Representation Learning in Neural Networks

Keith L. Downing

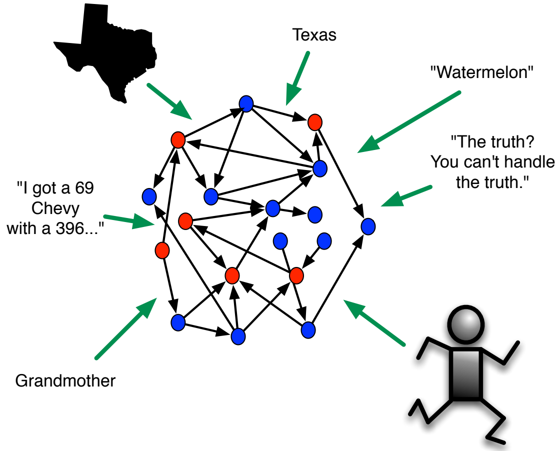
The Norwegian University of Science and Technology (NTNU)
Trondheim, Norway
keithd@idi.ntnu.no

November 13, 2017

Differences that make a Difference (Edelman)

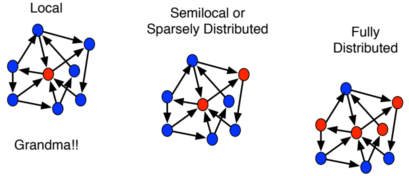
- Neural networks (biological and artificial) need to learn which differences between input patterns are significant (i.e. *make a difference*, call for different actions) and which are not, and hence can be *generalized* to evoke the same response.
- In an ANN, ”response” often means ”classification”, e.g., ”Is this image a Golden Retriever or a Husky?”
- If inputs have salient differences, then the network should map them to internal states that have enough separation from one another to insure the output patterns (i.e. classifications) also differ.
- Conversely, if the inputs should be classified similarly (even if their features are quite different) then the network should map them to the same (or very similar) internal states to insure that the final classifications are the same.

Distributed Memory


Texas
"Watermelon"
"The truth? You can't handle the truth."
"I got a 69 Chevy with a 396..."
Grandmother

- In natural and artificial neural nets, memories are spread across many neurons and synapses.
- Each neuron and synapse encodes parts of many memories.

Neural Coding Schemes

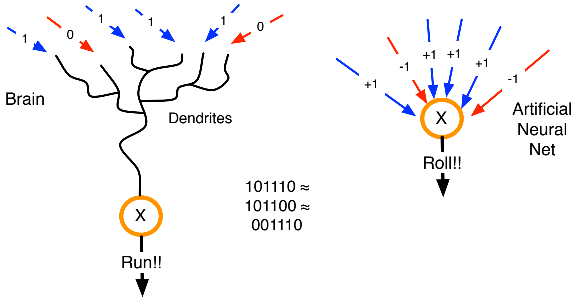


Given N neurons

- Local: $\binom{N}{1} = N$ patterns
- Sparsely Distributed: $\binom{N}{k} = \frac{N!}{(N-k)!k!}$ patterns. N = 20, k= 3 $\rightarrow$ 1140
- Fully Distributed: $\binom{N}{\frac{N}{2}} = \frac{N!}{(\frac{N}{2}!)^2}$ patterns. N = 20 $\rightarrow$ 184756

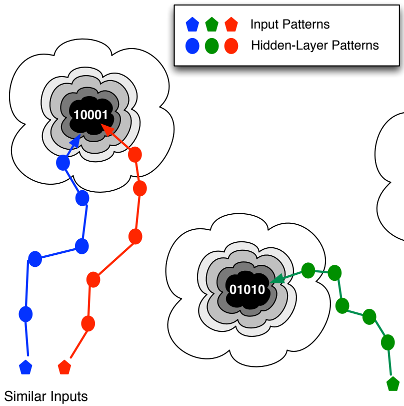
*But, to store 184756 patterns with 20 nodes in a **useful** manner isn't easy!

