CS772: Deep Learning for Natural Language Processing (DL-NLP)

Backpropagation, Small and Large LMs

Pushpak Bhattacharyya

Computer Science and Engineering Department

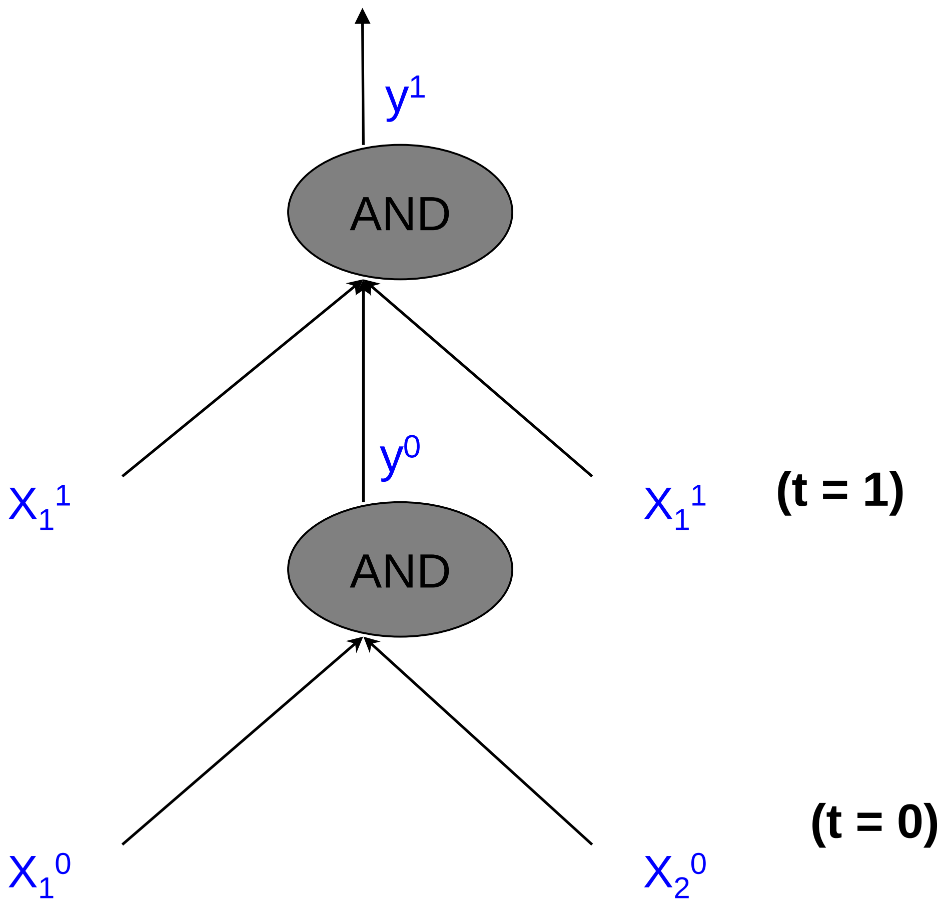
IIT Bombay

Week 4 of 22jan24

1-slide recap

- Recurrent Perceptron- concept of state, feedback
- Responsible AI- Toxicity, Hallucination, Deepfake
- Sigmoid and Softmax
- Derivative of sigmoid and softmax
- Weight change with gradient descent- sigmoid neuron and softmax neural net (without hidden layer)

Recurrent Perceptron



Perspective- Do Small Language Models have a place

Emergence of LLMs

- Very simple idea of training neural models with 1 (openAI) to 4 (Amazon's Olympus, $4billion, about 33K Cr INR) trillion parameters to predict **the next word or masked words**
- Three dimensions of **Language-Domain-Task**: e.g. Konakani-Agro-QA
- YouTube developing Summarization and CAI tools using GenAI
- Impossible for single organizations and small consortia to match up to such efforts
- However, **Prompt Engineering, Adapters and Fine Tuning** piggybacking on big industry created pretrained models is full of potential
- Looming also in the horizon- bias, hallucination, hate text/speech, fake news

Well known LMs: size

- **GPT-3 (Generative Pre-trained Transformer 3)** by OpenAI: 175 billion parameters; **GPT-4:** 1 trillion
- **BERT** by Google: BERT-base: 110 million; BERT-large: 340 million
- **XLNet** by Google AI and Carnegie Mellon University: Size: 340 million
- **T5 (Text-To-Text Transfer Transformer)** by Google AI: Size: 11 billion
- **LLAMA-2** by Meta**:** 34 billion
- **Olympus** by Amazon: 4 trillion
- **Mistral 7B** by Mistral.AI: 7.3 billion
- **OpenHathi** by Sarvam: 7 Billion
- **PaLM** by Google: 540 billion

