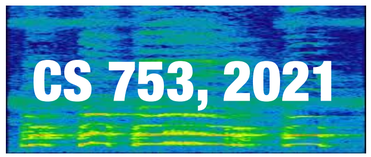
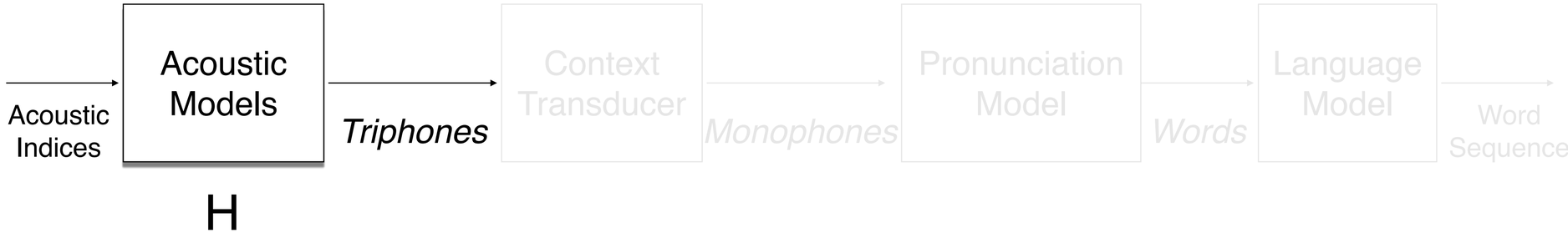
Feedforward Neural Network Acoustic Models

Lecture 6a

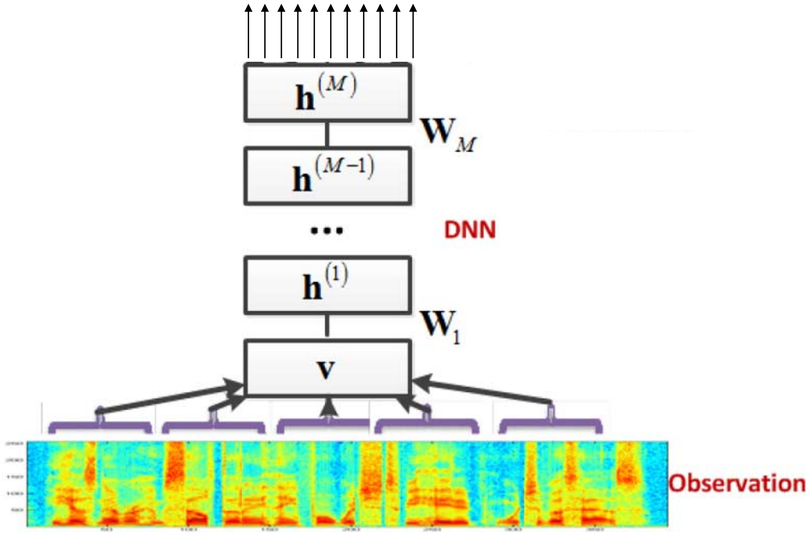


Instructor: Preethi Jyothi, IITB

DNN-based acoustic models?



Phone posteriors



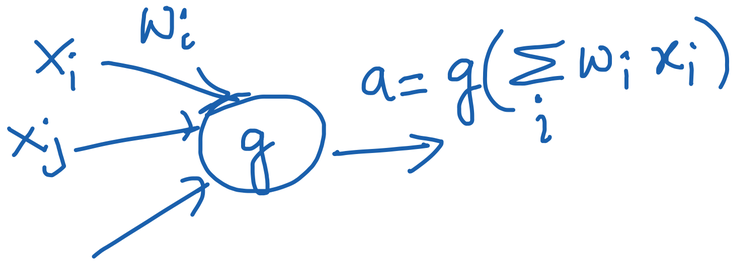
Can we use deep neural networks instead of HMMs to learn mappings between acoustics and phones?

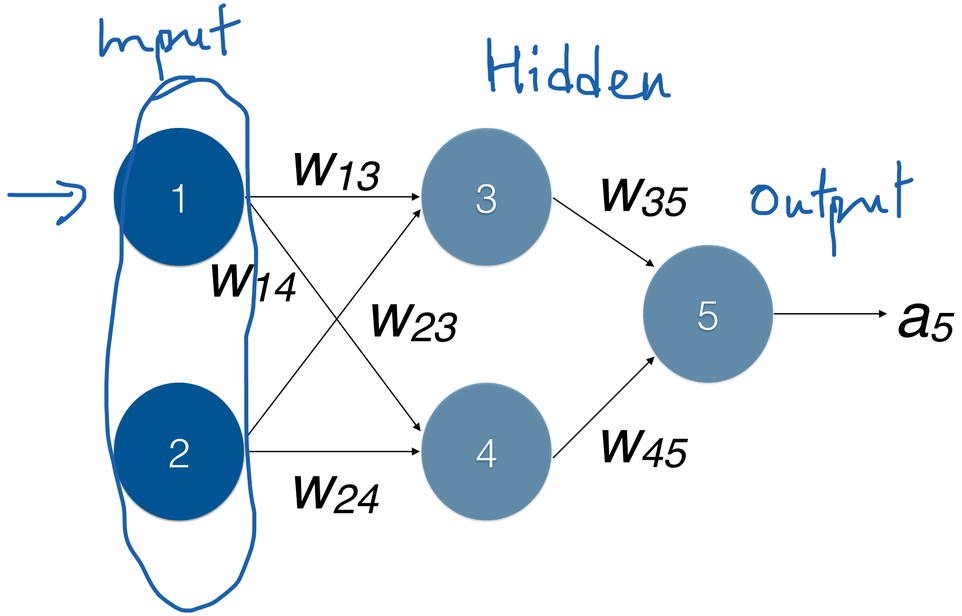
Resulting FST

H

Feed-forward Neural Network

Parameterized Model

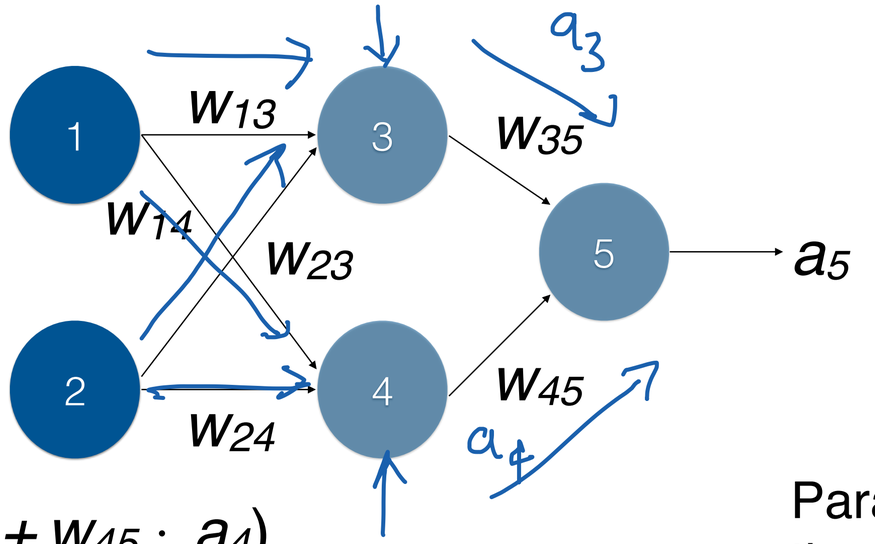




Affine

Feed-forward Neural Network

Parameterized Model



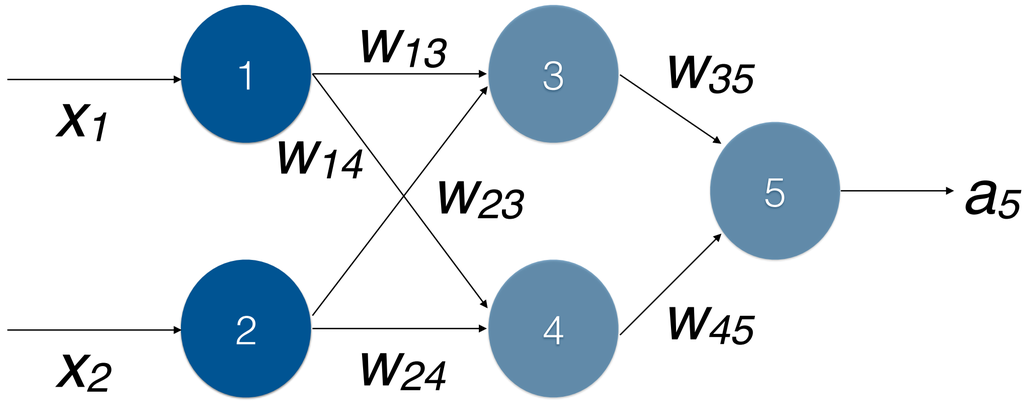
$$a_5 = g(w_{35} \cdot a_3 + w_{45} \cdot a_4)$$

$$= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))$$

Parameters of the network: all w_{ij} (and biases not shown here)

Feed-forward Neural Network

Parameterized Model



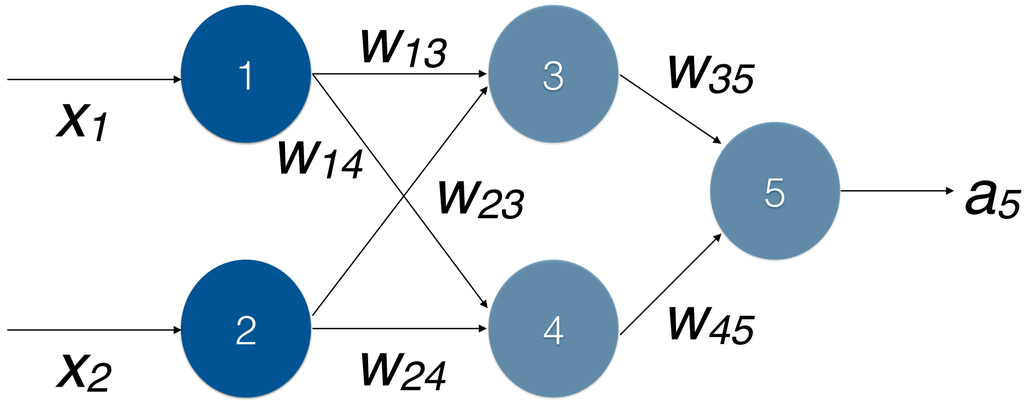
$$a_5 = g(w_{35} \cdot a_3 + w_{45} \cdot a_4)$$

$$= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))$$

Parameters of the network: all w_{ij} (and biases not shown here)

Feed-forward Neural Network

Parameterized Model



$$a_5 = g(w_{35} \cdot a_3 + w_{45} \cdot a_4)$$

$$= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))$$

Parameters of the network: all w_{ij} (and biases not shown here)

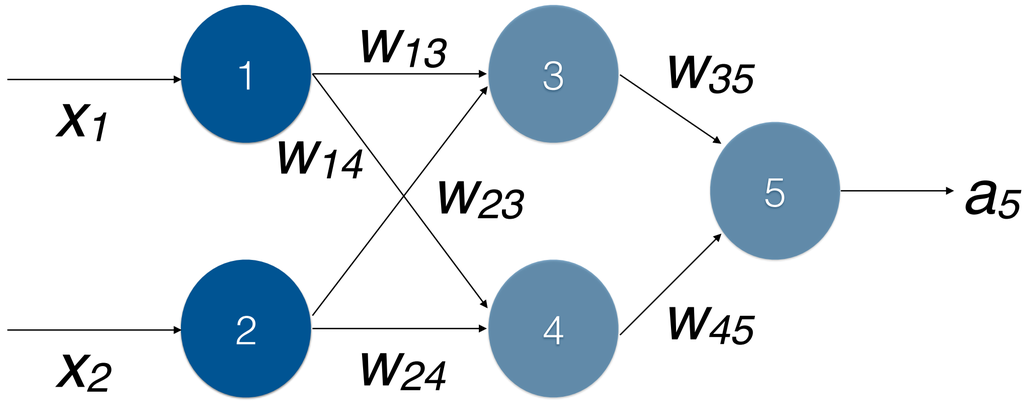
If **x** is a 2-dimensional vector and the layer above it is a 2-dimensional vector **h**, a fully-connected layer is associated with:

$$\mathbf{h} = \mathbf{x}\mathbf{W} + \mathbf{b}$$

where w_{ij} in **W** is the weight of the connection between i^{th} neuron in the input row and j^{th} neuron in the first hidden layer and **b** is the bias vector

