Information Processing and the Brain

Josh Felmeden

January 5, 2022

Contents

1 Information Theory **4**
1.1 Randomness . . . 4

2 Shannon's entropy **4**
2.1 Nice properties . . . 5
2.1.1 Case of n = 2 . . . 6
2.1.2 Source coding . . . 6
2.2 Joint and Conditional Entropy . . . 7
2.2.1 Conditional Entropy . . . 7
2.3 Mutual Information . . . 8
2.4 Correlation . . . 9
2.5 The data processing inequality . . . 9

3 Information Theory in the Brain **10**
3.1 Information in the Brain . . . 10
3.1.1 Discretisation size . . . 10
3.2 Differential Entropy . . . 11
3.2.1 Gaussian distribution . . . 11
3.3 Relationship between Differential entropy and Shannon's entropy . . . 12
3.4 Mutual information for continuous probabilities . . . 12
3.5 Applications of differential entropy . . . 12

4 Infomax **12**

5 The Bayesian Brain **17**
5.1 Bayesian Fusion . . . 17
5.2 Kalman Filter . . . 18

6 Neural Circuits **20**
6.1 A feed-forward neural network . . . 21
6.2 Supervised learning: in depth . . . 22
6.2.1 Pyramidal cells . . . 24
6.2.2 Supervised learning in the Cerebellum . . . 26
6.2.3 Summary . . . 26

7 Visual System and Learning **26**
7.1 Simple cells . . . 27
7.2 Complex cells . . . 27
7.3 Grandmother cell . . . 27
7.4 Convolution Neural Networks . . . 28
7.5 Summary . . . 28

8 Reinforcement Learning **28**
8.1 Temporal Difference (TD) Learning . . . 29
8.2 Q-Learning . . . 30
8.3 Bigger spaces . . . 30
8.4 Deep Q-learning . . . 31

9 Unsupervised Learning **31**
9.1 Sparse Coding 31
9.2 Sparse Coding in the Brain 32
9.3 Autoencoders 33

10 Temporal processing and Recurrent Neural Networks (RNNs) **33**
10.1 Temporal Processing 33
10.2 Recurrent Neural Networks (RNNs) 34
10.2.1 Attractor Neural Networks 34
10.3 Reservoir computing 35
10.4 Recurrent CNNs 35

11 Microcircuits and RNNs **35**
11.1 Long Short-Term Memory (LSTM) 36
11.2 Summary 36

1 Information Theory

Information theory quantifies the amount of information that is potentially available from the communication channel using. So, we look to answer the question: can the receiver decide how informative the channel is likely to be; how much information is in the channel.

1.1 Randomness

The key is that if the information in the channel is predictable, the receiver is not going to learn much from this. Therefore, we will look at randomness (coin flipping, geiger counters, etc).

We will also look and unexpectedness, and consider the relationship between randomness, unexpectedness, and information theory.

Netflix is a good example of this. Their star rating of films helps them to recommend films to you, and use your experiences to recommend films to others. The recommended tab is the *channel* in the information theory. The films are the medium, and Netflix is communicating to the user information about the film. If Netflix recommends a really popular film, the information is predictable, and the user is not learning much. Therefore, we are not gaining much information about the quality of the films.

Another example is the star rating of movies. The star rating is not 'random' enough for the consumer, in that there is not enough information that we as the user can learn from this. A good film with a lot of hype will likely have a good star rating, and therefore we would already know this.

Therefore, we can say that a statement, despite being correct, can be useless if we can predict it.

> *The theory of information starts with an attempt to allow us to quantify the informativeness of information, but not its salience or validity.*

2 Shannon's entropy

For a finite discrete distribution with a random variable X, possible outcomes $\{x_1, , x_2, \ldots, x_n\} \exists \mathcal{X}$ and a probability mass function p_x giving probabilities $p_X(x_i)$, the entropy is:

$$H(X) = -\sum_{x_i \exists \mathcal{X}} p_X(x_i) \log_2 p_X(x_i)$$

This formula *quantifies* some information received. In the example of star ratings on netflix:

1 star	0.016
2 star	0.310
3 star	0.627
4 star	0.057

We obtain the following formula:

$$H(X) = -0.016 \log_2 0.016 - 0.31 \log_2 0.31 - 0.627 \log_2 0.627 - 0.057 \log_2 0.057 \approx 1.28$$

Choosing base two is arbitrary, but because we are dealing with information, it is useful to deal in bits. In other circumstances, it is possible to choose a different base. The result, 1.28 is the entropy for the channel of Netflix film star ratings.

If the star ratings were all equally likely, we would get an entropy of:

$$H(X) = -4 \times 0.25 \log_2 0.25 = 2$$

This is higher value entropy meaning the information is more useful.

In contrast, if all films are rated one star, we end up with entropy 0, meaning that it is useless. We assume $\log 0$ is 0 for shannon's entropy despite this not being the case in reality.

2.1 Nice properties

Shannon's entropy works on any sample space. It's good because you can calculate this on things that would otherwise be impossible, such as items bought from a grocers. This is because it is defined on probability space, while other calculations are calculated in vector space (such as mean, etc).

It's **always positive**, unless it is zero, which is when the distribution *isn't random* (aka *determined*).

If the distribution is *uniform*:

$$p_X(x_i) = \frac{1}{n}$$

for all x_i where ($\#$ is equivalent to `len`)

$$n = \#\mathcal{X}$$

then, since $-\log_2(1/n) = \log_2 n$

$$H(X) = \log_2 n$$

We won't prove it here, but is not difficult to prove that:

$$0 \leq H(X) \leq \log_2 n$$

with $H(X) = 0$ *only* if one probability is one and the rest zero, and $H(X) = \log_2 n$.

2.1.1 Case of n = 2

With two outcomes, a and b with $p(a) = p$ and $p(b) = 1 - p$ then

$$H = -p \log_2 p - (1 - p) \log_2 (1 - p)$$

The resulting graph of entropy will be symmetric; rising towards 1.

2.1.2 Source coding

the main reason to believe that Shannon's entropy is a good quantity for calculating entropy is its relationship to so called source coding.

Consider storing a long sequence of letters `A, B, C, D` as binary (AABACBDA for example). If we wanted to digitise this, we might use a very simple binary representation of these characters (`A = 00, B = 01, ...`). Using this representation means the average bits per letter is 2. Now, if we had different distribution of letters, then we could possibly come up with a smaller, more compact version of the code. For example, if we had the following distribution:

A	B	C	D
0.5	0.25	0.125	0.125

We could use the following encoding:

A	B	C	D
0	10	110	111

Don't forget that this resulting code should be *prefix free*.

Now, we can prove that this code is shorter on average:

$$L = 0.5 \times 1 + 0.25 \times 2 + 0.125 \times 3 + 0.125 \times 3 = 1.75 < 2$$

This is also the same as the Shannon's entropy of the code. Each of the logs are giving us the length of the corresponding code word.

The source coding theorem states that, for the most efficient code, we get:

$$H(X) \leq L < H(X) + 1$$

This means that if you have a sequence of objects that have some regularity and you want to code them into binary, the channel's entropy is a lower bound on the average length of a message and you can get a *prefix free* code within one of Shannon's entropy.

2.2 Joint and Conditional Entropy

Typically, we want to use information theory to study the relationship between two random variables. So far, we have only looked at single variables.

