Deep Learning

Chapter 12 - Applications

Cherif Jazra
@jazracherif
Part 1 Part 2

Comments section is open

Chapter Content

Chapter Content

Introduction

How do we use deep learning to solve various applications?

Chapter is divided into 2 main parts:

- Large-scale systems and how they are implemented
- Specific Application areas

While NN tries to broadly solve learning problems, **specialization** is still needed, as inputs differs for example between Vision problems and Language problems.

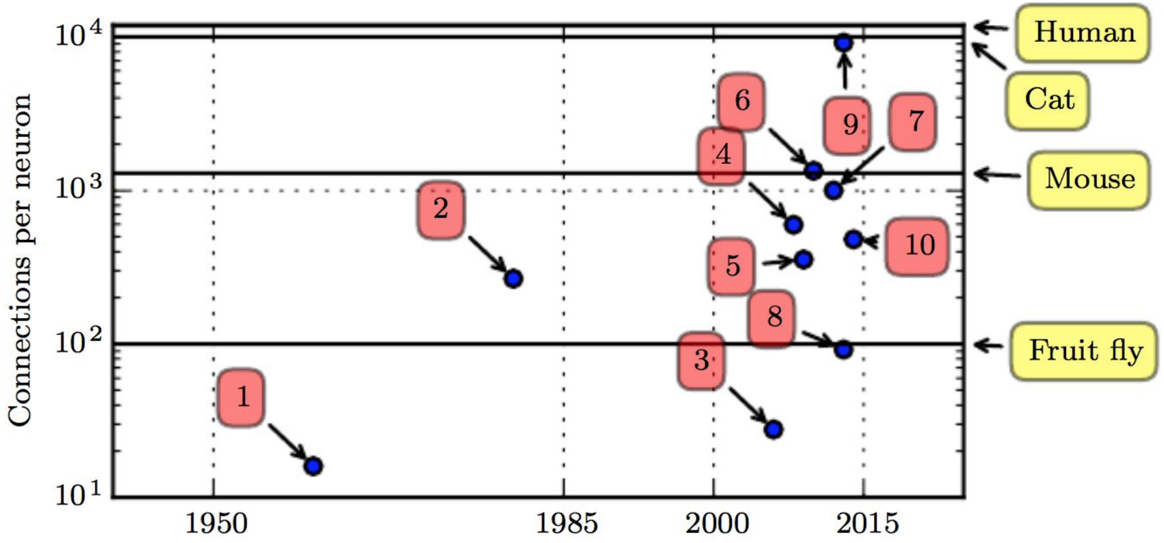
In many cases, we are trying to achieve **acceleration** of the process or reduction in **memory** cost.

12.1 Large-Scale Deep Learning

NN are based on the philosophy of **Connectionism**.

We need **large** amount of neurons to exhibit intelligent behavior.

Improvement in **accuracy** and **task complexity** since the 1980s is due to the increase in the sizes of networks



1. **Adaptive linear element** (Widrow and Hoff, 1960)
2. **Neocognitron** (Fukushima, 1980)
3. **GPU-accelerated convolutional network** (Chellapilla et al., 2006)
4. **Deep Boltzmann machine** (Salakhutdinov and Hinton, 2009a)
5. **Unsupervised convolutional network** (Jarrett et al., 2009)
6. **GPU-accelerated multilayer perceptron** (Ciresan et al., 2010)
7. **Distributed autoencoder** (Le et al., 2012)
8. **Multi-GPU convolutional network** (Krizhevsky et al., 2012)
9. **COTS HPC unsupervised convolutional network** (Coates et al., 2013)
10. **GoogLeNet** (Szegedy et al., 2014a)

12.2 Fast CPU Implementations

Today, training using a CPU is considered insufficient. GPU with networked machines is the standard.

But there were efforts to make CPU implementation efficients:
Ex: Vanhoucke (2011), obtained 3x speedup by using a **fixed-point** rather than **floating point** arithmetic.

Other strategies:

- Optimizing data structures to avoid cache misses.
- Using Vector instructions.

⇒ **Take away**: Careful specialisation of numerical computation routines can yield large payoff, often neglected by ML engineers!!

12.2 GPU Implementations

GPU: Graphics Processing Units

Specialized hardware for graphic applications, historically largely driven by the the video games industry.

Why video games?

- Many operations in parallel required
- Matrix multiplications and division in parallel to convert 3D coordinates (character and environment) into 2-D screen.
- All pixel colors determined in parallel.

⇒ Optimized for many independent parallel computation, few branching.

12.2 GPU Implementations

Neural Networks require the same kind of performance characteristics:

- Large and numerous buffers of **parameters**, **activation** values, **gradient** values
- Large **memory bandwidth** required (larger than typical cache)
- NN as a graph can be divided into multiple individual neurons that can be processed **independently** from others in the same layer

Early trials of using GPU:

- 2005 Steinkrau: 2 layers, fully connected NN on a GPU achieves 3x speedup compared to CPU
- 2006 Chellapilla: Supervised convolutional networks achieves speedup over CPU.

12.2 GPU Implementations

Over time, GPUs became more flexible, and their popularity increased with the introduction of **General Purpose** GPU (GP-GPU).

- Arbitrary code can be executed.
- NVIDIA CUDA programming language (proprietary, launched in 2006) C like language.

But it is still difficult to program:

- Different kind of speed optimizations than CPU, i.e speedup obtained by not using caching
- Inherently multithreaded, careful thread coordination needed → **Coalescing** techniques are used: when several thread can read/write simultaneously with 1 single memory transaction.
- Thread in group (a warp) should executes the same instruction simultaneously. Makes branching code more difficult.

High level libraries available: Pylearn2 and Thenao, Cuda-convnet, Tensorflow, Torch.

12.1.3 Large-Scale Distributed Implementations

2 types of parallelism for distributing the workload of training and inference across many machines

Data parallelism (DP): each input example is handled by a different machine
Model Parallelism (MP): each part of the model is handled by a different machine

DP is usually harder for training. We could increase the batch size for single SGD step but get less than linear returns in optimization performance.
Better to allow multiple machines to compute multiple gradient descent steps in parallel
But gradient descent is a sequential algorithm!

