Deep Learning for Text 1

Applied Text Mining

Ayoub Bagheri

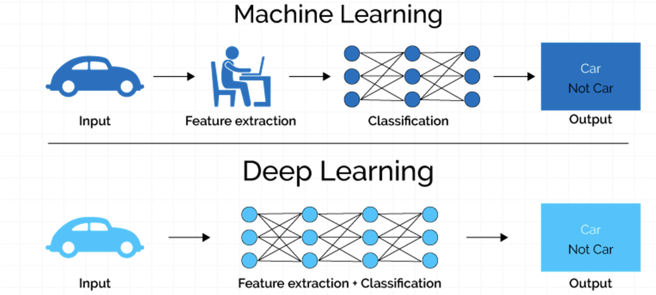
Lecture's plan

1. Feed-forward neural networks
2. Recurrent neural networks
    2.1 SRN
    2.2 LSTM
    2.3 Bi-LSTM
    2.4 GRU

What is Deep Learning (DL) ?

A machine learning subfield of learning representations of data. Exceptional effective at learning patterns.

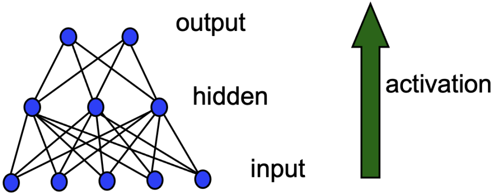
Deep learning algorithms attempt to learn (multiple levels of) representation by using a hierarchy of multiple layers.



https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

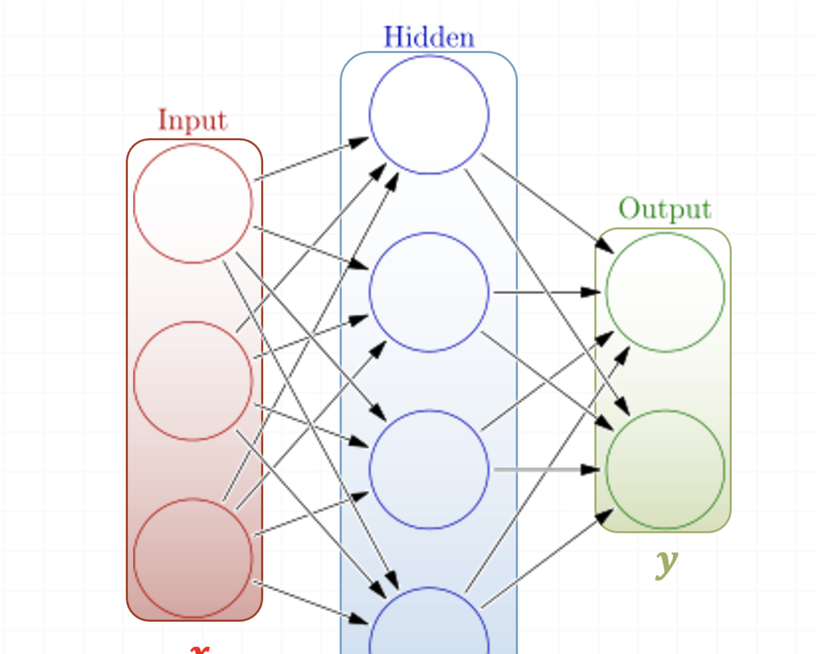
Feed-forward neural networks

- A typical multi-layer network consists of an input, hidden and output layer, each fully connected to the next, with activation feeding forward.

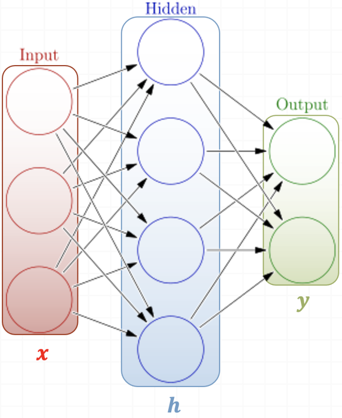


- The weights determine the function computed.

