Differentiable Programming

(and some Deep Learning)

Jonathon Hare

Vision, Learning and Control
University of Southampton

Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data	$\{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N$	$\{\boldsymbol{x}_n\}_{n=1}^N$

Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data	$\{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N$	$\{\boldsymbol{x}_n\}_{n=1}^N$
Function Approximator	$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta}) + \nu$	

Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data	$\{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N$	$\{\boldsymbol{x}_n\}_{n=1}^N$
Function Approximator	$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta}) + \nu$	
Parameter Estimation	$E_0 = \sum_{n=1}^N \{\|\boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})\|\}^2$	

Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data	$\{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N$	$\{\boldsymbol{x}_n\}_{n=1}^N$
Function Approximator	$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta}) + \nu$	
Parameter Estimation	$E_0 = \sum_{n=1}^N \{\|\boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})\|\}^2$	
Prediction	$\hat{\boldsymbol{y}}_{N+1} = f(\boldsymbol{x}_{N+1}, \hat{\boldsymbol{\theta}})$	

Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data	$\{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N \qquad \{\boldsymbol{x}_n\}_{n=1}^N$
Function Approximator	$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta}) + \nu$
Parameter Estimation	$E_0 = \sum_{n=1}^N \{\|\boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})\|\}^2$
Prediction	$\hat{\boldsymbol{y}}_{N+1} = f(\boldsymbol{x}_{N+1}, \hat{\boldsymbol{\theta}})$
Regularisation	$E_1 = \sum_{n=1}^N \{\|\boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})\|\}^2 + r(\|\boldsymbol{\theta}\|)$

Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data	$\{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N \qquad \{\boldsymbol{x}_n\}_{n=1}^N$
Function Approximator	$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta}) + \nu$
Parameter Estimation	$E_0 = \sum_{n=1}^N \{\|\boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})\|\}^2$
Prediction	$\hat{\boldsymbol{y}}_{N+1} = f(\boldsymbol{x}_{N+1}, \hat{\boldsymbol{\theta}})$
Regularisation	$E_1 = \sum_{n=1}^N \{\|\boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})\|\}^2 + r(\|\boldsymbol{\theta}\|)$
Modelling Uncertainty	$p(\boldsymbol{\theta} \| \{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N)$

Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data	$\{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N \qquad \{\boldsymbol{x}_n\}_{n=1}^N$
Function Approximator	$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta}) + \nu$
Parameter Estimation	$E_0 = \sum_{n=1}^N \{\|\boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})\|\}^2$
Prediction	$\hat{\boldsymbol{y}}_{N+1} = f(\boldsymbol{x}_{N+1}, \hat{\boldsymbol{\theta}})$
Regularisation	$E_1 = \sum_{n=1}^N \{\|\boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})\|\}^2 + r(\|\boldsymbol{\theta}\|)$
Modelling Uncertainty	$p(\boldsymbol{\theta}\|\{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N)$
Probabilistic Inference	$\mathbb{E}[g(\boldsymbol{\theta})] = \int g(\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} = \frac{1}{N_s} \sum_{n=1}^{N_s} g(\boldsymbol{\theta}^{(n)})$

Machine Learning - A Recap

All credit for this slide goes to Niranjan

Data	$\{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N \qquad \{\boldsymbol{x}_n\}_{n=1}^N$
Function Approximator	$\boldsymbol{y} = f(\boldsymbol{x}, \boldsymbol{\theta}) + \nu$
Parameter Estimation	$E_0 = \sum_{n=1}^N \{\|\boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})\|\}^2$
Prediction	$\hat{\boldsymbol{y}}_{N+1} = f(\boldsymbol{x}_{N+1}, \hat{\boldsymbol{\theta}})$
Regularisation	$E_1 = \sum_{n=1}^N \{\|\boldsymbol{y}_n - f(\boldsymbol{x}_n; \boldsymbol{\theta})\|\}^2 + r(\|\boldsymbol{\theta}\|)$
Modelling Uncertainty	$p(\boldsymbol{\theta}\|\{\boldsymbol{x}_n, \boldsymbol{y}_n\}_{n=1}^N)$
Probabilistic Inference	$\mathbb{E}[g(\boldsymbol{\theta})] = \int g(\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} = \frac{1}{N_s} \sum_{n=1}^{N_s} g(\boldsymbol{\theta}^{(n)})$
Sequence Modelling	$\boldsymbol{x}_n = f(\boldsymbol{x}_{n-1}, \boldsymbol{\theta})$

What is Deep Learning?

Deep learning is primarily characterised by function compositions:

What is Deep Learning?

Deep learning is primarily characterised by function compositions:

- Feedforward networks: $\boldsymbol{y} = f(g(\boldsymbol{x}, \boldsymbol{\theta}_g), \boldsymbol{\theta_f})$
  - Often with relatively simple functions (e.g. $f(\boldsymbol{x}, \boldsymbol{\theta}_f) = \sigma(\boldsymbol{x}^\top \boldsymbol{\theta}_f)$)

What is Deep Learning?

Deep learning is primarily characterised by function compositions:

- Feedforward networks: $\boldsymbol{y} = f(g(\boldsymbol{x}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_f)$
  - Often with relatively simple functions (e.g. $f(\boldsymbol{x}, \boldsymbol{\theta}_f) = \sigma(\boldsymbol{x}^\top \boldsymbol{\theta}_f)$)
- Recurrent networks:
  $\boldsymbol{y}_t = f(\boldsymbol{y}_{t-1}, \boldsymbol{x}_t, \boldsymbol{\theta}) = f(f(\boldsymbol{y}_{t-2}, \boldsymbol{x}_{t-1}, \boldsymbol{\theta}), \boldsymbol{x}_t, \boldsymbol{\theta}) = \ldots$

What is Deep Learning?

Deep learning is primarily characterised by function compositions:

- Feedforward networks: $\boldsymbol{y} = f(g(\boldsymbol{x}, \boldsymbol{\theta}_g), \boldsymbol{\theta}_f)$
  - Often with relatively simple functions (e.g. $f(\boldsymbol{x}, \boldsymbol{\theta}_f) = \sigma(\boldsymbol{x}^\top \boldsymbol{\theta}_f)$)
- Recurrent networks:
  $\boldsymbol{y}_t = f(\boldsymbol{y}_{t-1}, \boldsymbol{x}_t, \boldsymbol{\theta}) = f(f(\boldsymbol{y}_{t-2}, \boldsymbol{x}_{t-1}, \boldsymbol{\theta}), \boldsymbol{x}_t, \boldsymbol{\theta}) = \ldots$

In the early days the focus of deep learning was on learning functions for classification. Nowadays the functions are much more general in their inputs and outputs.

What is Differentiable Programming?

- Differentiable programming is a term coined by Yann Lecun[1] to describe a superset of Deep Learning.