⇒ Asynchronous Stochastic Gradient Descent **A-SGD** (Bengio 2001, Recht 2001)
Several processors share the memory representing the parameters:

- Read parameter without a lock
- Compute gradient
- Increment parameters without a lock

Dean 2012: pioneers the multimachine implementation, uses **parameter server** rather than use shared memory

12.1.4 Model Compression

In many application, optimization **inference** is more important than optimizing **training** (ex: a model trained and deployed on a mobile phone)

How to reduce the **speed** and **memory** cost of inference?
Use **Model Compression** (Bucilua et al. 2006)

- Goal is to replace original model with a smaller model
- Good when the size of the original model was driven by a need to prevent overfitting.
- Large model is often an Ensemble of several independently trained models, and accurately learns to represent the true distribution function f(x)
- We can generate infinitely more data samples by drawing from a distribution similar to the test input and labeled by f(x)
- With so much more data, we can tune the capacity better to match a smaller network.

Other method: a way to transfer the generalization ability of a larger (cumbersome) model into a smaller one (Distillation, in Hinton et al. 2015)

- Uses the output probabilities of the larger model as a soft target for the smaller one (instead of using the hard values of the truth target)

12.1.5 Dynamic Structure

Acceleration can be achieved when the computation graph are built with dynamic structure
→ Allows decomposition of the processing in a dynamic way, depending on input values

Conditional Computation: determine which subset of hidden units to compute given information from the input.

Multiple types of Dynamic structures:

Cascade of classifiers.
Ex: Goal is to detect presence of a rare objects. Use less sophisticated classifier which rejects inputs that don’t contain the object.
First classifier: Low capacity, high recall: don’t reject an item if its present
Last classifier: has high precision

Abandon an example as soon as one classifier rejects it
→ we can reject more cases (since object is rare), more quickly, with small cost.

12.1.5 Dynamic Structure

How does cascade achieve high capacity? Either of the two:

1. Late members in the cascade should have high capacity
2. All have low capacity, but the total has high capacity. (ex viola and jones 2001, a cascade of boosted trees for face detection: various sliding windows in a picture are examined, rejected if no face)

Decision tree, each node decides which of the subset trees will be evaluated. One could use a neural network to learn the decision function at each tree (Guo and Gelfand, 1992)

Gater

learns which of different expert system to use depending on inputs.

- Mixture of experts: final output is the weighted combination of output from each expert
- Hard mixture of expert: only 1 expert is used for final output → reduces computation

Switch

Like an attention mechanism, selectively turns on off inputs to hidden units

Soft switch more popular than hard switch, used for NLP applications

12.1.5 Dynamic Structure

Problems:

- Decreased **degree of parallelism**, as different inputs could need different code branches
- Fewer efficient operation: matrix multiplication / batch convolution / mini batch
- Specialized functions would be harder to efficiently implement on CPU or GPU (lack of coalesced memory transaction)
- Partitioning examples in group may lead to load balancing issues, underuse of machines.

12.1.6 Specialized Hardware Implementation of Deep Learning

Speed of CPU/GPU cores have slowed down, specialization needed to stay ahead.

Forms of specialized format:

- ASIC
    - Digital: binary representation
    - Analog: voltage/current
- FPGA

Degree of precision can be less than 32/64 bits, to speed up inference
8/16 bits of precision may be enough for inference (See Vanhoucke et al, 2011, Courbariaux et al, 2015, Gupta et al. 2015).

Use dynamic fixed-point representation (shared range between weights in one layer for example)
→ reduces hardware surface area, power requirements, and computing time.

12.2 Computer Vision

One of the most active area of research, because vision is effortless for human but hard for computers.

Most popular benchmarks are for **object recognition** and **Optical Character Recognition** (OCR)

Popular type of vision problems:

- Which object is present in an image
- Annotating an image with bounding boxes
- Transcribing a sequence of symbols from an image
- Labeling each pixel with the identity of the object it belongs to
- Image synthesis for restoration, repairing defects, or removing objects from images.

12.2.1 Preprocessing

Usually little preprocessing

- 1 requirement is to standardize pixel values to be in range [0,1] or [-1,1]
- Often all images must be of same size → cropping / scaling required
- Some convNet models accept variable sized input and adjust the size of their pooling regions. (waibel et al, 1989)
- Dataset augmentation: Excellent way to reduce generalization error
- Related method at test time: Show many versions of the same input and the model vote: ex same image cropped slightly at different locations
- Reduce amount of variation the model should learn. Helps reduce generalization error and lower model capacity. ex : Alexnet subtracts the mean across training examples at each pixel

12.2.1.1 Contrast Normalization

Amount of contrast: A form of variation that can be removed from an image.

What is contrast? The magnitude of the difference between the **bright** and the **dark** pixels in an image = Standard Deviation for pixel values

Assume 3 channel pixel intensity is $\mathbf{X} \in \mathbb{R}^{r \times c \times 3}$

$\Rightarrow$ Contrast = $\sqrt{\frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3}\left(X_{i,j,k} - \bar{\mathbf{X}}\right)^2}$

$$\bar{\mathbf{X}} = \frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3} X_{i,j,k}.$$

12.2.1.1 Contrast Normalization

Global Contrast Normalization (GCN)

Rescaling pixels such as the standard deviation is a constant value **s**

$$X'_{i,j,k} = s.\frac{X_{i,j,k} - \bar{X}}{contrast}$$

Problem: pictures with zero or very little contrast.