Compare with Human Brain

- The human brain has some 8.6 x **10^{10} (eighty six billion) neurons**. Each neuron has on average 7,000 synaptic connections to other neurons.
- Hence total no. of “parameters”= average **600 trillion**

Well known LLMs: Data Used

- **GPT-3**: data amount not publicly known; probably hundreds of terrabytes
- BERT-base: English Wikipedia (about 2.5 billion words) and the BookCorpus dataset (11,038 books); BERT-large: even more
- **XLNet**: as in BERT base
- **T5**: mixture of books, articles, and websites

Well known LLMs: Pre-training resource requirement

- **GPT-3**: Trained using thousands of powerful GPUs over the span of weeks
- BERT-large: GPU and (possibly) TPU clusters; weeks of pretraining time
- **XLNet**: as above
- **T5**: as above

A few SLMs

Credit: https://analyticsindiamag.com/9-best-small-language-models-released-in-2023/

A few “Small” Language Models (SLMs): 1/3

- **Llama-2 7B:** many current open source models built on top of Llama family of models; multilingual; various NLP tasks
- **Phi2 and Orca**: 13 B; Microsoft; Reasoning and explainability capability
- **Stable Beluga 7B**: leverages Meta’s Llama; multilingual: text generation, translation, question answering, and code completion.

SLMs: 2/3

- **X Gen**: 7B; by Salesforce; multilingual; creative writing and content creation and language learning
- **Alibaba’s Qwen**: Qwen-1.8B, Qwen-7B, Qwen-14B, and Qwen-72B; standard NLP tasks, and audio processing
- **Alpaca 7B**: leverages Meta; renowned for its remarkable compactness and cost-effectiveness, requiring less than $600 in building costs

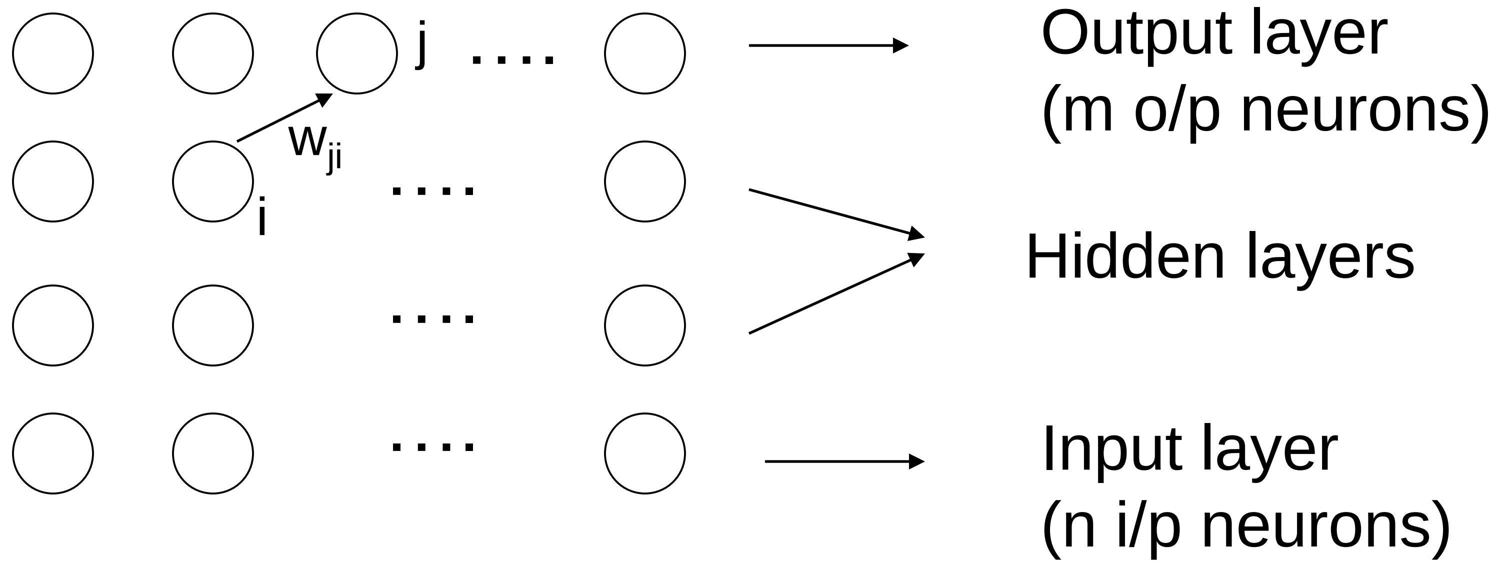
SLMs: 3/3

- **MPT**: 7 B; by Mosaic ML; creative writing, content creation, education, and accessibility tools
- **Falcon 7B**: by Technology Innovation Institute (TII); Tailored for chatting and question answering; tops the Huggingface leaderboard for the longest time
- **Zephyr**: 7B; by Huggingface; specialized for chatbots

