Neural Networks Lecture 1, part 2:

A BSM Search Analysis demonstration using simple ‘no black box’ code

for the
“Beyond Standard Model”
RTG

7th April, 2017

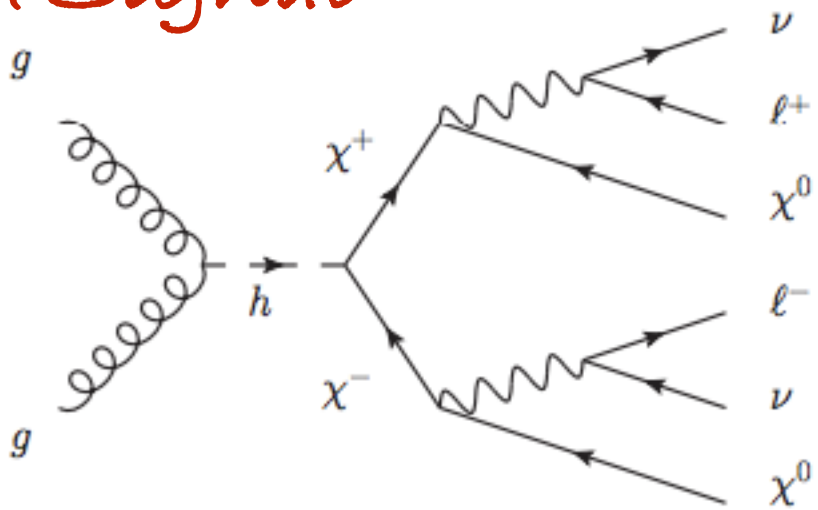
Searching for Exotic Particles in High-Energy Physics with Deep Learning

Pierre Baldi, Peter Sadowski (UC, Irvine) , Daniel Whiteson (UC, Irvine & Pennsylvania U. & UC, Irvine)

Feb 19, 2014 - 9 pages

Nature Commun. 5 (2014) 4308

BSM Signal

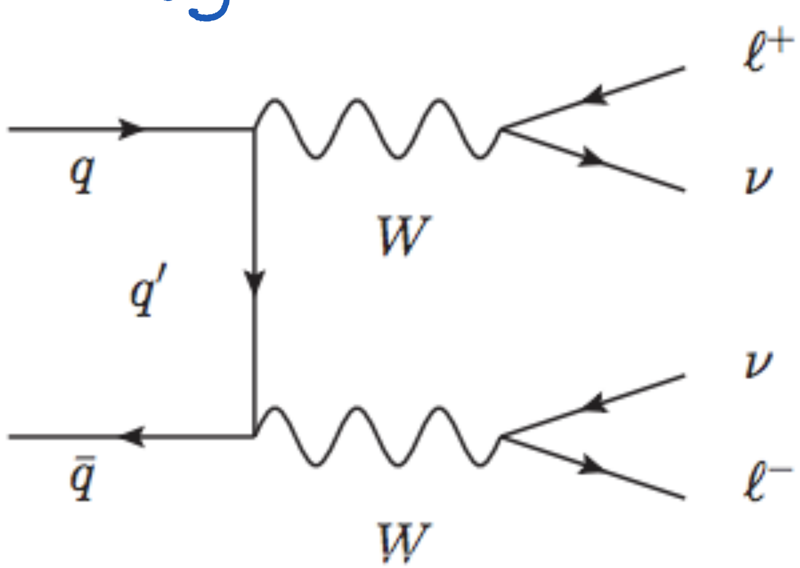


(a)

