Deep Learning for NLP
Part 3

CS224N
Christopher Manning

(Many slides borrowed from ACL 2012/NAACL 2013 Tutorials by me, Richard Socher and Yoshua Bengio)

Part 1.5: The Basics

Backpropagation Training

2

Backprop

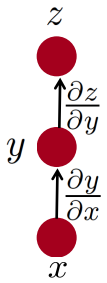
- Compute gradient of example-wise loss wrt parameters
- Simply applying the derivative chain rule wisely

$$z = f(y) \quad y = g(x) \quad \frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

- If computing the *loss*(*example*, *parameters*) is *O*(*n*) computation, then so is computing the gradient

3

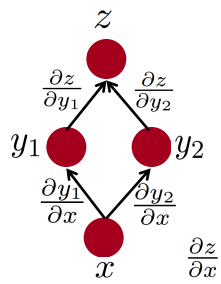
Simple Chain Rule



$$\Delta z = \frac{\partial z}{\partial y}\Delta y$$
$$\Delta y = \frac{\partial y}{\partial x}\Delta x$$
$$\Delta z = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$$
$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

4

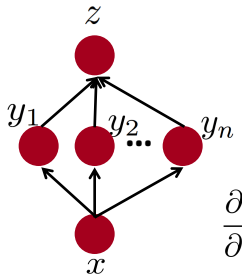
Multiple Paths Chain Rule



$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y_1}\frac{\partial y_1}{\partial x} + \frac{\partial z}{\partial y_2}\frac{\partial y_2}{\partial x}$$

5

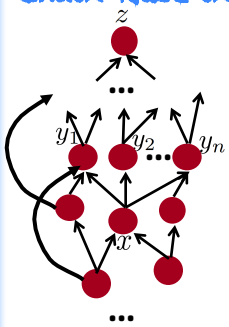
Multiple Paths Chain Rule – General



$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n}\frac{\partial z}{\partial y_i}\frac{\partial y_i}{\partial x}$$

6

Chain Rule in Flow Graph



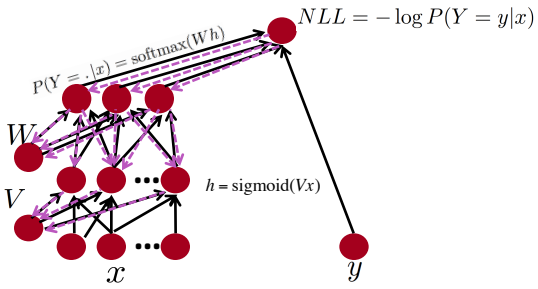
Flow graph: any directed acyclic graph
node = computation result
arc = computation dependency

$\{y_1, y_2, \ldots y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

7

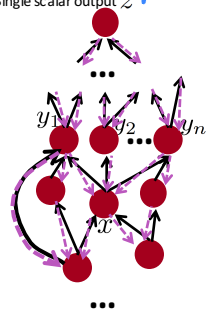
Back-Prop in Multi-Layer Net



8

Back-Prop in General Flow Graph

Single scalar output z



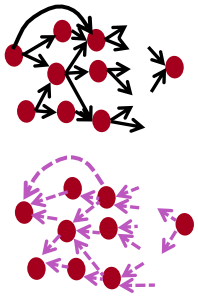
1. Fprop: visit nodes in topo-sort order
   - Compute value of node given predecessors
2. Bprop:
   - initialize output gradient = 1
   - visit nodes in reverse order:
     Compute gradient wrt each node using gradient wrt successors

$\{y_1, y_2, \ldots y_n\}$ = successors of x

$$\frac{\partial z}{\partial x} = \sum_{i=1}^{n} \frac{\partial z}{\partial y_i} \frac{\partial y_i}{\partial x}$$

9

Automatic Differentiation



- The gradient computation can be automatically inferred from the symbolic expression of the fprop.
- Each node type needs to know how to compute its output and how to compute the gradient wrt its inputs given the gradient wrt its output.
- Easy and fast prototyping. See:
  - Theano (Python),
  - TensorFlow (Python/C++), or
  - Autograd (Lua/C++ for Torch)

10

Deep Learning General Strategy and Tricks

11

General Strategy

1. Select network structure appropriate for problem
   1. Structure: Single words, fixed windows vs. convolutional vs. recurrent/recursive sentence based vs. bag of words
   2. Nonlinearities [covered earlier]
2. Check for implementation bugs with gradient checks
3. Parameter initialization
4. Optimization
5. Check if the model is powerful enough to overfit
   1. If not, change model structure or make model "larger"
   2. If you can overfit: Regularize

12

Gradient Checks are Awesome!

