CSCI 567: Machine Learning

Vatsal Sharan

Spring 2024

Lecture 6, Feb 16

USC University of Southern California

Administrivia

- No OH or mentoring sessions on Monday, can find schedule for rest of the week on course calendar.
- HW2 due next Thursday at 12 noon (12 hour extension due to holiday on Monday)
- Exam 1 is in 2 weeks (March 1, 1pm-3:20pm).

Multiclass classification

1.1 Setup

Recall the setup:

- input (feature vector): $\boldsymbol{x} \in \mathbb{R}^d$
- output (label): $y \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$
- goal: learn a mapping $f : \mathbb{R}^d \rightarrow [\mathsf{C}]$

Examples:

- recognizing digits ($\mathsf{C} = 10$) or letters ($\mathsf{C} = 26$ or 52)
- predicting weather: sunny, cloudy, rainy, etc
- predicting image category: ImageNet dataset ($\mathsf{C} \approx 20K$)

1.2 Linear models: Binary to multiclass

Step 1: *What should a linear model look like for multiclass tasks?*

Note: a linear model for binary tasks (switching from $\{-1, +1\}$ to $\{1, 2\}$)

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \geq 0 \\ 2 & \text{if } \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} < 0 \end{cases}$$

can be written as

$$f(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} \geq \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} \\ 2 & \text{if } \boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x} > \boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x} \end{cases}$$

$$= \operatorname*{argmax}_{k \in \{1,2\}} \boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$$

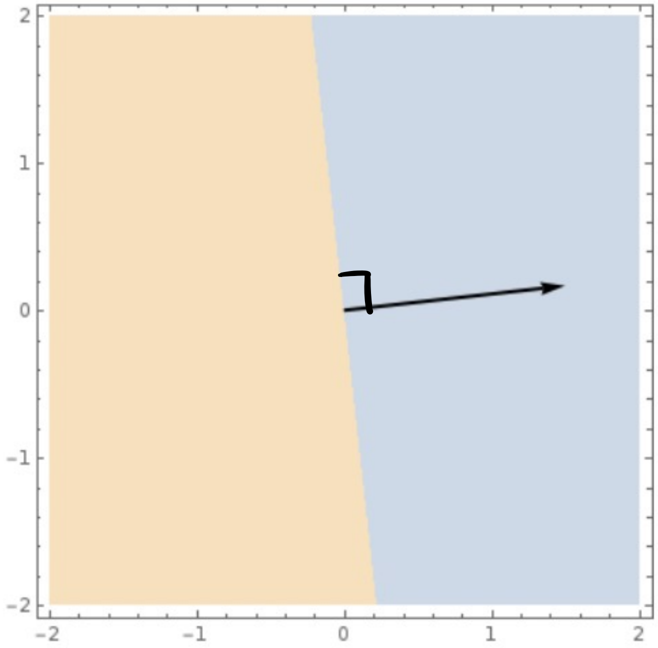
w_1, w_2 s.t.

$w = w_1 - w_2$

for any $\boldsymbol{w}_1, \boldsymbol{w}_2$ s.t. $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_2$

Think of $\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$ as **a score for class k**.

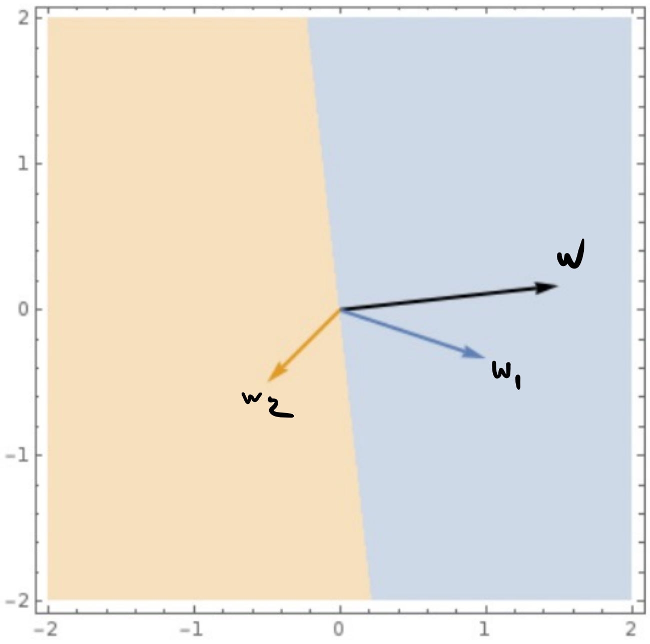
Linear models: Binary to multiclass



$$\boldsymbol{w} = (\tfrac{3}{2}, \tfrac{1}{6})$$

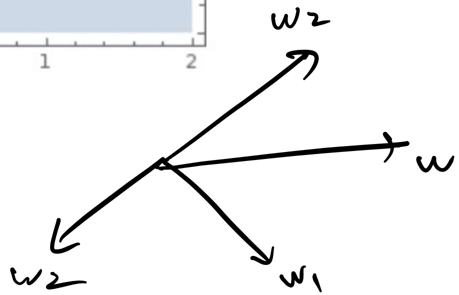
- Blue class:
  $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} \geq 0\}$
- Orange class:
  $\{\boldsymbol{x} : \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} < 0\}$

Linear models: Binary to multiclass

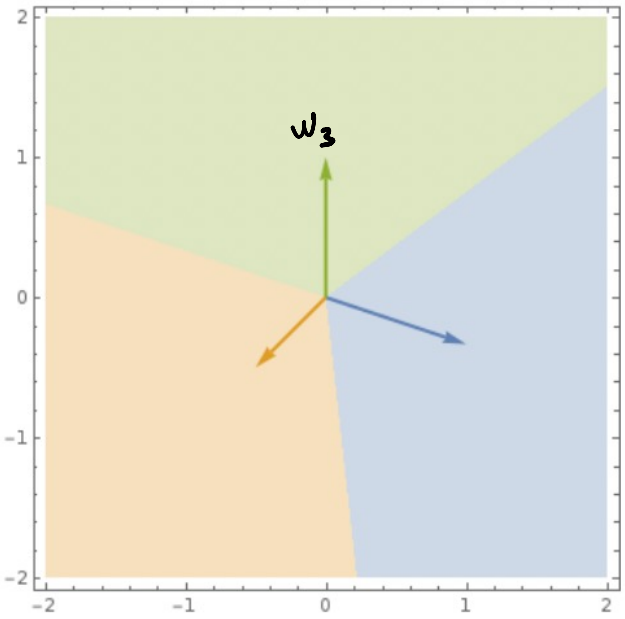


$$\boldsymbol{w} = (\tfrac{3}{2}, \tfrac{1}{6}) = \boldsymbol{w}_1 - \boldsymbol{w}_2$$
$$\boldsymbol{w}_1 = (1, -\tfrac{1}{3})$$
$$\boldsymbol{w}_2 = (-\tfrac{1}{2}, -\tfrac{1}{2})$$

- Blue class:
  $\{\boldsymbol{x} : 1 = \text{argmax}_k\, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Orange class:
  $\{\boldsymbol{x} : 2 = \text{argmax}_k\, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$



Linear models: Binary to multiclass



$\boldsymbol{w}_1 = (1, -\frac{1}{3})$
$\boldsymbol{w}_2 = (-\frac{1}{2}, -\frac{1}{2})$
$\boldsymbol{w}_3 = (0, 1)$