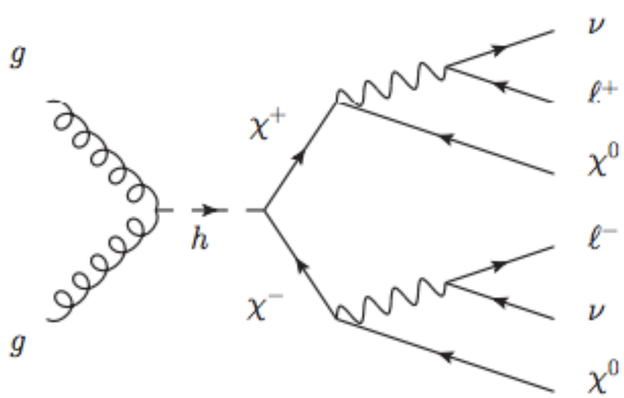
Signature

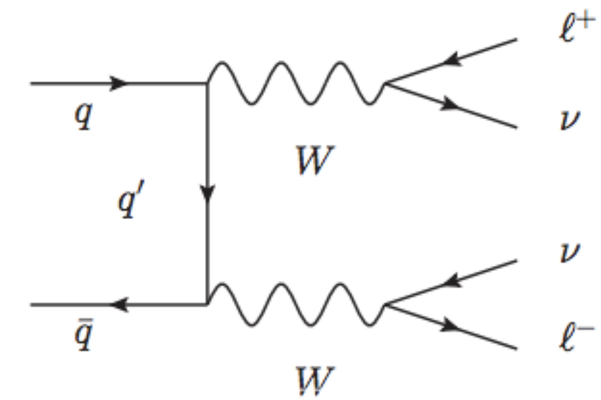
MET + $\ell^{-}\ell^{+}$

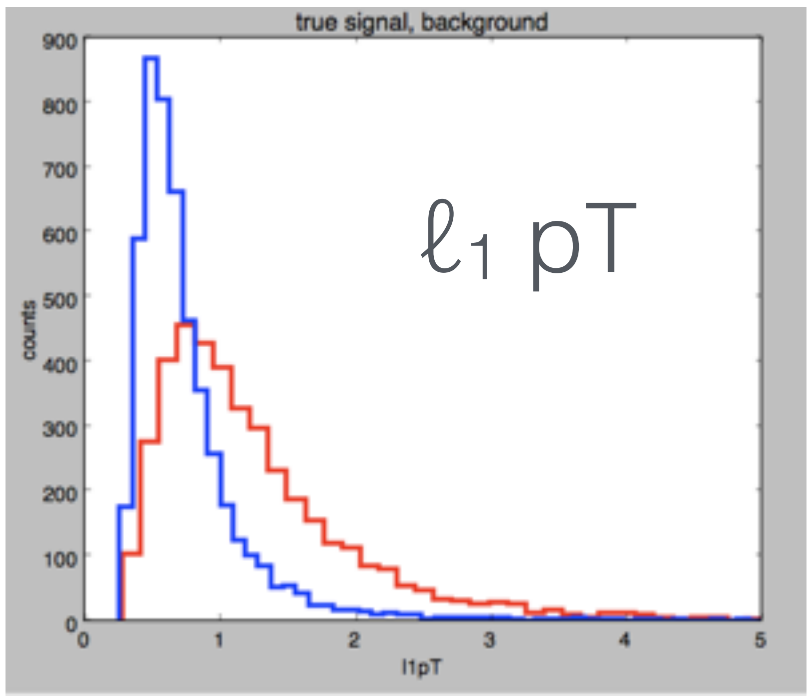
SM Background

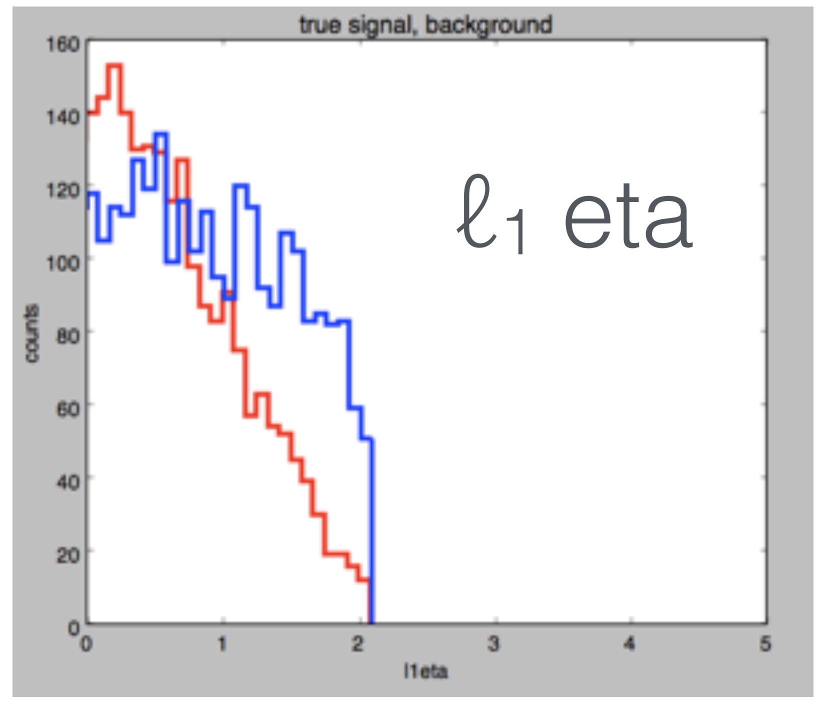


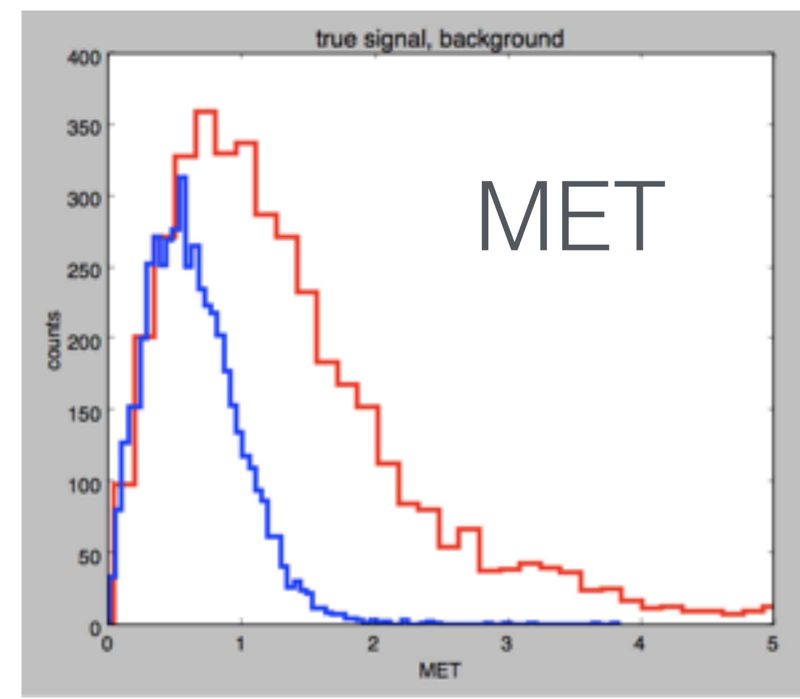
Data*
raw features

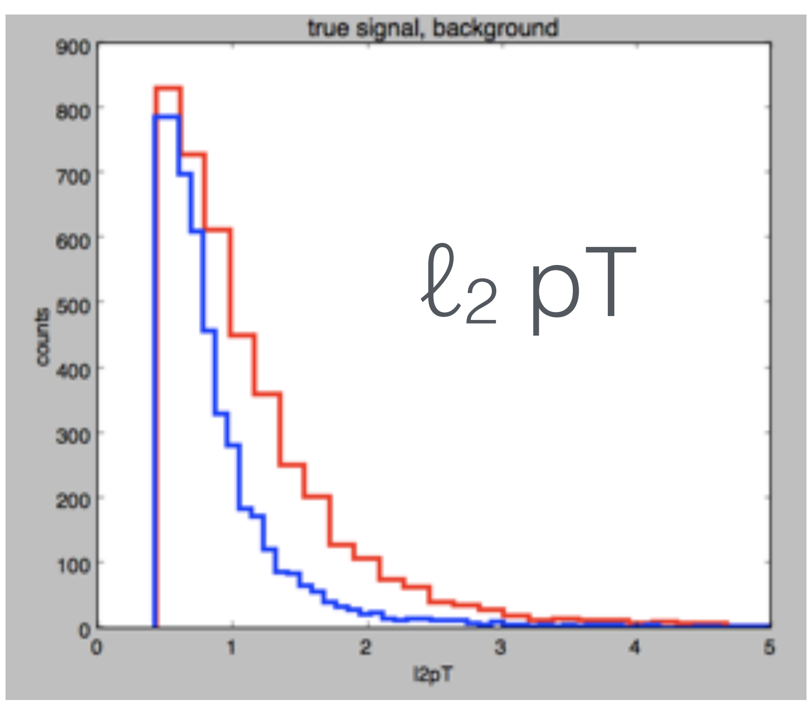


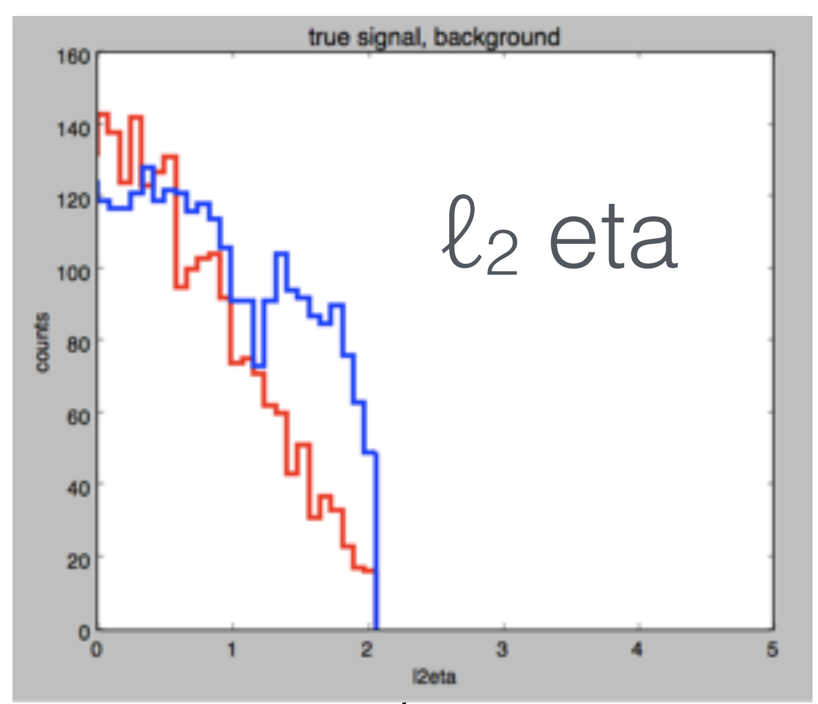












* standardized

Code

*   *   *
[# nodes1, # nodes2, #nodes3,…]

```
class Network(object):

    def __init__(self, sizes, fname):
        self.num_layers*= len(sizes)
        self.sizes = sizes
        self.biases = [np.random.randn(y, 1) for y in sizes[1:]]
        #self.weights = [np.random.randn(y, x)
        #                 for x, y in zip(sizes[:-1], sizes[1:])]
        self.weights = [np.random.randn(y, x)/np.sqrt(x)
                        for x, y in zip(self.sizes[:-1], self.sizes[1:])]
        self.f = open(fname, 'w')
```

initialization
of network

***hyper parameters!**

```
    def SGD(self, training_data, epochs*, mini_batch_size*, eta*,
            test_data=None):
        if test_data: n_test = len(test_data)
        n = len(training_data)
        for j in xrange(epochs):
            random.shuffle(training_data)
            mini_batches = [
                training_data[k:k+mini_batch_size]
                for k in xrange(0, n, mini_batch_size)]
            for mini_batch in mini_batches:
                self.update_mini_batch(mini_batch, eta)
```

splitting of data into
epochs and mini batches

entry point to
weights and bias updates…

Code

```python
def SGD(self, training_data, epochs, mini_batch_size, eta,
        test_data=None):
    if test_data: n_test = len(test_data)
    n = len(training_data)
    for j in xrange(epochs):
        random.shuffle(training_data)
        mini_batches = [
            training_data[k:k+mini_batch_size]
            for k in xrange(0, n, mini_batch_size)]
        for mini_batch in mini_batches:
            self.update_mini_batch(mini_batch, eta)
```

```python
def update_mini_batch(self, mini_batch, eta):
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
    self.weights = [w-(eta/len(mini_batch))*nw
                    for w, nw in zip(self.weights, nabla_w)]
    self.biases = [b-(eta/len(mini_batch))*nb
                   for b, nb in zip(self.biases, nabla_b)]