⇒ use small positive regularization parameter **λ** or use a floor value **ε**

$$X'_{i,j,k} = s\frac{X_{i,j,k} - \bar{X}}{\max\left\{\epsilon, \sqrt{\lambda + \frac{1}{3rc}\sum_{i=1}^{r}\sum_{j=1}^{c}\sum_{k=1}^{3}\left(X_{i,j,k} - \bar{X}\right)^2}\right\}}$$

12.2.1.1 Contrast Normalization

Parameter settings

Large images with interesting objects usually have contrast: set $\boldsymbol{\lambda}$= 0, $\boldsymbol{\varepsilon}$ = 10^{-8}

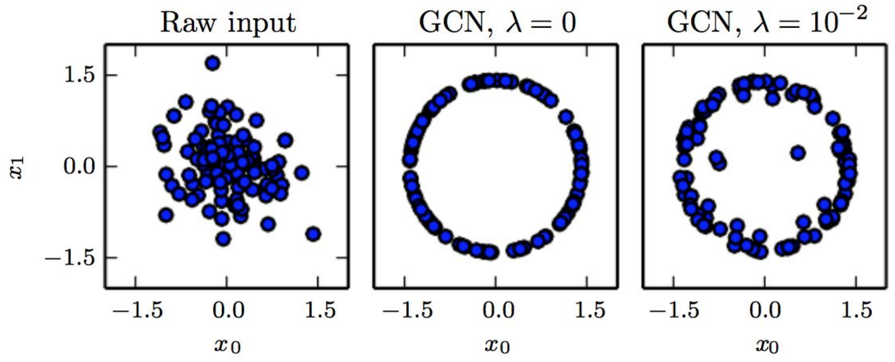
Small images cropped randomly are more likely to have constant intensity: Set $\boldsymbol{\varepsilon}$ = 0, $\boldsymbol{\lambda}$ = 10

S = 1 is common. Choose it in a way where each individual pixel has standard deviation close to 1

12.2.1.1 Contrast Normalization

Interpretation of GCN

A mapping of examples to a spherical shell.
In simplest case (λ=ε=0), the norm of each point will a constant



⇒ Directions in space are easier to learn than exact locations
⇒ Responding to multiple distances in the same direction will require hidden units with collinear weights vectors but different biases

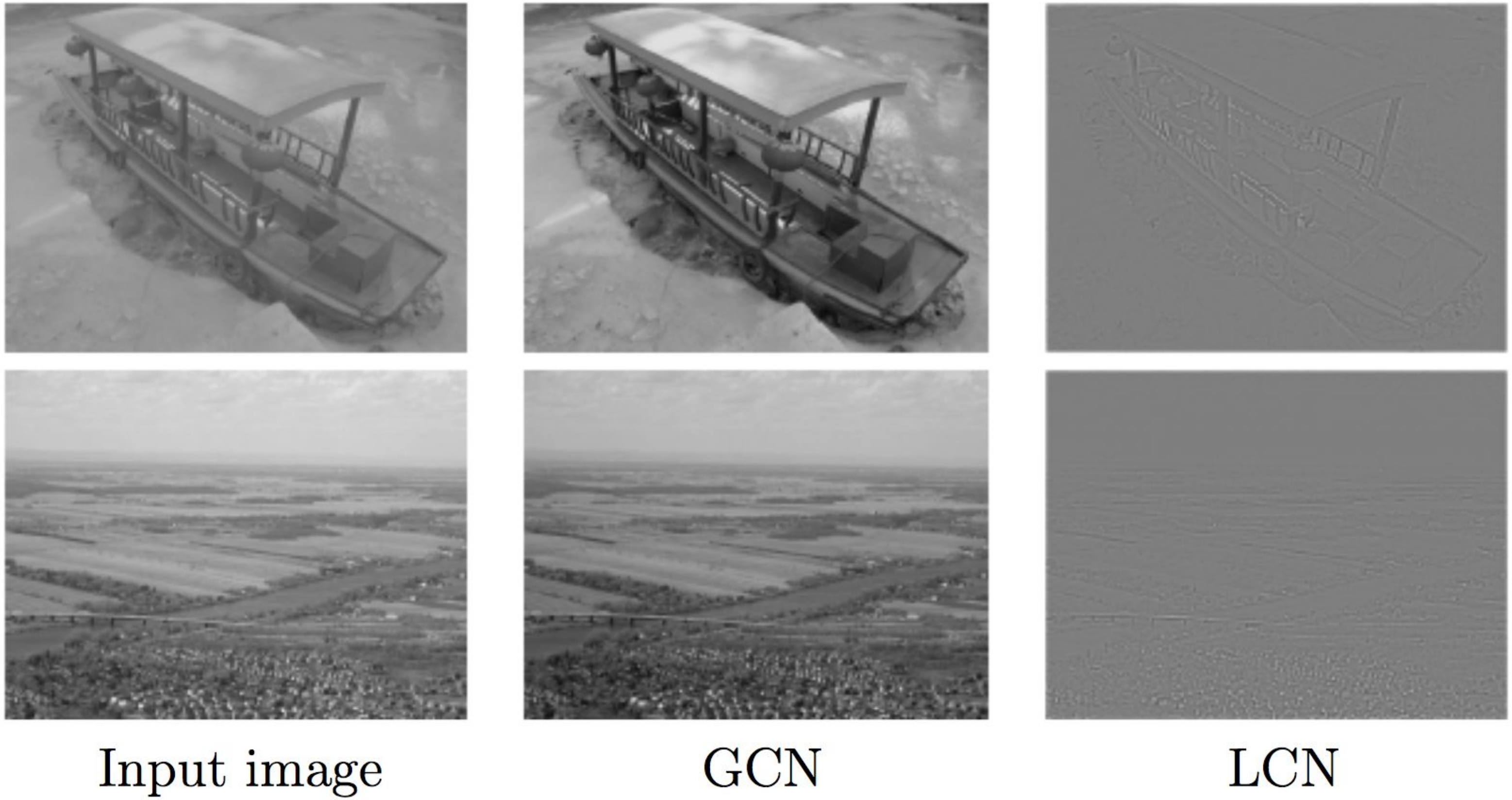
12.2.1.1 Contrast Normalization

Problem with GNC:
Edges and **corners** are not strongly highlighted. GCN ensure large regions of different contrast will result in different in brightness but local edges and corners will not stand out.

