Neural Networks and Deep Learning

Gathering & Summarizing multiple resources in one PP Deck

By Yasmine Badawy

Neural Nets

Focusing on Business Usage

Business Problems as:

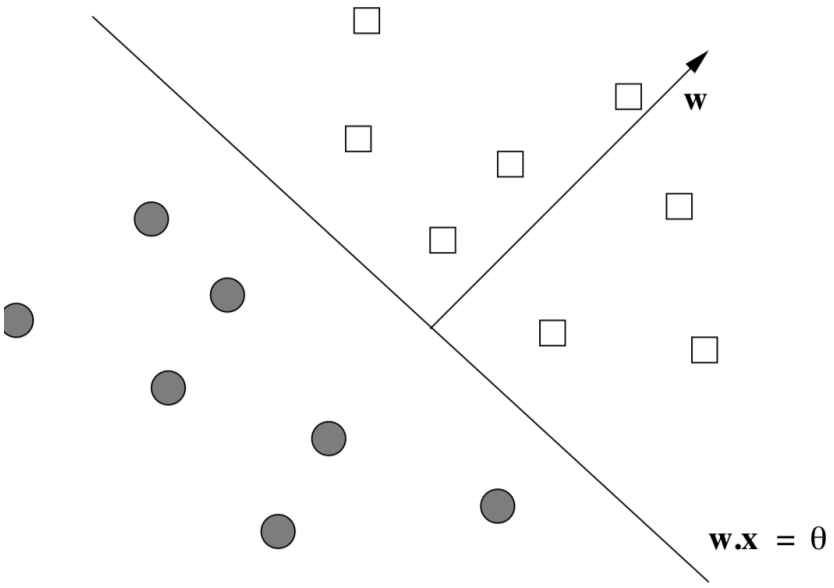
- Sales Forecasting
- Customer Research
- Data Validation
- Risk Management

Examples:

- Time Series Predictions
- Anomaly Detection
- Natural Language Understanding

Perceptron Review

- **A perceptron is a linear binary classifier.**
- Its input is a vector x = [x1, x2, . . . , xd] with real-valued components. Associated with the perceptron is a vector of weights w = [w1, w2, . . . , wd], also with real-valued components. Each perceptron has a threshold θ.
  - The output of the perceptron is +1 if w.x > θ, and the output is −1 if w.x < θ.
- Perceptron classifier works only for data that is linearly separable
  - In the sense, there is some hyperplane that separates all the positive points from all the negative points.
  - If no hyperplane exists, then the perceptron cannot converge to any particular point.



NN High Level

Each neuron multiplies an initial value by some weight, then sums results with other values coming into the same neuron, adjusts the resulting number by the neuron's bias, and then normalizes the output with an activation function.

- The computational part of training, is primarily, the choice of weights for the inputs to each node.

Perceptron - cont’d

- Perceptron is a conservative algorithm. It ignores samples that it classifies correctly
- The special case where $w.x = \theta$, will always be wrong

NN Cont'd

Would explain by example following slides

Neural networks takes a large number of handwritten digits, known as training examples, and then develops a system which can learn from those training examples.

Perceptron: *y' = sign(w · x)*

Consider each training example **t = (x, y)** in turn.

(a) Let ***y'*** = w.x.

(b) If y' and y have the same sign, then do nothing; t is properly classified.

(c) However, if y' and y have different signs, or y' = 0, replace w by w + ηyx. That is, adjust w slightly in the direction of x.

η is a hyperparameter, the learning rate

Summary

If y' is correct (i.e., yt = y')

No change: $w^{(t+1)} = w^{(t)}$

If y' is wrong: adjust w(t)

$w(t+1) = w(t) + \eta \cdot y(t) \cdot x(t)$

$x^{(t)}$ is the t-th training example

$y^{(t)}$ is true t-th class label ({+1, -1})

NN Basics

- Neural Nets are collections of perceptions or nodes
- The sum of the products of the weights and the inputs is calculated in each node, and if the value is above some threshold (typically 0) the neuron fires and takes the activated value (typically 1); otherwise it takes the deactivated value (typically -1).
- Outputs becomes the input of nodes for next level
- Last output produces output of entire NN
- The training of neural nets with many layers requires enormous numbers of training examples, but has proven to be an extremely powerful technique, referred to as deep learning,
- In general, if number of middle layer is above 3, this would be called Deep Neural Network. That is the only difference.

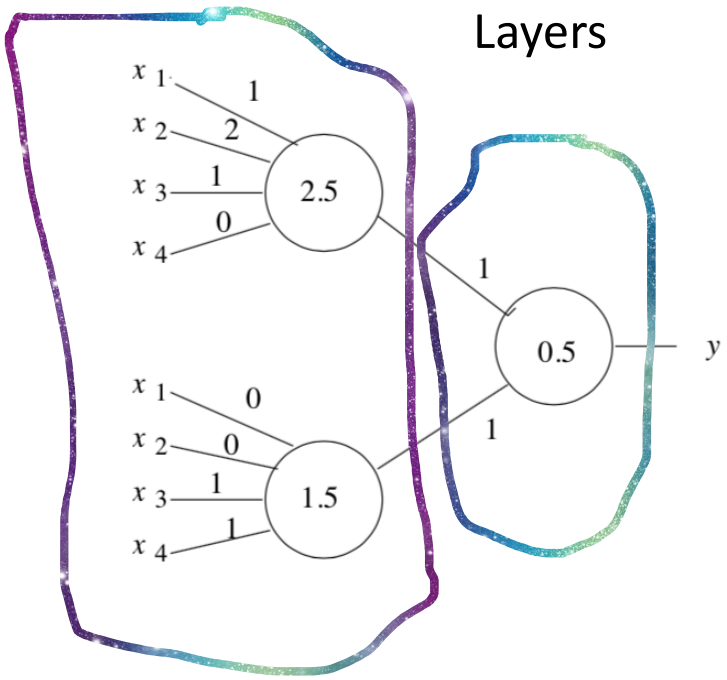
NN Architecture Review

- Input Layers
- Hidden Layers
- Output Layer
- Connection

Hidden Layers:

