# Machine learning for particle physicists

## II. Introduction to neural networks

Anja Butter

26th Vietnam School of Physics

# Recap I - Linear Regression

Task: Model calorimeter response

Model: $h(\boldsymbol{x}) = \boldsymbol{x}\boldsymbol{w} = \tilde{y} \leftarrow$ prediction

Loss function: $$\mathcal{L} = \sum_{i}^{n_{data}} (y_i - \tilde{y})^2$$

Optimized parameters: $\boldsymbol{w}_{opt} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$

# Recap II - Logistic Regression

Task: Classify jets

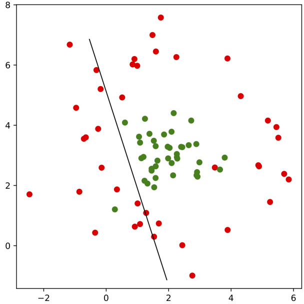
Model: $h(\boldsymbol{x}) = \sigma(\boldsymbol{x}\boldsymbol{w}) = P(y = 1|\boldsymbol{x}) \leftarrow$ probability

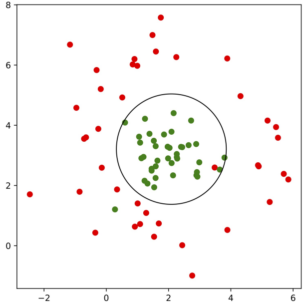
Loss function: $$\mathcal{L} = \sum_i -y_i \log\left(h(\boldsymbol{x}_i)\right) - (1 - y_i) \log\left(1 - h(\boldsymbol{x}_i)\right)$$

Parameter optimization: $\boldsymbol{w}_n \to \boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \alpha \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_n)$

# Limitations

So far we were limited to linear separations:



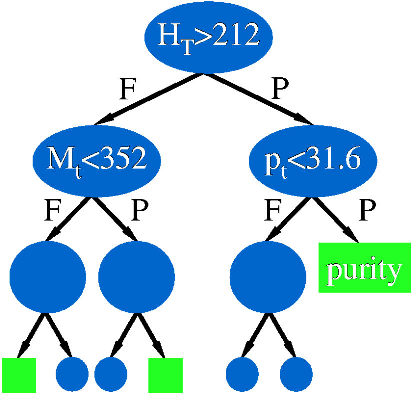

$\rightarrow$ Need more complex models to learn complex structures

Many possibilities:

boost decision trees $\rightarrow$ popular, close to intuitive cut & count analysis

# Decision Trees

Example from single top measurement

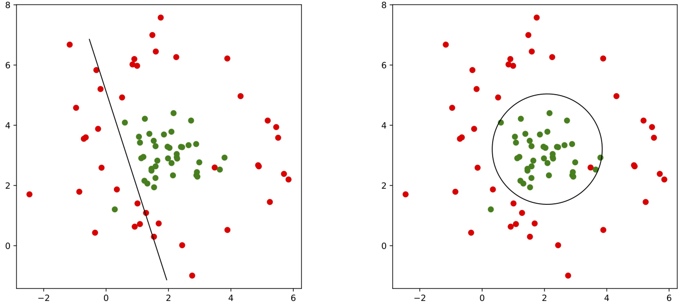



$$\text{purity} = \frac{s}{s+b}$$

$\Rightarrow$ Further information: boosting, random forest

# Limitations

So far we were limited to linear separations:



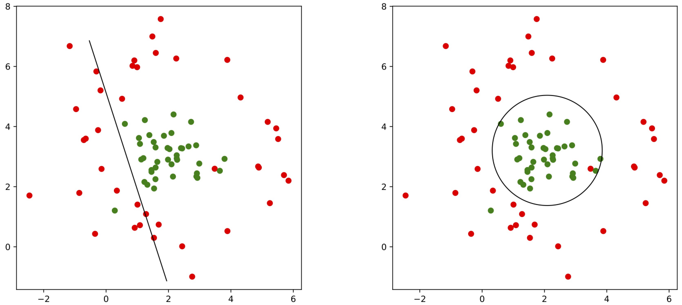

→ Need more complex models to learn complex structures

Many possibilities:

boost decision trees → popular, close to intuitive cut & count analysis

# Limitations

So far we were limited to linear separations:




