APS360
Applied Fundamentals of Machine Learning

Michael Boyadjian

January 14, 2021

Contents

1 Introduction to Machine Learning

1.1 Terminology

- **Artificial Intelligence**
  - The intelligence of machines and the branch of computer science that aims to create it.
  - The science and engineering of making intelligent machines, especially intelligent computer programs.
  - Often used to describe machines (or computers) that mimic "cognitive" functions that humans associate with the human mind, such as "learning" and "problem solving".
  - *Create intelligent machines that work and act like humans.*
- **Machine Learning**
  - A branch of artificial intelligence concerned with design and development of algorithms to build mathematical models based on sample data, known as "training data", in order to make predictions or decisions without being explicitly programmed to perform the task
  - *Find an algorithm that automatically learns from structured (preprocessed) example data.*
- **Deep Learning**
  - Deep learning is a subset of machine learning that has networks capable of learning unsupervised from data that is unstructured or unlabeled. Also known as deep neural learning or deep neural network.
  - *Uses deep neural networks to automatically learn from unstructured example data.*

1.2 AI Reality

- AI works well on pattern recognition problems with big data ; examples include voice recognition, facial recognition, etc.
- AI does not work as well on on planning and prediction; examples include stock market, language, comedy, etc.

1.3 Types of Learning

- **Supervised Learning**
  - Learning model that maps an input to an output based on example input-output pairs *(ex. age prediction given a headshot)*
  - Machine learning is a game of balance, with our objective being to generalize to all possible future data

- **Unsupervised Learning**
  - Focuses on finding patterns, regularities or structure in (unlabeled) data *(ex. clustering, style transfer, etc.)*
- **Reinforcement Learning**
  - Learning what actions to take to optimize long-term reward *(ex. playing a video game, learning to walk, etc.)*
  - Correct input/label pairs are never presented, nor are sub-optimal actions explicitly corrected
  - Payoff is often delayed

1.4 Data Splitting

- **Training and Testing Data**
  - More data leads to a better model, but we need to set some aside for testing
  - Have a training and testing split
  - Trying until favourable outcomes are achieved would make testing data effectively same as training data and would lead to overfitting
- **Validation and Holdout Data**
  - Split the testing data into validation and holdout
  - Only use holdout data once
  - Typical splits are 60/20/20, 70/15/15, or 80/10/10

2 Artificial Neural Networks (ANNs)

2.1 Biological Neurons

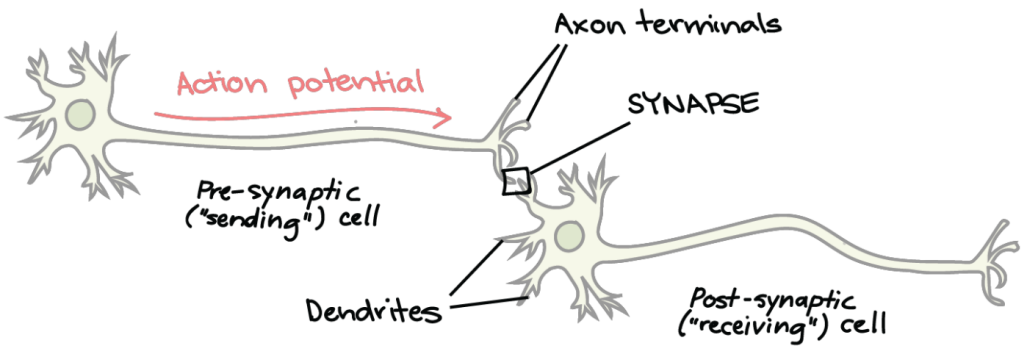
2.1.1 Artificial Pigeon

In order to use an artificial neural network (pigeon) to solve a simple classification problem, we will need to:

1. Build an artificial neural network – or rather, an artificial (pigeon) brain
2. Decide how to reward the artificial pigeon
3. Decide how to train the artificial pigeon
4. Determine how well our artificial pigeon performs the classification task

2.1.2 Neuron Anatomy

- The **dendrites**, which are connected to other cells that provides information.
- The **cell body** (soma), which consolidates information from the dendrites.
- The **axon**, which is an extension from the cell body that passes information to other cells.
- The **synapse**, which is the area where the axon of one neuron and the dendrite of another connect.
  - Small voltage difference between inside and outside of cell
  - When a neuron receives "information" in its dendrites, the voltage difference along that part of the cell lowers.
  - If the total activity in a neuron's dendrites lowers the voltage difference enough, the entire cell depolarizes and the cell fires.
  - The voltage signal spread along the axon and to the synapse, then to the next cells.

