CS546: Machine Learning in NLP (Spring 2018)

http://courses.engr.illinois.edu/cs546/

Lecture 6
Neural network training

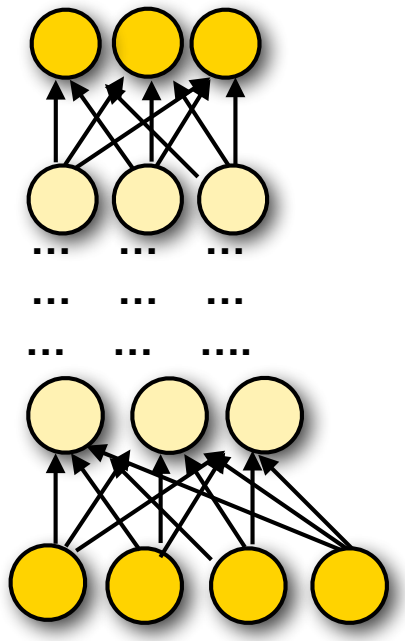
Julia Hockenmaier

juliahmr@illinois.edu

3324 Siebel Center

Office hours: Tue/Thu 2pm-3pm

Multi-layer feedforward networks



Output layer: vector **y**

Hidden layer: vector $\mathbf{h_n}$

Hidden layer: vector $\mathbf{h_1}$

Input layer: vector **x**

Multi-layer feedforward networks

- Each input unit *i* is a scalar *v(i)*
- Each unit *i* in a hidden or the output layer is connected to every unit *j* in the preceding (hidden or input) layer.
- Each unit *i* in a hidden our the output layer computes a scalar that depends on a (nonlinear) function g() and the outputs of the units j in the preceding layer

  $v(i) = g(\sum_j w_{ji}\, v(j) + b)$
- The learned weight w_{ji} specifies by how much to multiply the output of unit *j* when going to unit *i.*
- The nonlinear function g() is typically specified on a layer-by-layer basis
- Each unit in the output layer is also a scalar.
- multiclass classification: k output units.

  argmax = pick one element out of this vector

Neural Network Training (Ch. 5)

Neural Network Training

Neural networks are differentiable parametrized functions.

- Training a NN = setting its parameters (weight matrices).

Training NNs is done via gradient-based optimization techniques.

How can we compute gradients for complex networks?

- Basic idea: use the chain rule of differentiation
- This can be done automatically via the backprop(agation) algorithm
- Common abstraction (used in many NN libraries): computation graph

