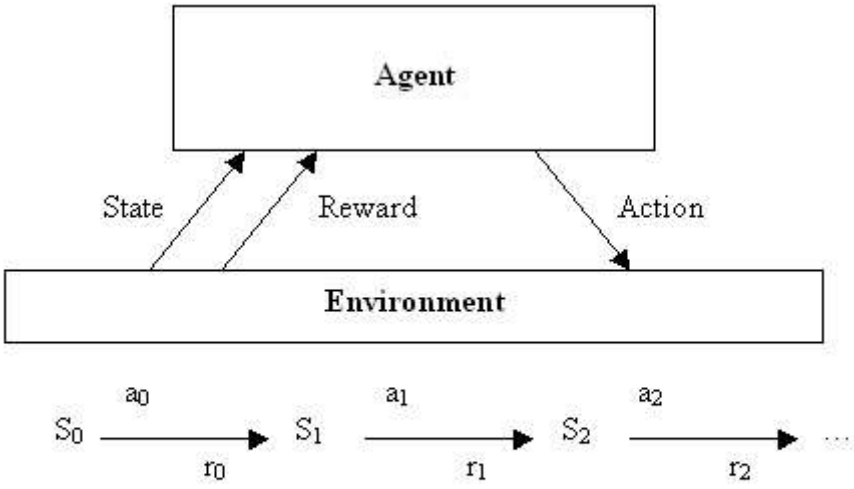
A 'Brief' History of Neural Nets and Deep Learning, Part 3

About neural nets' flaws in the 90s, and how a new AI Winter dawned | December 24, 2015

This is the third part of 'A Brief History of Neural Nets and Deep Learning'. Parts 1 and 2 are here and here, and part 4 is here. In this part, we will continue to see the swift pace of research in the 90s, and see why neural nets ultimately lost favor much as they did in the late 60s.

Neural Nets Make Decisions

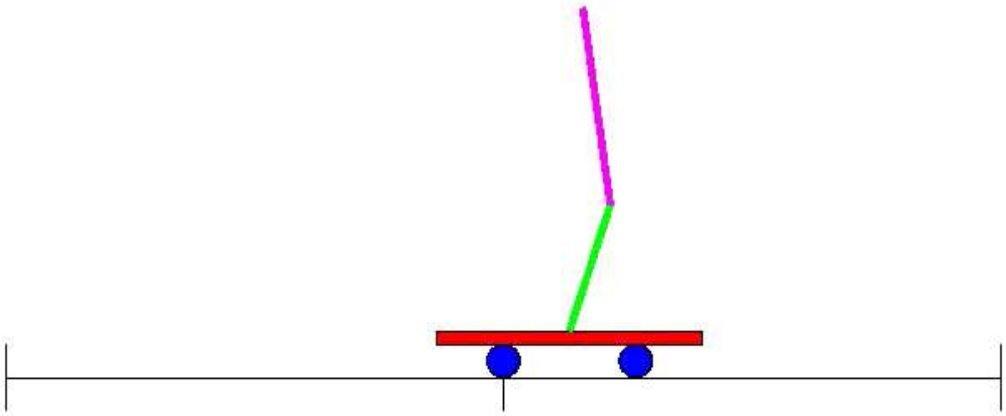
Having discovered the application of neural nets to unsupervised learning, let us also quickly see how they were used in the third branch of machine learning: **reinforcement learning**. This one requires the most mathy notation to explain formally, but also has a goal that is very easy to describe informally: learn to make good decisions. Given some theoretical agent (a little software program, for instance), the idea is to make that agent able to decide on an **action** based on its current **state**, with the reception of some **reward** for each action and the intent of getting the maximum **utility** in the long term. So, whereas supervised learning tells the learning algorithm exactly what it should learn to output, reinforcement learning provides 'rewards' as a by-product of making good decisions over time, and does not directly tell the algorithm the correct decisions to choose. From the outset it was a very abstracted decision making model - there were a finite number of states, and a known set of actions with known rewards for each state. This made it easy to write very elegant equations for finding the optimal set of actions, but hard to apply to real problems - problems with continuous states or hard-to-define rewards.



