Learning rate and regularization

Glenn Bruns
CSUMB

Much material in this deck from Géron, Hands-on Machine Learning with Scikit-Learn and TensorFlow

Learning outcomes

After this lecture you should be able to describe various methods for:

- learning rate scheduling in DNNs
- regularization in DNNs

Motivation

Were still addressing training problems in large DNNs:

- training can be slow
- with many layers and neurons per layer, the model can overfit

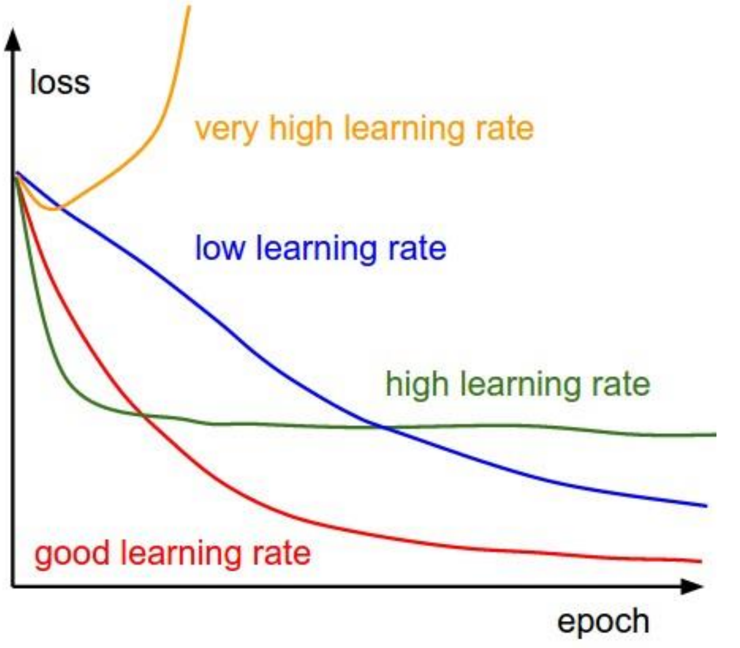
Learning rate

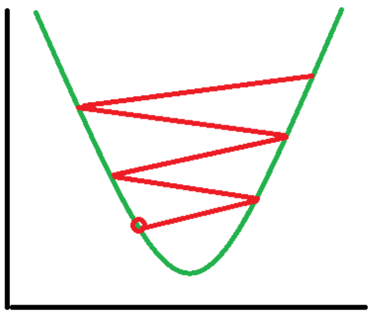
Learning rate too low →

takes too long to converge to optimum weights

Learning rate too high →

bounces around optimum, or diverges



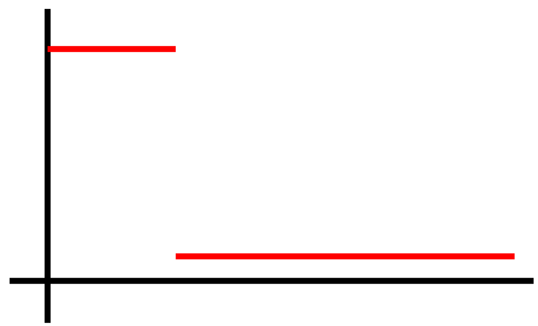


too high

Learning rate scheduling

Predetermined piecewise constant learning rate:

- reduce learning rate every so many epochs
- example: initially $\eta = 0.1$, then 0.001 after 50 epochs



Performance scheduling:

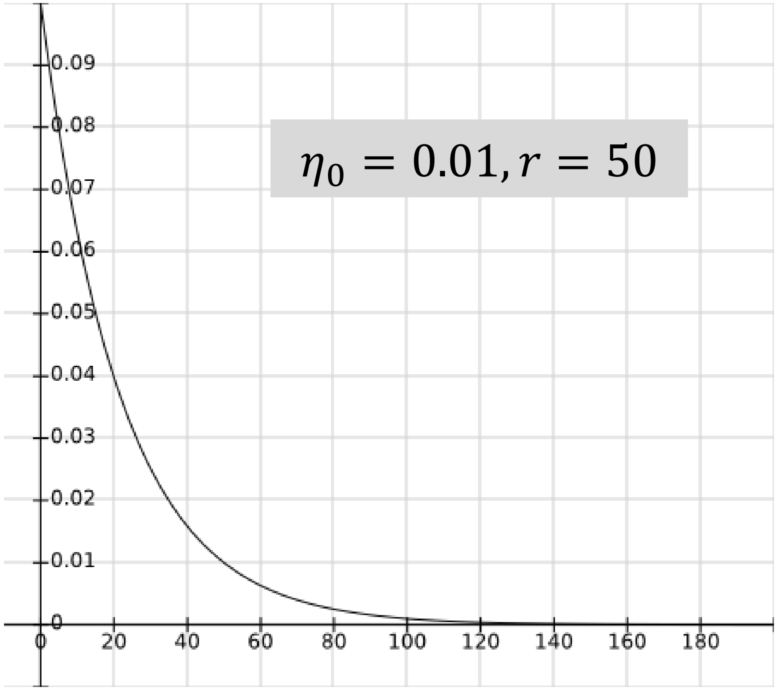
- measure test error every N steps
- reduce learning rate by a factor of λ when the error stops dropping

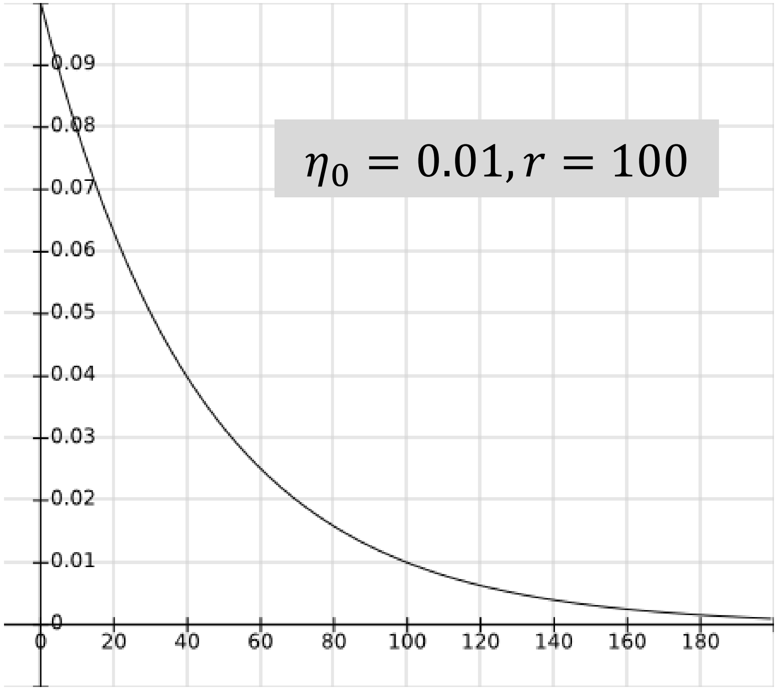
Exponential scheduling

- learning rate a function of iteration number:

$$\eta(t) = \eta_0 10^{-t/r}$$

- two tuning parameters: η_0, r



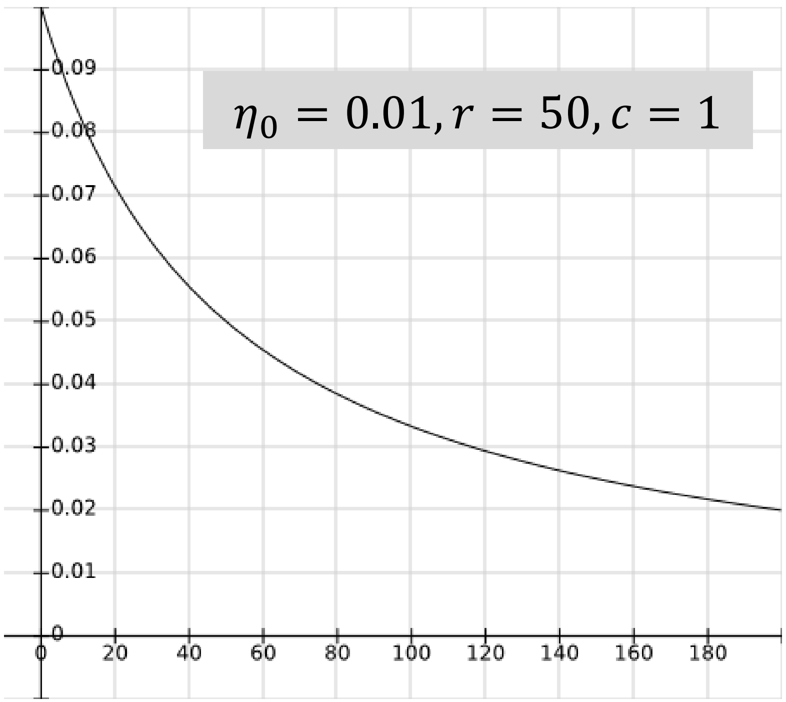


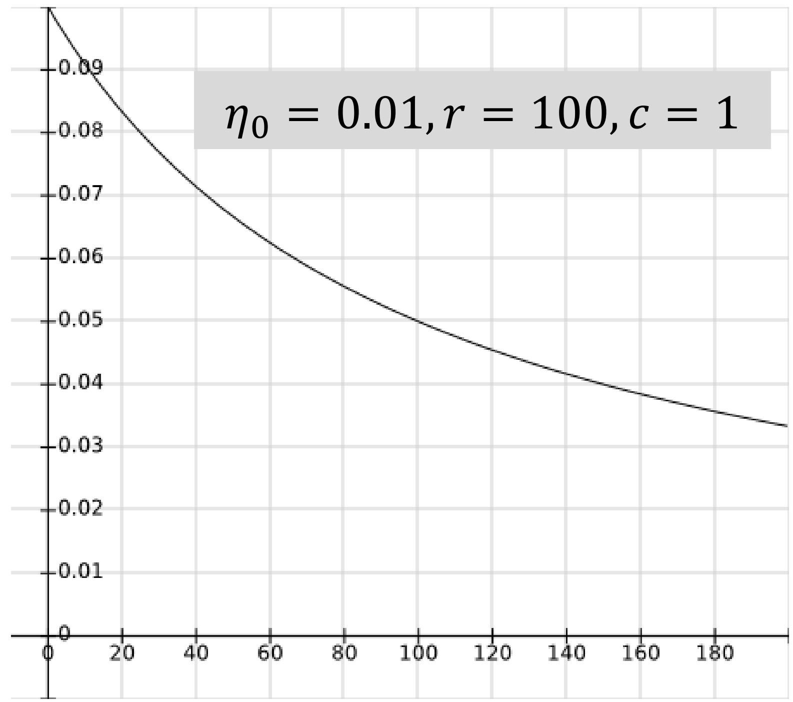
Power scheduling

- like exponential scheduling

$$\eta(t) = \eta_0 (1 + t/r)^{-c} \qquad (c \text{ usually } 1)$$

- … but learning rate drops more slowly





Regularization

Regularization methods address overfitting

Question: what are some examples we've seen?

- linear regression: the lasso, ridge regression
- kNN: make k larger
- reduce number of polynomial degrees
- early stopping

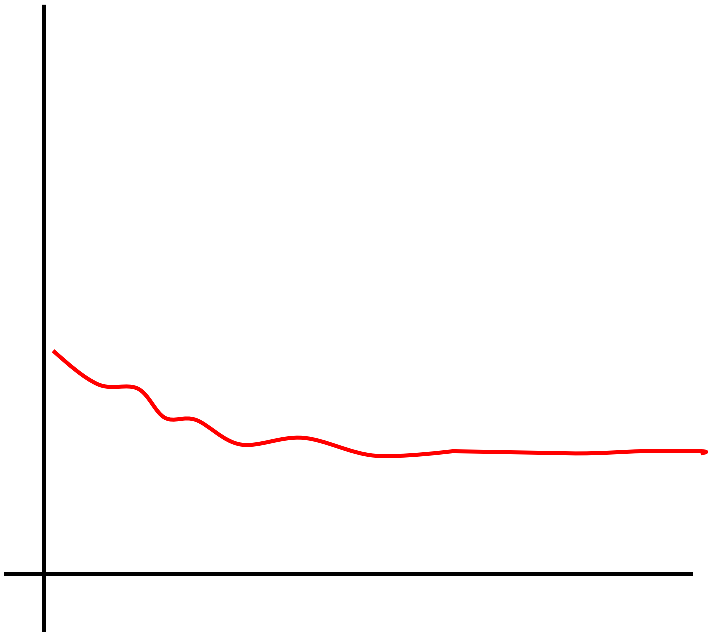
Question: how to regularize a neural net?