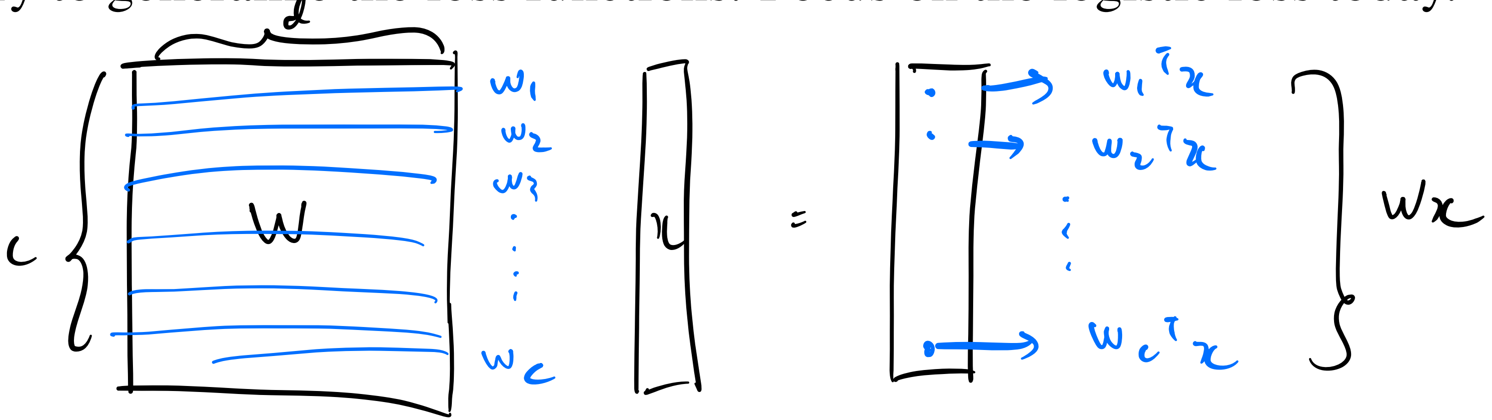
- Blue class:
  $\{\boldsymbol{x} : 1 = \text{argmax}_k \, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Orange class:
  $\{\boldsymbol{x} : 2 = \text{argmax}_k \, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$
- Green class:
  $\{\boldsymbol{x} : 3 = \text{argmax}_k \, \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}\}$

1.3 **Function class:** Linear models for multiclass classification

$$\mathcal{F} = \left\{ f(\boldsymbol{x}) = \underset{k \in [\mathsf{C}]}{\operatorname{argmax}} \; \boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x} \mid \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{\mathsf{C}} \in \mathbb{R}^d \right\}$$

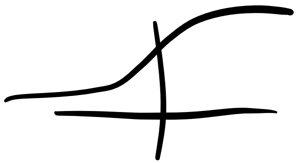
$$= \left\{ f(\boldsymbol{x}) = \underset{k \in [\mathsf{C}]}{\operatorname{argmax}} \; (\boldsymbol{W}\boldsymbol{x})_k \mid \boldsymbol{W} \in \mathbb{R}^{\mathsf{C} \times d} \right\}$$

Next, lets try to generalize the loss functions. Focus on the logistic loss today.



1.4 Multinomial logistic regression: a probabilistic view

Observe: for binary logistic regression, with $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_2$:



$$\Pr(y = 1 \mid \boldsymbol{x}; \boldsymbol{w}) = \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}}} = \frac{e^{\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x}}}{e^{\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x}} + e^{\boldsymbol{w}_2^{\mathrm{T}}\boldsymbol{x}}} \propto e^{\boldsymbol{w}_1^{\mathrm{T}}\boldsymbol{x}}$$

$$\Pr(y=2 \mid x; w) = \frac{e^{w_2^T x}}{e^{w_1^T x} + e^{w_2^T x}} \propto e^{w_2^T x}$$

Naturally, for multiclass:

$$\Pr(y = i \mid \boldsymbol{x}; \boldsymbol{W}) = \frac{e^{\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}}}{\sum_{k \in [\mathsf{C}]} e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}}} \propto e^{\boldsymbol{w}_i^{\mathrm{T}}\boldsymbol{x}}$$

This is called the *softmax function*.

Converts scores $w_i^T x \rightarrow \Pr(y=i \mid x; w) = \frac{e^{w_i^T x}}{\sum_{k \in [C]} e^{w_k^T x}}$

1.5 Let's find the MLE

Maximize probability of seeing labels $y_1, \ldots, y_n$ given $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$

(iid assumption)

$$P(\boldsymbol{W}) = \prod_{i=1}^{n} \Pr(y_i \mid \boldsymbol{x}_i; \boldsymbol{W}) = \prod_{i=1}^{n} \frac{e^{\boldsymbol{w}_{y_i}^{\mathrm{T}} \boldsymbol{x}_i}}{\sum_{k \in [\mathsf{C}]} e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}_i}}$$

By taking **negative log**, this is equivalent to minimizing

$$F(\boldsymbol{W}) = \sum_{i=1}^{n} \ln \left(\frac{\sum_{k \in [\mathsf{C}]} e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}_i}}{e^{\boldsymbol{w}_{y_i}^{\mathrm{T}} \boldsymbol{x}_i}} \right) = \sum_{i=1}^{n} \ln \left(1 + \sum_{k \neq y_i} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i} \right)$$

This is the *multiclass logistic loss*. It is an upper-bound on the 0-1 misclassification loss:

$$\mathbb{I}[f(\boldsymbol{x}) \neq y] \leq \log_2 \left(1 + \sum_{k \neq y} e^{(\boldsymbol{w}_k - \boldsymbol{w}_y)^{\mathrm{T}} \boldsymbol{x}} \right)$$

$\left(\log_2(1 + e^x) \geq 1 \text{ if } x \geq 0 \right)$

When $\mathsf{C} = 2$, multiclass logistic loss is the same as binary logistic loss (let's verify).

Relating binary and multiclass logistic loss

$$F(W) = \sum_{i=1}^{n} \ln\left(1 + \sum_{k \neq y_i} e^{(w_k - w_{y_i})^T x}\right)$$

Consider any $i \in [n]$,

For $y_i = 1$, $\ln\left(1 + e^{(w_2 - w_1)^T x}\right)$

For $y_i = 2$, $\ln\left(1 + e^{(w_1 - w_2)^T x}\right)$

For $w = w_1 - w_2$, and transfering labels from $\{1,2\} \to \{1,-1\}$,

$$F(w) = \sum_{i=1}^{n} \ln\left(1 + e^{-y_i w^T x_i}\right)$$

1.6 Next, optimization

Apply **SGD**: what is the gradient of

$$F(\boldsymbol{W}) = \ln\left(1 + \sum_{k \neq y_i} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}\right)?$$

It's a C $\times$ d matrix. Let's focus on the k-th row:

If $k \neq y_i$:

$$\nabla_{\boldsymbol{w}_k^{\mathrm{T}}} F(\boldsymbol{W}) = \frac{e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}}{1 + \sum_{k \neq y_i} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}} \boldsymbol{x}_i^{\mathrm{T}} = \frac{e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}_i}}{e^{\boldsymbol{w}_{y_i}^{\mathrm{T}} \boldsymbol{x}_i} + \sum_{k \neq y_i} e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}_i}} \boldsymbol{x}_i^{\mathrm{T}} = \Pr(y = k \mid \boldsymbol{x}_i; \boldsymbol{W}) \boldsymbol{x}_i^{\mathrm{T}}$$

else:

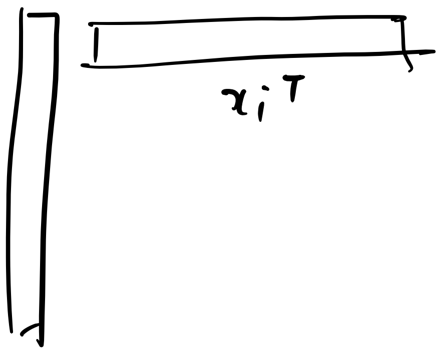
$$\nabla_{\boldsymbol{w}_k^{\mathrm{T}}} F(\boldsymbol{W}) = \frac{-\left(\sum_{k \neq y_i} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}\right)}{1 + \sum_{k \neq y_i} e^{(\boldsymbol{w}_k - \boldsymbol{w}_{y_i})^{\mathrm{T}} \boldsymbol{x}_i}} \boldsymbol{x}_i^{\mathrm{T}} = \frac{-\left(\sum_{k \neq y_i} e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}_i}\right)}{e^{\boldsymbol{w}_{y_i}^{\mathrm{T}} \boldsymbol{x}_i} + \sum_{k \neq y_i} e^{\boldsymbol{w}_k^{\mathrm{T}} \boldsymbol{x}_i}} \boldsymbol{x}_i^{\mathrm{T}} = (\Pr(y = y_i \mid \boldsymbol{x}_i; \boldsymbol{W}) - 1) \boldsymbol{x}_i^{\mathrm{T}}$$

SGD for multinomial logistic regression

Initialize $\boldsymbol{W} = \mathbf{0}$ (or randomly). Repeat:

1. pick $i \in [n]$ uniformly at random
2. update the parameters

$$\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta \begin{pmatrix} \Pr(y = 1 \mid \boldsymbol{x}_i; \boldsymbol{W}) \\ \vdots \\ \Pr(y = y_i \mid \boldsymbol{x}_i; \boldsymbol{W}) - 1 \\ \vdots \\ \Pr(y = \mathsf{C} \mid \boldsymbol{x}_i; \boldsymbol{W}) \end{pmatrix} \boldsymbol{x}_i^{\mathrm{T}}$$



Think about why the algorithm makes sense intuitively.

