We are starting at 14:00!

Grab a seat and get ready

Saturdays.AI
Kigali

Saturdays.AI
Kigali

#3 MultiLayer Perceptron

AI Saturdays Kigali

Agenda

14:00 - 14:45: The computational benefits of nonlinearity

14:45 - 15:30: Building Multi Layer Perceptrons in PyTorch

15:00 - 16:00: MLPs for classification

16:00 - 16:30: Break

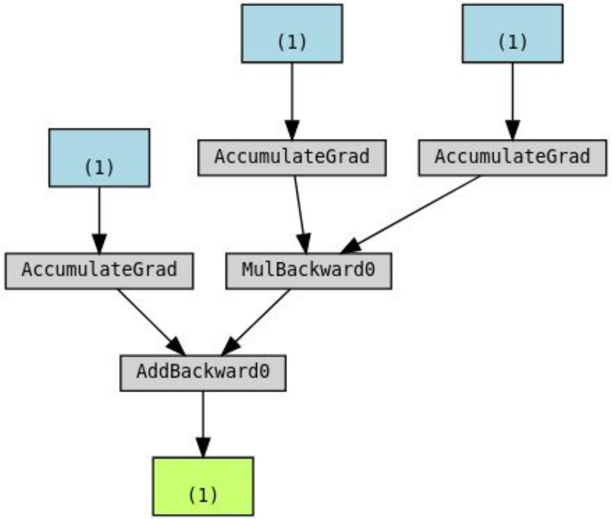
16:30 - 17:00: Putting it together

17:30 - 18:00: Challenges & Next steps

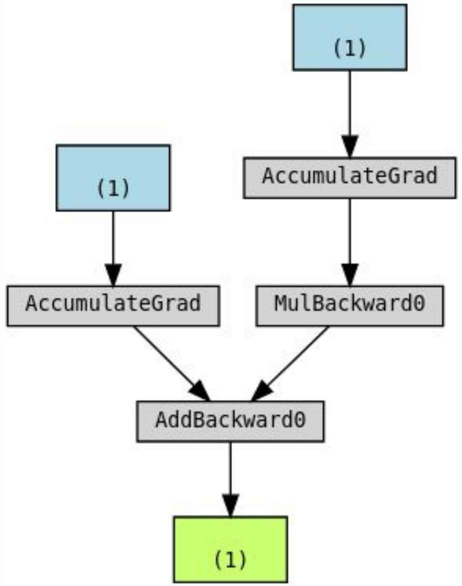
TYU

If b and w are trainable parameters and x is a feature vector, which computation graph best represents the operation yhat = b + w * x?

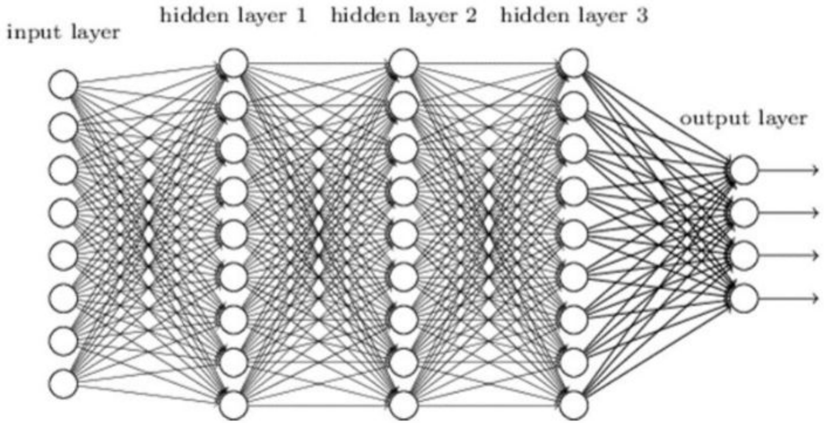
A)

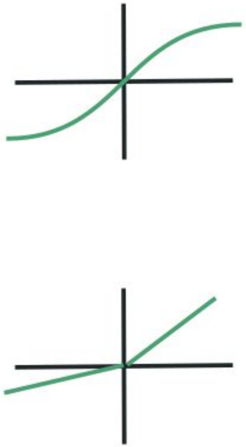


B)



Our first computationally powerful deep net: a multilayer perceptron (MLP)