Feedforward N/W and Backpropagation

With total sum square loss (TSS)

Backpropagation algorithm



- Fully connected feed forward network
- Pure FF network (no jumping of connections over layers)

Gradient Descent Equations

$$\Delta w_{ji} = -\eta \frac{\delta E}{\delta w_{ji}} (\eta = \text{learning rate}, 0 \le \eta \le 1)$$

$$\frac{\delta E}{\delta w_{ji}} = \frac{\delta E}{\delta net_j} \times \frac{\delta net_j}{\delta w_{ji}} (net_j = \text{input at the j}^{th} \text{ neuron})$$

$$\boxed{\frac{\delta E}{\delta net_j} = -\delta j}$$

$$\Delta w_{ji} = \eta \delta j \frac{\delta net_j}{\delta w_{ji}} = \eta \delta j o_i$$

A quantity of great importance

Backpropagation – for outermost layer

$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j} (net_j = \text{input at the } \mathrm{j}^{th} \text{ layer})$$

$$E = \frac{1}{2}\sum_{i=1}^{N}(t_j - o_j)^2$$

$$\text{Hence, } \delta j = -(-(t_j - o_j)o_j(1-o_j))$$

$$\Delta w_{ji} = \eta(t_j - o_j)o_j(1-o_j)o_i$$

Observations from Δw_{ji}

$$\Delta w_{ji} = \eta(t_j - o_j)o_j(1 - o_j)o_i$$

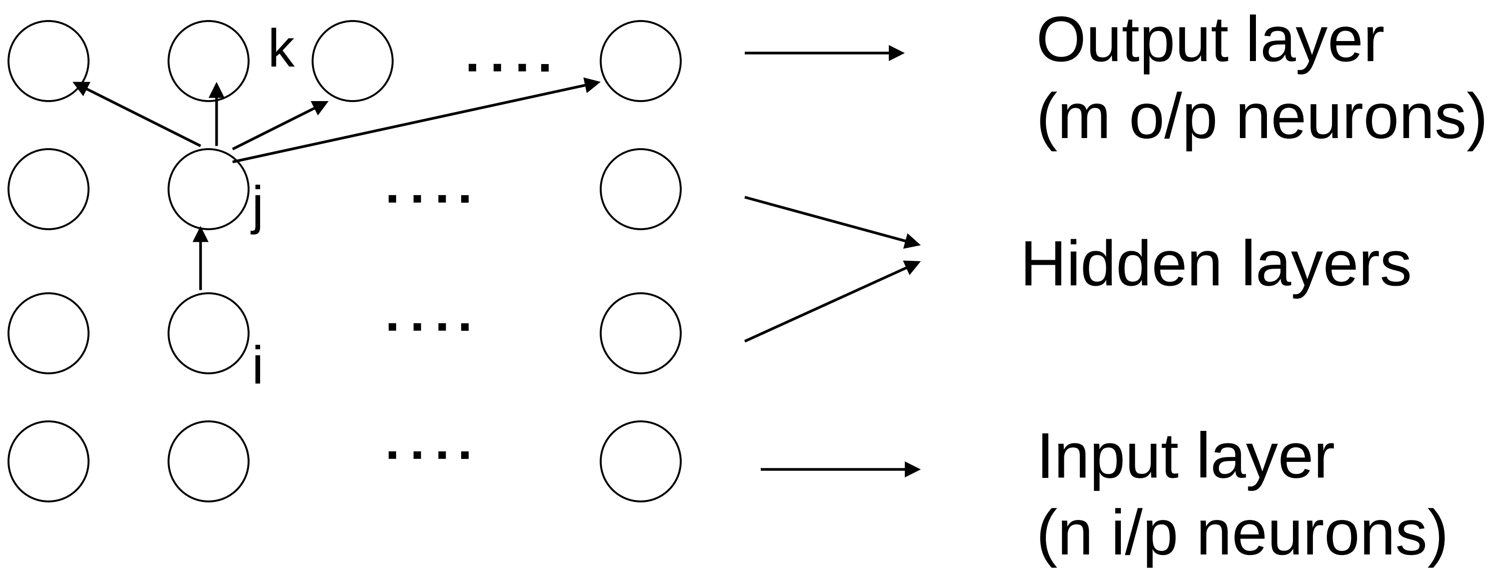
$\Delta w_{ji} \rightarrow 0$ if,

$$\left.\begin{array}{ll} 1.\ o_j \rightarrow t_j & \text{and/or} \end{array}\right.$$

$$\left.\begin{array}{ll} 2.\ o_j \rightarrow 1 & \text{and/or} \\ 3.\ o_j \rightarrow 0 & \text{and/or} \end{array}\right\} \text{Saturation behaviour}$$

$$\left.\begin{array}{ll} 4.\ o_i \rightarrow 0 & \end{array}\right\} \text{Credit/Blame assignment}$$

