Foundations

AI vs ML vs NC

- *Non-examinable*
- Artificial intelligence (AI) = any way of making computers appear intelligent e.g. may just be good hardcoded rules (if else blocks) e.g. ELIZA
- Machine learning (ML) = using statistics to generate AI rules from data e.g. SVM
- Neural computing (NC) = brain-inspired statistical models for ML i.e. artificial neural networks
- Deep learning (DL) = NC with many layers e.g. Transformers
- DL $\subset$ NC $\subset$ ML $\subset$ AI
- **ML is more dependent on good human feature engineering than NC which can in theory extract the best features itself from the raw data**
- NC requires more compute power and more data than ML but will typically give better accuracy given these than ML which will hit a ceiling

Learning paradigms

- Supervised learning = labelled data e.g. most NC — learning from examples
- Unsupervised learning = unlabeled data e.g. k-means clustering
- Reinforcement learning = learning from feedback on decisions made — learning from experience
- **Transductive learning = make predictions for the unlabelled portion of a seen dataset**
- **Inductive learning = learn a model for a labeled dataset that can applied to make predictions on unseen data —** generalises transductive learning
    - **Self-learning for semi-supervised-learning = train model on labeled data; add most confidently classified unlabeled data to labelled data with the predicted label (pseudo label); repeat** — bootstraps learning over data for which annotation is expensive

Artificial Neural Networks architecture

- An N-layer neural network has N+1 layers including the input layer i.e. has N sets of edges in a feedforward network (input → layer 1; layer 1 → layer 2; …; layer n − 1 → layer n)
- A single layer neural network has only an input layer and an output layer (no hidden layers)
- A shallow neural network has 1 or 2 hidden layers
- A deep neural network has at least 3 hidden layers
- **In a feedforward neural network, data-flow is unidirectional, there are no loops**
- **In a feedback neural network (e.g. RNN), there are loops**

Biology

Reflex arc

- *This is examinable!*
- We are able to interact with the world thanks to sensory, motor, and relay neurons working together
- **Sensory neurons transfer inputs from receptors (e.g. eye, ear) to the central nervous system (brain, spinal cord)**
- **Relay neurons are located within the central nervous system**
- **Motor neurons transfer outputs from the central nervous system to effectors (muscles)**
- **A case study: heat receptors in skin detect dangerous levels of heat on hand ⇒ sensory neurons in hand send signal to relay neurons in spinal cord ⇒ these relay neurons** *(bypassing the higher-order thinking of the brain for speed as this is a reflex)* **send signal to motor neurons in same arm ⇒ these motor neurons cause the muscle to effect a response (contract to move the hand away from the hazard)**

Neuron anatomy

- *Non-examinable*
- *The human brain contains ~100 billion neurons, which is roughly the parameter count of current transformers such as DeepSeek R1, but requires way way way less resources*
- Biological neurons generate action potentials (electrical signals) to transmit information around the body, alterations to the signals during this transmission is how signals are processed causing intelligence to emerge
- **Dendrites** have protrusions (spines) that **receive signals from other neurons**
- The **dendrites converge at the soma** (cell body: nucleus etc is stored here)
- **If the total strength of the signals suming at the soma exceeds the threshold of the axon hillock the neuron will send on a signal otherwise it will not**
- **The axon is a tube-shaped nerve fibre that transmits the signal produced at the axon hillock** to other neurons (or to muscles)

Further axon biology: Synapses

- **The protrusions at the other end of the axon are called the axon terminals and connect to the dendrite spines of other neurons or other cells** (e.g. muscle cells)
- **The junction between an axon terminal and another cell is called a synapse** — there is a synaptic gap that the message of the action potential must cross, typically chemically but sometimes electrically
- **The action potential causes chemical neurotransmitters to be released from the presynaptic neuron which travel across the synaptic cleft (gap) and bind to receptors in the postsynaptic cell**
    - Excitatory neurotransmitters cause an action
    - Inhibitory neurotransmitters prevent other neurotransmitters from causing an action
- **Hebbian theory of neuroplasticity: Neurons that fire together wire together. Neurons that fire out of sync lose their link**
    - *True biology: If neuron A fires just before neuron B the link from A to B strengthens and the link from B to A weakens. However, as action potentials are quick they must fire very close to each other for there to be any effect.*

