Deep Learning for Text

Ayoub Bagheri

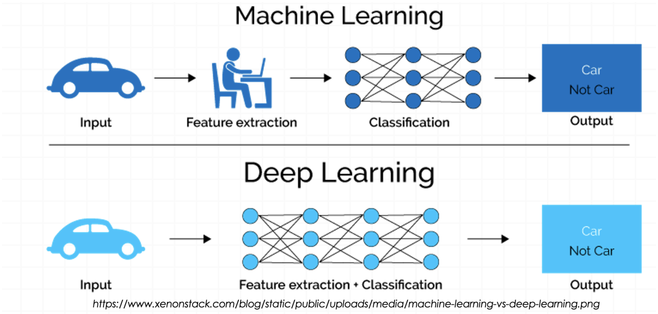
Lecture plan

1. Deep learning
2. Feed-forward neural networks
3. Recurrent neural networks

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.


Machine Learning
Input
Feature extraction
Classification
Car
Not Car
Output
Deep Learning
Input
Feature extraction + Classification
Car
Not Car
Output
https://www.xenonstack.com/blog/static/public/uploads/media/machine-learning-vs-deep-learning.png

Deep learning vs neural networks

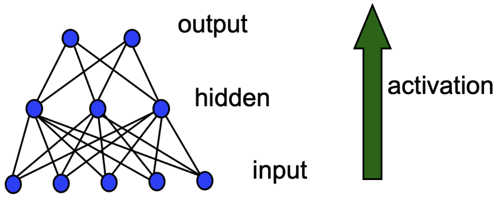
- Deep learning is only “deep” neural networks, such that with multiple ($>$2) layers.

Deep learning architechtures

- Feed-forward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Self-organizing maps
- Autoencoders
- Transformers: Large Language Models (LLMs)

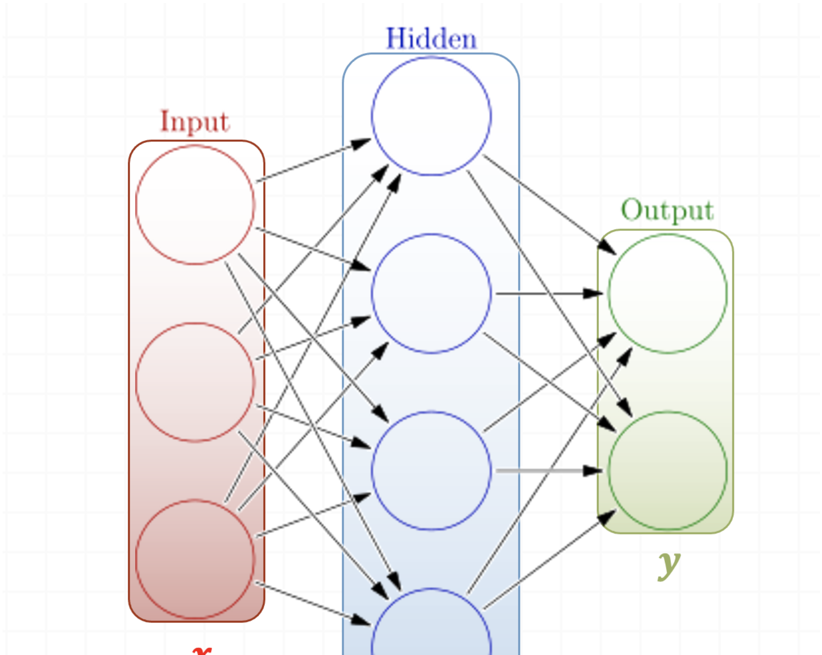
Feed-forward neural networks

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

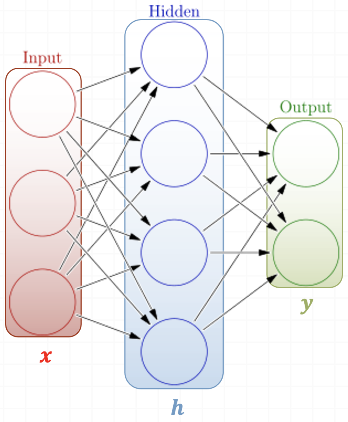


- The weights determine the function computed.