The Computation Graph

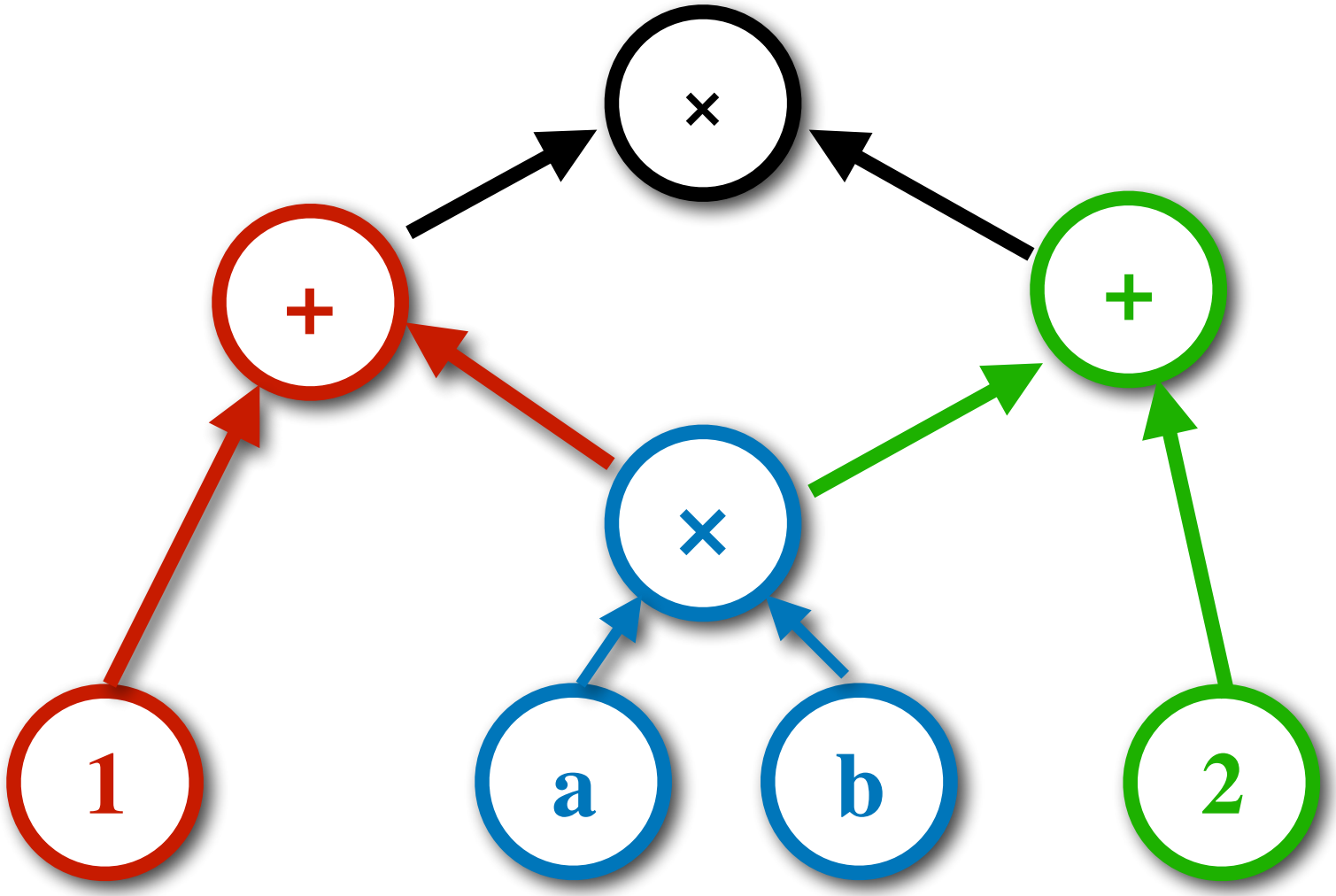
A representation of a mathematical computation as a directed acyclic graph (DAG)

Nodes represent mathematical operations or variables

- One node may be reused multiple times (e.g. bound variables)
- Hence, we're dealing with DAGs, not trees

Edges capture the flow of intermediary values (define the order of computation)

$$(a \times b + 1) \times (a \times b + 2)$$



Computation Graphs for NNs

1. Graphs with unbound input

Specify the network architecture

2. Graphs with concrete input

Specify the instantiation to one particular example

Sufficient for prediction/testing

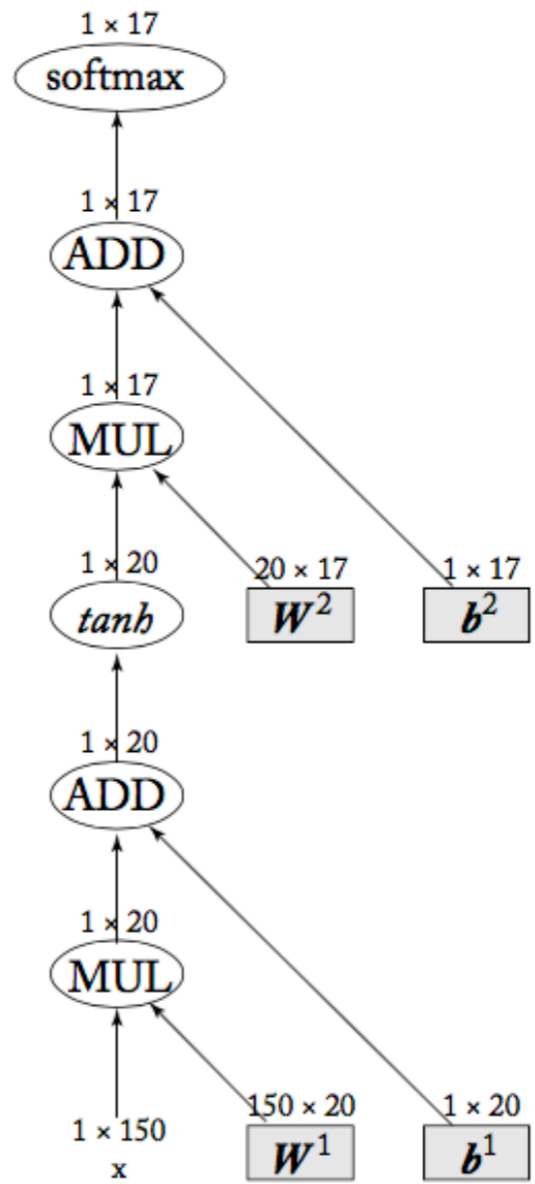
3. Graphs with concrete input, expected output and final loss node