- There are one or more hidden layers
- Each of the layers, can have different number of nodes
  - Choosing the right number of nodes in each layer is the important part of the design process for Neural Nets

Understanding layers and Nodes



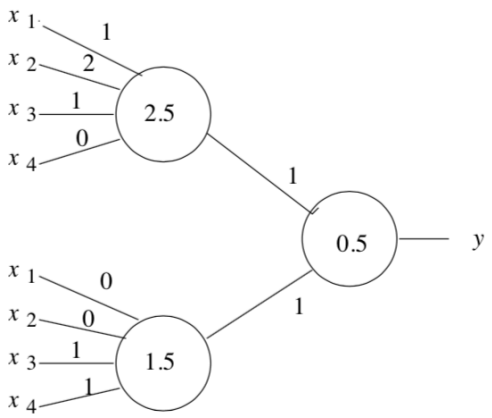
- Layer 1 has two nodes
- Layer 2 has one node

- To reach threshold 2.5 in node1 (layer1), x3 must equal 1, and either x1 or x3 should equal 1
- To reach threshold 1.5 in node2(layer1), both x3 and x4 should equal 1.

- In layers 2, It gives output y = 1 whenever either or both of the nodes in the first layer have output 1, but gives output y = 0 if both of the first-layer nodes give output 0

Assuming threshold is zero in NN

- To assume threshold equal zero, we add an additional input to each node, then transform threshold (t) as w with value (-t), while x for that input must equal 1



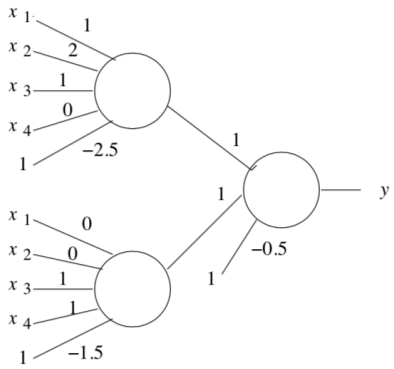


Figure 13.2: Making the threshold 0 for all nodes

Interconnections Among Nodes

A layer that receives all outputs from the previous layer is said to be fully connected.

Other options for choosing interconnections:

- Random. Random picking from previous layer
- Pooled.
  - One node for each cluster
  - This node has all and only the members of it's clusters as inputs
- Convolutional (useful in image recognition)
  - The weights on the inputs must be the same for all nodes of a single convolutional layer
  - More efficient than general NN
  - Fewer parameters at each layer. Therefore, fewer training examples can be used.

Linear algebra notation & Bias Vector

Multiplying a matrix and a vector (x's and w's) using a single linear algebra operator is much faster than coding the same operator using nested loops.

Modern Deep Learning Frameworks:

- TensorFlow ; PyTorch; Caffe

Outcome: Speeding up neural network computations

The threshold inputs to the hidden layer nodes (the negatives of the thresholds), often called the bias vector.

"A **bias vector** is an additional set of weights in a neural network that require no input, and this it corresponds to the output of an artificial neural network when it has zero input. Bias represents an extra neuron included with each pre-output layer and stores the value of "1," for each action."

Linear algebra notation & bias vector (cont'd)

Linear-algebra notation works just as well when we have larger inputs and many more nodes in one hidden layer. We just need to scale the weight matrix and bias vector appropriately.

- Perceptrons can also apply non-linear step functions. For a given input in a multiclass classification problem, the outputs specify the probability that the input belongs to the corresponding class.
  - Other non-linear step functions: Activation function following the linear transformation.

Activation Function & Feedforward Network

Complexity is in hidden layers. Each hidden layer introduces an additional matrix of weights and vector of biases, as well as its own activation function.

- This kind of network is called a feedforward network, since all edges are oriented “forward” from input to output, without cycles.

Activation Function

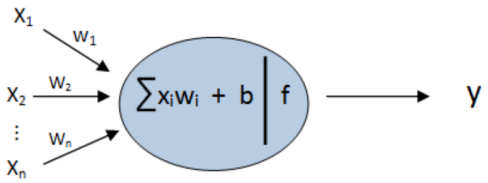
Recall: A node in NN is designed to give 0 or 1 (yes or no).

What if we want to modify the output? We apply an activation function to the output of the node.

- Some cases, the activation function takes all the outputs of a layer and modifies them as a group.
- To learn good parameter values for the network, we use gradient descent. “Activation functions add learning power to neural networks.”

Activation Function

When we add activation functions to the neurons, we leave the curse of linearity behind and are able to model complex, non-linear relationships.