- Allows you to know that there are no bugs in your neural network implementation! (But makes it run really slow.)
- Steps:
  1. Implement your gradient
  2. Implement a finite difference computation by looping through the parameters of your network, adding and subtracting a small epsilon (~10^{-4}) and estimate derivatives

$$g_i(\theta) \approx \frac{J(\theta^{(i+)}) - J(\theta^{(i-)})}{2 \times \text{EPSILON}}. \qquad \theta^{(i+)} = \theta + \text{EPSILON} \times \vec{e}_i$$

  3. Compare the two and make sure they are almost the same

13

Parameter Initialization

- Parameter Initialization can be very important for success!
- Initialize hidden layer biases to 0 and output (or reconstruction) biases to optimal value if weights were 0 (e.g., mean target or inverse sigmoid of mean target).
- Initialize weights ~ Uniform(−*r*, *r*), *r* inversely proportional to fan-in (previous layer size) and fan-out (next layer size):

$$\sqrt{6/(\text{fan-in} + \text{fan-out})}$$

  for tanh units, and 4x bigger for sigmoid units [Glorot AISTATS 2010]
- Make initialization slighly positive for ReLU – to avoid dead units

14

Stochastic Gradient Descent (SGD)

- Gradient descent uses total gradient over all examples per update
  - Shouldn't be used. Very slow.
- SGD updates after each example:

$$\theta^{(t)} \leftarrow \theta^{(t-1)} - \epsilon_t \frac{\partial L(z_t, \theta)}{\partial \theta}$$

- L = loss function, z_t = current example, θ = parameter vector, and ε_t = learning rate.
- You process an example and then move each parameter a small distance by subtracting a fraction of the gradient
- ε_t should be small … more in following slide
- Important: apply all SGD updates at once after backprop pass

15

Stochastic Gradient Descent (SGD)

- Rather than do SGD on a single example, people usually do it on a *minibatch* of 32, 64, or 128 examples.
- You sum the gradients in minibatch (and scale down learning rate)
  - Minor advantage: gradient estimate is much more robust when estimated on a bunch of examples rather than just one.
  - Major advantage: code can run much faster *iff* you can do a whole minibatch at once via matrix-matrix multiplies
- There is a whole panoply of fancier online learning algorithms commonly used now with NNs. Good ones include:
  - Adagrad
  - RMSprop
  - ADAM

16

Learning Rates

- Setting α correctly is tricky
- Simplest recipe: α fixed and same for all parameters
- Or start with learning rate just small enough to be stable in first pass through data (epoch), then halve it on subsequent epochs
- Better results can usually be obtained by using a curriculum for decreasing learning rates, typically in $O(1/t)$ because of theoretical convergence guarantees, e.g. $\epsilon_t = \frac{\epsilon_0 \tau}{\max(t, \tau)}$ with hyper-parameters ε_0 and τ
- Better yet: No hand-set learning rates by using methods like AdaGrad (Duchi, Hazan, & Singer 2011) [but may converge too soon – try resetting accumulated gradients]

17

Attempt to overfit training data

Assuming you found the right network structure, implemented it correctly, and optimized it properly, you can make your model totally overfit on your training data (99%+ accuracy)

- If not:
  - Change architecture
  - Make model bigger (bigger vectors, more layers)
  - Fix optimization
- If yes:
  - Now, it's time to regularize the network

