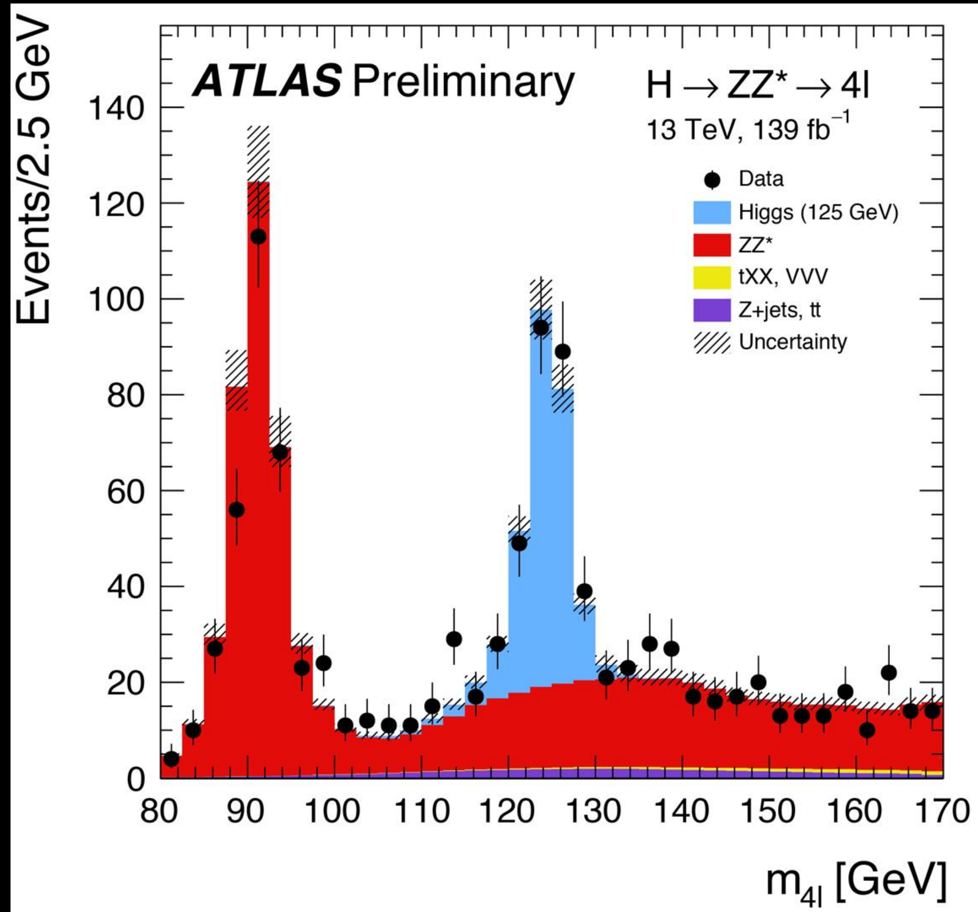
Machine Learning
Day 1 Exercises

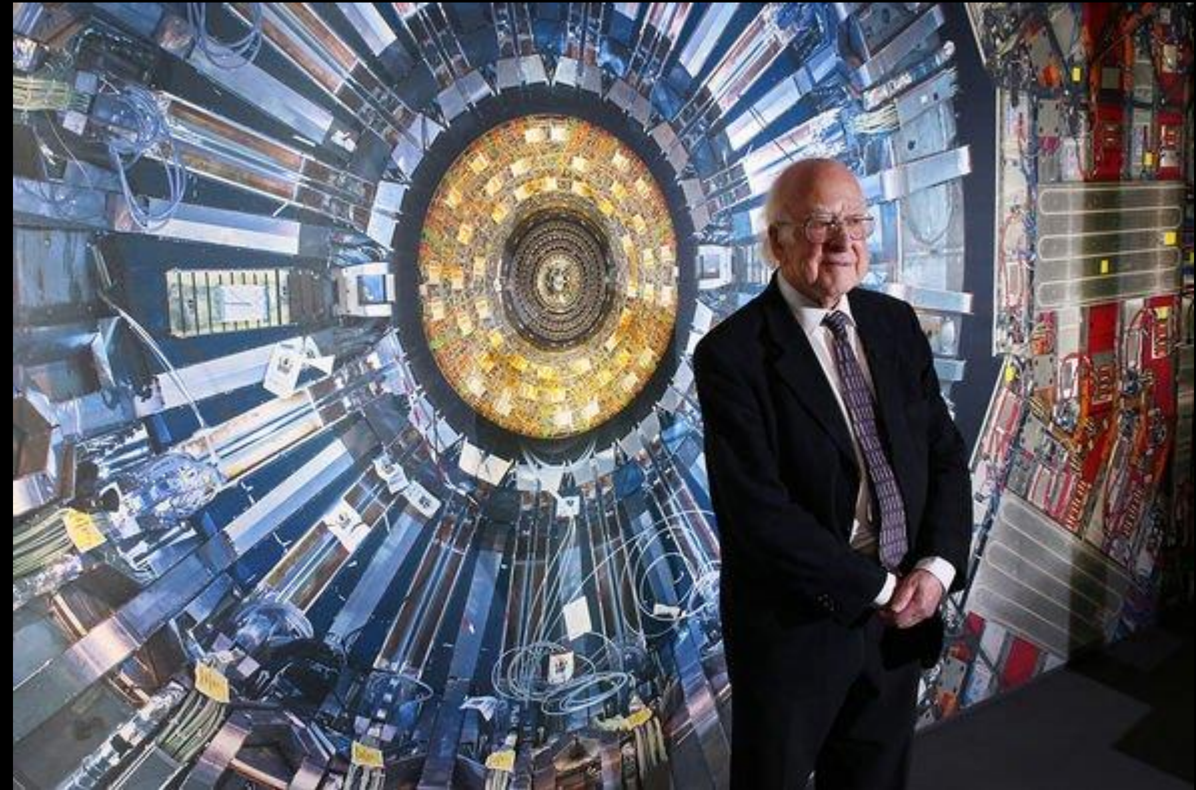
C. Tunnell (Rice) G. Watts (University of Washington/Seattle)

Start your binder!!!

