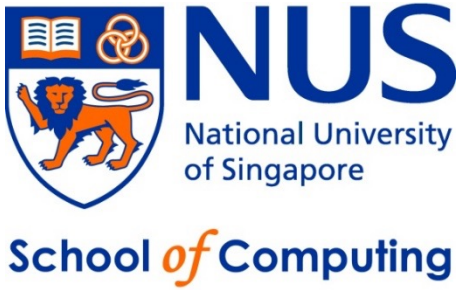
SWS3009A

Robotics and Deep Learning

Deep Learning Trial Lecture

colintan@nus.edu.sg

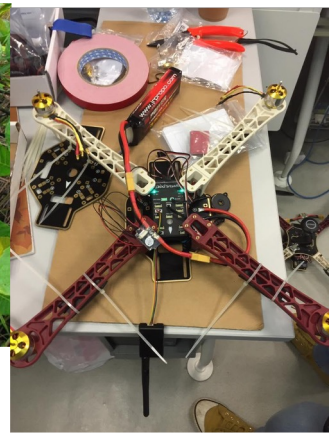


About Me

Colin Tan

Department of Computer Science, NUS School of Computing.

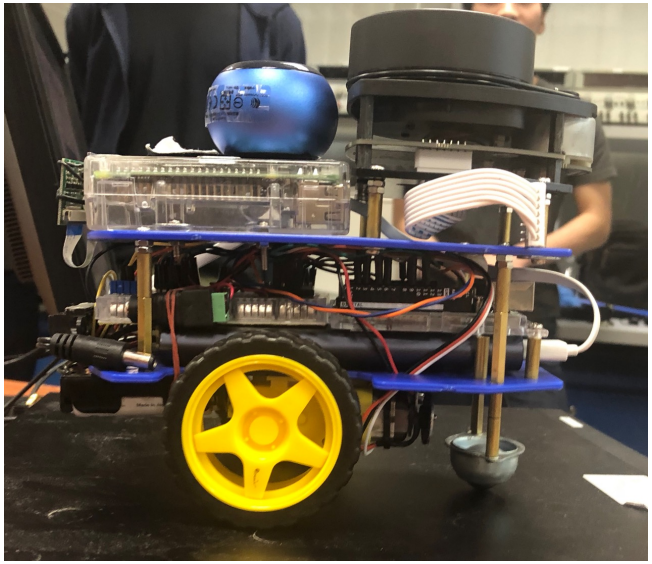
Email: colintan@nus.edu.sg



NUS National University of Singapore
School of Computing

Our Motivation

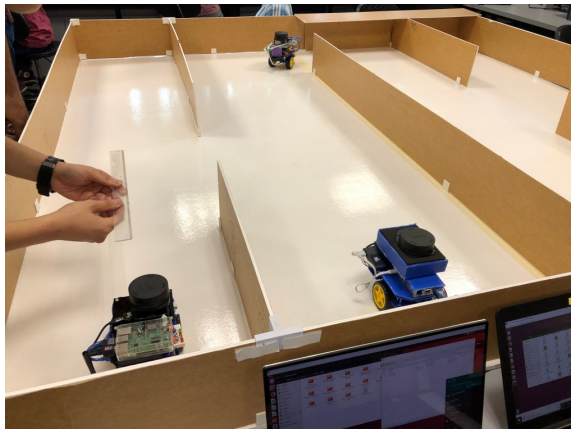
- **ChatGPT is (sort of) awesome!**
  - GPT4 has digested almost all the text in the universe! (I.e. what's available on the internet)
  - But it's just text; life is more than text!
- **Our goal then is to integrate AI with sensors and actuators.**
  - What could be more fun than working with robots?
- **Our methodology:**
  - Two tracks – One on robotics and one on AI.
  - Two projects:
    - ✓**A baseline project to ensure you've gotten all the basic concepts down.**
    - ✓**A project that YOU dream up.**

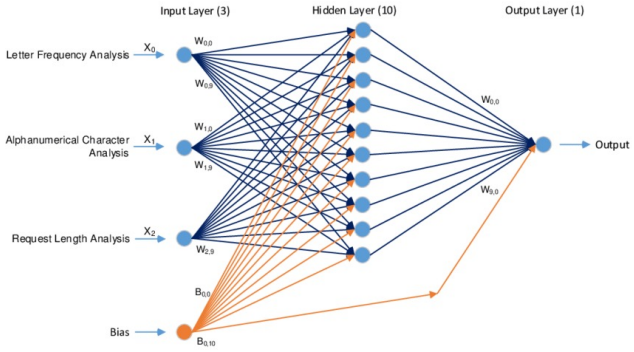


NUS National University of Singapore
School of Computing

What You Will Learn (Quick Summary):

- **Boyd's Part: Robotics**
  - The hardware aspects of building a robot
    - ✓**Connecting and programming robot sensors.**
    - ✓**Controlling the robot remotely**
  - Hardware security.
- **My Part: The "brain" of the robot**
  - Machine learning:
    - ✓**Statistical Methods.**
    - ✓**Neural Networks.**
    - ✓**Deep Learning Networks.**
  - Communications.
    - ✓**Reverse tunnels:**
      - *–Control your robot from any part of the world*
    - ✓**Transport layer security**





Course Structure

- **Trial Lecture Only: You will be attending both my part and Boyd's part.**
  - Decide whether you want to do hardware or deep learning.
    - ✓**You will be doing this topic throughout the workshop.**
  - You are STRONGLY encouraged to take the topic you are not familiar with:
    - ✓**Take hardware if you're a deep learning person.**
    - ✓**Take deep learning if you're a hardware person.**
    - ✓**If you are neither, take the part that excites you most.**
  - This way you can maximize what you get from this Summer Workshop.

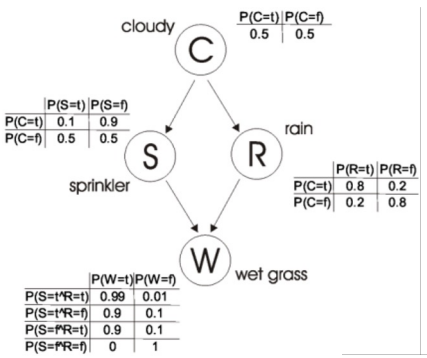
NUS National University of Singapore
School of Computing

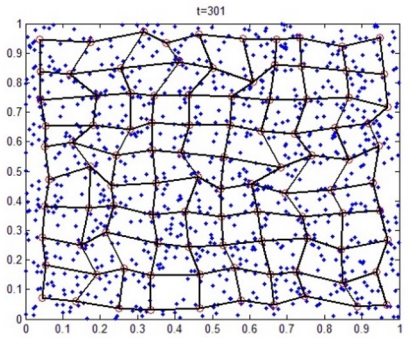
Course Structure

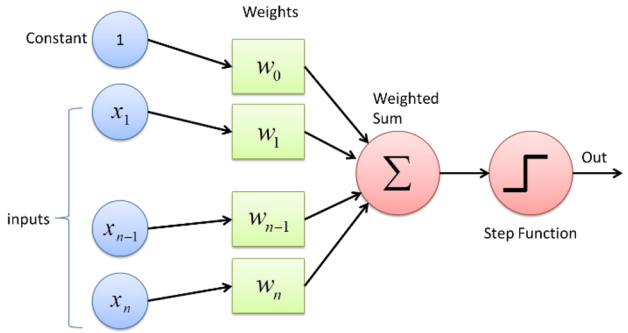
- **Structure:**
  - You will be organized into teams:
    - ✓**Hopefully 2 hardware, 2 deep learning per team.**
  - Hardware and deep learning groups will have separate lectures.
    - ✓**In Phase 2 you will attend only ONE track of lectures, not both.**
  - You will be working together (hardware + deep learning) on a baseline project.
    - ✓**Build a robot to explore a maze and find targets.**
  - After the baseline project, you will work on your own project.
    - ✓**We will be looking at several very interesting deep learning models:**
      - *–Convolutional networks to recognize objects.*
      - *–Generative networks to produce new data from old.*
      - *–Object detection networks to track objects.*
      - *–LSTM and Transformers for understanding time-series data.*
      - *–Auto-encoders to detect problems.*

What Is Machine Learning?

- In machine learning:
  - ✓**We present the machine with A LOT of data.**
  - ✓**Machine learns on its own to recognize patterns in the data.**







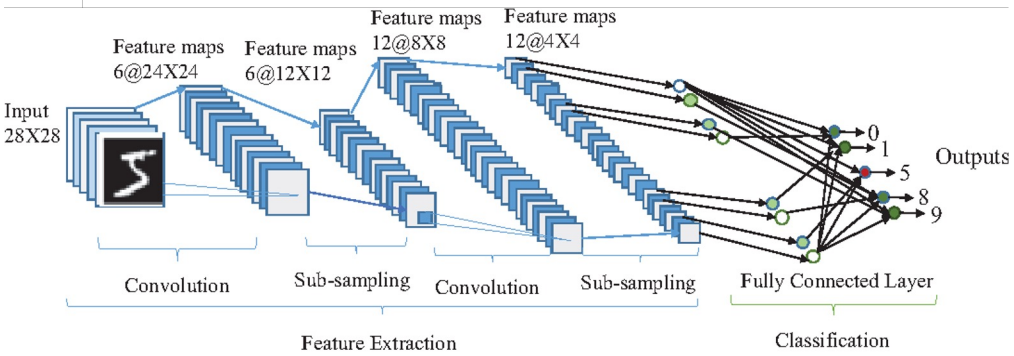
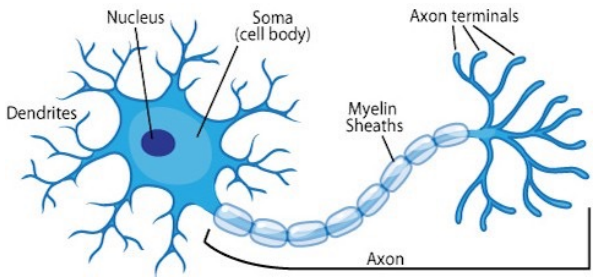
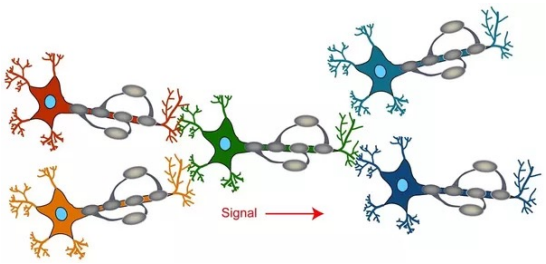