[1] https://www.facebook.com/yann.lecun/posts/10155003011462143

[2] See our ICLR 2019 paper: https://arxiv.org/abs/1812.03928 and NeurIPS 2019 paper: https://arxiv.org/abs/1906.06565

What is Differentiable Programming?

- Differentiable programming is a term coined by Yann Lecun[1] to describe a superset of Deep Learning.
- Captures the idea that computer programs can be constructed of parameterised functional blocks in which the parameters are learned using some form of gradient-based optimisation.

[1] https://www.facebook.com/yann.lecun/posts/10155003011462143

[2] See our ICLR 2019 paper: https://arxiv.org/abs/1812.03928 and NeurIPS 2019 paper: https://arxiv.org/abs/1906.06565

What is Differentiable Programming?

- Differentiable programming is a term coined by Yann Lecun[1] to describe a superset of Deep Learning.
- Captures the idea that computer programs can be constructed of parameterised functional blocks in which the parameters are learned using some form of gradient-based optimisation.
  - The implication is that we need to be able to compute gradients with respect to the parameters of these functional blocks. We'll start explore this in detail next week...

[1] https://www.facebook.com/yann.lecun/posts/10155003011462143

[2] See our ICLR 2019 paper: https://arxiv.org/abs/1812.03928 and NeurIPS 2019 paper: https://arxiv.org/abs/1906.06565

What is Differentiable Programming?

- Differentiable programming is a term coined by Yann Lecun[1] to describe a superset of Deep Learning.
- Captures the idea that computer programs can be constructed of parameterised functional blocks in which the parameters are learned using some form of gradient-based optimisation.
  - The implication is that we need to be able to compute gradients with respect to the parameters of these functional blocks. We'll start explore this in detail next week...
  - The idea of Differentiable Programming also opens up interesting possibilities:
    - The functional blocks don't need to be direct functions in a mathematical sense; more generally they can be *algorithms*.
    - What if the functional block we're learning parameters for is itself an algorithm that optimises the parameters of an internal algorithm using a gradient based optimiser?![2]

[1] https://www.facebook.com/yann.lecun/posts/10155003011462143

[2] See our ICLR 2019 paper: https://arxiv.org/abs/1812.03928 and NeurIPS 2019 paper: https://arxiv.org/abs/1906.06565

Is all Deep Learning Differentiable Programming?

- Not necessarily!
  - Most deep learning systems are trained using first order gradient-based optimisers, but there is an active body of research on gradient-free methods.

[3] including at least myself, my PhD students and Geoff Hinton!

Is all Deep Learning Differentiable Programming?

- Not necessarily!
  - Most deep learning systems are trained using first order gradient-based optimisers, but there is an active body of research on gradient-free methods.
  - There is an increasing interest in methods that use different styles of learning, such as Hebbian learning, within deep networks. More broadly there are a number of us[3] who are interested in biologically motivated models and learning methods.

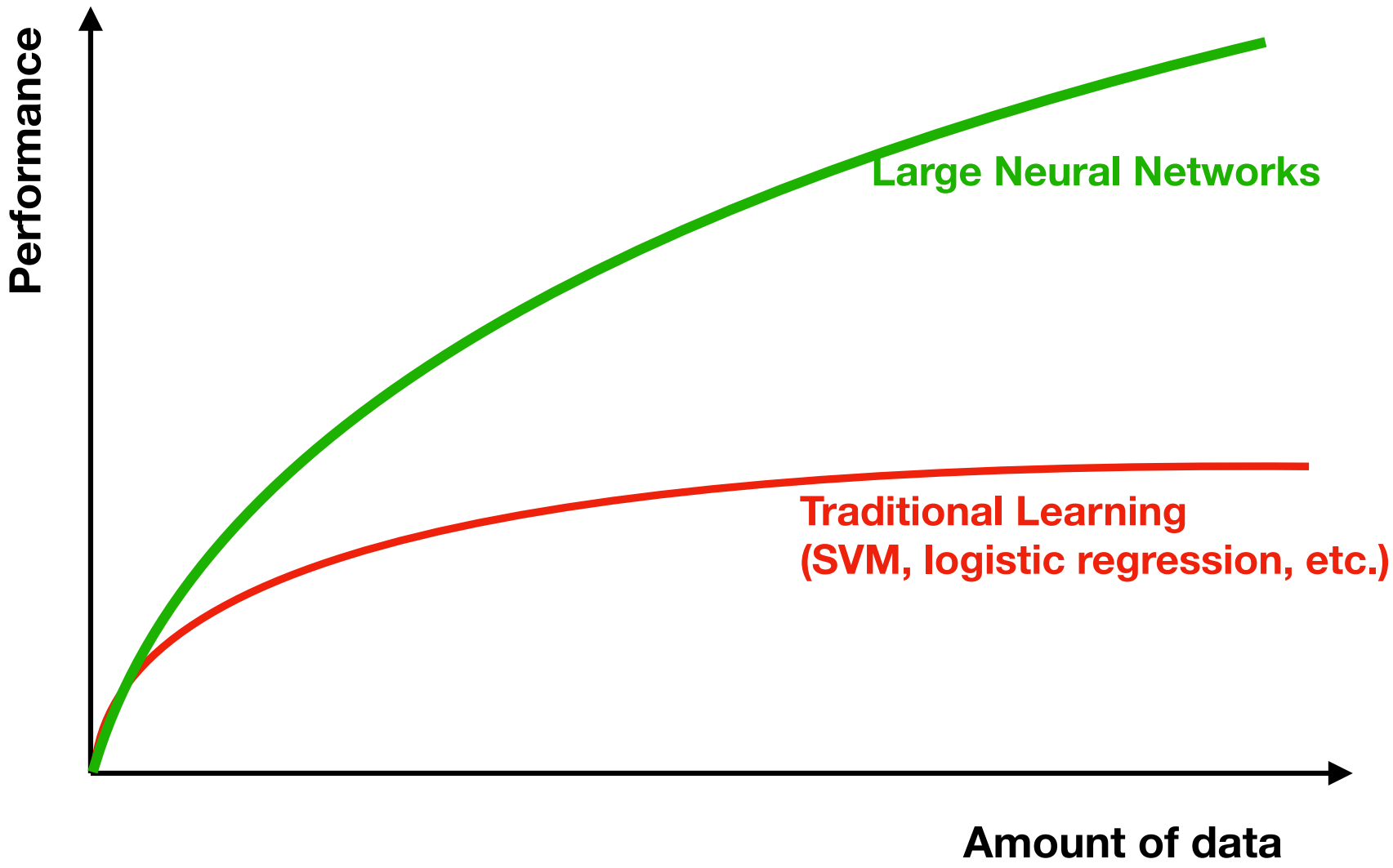
[3] including at least myself, my PhD students and Geoff Hinton!

Is all Deep Learning Differentiable Programming?