Joint Entropy

Given two random variables X and Y, the probability of getting the pair (x_i, y_j) is given by the **joint probability** $p(X,Y)(x_i, y_j)$. THe joint entropy is just the entropy of the join distribution:

$$H(X,Y) = -\sum_{i,j} p_{X,Y}(x_i, y_j) \log_2 p_{X,Y}(x_i, y_j)$$

This is essentially the probability that x will produce x_i and that y will produce y_j

Here's an example:

	x_0	x_1
y_0	$1/4$	$1/4$
y_1	$1/2$	0

$$H(X,Y) = -\frac{1}{2}\log_2\frac{1}{4} - \frac{1}{2}\log_2\frac{1}{2} = \frac{3}{2}$$

2.2.1 Conditional Entropy

$p_{X|Y}(x_i|x_j)$ is the **conditional probability** of x_i given y_j. If we know that $Y = y_j$, it gives us the probability that the pair is (x_i, y_j).

$$p_{X|Y}(x_i|y_j) = \frac{p_{(X,Y)}(x_i, y_j)}{p_Y(y_j)}$$

The **marginal probability** for x is: $p_X(x_i) = \sum_j p_{(X,Y)}(x_i, y_j)$. This is the way of getting the probabilities for one of the two random variables from the joint distribution.

Let's now substitute the conditional probability into the formula for entropy:

$$H(X|Y = y_j) = -\sum_i p_{(X|Y)}(x_i|y_j) \log_2 p_{X|Y}(x_i, y_j)$$

This is the entropy of X as we know $Y = y_j$. We call this the **conditioned entropy**.

The conditional entropy is the average amount of information still in X when we know Y. It also has some pretty nice properties:

If X, Y are *independent*, then:

$$p_{X,Y}(x_i, y_j) = p_X(x_i)p_Y(y_j)$$

For all i, j and

$$p_{X|Y}(x_i|y_j) = p_X(x_i)$$

so

$$H(X|Y) = -\sum_{i,j} p_{X,Y}(x_i, y_j) \log_2 p_{X|Y}(x_i|y_j) = H(X)$$

Which is what we want, since if Y tells us nothing about X, then the conditional entropy should just be the same as the entropy of X.

Conversely, if X is *determined* by Y, then $H(X|Y) = 0$. This could happen if the only x_j, y_i pairs that actually occur are (x_i, y_i).

2.3 Mutual Information

Given two (overlapping) variables, there will be some information that is shared by these two entropies, and this is called **mutual information**, and is defined as:

$$I(X, Y) = H(X) + H(Y) - H(X, Y)$$

Or:

$$I(X, Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

By substituting in the formulas, we end up with:

$$I(X, Y) = \sum_{i,j} p_{X,Y}(x_i, y_j) \log_2 \frac{p_{X,Y} x_i y_j}{p_X(x_j)p_Y(y_j)}$$

Going back to the previous example:

	x_0	x_1
y_0	$1/4$	$1/4$
y_1	$1/2$	0

This has $H(X, Y) = 3/2, H(X) \approx 0.81$ and $H(Y) = 1$ so $I(X, Y) \approx 0.31$

If X and Y are independent, we just end up with $I(X, Y) = 0$. In fact, $I(X, Y) \geq 0$

2.4 Correlation

The correlation is defined as:

$$C(X,Y) = \frac{\langle (X-\mu_X)(Y-\mu_Y)}{\sigma_X \sigma_Y}$$

Where μ_X is the average of X and σ_X is the standard deviation.

Correlation only works if X, Y take their values in vector space (meaningfully be able to multiply the two together).

Consider:

	-1	0	1
1	1/4	0	1/4
0	0	1/2	0

Then:

$$C(X,Y) = 0$$

Whereas $I(X,Y) = 1$.

2.5 The data processing inequality

There is something called **conditional independence**. Imagine playing a game of snakes and ladders, and X, Y, Z are the resulting position after moves 1,2,3 respectively. Knowing X will change the probability of Z, but only if we don't know Y, since if we know Y, it doesn't really matter what X was. This means that X and Z are conditionally independent (conditioned on Y).

The data processing inequality states that if:

$$X \to Y \to Z$$

then:

$$I(X,Y) \geq I(X,Z)$$

With equality if and only if $X \to Z \to Y$

3 Information Theory in the Brain

3.1 Information in the Brain

Spike trains are when neurons spike with some level of voltage when they are activated. This spike passes from neuron to neuron, which is how information is passed along an axon.

We can use Shannon's entropy to look at the similarities in these spikes to see the shared information.

An experiment was performed on a fly where it was shown a series of bars on a TV screen and the spike trains from the neurons at the back of the brain (the ones that observe horizontal movement) was recorded. More specifically, the timing of the spikes was recorded. This information was then *discretised*, and evaluated. The information recorded are the action spikes. *Discretisation* meant that the information was looked at in 3ms, and then return a 1 if there is a spike, or a 0 if not (think time bins). This binary sequence was then split up into words. Each word corresponds to the amount of time that an entire piece of information has been encoded. The calculations performed revealed that the neurons that were being looked at only remember things in 30ms, so there was a natural 'word length' for splitting up the binary sequence, thus 30ms was taken.

These words can now be converted into a set of objects. The random variable that is taken as the communication channel is a chunk of spike train for 30ms, represented by one of the words. Now, we want to look at the probability of different words:

$$p(w_0) \approx \frac{\#(\text{occurance of } w_0)}{\#(\text{trials})}$$

Now, two different tests were performed. The first is where the stimulus is random. The second is where a fixed five minute segment of the stimulus is shown ($H(W)$), and the responses are considered ($H(W|S)$).

Now, we can get the mutual information about the words given:

$$I(W, S) = H(W) - H(W|S)$$

Or, the information about S is the total information in W subtract the noise.

3.1.1 Discretisation size

One important question is how to decide how small to make hte discretisation time and how long to make the words. In our example, given that $\delta t = 3ms$ gives 78 ± 5 bits per second or 1.8 ± 0.1 bits per spike.

Unfortunately, if we have a 30ms word with 3ms letters, this gives us 10 letters per word giving $2^{10} = 1024$. If six seconds of data are used for the repeating stimulus, that is 100 different stimuli, then even for a three hour recording, there are 1800 trails for each stimulus. This is not a big amount for estimating 1024 probabilities.

3.2 Differential Entropy

Differential Entropy is the name given to Shannon's entropy for continuous probability distributions where the sample space is $\mathcal{X} \subseteq \mathbf{R}^d$. In the examples we will be looking at, $d = 1$. The idea is that: $h(X) = \int dx p(x) \log_2 p(x)$. This is a good guess as to what the continuous distribution would look like of Shannon's entropy.

If we consider a uniform distribution where:

$$p(x) = \begin{cases} 1/a & x \in [0, a] \\ 0 & \text{otherwise} \end{cases}$$

So

$$h(X) = \int_{-\infty}^{\infty} dx p(x) \log_2 p(x) = -\frac{1}{a} \int_0^a dx \log_2 \frac{1}{a}$$

And so:

$$h(X) = \log_2 a$$

This, unfortunately, does not guarantee that the result will be positive, which is one of the useful properties of Shannon's entropy.

3.2.1 Gaussian distribution

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-x^2}{2\sigma^2}}$$

If we substitute and integrate by parts, we get:

$$h(x) = \frac{1}{2} \log_2 2\pi e \sigma^2$$

As you expand the distributions, we can prove that for fixed variance, Gaussian has the highest entropy.

Densities are not probabilities. The discrete case $p(x)$ is the probability that $X = x$. For the continuous case:

$$\int_{x_0}^{x_1} dx p(x)$$

is the probability that $x_0 \leq x \leq x_1$. The usual sums and probabilities go to integrals and densities doesn't work because there is a p in the log.

We also need to model the sensitivity of the receiver as well as the behaviour of the source.