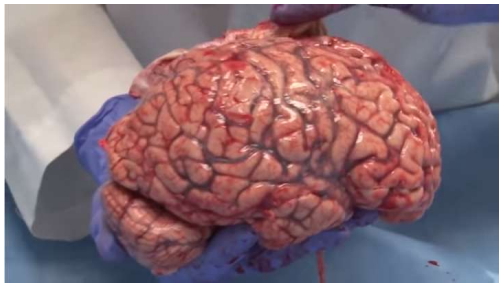
Fig. 1. CNN Block Diagram

Neural Networks: Biological Inspiration

- **Neural networks are inspired by biological neurons: Units of computation that are in our brain and nervous system:**
  - Each neuron takes electrical signals from surrounding neurons.
  - Signals pass through gaps called "synapses" that pass a portion of the signal through.
    - ✓**"Strong" synapses pass more signals, "weak" synapses pass weaker signals.**
  - The neuron sums these signals and fire if they exceed a "threshold".
  - To lean, the synapses are strengthened or weakened.

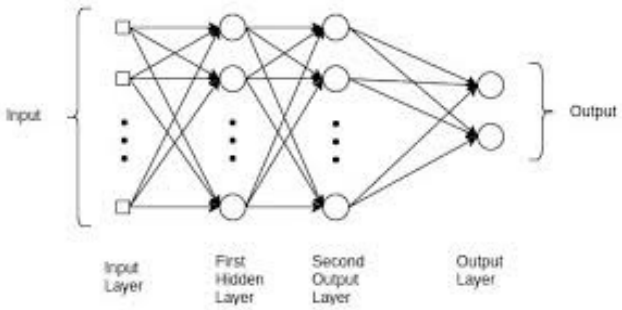


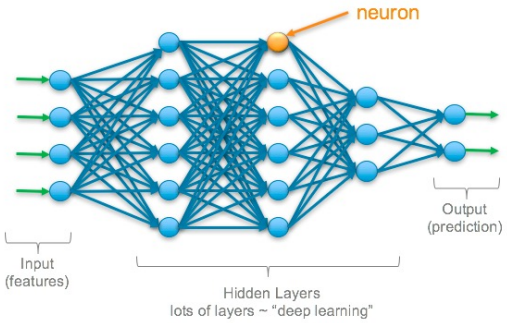


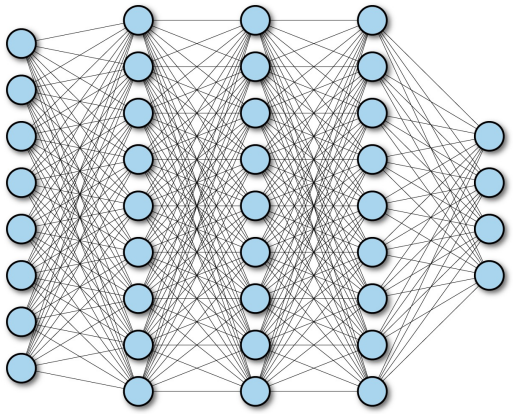


Deep Learning

- **Primary limitation of neural networks:**
  - Neural networks learn by adjusting "parameters" according to a "learning law" (We will cover learning laws later), based on the data presented.
  - Need large number of "parameters" to learn highly complex tasks.
    - ✓**E.g. image classification**
  - Curse of dimensionality:
    - ✓**As the number of parameters increases, the amount of training data required grows exponentially.**
    - ✓**Otherwise the neural network "overfits" – It memorizes the data it is presented instead of learning from it.**

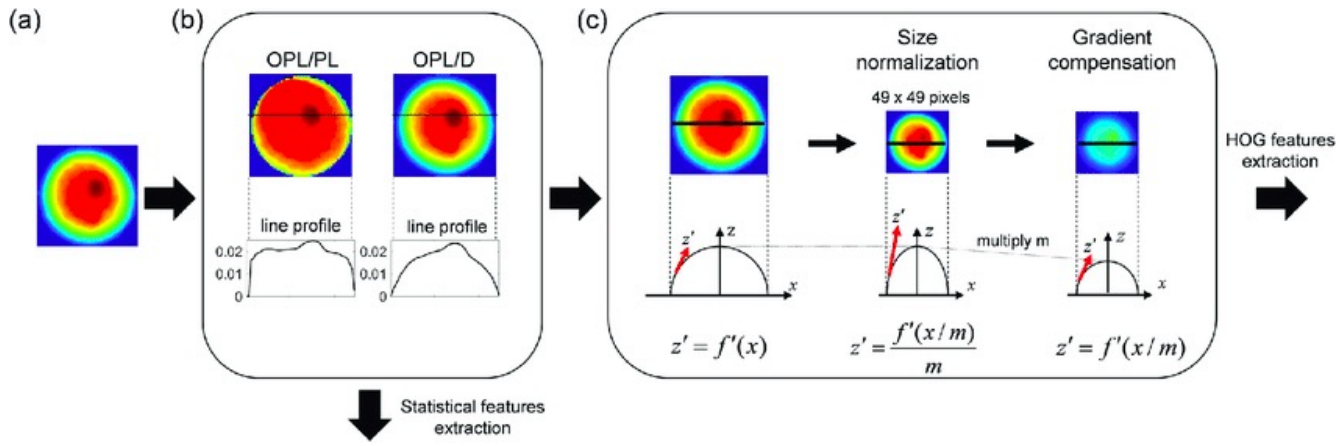






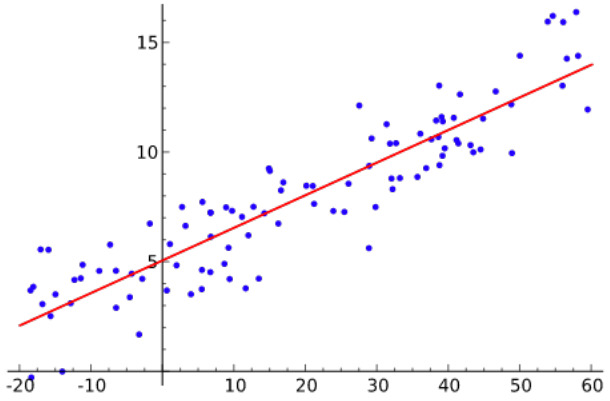
Deep Learning

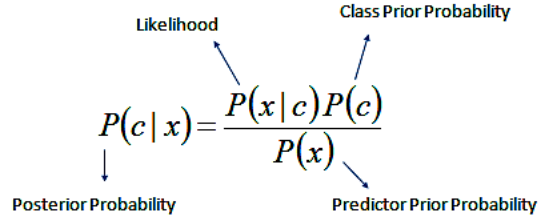
- In Deep Learning we apply various techniques to "simplify" the data:
  - Extract features.
  - Make the data invariant:
    - ✓Shift invariance.
    - ✓Scale invariance.



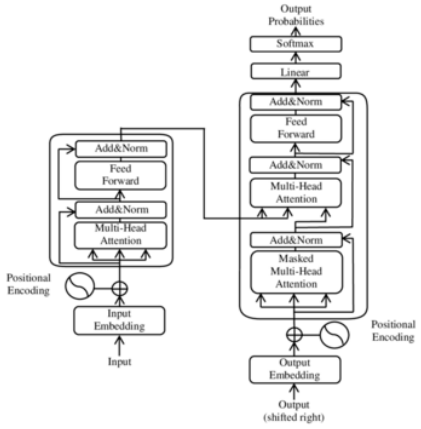
What We Will Be Doing

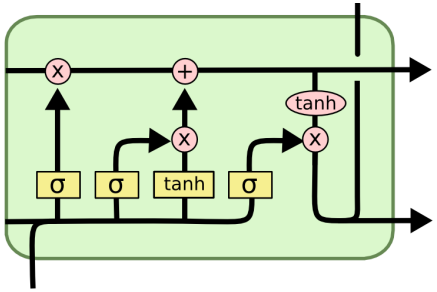
- **We will now go through all the different algorithms we will be looking at.**
  - I will only give a quick summary. Read through the list of topics yourself.
  - Think about what you want to do for your project.





$$P(c \mid \mathbf{X}) = P(x_1 \mid c) \times P(x_2 \mid c) \times \cdots \times P(x_n \mid c) \times P(c)$$

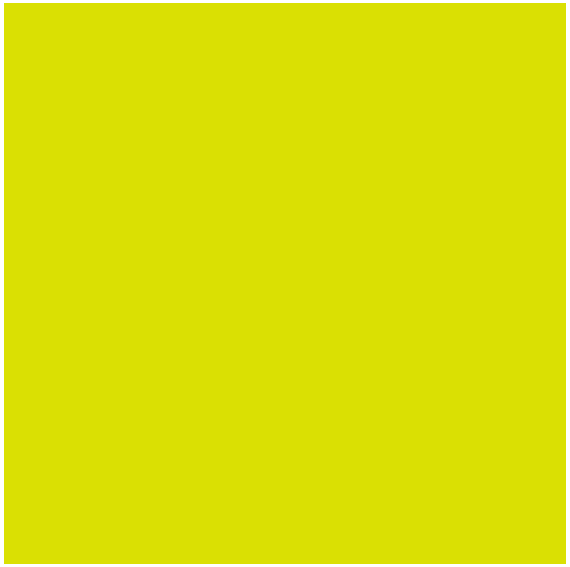




NUS National University of Singapore
School of Computing

What We Will Be Doing (Read this list yourself)