Signal transmission: Membrane potential

- The lipids and proteins in the cell membrane separate the inside of the cell from the outside, thus the contents of the cell are electrically insulated from the external environment
- **Membrane potential = electric potential outside the cell − electric potential inside of a cell — caused by unequal distribution of (Na+ and K+) ions**
- Resting potential = membrane potential of a neuron when it is not firing
    - Typically −70mV — **inside of cell has fewer positive ions than the outside**
- The cell membrane contains channels to allow the exchange of ions across the membrane
    - Leaky channels are always open
    - Voltage gated channels only open at certain voltages e.g. are closed at resting potential
- **Neurons contain a sodium-potassium pump — expends energy to move 3 Na+ ions out and 2 K+ ions in** — now it is clear why the inside is more negative than the outside
    - **If the pump stopped pumping, the leaky channels would cause K+ ions to diffuse out and Na+ ions to diffuse in**

Signal transmission: Action potential

- Action potential = a rapid change in membrane potential to send information down an axon
    - **All or nothing law: Any stimulus to the axon hillock below its threshold membrane potential (e.g. -55mV) will produce no response on the axon whereas anything at or above it will produce a full spike**
- **A spike consists of:**
    1. **Depolarization: Increase in membrane potential from stimulus causes voltage gated sodium channels to open** allowing Na+ ions to diffuse into the cell making it more positive (e.g. become more positive than outside for a membrane potential of +30mV)
    2. **Repolarization: Increase in membrane potential from depolarization causes voltage gated potassium channels to open** allowing K+ ions to diffuse out of the cell making it more negative
    3. **Return to the resting potential** (after a small overshoot into being even more negative than the resting potential): **Drop in membrane potential from repolarization causes voltage gated channels to reclose** allowing the sodium-potassium pump to return the cell to steady state
- A failure to spike consists of: Insufficient increase in membrane potential to open voltage gated channels followed by sodium-potassium pump quickly returning membrane potential to resting potential

Perceptron

McCulloch-Pitts neuron

- *The MP neuron was the first mathematical model of a biological neuron*
- **MP neuron: Sum up 0/1 inputs; output is 1 iff sum is at least the neuron's threshold θ ; 0 otherwise**
  - This describes the behaviour of excitatory inputs — **some inputs can be designated as inhibitory inputs: if any of the inhibitory inputs is 0 the neuron is forced to output 0**
  - θ is a manually set parameter — no algorithm is provided to learn it
- Heaviside activation function with threshold $\theta = H_\theta(z) = 1$ iff $z \geq \theta$; 0 otherwise
- Rojas Diagram: Right hand side of neuron is filled with black; threshold θ is written on left hand side; small white circle is drawn between inhibitory input and neuron
- An MP neuron can be used to implement some (in particular those that correspond to linearly separable functions) logic gates:
  - N-input AND gate = Threshold of N and all inputs excitatory
  - N-input OR gate = Threshold of 1 and all inputs excitatory
  - NOT GATE = Threshold of 0 and input inhibitory
  - XOR gate = not possible as is not linearly separable