- Not necessarily!
  - Most deep learning systems are trained using first order gradient-based optimisers, but there is an active body of research on gradient-free methods.
  - There is an increasing interest in methods that use different styles of learning, such as Hebbian learning, within deep networks. More broadly there are a number of us[3] who are interested in biologically motivated models and learning methods.
  - This course will primarily focus on differentiable methods, but we'll look at how relaxations can be made to make non-differentiable operators learnable with gradient-based optimisers.

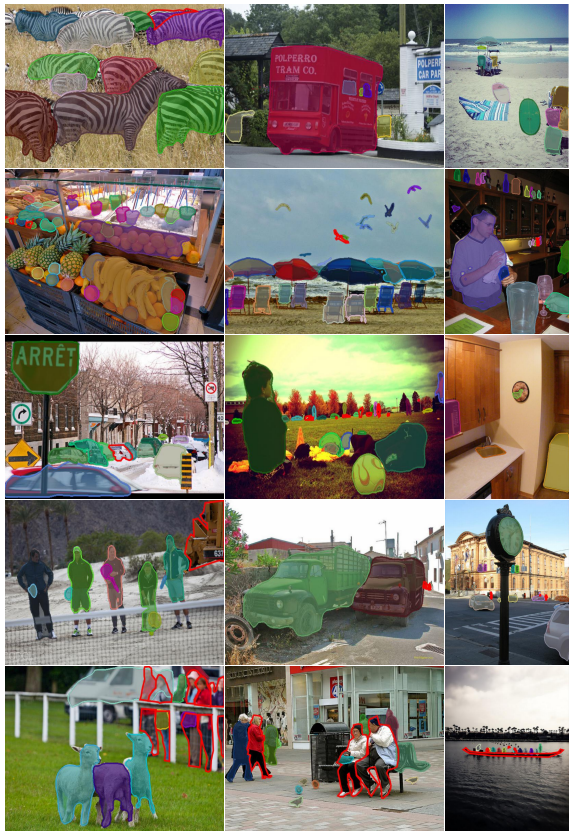
[3] including at least myself, my PhD students and Geoff Hinton!

Why should we care about this?



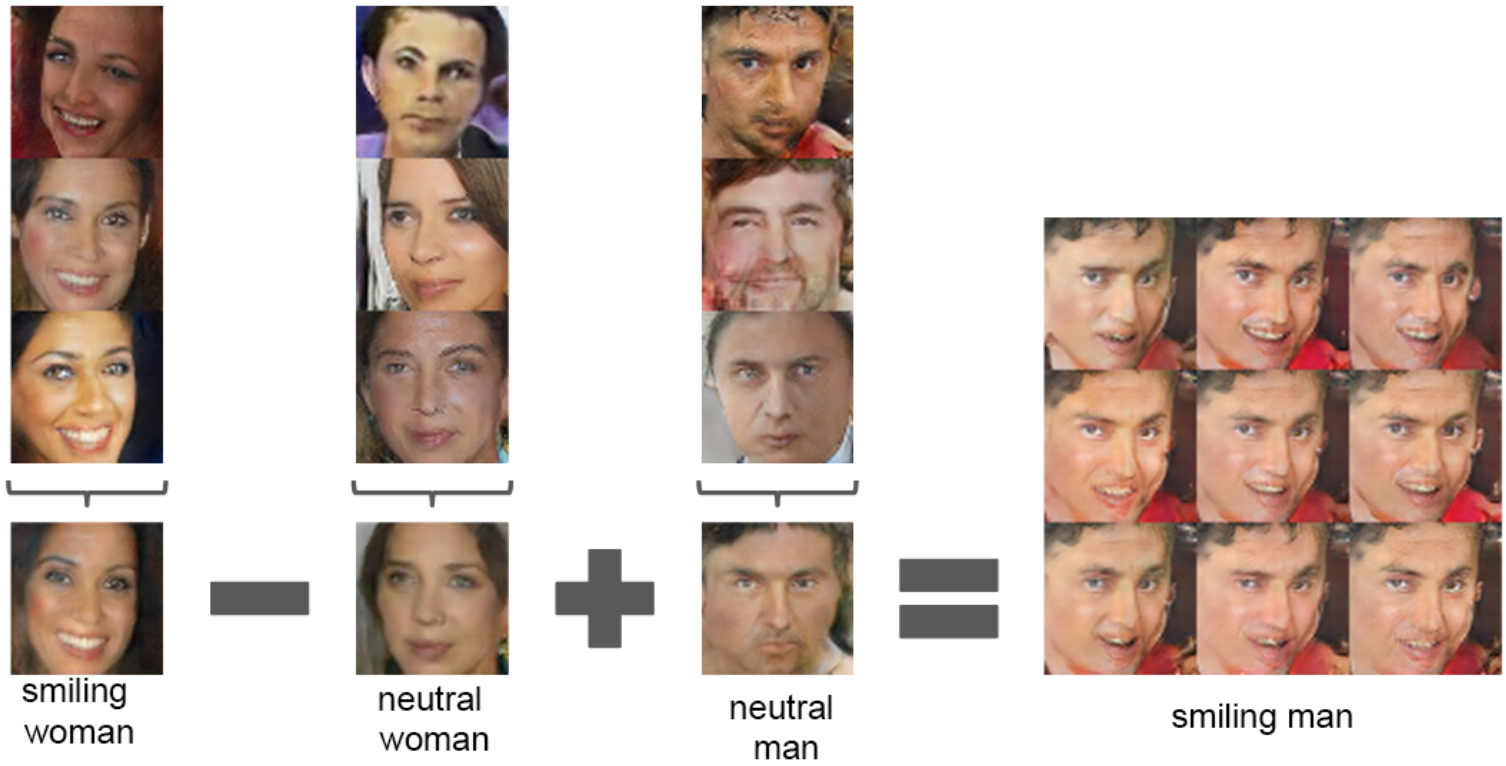
Reference: Andrew Ng

Success stories - Object detection and segmentation



Pinheiro, Pedro O., et al. "Learning to refine object segments." European Conference on Computer Vision. Springer, 2016.

Success stories - Image generation



Radford, Alec, Luke Metz, and Soumith Chintala. "Unsupervised representation learning with deep convolutional generative adversarial networks." arXiv preprint arXiv:1511.06434 (2015).

Success stories - Translation

ENGLISH TEXT

The reason Boeing are doing this is to cram more seats in to make their plane more competitive with our products," said Kevin Keniston, head of passenger comfort at Europe's Airbus.

TRANSLATED TO FRENCH

La raison pour laquelle Boeing fait cela est de creer plus de sieges pour rendre son avion plus competitif avec nos produits", a declare Kevin Keniston, chef du confort des passagers chez Airbus.

Wu, Yonghui, et al. "Google's neural machine translation system: Bridging the gap between human and machine translation." arXiv preprint arXiv:1609.08144 (2016).

