Deep Learning for Text

Ayoub Bagheri

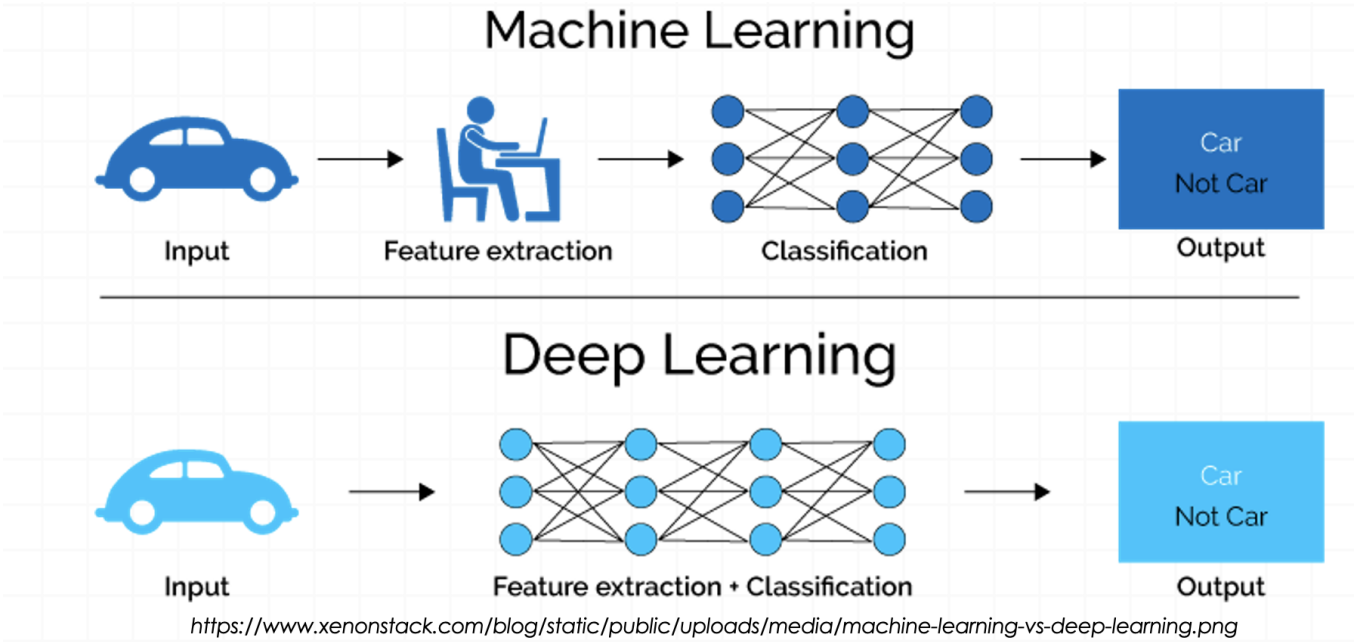
Lecture plan

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.



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

Deep learning vs neural networks

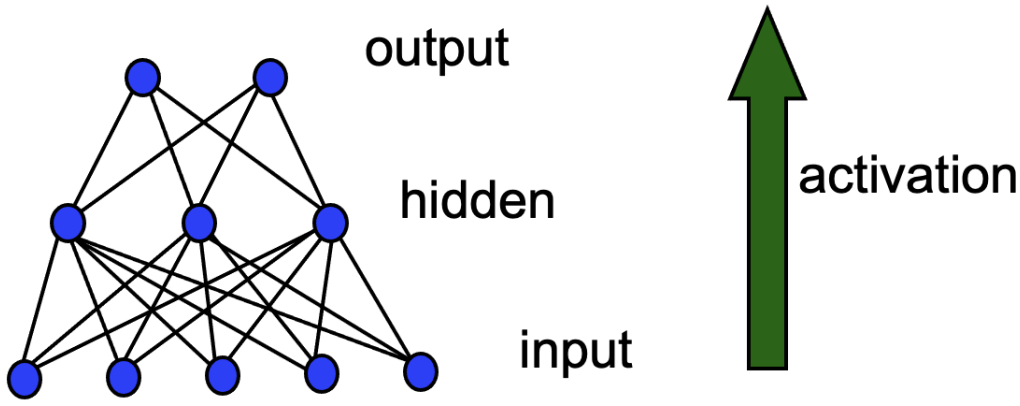
- Deep learning is only "deep" neural networks, such that with multiple ($>$2) layers.