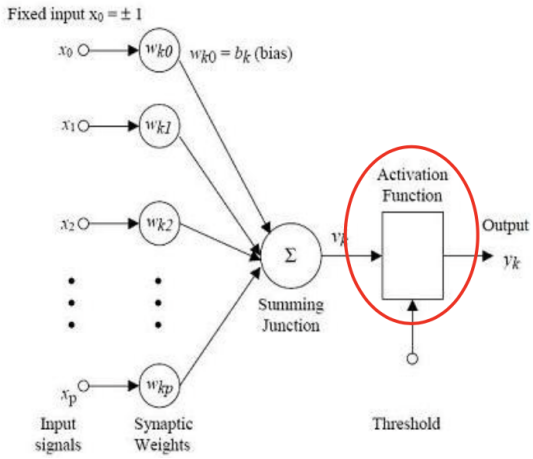


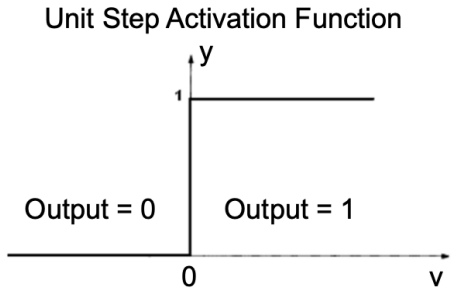
2.2 Artificial Neurons

- Here the neuron is actually a processing unit, it calculates the weighted sum of the input signal to the neuron to generate the activation signal v_k

$$v_k = \sum_{i=0}^{p} w_i x_i = w^T x$$

- Then a hard threshold is applied on the activation signal to get the output signal y_k
- x_0 is reserved for the bias





2.3 Forward Pass

2.3.1 Error

In order to train an ANN we first have to define the error on our classifications (loss function), the error depends on the problem. For **regression** problems we use mean squared error (MSE):

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(y_i - t_i)^2$$

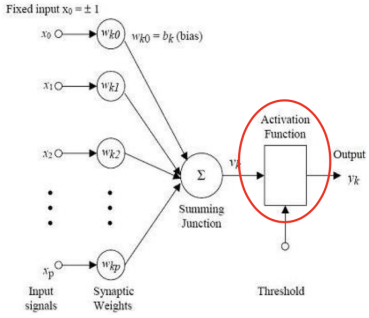
where N is the number of training samples, y is the model predicted output and t is the desired output. For **classification** problems we use cross entropy (CE):

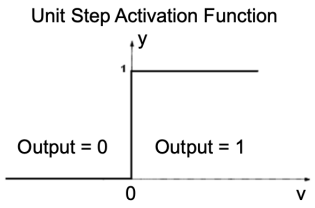
$$CE = -\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K} t_{n,k}\log(y_{n,k})$$

where N is the number of training samples, K is the number of classes, y is the model predicted output and t is the target label. The objective is to get $error \approx 0$.

2.3.2 Weight Tuning

- Randomly pick weights until it works
- Change one weight at a time in the direction that reduces error
- Gradient descent - Adjusting weights according to the slope (gradient) will guide us to the minimum error





2.4 Back Propagation

2.4.1 Gradient Descent

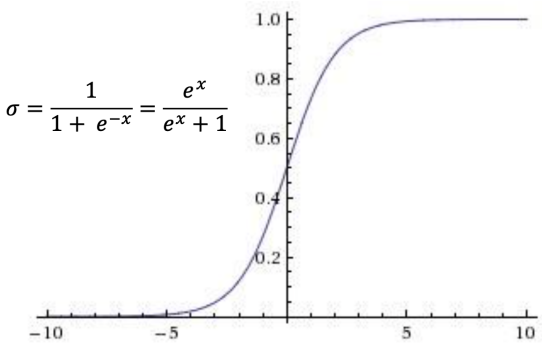
- Find how the error changes as you change each weight ($\frac{dE}{dw}$)

- Update weight in the opposite direction of the gradient

$$W = W - \eta \frac{dE}{dW}$$

where η is the learning rate, the step size taken according to the gradient

- Need the function be differentiable
- Step function gradient undefined at $f(0)$, and 0 everywhere else, require another activation function - we can use sigmoid (logistic) activation function



- Other activation functions include tanh activation and $ReLU$ activation

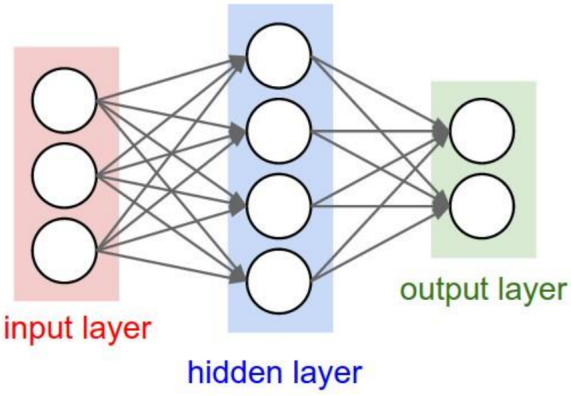
$$tanh = \frac{e^x - e^{-x}}{e^x + e^{-x}} = \frac{e^{2x} - 1}{e^{2x} + 1}$$

$$ReLU = \max\{0, x\}$$

- $ReLU$ is a preferred activation function
  - Resolves vanishing gradients issues resulting from deep networks
  - Computationally more efficient

2.5 ANN Architecture

- An architecture of a neural network describes the neurons and their connectivity in the network
- Information only flows from one layer to a later layer, from the input to the output.
- Neurons between adjacent layers are fully pairwise connected.
- Input layer is not counted in number of layers



- Adding more linear layers does not help increase the capacity of the model
- Any sequence of linear layers can be equivalently represented with a single linear layer.