Feed-forward neural networks



Feed-forward neural networks



Weights

$$h = \sigma(\mathbf{W_1} x + b_1)$$

$$y = \sigma(W_2 h + b_2)$$

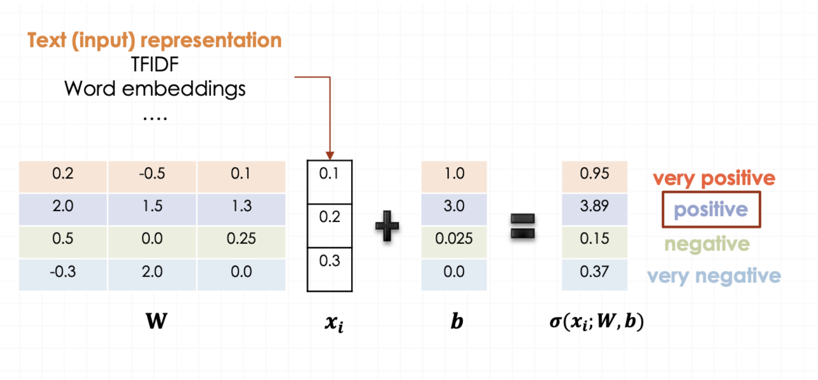
Activation functions

4 + 2 = 6 neurons (not counting inputs)
[3 x 4] + [4 x 2] = 20 weights
4 + 2 = 6 biases

---

26 learnable **parameters**

One forward pass



Training

https://medium.com/@ramrajchandradevan/the-evolution-of-gradient-descend-optimization-algorithm-4106a6702d39

Optimize objective/cost function $J(\theta)$

Generate error signal that measures difference between predictions and target values

Use error signal to change the weights and get more accurate predictions

Subtracting a fraction of the gradient moves you towards the (local) minimum of the cost function

Updating weights

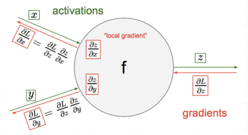
objective/cost function $J(\theta)$

Update each element of θ:

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{d}{\theta_j^{old}} J(\theta)$$

Matrix notation for all parameters (α: learning rate):

$$\theta_j^{new} = \theta_j^{old} - \alpha \nabla_\theta J(\theta)$$



Recursively apply chain rule though each node

Notes on training

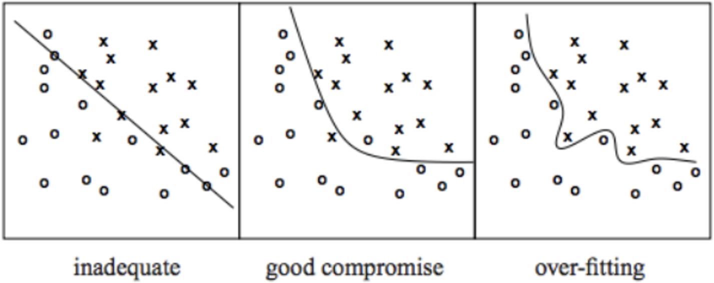
- Not guaranteed to converge to zero training error, may converge to local optima or oscillate indefinitely.
- However, in practice, does converge to low error for many large networks on real data.
- Many epochs (thousands) may be required, hours or days of training for large networks.
- To avoid local-minima problems, run several trials starting with different random weights (*random restarts*).
  - Take results of trial with lowest training set error.
  - Build a committee of results from multiple trials (possibly weighting votes by training set accuracy).

Hidden unit representations

- Trained hidden units can be seen as newly constructed features that make the target concept linearly separable in the transformed space.
- On many real domains, hidden units can be interpreted as representing meaningful features such as vowel detectors or edge detectors, etc..
- However, the hidden layer can also become a distributed representation of the input in which each individual unit is not easily interpretable as a meaningful feature.

Overfitting

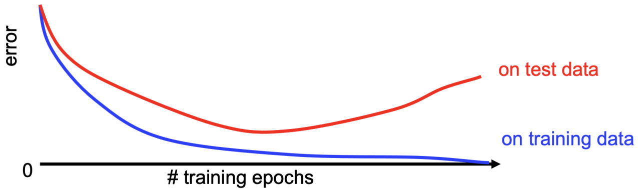
Learned hypothesis may fit the training data very well, even outliers (noise) but fail to generalize to new examples (test data)



http://wiki.bethanycrane.com/overfitting-of-data

Overfitting prevention

- Running too many epochs can result in over-fitting.



- Keep a hold-out validation set and test accuracy on it after every epoch. Stop training when additional epochs actually increase validation error.
- To avoid losing training data for validation:
  - Use internal K-fold CV on the training set to compute the average number of epochs that maximizes generalization accuracy.
  - Train final network on complete training set for this many epochs.

