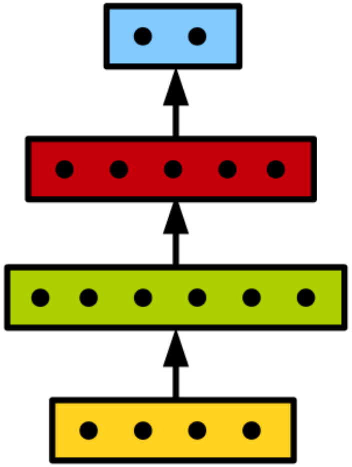
Más detalles, contenido del master haplap

Overfitting and regularization



The more layers / hidden units, the more capacity and risk of overfitting

- W can be **very good for training**, with enough layers and capacity the model can memorize the training data!

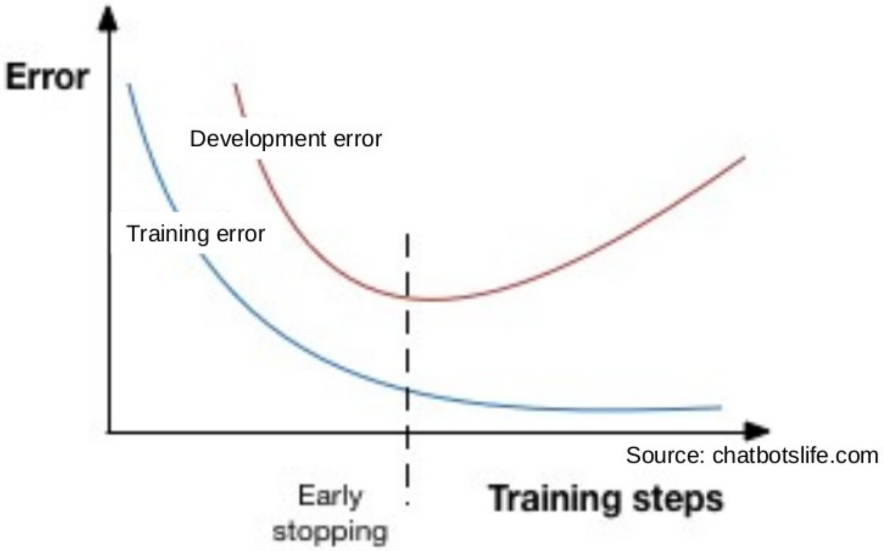
Low train error

- Generalize **very poorly to test** data (i.e. the real world)

High test error

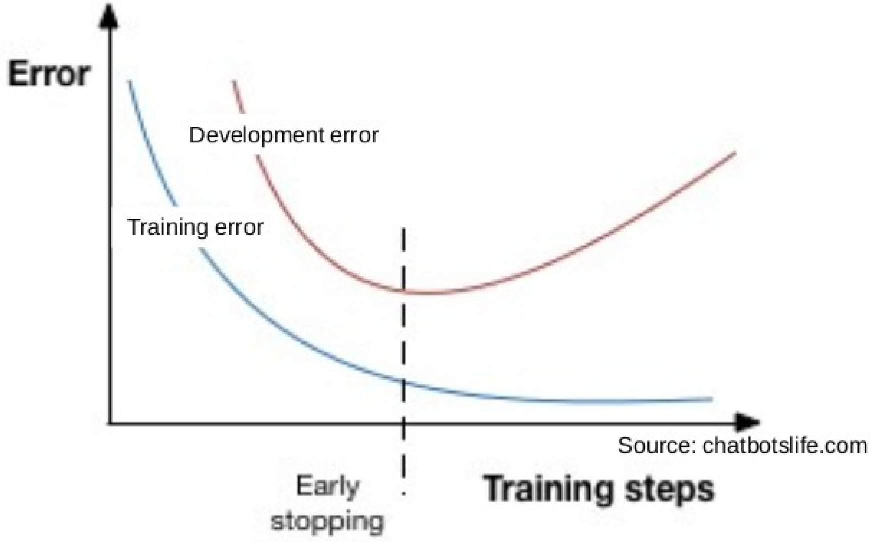
Overfitting, experimental design, Early stopping

- **Overfitting** can be seen in this graph



Overfitting, experimental design, Early stopping

- **Overfitting** can be seen in this graph
- **Experimental setup**: **%80 train, %10 dev,** %10 test (blind!!)
- **Early stopping:** finishes training as soon as development error starts to increase
- **Model selection**: best accuracy (lowest error) at development



Overfitting and regularization

Second solution: add a regularizer to the loss function that avoids the model to fit the training data

