tf.nn.weighted_cross_entropy_with_logits

```
weighted_cross_entropy_with_logits(
    targets,
    logits,
    pos_weight,
    name=None
)
```

Defined in `tensorflow/python/ops/nn_impl.py`.

See the guide: Neural Network > Classification

Computes a weighted cross entropy.

This is like `sigmoid_cross_entropy_with_logits()` except that `pos_weight`, allows one to trade off recall and precision by up- or down-weighting the cost of a positive error relative to a negative error.

The usual cross-entropy cost is defined as:

```
targets * -log(sigmoid(logits)) +
    (1 - targets) * -log(1 - sigmoid(logits))
```

The argument `pos_weight` is used as a multiplier for the positive targets:

```
targets * -log(sigmoid(logits)) * pos_weight +
    (1 - targets) * -log(1 - sigmoid(logits))
```

For brevity, let `x = logits`, `z = targets`, `q = pos_weight`. The loss is:

```
  qz * -log(sigmoid(x)) + (1 - z) * -log(1 - sigmoid(x))
= qz * -log(1 / (1 + exp(-x))) + (1 - z) * -log(exp(-x) / (1 + exp(-x)))
= qz * log(1 + exp(-x)) + (1 - z) * (-log(exp(-x)) + log(1 + exp(-x)))
= qz * log(1 + exp(-x)) + (1 - z) * (x + log(1 + exp(-x))
= (1 - z) * x + (qz +  1 - z) * log(1 + exp(-x))
= (1 - z) * x + (1 + (q - 1) * z) * log(1 + exp(-x))
```

Setting `l = (1 + (q - 1) * z)`, to ensure stability and avoid overflow, the implementation uses

```
(1 - z) * x + l * (log(1 + exp(-abs(x))) + max(-x, 0))
```

`logits` and `targets` must have the same type and shape.

Args:

- `targets`: A `Tensor` of the same type and shape as `logits`.
- `logits`: A `Tensor` of type `float32` or `float64`.
- `pos_weight`: A coefficient to use on the positive examples.
- `name`: A name for the operation (optional).

Returns:

A `Tensor` of the same shape as `logits` with the componentwise weighted logistic losses.

Raises:

- `ValueError`: If `logits` and `targets` do not have the same shape.