Feed-forward Neural Network

Parameterized Model

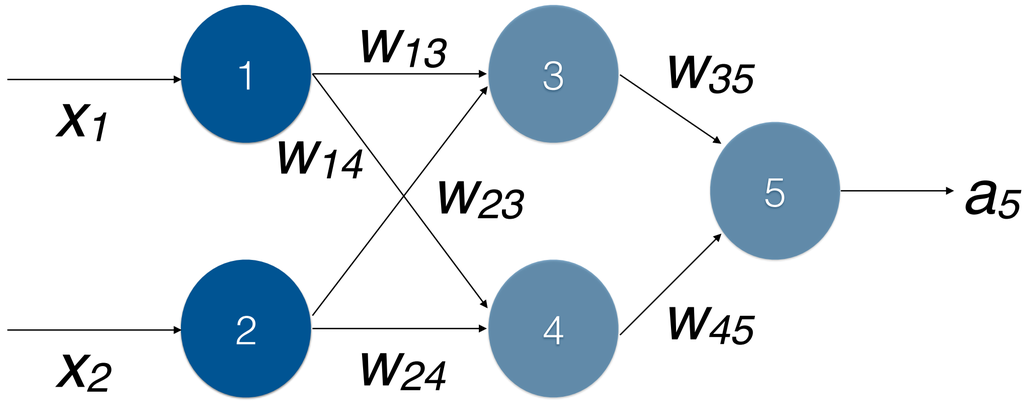


$$a_5 = g(w_{35} \cdot a_3 + w_{45} \cdot a_4)$$

$$= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))$$

Feed-forward Neural Network

Parameterized Model



$$a_5 = g(w_{35} \cdot a_3 + w_{45} \cdot a_4)$$

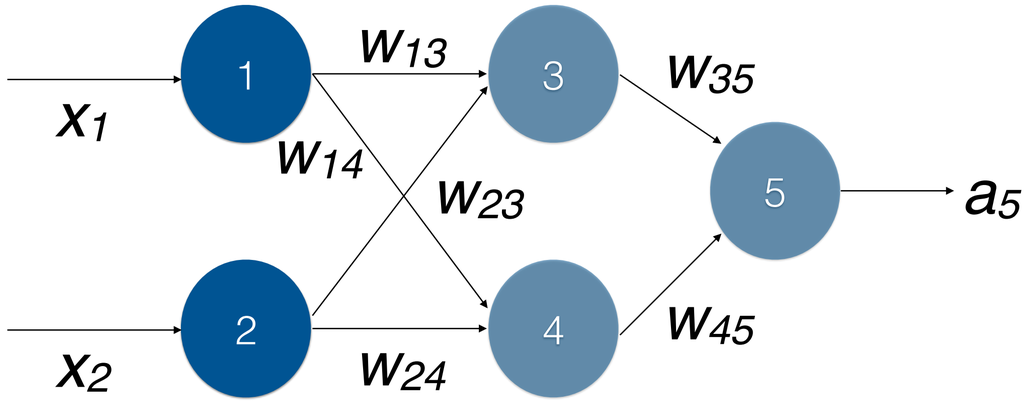
$$= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))$$

The simplest neural network is the perceptron:

$$\text{Perceptron}(x) = \mathbf{xW} + \mathbf{b}$$

Feed-forward Neural Network

Parameterized Model



$$a_5 = g(w_{35} \cdot a_3 + w_{45} \cdot a_4)$$

$$= g(w_{35} \cdot (g(w_{13} \cdot a_1 + w_{23} \cdot a_2)) + w_{45} \cdot (g(w_{14} \cdot a_1 + w_{24} \cdot a_2)))$$

The simplest neural network is the perceptron:

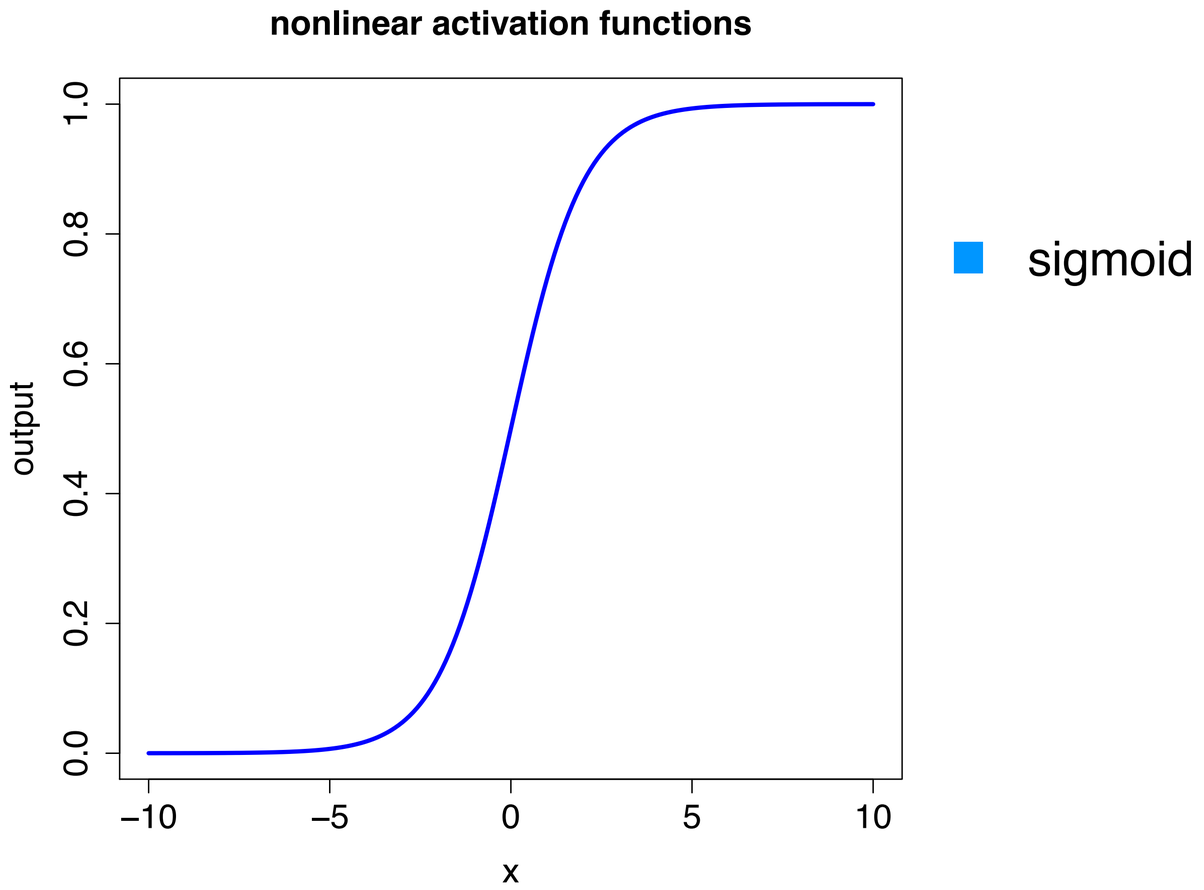
$$\text{Perceptron}(x) = \mathbf{xW} + \mathbf{b}$$

A 1-layer feedforward neural network has the form:

$$\text{MLP}(x) = g(\mathbf{xW}_1 + \mathbf{b}_1)\ \mathbf{W}_2 + \mathbf{b}_2$$

Common Activation Functions (g)

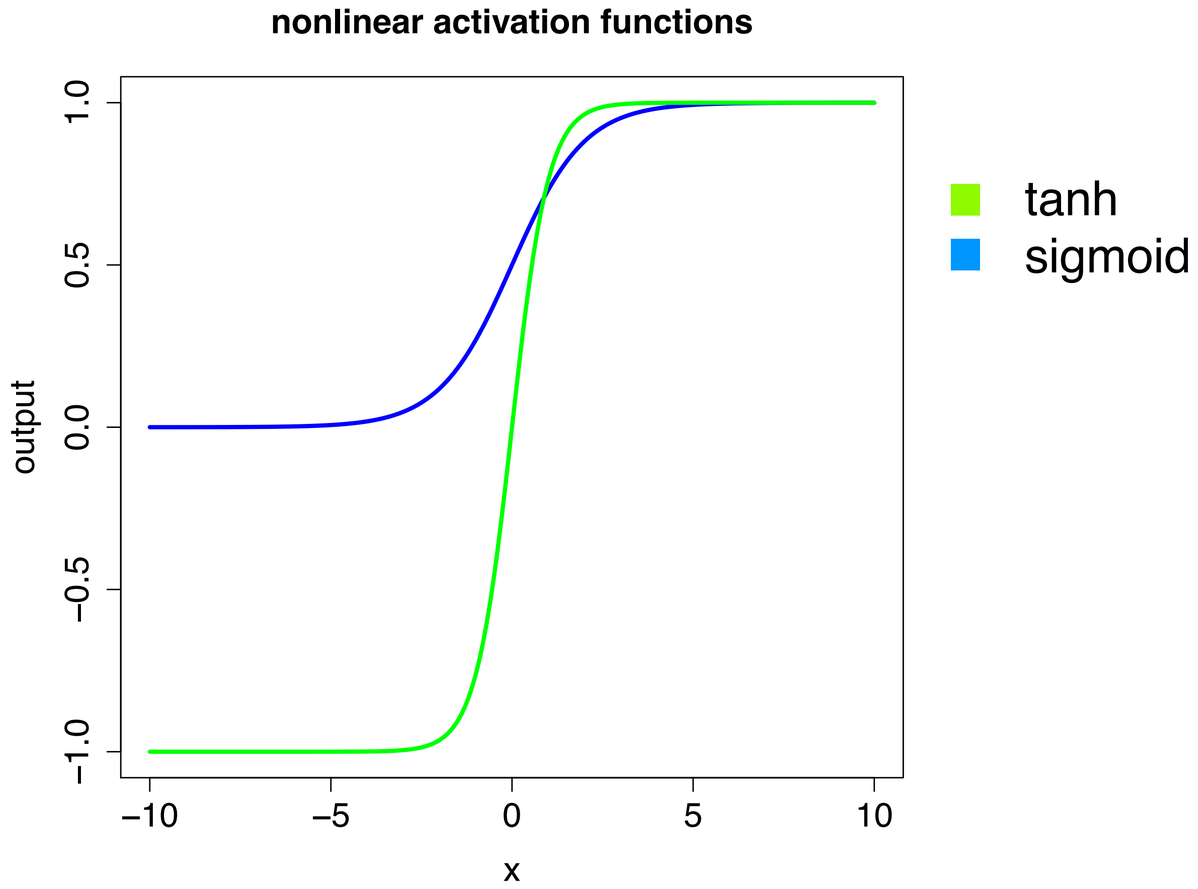
Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$



Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

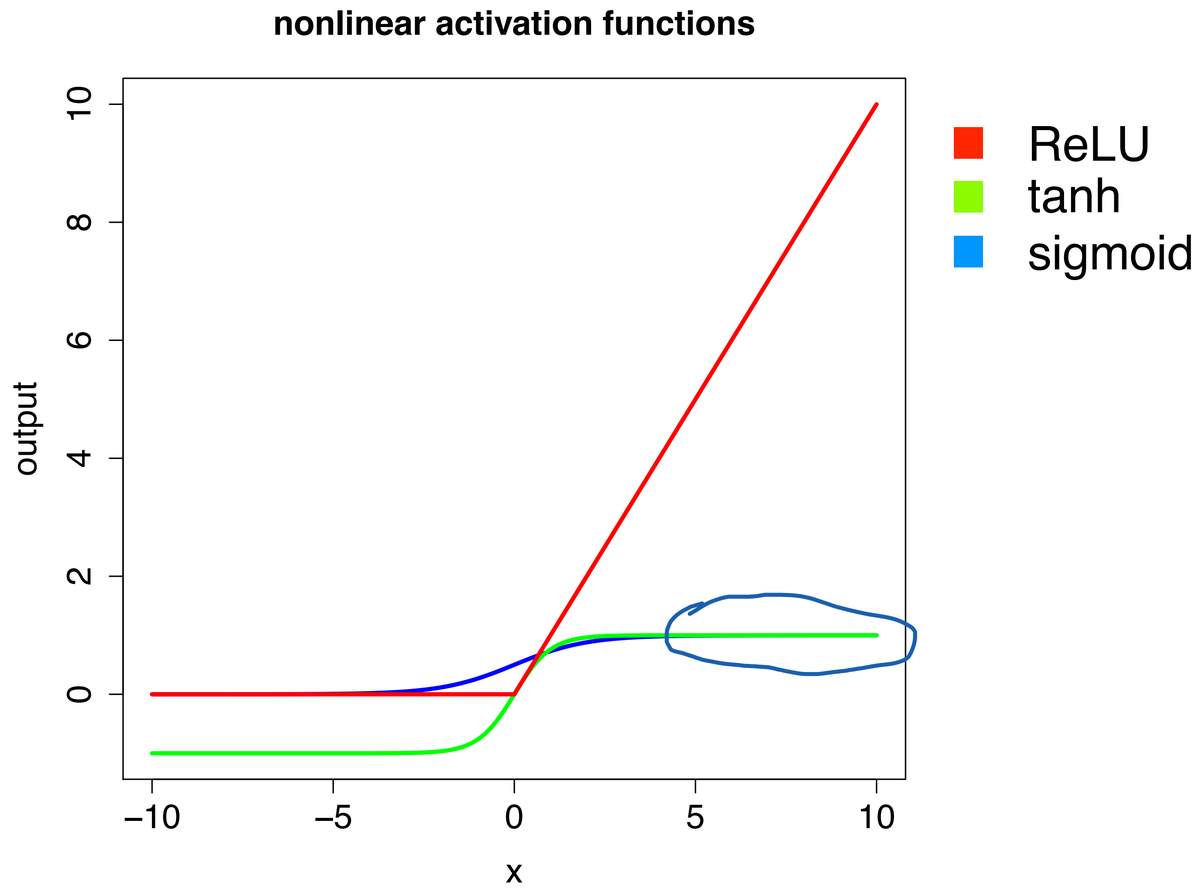


Common Activation Functions (g)

Sigmoid: $\sigma(x) = 1/(1 + e^{-x})$

Hyperbolic tangent (tanh): $\tanh(x) = (e^{2x} - 1)/(e^{2x} + 1)$

Rectified Linear Unit (ReLU): $\text{RELU}(x) = \max(0, x)$



Optimization Problem

- To train a neural network, define a loss function $L(y,\tilde{y})$: a function of the true output y and the predicted output $\tilde{y}$
- $L(y,\tilde{y})$ assigns a non-negative numerical score to the neural network's output, $\tilde{y}$
- The parameters of the network are set to minimise L over the training examples (i.e. a sum of losses over different training samples)
- L is typically minimised using a *gradient-based method*

Stochastic Gradient Descent (SGD)

SGD Algorithm

Inputs:

Function NN(x; θ), Training examples, $x_1 \ldots x_n$ and outputs, $y_1 \ldots y_n$ and Loss function L.

do until stopping criterion

Pick a training example x_i, y_i

Compute the loss $L(NN(x_i; \theta), y_i)$

Compute gradient of L, ∇L with respect to θ

$\theta \leftarrow \theta - \eta \nabla L$

weight update rule

done

Return: θ

Stochastic Gradient Descent (SGD)

SGD Algorithm

Inputs:

Function NN(x; θ), Training examples, $x_1 \ldots x_n$ and outputs, $y_1 \ldots y_n$ and Loss function L.

do until stopping criterion

Pick a training example x_i, y_i

Compute the loss $L(NN(x_i; \theta), y_i)$

Compute gradient of L, ∇L with respect to θ

$\theta \leftarrow \theta - \eta \nabla L$

done

Return: θ

Training a Neural Network

Training a Neural Network

Define the **Loss function** to be minimised as a node L

Training a Neural Network

Define the **Loss function** to be minimised as a node L

Goal: Learn weights for the neural network which minimise L

Gradient Descent: Find $\partial L/\partial w$ for every weight w, and update it as $w \leftarrow w - \eta \, \partial L / \partial w$

Training a Neural Network

Define the **Loss function** to be minimised as a node L

Goal: Learn weights for the neural network which minimise L

Gradient Descent: Find $\partial L/\partial w$ for every weight w, and update it as $w \leftarrow w - \eta \, \partial L / \partial w$

How do we efficiently compute $\partial L/\partial w$ for all w?

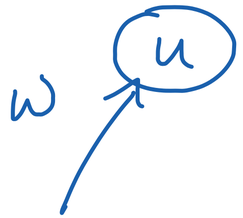
Training a Neural Network

Define the **Loss function** to be minimised as a node L

Goal: Learn weights for the neural network which minimise L

Gradient Descent: Find $\partial L/\partial w$ for every weight w, and update it as $w \leftarrow w - \eta\, \partial L/\partial w$

How do we efficiently compute $\partial L/\partial w$ for all w?

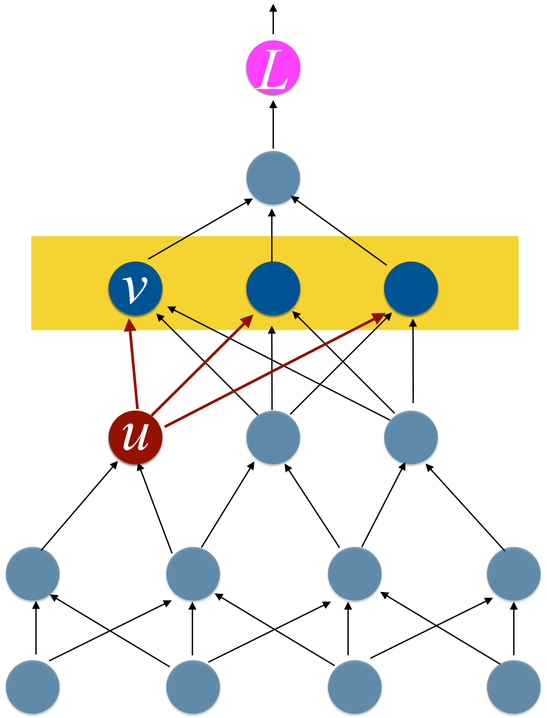


Will compute $\partial L/\partial u$ for every node u in the network!

$\partial L/\partial w = \partial L/\partial u \cdot \partial u/\partial w$ where u is the node which uses w

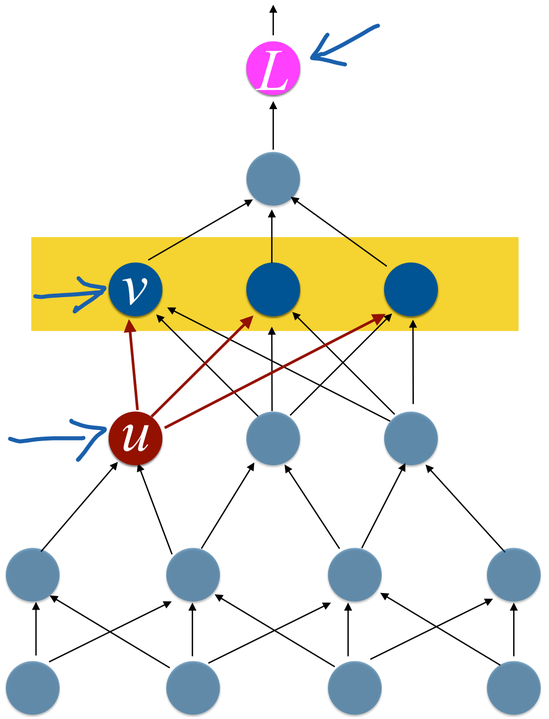
Backpropagation

$$\partial L/\partial u = \Sigma_{v \in \Gamma(u)} \; \partial L/\partial v \cdot \partial v/\partial u$$



Backpropagation

$$\partial L/\partial u = \Sigma_{v \in \Gamma(u)} \; \partial L/\partial v \cdot \partial v/\partial u$$



Forward Pass

First, in a forward pass, compute values of all nodes given an input
(The values of each node will be needed during backprop)

Backpropagation

$$\partial L/\partial u = \Sigma_{v \in \Gamma(u)} \; \partial L/\partial v \cdot \partial v/\partial u$$

$$v = g\left(\sum_i w_i u_i\right)$$

$$\frac{\partial v}{\partial u} = g' \cdot w$$

Backpropagation

Base case: $\partial L/\partial L = 1$

For each u (top to bottom):

For each $v \in \Gamma(u)$:

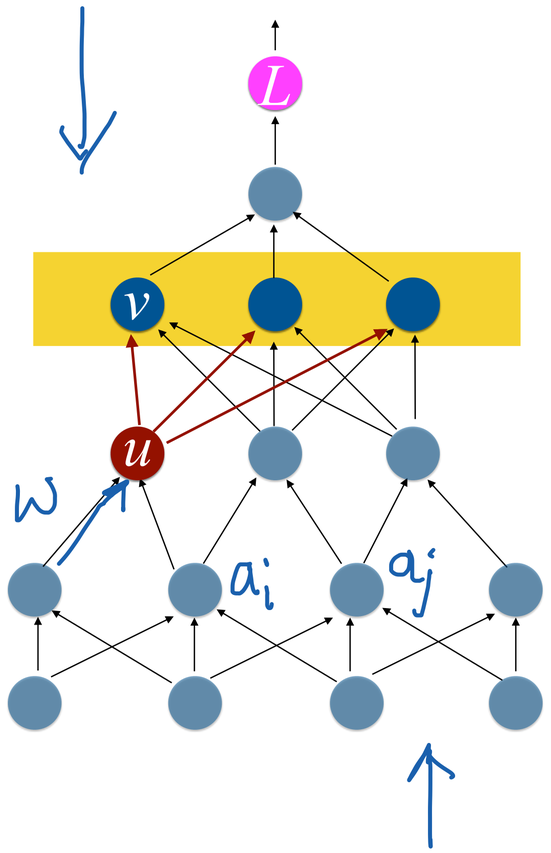
Inductively, have computed $\partial L/\partial v$

Directly compute $\partial v/\partial u$

Compute $\partial L/\partial u$

Compute $\partial L/\partial w$

where $\partial L/\partial w = \partial L/\partial u \cdot \partial u/\partial w$



Forward Pass

First, in a forward pass, compute values of all nodes given an input
(The values of each node will be needed during backprop)

Backpropagation

$$\partial L/\partial u = \Sigma_{v \in \Gamma(u)} \; \partial L/\partial v \cdot \partial v/\partial u$$

Backpropagation

Base case: $\partial L/\partial L = 1$

For each u (top to bottom):

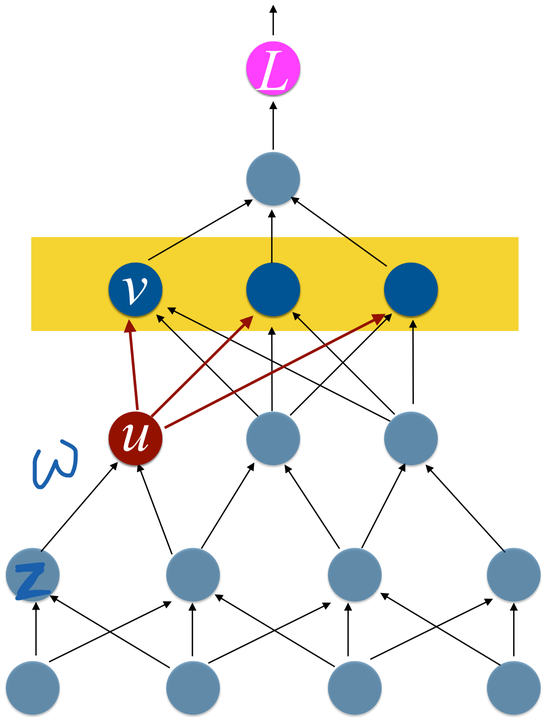
For each $v \in \Gamma(u)$:

Inductively, have computed $\partial L/\partial v$

Directly compute $\partial v/\partial u$

Compute $\partial L/\partial u$

Compute $\partial L/\partial w$
where $\partial L/\partial w = \partial L/\partial u \cdot \partial u/\partial w$



Where values computed in the forward pass may be needed

Forward Pass

First, in a forward pass, compute values of all nodes given an input
(The values of each node will be needed during backprop)

History of Neural Networks in ASR

History of Neural Networks in ASR

- Neural networks for speech recognition were explored as early as 1987

History of Neural Networks in ASR

- Neural networks for speech recognition were explored as early as 1987
- Deep neural networks for speech
  - Beat state-of-the-art on the TIMIT corpus [M09]
  - Significant improvements shown on large-vocabulary systems [D11]
  - Dominant ASR paradigm [H12]

[M09] A. Mohamed, G. Dahl, and G. Hinton, “Deep belief networks for phone recognition,” NIPS Workshop on Deep Learning for Speech Recognition, 2009.