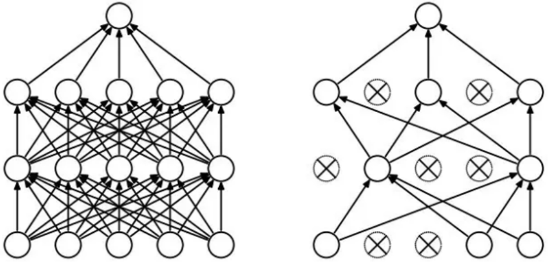
Regularization

Dropout

Randomly drop units (along with their connections) during training

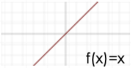
Each unit retained with fixed probability p, independent of other units

Hyper-parameter p to be chosen (tuned)



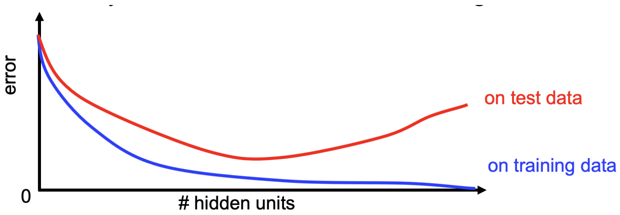
Srivastava, Nitish, et al. "Dropout: a simple way to prevent neural networks from overfitting." Journal of machine learning research (2014)

Loss functions and output

	Classification	Regression
Training examples	R^n x {class_1, ..., class_n} (one-hot encoding)	R^n x R^m
Output Layer	Soft-max [map R^n to a probability $P(y=j \mid \mathbf{x}) = \frac{e^{\mathbf{x}^\mathsf{T}\mathbf{w}_j}}{\sum_{k=1}^{K} e^{\mathbf{x}^\mathsf{T}\mathbf{w}_k}}$	Linear (Identity) or Sigmoid 
Cost (loss) function	Cross-entropy $J(\theta) = -\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{K}\left[y_k^{(i)} \log \hat{y}_k^{(i)} + \left(1-y_k^{(i)}\right)\log\left(1-\hat{y}_k^{(i)}\right)\right]$	Mean Squared Error $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}^{(i)}\right)^2$ Mean Absolute Error $J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left\lvert y^{(i)} - \hat{y}^{(i)}\right\rvert$

Determining the best number of hidden units

- Too few hidden units prevents the network from adequately fitting the data.
- Too many hidden units can result in over-fitting.



- Use internal cross-validation to empirically determine an optimal number of hidden units.
- Hyperparameter tuning

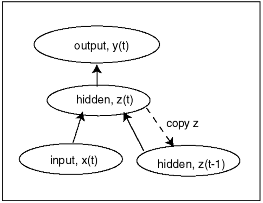
Recurrent Neural Networks

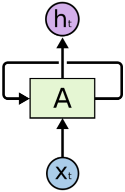
Recurrent Neural Network (RNN)

- Add feedback loops where some units' current outputs determine some future network inputs.
- RNNs can model dynamic finite-state machines, beyond the static combinatorial circuits modeled by feed-forward networks.

Simple Recurrent Network (SRN)

- Initially developed by Jeff Elman ("*Finding structure in time*," 1990).
- Additional input to hidden layer is the state of the hidden layer in the previous time step.

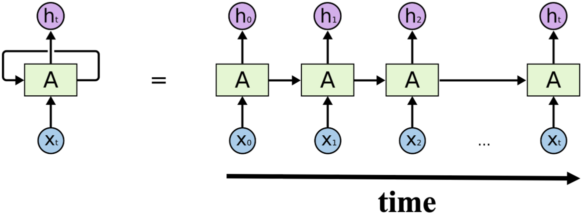




http://colah.github.io/posts/2015-08-Understanding-LSTMs/

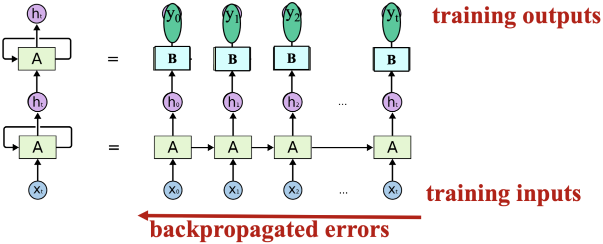
Unrolled RNN

- Behavior of RNN is perhaps best viewed by "unrolling" the network over time.



Training RNNs

- RNNs can be trained using "backpropagation through time."
- Can viewed as applying normal backprop to the unrolled network.