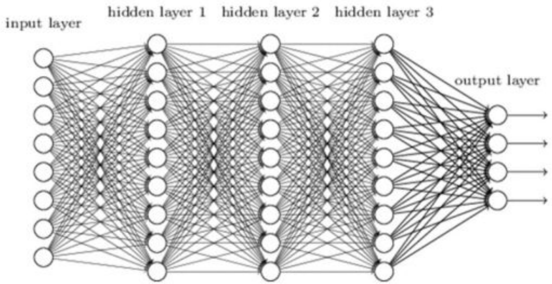


Sigmoid

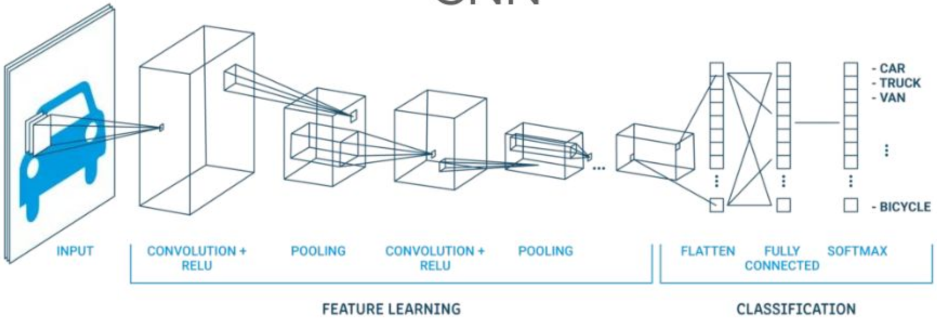
Leaky ReLU

MLPs are a basis for CNNs and RNNs

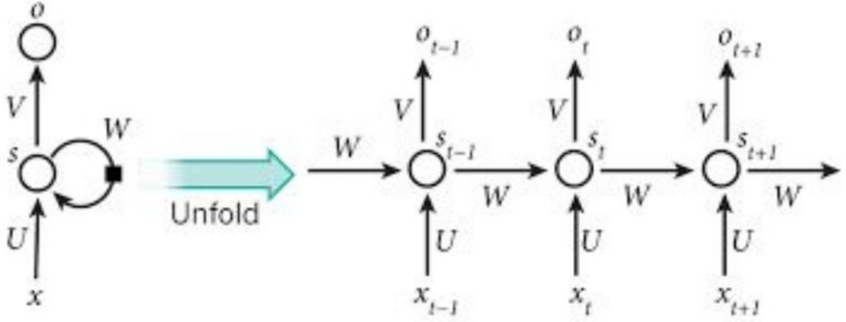
MLP



CNN



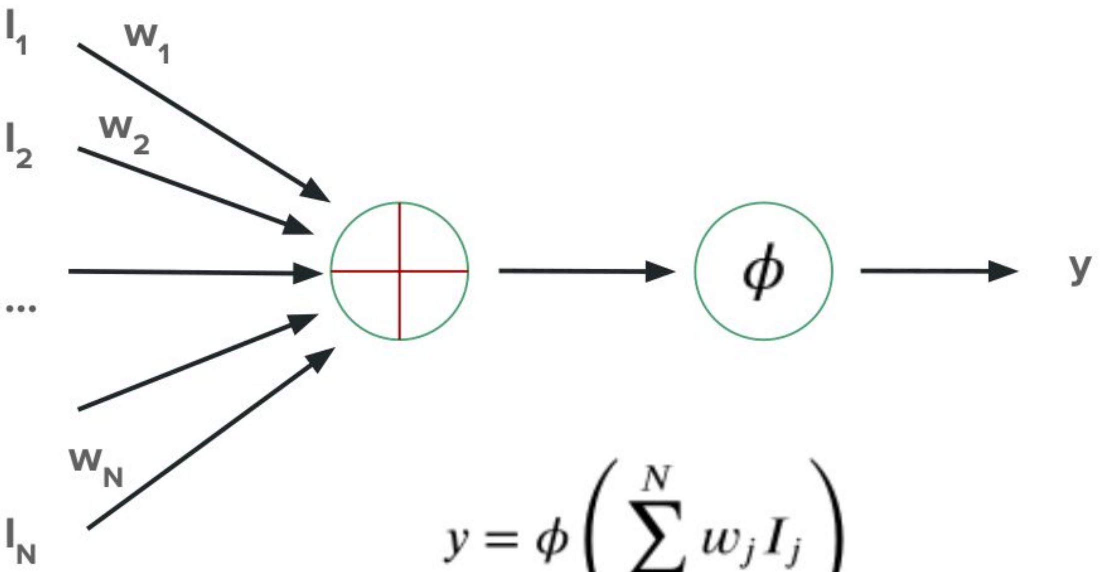
RNN



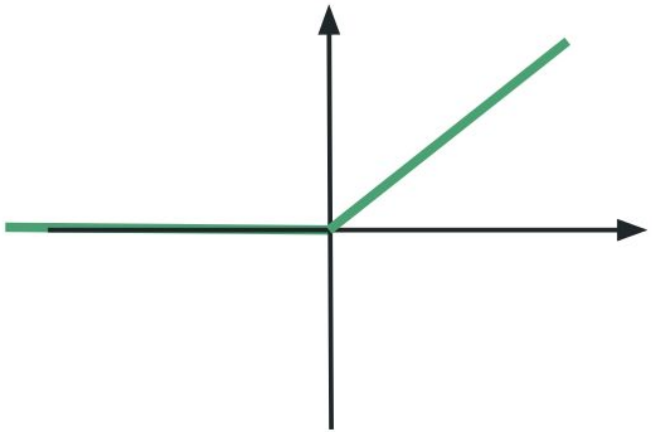
The computational benefits of nonlinearity

What can even a shallow nonlinear network with 1 hidden layer do that a linear network cannot?

The rectified linear unit (ReLU) in AI



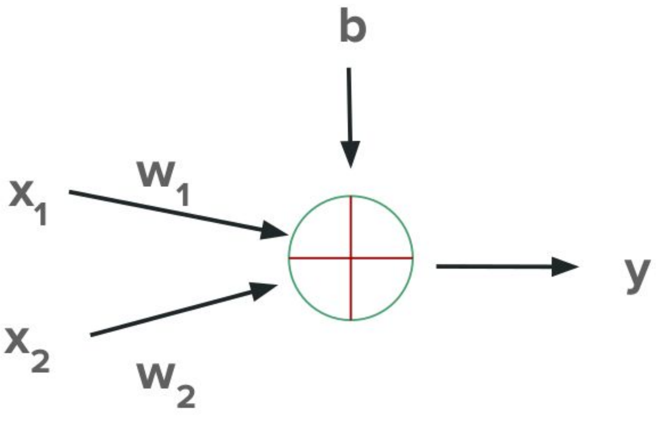
$$y = \phi\left(\sum_{j=1}^{N} w_j I_j\right)$$