Goal: learn to choose actions that maximize:
$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$, where $0 \leq \gamma < 1$

Reinforcement learning. (Source)

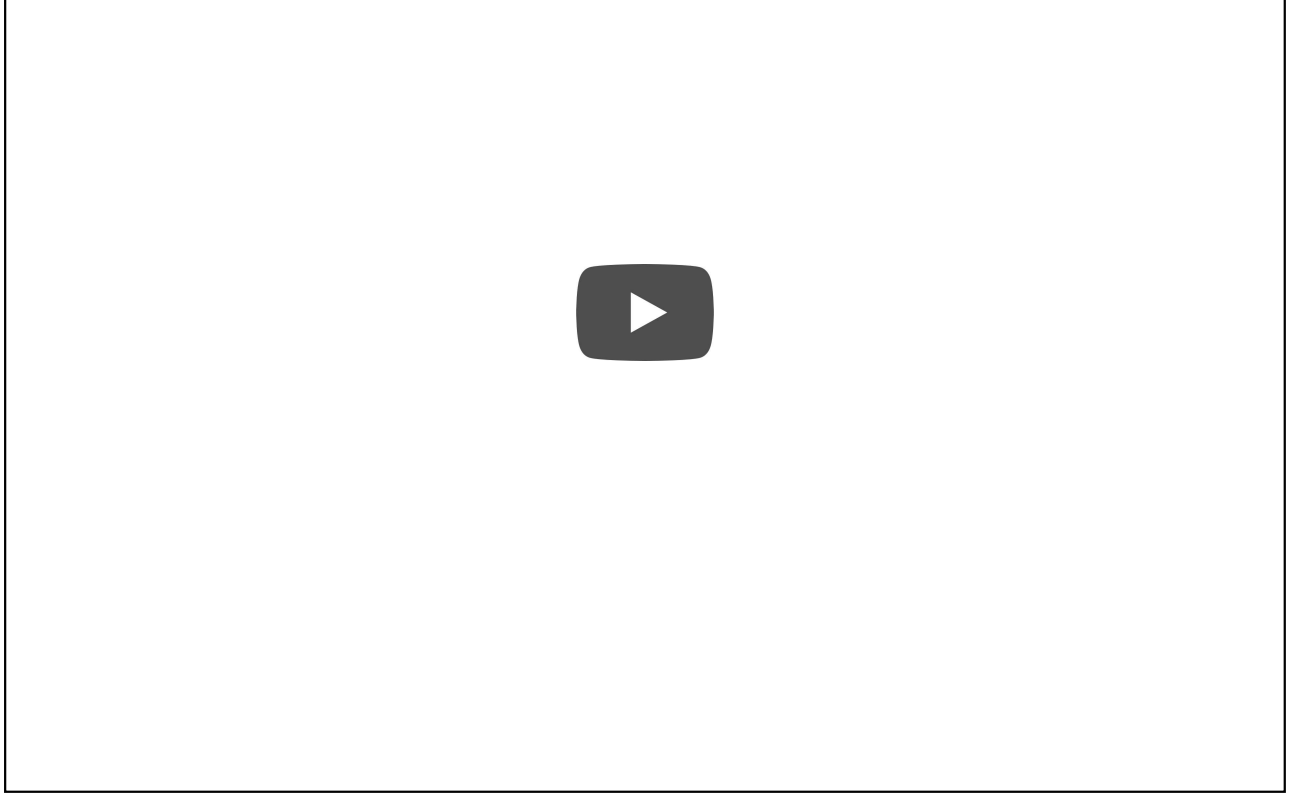
This is where neural nets come in. Machine learning in general, and neural nets in particular, are good at dealing with messy continuous data or dealing with hard to define functions by learning them from examples. Although classification is the bread and butter of neural nets, they are general enough to be useful for many types of problems - the descendants of Bernard Widrow's and Ted Hoff's Adaline were used for adaptive filters in the context of electrical circuits, for instance. And so, following the resurgence of research caused by backpropagation, people soon devised ways of leveraging the power of neural nets to perform reinforcement learning. One of the early examples of this was solving a simple yet classic problem: the balancing of a stick on a moving platform, known to students in control classes everywhere as the inverted pendulum problem [1].



