# Fast Shower Simulation with Deep Learning

## Final Summer Student Session

Peter McKeown[1,2]

Supervisors: Engin Eren[1] and Frank Gaede[1]

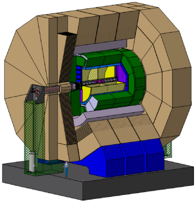
[1]DESY, Hamburg

[2]University of Nottingham

September 5th, 2019

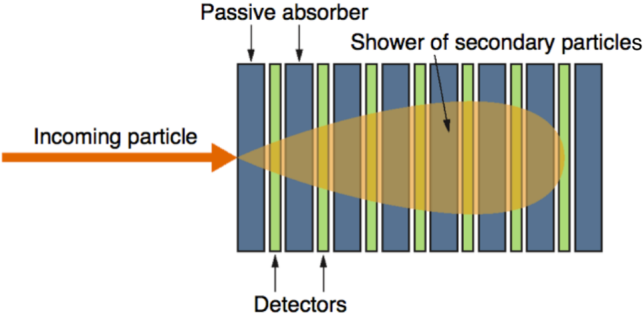
# Simulation for HEP (and why it needs to be fast)

- Detector simulation required for:
  - physics analysis $\rightarrow$ **rare signals require large statistics**
  - detector design and optimisation etc.
- Monte Carlo simulation time consuming and CPU intensive e.g **WLCG**
- Start with calorimeters - most computationally intensive part
- ILD has highly granular calorimeter

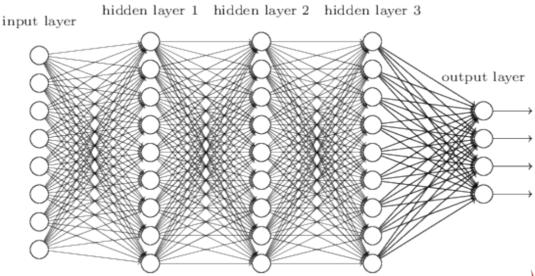


# Electromagnetic Calorimeter Showers

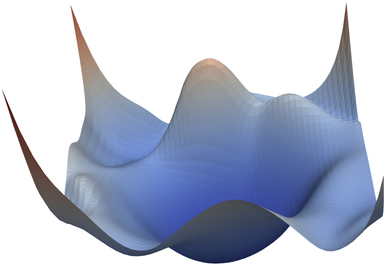
- Measure particle's energy destructively
- Produce a shower of secondary particles until absorbed totally
- ILD proposal uses an Si-W **sampling calorimeter**



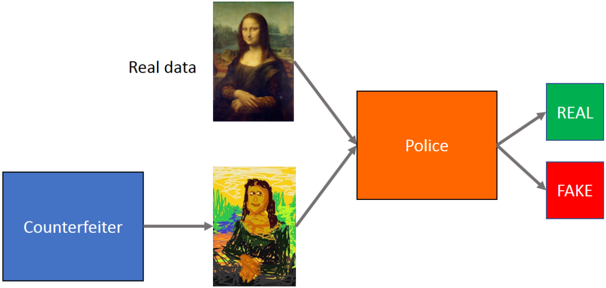
# Neural Network Overview



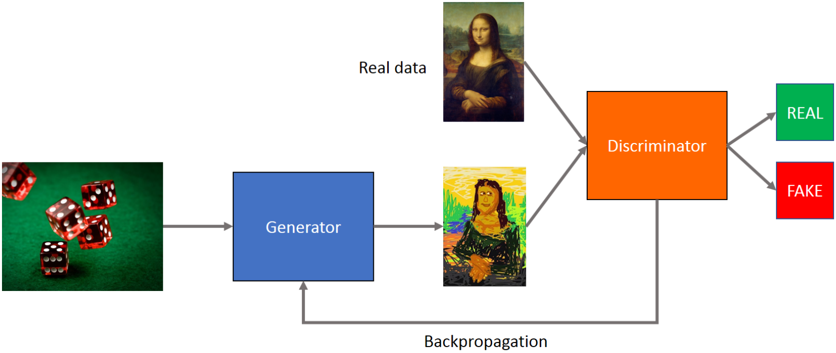

- Deep $\rightarrow$ fit non-linear functions
- Loss function $\rightarrow$ performance
- Learning $\rightarrow$ gradient descent (minimise loss)
- Backpropagation $\rightarrow$ update parameters