3.3 Relationship between Differential entropy and Shannon's entropy

Let $\delta = x_{i+1} - x_i$. Consider the discrete random variable X^δ. Now, instead of the continuous distribution we had before, we now have a discrete representation of the interval (differentiation by first principle (by faking it)). A more elegant approach to this is the **mean value theorem**. This is where you choose two heights for the histogram: the first being the lowest possible point, and the other being the highest, and then choosing the mean point of this.

3.4 Mutual information for continuous probabilities

The differential mutual information is:

$$I(X,Y) = \int p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)}$$

And satisfies the same identities as the mutual information:

$$I(X,Y) = h(X) + h(Y) - h(X,Y)$$

It has some advantages over the differential entropy; in particular, it is invariant under changes of variable in X and Y. The differential mutual information is the same as the mutual information in the sense that, using the notation above:

$$I(X^{\delta x}, Y^{\delta y}) \to I(X,Y)$$

as δx and δy approach zero. Furthermore,

$$I(X,Y)) \geq 0$$

3.5 Applications of differential entropy

The sick part of entropy is its relationship to communication through the source coding theorem. At first, it might appear that there is no analogous set of ideas for differential entropy since, in a sense, the amount of information encoded in the outcome of a continuous variable must be infinite if read to infinite precision. However, of course, in real world communication, nothing is read to infinite precision and there is a theory of communication using continuous signals which includes the signal noise and imprecision of signal transmission. we won't look at this here, and instead consider the example of infomax.

4 Infomax

This is another information theory example we will consider. It is an approach to *unmixing* data, and gives an in-principle explanation for how the brain might perform auditory source separation. In other words, it shows how source separation might be performed.

The problem is as follows: imagine you are in a crowded room (classically at a *cocktail party*. Lots of people are talking, but if oyu concentrate on one voice at a time, you can separate it from the racket. The opposite effect can be observed when meditating or sitting in thoughtful silence, where we suddenly notice how loud distant noises are. These sounds are filtered without thinking. This is called auditory source separation and the question of how to do this is called the **cocktail party problem**.

The problem can be formalised. Given the sources $\mathbf{s}(t)$ where $\mathbf{s}$ is a vector over multiple sources. Now we do source separation with only two recordings, one for each ear. Here, we are just going to consider the simpler problem of source separation when there are many recordings as there are sources, we also assume the mixin is linear and instantaneous. Real mixing of auditory sources in a room will only have these properties approximately. Using this assumptions, we have:

$$\mathbf{r}(t) = M\mathbf{s}(t)$$

with M being a square mixing matrix. Our goal is to find the unknown source signals $\mathbf{s}(t)$ from the known recordings $\mathbf{t}$

Although it makes little difference, we will restrict ourselves to two sources, so $\mathbf{s}$ and $\mathbf{r}$ are two-dimensional vectors and M is a two by two matrix.

We assume the two sources are independent $p_{s1,s2} = p_{s1}p_{s2}$; the point of source separation is to unmix independent sources. Now, we want to find the unmixing matrix W so that knowing:

$$\mathbf{x}(t) = W\mathbf{r}(t)$$

is as good as knowing the sources. Since we know $\mathbf{x} = WM\mathbf{s}$, we want WM to be a diagonal matrix multiplied by a permutation matrix. Changing the amplitude of the source, or reordering doesn't matter. This means that

$$WM = \mathsf{diag}(d_1, d_2)$$

or

$$WM = \begin{pmatrix} 0 & d_1 \\ d_2 & 0 \end{pmatrix}$$

where d_1 and d_2 are real numbers. Hence:

$$\mathbf{s} \rightarrow^{\text{mixing}} \mathbf{r} = M\mathbf{s} \rightarrow^{\text{unmixing}} \mathbf{x} = W\mathbf{r}$$

One difficulty with looking at this problem is that it involves continuous random variables, while we have so far only looked at the discrete case. For this reason, some of the results will just be quoted. Some changes are that we get:

$$h(X) = -\int p(x) \log p(x) dx$$

But it is no longer always positive.

now, the idea is to solve the problem by using the face that S_1 and S_2 are independent. We only need to find W so that X_1 and X_2 are also independent. One approach could be to decorrellate the random variables:

$$C(X_1, X_2) = \langle (X_1 - \langle X_1 \rangle)(X_2 - \langle X_2 \rangle) \rangle_{(X_1, X_2)}$$

where the expectation value for continuous random variable has the obvious definition:

$$\langle g(X) \rangle = \int p_X(x) g(x) dx$$

It is easy to check that the correlation vanishes if X_1 and X_2 are independent, however, the flaw in this approach is that the converse is not true. It is possible to have zero correlation while still having statistical dependence. to see this, imagine that EX_1 and EX_2 are zero and that we have chosen W so that the correlation matrix is the identity:

$$C_a b = C(X_a, X_b) = 1$$

It is then easy to see that rotations:

$$\begin{pmatrix} X_1' \\ X_2' \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ = \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} X_1 \\ X_2 \end{pmatrix}$$

do not change the correlation matrix. For this reason, the decorrelation prescription has a rotational ambiguity and something more is needed. That is to require that $I(X_1, X_2) = 0$ since this happens if and only if X_1, X_2 are independent. The problem is that $I(X_1, X_2)$ is pretty had to calculate. So, infomax basically just looks at $h(X_1, X_2)$:

$$I(X_1, X_2) = h(X_1) + h(X_2) - h(X_1, X_2)$$

Maximising the joint entropy $h(X_1, X_2)$ will give a *minimum* of the mutual information, meaning the variations in the individual entropies $h(X_1)$ and $h(X_2)$ can be ignored. Unfortunately, due to the possibility of trivially scaling the entropy, $X_a \to \lambda X_a$ causes the joint entropy $h(X_1, X_2) \to h(X_1, X_2) + \log|\lambda|$ so $h(X_1, X_2)$ can be made arbitrarily large by scaling, something that tells us nothing about mixing and unmixing. Inspired by the behaviour of neurons, this is solved by adding a saturation non-linearity:

$$y_1 = g(x_1 + w_1)$$
$$y_2 = g(x_2 + w_2)$$

where

$$g(u) = \frac{1}{1 + e^{-u}}$$

Is a saturating non-linearity so $g : (-\infty, \infty) \to (0, 1)$

This leaves us with:

$$\mathbf{s} \rightarrow^{\text{mixing}} \mathbf{r} = M\mathbf{s} \rightarrow^{\text{unmixing}} \mathbf{x} = W\mathbf{r} \rightarrow^{\text{non-linearity}} \mathbf{y} : y_a = g(x_a + w_a)$$

From here on, we will write

$$y_a = g(x_a + w_a) = f(r_1, r_2 : W, w_1)$$

where f is the function parameterised by W and w_a mapping from the recording to y. Before considering the source separation problem, lets look at the effect of the non-linearity on its own. We consider the one-to-one case:

$$r \rightarrow^{\text{multiply}} x = Wr \rightarrow^{\text{non-linearity}} y = g(x + w) = f(r; w, W)$$

Where W and w are now both scalars and r, x, y are outcomes for random variables R, X, Y. We consider maximising the entropy $h(Y)$, which maximises the information in Y about R:

$$I(R;Y) = h(Y) - h(Y|R)$$

However, $h(Y|R)$ is constant since R determines Y. In the discrete case, we are familiar with, this would be easy to discuss since it would just be zero. In the continuous case, its not that simple. Its actually negative infinity, but the consequence is the same (it doesn't depend on W and w). To maximise $h(Y)$ we need to calculate derivatives and these are calculable and well defined, even if the quantity being differentiated is not. In this case:

$$h(Y) = -\int p(y) \log p(y) dy$$

and this is estimated by

$$h(y) = -\log p(y)$$

In other words, if n values of y are drawn from Y then

$$\frac{1}{n} \sum_i h(y_i) \rightarrow h(Y)$$

as n gets large.