Feed-forward neural networks



Feed-forward neural networks



Weights

$$h = \sigma(W_1 x + b_1)$$

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

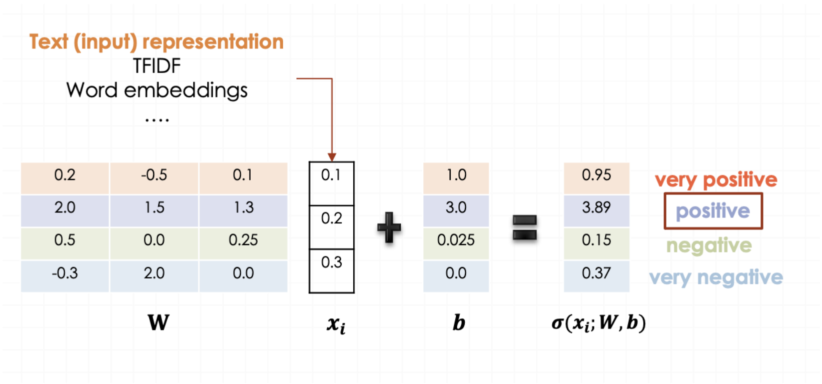
Activation functions

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

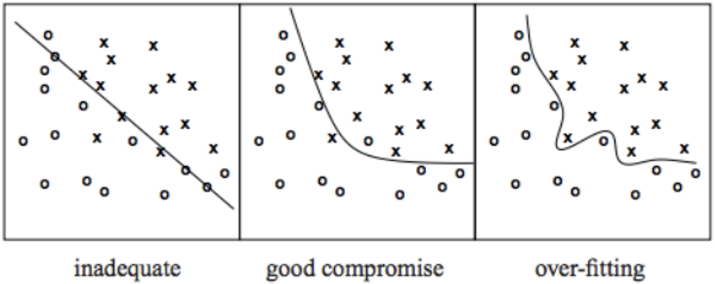


Hidden unit representations

- 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

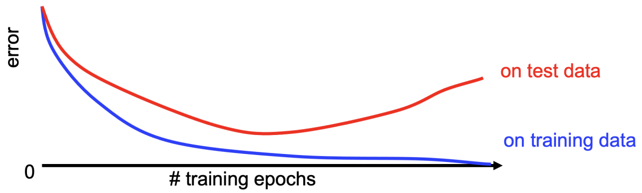
Learned hypothesis may fit the training data very well, even outliers (noise) but fail to generalize to new examples (test data)



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.

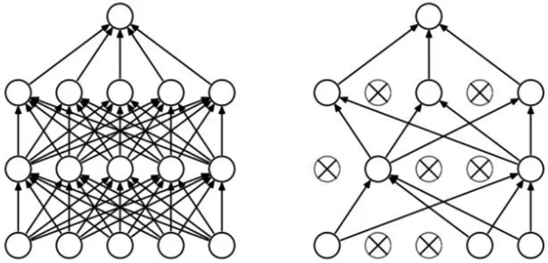
Regularization

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)

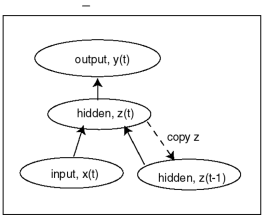
Recurrent Neural Networks

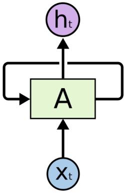
Recurrent Neural Network (RNN)

- 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.