- **Connectionist Models:**
  - Unsupervised Learning:
    - ✓**Machine looks at data, and figures how on its own how to sort out the data into classes.**
    - ✓**Good for "clustering" – Is this colour green, yellow, or greenish-yellow?**
    - ✓**Algorithms:**
      - *–k-Means Clustering*
      - *–Kohonen Self Organizing Maps*
    - ✓**Kohonen Self Organizing Map can be automatically trained to sort out colours to do colour classification.**
    - ✓**No need for tagged data.**



NUS National University of Singapore
School of Computing

What We Will Be Doing (Read this list yourself)

- **Connectionist Models:**
  - Supervised Learning:
    - ✓**These are great when you have enough "tagged" data to teach the computer.**
    - ✓**Creation / acquisition of tagged data can be extremely expensive.**
    - ✓**Techniques like Generative Adverserial Networks (GANs) can produce "fake" data that looks like real data, to augment real data sets.**
    - ✓**Algorithms:**
      - *–Perceptrons (Grandfather of deep learning)*
      - *–Multilayer Perceptrons*

NUS National University of Singapore
School of Computing

What We Will Be Doing (Read this list yourself)

- **Deep Learning Models:**
  - Recurrent Neural Networks
    - ✓**Great for learning patterns across time.**
  - Long-Short Term Memories
    - ✓**Like RNNs but much more robust.**
  - Transformers
    - ✓**Able to learn sequences in parallel.**
    - ✓**Attention mechanisms**
  - Convolutional Neural Networks
    - ✓**Most commonly used for images, but can be used elsewhere.**
    - ✓**Key strength are its convolutional layers (detects key features) and pooling layers (summarizes features / removes noise).**
  - Object Detection Networks
    - ✓**Detect, recognize and track objects in real-time.**

What We Will Be Doing (Read this list yourself)

- Generative Adversarial Networks
  - ✓**Forgers and police.**
  - ✓**Forgers fake data, police try to catch them. Forgers improve their fakery to avoid getting caught.**
  - ✓**Used to generate more data from existing samples of data.**
- Autoencoders
  - ✓**Train neural networks on this function f(x) = g(x).**
  - ✓**Why?**
    - *–Assume that g(x) is the output of some system, e.g. torque from an engine, and x is the input to the engine, e.g. throttle position.*
    - *–When f(.) is fully trained, we will get $|f(x) – g(x)| < \varepsilon$ where ε is some very small value (f(x) is never exactly g(x) because of noise in g(x), e.g. variations in oxygen levels in the air taken in by the engine, variations in temperature, etc.)*
    - *–If at some point $|f(x) – g(x)| > k\varepsilon$ where k is some constant, we know that g(x) is producing abnormal values, possibly signalling a fault.*
  - ✓**Autoencoders are thus very good at detecting faults in a system.**

NUS National University of Singapore
School of Computing

What We Will Be Doing (Read this list yourself)

- **Using Machine Learning is HARD!! We will look at:**
  - Network sizing.
    - ✓**Kernel dimensions?**
    - ✓**# of layers?**
    - ✓**Pooling?**
    - ✓**# of neurons in each layer?**
  - Selection of training hyper-parameters:
    - ✓**Learning algorithms**
      - *–Stochastic Gradient Descent?*
      - *–Adam?*
      - *–AdaMax?*
      - *–???*
    - ✓**Learning parameters**
      - *–Learning Rate*
      - *–Decay rate*
      - *–Momentum*

NUS National University of Singapore
School of Computing

What We Will Be Doing (Read this list yourself)

- Selecting Transfer Functions
  - ✓**Identity?**
  - ✓**Sine, cosine?**
  - ✓**Sigmoid?**
  - ✓**ReLU, Leaky-ReLU?**
- Dealing with Overfitting
  - ✓**Dropouts?**
  - ✓**L1 or L2 regularizers? Or none? Or both?**
  - ✓**How much L1 and how much L2 to use?**
- Getting Data
  - ✓**Hunting for data.**
  - ✓**Cleaning up the data.**
  - ✓**Finding more data.**
  - ✓**DATA DATA DATA!**

NUS National University of Singapore
School of Computing

What We Will Be Doing (Read this list yourself)

- **Statistical Models:**
  - Convenient for many applications – fast, few hyperparameters, easy to train.
  - Linear Regression
    - ✓**Given a set of data, produce a function that can predict outcomes for unseen data.**
    - ✓**E.g. Given historical room temperature by time and by season, predict the level of air-conditioning/heating needed for tomorrow.**
  - Naïve Bayes Classifier
    - ✓**Model the statistical characteristics of the data.**
    - ✓**Classify new data based on these characteristics.**
    - ✓**Good for classifying data – E.g. given a series of temperature and light readings, what season are we in? (Autumn, spring?)**

NUS National University of Singapore
School of Computing

What We Will Be Doing (Read this list yourself)

- **Statistical Models:**
  - Decision Trees
    - ✓**Look at the data and infer how decisions were made.**
  - Support Vector Machines
    - ✓**Once viewed as the "saviour" of machine learning.**
    - ✓**Look at the data and create "hyperplanes" to separate different classes of data.**
    - ✓**Good for classifying data – More complex than Naïve Bayers but more robust.**

NUS National University of Singapore
School of Computing

SWS3009A Robotics and Deep Learning (Deep Learning)

- **Lectures will be a mix of theory and hands-on.**
  - Language used: Python 3 (Python 2 is not supported).
  - Environment:
    - ✓**Lecture hands-on using Jupyter.**
    - ✓**Labs: Any IDE you prefer. But real programmers use vim. ;)**
- **There will be 5 lectures, and a lot of hands-on.**
  - Covers key topics we don't do in the lecture:
    - ✓**Creating and training statistical, neural network and deep learning models.**
    - ✓**Generation of crypto keys.**
    - ✓**Reverse proxies to control your robot from any part of the world.**
- **This is a HEAVY but FUN course with a lot of work!**

NUS National University of Singapore
School of Computing

Installing the Course Environment (Intel Macbooks/Windows Machines Only)

- **You will need to install Python 3.x (preferably 3.7 or better). Python 2.x will NOT be supported.**
- **Once installed:**
  1. Create a course directory. E.g. sws3009cool.
  2. Change to that directory.
  3. Create a virtual environment called venv:

     ```
     python –m venv venv
     ```
  4. Activate the environment:

     ```
     source venv/bin/activate
     ```
  5. Install all requirements by executing (this is one command):

     ```
     pip3 install pandas pymongo tensorflow sklearn jupyter
     requests flask pillow matplotlib tqdm
     ```

  Note: To exit the virtual environment, enter the following command:

  ```
  deactivate
  ```

NUS National University of Singapore
School of Computing

Special Note:
Students with Apple M1 / M2 Macbooks

- **Please follow the instructions here:**

https://developer.apple.com/metal/tensorflow-plugin/

- **Once Tensorflow has been installed, install the rest of your software (this is one command):**

```
pip3 install pandas pymongo sklearn jupyter requests
flask pillow matplotlib tqdm
```

NUS National University of Singapore
School of Computing

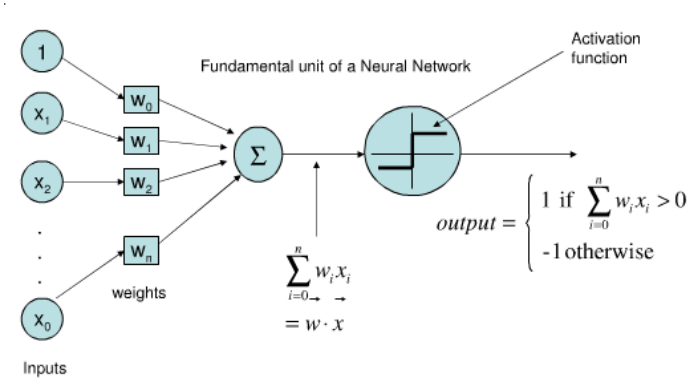
Neural Networks
Introduction

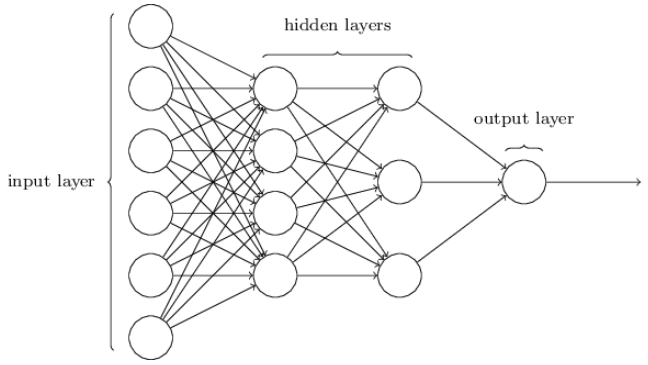
- **Now we will look at the basics of deep learning: Neural network learning laws.**
  - Unsupervised learning.
  - Supervised learning using gradient descent.
  - Problems with neural networks and how to solve them.
- **We will also be looking at some hands-on for programming neural networks using Keras (built into TensorFlow).**

INTRODUCTION TO LEARNING LAWS

Revision: Neural Networks

- **Recall: Neural networks are made of neurons.**
  - Like the biological neurons, the "neurons" here also sum inputs through weights.
  - The neuron "fires" (outputs a "1") if the sum exceeds a threshold, through an "activation function".
  - The summation is essentially a dot product $w^T x$, and the neuron can be specified by $g(w^T x)$ where $g(.)$ is the "activation function".