```

functions

```python
def cost_derivative(self, output_activations, y):
    return (output_activations-y)
```

```python
def sigmoid(z):
    """The sigmoid function."""
    return 1.0/(1.0+np.exp(-z))

def sigmoid_prime(z):
    """Derivative of the sigmoid function."""
    return sigmoid(z)*(1-sigmoid(z))
```

Code

```
def SGD(self, training_data, epochs, mini_batch_size, eta,
        test_data=None):
    if test_data: n_test = len(test_data)
    n = len(training_data)
    for j in xrange(epochs):
        random.shuffle(training_data)
        mini_batches = [
            training_data[k:k+mini_batch_size]
            for k in xrange(0, n, mini_batch_size)]
        for mini_batch in mini_batches:
            self.update_mini_batch(mini_batch, eta)
```

```
def update_mini_batch(self, mini_batch, eta):
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    for x, y in mini_batch:
        delta_nabla_b, delta_nabla_w = self.backprop(x, y)
        nabla_b = [nb+dnb for nb, dnb in zip(nabla_b, delta_nabla_b)]
        nabla_w = [nw+dnw for nw, dnw in zip(nabla_w, delta_nabla_w)]
    self.weights = [w-(eta/len(mini_batch))*nw
                    for w, nw in zip(self.weights, nabla_w)]
    self.biases = [b-(eta/len(mini_batch))*nb
                   for b, nb in zip(self.biases, nabla_b)]
```

```
def backprop(self, x, y):
    nabla_b = [np.zeros(b.shape) for b in self.biases]
    nabla_w = [np.zeros(w.shape) for w in self.weights]
    # feedforward
    activation = x
    activations = [x] # list to store all the activations, layer by layer
    zs = [] # list to store all the z vectors, layer by layer
    for b, w in zip(self.biases, self.weights):
        z = np.dot(w, activation)+b
        zs.append(z)
        activation = sigmoid(z)
        activations.append(activation)
    # backward pass
    delta = self.cost_derivative(activations[-1], y) * \
        sigmoid_prime(zs[-1])
    nabla_b[-1] = delta
    nabla_w[-1] = np.dot(delta, activations[-2].transpose())
    for l in xrange(2, self.num_layers):
        z = zs[-l]
        sp = sigmoid_prime(z)
        delta = np.dot(self.weights[-l+1].transpose(), delta) * sp
        nabla_b[-l] = delta
        nabla_w[-l] = np.dot(delta, activations[-l-1].transpose())
    return (nabla_b, nabla_w)
```