# Generative Adversarial Networks (GANs) The concept



# Generative Adversarial Networks (GANs) The concept

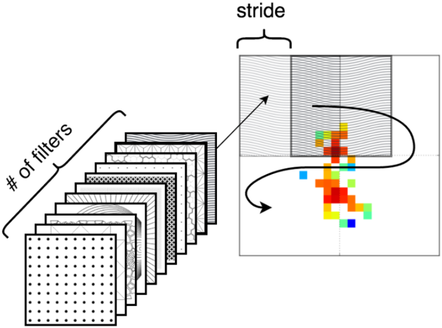


# The CaloGAN Architecture M. Paganini et al.

- Keras API with Tensorflow backend
- One generative model per particle type
- Loss function **encourages** conservation of energy
- Attentional mechanism
- Convolutional layers are replaced with locally connected layers

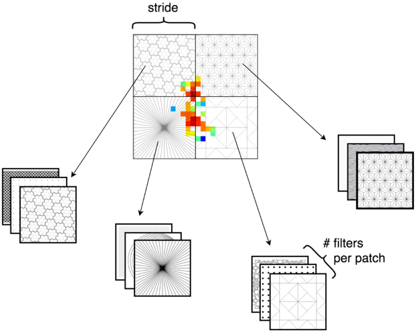
# The CaloGAN Architecture M. Paganini et al.

- Keras API with Tensorflow backend
- One generative model per particle type
- Loss function **encourages** conservation of energy
- Attentional mechanism
- Convolutional layers are replaced with locally connected layers



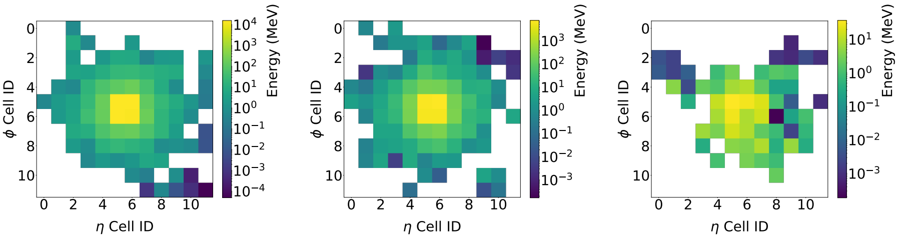
# The CaloGAN Architecture M. Paganini et al.

- Keras API with Tensorflow backend
- One generative model per particle type
- Loss function **encourages** conservation of energy
- Attentional mechanism
- Convolutional layers are replaced with locally connected layers

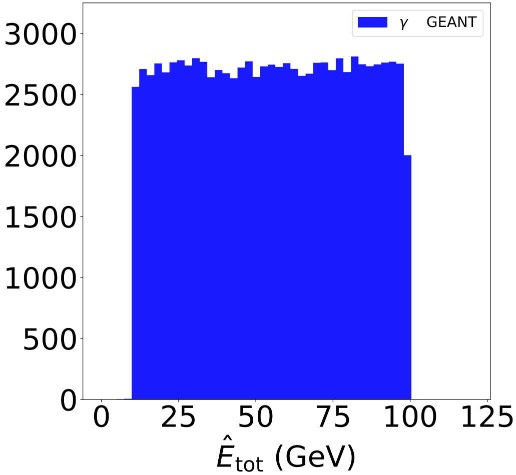


# Training data The Calorimeter Model

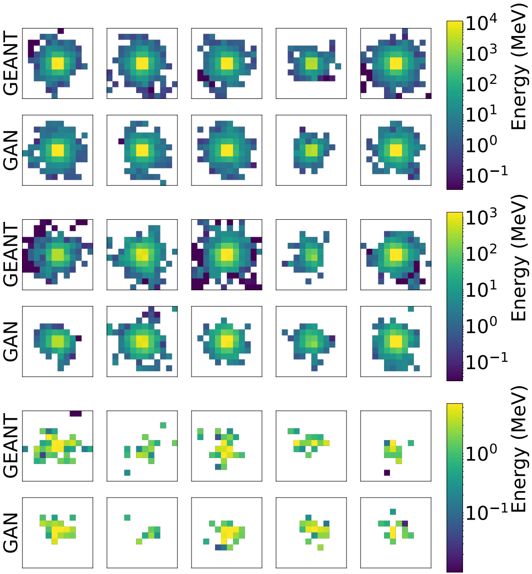
- Created with GEANT4
- Use Si active material and W passive material
- Three layers $\rightarrow$ created by **summing the energies** in absorber and readout
- Uniform segmentation in each layer