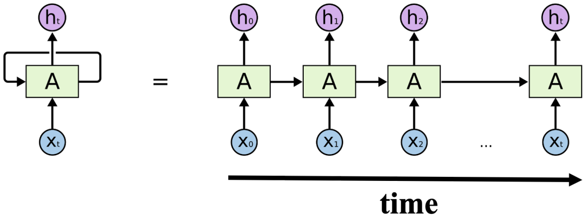




http://colah.github.io/posts/2015-08-Understanding-LSTMs/

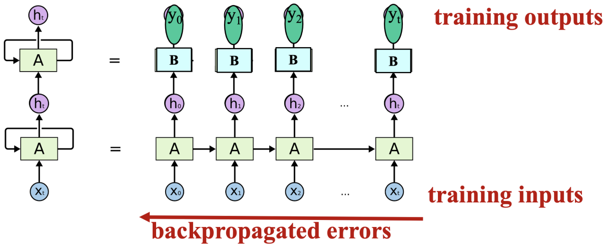
Unrolled RNN

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



Training RNNs

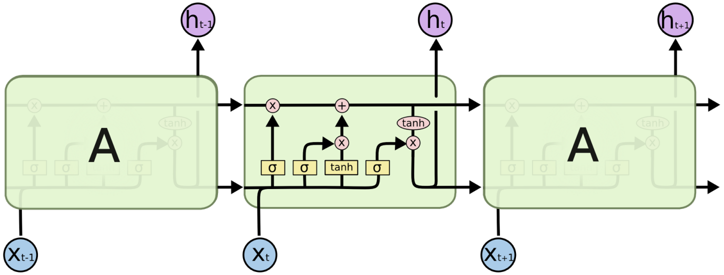
- RNNs can be trained using "backpropagation through time."
- Can viewed as applying normal backprop to the unrolled network.

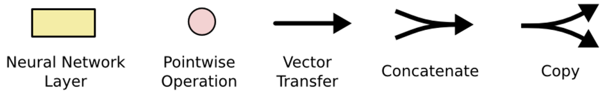


Long Short Term Memory (LSTM)

- 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/





In R

```r
# Use Keras Functional API
input <- layer_input(shape = list(maxlen), name = "input")

model <- input %>%
  layer_embedding(input_dim = max_words, output_dim = dim_
                  weights = list(word_embeds), trainable =
  layer_lstm(units = 80, return_sequences = TRUE)

output <- model                  %>%
  layer_global_max_pooling_1d() %>%
  layer_dense(units = 1, activation = "sigmoid")

model <- keras_model(input, output)

summary(model)
```

In R

```
## Model: "model"
## ________________________________________________________________________________
##  Layer (type)                Output Shape              Param #    Trainable
## ================================================================================
##  input (InputLayer)          [(None, 60)]              0          Y
##  embedding (Embedding)       (None, 60, 300)           3000000    N
##  lstm (LSTM)                 (None, 60, 80)            121920     Y
##  global_max_pooling1d (GlobalM  (None, 80)             0          Y
##  axPooling1D)
##  dense (Dense)               (None, 1)                 81         Y
## ================================================================================
## Total params: 3,122,001
## Trainable params: 122,001
## Non-trainable params: 3,000,000
## ________________________________________________________________________________
```

In R

```r
# instead of accuracy we can use "AUC" metrics from "tenso
model %>% compile(
  optimizer = "adam",
  loss = "binary_crossentropy",
  metrics = tensorflow::tf$keras$metrics$AUC() # metrics =
)
```

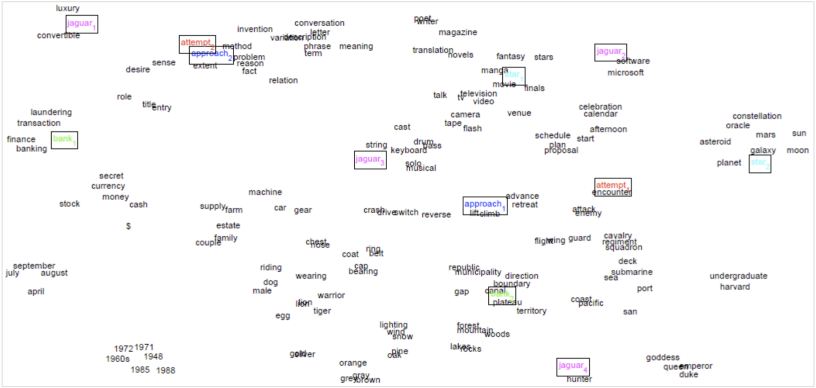
In R

```r
history <- model %>% keras::fit(
  x_train, y_train,
  epochs = 10,
  batch_size = 32,
  validation_split = 0.2
)
```

