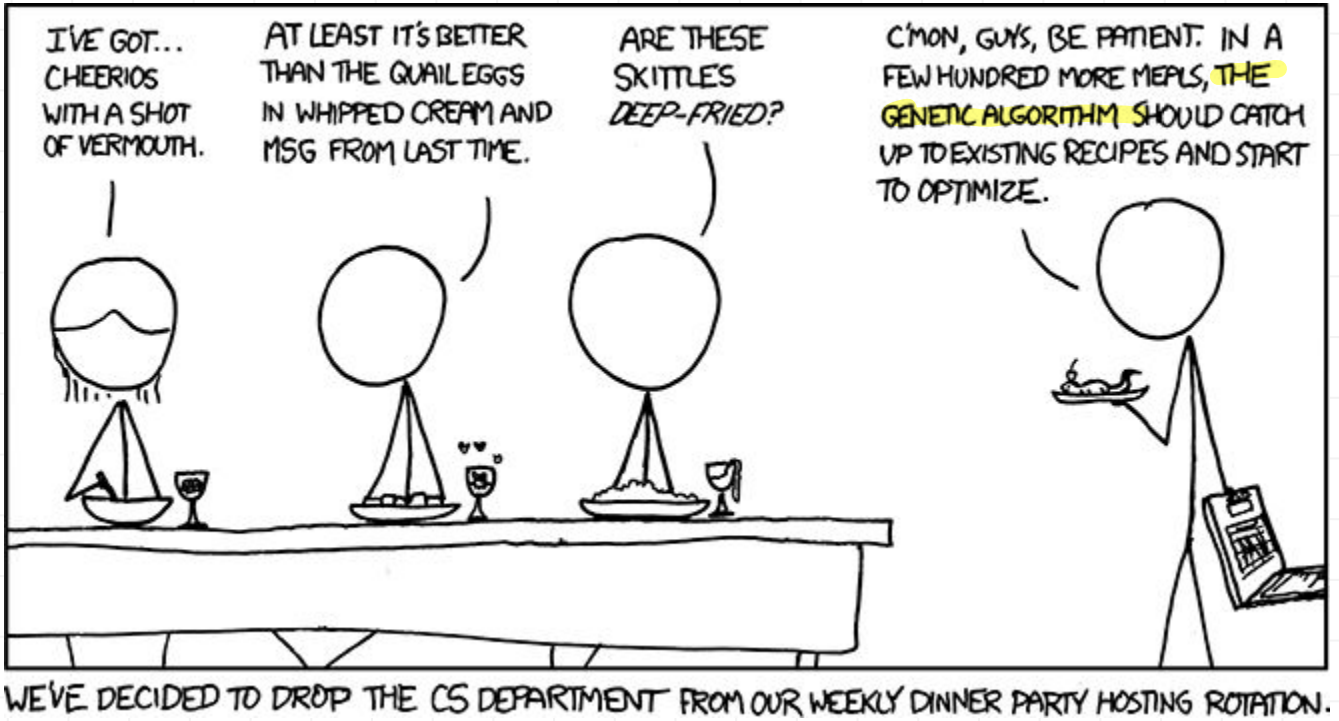
https://xkcd.com/720/



Announcements

Quiz 1 recap – today after the main lecture

287 submit mean ~ 73

Assignment 1 due next Mon

(Extra support sessions this week will be announced soon)

Neural networks

Chap 5 PRML

Neural network as adaptive basis functions
- Weight-space symmetries

Network training
- parameter optimisation
- gradient descent

Error backpropagation and automatic differentiation

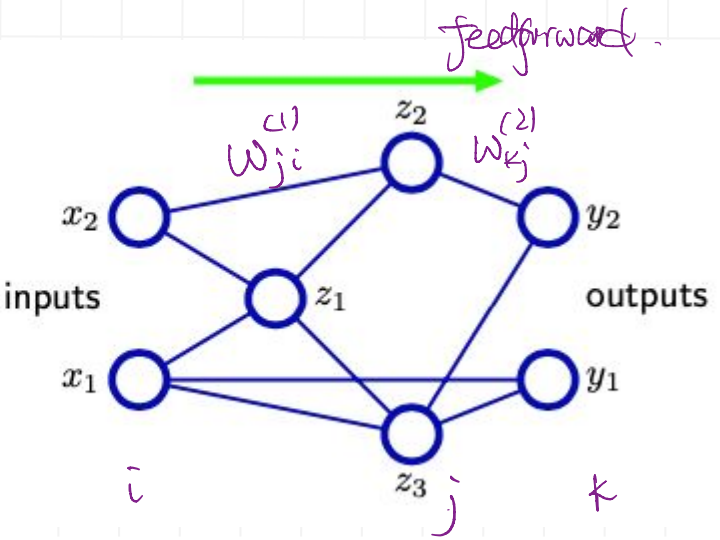
Regularisation

Feed-forward network functions

$$y(\mathbf{x}, \mathbf{w}) = f\left(\sum_{j=1}^{M} w_j \phi_j(\mathbf{x})\right) \tag{5.1}$$

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right). \tag{5.9}$$

Figure 5.2 Example of a neural network having a general feed-forward topology. Note that each hidden and output unit has an associated bias parameter (omitted for clarity).



- Can have skip connections;
- Connections can be sparse;
- Require feed-forward structure, no directed cycles
- Convention: number of layers refers to the number of weights (rather than nodes)

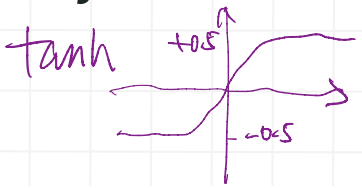
Two key ideas:

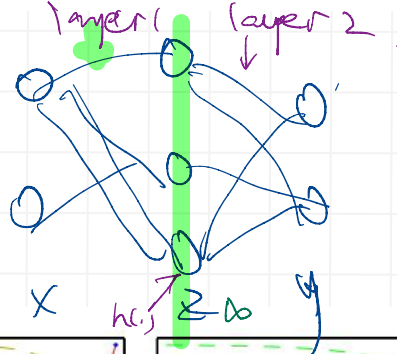
1) Generalised linear model with activation function
2) Recursive composition of these

Universal approximation property

(regression) a two-layer network with linear outputs can uniformly approximate *any continuous function* on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units.

(classification) a two-layer network can uniformly approximate *any discriminative function* on a compact input domain to arbitrary accuracy provided the network has a sufficiently large number of hidden units.