Of course, we do not have $p_Y(y)$ and it would be difficult to estimate, but we actually don't need it to get hte derivative of $h(y)$. We have seen that already since $y = f(r; W, w)$

$$p_Y(y) = \frac{p_R[r = f^{-1}(y)]}{|f'(f^{-1}(y))|}$$

so

$$h(y) = -\log p_R(r) + \log |f'|$$

and $p_R(r)$ is independent of the parameters. Now, for our choice of saturating and non-linearity

$$g(u) = \frac{1}{1+\exp(-u)}$$
$$\frac{dg}{du} = g(1-g)$$

and hence

$$\log|f'| = \log W + \log f + \log(1-f)$$

Now we know f:

$$f = g(Wr + w)$$

so

$$\frac{df}{dW} = rf(1-f)$$

and hence

$$\frac{dh(y)}{dW} = \frac{1}{W} + \frac{1}{f}rf(1-f) - \frac{1}{1-f}rf(1-f) = \frac{1}{W} + r(1-2y)$$

Similarly

$$\frac{dh(y)}{dw} = 1 - 2y$$

these quantities r, y, W are numbers we have access to:

- W is a parameter,
- r is the recorded signal
- we can sample $r(t)$ at a set of times to get a set of rs
- y is a function of s.

Now, we choose a starting W and w and estimate the gradient and then change W and w a small amount, repeating until the optimum values are found. The optimum value would look like

$$f(r) = \int_{-\infty}^{r} p_R(u)du$$

Now, we don't know $p_R(r)$ and we chose $f(r)$ at the start, here it is a member of a two-parameter family of functions parameterised by W and w. Ideally, if the derivative of the saturating non-linearity is somewhat close to the distribution of R, then infomax will find the W and w that line everything up so that Y will have something close to an even distribution.

In out two to two case, we want to maximise $h(Y_1, Y_2)$, the idea being that this should find a matrix W whose eigen-directions give statistically independent Y_a, this is the bit we want since it will make X_a independent. Doing this calculation gives:

$$\frac{dh(y)}{dW_{ab}} = (W^T)^{-1}_{ab} + r_a(1 - 2_{yb})$$
$$\frac{dh(y)}{dw_a} = 1 - 2_{ya}$$

allowing the maximum of $h(Y_1, Y_2)$ to hopefully be found. This should unmix the signal.

5 The Bayesian Brain

Here, we will look at the idea that the brain performs Bayesian inference on data.

Remember Bayes' rule:

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)}$$

Imagine there is a machine that detects if the sun has exploded, but before it tells you the answer, it rolls two dice and then if it gets two sixes it lies. If the machine says the sun has exploded, the probability that the sun has actually exploded is obviously not very high, but if you look at it a certain way, (probability that the machine is lying given sun has exploded is low) then you could convince yourself that the sun HAS exploded.

5.1 Bayesian Fusion

The brain is constantly making decisions based on fusion of different inputs. For example, we can make estimates of an objects height based on feel and sight. This estimate is based on a combination of both of these sources of input. If we define the deviation from haptic feedback as σ_h and from visual feedback as σ_v and the variation of the estimate of the TRUE value σ, we get the following:

$$\frac{1}{\sigma^2} = \frac{1}{\sigma_v^2}\frac{1}{\sigma_h^2}$$

Now, let

$$\lambda = \frac{\sigma^2}{\sigma_h^2}$$

then

$$1 - \lambda = \frac{\sigma^2}{\sigma_v^2}$$

so

$$\bar{x} = (1 - \lambda)v + \lambda h$$

This shows how the standard deviation affects bayesian fusion, as those with a smaller variance will be weighted highly.

5.2 Kalman Filter

The Kalman filter works like the following example. At time t we have a belief about position and speed of an object. We believe that they are $\bar{x}$ but our estimate is uncertain, since $\bar{x}$ is the mean of a two-dimensional Gaussian distribution with covariance P representing our belief.

We want to update this to new values for time $t + \delta t$ using two noisy pieces of information: dead reckoning and direct measurement. The new belief will have mean $\bar{x}_n$ and covariance P_n

The position changes according to the standard law of motion:

$$s \to s_a = s + v\delta t$$

We are assuming speed is constant, but we can not assume this by including a control vector detailing the changes of speed.

Dead reckoning is defined as:

$$X_d = FX + W$$

where F is the motion matrix

$$F = \begin{pmatrix} 1 & \delta t \\ 0 & 1 \end{pmatrix}$$

and W is the zero mean Gaussian noise with covariance matrix Q.

If we take the average:

$$\bar{x}_d = F\bar{x}$$

This tells us how we update the mean we use to describe the distribution.

To look at the **covariance**, we will look at a one-dimensional example. Let's say we have a variable Y:

$$Y \sim \mathcal{N}(\bar{y}, p)$$

And say the true update has the form:

$$y_a = fy$$

where f is a constant, so:

$$Y_d = fY + U$$

where

$$U \sim \mathcal{N}(0, q)$$

Since y_d is drawn from the sum of two gaussians, it is also gaussian with mean and variance that can be calculated directly:

$$\bar{y}_d = \langle Y_d \rangle = \langle fY + U \rangle = f\langle Y \rangle = f\bar{y}$$

and variance:

$$\begin{aligned}\langle (Y_d - \bar{y}_d)^2 \rangle &= \langle (fY + U)^2 \rangle - \bar{Y}_d^2 = f^2\langle Y^2 \rangle + \langle U^2 \rangle + \langle U^2 \rangle - f^2\bar{y}^2 \\ &= f^2(p + \bar{y}^2) + q - f^2\bar{y} = f^2 p + q\end{aligned}$$

For two-dimensional case, we have:

$$P_d = FPF^T + Q$$

In our one dimensional example, with ys where the xs are, little letters for variances, the equivalent of the covariances before:

$$p(y_a|y_d, y_s) \propto p(y_d|y_a)p(y_s|y_a)$$

where $p(y_d|y_a) \sim \mathcal{N}(y_a, p_d)$ and $p(y_s|y_a) \sim \mathcal{N}(y_a, r)$

This is just Bayesian fusion (from before):

$$\frac{1}{p_n} = \frac{1}{p_d} + \frac{1}{r}$$

and gives new mean:

$$\bar{y}_n = \frac{p_n}{p_d} y_d + \frac{p_n}{r} y_s$$

with this new mean we can rewrite

$$\frac{p_n}{p_d} = 1 - \frac{p_n}{r}$$

so

$$\bar{y}_n = y_d + k(y_s - y_d)$$

where

$$k = \frac{p_n}{r}$$

This gives us Kalman gain:

$$\bar{y}_n = y_d + k(y_s - y_d)$$

where

$$k = \frac{p_n}{r} = \frac{p_d}{p_d + r}$$

and

$$\frac{1}{p_n} = \frac{1}{p_d} + \frac{1}{r}$$

and finally, after some rejigging:

$$p_n = \frac{rp_d}{p_d + r} = \frac{p_d(r + p_d)}{p_d + r} - \frac{p_d^2}{p_d + r} = (1 - k)p_d$$

Back to two-d:

$$S = P_d = R$$

and

$$K = P_d S^{-1}$$

This factor, called the **Kalman gain** is the analogue of k above:

$$\bar{x}_n = x_d + K(x_s - x_d)$$

and

$$P_n = (1 - K)P_d$$

6 Neural Circuits

The brain learns by taking visual input, preparing movement, and assigning an appropriate kinetic output. We use synapses to assign credit to *parameters* in the brain. The image is the input, and is processed by the **primary visual cortex** (V1). The **secondary visual cortex** (V2) is then used to process this further, and this is finally passed to the **frontal cortex** which is the decision making centre.