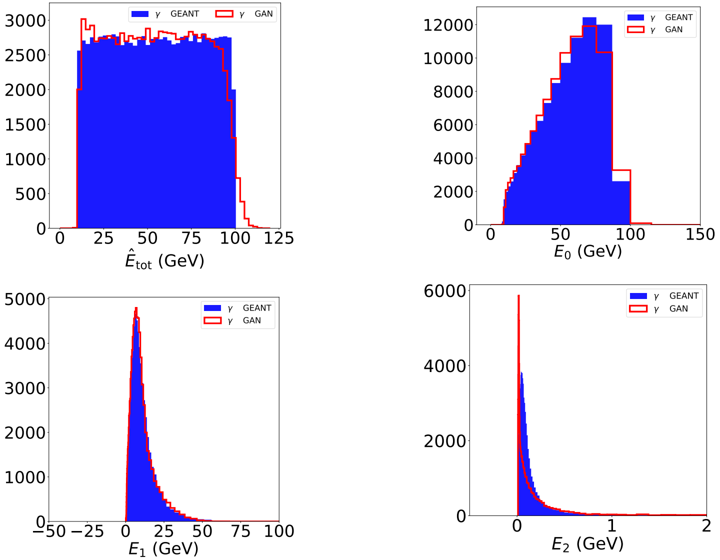
# Training strategy



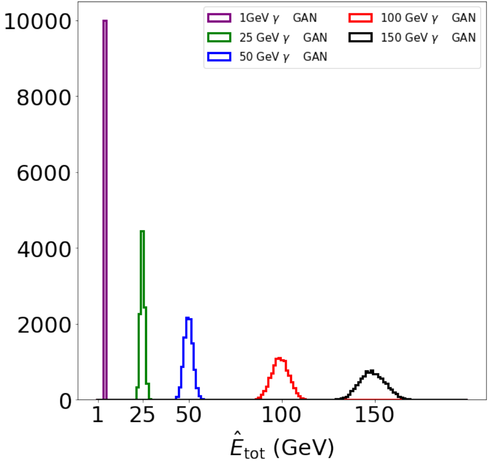
# Results Qualitative review



# Results Quantitative review



# Results Selecting a single energy




- $\frac{\sigma(E)}{E} \approx 5\%$
- Expect $\frac{\sigma(E)}{E} \sim \frac{1}{\sqrt{E}}$

# Outlook

- CaloGAN architecture is a good first step
- Altering loss function
- More layers
- Already been further steps
  - Different metric (e.g. Wasserstein)
  - Condition on more than energy (momentum, position etc.)

```
In [356]: # 1,000 is the number of showers we want to generate right now - 25 GeV
          noise_2 = np.random.normal(0, 1, (1000, latent_size))
          sampled_energy_2 = np.random.uniform(25, 25, (1000, 1))

In [357]: images_2 = generator.predict([noise_2, sampled_energy_2], verbose=True)

          1000/1000 [==============================] - 3s 3ms/step
```

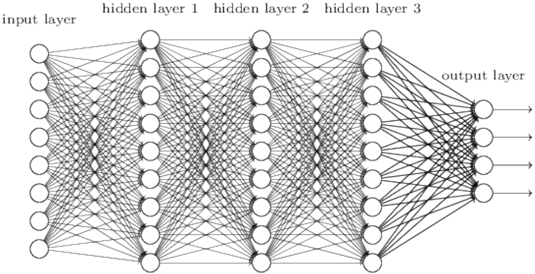
```
Run Summary
  Number of events processed : 1000
  User=3241.99s Real=3275.17s Sys=3.94s
```

# Backup

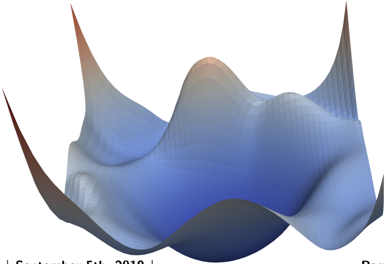
# Simulation for HEP (and why it needs to be fast)