→ Need more complex models to learn complex structures

Many possibilities:

boost decision trees → popular, close to intuitive cut & count analysis
support vector machines → often very good baseline
neural networks → more flexibility, excellent performance

# Why neural?

Inspiration from brain

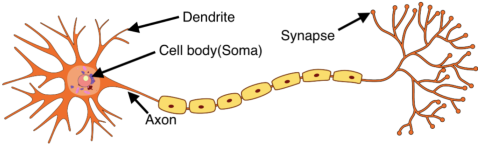
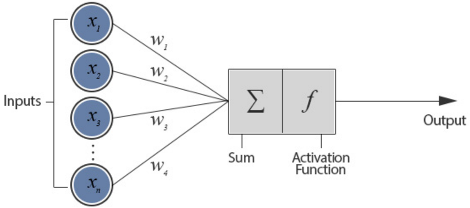



Image source: Wikimedia Commons

1. Dendrites get incoming signal
2. Soma processes signal (fire?!)
3. Axon transports signal to cells
4. Synapse connects to other dendrites

Intelligence is *somehow* the product of the connection of many neurons.

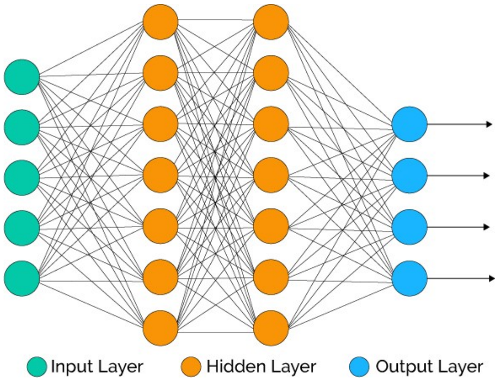
# An artificial neuron




$$\rightarrow output = f\left(\sum_i x_i w_i\right)$$
$$= f(\boldsymbol{xw})$$

Combination of linear mapping & non-linear activation function

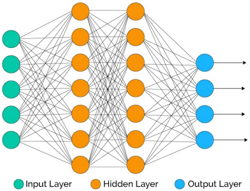
# A simple neural network




Every line represents a free parameter called weight

# The math behind a neural network

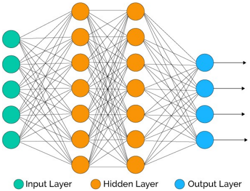
## Matrix multiplications...




input
$\boldsymbol{X} = [n_{data} \times d_{feat}]$

# The math behind a neural network

## Matrix multiplications...



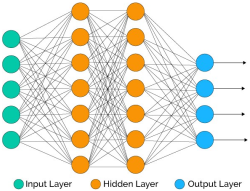

input
$\boldsymbol{X} = [n_{data} \times d_{feat}]$

$\downarrow$

$\boldsymbol{X}$
$[n_{data} \times d_{feat}]$

# The math behind a neural network

## Matrix multiplications...




input
$\boldsymbol{X} = [n_{data} \times d_{feat}]$

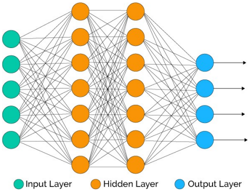
layer 1
$W_1 = [d_{feat} \times d_{l1}]$

$\boldsymbol{X}$
$[n_{data} \times d_{feat}]$

$\boldsymbol{X} W_1$
$[n_{data} \times d_{l1}]$

# The math behind a neural network

## Matrix multiplications...




input
$\boldsymbol{X} = [n_{data} \times d_{feat}]$

layer 1
$W_1 = [d_{feat} \times d_{l1}]$

layer 2
$W_2 = [d_{l1} \times d_{l2}]$

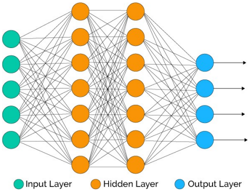
$\boldsymbol{X}$
$[n_{data} \times d_{feat}]$