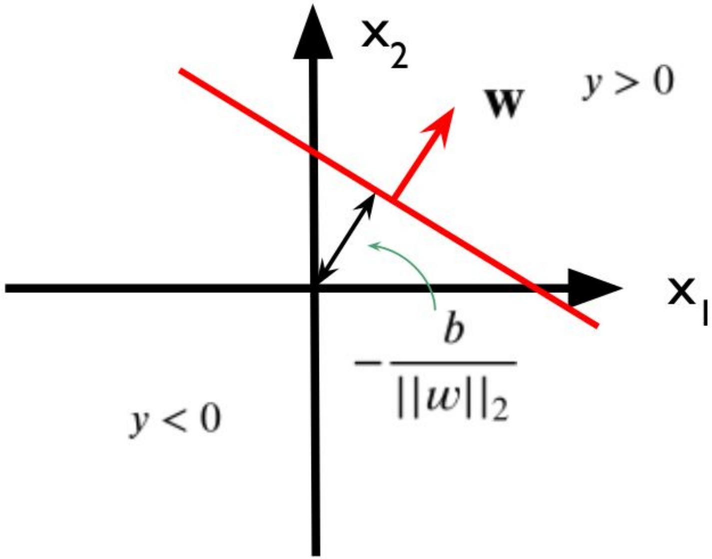
$$\phi(x) = \max(0, x)$$

Saturdays.AI
Kigali

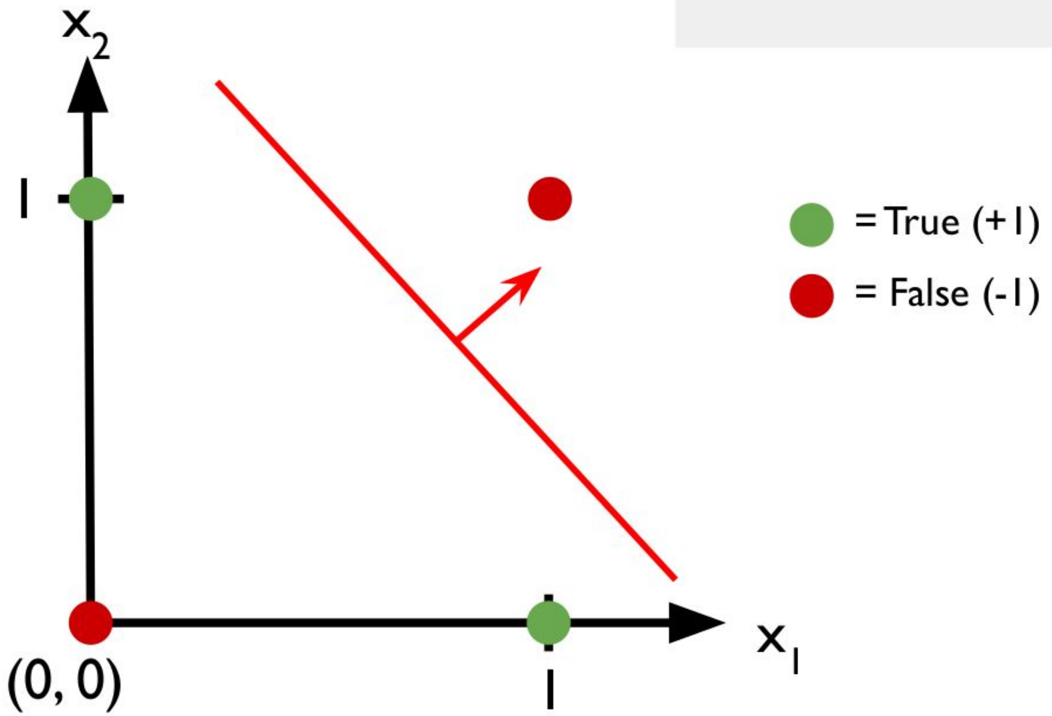
A single linear neuron can only construct linear functions and decision boundaries



$$y = \mathbf{w} \cdot \mathbf{x} + b$$

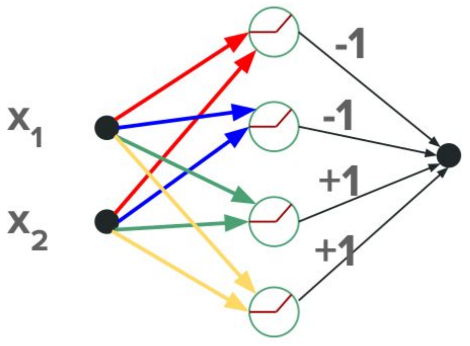


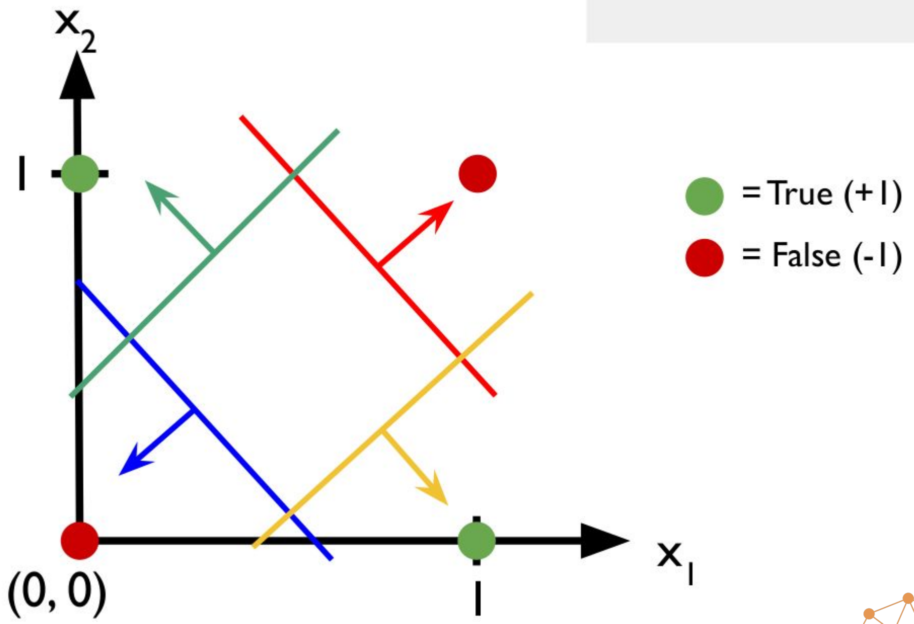
A single linear neuron can't solve XOR



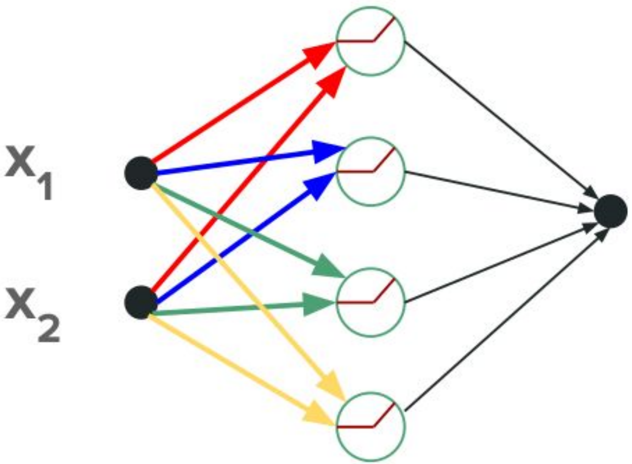
Saturdays.AI
Kigali

But a 1 hidden layer MLP can!





Ok... so what else can a 1 hidden layer MLP solve?



Answer: almost anything!*

(* If you give it enough hidden neurons)

Universal function approximation theorem

Let $\phi : \mathbb{R} \to \mathbb{R}$ be a continuous activation function (that is not a simple polynomial)

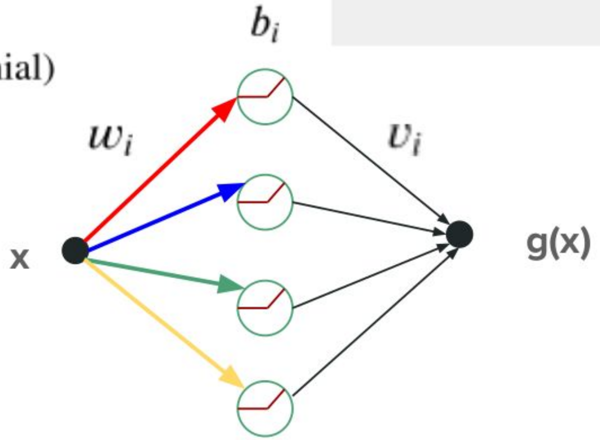
Let $f : \mathbb{R} \to \mathbb{R}$ be a continuous target function.