2.6 ANN Training

Training deals with techniques for effectively and efficiently tuning the model weights (parameters). To train we need to calculate the error on the predictions (classifications), which we can achieve with a Loss Function. An optimizer is then used to achieve this more efficiently

2.6.1 Loss Function

- Computes how "bad" a set of predictions was, compared to the ground truth (labels).
  - **Large loss** = the network's prediction differs from the ground truth
  - **Small loss** = the network's prediction matches the ground truth

2.6.2 Optimizer

- An optimizer determines, based on the value of the loss function, how each parameter (weight) should change.
- Solves the credit assignment problem: how do we assign credit (blame) to the parameters based on how the network performs?
- Each update of the weights takes one step towards solving the optimization problem
- We use the derivative of the loss function at a training example, and take a step towards its negative gradient

2.6.3 Forward and Backward Pass

- **Forward Pass**
  - Makes a prediction
  - Information flows forwards from input to output layer
- **Backward Pass**
  - Computes gradients for making changes to weights
  - Information flows backwards from output to input layer

2.7 Hyperparameter Tuning

Hyperparameters are neural network settings that cannot be optimized using an optimizer

2.7.1 Batch Size

- Instead of working with one sample at a time we apply batching:
  1. Use our network to make the predictions for n images
  2. Compute the average loss for those n images
  3. Take a "step" to optimize the average loss of those n images
- The batch size is the number of training examples used per optimization "step".
- Each optimization "step" is known as an iteration.
- The parameters are updated once per iteration.
- If batch size **too small** = we optimize a (possibly very) different function; noisy loss at each iteration
- If batch size **too large** = average loss might not change very much as batch size grows; expensive

2.7.2 Epochs

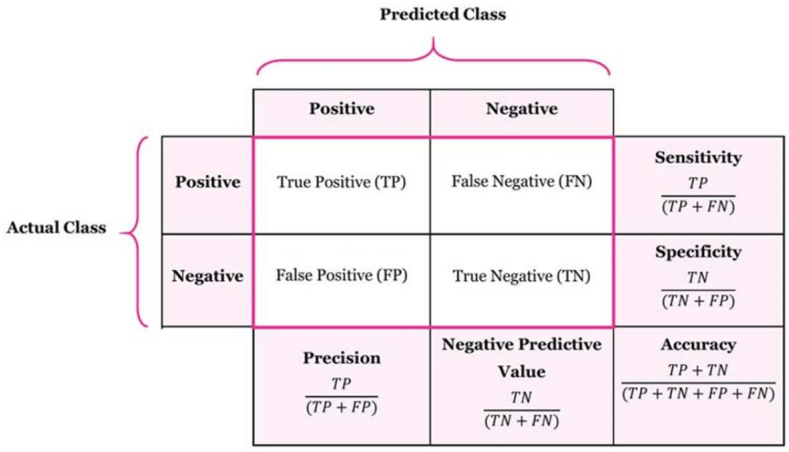
- An epoch is a measure of the number of times all training data are used once to update the parameters.
- **example**: 1000 images for training $\rightarrow$ uf $batchsize = 10$ then 100 $iterations = 1$ $epoch$

2.7.3 Learning Rate

- The learning rate determines the size of the "step" that an optimizer takes during each iteration.
- **Larger step size** = make a bigger change in the parameters (weights) in each iteration.
- If learning rate **too small**: parameters don't change very much in each iteration; takes a long time to train the network
- If learning rate **too large:** noisy; average loss might not change very much as batch size grows; very large can be detrimental to neural network training
- Appropriate learning rate depends on
  - The learning problem
  - The optimizer
  - The batch size
    - Large batch size allows for larger learning rates.
    - Small batch size requires a smaller learning rate.
  - The stage of training
- Reduce learning rate as training progresses

2.8 Debugging

- Make sure that your network can overfit to a small dataset; ensures that you are using the right variable names, and rules out other programming bugs that are difficult to discern from architecture issues
- Create a confusion matrix to provides visualization of the classification performance over each class (ground truth label)



2.9 Multiclass Classification

- Requires minor changes to our PyTorch implementation:
  1. The final output layer has as many neurons as classes.
  2. Apply the softmax activation function on the final layer to obtain class probabilities
  3. Use the multiclass cross-entropy loss function
- Softmax function calculates the probabilities distribution of the event over 'k' different events.
  - Calculates the probabilities class over all possible classes.
  - The range will be 0 to 1, and the sum of all the probabilities will be equal to one.

$$\sigma(z)_j = \frac{e_j^z}{\sum_{k=1}^{K} e_k^z} \qquad \text{for } j = 1, \cdots, k$$