LSTM

Vanishing gradient problem

Suppose we had the following scenario:

Day 1: Lift Weights

Day 2: Swimming

Day 3: At this point, our model must decide whether we should take a rest day or yoga. Unfortunately, it only has access to the previous day. In other words, it knows we swam yesterday but it doesn't know whether had taken a break the day before. Therefore, it can end up predicting yoga.

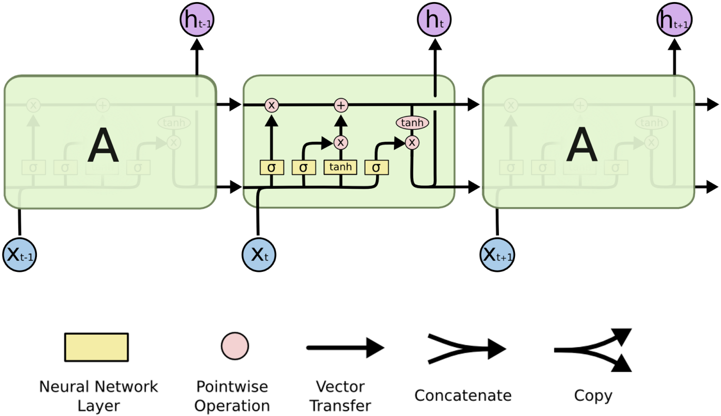
- Backpropagated errors multiply at each layer, resulting in exponential decay (if derivative is small) or growth (if derivative is large).
- Makes it very difficult train deep networks, or simple recurrent networks over many time steps.
- LSTMs were invented, to get around this problem.

https://towardsdatascience.com/

Long Short Term Memory

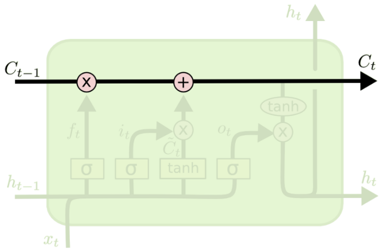
- LSTM networks, add additional gating units in each memory cell.
  - Forget gate
  - Input gate
  - Output gate
- Prevents vanishing/exploding gradient problem and allows network to retain state information over longer periods of time.

LSTM network architecture | https://colah.github.io/posts/2015-08-Understanding-LSTMs/



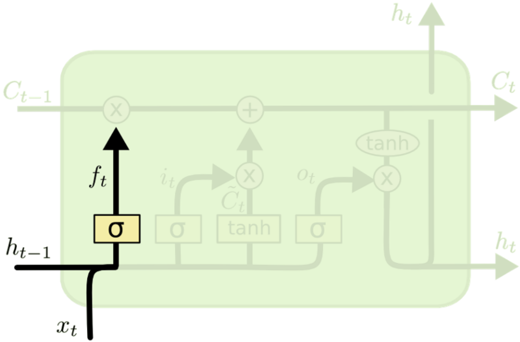
Cell state

- Maintains a vector C_t that is the same dimensionality as the hidden state, h_t
- Information can be added or deleted from this state vector via the forget and input gates.



Forget gate

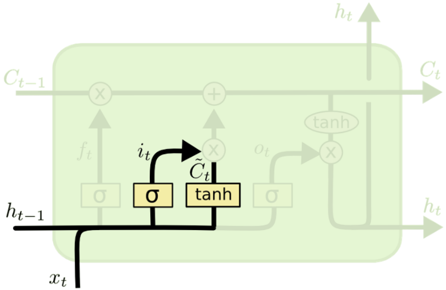
- Forget gate computes a 0-1 value using a logistic sigmoid output function from the input, x_t, and the current hidden state, h_t:
- Multiplicatively combined with cell state, "forgetting" information where the gate outputs something close to 0.



$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Input gate

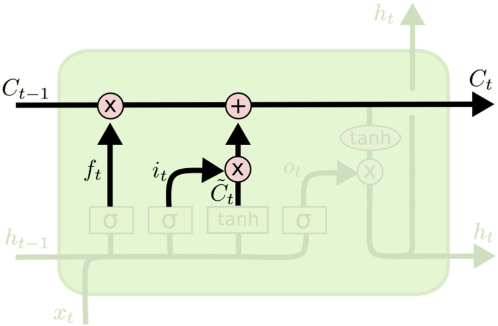
- First, determine which entries in the cell state to update by computing 0-1 sigmoid output.
- Then determine what amount to add/subtract from these entries by computing a tanh output (valued –1 to 1) function of the input and hidden state.



