every batch) for hidden to hidden transitions
- Embedding dropout: remove some words **completely** with p_ϵ, scale others by $\frac{1}{1-p_\epsilon}$
- Weight tying: input and output embedding spaces hare parameters
- Activation regularization: regularize activations that are significantly larger than 0

$$AR = \alpha L_2(m \odot h_t)$$

Where m is the dropout mask applied to the outputs, h_t are the inputs to the nonlinearity and α is a scaling parameter.

- Temporal activation regularization: penalize differences between subsequent hidden states (slowness regularizer)

$$TAR = \beta L_2(h_t - h_{t-1})$$

References

Campos, Víctor, Brendan Jou, Xavier Giró-i-Nieto, Jordi Torres, and Shih-Fu Chang. 2017. "Skip Rnn: Learning to Skip State Updates in Recurrent Neural Networks." *arXiv Preprint arXiv:1708.06834*.

Golovin, Daniel, Benjamin Solnik, Subhodeep Moitra, Greg Kochanski, John Karro, and D Sculley. 2017. "Google Vizier: A Service for Black-Box Optimization." In *Proceedings of the 23rd Acm Sigkdd International Conference on Knowledge Discovery and Data Mining*, 1487–95. ACM.

Goodfellow, Ian J, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. 2013. "Maxout Networks." *arXiv Preprint arXiv:1302.4389*.

Graves, Alex, and Jürgen Schmidhuber. 2009. "Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks." In *Advances in Neural Information Processing Systems*, 545–52.

Gururangan, Suchin, Swabha Swayamdipta, Omer Levy, Roy Schwartz, Samuel R Bowman, and Noah A Smith. 2018. "Annotation Artifacts in Natural Language Inference Data." *arXiv Preprint arXiv:1803.02324*.

Hochreiter, Sepp, and Jürgen Schmidhuber. 1997. "Long Short-Term Memory." *Neural Computation* 9 (8). MIT Press: 1735–80.

Jang, Eric, Shixiang Gu, and Ben Poole. 2016. "Categorical Reparameterization with Gumbel-Softmax." *arXiv Preprint arXiv:1611.01144*.

Jernite, Yacine, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. "Variable Computation in Recurrent Neural Networks." *arXiv Preprint arXiv:1611.06188*.

Johansen, Alexander Rosenberg, and Richard Socher. 2017. "Learning When to Skim and When to Read." *arXiv Preprint arXiv:1712.05483*.

Kalchbrenner, Nal, Ivo Danihelka, and Alex Graves. 2015. "Grid Long Short-Term Memory." *arXiv Preprint arXiv:1507.01526*.

Le, Quoc V, Navdeep Jaitly, and Geoffrey E Hinton. 2015. "A Simple Way to Initialize Recurrent Networks of Rectified Linear Units." *arXiv Preprint arXiv:1504.00941*.

Melis, Gábor, Chris Dyer, and Phil Blunsom. 2017. "On the State of the Art of Evaluation in Neural Language Models." *arXiv Preprint arXiv:1707.05589*.

Merity, Stephen, Nitish Shirish Keskar, and Richard Socher. 2017. "Regularizing and Optimizing Lstm Language Models." *arXiv Preprint arXiv:1708.02182*.

Morcos, Ari, David GT Barrett, Neil C Rabinowitz, and Matthew Botvinick. 2018. "On the Importance of Single Directions for Generalization." In *Proceeding*

of the International Conference on Learning Representations.

Polyak, Boris T, and Anatoli B Juditsky. 1992. "Acceleration of Stochastic Approximation by Averaging." *SIAM Journal on Control and Optimization* 30 (4). SIAM: 838–55.

See, Abigail, Peter J Liu, and Christopher D Manning. 2017. "Get to the Point: Summarization with Pointer-Generator Networks." *arXiv Preprint arXiv:1704.04368*.

Seo, Minjoon, Sewon Min, Ali Farhadi, and Hannaneh Hajishirzi. 2017. "Neural Speed Reading via Skim-Rnn." *arXiv Preprint arXiv:1711.02085*.

Srivastava, Rupesh Kumar, Klaus Greff, and Jürgen Schmidhuber. 2015. "Highway Networks." *arXiv Preprint arXiv:1505.00387*.

Trinh, Trieu H, Andrew M Dai, Minh-Thang Luong, and Quoc V Le. 2018. "Learning Longer-Term Dependencies in Rnns with Auxiliary Losses."

Yu, Adams Wei, Hongrae Lee, and Quoc V Le. 2017. "Learning to Skim Text." *arXiv Preprint arXiv:1704.06877*.

Zilly, Julian Georg, Rupesh Kumar Srivastava, Jan Koutník, and Jürgen Schmidhuber. 2016. "Recurrent Highway Networks." *arXiv Preprint arXiv:1607.03474*.

Zoph, Barret, and Quoc V Le. 2016. "Neural Architecture Search with Reinforcement Learning." *arXiv Preprint arXiv:1611.01578*.