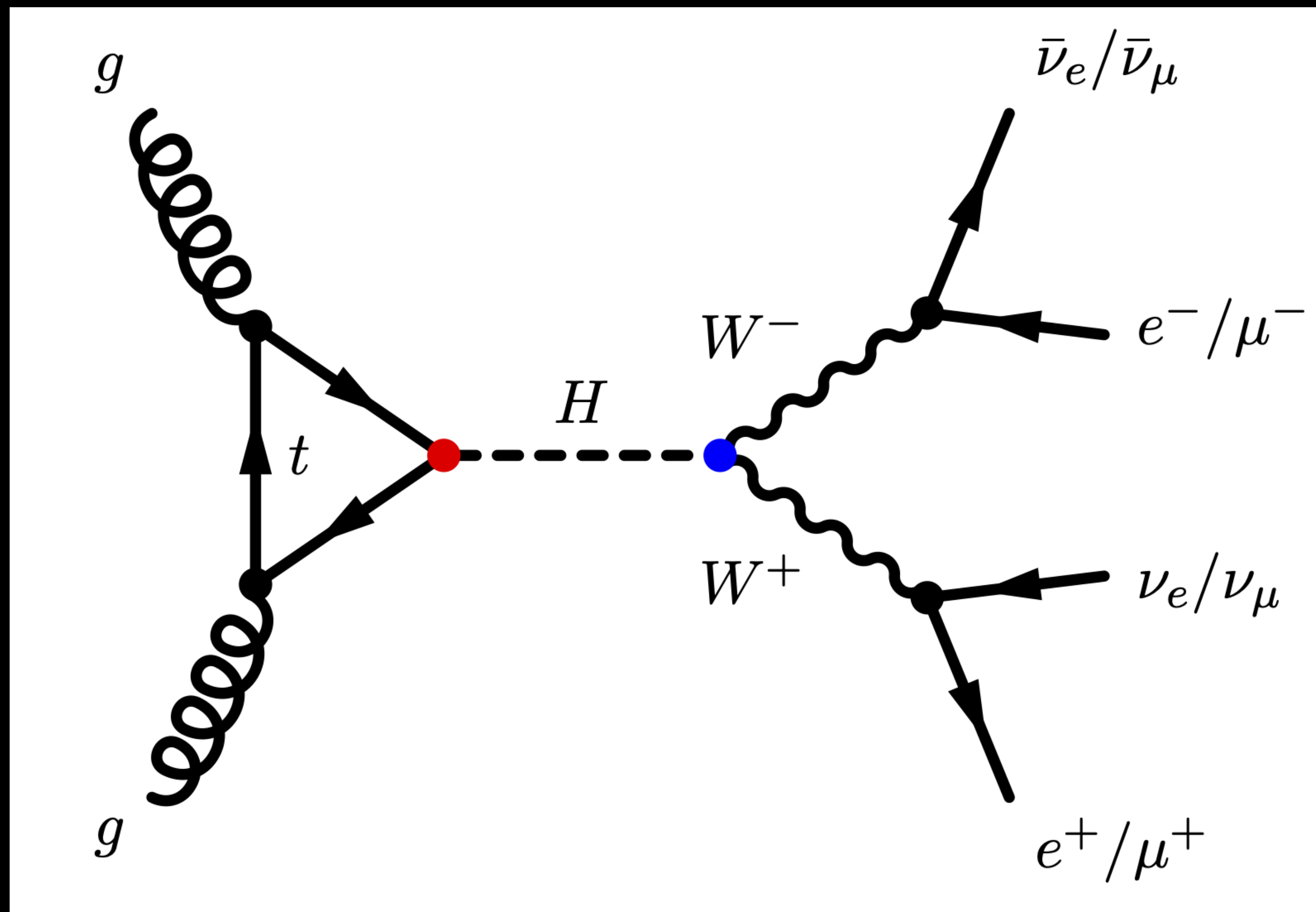
(linked from the agenda)