Deep learning architechtures

- Feed-forward neural networks
- Convolutional neural networks
- Recurrent neural networks
- Self-organizing maps
- Autoencoders

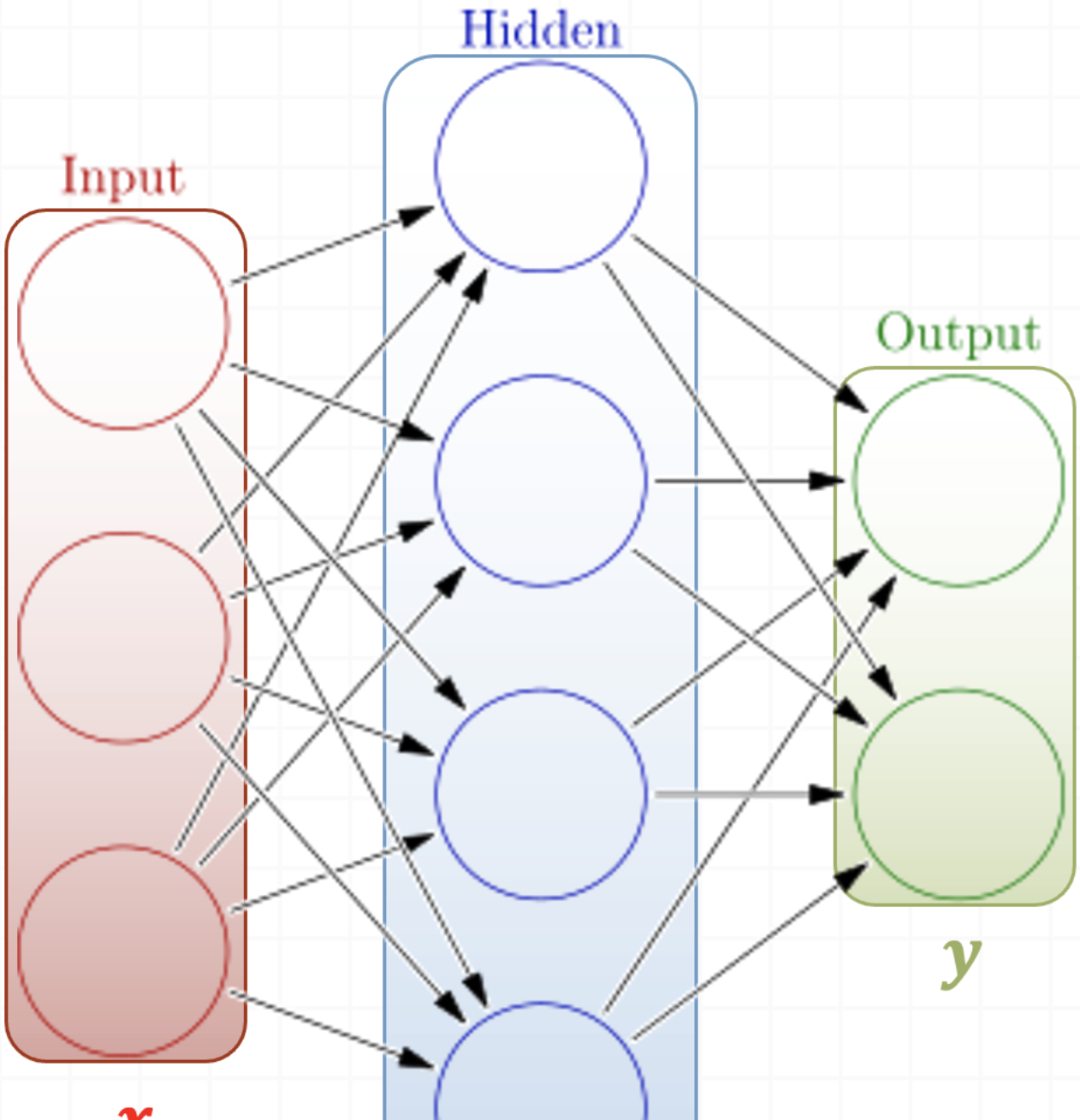