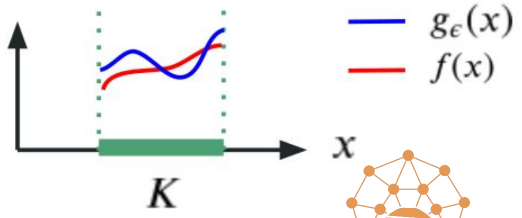
Let $g(x) = \sum_{i=1}^{N} v_i \phi(w_i x + b_i)$ be a family of neural network functions.

For every compact subset $K \in \mathbb{R}$ and for every error tolerance level ϵ

there exists an integer N and a set of parameters $\{v_i, w_i, b_i\}_{i=1}^{N}$

such that the corresponding function $g_\epsilon(x)$ obeys $\sup_{x \in K} |f(x) - g_\epsilon(x)| < \epsilon$



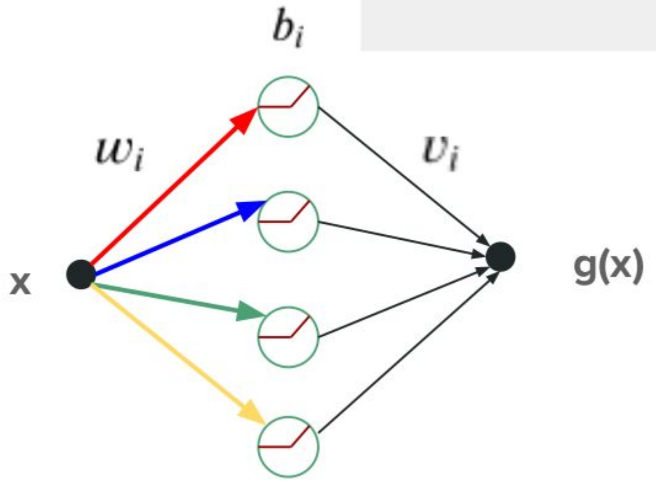


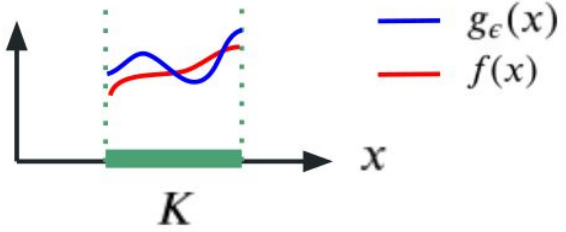
Okay… so why do we need deep nets with more than one hidden layer?

While the universal approximation theorem says we can approximate a function to some accuracy with a one hidden layer neural network,

It does not tell us how many hidden neurons we will need:

there exists an integer N and a set of parameters $\{v_i, w_i, b_i\}_{i=1}^{N}$



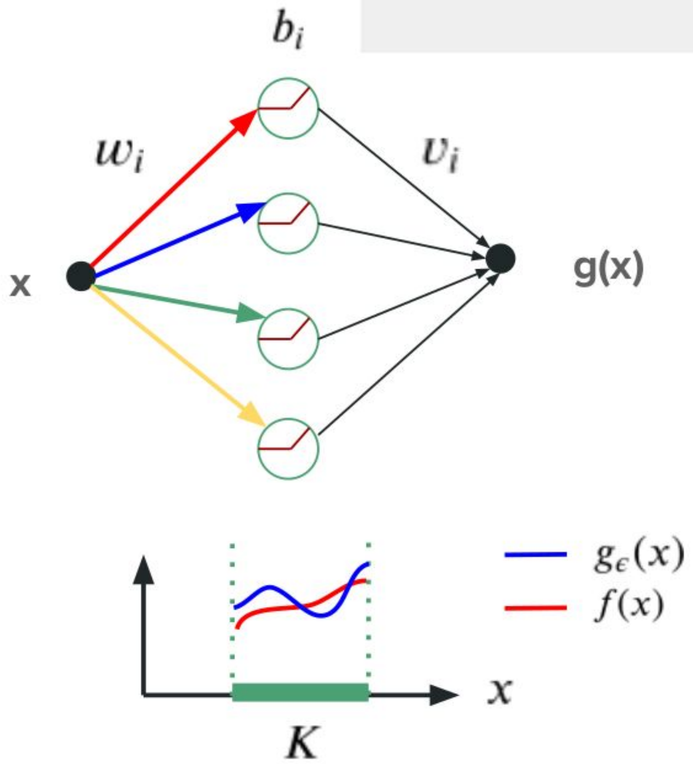


The phenomenon of deep expressivity

For example, there exist some functions which can be efficiently approximated by a deep network.

But to approximate these same functions with a 1 hidden layer network would require exponentially many more neurons.

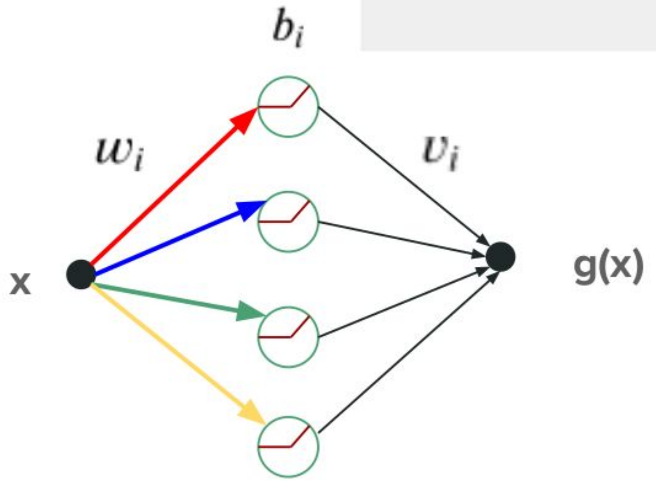
We will explore this in a later tutorial.



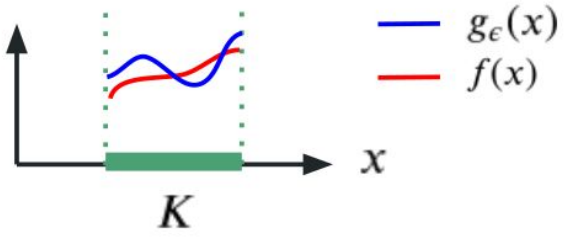
What else the universal approximation theorem not tell us: how to learn.