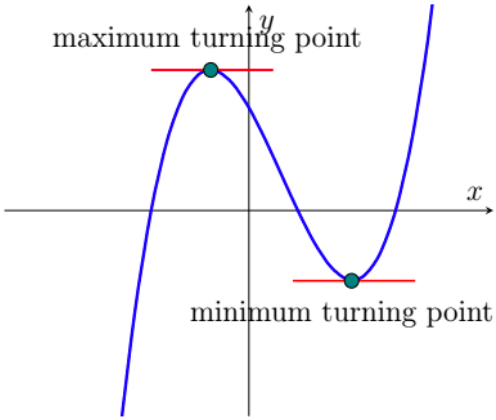
NUS National University of Singapore
School of Computing

Revision: Neural Networks

- **The chief problem in neural networks is how to fix the weights w_0 to w_{n-1}.**
  - Similar problem to linear regression: We need to find weight values that minimize an error function E.
  - Mathematically we want we want to find a set of weights W such that:

$$\frac{dE}{dW} = 0$$

  - As we shall see, unlike Linear Regression, this problem has no analytical solutions.
- **We will instead look at a class of numerical solutions called "Learning Laws".**
  - Note that $\frac{dE}{dW} = 0$ does not guarantee a minima: It also occurs at maxima.
  - The algorithm has to be designed to guarantee only minimum solutions.



Learning Laws

- **Two classes of learning laws:**
  - Unsupervised learning:
    - ✓**Tons of data is thrown at the NN.**
    - ✓**The NN does its own inference on the structure and relationships in the data.**
    - ✓**Due to lack of time, we will not look at unsupervised learning. Materials are provided for your own self-learning.**
  - Supervised learning:
    - ✓**We give the NN the data and the correct labels (or values we want to produce) from the data.**
    - ✓**The NN then optimizes its weights to mimic the generator function for the data, based on what we tell it.**

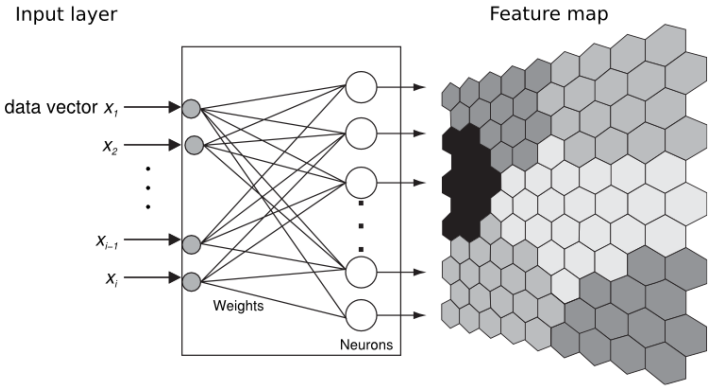
UNSUPERVISED LEARNING

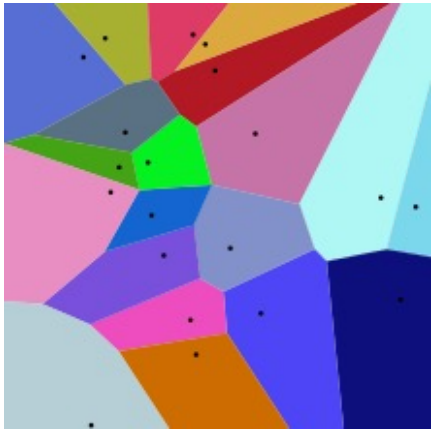
Unsupervised Learning

- **In unsupervised learning, the neural network is given data samples, but no descriptions of what the data samples mean.**
- **The job of the neural network is then to automatically learn relationships and structure between the data samples.**
- **There are several such networks:**
  - Hebbian Learning: Neural networks that strengthen or weaken connections between neurons based on sample data.
  - Adverserial Learning: Two neural networks that compete with each other based on sample data.
  - Self Organizing Maps: A neural network that partitions the vector space based on sample data – "Clustering".
- **In our lecture we will only look at self organizing maps.**

Unsupervised Learning
Self Organizing Maps

- **The Self Organizing Map (SOM) is based on the following principles:**
  - Given a sample vector v_k,
  - The neuron (which we will call a "centroid") that best matches v_k (bmu) AND
  - Its nearest neighbors
  - Are adjusted to look like v_k. This partitions the vector space into a "tesellated Voronoi surface."
- **We begin first with a set of *m* centroids of dimension *d*:**
  - We usually use values of between 0 and 1.
  - INITIAL VALUES DO AFFECT OUTCOME!
    - ✓**Can optimize performance by carefully choosing initial values.**





NUS National University of Singapore
School of Computing

Unsupervised Learning
The Kohonen SOM Algorithm

- **Given a sample vector v_k** (Input vector v_k and centroids n_i are all of dimension d)**:**
  - Find the nearest neuron (called a "best-matching-unit" or bmu) n_{bmu} using:

$$n_{bmu} = argmin_{1 \le i \le m} \|v_k - n_i\|$$

  - Update every centroid n_i:

$$n_i = n_i + \alpha(t)h(i - bmu)(v_k - n_i)$$

Where:

$$h(t) = e^{-\frac{\|t\|^2}{2w(t)}}$$

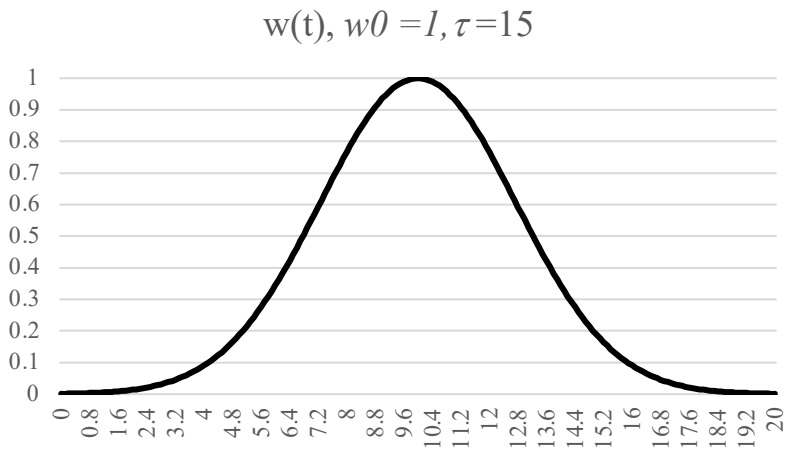
  - h is known as the "neighborhood function".

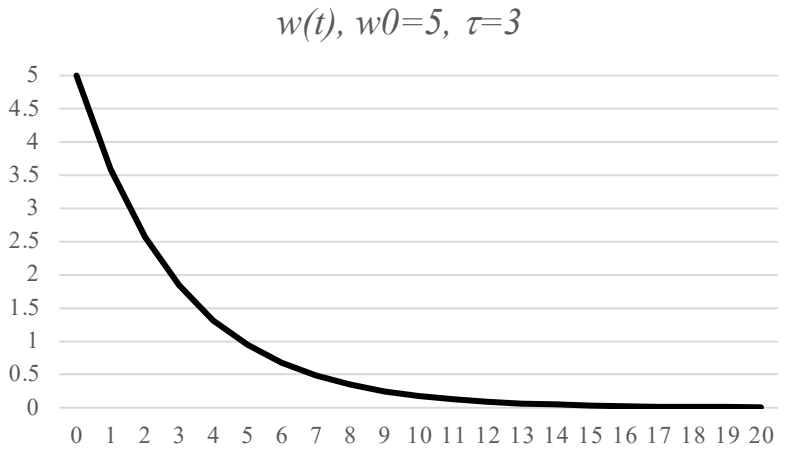
Unsupervised Learning
The Kohonen SOM Algorithm

- **The neighbourhood function *h* is 1 at bmu, and gets smaller around neighbors.**
  - Maximal change to bmu, reduced changes to neighbors.
- **The parameter "w" in the neighbourhood function controls the width of the "hat".**
  - This is progressively decayed to reduce influence of new samples on the neighbors, but it will always be 1 at bmu.

$$w(t) = w_0 e^{-\frac{t}{\tau}}$$

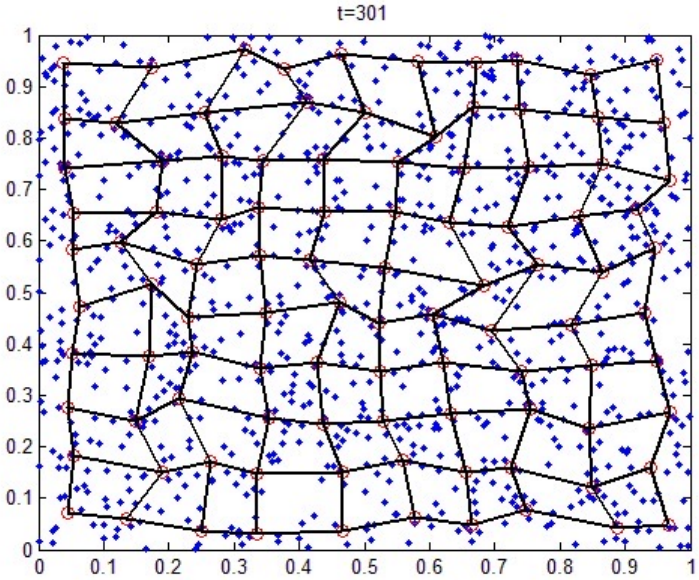
where w_0 is the initial width, and τ is a parameter to control the decay of $w(t)$.





Unsupervised Learning
The Kohonen SOM Algorithm

- **The learning rate $\alpha(t)$ is similarly decayed.**
- **The end-result:**
  - Neurons (sometimes called "centroids") and their nearest neighbors are adapted to look like the sample data that are closest to them.
  - Likewise new sample data coming in will automatically be "classified" as the centroid they closest to:
    - ✓**This means that they will be "classified" together with other data that "looks" like them.**
  - Thus the Kohonen SOM automatically infers structure from the sample data presented to it.