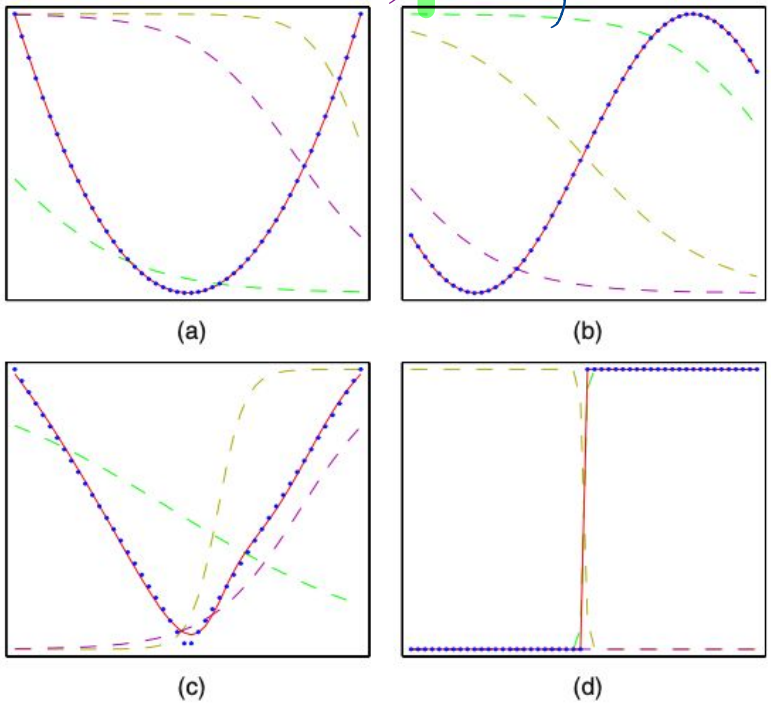


Figure 5.3 Illustration of the capability of a multilayer perceptron to approximate four different functions comprising (a) $f(x) = x^2$, (b) $f(x) = \sin(x)$, (c), $f(x) = |x|$, and (d) $f(x) = H(x)$ where $H(x)$ is the Heaviside step function. In each case, $N = 50$ data points, shown as blue dots, have been sampled uniformly in x over the interval $(-1, 1)$ and the corresponding values of $f(x)$ evaluated. These data points are then used to train a two-layer network having 3 hidden units with 'tanh' activation functions and linear output units. The resulting network functions are shown by the red curves, and the outputs of the three hidden units are shown by the three dashed curves.

select # of layers, neurons, etc.
Bayesian Opt. / AutoML / NN architecture search

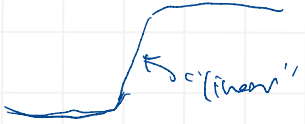
Comparison to Perceptron

- A neural network looks like a multilayer perceptron.
- But perceptron's nonlinear activation function was a step function — neither smooth nor differentiable.

$$f(a) = \begin{cases} +1, & a \geq 0 \\ -1, & a < 0 \end{cases}$$

- The activation functions $h(\cdot)$ and $g(\cdot)$ of a neural network are smooth and differentiable.

having gradients are nice ; everywhere.



What if:

All activation functions are linear? → (linear layer

Number of hidden units smaller than the number of input dimensions?

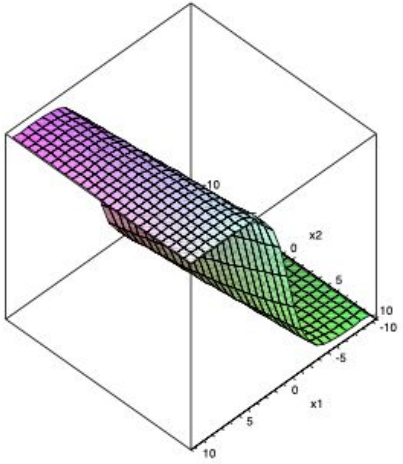
→ No UA

don't need $\phi(x)$

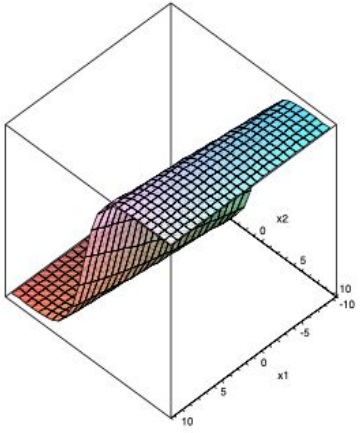
Neural network as variable basis functions

$$z = \sigma(w_0 + w_1 x_1 + w_2 x_2)$$

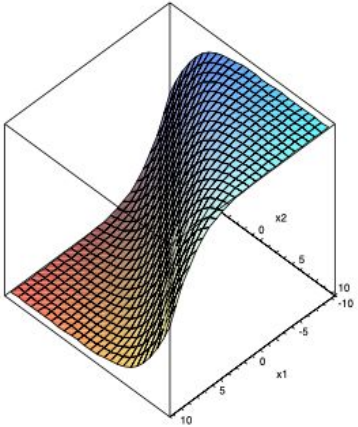
"tuned to training data"



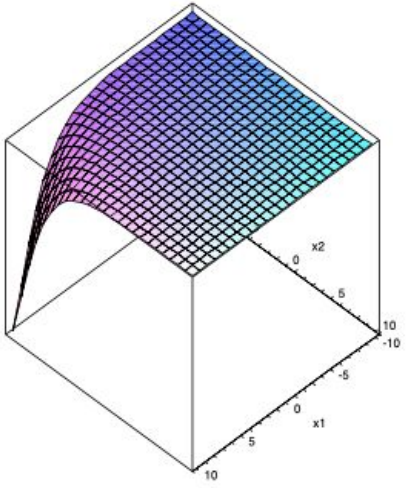
$(w_0, w_1, w_2) = (0.0, 1.0, 0.1)$



$(w_0, w_1, w_2) = (0.0, 0.1, 1.0)$



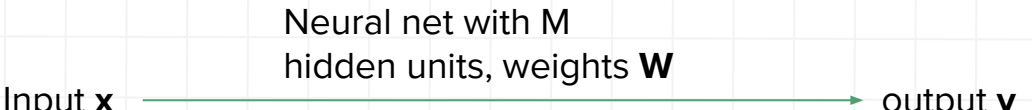
$(w_0, w_1, w_2) = (0.0, -0.5, 0.5)$

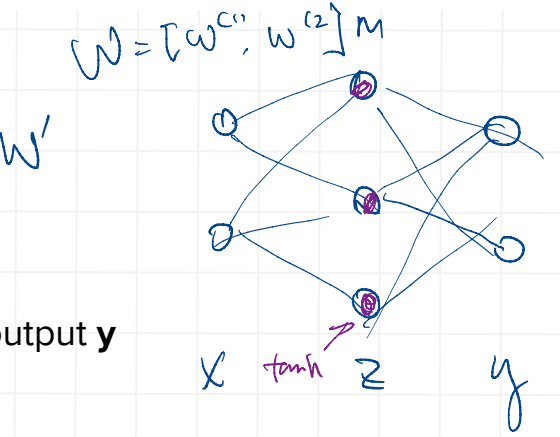


$(w_0, w_1, w_2) = (10.0, -0.5, 0.5)$

(the source of these figures is lost -- let me know if you encounter them, with apologies to the author)

Weight-space symmetries





Q: is there another set of weights **W'** that always map the same input to the same output as **W** .