While there may exist a network that approximates our function

There is no guarantee we can find this function given a finite set of example input output pairs.



there exists an integer N and a set of parameters $\{v_i, w_i, b_i\}_{i=1}^{N}$



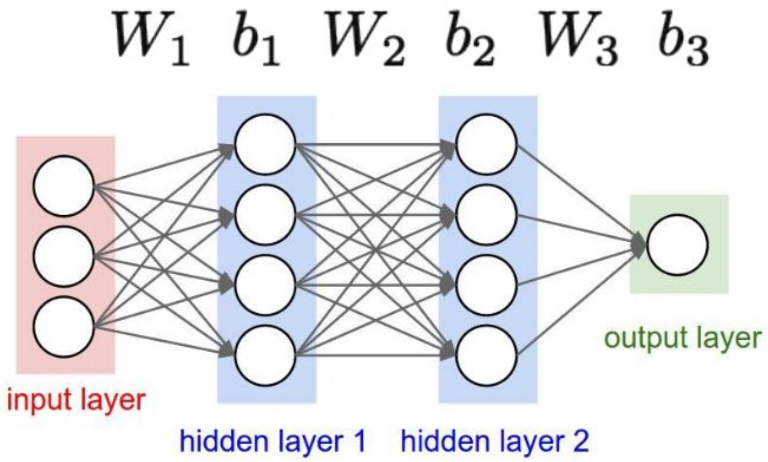
Building Multi Layer Perceptrons in PyTorch

Saturdays.AI
Kigali

Multi-layer perceptrons (MLPs)

$$y(x) \approx N(x) = W_k\sigma(\cdots\sigma(W_2\sigma(W_1x + b_1) + b_2)\cdots + b_{k-1}) + b_k$$

- **Layer**: one of the intermediate vectors
- **Neuron**: one of the entries of a layer vector
- **Depth**: the number of layers
- **Width**: the layer's dimension
- **Weights**: coefficients of the matrice Wk
- **Biases** coefficients of the vector bk
- The **activation function** or **non-linearity** is the function σ.



Saturdays.AI
Kigali

So let us create a general MLP

Input/Output behaviour:

- We tell it sizes of input, each hidden, and output layers
- And which activation function to use for hidden layers
- Then it will construct an MLP with no output activation since it's general!

PyTorch Design

```
class Net(nn.Module):
```

Create a model class called "Net" which subclasses nn.Module, the base class for all neural network modules.

nn.Module takes care of backprop for you so you don't need to define a backward() function!

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
```

Define the initialization inputs of the model class

Saturdays.AI
Kigali

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
```

```
nn.ELU, nn.Hardshrink, nn.Hardsigmoid,
nn.Hardtanh, nn.Hardswish, nn.LeakyReLU,
nn.LogSigmoid, nn.MultiheadAttention,
nn.PReLU, nn.ReLU, nn.ReLU6, nn.RReLU,
nn.SELU, nn.CELU, nn.GELU, nn.Sigmoid,
nn.SiLU, nn.Mish, nn.Softplus, nn.Softshrink,
nn.Softsign, nn.Tanh, nn.Tanhshrink,
nn.Threshold
```

"actv" is the string of the activation function with arguments which exists in torch.nn, e.g., "LeakyReLU(0.1)"

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
```

Input layer size,
E.g., for an RGB image of 32x32 it is 32x32x3=3072

Saturdays.AI
Kigali

PyTorch Design

```python
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
```

List of hidden layer sizes,
E.g., for 3 hidden layers,
[256, 128, 64]

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
```

Output layer size,
E.g., for a 3 way classification
it would be 3

Saturdays.AI
Kigali

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
```

Calls the init function of its base class (nn.Module)

Saturdays.AI
Kigali

PyTorch Design

```python
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
```

Save the input size for later use in forward(), since we will reshape inputs using this

Saturdays.AI
Kigali

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
        self.mlp = nn.Sequential()
```

Initialize another subclass of nn.Module with the functionality to run the given modules in sequence (we'll give the modules next)

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
        self.mlp = nn.Sequential()

        in_num = input_feature_num
        for i in range(len(hidden_unit_nums)):
```

Initialize the variable that will determine the **input size** of each layer (nn.Linear module)

Saturdays.AI
Kigali

PyTorch Design

```python
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
        self.mlp = nn.Sequential()

        in_num = input_feature_num
        for i in range(len(hidden_unit_nums)):

          out_num = hidden_unit_nums[i]
          layer = ...
          in_num = out_num
          self.mlp.add_module('Linear_%d'%i, layer)
```

Initialize the variable that will determine the **output size** of each layer which is same as the hidden units in that layer

Saturdays.AI
Kigali

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
        self.mlp = nn.Sequential()

        in_num = input_feature_num
        for i in range(len(hidden_unit_nums)):

          out_num = hidden_unit_nums[i]
          layer = ...
          in_num = out_num
          self.mlp.add_module('Linear_%d'%i, layer)
```

Determine the <u>input size of next layer</u> which is <u>the output size of current layer</u>

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
        self.mlp = nn.Sequential()

        in_num = input_feature_num
        for i in range(len(hidden_unit_nums)):

          out_num = hidden_unit_nums[i]
          layer = ...
          in_num = out_num
          self.mlp.add_module('Linear_%d'%i, layer)
```

Now we can append the module you just constructed with a name to the Sequential module we initialized

Saturdays.AI
Kigali

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
        self.mlp = nn.Sequential()

        in_num = input_feature_num
        for i in range(len(hidden_unit_nums)):

          out_num = hidden_unit_nums[i]
          layer = ...
          in_num = out_num
          self.mlp.add_module('Linear_%d'%i, layer)

          actv_layer = eval('nn.%s'%actv)
          self.mlp.add_module('Activation_%d'%i, actv_layer)
```

We initialize the activation module for that layer and append it similar to before

Saturdays.AI
Kigali

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
        self.mlp = nn.Sequential()

        in_num = input_feature_num
        for i in range(len(hidden_unit_nums)):

          out_num = hidden_unit_nums[i]
          layer = ...
          in_num = out_num
          self.mlp.add_module('Linear_%d'%i, layer)

          actv_layer = eval('nn.%s'%actv)
          self.mlp.add_module('Activation_%d'%i, actv_layer)
```

Caution!
some activation modules have learnable parameters so it is important to initialize them separately for each layer

We initialize the activation module for that layer and append it similar to before

Saturdays.AI
Kigali

PyTorch Design

```
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
        self.mlp = nn.Sequential()

        in_num = input_feature_num
        for i in range(len(hidden_unit_nums)):

          out_num = hidden_unit_nums[i]
          layer = ...
          in_num = out_num
          self.mlp.add_module('Linear_%d'%i, layer)

          actv_layer = eval('nn.%s'%actv)
          self.mlp.add_module('Activation_%d'%i, actv_layer)

        out_layer = nn.Linear(in_num, output_feature_num)
        self.mlp.add_module('Output_Linear', out_layer)
```

Finally, we define and append the output layer separately since it does not have an activation layer and its output size is not in the hidden unit list

Saturdays.AI
Kigali

PyTorch Design

