Lecture 13

Deep Belief Networks

Michael Picheny, Bhuvana Ramabhadran, Stanley F. Chen, Markus Nussbaum-Thom

Watson Group
IBM T.J. Watson Research Center
Yorktown Heights, New York, USA
{picheny,bhuvana,stanchen,nussbaum}@us.ibm.com

20 April 2016

Administrivia

- Lab 4 handed back today?
- E-mail reading project selections to `stanchen@us.ibm.com` by this Friday (4/22).
- Still working on tooling for experimental projects; please get started!

Outline for the next two lectures

- Introduction to Neural Networks, Definitions
- Training Neural Networks (Gradient Descent, Backpropagation, Estimating parameters)
- Neural networks in Speech Recognition (Acoustic modeling)
- Objective Functions
- Computational Issues
- Neural networks in Speech Recognition (Language modeling)
- Neural Network Architectures (CNN, RNN, LSTM, etc.)
- Regularization (Dropout, Maxnorm, etc.)
- Advanced Optimization methods
- Applications: Multilingual representations, autoencoders, etc.
- What's next ?

A spectrum of Machine Learning Tasks

Typical Statistics

- Low-dimensional data (e.g. less than 100 dimensions)
- Lots of noise in the data
- There is not much structure in the data, and what structure there is, can be represented by a fairly simple model.
- The main problem is distinguishing true structure from noise.

A spectrum of Machine Learning Tasks Cont'd

Machine Learning

- High-dimensional data (e.g. more than 100 dimensions)
- The noise is not sufficient to obscure the structure in the data if we process it right.
- There is a huge amount of structure in the data, but the structure is too complicated to be represented by a simple model.
- The main problem is figuring out a way to represent the complicated structure so that it can be learned.

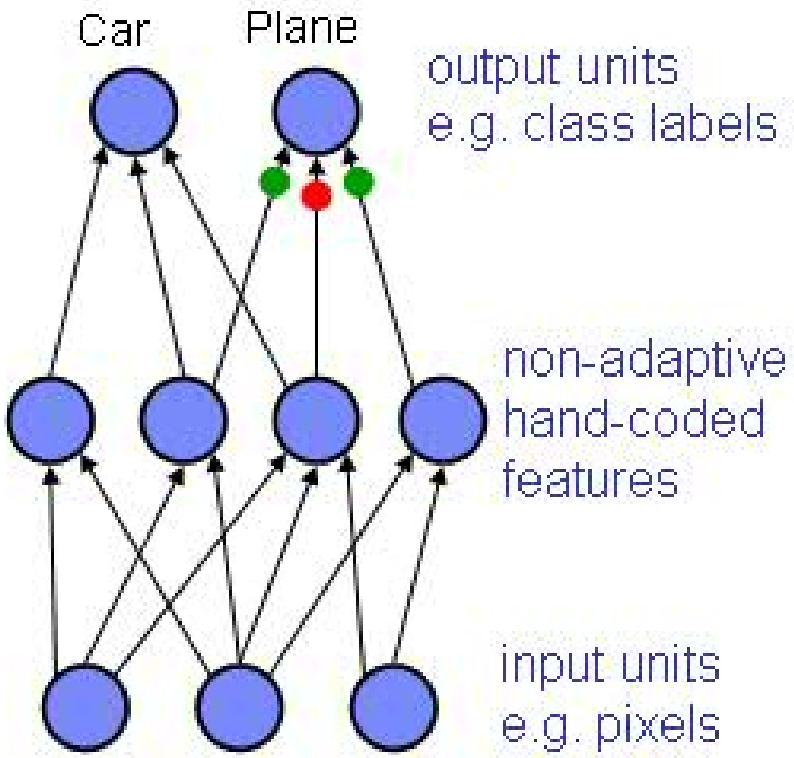
Why are Neural Networks interesting?

- GMMs and HMMs to model our data
- Neural networks give a way of defining a complex, non-linear model with parameters W (weights) and biases (b) that we can fit to our data

- In past 3 years, Neural Networks have shown large improvements on small tasks in image recognition and computer vision
- Deep Belief Networks (DBNs) ??
- Complex Neural Networks are slow to train, limiting research for large tasks
- More recently extensive use of various Neural Network architectures for large vocabulary speech recognition tasks

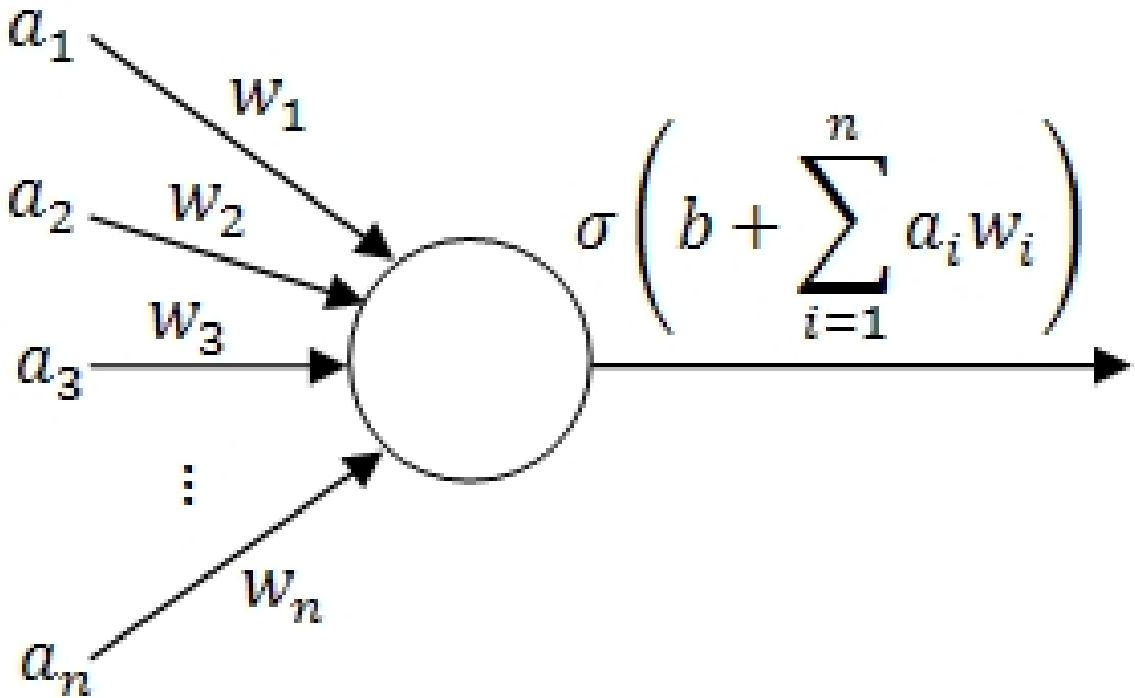
Initial Neural Networks

- Perceptrons (1960) used a layer of hand-coded features and tried to recognize objects by learning how to weight these features.
- Simple learning algorithm for adjusting the weights.
- Building Blocks of modern day networks



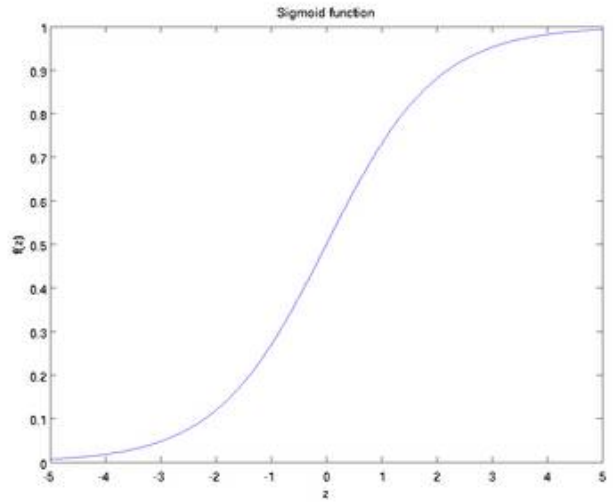
Perceptrons

- The simplest classifiers from which neural networks are built are perceptrons.
- A perceptron is a linear classifier which takes a number of inputs $a_1, \ldots, a_n$, scales them using some weights $w_1, \ldots, w_n$, adds them all up (together with some bias b) and feeds the result through a linear activation function, σ (eg. sum)