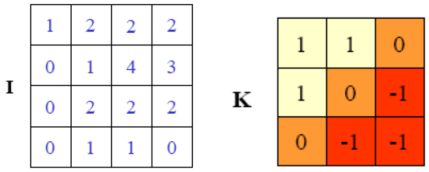
3 Convolutional Neural Networks (CNNs)

3.1 Large Networks

- Using a large fully connected layer could be problematic
  - Computing predictions (forward pass) will take longer
  - A large number of weights requires a lot of training data to avoid overfitting (exponential relationship)
  - Small shift in image can result in large change in prediction (each pixel would be multiplied by a different weight)
  - Does not make use of the geometry of the image

3.2 Convolutional Filters

- People have used the idea of convolutional filters in computer vision even before the rise of machine learning.
- They hand-coded filters that can detect simple features, for example to detect edges in various orientations.
- Computing the convolution of image I with filter K
  1. Flip rows and columns of kernel
  2. Multiply each pixel in range of kernel by the corresponding element of flipped kernel, sum all these products and write to a new 2d array; essentially a dot product
  3. Slide kernel across all areas of the image until you reach the ends; optional, can use zero padding to maintain size of output

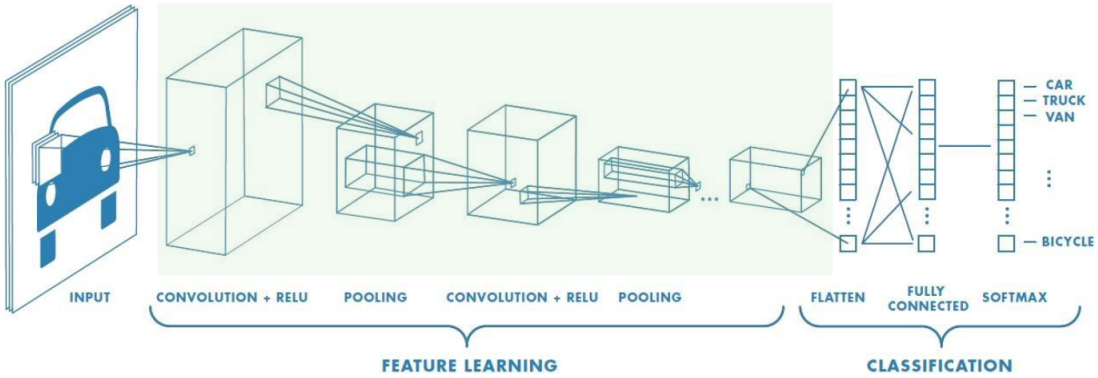


3.3 Convolutional Neural Networks

- We can introduce convolutional filters into our neural network so that we don't have to hand craft the features
- Locally-connected layers: look for local features in small regions of the image

- Weight-sharing: detect the same local features across the entire image
- Neural Network learns the kernel values (or weights)
- Two types of layers: **convolution** and **pooling**
- Can express the output dimension as

$$n_{i+1} = \frac{n_i + 2p - f}{s} + 1$$



3.4 Pooling Layers

- Dimensionality reduction (downsampling) is achieved using pooling layers
- Performed independently along the depth of a tensor
- Max pooling is the most common choice for pooling
- No parameters to learn!

3.5 Deeper CNN Architectures

- **LeNet**
  - First introduced by Yann LeCun in 1989
  - 7 layers
  - 60,000 parameters to learn
- **AlexNet**
  - Won ImageNet competition in 2012 with 15.3 % error rate
  - 60 million parameters
  - Over 90 epochs and 6 days of training

  - Can be implemented with transfer learning

- **Inception**
  - Won ImageNet competition in 2014 with 6.67 % error rate (close to human level performance)
  - 22 layer deep
  - Parameters reduced from 60 million to 4 million

- **VGG**
  - Runner up at ImageNet competition in 2014
  - Initially 16 convolutional layers, then 19
  - 138 million parameters which can be challenging to handle

- **ResNet**
  - Won ImageNet competition in 2015 with 3.57 % error rate (better than human level performance)
  - 152 layers

3.6 CNN Limitations

- Dependent on data
  - Scale invariance
  - Rotation invariance
  - Translation invariance

- They do not code position and orientation into their predictions
  - The final label of an image or the final prediction of the CNN is position invariant.
  - CNN should be able to identify the image as being small, large or upside down and it should have an internal representation of the position of that face.
  - Take an image and create many versions of it by tilting it, inverting it, rotating it and so on (**Image Augmentation**)
  - This helped CNNs to learn an internal representation of an image with many different viewpoints.

- Point of reference is not tracked
- Hyperparameter tuning is not trivial
- Convolution is computationally expensive
- Require large datasets

4 Autoencoders

4.1 Unsupervised Learning

- There are some challenges with supervised learning
  - Requires large amounts of labeled data
  - Obtaining labeled data is expensive
  - Often there is a lot more unlabeled data than labeled.
  - Not what we see in biology
- Our brains are constantly observing the world around us for patterns, or some structure to relate objects.
- Patterns or clusters of similar features can tell us a great deal about the data before we even have a label.