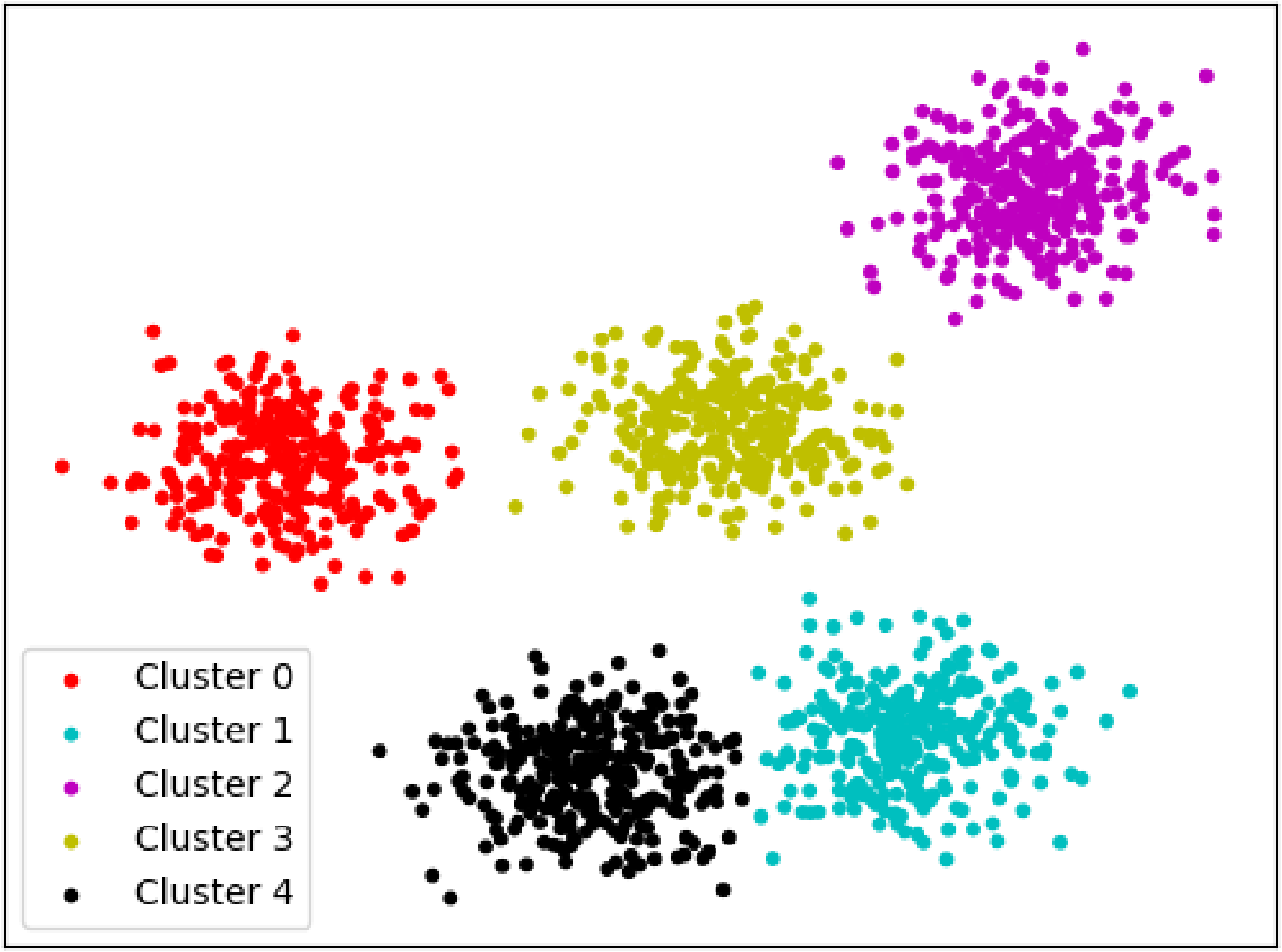
Types of Learning

- Supervised Learning - learn to predict an output when given an input vector
- Unsupervised Learning - discover a good internal representation of the input
- Reinforcement Learning - learn to select an action to maximize the expectation of future rewards (payoff)
- Self-supervised Learning - learn with targets induced by a prior on the unlabelled training data
- Semi-supervised Learning - learn with few labelled examples and many unlabelled ones