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$
$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

Updating the cell state

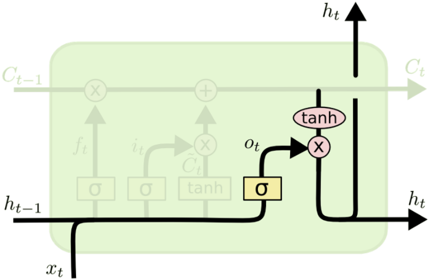
- Cell state is updated by using component-wise vector multiply to "forget" and vector addition to "input" new information.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Output gate

- Hidden state is updated based on a "filtered" version of the cell state, scaled to –1 to 1 using tanh.
- Output gate computes a sigmoid function of the input and current hidden state to determine which elements of the cell state to "output".



$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$

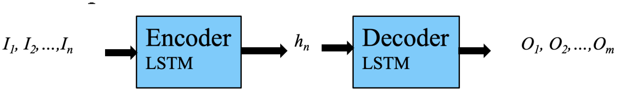
$$h_t = o_t * tanh(C_t)$$

General problems solved with LSTM

- Sequence labeling
  - Train with supervised output at each time step computed using a single or multilayer network that maps the hidden state (h_t) to an output vector (O_t).
- Language modeling
  - Train to predict next input ($O_t = I_{t+1}$)
- Sequence (e.g. text) classification
  - Train a single or multilayer network that maps the final hidden state (h_n) to an output vector (O).

Sequence to sequence transduction (mapping)

- Encoder/Decoder framework maps one sequence to a "deep vector" then another LSTM maps this vector to an output sequence.



- Train model "end to end" on I/O pairs of sequences.

LSTM application architectures

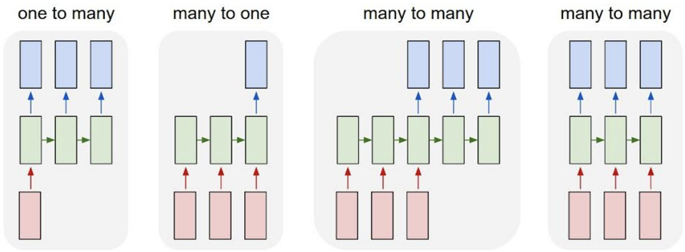
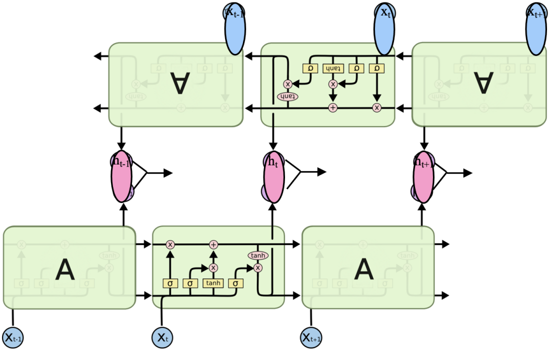


Image Captioning | Video Activity Recog, Text Classification | Video Captioning, Machine Translation | POS Tagging, Language Modeling

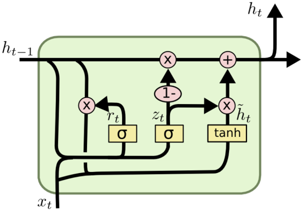
Bi-directional LSTM (Bi-LSTM)

- Separate LSTMs process sequence forward and backward and hidden layers at each time step are concatenated to form the cell output.



Gated Recurrent Unit (GRU)

- Alternative RNN to LSTM that uses fewer gates (Cho, et al., 2014)
  - Combines forget and input gates into "update" gate.
  - Eliminates cell state vector



$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = tanh(W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{(h)}_t$$

Attention

- For many applications, it helps to add “attention” to RNNs.
- Allows network to learn to attend to different parts of the input at different time steps, shifting its attention to focus on different aspects during its processing.
- Used in image captioning to focus on different parts of an image when generating different parts of the output sentence.
- In MT, allows focusing attention on different parts of the source sentence when generating different parts of the translation.

Summary

Summary

- Feed-forward neural networks
- Backpropagation
- Recurrent neural networks
- SRN
- LSTM
- Bi-LSTM
- GRU

Time for Practical 6!