Advantages of Distributed Coding



1. **Structured Representation**: Salient similarities and differences among input patterns can be reflected in internal firing patterns (a.k.a. representations).
2. **Generalization**: Similar codes can be treated similarly by the network.
3. **Fault Tolerance**: A few noisy input bits or malfunctioning neurons won't affect the response.
4. **Specialization**: Many stimuli can have unique neural states.

The Representation Landscape



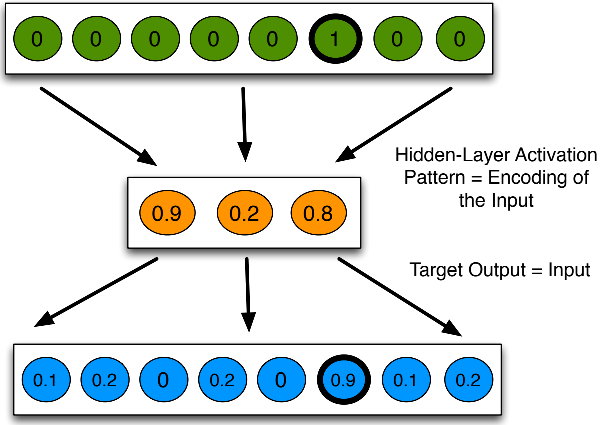
- Feedforward processing → Transitioning from an input state through intermediate representations defined by the activation patterns of each hidden layer.
- Learning → Creating large basins of attraction with minimal overlap.

Representation Learning

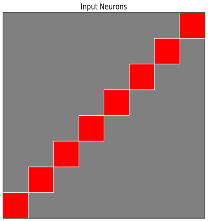
- Any network with a hidden layer learns representations in the course of training.
- Each representation vector may house subpatterns that embody features of the input data.
- Common statement: *Early NN layers learn salient features of the data.*
- The purpose of representations is to **make learning easier** for downstream layers.
- Similarly, we can save the representation and use it to facilitate learning in another system, whether NN or other ML tool.
- Transfer Learning: Generate a representation for task 1, but then use it for a good head start on task 2.
- Key Tradeoff: Specialised for task 1 -vs- General enough for other tasks.
- Ideally, tasks 1 and 2 share many salient features, so adapting for task 2 requires only a little tuning of the high-level weights (i.e. those near the output end).

The Autoencoder

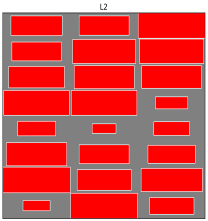
- A neural network designed to map input patterns to themselves, but via one or more hidden layers.
- The patterns produced in these hidden layers constitute alternate representations of the input patterns.



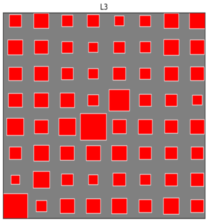
Initial Internal Representation



Input
Patterns



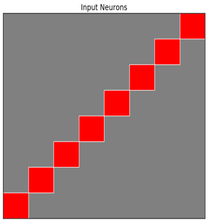
Hidden-Layer
Patterns



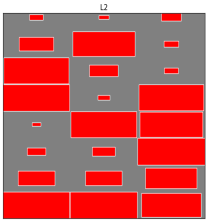
Output
Patterns

Initially, many of the inputs evoke the same internal states. The net cannot differentiate (separate) the inputs internally, so it cannot map them to different classes.

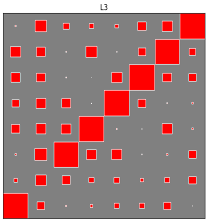
Learning Useful Internal Representations



Input
Patterns



Hidden-Layer
Patterns

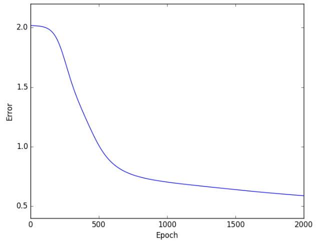


Output
Patterns

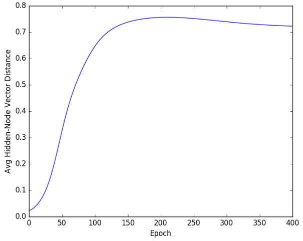
After training, the network produces different internal activation patterns (i.e. representations) with good separation between them. Each of these input patterns can then evoke a different output pattern.