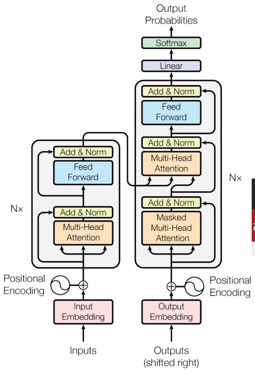
Transformers

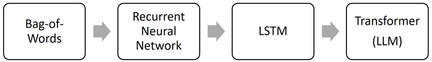
Transformers

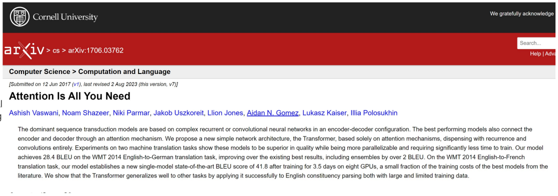
Contextual Word Embeddings



Transformers







Computer Science > Computation and Language

[Submitted on 12 Jun 2017 (v1), last revised 2 Aug 2023 (this version, v7)]

Attention Is All You Need

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin

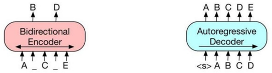
The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

Transformers

- A transformer adopts an encoder-decoder architecture.
- Transformers were developed to solve the problem of sequence transduction, or neural machine translation. That means any task that transforms an input sequence to an output sequence.
- More details on the architecture and implementation:
  - https://arxiv.org/abs/1810.04805
  - http://nlp.seas.harvard.edu/2018/04/03/attention.html
  - https://jalammar.github.io/illustrated-transformer/

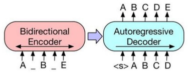
Transformer foundation models: BERT, GPT, BART

- **BERT**: **B**idirectional **E**ncoder **R**epresentations from **T**ransformers.
  - *Masked word prediction, text representation*
- **GPT**: **G**enerative **P**re-trained **T**ransformer.
  - *Next word prediction, text generation, chat*
- **BART** = "BERT+GPT": **B**idirectional encoder and **A**uto-**R**egressive decoder **T**ransformers.
  - *Noised text reconstruction, summarization, translation, spelling correction*



(a) BERT: Random tokens are replaced with masks, and the document is encoded bidirectionally. Missing tokens are predicted independently, so BERT cannot easily be used for generation.

(b) GPT: Tokens are predicted auto-regressively, meaning GPT can be used for generation. However words can only condition on leftward context, so it cannot learn bidirectional interactions.



(c) BART: Inputs to the encoder need not be aligned with decoder outputs, allowing arbitary noise transformations. Here, a document has been corrupted by replacing spans of text with a mask symbols. The corrupted document (left) is encoded with a bidirectional model, and then the likelihood of the original document (right) is calculated with an autoregressive decoder. For fine-tuning, an uncorrupted document is input to both the encoder and decoder, and we use representations from the final hidden state of the decoder.

BERT: Bidirectional Encoder Representations from Tranformers

BERT: Bidirectional Encoder Representations from Tranformers

Transformers

- ChatGPT: https://chat.openai.com/
- Write with Transformer: https://transformer.huggingface.co/
- Talk to Transformer: https://app.inferkit.com/demo
- Transformer model for language understanding: https://www.tensorflow.org/text/tutorials/transformer
- Pre-trained models: https://huggingface.co/transformers/pretrained_models.html

ChatGPT (5-min exercise)

- Go to https://chat.openai.com/ and login
- How many parameters has chatgpt-3 model been trained on?
- How many parameters has chatgpt-4 model been trained on?
- What is the next generation NLP?
- Suppose we want to build an application to help a user buy a car from textual catalogues. The user looks for any car cheaper than \$10,000.00. Assume we are using the following data: txt <- c("Price of Tesla S is \$8599.99.", "Audi Q4 is \$7000.", "BMW X5 costs \$900"). Could you give me a regular expression to do this in R?

Summary

Summary

- Deep learning
- Feed-forward neural networks
- Recurrent neural networks
- State-of-the-art LLMs

Practical 8