4.2 Autoencoder Overview

- Find efficient representations of input data that could be used to reconstruct the original images.
- Composed of two parts:
  - **Encoder** converts the inputs to an internal representation (dimensionality reduction)
  - **Decoder** converts the internal representation to the outputs (generative network)
- Applications include:
  - Feature Extraction
  - Unsupervised Pretraining
  - Dimensionality Reduction
  - Generate new data
  - Anomaly detection

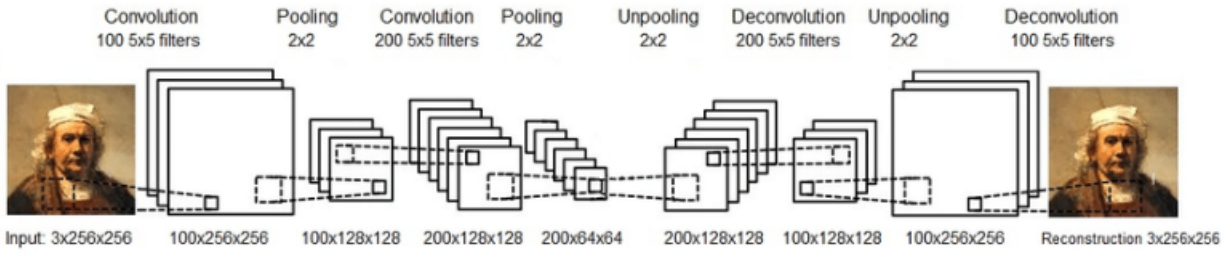
4.3 Autoencoder Architecture

- Typically the number of outputs is the same as the inputs
- Hour glass shape creating a bottleneck layer or lower dimensional representation
- Not used for supervised learning. The task is not to predict something about the data!
- It is forced to learn the most important features in the input data and drop the unimportant ones

- Potential for Overfitting!
  - If encoder is too deep it could map each input to a single arbitrary number
  - Such an encoder would reconstruct the training data perfectly and would not have learned any useful representations
  - Unlikely to generalize to new instances.
- One way to ensure that an autoencoder is properly trained is to compare the inputs and the outputs.
- Noise can be added to the input images of the autoencoder to force it to learn useful features; prevents the autoencoder from trivially copying its inputs to its outputs, has to find patterns in the data.

4.4 Convolutional Autoencoder

- The autoencoders mentioned up until now did not take advantage of spatial information
- Adding convolutional layers can add improvements when trying to generate image data
- The encoder works with convolution layers (pooling layers optional).
- The decoder tries to mirror the encoder but instead of "making everything smaller" it has the goal of "making everything bigger" to match the original size of the image — use **Transposed Convolution**:
  1. Take each pixel of your input image
  2. Multiply each value of your kernel with the input pixel to get a weighted kernel
  3. Insert this in your output to create an image
  4. Sum the outputs where they overlap



5 Recurrent Neural Networks (RNNs)

5.1 One-Hot Encoding

- For word (or categorical data) where there is no order, integer encoding is not enough.
- A better way... convert word features into numerical features with one-hot encoding.
- Some advantages/disadvantages
  - Dimensions grow with number of words
  - One-hot encoding assumes each word is completely independent

5.2 Word Embeddings

In order to talk about which words have "similar" GloVe embeddings, We need to introduce a measure of distance in the embedding space.

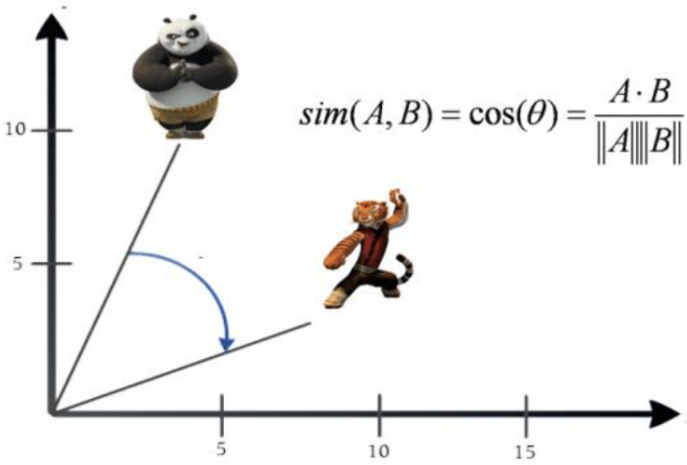
5.2.1 Euclidean Distance

The Euclidean distance of two vectors $x = [x_1, x_2, \cdots, x_n]$ and $y = [y_1, y_2, \cdots, y_n]$ is the 2-norm of their difference $x - y$:

$$\sqrt{\sum_i (x_i - y_i)^2}$$

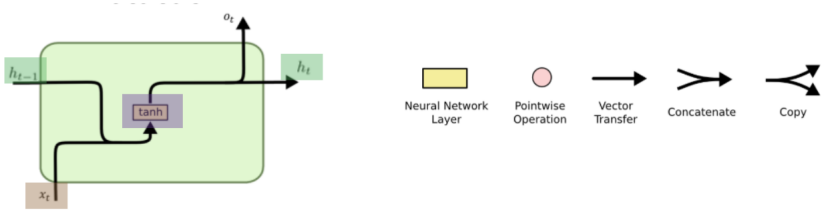
5.2.2 Cosine Similarity

The cosine similarity of two vectors x and y is the cosine of the angle between the two vectors. Cosine similarity is useful when we want a distance measure that is invariant to the magnitude of the vectors.