Quantifying Progress

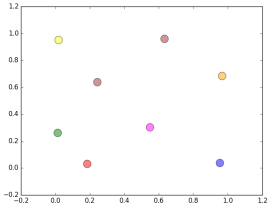
Training Error



Separation of Hidden-Layer Patterns



Final Hidden-Layer Patterns



Types of Autoencoders

Undercomplete

- Hidden layers are smaller than the input layer.
- Useful for finding compression codes.
- Forces hidden layer(s) to extract the most salient input features.

Overcomplete

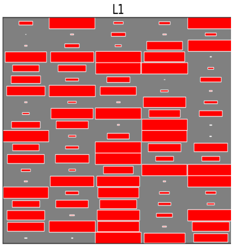
- Hidden layers are larger than the input layer.
- With a standard loss function (L), such as MSE, the net can just copy the input to the hidden layer(s) and then onto the output layer, without extracting any useful features.
- By adding a regularizer to L, we can prevent the net from merely copying and encourage it to produce more useful representations.

Regularized Autoencoders

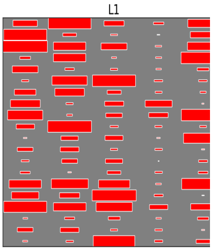
Notation: X = input patterns; f(X) = encoding phase (that produces H), g(H) = decoding phase (that should produce something similar to X)

1. Sparse Autoencoder
   - Penalizes dense hidden-layer activation patterns.
   - $\tilde{L} = L(X, g(f(X))) + \Omega(H)$
   - Used to learn representations (features) for another task.
   - To get actual zeros, not just small activations, use ReLU.
2. Denoising Autoencoder
   - Add noise to inputs (X) to get $\tilde{X}$, but keeps targets as X.
   - $\tilde{L} = L(X, g(f(\tilde{X})))$
   - Trained net is more robust and captures the general **structure** of X, not the individual cases.
3. Contractive Autoencoder
   - Enforce stability of internal representation (H) by reducing its sensitivity to small changes in X, as quantified by the Jacobian, $\nabla_X H$.
   - $\tilde{L} = L(X, g(f(X))) + \Omega(H, X)$
   - $\Omega(H, X) = \lambda \sum_i \| \nabla_X h_i \|^2$

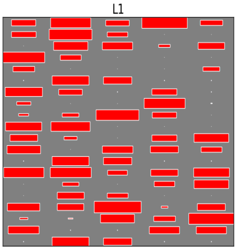
Sparse Autoencoder Example



Basic
Autoencoder
(Sigmoid)



Sparse
Autoencoder
(Sigmoid)

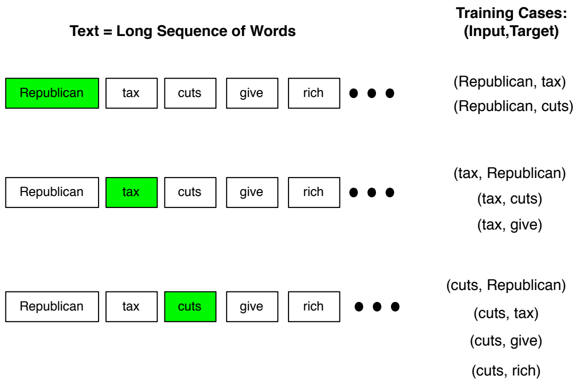


Sparse
Autoencoder
(ReLU)