1.7 Probabilities -> Prediction

Having learned $\boldsymbol{W}$, we can either

- make a *deterministic* prediction $\mathrm{argmax}_{k\in[\mathsf{C}]}\ \boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}$
- make a *randomized* prediction according to $\Pr(y = k \mid \boldsymbol{x}; \boldsymbol{W}) \propto e^{\boldsymbol{w}_k^{\mathrm{T}}\boldsymbol{x}}$

1.8 Beyond linear models

Suppose we have any model f (not necessary linear) which gives some score $f_k(\boldsymbol{x})$ for the datapoint $\boldsymbol{x}$ having the k-th label.

For linear model, $f_k(x) = w_k^T x$

How can we convert this score to probabilities? Use the *softmax function*!

$$\tilde{f}_k(\boldsymbol{x}) = \Pr(y = k \mid \boldsymbol{x}; f) = \frac{e^{f_k(\boldsymbol{x})}}{\sum_{k' \in [\mathsf{C}]} e^{f_{k'}(\boldsymbol{x})}} \propto e^{f_k(\boldsymbol{x})}$$

Once we have probability estimates, what is suitable loss function to train the model? Use the *log loss*. Also known as the *cross-entropy loss*.

Log Loss/Cross-entropy loss: Binary case

Let's start with binary classification again. Consider a model which predicts $\tilde{f}(\boldsymbol{x})$ as the probability of label being 1 for labelled datapoint $(\boldsymbol{x}, y)$. The log loss is defined as,

$$\text{LogLoss} = \mathbf{1}(y=1)\ln\left(\frac{1}{\tilde{f}(\boldsymbol{x})}\right) + \mathbf{1}(y=-1)\ln\left(\frac{1}{1-\tilde{f}(\boldsymbol{x})}\right)$$

$$= -\mathbf{1}(y=1)\ln(\tilde{f}(\boldsymbol{x})) - \mathbf{1}(y=-1)\ln((1-\tilde{f}(\boldsymbol{x}))).$$

Why?
If $y=1$, want to max $\ln(\tilde{f}(x))$
$\Leftrightarrow$ min $\ln(1/\tilde{f}(x))$

When the model is linear, this reduces to the logistic regression loss we defined before!

Linear model: $\tilde{f}(x) = \sigma(w^T x) = \frac{1}{1+e^{-w^T x}}$, $1-\sigma(w^T x) = \sigma(-w^T x)$

Log loss $= -\mathbb{1}(y=1)\ln((1+e^{-w^T x})^{-1}) + \mathbb{1}(y=-1)\ln((1+e^{w^T x})^{-1})$

$= \ln(1+e^{-y w^T x})$ (logistic regression loss)

Log Loss/Cross-entropy loss: Multiclass case

This generalizes easily to the multiclass case. For datapoint $(\boldsymbol{x}, y)$, if $\tilde{f}_k(\boldsymbol{x})$ is the predicted probability of label k,

$$\text{LogLoss} = \sum_{k=1}^{\mathsf{C}} \mathbf{1}(y = k) \ln\left(\frac{1}{\tilde{f}_k(\boldsymbol{x})}\right)$$

$$= -\sum_{k=1}^{\mathsf{C}} \mathbf{1}(y = k) \ln\left(\tilde{f}_k(\boldsymbol{x})\right)).$$

When the model is linear, this also reduces to the multiclass logistic regression loss we defined earlier today.

Log Loss/Cross-entropy loss: Multiclass case

By combining the softmax and the log-loss, we have a general loss $\ell(f(\boldsymbol{x}), y)$ which we can use to train a multi-class classification model which assigns scores $f_k(\boldsymbol{x})$ to the k-th class. (These scores $f_k(\boldsymbol{x})$ are sometimes referred to as logits).

$$\begin{aligned}
\ell(f(\boldsymbol{x}), y) &= -\sum_{k=1}^{\mathsf{C}} \mathbf{1}(y = k) \ln\left(\tilde{f}_k(\boldsymbol{x})\right) \\
&= \ln\left(\frac{\sum_{k\in[\mathsf{C}]} e^{f_k(\boldsymbol{x})}}{e^{f_y(\boldsymbol{x})}}\right) \\
&= \ln\left(1 + \sum_{k\neq y} e^{f_k(\boldsymbol{x}) - f_y(\boldsymbol{x})}\right).
\end{aligned}$$

Multiclass logistic loss: Another view

Recall that we predict using

$$\arg\max_k f_k(x)$$

$$\ell(f(x), y) = \ln\left(\sum_k \exp(f_k(x))\right) - \ln(\exp(f_y(x)))$$

$$= \ln\left(\sum_k \exp(f_k(x))\right) - f_y(x)$$

$$\ln\left(\sum_k \exp(f_k(x))\right) \simeq \max_k f_k(x)$$

Verify: $\max_{k \in [c]} f_k(x) \leq \ln\left(\sum_{k \in [c]} \exp(f_k(x))\right) \leq \max_k f_k(x) + \ln($

$$\therefore \ \ell(f(x), y) \simeq \max_k f_k(x) - f_y(x)$$

1.8 Other techniques for multiclass classification

Cross-entropy is the most popular, but there are other *black-box techniques* to convert multiclass classification to binary classification.

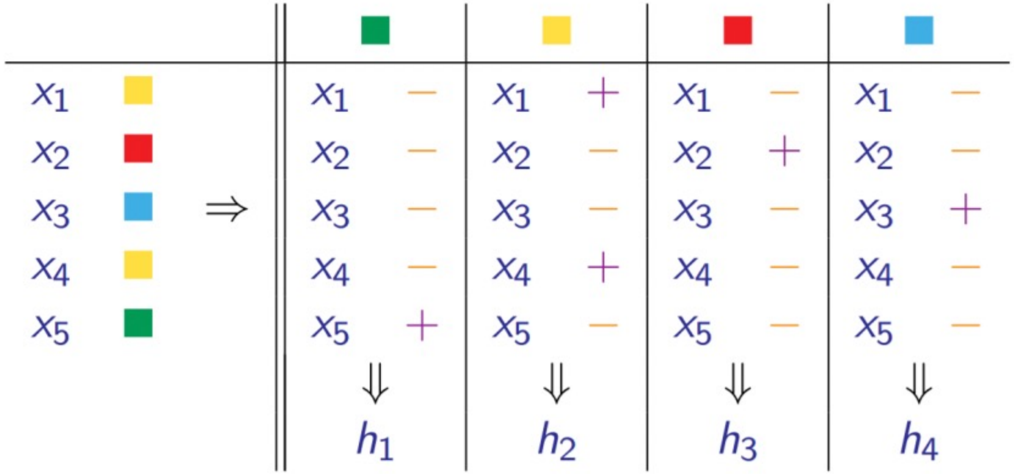
- **one-versus-all** (one-versus-rest, one-against-all, etc.)
- **one-versus-one** (all-versus-all, etc.)
- **Error-Correcting Output Codes** (ECOC)
- **tree-based reduction**

1.9 One-versus-all

Idea: train C binary classifiers to learn "**is class k or not?**" for each k.

Training: for each class $k \in [C]$,

- relabel examples with class k as $+1$, and all others as -1
- train a binary classifier h_k using this new dataset



Picture credits link

1.9 One-versus-all

Idea: train C binary classifiers to learn "**is class k or not?**" for each k.

Prediction: for a new example $\boldsymbol{x}$

- ask each h_k: **does this belong to class k?** (i.e. $h_k(\boldsymbol{x})$)
- randomly pick among all k's s.t. $h_k(\boldsymbol{x}) = +1$.