Kohonen SOM Example – Color Classification

- **Color is described by a triplet (r,g,b), with r, g and b ranging from 0 to 255:**
  - (0,0,0) is black, (255, 255, 255) is white.
  - (255,0,0) is pure red, (0, 255, 0) is pure green, (0, 0, 255) is pure blue.
  - Total number of possible colors = 256 x 256 x 256 = 16,777,216 colors.
  - Usual to scale all values to between 0 and 1 to minimize chance of overflow.
- **Full video: https://www.youtube.com/watch?v=71wmOT4lHWc**

SUPERVISED LEARNING

NUS National University of Singapore
School of Computing

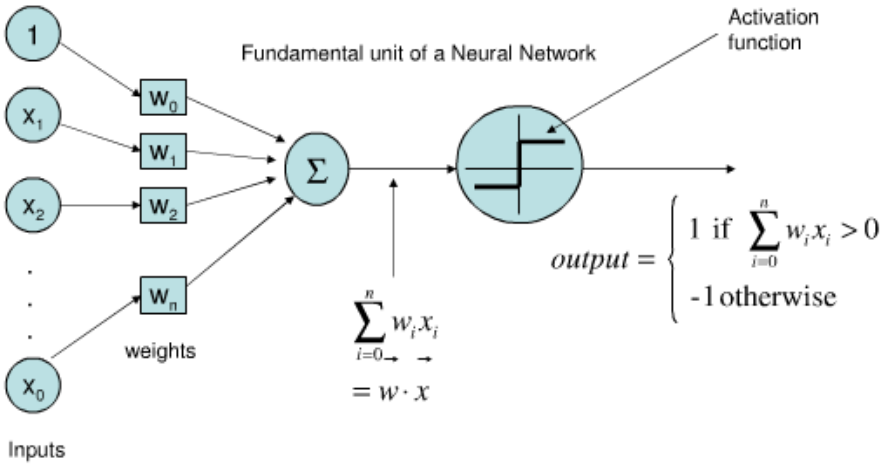
Supervised Learning

- **Unlike unsupervised learning algorithms like Kohonen SOMs, in supervised learning we must tell the network what it is looking at.**
  - Generally data is presented in the form of (x_i, y_i), where x_i is the sample input, and y_i is some kind of target or class that the neural network needs to learn.
- **We will look at two related supervised learning algorithms:**
  - Linear Perceptrons:
    - ✓**Simple, good for classifying linearly separable data.**
  - Multi-layer Perceptrons:
    - ✓**More complex, good for classifying non-linearly separable data.**

SUPERVISED LEARNING : PERCEPTRONS

Supervised Learning
Perceptrons

- **A single Perceptron (Mculloch – Pitts) neuron consists of:**
  - A set of inputs **x**.
  - A set of "weights" **w.**
  - A unit that does a dot product $\mathbf{w^T x}$.
  - An activation function that does outputs 1 when $\mathbf{w^T x + b} > 0$, and -1 otherwise. (other functions are possible, e.g. the identity function id(x)=x)
  - The bias b is normally modelled as a unit input, with the weight w_0 becoming the bias.



NUS National University of Singapore
School of Computing

Supervised Learning
Perceptrons

- **The learning algorithm is very simple:**
  - Present $(\boldsymbol{x_i}, y_i)$. Input $\boldsymbol{x_i}$ may be a $(k+1)$-element vector of elements $[x_{i1}, x_{i,2}, \ldots, x_{i,k}, 1]$. The additional $+1$ element in the vector is the bias.
  - FEEDFORWARD: Compute:

$$y_{out} = f\left(\sum_{j=1}^{k+1} w_j x_{ij}\right)$$

Here f(.) is the activation function.

  - UPDATE: Update each w_j with:

$$w_j = w_j + \alpha(y_i - y_{out})x_{ij}$$

NUS National University of Singapore
School of Computing

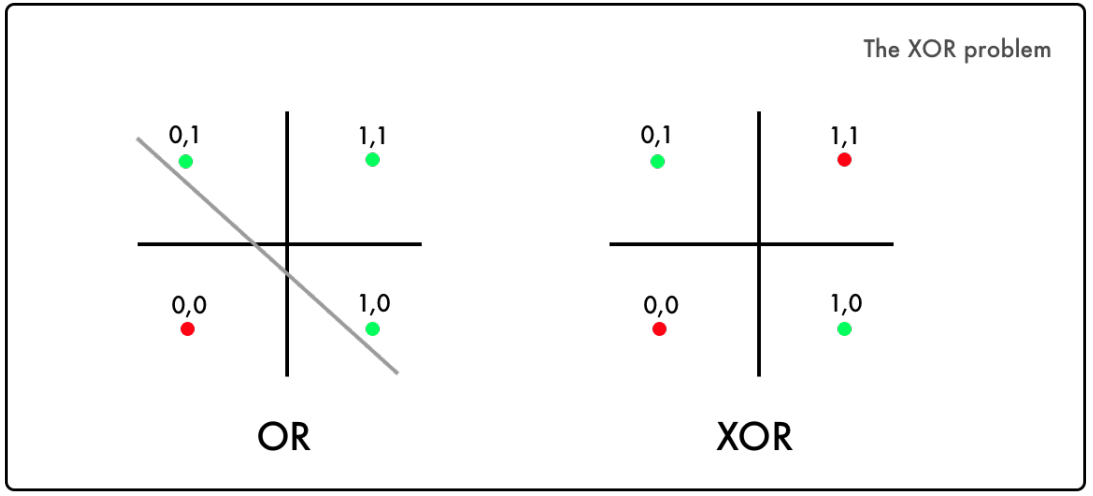
Supervised Learning
Perceptrons (perceptron.xlsx)

- **Function learning (Activation Function: id(x)=x)**
  - y(x) = 2x
  - y(x)=2x+5
  - y(x) = 2x+5 + noise
  - y(x) = 2x+5 + large noise
- **Classification:**
  - Given the length and weight of a vehicle we want to classify either as a lorry (-1) or a van (1)
  - Activation function f(x)=1 if x>0, -1 otherwise.
- **XOR Problem:**
  - Learn the XOR table.
  - Activation: f(x)=1 if x>0, 0 otherwise.

Inputs		Outputs
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	0

Supervised Learning
Perceptrons

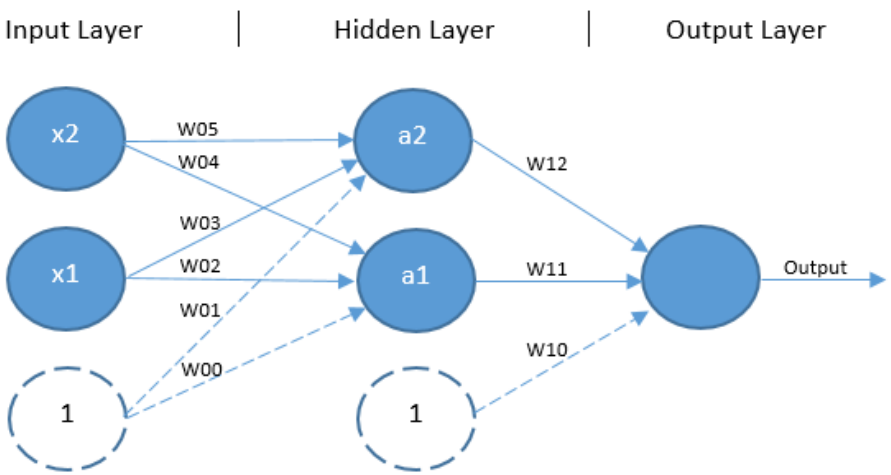
- **From our example we see that Perceptrons cannot learn the XOR problem.**
  - This is because it is not possible to draw a straight line to separate the samples into two classes.
  - Neural network research was killed for several years because no solution could be found for this problem.

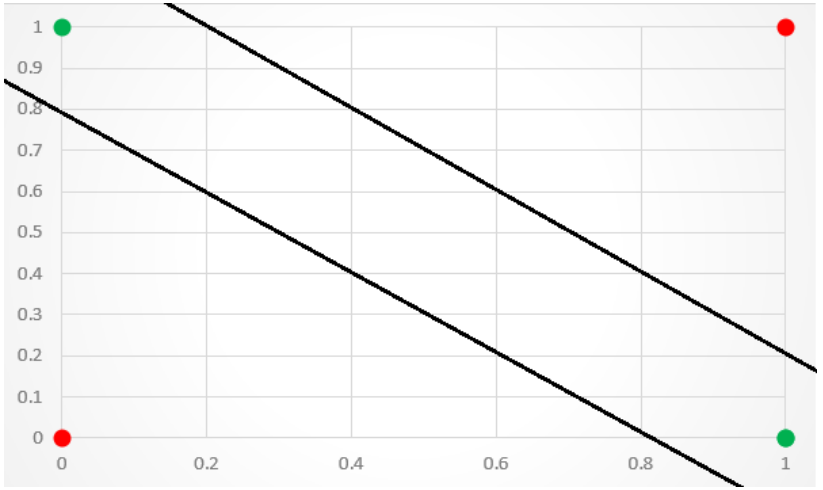


SUPERVISED LEARNING: MULTI-LAYER PERCEPTRONS

Supervised Learning
Multi-Layer Perceptrons

- **Our key issue in the XOR problem with a single Perceptron is that we cannot separate the points in the XOR problem with a SINGLE linear plane.**
- **However if we add a second Perceptron layer, we introduce a second plane that can neatly separate the points:**