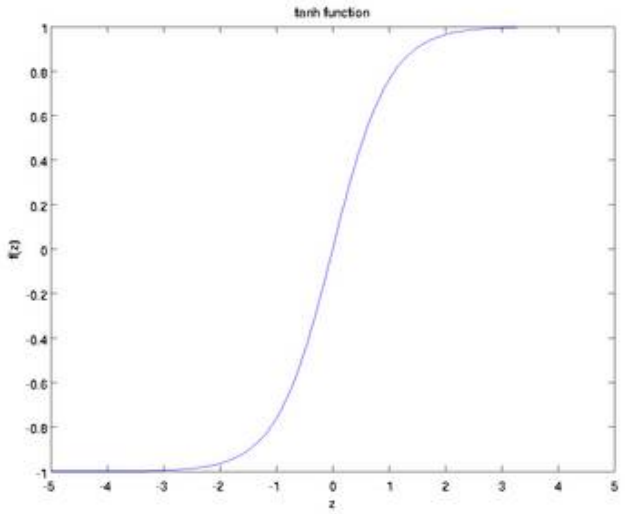


Activation Function

- Sigmoid $f(z) = \frac{1}{1+\exp(-z)}$



- Hyperbolic tangent $f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

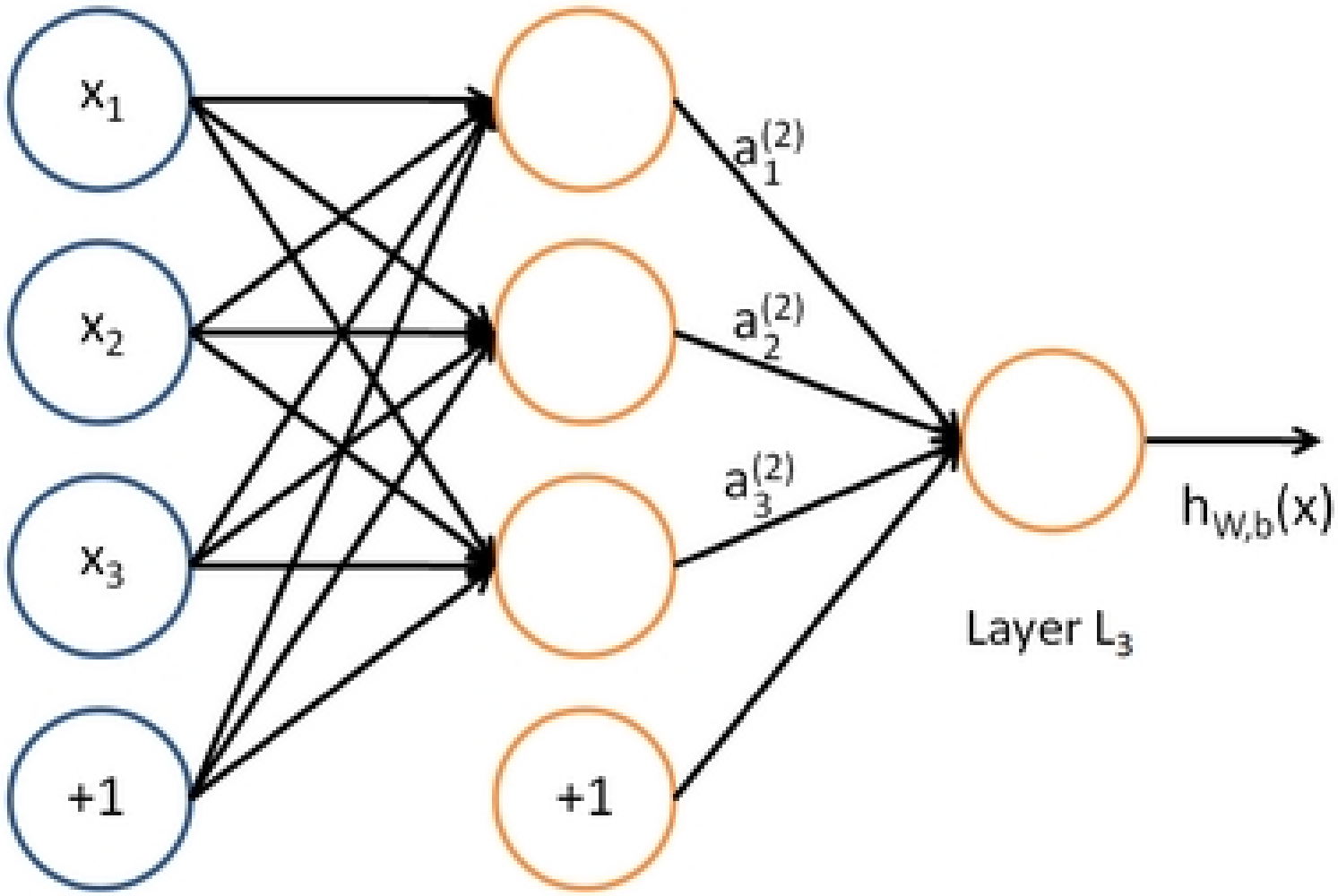


Derivatives of these activation functions

- If $f(z)$ is the sigmoid function, then its derivative is given by $f'(z) = f(z)(1 - f(z))$.
- If $f(z)$ is the tanh function, then its derivative is given by $f'(z) = 1 - (f(z))^2$.
- Remember this for later!

Neural Network

A neural network is put together by putting together many of our simple building blocks.



Definitions

- n_l denotes the number of layers in the network;
- L_1 is the input layer, and layer L_{n_l} the output layer.
- Parameters $(W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)}$, where
- $W_{ij}^{(l)}$ is the parameter (or weight) associated with the connection between unit j in layer l, and unit i in layer $l + 1$.
- $b_i^{(l)}$ is the bias associated with unit i in layer $l + 1$ Note that bias units don't have inputs or connections going into them, since they always output
- $a_i^{(l)}$ denotes the "'activation"' (meaning output value) of unit i in layer l.

Definitions

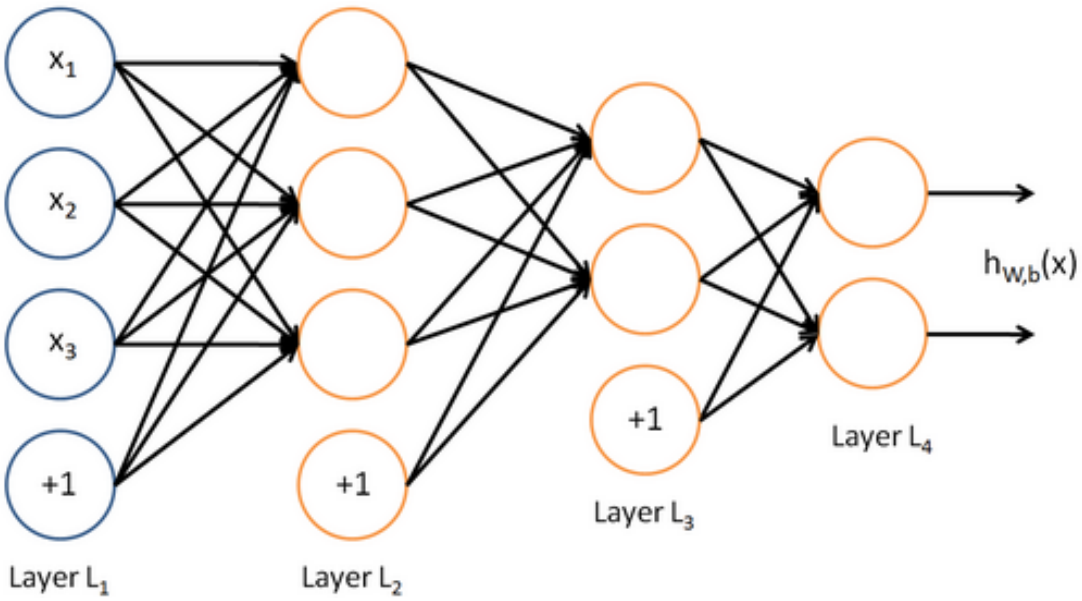
- This neural network defines $h_{W,b}(x)$ that outputs a real number. Specifically, the computation that this neural network represents is given by:

$$a_1^{(2)} = f(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)})$$
$$a_2^{(2)} = f(W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)})$$
$$a_3^{(2)} = f(W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3 + b_3^{(1)})$$
$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} + b_1^{(2)})$$

- This is called forward propogation.
- Use matrix vector notation and take advantage of linear algebra for efficient computations.

Another Example

- Generally networks have multiple layers and predict more than one output value.
- Another example of a feed forward network



How do you specify output targets?

- Output targets are specified with a 1 for the label corresponding to each feature vector
- What would these targets be for speech?
- Number of targets is equal to the number of classes

Training set: $(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots, (x^{(m)}, y^{(m)})$

$y^{(i)}$ one of $\begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$

How do you train these networks?

- Use Gradient Descent (batch)
- Given a training set $(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$
- Define the cost function (error function) with respect to a single example to be:

$$J(W, b; x, y) = \frac{1}{2} \|h_{W,b}(x) - y\|^2$$

Training (contd.)

- For m samples, the overall cost function becomes

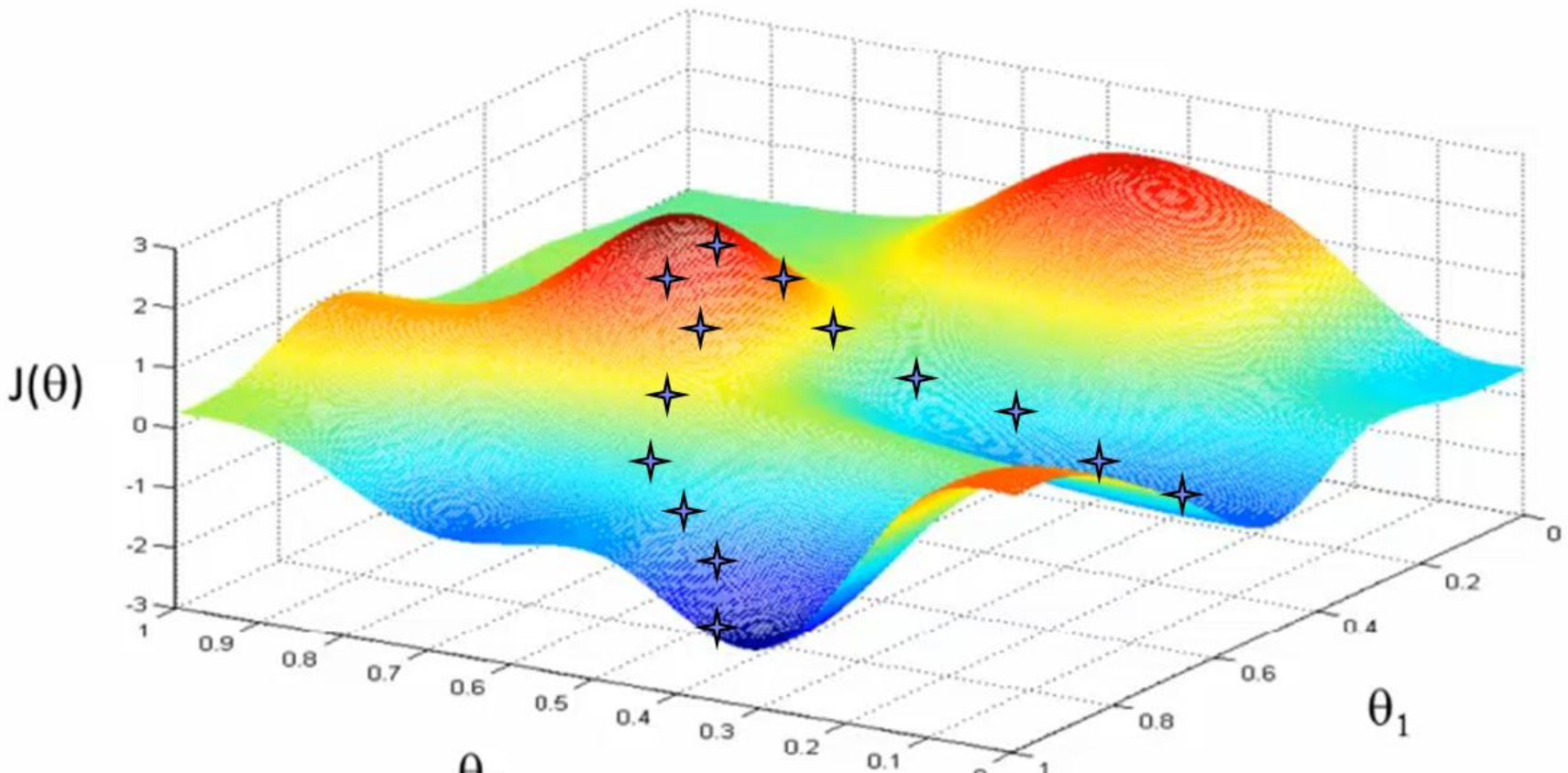
$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m} J(W,b;x^{(i)},y^{(i)})\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

$$= \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{(l)}\right)^2$$

- The second term is a regularization term (”’weight decay”’) that prevent overfitting.
- Goal: minimize $J(W,b)$ as a function of W and b.