$y = f_W(x)$

$[w^{(1)}, w^{(2)}] \to W$

$y = f_{W'}(x)$

① sign flips

e.g $\tanh(\cdot)$

$-w^{(1)} \quad -w^{(2)}$

$\to 2^M$

② $z = [z_1, z_2, z_3]$ — "permutation invariant"

$M!$

combined $2^M \cdot M!$

What we covered so far

Neural network as adaptive basis functions
- Weight-space symmetries

Network training
- parameter optimisation
- gradient descent

Error backpropagation and automatic differentiation

Regularisation

Neural network training -- objectives

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right). \qquad (5.9)$$

Regression (linear activation at output)

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(\mathbf{x}_n, \mathbf{w}) - t_n\}^2 \qquad (5.14)$$

↑ over fitting.

Binary classification (logistic output activation)

$t_n \in \{0, 1\}$

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\{t_n \ln y_n + (1 - t_n)\ln(1 - y_n)\} \qquad (5.21)$$

also (4.90)

either term is active
not both.

Different versions of classification

Binary classification (logistic output activation)

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \{t_n \ln y_n + (1 - t_n) \ln(1 - y_n)\} \tag{5.21}$$

Multiple/independent binary classification ← cat? Y/N car? Y/N

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} \{t_{nk} \ln y_{nk} + (1 - t_{nk}) \ln(1 - y_{nk})\} \tag{5.23}$$

Multi-class classification {truck, crane, digger, car, ...}

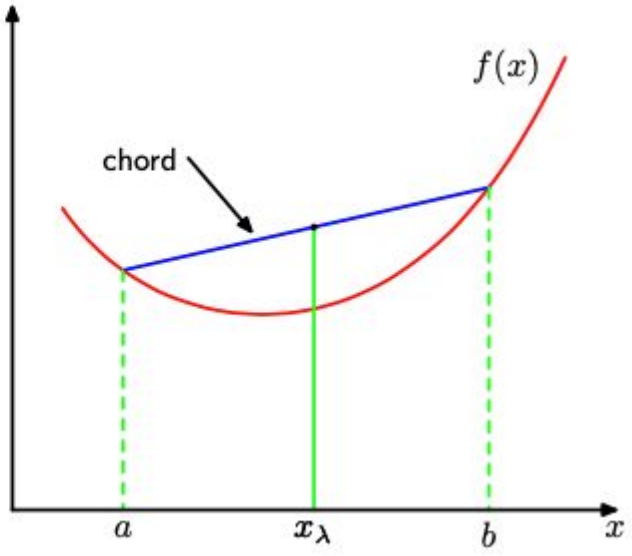
1-hot encoding [0, 0, 1, ..., 0]

$$E(\mathbf{w}) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{kn} \ln y_k(\mathbf{x}_n, \mathbf{w}). \tag{5.24}$$

only 1 out of K terms active,

Recap: Convex functions

Figure 1.31 A convex function $f(x)$ is one for which every chord (shown in blue) lies on or above the function (shown in red).



$$f(\lambda a + (1-\lambda)b) \leqslant \lambda f(a) + (1-\lambda)f(b). \qquad (1.114)$$

f" >= 0

Examples: x^2, xlnx (x>0), ... $\mathbf{x}^T$ S $\mathbf{x}$ (S positive semi definite)

$\frac{\partial^2 f}{\partial x^2} \geq 0$ positive semi-definite.

Jesen's inequality $f(\mathbb{E}[x]) \leqslant \mathbb{E}[f(x)]$ (1.116)

Non-linear optimisation

σ(·) / h(·) non-linear
many options.

Task: find weight vector **w**, that minimizes the function E(**w**)

$$\nabla E(\mathbf{w}) = 0 \qquad (5.26)$$

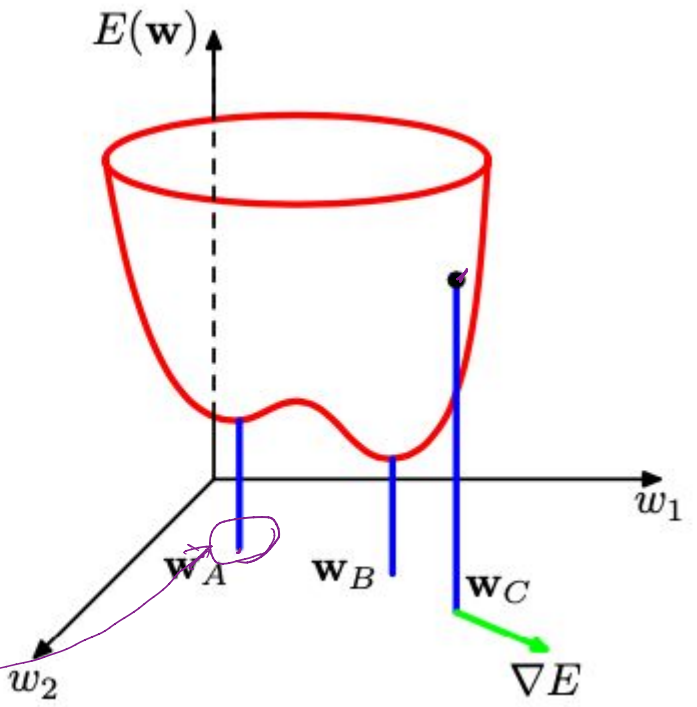
stationary point

Definition (Global Minimum)

A point $\mathbf{w}^*$ for which the error $E(\mathbf{w}^*)$ is smaller than any other error $E(\mathbf{w})$.

Definition (Local Minimum)

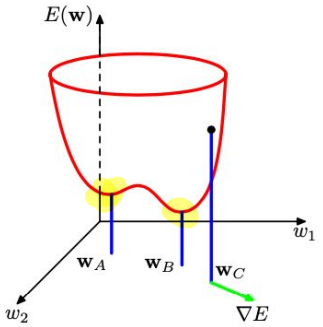
A point $\mathbf{w}^*$ for which the error $E(\mathbf{w}^*)$ is smaller than any other error $E(\mathbf{w})$ in some neighbourhood of $\mathbf{w}^*$.