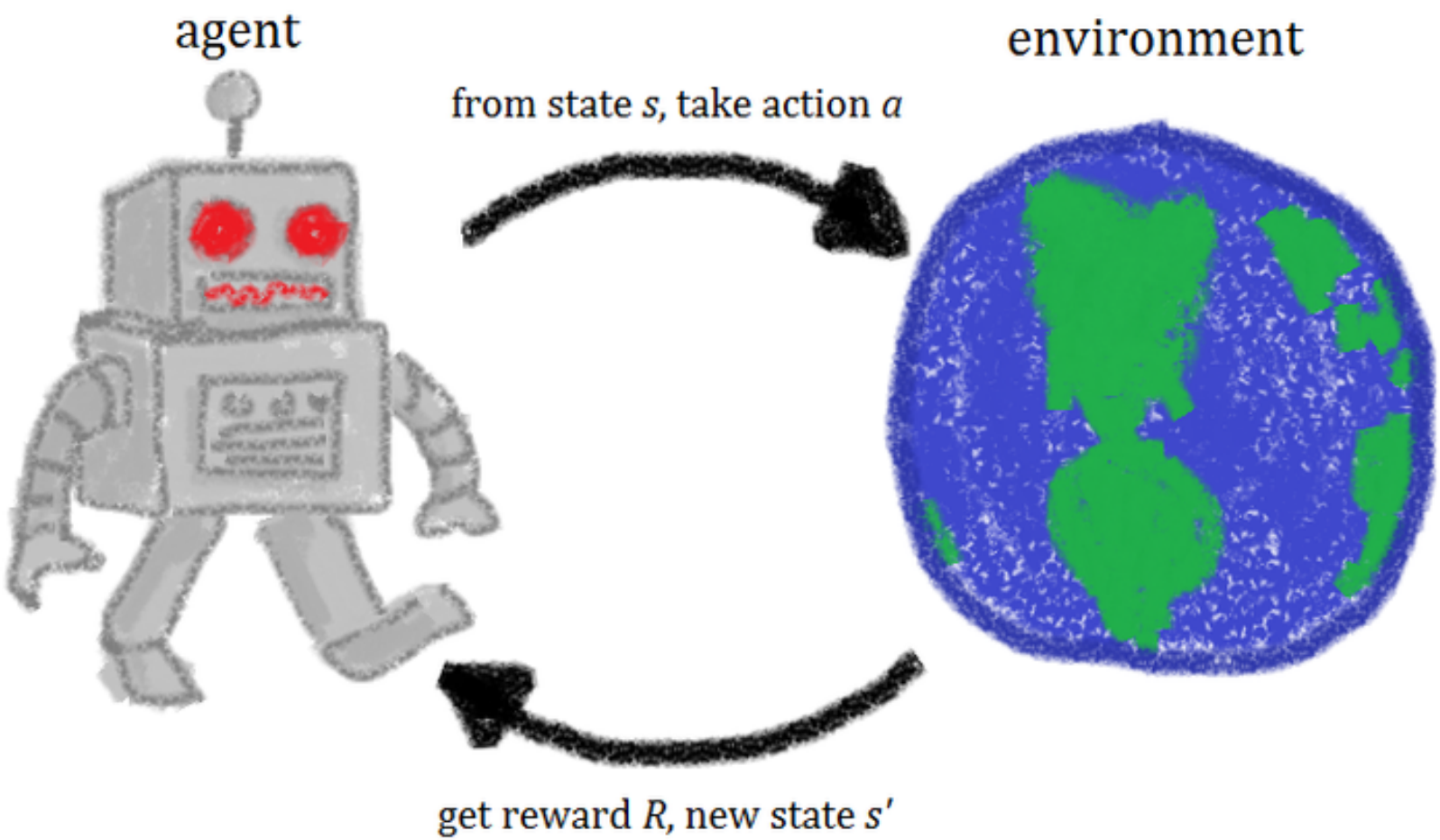
Supervised Learning

Newell, Alejandro, Kaiyu Yang, and Jia Deng. "Stacked hourglass networks for human pose estimation." ECCV'16. Springer, 2016.

Unsupervised Learning

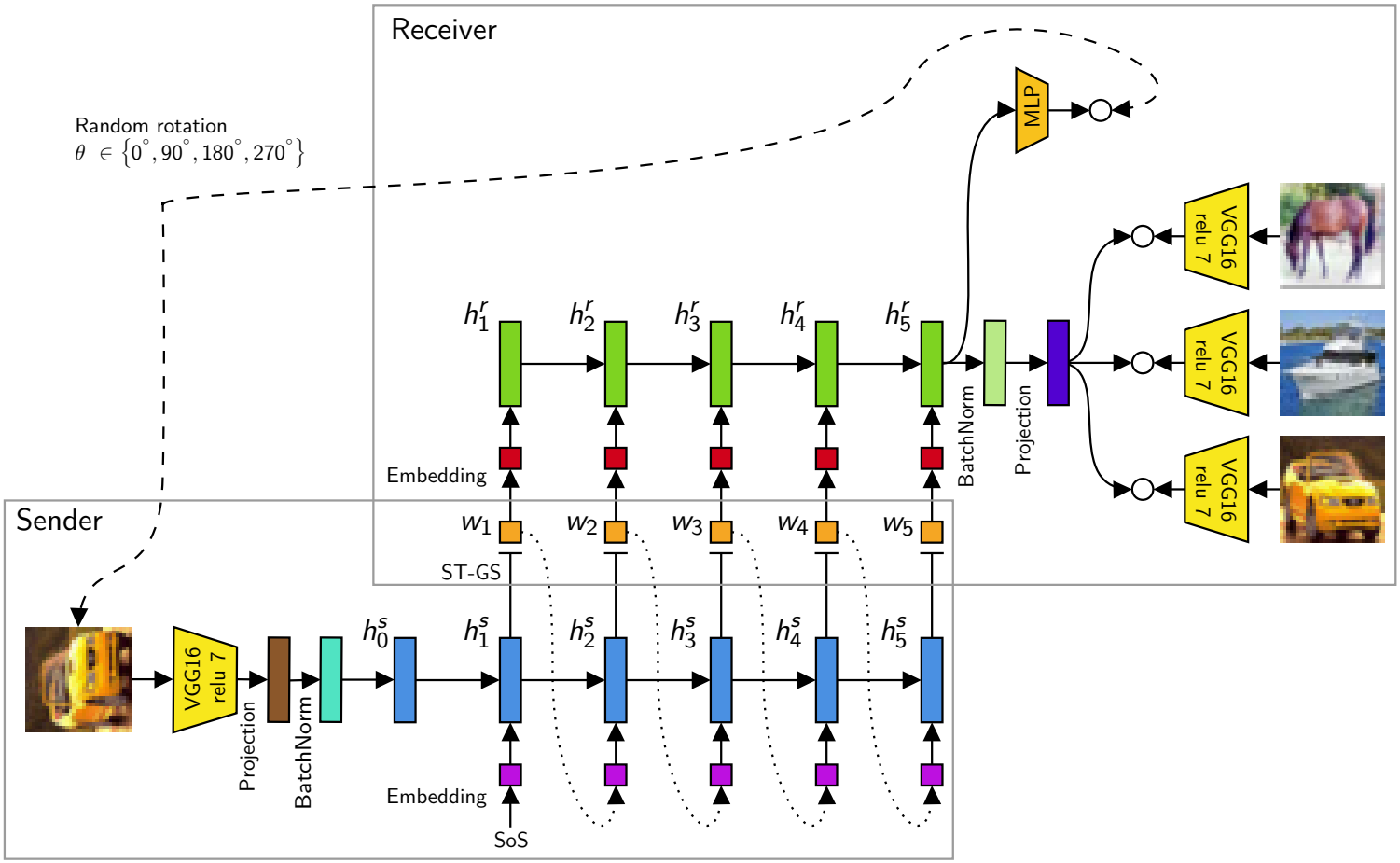


Reinforcement Learning



Reference: Wikipedia
`https://simple.wikipedia.org/wiki/Reinforcement_learning`

Self-supervised Learning



Daniela Mihai and Jonathon Hare. Avoiding hashing and encouraging visual semantics in referential emergent language games. EmeCom @ NeurIPS 2019. https://arxiv.org/abs/1911.05546

Semi-supervised Learning

1. Start with unlabelled image data.

2. Use deep learning to automatically find structure in image.

3. Human adds labels to different clusters successfully classified, and distinguishes between images classified incorrectly.

4. Repeat stages 2&3 (re-train classifier using this additional info.

Jeremy Howard. The wonderful and terrifying implications of computers that can learn. TEDxBrussels. `http://www.ted.com/talks/jeremy_howard_the_wonderful_and_`

Generative Models

- Many unsupervised and self-supervised models can be classed as 'Generative Models'.
- Given unlabelled data X, a unsupervised generative model learns $P[X]$.
  - Could be direct modelling of the data (e.g. Gaussian Mixture Models)
  - Could be indirect modelling by learning to map the data to a parametric distribution in a lower dimensional space (e.g. a VAEs Encoder) or by learning a mapping from a parameterised distribution to the real data space (e.g. a VAE Decoder or GAN)
- These are characterised by an ability to 'sample' the model to 'create' new data

Generative vs. Discriminative Models (II)

Generative vs. discriminative approaches to classification use different statistical modelling.