X = input

W = weight

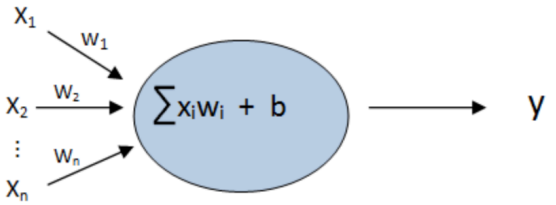
b = bias term

y = output

f = activation function

$$y = f(\sum(weight * input) + bias)$$

A neuron with an activation function



X = input

W = weight

b = bias term

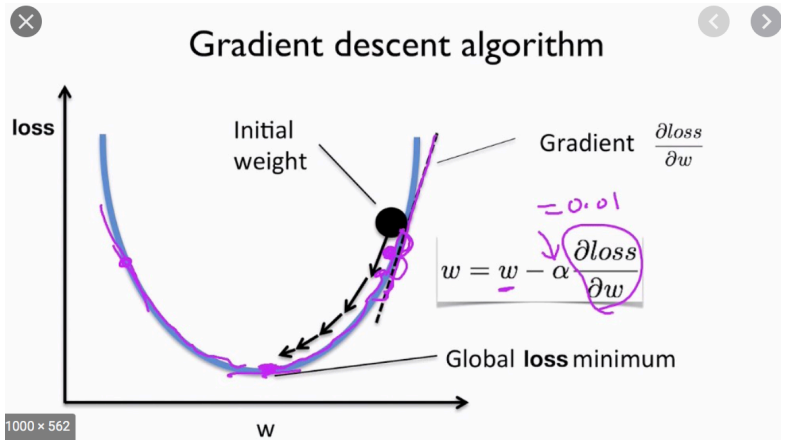
y = output

$$y = \sum(weight * input) + bias$$

A neuron without an activation function

Gradient descent

- **Gradient descent** is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. To find a local minimum of a function using gradient descent, we take steps proportional to the negative of the gradient of the function at the current point.



Activation function properties to work well w/ gradient descent

We look for the activation function with the following properties to play well with gradient descent:

1. The function is continuous and differentiable everywhere (or almost everywhere)
2. The derivate of the function does not become very small, tending towards zero. This would lead to comp / learning process to quit.
3. The derivative does not explode (becomes very large tending towards infinity). This would lead to issues of numerical instability

Iterating Gradient Descent

Given a set of training examples, we run the compute graph in both directions for each example: forward (to compute the loss) and backwards (to compute the gradients). We average the loss and gradients across the training set to compute the average loss and the average gradient for each parameter vector.

- At each iteration we update each parameter vector in the direction opposite to its gradient, so the loss will tend to decrease.

Iterating Gradient Descent and Learning Rate impact

- We stop gradient descent either when the loss between successive iterations changes by only a tiny amount (i.e., we have reached a local minimum) or after a fixed number of iterations.
- It is important to pick the learning rate carefully. Too small a value means gradient descent might take a very large number of iterations to converge. Too large a value might cause oscillations in the parameter values and never lead to convergence.
  - A good approach: start with an initial learning rate. Then, at each iteration, multiply the learning rate by a factor until the learning rate reaches a sufficiently low value.

Convergence & Learning Rate In-depth explanations

- **Convergence** describes a progression towards a network state where the network has learned to properly respond to a set of training patterns within some margin of error.

- **Adaptive Convergence** is just "convergence." This is the conventional form of the word, as it is used within most existing artificial neural network literature. It describes a set of weights during supervised training, as they begin to find (converge on) the values needed to produce the correct (trained) response.

- **Learning rate** is a tuning parameter in an optimization algorithm that determines the step size at each iteration while moving toward a minimum of a loss function.
  - Learning rate: to what extent newly acquired information overrides old information
  - Goal: to find the fastest **rate** that still decreases the loss
  - Pick a learning-rate parameter **η,** which is a small, positive real number. The choice of η affects the convergence of the perceptron. If η is too small, then convergence is slow; if it is too big, then the decision boundary will "dance around" and again will converge slowly, if at all.
  - A **learning rate schedule** changes the learning rate during learning and is most often changed between epochs/iterations

Iterating learning process

- A key feature of neural networks is an iterative learning process in which records (rows) are presented to the network one at a time, and the weights associated with the input values are adjusted each time.
- After all, cases are presented, the process is often repeated.
- During this learning phase, the network trains by adjusting the weights to predict the correct class label of input samples.
- “The iterative training process of neural networks solves an optimization problem that finds for parameters (model weights) that result in a minimum error or loss when evaluating the examples in the training dataset.”

Exploding Gradients

Exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training. This has the effect of your model being unstable and unable to learn from your training data.

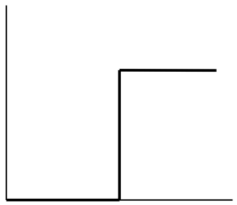
Stochastic gradient descent (sampling)

When we have a large training set, we may not want to use the entire training set for each iteration, as it might be too time-consuming.

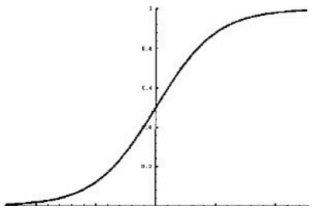
- So for each iteration, we randomly sample a “minibatch” of training examples. This variant is called stochastic gradient descent.

Alternatives to step functions

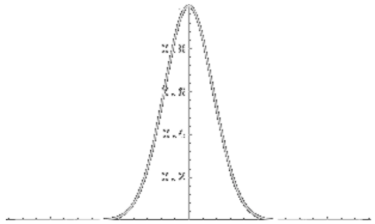
There have been many proposals for activation functions, including thresholds, sigmoid, symmetric sigmoid and Gaussian etc.,