```python
class Net(nn.Module):
    def __init__(self, actv, input_feature_num, hidden_unit_nums, output_feature_num):
        super(Net, self).__init__()
        self.input_feature_num = input_feature_num
        self.mlp = nn.Sequential()

        in_num = input_feature_num
        for i in range(len(hidden_unit_nums)):

          out_num = hidden_unit_nums[i]
          layer = ...
          in_num = out_num
          self.mlp.add_module('Linear_%d'%i, layer)

          actv_layer = eval('nn.%s'%actv)
          self.mlp.add_module('Activation_%d'%i, actv_layer)

        out_layer = nn.Linear(in_num, output_feature_num)
        self.mlp.add_module('Output_Linear', out_layer)
```

Now only left is the forward() function which should be easy since we designed it right ;)

You complete it!

Saturdays.AI Kigali

Practice

Implement a general-purpose MLP in Pytorch

- Design an MLP with any input (1D, 2D, etc.)!

[Coding Exercise 2]

Saturdays.AI
Kigali

MLPs for classification: softmax and cross-entropy

How to classify data by expressing and training probability distributions over a finite set of class labels.

Saturdays.AI
Kigali

Many DL problems involve classification

MNIST: Which one of 10 digits is an input image?

ImageNET: which one of 1000 classes is an input image?

Language models: which word out of a given vocabulary is the most likely next word?

Medical diagnosis: does certain patient medical data signify a diseased state or not?

Classification involves returning a probability distribution over possible label values

MNIST: Probability distribution over 10 digit labels given image.

ImageNET: Probability distribution over 1000 classes given image.

Language models: Probability distribution over next word given previous words.

Medical diagnosis: Probability of disease given medical data.

How to express and train a probability distribution?

Training data:

$\{x_i\}$ A set of inputs (images, medical data, etc...)

$\{l_i\}$ A set of true labels (class, disease state, etc...)

$l_i \in \{1, 2, 3, \ldots, C\}$ C is the total number of classes

How to express and train a probability distribution?

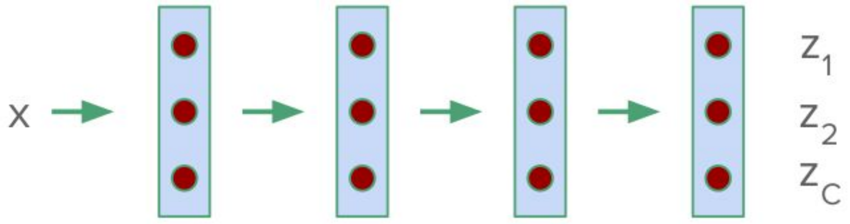
Training data:

$\{x_i\}$ A set of inputs (images, medical data, etc...)

$\{l_i\}$ A set of true labels (class, disease state, etc...)

$l_i \in \{1, 2, 3, \ldots, C\}$ C is the total number of classes

MLP:



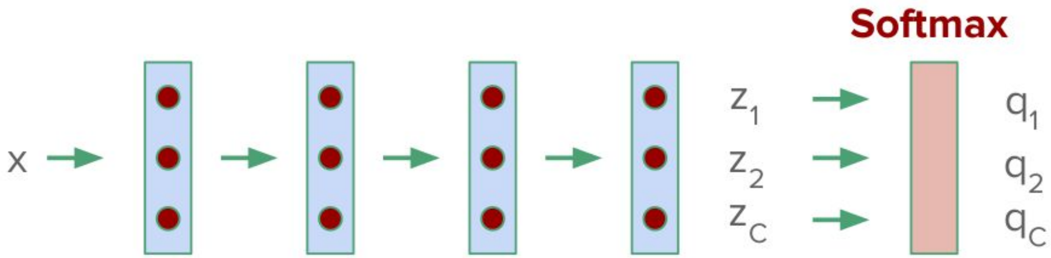
How to express and train a probability distribution?

Training data: $\{x_i\}$ A set of inputs (images, medical data, etc...)

$\{l_i\}$ A set of true labels (class, disease state, etc...)

$l_i \in \{1, 2, 3, \ldots, C\}$ C is the total number of classes

MLP:



How to express and train a probability distribution?