- Discriminative models learn the boundary between classes. A discriminative models is a model of the conditional probability of the target Y given an observation X: $P[Y|X]$.
- Generative models of labelled data model the distribution of individual classes. Given an observable variable X and a target variable Y, a generative model is a statistical model that tries to model $P[X|Y]$ and $P[Y]$ in order to model the joint probability distribution $P[X, Y]$.[4]

[4] Some such models can be sampled conditionally based on a prior Y - e.g. a Conditional VAE: `https://papers.nips.cc/paper/5775-learning-structured-output-representation-using-deep-conditional-gene`

Two Types of Supervised Learning

- Classification: The machine is asked to specify which of k categories some input belongs to.
  - Multiclass classification - target is one of the k classes
  - Multilabel classification - target is some number of the k classes
  - In both cases, the machine is a function $f : \mathbb{R}^n \to \{1, ..., k\}$ (although it is most common for the learning algorithm to actually learn $\hat{f} : \mathbb{R}^n \to \mathbb{R}^k$).

Two Types of Supervised Learning

- Classification: The machine is asked to specify which of k categories some input belongs to.
  - Multiclass classification - target is one of the k classes
  - Multilabel classification - target is some number of the k classes
  - In both cases, the machine is a function $f : \mathbb{R}^n \to \{1, ..., k\}$ (although it is most common for the learning algorithm to actually learn $\hat{f} : \mathbb{R}^n \to \mathbb{R}^k$).
- Regression: The machine is asked predict k numerical values given some input. The machine is a function $f : \mathbb{R}^n \to \mathbb{R}^k$.

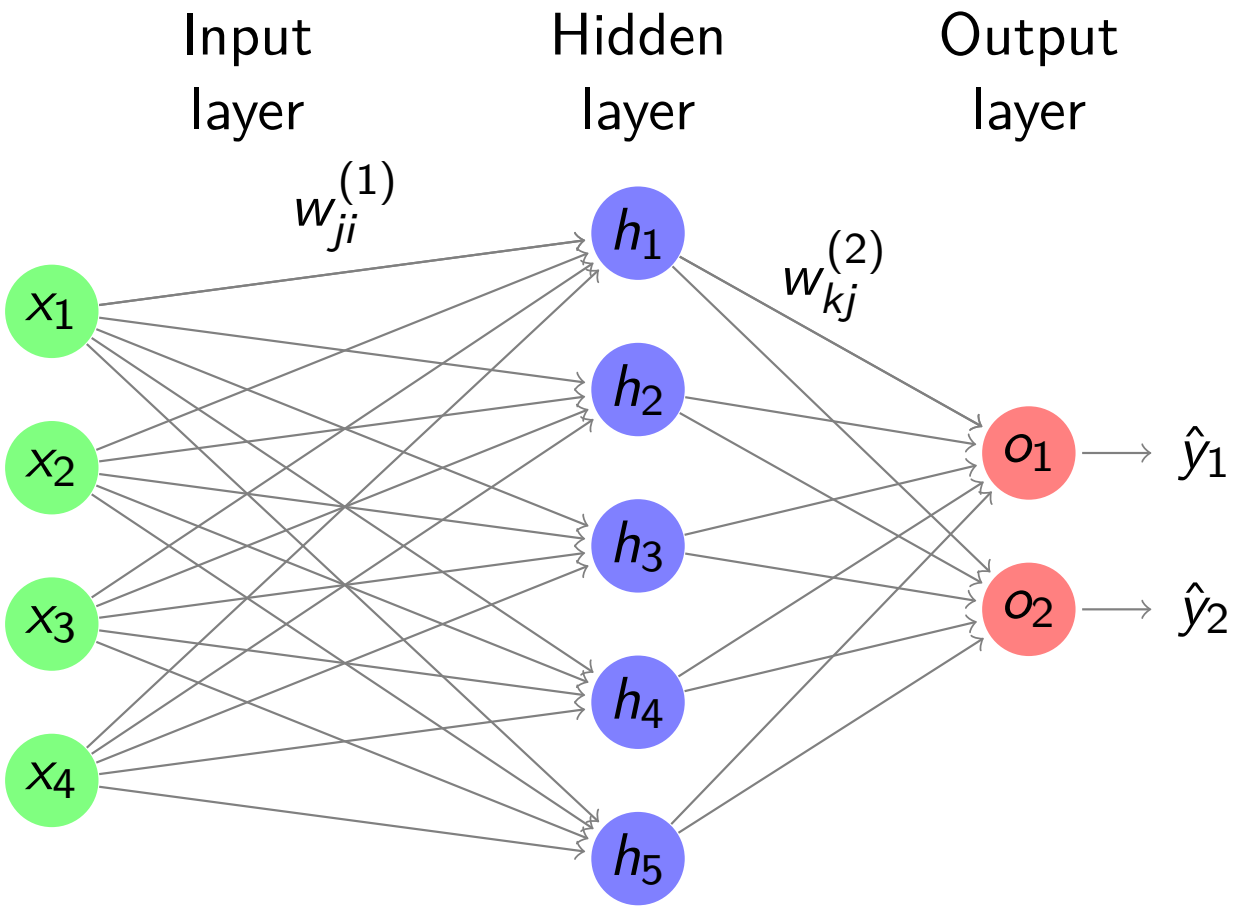
Two Types of Supervised Learning

- Classification: The machine is asked to specify which of k categories some input belongs to.
  - Multiclass classification - target is one of the k classes
  - Multilabel classification - target is some number of the k classes
  - In both cases, the machine is a function $f : \mathbb{R}^n \to \{1, ..., k\}$ (although it is most common for the learning algorithm to actually learn $\hat{f} : \mathbb{R}^n \to \mathbb{R}^k$).
- Regression: The machine is asked predict k numerical values given some input. The machine is a function $f : \mathbb{R}^n \to \mathbb{R}^k$.
- Note that there are lots of exceptions in the form the inputs (and outputs) can take though! We'll see lots of variations in the coming weeks.

How Supervised Learning Typically Works

- Start by choosing a model-class: $\hat{y} = f(\boldsymbol{x}; \boldsymbol{W})$ where the model-class f is a way of using some numerical parameters, $\boldsymbol{W}$, to map each input vector $\boldsymbol{x}$ to a predicted output $\hat{y}$.
- Learning means adjusting the parameters to reduce the discrepancy between the true target output y on each training case and the output $\hat{y}$, predicted by the model.

Let's look at an unbiased Multilayer Perceptron...



Without loss of generality, we can write the above as:

$$\hat{\boldsymbol{y}} = g(f(\boldsymbol{x}; \boldsymbol{W}^{(1)}); \boldsymbol{W}^{(2)}) = g(\boldsymbol{W}^{(2)} f(\boldsymbol{W}^{(1)} \boldsymbol{x}))$$

where f and g are activation functions.

Common Activation Functions

- Identity
- Sigmoid (aka Logistic)
- Hyperbolic Tangent (tanh)
- Rectified Linear Unit (ReLU) (aka Threshold Linear)

Final layer activations

$$\hat{y} = g(\boldsymbol{W}^{(2)} f(\boldsymbol{W}^{(1)} \boldsymbol{x}))$$

- What form should the final layer function g take?