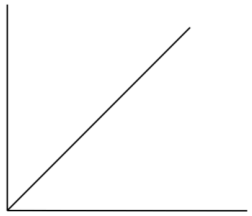
Threshold or step



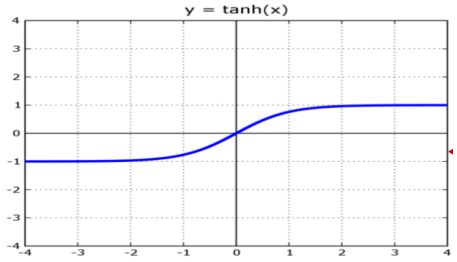
Sigmoid



Gaussian



Linear



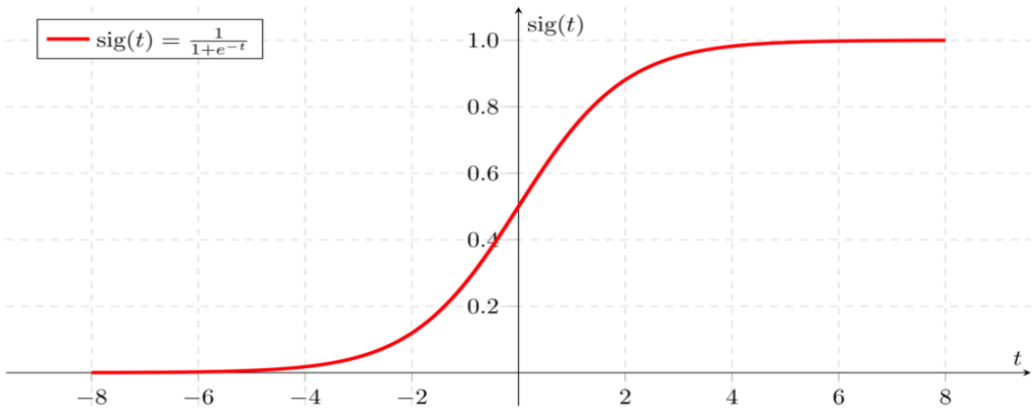
Tanh

Sigmoid

Sigmoid function, unlike step function, introduces non-linearity into our neural network model. ... This non-linear activation function, when used by each neuron in a multi-layer neural network, produces a new “representation” of the original data, and ultimately allows for non-linear decision boundary.

- This is an S-shaped curve with output values between -1 and 1 (or 0 and 1), and which is monotonically increasing.

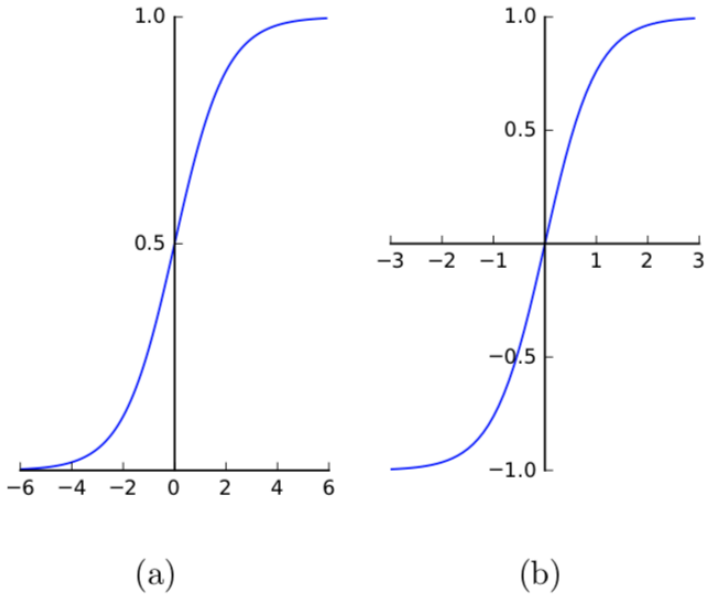
$$f(S) = \frac{1}{1 + e^{-cs}}$$



The Hyperbolic Tangent (tanh)

The hyperbolic tangent is just a scaled and shifted version of the sigmoid. It's output is in the range [-1, 1] and is symmetric around 0.

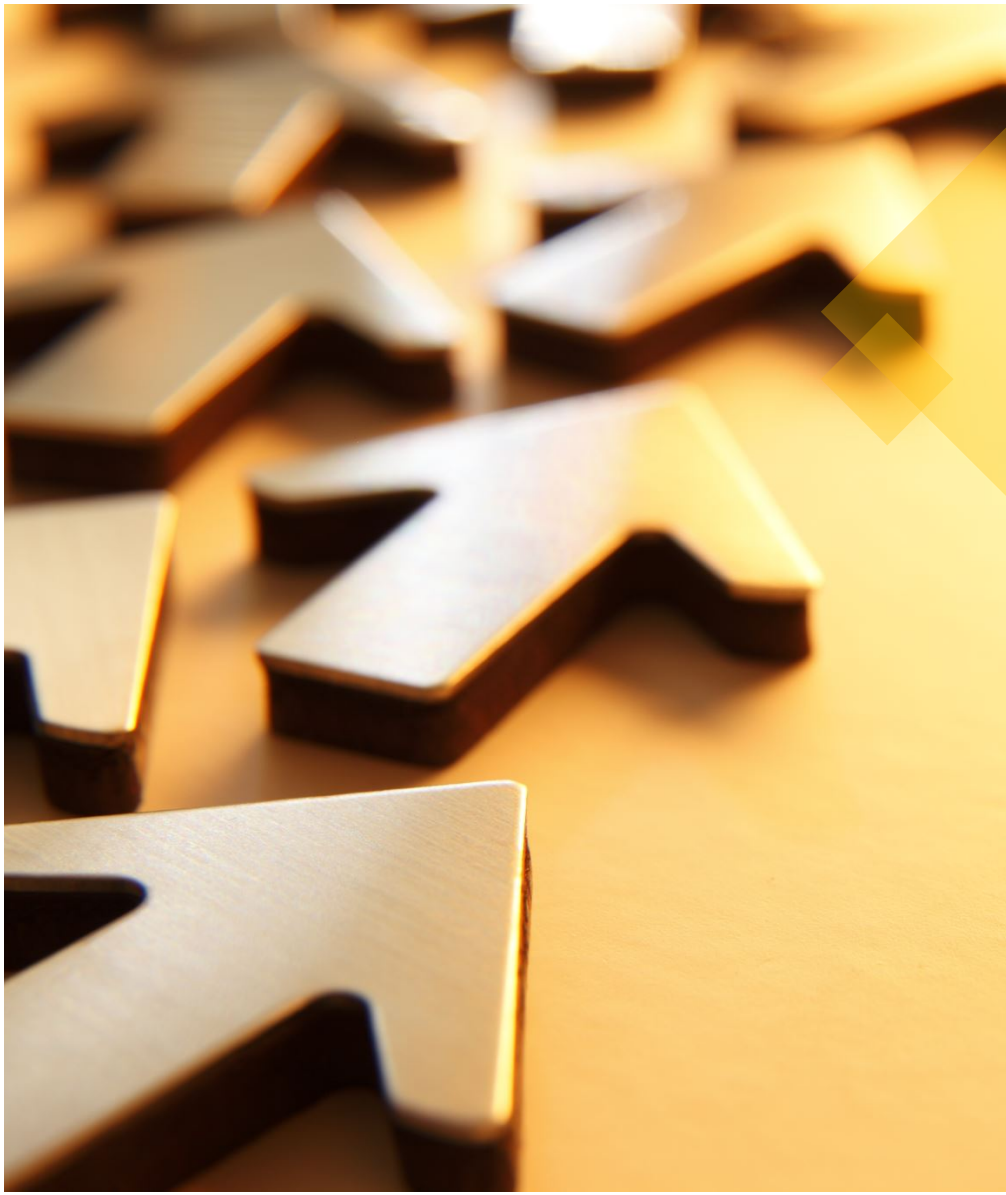
$$f(S) = \frac{(1-e^{-s})}{(1+e^{-cs})}$$