18

Prevent Overfitting: Model Size and Regularization

- Simple first step: Reduce model size by lowering number of units and layers and other parameters
- **Standard L1 or L2 regularization on weights**
- **Early Stopping: Use parameters that gave best validation error**
- Sparsity constraints on hidden activations, e.g., add to cost:

$$KL\left(1/N \sum_{n=1}^{N} a_i^{(n)} \| 0.0001\right)$$

19

Prevent Feature Co-adaptation

Dropout (Hinton et al. 2012) http://jmlr.org/papers/v15/srivastava14a.html

- Training time: at each instance of evaluation (in online SGD-training), randomly set 50% of the inputs to each neuron to 0
- Test time: halve the model weights (now twice as many)
- This prevents feature co-adaptation: A feature cannot only be useful in the presence of particular other features
- A kind of middle-ground between Naïve Bayes (where all feature weights are set independently) and logistic regression models (where weights are set in the context of all others)
- Can be thought of as a form of model bagging
- It acts as a strong regularizer; see (Wager et al. 2013) http://arxiv.org/abs/1307.1493

20

Deep Learning Tricks of the Trade

- Y. Bengio (2012), "Practical Recommendations for Gradient-Based Training of Deep Architectures" http://arxiv.org/abs/1206.5533
  - Unsupervised pre-training
  - Stochastic gradient descent and setting learning rates
  - Main hyper-parameters
    - Learning rate schedule & early stopping, Minibatches, Parameter initialization, Number of hidden units, L1 or L2 weight decay, ...

- Y. Bengio, I. Goodfellow, and A. Courville (in press), "Deep Learning". MIT Press, ms. http://goodfeli.github.io/dlbook/
  - Many chapters on deep learning, including optimization tricks
  - Some more recent stuff than 2012 paper

21

Sharing statistical strength

22

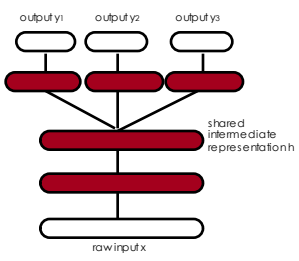
Sharing Statistical Strength

- Besides very fast prediction, the main advantage of deep learning is statistical
- Potential to learn from less labeled examples because of sharing of statistical strength:
  - Unsupervised pre-training
  - Multi-task learning
  - Semi-supervised learning

23

Multi-Task Learning

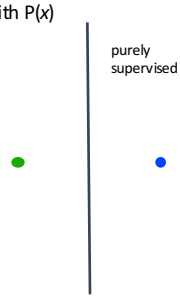
- Generalizing better to new tasks is crucial to approach AI
- Deep architectures learn good intermediate representations that can be shared across tasks
- Good representations make sense for many tasks



24

Semi-Supervised Learning

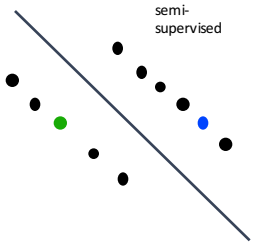
- Hypothesis: P(*c*|*x*) can be more accurately computed using shared structure with P(*x*)



25

Semi-Supervised Learning

- Hypothesis: P(*c*|*x*) can be more accurately computed using shared structure with P(*x*)



26

Advantages of Deep Learning Part 2

27

#4 Unsupervised feature learning

Today, most practical, good NLP& ML methods require labeled training data (i.e., supervised learning)

But almost all data is unlabeled

Most information must be acquired **unsupervised**

Fortunately, a good model of observed data can really help you learn classification decisions

Commentary: This is more the dream than the reality; most of the recent successes of deep learning have come from regular supervised learning over very large data sets

28

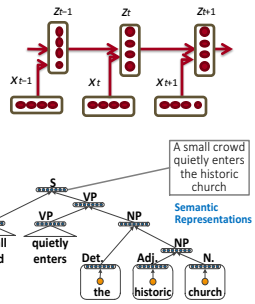
#5 Handling the recursivity of language

Human sentences are composed from words and phrases

We need compositionality in our ML models

Recursion: the same operator (same parameters) is applied repeatedly on different components

Recurrent models: Recursion along a temporal sequence



29

#6 Why now?

Despite prior investigation and understanding of many of the algorithmic techniques …

Before 2006 training deep architectures was **unsuccessful** ☹

What has changed?