- Detector simulation required for:
  - physics analysis $\rightarrow$ **rare signals require large statistics**
  - detector design and optimisation etc.
- MC simulation very time consuming and CPU intensive e.g **WLCG**
- Several possible speed up methods proposed:
  - Reduce **quantity** of simulation
  - **Optimise** simulation
- Cannot use MC for further speed up
  - Start with calorimeters - most computationally intensive part

# Neural Network Overview




- Deep $\rightarrow$ fit non-linear functions
- Activation of given neuron:
  $\vec{\mathbf{a}}^{(1)} = \sigma(W\vec{\mathbf{a}}^{(0)} + \vec{\mathbf{b}})$
- Loss function $\rightarrow$ problem dependent
- Learning $\rightarrow$ gradient descent (minimise loss)
- Backpropagation $\rightarrow$ update $W$ and $\vec{\mathbf{b}}$



# Generative Adversarial Networks (GANs) Further details

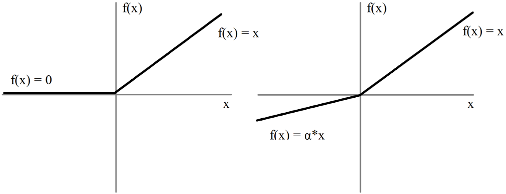
- Generator (G) tries to match the training data distribution
- Discriminator (D) is a binary classifier
- Two player minmax game:
  - D wants to **maximise** objective function
  - G wants to **minimise** objective function
- Unique solution $\rightarrow$ **Nash equilibrium**
  - G recovers training data
  - $D = 0.5$

# Objective function for the original GAN

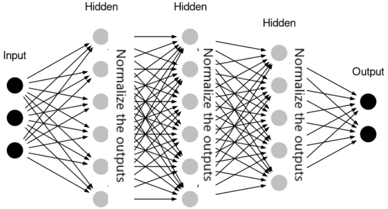
- Discriminator objective:
  $\max_D V(D) = \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$
- Generator objective: $\min_G V(G) = \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$
- Loss: $\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{data}}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))]$

# Challenges sparsity, mode collapse, HDR

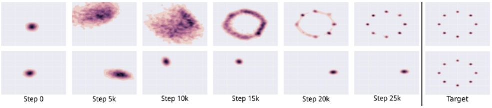
- Sparsity → ReLU and leaky ReLU activation functions



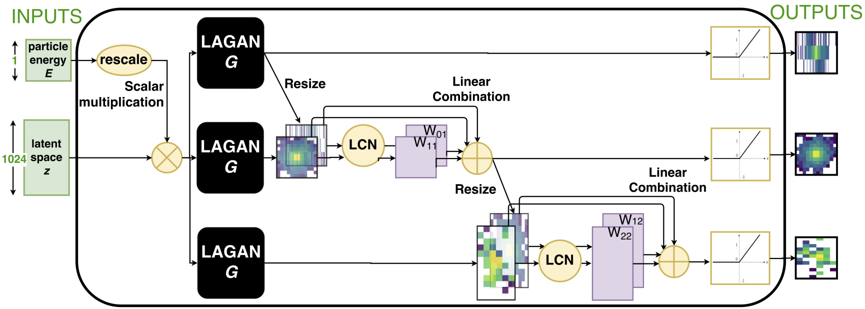

- High Dynamic Range → Batch normalisation



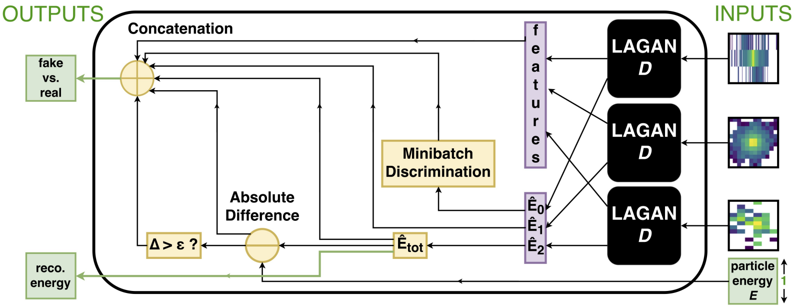

- Mode Collapse → Mini-batch Discrimination



# CaloGAN generator



# CaloGAN discriminator



# Results Selecting a single energy