Local Contrast Normalization
Normalize contrast across small window within an image
How? Normalize each pixel using neighboring pixels only

- Mean + std
- Weighted mean + std using Gaussian weights centered on the pixel. Could apply different weights for different color channels

12.2.1.1 Contrast Normalization



12.2.1.2 Dataset Augmentation

Particular useful for object detection tasks

Object class is invariant to many geometric transformation: random translations, rotations, flips

More advanced techniques:

- Random perturbation of the colors of an image (Krizhevksy et al., 2012)
- Nonlinear geometric distortions of the input (LeCun et al., 1998)

12.3 Speech Recognition

Automatic Speech Recognition (ASR)

Goal is to map an acoustic signal containing spoken language into the proper sequence of words.

Split the audio into 20ms frame symbols: $X = (x^{(1)}, x^{(2)}, .. \, x^{(T)})$

Then, create a function f^* that will give the most probable output sequence:

$$f^*_{ASR}(X) = \arg\max_y P^*(y \mid X = \mathcal{X})$$

Where P^* is the true conditional distribution

A typical pipeline of ASR

Raw acoustic sound → intermediary feature representation → phonemes (the perceptually distinct units of sound) → Word sequence

12.3 Speech Recognition

Historical Development

From 1980s to 2009-2012, SOA was GMM-HMM
GMM: models the relation from acoustic features ⇔ Phonemes
HMM: models the sequence of phonemes

However, many audio systems developed neural net approach during this period: Bourlard and Wellekens, 1989, Waibel et al., 1989, Robinson and Fallside, 1991, Bengio et al., 1991 1992, Konig et al., 1996

TIMIT: the equivalent of MNIST for phoneme recognition
NN achieved similar results as HMM-GMM ~ 26% error rate

But complex engineering effort to deploy it

12.3 Speech Recognition

Starting 2009, use of much larger and deeper neural networks.
First, Unsupervised learning (RBM) techniques used to pretrain layers in the feedforward network
⇒ Improved accuracy from 26% to 20.7%

- Speaker adaptive feature improved error

Eventually, RBM replaced by using ReLus and Dropout. See Hinton et .al 2012 for the breakthroughs

2013: use of convolutional network: model the spectral representation of the acoustic sound as an image. (Sainath et al., 2013)

Finally, End-to-end deep learning without HMM: first breakthrough with Graves et al., 2013: Deep LSTM with MAP inference. Achieved phoneme rates error of 17.7%

12.4 Natural Language Processing

Goal: the use of human languages by computers.

Challenges with human language:

- Ambiguous meaning
- Defy formal description

Ex: Machine Translation

Many NLP system are based on **Language models**: they define a probability distribution over sequence of words, character, or bytes.

12.4.1 n-grams

N-grams are an example of **fixed-length** sequence of tokens used to create a language model.

Captures the probability distribution of a sequence of words.

Consider the **chain rule** for the probability of a sequence of words (x_1, .., x_τ):

$$p(x_1, .., x_\tau) = P(x_\tau | x_1, ..x_{\tau-1}) * p(x_1, ..x_{\tau-1})$$
$$= P(x_\tau | x_1, ..x_{\tau-1}) * P(x_{\tau-1} | x_1, ..x_{\tau-2}) * p(x_1, ..x_{\tau-2})$$

By expanding the rule for τ - n times, we get Eq (12.5)

$$P(x_1, \ldots, x_\tau) = P(x_1, \ldots, x_{n-1}) \prod_{t=n}^{\tau} P(x_t \mid x_{t-n+1}, \ldots, x_{t-1}).$$

12.4.1 n-grams

Maximum Likelihood estimate: just count how many times each possible n-grams occur in the training set

Unigram: n = 1
Bigram : n = 2
Trigram : n = 3

Gram = from Greek, meaning “something written”

Long history of using n-gram:
Jelinek and mercer, 1980
Katz, 1987
Chen and Goodman, 1999

12.4.1 n-grams

Typically, train both **n-gram** and **n-1 gram** model together to compute the conditional probability of the language model:

$$P(x_t \mid x_{t-n+1}, \ldots, x_{t-1}) = \frac{P_n(x_{t-n+1}, \ldots, x_t)}{P_{n-1}(x_{t-n+1}, \ldots, x_{t-1})}$$

Ex: using 2-gram/3-gram model, What is the probability of the sentence “THE DOG RAN AWAY”

P(“THE DOG RAN AWAY”) = P_3(THE DOG RAN) * P(AWAY | DOG RAN)

= P_3(THE DOG RAN) * P_3(DOG RAN AWAY) / P_2(DOG RAN)

12.4.1 n-grams

Problem: many tuples may not appear in the training set.

P_n or P_{n-1} might be zero creating either an negative infinite log likelihood or an undefined ratio.

Possible Solutions:

1. Use **Smoothing**: shifts some probability mass to unobserved tuples (more in Chen and Goodman, 1999). Same as adding a Dirichlet prior over the count parameters.
2. **Back-off methods**: use mixture model with higher and lower order n-gram (lower are less likely to have zero or close to zero values)

Curse of Dimensionality: There are $|V|^n$ possible n-grams with V, the very large vocabulary size

No good neighbor distance metrics for n-grams: any two different words have the same distance from each other in the one-hot vector space.

12.4.1 n-grams

Improvements: Class-based language models

Word categories in which words share statistical strength.