Feed-forward neural networks

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

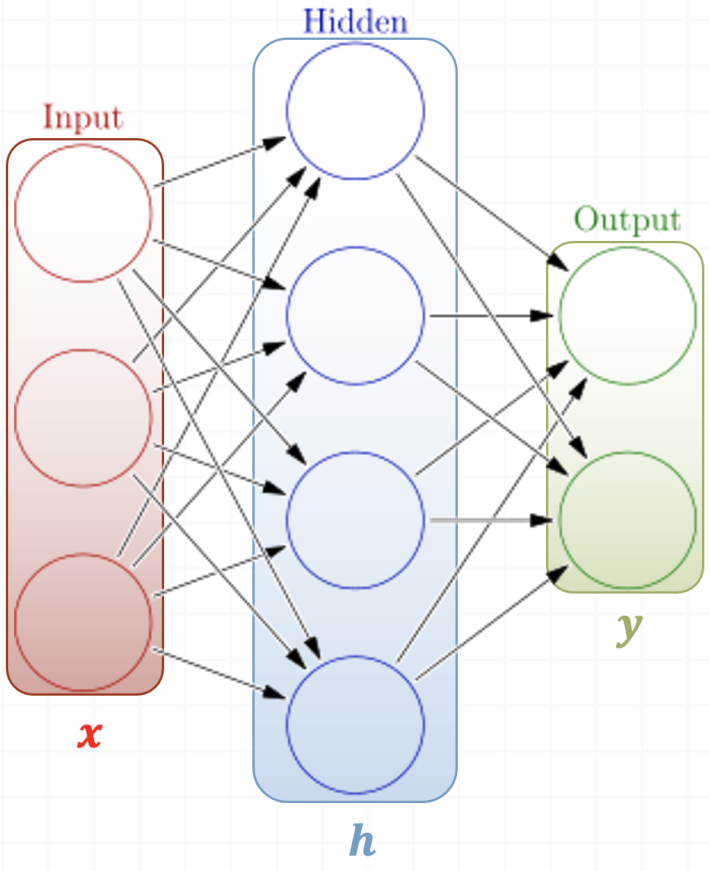


- The weights determine the function computed.