The Famous $H \rightarrow WW$ Discovery Plot

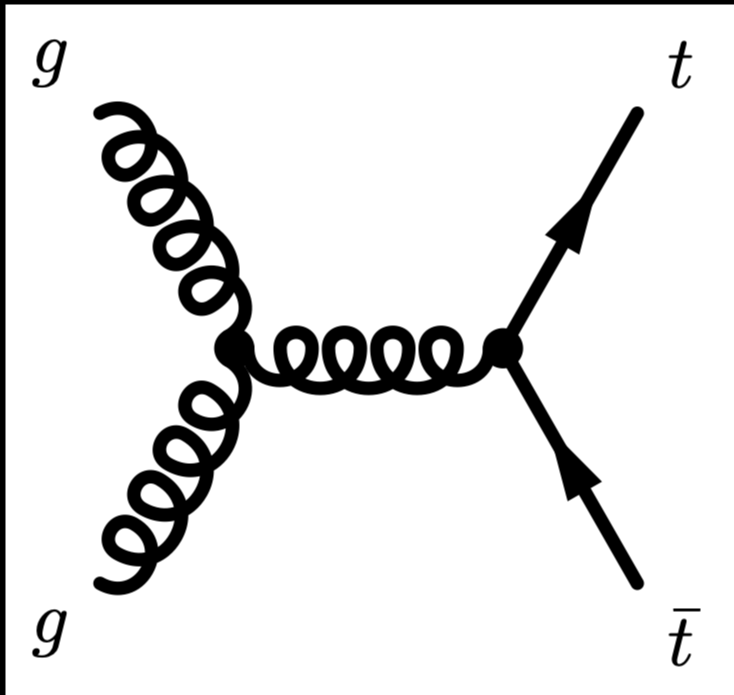


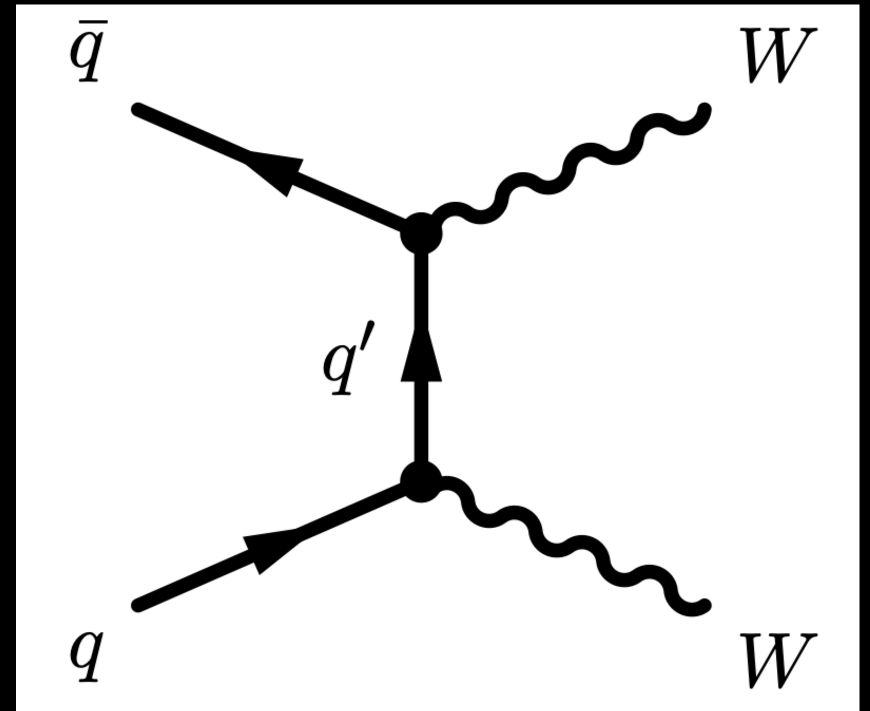


$H \to WW \to 2\ell 2\nu$



Large Backgrounds



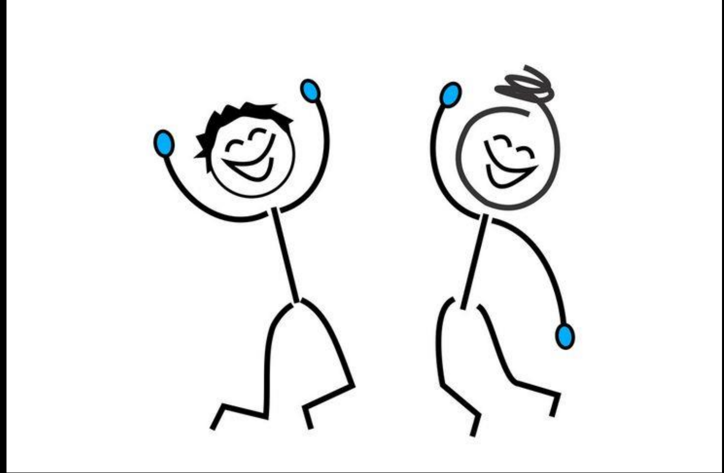


What are the differences?

Final states

HWW: $2\ell 2\nu$
$t\bar{t}: WbWb \rightarrow 2\ell 2\nu + 2b$
$WW: 2\ell 2\nu$

Pair Up!



Exercise…

1. Load the data
2. Look at the data
3. Build a network
4. Train
5. Adjust the network to improve the training…

The **same** binder/Jupyter server instance can be used for all today's work

No GPU required!

What kind of network should we create?