Partition words using a clustering algorithms, based on co-occurrence frequencies with other words (clusters are the categories)

Use class category id as the **conditioning** context, thus reduces the number of needed combinations

Caveat: much information is lost in kind of representation

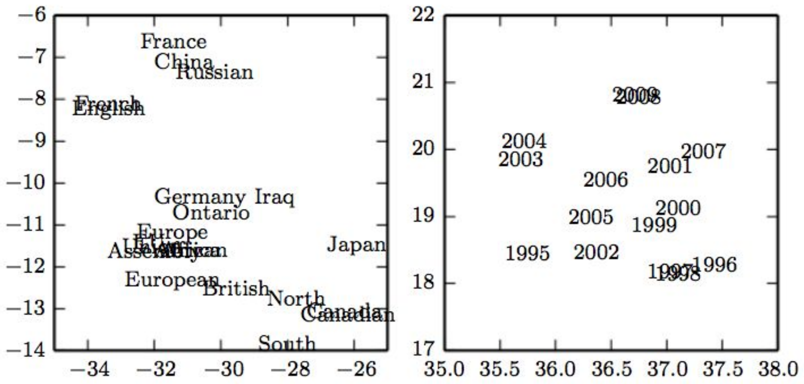
A mix of word-based and class-based models may be used

12.4.2 Neural Language Models

Overcome the Curse of Dimensionality by using a **distributed representation of words** (Bengio et al, 2001)

The semantic meaning of words is captured in a low dimension space, where the relationship between words is captured.
→ word embeddings

In the original space, for any 2 words w_i and w_j one-hot vector, $d_{wi,\ wj} = \sqrt{2}$
In the embedding space, distance between words reflects the fact that words appear frequently in the same context



12.4.3 High-Dimensional Output

How should we deal with large output vocabulary?

|V| ~ hundreds of thousands of words

Naive implementation: SOFTMAX(x_N*w + b)

High memory cost: **w** has large dimensions
High computational cost: Softmax must be normalized across all |V| outputs

$$a_i = b_i + \sum_j W_{ij} h_j \quad \forall i \in \{1, \ldots, |\mathbb{V}|\},$$

$$\hat{y}_i = \frac{e^{a_i}}{\sum_{i'=1}^{|\mathbb{V}|} e^{a_{i'}}}.$$

12.4.3.1 Use of a Short List

Solution?

Some early solutions limit the output vocabulary to 10-20k words.

Split vocabulary into **shortlist** (L most frequent words) and **tail list** (most rare words: size =V \ L)

Use an extra sigmoid output unit to predict whether the word appearing after the context is in the tail list.
⇒ Combine **Neural language** with **n-gram** model

Language Model
(L words)

Sigmoid

$$\begin{aligned} P(y=i \mid C) = & 1_{i\in\mathbb{L}} P(y=i \mid C, i\in\mathbb{L})(1-P(i\in\mathbb{T} \mid C)) \\ & + 1_{i\in\mathbb{T}} P(y=i \mid C, i\in\mathbb{T}) P(i\in\mathbb{T} \mid C) \end{aligned}$$

N-gram model (V-L words)

12.4.3.2 Hierarchical Softmax

Decompose probabilities hierarchically

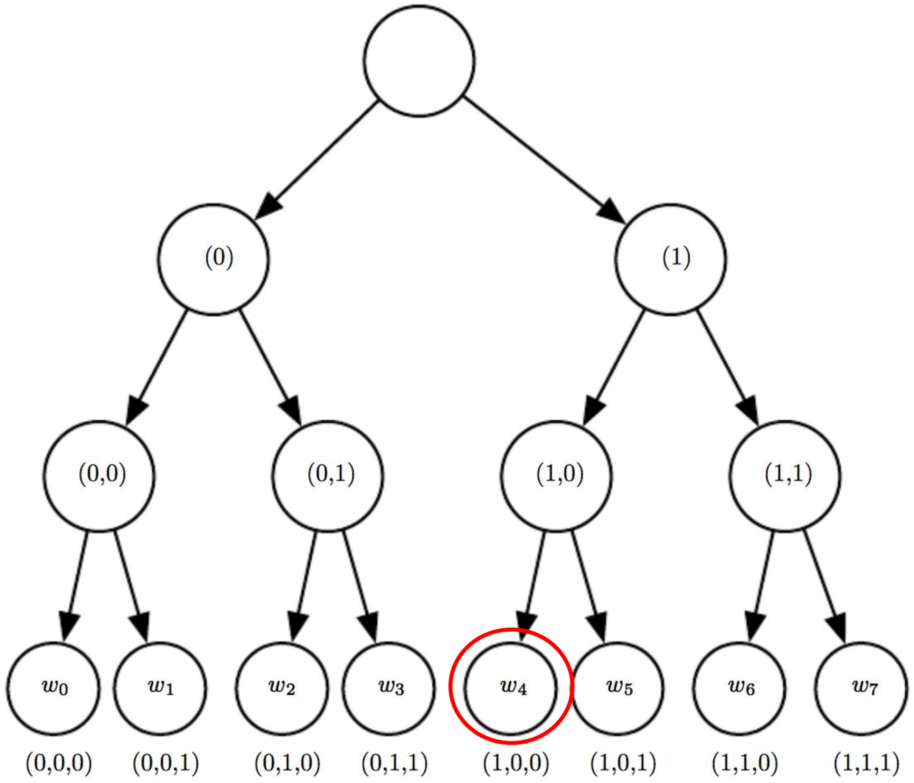
Reduces number of computations from O(|V|) → O(log|V|)

Builds multilevel nested categories of words into a balanced tree.

Probability of choosing a words = product of probabilities of choosing the branch leading to word at every node

12.4.3.2 Hierarchical Softmax

$$\begin{aligned} P(\mathrm{y} = w_4) &= P(\mathrm{b}_0 = 1, \mathrm{b}_1 = 0, \mathrm{b}_2 = 0) \\ &= P(\mathrm{b}_0 = 1) P(\mathrm{b}_1 = 0 \mid \mathrm{b}_0 = 1) P(\mathrm{b}_2 = 0 \mid \mathrm{b}_0 = 1, \mathrm{b}_1 = 0). \end{aligned}$$



12.4.3.2 Hierarchical Softmax

We replace the softmax unit with a tree structure ⇒ No need to calculate the normalizing factor.

But how to compute the conditional probabilities at each node?
Use logistic regression, and provide the same context C as input to all these node.
→ a supervised learning approach, can use cross-entropy as a loss, to maximize the log likelihood of the correct sequence of decisions

How to choose the shape of the tree?
The number of bits associated with a word is ~ log freq of the word
Or define a tree with depth two and a branching factor of $\sqrt{|V|}$

Learning the hierarchy is difficult: discret, not amenable to gradient-based optimization.

Hierarchical Softmax is good for both training and test time.
Computing the probability of all $|V|$ words is still expensive.
Tree structure is also not efficient for providing the most likely words.