Perceptron

- *Rosenblatt extended the MP neuron by incorporating the Hebbian learning rule and called this new type of neuron a Perceptron*
- In modern terms, a Perceptron is a single layer ANN that takes continuous 0-1 inputs and produces a binary 0/1 output
- Perceptron neuron generalises MP Neuron: Takes a weighted sum of inputs (dot product wx); output is 1 iff sum is at least the neuron's threshold θ ; 0 otherwise — weights w and threshold θ can both be learnt
    - This is equivalent to having a fixed threshold of 0 and a learnt bias (extra weight) of −θ
    - **Heaviside step function = H(z) = 1 iff z > 0; 0 otherwise**
- **Although we now allow continuous inputs, this is still a linear binary classifier so we still can't solve XOR**
- **MP and perceptron both produce binary 0/1 output using Heaviside step function but MP takes binary 0/1 input whereas Perceptron takes continuous 0-1 input**
- **MP and Perceptron both sum up their inputs but only Perceptron weights the sum (using learnt parameters)**
    - **MP essentially only has weights 0 (not connected), 1 (excitatory), −∞ (inhibitory)**

Representing logic gates

- The maximum boundary perceptron for a 2 input $\wedge$ gate is w = (1, 1, −1.5)
- The maximum boundary perceptron for a 2 input $\vee$ gate is w = (1, 1, −0.5)
- A single pure perceptron cannot exactly solve XOR as it is linearly inseparable — this lead to the AI winter in the 1970s
  - Considering all 4 possible inputs gives the system of constraints $b < 0$; $w_2 + b \geq 0$, $w_1 + b \geq 0$; $w_1 + w_2 + b < 0$ and these contradict each other
- **Because it has heaviside activation, MLP can solve XOR by computing intermediate values**
  - A XOR B = (A OR B) AND (A NAND B) — OR, AND, NAND are all linearly separable
- A single layer perceptron can solve XOR with a suitable activation function (instead of the Heaviside threshold) e.g. 0 → 0, 1 → 1, 2 → 0 — surely this is cheating though

Multilayer Perceptron

MLP decision boundaries

- **Theorem: An MLP with only 1 hidden layer (with suitably many neurons) can emulate any polygonal acceptance region**
  Proof: Each neuron in the hidden layer acts as a single layer perceptron, creating a half-plane acceptance region. The single neuron in the final layer can act as an n-input AND gate and thus will take the intersection of the half-planes. Hence, by learning suitable equations for the half-planes in the middle layer the polygon can be constructed
    - Corollary: An MLP with only 1 hidden layer (with suitably many neurons) can emulate any acceptance region which is the complement of a polygon as the complement of a half-plane is still a half-plane
- **An MLP can even recognize an acceptance region that is a union of polygons by adding a third layer to union together the polygons (n-input OR gate) after they have each been intersected from the half planes** (consider a simple tweak to the ANDs of each to mask out the half planes it does not use)

Activation functions

- A multilayer network with linear activation functions (i.e. effectively no activation function) is mathematically equivalent to a single-layer model — activiaton functions provide non-linearity so that we get benefits from depth
- Activation functions can also provide normalization

Activation functions: Sigmoids

- **Sigmoid** is a smooth approximation of the Heaviside step: $\boldsymbol{\sigma(x) = (1 + e^{-x})^{-1}}$
  - Proposition: $\boldsymbol{\sigma'(x) = \sigma(x)(1 - \sigma(x))}$
    Proof: Let $y = 1 + e^{-x}$. Then, $\sigma(x) = y^{-1}$ and so $d\sigma/dy = -y^{-2}$. Moreover, $dy/dx = -e^{-x}$. Thus, $d\sigma/dx = (y - 1)/y^2 = y^{-1} - y^{-2}$. Recall that $\sigma(x) = y^{-1}$ and hence conclude that $\sigma'(x) = \sigma(x)(1 - \sigma(x))$
  - **Has vanishing gradient problem** (for all x, $0 < \sigma'(x) \leq ¼$)
  - **exp is expensive to compute** but at least once we've computed the function value it is cheap to compute the derivative
- Tanh is rescaled sigmoid: $\tanh(x) = 2\sigma(2x) - 1 = (e^x - e^{-x})/(e^x + e^{-x})$
  - Proposition: $\tanh'(x) = 1 - \tanh^2(x)$
    Proof: As $\tanh(x) = 2\sigma(2x) - 1$, $\tanh'(x) = 2*2\sigma'(2x) = \ldots = (\tanh(x) - 1)(\tanh(x) + 1)$
  - Still have computationally expensive exp(s) but still has nice derivative
  - Still have vanishing gradients (but not so severe so soon)
- **Let $y = \text{Softmax}(z)$. Then, $y_i = \exp(z_i)/\sum_j \exp(z_j)$**
  - $\sum y_i = 1$ and $\forall i.\ 0 \leq y \leq 1$
  - $\sigma(x) = \text{Softmax}((x, 0))$