Gradient Descent

- Cost function is $J(\theta)$
- minimize$_{\theta} J(\theta)$
- θ are the parameters we want to vary



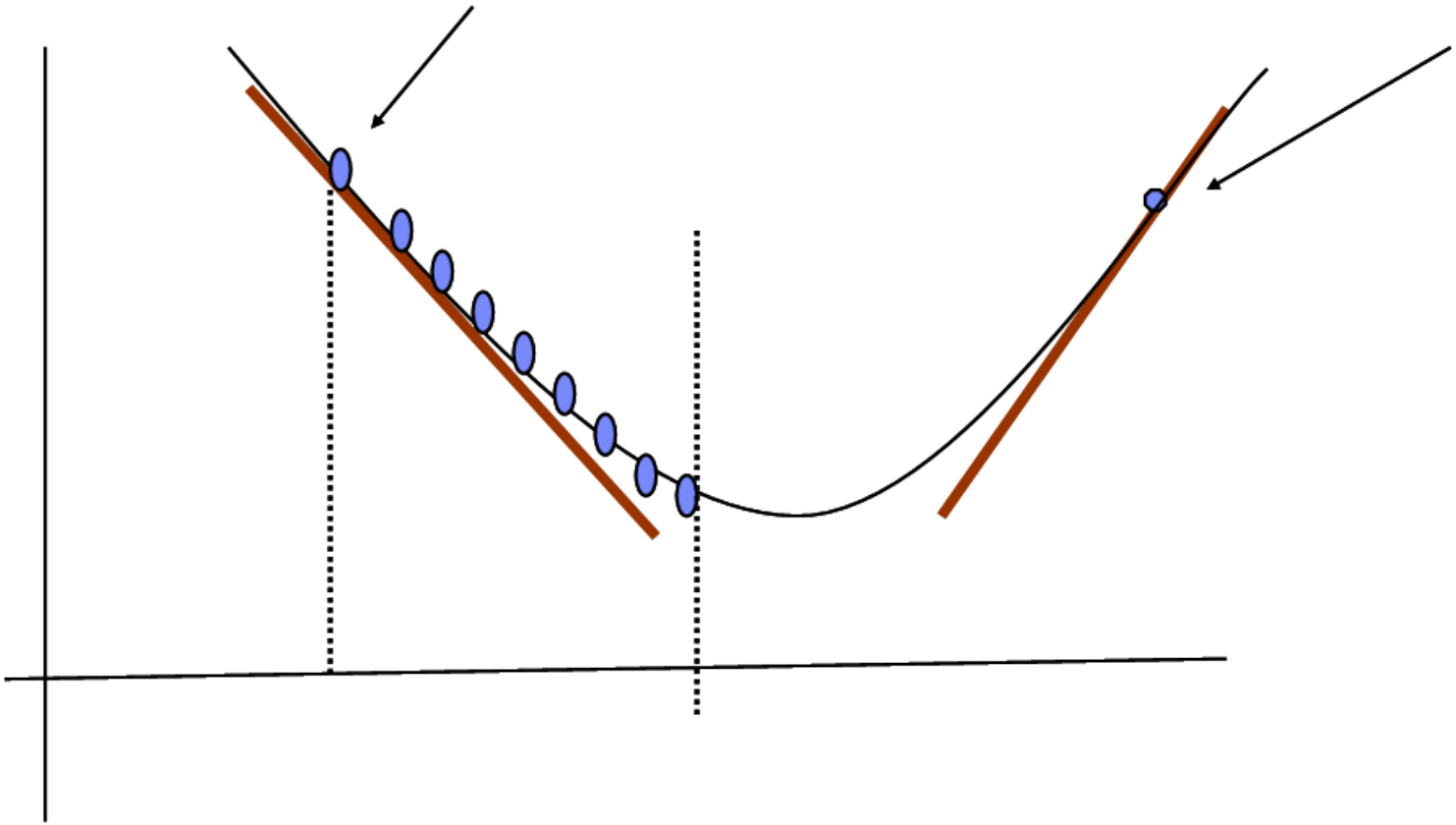
Gradient Descent

- Repeat until convergence
- Update θ as

$$\theta_j - \alpha * \frac{\partial}{\partial \theta_j} \boldsymbol{J}(\theta) \forall \boldsymbol{j}$$

- α determines how big a step in the right direction and is called the learning rate.
- Why is taking the derivative the correct thing to do?
- . . . direction of steepest descent

Gradient Descent



- As you approach the minimum, you take smaller steps as the gradient gets smaller

Returning to our network...

- Goal: minimize $J(W, b)$ as a function of W and b.
- Initialize each parameter $W_{ij}^{(l)}$ and each $b_i^{(l)}$ to a small random value near zero (for example, according to a Normal distribution)
- Apply an optimization algorithm such as gradient descent.
- $J(W, b)$ is a non-convex function, gradient descent is susceptible to local optima; however, in practice gradient descent usually works fairly well.

Estimating Parameters

- It is important to initialize the parameters randomly, rather than to all 0's. If all the parameters start off at identical values, then all the hidden layer units will end up learning the same function of the input.
- One iteration of Gradient Descent yields the following parameter updates:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b)$$

- The backpropogation algorithm is an efficient way to computing these partial derivatives.

Backpropagation Algorithm

- Let's compute $\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y)$ and $\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y)$, the partial derivatives of the cost function $J(W, b; x, y)$ with respect to a single example (x, y).
- Given the training sample, run a forward pass through the network and compute all teh activations
- For each node i in layer l, compute an "error term" $\delta_i^{(l)}$. This measures how much that node was "responsible" for any errors in the output.

Backpropagation Algorithm

- This error term will be different for the output units and the hidden units.
- Output node: Difference between the network's activation and the true target value defines $delta_i^{(n_l)}$
- Hidden node: Use a weighted average of the error terms of the nodes that uses $delta_i^{(n_l)}$ as an input.

Backpropogation Algorithm

- Let $z_i^{(l)}$ denote the total weighted sum of inputs to unit i in layer l, including the bias term

$$z_i^{(2)} = \sum_{j=1}^{n} W_{ij}^{(1)} x_j + b_i^{(1)}$$

- Perform a feedforward pass, computing the activations for layers L_2, L_3, and so on up to the output layer L_{n_l}.
- For each output unit i in layer n_l (the output layer), define

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} \|y - h_{W,b}(x)\|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

Backpropogation Algorithm Cont'd

- For $l = n_l - 1, n_l - 2, n_l - 3, \ldots, 2$, define
- For each node i in layer l, define

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

- We can now compute the desired partial derivatives as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b; x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b; x, y) = \delta_i^{(l+1)}$$

- Note If $f(z)$ is the sigmoid function, then its derivative is given by $f'(z) = f(z)(1 - f(z))$ which was computed in the forward pass.

Backpropogation Algorithm Cont'd

- Derivative of the overall cost function J(W,b) over all training samples can be computed as:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial W_{ij}^{(l)}} J(W,b; x^{(i)}, y^{(i)}) \right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial}{\partial b_{i}^{(l)}} J(W,b) = \frac{1}{m} \sum_{i=1}^{m} \frac{\partial}{\partial b_{i}^{(l)}} J(W,b; x^{(i)}, y^{(i)})$$

- Once we have the derivatives, we can now perform gradient descent to update our parameters.

What was the Intuition Behind Backpropagation

Logistic regression: $Cost(i) = (h_\theta x^{(i)} - y^{(i)})^2$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log(h_\Theta(x^{(i)})) + (1 - y^{(i)}) \log(1 - (h_\Theta(x^{(i)})))\right]$$
$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

Focusing on a single example $x^{(i)}$, $y^{(i)}$, the case of 1 output unit, and ignoring regularization ($\lambda = 0$),

$$\text{cost(i)} = y^{(i)} \log h_\Theta(x^{(i)}) + (1 - y^{(i)}) \log h_\Theta(x^{(i)})$$

(Think of $\text{cost(i)} \approx (h_\Theta(x^{(i)}) - y^{(i)})^2$)

Updating Parameters via Gradient Descent

- Using matrix notation

$$W^{(l)} = W^{(l)} - \alpha \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$

$$b^{(l)} = b^{(l)} - \alpha \left[\frac{1}{m} \Delta b^{(l)} \right]$$

Now we can repeatedly take steps of gradient descent to reduce the cost function $J(W, b)$ till convergence.

Optimization Algorithm

- We used Gradient Descent. But that is not the only algoritm.
- More sophisticated algorithms to minimize $J(\theta)$ exist.
- An algorithm that uses gradient descent, but automatically tunes the learning rate α so that the step-size used will approach a local optimum as quickly as possible.
- Other algorithms try to find an approximation to the Hessian matrix, so that we can take more rapid steps towards a local optimum (similar to Newton's method).

Optimization Algorithm

- Examples include the "'L-BFGS"' algorithm, "'conjugate gradient"' algorithm, etc.
- These algorithms need for any θ, $J(\theta)$ and $\nabla_\theta J(\theta)$. These optimization algorithms will then do their own internal tuning of the learning rate/step-size and compute its own approximation to the Hessian, etc., to automatically search for a value of θ that minimizes $J(\theta)$.
- Algorithms such as L-BFGS and conjugate gradient can often be much faster than gradient descent.

Optimization Algorithm Cont'd

- In practice, on-line or Stochastic Gradient Descent is used
- The true gradient is approximated by the gradient from a single or a small number of training samples (mini-batches)
- Typical implementations may also randomly shuffle training examples at each pass and use an adaptive learning rate.

Recap of Neural Networks

- A neural network has multiple hidden layers, where each layer consists of a linear weight matrix a non-linear function (sigmoid)
- Outputs targets: Number of classes (sub-word units)
- Output probabilities used as acoustic model scores (HMM scores)
- Objective function that minimizes loss between target and hypothesized classes
- Benefits: No assumption about a specific data distribution and parameters are shared across all data
- Training is extremely challenging with the objective function being non-convex.
- Recall the weights randomly initialized and can get stuck in local optima.

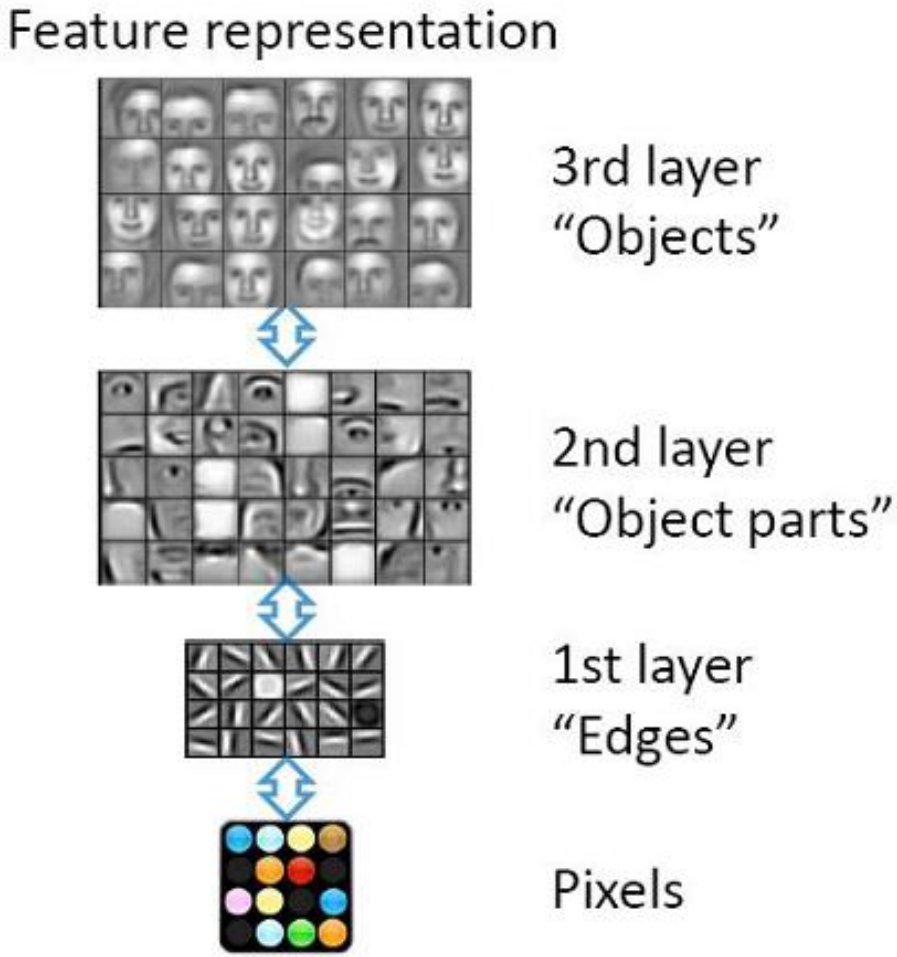
Neural Networks and Speech Recognition

- Introduced in the 80s and 90s to speech recognition, but extremely slow and poor in performance compared to the state-of-the-art GMMs/HMMs
- Several papers published by ICSI, CMU, IDIAP several decades ago!
- Over the last couple of years, renewed interest with what is known as Deep Belief Networks, or renamed as Deep Neural Networks.

