Points for presentation

* Deep = many layers
* Neural nets use the same mathematics as logistic regression but are a more powerful classifier
* Activation functions: sigmoid is not commonly used, tanh is better. However, the simplest and most used is the rectified linear unit (ReLU – y = max(x,0))

Feedforward network

* The computation proceeds iteratively from one layer of units to the next
* Weights for neural nets are learned automatically using the error backpropagation algorithm
* Layers are connected with no cycles – each output is passed to a higher layer but no output is passed to a lower layer
* Also called multi-layer perceptrons (MLPs)
* Contain input, hidden and output units

A picture containing umbrella

Description automatically generated

* A single matrix W for the weights of the entire layer is used so that the computation can be done efficiently using simple matrix operations. h = σ(W x+b)
* Normally, no bias is included in the output layer, therefore, the weight matrix is multiplied (U) is multiplied by the input vector (h). z = Uh
* The softmax function is used to normalize he vector of real number results into a vector of probabilities (between 0 and 1) for classification. The result after the softmax function is a probability distribution (all numbers are between 0 and 1 and sum to 1)
* Common to use the cross-entropy loss function to find the difference between the system output and the gold output – this can be minimized using gradient descent
* To find the gradient of the loss function, since it is much more complex than logistic regression since losses are attached to other layers, we use the algorithm called error backpropagation or reverse differentiation.
* We need to intialise the weights and bias to a small random number. We should also normalize the input values to have 0 mean and unit variance.
* Hyperparameters – learning rate, mini-batch size and model architecture (number of layers, number of hidden nodes per layer, and activation function) – These need to be tuned

Neural Language models

* Language modelling – predicting upcoming words from prior word context
* Advantages of neural net-based language models over n-gram model
* No smoothing required
* Handle longer histories
* Generalize over context of similar words
* Disadvantages of neural net-based language models over n-gram model
* Slower to train
* For many tasks, n-gram is still good
* Modern neural language models are generally recurrent and not feedforward
* A feedforward neural language model takes an input at time t a representation of some words and outputs a probability distribution over possible next words
* In neural language models, the prior context is represented by embeddings.
* Representing the prior context as embeddings rather than exact words (like n-gram) allows neural language models to generalize better to unseen data
* We can learn the embeddings using methods like word2vec separately. This is called pretraining.
* Sometimes, we need to learn the embeddings at the same time when training the network, an example is when the task places strong constraints on what makes a good representation.
* We can add another layer to learn the embeddings and propagate the error all the way back to the embedding vectors.
* At the input layer, we use a one-hot-vector instead of pretrained embeddings. One hot vector per each context word and each vector is connected to the embedding layer
* We can have a shared weight matrix (E) across words rather than having a weight matrix for each previous word to the embedding layer. Each row of the input matrix E is an embedding for a word. This E is multiplied to the one hot vector of the word

A picture containing text, map

Description automatically generated

* To train the model, the parameters E, W, U and b have to be learned. Therefore we do gradient descent, using error backpropagation on the computation graph to compute the gradient. While training the model to achieve better parameters, new embeddings are learned and these can be used as word representations for other tasks.

A screenshot of a cell phone

Description automatically generated

Recurrent Neural Networks

* Convolutional Neural Networks – for image classification problems
* These are used to deal with sequence information, ex time series (Ex stock price changing with time), sequences (natural language), audio, music.
* Can we learn how to predict patterns in sequences?
* A recurrent neuron sends its output back to itself as time goes by unlike in a normal neuron which sends its output to the next neuron in the network

Diagram

Description automatically generated

* As you can see in the image above, these kind of neurons receive two types of inputs. Inputs from previous time steps and inputs from the current time steps.
* Memory Cells - Cells that are a function of inputs from previous time steps. These hold inputs from past or historical cells. Hence they hold memory
* We can also have a later with multiple recurrent neurons
* RNN are flexible in their inputs and outputs – they can have sequences as inputs and then outputs a vector – Ex input a sequences of sentences in a review and output a vector of the sentences’ sentiment score. Or input a sequence and output a sequence shifted in time. You can also pass a vector as input and receive a sequence as output (Ex pass an image and receive a sequence of sentences describing that image)
* Vanishing gradients issue
* Backpropagation goes backwards from the output to the input layer, propagating the error gradient
* When networks are deep, these gradients can be lost or vanished.
* As you go back to the lower layers, closer to the inputs, the gradients get smaller and eventually weights never change
* On the other hand, the extreme opposite can occur and gradients may explode causing issues – BUT not as bad as vanishing gradients
* We can use LSTM or GRU to fix these problems
* These occur due to the choice of the activation function
* Why does this happen? Ex in sigmoid and hyperbolic tangent activation function (same shape but goes from -1 to 1). In these functions, backpropagation computes these gradients using the chain rule. If the input is very large positive or negative number, the gradient would be very close to zero so the chain rule will multiply n of these small numbers in an n layer network. This means that the gradient will decrease exponentially with the number of layers in the network and the front layers tarin very slowly. We can solve this by using the ReLu activation function (still there is an issue that for negative numbers it will always output 0). In fact to continue solve this issue then there is the “leaky” ReLu which has a sort of negative slope for the negative numbers. Exponential Linear Unit (ELU) activation function also attempts to solve this problem. Another solution is performing batch normalization where each batch is normalized using the batch mean and standard deviation. Gradient clipping is also used to cut off gradients when reaching a predetermined limit