Activation functions: *LUs

- **ReLU(x) = max(0, x)**
  - **Cheap to compute** (and to differentiate)
  - ReLU'(x) = 1 if x > 0, 0 if x < 0
  - **LeakyReLU$_\alpha$(x) = max(αx, x) fixes dying ReLU problem** and keeps ease of computation and nice magnitude gradient (in fact exactly 1) for positive inputs
  - **PReLU: LeakyReLU but α is learnt instead of hyperparameter**
- **ELU$_\alpha$(x) = x if x ≥ 0; α(e^{-x} − 1) if x < 0**
  - **ReLU for x > 0**
  - **Smooth derivative unlike ReLUs** which have undefined derivative at x=0
  - Tends towards −α from above for x < 0 but this does mean the gradient is vanishing

Regularisation

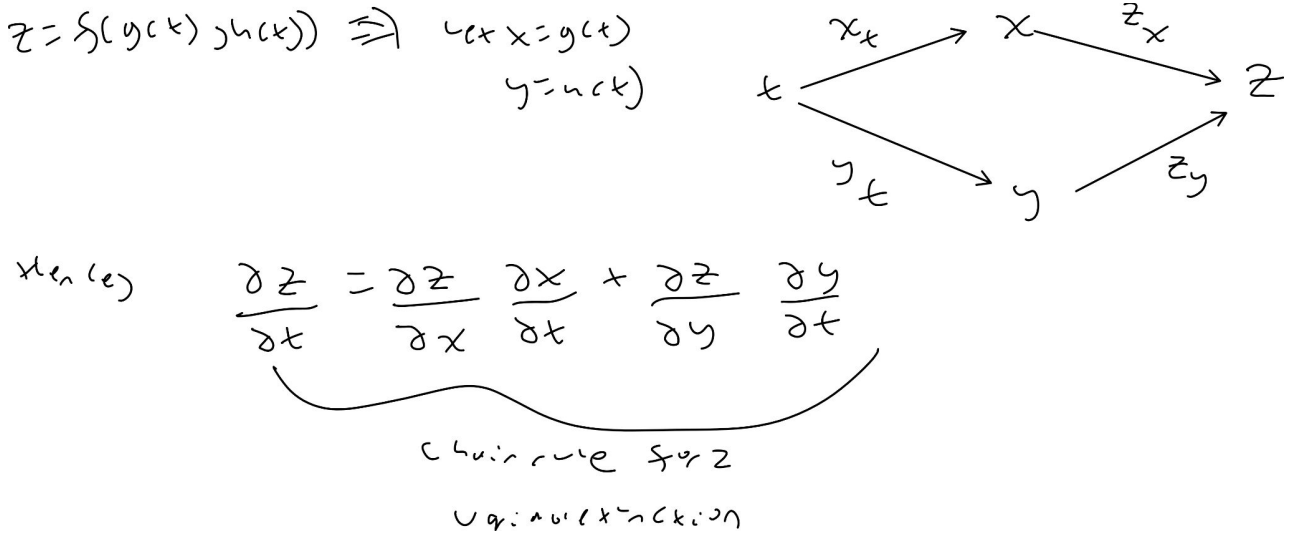
- The loss function measures the error of the network output for a single example
- The cost function measures the penalty of the network overall (e.g. mean loss over all examples with regularisation penalty terms added)
- **L_1 penalty term = sum of absolute values of weights**
- **$L_2^{(2)}$ penalty term = sum of squares of weights**
- Using L_1 for regularisation is called lasso regularisation
- Using L_2 for regularisation is called ridge regularisation
- **Regularisation helps to prevent overfitting** — the larger the weights are, the more sensitive the model is to small fluctuations in the input