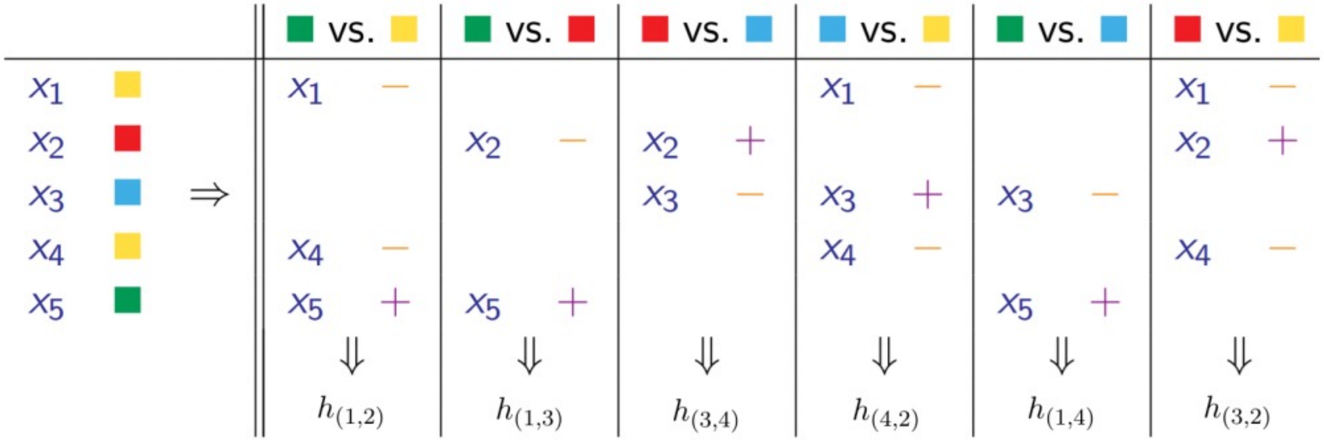
Issue: will (probably) make a mistake *as long as one of h_k errs*.

1.10 One-versus-one

Idea: train $\binom{C}{2}$ binary classifiers to learn "**is class k or k'?**".

Training: for each pair (k, k'),

- relabel examples with class k as $+1$ and examples with class k' as -1
- *discard all other examples*
- train a binary classifier $h_{(k,k')}$ using this new dataset



Picture credits link

1.10 One-versus-one

Idea: train $\binom{\mathsf{C}}{2}$ binary classifiers to learn "**is class k or k'?**".

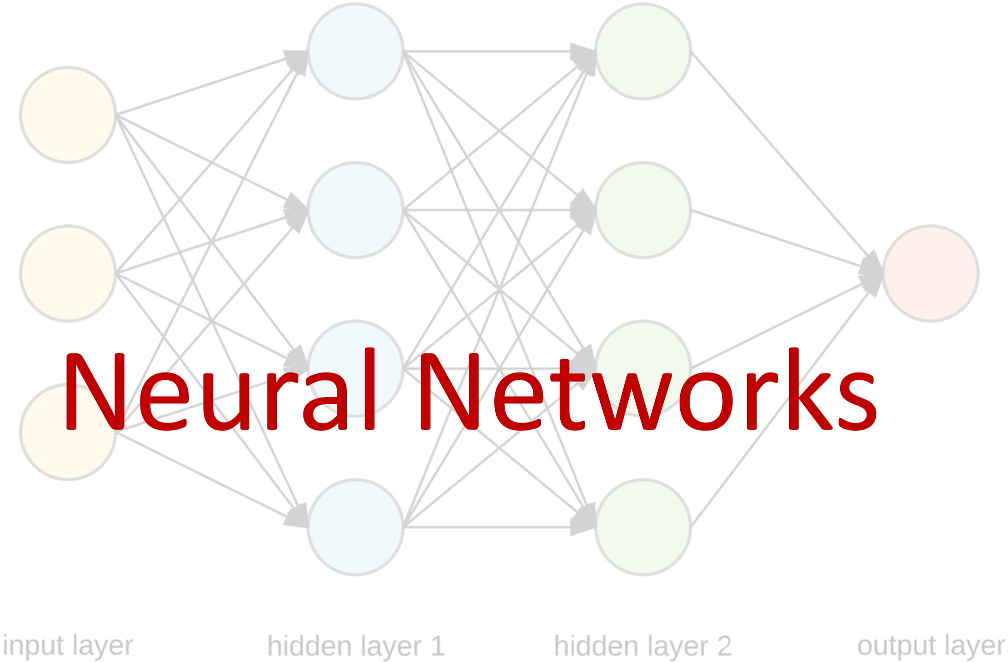
Prediction: for a new example $\boldsymbol{x}$

- ask each classifier $h_{(k,k')}$ to **vote for either class k or k'**
- predict the class with the most votes (break tie in some way)

More robust than one-versus-all, but *slower* in prediction.

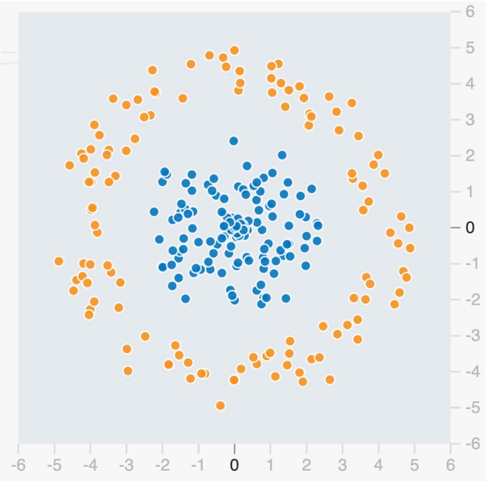
Other techniques such as tree-based methods and error-correcting codes can achieve intermediate tradeoffs.

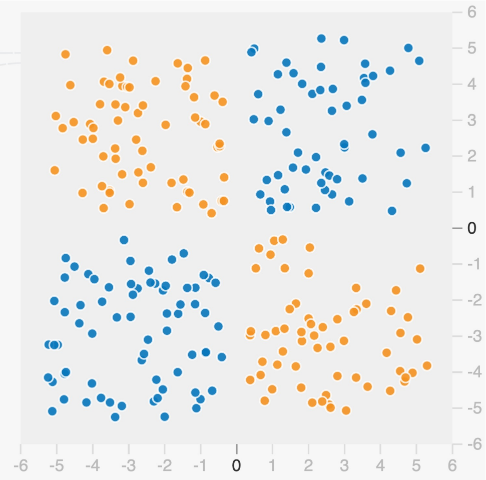
Neural Networks



Linear -> Fixed non-linear -> Learned non-linear map

Take polynomial kernel (add x_1^2, x_2^2)





Linear models aren't always enough. As we discussed, we can use a nonlinear mapping and learn a linear model in the feature space:

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^d \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

But what kind of nonlinear mapping ϕ should be used?

Can we just learn the nonlinear mapping itself?

Supervised learning in one slide

Loss function: What is the right loss function for the task?

Representation: What class of functions should we use?

Optimization: How can we efficiently solve the empirical risk minimization problem?

Generalization: Will the predictions of our model transfer gracefully to unseen examples?

All related! And the fuel which powers everything is data.

2.1 Loss function

For model which makes predictions $f(\boldsymbol{x})$ on labelled datapoint $(\boldsymbol{x}, y)$, we can use the following losses.

Regression:

$$\ell(f(\boldsymbol{x}), y) = (f(\boldsymbol{x}) - y)^2 .$$

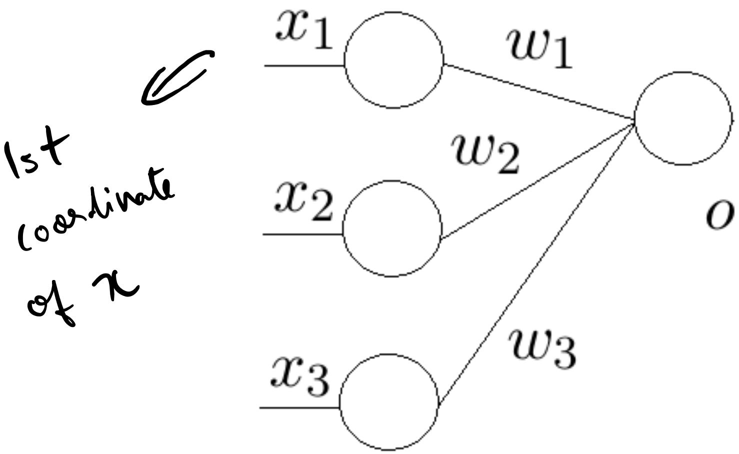
Classification:

$$\ell(f(\boldsymbol{x}), y) = \ln\left(\frac{\sum_{k \in [\mathsf{C}]} e^{f_k(\boldsymbol{x})}}{e^{f_y(\boldsymbol{x})}}\right) = \ln\left(1 + \sum_{k \neq y} e^{f_k(\boldsymbol{x}) - f_y(\boldsymbol{x})}\right) .$$

There maybe other, more suitable options for the problem at hand, but these are the most popular for supervised problems.