Chart, line chart

Description automatically generated Leaky ReLu

* Specially designed neuron units such as LSTM (long short-term memory) and GRU solve these issues. These are better than the practises explained above because if the inputs are long series of time, training could be slowed down. This could be solved by shortening the time steps used for prediction but this causes the model to perform worse in predicting longer trends. RNN after a while the network forgets the first inputs since information is lost at each step, in fact, we need some sort of long-term memory for our networks.
* LSTM cells help to address these issues
* LSTM steps

1. Forget gates layer – we decide which information we are going to forgot or throw from the cell state. Sigmoid layer – outputs a number between 0 or 1). We multiply the history from the previous time step and the input at the current time by the weights, we add the bias and pass the result through the sigmoid function. 1-keep it, 0 – forget it
2. Decide what new information you are going to store in the cell state. W parts. The first part is a sigmoid layer and is called the input gate layer. The second part is a hyperbolic tangent layer. This results in a vector of candidate values which can be added to the cell state
3. Update the old cell state by multiplying the cell of t-1 (Ct-1) by the ft  and add the input gate layer (it) by the candidate values (C(tilde)t).
4. What do we output for Ht. This is a filtered version of the cell state.

* There also exists a variant called LSTM with peepholes which allows ft and Ot to see the previous cell state.
* Another variant is the GRU (gated recurrent unit). This simplifies things by combining the input and forget gates unto a single update gate. It also merges the cell state and the hidden state. Popular recently.

Word2Vec

* NLP has 2 approaches:

1. Count based – counts how many times words occur next to their neighbours in a large text and these counts are mapped into a small dense vector for each word which replaces the words with numbers
2. Predictive based – These try to predict next words from learned small dense embeddings which are considered the parameters of the model (like Word2Vec)

* The aim of Word2Vec is to learn word embeddings by modelling each word as a vector in an n dimensional space
* Why word embeddings? When looking at audio or images and you take a count based approach, you end up with very dense data sets, however when you apply a count based approach on text, you end up with a sparse data set. Given any two words, In a count based approach, you only get information related to their frequency in a document rather than their relationships.
* Word2Vec embeds words in a vector space with words represented as vectors giving the opportunity to perform vector mathematics on words. For example you can check the similarity of 2 vectors using cosine similarity, that is checking how close or similar 2 words are to each other.
* At the beginning of training these embeddings are random but through backpropagation, these vectors are adjusted. This causes that the model eventually produces axes that represent concepts with words related to it based on neighbouring words
* How does word2vec creates these embeddings and learn them from raw text?
* 2 Models which are similar but different in the way they end up predicting words:

1. CBOW (continuous bag of words) – This approach takes example some words and then tries to predict the missing next word. Ex (Takes in: The dog chews the, Tries to predict: Bone). Better for smaller data sets
2. Skip-Gram model – This takes in the target word and tries to predict the context around it. Ex (Takes in: bone, Tries to predict: The dog chews the). Better for larger data sets

* How it trains the model? Noise contrastive training

1. We use a binary classification objective such as logistic regression to separate the real target words from the imaginary noise words in the same context. That is, contrasting the noise from the actual target word
2. Word2Vec draws an amount K of noise words drawn from the vocabulary and not all the words in the vocabulary. This makes it a lot faster to train. The larger the number of noise words, the longer the training time.
3. The goal is to assign high probability to correct words and low probability to noise words. After this, you will end up with vectors for each word
4. Once we have these vectors, we can visualise relationships by reducing dimensions from 150 to 2 using t-Distributed Stochastic Neighbour Embedding. Therefore we end up with a. 2 dimensional space, which is easy to plot. From this plot, the closer 2 points are, the bigger the similarity between the words represented by these points is.