Gradient descent

- If $f(x_1, \ldots, x_n) = \ldots$, **gradient of f = $\nabla f = [\delta f/\delta x_1, \ldots, \delta f/\delta x_n]$**
  - $(-)\nabla f(x_1, \ldots, x_n)$ gives the direction of fastest instantaneous decrease/increase of f at the point $(x_1, \ldots, x_n)$
- Gradient descent: $x_{t-1} = x_t - \alpha \nabla f(x_t)$
  - If converges, converges to <u>an</u> x_L s.t. $\nabla f(x_L) = 0$ (i.e. x_L is a stationary point)
    - *Moreover, if the expression is convergent and a sufficiently small α is used, will converge to the nearest stationary point to x_0*
- Single-variable chain rule: Let f(x) = g(h(t)), then d/dt(f(x)) = dg/dh * dh/dt = f'(g(t)) * g'(t)
- **Multivariable chain rule: Let $f(x_1, \ldots, x_n) = f(v(t))$** be a function with multivariable input and scalar output written in terms of a function v with scalar input and multivariable output, **then $d/dt(f(x_1, \ldots, x_n)) = \nabla f(v(t)) \cdot v'(t)$** where v is differentiated element-wise and f is differentiated with respect to v

Dependency graphs

- Dependency graphs make computing ∇f easy
- **A computation graph is created for the function**
- **Each edge (a, b) is annotated with $\delta b/\delta a$**
- **Any partial derivative $\delta c_1/\delta c_0$ can be calculated as the sum over all paths from c_0 to c_1 of the product of the edges along the path**



Backward propagation (backprop)

Let z_{ij} = pre-activation output of neuron j in layer i and a_{ij} = post-activation output of neuron j in layer i. Then, $a_i = \sigma_i(z_i)$ where σ_i is the activation function of layer i and $z_i = w_i a_{i-1} + b$ if $i > 1$ and $w_1 x + b$ if $i = 1$.

Thus:

$\delta L/\delta w_{ijk} = \delta L/\delta z_{ik} * \delta z_{ik}/\delta w_{ijk} \quad \forall i. \ \forall j. \ \forall k.$