- New methods for unsupervised pre-training (Restricted Boltzmann Machines = RBMs, autoencoders, contrastive estimation, etc.) and deep model training developed
- More efficient parameter estimation methods
- Better understanding of model regularization
- **More data** and **more computational power**

Deep Learning models have already achieved impressive results for HLT

Neural Language Model
[Mikolov et al. Interspeech 2011]

Model \ WSJ ASR task	Eval WER
KN5 Baseline	17.2
Discriminative LM	16.9
Recurrent NN combination	14.4

MSR MAVIS Speech System
[Dahl et al. 2012; Seide et al. 2011; following Mohamed et al. 2011]

"The algorithms represent the first time a company has released a deep-neural-networks (DNN)-based speech-recognition algorithm in a commercial product."

Acoustic model & training	Recog \ WER	RT03S FSH	Hub5 SWB
GMM 40-mix, BMMI, SWB 309h	1-pass –adapt	27.4	23.6
DBN-DNN 7 layer x 2048, SWB 309h	1-pass –adapt	18.5 (–33%)	16.1 (–32%)
GMM 72-mix, BMMI, FSH 2000h	*k*-pass +adapt	18.6	17.1

31

Deep Learn Models Have Interesting Performance Characteristics

Deep learning models can now be very fast in some circumstances

- SENNA [Collobert et al. 2011] can do POS or NER faster than other SOTA taggers (16x to 122x), using 25x less memory
- WSJ POS 97.29% acc; CoNLL NER 89.59% F1; CoNLL Chunking 94.32% F1

Changes in computing technology favor deep learning

- In NLP, speed has traditionally come from exploiting sparsity
- But with modern machines, branches and widely spaced memory accesses are costly
- Uniform parallel operations on dense vectors are faster

These trends are even stronger with multi-core CPUs and GPUs

32

TREE STRUCTURES WITH CONTINUOUS VECTORS

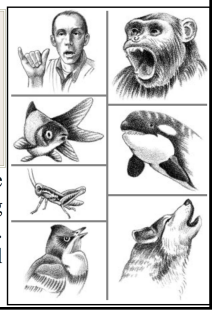
SCIENCE'S COMPASS • REVIEW

REVIEW: NEUROSCIENCE

The Faculty of Language: What Is It, Who Has It, and How Did It Evolve?

Marc D. Hauser,[1]* Noam Chomsky,[2] W. Tecumseh Fitch[1]

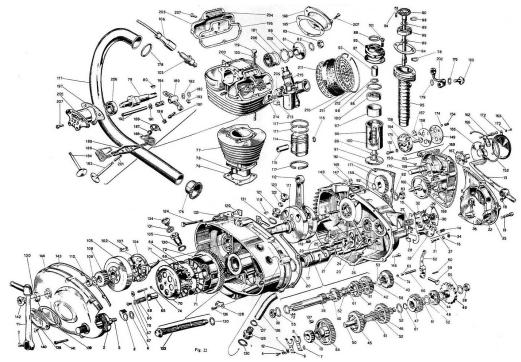
We argue that an understanding of the faculty of language requires substantial interdisciplinary cooperation. We suggest how current developments in linguistics can be profitably wedded to work in evolutionary biology, anthropology, psychology, and neuroscience. We submit that a distinction should be made between the faculty of language in the broad sense (FLB) and in the narrow sense (FLN). FLB includes a sensory-motor system, a conceptual-intentional system, and the computational mechanisms for recursion, providing the capacity to generate an infinite range of expressions from a finite set of elements. We hypothesize that FLN only includes recursion and is the only uniquely human component of the faculty of language. We further argue that FLN may have evolved for reasons other than language, hence comparative studies might look for evidence of such computations outside of the domain of communication (for example, number, navigation, and social relations).



If a martian graced our planet, it would be struck by one remarkable similarity among Earth's living creatures and a key difference. Concerning similarity, it would note that all

34

Compositionality



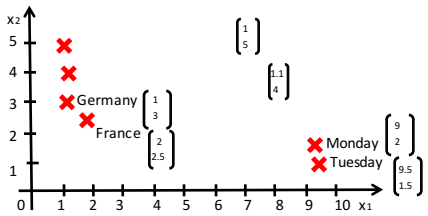
We need more than word vectors and bags! What of larger semantic units?

How can we know when larger units are similar in meaning?