Simple multi-layer network (similar to earlier today)…

These are always good starting points…

This Layer…

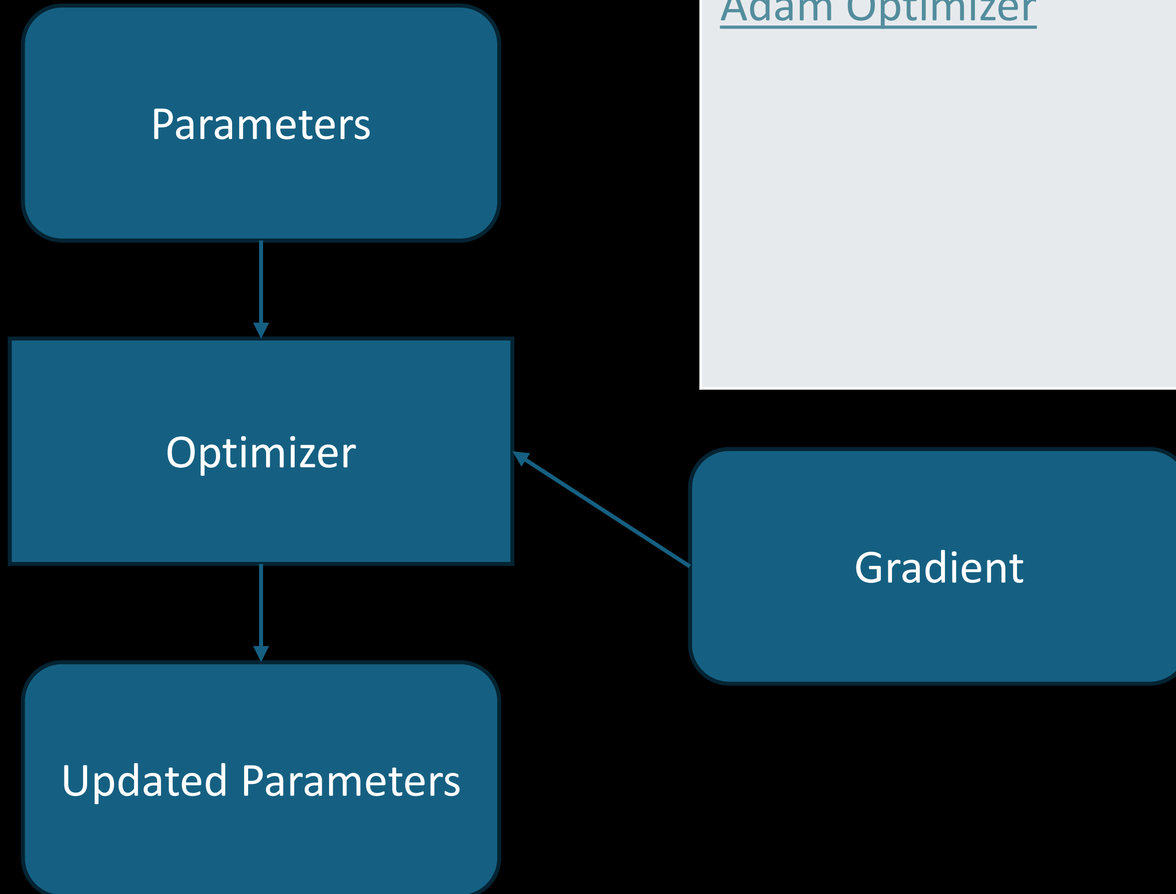
```
import jax
import jax.numpy as jnp
import jax.nn
import haiku as hk
from optax import adam, apply_updates

def net_fn(x):
    mlp = hk.Sequential([
        hk.Linear(12), jax.nn.relu,
        hk.Linear(60), jax.nn.relu,
        hk.Linear(32), jax.nn.relu,
        hk.Linear(1), jax.nn.sigmoid
    ])
    return mlp(x)

net = hk.transform(net_fn)
```

What is an optimizer in JAX?

We are using the optax library



Stocastic Gradient Descent	$\mu_t = g_t \times \alpha$
Adam Optimizer	Uses "momentum" to help adjust the parameters μ_t. "The scaling used for each parameter is computed from estimates of first and second-order moments of the gradients (using suitable exponential moving averages)."