The double pendulum control problem - a step up from the single pendulum version, which is a classic control and reinforcement learning task. (Source)

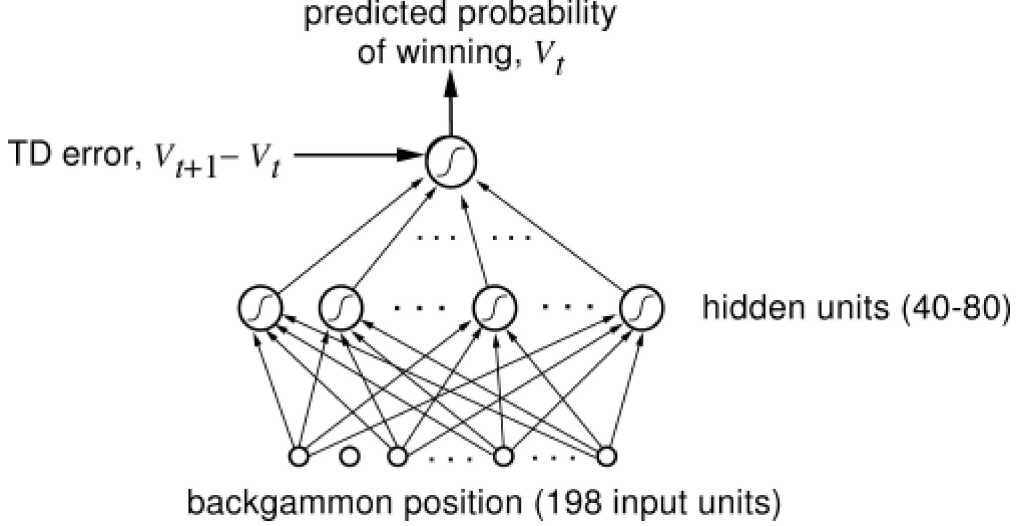
As with adaptive filtering, this research was strongly relevant to the field of Electrical Engineering, where control theory had been a major subfield for many decades prior to neural nets' arrival. Though the field had devised ways to deal with many problems through direct analysis, having a means to deal with more complex situations through learning proved useful as evidenced by the hefty 7000 (!) citations of the 1990 "Identification and control of dynamical systems using neural networks"[2]. Perhaps predictably, there was another field separate from Machine Learning where neural nets were useful - robotics. A major example of early neural net use for robotics came from CMU's NavLab with 1989's "Alvinn: An autonomous land vehicle in a neural network"[3]:

NavLab 1984 - 1994



As discussed in the paper, the neural net in this system learned to control the vehicle through plain supervised learning using sensor and steering data recorded while a human drove. There was also research into teaching robots using reinforcement learning specifically, as exemplified by the 1993 PhD thesis "Reinforcement learning for robots using neural networks"[4]. The thesis showed that robots could be taught behaviors such as wall following and door passing in reasonable amounts of time, which was a good thing considering the prior inverted pendulum work requires impractical lengths of training.

These disparate applications in other fields are certainly cool, but of course the most research on reinforcement learning and neural nets was happening within AI and Machine Learning. And here, one of the most significant results in the history of reinforcement learning was achieved: a neural net that learned to be a world class backgammon player. Dubbed TD-Gammon, the neural net was trained using a standard reinforcement learning algorithm and was one of the first demonstrations of reinforcement learning being able to outperform humans on relatively complicated tasks [5]. And it was specifically a reinforcement learning approach that worked here, as the same research showed just using a neural net without reinforcement learning did not work nearly as well.



The neural net that learned to play expert-level Backgammon. (Source)

But, as we have seen happen before and will see happen again in AI, research hit a dead end. The predictable next problem to tackle using the TD-Gammon approach was investigated by Sebastian Thrun in the 1995 "Learning To Play the Game of Chess", and the results were not good [6]. Though the neural net learned decent play, certainly better than a complete novice at the game, it was still far worse than a standard computer program (GNU-Chess) implemented long before. The same was true for the other perennial challenge of AI, Go [7]. See, TD-Gammon sort of cheated - it learned to evaluate positions quite well, and so could get away with not doing any 'search' over multiple future moves and instead just picking the one that led to the best next position. But the same is simply not possible in chess or Go, games which are a challenge to AI precisely because of needing to look many moves ahead and having so many possible move combinations. Besides, even if the algorithm were smarter, the hardware of the time just was not up to the task - Thrun reported that "NeuroChess does a poor job, because it spends most of its time computing board evaluations. Computing a large neural network function takes two orders of magnitude longer than evaluating an optimized linear evaluation function (like that of GNU-Chess)." The weakness of computers of the time relative to the needs of the neural nets was a very real issue, and as we shall see not the only one…