History: Deep Belief Networks (DBNs)

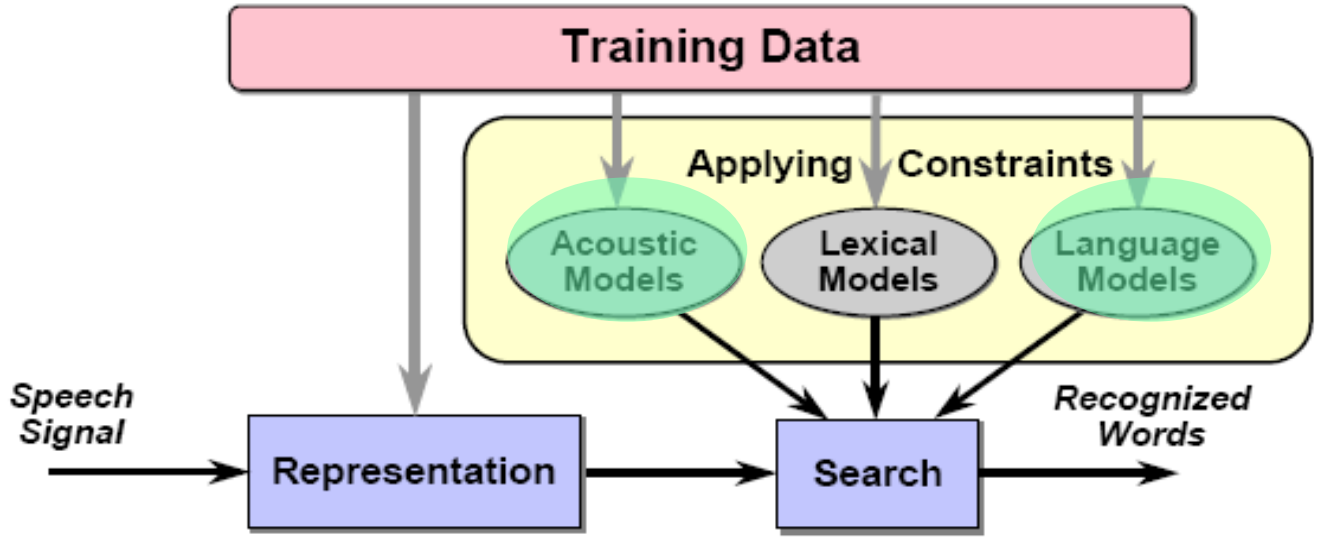
- Deep Belief Networks [Hinton, 2006] Capture higher-level representations of input features Pre-train ANN weights in an unsupervised fashion, followed by fine-tuning (backpropagation)
- Address issues with MLPs getting stuck at local optima.
- DBN Advantages first applied to image recognition tasks, showing gains between 10-30% relative successful application on small vocabulary phonetic recognition task
- Around 2012, DBNs get renamed as Deep Neural Networks (DNNs)

What does a Deep Network learn?



Neural Networks and Speech Recognition

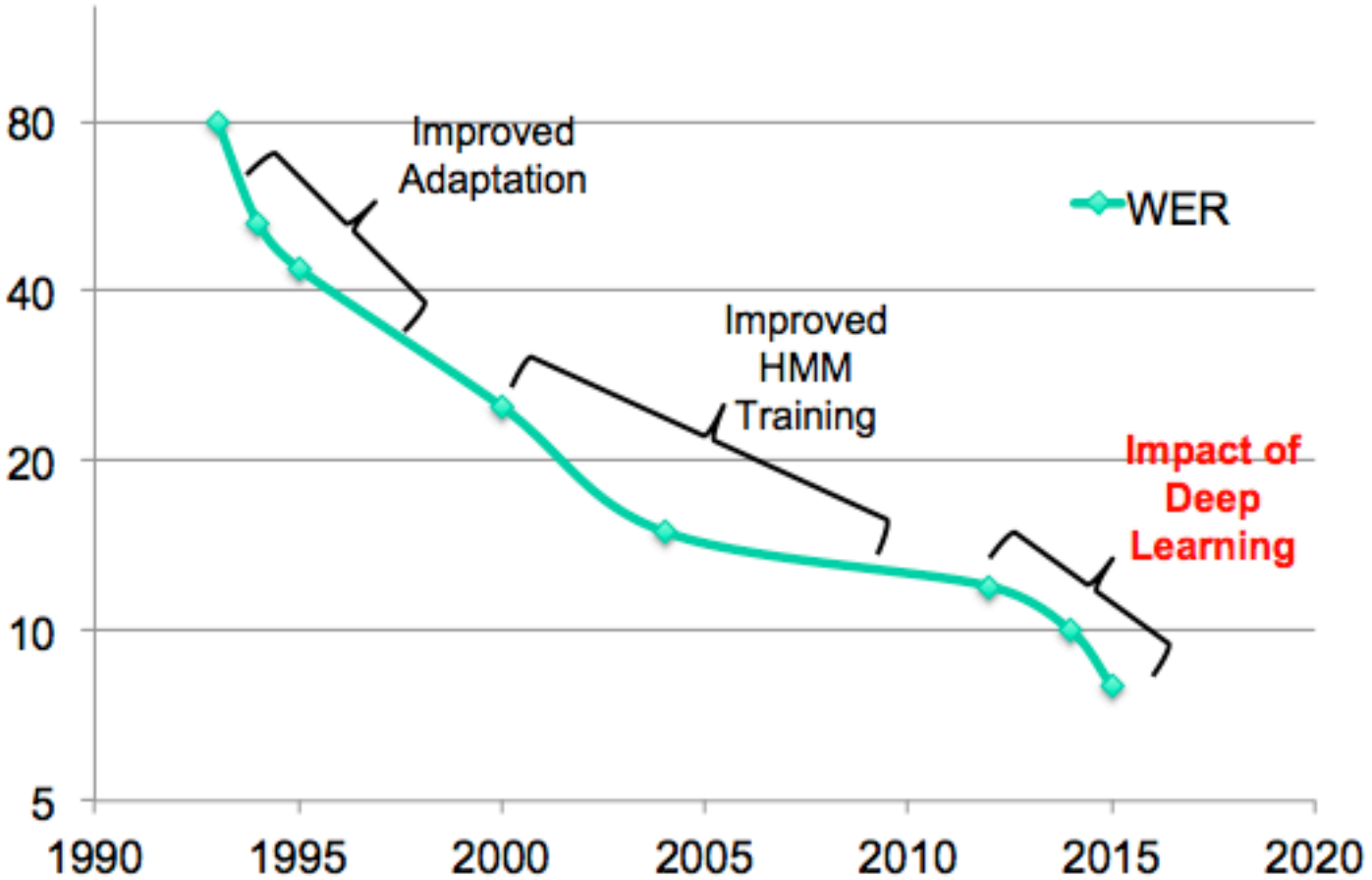
Networks for Individual Components of a speech recognition system



Used in both, acoustic and language modeling!

Good improvement in speech recognition

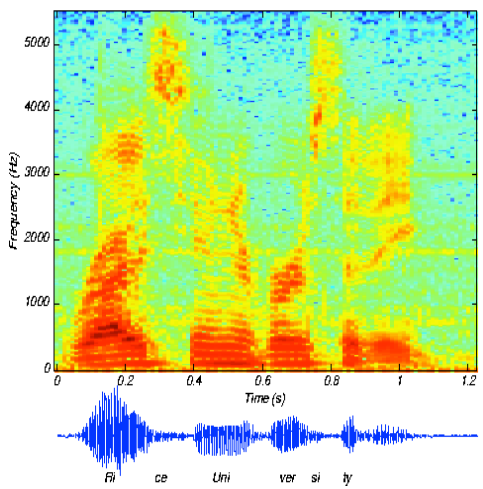
On a conversational telephone-bandwidth task in English:



Acoustic Modeling

What makes speech so unique?

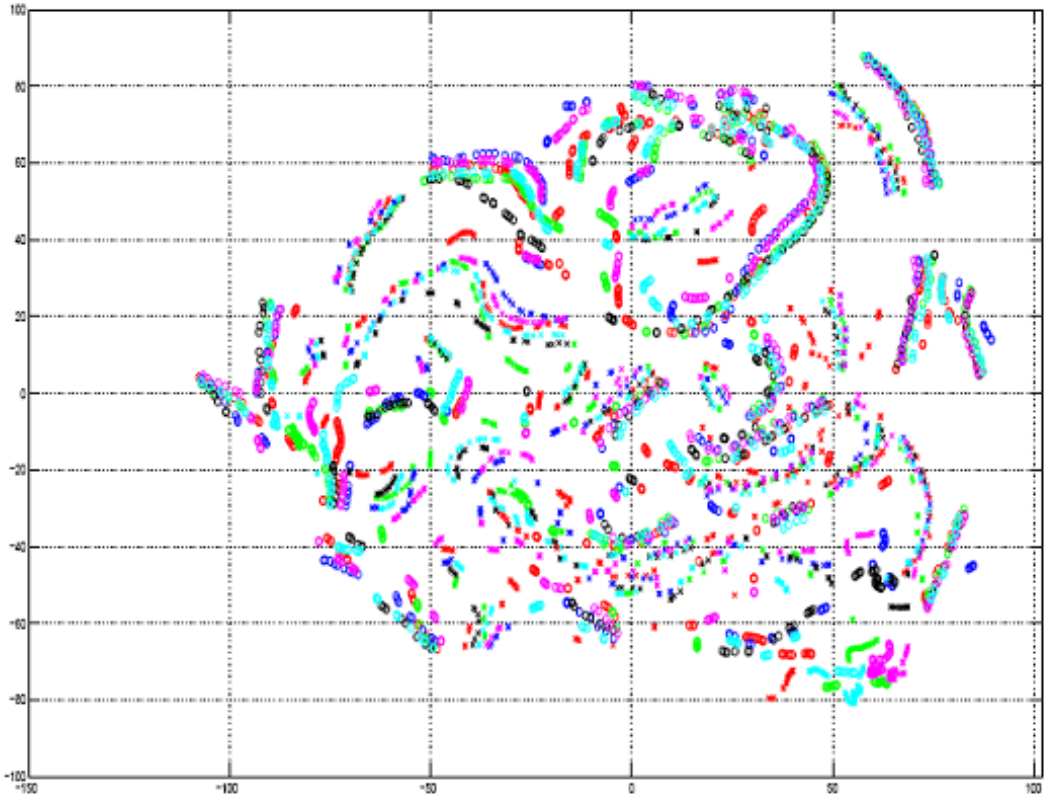
- Non-linear temporal sequence
- Speaking styles (conversational, paralinguistic information)
- Speaker variations (accents, dialects)
- Noise conditions (speech, noise, music, etc.)
- Multiple concurrent speakers
- Enormous variability in the spectral space with temporal and frequency correlations