Let's use the Adam optimizer, with a learning rate of 0.0005

Adam

```
optimizer = optax.adam(0.0005)
```

Loss Functions

Optax also has a huge number of loss functions to help us get started

`ntxent`(embeddings, labels[, temperature])	Normalized temperature scaled cross entropy loss (NT-Xent).
`safe_softmax_cross_entropy`(logits, labels)	Computes the softmax cross entropy between sets of logits and labels.
`sigmoid_binary_cross_entropy`(logits, labels)	Computes element-wise sigmoid cross entropy given logits and labels.
`sigmoid_focal_loss`(logits, labels[, alpha, ...])	Sigmoid focal loss.
`smooth_labels`(labels, alpha)	Apply label smoothing.
`softmax_cross_entropy`(logits, labels)	Computes the softmax cross entropy between sets of logits and labels.
`softmax_cross_entropy_with_integer_labels`(...)	Computes softmax cross entropy between sets of logits and integer labels.
`squared_error`(predictions[, targets])	Calculates the squared error for a set of predictions.

↑ Back to top

Loss Function

This is a classification problem – binary cross entropy!

But which one? Take a look and a guess before we go on…

Loss Function

`optax.losses.sigmoid_binary_cross_entropy(logits, labels)` [source]

Computes element-wise sigmoid cross entropy given logits and labels.

This function can be used for binary or multiclass classification (where each class is an independent binary prediction and different classes are not mutually exclusive e.g. predicting that an image contains both a cat and a dog.)

Because this function is overloaded, please ensure your *logits* and *labels* are compatible with each other. If you're passing in binary *labels* (values in {0, 1}), ensure your *logits* correspond to class 1 only. If you're passing in per-class target probabilities or one-hot *labels*, please ensure your *logits* are also multiclass. Be particularly careful if you're relying on implicit broadcasting to reshape *logits* or *labels*.

References

[Goodfellow et al, 2016](http://www.deeplearningbook.org/contents/prob.html)

Parameters:

- **logits** – Each element is the unnormalized log probability of a binary prediction. See note about compatibility with *labels* above.
- **labels** – Binary labels whose values are {0,1} or multi-class target probabilities. See note about compatibility with *logits* above.

Returns:

cross entropy for each binary prediction, same shape as *logits*.

Loss Function

```python
from optax import sigmoid_binary_cross_entropy

@jax.jit
def loss_fn(params, x, y):
    preds = net.apply(params, rng, x)
    # This next line provides a x10 speed up.
    preds = preds.reshape(-1)
    sm_array = sigmoid_binary_cross_entropy(preds, y)
    return jnp.mean(sm_array)
```

Training Step

```python
@jax.jit
def train_step(params, opt_state, x, y):
    grads = jax.grad(loss_fn)(params, x, y)
    updates, opt_state = optimizer.update(grads, opt_state)
    new_params = apply_updates(params, updates)
    return new_params, opt_state
```

Training Loop

```
starting_time = time.time()

print("starting training...")

if len(losses_training) == 0:
    # Prime the loss arrays.
    losses_training.append(loss_fn(params, X_train[:n_loss], y_train[:n_loss]))
    losses_test.append(loss_fn(params, X_test[:n_loss], y_test[:n_loss]))

for epoch in tqdm(range(1000)):
    params, opt_state = train_step(params, opt_state, X_train, y_train)
    losses_training.append(loss_fn(params, X_train[:n_loss], y_train[:n_loss]))
    losses_test.append(loss_fn(params, X_test[:n_loss], y_test[:n_loss]))

print("done training...")

training_time = time.time() - starting_time
print("Training time:", training_time)
```

Challenge!!

1. Get the lowest loss training you can!
2. Tomorrow (Friday), as lunch starts, I'll release the *https* location of independent data file on Slack.
3. You run your pre-trained NN against that file and see who gets the lowest loss!
    1. Post your loss number of Slack
4. We'll want to see your code and a demonstration of your training!
    1. No training on the new file tomorrow morning!

Completely done tutorial…

Final Version of the data