There are three forms of *credit assignment*:

- Supervised learning
  - Relies of a teaching signal
- Unsupervised learning
  - Extracts useful representations of input
- Reinforced learning
  - Learn to navigate/survive an environment
  - Usually using some kind of reward fed back into the system to change the parameters

6.1 A feed-forward neural network

The brain is like a tropical forest with lots of different neuron types and architectures. In theoretical neuroscience, we need to abstract out some of this complexity to get at the principles of information processing in the brain. We therefore need to classify input into different categories.

For *supervised learning*, we have some output $v = f(wu)$, where:

- v: output
- w: weights
- u: input

In nature, this might be working out whether something is a predator, with an target $y = \{1, 0\}$.

Supervised learning is just minimising some *cost* function $(v - y)^2$. The learning rules for the weights are derived from this cost function.

For *unsupervised learning*, there is no such assumption. We have:

- v: output
- h: hidden layer
- u: input

The hidden layer is of lower *dimensionality* than the other layers. This means that our network is forced to learn a lower dimension representation of the input, thereby capturing the key information of the input.

Again, we are minimising the cost function $(v - u)^2$. The learning rules for the weights are again derived from the cost function.

For *reinforcement learning*, we need to find the best policy to maximise the rewards. For the parameters at time t, we have:

- a_t: action
- r_t: reward
- s_t: state

Using an example of gridworld, where the world is a grid with recharge points and bombs, we might have a table with different reward values for each square in the grid. We update the table with

temporal difference learning:

$$V(S_t) = V(S_t) + (R_{t+1} + \lambda V(S_{t+1}) - V_t)$$

Where:

- $V(S_t)$ is the value
- R_{t+1} is the reward
- $V(S_{t+1})$ is the future value
- λ is the discount factor

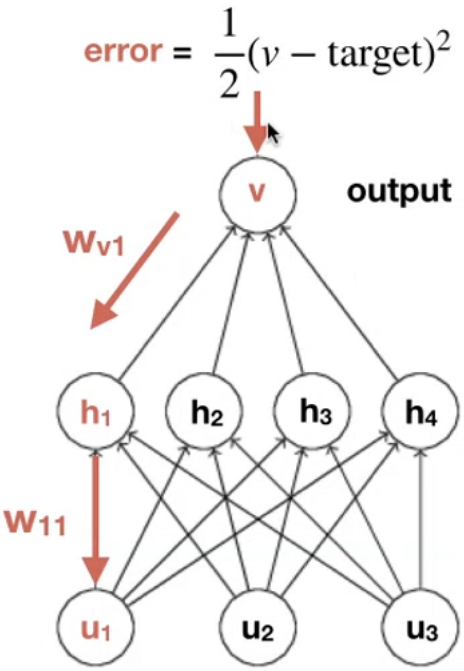
6.2 Supervised learning: in depth

Supervised learning can be difficult to train. The solution ot this is the *backprop algorithm*, although this is not explicitly bound to supervised learning.

Backpropagation is an algorithm used to train artificial neural networks that calculates a gradient from the error function with respect to a given parameter (i.e. weight). With htis gradient, errors are effectively back propagated during training throughout the network.

$$\Delta w = \eta(v - \text{target})u$$

It is a generalisation of the **delta rule** (above) to deep feedforward networks, which relies on the use of the chain rule to compute gradients for each layer.



Backpropagation is typically used to perform gradient descent optimisation, adjusting the weights of neurons while optimising a desired loss function (error).

The backpropagation algorithm first calculates forward activity. The output is $v = W_v h$ (linear activation for v), and hidden layer is $h = f(W_u u)$ where f denotes the activation function which is typically modelled as sigmoid or Rectifying linear unit function (ReLU).

Next, it calculates **error**. The error signal is typically modelled as:

$$\frac{1}{2}(v - \text{target})^2$$

Finally, we **backpropagate** this error (how to calculate this gradient with respect to w_{11}):

$$\frac{\delta \text{Error}}{\delta w_{11}} = -\eta \frac{\delta \text{Error}}{\delta w_{11}} \frac{\delta v}{\delta h_1} \frac{\delta h_1}{\delta w_{11}}$$

Resulting in:

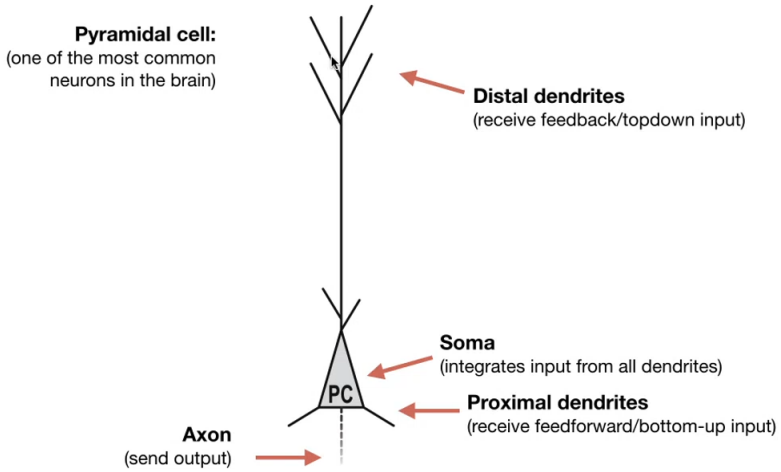
$$\Delta w_{11} = -\eta \frac{\delta \text{Error}}{\delta w_{11}} = (v - \text{target}) w_{v1} f'_{h_1} u_1$$

Unfortunately, at first glance, backpropagation in the brain does not seem to be biologically plausible. The key issues are:

1. Weight transport problem: Suggests that feedback weights = feedforward weights
    - The chain rule predicts feedback weights that are equal to feedforward weights, but there is no evidence that feedback connections exactly mirror feedforward ones
    - The solution to this remains unclear, but it has been found that exact feedback weights can be replaced by random weights, without a major impact on performance. This method is known as feedback alignment and suggests that the brain may not need an exact feedback signal to implement 'backprop'.
2. Derivative of activation functions: relies on calculating derivatives of activation functions
    - Backprop relies on the derivative of the activation function, but it is not clear how neurons could calculate their own derivative.
    - The solution to this is also unclear, but as long as the gradient direction is not impacted, the derivative is not critical for performance.
3. Two phase learning: feedforward propagation of activity and error backpropagation
    - Backprop relies on two phases, a feedfoward pass of activity and then a backpass of errors. In the brain, there is no clear separation between perception and learning.
4. Separate error network: Suggest the need for a separate biological network computing the learning rules
    - The chain rule in backprop can be seen as a separate network, but there is minimal evidence for the existence of such gradient or error networks.
5. Non-local learning rules: The weight update depends on non-local information
    - Backprop relies on non-local learning rules, but learning is believed to occur through local changes at synapses, which is a process known as synaptic plasticity which can only access locally available information.
6. Target: the target label guides supervised learning.