2.2 Representation: Defining neural networks

Linear model as a one-layer neural network:



$$o = h(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x})$$

For a linear model, $h(a) = a$.

To create non-linearity, can use some nonlinear (differentiable) function:

- Rectified Linear Unit (**ReLU**): $h(a) = \max\{0, a\}$
- Sigmoid function: $h(a) = \frac{1}{1+e^{-a}}$
- Tanh: $h(a) = \frac{e^{a}-e^{-a}}{e^{a}+e^{-a}}$
- many more

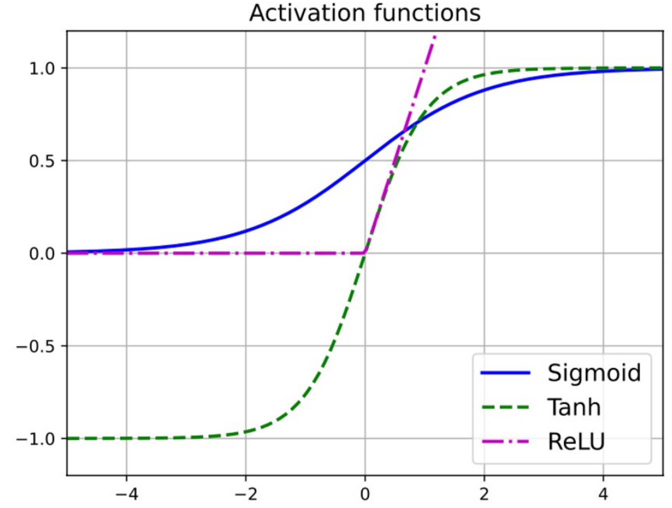
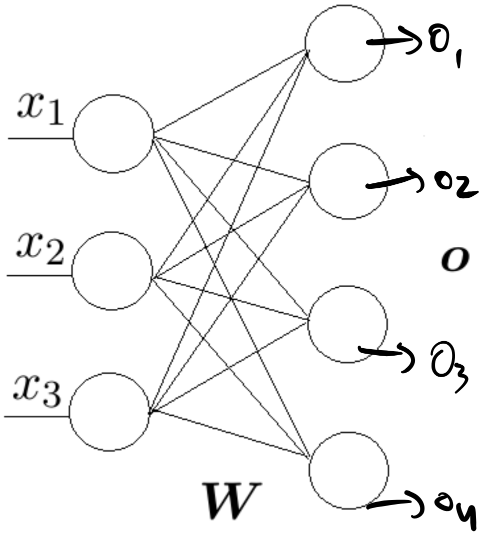
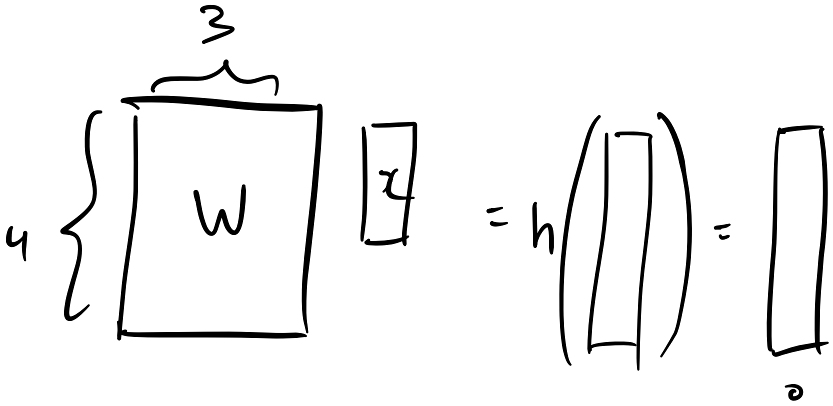


Figure 13.2 from PML

Adding a layer



$$\boldsymbol{o} = \boldsymbol{h}(\boldsymbol{W}\boldsymbol{x})$$



$\boldsymbol{W} \in \mathbb{R}^{4\times 3}$, $\boldsymbol{h} : \mathbb{R}^4 \to \mathbb{R}^4$ so $\boldsymbol{h}(\boldsymbol{a}) = (h_1(a_1), h_2(a_2), h_3(a_3), h_4(a_4))$

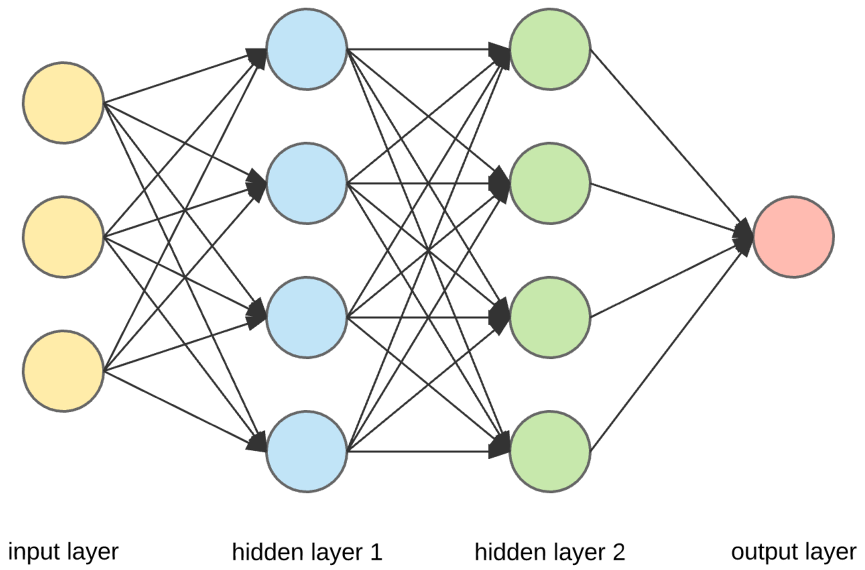
Can think of this as a nonlinear mapping: $\phi(\boldsymbol{x}) = \boldsymbol{h}(\boldsymbol{W}\boldsymbol{x})$

$$h(a) = (h(a_1), h(a_2), h(a_3), h(a_4))$$

Putting things together: a neural network

We now have a network:

- each node is called a **neuron**
- h is called the **activation function**
  - can use $h(a) = 1$ for one neuron in each layer to incorporate bias term
  - output neuron can use $h(a) = a$
- #layers refers to #hidden_layers (plus 1 or 2 for input/output layers)
- **deep** neural nets can have many layers and *millions* of parameters
- this is a **feedforward, fully connected** neural net, there are many variants (convolutional nets, residual nets, recurrent nets, etc.)