Non-linear optimisation (a local view)

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} + \Delta\mathbf{w}^{(\tau)} \tag{5.27}$$

Goal: figure out a good direction **Δw** (and how far)



Taylor expansion

$$E(\mathbf{w}) \simeq E(\widehat{\mathbf{w}}) + (\mathbf{w}-\widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{b} + \frac{1}{2}(\mathbf{w}-\widehat{\mathbf{w}})^{\mathrm{T}}\mathbf{H}(\mathbf{w}-\widehat{\mathbf{w}}) \tag{5.28}$$

gradient/Jacobian

$$\mathbf{b} \equiv \nabla E|_{\mathbf{w}=\widehat{\mathbf{w}}} \tag{5.29}$$

Hessian

$$(\mathbf{H})_{ij} \equiv \left.\frac{\partial E}{\partial w_i \partial w_j}\right|_{\mathbf{w}=\widehat{\mathbf{w}}}. \tag{5.30}$$

H is p.s.d → local minima. Many other geometric structures possible, maxima, saddle points, ring structure …

Gradient descent: the computational argument

$$\frac{\partial^2 E}{\partial w_i \partial w_j} = \frac{\partial^2 E}{\partial w_j \partial w_i}$$

- Hessian is symmetric and contains $W(W+1)/2$ independent entries where W is the total number of weights in the network.
- If we use function evaluations only:
  - Need to gather this $O(W^2)$ pieces of information by doing $O(W^2)$ number of function evaluations each of which cost $O(W)$ time, for an overall cost of order $O(W^3)$.
- If we use gradients of the function:
  - Surprisingly the gradient ∇E also costs only $O(W)$ time, although it provides W pieces of information.
  - We now need only $O(W)$ steps, so the order of time complexity is reduced to $O(W^2)$.

$O(W)$

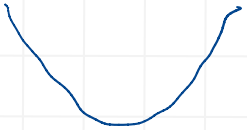
each w_i $i = 1, \ldots W$

↓ goes through one multiplying.

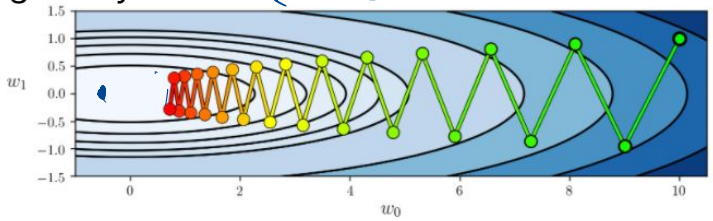
$$y_k \sim \sigma\left(\sum_j w_{ji}\, h\left(\sum_i w_i x_i\right)\right)$$

The **Cheap Gradient Principle** (Griewank 2008) --- the computational cost of computing the gradient of a scalar-valued function is nearly the same (often within a factor of 5) as that of simply computing the function itself --- is of central importance in optimization; it allows us to quickly obtain (high dimensional) gradients of scalar loss functions which are subsequently used in black box gradient-based optimization procedures.

Limitations of gradient-based methods



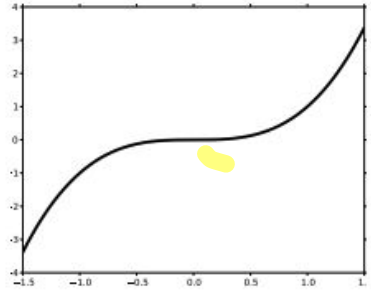
Long valleys

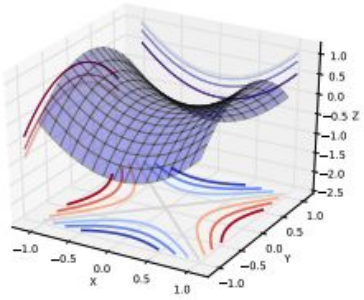


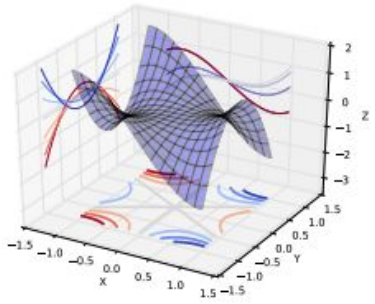
Follow-up topics:

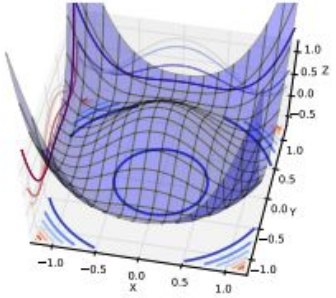
* Newton methods
* Quasi-Newton methods, e.g. L-BFGS
* conjugate gradient descent
* momentum
* ...

Saddle points









https://jermwatt.github.io/machine_learning_refined/notes/3_First_order_methods/3_7_Problems.html

"On the saddle point problem for non-convex optimization", Pascanu et al, https://arxiv.org/abs/1405.4604

What we covered so far

Neural network as adaptive basis functions
- Weight-space symmetries

Network training
- parameter optimisation
- gradient descent

Error backpropagation and automatic differentiation

Regularisation

Gradient descent optimisation

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E(\mathbf{w}^{(\tau)}) \tag{5.41}$$

Online version, stochastic gradient

$$E(\mathbf{w}) = \sum_{n=1}^{N} E_n(\mathbf{w}). \tag{5.42}$$

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta \nabla E_n(\mathbf{w}^{(\tau)}). \tag{5.43}$$

Error backpropagation: linear version

$$y_k = \sum_i w_{ki} x_i \tag{5.45}$$

$$y_{nk} = y_k(\mathbf{x}_n, \mathbf{w})$$

$$E_n = \frac{1}{2} \sum_k (y_{nk} - t_{nk})^2 \tag{5.46}$$

$$E_{n_k} = \frac{1}{2} \quad (y_{nk} - t_{nk})^2$$

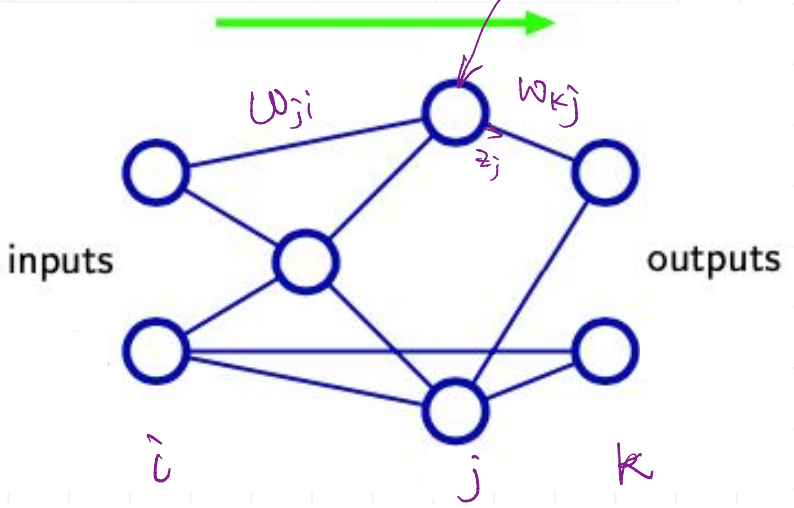
$$\frac{\partial E_n}{\partial w_{ji}} = (y_{nj} - t_{nj}) x_{ni} \tag{5.47}$$