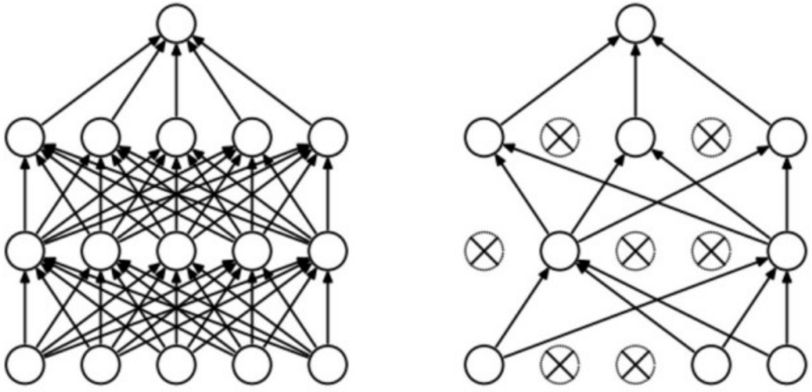
$$J_i(W) = -\log\left(\frac{\exp(W_{c_i}^T x)}{\sum_{c' \in C} \exp(W_{c'}^T x)}\right) + \lambda \sum_k W_k^2$$

squared L2 norm

Overfitting and regularization

Third solution: Dropout

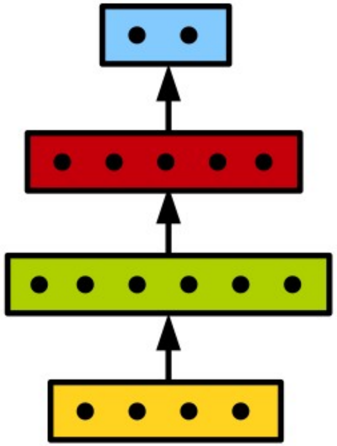
- At training time deactivate 50% of the activations at random (hyperparameter dropout rate)
- At test time use all activations

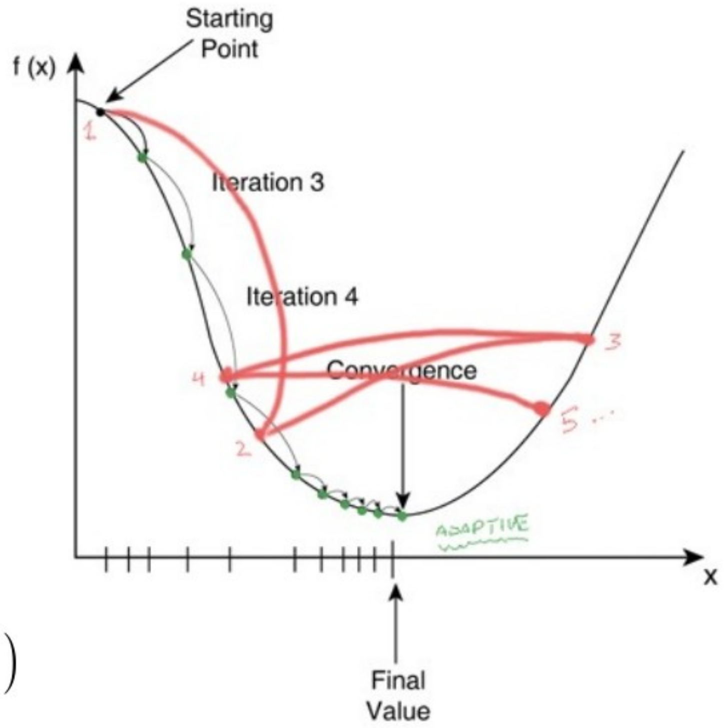


(a) Standard Neural Net (b) After applying dropout.

Source: "Dropout: a simple way to prevent neural networks from overfitting", JMLR 2014