Final layer activations

$$\hat{y} = g(\boldsymbol{W}^{(2)} f(\boldsymbol{W}^{(1)} \boldsymbol{x}))$$

- What form should the final layer function g take?
- It depends on the task (and on the chosen loss function)...
  - For regression it is typically linear (e.g. identity), but you might choose others if you say wanted to clamp the range of the network.
  - For binary classification (MLP has a single output), one would choose Sigmoid
  - For multilabel classification, typically one would choose Sigmoid
  - For multiclass classification, typically you would use the Softmax function

Softmax

The softmax is an activation function used at the output layer of a neural network that forces the outputs to sum to 1 so that they can represent a probability distribution across a discrete mutually exclusive alternatives.

$$\text{softmax}(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \qquad \forall i = 1, 2, \ldots, K$$

- Note that unlike the other activation functions you've seen, softmax makes reference to all the elements in the output.
- The output of a softmax layer is a set of positive numbers which sum up to 1 and can be thought of as a probability distribution.
- Note:

$$\begin{aligned}
\frac{\partial \,\text{softmax}(\boldsymbol{z})_i}{\partial z_i} &= \text{softmax}(z_i)(1 - \text{softmax}(z_i)) \\
\frac{\partial \,\text{softmax}(\boldsymbol{z})_i}{\partial z_j} &= \text{softmax}(z_i)(1(i = j) - \text{softmax}(z_j)) \\
&= \text{softmax}(z_i)(\delta_{ij} - \text{softmax}(z_j))
\end{aligned}$$

Ok, so let's talk loss functions

- The choice of loss function depends on the task (e.g. classification/regression/something else)

Ok, so let's talk loss functions

- The choice of loss function depends on the task (e.g. classification/regression/something else)
- The choice also depends on the activation function of the last layer

Ok, so let's talk loss functions

- The choice of loss function depends on the task (e.g. classification/regression/something else)
- The choice also depends on the activation function of the last layer
  - For numerical reasons (see Log-Sum-Exp in a few slides) many times the activation is computed directly within the loss rather than being part of the model

Ok, so let's talk loss functions

- The choice of loss function depends on the task (e.g. classification/regression/something else)
- The choice also depends on the activation function of the last layer
  - For numerical reasons (see Log-Sum-Exp in a few slides) many times the activation is computed directly within the loss rather than being part of the model
  - Some classification losses require *raw outputs* (e.g. a linear layer) of the network as their input
    - These are often called *unnormalised log probabilities* or *logits*
    - An example would be hinge-loss used to create a Support Vector Machine that maximises the margin — e.g.: $\ell_{hinge}(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$ with a true label, $y \in \{-1, 1\}$, for binary classification.

Ok, so let's talk loss functions

- The choice of loss function depends on the task (e.g. classification/regression/something else)
- The choice also depends on the activation function of the last layer
  - For numerical reasons (see Log-Sum-Exp in a few slides) many times the activation is computed directly within the loss rather than being part of the model
  - Some classification losses require *raw outputs* (e.g. a linear layer) of the network as their input
    - These are often called *unnormalised log probabilities* or *logits*
    - An example would be hinge-loss used to create a Support Vector Machine that maximises the margin — e.g.: $\ell_{hinge}(\hat{y}, y) = \max(0, 1 - y \cdot \hat{y})$ with a true label, $y \in \{-1, 1\}$, for binary classification.
- There are many different loss functions we might encounter (MSE, Cross-Entropy, KL-Divergence, huber, L1 (MAE), CTC, Triplet, ...) for different tasks.

The Cost Function (measure of discrepancy)

Recall:

- Mean Squared Error (MSE) loss for a single data point (here assumed to be a vector, but equally applicable to a scalar) is given by $\ell_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_i (\hat{y}_i - y_i)^2 = (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top (\hat{\boldsymbol{y}} - \boldsymbol{y})$
- We often multiply this by a constant factor of $\frac{1}{2}$ — can anyone guess/remember why?

[5] http://neuralnetworksanddeeplearning.com/chap3.html

The Cost Function (measure of discrepancy)

Recall:

- Mean Squared Error (MSE) loss for a single data point (here assumed to be a vector, but equally applicable to a scalar) is given by $\ell_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_i (\hat{y}_i - y_i)^2 = (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top (\hat{\boldsymbol{y}} - \boldsymbol{y})$
- We often multiply this by a constant factor of $\frac{1}{2}$ — can anyone guess/remember why?
- $\ell_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ is the predominant choice for regression problems with linear activation in the last layer

[5] http://neuralnetworksanddeeplearning.com/chap3.html

The Cost Function (measure of discrepancy)

Recall:

- Mean Squared Error (MSE) loss for a single data point (here assumed to be a vector, but equally applicable to a scalar) is given by $\ell_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \sum_i (\hat{y}_i - y_i)^2 = (\hat{\boldsymbol{y}} - \boldsymbol{y})^\top (\hat{\boldsymbol{y}} - \boldsymbol{y})$
- We often multiply this by a constant factor of $\frac{1}{2}$ — can anyone guess/remember why?
- $\ell_{MSE}(\hat{\boldsymbol{y}}, \boldsymbol{y})$ is the predominant choice for regression problems with linear activation in the last layer
- For a classification problem with Softmax or Sigmoidal (or really anything non-linear) activations, MSE can cause slow learning, especially if the predictions are very far off the targets
  - Gradients of ℓ_{MSE} are proportional to the difference in target and predicted multiplied by the gradient of the activation function[5]
  - The Cross-Entropy loss function is generally a better choice in this case

[5] http://neuralnetworksanddeeplearning.com/chap3.html

Binary Cross-Entropy

For the binary classification case:

$$\ell_{BCE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

- The cross-entropy cost function is non-negative, $\ell_{BCE} > 0$
- $\ell_{BCE} \approx 0$ when the prediction and targets are equal (i.e. $y = 0$ and $\hat{y} = 0$ or when $y = 1$ and $\hat{y} = 1$)
- With Sigmoidal final layer, $\frac{\partial \ell_{BCE}}{\partial \boldsymbol{W}_i^{(2)}}$ is proportional to just the error in the output $(\hat{y} - y)$ and therefore, the larger the error, the faster the network will learn!
- Note that the BCE is the negative log likelihood of the Bernoulli Distribution

Binary Cross-Entropy — Intuition

- The cross-entropy can be thought of as a **measure of surprise**.
- Given some input x_i, we can think of $\hat{y}_i$ as the estimated probability that x_i belongs to class 1, and $1 - \hat{y}_i$ is the estimated probability that it belongs to class 0.
- Note the extreme case of infinite cross-entropy, if your model believes that a class has 0 probability of occurrence, and yet the class appears in the data, the 'surprise' of your model will be infinitely great.