In practice, hierarchical softmax gives worse result than sampling-based method

12.4.3.3 Importance Sampling

Avoid explicitly computing the contribution of the gradient from all the words that do not appear in the next position.
→ Approximate the likelihood by sampling from a smaller dataset.

$$\begin{aligned}
\frac{\partial \log P(y \mid C)}{\partial \theta} &= \frac{\partial \log \operatorname{softmax}_y(\boldsymbol{a})}{\partial \theta} \\
&= \frac{\partial}{\partial \theta} \log \frac{e^{a_y}}{\sum_i e^{a_i}} \\
&= \frac{\partial}{\partial \theta}\left(a_y - \log \sum_i e^{a_i}\right) \\
&= \frac{\partial a_y}{\partial \theta} - \sum_i P(y = i \mid C) \frac{\partial a_i}{\partial \theta}
\end{aligned}$$

12.4.3.3 Importance Sampling

$$\frac{\partial \log P(y \mid C)}{\partial \theta} = \underbrace{\frac{\partial a_y}{\partial \theta}}_{\text{Positive Term}} - \underbrace{\sum_i P(y = i \mid C) \frac{\partial a_i}{\partial \theta}}_{\substack{\text{Negative Term} \\ = \text{Expectation}}}$$

The expectation can be computed with **Monte Carlo**, but that requires computing P(i | C) for all i
Instead, use a proposal distribution (q) corrected by appropriate biases to approximate that distribution
⇒ An application of **Importance Sampling** (See section 17.2)

Still, this requires computing weights p_i / q_i where p_i = P(i|C), which requires all a_i to be computed
⇒ Use **biased importance sampling**, where the importance weights are normalized to 1

12.4.3.3 Importance Sampling

Biased importance sampling

For each negative words sampled, the associated gradient is weighted by w$_i$

$$w_i = \frac{p_{n_i}/q_{n_i}}{\sum_{j=1}^{N} p_{n_j}/q_{n_j}}$$

Gives the appropriate importance of these m negative words

$$\sum_{i=1}^{|\mathbb{V}|} P(i \mid C)\frac{\partial a_i}{\partial \theta} \approx \frac{1}{m}\sum_{i=1}^{m} w_i \frac{\partial a_{n_i}}{\partial \theta}$$

Use **unigram** or **bigram distribution** for q, and estimate its parameters from the data
Sampling is also easy from these distribution

12.4.3.4 Noise-Constrastive Estimation and Ranking Loss

Ranking loss: proposed by Collobert and Weston, 2008
View the output as a ranking loss, tries to make the score larger for the correct words.

$$L = \sum_i \max(0, 1 - a_y + a_i)$$

No change in gradient if the score for the observed word is already larger than others by a margin of 1

Problem: doesn't estimate conditional probabilities

See section 18.6 for Noise-contrastive estimation

12.4.4 Combining Neural Language models with n-grams

N-grams achieve high model capacity with little computation to process an example.

Hash table and trees can make the computation almost independent of capacity.

However, doubling a neural network doubles its computation time

Use ensemble to combine both approaches
Could use uniform weighting or weights chosen on a validation set

Mikolov 2001: use large array of models
Mikolov 2001: Pair a NN with a max-entropy model trained jointly

12.4.5 Neural Machine Translation

Goal: Translate a sentence from 1 language to another

Many components

- Proposals module: proposes many candidates including ones with grammatical errors
- Language model: to evaluate the proposals

Early NN explorations for NMT already incorporate the idea of **encoder** and **decoder**:
Allen 1987, Chrisman 1991, Forcada and Neco 1997

12.4.5 Neural Machine Translation

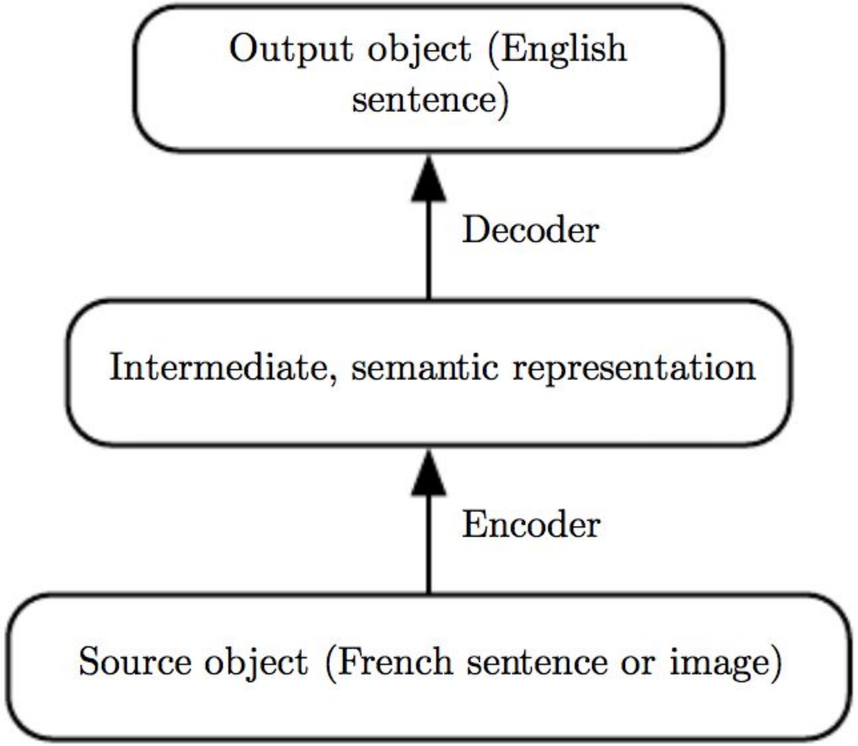
Most recent implementations use RNN to accommodate **variable** length inputs and outputs

Section 10.2.4 Modeling sequence conditioned on context
Section 10.4: Encoder-Decoder Architecture

The encoder will read the input sequence and summarizes into a context “C”
This could be either an RNN (Cho et al. 2014a; Sutskever et al., 2014; Jean et al., 2014) or a ConvNet (Kalchbrenner and Blunsom, 2013)

The decoder, usually an RNN, will read the context C and generate a sequence in the output language.