Feed-forward neural networks



Feed-forward neural networks



Weights

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

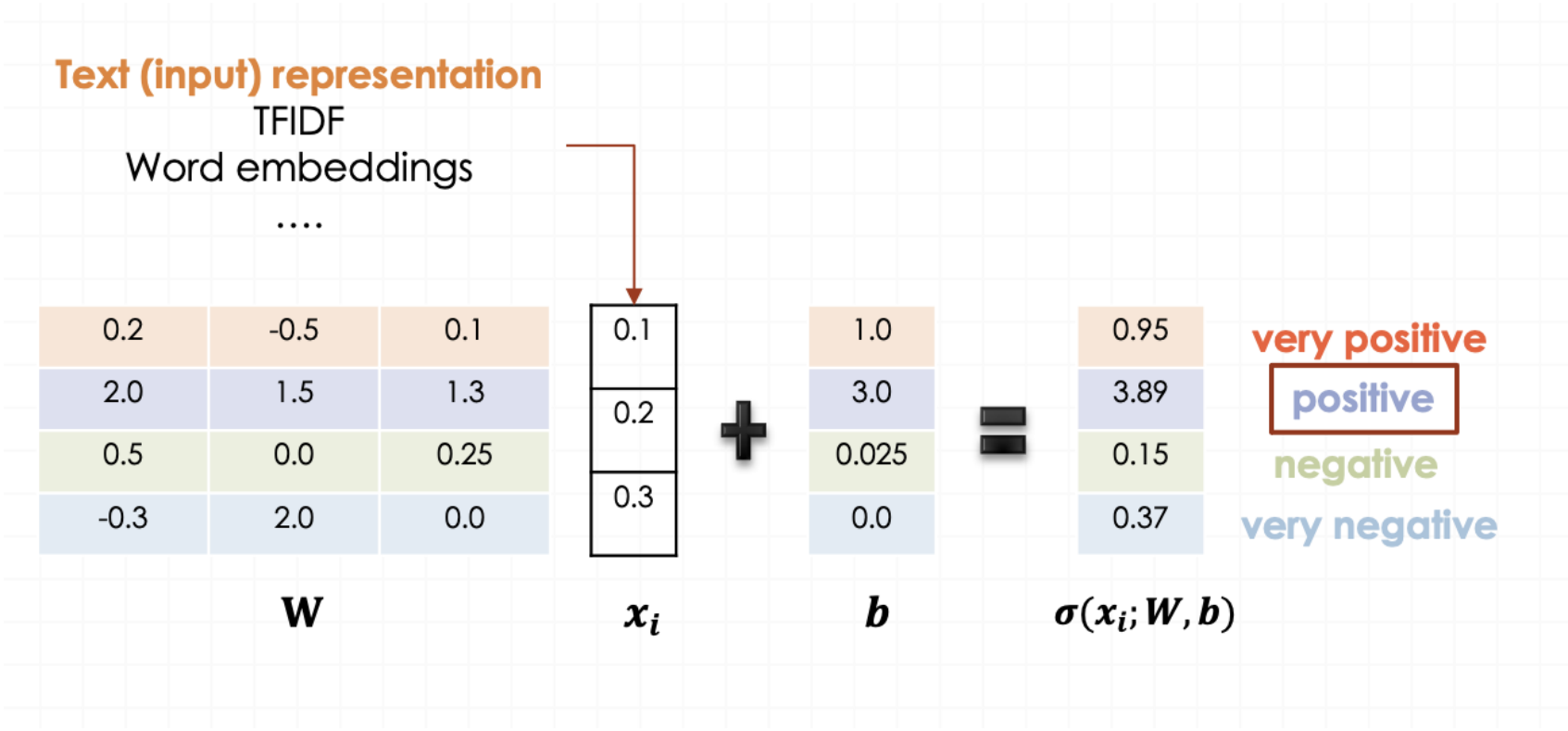
$$y = \sigma(W_2 h + b_2)$$

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

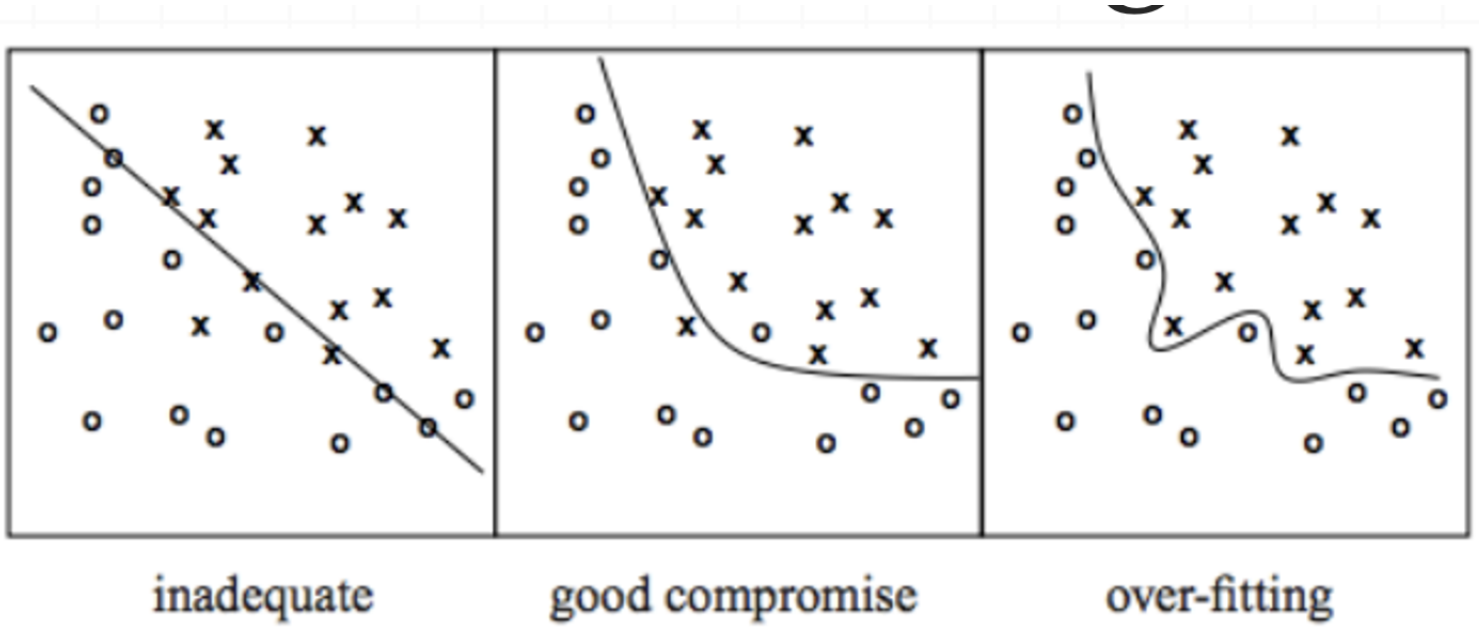


Hidden unit representations

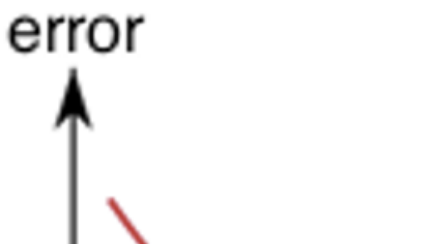
- Trained hidden units can be seen as newly constructed features that