Logistic sigmoid (a) and hyperbolic tangent (b)

Forward Propagation (review)

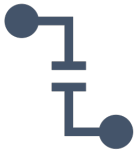
Forward implies moving ahead and **propagation** is a term for saying spreading of anything. **Forward propagation** means we are moving in only one direction, from input to the output, in a **neural network**.



Backpropagation Algorithm

Most popular neural network algorithm

The network processes the records in the “training set” one at a time, using the weights and functions in the hidden layers, then compares the resulting outputs against the desired outputs.



Errors are then propagated back through the system, causing the system to adjust the weights for application to the next record.

Backpropagation- explained again

- **The goal of the backpropagation algorithm is to compute the gradient of the loss function with respect to the parameter of the network.**

  - We work backwards through the compute graph, applying the chain rule at each stage.
  - At each point, we pick a node all of whose successors have already been processed.

- *Since* we need to compute these gradients several times, once for each iteration of gradient descent, we can avoid repeated computation by adding additional nodes to the compute graph for backpropagation: one node for each gradient computation.
- *This* combined gradient does not saturate or explode, and leads to good learning behavior. That observation explains why softmax and cross entropy loss work so well together in practice. (no need to go further here)

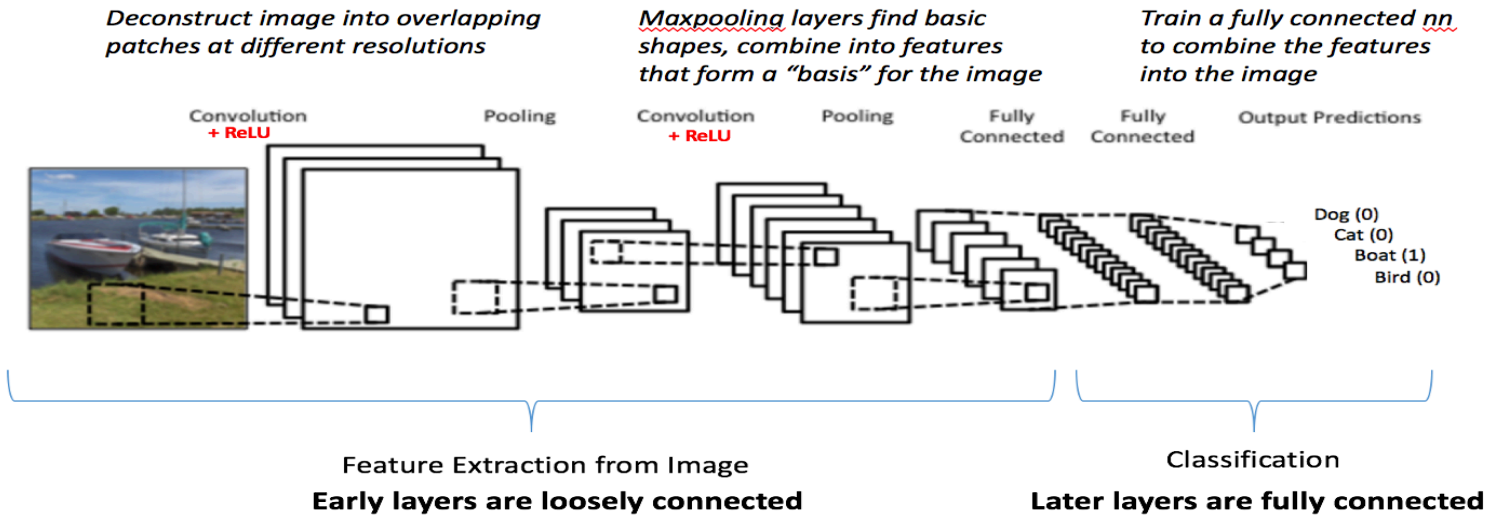
Special forms of NN

- Since learning all the weights on all the inputs to all the nodes of the network is in general a hard and time-consuming task, below are special forms of network that greatly simplify the process of training the network to recognize the desired class or classes of inputs.
- (CNN's), which are specially designed to recognize classes of images.
- Recurrent neural networks (RNN's) and long short-term memory networks (LSTM's), are designed to recognize classes of sequences, such as sentences (sequences of words).