12.4.5 Neural Machine Translation


Output object (English sentence)
Decoder
Intermediate, semantic representation
Encoder
Source object (French sentence or image)

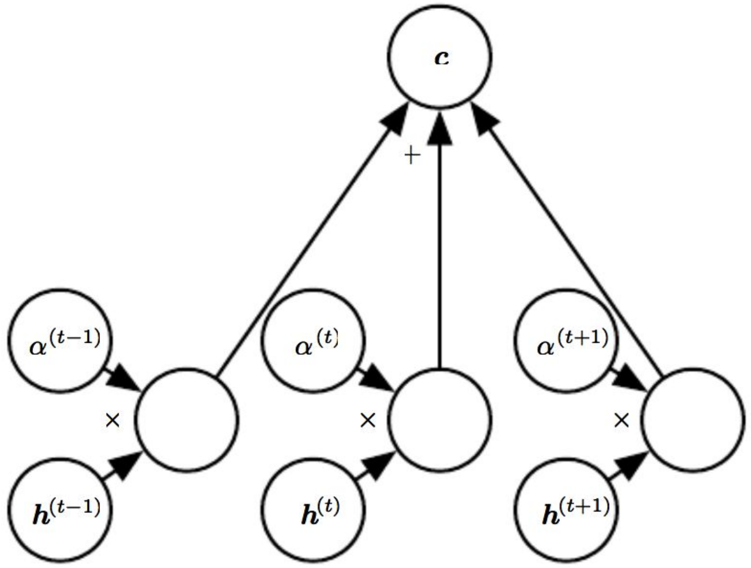
12.4.5.1 Using an Attention Mechanism and Aligning PIeces of Data

Using the context C, a fixed-size representation, for the full semantic details of a long sentence (i.e. 60 words) is very difficult

Instead, read the whole input sentence and produce one word at a time, each while focusing on different parts of the input sentence
→ Bahdanau et al. (2015)

C is a weighted average of all the hidden unit feature vector h$^{(t)}$, with α$^{(t)}$, which must be learned.

α$^{(t)}$ = output of a softmax of a **relevance scores**

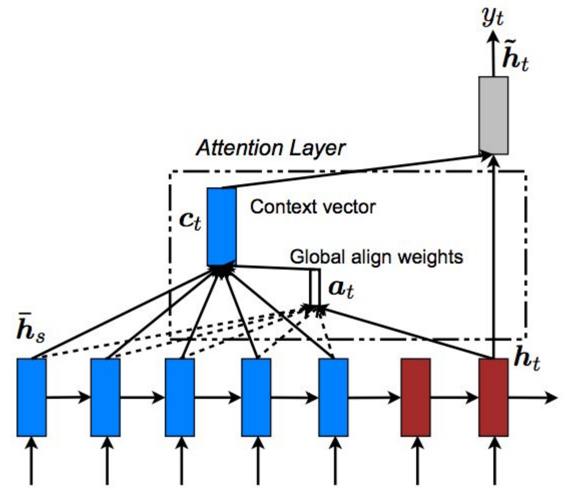


12.4.5.1 Using an Attention Mechanism and Aligning PIeces of Data

Example $\alpha^{(t)}$ (below as $a_t(s)$) computed for every target word, and possible score functions (from Luang, 2015)

$$\begin{aligned}\boldsymbol{a}_t(s) &= \operatorname{align}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\\ &= \frac{\exp\left(\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'} \exp\left(\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)}\end{aligned}$$

$$\operatorname{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases}\boldsymbol{h}_t^\top \bar{\boldsymbol{h}}_s\\ \boldsymbol{h}_t^\top \boldsymbol{W_a} \bar{\boldsymbol{h}}_s\\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_a}[\boldsymbol{h}_t; \bar{\boldsymbol{h}}_s]\right)\end{cases}$$



12.4.5.1 Using an Attention Mechanism and Aligning PIeces of Data

Attention-based system can be thought of having 3 components:

1) A process that reads raw data and converts them into distributed representation, 1 word having 1 feature vector
2) A list of feature vectors storing the output of the reader. Could be thought of as a memory of a sequence of facts, which can be accessed later individually
3) A process that exploits the content of the memory to sequentially perform a task, with the ability to put attention on one memory element

The 3rd generates the translated sentence

Alignement
If we can align words from source to destination language, we could also relate the word embeddings of each.
→ a translation matrix relating the two word embedding (Kocisky et al., 2014), efficient version for large dataset in Gouws et al., 2014.

12.4.6 Historical Perspective

- The idea of Distributed Representation (Rumelhart et al., 1986a): first exploration of backprop, where NN learns the relationship between family members
- From embedding for a symbol, to embedding for a word: Deerwester et al., 1990. SVD were used initially, then NN
- Early NN application for NLP represented the input as sequence of characters. See Miikkulainen and Dyer, 1991; Schmidhuber, 1996
- First neural language models with interpretable word embeddings by Bengio et al., 2001: help scale up from small set of symbols in the 80s to millions of words.
- Recent techniques use words as well as characters (Sutskever et al., 2011) and bytes of Unicode character (Gillick et al., 2015) as basic unit.
- NN also used for parsing, part-of-speech tagging, semantic role labeling, chunking
- 2D visualization of embedding became popular with development of t-SNE dimensionality reduction algorithm (van der Maaten and Hinton, 2008), and its application to word embeddings by Joseph Turian (2009)

12.5 Other Applications

Topics:

- Recommender Systems
- Exploration v. Exploitation
- Knowledge Representation, Reasoning and Question Answers

12.5.1 Recommender Systems

Recommending items to potential users or customers. 2 major types

1) Online advertising (which ads to show)
2) Item recommendations (which items to buy next)

Learn association between User ⇔ item

Examples:

- Selecting posts to display on news feeds
- Recommending movies to watch
- Recommending jokes
- Recommending advice from experts
- Matching players for video games
- Matching people in dating services

12.5.1 Recommender Systems

Often a supervised learning problem: using info about the item and user, predict a proxy of interest (user clicks on ad, user rating, user “liking”..)
→ Regression problem or probabilistic classification problem

Early work
Relied simply on User ID and Item ID, and learned the similarity between different users or different items: Users who like similar items have similar tastes.
⇒ **Collaborative Filtering**

Bilinear prediction of the target variable (such as a rating for a user/item pair) have been successful.