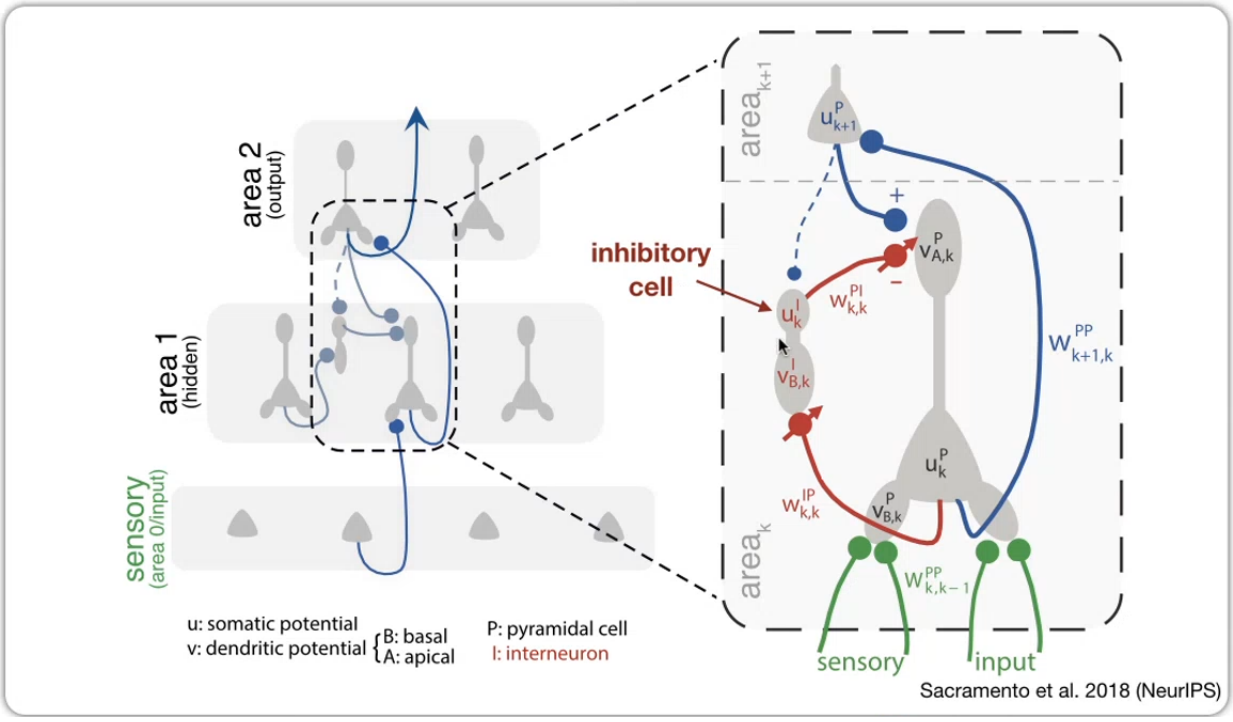
6.2.1 Pyramidal cells

We will now take inspiration from real neurons (pyramidal cells). These are one of the most common neurons in the brain.



The the brain, we have multiple areas, each made up of multiple pyramidal cells. This is the inspiration from which we get our new models of computer representations of learning in the brain. There are two cell types in the cortex, **Pyramidal** cells (excitatory cells) and **Martinotti** cells (inhibitory cells). These two work together.

Dendritic microcircuits are those that approximate backpropagation, and look like this (NOTE!: u and v have a different meaning from before!):



The general idea of this is the difference between the topdown activity from u^P_{k+1} and lateral inhibition u^I_k generates an **error** signal: $v^P_{A,k} \sim u^P_{k+1} - u^I_k$

The **error** signal triggers changes in bottom weights $w^{PP}_{k,k-1}$.

The model learns recursively across areas/layers (just like in backprop) to correctly predict an output. During this process, lateral inhibition learns to predict the top-down input, leading to a zero error after learning, i.e.: $u^I_k \sim u^P_{k+1}$

The general form of the local learning rule is

$$\frac{d}{dt}w \sim \eta(u - v)r$$

Where:

- u: somatic activity
- v: input
- η: learning rate
- r: presynaptic activity

The learning rule for the bottom-up synapses is:

$$\frac{d}{dt}w^{PP}_{k,k-1} \sim \eta(u^P_{k+1} - u^P_k)r^P_{k-1}$$

Note that this is really similar to the backprop learning rule:

$$\Delta w^{PP} \sim \eta \text{ error}_k \; r^p_{k-1}$$

By being closer to biology, we were able to solve three of the key problems we discussed above:

- Two phase learning: No need to two separate phases
- Separate error network: Neuron encodes both activity and errors
- Non-local learning rules: learning is local

Finally, if we look at the problem of the *target* we know that backprop needs a specific target (or *teaching signal*). It is really hard to imagine how the brain would be able to do this. One solution is that backprop works in unsupervised learning, so we can use that. Another solution is that specific brain areas may be calculating teaching signals that are used for internal supervised learning at other brain areas.

6.2.2 Supervised learning in the Cerebellum

The **cerebellum** has a stereotypical structure and contains more neurons than the rest of the brain. It is typically involved in motor error correction, but growing evidence suggests that it is also involved in regulating many other aspects of behaviour.

It seems to use a form of supervised learning but may also provide teaching signals to the cerebral cortex. So, the cerebellum can be used as an adaptive filter or controller.

The cerebellum's circuit looks really similar to reinforcement learning, because the forward model has a prediction that it compares with the outcome.

The fact that the cerebellum has so many neurons leads us to believe that it could be used in much more than simple motor control. A 2D cognitive map of the cerebellum shows that there is activation to many different tasks.

6.2.3 Summary

In summary:

- Deep learning successes rely on the backprop algorithm.
- Backprop is slowly becoming a possible model for learning in the brain, but several key issues remain unsolved.
- Dendritic microcircuits provide a powerful model of learning in the brain that approximates backprop.

7 Visual System and Learning

Our visual system works by projecting the image of the world onto the back of the eye (the retina) which is then relayed by the optic tract and sent to the visual cortex at the back of the brain. We can then record individual neurons and hope they respond to the visual stimulus presented to the brain.

The visual cortex has a few sub-areas. The primary visual cortex, the secondary visual cortex, and the inferior temporal cortex. The visual cortex also has receptive field which is a region of sensory space in which as stimulus modifies the firing rate of the neuron. The simple model of a receptive field is just a grid (visual input, u). This is then relayed via weights (w) to the visual cortex neuron (v), giving us this familiar equation $v = f(wu)$.

7.1 Simple cells

A *simple cell* is a representation of a cell in the primary visual cortex. We can give a test subject a certain orientation of a white bar on a black background and see which neurons react to the particular stimulus on a particular location. If the bar is moved, the neuron might not respond as much, or might be a bit delayed. If the bar orientation is moved, the neuron may not fire at all.

The classic model for a simple receptive field (as seen in simple cells) is a **gabor filter**:

$$w \sim g(x, y; \lambda, \theta, \phi, \gamma) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right)\cos\left(2\pi\frac{x'}{\lambda} + \phi\right)$$

where:

- x, y are coordinates
- σ is the standard deviation
- γ defines the aspect ratio. If $\gamma = 1$, then the kernel is very circular. If $\gamma \sim 0$, then the kernel is elliptical in one direction, and almost 0.
- θ is wavelength
- ϕ is the phase offset

A Gabor filter is a product of a *Gaussian* and *sinusoid* that can model a periodic pattern similar to simple cells.

7.2 Complex cells

Complex cells are able to fire neurons even if the stimulus is moved. However, if the orientation changes, it stops responding. Complex cells can be modelled as a set of simple cells with the same orientation but different locations, and simple cells can be modelled with Gabor Filters.

Generally speaking, simple cells are encoded in lower levels, such as V1, while complex cells are encoded higher up in places like V2. As you go higher and higher, you can have more complex components, such as edge cells (that encode edges) or even face cells (that encode faces).