$\boldsymbol{X} W_1$
$[n_{data} \times d_{l1}]$

$\boldsymbol{X} W_1 W_2$
$[n_{data} \times d_{l2}]$

# The math behind a neural network

## Matrix multiplications...




$$
\begin{array}{ccccccc}
\boxed{\begin{array}{c}\text{input}\\ \boldsymbol{X} = [n_{data} \times d_{feat}]\end{array}} & \rightarrow & \boxed{\begin{array}{c}\text{layer 1}\\ W_1 = [d_{feat} \times d_{l1}]\end{array}} & \rightarrow & \boxed{\begin{array}{c}\text{layer 2}\\ W_2 = [d_{l1} \times d_{l2}]\end{array}} & \rightarrow & \boxed{\begin{array}{c}\text{output}\\ w_{out} = [d_{l2} \times 1]\end{array}} \\
\downarrow & & \downarrow & & \downarrow & & \downarrow \\
\boxed{\begin{array}{c}\boldsymbol{X}\\ [n_{data} \times d_{feat}]\end{array}} & & \boxed{\begin{array}{c}\boldsymbol{X} W_1\\ [n_{data} \times d_{l1}]\end{array}} & & \boxed{\begin{array}{c}\boldsymbol{X} W_1 W_2\\ [n_{data} \times d_{l2}]\end{array}} & & \boxed{\begin{array}{c}\boldsymbol{X} W_1 W_2 w_{out}\\ [n_{data} \times 1]\end{array}}
\end{array}
$$

# If we combine several linear layers...

$$
\begin{aligned}
\boldsymbol{Y} &= \boldsymbol{X} \cdot W_1 \cdot W_2 \cdot w_{out} \\
&= \boldsymbol{X}_{ij} W_{1,jk} W_{2,kl} W_{out,l} && \leftarrow \text{ Einstein summation convention} \\
&= \boldsymbol{X}_{ij} \tilde{\boldsymbol{w}}_j && \text{with } \tilde{\boldsymbol{w}} = W_{1,jk} W_{2,kl} W_{out,l}
\end{aligned}
$$

... we obtain a linear layer!

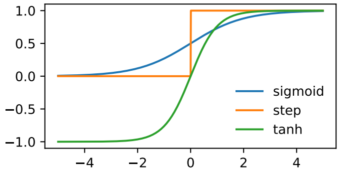
Not more expressiveness than a simple scalar product!

$\rightarrow$ Include non-linearities

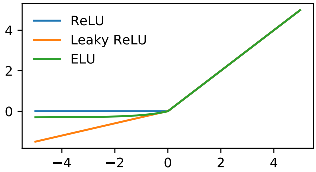
# Activation functions

The right choice can facilitate the training:
$\rightarrow$ smooth/sharp, limited/unlimited, computing time efficient, ...

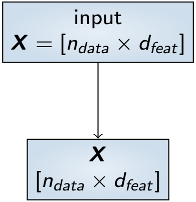
- Limited (typically for classification)
  - Sigmoid: $\sigma(x) = \dfrac{1}{1 + e - x}$
  - Step: $\theta(x) = \text{sign}(x)$
  - tanh: $\tanh(x)$



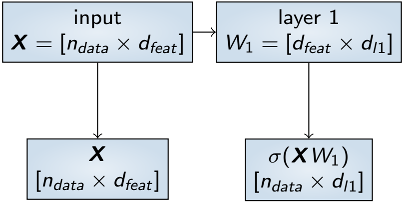

- Unlimited
  - ReLU: $\max(0, x)$
  - Leaky ReLU: $\max(\alpha x, x)$
  - ELU: $\begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x < 0 \end{cases}$



