Deep Neural Networks

Project week9:
Adding BatchNormalization

Initial model: 6 classes, not too complex

```
model.summary()
```

```
Model: "sequential_13"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_11 (ZeroPaddi (None, 226, 226, 3)       0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 224, 224, 16)      448
_________________________________________________________________
max_pooling2d_30 (MaxPooling (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_31 (Conv2D)           (None, 110, 110, 32)      4640
_________________________________________________________________
max_pooling2d_31 (MaxPooling (None, 109, 109, 32)      0
_________________________________________________________________
flatten_13 (Flatten)         (None, 380192)            0
_________________________________________________________________
dropout_13 (Dropout)         (None, 380192)            0
_________________________________________________________________
dense_13 (Dense)             (None, 6)                 2281158
=================================================================
Total params: 2,286,246
Trainable params: 2,286,246
Non-trainable params: 0
_________________________________________________________________
```

Training accuracy: 0.83
Validation accuracy: 0.88
Test accuracy: 0.83

More layers

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_14 (ZeroPaddi (None, 226, 226, 3)       0
_________________________________________________________________
conv2d_38 (Conv2D)           (None, 224, 224, 16)      448
_________________________________________________________________
max_pooling2d_38 (MaxPooling (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_39 (Conv2D)           (None, 110, 110, 32)      4640
_________________________________________________________________
max_pooling2d_39 (MaxPooling (None, 109, 109, 32)      0
_________________________________________________________________
conv2d_40 (Conv2D)           (None, 107, 107, 16)      4624
_________________________________________________________________
max_pooling2d_40 (MaxPooling (None, 53, 53, 16)        0
_________________________________________________________________
flatten_16 (Flatten)         (None, 44944)             0
_________________________________________________________________
dropout_16 (Dropout)         (None, 44944)             0
_________________________________________________________________
dense_16 (Dense)             (None, 6)                 269670
=================================================================
Total params: 279,382
Trainable params: 279,382
Non-trainable params: 0
_________________________________________________________________
```

+ Code + Markdown

Training accuracy: 0.97
Validation accuracy: 0.97
Test accuracy: 0.98

Adding BatchNormalization

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
zero_padding2d_16 (ZeroPaddi (None, 226, 226, 3)       0
_________________________________________________________________
conv2d_44 (Conv2D)           (None, 224, 224, 16)      448
_________________________________________________________________
batch_normalization_17 (Batc (None, 224, 224, 16)      64
_________________________________________________________________
max_pooling2d_44 (MaxPooling (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_45 (Conv2D)           (None, 110, 110, 32)      4640
_________________________________________________________________
batch_normalization_18 (Batc (None, 110, 110, 32)      128
_________________________________________________________________
max_pooling2d_45 (MaxPooling (None, 109, 109, 32)      0
_________________________________________________________________
conv2d_46 (Conv2D)           (None, 107, 107, 16)      4624
_________________________________________________________________
batch_normalization_19 (Batc (None, 107, 107, 16)      64
_________________________________________________________________
max_pooling2d_46 (MaxPooling (None, 53, 53, 16)        0
_________________________________________________________________
flatten_18 (Flatten)         (None, 44944)             0
_________________________________________________________________
dropout_18 (Dropout)         (None, 44944)             0
_________________________________________________________________
dense_18 (Dense)             (None, 6)                 269670
=================================================================
Total params: 279,638
Trainable params: 279,510
Non-trainable params: 128
_________________________________________________________________
```

Training accuracy: 0.99
Validation accuracy: 0.44
Test accuracy: 0.46

Why is batch normalization reducing my model training accuracy?

Asked 1 year, 8 months ago Modified 1 year, 8 months ago Viewed 702 times

1 Answer

Sorted by: Highest score (default)

2

Batch Normalisation doesnt guarantee that your performance will increase. But it does work well in some cases.

One of things you can try to do is:

1. Increase the batch size of the training. This will give a more appropiate mean and standard deviation for normalisation.
2. Play around with the BN parameters, specifically the momentum parameter. See more here about the params https://keras.io/api/layers/normalization_layers/batch_normalization/ I would suggest to decrease the momentum and try again.
3. If it still doesnt work, leave it out.

Share Follow

answered Mar 4, 2021 at 8:26

s510

1,741 ●6 ●17

Parameters in BatchNormalization

BatchNormalization layer

BatchNormalization class

```
tf.keras.layers.BatchNormalization(
    axis=-1,
    momentum=0.99,
    epsilon=0.001,
    center=True,
    scale=True,
    beta_initializer="zeros",
    gamma_initializer="ones",
    moving_mean_initializer="zeros",
    moving_variance_initializer="ones",
    beta_regularizer=None,
    gamma_regularizer=None,
    beta_constraint=None,
    gamma_constraint=None,
    **kwargs
)
```

- https://keras.io/api/layers/normalization_layers/batch_normalization/

Recap: why BatchNormalization?

- Issues without BatchNormalization
  - Internal covariate shift
    - update of weights → slight changes → input distribution of every neuron is different at every epoch
    - in deep networks: add up fast, amplify greatly
    - → these neurons need to continuously adapt to the changing input distribution, meaning that their learning capabilities are severely bottlenecked
  - Vanishing and exploding gradients when using larger learning rates
- Advantages:
  - Accelerate deep network training by reducing internal covariate shift
  - can use larger learning rates
  - Reduces overfitting
    - regularising effect since it adds noise to the inputs of every layer

BatchNormalization during training vs. test phase

- During training:
  - Mean and standard deviation calculated using samples in the mini-batch
- During testing:
  - Does not make sense to calculate new values
  - → use a running mean and running variance that is calculated during training
  - Parameter: momentum

“momentum”

```
running_mean = momentum * running_mean + (1-momentum) * new_mean
running_var = momentum* running_var + (1-momentum) * new_var
```

Momentum is the importance given to the last seen mini-batch, a.k.a “lag”. If the momentum is set to 0, the running mean and variance come from the last seen mini-batch. However, this may be biased and not the desirable one for testing. Conversely, if momentum is set to 1, it uses the running mean and variance from the first mini-batch. Essentially, momentum controls how much each new mini-batch contributes to the running averages.

Ideally, the momentum should be set close to 1 (>0.9) to ensure slow learning of the running mean and variance such that the noise in a mini-batch is ignored.

Final model

[564]:

```
# define the keras model
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.ZeroPadding2D(),
        layers.Conv2D(16, kernel_size=(3, 3), activation="relu"),
        layers.BatchNormalization(momentum=0.8),
        layers.MaxPooling2D(pool_size=(2, 2), strides=2),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.BatchNormalization(momentum=0.8),
        layers.MaxPooling2D(pool_size=(2, 2), strides=1),
        layers.Conv2D(16, kernel_size=(3, 3), activation="relu"),
        layers.BatchNormalization(momentum=0.8),
        layers.MaxPooling2D(pool_size=(2, 2), strides=2),
        layers.Flatten(),
        #layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)
```

Training accuracy: 1
Validation accuracy: 0.99
Test accuracy: 0.99

- https://towardsdatascience.com/batch-normalisation-explained-5f4bd9de5feb?gi=21f16071fb92