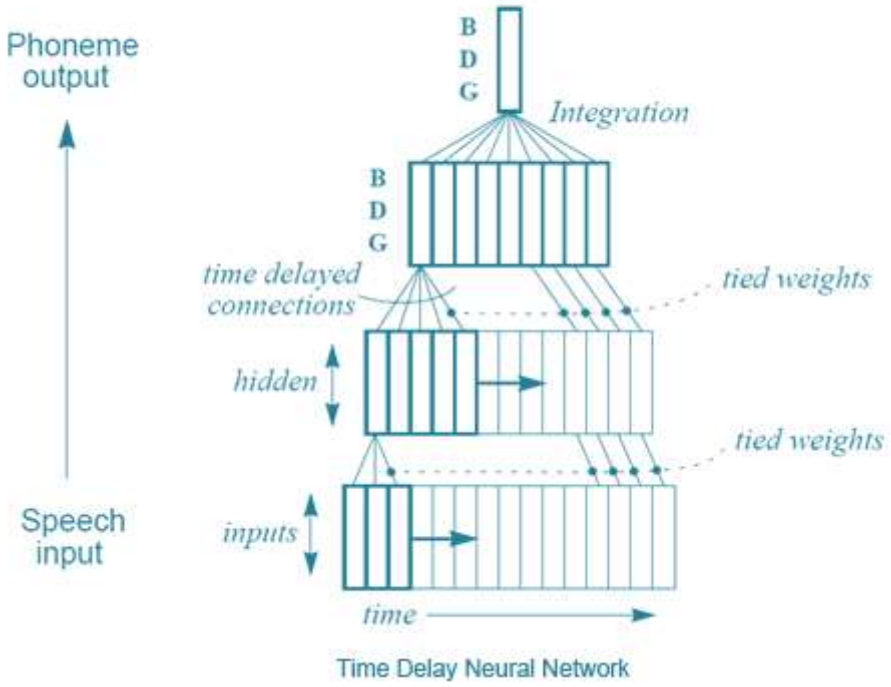
Neural Nets Get Loopy

As neat as unsupervised and reinforcement learning are, I think supervised learning is still my favorite use case for neural nets. Sure, learning probabilistic models of data is cool, but it's simply much easier to get excited for the sorts of concrete problems solved by backpropagation. We already saw how Yann Lecun achieved quite good recognition of handwritten text (a technology which went on to be nationally deployed for check-reading, and much more a while later…), but there was another obvious and greatly important task being worked on at the same time: understanding human speech.

As with writing, understanding human speech is quite difficult due to the practically infinite variation in how the same word can be spoken. But, here there is an extra challenge: long sequences of input. See, for images it's fairly simple to crop out a single letter from an image and have a neural net tell you which letter that is, input->output style. But with audio it's not so simple - separating out speech into characters is completely impractical, and even finding individual words within speech is less simple. Plus, if you think about human speech, generally hearing words in context makes them easier to understand than being separated. While this

structure works quite well for processing things such as images one at a time, input->output style, it is not at all well suited to long streams of information such as audio or text. The neural net has no 'memory' with which an input can affect another input processed afterward, but this is precisely how we humans process audio or text - a string of word or sound inputs, rather than a single large input. Point being: to tackle the problem of understanding speech, researchers sought to modify neural nets to process input as a stream of input as in speech rather than one batch as with an image.

One approach to this, by Alexander Waibel et. al (including Hinton), was introduced in the 1989 "Phoneme recognition using **time-delay neural networks**"[8]. These time-delay neural networks (TDNN) were very similar to normal neural networks, except each neuron processed only a subset of the input and had several sets of weights for different delays of the input data. In other words, for a sequence of audio input, a 'moving window' of the audio is input into the network and as the window moves the same bits of audio are processed by each neuron with different sets of weights based on where in the window the bit of audio is. This is best understood with a quick illustration:



Time delay neural networks. (Source)

In a sense, this is quite similar to what CNNs do - instead of looking at the whole input at once, each unit looks at just a subset of the input at a time and does the same computation for each small subset. The main difference here is that there is no idea of time in a CNN, and the 'window' of input for each neuron is always moved across the whole input image to compute a result, whereas in a TDNN there actually is sequential input and output of data. Fun fact: according to Hinton, the idea of TDNNs is what inspired LeCun to develop convolutional neural nets. But, funnily enough CNNs became essential for image processing, whereas in speech recognition TDNNs have been surpassed to another approach - **recurrent neural nets** (RNNs). See, all the networks that have been discussed so far have been **feedforward** networks, meaning that the output of neurons in a given layer acts as input to only neurons in a next layer. But, it does not have to be so - there is nothing prohibiting us brave computer scientists from connecting output of the last layer act as an input to the first layer, or just connecting the

output of a neuron to itself. By having the output of the network 'loop' back into the network, the problem of giving the network memory as to past inputs is solved so elegantly!