error for output nth data
jth component

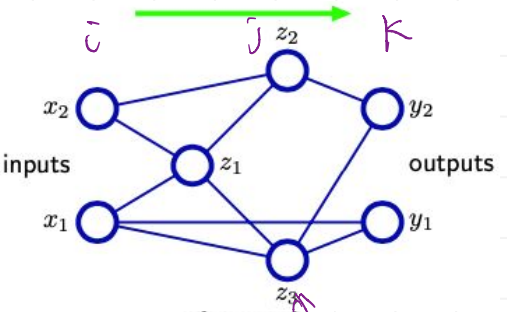
Forward propagation in general

$$a_j = \sum_i w_{ji} z_i \tag{5.48}$$

$$z_j = h(a_j). \tag{5.49}$$



Backpropagation for 1 layer



$$a_j = \sum_i w_{ji} z_i$$

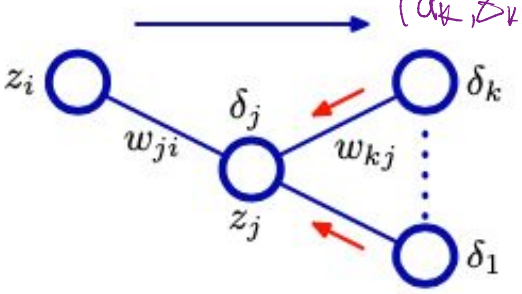
$$z_j = h(a_j). \qquad (5.49)$$

$$\frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}. \qquad (5.50)$$

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} \qquad \frac{\partial a_j}{\partial w_{ji}} = z_i.$$

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i. \qquad (5.53)$$

Figure 5.7 Illustration of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forward propagation, and the red arrows indicate the backward propagation of error information.



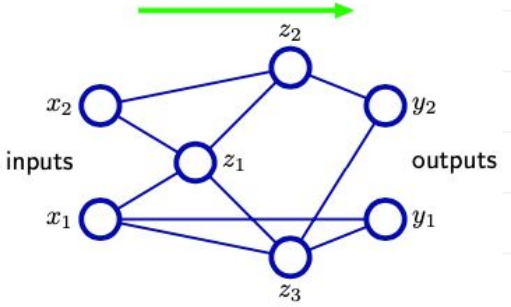
over all output units z_k, $k=1,\dots,K$

$$\delta_j \equiv \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{5.55}$$

$$\delta_j = h'(a_j)\sum_k w_{kj}\delta_k \tag{5.56}$$

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\{t_{nk}\ln y_{nk} + (1-t_{nk})\ln(1-y_{nk})\} \tag{5.23}$$

$$y_k(\mathbf{x},\mathbf{w}) = \sigma\left(\sum_{j=0}^{M} w_{kj}^{(2)} h\left(\sum_{i=0}^{D} w_{ji}^{(1)} x_i\right)\right). \tag{5.9}$$

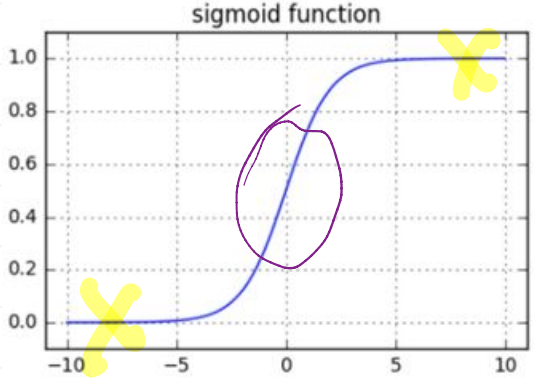


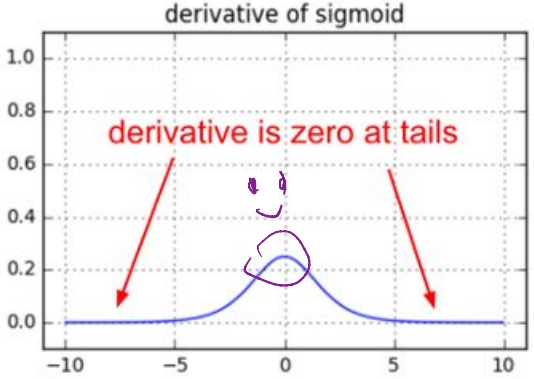
“Yes you should understand backprop”

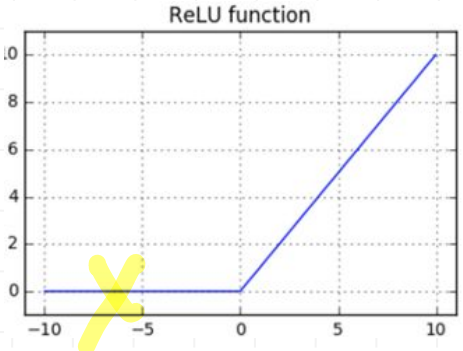
https://medium.com/@karpathy/yes-you-should-understand-backprop-e2f06eab496b

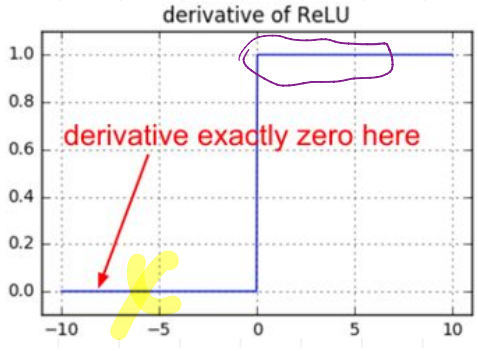
It’s a *leaky abstraction* - it is easy to fall into the trap of abstracting away the learning process — believing that you can simply stack arbitrary layers together and backprop will “magically make them work” on your data

https://xkcd.com/1838/









https://cs231n.github.io/optimization-2/, also see MML Sec 5.6, esp 5.6.2

Implementing Backprop using Automatic Differentiation

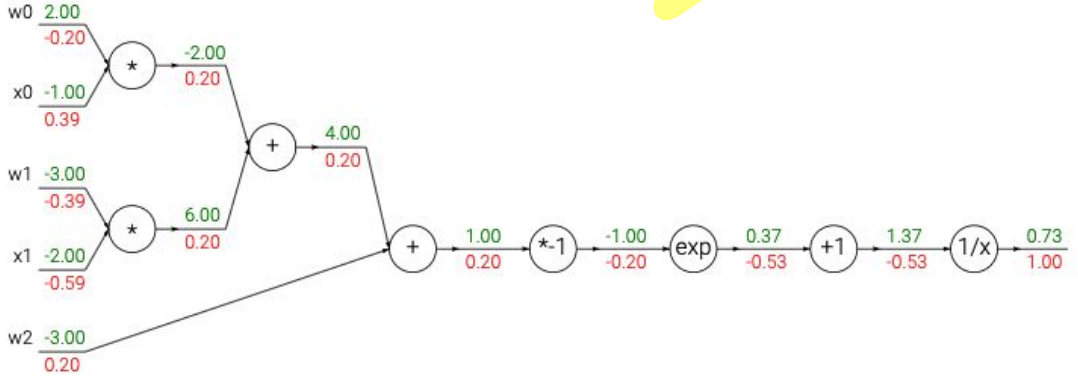
$$f(w,x) = \frac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}}$$

