A jet = collection of ptcl 4-vectors "LLFs"

$$J_i = \begin{pmatrix} p_x^1, p_y^1, p_z^1, E^1 \\ \vdots \\ p_x^{N_i}, p_y^{N_i}, p_z^{N_i}, E^{N_i} \end{pmatrix}$$

N_i diff from jet to jet

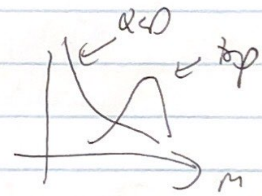
$(E^a)^2 = (\vec{p}^{\,a})^2$

HLFs like jet mass $m_i^2 = \left(\sum_{a=1}^{N_i} E^a\right)^2 - \left(\sum_{a=1}^{N_i} \vec{p}^{\,a}\right)^2$

and N-subjettiness τ_{32}

↓

physics motivated fns of LLFs



We have seen:

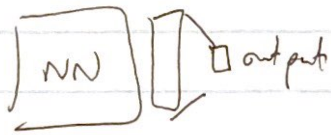
cut based classifier on HLFs $\simeq$ DNN on HLFs

$\simeq$ DNN on LLFs (30 highest p_T constituents)

"Automated feature engineering"

- DNN can construct HLFs from LLFs that perform as well as physics HLFs!
  - Q: is it learning the physics HLFs or something else? How would we know?

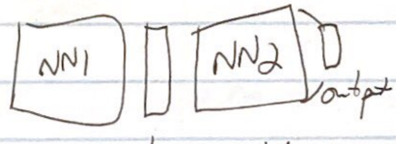
    A: Could add physics HLFs to inputs - perf. improves?

- 

last layer → these are like NN engineered HLFs!

automated the HLF engineer

not necessarily very last layer

- 

↳ could interpret these as HLFs

- Is this all or can we surpass physics HLF performance?

→ yes but need more powerful NN architectures!

General lesson: → leverage symmetry / structure of data!

like getting more data for free

"inductive bias" → helps machine learn more effectively

our DNN p_T ordered constituents

selected 30 hardest

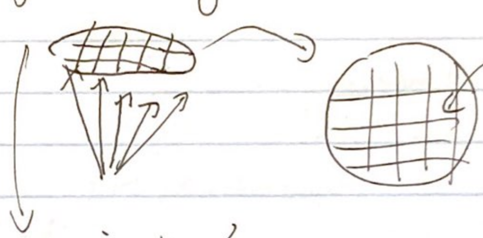
But jets have <u>permutation inv'ce</u>! order of constituents doesn't matter

also: <u>Lorentz symmetry</u> → spatial <u>rotations</u> & boosts

"point cloud"

· A rich playground for different ML architectures

- CNNs — jets as images



each pixel p_T or E deposited

↓

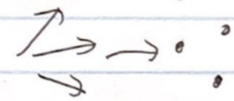
like a real detector!

perm inv't ✓

Lorentz sym? → spatial rotations & translations

- RNNs, LSTMs, ⋯ — jets as sequences
  — not perm inv't! what ordering?

- Graph NNs — perm inv't!
  (can add Lorentz)



- Deepsets — a perm inv't DNN

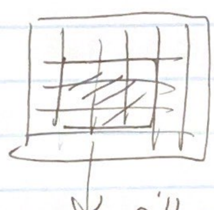
- Transformers — a perm inv't sequence NN.

→ Plan for next few lectures + (detour into Astro w/ CNNs)

Convolutional NNs

Originally designed for natural images (MNIST, ImageNet, ...)
Work well for jet tagging too!

Idea: learn "filters" that encode key features of images
"feature detector" (eg noses, eyebrows, wheels, ...)
(or jet substructure, ...)



Input
Data structured as 2d array instead of 1d vector

x_{ij} $\quad i,j = 1, \dots N_p$

$m \times n$ filter $W_{\alpha\beta}$ $\quad \alpha = 1, \dots m$, $\beta = 1, \dots n$

biases

$$A\left(\sum_{\beta=1}^{n}\sum_{\alpha=1}^{m} W_{\alpha\beta}\, x_{i+\alpha,\, j+\beta} + b_{ij}\right) = x^{(1)}_{ij}$$

hidden layer is also an image!

can repeat $\rightarrow$ "convolutional layer"

- each layer can have multiple filters $W^{(f)}_{\alpha\beta}$ $\quad f = 1, \dots N_f$

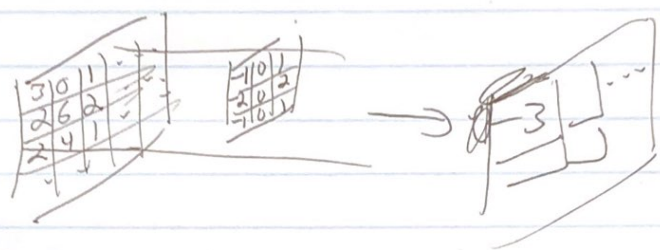
$\longrightarrow x^{(1)}_{ijf}$ is $N_p \times N_p \times N_f$ ← "channels" like colors.

$$A\left(\sum_{\alpha,\beta,f} W_{\alpha\beta f}\, x_{i+\alpha,\, j+\beta,\, f} + b_{ij}\right) \longrightarrow x^{(1)}_{ijf}$$

then need filters $W_{\alpha\beta f}$ ← not convolved over channels just summed

- Also ~~works for~~ applies to input — channels = colors (eg RGB)

- Example 0:



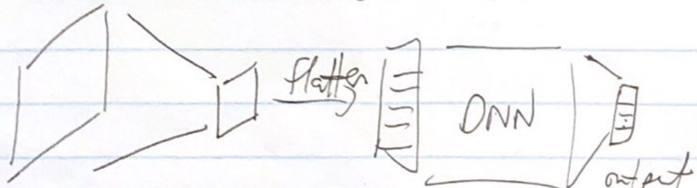
- This is the basic idea of CNN layer
  - many details related to boundaries — padding or no padding
    - image gets smaller w/ each layer

- Need some way to reduce dimensionality / aggregate info

  → "pooling layer"

$$\begin{array}{|c|c|}\hline a & b \\ \hline c & d \\ \hline\end{array} \rightarrow \max(a,b,c,d) \text{ or } \mathrm{avg}(a,b,c,d)$$

example of 2x2 pooling
reduces image by 2x.



conv layers
+pooling

classic CNN architecture

idea: successive layers learning layer, higher level features

edges etc → noses, eyes → fac