functions

```
def cost_derivative(self, output_activations, y):
    return (output_activations-y)
```

```
def sigmoid(z):
    """The sigmoid function."""
    return 1.0/(1.0+np.exp(-z))

def sigmoid_prime(z):
    """Derivative of the sigmoid function."""
    return sigmoid(z)*(1-sigmoid(z))
```

References

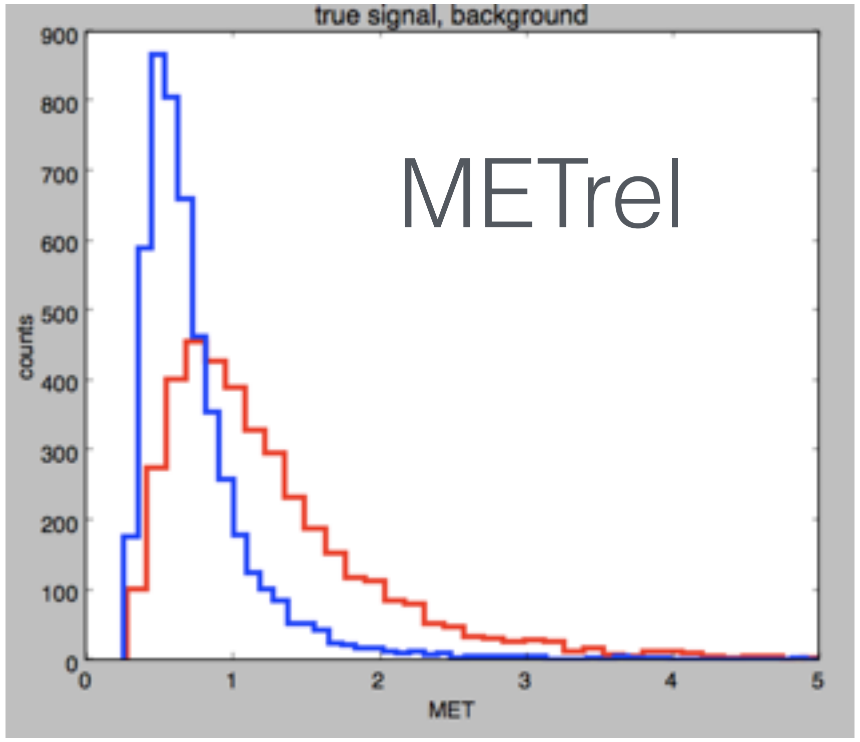
- **"Searching for Exotic Particles in High-Energy Physics with Deep Learning"**, P. Baldi, P. Sadowski, and D. Whiteson (2014): https://arxiv.org/pdf/1402.4735.pdf
    - github: https://github.com/uci-igb/higgs-susy
    - dataset: http://archive.ics.uci.edu/ml/datasets/HIGGS
- **online reads:**
    - online book "Neural Networks and Deep Learning" by Michael Nielsen
        - http://neuralnetworksanddeeplearning.com/
        - https://github.com/mnielsen/neural-networks-and-deep-learning.git
- **python package**
    - pylearn2 (used by P. Baldi et al.)
        - git://github.com/lisa-lab/pylearn2.git
        - Warning: no longer being developed
    - Scikit-learn: best way to get started
    - Keras
        - GPU processing
        - high level deep learning library for Theano/TensorFlow

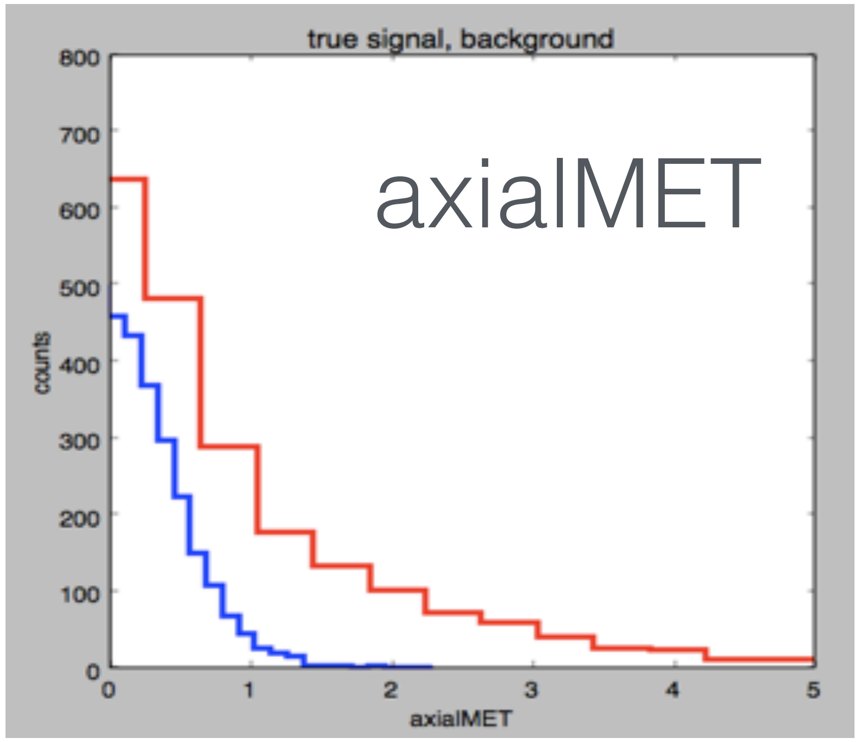
Back-up

SUSY physics case:

Data*

derived features





…and many more ‘hidden’ correlations (see paper).