Required for training

Unbound input

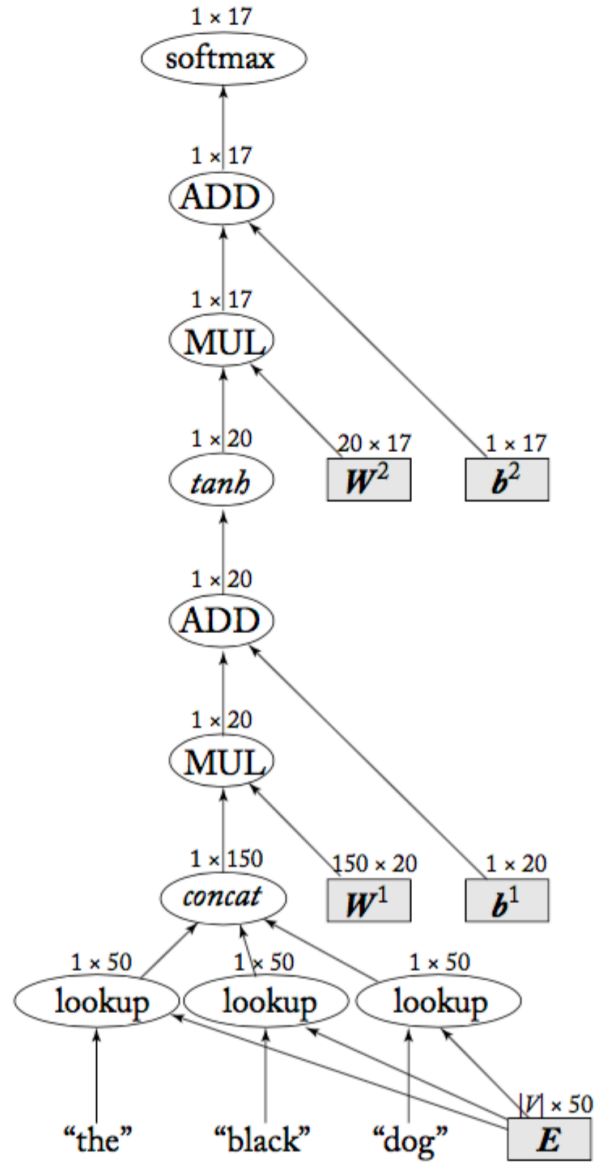
Multilayer feedforward network:

- 150 input units
- 1 hidden layer with 20 units and tanh activation function
- 17 output units with softmax



With concrete input

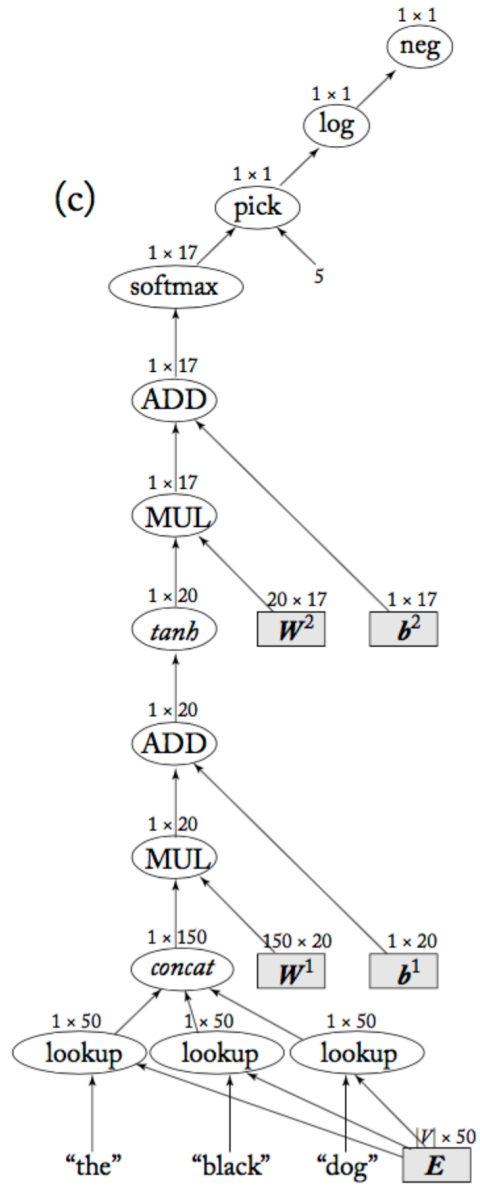
Three input words, represented as 50-dimensional vectors