$$f(x) = \frac{1}{x} \rightarrow \frac{df}{dx} = -1/x^2$$

$$f_c(x) = c + x \rightarrow \frac{df}{dx} = 1$$

$$f(x) = e^x \rightarrow \frac{df}{dx} = e^x$$

$$f_a(x) = ax \rightarrow \frac{df}{dx} = a$$

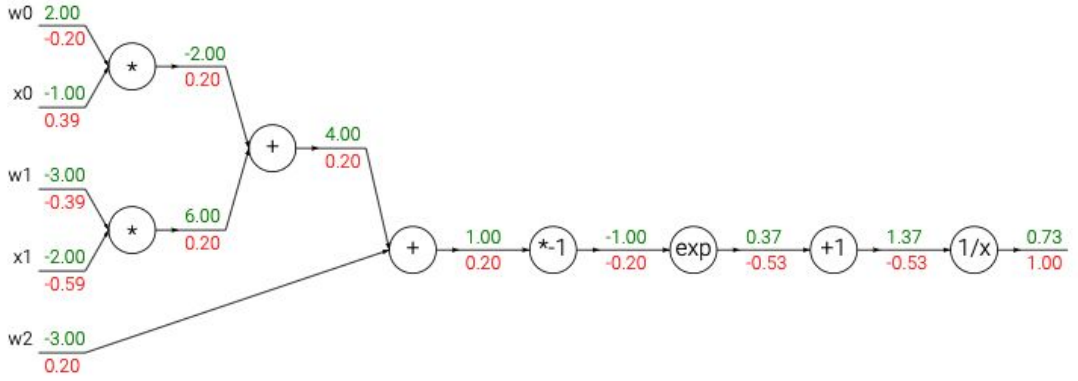


Example circuit for a 2D neuron with a sigmoid activation function. The inputs are [x0,x1] and the (learnable) weights of the neuron are [w0,w1,w2]. As we will see later, the neuron computes a dot product with the input and then its activation is softly squashed by the sigmoid function to be in range from 0 to 1.

https://cs231n.github.io/optimization-2/, also see MML Sec 5.6, esp 5.6.2

Implementing Backprop using Automatic Differentiation

- Have differentials of elementary function
- Use chain rule to compose the stage-wise gradient
- Caching - store the differentials and function values as computer variables



Example circuit for a 2D neuron with a sigmoid activation function. The inputs are [x0,x1] and the (learnable) weights of the neuron are [w0,w1,w2]. As we will see later, the neuron computes a dot product with the input and then its activation is softly squashed by the sigmoid function to be in range from 0 to 1.

What we covered so far

Neural network as adaptive basis functions
- Weight-space symmetries

Network training
- parameter optimisation
- gradient descent

Error backpropagation and automatic differentiation

Regularisation

Regularising neural networks

The number of input and output nodes determined by the application.

The number of hidden nodes is a free parameter.

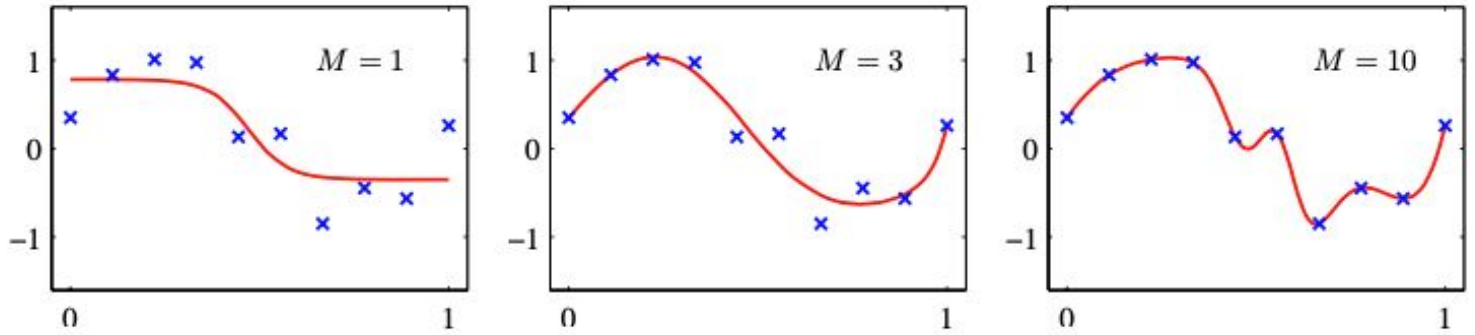


Figure 5.9 Examples of two-layer networks trained on 10 data points drawn from the sinusoidal data set. The graphs show the result of fitting networks having $M = 1$, 3 and 10 hidden units, respectively, by minimizing a sum-of-squares error function using a scaled conjugate-gradient algorithm.

Recipe: use the L2 regularisation that we know

$$\widetilde{E}(\mathbf{w}) = E(\mathbf{w}) + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}. \tag{5.112}$$