Neural network: Definition

An L-layer neural net can be written as

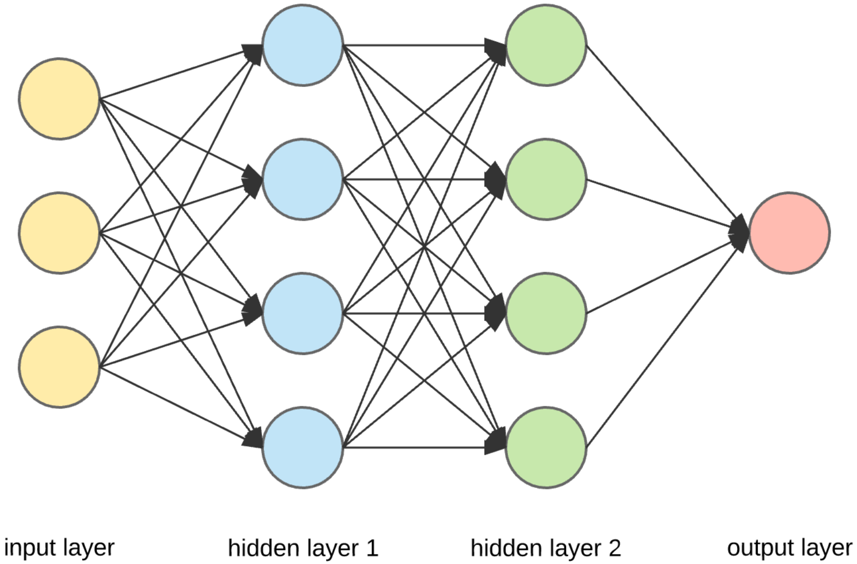
$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{h}_{\mathsf{L}}\left(\boldsymbol{W}_L \boldsymbol{h}_{\mathsf{L}-1}\left(\boldsymbol{W}_{L-1} \cdots \boldsymbol{h}_1\left(\boldsymbol{W}_1 \boldsymbol{x}\right)\right)\right).$$

input: output of previous layer
output: input of next layer

Define

- $\boldsymbol{W}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ is the weights between layer $\ell - 1$ and ℓ
- $d_0 = d, d_1, \ldots, d_{\mathsf{L}}$ are numbers of neurons at each layer
- $\boldsymbol{a}_\ell \in \mathbb{R}^{d_\ell}$ is input to layer ℓ
- $\boldsymbol{o}_\ell \in \mathbb{R}^{d_\ell}$ is output of layer ℓ
- $\boldsymbol{h}_\ell : \mathbb{R}^{d_\ell} \to \mathbb{R}^{d_\ell}$ is activation functions at layer ℓ

input: $W_1 x$
output: $h_1(W_1 x)$



Now, for a given input $\boldsymbol{x}$, we have recursive relations:

$$\boldsymbol{o}_0 = \boldsymbol{x}, \boldsymbol{a}_\ell = \boldsymbol{W}_\ell \boldsymbol{o}_{\ell-1}, \boldsymbol{o}_\ell = \boldsymbol{h}_\ell(\boldsymbol{a}_\ell), \quad (\ell = 1, \ldots, \mathsf{L}).$$

2.3 Optimization

Our optimization problem is to minimize,

$$F(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_\mathsf{L}) = \frac{1}{n} \sum_{i=1}^{n} F_i(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_\mathsf{L})$$

where

$$F_i(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_\mathsf{L}) = \begin{cases} \|\boldsymbol{f}(\boldsymbol{x}_i) - \boldsymbol{y}_i\|_2^2 & \text{for regression} \\ \ln\left(1 + \sum_{k \neq y_i} e^{f_k(\boldsymbol{x}_i) - f_{y_i}(\boldsymbol{x}_i)}\right) & \text{for classification} \end{cases}$$

How to solve this? Apply **SGD**!

To compute the gradient efficiently, we use *backpropogation*. More on this soon.

2.4 Generalization

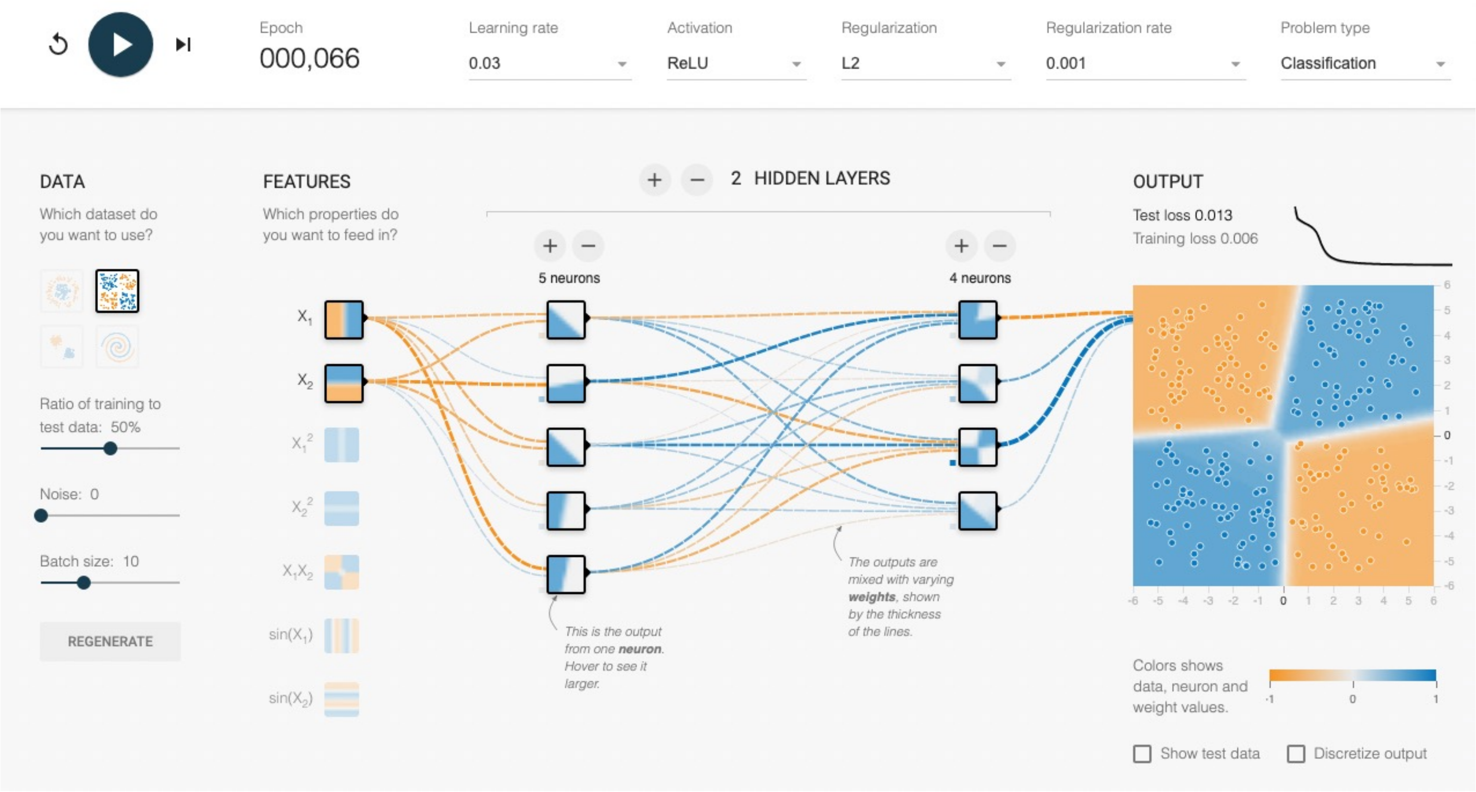
Overfitting is a concern for such a complex model, but there are ways to handle it.

For example, we can add ℓ_2 regularization.

ℓ_2 **regularization**: minimize

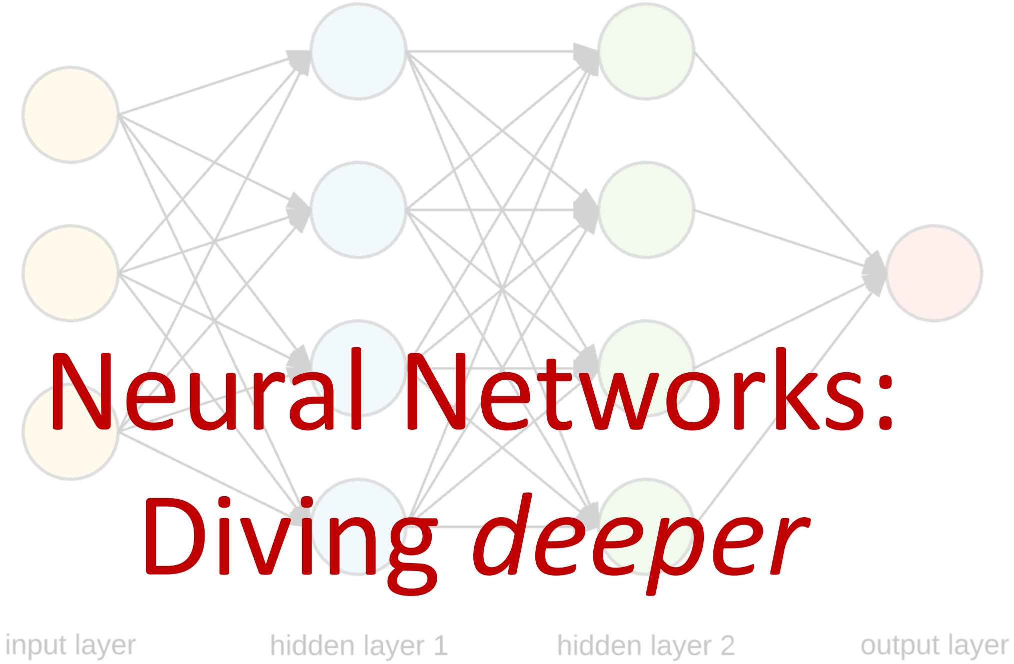
$$G(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{\mathsf{L}}) = F(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{\mathsf{L}}) + \lambda \sum_{\substack{\text{all weights } w \\ \text{in network}}} w^2$$