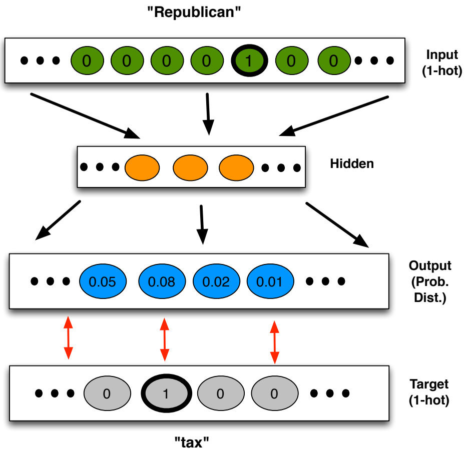
- Network = 10 x 5 x 10 Autoencoder, trained on 40 random bit vectors
- Hidden-layer reps for 20 vectors shown above: one per row.
- Size of red box denotes activation level.

Embeddings

- Embeddings = hidden-layer activation patterns used as alternate representations for input vectors
- Particularly useful for natural-language processing, where input vectors for words typically have no meaningful structure
- Description based on Chris McCormick's Skip-Gram Model Tutorial.

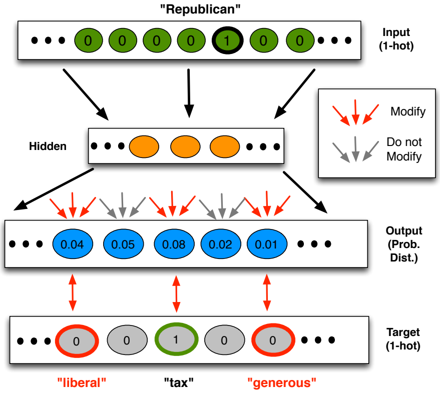


Word2Vec Skip-Gram Network: Training



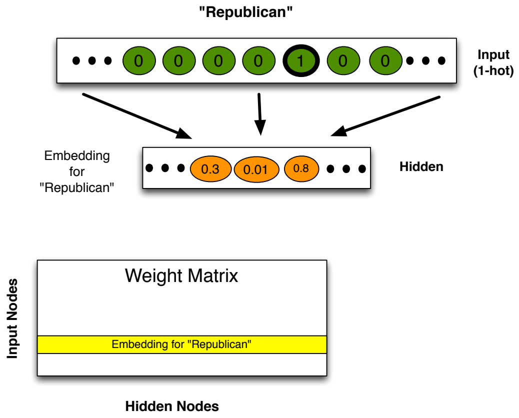
- Hidden layer uses simple linear activation: output = sum wgt'd inputs.
- Output layer uses softmax to produce probability distribution. Each output node = word probability.

Negative Sampling



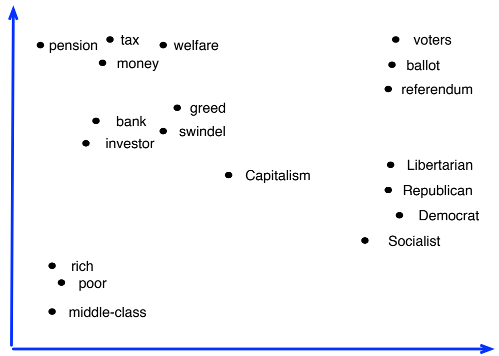
- **Problem:** Word2Vec networks have many weights. Do we have to modify ALL of them for every case? No.
- Choose (5-20) words (red) that are **not** associated with the key word.
- Only modify weights associated with the correct word along with those for the selected negative examples.

After Training: Generating the Embedding



- Embedding = Hidden Layer activations produced by 1-hot input.
- Since inputs are 1-hot, and hidden-layer activation is simple linear, embedding = kth row of the input-to-hidden weight matrix, where k = index of hot bit $\rightarrow$ Read whole embedding from wgt matrix.