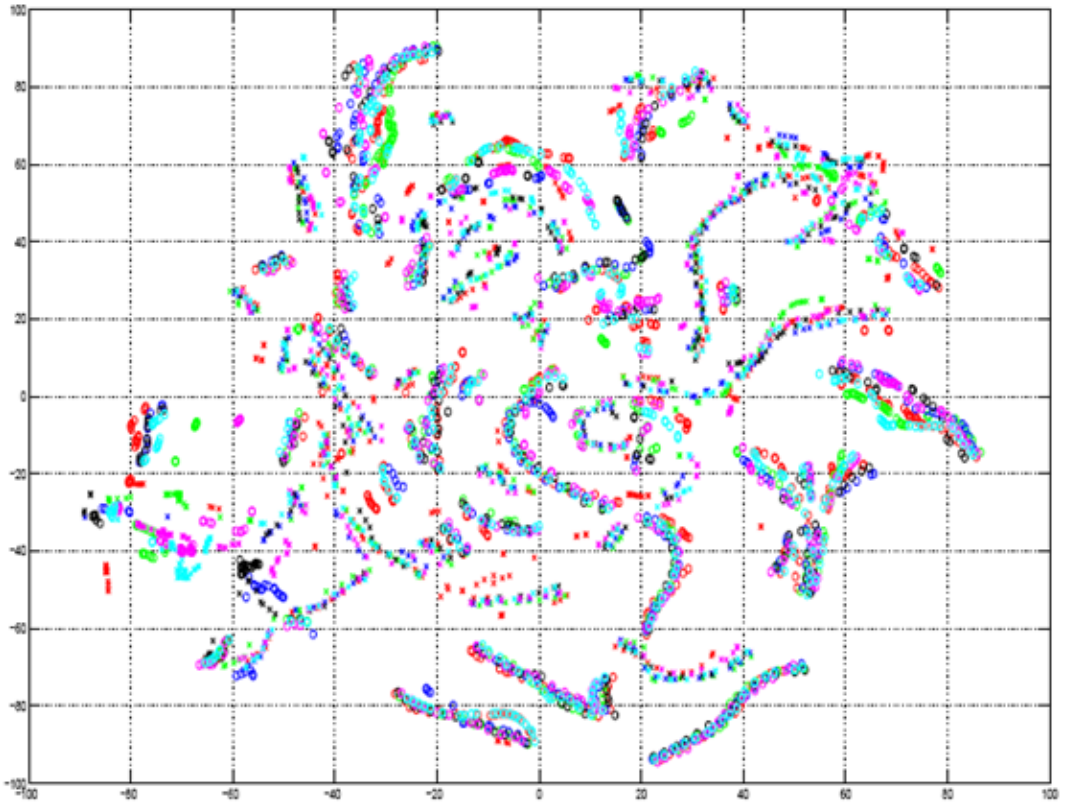
Why Deepness in Speech?

- Want to analyze activations by different speakers to see what DBN is capturing
- t-SNE [van der Maaten, JMLR 2008] (Stochastic neighbor Embedding)Nplots produce 2-D embeddings in which points that are close together in the high-dimensional vector space remain close in the 2-D space
- Similar phones from different-speakers are grouped together better at higher layers
- Better discrimination between classes is performed at higher layers

What does each layer capture?

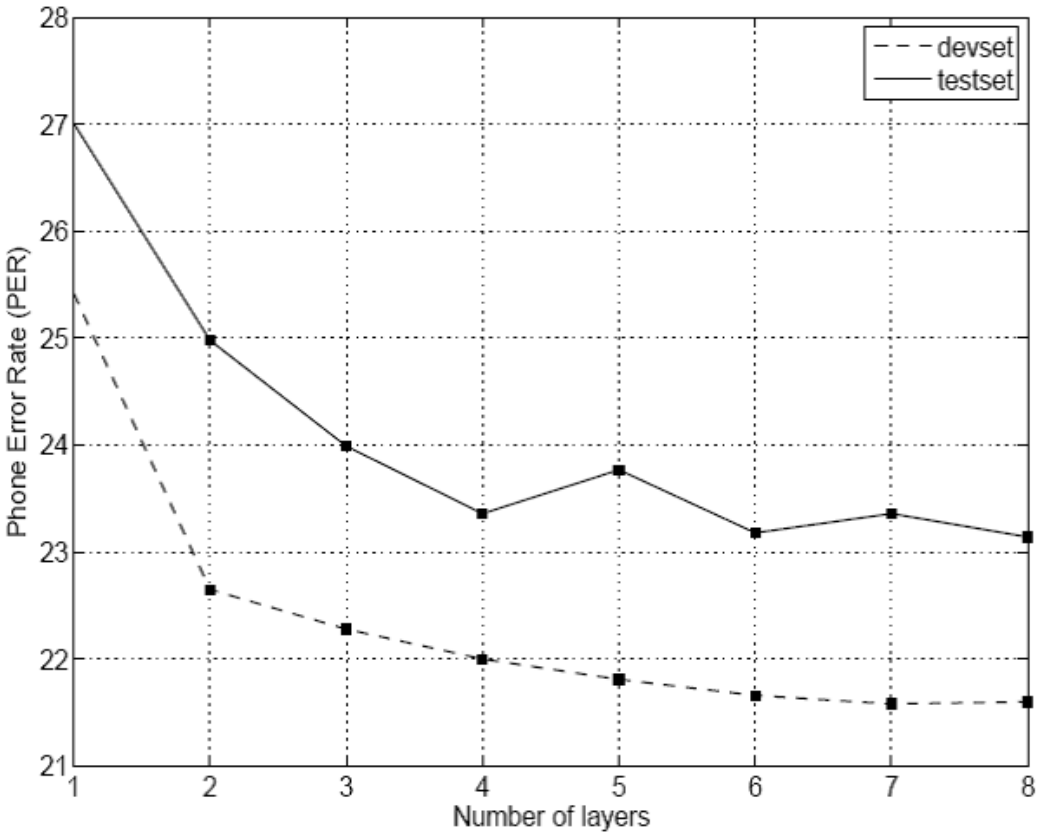


Second layer



Experimental Observation for impact of many layers

On TIMIT phone recognition task:



Initialization of Neural networks

- The training of NNs is a non-convex optimization problem.
- Training can get stuck in local optima.
- DNNs can also overfit strongly due to the large number of parameters.
- Random initialization works well for shallow networks.
- Deep NNs can be initialized with pre-training algorithms - either unsupervised or supervised.

Initialization of Neural networks

- What is unsupervised pretraining?
  - Learning of multi-layer generative models of unlabelled data by learning one layer of features at a time.
- Keep the efficiency and simplicity of using a gradient method for adjusting the weights, but use it for modeling the structure of the input.
- Adjust the weights to maximize the probability that a generative model would have produced the input.
- But this is hard to do.

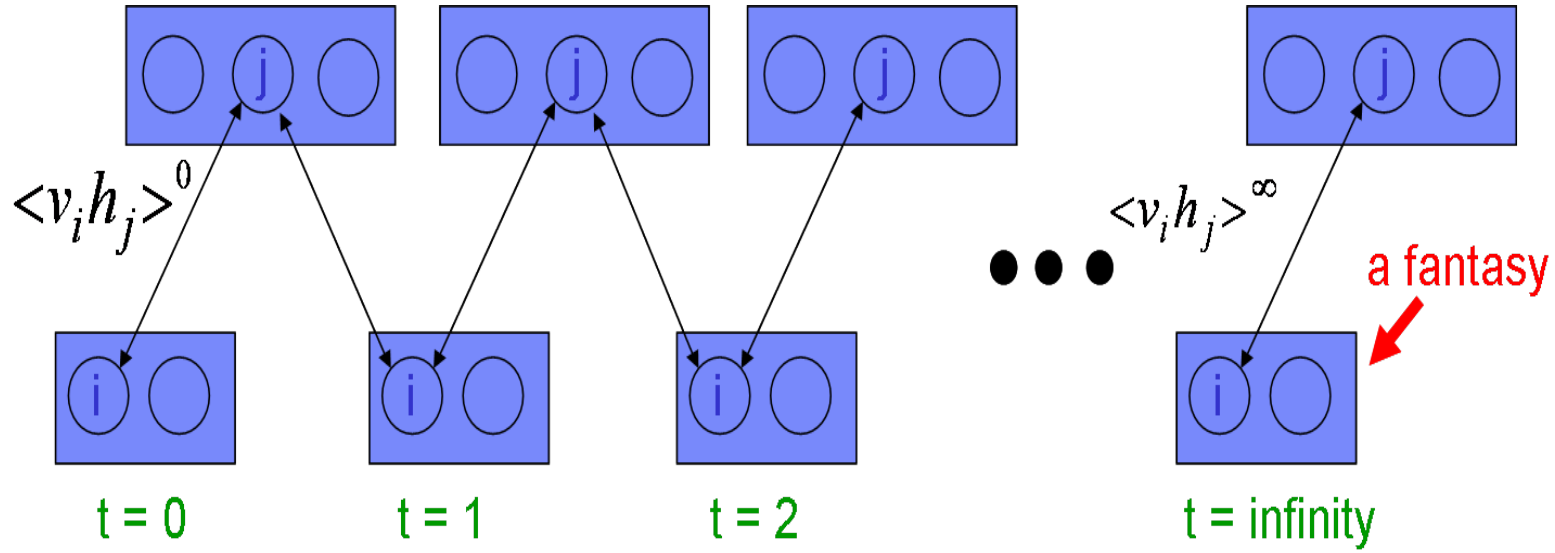
Pretraining

- Learning is easy if we can get an unbiased sample from the posterior distribution over hidden states given the observed data.
- Monte Carlo methods can be used to sample from the posterior. But its painfully slow for large, deep models.
- In the 1990s people developed variational methods for learning deep belief nets These only get approximate samples from the posterior. Nevetheless, the learning is still guaranteed to improve a variational bound on the log probability of generating the observed data.
- If we connect the stochastic units using symmetric connections we get a Boltzmann Machine (Hinton and Sejnowski, 1983). If we restrict the connectivity in a special way, it is easy to learn a Restricted Boltzmann machine.

Restricted Boltzmann Machines

In an RBM, the hidden units are conditionally independent given the visible states. This enables us to get an unbiased sample from the posterior distribution when given a data-vector.

A Maxmimum Likelihood Learning Algorithm for an RBM



Start with a training vector on the visible units.

Then alternate between updating all the hidden units in parallel and updating all the visible units in parallel.

$$\frac{\partial \log p(v)}{\partial w_{ij}} = <v_i h_j>^0 - <v_i h_j>^\infty$$

Training a deep network

- First train a layer of features that receive input directly from the audio.
- Then treat the activations of the trained features as if they were input features and learn features of features in a second hidden layer.
- It can be proved that each time we add another layer of features we improve a variational lower bound on the log probability of the training data.
- The proof is complicated. But it is based on a neat equivalence between an RBM and a deep directed graphical model

Initialzing a deep network in a supervised fashion

- First learn one layer at a time greedily. Then treat this as pre-training that finds a good initial set of weights which can be fine-tuned by a local search procedure.
- Start with training a NN with a single hidden layer, discard output layer, add a second hidden layer and a new output layer to the NN, . . .
- Backpropagation can be used to fine-tune the model for better discrimination.