Demo



http://playground.tensorflow.org/

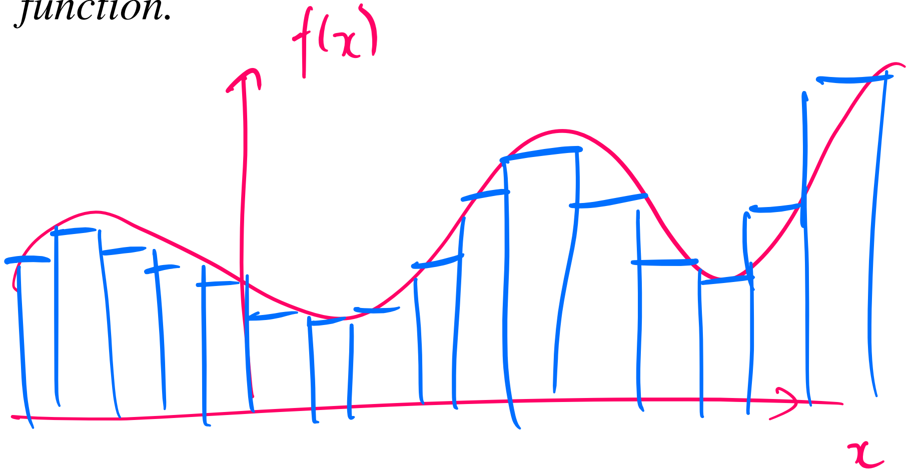
Neural Networks: Diving *deeper*

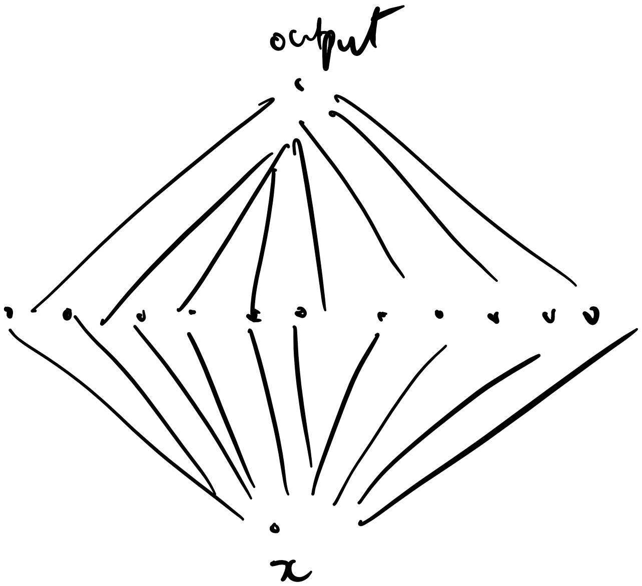


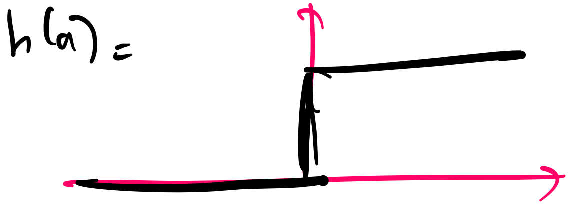
3.1 Representation: Very powerful function class!

Universal approximation theorem (Cybenko, 89; Hornik, 91):

A feedforward neural net with a single hidden layer can approximate any continuous function.



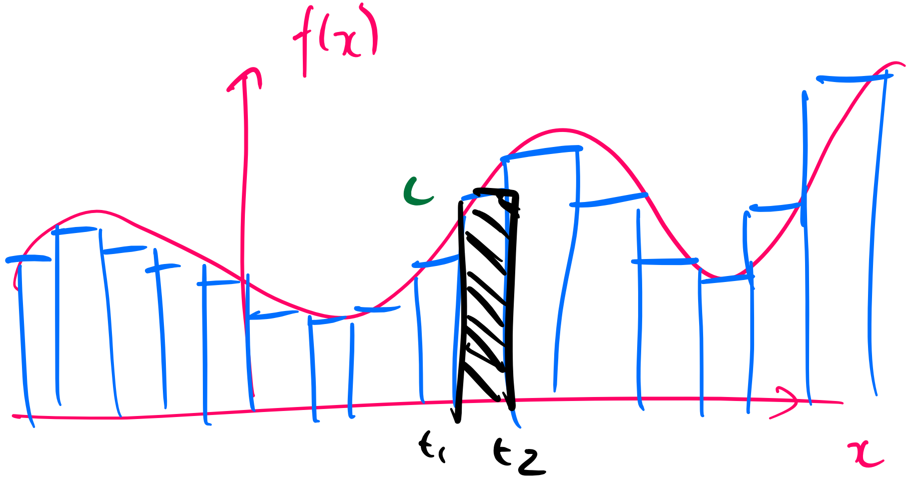


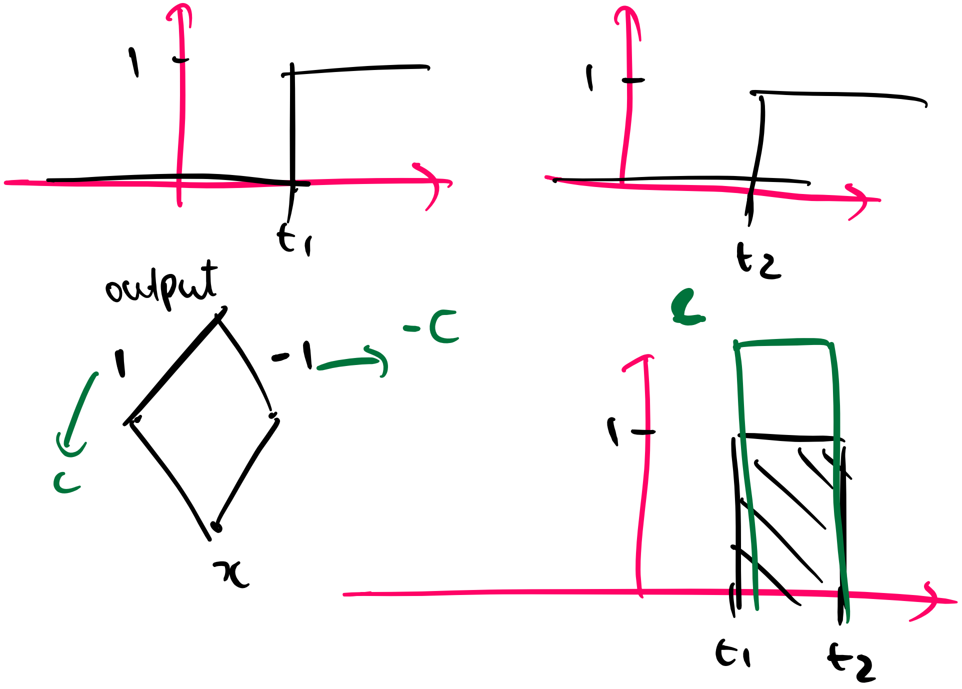


3.1 Representation: Very powerful function class!

Universal approximation theorem (Cybenko, 89; Hornik, 91):

A feedforward neural net with a single hidden layer can approximate any continuous function.





3.1 Representation: Very powerful function class!

Universal approximation theorem (Cybenko, 89; Hornik, 91):

A feedforward neural net with a single hidden layer can approximate any continuous function.

It might need a huge number of neurons though, and *depth helps!*

Choosing the network architecture is important.

- for feedforward network, need to decide number of hidden layers, number of neurons at each layer, activation functions, etc.