[D11] G. Dahl, D. Yu, L. Deng, and A. Acero, “Context-Dependent Pre-Trained Deep Neural Networks for Large-Vocabulary Speech Recognition,” TASL 20(1

[H12] G. Hinton, et al., “Deep Neural Networks for Acoustic Modeling in Speech Recognition”, IEEE Signal Processing Magazine, 2012.

Feedforward Neural Networks for ASR

- Two main categories of approaches have been explored:

1. Hybrid neural network-HMM systems: Use DNNs to estimate HMM observation probabilities

2. Tandem system: NNs used to generate input features that are fed to an HMM-GMM acoustic model

Feedforward Neural Networks for ASR

- Two main categories of approaches have been explored:

1. Hybrid neural network-HMM systems: Use DNNs to estimate HMM observation probabilities

2. Tandem system: DNNs used to generate input features that are fed to an HMM-GMM acoustic model

Decoding an ASR system

- Recall how we decode the most likely word sequence W for an acoustic sequence O:

$$W^* = \arg\max_W \Pr(O|W) \Pr(W)$$

- The acoustic model $\Pr(O \mid W)$ can be further decomposed as (here, Q represents a triphone state sequence):

$$\begin{aligned}
\Pr(O|W) &= \sum_Q \Pr(O, Q|W) \\
&= \sum_Q \Pr(O|Q, W) \Pr(Q|W) \Pr(W) \\
&\approx \sum_Q \Pr(O|Q) \Pr(Q|W) \Pr(W)
\end{aligned}$$

Hybrid system decoding

Hybrid system decoding

We've seen Pr(O|Q) estimated using a Gaussian Mixture Model.
Let's use a neural network instead to model Pr(O|Q).

Hybrid system decoding

We've seen Pr(O|Q) estimated using a Gaussian Mixture Model.
Let's use a neural network instead to model Pr(O|Q).

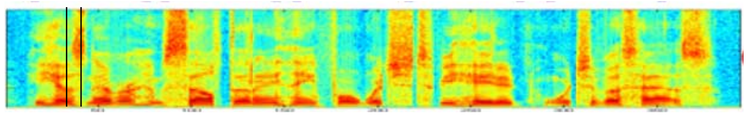
$$\Pr(O|Q) = \prod_t \Pr(o_t|q_t)$$

$$\Pr(o_t|q_t) = \frac{\Pr(q_t|o_t)\Pr(o_t)}{\Pr(q_t)}$$

$$\propto \frac{\Pr(q_t|o_t)}{\Pr(q_t)}$$

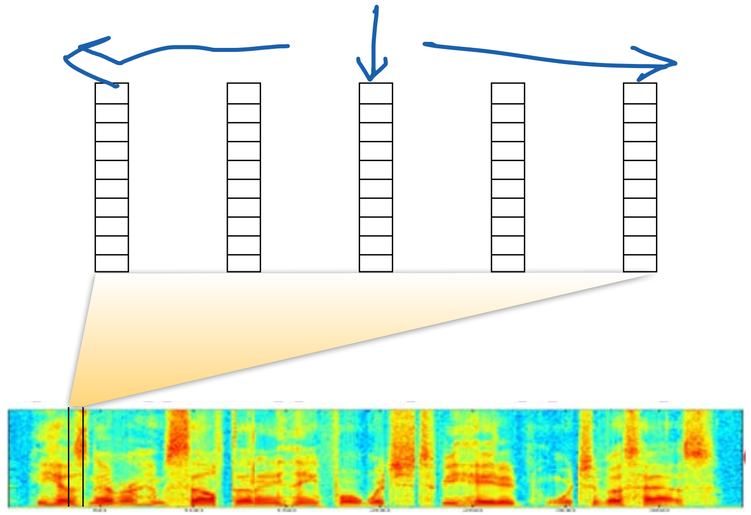
where o_t is the acoustic vector at time t and q_t is a triphone HMM state
Here, Pr(q_t|o_t) are posteriors from a trained neural network.
Pr(o_t|q_t) is then a scaled posterior.