Why does it work?

- Greedily learning one layer at a time scales well to really big networks, especially if we have locality in each layer.
- We do not start backpropagation until we already have sensible feature detectors that should already be very helpful for the discrimination task. So the initial gradients are sensible and backprop only needs to perform a local search from a sensible starting point.

Another view

- Most of the information in the final weights comes from modeling the distribution of input vectors.
- The input vectors generally contain a lot more information than the labels.
- The precious information in the labels is only used for the final fine-tuning.
- The fine-tuning only modifies the features slightly to get the category boundaries right. It does not need to discover features.
- This type of backpropagation works well even if most of the training data is unlabeled. The unlabeled data is still very useful for discovering good features.

In speech recognition...

- We know with GMM/HMMs, increasing the number of context-dependent states (i.e., classes) improves performance
- MLPs typically trained with small number of output targets increasing output targets becomes a harder optimization problem and does not always improve WER
- It increases parameters and increases training time
- With Deep networks, pre-training putting weights in better space, and thus we can increase output targets effectively

Decoding: Hybrid Systems

How can NNs be used for acoustic modeling?

- In most approaches, NNs model the posterior probability p(s|x) of an HMM state s given an acoustic observation x.
- Advantage: existing HMM speech recognizers can be used.
- In recognition, the class-conditional probability p(x|s) is required, which can be calculated using Bayes rule: $p(x|s) = p(s|x)p(x)/p(s)$.
- $p(s)$ can be estimated as the relative frequency of s (priors).
- $p(x)$ is a constant in the maximization problem and can be discarded.
- This model is known as hybrid NN-HMM and was introduced by [Bourlard, 1993].

Performance of Deep Neural Networks

Impact of output targets on TIMIT phone recognition: Should the targets be Context-independent or context-dependent?

Number of Targets	WER
384	21.3
512	20.8
1024	19.4
2,220	**18.5**

Training Criteria

- The MSE criterion (ore robust to outliers) has been used in earlier work on NNs [Rumelhart, Hinton, et al. 1986].
- Better results are achieved with the Cross Entropy (CE) criterion. The MSE criterion has many plateaus which make it hard to optimize, see for example [Glorot, Bengio et al.2010].
- The CE criterion for NNs without hidden layers is a convex optimization problem (softmax activation function leads to log-linear model in this case).
- The CE criterion is very popular for speech recognition.

Cross Entropy Criterion

$$L_{\text{XENT}}(\theta) = \sum_{r=1}^{R} \sum_{t=1}^{T_r} \sum_{i=1}^{I} \hat{y}_{rt}(i) \log \frac{\hat{y}_{rt}(i)}{y_{rt}(i)}$$

Backpropagation adjusts θ to minimize the above loss function

Typically this criterion is used in conjunction with soft-max non-linearity in the output layer.

Then, the derivative of the loss function with respect to the activations reduces to a simple expression:

$y_{rt}(i) - \hat{y}_{rt}(i)$

Cross Entropy Criterion

How to set learning rates properly?

- Newbob is an effective heuristic that prevents overfitting by early stopping.
- Use a fixed learning rate per epoch, start with a large learning rate.
- After every epoch, check the error rate on a held-out set. If it does not decrease sufficiently, halve the learning rate.
- Terminate when there is no further improvement on the validation set.

Cross Entropy: Results

Method	WER
Speaker-independent GMM/HMM System	24.5
DNN [University of Toronto, 2010]	**20.7**

Method	WER
Speaker-independent, discriminatively-trained GMM/HMM System	18.9
DNN [Microsoft, 2011]	**16.1**

Sequence Training

- Criteria based on sequence classification are more closely related to word error rate than criteria based on frame classification.
- Intuitively, it performs discriminative training: maximize the probability of the correct sequence while minimizing the probability of competing sequences
- Use of word lattices to compactly represent the reference and the competing hypotheses, makes it possible to train on large-vocabulary tasks with large number of training samples.
- Use of a scalable Minimum Bayes-Risk criterion (sMBR) for sequence discrimination, wherein the gradient is now computed with respect to the sequence-classification criterion and Hessian-free optimization.

Sequence Training

$$L_{\text{sMBR}}(\theta) = \sum_{r=1}^{R} \frac{\sum_{W \in \mathcal{W}_r} P(X_r|W_r,\theta)^{\kappa} P(W_r) d(Y, Y_r)}{\sum_{W \in \mathcal{W}} P(X_r|W,\theta)^{\kappa} P(W)}$$

Variants of MMI, MPE that you saw in last week's lecture.

Sequence Training: Results

10-15% relative improvement on speech recognition tasks:

	Cross Entropy	Sequence Training
RNN/CNN (English Conversational)	10.4	9.2
DNNs (English Broadcast news)	16.5	15.1
DNNs (Noisy Levantine Arabic)	58.1	50.7

LVCSR Performance across well-benchmarked tasks

	50 Hour English Broadcast News	300 Hour Switchboard Telephony	400 Hour English Broadcast News
GMM/HMM	17.2	14.3	16.5
DBN	**14.9**	**12.2**	**15.2**
% Relative Improvement	**13.4**	**14.7**	**7.9**

Issues with Neural Networks

- Training time!!
  - Architecture: Context of 11 Frames, 2,048 Hidden Units, 5 layers, 9,300 output targets implies 43 million parameters !!
  - Training time on 300 hours (100 M frames) of data takes 30 days on one 12 core CPU)!
  - Compare to a GMM/HMM system with 16 M parameters that takes roughly 2 days to train!!
  - GPUs to the rescue!
- Size, connectivity, feature representations . . .

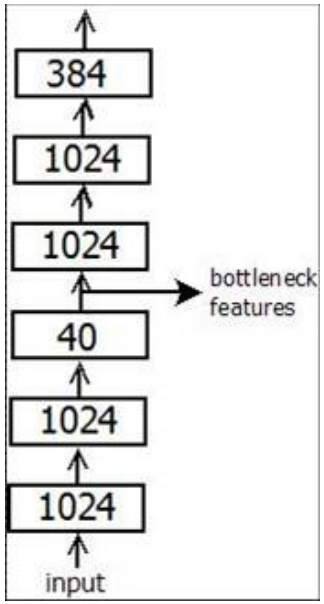
One way to speed up..

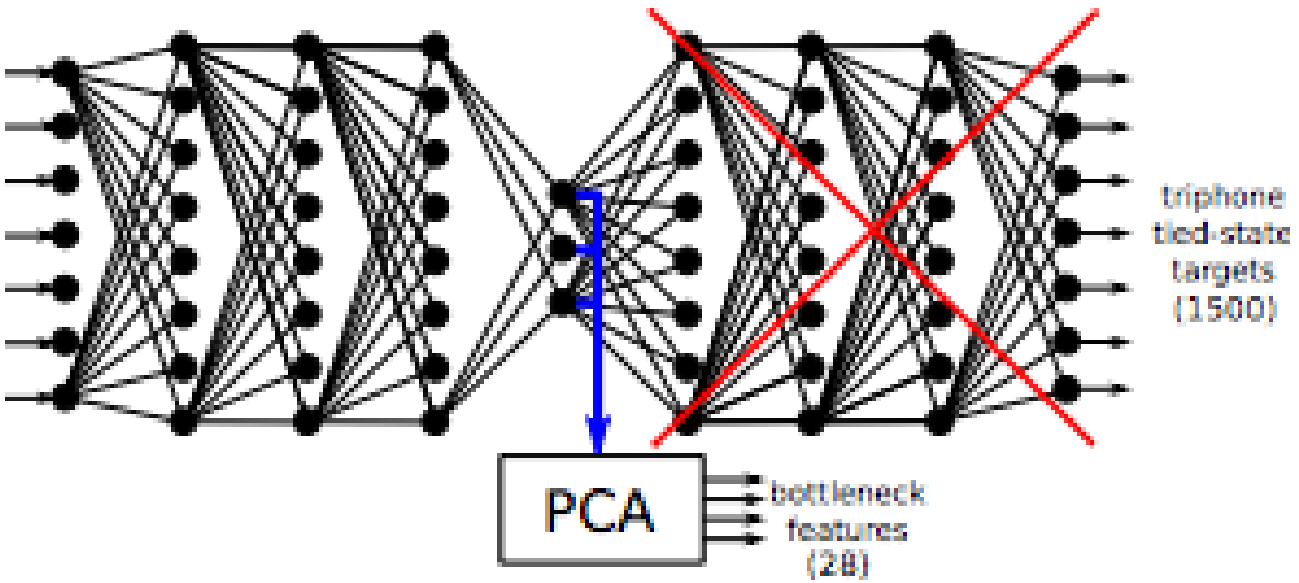
- Matrix computations on the GPUs (4x to 6x speed ups)
- Distributed training on GPUs: Synchronous SGD.
- Asynchronous algorithms, such as ASGD (and its variants, e.g., elastic averaging) on CPUs or GPUs. These can operate on data or on layers (less trivial)
- Hessian-free training on multiple-GPUs
- Massively parallel hardware (Blue Gene)

Another way to speed up..

- One reason NN training is slow is because we use a large number of output targets (context dependent targets) - as high as 10000.
- Bottleneck NNs
  - Generally have few output targets - these are features we extract to train GMMs on them.
  - Traditional bottleck configurations, introduce a bottlneck layer just prior to the output targets that can have fewer units and no non-linearity (low rank methods).
- Once, bottleneck features are extracted, we can use standard GMM processing techniques on these features.

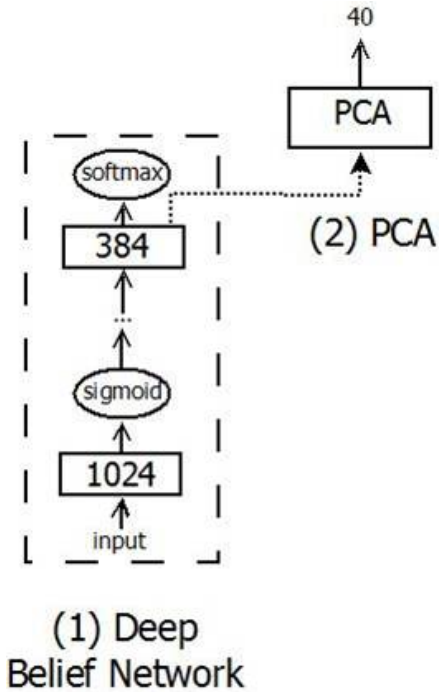
Example bottleneck feature extraction





Example bottleneck feature extraction

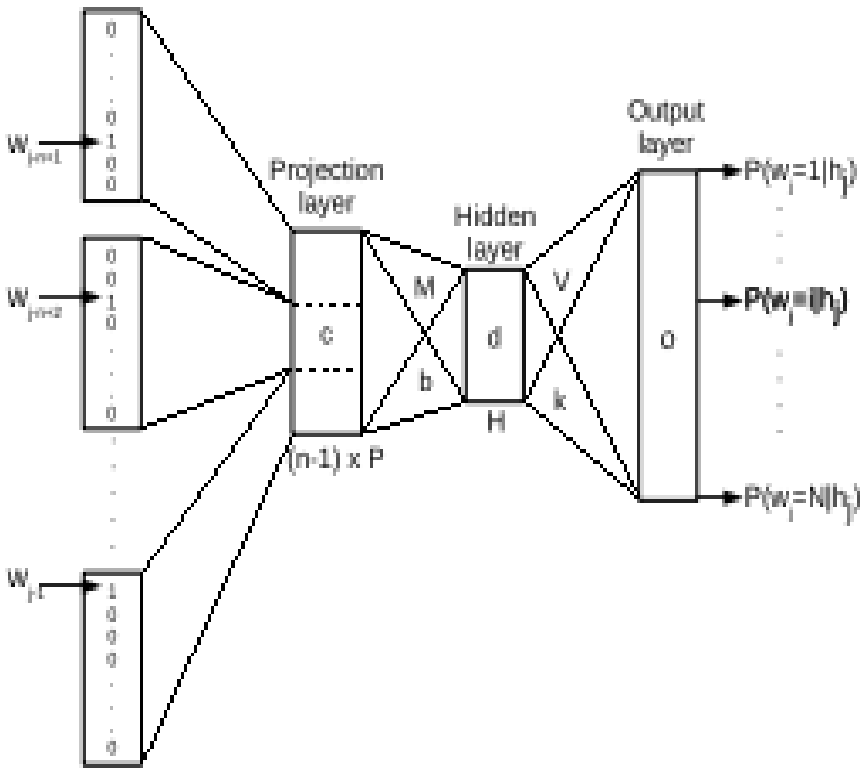
In this configuration, the bottleneck features are extracted at the output layer, just before the non-linearity.