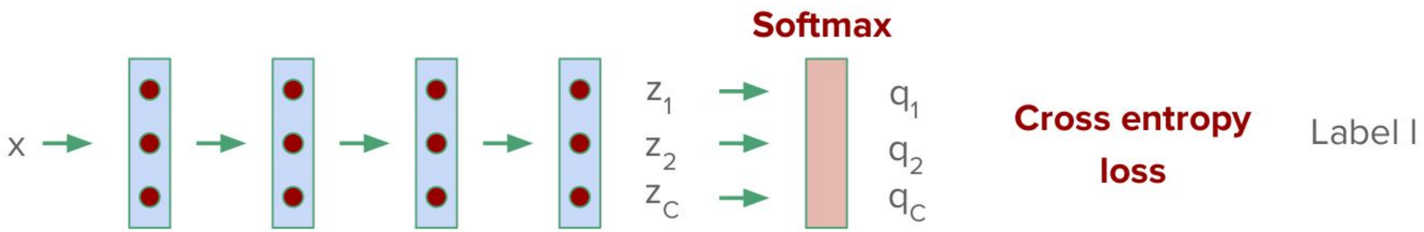
Training data:

$\{x_i\}$ A set of inputs (images, medical data, etc...)

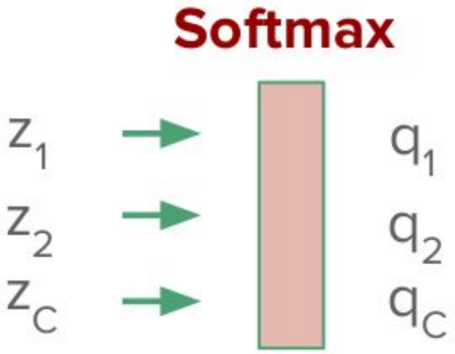
$\{l_i\}$ A set of true labels (class, disease state, etc...)

$l_i \in \{1, 2, 3, \ldots, C\}$ C is the total number of classes

MLP:



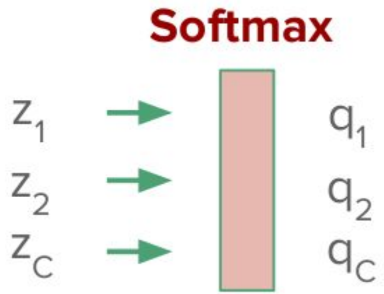
Converting logits (z) to probabilities (q)



$$q_i > 0$$

$$\sum_{i=1}^{C} q_i = 1$$

Converting logits (z) to probabilities (q)



$$q_i > 0$$

$$\sum_{i=1}^{C} q_i = 1$$

The softmax function solves these constraints:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

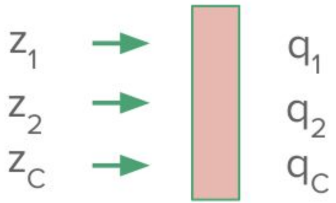
Saturdays.AI
Kigali

Training: inc probability of correct class
dec probability of incorrect classes

Goal: increase q_j if and only if $j = l_i$

Training: inc probability of correct class
dec probability of incorrect classes

Softmax



Cross entropy loss

Label l_i

Goal: increase q_j if and only if $j = l_i$

$$\mathcal{L}_i = \sum_j -Y_{ij} \log q_j = -\log q_{l_i}$$

One hot encoding of labels

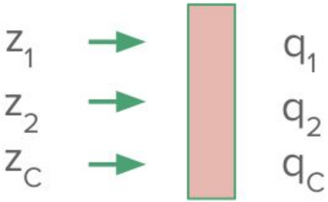
$$l_i \rightarrow \begin{array}{ll} y_{ij} = 1 & \text{if } j = l_i \\ y_{ij} = 0 & \text{if } j \neq l_i \end{array}$$

$l_1 = 1$ $\quad$ $l_1 = 2$ $\quad$ $l_1 = 3$

$$\boxed{\begin{matrix}1\\0\\0\end{matrix}} \quad \boxed{\begin{matrix}0\\1\\0\end{matrix}} \quad \boxed{\begin{matrix}0\\0\\1\end{matrix}}$$

Training: inc probability of correct class
dec probability of incorrect classes

Softmax



Cross entropy loss

Label l_i

Goal: increase q_j if and only if $j = l_i$

$$\mathcal{L}_i = \sum_j -Y_{ij} \log q_j = -\log q_{l_i}$$

Loss on training example i

One hot encoding of labels

$$l_i \quad \rightarrow \quad \begin{array}{ll} y_{ij} = 1 & \text{if } j = l_i \\ y_{ij} = 0 & \text{if } j \neq l_i \end{array}$$

$$l_1 = 1 \qquad l_1 = 2 \qquad l_1 = 3$$

$$\boxed{\begin{matrix}1\\0\\0\end{matrix}} \qquad \boxed{\begin{matrix}0\\1\\0\end{matrix}} \qquad \boxed{\begin{matrix}0\\0\\1\end{matrix}}$$

Cross entropy, entropy and KL divergence

Y_j correct distribution over C labels

q_j network's distribution over C labels

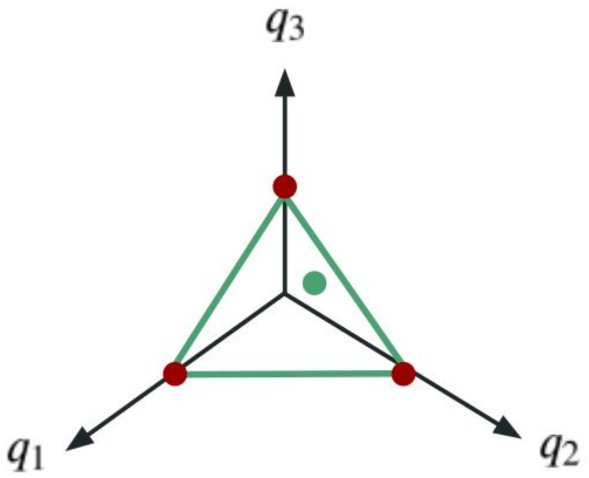
$$\mathcal{L} = \underbrace{-\sum_{j=1}^{C} Y_j \log q_j}_{\text{Cross entropy}} = \underbrace{-\sum_j Y_j \log Y_j}_{\text{Entropy of Y}} + \underbrace{\sum_j Y_j \log \frac{Y_j}{q_j}}_{\text{KL divergence}}$$

$H(Y)$

$D_{\mathrm{KL}}(Y||q)$

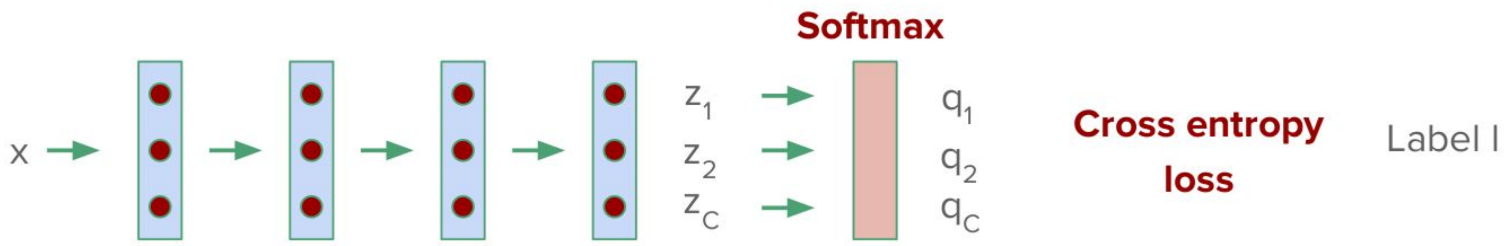
≥ 0

$= 0 \iff Y = q$



Saturdays.AI Kigali

Training scheme for classification



$$\mathcal{L} = \sum_i \sum_j -Y_{ij} \log q_j(x_i, \mathbf{w})$$

Now your turn! You get to play with code to implement the softmax + cross entropy

Practice

Implement Batch Cross Entropy Loss

- Implement Batch CE Loss!

