The Pipeline

• Extracting information from natural language text is a complex process
• We have been able to make it manageable by dividing it into many separate stages, each realized by its own (relatively simple) model
Introducing Errors

Each stage introduces some errors because

• model is an oversimplification of linguistic phenomenon

• (hand-prepared) training data may be noisy

• typically 10% error rate per stage
  – error rates range from 3% (POS) to 15% (name tagging)
Compounding Errors

- Errors in output of stage = errors due to faulty input + errors introduced by stage
- Final output error rate > 50%
Helpful Feedback

• We will reduce the error rate by in effect providing feedback from later stages to earlier ones

Example: “Roger Park began to work for IBM.”

• NE tagger says “Roger Park” is most likely a *location* but could also be a *person*

• relation extraction pattern (in IE stage) indicates a preference for “*person* works” over “*location* works”, fixing the error
Joint Inference

• To perform joint inference between stages A and B,
  – we define an objective function combining A and B
  – we search the combined space of A and B
    • where possible B outputs may depend on A output
    • seeking to maximize combined objective
    • much larger search space than with independent components
More Examples

• “Meet me in front of the White House.”
  – “White House” may refer to the building or the organization therein
    • 1-token context used by NE doesn’t help resolve ambiguity
  – relation pattern determines this is a reference to building

• “Ford employs 4000 in Detroit.”
  – event pattern determines that 4000 is of type people
Benefit

• For 2 or 3 stages, reductions of 2 - 3% (absolute) in error rate are reported
  – can only correct errors which change a valid (more likely) input to second stage to an invalid (less likely) input
Cost

• Much larger space to search
  – full search of product space infeasible
  – joint token-by-token scan updating multiple models (NE, relation, event) concurrently
    • use beam search to limit search space
      – follow only top n hypotheses at each token
        OR
      – follow only hypotheses within m% of best hypothesis
  OR
  – build graphical model connecting stages
Deep Learning

• Instead of training separate models for each stage and then coupling them, can we train a unified model to perform the entire analysis starting from a sequence of tokens:

deep learning using multi-layer neural networks
2 minute guide to neural nets

• building block = node (artificial neuron)
• node takes in multiple real-valued inputs, produces one real-valued output

\[ f(x) = K(\sum_{i} w_i \cdot x_i) \]

• \( w_i \) = weights (to be learned)
• \( K \) = (non-linear) activation function
Layers

Input Layer → Hidden Layer → Output Layer

Input #1 → Input #2 → Input #3 → Input #4
Training

- The network is a powerful system with a simple yet efficient training strategy termed *backpropagation*.
  - we define a cost function (given an input \( x \), the difference between the desired and current outputs)
  - we compute the derivative of the cost function w.r.t. each of the weights in the network
  - given an input/output pair, we update each of the weights (gradient descent)
Input Data

• We need to convert the text into a form which can be effectively used by the NN

• Natural form of information is a fixed-length vector of real values
  – these are the word embeddings discussed earlier
Convolution Layer

• In conventional feature-based systems, we program a set of features
  – each feature is a function of a set of consecutive tokens: $f(w_{i-1}, w_i, w_{i+1})$
  – feature is applied starting at each token in succession

• In NN, we want the network to learn the appropriate features given only the input and final output
  – We do this through a convolution layer – a row of nodes with a common set of weights
  – Has potential to create features best suited to main task