Backpropagation for hidden layers



δ_k is propagated backwards to find value of δ_j

Backpropagation – for hidden layers

$$\Delta w_{ji} = \eta \delta j o_i$$

$$\delta j = -\frac{\delta E}{\delta net_j} = -\frac{\delta E}{\delta o_j} \times \frac{\delta o_j}{\delta net_j}$$

$$= -\frac{\delta E}{\delta o_j} \times o_j(1-o_j)$$

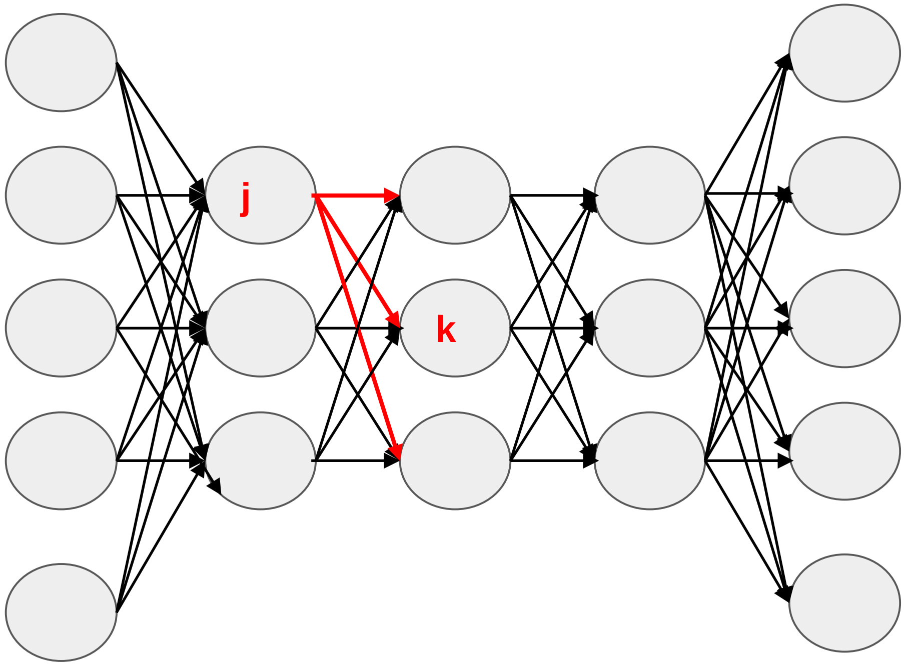
This recursion can give rise to vanishing and exploding Gradient problem

$$= -\sum_{k \in \text{next layer}} \left(\frac{\delta E}{\delta net_k} \times \frac{\delta net_k}{\delta o_j}\right) \times o_j(1-o_j)$$

$$\text{Hence}, \delta_j = -\sum_{k \in \text{next layer}} (-\delta_k \times w_{kj}) \times o_j(1-o_j)$$

$$= \sum_{k \in \text{next layer}} (w_{kj}\delta_k) o_j(1-o_j)$$

Back-propagation- for hidden layers: Impact on net input on a neuron



- O_j affects the net input coming to all the neurons in next layer

General Backpropagation Rule

- General weight updating rule:

$$\Delta w_{ji} = \eta \delta j o_i$$

- Where

$$\delta_j = (t_j - o_j) o_j (1 - o_j) \quad \text{for outermost layer}$$

$$= \sum_{k \in \text{next layer}} (w_{kj} \delta_k) o_j (1 - o_j) \quad \text{for hidden layers}$$

An application in Medical Domain

Expert System for Skin Diseases Diagnosis

- Bumpiness and scaliness of skin
- Mostly for symptom gathering and for developing diagnosis skills
- Not replacing doctor’s diagnosis

Architecture of the FF NN

- 96-20-10
- 96 input neurons, 20 hidden layer neurons, 10 output neurons
- Inputs: skin disease symptoms and their parameters
  - *Location, distribution, shape, arrangement, pattern, number of lesions, presence of an active norder, amount of scale, elevation of papuls, color, altered pigmentation, itching, pustules, lymphadenopathy, palmer thickening, results of microscopic examination, presence of herald pathc, result of dermatology test called KOH*

Output

- 10 neurons indicative of the diseases:
  - *psoriasis, pityriasis rubra pilaris, lichen planus, pityriasis rosea, tinea versicolor, dermatophytosis, cutaneous T-cell lymphoma, secondery syphilis, chronic contact dermatitis, soberrheic dermatitis*