Binary Cross-Entropy for multiple labels

In the case of multi-label classification with a network with multiple sigmoidal outputs you just sum the BCE over the outputs:

$$\ell_{BCE} = -\sum_{k=1}^{K}[y_k \log(\hat{y}_k) + (1 - y_k) \log(1 - \hat{y}_k)]$$

where K is the number of classes of the classification problem, $\hat{y} \in \mathbb{R}^K$.

Numerical Stability: The Log-Sum-Exp trick

$$\ell_{BCE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

- Consider what might happen early in training when the model might confidently predict a positive example as negative
  - $\hat{y} = \sigma(z) \approx 0 \implies z << 0$

[6] https://www.xarg.org/2016/06/the-log-sum-exp-trick-in-machine-learning/

Numerical Stability: The Log-Sum-Exp trick

$$\ell_{BCE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

- Consider what might happen early in training when the model might confidently predict a positive example as negative
  - $\hat{y} = \sigma(z) \approx 0 \implies z << 0$
  - if $\hat{y}$ is small enough, it will become 0 due to limited precision of floating-point representations

[6] https://www.xarg.org/2016/06/the-log-sum-exp-trick-in-machine-learning/

Numerical Stability: The Log-Sum-Exp trick

$$\ell_{BCE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

- Consider what might happen early in training when the model might confidently predict a positive example as negative
  - $\hat{y} = \sigma(z) \approx 0 \implies z << 0$
  - if $\hat{y}$ is small enough, it will become 0 due to limited precision of floating-point representations
  - but then $\log(\hat{y}) = -\inf$, and everything will break!

[6] https://www.xarg.org/2016/06/the-log-sum-exp-trick-in-machine-learning/

Numerical Stability: The Log-Sum-Exp trick

$$\ell_{BCE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

- Consider what might happen early in training when the model might confidently predict a positive example as negative
  - $\hat{y} = \sigma(z) \approx 0 \implies z << 0$
  - if $\hat{y}$ is small enough, it will become 0 due to limited precision of floating-point representations
  - but then $\log(\hat{y}) = -\inf$, and everything will break!
- To tackle this problem implementations usually combine the sigmoid computation and BCE into a single loss function that you would apply to a network with linear outputs (e.g. `BCEWithLogitsLoss`).

[6] https://www.xarg.org/2016/06/the-log-sum-exp-trick-in-machine-learning/

Numerical Stability: The Log-Sum-Exp trick

$$\ell_{BCE}(\hat{y}, y) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y})$$

- Consider what might happen early in training when the model might confidently predict a positive example as negative
  - $\hat{y} = \sigma(z) \approx 0 \implies z << 0$
  - if $\hat{y}$ is small enough, it will become 0 due to limited precision of floating-point representations
  - but then $\log(\hat{y}) = -\inf$, and everything will break!
- To tackle this problem implementations usually combine the sigmoid computation and BCE into a single loss function that you would apply to a network with linear outputs (e.g. `BCEWithLogitsLoss`).
- Internally, a trick called 'log-sum-exp' is used to *shift* the centre of an exponential sum so that only numerical underflow can potentially happen, rather than overflow[6].
  - Ultimately this means you'll always get a numerically reasonable result (and will avoid NaNs and Infs originating from this point).

[6] https://www.xarg.org/2016/06/the-log-sum-exp-trick-in-machine-learning/

Multiclass classification with Softmax Outputs

- Softmax can be thought of making the K outputs of the network mimic a probability distribution.

[7] Note: Keras calls this function 'Categorical Cross-Entropy'; you would need to have a Softmax output layer to use this

Multiclass classification with Softmax Outputs

- Softmax can be thought of making the K outputs of the network mimic a probability distribution.
- The target label y could also be represented as a distribution with a single 1 and zeros everywhere else.
  - e.g. they are "one-hot encoded".

[7] Note: Keras calls this function 'Categorical Cross-Entropy'; you would need to have a Softmax output layer to use this

Multiclass classification with Softmax Outputs

- Softmax can be thought of making the K outputs of the network mimic a probability distribution.
- The target label y could also be represented as a distribution with a single 1 and zeros everywhere else.
  - e.g. they are "one-hot encoded".
- In such a case, the obvious loss function is the *negative log likelihood* of the Categorical distribution (aka Multinoulli, Generalised Bernoulli, Multinomial with one sample)[7]: $\ell_{NNL} = -\sum_{k=1}^{K} y_k \log \hat{y}_k$
  - Note that in practice as y_k is zero for all but one class you don't actually do this summation, and if y is an integer class index you can write $\ell_{NNL} = -\log \hat{y}_y$.

[7] Note: Keras calls this function 'Categorical Cross-Entropy'; you would need to have a Softmax output layer to use this

Multiclass classification with Softmax Outputs

- Softmax can be thought of making the K outputs of the network mimic a probability distribution.
- The target label y could also be represented as a distribution with a single 1 and zeros everywhere else.
  - e.g. they are "one-hot encoded".
- In such a case, the obvious loss function is the *negative log likelihood* of the Categorical distribution (aka Multinoulli, Generalised Bernoulli, Multinomial with one sample)[7]: $\ell_{NNL} = -\sum_{k=1}^{K} y_k \log \hat{y}_k$
  - Note that in practice as y_k is zero for all but one class you don't actually do this summation, and if y is an integer class index you can write $\ell_{NNL} = -\log \hat{y}_y$.
- Analogously to what we saw for BCE, Log-Sum-Exp can be used for better numerical stability.
  - PyTorch combines LogSoftmax with NNL in one loss and calls this "Categorical Cross-Entropy" (so you would use this with a *linear output layer*)

[7] Note: Keras calls this function 'Categorical Cross-Entropy'; you would need to have a Softmax output layer to use this

Reminder: Gradient Descent

- Define total loss as $\mathcal{L} = -\sum_{(\boldsymbol{x},y)\in\boldsymbol{D}} \ell(g(\boldsymbol{x},\boldsymbol{\theta}),y)$ for some loss function ℓ, dataset $\boldsymbol{D}$ and model g with learnable parameters $\boldsymbol{\theta}$.
- Define how many passes over the data to make (each one known as an Epoch)
- Define a learning rate η

Gradient Descent updates the parameters $\boldsymbol{\theta}$ by moving them in the direction of the negative gradient with respect to the **total loss** $\mathcal{L}$ by the learning rate η multiplied by the gradient:

```
for each Epoch:
```
$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta\nabla_{\boldsymbol{\theta}}\mathcal{L}$$

Reminder: Stochastic Gradient Descent

- Define loss function ℓ, dataset $\boldsymbol{D}$ and model g with learnable parameters $\boldsymbol{\theta}$.
- Define how many passes over the data to make (each one known as an Epoch)
- Define a learning rate η

Stochastic Gradient Descent updates the parameters $\boldsymbol{\theta}$ by moving them in the direction of the negative gradient with respect to the loss of a **single item** ℓ by the learning rate η multiplied by the gradient:

```
for each Epoch:
    for each (x, y) ∈ D:
        θ ← θ − η∇_θ ℓ
```