Embeddings = Structured Representations



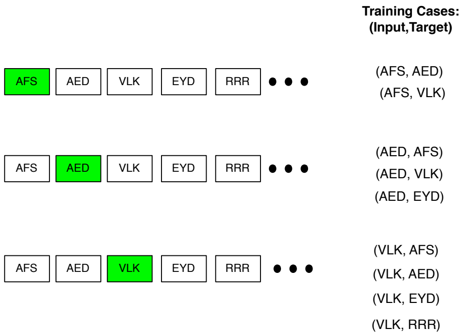
- Distance between **any** pair of **one-hot** vectors = $\sqrt{2} \rightarrow$ the space of 1-hot representations for words has no meaningful structure.
- However, **embeddings** for related words have shorter distances in m-dimensional space, where m = hidden-layer size.
- Vector proximity reflects **semantic** similarity: structure.

Using Embeddings

- Notation: E(w) = Embedding for word w; W*(v) = word corresponding to the embedding closest to vector v.
- Not only do similar words have similar embeddings, but the **vector differences** between pairs of words that have a similar relationship are also similar.
- E('king') - E('man') $\approx$ E('queen') - E('woman')
- E('horses') - E('horse') $\approx$ E('clowns') - E('clown')
- Answer questions such as "What is the female equivalent of a king?"
- W*(E('king') - E('man') + E('woman')) = 'queen'
- This is the basis of Analogical Reasoning
  - Human: House as Bird: ???
  - W*(E('bird') + (E('house') - E('human'))) = 'nest'
  - Paris : France as Rome: ???
  - W*(E('Rome') + (E('France') - E('Paris'))) = 'Italy'
- Pre-generated embeddings give structured starting points (similar to pre-training) for further AI problem-solving and learning.
- Many Apps: Lang.Translation, DNA Analysis, Proteonomics....

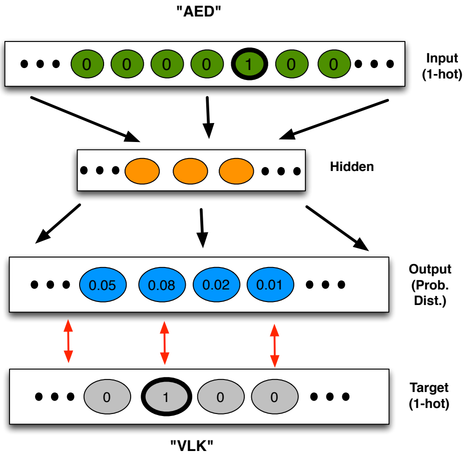
Embeddings for Proteonomics

- Proteins = long sequences of amino acids: AFSAEDVLKEYDRRR...
- Choose a group size K (e.g. K=3) and do K *splits*:
- (AFS, AED, VLK, EYD, RRR) , (FSA,EDV,LKE,YDR,...), (SAE, DVL, KEY, DRR,...). Generate training cases from each.



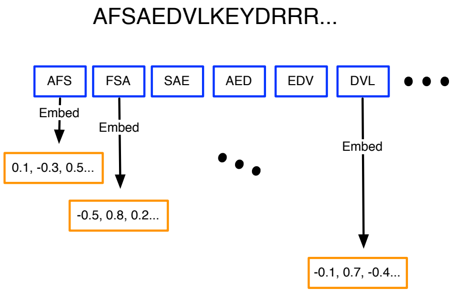
Continuous Distr. Rep. of Bio. Sequences, Asgari and Mofrad, 2015.

Skip-Gram Training



- Hidden layer uses simple linear activation: output = sum wgt'd inputs.
- Output layer uses softmax to produce probability distribution. Each output node = N-gram prob.

Generating Protein Embeddings

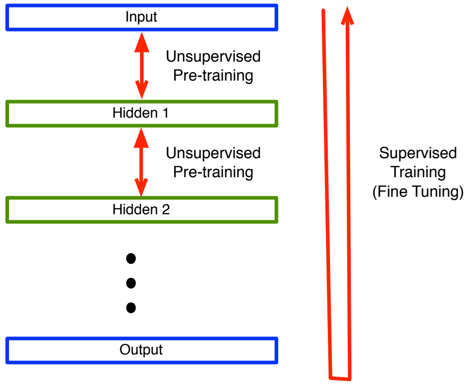


- The protein's embedding (a.k.a. protovec) is the sum of it's N-gram embeddings.
- Use protovecs as inputs to NNs that can be trained to predict the properties of proteins.