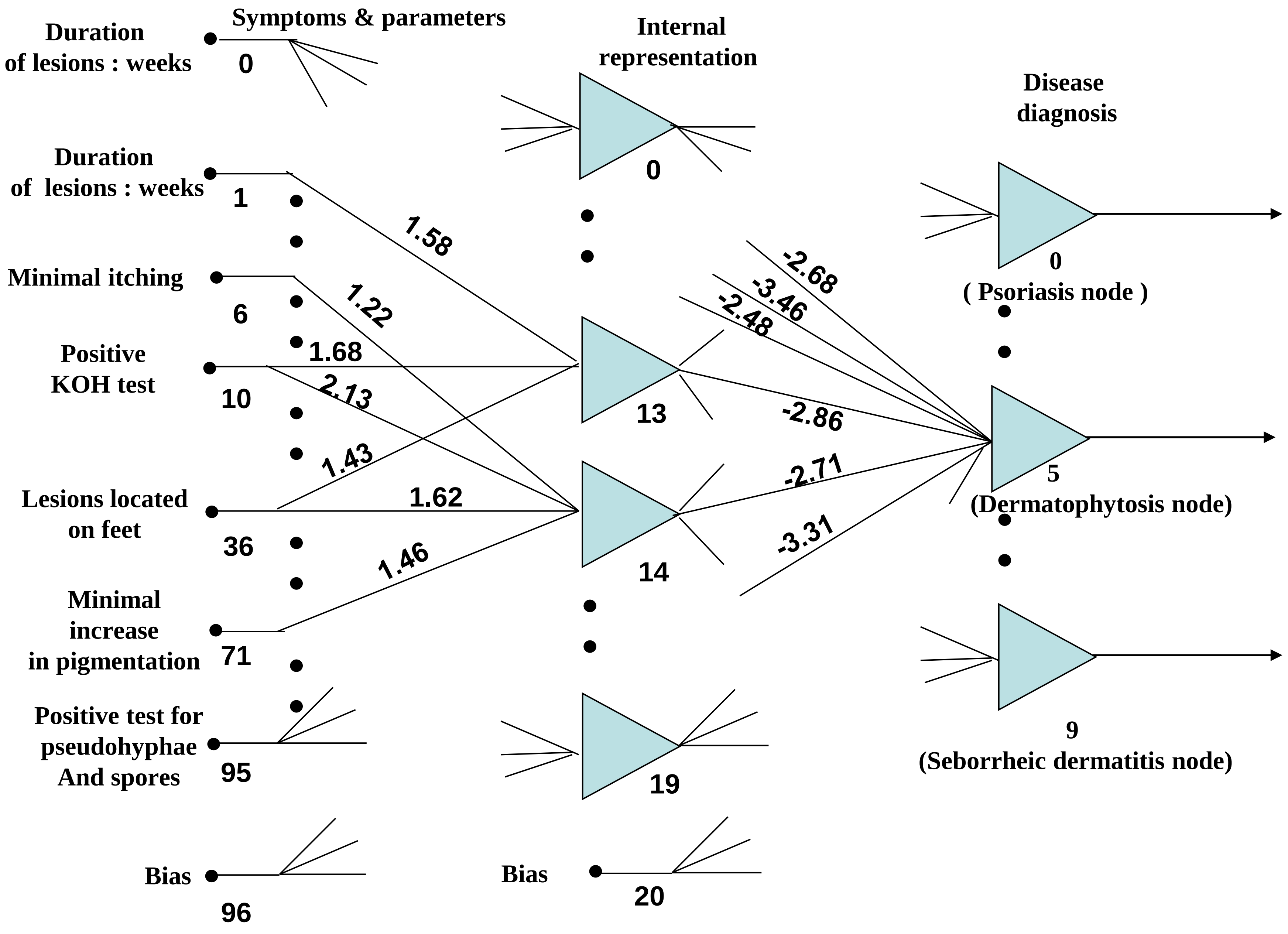


Figure : Explanation of dermatophytosis diagnosis using the DESKNET expert system.

Training data

- Input specs of 10 model diseases from 250 patients
- 0.5 is some specific symptom value is not known
- Trained using standard error backpropagation algorithm

Testing

- Previously unused symptom and disease data of 99 patients
- Result:
- Correct diagnosis achieved for 70% of papulosquamous group skin diseases
- Success rate above 80% for the remaining diseases except for psoriasis
- psoriasis diagnosed correctly only in 30% of the cases
- Psoriasis resembles other diseases within the papulosquamous group of diseases, and is somewhat difficult even for specialists to recognise.

Explanation capability

- Rule based systems reveal the explicit path of reasoning through the textual statements
- Connectionist expert systems reach conclusions through complex, non linear and simultaneous interaction of many units
- Analysing the effect of a single input or a single group of inputs would be difficult and would yield incorrect results

Explanation contd.