Computing $\Pr(q_t|o_t)$ using a deep NN

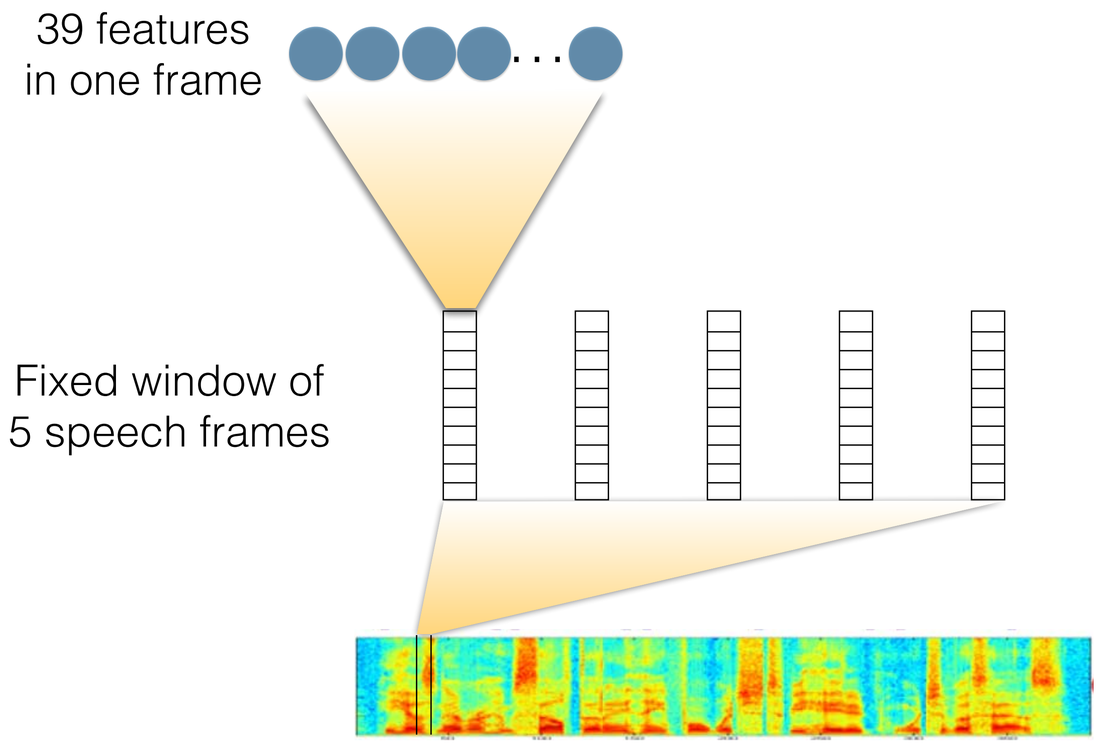


Computing $\Pr(q_t|o_t)$ using a deep NN

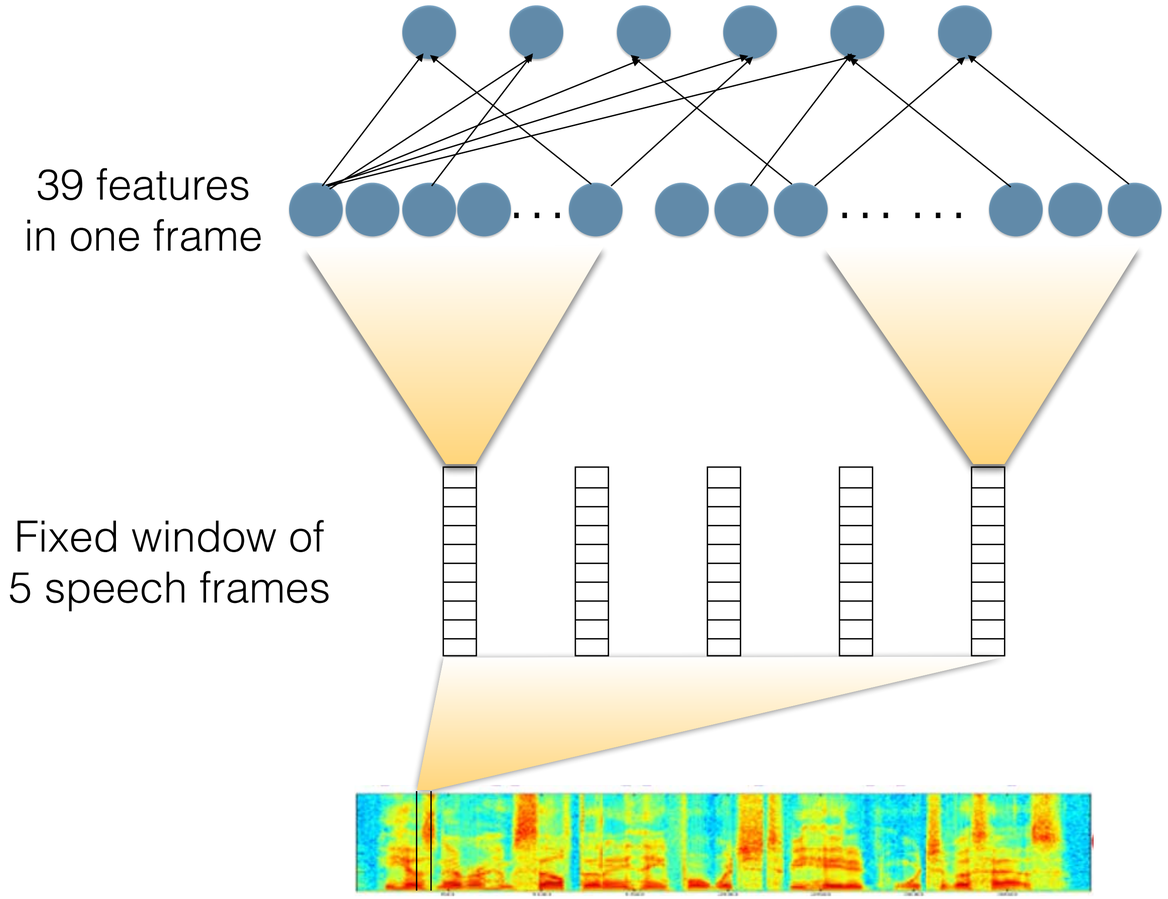
Fixed window of
5 speech frames



Computing $\Pr(q_t|o_t)$ using a deep NN

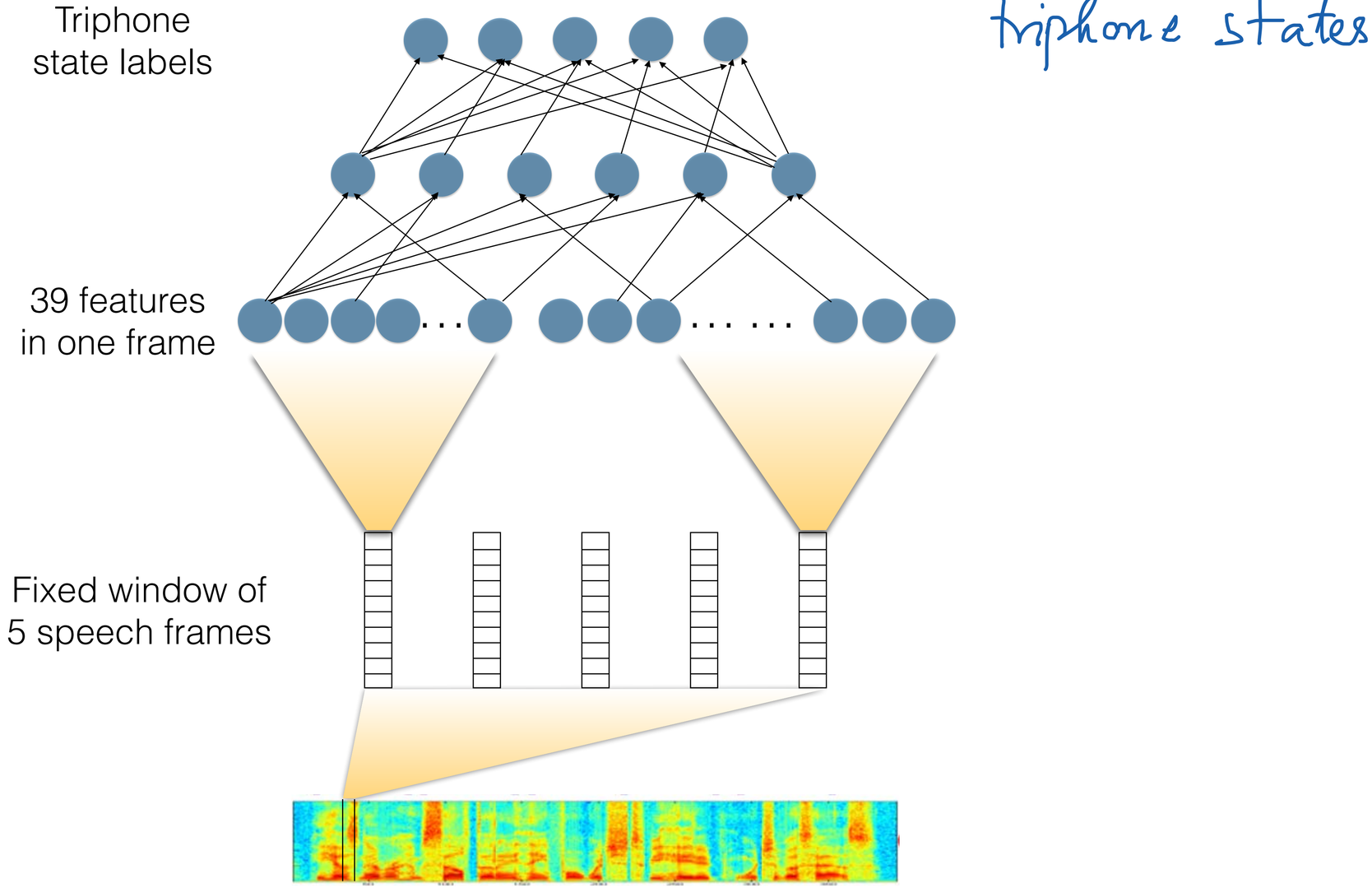


Computing $\Pr(q_t|o_t)$ using a deep NN

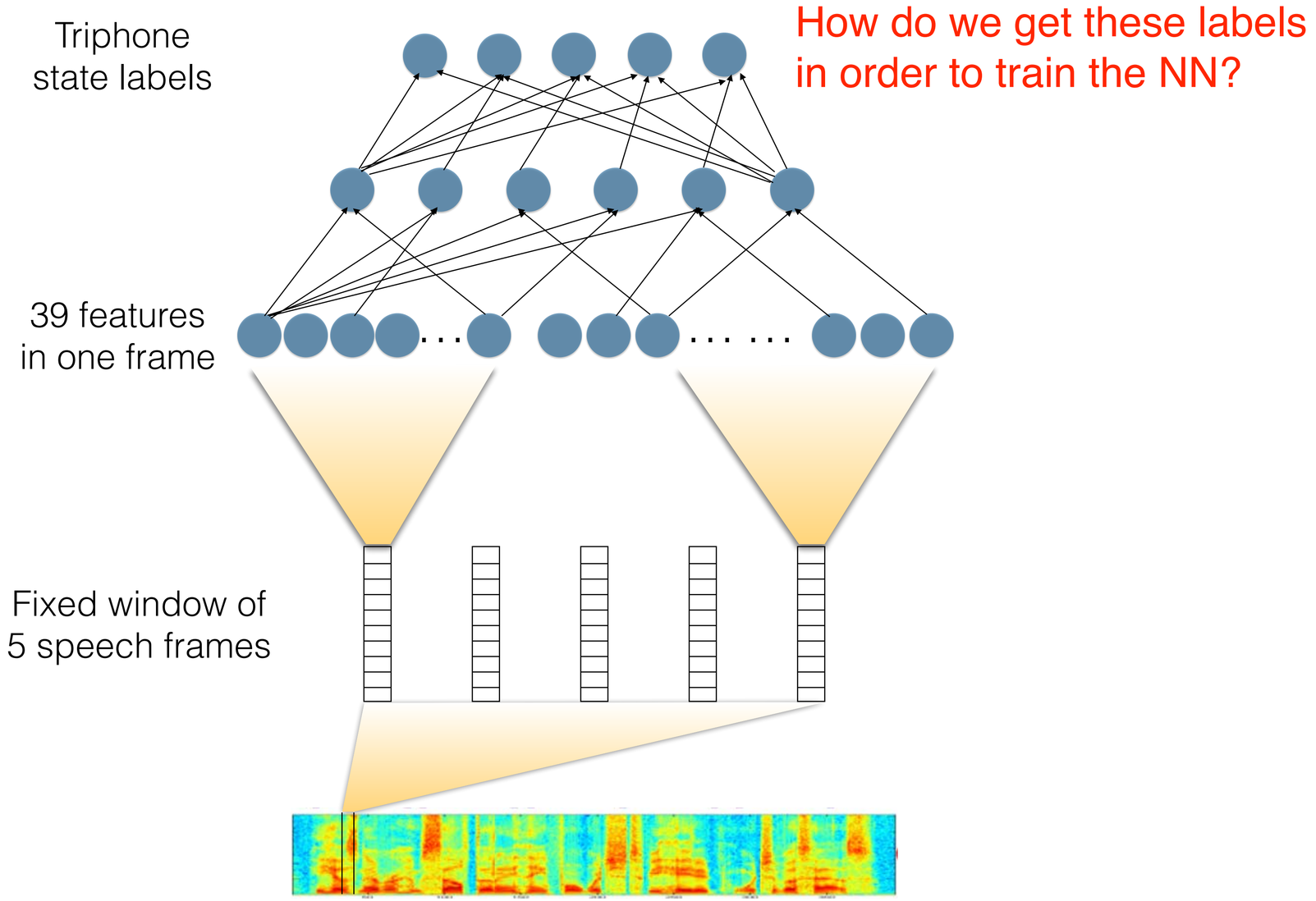


Computing $\Pr(q_t|o_t)$ using a deep NN

Softmax distribution over triphone states



Computing $\Pr(q_t|o_t)$ using a deep NN



Triphone labels

Triphone labels

- Forced alignment: Use current acoustic model to find the most likely sequence of HMM states given a sequence of acoustic vectors. (Algorithm to help compute this?)

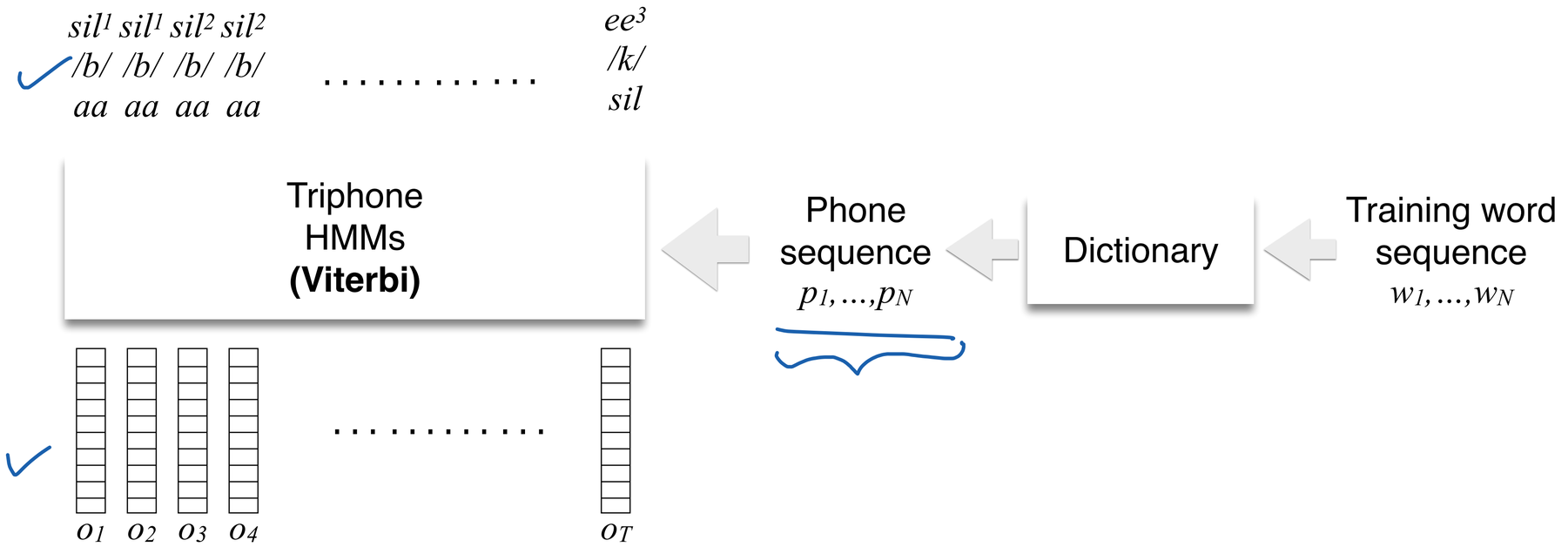
Viterbi

Triphone labels

- Forced alignment: Use current acoustic model to find the most likely sequence of HMM states given a sequence of acoustic vectors. (Algorithm to help compute this?)
- The “Viterbi paths” for the training data, are also referred to as forced alignments