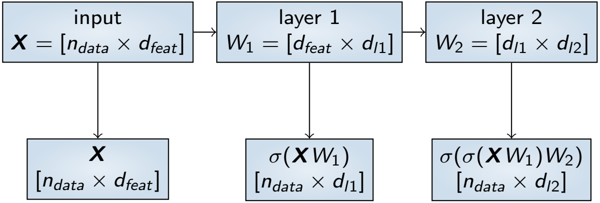
# A simple neural network



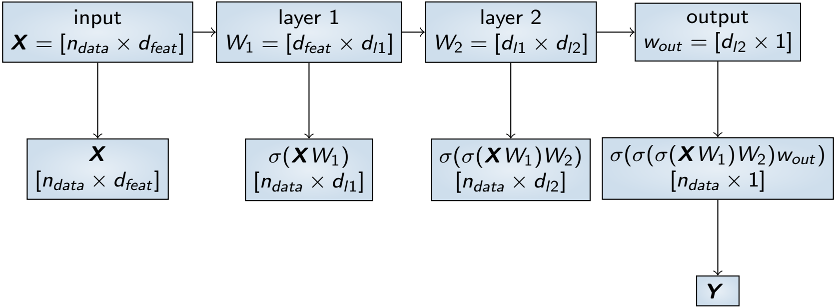
# A simple neural network



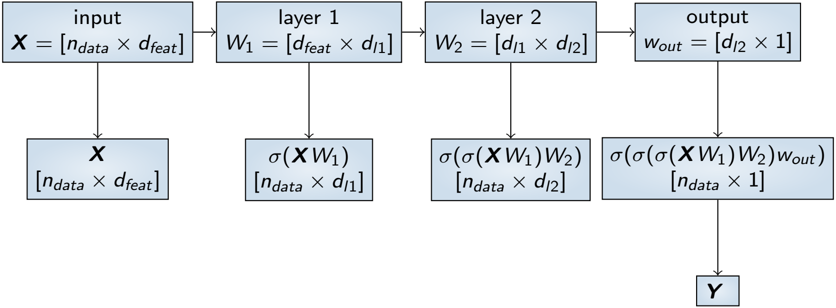
# A simple neural network



# A simple neural network



# A simple neural network




Finally we have obtained an expressive network!
But how can we train it?
Gradient descent?!

# How to minimize the loss

Remember gradient descent for logistic regression:

$$\rightarrow \boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \alpha \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_n)$$

We start from the last layer:
Let $\tilde{\boldsymbol{X}}$ be the latent representation after the 2. layer

$$\mathcal{L}(\boldsymbol{w}_{out}) = \sum_{j=1}^{n_{data}} (\sigma(\tilde{\boldsymbol{X}}\boldsymbol{w}_{out})_j - Y_j)^2$$

$$\frac{\partial \mathcal{L}(\boldsymbol{w}_{out})}{\partial w_{out,i}} = \sum_{j=1}^{n_{data}} 2(\sigma(\tilde{\boldsymbol{X}}\boldsymbol{w}_{out})_j - Y_j) \cdot \frac{\partial \sigma(\tilde{\boldsymbol{X}}\boldsymbol{w}_{out})_j}{\partial w_{out,i}} \leftarrow \text{ chain rule}$$

$$= \sum_{j=1}^{n_{data}} 2(\sigma(\tilde{\boldsymbol{X}}\boldsymbol{w}_{out})_j - Y_j) \cdot \frac{\partial \sigma(\tilde{\boldsymbol{X}}\boldsymbol{w}_{out})_j}{\partial (\tilde{\boldsymbol{X}}\boldsymbol{w}_{out})_j} \cdot \frac{\partial (\tilde{\boldsymbol{X}}\boldsymbol{w}_{out})_j}{\partial w_{out,i}}$$

# How to minimize the loss

Remember gradient descent for logistic regression:

$$\rightarrow \boldsymbol{w}_{n+1} = \boldsymbol{w}_n + \alpha \nabla_{\boldsymbol{w}} \mathcal{L}(\boldsymbol{w}_n)$$

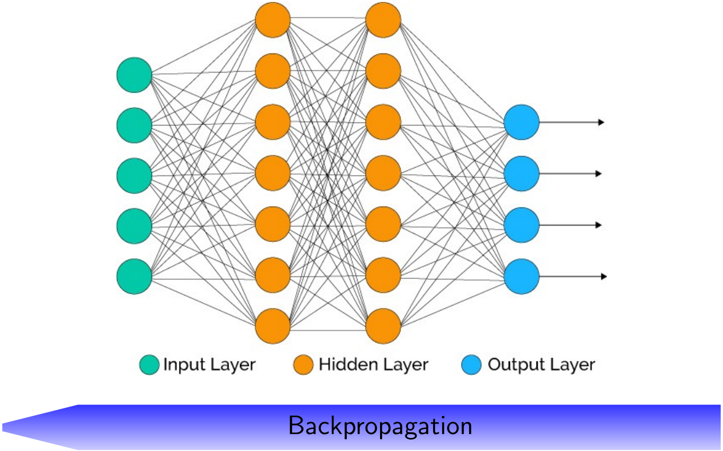
We start from the last layer:
Let $\tilde{\boldsymbol{X}}$ be the latent representation after the 2. layer