CNN (for image processing)

A Type of NN

- Specific kind of deep learning, designed for modeling on images.
- Use a sliding patch at various resolutions to automatically discover basic building block features like edges, curves…
- The basic features are combined into more complex features in further layers.
- The final image is learned as a combination of the complex, automatically-discovered features.



From Dr. Stephen Coggeshall class slides

RNN (Recurrent Neural Networks)

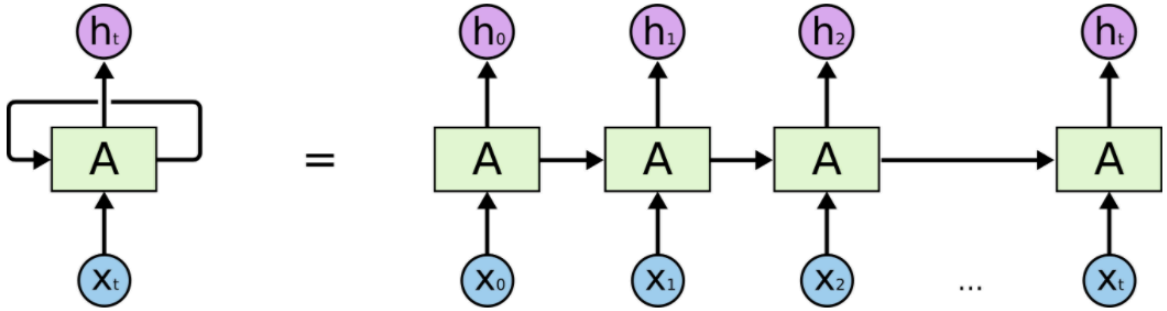
A type of NN

RNN remembers past information. Regular NN, in general, lacks reasoning about previous events.

- RNN are networks with loops in them, allowing information to persist.
- RNNs, last few years, were applied to a variety of problems: speech recognition, language modeling, translation, image captioning
- RNN can be thought of as multiple copies of the same network, each passing a message to a successor.

In diagram: x is input,

while h is output.



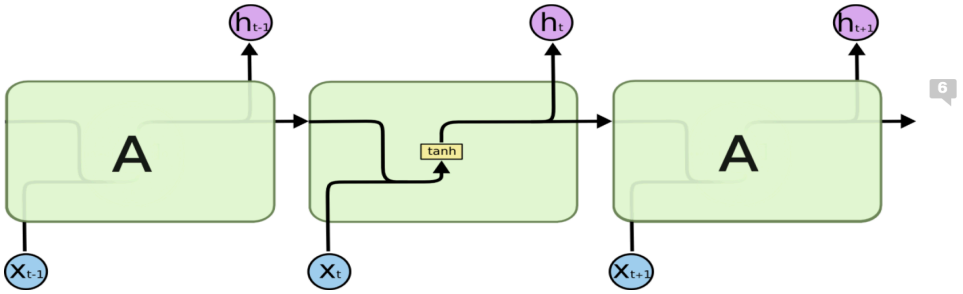
LSTM (Long Short Term Memory) Units
For natural language processing

LSTM, a special kind and a much better version than the standard RNN.

- LSTM capable of learning long term dependencies
- The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates.
  - The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It's very easy for information to just flow along it unchanged.

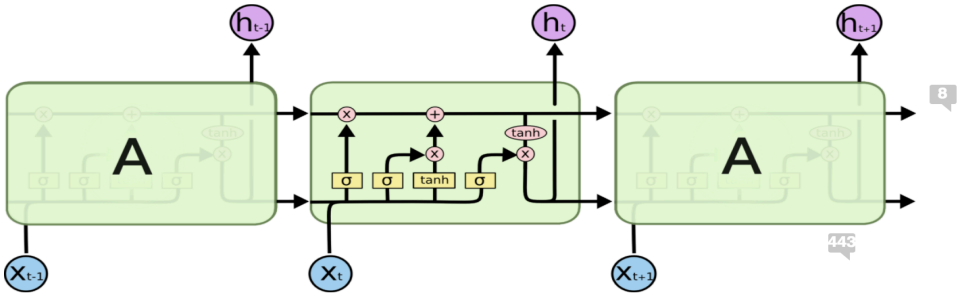
LSTM Units cont'd

For natural language processing



The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.