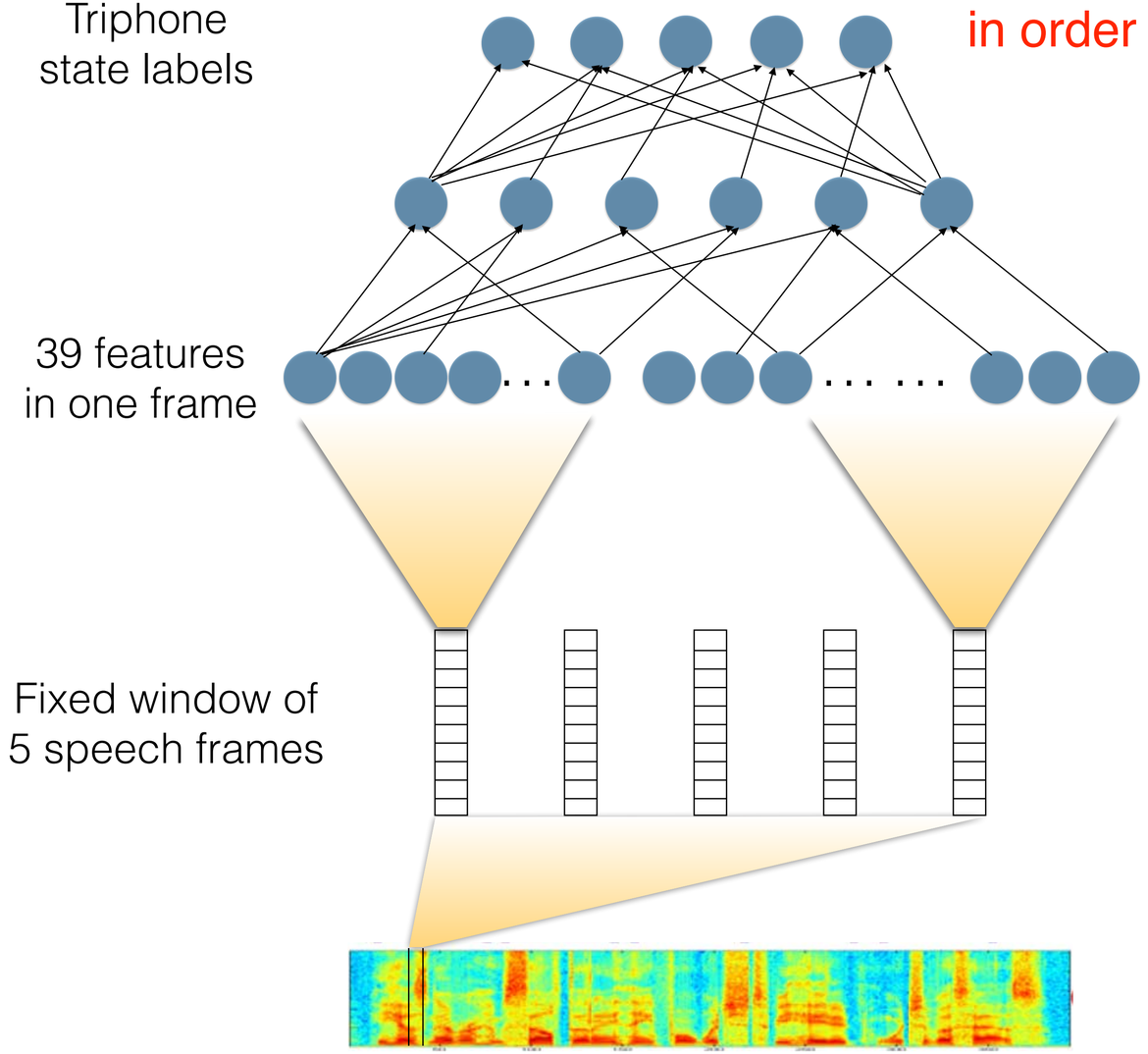
Triphone labels

- Forced alignment: Use current acoustic model to find the most likely sequence of HMM states given a sequence of acoustic vectors. (Algorithm to help compute this?)
- The “Viterbi paths” for the training data, are also referred to as forced alignments



Computing $Pr(q_t|o_t)$ using a deep NN

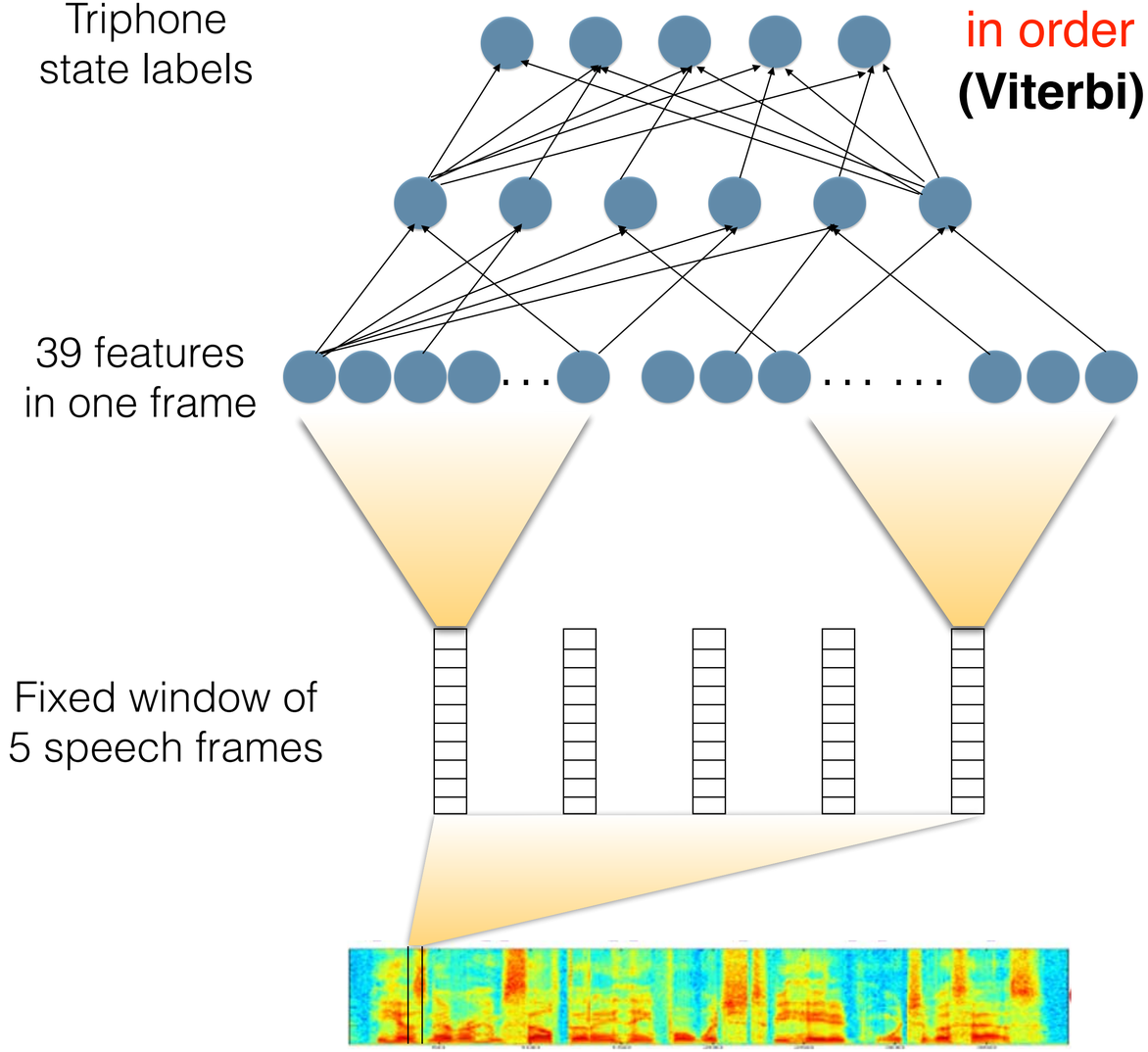
How do we get these labels in order to train the NN?



Computing $Pr(q_t|o_t)$ using a deep NN

How do we get these labels in order to train the NN?

(Viterbi) Forced alignment



Computing priors Pr(q_t)

Computing priors Pr(q_t)

$$\Pr(o_t|q_t) \propto \frac{\Pr(q_t|o_t)}{\Pr(q_t)}$$

- To compute HMM observation probabilities, Pr(o_t|q_t), we need both Pr(q_t|o_t) and Pr(q_t)

Computing priors Pr(q_t)

- To compute HMM observation probabilities, Pr($o_t|q_t$), we need both Pr($q_t|o_t$) and Pr(q_t)

- The posterior probabilities Pr($q_t|o_t$) are computed using a trained neural network

Computing priors Pr(q_t)

- To compute HMM observation probabilities, Pr($o_t|q_t$), we need both Pr($q_t|o_t$) and Pr(q_t)

- The posterior probabilities Pr($q_t|o_t$) are computed using a trained neural network

- Pr(q_t) are relative frequencies of each triphone state as determined by the forced Viterbi alignment of the training data

Hybrid Networks

Hybrid Networks

- The networks are trained with a minimum cross-entropy criterion

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$$

Hybrid Networks

- The networks are trained with a minimum cross-entropy criterion

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$$

Hybrid Networks

- The networks are trained with a minimum cross-entropy criterion

$$L(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i)$$

- Advantages of hybrid systems:

1. Fewer assumptions made about acoustic vectors being uncorrelated: Multiple inputs used from a window of time steps

2. Discriminative objective function used to learn the observation probabilities

hybrid

Summary of DNN-HMM acoustic models

Comparison against HMM-GMM on different tasks

[TABLE 3] A COMPARISON OF THE PERCENTAGE WERs USING DNN-HMMs AND GMM-HMMs ON FIVE DIFFERENT LARGE VOCABULARY TASKS.

TASK	HOURS OF TRAINING DATA	DNN-HMM	GMM-HMM WITH SAME DATA	GMM-HMM WITH MORE DATA
SWITCHBOARD (TEST SET 1)	309	18.5	27.4	18.6 (2,000 H)
SWITCHBOARD (TEST SET 2)	309	16.1	23.6	17.1 (2,000 H)
ENGLISH BROADCAST NEWS	50	17.5	18.8	
BING VOICE SEARCH (SENTENCE ERROR RATES)	24	30.4	36.2	
GOOGLE VOICE INPUT	5,870	12.3		16.0 (>> 5,870 H)
YOUTUBE	1,400	47.6	52.3	

Hybrid DNN-HMM systems consistently outperform GMM-HMM systems (sometimes even when the latter is trained with lots more data)

Table from G. Hinton, et al., “Deep Neural Networks for Acoustic Modeling in Speech Recognition”, IEEE Signal Processing Magazine, 2012.

Neural Networks for ASR

- Two main categories of approaches have been explored:

1. Hybrid neural network-HMM systems: Use DNNs to estimate HMM observation probabilities

2. Tandem system: NNs used to generate input features that are fed to an HMM-GMM acoustic model

Tandem system

Tandem system

- First, train a DNN to estimate the posterior probabilities of each subword unit (monophone, triphone state, etc.)

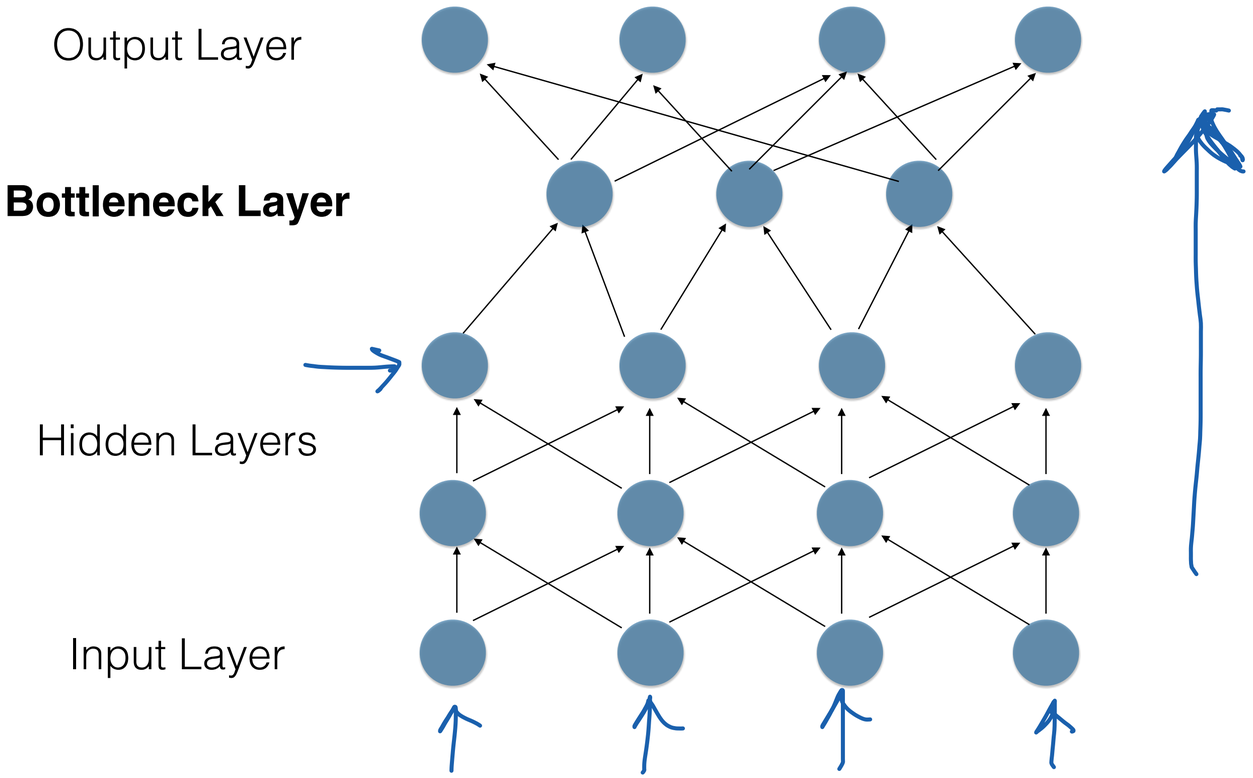
Tandem system

- First, train a DNN to estimate the posterior probabilities of each subword unit (monophone, triphone state, etc.)
- In a hybrid system, these posteriors (after scaling) would be used as observation probabilities for the HMM acoustic models