5.3 RNN Architecture

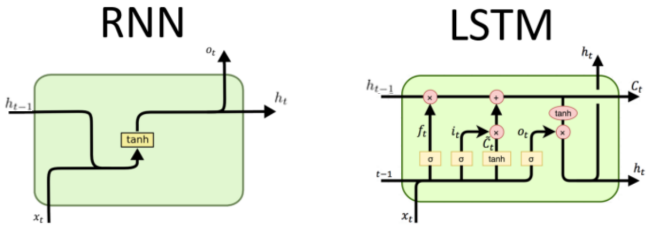
- Can take in variable-sized sequential input
- Can remember things over time, or has some sort of memory or state
- General procedure :
  1. Start with an initial hidden state with a blank slate (can be a vector of all zeros)
  2. Hidden state is updated based on previous hidden state, and the input using the same neural network as before (weight sharing)
  3. Continue updating the hidden state until we run out of tokens.
  4. Use the last hidden state as input to a prediction network
- The forward path in RNNs shares similarities with the a fully-connected neural network
- Weights applied on input and hidden state

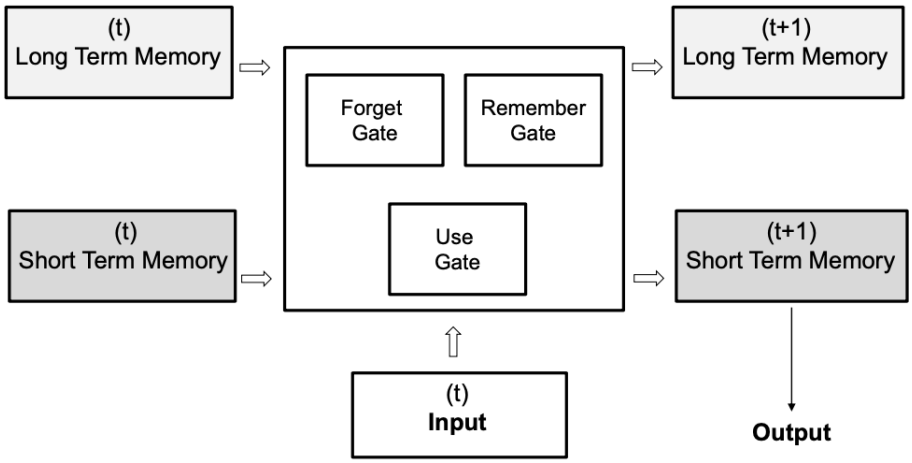


hidden(t) = tanh(weight_hidden*hidden(t-1) + weight_input*(input(t))

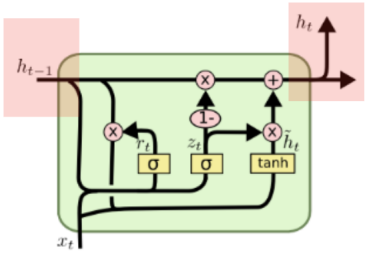
5.4 Long Term Dependencies

- RNNs historically hard to train; good for capturing recent information, but difficulties with long-term dependencies
- **Long Short-Term Memory (LSTM) networks**
  - take advantage of an extra hidden state referred to as the cell state
  - gates to manage long-term memory and regulate vanishing/exploding gradients
  - adds more complexity





- **Gated Recurrent Units (GRUs)**
  - Variant on LSTM
  - Introduced in 2014, it combines gates and merges cell state and hidden state
  - It is much simpler to implement and faster than standard LSTM



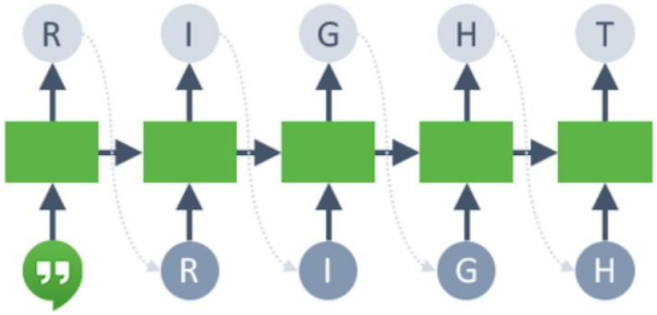
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