* standardized

2nd physics case:
BSM Higgs

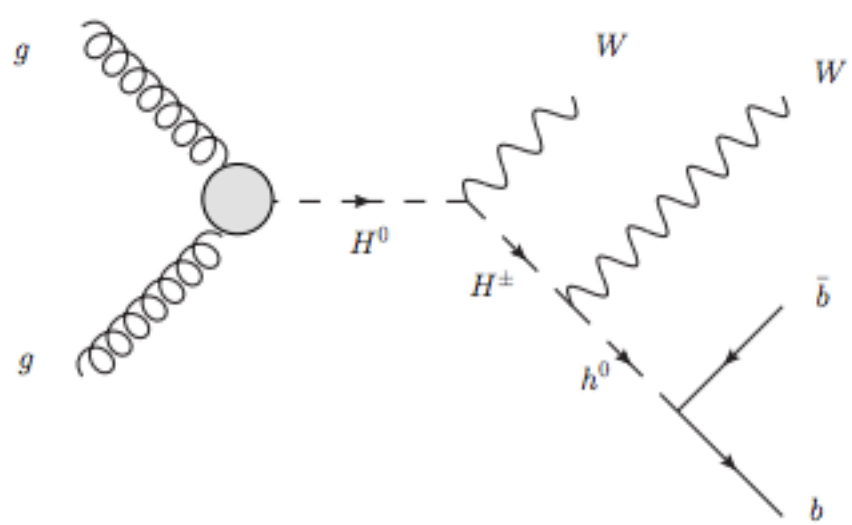
Searching for Exotic Particles in High-Energy Physics with Deep Learning

Pierre Baldi, Peter Sadowski (UC, Irvine) , Daniel Whiteson (UC, Irvine & Pennsylvania U. & UC, Irvine)

Feb 19, 2014 - 9 pages

Nature Commun. 5 (2014) 4308

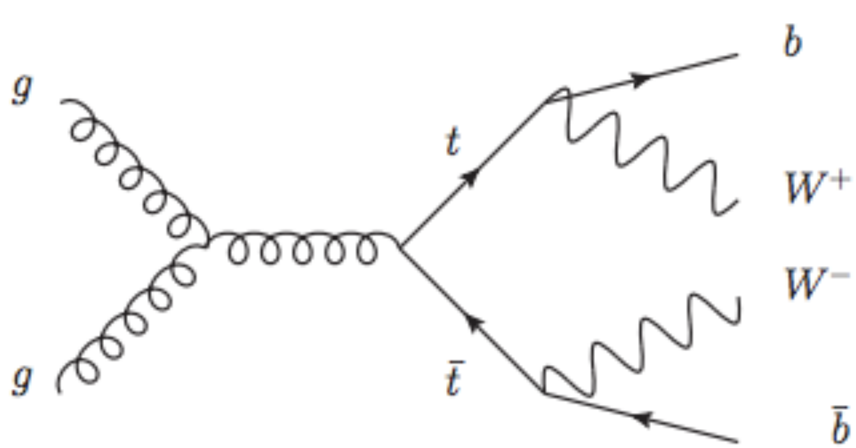
BSM Signal



(a)