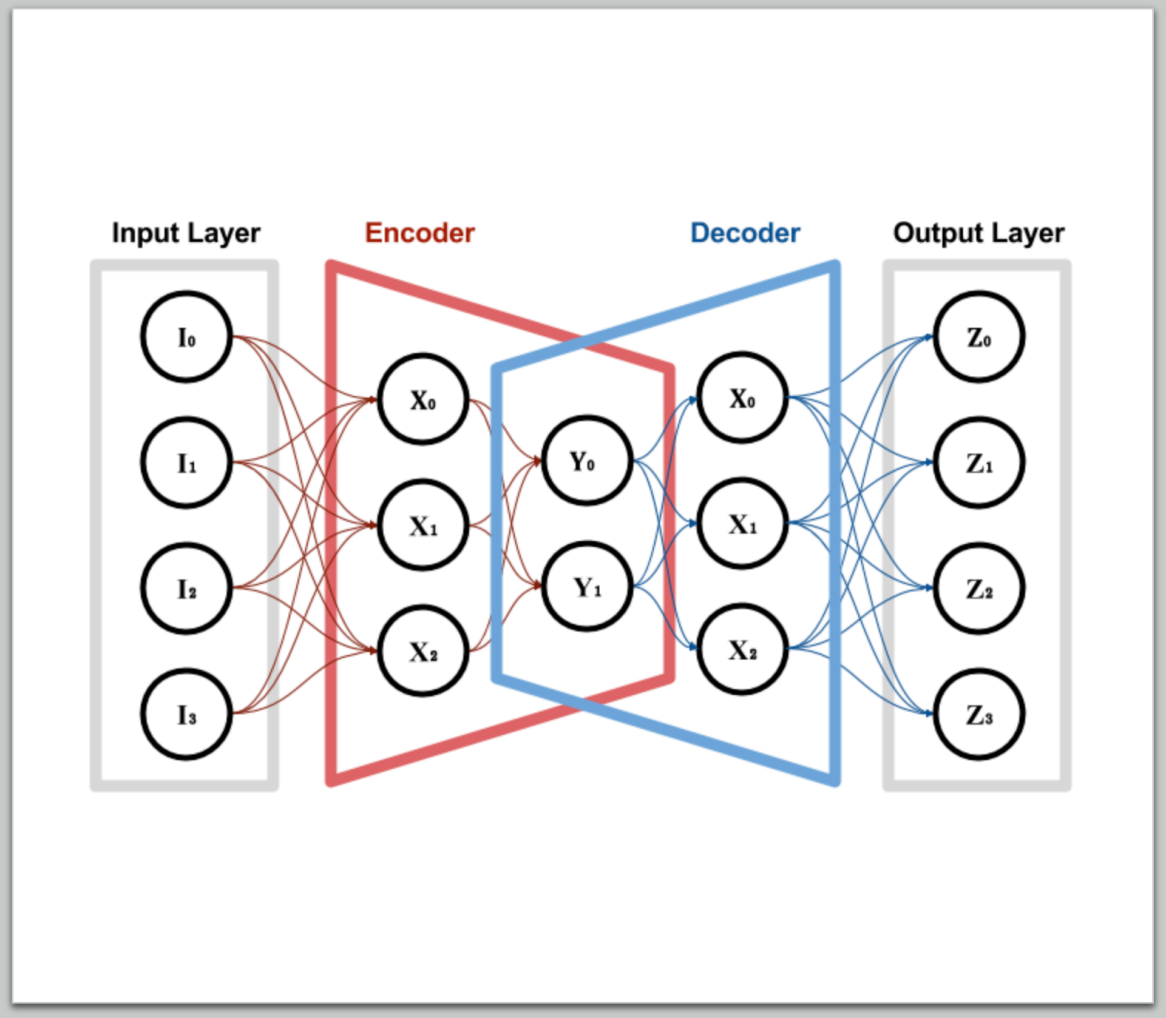
Understanding LSTM Networks, Colah's blog

Autoencoders

An **autoencoder** is a special type of neural network whose objective is to match the input that was provided with.

- In order to succeed in connecting the input values to their respective output nodes, the autoencoder has to somehow *compress* the information provided and to reconstruct it before presenting it as its final output.

https://www.alanzucconi.com/2018/03/14/an-introduction-to-autoencoders/



Autoencoder in wikipedia

“ An autoencoder is a type of artificial neural network used to learn efficient data codings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal “noise”. Along with the reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduced encoding a representation as close as possible to its original input, hence its name. ”

Executing Neural Network Model

Data Preparation

Data with different scales can induce instability in NN

- Even if stable, it can increase the time necessary
- The algorithm would spend more time adjusting for variation

Solution: Normalization. If using JMP, just transform Covariates

Data scaling or normalization is a process of making model data in a standard format so that the training is improved, accurate, and faster. The method of scaling data in neural networks is similar to data normalization in any machine learning problem.

Determining Number of Nodes

For output:

- As number of output nodes increase:
  - The number of variables increases the local optima while decreasing the probability of finding the global optimum
  - Amount of time increases as variables increases

Recommendation:

- To predict a single continuous variable or a binary variable, use one output node.
- For a categorical variable with five levels, use five output nodes, one for each level.

Determining Number of Nodes

For Hidden node:

Guideline principle for determining the number of nodes is that there should be enough nodes to model the target variables, but not so many to overfit.

Rule of thumb for determining number of nodes:

- Between number of inputs and number of outputs
- Equal the number of inputs plus the number of outputs times two-thirds
- No more than twice the number of inputs
- Approximately ln(T) where T is the sample size
- Between 5% and 10% of the number of variables

Boosting

Boosting is an option that can enhance the predictive ability of the NN. Boosting developed for predicting continuous target variables. Gradient boosting is a form of boosting used for NN.

Method of Boosting:

- Run the algorithm where all observations have equal weight
- Next step: give more weight to the incorrectly classified observations and less weight to the correctly classified observations
- Rerun the algorithm; repeat until algorithm has run T “number of models” times

❖ If observation has been classified more times as zero than one, then zero is its final classification. Same with one.

Overfitting

Overfitting is a general problem that affects all machine-learning models. However, deep neural networks are particularly susceptible to overfitting, because they use many more parameters (weights and biases) than other kinds of models. Several techniques have been developed to reduce overfitting in deep networks, usually by trading higher training error for better generalization. The process is referred to as model regularization.

Regularization (high level)

A larger number of gradient boosting iterations reduces training set errors. Raising the number of gradients boosting iterations too high increases overfitting. The monitoring of the error of prediction from a distinct validation data set can help in choosing the optimal value for the number of gradients boosting iterations.

Regularization (in depth)

We have presented our goal as one of minimizing loss (i.e., prediction error) on the training set. Gradient descent and stochastic gradient descent help us achieve this objective. In practice, the real objective of training is to minimize the loss on new and hitherto unseen inputs. Our hope is that our training set is representative of unknown future inputs, so a low loss on the training set translates into good performance on new inputs. Unfortunately, the trained model sometimes learns idiosyncrasies of the training data that allow it to have low training loss, but not generalize well to new inputs – the familiar problem of overfitting.