5.5 Generative RNNs

- Learning to generate new sequences will require some changes:
  - Input sequence representation — start and end token
  - Training-time behaviour — teacher-forcing
  - Test-time behaviour — sampling and temperature
- **RNN For Prediction:**
  - Process tokens one at a time
  - Hidden state is a representation of all the tokens read thus far
- **RNN For Generation:**

  - Generate tokens one at a time
  - Hidden state is a representation of all the tokens to be generated

- Process:
  - $token = preprocess(input)$
  - $hidden = update\_function(hidden, token)$
  - $token\_distribution = prediction\_function(hidden)$
  - $token = sample\_from(token_distribution)$



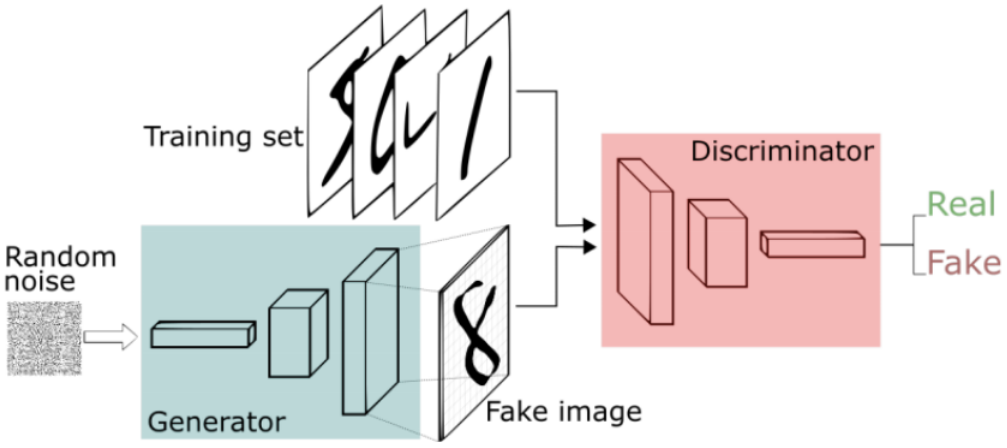
6 Generative Adversarial Networks (GANs)

6.1 Generative Models

- Learns the structure of a set of input data, and can be used to generate new data
- **examples**: Autoencoders, RNNs for text generation

6.2 GAN Architecture

- The idea is to train two networks
  - **Generator Network**: try to fool the discriminator by generating real-looking images
    - Input: A random noise vector
    - Output: A generated image
  - **Discriminator Network**: try to distinguish between real and fake images
    - Input: An image
    - Output: A binary label (real or fake)
- The loss function of the generator (the model we care about) is defined by the discriminator!
- Applies a minimax strategy
  - the discriminator will try to do the best job it can
  - the generator is set to make the discriminator as wrong as possible



6.3 Loss Function

- Tune **discriminator** weights to maximize the probability that the ...
  - discriminator labels a real image as real
  - discriminator labels a generated image as fake
- Tune **generator** weights to maximize the probability that the ...
  - discriminator labels a generated image as real

6.4 Adversarial Attack

- **Goal** - choose a small perturbation ϵ on an image x so that a neural network f misclassifies $x + \epsilon$
- **Approach** - use the same optimization process to choose ϵ to minimize the probability that

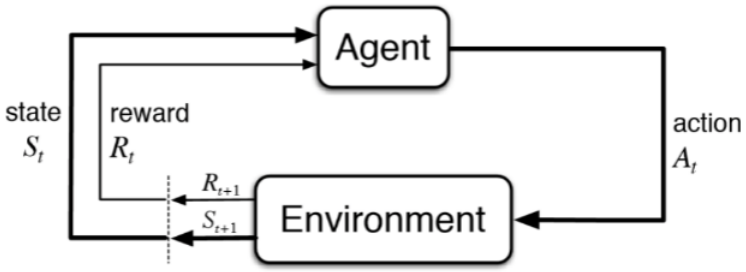
$$f(x + \epsilon) = correct\ class$$

- Treating ϵ as the parameters
- **Non-Targeted Attack** - minimize the probability that $f(x + \epsilon) = correct\ class$
- **Targeted Attack** - maximize the probability that $f(x + \epsilon) = targeted\ class$
- **White Box** - assumes the model is known, we need to know the architectures and weights of f to optimize ϵ
- **Black Box** - don't know the architectures and weights of f to optimize ϵ; substitute model mimicking target model with known, differentiable function

7 Reinforcement Learning

7.1 Overview

- Reinforcement learning is different from supervised learning
  - There is no supervision - only a scalar reward signal
  - There is a notion of "time" (steps/moves)
  - Feedback is delayed, not instantaneous
  - Decisions/actions will affect the subsequent data received
- There are two key components
  - **Environment** - provides the agent with the current state and provides a reward at each time step (mostly 0 reward)
  - **Agent** - chooses an action at each time step, given the state