Consider **User** Embedding matrix ($A_{u,j}$) and **Item** Embedding matrix ($B_{j,i}$), together with user and item biases b_u,c_i, the rating can be modeled as:

$$\hat{R}_{u,i} = b_u + c_i + \sum_j A_{u,j} B_{j,i}.$$

12.5.1 Recommender Systems

$$\hat{R}_{u,i} = b_u + c_i + \sum_j A_{u,j} B_{j,i}$$

Goal: to minimize the **square error** between predicted ratings and actual ratings (R)

How to obtain A, B?

Could do a Singular Value Decomposition (SVD) of R
$R = U\ D\ V^T$
Where $A = U\ D$, and $B = V^T$

Problem: missing entries in R will be treated as 0.

Instead of computed SVD directly, Minimize the square loss between R and A.B by gradient descent

12.5.1 Recommender Systems

These strategies performed well on **Netflix prize**, a competition which took place between 2006 and 2009 (competition description: Bennett and Lanning, 2007)

Winners used these as a component of the final model. Tosher et al, 2009 Koren, 2009

Other models include the RBM undirected probabilistic model (which were also used by Tosher’s netflix prize winner)

Limitation with Collaborative Filtering:
New items lack historic ratings. New users have not rated any items ⇒ The **Cold Start** problem.
General solution is to introduce more information about the users and items: user profile info, item features.
⇒ **Content-based recommender** Systems. One can learn a mapping from a set of user features or item features to an embedding space with a deep learning architecture: Huang et al., 2013; Elkahky et al, 2015

12.5.1.1 Exploration versus Exploitation

When we use the recommendation system itself to collect data, we run into **biases**:

- See responses of users only to items recommended to them and not to the other items.
- No information on users for whom no recommendation was made
- No information about outcome for items that were not recommended.

⇒ We don't know the real recommendation the user would have wanted: It's like training a classifier by picking one class for each point and asking whether it is the true or the false class.
→ less informative than supervised learning where we have the real class

For true recommendations that have low probability in the model, the recommender would not even learn to predict them until the user picks them.

12.5.1.1 Exploration versus Exploitation

→ Framing the problem as **Contextual Bandits**
A special case of Reinforcement learning: learner picks a single action and receives a single reward

The action is taken in the context of some input variable that can inform the decision.

Ex: when we make an item recommendation for a new user, the contextual information is the user ID. We know who the user is

A **policy** = The mapping from **context** → **action**

12.5.1.1 Exploration versus Exploitation

In RL, tradeoff between **Exploration** and **Exploitation**

Exploration: taking an action to get more data, Actions that would have lower expected reward based on current policy → goal is to find actions with higher rewards than expected.

Exploitation: taking an action based on maximization of the learned policy. → Reinforcing what we already know

When do to exploration v.s exploitation? One distinction is along available time-scale:
If we have little time to accrue reward, prefer exploitation.
If we have longer time span, do more exploration in the beginning and later on shift to exploitation.

No such tradeoff in supervised learning since we have the correct output for every input. No need to try different actions.

12.5.1.1 Exploration versus Exploitation

problem: evaluating and comparing different policies.

RL involves an interaction between agent and environment: a feedback loop. The policy itself determines the agent’s action
→ no simple way to evaluate the learner’s performance based on fixed set of test set input values. (see Dudik et al., 2011 for contextual bandits evaluation methods)

12.5.2 Knowledge Representation, Reasoning, and Question Answering

The success of deep learning in NLP is based on the concept of word and symbol embeddings

Embeddings encapsulate **semantic knowledge** about words and the concept they belong to.

Next steps include developing embeddings for **phrases**, and for **relations between words and facts**

12.5.2.1 Knowledge, Relations, and Question Answering

How should we learn distributed representations that capture relations between entity?

Ex: a binary relation is a pair of item connected by a certain relation
If relation is “is less than”, entities {1,2,3} would be defined by ordered pairs: {(1,2), (1,3), (2,3)}

Relation as a verb is a tuple:
(Subject, Verb, Object),
with values (entity$_i$, relation$_j$, entity$_k$)

Or an attribute: (entity$_i$, attribute$_j$)

12.5.2.1 Knowledge, Relations, and Question Answering

What data for the training set?

Structured database that identify relations explicitly stored in **relational databases.**

A database for common sense knowledge is usually called a **Knowledge base**.
Ex: Freebase, OpenCyc, WordNet, Wikibase, GeneOntology

Consider each triplet in a database as a training example and maximize an objective that capture the joint distribution (see for example Bordes et al., 2013a)

Modelling approach
Similarly to word embedding, we learn an embedding vector for each relation.

12.5.2.1 Knowledge, Relations, and Question Answering

Practical application: **link prediction**
→ to predict missing links in a knowledge graph: speed up from the manual labor involved.
(See Wang et al. 2014b, Lin et al. 2015, Garcia-Duran et al. 2015)

Evaluation problem: we have a dataset of only positive problems
Are the model predictions new facts or fake facts?
→ Construct a set of facts that are probably false by corrupting a true fact, such as replacing one entity in the relation with a different one selected at random.

Popular metric: the 10% metric count = how many times the model ranks a correct fact among the top 10% of all corrupted versions of that fact.

12.5.2.1 Knowledge, Relations, and Question Answering

Other application: **Word-Sense Disambiguation**
Which sense of a word is the correct one in the current context?

Goal is to build a general Question-answering system.
Must be able to remember important facts and reason about them later

Best method today based on **Memory Networks** (section 10.12) from Weston et al., 2014
Kumar et al.,2015 extends the model to use GRU networks.