Aside: more on RNNs vs TDNNs »

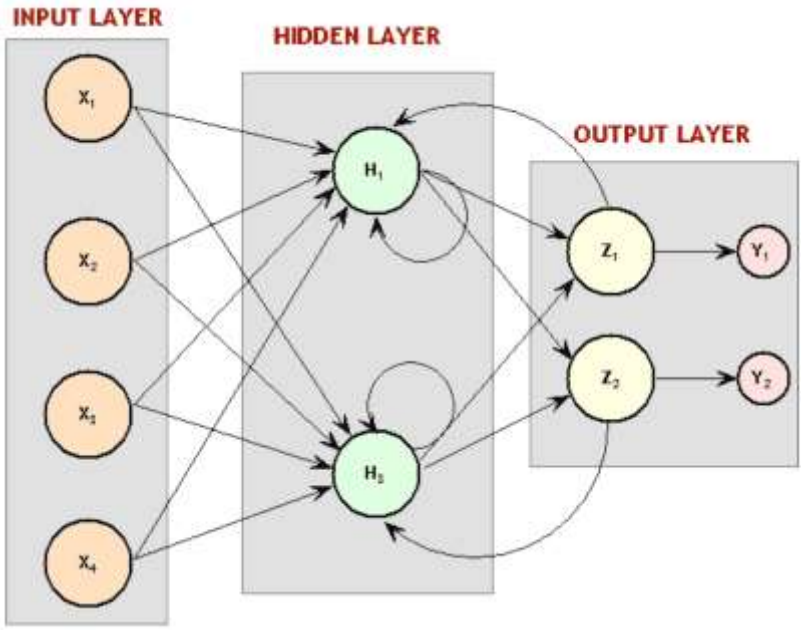
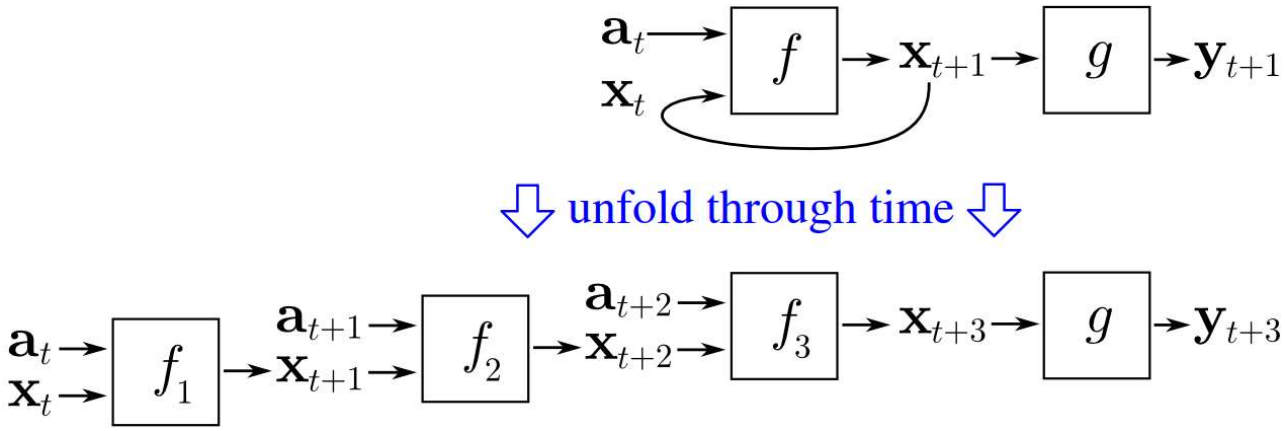


Diagram of a Recurrent Neural Net. Recall Boltzmann Machines from before? Surprise! Those were recurrent neural nets. (Source)

Well, it's not quite so simple. Notice the problem - if backpropagation relies on 'propagating' the error from the output layer backward, how do things work if the first layer connects back to the output layer? The error would go ahead and propagate from the first layer back to the output layer, and could just keep looping through the network, infinitely. The solution, independently derived by multiple groups, is **backpropagation through time**. Basically, the idea is to 'unroll' the recurrent neural network by treating each loop through the neural network as an input to another neural network, and looping only a limited number of times.