Neural networks in Speech Recognition: Language modeling

- Conventional n-gram LM
  - Words are treated as discrete entities
  - Data sparseness issues mitigated by smoothness techniques
- Neural Network LM [Bengio et al., 2003, Schwenk, 2007]
  - Words are embedded in a continuous space
    - Semantically or grammatically related words can be mapped to similar locations
  - Probability estimation is done in this continuous space
  - NNLMs can achieve better generalization for unseen n-grams

Neural networks in Speech Recognition: Language modeling



Model Parameters

- Projection layer

  $c : (n-1) \times P$ dimensional

- Hidden layer

  $d_j = \tanh\left(\sum_{l=1}^{(n-1)\times P} M_{jl} c_l + b_j\right)$

- Output layer

  $o_i = \sum_{j=1}^{H} V_{ij} d_j + k_i$

  $p_i = \frac{\exp(o_i)}{\sum_{r=1}^{N} \exp(o_r)} = P(w_j = i | h_j)$

Single layer NNLM = 1 hidden layer with nonlinear activations.

Neural networks in Speech Recognition: Language modeling

- Introduced by [Bengio et al., 2003].
- Extended to large vocabulary speech recognition [Schwenk, 2007].
- Used for syntactic-based language modeling [Emami, 2006], [Kuo et al., 2009].
- Reducing computational complexity:
  - Using shortlist at output layer [Schwenk, 2007].
  - Hierarchical decomposition of output probabilities [Morin and Bengio, 2005], [Mnih and Hinton, 2008], [Son Le et al., 2011].
- Recurrent neural networks were used in LM training [Mikolov et al., 2010], [Mikolov et al., 2011].
- Deep Neural Network Models [Arisoy et al., 2012]

Neural networks in Speech Recognition: Language modeling

How do you use a NN LM in speech recognition?

- Rescoring a lattice (most commonly used approach)
- During decoding (represent the NN LM as a comventional n-gram model) [Arisoy et al, 2014]

Do NN LMs help?

Results on WSJ (23.5M words)

Evaluation: Lattice rescoring with DNN LMs

- 4-gram DNN LMs only (no interpolation with a baseline LM!)

Models	Perplexity	WER(%)
4-gram LM	114.4	22.3
DNN LM: h=500, d=30		
with 1 layer (NNLM)	115.8	22.0
with 4 layers	108.0	21.6
DNN LM: h=500, d=60		
with 1 layer (NNLM)	109.3	21.5
with 3 layers	105.0	21.3
DNN LM: h=500, d=120		
with 1 layer (NNLM)	104.0	21.2
with 3 layers	102.8	20.8

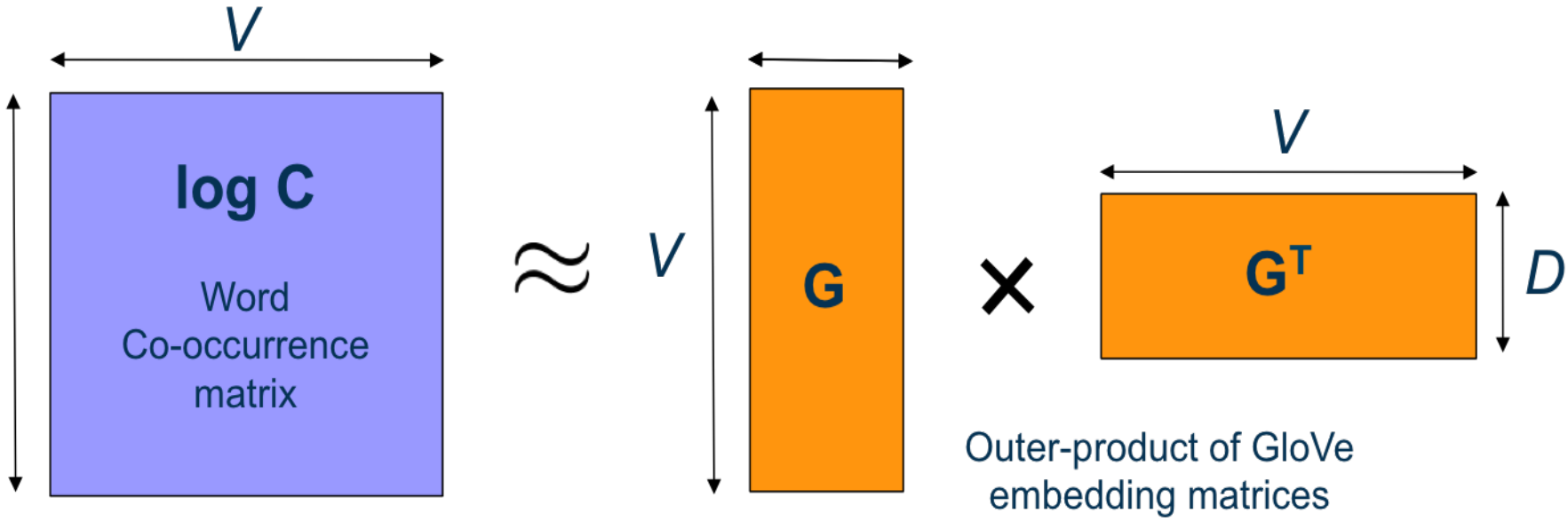
Semantic word embeddings

Semantic word embedding algorithms such as, word2vec and GloVe (Global Vectors for Word Representations) aim to capture semantic information from text.

Sim. Rank	Speech		Machine		Learning	
	GloVe	FNNLM	**GloVe**	FNNLM	**GloVe**	FNNLM
1	**Remarks**	Address	**Machines**	Stun	**Learn**	Learn
2	**Address**	Event	**Guns**	Pellet	**Teaching**	Learned
3	**Speeches**	Ceremony	**Gun**	Celebratory	**Learned**	Learns
4	**Comments**	Statement	**Hand**	Millimeter	**Skills**	Complain
5	**Bush**	Remarks	**Automatic**	Sharpnel	**Teach**	Confirmation

Semantic word embeddings

GloVe is bi-linear approximation of the word co-occurence matrix computed on the training data



The embedding matrix

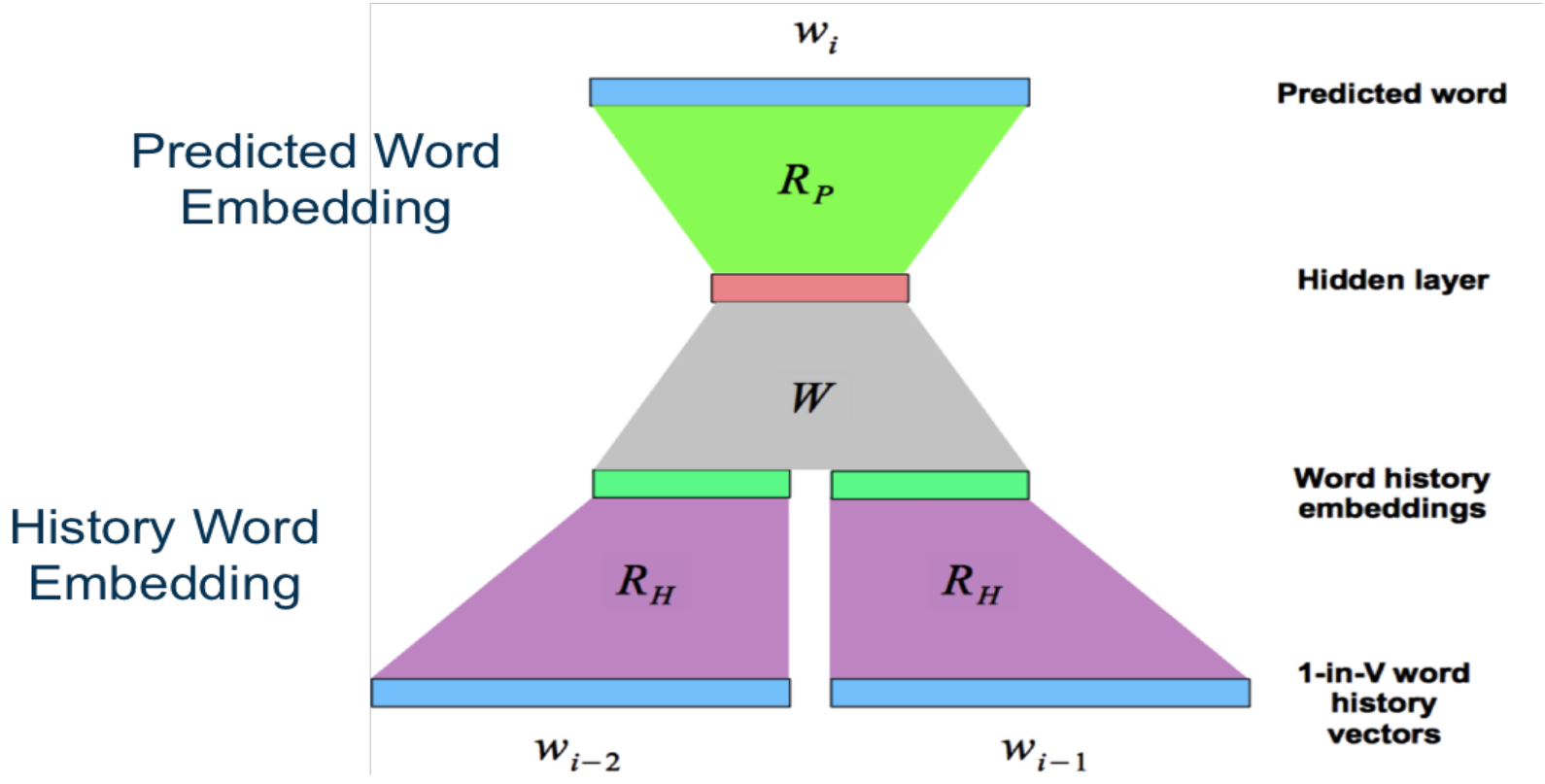
Embedding matrix is estimated as:

$$\mathbf{G}^*, \mathbf{b}^* = \arg\min_{\mathbf{G},\mathbf{b}} \sum_{i,j=1}^{V} f(\mathbf{C}(i,j)) \Big(\mathbf{G}(i)\mathbf{G}(j)^T + b(i) + b(j) - \log \mathbf{C}(i,j) \Big)^2$$

$$f(x) = \min\{1, (x/x_{\min})^{\alpha}\}$$

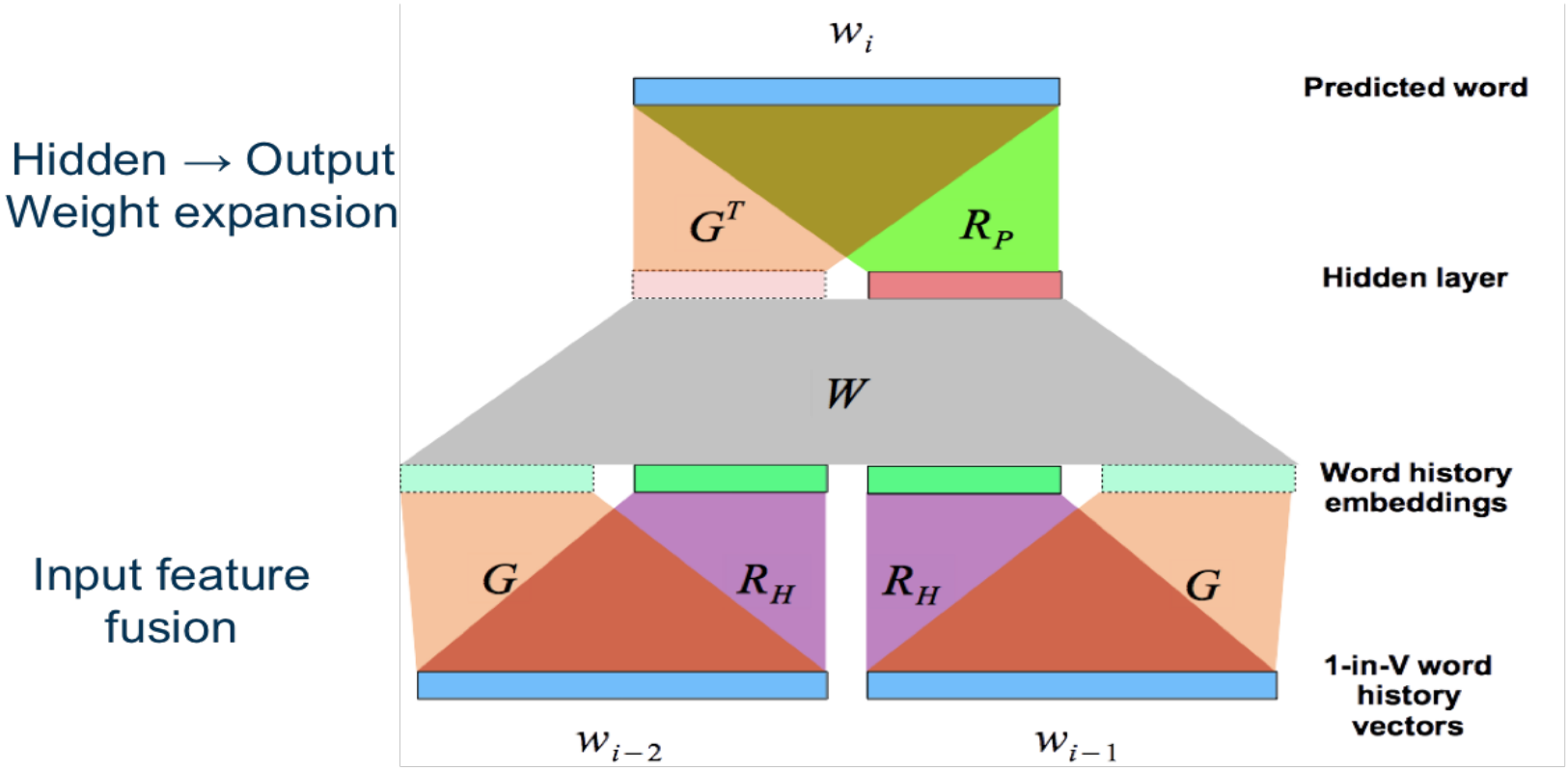
Recall ...

The feed forward NNLM predicts the next word by passing the continuous embeddings of the history words through a feed forward NN LM



Now ...

Now, to use the semantic word embeddings, input feature concatenation fuses two diverse embeddings, semantic and previous word embedding



Results on a LVCSR task

Results on a broadcast news transcription task:

LM	WER	% Reduction
4gm KN	11.3%	-
4gm KN + FNNLM	11.0%	2.6%
4gm KN + SWE-FNNLM	**10.7%**	**5.3%**

References

Saon, George, et al. "The IBM 2015 English Conversational Telephone Speech Recognition System." arXiv preprint arXiv:1505.05899 (2015).

Berg, Ewout van den, Daniel Brand, Rajesh Bordawekar, Leonid Rachevsky, and Bhuvana Ramabhadran, "Efficient GPU Implementation of Convolutional Neural Networks for Speech Recognition." In Sixteenth Annua Conference of the International Speech Communication Association. 2015.

Bordawekar et al., "Accelerating Deep Convolution Neural Networks For Large-Scale Speech Tasks Using GPUs", GOU Technology Conference, 2015.

Audhkhasi, Kartik, Abhinav Sethy, and Bhuvana Ramabhadran. "Diverse Embedding Neural Network Langua Models." arXiv preprint arXiv:1412.7063 (2014).

Arisoy, Ebru, Abhinav Sethy, Bhuvana Ramabhadran, and Stanley Chen. "Bidirectional recurrent neural netw

language models for automatic speech recognition." In IEEE ICASSP, 2015.

Sethy, Abhinav, Stanley Chen, Ebru Arisoy, and Bhuvana Ramabhadran. "UNNORMALIZED EXPONENTIAL AND NEURAL NETWORK LANGUAGE MODELS.", ICASSP, 2015

Audhkhasi, K., Sethy, A., Ramabhadran, B., "SEMANTIC WORD EMBEDDING NEURAL NETWORK LANGUAGE MODELS FOR AUTOMATIC SPEECH RECOGNITION", submitted ICASSP 2016.

Graves, A. and Jaitly, N. (2014). Towards end-to-end speech recognition with recurrent neural networks. In Proceedings of the 31st International Conference on Machine Learning, 2014.

References

J. Cui et al., „MULTILINGUAL REPRESENTATIONS FOR LOW RESOURCE SPEECH RECOGNITION AND KEYWORD SEARCH", *Proc. of IEEE ASRU*, 2015.

R. Fernandez, "Using Deep Bidirectional Recurrent Neural Networks for Prosodic-Target Prediction in a Unit-Selection Text-to-Speech System", *Proc. Interspeech*, 2015.

B. Kingsbury, "Lattice-Based Optimization of Sequence Classification Criteria for Neural-Network Acoustic Mo eling," in *Proc. ICASSP*, 2009.

B. Kingsbury, T. N. Sainath, and H. Soltau, "Scalable Minimum Bayes Risk Training of Deep Neural Network Acoustic Models Using Distributed Hessian-free Opti- mization," in *Proc. Interspeech*, 2012.

T. Sainath, I. Chung, B, Ramabhadran, et al, "Parallel Deep Neural Network Training for LVCSR Tasks using Blue Gene/Q" in *Proc. Interspeech* 2014.

Chung, I.H., Sainath, T.N., Ramabhadran, B., Pichen, M., Gunnels, J., Austel, V., Chauhari, U. and Kingsbury, B., 2014, November. Parallel deep neural network training for big data on blue gene/Q. In High Performance Computing, Networking, Storage and Analysis, SC14: International Conference for (pp. 745-753). IEEE.

References

Jeff Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Quoc Le, Mark Mao, Marc'Aurelio Ranzato, Antrew Senior, Paul Tucker, Ke Yang, Andrew Ng Large Scale Distributed Deep Networks. NIPS 2012.

B. Kingsbury, T. N. Sainath, and H. Soltau, "Scalable Minimum Bayes Risk Training of Deep Neural Network Acoustic Models Using Distributed Hessian-free Optimization," in Proc. Interspeech, 2012.

G. Saon and H. Soltau, "A comparison of Two Optimization Techniques for Sequence Training of Deep Neural Networks," in Proc. ICASSP 2014.

T. N. Sainath, I. Chung, B. Ramabhadran, M. Picheny, J. Gunnels, B. Kingsbury, G. Saon, V. Austel, U. Chaudhri, "Parllel Deep Neural Netowkr Training for LVCSR using Blue Gene/Q," in Proc. Interspeech, September 2014.

F. Seide, H. Fu, J. Droppo, G. Li and D. Yu, "1-Bit Stochastic Gradient Descent and Application to Data-Parallel Distributed Training of Speech DNNs," in Interspeech 2014.

Language Modeling References

- Holger Schwenk, Jean-Luc Gauvain, *Continuous space language models*, Comput. Speech Lang., 21(3):492-518, July 2007.
- Yoshua Bengio, Rejean Ducharme, Pascal Vincent, and Christian Jauvin, *A neural probabilistic language model*, Journal of Machine Learning Research,3:1137-1155, 2003.
- Ahmad Emami, *A neural syntactic language model*, Ph.D. thesis, Johns Hopkins University, Baltimore, MD, USA, 2006.
- H-K. J. Kuo, L. Mangu, A. Emami, I. Zitouni, and Y-S. Lee, *Syntactic features for Arabic speech recognition*, In proc. ASRU, pp.327-332, Merano, Italy, 2009.
- Andriy Mnih and Geoffrey Hinton, *A scalable hierarchical distributed language model*, In Proc. NIPS, 2008.
- Frederic Morin and Yoshua Bengio, *Hierarchical probabilistic neural network language model*, In Proc. AISTATS, pp. 246-252, 2005.
- Hai Son Le, Ilya Oparin, Alexandre Allauzen, Jean-Luc Gauvain, and Francois Yvon, *Structured output layer neural network language model*, In. Proc. ICASSP, pp. 5524-5527, Prague, 2011.

Language Modeling References

- Tomas Mikolov, Martin Karafiat, Lukas Burget, Jan Cernocky, and Sanjeev Khudanpur, *Recurrent neural network based language model*, In Proc. Interspeech, pp. 1045-1048, 2010.
- Tomas Mikolov, Anoop Deoras, Daniel Povey, Lukas Burget, and Jan Cernocky, *Strategies for training large scale neural network language models*, In Proc. ASRU, pp. 196-201, 2011.
- E. Arisoy, T. N. Sainath, B. Kingsbury, B. Ramabhadran, *Deep Neural Network Language Models*, In Proc. of NAACL-HLT, 2012.
- K. Audhkhasi, A. Sethy, and B. Ramabhadran, *Semantic Word Embedding Neural Network Language Models for Automatic Speech Recognition*, In Proc. ICASSP, 2016.