- The hidden layer re-represents the data
- Outputs of hidden neurons are neither symtoms nor decisions

Discussion

- Symptoms and parameters contributing to the diagnosis found from the n/w
- Standard deviation, mean and other tests of significance used to arrive at the importance of contributing parameters
- The n/w acts as apprentice to the expert

First Major Application of BP in NLP: Word Vectors

Deriving the word vector: setting

$$W^s : w_0^s, w_1^s, w_2^s, \ldots w_i^s, \ldots w_m^s$$

W^S: word sequence in the *s*th Sentence

$$V_{w_i} : [v_0^i, v_1^i, v_2^i, \ldots v_k^i, \ldots v_d^i]$$

V_{wi}: word vector of w_i

$$J = P(w_j \mid w_i)$$

J to be maximized

$$P(w_j \mid w_i) = \frac{e^{V_{w_i} . V_{w_j}}}{\sum_{j'=1}^{|V|} e^{V_{w_i} . V_{w_{j'}}}}$$

Loss L to be minimized

$$LL = -V_{w_i} . V_{w_j} + \ln\left(\sum_{j'=1}^{|V|} e^{V_{w_i} . V_{w_{j'}}} \right)$$

Work with Log Loss

Deriving the word vector: Optimization (1/2)

$$V_{w_i} : [v_0^i, v_1^i, v_2^i, \ldots v_k^i, \ldots v_d^i] = [u_0, u_1, u_2, \ldots u_k, \ldots u_d]$$

$$V_{w_j} : [v_0^j, v_1^j, v_2^j, \ldots v_k^j, \ldots v_d^j] = [v_0, v_1, v_2, \ldots v_k, \ldots v_d]$$

$$V_{w_{j'}} : [v'_0, v'_1, v'_2, \ldots v'_k, \ldots v'_d]$$

$$V_{w_i} . V_{w_j} = \sum_{k=0}^{d} u_k v_k$$

$$\frac{\partial LL}{\partial u_k} = -v_k + \frac{\frac{\partial}{\partial u_k}\left(\sum_{j'=1}^{|V|} e^{\sum_{k=0}^{d} u_k v_k^{'}} \right)}{\sum_{j'=1}^{|V|} e^{\sum_{k=0}^{d} u_k v_k^{'}}}$$

Deriving the word vector: Optimization

$$= -v_k + \frac{\sum_{j'=1}^{|V|} \frac{\partial}{\partial u_k}\left(e^{\sum_{k=0}^{d} u_k v_k^{'}} \right)}{\sum_{j'=1}^{|V|} e^{\sum_{k=0}^{d} u_k v_k^{'}}} = -v_k + \frac{\sum_{j'=1}^{|V|} e^{\sum_{k=0}^{d} u_k v_k^{'}} \frac{\partial}{\partial u_k}\left(\sum_{k=0}^{d} u_k v_k^{'} \right)}{\sum_{j'=1}^{|V|} e^{\sum_{k=0}^{d} u_k v_k^{'}}}$$