Supervised Learning
Multi-Layer Perceptrons

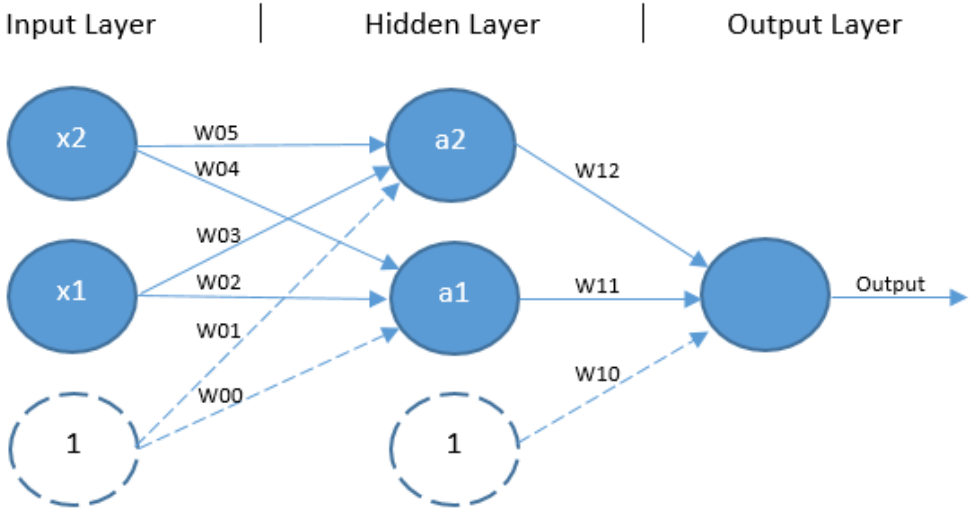
- **An MLP consists of:**
  - A set of inputs $\boldsymbol{x}$ with $n+1$ elements $[x_1, x_2, \ldots, x_n, 1]$. As before the $n+1$ item is the bias.
  - A set of m Perceptron nodes called "hidden nodes", $\boldsymbol{h}$. As always we add an $m+1$ term for the bias.
  - A set of weights $\boldsymbol{w_h}$ connecting $\boldsymbol{x}$ to $\boldsymbol{h}$.
  - A set of q Perceptron nodes $\boldsymbol{p}$ for the output.
  - A set of weights $\boldsymbol{w_p}$ connecting $\boldsymbol{h}$ to $\boldsymbol{p}$.
  - FEEDFORWARD:

$$h_j = f\left(\sum_{i=1}^{m+1} w_{ij} x_i\right)$$

$$p_k = g\left(\sum_{j=1}^{q+1} w_{jk} h_j\right)$$

Supervised Learning
Multi-Layer Perceptrons

- **The update step is considerably more complicated than single layer Perceptrons:**
  - In SLP there is only one "output" layer. It is relatively easy to compute the error, which is simply $y_t - y_{out}$.
  - In MLP, things get more complicated in the hidden layer. How do we assign errors to this layer?



Supervised Learning
Multi-Layer Perceptrons

- **In MLPs we use a technique called "gradient descent":**
  - In the output layer we find the quadratic error for ouput *k*:

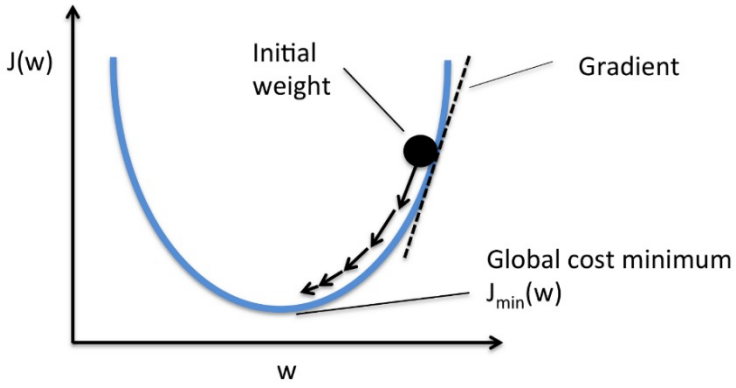
$$E = \frac{1}{2}(p_k - y_t)^2$$

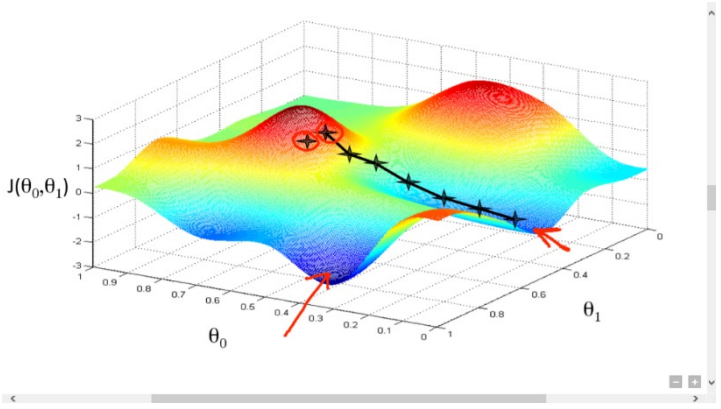
  - We want to minimize *E*:

$$\frac{dE}{dw_{jk}} = 0$$

✓**This points us in the direction of the maximum.**

✓**We therefore go in the OPPOSITE direction of the derivative, towards to minimum.**

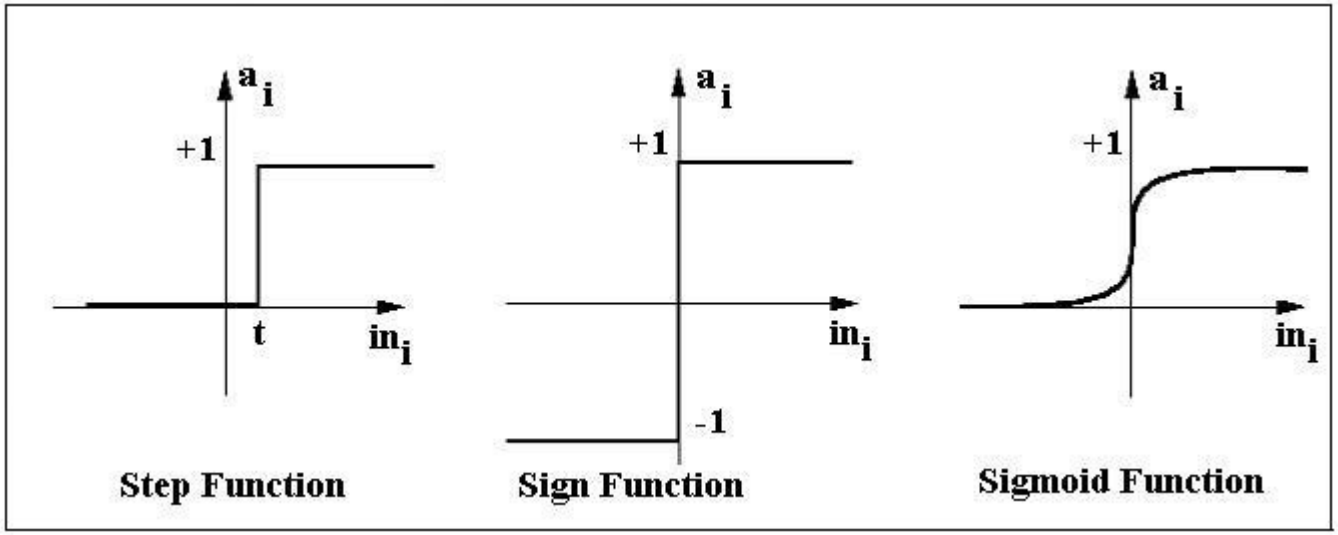




Supervised Learning
Multi-Layer Perceptrons

- **This means that the activation functions *f(.)* and *g(.)* must be differentiable:**
  - Our step (and sign) function has a discontinuity at 0, where the gradient is infinite. We choose instead the sigmoid function:

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



Supervised Learning
Multi-Layer Perceptrons

- **The sigmoid has a very convenient derivative:**

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

$$\frac{df(x)}{dx} = f(x)\big(1 - f(x)\big)$$

- **From here we can find the derivative of the quadratic error E of an output node p_k:**

$$E = \frac{1}{2}(p_k - y_t)^2$$

$$\frac{dE}{dw_{jk}} = (p_k - y_t)\frac{dp_k}{dw_{jk}}$$

$$= p_k(1 - p_k)(p_k - y_t)$$

NUS National University of Singapore
School of Computing

Supervised Learning
Multi-Layer Perceptrons

- **Similar to what we did for SLPs, we now "backpropagate" the error to the hidden layer:**

$$\delta_{jk} = h_j p_k (1 - p_k)(p_k - y_t)$$

- **We can now update the weight w_{jk}**

$$w_{jk} = w_{jk} - \alpha \delta_{jk}$$

where α is the learning rate.

Supervised Learning
Multi-Layer Perceptrons

- **Now we are ready to tackle the more challenging issue: Adjusting the weights connecting the inputs to the hidden layer.**
  - In the output layer we took into account the error in the output layer.
  - In the hidden layer we must similarly take into account all the corrections applied to every weight connecting h_j to every output node p_k. We do this by summing over all the corrections derived from each output node p_k:

$$\sum_{k=1}^{m+1} w_{jk}\delta_{jk}$$

  - We multiply this by the derivative of h_j:

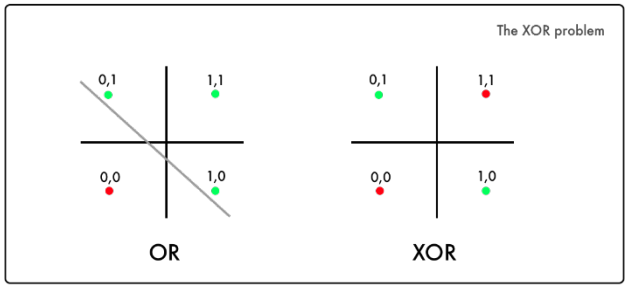
$$h_j(1-h_j)\sum_{k=1}^{m+1} w_{jk}\delta_{jk}$$

  - Finally our correction to w_{ij} is:

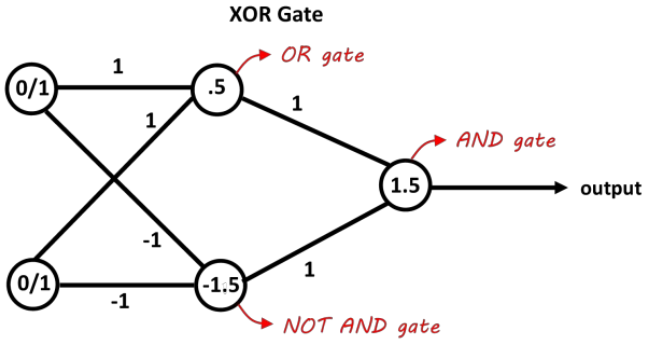
$$w_{ij} = w_{ij} - \alpha x_i h_j(1-h_j)\sum_{k=1}^{m+1} w_{jk}\delta_{jk}$$