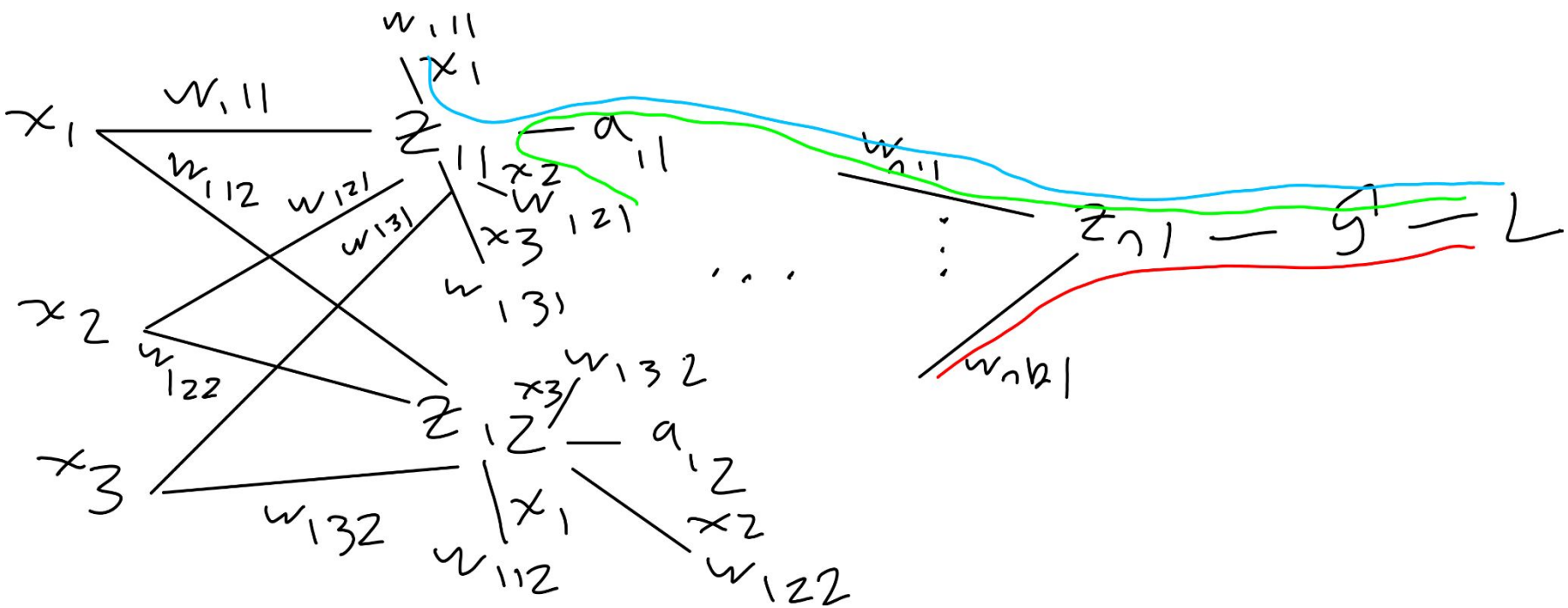
$= \delta L/\delta z_{ik} * a_{i-1,j}$ if $i > 1$ and $\delta L/\delta z_{1k} * x_j$ if $i = 1$

where $\delta L/\delta z_{ik}$ $(i < n) = \delta L/\delta a_{ik} * \delta a_{ik}/\delta z_{ik}$ where $\delta L/\delta a_{ik} = \sum_k \delta L/\delta z_{i+1,k} * w_{i+1,j,k}$

and $\delta L/\delta z_{nk} = \delta L/\delta a_{nk} * \delta a_{nk}/\delta z_{nk}$

- Thus, **the $\delta L/\delta z_{i,k}$ we compute for layer i is needed again for all the previous layers so is memoized for efficiency**

Backprop: Visualisation



Hopfield Network

Hopfield network

- *John Hopfield won the 2024 Nobel Prize in Physics alongside Geoffrey Hinton for work based on the Hopfield network*
- A Hopfield network is a fully-connected (but with no self-loops) undirected graph of standard perceptrons (except with outputs 1/−1)
    - As the network is undirected (and complete), weight of input from a node = weight of output into that node
- **A Hopfield network repeatedly applies a transformation to pull an input towards its nearest stored attractor pattern** (a fixed point of this iterative process)
- Content-addressable memory = ability to retrieve an item based on only approximately (i.e. a noisy version of) its content
- An nxn black and white image can be represented in a flattened form as a 0/1 vector with n^2 entries. However, the bipolar encoding (1s as 1s but 0s as −1s) will be easiest

Hopfield network: Learning rule

- **Learning rule: For a <u>bipolar</u> network which only needs to store a <u>single</u> pattern x: $w_{ij} = x_i x_j \ \forall i \in [|x|]. \ \forall j \in [|x|] \backslash \{i\}.$** That is that $W = xx^T - I$
- **Learning rule: For a <u>bipolar</u> network given a collection of patterns $x = \{x^{(1)}, \ldots, x^{(m)}\}$: W = average of weights for each pattern** $= (1/m * \sum_{j \in [m]} x^{(j)} x^{(j)T}) - I$
- For a binary network, apply the bipolar rule with the necessary transformation on the patterns $(y(j) = 2x^{(j)} - 1)$
- These learning rules follow the Hebbian associative learning principle: If both in a pair agree the weight connecting them increases, if they disagree the weight connecting them decreases