$$\mathcal{L}(\boldsymbol{w}_{out}) = \sum_{j=1}^{n_{data}} (\sigma(\tilde{\boldsymbol{X}} \boldsymbol{w}_{out})_j - Y_j)^2$$

$$\frac{\partial \mathcal{L}(\boldsymbol{w}_{out})}{\partial w_{out,i}} = \sum_{j=1}^{n_{data}} 2(\sigma(\tilde{\boldsymbol{X}} \boldsymbol{w}_{out})_j - Y_j) \cdot \frac{\partial \sigma(\tilde{\boldsymbol{X}} \boldsymbol{w}_{out})_j}{\partial w_{out,i}} \leftarrow \text{ chain rule}$$

$$= \sum_{j=1}^{n_{data}} 2(\sigma(\tilde{\boldsymbol{X}} \boldsymbol{w}_{out})_j - Y_j) \cdot \frac{\partial \sigma(\tilde{\boldsymbol{X}} \boldsymbol{w}_{out})_j}{\partial (\tilde{\boldsymbol{X}} \boldsymbol{w}_{out})_j} \cdot \tilde{X}_{ji}$$

# Backpropagation



# Updating the n-th layer

output
$w_{out} = [d_{l2} \times 1]$

$$\rightarrow \frac{\partial \mathcal{L}}{\partial \sigma_3} \frac{\partial \sigma_3}{\partial lin3}$$

# Updating the n-th layer

$$\text{layer 2} \quad W_2 = [d_{l1} \times d_{l2}] \longleftarrow \text{output} \quad w_{out} = [d_{l2} \times 1]$$

$$\rightarrow \frac{\partial \mathcal{L}}{\partial \sigma_3} \frac{\partial \sigma_3}{\partial lin3} \cdot \frac{\partial lin3}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial lin2}$$

# Updating the n-th layer

$$\boxed{\begin{array}{c}\text{layer 1}\\ W_1 = [d_{feat} \times d_{l1}]\end{array}} \leftarrow \boxed{\begin{array}{c}\text{layer 2}\\ W_2 = [d_{l1} \times d_{l2}]\end{array}} \leftarrow \boxed{\begin{array}{c}\text{output}\\ w_{out} = [d_{l2} \times 1]\end{array}}$$

$$\rightarrow \frac{\partial \mathcal{L}}{\partial \sigma_3} \frac{\partial \sigma_3}{\partial lin3} \cdot \frac{\partial lin3}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial lin2} \cdot \frac{\partial lin2}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial lin1}$$

# Updating the n-th layer

$$\begin{array}{c} \text{input} \\ \boldsymbol{X} = [n_{data} \times d_{feat}] \end{array} \leftarrow \begin{array}{c} \text{layer 1} \\ W_1 = [d_{feat} \times d_{l1}] \end{array} \leftarrow \begin{array}{c} \text{layer 2} \\ W_2 = [d_{l1} \times d_{l2}] \end{array} \leftarrow \begin{array}{c} \text{output} \\ w_{out} = [d_{l2} \times 1] \end{array}$$

$$\rightarrow \frac{\partial \mathcal{L}}{\partial \sigma_3} \frac{\partial \sigma_3}{\partial lin3} \cdot \frac{\partial lin3}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial lin2} \cdot \frac{\partial lin2}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial lin1} \cdot \frac{\partial lin1}{\partial W_1}$$