make the target concept linearly separable in the transformed space.
- On many real domains, hidden units can be interpreted as representing meaningful features such as vowel detectors or edge detectors, etc..
- However, the hidden 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)

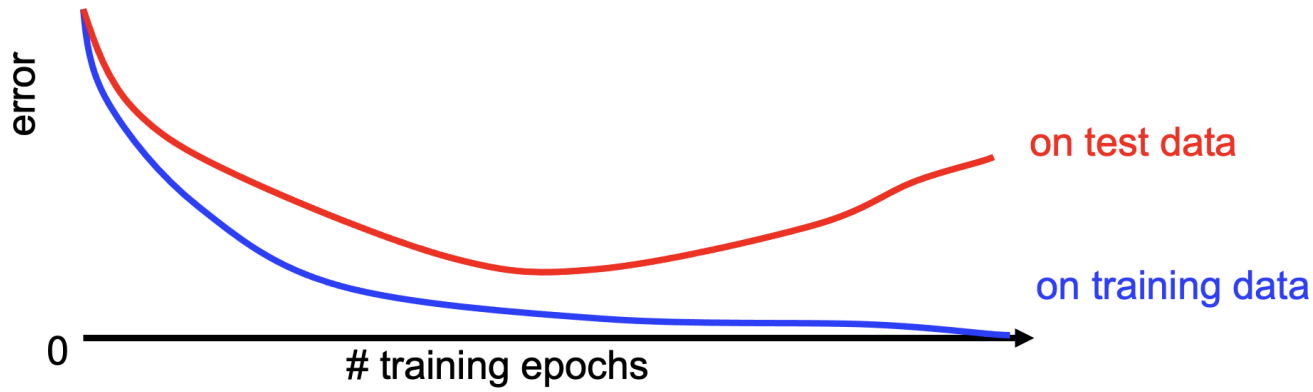


http://wiki.bethanycrane.com/overfitting-of-data



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.