The wonderfully intuitive backpropagation through time concept. (Source)

This fairly simple idea actually worked - it was possible to train recurrent neural nets. And indeed, multiple people explored the application of RNNs to speech recognition. But, here is a twist you should now be able to predict: this approach did not work very well. To find out why, let's meet another modern giant of Deep Learning: Yoshua Bengio. Starting work on speech

recognition with neural nets around 1986, he co-wrote many papers on using ANNs and RNNs for speech recognition, and ended up working at the AT&T Bell Labs on the problem just as Yann LeCun was working with CNNs there. In fact, in 1995 they co-wrote the summary paper "Convolutional Networks for Images, Speech, and Time-Series"[9], the first of many collaborations among them. But, before then Bengio wrote the 1993 "A Connectionist Approach to Speech Recognition"[10]. Here, he summarized the general failure of effectively teaching RNNs:

> *"Although recurrent networks can in many instances outperform static networks, they appear more difficult to train optimally. Our experiments tended to indicate that their parameters settle in a suboptimal solution which takes into account short term dependencies but not long term dependencies. For example in experiments described in (ctation) we found that simple duration constraints on phonemes had not at all been captured by the recurrent network. … Although this is a negative result, a better understanding of this problem could help in designing alternative systems for learning to map input sequences to output sequences with long term dependencies eg for learning finite state machines, grammars, and other language related tasks. Since gradient based methods appear inadequate for this kind of problem we want to consider alternative optimization methods that give acceptable results even when the criterion function is not smooth."*

A New Winter Dawns

So, there was a problem. A big problem. And the problem, basically, was what so recently was a huge advance: backpropagation. See, convolutional neural nets were important in part because backpropagation just did not work well for normal neural nets with many layers. And that's the real key to deep-learning - having many layers, in today's systems as many as 20 or more. But already by the late 1980's, it was known that deep neural nets trained with backpropagation just did not work very well, and particularly did not work as well as nets with fewer layers. The reason, in basic terms, is that backpropagation relies on finding the error at the output layer and successively splitting up blame for it for prior layers. Well, with many layers this calculus-based splitting of blame ends up with either huge or tiny numbers and the resulting neural net just does not work very well - the 'vanishing or exploding gradient problem'. Jurgen Schmidhuber, another Deep Learning luminary, summarizes the more formal explanation well [11]:

> *"A diploma thesis (Hochreiter, 1991) represented a milestone of explicit DL research. As mentioned in Sec. 5.6, by the late 1980s, experiments had indicated that traditional deep feedforward or recurrent networks are hard to train by backpropagation (BP) (Sec. 5.5). Hochreiter's work formally identified a major reason: Typical deep NNs suffer from the now famous problem of vanishing or exploding gradients. With standard activation functions (Sec. 1), cumulative backpropagated error signals (Sec. 5.5.1) either shrink rapidly, or grow out of bounds. In fact, they decay exponentially in the number of layers or CAP depth (Sec. 3), or they explode. "*

Backpropagation through time is essentially equivalent to a neural net with a whole lot of layers, so RNNs were particularly difficult to train with Backpropagation. Both Sepp Hochreiter, advised by Schmidhuber, and Yoshua Bengio published papers on the inability of learning long-term information due to limitations of backpropagation [12][13]. The analysis of the problem did

reveal a solution - Schmidhuber and Hochreiter introduced a very important concept in 1997 that essentially solved the problem of how to train recurrent neural nets, much as CNNs did for feedforward neural nets - **Long Short Term Memory** (LSTM)[14]. In simple terms, as with CNNs the LTSM breakthrough ended up being a small alteration to the normal neural net model [11]:

> *"The basic LSTM idea is very simple. Some of the units are called Constant Error Carousels (CECs). Each CEC uses as an activation function f, the identity function, and has a connection to itself with fixed weight of 1.0. Due to f's constant derivative of 1.0, errors backpropagated through a CEC cannot vanish or explode (Sec. 5.9) but stay as they are (unless they "flow out" of the CEC to other, typically adaptive parts of the NN). CECs are connected to several nonlinear adaptive units (some with multiplicative activation functions) needed for learning nonlinear behavior. Weight changes of these units often profit from error signals propagated far back in time through CECs. CECs are the main reason why LSTM nets can learn to discover the importance of (and memorize) events that happened thousands of discrete time steps ago, while previous RNNs already failed in case of minimal time lags of 10 steps."*