# Updating the n-th layer

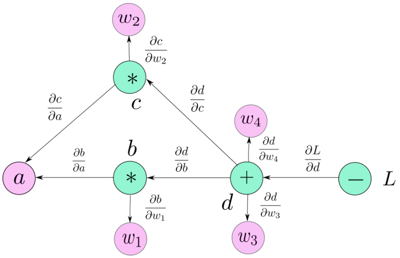
| input | | layer 1 | | layer 2 | | output |
|---|---|---|---|---|---|---|
| $\boldsymbol{X} = [n_{data} \times d_{feat}]$ | $\leftarrow$ | $W_1 = [d_{feat} \times d_{l1}]$ | $\leftarrow$ | $W_2 = [d_{l1} \times d_{l2}]$ | $\leftarrow$ | $w_{out} = [d_{l2} \times 1]$ |

$$\rightarrow \frac{\partial \mathcal{L}}{\partial \sigma_3} \frac{\partial \sigma_3}{\partial lin3} \cdot \frac{\partial lin3}{\partial \sigma_2} \frac{\partial \sigma_2}{\partial lin2} \cdot \frac{\partial lin2}{\partial \sigma_1} \frac{\partial \sigma_1}{\partial lin1} \cdot \frac{\partial lin1}{\partial W_1}$$

Latest winner of ImageNet 'ViT-H/14' has 32 layers with a total of 632M parameters
Let's start calculating...?

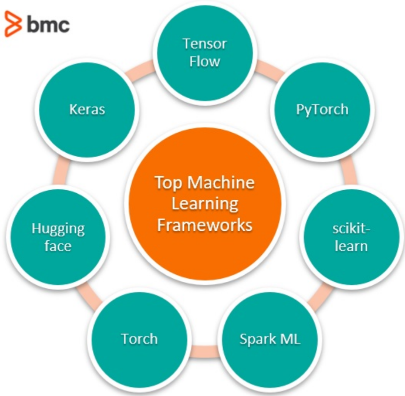
# Automatic gradient computations

Backpropagation implemented via computation graphs in dedicated frameworks




Ayoosh Kathuria

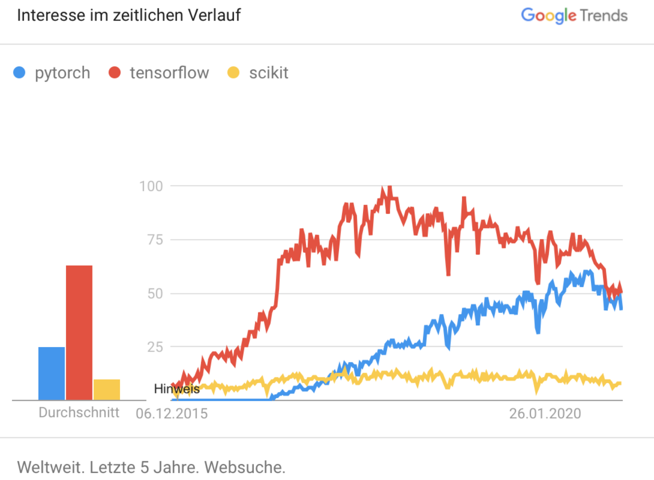
# High-level ML frameworks




- TensorFlow & PyTorch → DNN
- Keras → user friendly TF interface
- Scikit-learn → SVM, BDT, clustering
- Spark ML → part of Spark, basic ML
- Hugging face → NLP, transformers

# Which framework to choose for NN?

Biggest players: PyTorch (facebook) and TensorFlow (google)



# Before you start training your own neural network

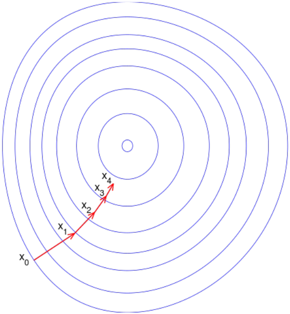
Let's talk about:

$\rightarrow$ efficient training

$\rightarrow$ overtraining (overfitting)

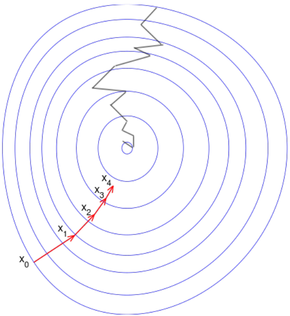
# Stochastic gradient descent

- Loss defined on entire dataset
- Each weight update: gradient for full dataset
  $\rightarrow$ very computing expensive!