With input, output, loss

Correct class here is 5 (‘Noun’)

“pick” = indexing operation



Forward computation

Computes the output of the nodes

Traverses nodes i = 1…N in topological order:

f_i: function computed at node i

(this is a constant if the node is a variable, or an input)

$\pi(i)$ = parent node of node i

$\pi^{-1}(i)$ = children nodes of node i

(this is an empty set if the node is a variable, or an input)

v(i): output of node i (application of f_i to outputs of $\pi^{-1}(i)$

Forward pass algorithm:

for i = 1…N **do**:

let $a_1 \ldots a_m = \pi^{-1}(i)$

$v(i) := f_i(a_1, \ldots, a_m)$

Backward computation (backprop)

Assume node N with scalar (1x1) output is the loss node: After forward pass, v(N) contains the loss.

Backward computation computes $d(i) = \delta N/\delta i$ (gradients of parameter at node i wrt. v(N)) for all nodes i = 1..N

$\delta f_j/\delta i$: partial derivative of $f_j(\pi^{-1}(j))$ wrt. the argument $i \in \pi^{-1}(j))$

Algorithm 5.4 Computation graph backward pass (backpropagation).

1: $d(N) \leftarrow 1$ $\quad \triangleright \frac{\partial N}{\partial N} = 1$

2: **for** i = N-1 to 1 **do**

3: $\quad d(i) \leftarrow \sum_{j \in \pi(i)} d(j) \cdot \boxed{\frac{\partial f_j}{\partial i}}$

$\triangleright d(i) = \frac{\partial N}{\partial i} = \sum_{j \in \pi(i)} \frac{\partial N}{\partial j} \frac{\partial j}{\partial i}$

In practice

Each node type needs to implement $\delta f_j/\delta i$

NN libraries provide this functionality for standard node types

What does this mean for operations like pick(**x**, 5)?

Only the contribution of that node to the computation matters.

pick(**x**, 5) results in only one scalar (the 5th element of the vector **x**)

The gradient of pick(**x**, 5) is a vector of the same dimensionality as **x**, where the 5th element = 1, and all other elements are 0.

max(0,x): value of gradient is 1 for x > 0, 0 otherwise

How to train a neural net

The architecture defines the computation graph. You need to specify the (unbound) graph for the desired architecture, and how to instantiate this graph for a given training example.

For each iteration:
  For each training example:
    Instantiate graph
    Compute forward pass to get the loss at the loss node N
    Compute the gradients by running a backward pass (backprop)
    Update the parameters
Return the parameters

Practical considerations

How do you optimize the parameters:

- Variants of SGD that take care of the learning rate (e.g. Adam)

How do you initialize the parameters:

- Random initialization can have a big effect on performance
  - (but can also be an effective way to produce ensembles)
- Using random restarts can help (but is very slow)
- Magnitude of weights matters.
  - The book suggests schemes for tanh (Xavier) and ReLU units

Gradients may vanish or explode (get very large):

- Motivation for LSTMs and GRUs (we'll get to that)
- Batch normalization (normalize inputs to each layer in each minibatch to have zero mean and unit variance)
- Gradient clipping helps too

Practical considerations

How do you go over the training data

- Shuffle after each iteration
- Use minibatches (k examples): average the loss of all k examples with one averaging node in the computation graph

How do you set learning rates?

- Explore a range of rates in [0,1], e.g. [0.001, 0.01, 0.1, 1]
- Monitor loss over time, decrease rate after loss stops improving on held out data
- Alternative: use decaying learning rate

Dropout training can be effective:

- randomly set hidden units to 0 during training

Logistics

Schedule

Week 1—Week 4: Lectures

Paper presentations: Lectures 9-28

1. Word embeddings
2. Language models
3. More on RNNs for NLP
4. CNNs for NLP
5. Multitask learning for NLP
6. Syntactic parsing
7. Information extraction
8. Semantic parsing
9. Coreference resolution
10. Machine translation I
11. Machine translation II
12. Generation
13. Discourse
14. Dialog I
15. Dialog II
16. Multimodal NLP
17. Question answering
18. Entailment recognition
19. Reading comprehension
20. Knowledge graph modeling