Regularisation by early-stopping

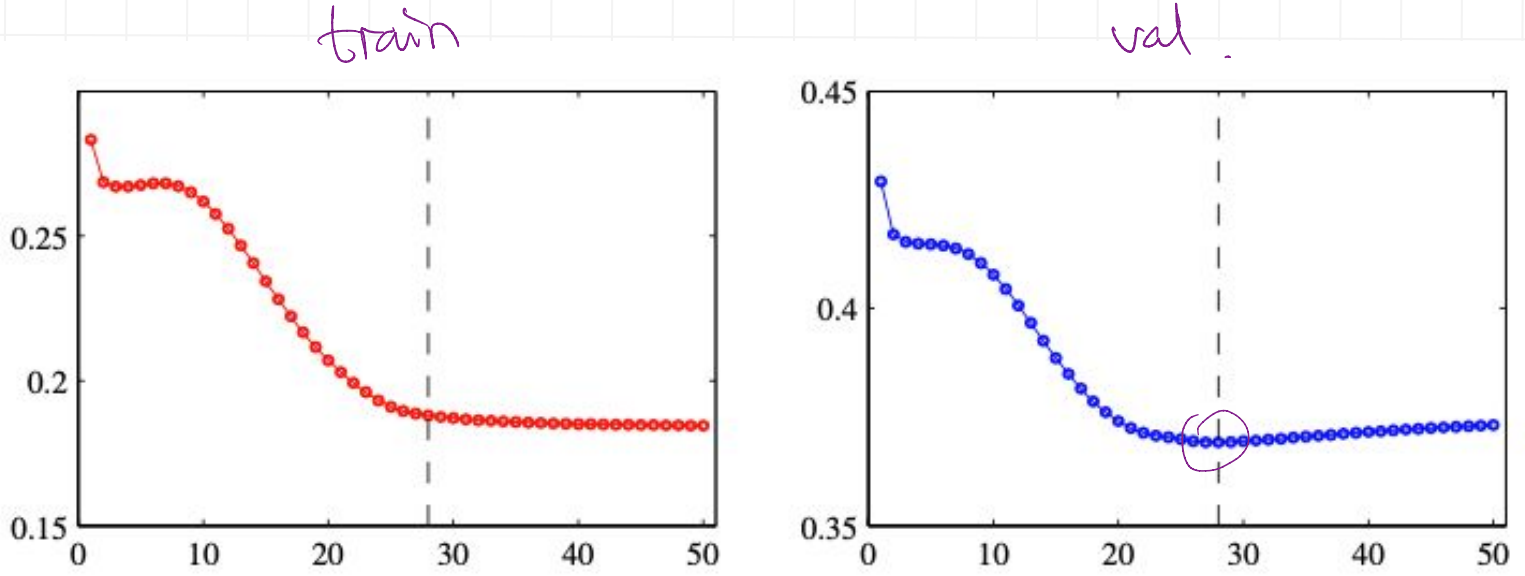


Figure 5.12 An illustration of the behaviour of training set error (left) and validation set error (right) during a typical training session, as a function of the iteration step, for the sinusoidal data set. The goal of achieving the best generalization performance suggests that training should be stopped at the point shown by the vertical dashed lines, corresponding to the minimum of the validation set error.

Invariances

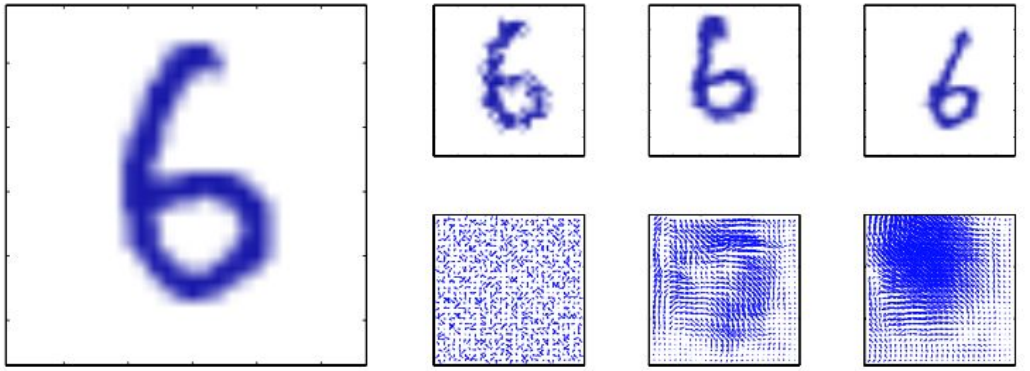


Figure 5.14 Illustration of the synthetic warping of a handwritten digit. The original image is shown on the left. On the right, the top row shows three examples of warped digits, with the corresponding displacement fields shown on the bottom row. These displacement fields are generated by sampling random displacements $\Delta x, \Delta y \in (0, 1)$ at each pixel and then smoothing by convolution with Gaussians of width 0.01, 30 and 60 respectively.

+ Augment training set with perturbed/transformed training patterns (Fig 5.14)
+ Preprocess input by normalising against transformations (e.g. rectifying faces)
+ Build important invariance into network structure -- e.g. Convolutional Nets
- Explicitly allow invariances using improper priors (as regulariser) - 5.5.1 , tangent propagation (5.5.4)

Bayesian Neural Networks

- Predict a single target t from a vector of inputs $\mathbf{x}$
- Assume conditional distribution to be Gaussian with precision β

$$p(t \mid \mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t \mid y(\mathbf{x}, \mathbf{w}), \beta^{-1})$$

- Prior distribution over weights $\mathbf{w}$ is also assumed to be Gaussian

$$p(\mathbf{w} \mid \alpha) = \mathcal{N}(\mathbf{w} \mid \mathbf{0}, \alpha^{-1}\mathbf{I})$$

- For an i.i.d training data set $\{\mathbf{x}_n, t_n\}_{n=1}^{N}$, the likelihood of the targets $\mathcal{D} = \{t_1, \ldots, t_N\}$ is

$$p(\mathcal{D} \mid \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n \mid y(\mathbf{x_n}, \mathbf{w}), \beta^{-1})$$

- Posterior distribution

$$p(\mathbf{w} \mid \mathcal{D}, \alpha, \beta) \propto p(\mathbf{w} \mid \alpha) p(\mathcal{D} \mid \mathbf{w}, \beta)$$

Troubling Trends in Machine Learning Scholarship

Zachary C. Lipton* & Jacob Steinhardt*
Carnegie Mellon University, Stanford University
zlipton@cmu.edu, jsteinhardt@cs.stanford.edu

July 27, 2018

3.4 Misuse of Language

We identify three common avenues of language misuse in machine learning: *suggestive definitions*, *overloaded terminology*, and *suitcase words*.

3.4.1 Suggestive Definitions

In the first avenue, a new technical term is coined that has a suggestive colloquial meaning, thus sneaking in connotations without the need to argue for them. This often manifests in anthropomorphic characterizations of tasks (*reading comprehension* [31] and *music composition* [59]) and techniques (*curiosity* [66] and *fear* [48]). A number of papers name components of proposed models in a manner suggestive of human cognition, e.g. "thought vectors" [36] and the "consciousness prior" [4]. Our goal is not to rid the academic literature of all such language; when properly qualified, these connections might communicate a fruitful source of inspiration. However, when a suggestive term is assigned technical meaning, each subsequent paper has no choice but to confuse its readers, either by embracing the term or by replacing it.

