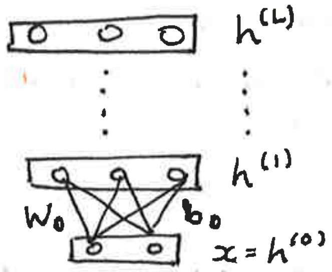
Feed-Forward Neural Networks



matrix notation:

non-linearity

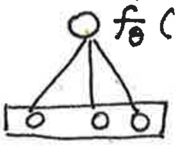
$$h^{(\ell)} = \sigma(W_{\ell-1} h^{(\ell-1)} + b_{\ell-1})$$

"Deep Learning" $\iff L \geq 2$

L+1 layers in total (L hidden layers + 1 output layer)
Don't count input layer.

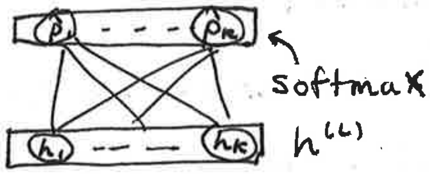
- Regression

$\hat{f}_\theta(x) = W_L h^{(L)} + b_L$



- trained by minimising L2 loss $\sum_i \| y_i - \hat{f}_\theta(x_i) \|^2$
  via backpropagation - gradient descent via chain rule (update weights from top to bottom)
- equivalent to max. lkhd under $p(y_i | x_i) = \mathcal{N}(y_i ; \hat{f}_\theta(x_i), \sigma^2)$.
  - if $h^{(L)} = x$, equiv. to linear regression

- Classification (K-class)



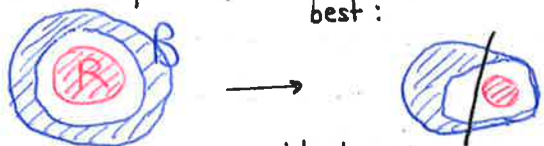
where $\hat{p}_k(x_i) = \dfrac{\exp(h_k(x_i))}{\sum_k \exp(h_k(x_i))}$

- train by min. cross-entropy loss $-\sum_i \log \hat{p}_{c_i}(x_i)$
- equiv. to max. lkhd under $p(c_i | x_i) = \mathrm{Cat}(c_i | \hat{p}_1(x_i), \cdots, \hat{p}_K(x_i))$
- note redundancy in parameterisation; subtracting any constant α from $h_k(\cdot)$ won't change $\hat{p}_k(\cdot)$.
- so for binary classification: $\hat{p}_1(x) = \dfrac{1}{1 + \exp(-h_1(x))} = \sigma(h_1(x))$
  $\hat{p}_0(x) = 1 - \hat{p}_1(x)$.

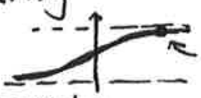
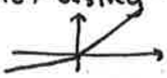
so $p(c_i | x_i) = \hat{p}_1(x_i)^{c_i} (1 - \hat{p}_1(x_i))^{1-c_i}$

- if we have just one hidden layer and no non-linearity, equiv. to logistic regression.
  i.e. finding straight line / hyperplane in x-space that separates two classes best.
- intuition on role of non-linearities & # hidden units:
  - assume ~~non~~ last layer has no non-linearity ($\sigma = \mathrm{id}$) for now.
  - can think of $x \to h^{(L-1)}$ as a non-linear, cts mapping from x-space to space of $h^{(L-1)}$
  - want to learn a mapping st in this new space, the classes are separable by a line / hyperplane.
  - # hidden units controls dimensionality of this new space.
  - e.g. Suppose $x \in \mathbb{R}^2$, with two classes:
    - is it possible to get perfect classification with NN of 2 hidden units? No.
    - but possible with 3 hidden units: $\exists$ map $x \to \mathbb{R}^3$ st R & B separable by plane.

best:



Architecture

- so the more hidden units, the better fit for training data.
- But can overfit with too many → poor generalisation.
- But in practice, better to regularise than use fewer hidden units
  ↳ turns out empirically, with more hidden units ∃ more local min, but these are better than the local min with few hidden units.
- Depth (# hidden layers)
  - deeper ⇒ more complex mappings from $x \longrightarrow h^{(L-1)}$
  - for non-image data, usually 3 layers does better than 2 layers, but going deeper rarely helps more.
  - for image data, depth is crucial - perhaps due to hierarchical structure of images (e.g. faces consist of eyes, eyes made of lines, etc.)
  - Universal approx. Thm for NN with single hidden layer says it can model any cts f^n arbitrarily well. However, usually need v. wide layers to model useful f^ns. Going deeper shown empirically to model complicated f^ns in a more compact way.
- Choice of non-linearity
  ① sigmoid/tanh :  saturating regime. ← no gradient passing thru' in chain rule when units are saturated.
  "saturation/gradient vanishing problem"
  └ why researchers struggled to train NNs in 90s.
  rule of thumb → ② ReLu (rectified linear unit)  - gives NNs that train fast & well.
  1 problem: some hidden units can become 'dead' - value fixed at 0 ∀ inputs, especially if learning rates too big initially.
  so not using full capacity of model.
  ③ Leaky ReLu :  solves above problem, but not consistently better than RELu.
  ④ maxout : generalises (L)ReLu : $\max(W^{(1)}h + b^{(1)}, W^{(2)}h + b^{(2)})$
  └ doubles #params.

Optimisation (to prevent vanishing/exploding gradients)

- Weight initialisation : need symmetry breaking - if all weights are the same, then they will receive same grad. updates, so remain the same.
  - symmetry breaking usually done by random initialisation.
    e.g. Xavier init : $(W_e)_{ij} \overset{iid}{\sim} U[-k,k]$, $k = \sqrt{\frac{6}{|h_e| + |h_{e+1}|}}$ (controls scale of each layer to be similar)
- Bias init: use zero init.
- Normalise data - zero mean, unit variance.
- Adaptive learning rates: Adagrad, RMSprop, ADAM.
- mini-batches (SGD) - also helps with regularisation by adding stochasticity to grads
- batch normalisation - significantly more robust to bad init.

Regularisation (prevent overfitting)

- L2/L1 regularisation - add penalty term (magnitude chosen by cross-validation)
- Dropout - make weights = 0 during training with fixed prob. p (usually p = 0.5, but can be chosen by CV)
- Early stopping.
- Unsupervised pretraining (modelling p(x) to initialise weights) used to be a form of regularisation, ~~(force netw~~ but nowadays it is shown to be outperformed by dropout + RELU.