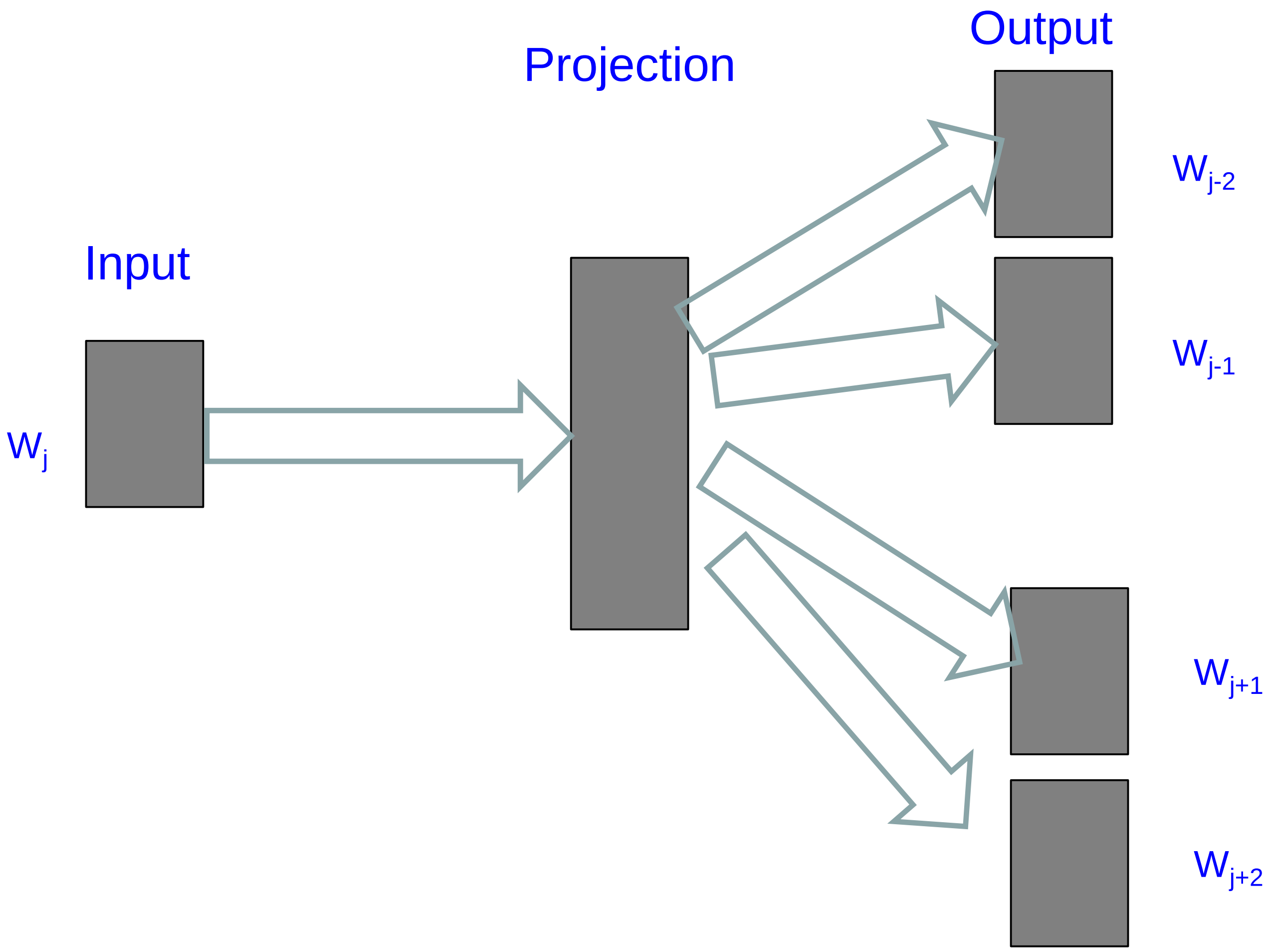
$$= -v_k + \frac{\sum_{j'=1}^{|V|} e^{\sum_{k=0}^{d} u_k v_k^{'}} v_k^{'}}{\sum_{j'=1}^{|V|} e^{\sum_{k=0}^{d} u_k v_k^{'}}} = -v_k + \sum_{j'=1}^{|V|} P(w_{j'} \mid w_i).v_{k'} = -v_k + E(v_{k'})$$

Deriving the word vector, Gradient Descent: Δu_k

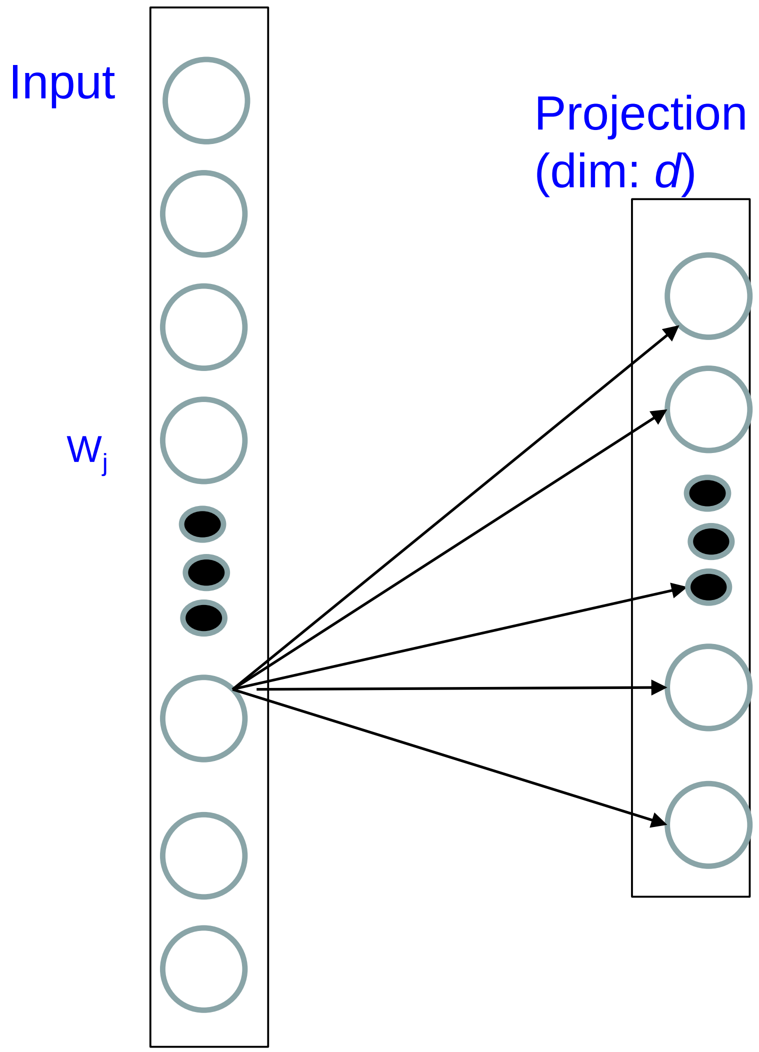
$$\Delta u_k = -\eta \frac{\partial LL}{\partial u_k} = \eta [v_k - E(v_{k'})]$$