Semi-Supervised Learning

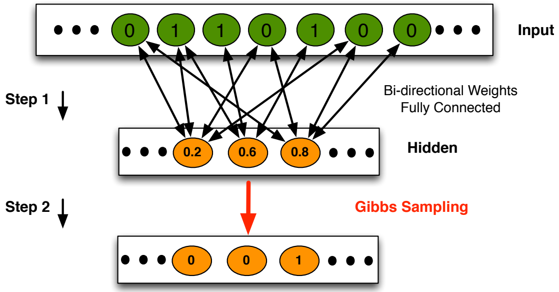
- Well-structure representations can be learned by supervised or unsupervised methods.
- They can then be reused as starting points for a fully supervised process.
- Useful when only some of the dataset is labeled. Then unlabeled cases support unsupervised representational learning, while the labeled cases drive the second, supervised, phase.
- Alternatively, unsupervised and supervised can be interleaved.
- 2006: Beginning of Deep Learning Renaissance - Unsupervised pre-training of ”Deep Belief Networks” (using Restricted Boltzmann Machines) allowed **deep** nets to learn.
- A few years later, Convolution nets and LSTMs beat out Deep Belief Nets, but Hinton (and others) still believe that unsupervised pretraining is the future of DL. It's also more biologically realistic: our brains appear to do a lot more unsupervised than supervised learning.

Deep Belief Networks



- Weights between pre-trained layers are bi-directional: $w_{j,k} = w_{k,j}$
- Layer K+1 serves as hidden layer for recreating (autoencoding) Layer K **back on** Layer K.
- Thus, Layer K+1 forms a representation of Layer K.

Restricted Boltzmann Machine (RBM): Forward Phase

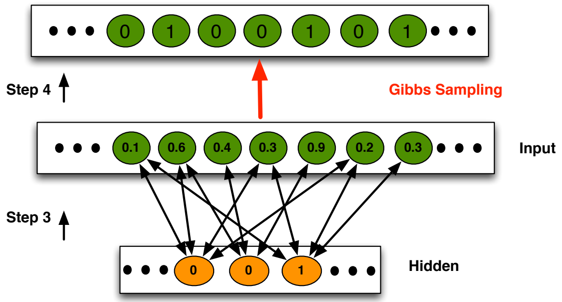


- Feed the input pattern (V) forward over weights W.
- Hidden layer (H) uses sigmoid (σ) activation function.
- Use H values as probabilities for Gibbs sampling.

$$p(h_k = 1 | V) = \sigma(b_k + \sum_j v_j w_{k,j})$$

where b_k = bias for kth hidden node.

RBM Construction Phase



- Send hidden-layer pattern back over the same weights (W).
- Input layer becomes a reconstruction layer and also uses σ activation.
- Use reconstructed values (V) as probabilities for Gibbs sampling.

$$p(v_k = 1|H) = \sigma(c_k + \sum_j h_j w_{j,k})$$

where c_k = bias for kth reconstruction node.

RBM Training Overview

1. Present an input case
2. Alternate forward and reconstruction phases until binary states V and H_0 stabilize.
3. Modify weights based on gradient $\frac{\partial E}{\partial W}$ where E is an energy function to be minimized (details not included).
4. Go to step 1

- After many rounds through the training set, consider the RBM "trained" and move on to the next RBM, which will have H_0 as its input layer. The case set for H_0 is the mapping of the original input cases to H_0: The representation at H_0 is the data set for the H_0-H_1 RBM.
- Keep training RBMs and generating new representations deeper and deeper into the network.
- When all RBMs are trained, return to the original inputs and their corresponding targets and apply supervised learning throughout the net for fine tuning.