[Lec 04]

Back-propagation and computation graphs

Matrix Gradients for simple Neural Net

Derivative with respect to weight matrix

$$\begin{array}{ccc} \dfrac{\partial s}{\partial \boldsymbol{W}} & = & \boldsymbol{\delta}^T \quad \boldsymbol{x}^T \\ [n \times m] & & [n \times 1][1 \times m] \end{array}$$

Tips

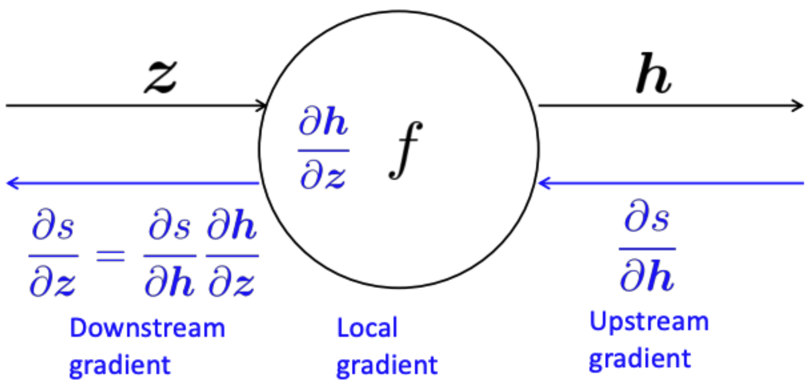
- Carefully Define Variables
- Use Chain Rule and Shape Convention

Pitfall

- Always use pre-trained word vectors
- Fine-tune it only when size of data is large

Computation Graphs and Back propagation

→ Taking Derivatives using the chain rule



- + : distributes the upstream gradient
- max: routes the upstream gradient
- * : switches the upstream gradient

→ For efficiency, compute all gradients at once

Back-Prop in General Computation Graph

- Fprop: visit nodes in topological order
- Bprop: Recursively apply chain rule along computation graph
    - initialize output gradient = 1
    - visit nodes inreverse order

Regularization

→ prevents overfitting when having lots of features

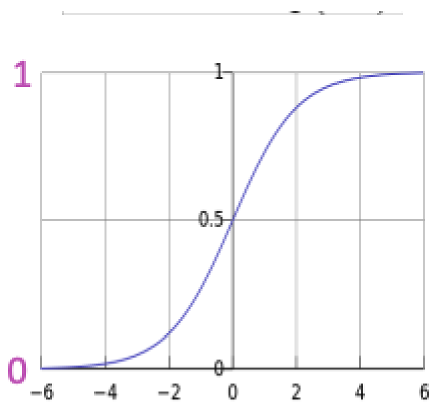
Vectorization

→ looping over word vectors verses concatenating them all into one large matrix and then multiplying soft-max weights

→ Matrices preferred

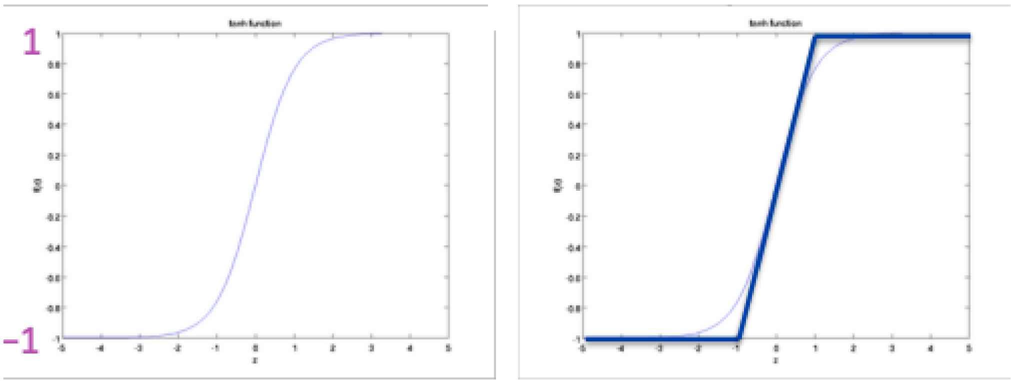
Nonlinearities

- sigmoid (logistic)

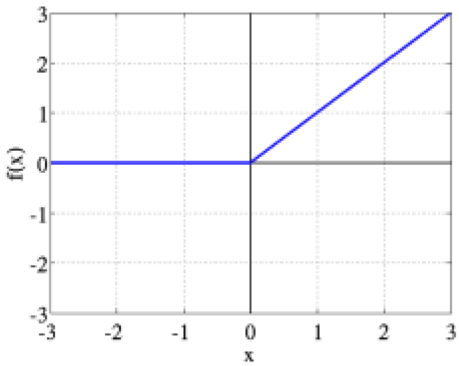


- tanh & hard tanh

  → rescaled and shifted sigmoid (tanh)



- ReLU (Rectified Linear Unit)
  - → Best or building feed-forward deep network
  - → variants exist (Leaky Relu, Parametric Relu)



Initialization

→ normally must initialize weights to small random values

→ biases

- hidden: Initialize to zero
- Output: Initialize to optimal value if weights were 0

Optimizers

→ SGD works fine

- But hand tune learning rate

→ Adaptive Optimizers

- Adagrad, RMSprop, **Adam**

Learning Rate

→ start around 0.001

must be order of magnitude (powers of 10)

→ Too Big: may diverge vs Too Small: May not be trained by deadline

→ Better to have decreasing learning rate while training