7.2 Rewards

- A **reward** R_t is a scalar feedback signal
- The goal of the agent is to select actions to maximize total future reward.
- Setting up rewards to get good agent behaviour is tricky, If the reward is not well designed, it can lead to reward hacking
- Two types of rewards:
  - **Sparse Reward** - there is a nonzero reward in only a few time steps (*i.e. win/loss at the end*)
  - **Dense Reward** - there is a nonzero reward in most time steps (*i.e. some form of score*)

7.3 Major Components

7.3.1 Agent

- algorithm that makes the decision on what action to take
- can contain machine learning algorithms that learn one or more of the policy, value function, or model.
- two main types:
  - **Value Based Agent** - learn value function only; choose the action that maximize the value function of the next state
  - **Policy Based Agent** - learn policy function only (no estimate of value function)

7.3.2 Policy

- the agent's behaviour, the action it chooses at a state.
- usually represented with the letter π
- a function that maps state to action
  - **deterministic:** $a = \pi(s)$
  - **stochastic**: $\pi(a|s) = P(A_t = a|S_t = s)$
- can parameterize π using a neural network

7.3.3 Value Function

- a prediction of future reward, assuming we follow a particular policy
- used to evaluate the goodness or badness of states
- the value function is a function that maps state to expected discounted return

7.3.4 Q-Function

- maps (state, action) pairs to expected discounted return
- can parameterize the Q-function using a neural network

7.3.5 Model

- predicts what the environment will do next
  - predict the next state (given the current state and an action to take)
  - predict the next reward

7.4 Value Iteration Algorithm

1. For all s set $V(s) = 0$
2. Repeat until convergence:
   (a) For all actions, a , determine and set $Q(s, a)$
   (b) $V(s) = \max_a Q(s, a)$
3. Return Q

$$Q(s, a) = \sum_s T(s, a, s') \cdot [R(s, a, s') + \gamma \cdot V(s')]$$

- $T(s, a, s')$ is the transition probability from state s to state s', given that the agent chose action a
- $R(s, a, s')$ is the immediate reward that the agent gets when it goes from state s to state s', given that the agent chose action a
- γ is the discount rate

7.5 Q-Learning Algorithm

1. For all s set $Q(s, a) = 0$, and $V(s) = 0$
2. Repeat until convergence:
   (a) Randomly select state s and randomly select action a from available actions at state s
   (b) If immediate reward available update matrix $R(s, a)$
   (c) Update $Q(s, a)$
   (d) $V(s) = \max_a Q(s, a)$
3. Return Q

$$Q(s, a) = (1 - \alpha) \cdot Q(s, a) + \alpha \cdot [R(s, a, s') + \gamma \cdot V(s')]$$

7.6 Exploration vs Exploitation

- **Exploitation**: Take actions the function already thinks will lead to a good outcome
- **Exploration**: Try making novel actions and see if you discover a way to adjust the function to get even better outcomes

8 Ethics and Fairness

8.1 What is Ethics?

- Ethics deals with moral principles that govern a person's behavior or the conducting of an activity
- Ethics in AI is primarily focused on social, political and cultural impact of the application of AI software

8.2 Convergence of Ethical Principles

- Transparency
- Justice and Fairness
- Non-Maleficence
- Responsibility
- Privacy

8.3 Causes of Bias

- **Skewed Sample:** If by chance bias is introduced,then such bias may compound over time: future observations confirm prediction and fewer opportunity to make observations that contradict prediction.
- **Tainted Examples:** Human bias can be captured by the model.
- **Limited Features:** Minority groups data may have less information, or unreliably collected data. System tends to have much lower accuracy for the prediction of the minority group.
- **Sample Size Disparity:** When there is much less training data coming from the minority group than those from the majority group. Model has less information to accurately model the minority group.
- **Proxies:** Excluding sensitive attributes such as race/gender may not be enough. Other features such as neighbourhood could act as proxies of sensitive attribute. Can be difficult to decide if we should include feature or not.

8.4 Defining Fairness

8.4.1 Disparate Treatment

- Model suffers from disparate treatment if decisionsare correlated with the subject's sensitive attribute.

- For example, in the sentencing model, does the model treat people of different ethnicities similarly?

8.4.2 Disparate Impact

- Model suffers from disparate impact if decisions disproportionally hurt (or benefit) people with sensitive attributes
- For example, suppose we are building a model to determine whether or not an applicant is admitted to graduate school.

8.5 Measuring Fairness

8.5.1 Demographic Parity

- Acceptance ratesof applications from both groups must be equal
- **Problem**: Fairness is measured at a group level; model can hire qualified people from one group, and random people from the other

8.5.2 Equalized Odds

- Model should be equally accurate across both groups
- Also known as "accuracy parity"
- **Problem**: False positives and false negatives have different impacts; does not help to close the gap between the two groups

8.5.3 Individual Fairness

- Similar individuals from different groups should be treated similarly
- **Problem**: Hard to determine appropriate measure of "similarity" of inputs

8.6 Increasing Fairness

- **Pre-processing:** remove information correlated to sensitive attributes
- **Add regularization term:** add a "fairness" regularizer
- **Post-processing:** change the way we use a model to make predictions