7.3 Grandmother cell

This leads us to the concept of a grandmother cell. The idea here is that there are neurons responding specifically when you see people you are familiar with.

7.4 Convolution Neural Networks

CNNs can be used as a model of the visual system. This is a type of supervised learning, but the visual system is *unsupervised*. CNNs along with backprop have revolutionalised computer vision nonetheless. The performance is unmatched.

The two key operations are:

- Spatial convolution: Scanning the image with a filter
- Pooling: Subsampling input

CNNs learn hierarchical features similar to the ones found in the brain.

7.5 Summary

- The visual cortex is spread across multiple areas
- Receptive fields of increasing complexity
- Mathematical models of simple and complex cells
- CNNs as a representation of the visual system.

8 Reinforcement Learning

Reinforcement learning is the most common form of learning in the animal kingdom. The goal of this type of learning is to learn what actions to take maximise the rewards.

Some rewards are really sparse (such as winning or losing a game of something), and this poses a challenge because it's hard to realise what moves made should be rewarded since the final reward is the result of countless moves.

The general framework is the **Markov Decision Process**. In a basic form, this means that 'the future is independent of the past, given the present'. Mathematically, t his can be defined as the following transition probability of the future given the immediate past:

$$P(S_{t+1}|S_t = P(S_{t+1}|S_1, \ldots, S_t))$$

It assumes that the current state S_t has all the information needed to make a decision about the next state S_{t+1}.

In a Markov decision process, we define this by the following tuple:

$$(S, A, P, R, \gamma)$$

Where:

- S: Set of states

- A: Set of actions
- P: Transition probability $P^a_{ss'} = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$
- R: Reward function $R^a_s = \mathbb{E}(T_{t+1} = s' | S_t = s, A_t = a)$
- γ: Discount factor $[0, 1]$. This is usually set to 0.9 so the system slowly forgets the previous rewards

The goal is the find the **policy** that finds the maximises **discounted reward**.

A policy is the distribution over actions given states. It defines the behaviour of an agent:

$$\pi(a|s) = P(A_t = a | S_t = s)$$

A discounted reward is the total discounted reward received by the agent:

$$R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots = \sum_{t=0}^{\infty} \gamma^t r_t$$

It is useful to define the **value** of a state as the expected cumulative reward:

$$V^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t r_t | s_0 = s, \pi \right]$$

For that we have the *Bellman equation* which gives the expectatino over the policy for t and $t + 1$:

$$V^\pi(s) = \mathbb{E}_\pi[r_t + \gamma V^\pi(S_{t+1}) | S_t = S]$$

8.1 Temporal Difference (TD) Learning

TD Learning: is model-free reinforcement learning method that learns by bootstrapping based on current value function estimates. From the *Bellman equation*, we get an iterative method for updating hte value function:

$$V(S_t) = V(S_t) + (R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$

This gives us an estimate for the future rewards.

Tabular TD(0) for estimating v_π

Input: the policy π to be evaluated.
Algorithm parameter: step size $\alpha \in (0, 1]$
Initialise $V(s)$ for all $s \in S^+$, arbitrarily except that $V(terminal) = 0$
Loop for each episode:

- Initialize S
- Loop for each step of episode:
  - $A \leftarrow$ action given by π for S
  - Take action A, observe R, S'
  - $V(S) \leftarrow V(S) + \alpha[R + \gamma V(S') - V(S)]$
  - $S \leftarrow S'$
- Until S is terminal

8.2 Q-Learning

Another example of reinforcement learning is Q-Learning. Here, we learn an action-value function:

$$Q(S_t, A_t) = Q(S_t, A_t) + R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$$

Q-Learning for estimating $\pi \simeq \pi$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\epsilon > 0$
Initialise $Q(s, a)$ for all $s \in S^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$.
Loop for each episode:

- Initialise S
- Loop for each step of episode:
  - Choose A from S using policy derived from Q (such as ϵ-greedy)
  - Take action A, observe R, S'
  - $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$
  - $S \leftarrow S'$
- Until S is terminal

The main difference is that we use the **best action** in the next state ($\max_a Q(S', a)$). Other than this, the two algorithms are very similar. TD-Learning uses the value of the **state given** by the policy, while Q-learning uses the **best possible action**, not necessarily the one given by the policy.

8.3 Bigger spaces

What if we have something with hundreds or millions of states? Possible solutions include:

- Using exploration-exploitation strategies
- Use function approximates (e.g. neural rewards)

There is an **exploration-exploitation** tradeoff where **exploitation** makes the best decision given current information and **exploration** gathers more information. The classic policy that relies on this tradeoff is the ϵ-greedy policy, which is a policy for when to choose the best available action $a*$ or a random action:

$$a_t = \begin{cases} a_t^* & \text{with probability } 1-\epsilon \\ \text{random action} & \text{with probability } \epsilon \end{cases}$$

- $\epsilon = 1$ gives a purely exploratory policy
- $\epsilon = 0$ gives a purely exploitative policy
- $\epsilon =]0, 1[$ gives somewhere in-between these two ideas. It is generally a good idea to explore more in the beginning and then exploit more towards the end of the optimisation.

8.4 Deep Q-learning

Recent success in reinforcement learning relies on two tricks:

1. Use neural network as $Q(S, A; \theta)$ function approximator with parameters θ
2. Use separate memory system to avoid getting stuck at local optima

9 Unsupervised Learning

Most sensory data we receive is of the unsupervised variety. There is no teaching signal and no feasible reward for every bit of data.

9.1 Sparse Coding

Sparse coding is a form of encoding in networks. It's where you only have a small set of units responding for a given input. It provides a framework for learning sparse representations of the input.

The actual *definition* is that sparse coding is the representation of items by the strong activation of a relatively small set of neurons. For each stimulus, this is a subset of all available neurons

Sparse coding has these key properties:

- **Overcomplete dictionary W**: $d > m$ (i.e. more dictionary elements than input elements)
- **Sparseness**: Number of zero elements in v is much larger than then number of non-zero elements
- Overcomplete and sparse representation leads to **richer features**.

These properties are represented in the cost function, where we have the convention of two important terms: *preserve information* and *sparseness*. The cost function looks like this:

$$\text{cost } = [\text{preserve information}] + \lambda[\text{sparseness}]$$

Sparse coding with L1 norm is:

$$\arg\min_{W,V} = ||U - WV||_2^2 + \lambda||V|| + 1$$

- To find a good sparse code, we need to optimise both the dictionary W and the representation V.
- This optimisation is convex in V and W separately, but not both.
- Several methods exist to optimise V and W, but we will only discuss one
- λ controls the tradeoff between sparseness and reconstruction.

Sparse coding is really good for noise reduction of an image or image restoration.

In the visual system, this is an easy mapping from conceptual to real. The inputs are mapped to the **LGN**, that is a region between the retina and the back of the brain. This input goes through some set of weights (maps to our W). Finally, the visual cortex itself is the output.

9.2 Sparse Coding in the Brain

We are now going to look at one type of cost function:

$$\text{argmin}_{W,V} = ||U - WV||_2^2 + \lambda \sum_i S(v_i)$$

Where $S(v_i) = \log(1 + v_i^2)$.

We need to derive a way to optimise these two sets of components, as discussed previously. Let's start with V.

$$w_i U - \sum_{j \neq i} G_{ij} v_j - f_\lambda(v_i)$$

where $G_{ij} = w_i w_j$ and $f_\lambda(v_i) = v_i + \lambda S'(v_i)$.