This insists that u and v be identical or close which is unrealistic; we need learnable parameters with many degrees of freedom.

Modelling $p(w_{t+j}|w_t)$

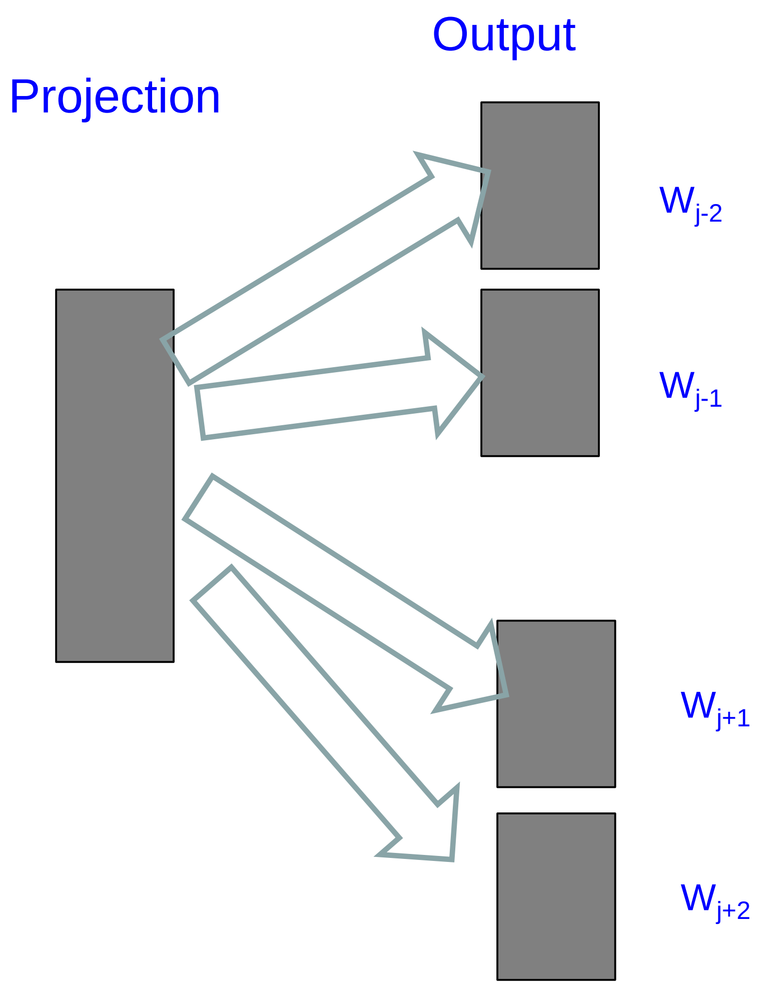


Input to Projection (shown for one neuron only)



- From each input neuron, a weight vector of dim *d*
- Input vector is of dim *V*, where V is the vocab size
- Input to projection we have a weight matrix *W* which is *V X d*
- Each row gives the weight vector of dim *d* REPRESENTING that word
- E.g., rows for 'dog', 'cat, 'lamp', 'table' etc.

Projection to output



- From the whole projection layer a weight vector of dim *d* to each neuron in each compartment, where the compartment represents a context word
- Each fat arrow is a *d X V* matrix

Capturing word association

Basic concept: Co-occurrence Matrix

Corpora: I enjoy cricket. I like music. I like deep learning

	I	**enjoy**	**cricket**	**like**	**music**	**deep**	**learning**
I	-	1	1	2	1	1	1
enjoy	1	-	1	0	0	0	0
cricket	1	1	-	0	0	0	0
like	2	0	0	-	1	1	1
music	1	0	0	1	-	0	0
deep	1	0	0	1	0	-	1
learning	1	0	0	1	0	1	-

Co-occurence Matrix

Fundamental to NLP

Also called **Lexical Semantic Association (LSA)**

Very sparse, many 0s in each row

Apply Principal Component Analysis (PCA) or Singular Value Decomposition (SVD)

Do Dimensionality Reduction; merge columns with high internal affinity (e.g., *cricket* and *bat*)

Compression achieves better semantics capture