Early stopping

Stop training when performance on test set starts dropping

One way, in TensorFlow:

- periodically evaluate model on test set
- save a 'winner' snapshot if it outperforms earlier winner snapshots
- stop training if number of steps since last winner exceeds a threashold (like 2000 steps)

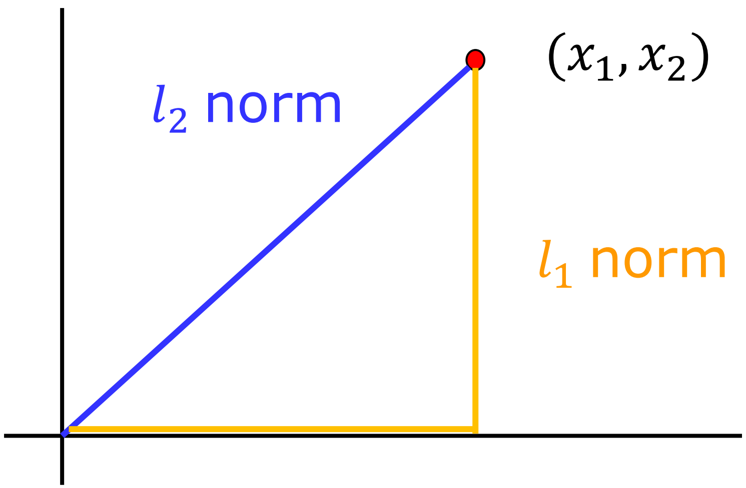


l_1, l_2 regularization

A norm gives a scalar 'size' to a vector

l_1 norm: "taxicab distances"

l_2 norm: Euclidean distance



Ridge regression penalizes cofficient using this term: $\alpha \frac{1}{2} \sum_{i=1}^{n} \theta_i^2$ (l_2 regularization)

Lasso regression penalizes coefficients using this term: $\alpha \sum_{i=1}^{n} |\theta_i|$ ($l_1$regularization)

l_1, l_2 regularization in TensorFlow

Add regularization penalties to connection weights.

```
scale = 0.001
my_dense_layer = partial(
    tf.layers.dense, activation=tf.nn.relu,
    kernel_regularizer=tf.contrib.layers.l1_regularizer(scale))

with tf.name_scope("dnn"):
    hidden1 = my_dense_layer(X, n_hidden1, name="hidden1")
    hidden2 = my_dense_layer(hidden1, n_hidden2, name="hidden2")
    logits = my_dense_layer(hidden2, n_outputs, activation=None,
                            name="outputs")
```

```
with tf.name_scope("loss"):
   xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,
                    logits=logits)
   base_loss = tf.reduce_mean(xentropy, name="avg_xentropy")
   reg_losses = tf.get_collection(tf.GraphKeys.REGULARIZATION_LOSSES)
   loss = tf.add_n([base_loss] + reg_losses, name="loss")
```

Dropout

Super effective.