SGD convergence: learning rate

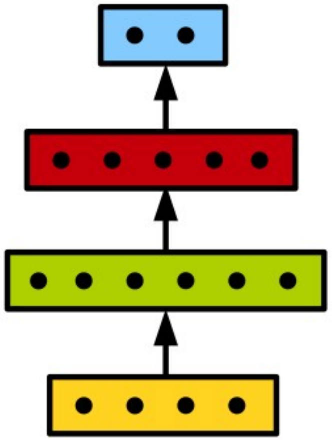


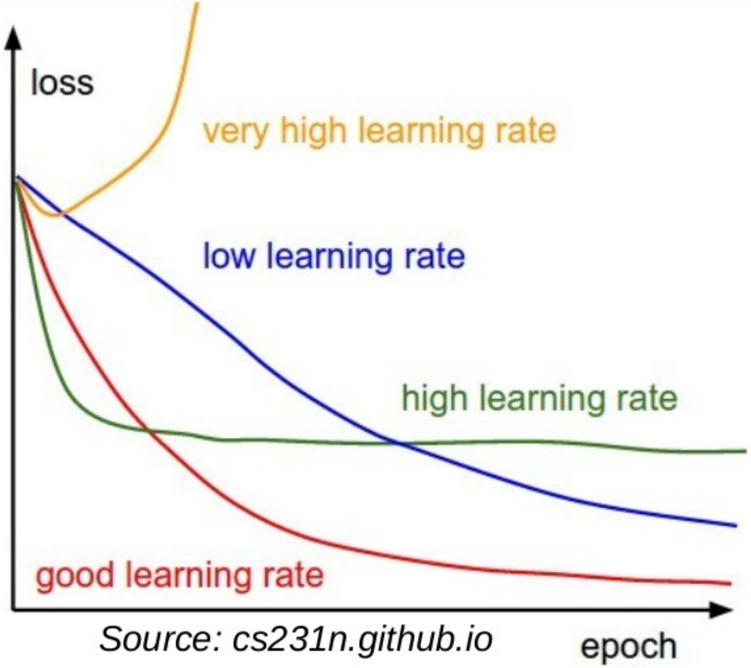


$$W = W - \eta \frac{1}{K} \sum_{i}^{K-1} \nabla J_i(W)$$

Source: cs231n.github.io

SGD convergence: learning rate

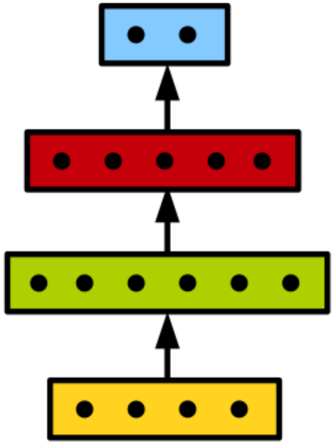




$$W = W - \eta \frac{1}{K} \sum_{i}^{K-1} \nabla J_i(W)$$

Source: cs231n.github.io

SGD convergence



Learning rate: Careful selection

- Optimizers (Adam, sgd, ...)

Don't expect convergence: use early stopping

Don't expect a unique solution: ensembling helps

$$W = W - \eta \frac{1}{K} \sum_{i}^{K-1} \nabla J_i(W)$$

MLP in Keras

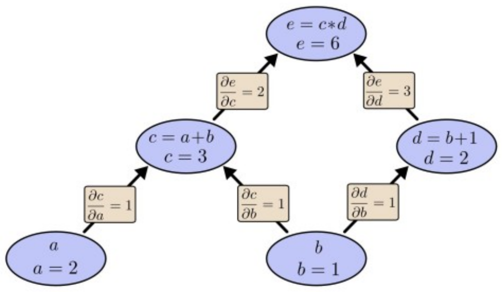
```
model = Sequential()

model.add(Dense(units=16,
                activation='sigmoid',
                input_shape=(x_train.shape[1],)))

model.add(Dense(units=1,
                activation='sigmoid'))


# input:  (None, 1000)
# hidden: (None, 16)
# output: (None, 1)
```

Introducing TensorFlow Keras

- High-level neural networks API, open source, Python
- High-level interface for Tensorflow 2
  - Works with on top of Pytorch too!
- Conceived as an interface for programmers
- Modular: model is a stack of layers
- Fast prototyping, hides details
  - Also its weakness: need to know lower level Tensorflow if layer/function is not available

Under the hood (back-end)

- Computational graphs
  - Numeric computation as data flow graphs
    - Nodes are operations, with inputs and outputs
    - Edges are tensors flowing between nodes
  - Forward computation
  - Backward (backpropagation) Autodiff: each node “knows” its derivative
    - Chain rule



Steps to implement a NN in keras

- Prepare the input and specify dimensions
- Define the model architecture (graph)
- Specify optimizer, loss, compile graph
- Train and test

Steps to implement a NN in keras

- **Prepare the input and specify dimensions**
- Define the model architecture (graph)
- Specify optimizer, loss, compile graph
- Train and test

```
(x_train,y_train), (x_dev,y_dev) = load_my_data()
print(x_train.shape) # (6920, 1000)
print(y_train.shape) # (6920,)
print(x_dev.shape) # (872, 1000)
print(y_dev.shape) # (872,)
```

Steps to implement a NN in keras

sigmoid(Wx + b)

- Prepare the input and specify dimensions
- **Define the model architecture (graph)**
- Specify optimizer, loss, compile graph
- Train and test

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential()
model.add(Dense(units=1,  activation='sigmoid',
        input_shape=(x_train.shape[1],)))

# input:  (None, 1000)
# output: (None, 1)
```

Steps to implement a NN in keras

- Prepare the input and specify dimensions
- Define the model architecture (graph)
- **Specify optimizer, loss, compile graph**
- Train and test

```
model.compile( loss='binary_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])
```

```
model.summary()
```

```
Layer (type)      Output Shape    Param
===================================== dense
(Dense)      (None, 1)       1001
```

Steps to implement a NN in keras

- Prepare the input and specify dimensions
- Define the model architecture (graph)
- Specify optimizer, loss, compile graph
- **Train and test**

```
model.fit(x_train, y_train,
          epochs=50, batch_size=32,
          validation_data=(x_dev, y_dev))

score = model.evaluate(x_test, y_test)

# model.metrics_names:  ['loss', 'acc']
# score:                [0.484, 0.759]
```

Steps to implement a NN in keras

- Prepare the input and specify dimensions
- Define the model architecture (graph)
- Specify optimizer, loss, compile graph
- **Train and test**