Supervised Learning
Multi-Layer Perceptrons – XOR Problem

- **Recall that Perceptrons cannot solve the XOR problem:**



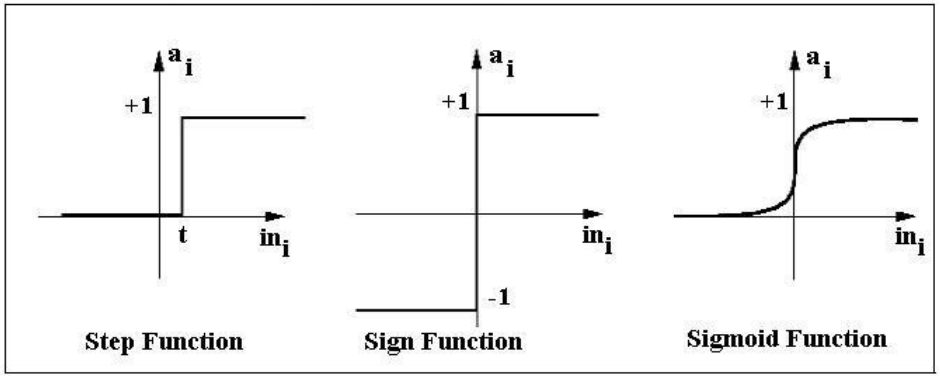
- **Adding a hidden layer solves this problem:**

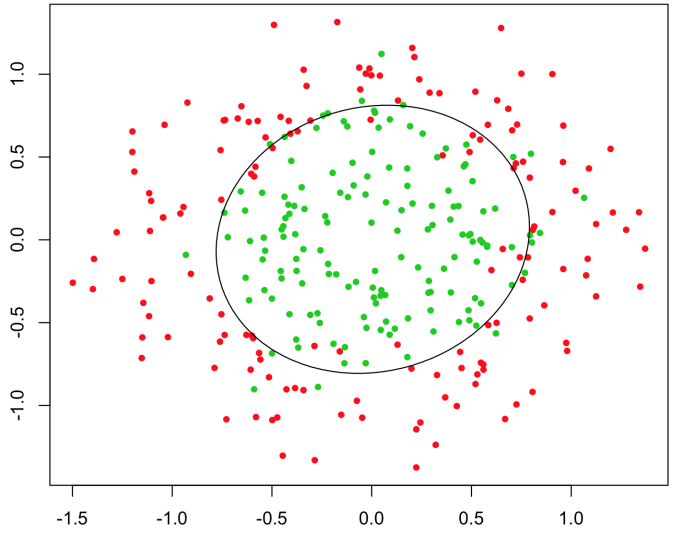


SUPERVISED LEARNING: INTRODUCING NON-LINEARITY

Supervised Learning
Non-Linearity

- **The addition of a hidden layer only partially solves the problem of classifying data points:**
  - What if the data points occur in a circle? No straight hyperplane can solve this.
- **Solution: Use non-linear decision boundaries.**
  - Popular ones: Sigmoid for (0, 1) outputs, tanh for [-1, 1] outputs, and ReLu.





Neural Network Hyperparameters

- **The weights that are being adjusted by our optimization algorithms are called "parameters", and neural networks are known as "parameterized models".**
- **There is another dimension of neural networks that must be "optimized" as well to get good results, called "hyperparameters". This is related to the design of the neural network. Important hyperparameters include:**
  - Architecture (Dense networks, Long-Short-Term Memories, Convolutional Neural Networks, Autoencoders, Generative-Adversarial Networks, etc)
  - # of input nodes (usually constraints by the problem itself)
  - Input and output encoding.
  - # of hidden layers.
  - Size of each hidden layer.
  - Loss functions
  - Transfer functions.
  - Optimization functions and their parameters (learning rate, momentum, etc.)
  - Dropouts and Regularizers.

Neural Network Hyperparameters

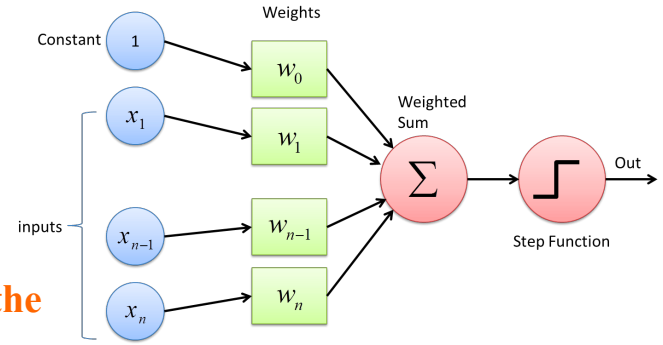
- **Some of the parameters are constrained by the problem itself (e.g. input and output size), others are found by trial-and-error (yes really).**
- **We will look at a subset of hyperparameters that are slightly more "scientific"**
  - Transfer Functions
    - ✓**We've already seen the sigmoid and step functions.**
  - Dropouts and Regularizers
    - ✓**These control the issue of "overfitting"**

Recall: Neural Networks and How They Work.

- **Generally neural network applications can be split into two classes:**
  - Classification:
    - ✓**Given a set of inputs $\{x_0, x_1, \ldots, x_{n-1}\}$, assign labels y_j to each input x_i from the set of classes $\{y_0, y_1, \ldots, y_{m=1}\}$, to mean that x_i is a member of class y_j.**
    - ✓**E.g. we may classify a library of books into the genres of novels, cookbooks, self-improvement books, etc.**
  - Regression:
    - ✓**Give a series of data of data $\{x_0, x_1, \ldots, x_{n-1}\}$, can we predict x_n?**
- **In either case we are attempting to learn a function *f(x)*:**
  - Classification: What $f(.)$ gives $f(x_i) = y_i$?
  - Regression: What $g(.)$ gives $g(x_{n-1}) = x_n$?

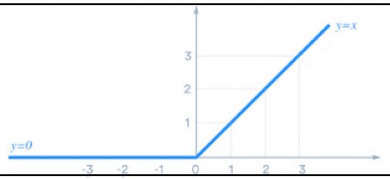
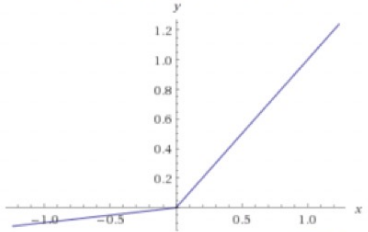
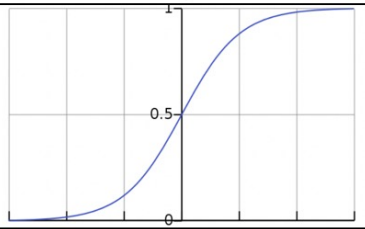
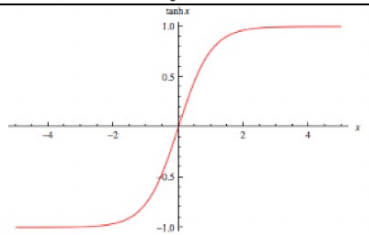
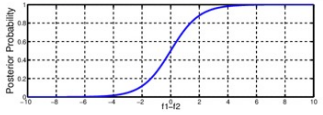
Recall: Neural Networks and How They Work.

- **The figure below shows an example neuron:**
  - The sum of each neuron is passed through an *activation function* to transform it in some way.
    - ✓**E.g. scale it from -1 to 1 or 0 to 1.**
    - ✓**Destroy the linearity of the output.**
    - ✓**Shape it into a statistical distribution.**
  - The weights need to be adjusted to create $f(x)$:
    - ✓**We must know the error $||y_i - f(x_i)||$. This is called the "loss function"**
    - ✓**We must have an algorithm to adjust the weights accordingly. This is done by "optimizers".**
- **Let's now take a closer look at each of these important parts.**



Activation Functions

- **An activation function (often also called "transfer function") transforms the dot product of the weights and inputs.**
  - Needed because this dot product is usually unbounded.
  - Also because the dot-product is a linear operation. Linearity prevents NNs from solving certain classes of problems. Explained in next section.
    - ✓**One obvious limitation: A linear function cannot regress over non-linear functions.**
- **The table on the next slide shows the various common activation functions.**

Function	Description	Shape	Range	Primary Use
Identity	f(x) = x	Straight line	[-inf, inf]	Linear regression.
ReLU	f(x) = x if x > 0 f(x) = 0 if x<=0		[0, inf]	Linear regression, classification
Leaky ReLU	f(x) = x if x > 0 f(x) = αx if x<=0, 0 α = [0, 1]		[-inf, inf]	Linear regression, classification
Sigmoid (Logistic)	$S(x) = \frac{1}{1+e^{-x}}$		[0, 1]	Classification, regression (scaled)
tanh	$\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$		[-1, 1]	Classification, regression (scaled)
Softmax	$S(x) = \frac{e^{x}}{\sum_j e^{x_j}}$		[0, 1]	Classification (Makes output a statistical distribution)

Loss Functions

- **A loss function measures the error between the function $f(x_i)$ learnt by the NN and the actual target value y_i.**
  - In regression y_i might just be x_{i+1}.
- **The table on the next page shows the common loss functions:**

Loss Function	Formulation	Use
Mean Squared Error (MSE)	$\forall t \in \{(x_i, y_i), x_i \in X, y_i \in Y\}$ $L_{MSE}(T) = \sum_{i=0}^{\lvert T \rvert} (y_i - f(x_i))^2$	Regression. Fast convergence, sensitive to outliers.
Mean Absolute Error (MAE)	$\forall t \in \{(x_i, y_i), x_i \in X, y_i \in Y\},$ $L_{MAE}(T) = \sum_{i=0}^{\lvert T \rvert} \lvert y_i - f(x_i) \rvert$	Regression. Less sensitive to outliers.
Binary Cross Entropy	$\forall t \in \{(x_i, y_i), x_i \in X, y_i \in Y\},$ $L_{BCE}(T) = -\frac{1}{\lvert T \rvert} \sum_{i=0}^{\lvert T \rvert} (y_i \log(f(x_i)) + (1 - y_i) \log(1 - f(x_i)))$	Binary classification. Use with softmax.
Hinge	$\forall t \in \{(x_i, y_i), x_i \in X, y_i \in Y\},$ $L_H(T) = \max(0, 1 - y_i f(x_i))$	Binary classification.
Categorical Cross Entropy	With *m* classes and *n* training samples: $L_{CE}(y, y') = -\sum_{j=1}^{m} \sum_{i=1}^{n} y_{ij} \log(y'_{ij})$ y'_{ij} is the *jth* output of $f(x_i)$, while y_{ij} is the *jth* output of the one-hot target vector y_i. I.e. $y_{ij} = 1$ iff x_i belongs to class j, 0 otherwise.	Multiclass classification. Use with softmax.