- *The snowboarder is leaping over the mogul*
- *A person on a snowboard jumps into the air*

People interpret the meaning of larger text units – entities, descriptive terms, facts, arguments, stories – by **semantic composition** of smaller elements

Representing Phrases as Vectors



Vector for single words are useful as features but limited

the country of my birth
the place where I was born

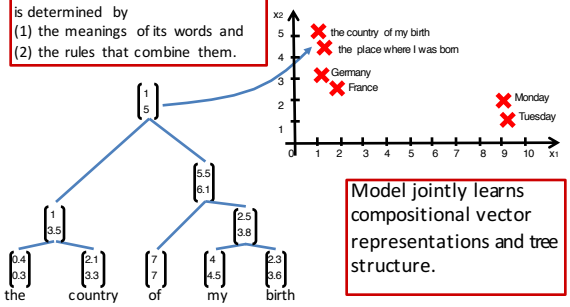
Can we extend ideas of word vector spaces to phrases?

If the vector space captures syntactic and semantic information, the vectors can be used as features for parsing and interpretation

How should we map phrases into a vector space?

Use the principle of compositionality!

The meaning (vector) of a sentence is determined by
(1) the meanings of its words and
(2) the rules that combine them.

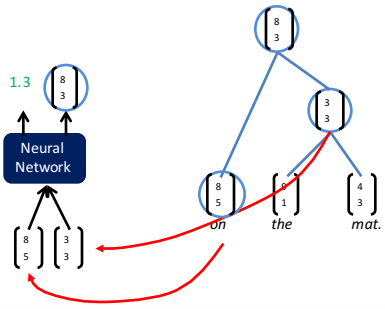


Model jointly learns compositional vector representations and tree structure.

Tree Recursive Neural Networks (Tree RNNs)

Computational unit:
Simple Neural Network layer, applied recursively

(Goller & Küchler 1996, Costa et al. 2003, Socher et al. ICML, 2011)

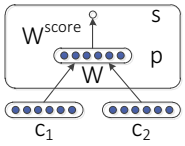


Version 1: Simple concatenation Tree RNN

$p = \tanh\left(W\begin{bmatrix}c_1\\c_2\end{bmatrix} + b\right)$,

where tanh:

score = $V^T p$



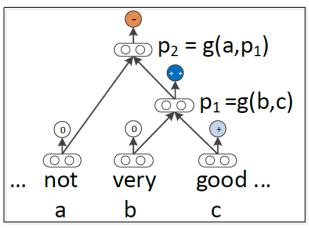
Only a single weight matrix = composition function!

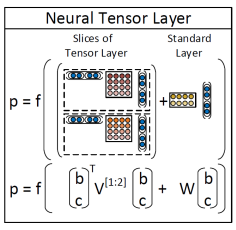
No really interaction between the input words!

Not adequate for human language composition function

Version 4: Recursive Neural Tensor Network

Allows the two word or phrase vectors to interact multiplicatively





Beyond the bag of words: Sentiment detection

Is the tone of a piece of text positive, negative, or neutral?

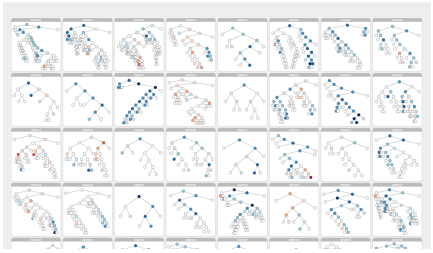
- Sentiment is that sentiment is "easy"
- Detection accuracy for longer documents ~90%, BUT

… … loved … … … … … great … … … … … …
impressed … … … … … … marvelous … … … …

With this cast, and this subject matter, the movie should have been funnier and more entertaining.

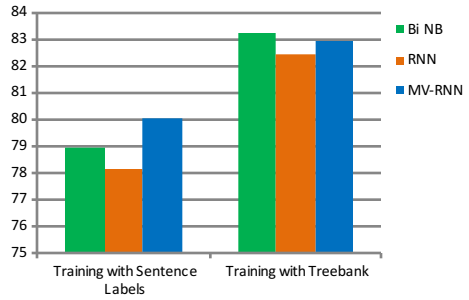
Stanford Sentiment Treebank

- 215,154 phrases labeled in 11,855 sentences
- Can actually train and test compositions



http://nlp.stanford.edu:8080/sentiment/

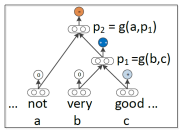
Better Dataset Helped All Models

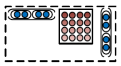


- Hard negation cases are still mostly incorrect
- We also need a more powerful model!