Tandem system

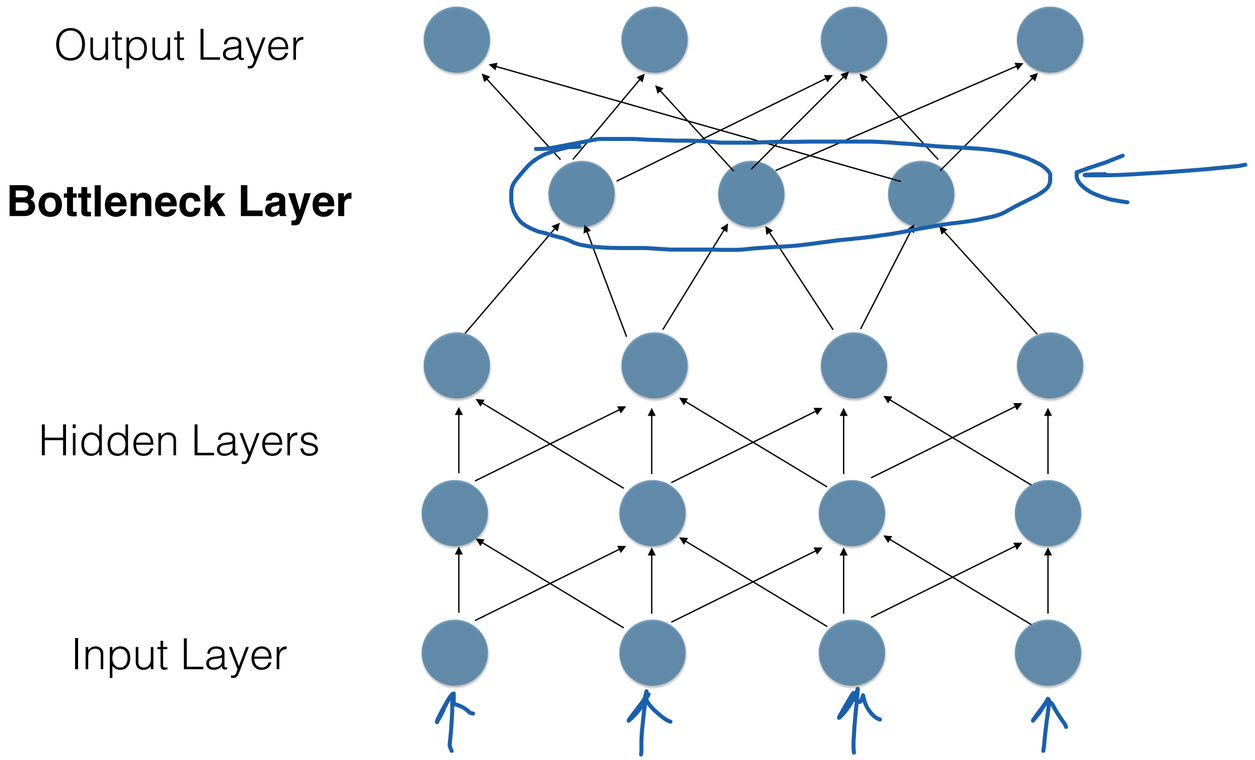
- First, train a DNN to estimate the posterior probabilities of each subword unit (monophone, triphone state, etc.)
- In a hybrid system, these posteriors (after scaling) would be used as observation probabilities for the HMM acoustic models
- In the tandem system, the DNN outputs are used as “feature” inputs to HMM-GMM models

Bottleneck Features



Bottleneck Features

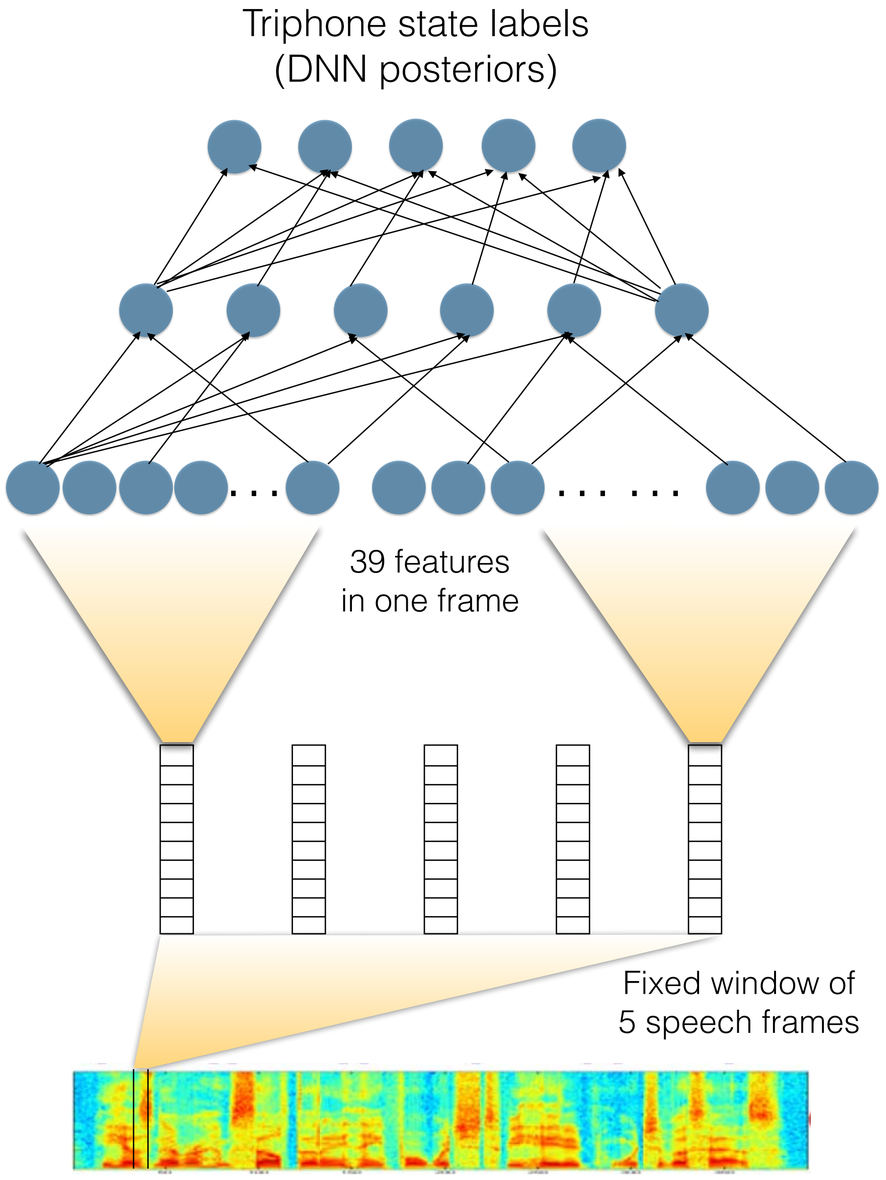
$P(q_t | o_t)$



Use a low-dimensional *bottleneck* layer representation to extract features

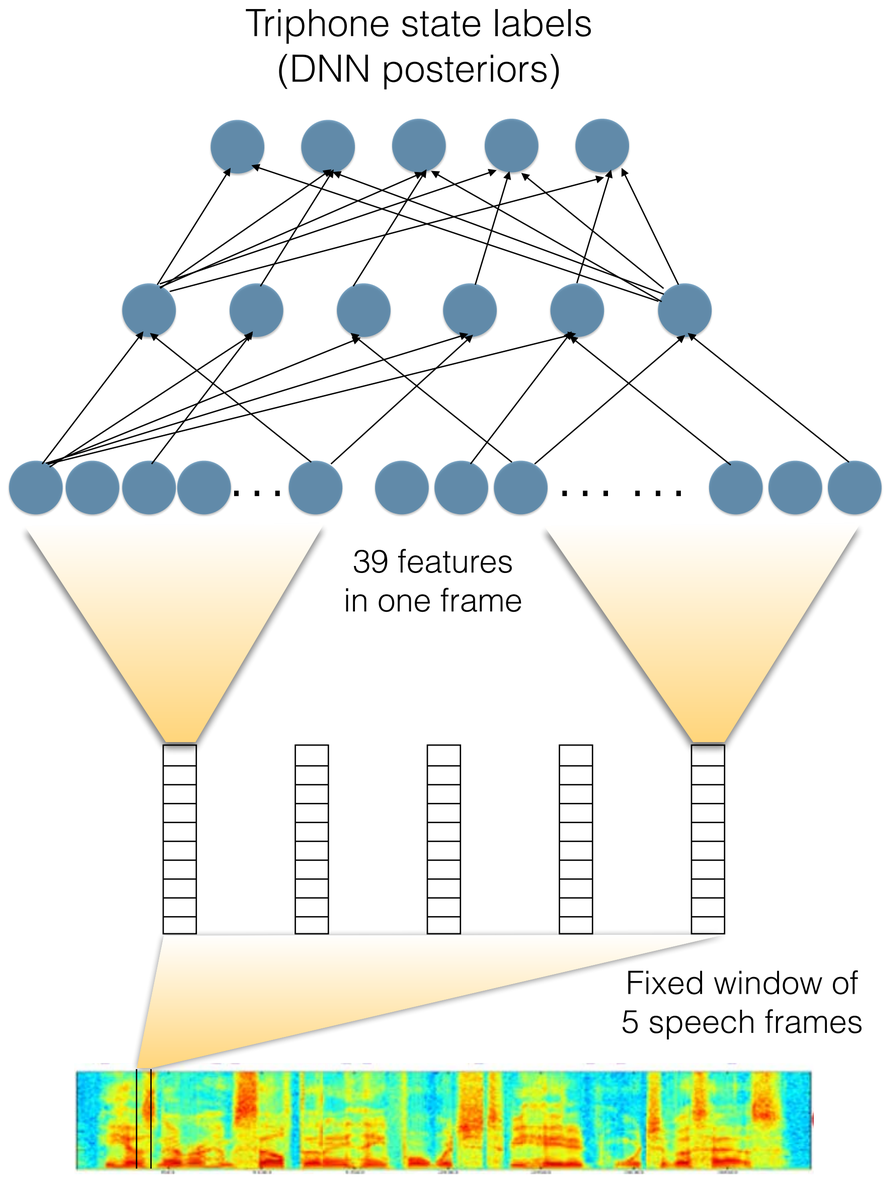
These bottleneck features are in turn used as inputs to HMM-GMM models