Optimizers

- **In NNs, optimizers are algorithms that derive the best set of parameter values that minimize the loss function (see later).**
- **Table shows a small set of optimizers and their characteristics – No straightforward way to choose!**

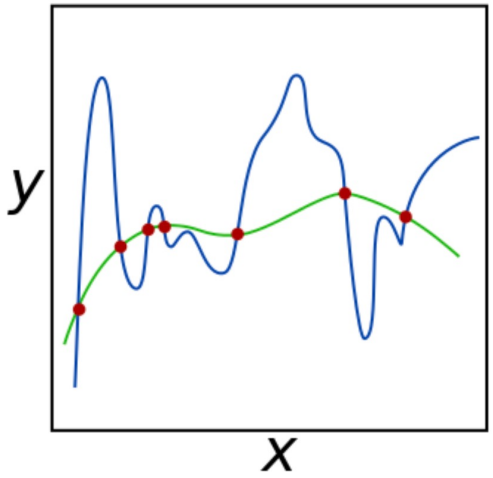
Algorithm	Characteristic
Stochastic Gradient Descent (SGD)	Classic algorithm, same learning rate is applied to all weights. Momentum may be used to speed up learning.
RProp	Each weight uses a different learning rate. Weights that have two consecutive update gradients of the same sign are updated more.
RMSProp	An improved version of RProp for mini-batches.
Adagrad	Weights that are updated more have smaller training rates (i.e. train slower). Weights that are not updated frequently have larger training rates. Good for sparse data.
Adadelta	Less aggressive form of Adagrad.
Adam	Similar to Adagrad

OVERFITTING

Overfitting

- **All data can be thought of as consisting of two things:**
  - Some sort of generator process *f(x)*. The input *x* could simply be time, or an actual input.
  - Noise
- **When we build a model, we want it to learn only the generator process *f(x)*.**
  - The noise is random and unique to the training data. If we learn the noise then our model cannot generalize well – "overfitting".
- **Two choices to avoid learning the noise:**
  - Have a simple model (i.e. fewer parameters)
  - Have more training data so that the network can "average out" the noise.
- **Catch:**
  - A simple model may not learn *f(x)* well – underfitting.
  - The amount of training data needed grows exponentially with the model complexity – "curse of dimensionality"

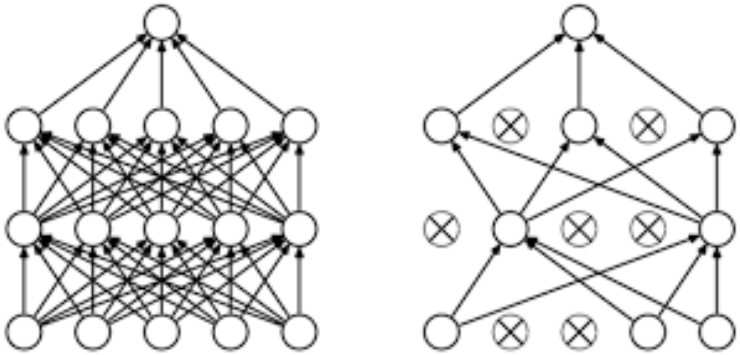
Overfitting



- **When we have too many parameters, we tend to learn the blue noisy line instead of the better green line:**
- **Marked by:**
  - Extremely low training loss.
  - Increasing validation loss.
  - Hence why we must check both!

Dealing with Overfitting
Noise Layers

- **Dropout Layers:**
  - A fixed percentage of neurons in the layer are dropped from training for one more epochs.
  - They are then put back in, and another percentage of neurons are dropped.
  - Reduces # of training paramters:



(a) Standard Neural Net (b) After applying dropout.

Dealing with Overfitting
Noise Layers

- **Noise layers add random noise to the outputs of the previous layer.**
  - E.g. add Gaussian noise to change the data to look like "new data" – augmentation.
    - ✓**In Gaussian noise the values of the noise are distributed according to the Gaussian probability distribution function:**

$$p_G(z) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

    - ✓**In images this simulates sensor noise, e.g. due to low-light.**

  - More details at https://keras.io/layers/noise/

Dealing with Overfitting
Regularizers

- **Regularizers:**
  - A "regularizer" is a penalty that is applied to the loss function of a layer.
  - E.g. In regression our loss function may look like this (squared error loss):

$$\text{RSS} = \sum_{i=1}^{n} \left(y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij} \right)^2 .$$

  - Our optimization algorithm will find values of the parameters (β) to minimize this loss function.
  - If there are too many parameters (β) the model starts of learn the unique noise of the training data – overfitting or "memorizing". Poor generalization.
  - We force a reduction – simplification - in parameters:
    - ✓**This is equivalent to forcing some of the parameters to 0.**

Dealing with Overfitting
Regularizers

- We can do this by adding the absolute values of the parameters (L1 regularization) or squares of the parameters (L2 regularization) to the loss function:

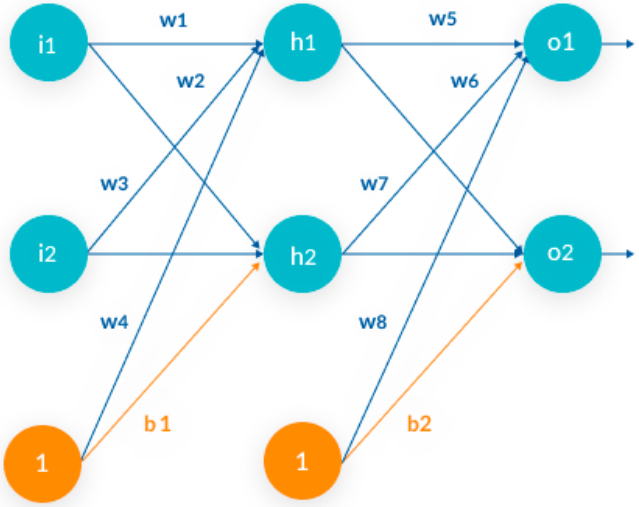
$$\sum_{i=1}^{n}\left(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij}\right)^2 + \lambda\sum_{j=1}^{p}\beta_j^2$$

- Now our optimizer will restrict the parameters themselves:
  - ✓**Some will be forced to zero while others are not allowed to grow too big.**
  - ✓**Otherwise the loss function will become too much.**
- The hyperparameter λ controls how much "flexibility" we give to the parameters. Higher λ means more strict control.
- L1: Eliminates less important parameters and simplifies the output.
- L2: More effective in severe overfitting since it squares the parameters.

- **If λ is too high the model will underfit.**

Dealing with Overfitting
Regularizers

- **Regularizers are added to individual layers:**
  - kernel_regularizer: Controls the main weights (lines connecting the green nodes)
  - bias_regularizer: Controls the bias weights (lines connecting the orange nodes)
  - activity_regularizer: Controls based on layer output (o1 and o2)



NUS National University of Singapore
School of Computing

Dealing with Overfitting
Regularizers

- **Example (From Keras, which we will see in the hands-on)**
  - The λ is specified in the brackets of regularizers.l1(..) or regularizers.l2(..).
  - Here it is set to 0.01.

```
from keras import regularizers
model.add(Dense(64, input_dim=64,
                kernel_regularizer=regularizers.l2(0.01),
                activity_regularizer=regularizers.l1(0.01)))
```

NUS National University of Singapore
School of Computing

Neural Network Hands-On

Load up NN.ipynb in Jupyter Notebook

NUS National University of Singapore
School of Computing

Keras Resources:

- **Sequential Model: https://keras.io/models/sequential/**
- **Functional API: https://keras.io/models/model/**
- **Core Layers (including Dropout): https://keras.io/layers/core/**
- **Noise Layers: https://keras.io/layers/noise/**
- **Convolution Layers: https://keras.io/layers/convolutional/**
- **Pooling Layers: https://keras.io/layers/pooling/**
- **Recurrent Layers (including LSTM): https://keras.io/layers/recurrent/**
- **Regularizers: https://keras.io/regularizers/**
- **Activations: https://keras.io/activations/**
- **Losses: https://keras.io/losses/**
- **Optimizers :https://keras.io/optimizers/**

Summary

- **In this lecture we saw:**
  - Our various options for setting up convolutional neural networks.
  - The various activation and loss functions and optimization algorithms.
  - How to deal with overfitting.
  - NN algorithms.
- **In the next lecture we will look at statistical models:**
  - These are often much simpler, have much better theoretical foundations, and require less training data.

NUS National University of Singapore
School of Computing

Trial Lecture Hands On

- **You will be randomly assigned into rooms.**
- **Spend 5-10 minutes getting to know each other.**
- **Download ”SWS3009 Trial – Neural Networks.ipynb” from Canvas.**
- **Load this into Jupyter.**
- **Work together to solve the questions.**
- **No need to submit, this is to give you some hands-on.**