Idea is simple:

- at each training step, each neuron (except output neurons) has some probability p of being ignored
- dropout rate p is typically set to 0.5
- dropout only occurs during training

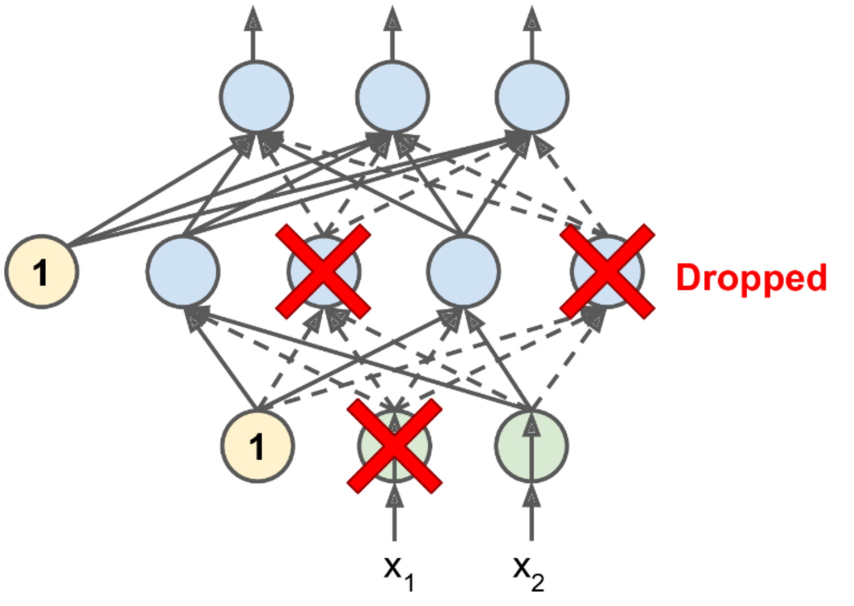


figure source: Géron

Why does dropout work?

Analogy with absent company employees:

- company could not depend on a few key employees
- present employees would learn to cooperate with more fellow employees
- in DNNs, neurons have to be useful on their own
- DNN cannot depend on a few input neurons

Diversity of neural nets: new one each step

- after 10,000 training steps you've trained 10,000 DNNs
- a kind of averaging ensemble

Adjusting weights in training

If p=0.5, half of neurons are absent in training

Input signal to neurons will be twice as big in testing!

To compensate, either:

- multiply each input connection weight by (1-p) after training   (1-p = "keep probability")
- divide each neuron's output by keep probability during training

Dropout in TensorFlow

```
[...]
training = tf.placeholder_with_default(False, shape=(),
                                       name='training')

dropout_rate = 0.5  # == 1 - keep_prob
X_drop = tf.layers.dropout(X, dropout_rate, training=training)

with tf.name_scope("dnn"):
    hidden1 = tf.layers.dense(X_drop, n_hidden1,
                              activation=tf.nn.relu, name="hidden1")
    hidden1_drop = tf.layers.dropout(hidden1, dropout_rate,
                              training=training)
    hidden2 = tf.layers.dense(hidden1_drop, n_hidden2,
                              activation=tf.nn.relu, name="hidden2")
    hidden2_drop = tf.layers.dropout(hidden2, dropout_rate,
                              training=training)
    logits = tf.layers.dense(hidden2_drop, n_outputs,
                              name="outputs")
```

Data augmentation

Reduce overfitting by creating addition training examples.

Create the new example by modify existing examples.

For example, if training on images:

- shift every image in training set
- rotate
- resize
- change contrast
- flip image horizontally

You can do this on the fly rather than creating a huge training set

Practical guidelines

Configure your DNN like this by default:

- initialization: He initialization
- activation function: ELU
- normalization: batch normalization
- regularization: dropout
- optimizer: Adam
- learning rate schedule: none

These practical guidelines are straight out of Géron's book.

Practical guidelines, cont'd.

Tweaking your configuration:

- Can't find good learning rate? Try a learning schedule, such as exponential delay
- Training set too small? Try data augmentation
- Need a sparse model? Add l_1 regularization, or FTRL instead of Adam optimization
- Need a fast model at runtime? Drop batch normalization, replace ELU with leaky ReLU

Summary

Learning rate scheduling:

- piecewise constant
- performance scheduling
- exponential scheduling
- power scheduling

Regularization

- early stopping
- l1, l2 regularization
- dropout
- data augmentation
- Max-Norm regularization (not covered)