[Coding Exercise 2.1]

Break

Saturdays.AI
Kigali

Putting it together: training and evaluating an MLP in PyTorch

Let's train an MLP!

But wait, how do we know if it actually works after we train it?

Saturdays.AI
Kigali

Cross-validation to combat overfitting

Training set

Test set

Use to train
the model

Use to tune
hyperparameters

Training error < Test error (if much less then you are overfitting)

Saturdays.AI
Kigali

If you tune too many hyperparameters...

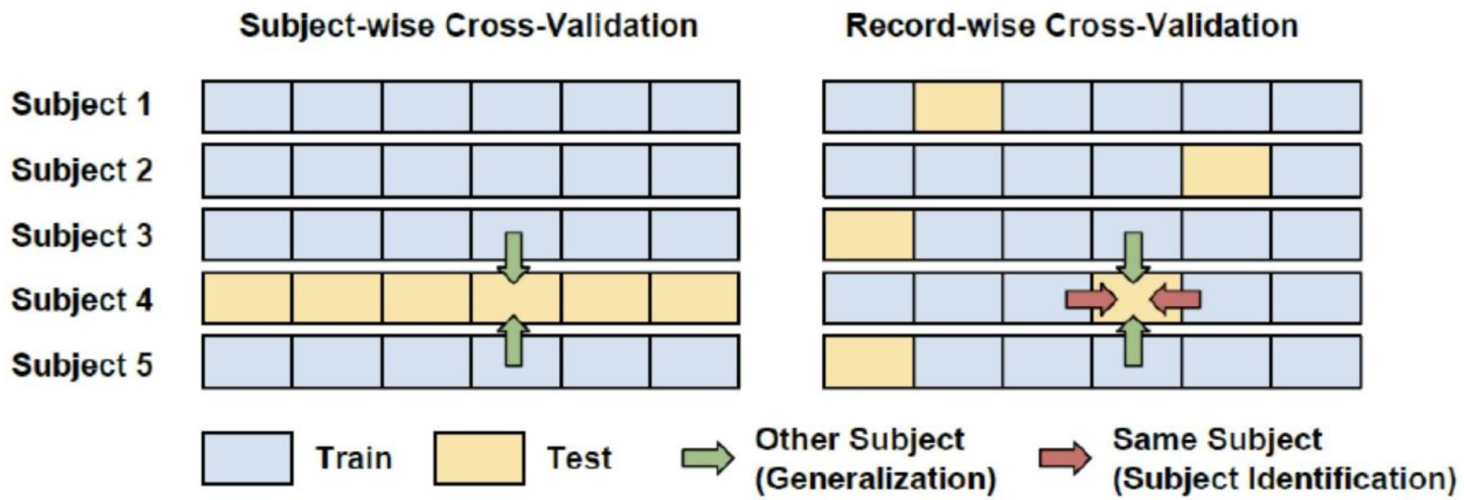
Training set	Test set
Use to train the model	Use to tune hyperparameters

Training error < Test error (if much less then you are overfitting)

Training set	Validation set	
Use to train the model	Use to tune hyperparameters	Use to test model and hyperparameters

Saturdays.AI
Kigali

The cross-validation strategy must match the use case



Saturdays.AI
Kigali

The evaluation metric must be meaningful

Example: Many people do mood estimates on mobile phones. R2~.7

$$R^2 = 1 - \frac{Var\ (\text{Model-Reality})}{Var\ (\text{Reality})}$$

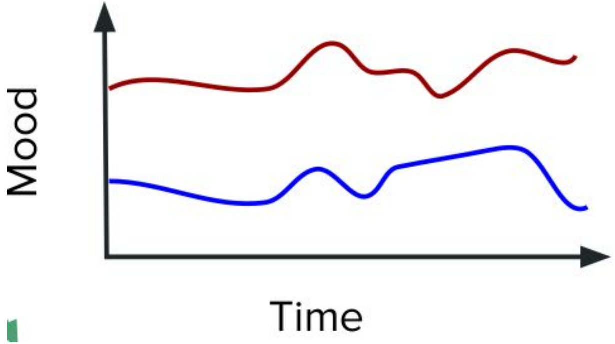
Used complex subject specific models.

What is the denominator?

Variance of reported mood across

Variance of reported mood within

But trivial within subject mean can get average R2~.7



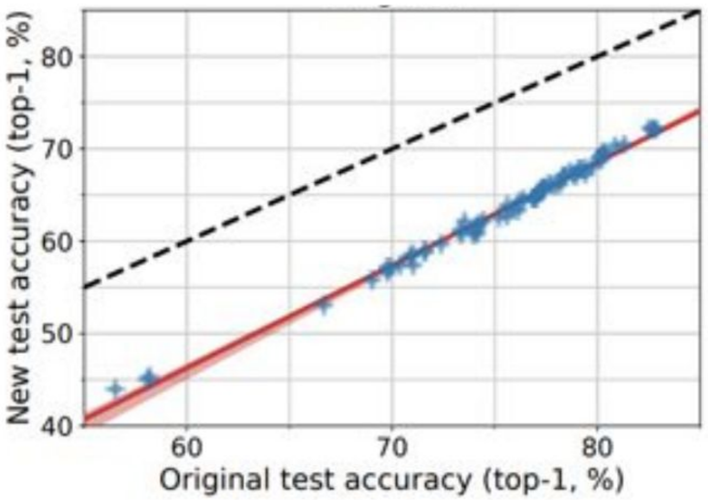
Saturdays.AI
Kigali

An entire field overfitting on a dataset?

Collect new images using same protocol as CIFAR10 / ImageNet

Get accuracy drops of ~ 10 percent

On ImageNet: corresponds to loss of 5 years of progress in performance



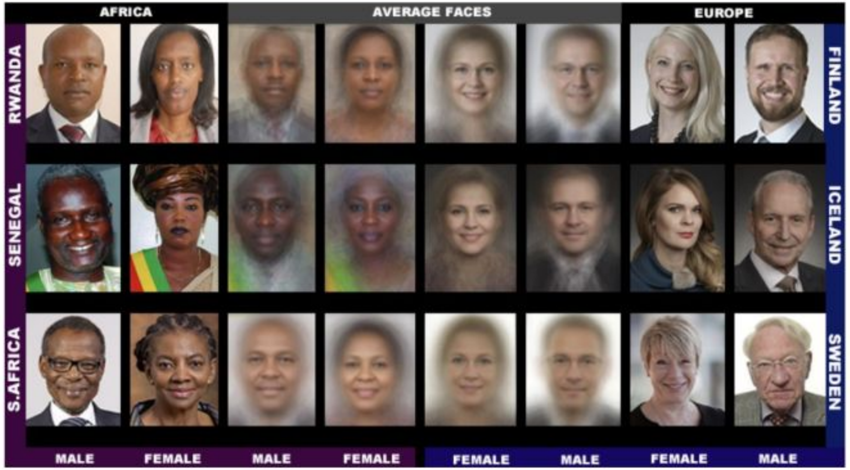
Saturdays.AI
Kigali

Overall accuracy on a test set is not enough: bias and fairness

3 commercial gender classification systems performed significantly worse on:

- females compared to males
- darker compared to lighter skinned faces

Need to make sure performance across important subpopulations is uniformly high



Buolamwini and Gebru, Conf on Fairness, Accountability and Transparency, 2018.

Saturdays.AI Kigali

So be careful in training and evaluation!

Need to make important choices of:

- What is the training set?
- Does the split between train and test match your use case?
- Is your metric for evaluation reasonable for real world deployment?
- Will future validation data drift from train and test settings?
- Are there biases due to problem selection, training data, algorithm design, evaluation metrics, or anywhere in ML pipeline?

Saturdays.AI
Kigali

Practice

Implement MLP for a classification

- Implement a simple train/test split for training and validation.

[Coding Exercise 2.3]

Challenges & Next steps!

Kahoot!

Saturdays.AI
Kigali

Any questions?

Saturdays.AI
Kigali

Saturdays.AI
Kigali

THANKS

kigali@saturdays.ai
coming soon
coming soon