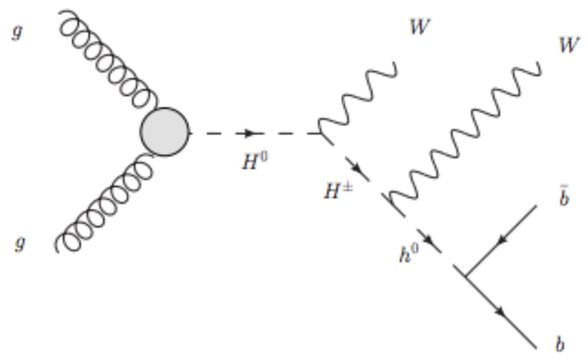
Signature
$W^+W^- b\bar{b}$

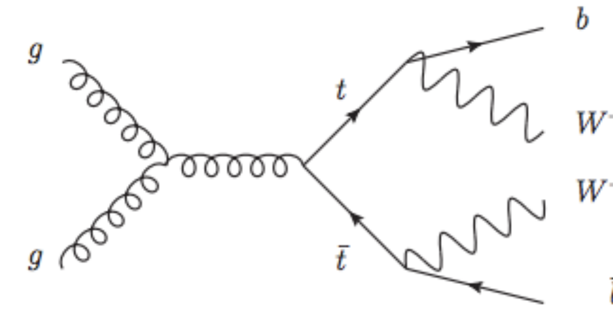
SM Background

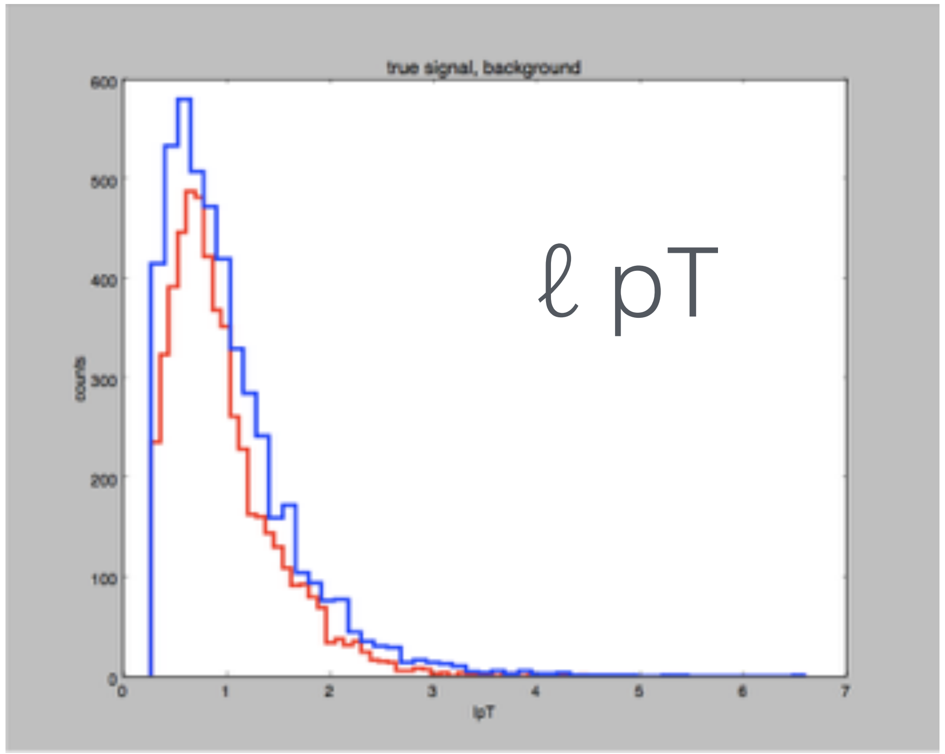


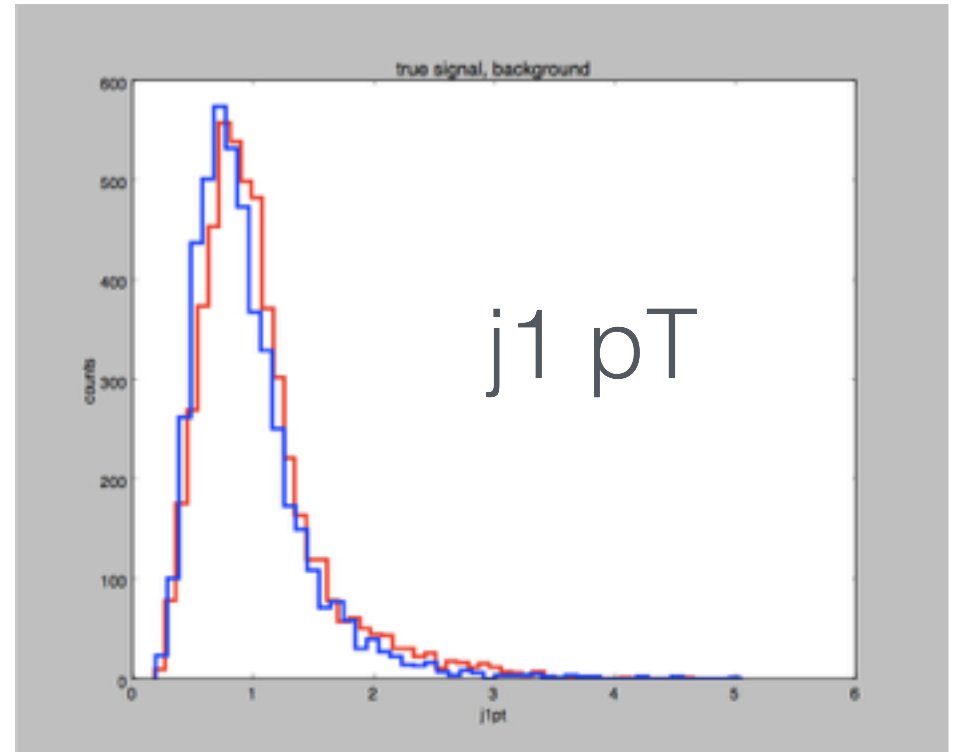
Data*

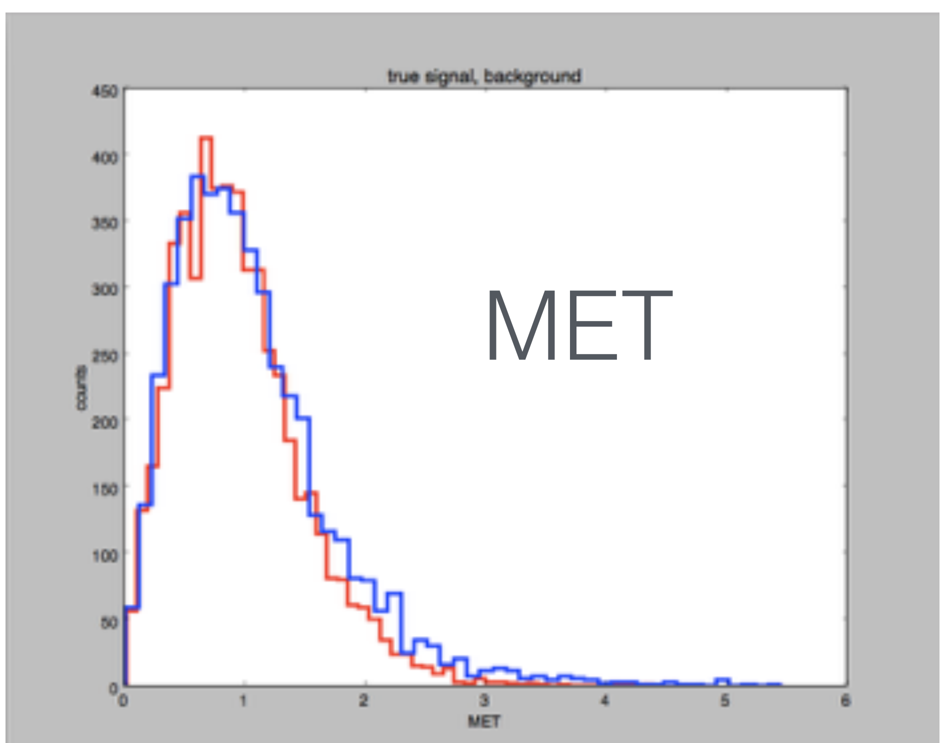
features

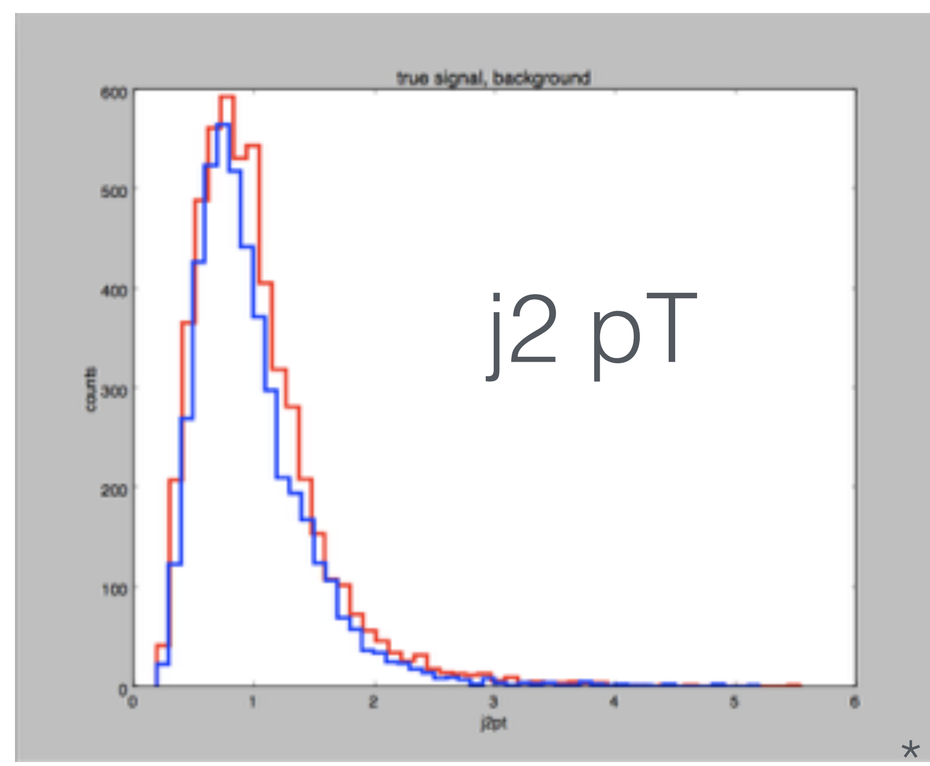




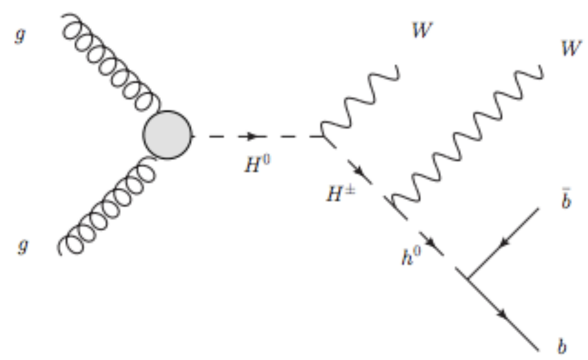