Secondly, we need to optimise the weights as

$$\begin{aligned}\Delta w_i &= \eta \frac{\delta \text{Cost}}{\delta w_i} \\ &= \eta < [U - \hat{U}]v_i > \\ &= \eta < [U - \sum_i^m w_i v_i]v_i >\end{aligned}$$

The learning rule is local and there is a Hebbian term (UV meaning a function of pre and post synaptic connectivity) and a postsynaptic only term (WV^2 that causes decay).

Sparse coding is really useful for the brain because it has good representations of information that can be used for decision making and discrimination. It's also really energy efficient, with only a few neurons active at a given time. Finally, it's also more robust to peturbations than purely localist codes.

9.3 Autoencoders

Autoencoders are very popular and use a similar cost function to the one discussed above. We give a network an input and then the network learns a sparse or compressed representation of the input. There is a network *bottleneck*, after which the network starts to decode the network. It essentially goes from lots of nodes, to few nodes (which is the bottleneck), and finally up to many (the re-representation).

The aim is to reconstruct the input, but without an explicit constraint on sparsity, and instead on other forms of regularisation.

The cost function for this, therefore is:

$$\text{argmin}_W = ||V_U||_2^2$$

10 Temporal processing and Recurrent Neural Networks (RNNs)

Temporal processing is crucial for our perception of the world. For example, with speech, you obviously are unable to understand things that are backwards (like reversed). The content of two clips where one is reversed is technically the same.

10.1 Temporal Processing

We have a few brain areas that are key for language processing in the brain. **Broca's area** is particularly important for speech production, while the many other cortices are used for speech understanding. I don't think this will be necessary content for the exam, but if it is, check the first part of lecture 6.

Receptive fields are also present for natural sounds in the auditory cortex, similar to the way that natural shapes are seen in the visual cortex. Receptive fields for audio seem to be localised in time and frequency domains, also similar to those in the visual cortex.

Feedforward networks do not model time *explicitly*. One option is to consider each input as being a different time point, but this is still quite hacky, because it is not how the brain work and does not seem to be a natural solution to the problem. Instead, in a **recurrent network**, each node is *laterally connected* and is also able to be connected to itself (have some memory about previous input).

We would model discrete time (the next time point v_{t+1}) as:

$$v_{t+1} = rv_t + f(Wu_t + Mv_t)$$

Here:

- r is the activity delay
- u is the feed forward input

- M is the recurrence input from other units

Furthermore, we can model continuous time as

$$\tau_r \frac{dv}{dt} = -v + f(Wu + Mv)$$

Here:

- dv/dt is the modelling of change in v over time
- τ_r controls this and is the timeconstant of activity decay
- $-v$ is the passive decay
- The rest is the same as the discrete case

10.2 Recurrent Neural Networks (RNNs)

An application of RNNs in neuroscience is that the brain remembering where it is looking at despite constant blinking. Is there some sort of memory for this? Remembering eye position seems to be encoded by three neurons. One encodes the direction clockwise, and the other counter clockwise. When the eye changes direction, one of these two neurons fire. There is a third neuron called an *integrator* neuron, that fires the whole time, and the rate determines where the eye is. The integrator gets input from the **on** and **off-direction** neurons resulting in an eye position. This integrator is related to the discrete time model equation discussed above.

- For $r = 1$ we have a perfect integrator, where the neuron remembers previous inputs forever
- For $r < 1$ the activity decays back to baseline
- For $r > 1$ the activity grows exponentially over time

So you choose or optimise r depending on the timescale of memory you need for a given task.

Some networks use a **winner-takes-all** network, which is similar to applying a *max* operator on the response of k neurons. Using lateral inhibitory neurons is a way of implementing such a max operation in a network. If we don't have this winner takes all, we might have multiple neurons firing for a certain eye position, but applying this means we only have a single, strongest neuron firing.

10.2.1 Attractor Neural Networks

Attractor neural nets are another form of RNN modelled as dynamic systems that contain a number of attractor states with low energy to which the network dynamics converge. An example of this is the **Hopfield** network.

These attractor neural nets have two key properties:

- $m_{ii} = 0$: no unit has a connection with itself
- $m_{ij} = m_{ij}$ connections are symmetric

There is also a stochastic version of the Hopfield network, called a **Boltzmann machine**. The total input is

$$z_i = b_i + \overbrace{\sum_j u_j w_{ij}}^{\text{external input}} + \overbrace{\sum_x v_x m_{ix}}^{\text{recurrent input}}$$

The probability of turning on a given neuron i is

$$P_{i=on} = \frac{1}{1 + \exp(-\frac{z_i}{T})}$$

If temperature $T = 0$, we just get a Hopfield network.

Boltzmann machines are good generative models, meaning they are able to recreate shapes pretty well when they have learned it.

10.3 Reservoir computing

Reservoir computing uses a large pool of sparsely connected neurons that generate a large repertoire of dynamics (they have *fading memory*) and learn a simple read-out to combine the interesting dynamics into a useful output.

There are two specific variants of reservoir computing: **echo state networks** and **Liquid state machines**

Echo state networks is the most common *rate-based* (meaning neurons are modelled at the level of firing rates) reservoir networks.

Liquid state machines are similar to echo state machines, but neurons are modelled at the level of individual spikes, which is a more realistic estimation of neurons.

10.4 Recurrent CNNs

Most brain areas exhibit a high degree of recurrence. Recent developments show that if CNNs are extended with specific forms of recurrence, they are able to outperform purely feedforward CNNs.

11 Microcircuits and RNNs

We have two umbrella types of neurons in the brain: *inhibitory* and *excitatory* neurons. The activity between these two is very balanced, which is good because it means that there is no runaway activity.

The inhibitory learning rule is:

$$\begin{aligned}\Delta w_{\text{inh}} &= \eta x(y - r_0)\\ 0 &= \eta x(y - r_0)\\ y &= r_0\end{aligned}$$

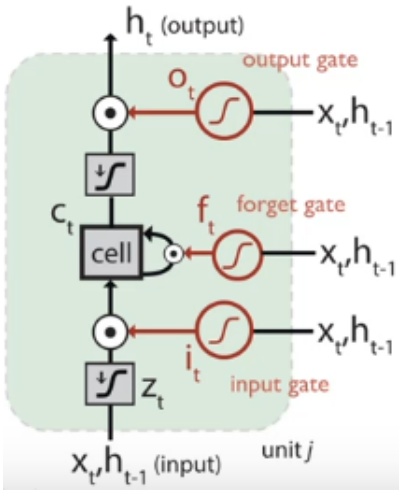
Where r_0 is the target rate of the post synaptic neuron y. r_0 is typically quite a low number.

What we have seen from inhibitory plasticity is that inhibition learns to follow excitation. It means that the receptive fields are balanced.

Inhibitory cells come in different flavours. Pyramidal cells are one type of inhibitory cell, but together with these, there are multiple more types. We won't talk about all these, but it's important to know. We will be looking at **basket cells**, which make up around 50% of the inhibitory neurons in the brain.

11.1 Long Short-Term Memory (LSTM)

LSTMs are the state of the art recurrent neural network model. They are useful because they are gated neural network that is able to capture both short and long term data. At the heart of this network is the memory cell.



This structure very closely resembles the circuits we see in the *cortical circuits*. There is an important difference, however. We do the gating with *multiplicative difference* in LSTM, but in neuroscience, we often interpret this gating as being inhibitory, meaning it should be subtractive. This moves us to a new model, **sub-LSTM**, where the gating is subtractive.

11.2 Summary

- Multiple excitatory and inhibitory cell types in the brain

- Intricate microcircuits across multiple layers
- Machine learning LSTMs are a form of gated-RNN good for capturing long-term dependencies such as language modelling
- Cortical microcircuits have similar features to gated-RNNs but may operate with subtractive gating (subLSTMs)