Version 4: Recursive Neural Tensor Network

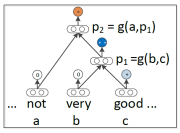
Idea: Allow both additive and mediated multiplicative interactions of vectors

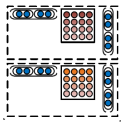




$$\begin{bmatrix} b \\ c \end{bmatrix}^T V \begin{bmatrix} b \\ c \end{bmatrix}$$

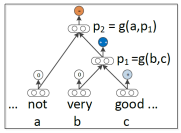
Recursive Neural Tensor Network

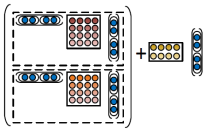




$$\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:2]} \begin{bmatrix} b \\ c \end{bmatrix}$$

Recursive Neural Tensor Network

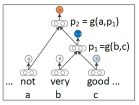




$$\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:2]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix}$$

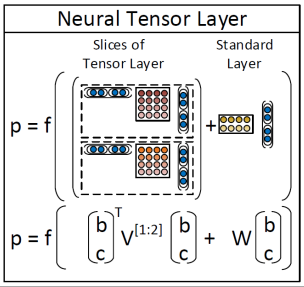
Recursive Neural Tensor Network

- Use resulting vectors in tree as input to a classifier like logistic regression
- Train all weights jointly with gradient descent



Neural Tensor Layer

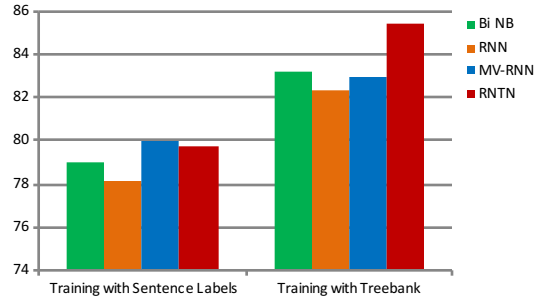
Slices of Tensor Layer | Standard Layer



$$p = f\left(\begin{bmatrix} b \\ c \end{bmatrix}^T V^{[1:2]} \begin{bmatrix} b \\ c \end{bmatrix} + W \begin{bmatrix} b \\ c \end{bmatrix} \right)$$

Positive/Negative Results on Treebank

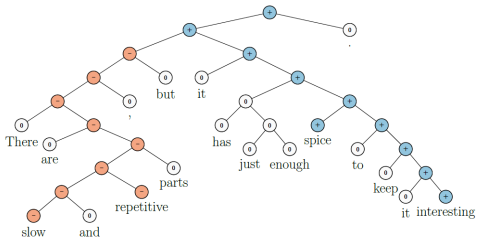
Classifying Sentences: Accuracy improves to 85.4



Note: for more recent work, see Le & Mikolov (2014), Irsoy & Cardie (2014), Tai et al. (2015)

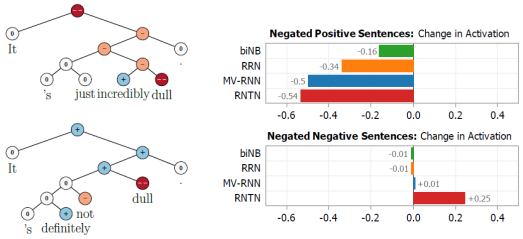
Experimental Results on Treebank

- RNTN can capture constructions like *X but Y*
- RNTN accuracy of 72%, compared to MV-RNN (65%), biword NB (58%) and RNN (54%)



Negation Results

When negating negatives, positive activation should increase!



Demo: http://nlp.stanford.edu:8080/sentiment/

Conclusion

Developing intelligent machines involves being able to recognize and exploit compositional structure

It also involves other things like top-down prediction, of course

Work is now underway on how to do more complex tasks than straight classification inside deep learning systems