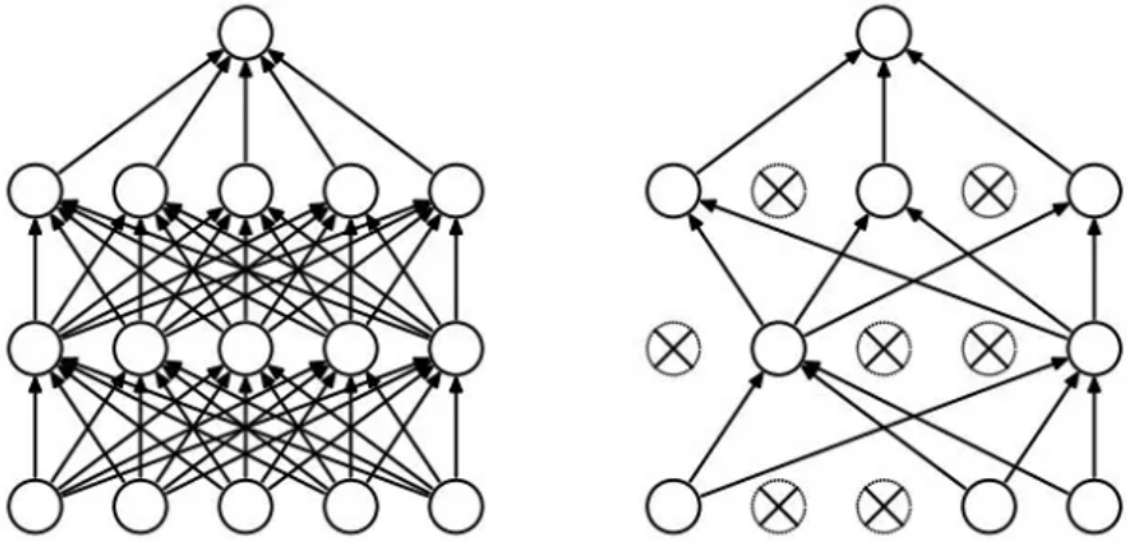
Regularization

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)

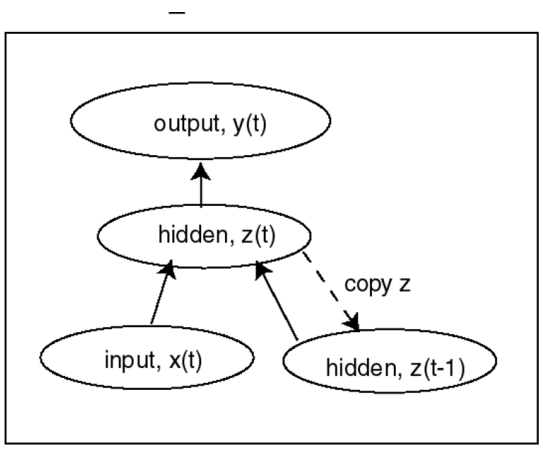
Recurrent Neural Networks

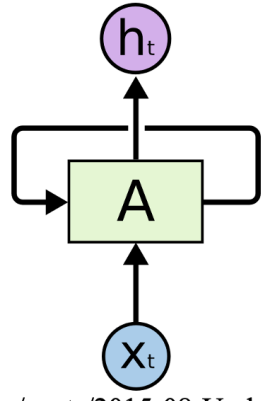
Recurrent Neural Network (RNN)

- 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.