Recap: Hybrid DNN-HMM Systems



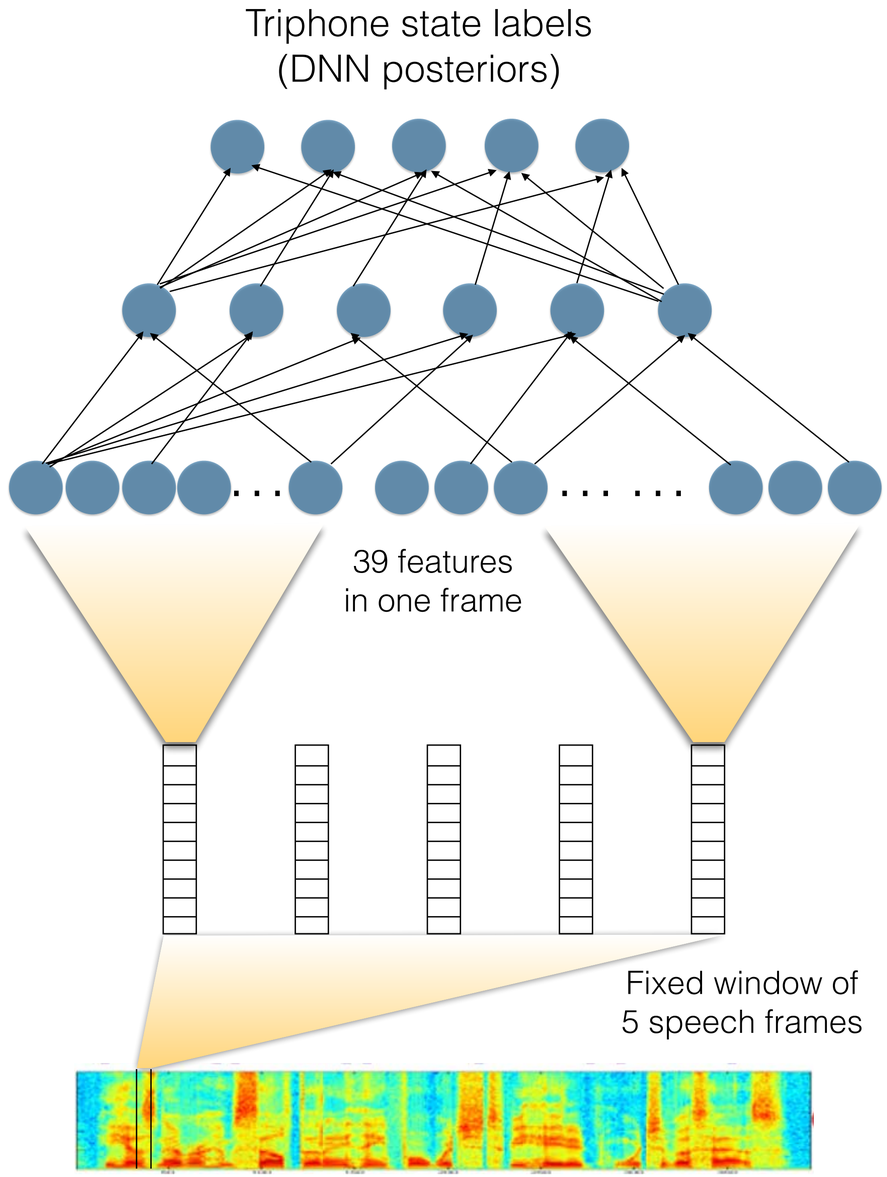
Recap: Hybrid DNN-HMM Systems

- Instead of GMMs, use scaled DNN posteriors as the HMM observation probabilities



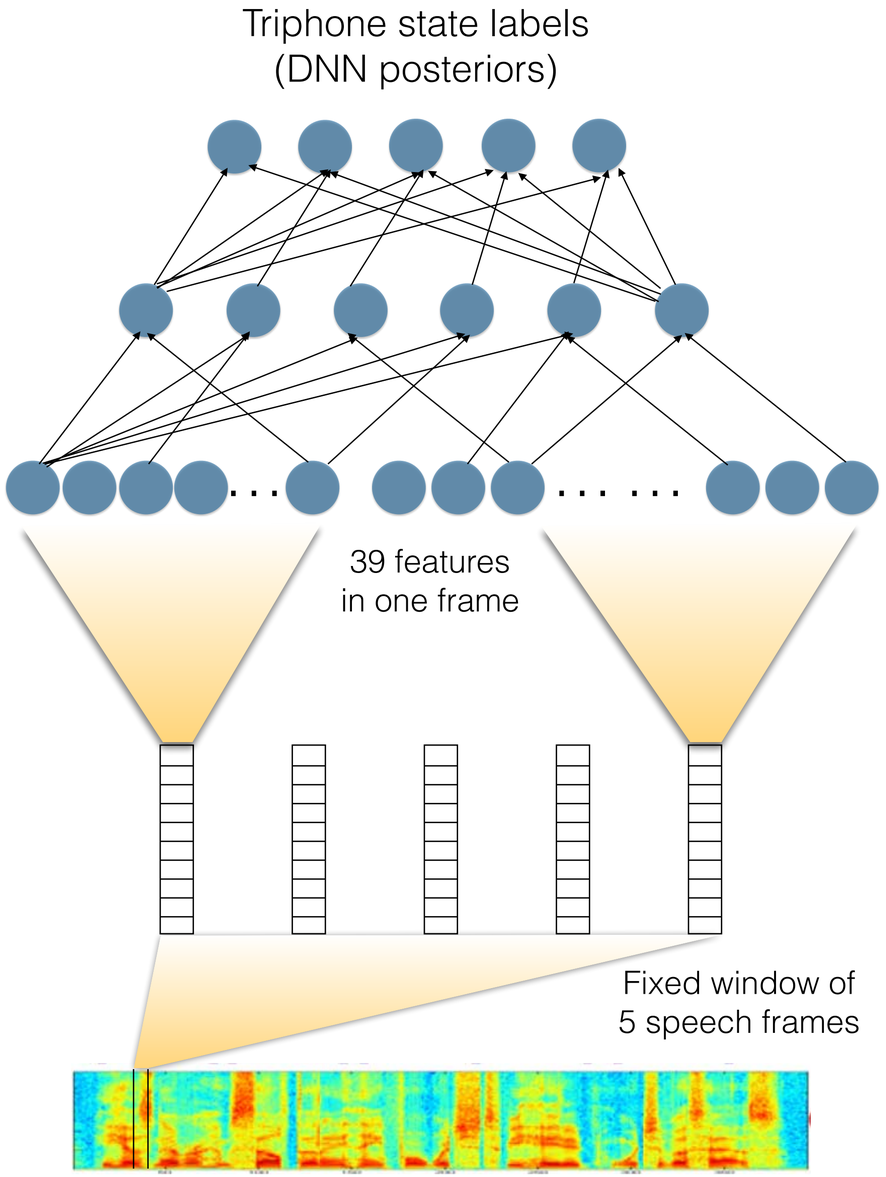
Recap: Hybrid DNN-HMM Systems

- Instead of GMMs, use scaled DNN posteriors as the HMM observation probabilities
- DNN trained using triphone labels derived from a forced alignment “Viterbi” step.

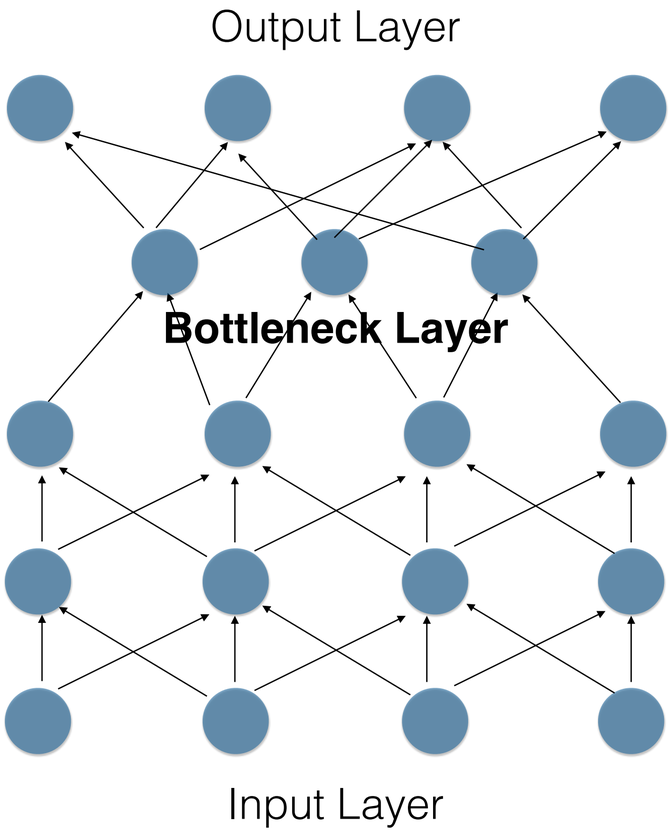


Recap: Hybrid DNN-HMM Systems

- Instead of GMMs, use scaled DNN posteriors as the HMM observation probabilities
- DNN trained using triphone labels derived from a forced alignment "Viterbi" step.
  - *Forced alignment:* Given a training utterance $\{O, W\}$, find the most likely sequence of states (and hence triphone state labels) using a set of trained triphone HMM models, M. Here M is constrained by the triphones in W.

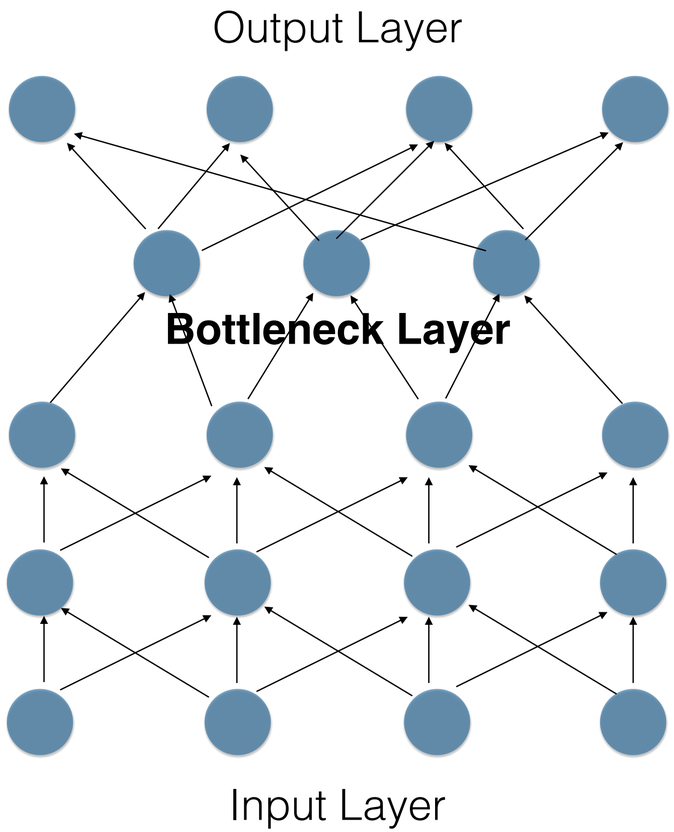


Recap: Tandem DNN-HMM Systems



Recap: Tandem DNN-HMM Systems

- Neural networks are used as “feature extractors” to train HMM-GMM models
- Use a low-dimensional *bottleneck* layer representation to extract features from the bottleneck layer
- These bottleneck features are subsequently fed to GMM-HMMs as input



Feedforward DNNs we've seen so far…

Feedforward DNNs we've seen so far…

- Assume independence among the training instances (modulo the context window of frames)
  - Independent decision made about classifying each individual speech frame
  - Network state is completely reset after each speech frame is processed

Feedforward DNNs we've seen so far…

- Assume independence among the training instances (modulo the context window of frames)
  - Independent decision made about classifying each individual speech frame
  - Network state is completely reset after each speech frame is processed
- This independence assumption fails for data like speech which has temporal and sequential structure

Feedforward DNNs we've seen so far…

- Assume independence among the training instances (modulo the context window of frames)
  - Independent decision made about classifying each individual speech frame
  - Network state is completely reset after each speech frame is processed
- This independence assumption fails for data like speech which has temporal and sequential structure
- Two model architectures that capture longer ranges of acoustic context:
  1. **Time delay neural networks (TDNNs)**
  2. Recurrent neural networks (RNNs)