Designing the architecture can be complicated, though various standard choices exist.

3.2 Optimization: Computing gradients efficiently using Backprop

To run SGD, need gradients of $F_i(\boldsymbol{W}_1, \ldots, \boldsymbol{W}_\mathsf{L})$ with respect to all the weights in all the layers. How do we get the gradient?

Here's a naive way to compute gradients. For some function $F(w)$ of a univariate parameter w,

$$\frac{dF(w)}{dw} = \lim_{\epsilon \to 0} \frac{F(w+\epsilon) - F(w-\epsilon)}{2\epsilon}$$

If our network has m weights, this requires O(m) evaluations of model.

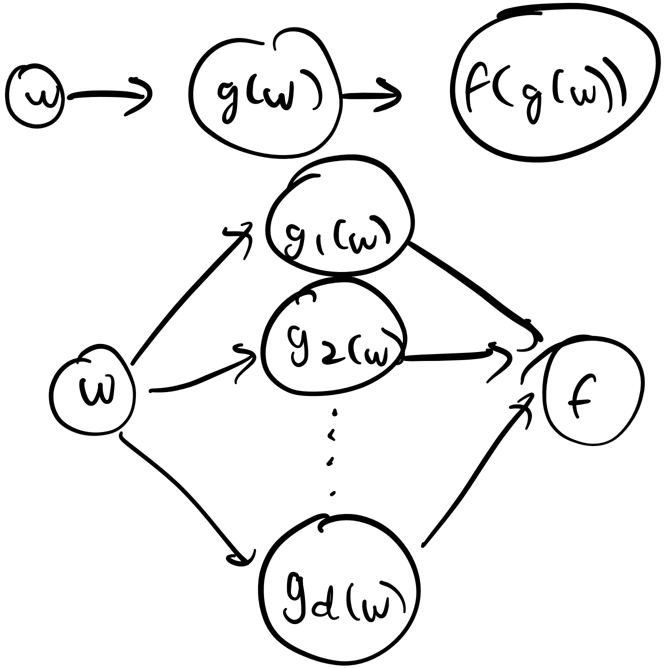
Backprop

Backpropogation: A very efficient way to compute gradients of neural networks using an application of the chain rule (similar to dynamic programming).

Chain rule:

- for a composite function $f(g(w))$

$$\frac{\partial f}{\partial w} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial w}$$

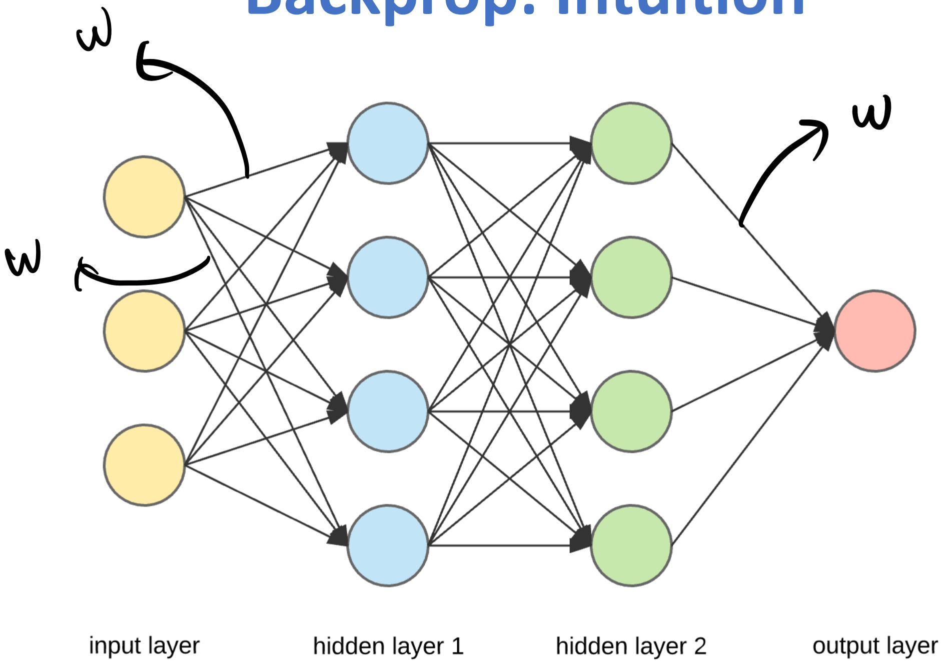


- for a composite function $f(g_1(w), \ldots, g_d(w))$

$$\frac{\partial f}{\partial w} = \sum_{i=1}^{d} \frac{\partial f}{\partial g_i}\frac{\partial g_i}{\partial w}$$

the simplest example $f(g_1(w), g_2(w)) = g_1(w)g_2(w)$

Backprop: Intuition

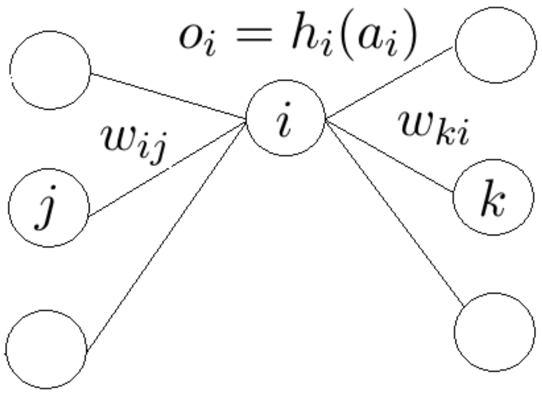


Naively: apply chain rule separately for each weight

Backprop: reuse computation by storing gradients wrt input to each layer (a_l)

Backprop: Derivation

Drop the subscript ℓ for layer for simplicity. For this derivation, refer to the loss function as F_m (instead of F_i) for convenience.



Find the **derivative of F_m w.r.t. to w_{ij}**

$$\frac{\partial F_m}{\partial w_{ij}} = \frac{\partial F_m}{\partial a_i}\frac{\partial a_i}{\partial w_{ij}} = \frac{\partial F_m}{\partial a_i}\frac{\partial (w_{ij} o_j)}{\partial w_{ij}} = \frac{\partial F_m}{\partial a_i} o_j$$

$$\frac{\partial F_m}{\partial a_i} = \frac{\partial F_m}{\partial o_i}\frac{\partial o_i}{\partial a_i} = \left(\sum_k \frac{\partial F_m}{\partial a_k}\frac{\partial a_k}{\partial o_i}\right) h_i'(a_i) = \left(\sum_k \frac{\partial F_m}{\partial a_k} w_{ki}\right) h_i'(a_i)$$

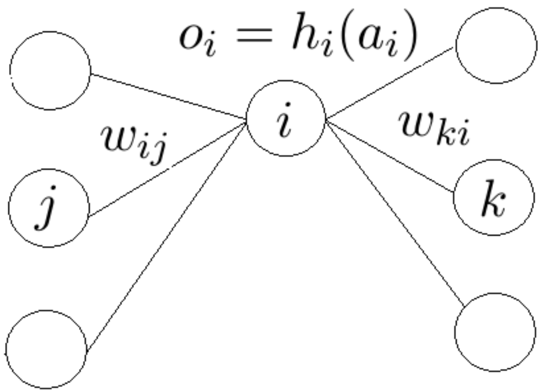
key quantity to store

Backprop: Derivation

Adding the subscript for layer:

$$\frac{\partial F_m}{\partial w_{\ell,ij}} = \frac{\partial F_m}{\partial a_{\ell,i}} o_{\ell-1,j}$$

$$\frac{\partial F_m}{\partial a_{\ell,i}} = \left(\sum_k \frac{\partial F_m}{\partial a_{\ell+1,k}} w_{\ell+1,ki} \right) h'_{\ell,i}(a_{\ell,i})$$



For the last layer, for square loss

$$\frac{\partial F_m}{\partial a_{\mathsf{L},i}} = \frac{\partial (h_{\mathsf{L},i}(a_{\mathsf{L},i}) - y_m)^2}{\partial a_{\mathsf{L},i}} = 2(h_{\mathsf{L},i}(a_{\mathsf{L},i}) - y_m) h'_{\mathsf{L},i}(a_{\mathsf{L},i})$$

Exercise: try to do it for logistic loss yourself.

We will continue with Backprop next time!