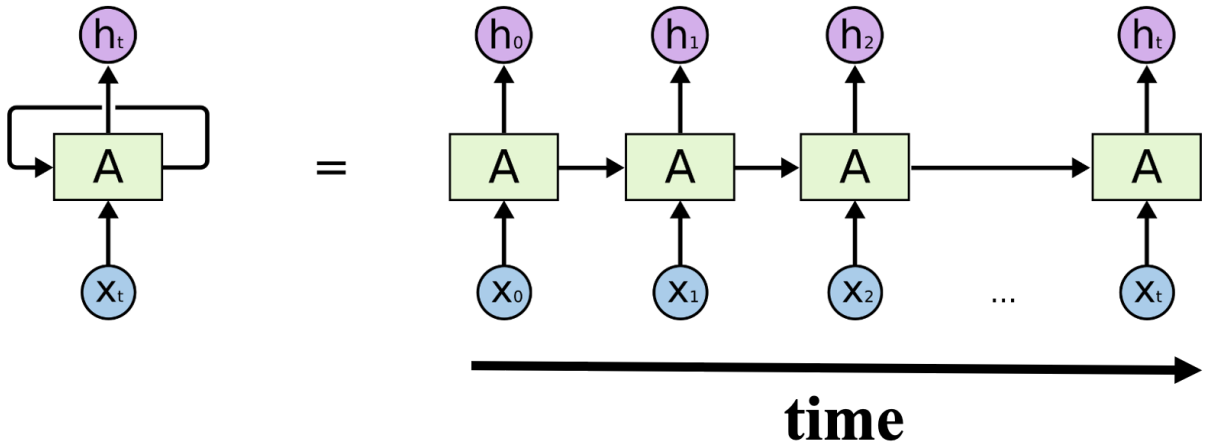




http://colah.github.io/posts/2015-08-Understanding-LSTMs/

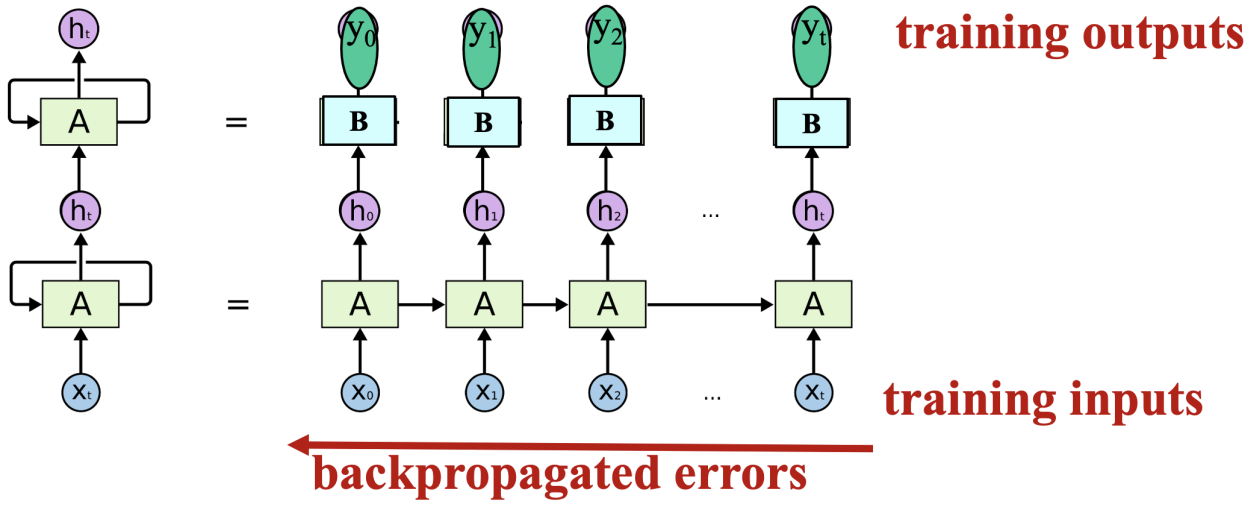
Unrolled RNN

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



Training RNNs

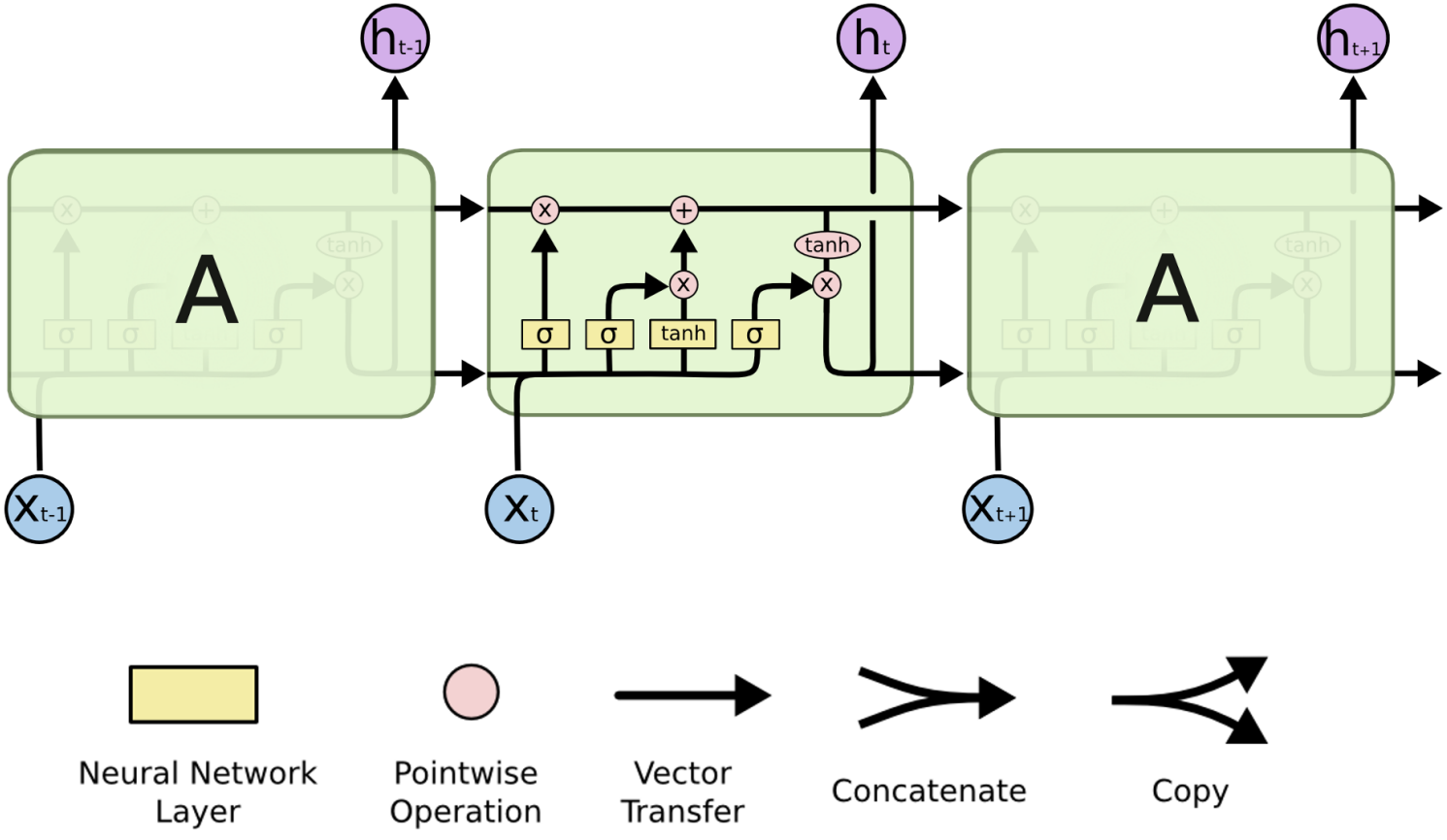
- RNNs can be trained using "backpropagation through time."
- Can viewed as applying normal backprop to the unrolled network.



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

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

model <- input %>%
  layer_embedding(input_dim = max_words, output_dim = dim_s
                  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

```
# 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

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

Summary

Summary

- Deep learning
- Feed-forward neural networks
- Recurrent neural networks

Practical 8