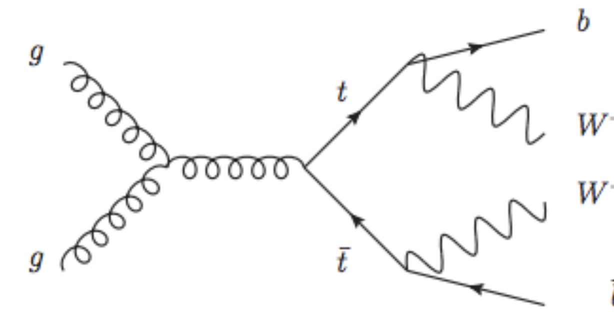


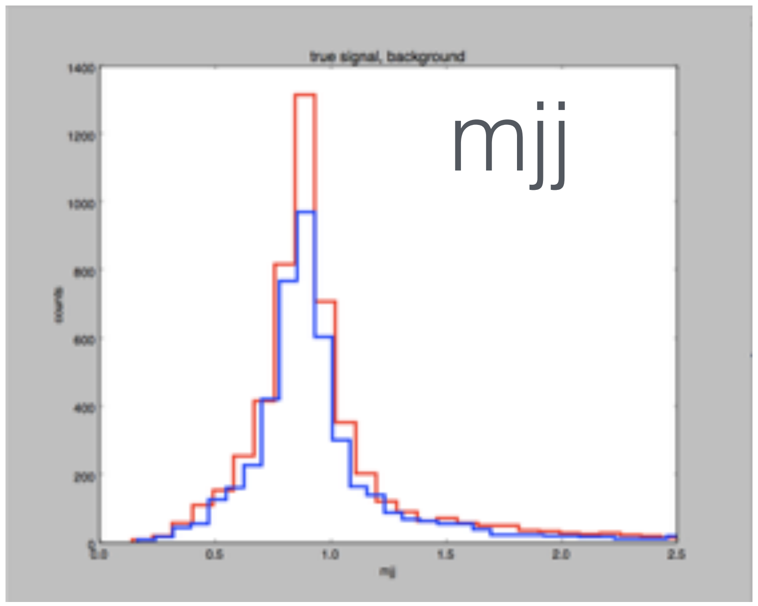
* standardized

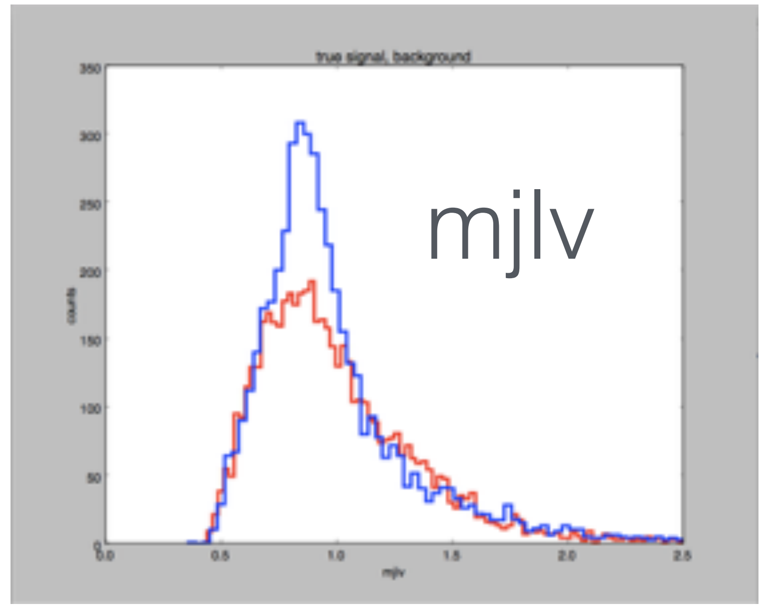


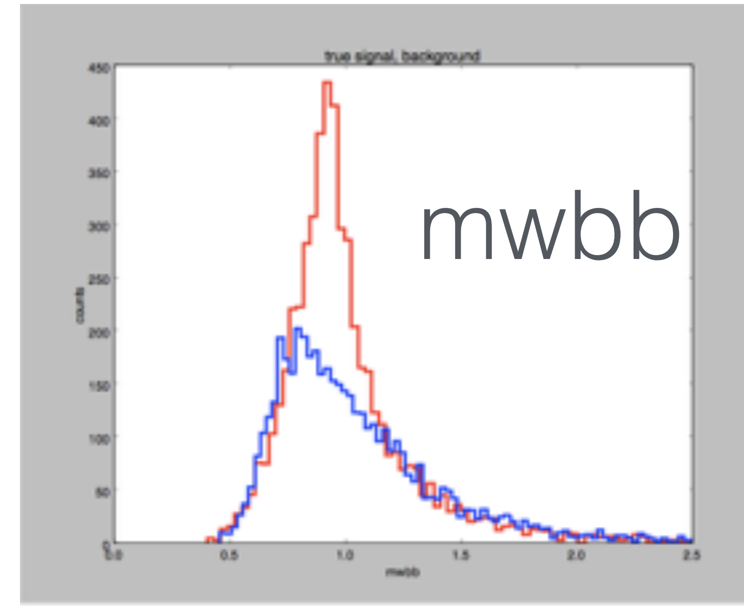
Data*

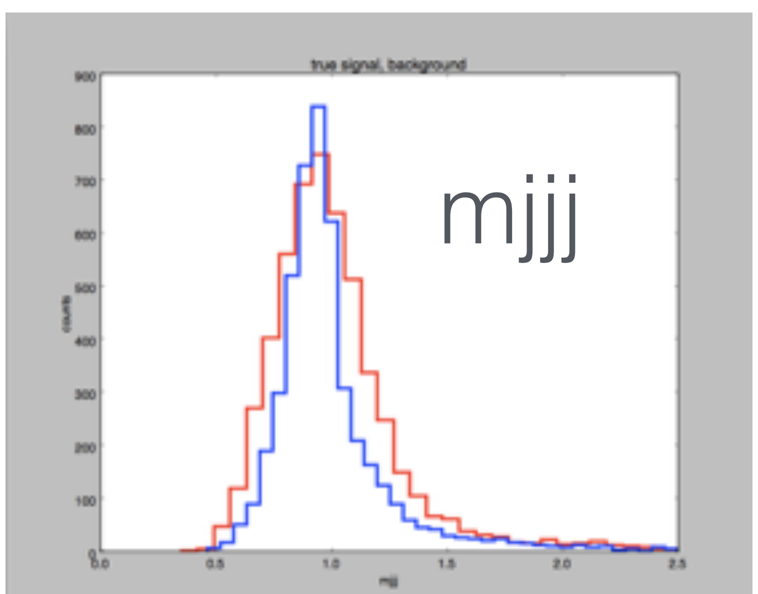
derived features

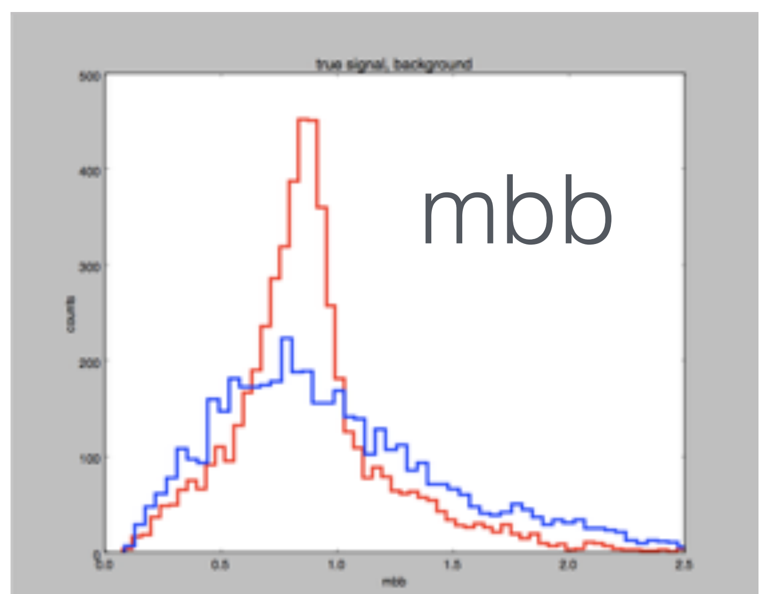


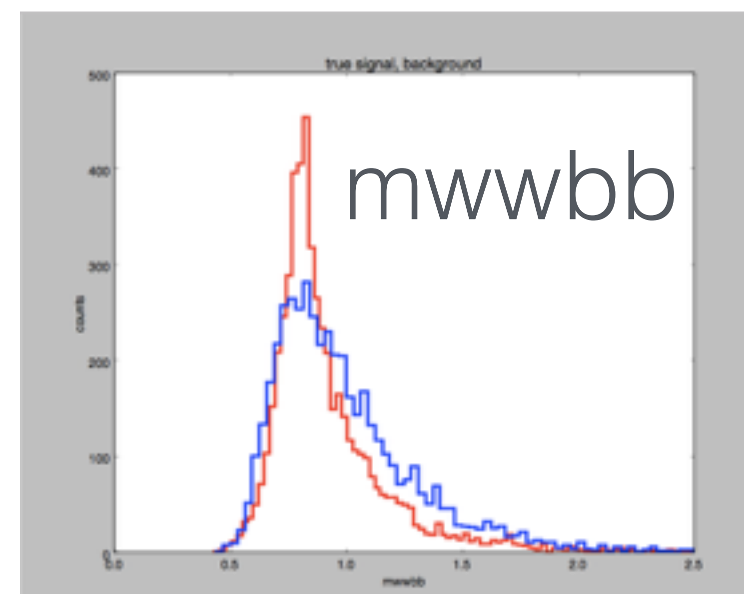












* standardized

```python
def feedforward(self, a):
    for b, w in zip(self.biases, self.weights):
        a = sigmoid(np.dot(w, a)+b)
    return a

def evaluate(self, test_data):
    test_results = [(np.argmax(self.feedforward(x)), y)
                    for (x, y) in test_data]
    return sum(int(x == y) for (x, y) in test_results)
```