Describing empirical results with loose claims of "human-level" performance can also portray a false sense of current capabilities. Take, for example, the "dermatologist-level classification of skin cancer" reported in [21]. The comparison to dermatologists conceals the fact that classifiers and dermatologists perform fundamentally different tasks. Real dermatologists encounter a wide variety of circumstances and must perform their jobs despite unpredictable changes. The machine

3.4.2 Overloading Technical Terminology

A second avenue of misuse consists of taking a term that holds precise technical meaning and using it in an imprecise or contradictory way. Consider the case of *deconvolution*, which formally describes the process of reversing a convolution, but is now used in the deep learning literature to refer to transpose convolutions (also called up-convolutions) as commonly found in auto-encoders and generative adversarial networks. This term first took root in deep learning in [79], which does address deconvolution, but was later over-generalized to refer to any neural architectures using upconvolutions [78, 50]. Such overloading of terminology can create lasting confusion. New machine learning papers referring to deconvolution might be (i) invoking its original meaning, (ii) describing

As another example, *generative models* are traditionally models of either the input distribution $p(x)$ or the joint distribution $p(x, y)$. In contrast, discriminative models address the conditional distribution $p(y \mid x)$ of the label given the inputs. However, in recent works, "generative model" imprecisely refers to any model that produces realistic-looking structured data. On the surface, this may seem consistent with the $p(x)$ definition, but it obscures several shortcomings—for instance, the inability of GANs or VAEs to perform conditional inference (e.g. sampling from $p(x_2 \mid x_1)$ where x_1 and x_2 are two distinct input features). Bending the term further, some discriminative models are now referred to as generative models on account of producing structured outputs [76], a mistake that we (ZL) make in [47]. Seeking to resolve the confusion and provide historical context, [58] distinguishes between *prescribed* and *implicit* generative models.

Among the consequences of mis-using language is that (as with generative models) we might conceal lack of progress by redefining an unsolved task to refer to something easier. This often combines with suggestive definitions via anthropomorphic naming. *Language understanding* and *reading comprehension*, once grand challenges of AI, now refer to making accurate predictions on specific datasets [31].

3.4.3 Suitcase Words

Finally, we discuss the overuse of *suitcase words* in ML papers. Coined by Minsky in the 2007 book *The Emotion Machine* [56], suitcase words pack together a variety of meanings. Minsky describes mental processes such as *consciousness*, *thinking*, *attention*, *emotion*, and *feeling* that may not share "a single cause or origin". Many terms in ML fall into this category. For example, [46] notes that *interpretability* holds no universally agreed-upon meaning, and often references disjoint methods and desiderata. As a consequence, even papers that appear to be in dialogue with each other may have different concepts in mind.

As another example, *generalization* has both a specific technical meaning (generalizing from train to test) and a more colloquial meaning that is closer to the notion of transfer (generalizing from one population to another) or of external validity (generalizing from an experimental setting to the real world) [67]. Conflating these notions leads to overestimating the capabilities of current systems.

Suggestive definitions and overloaded terminology can contribute to the creation of new suitcase words. In the fairness literature, where legal, philosophical, and statistical language are often overloaded, terms like *bias* become suitcase words that must be subsequently unpacked [17].

In common speech and as aspirational terms, suitcase words can serve a useful purpose. Perhaps the suitcase word reflects an overarching concept that unites the various meanings. For example, *artificial intelligence* might be well-suited as an aspirational name to organize an academic department. On the other hand, using suitcase words in technical arguments can lead to confusion. For example, [6] writes an equation (Box 4) involving the terms *intelligence* and *optimization power*, implicitly assuming that these suitcase words can be quantified with a one-dimensional scalar.

Neural networks

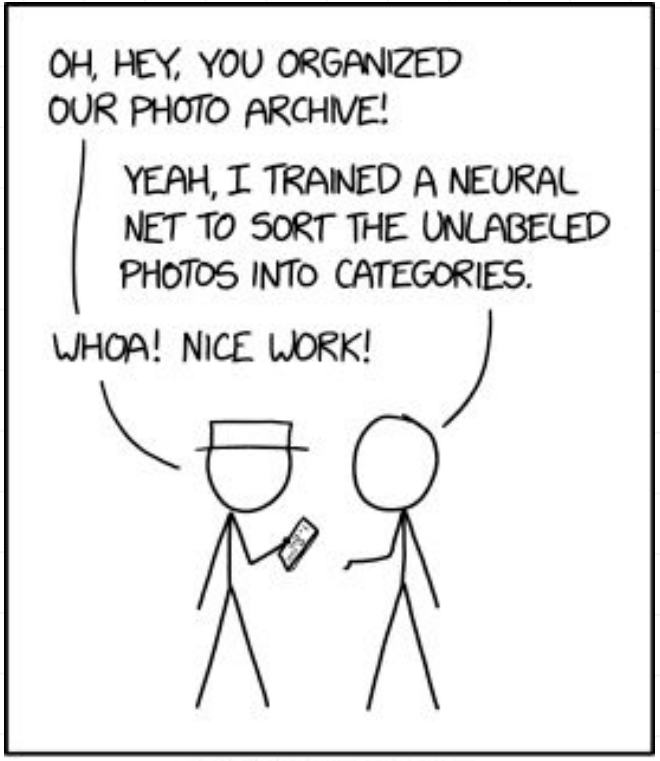
Neural network as adaptive basis functions
- Weight-space symmetries

Network training
- parameter optimisation
- gradient descent

Error backpropagation and automatic differentiation

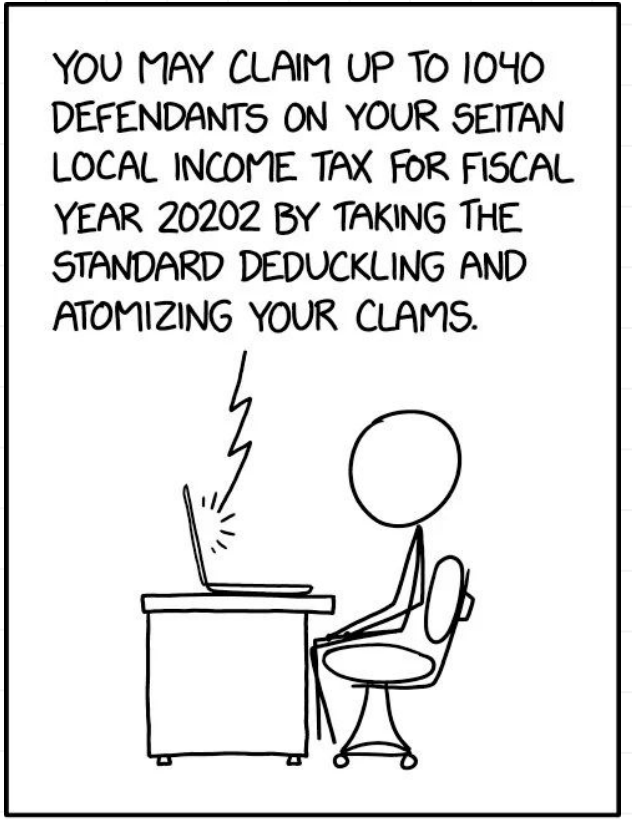
Regularisation

https://xkcd.com/2173/



ENGINEERING TIP:
WHEN YOU DO A TASK BY HAND, YOU CAN TECHNICALLY SAY YOU TRAINED A NEURAL NET TO DO IT.

https://xkcd.com/2265/



I USED A NEURAL NET TO PREPARE MY TAX RETURNS, BUT I THINK I CUT OFF ITS TRAINING TOO EARLY.