Hopfield network: Stability

- Lemma: **The state of a bipolar Hopfield network evolves according to $s(t+1) = sgn(W \cdot s(t))$ where $sgn(x) = 1$ iff $x \geq 0$; -1 iff $x < 0$** — note is 1 at 0
  Proof: Exercise
- **x is a stored pattern iff x is a stable point iff x is a fixed point iff $x = sgn(Wx)$**
- **Theorem: If a Hopfield network has only been trained on a single pattern x, then x is a stored pattern**
  Proof: Pick arbitrary $i \in |x|$. $sgn((Wx)_i) = sgn(\sum_j W_{ij}x_j) = sgn((\sum_j x_i x_j x_j) - x_i x_i x_i) = sgn((\sum_j x_i) - x_i x_i x_i)$ $[x_j \in \{-1, +1\}] = sgn((|x| - 1)x_i) = sgn(x_i) = x_i$. Thus, as i was arbitrary, $sgn((Wx)) = x$ as required.
- **Theorem: If a Hopfield network has been trained on multiple patterns, each may or may not be stable**
  Proof: Long
  Weaker version: If a Hopfield network has been trained on multiple patterns, a training pattern might not be stable
  Proof: Consider a HN trained on $P_1 = [-1, -1]$ and $P_2 = [-1, +1]$. Then, $W = (½)*([1] + [-1])) = [0]$. Thus, [1, 1] is the only fixed point

Time Series Data

Recurrent Neural Network (RNN)

- A recurrent neural network loops round an internal state in order to recursively process variable length sequential data — the loop can be unrolled for backpropagation through time
- A simple RNN has a single hidden recurrent layer followed by an ordinary readout layer
- A deep RNN has multiple hidden layers, potentially with multiple feedback loops or potentially with a single feedback loop
- **Let a be the output of a recurrent neuron, h be the feedback value and x be the input. Then at each time step t, $a_t = \tanh(W[h_{t-1} \# x_t])$** where # denotes matrix concatenation

Bidirectional RNN

- By default, RNNs are uni-directional, but sometimes we would like to “see the future” — for example to disambiguate a word based on the context of subsequent usage
- A bidirectional RNN has two copies of the same architecture and concatenates the outputs to produce pairs at each time step. The input is then fed into one copy in the normal order and one copy backwards

LSTM: Foundations

- Due to backpropagation through time, the effective network depth increases with every time step and so **exploding/vanishing gradients from long ago time steps can be a big issue when training RNNs (gradients >/< 1 grow/shrink exponentially in number of time steps)**
- **Gradient clipping can mitigate exploding gradients but to address vanishing we need gating** so we can manage the flow of data
    - Clipping to deal with exploding will still be needed with LSTM
- Extend our model of RNN to have a **memory cell between the output of one recurrent unit and the feedback input to the next**
    - Currently this simply stores the the value coming in and outputs the stored value

Long Short-Term Memory (LSTM)

- 3 gates control the flow of data through the memory cell of an LSTM
- **Let c_t, f_t, i_t o_t, a_t, h_t be the values at time t of the memory cell, forget gate, input gate, output gate, hidden layer output, and feedback into next time step respectively. Then:**
  - **For each gate g, $g_t = \sigma(W_g[h_{t-1} \# x_t])$** — other than σ instead of tanh (and gates having their own weights) this is the same as a_t
    - Sigmoid means values are continuous but squashed into 0 - 1
  - **$h_t = o_t * \tanh(c_t)$** where * denotes **element wise multiplication**
  - **$c_t = f_t * c_{t-1} + i_t * a_t$** — note forget gate 0 = forget everything, 1 = forget nothing
- The o_t term in h_t is quite directly influenced by h_{t-1} thus providing short term memory, whereas the $\tanh(c_t)$ provides long term memory thanks to the gating of i to making changes to c