But, this did little to fix the larger perception problem that neural nets were janky and did not work very well. They were seen as a hassle to work with - the computers were not fast enough, the algorithms were not smart enough, and people were not happy. So, around the mid 90s, a new AI Winter for neural nets began to emerge - the community once again lost faith in them. A new method called Support Vector Machines, which in the very simplest terms could be described as a mathematically optimal way of training an equivalent to a two layer neural net, was developed and started to be seen as superior to the difficult to work with neural nets. In fact, the 1995 "Comparison of Learning Algorithms For Handwritten Digit Recognition"[15] by LeCun et al. found that this new approach worked better or the same as all but the best designed neural nets:

> *"The [support vector machine] classifier has excellent accuracy, which is most remarkable, because unlike the other high performance classifiers, it does not include a priori knowledge about the problem. In fact, this classifier would do just as well if the image pixels were permuted with a fixed mapping. It is still much slower and memory hungry than the convolutional nets. However, improvements are expected as the technique is relatively new."*

Other new methods, notably Random Forests, also proved to be very effective and with lovely mathematical theory behind them. So, despite the fact that CNNs consistently had state of the art performance, enthusiasm for neural nets dissipated and the machine learning community at large once again disavowed them. Winter was back. In part 4, we shall see how a small group of researchers persevered in this research climate and ultimately made Deep Learning what it is today.

1. Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. Control Systems Magazine, IEEE, 9(3), 31-37. ↩

2. Narendra, K. S., & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. Neural Networks, IEEE Transactions on, 1(1), 4-27. ↩

3. Pomerleau, D. A. (1989). Alvinn: An autonomous land vehicle in a neural network (No. AIP-77). Carnegie-Mellon Univ Pittsburgh Pa Artificial Intelligence And Psychology Project. ↩

4. Lin, L. J. (1993). Reinforcement learning for robots using neural networks (No. CMU-CS-93-103). Carnegie-Mellon Univ Pittsburgh PA School of Computer Science. ↩

5. Tesauro, G. (1995). Temporal difference learning and TD-Gammon. Communications of the ACM, 38(3), 58-68. ↩

6. Thrun, S. (1995). Learning to play the game of chess. Advances in neural information processing systems, 7. ↩

7. Schraudolph, N. N., Dayan, P., & Sejnowski, T. J. (1994). Temporal difference learning of position evaluation in the game of Go. Advances in Neural Information Processing Systems, 817-817. ↩

8. Waibel, A., Hanazawa, T., Hinton, G., Shikano, K., & Lang, K. J. (1989). Phoneme recognition using time-delay neural networks. Acoustics, Speech and Signal Processing, IEEE Transactions on, 37(3), 328-339. ↩

9. Yann LeCun and Yoshua Bengio. 1998. Convolutional networks for images, speech, and time series. In The handbook of brain theory and neural networks, Michael A. Arbib (E()d.). MIT Press, Cambridge, MA, USA 255-258. ↩

10. Yoshua Bengio, A Connectionist Approach To Speech Recognition Int. J. Patt. Recogn. Artif. Intell., 07, 647 (1993). ↩

11. J. Schmidhuber. “Deep Learning in Neural Networks: An Overview”. “Neural Networks”, “61”, “85-117”. http://arxiv.org/abs/1404.7828 ↩ ↩[2]

12. Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. Diploma thesis, Institutfur Informatik, Lehrstuhl Prof. Brauer, Technische Universitat Munchen. Advisor: J. Schmidhuber. ↩

13. Bengio, Y.; Simard, P.; Frasconi, P., “Learning long-term dependencies with gradient descent is difficult,” in Neural Networks, IEEE Transactions on , vol.5, no.2, pp.157-166, Mar 1994 ↩

14. Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (November 1997), 1735-1780. DOI=http://dx.doi.org/10.1162/neco.1997.9.8.1735. ↩

15. Y. LeCun, L. D. Jackel, L. Bottou, A. Brunot, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard and V. Vapnik: Comparison of learning algorithms for handwritten digit recognition, in Fogelman, F. and Gallinari, P. (Eds), International Conference on Artificial Neural Networks, 53-60, EC2 & Cie, Paris, 1995 ↩

Previous Post | Next Post