*Mining of Massive Datasets by Jure Leskovec, Anand Rajaman, Jeffrey David Ullman

Drop Out

Dropout is a technique that **reduces overfitting** by making random changes to the underlying deep neural network. Recall that when we train using stochastic gradient descent, at each step we sample at random a minibatch of inputs to process. When using dropout, we also select at random a certain fraction (say half) of all the hidden nodes from the network and delete them, along with any edges connected to them.

Why does dropout reduce overfitting? Several hypotheses have been put forward, but perhaps the most convincing argument is that dropout allows a single neural network to behave effectively as a collection of neural networks. Imagine that we have a collection of neural networks, each with a different network topology. Suppose we trained each network independently using the training data, and used some kind of voting or averaging scheme to create a higher-level model. Such a scheme would perform better than any of the individual networks. The dropout technique simulates this setup without explicitly creating a collection of neural networks.

*Mining of Massive Datasets by Jure Leskovec, Anand Rajaman, Jeffrey David Ullman

Avoiding Overfitting / Norm Penalties

- Gradient descent is not guaranteed to learn parameters (weights and biases) that reduce the training loss to an absolute minimum. In practice, the procedure learns parameters that correspond to a local minimum in the training loss. There are usually many local minima, and some might lead to better generalization than others. In practice, it has been observed that solutions where the learned weights have low absolute values tend to generalize better than solutions with large weights.
- We can force gradient descent to favor solutions with low weight values by adding a term to the loss function. Suppose w is the vector of all the weight values in the model, and L0 is loss function used by our model. We define a new loss function L as follows:

$$L = L0 + \alpha \|w\|2$$

PCA in NN (if needed)

- “In principal, the linear transformation performed by PCA can be performed just as well by by the input layer weights of the neural network, so it isn't strictly speaking necessary. However, as the number of weights in the network increases, the amount of data needed to be able to reliably determine the weights of the network also increases (often quite rapidly), and over-fitting becomes more of an issue (using regularization is also a good idea). The benefit of dimensionality reduction is that it reduces the size of the network, and hence the amount of data needed to train it. The disadvantage of using PCA is that the discriminative information that distinguishes one class from another might be in the low variance components, so using PCA can make performance worse.
- Like most things in statistical pattern recognition, there is no single recipe that works reliably for all problems, and really the best thing to do is to try both approaches and see which works best.”

* https://stats.stackexchange.com/questions/67986/does-neural-networks-based-classification-need-a-dimension-reduction

Validation: Holdback Validation

Data divided is randomly divided into two parts: training and validation (holdback) sample.

- Weights are estimated on the training sample
- Then, weights are used on the holdback sample to calculate the error

❖ As the algorithm iterates, the error on both samples would decline as the NN learns the model

❖After sufficient number of iterations (that is, recalculations of the weights) the NN would have learned the model and NN would begin to overfit the noise in training sample (calculated error would increase)

Holdback Validation getaway

- As the number of iterations increase, we would experience overfitting.
- After n iterations, the neural network has determined the weights that minimize the error on the holdback sample. Any further iterations will only fit the noise in the training data and not the underlying model. Therefore, **the weights based on n iterations that minimize the error on the holdout sample should be used**.

Validation: K-fold cross-validation

Divide the data into k groups, or folds, that contain the same number of observations

- Consider k-1 to be the training data set
- Kth fold to be the validation data set

Repeat this k times, each time leaving out a different fold (obtaining k-measures of accuracy)

- Cross validation is often used when there are not enough data to divide the data into training and validation samples

Citations

- Mining of Massive Datasets by Jure Leskovec, Anand Rajaman, Jeffrey David Ullman
- Fundamentals of Predictive Analytics with JMP by Ron Klimberg.B.D. McCullough
- Dr. Stephen Coggeshall class slides / Dr. Cosimo Arnesano class slides
- https://blog.statsbot.co/neural-networks-for-beginners-d99f2235efca
- https://deepai.org/machine-learning-glossary-and-terms/bias-vector#:~:text=A%20bias%20vector%20is%20an,1%2C%E2%80%9D%20for%20each%20action.
- https://towardsdatascience.com/activation-functions-in-neural-networks-eb8c1ba565f8
- https://www.youtube.com/watch?v=b4Vyma9wPHo
- https://en.wikipedia.org/wiki/Gradient_descent
- http://standoutpublishing.com/g/convergence.html
- http://neuralnetworksanddeeplearning.com/chap1.html
- https://machinelearningmastery.com/why-training-a-neural-network-is-hard/
- https://towardsdatascience.com/first-neural-network-for-beginners-explained-with-code-4cfd37e06eaf
- https://stats.stackexchange.com/questions/67986/does-neural-networks-based-classification-need-a-dimension-reduction
- https://subscription.packtpub.com/book/big_data_and_business_intelligence/9781788397872/5/ch05lvl1sec62/scaling-of-data-in-neural-network-models#:~:text=Data%20scaling%20or%20normalization%20is,in%20any%20machine%20learning%20problem.
- https://blog.quantinsti.com/forward-propagation-neural-networks/
- https://colah.github.io/posts/2015-08-Understanding-LSTMs/#:~:text=Long%20Short%20Term%20Memory%20networks,of%20learning%20long%2Dterm%20dependencies.&text=In%20standard%20RNNs%2C%20this%20repeating,RNN%20contains%20a%20single%20layer.
- https://corporatefinanceinstitute.com/resources/knowledge/other/gradient-boosting/#:~:text=Gradient%20Boosting%20Regularization&text=M%20stands%20for%20the%20number,tree%20is%20the%20base%20learner.&text=In%20addition%20to%20using%20the,as%20an%20efficient%20regularization%20parameter.
- https://www.alanzucconi.com/2018/03/14/an-introduction-to-autoencoders/
- https://en.wikipedia.org/wiki/Autoencoder
- https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f