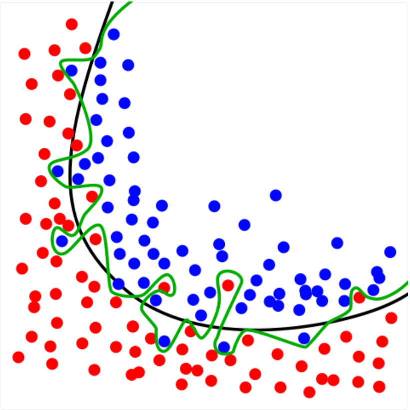
# Stochastic gradient descent

- Loss defined on entire dataset
- Each weight update: gradient for full dataset
  $\rightarrow$ very computing expensive!
- SGD: 1 iteration: gradient for 1 random data point
- Compromise: batch gradient descent (batch size: 32/62/.../1024)




\+ faster
\+ less sensitive to local minima
\+ profit from vectorization

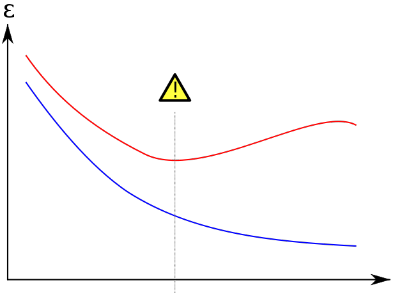
# Overtraining



… when networks start to learn "noise"

# How to control overtraining?

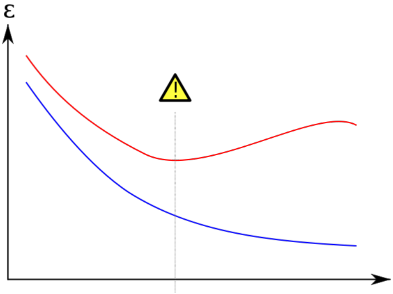
Split data into training/validation/test data




- Training loss $\nearrow$ validation loss $\searrow$ $\Rightarrow$ overfitting
- Why do we need test data?
- How would you split your dataset? 1:1:1? 8:1:1?

# How to control overtraining?

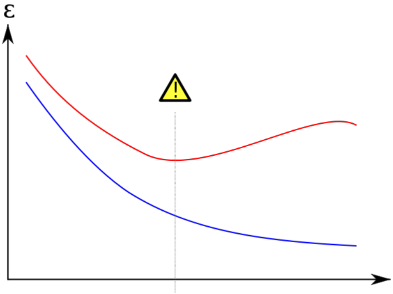
Split data into training/validation/test data




- Training loss $\nearrow$ validation loss $\searrow$ $\Rightarrow$ overfitting
- Why do we need test data?
  - parameter tests = training on validation dataset
- How would you split your dataset? 1:1:1? 8:1:1?

# How to control overtraining?

Split data into training/validation/test data




- Training loss $\nearrow$ validation loss $\searrow \Rightarrow$ overfitting
- Why do we need test data?
  - parameter tests = training on validation dataset
- How would you split your dataset? 1:1:1? 8:1:1?
  - dataset dependent
  - val/test data large enough to test performance

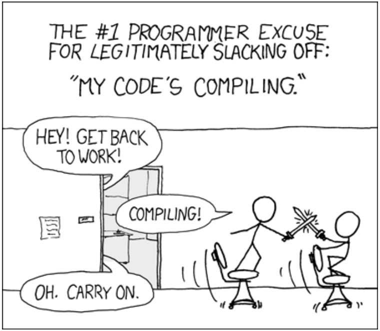
# Today's summary

We saw:

- How to build a network with multiple layers
- Why we need activation functions in each layer
- How to train a deep neural network $\rightarrow$ Backpropagation
- Stochastic/batch gradient descent
- Overtraining $\rightarrow$ Train - validate - test

Now you can start training your own neural network!

# Procrastination over the course of time





credits: XKCD & u/AmpyeriDracula