Hyperparameters

- A lot of hyperparameter choices
- Want to find the best configuration
- Usually done during the last steps of development

Learning Rate: 10^{-6}

Min 10^{-8}

Max 10^{-1}

L1 Dropout: 70%

Min 0%

Max 100%

...

Searching

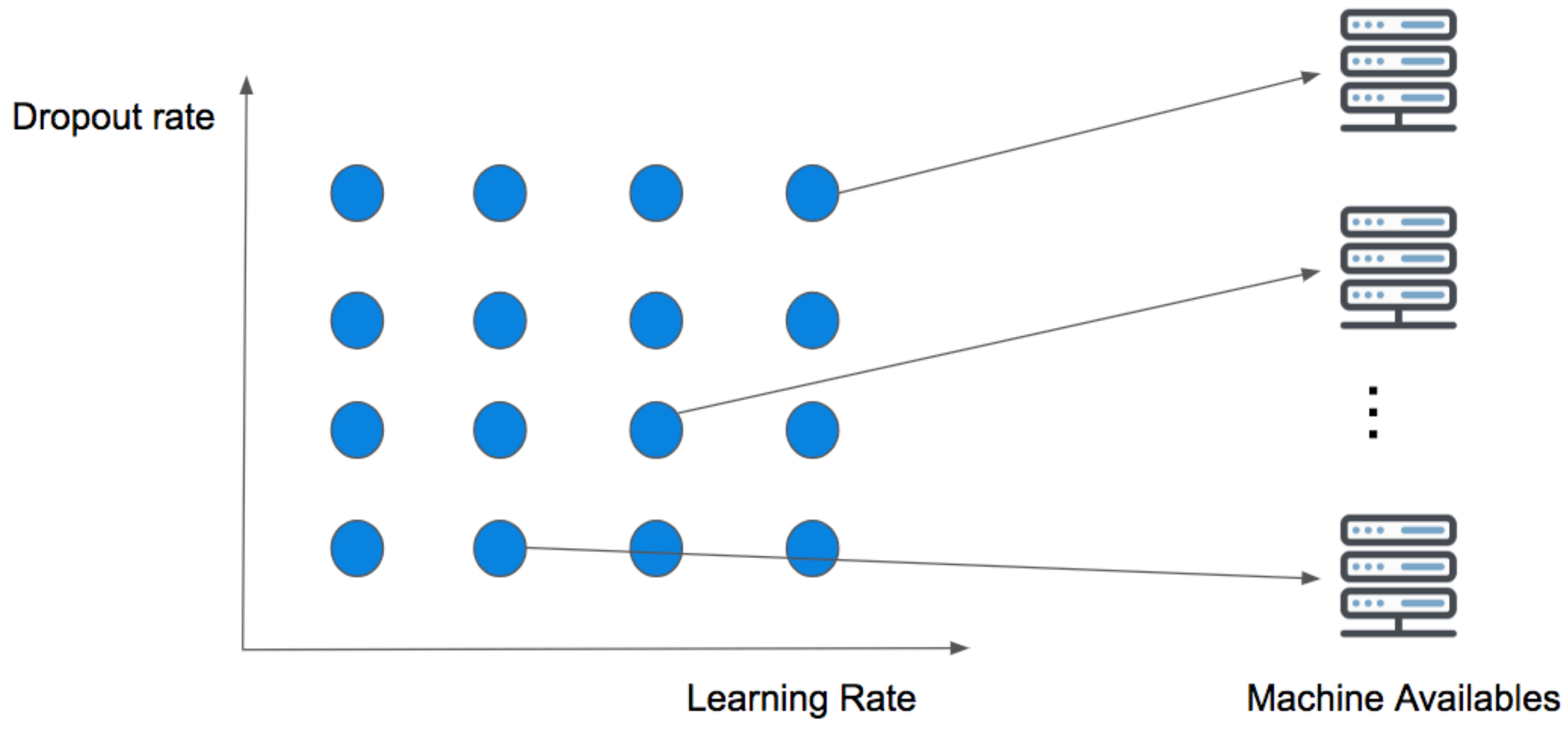
- Everything starts with a guess
- Wait until full training step
- Evaluate the performance
- Repeat!

Strategies

- Babysitting
- Grid search
- Random search
- Bayesian optimization

Grid search

- Grid of n dimensions one for each param
- For each dimension define range e.g. `batch_size = [16, 32, 128]`
- Search all possible configuration and return the best one



```python
def create_model(first_neuron=9,
                 activation='relu',
                 kernel_initializer='uniform',
                 dropout_rate=0,
                 optimizer='Adam'):
    # Create model
    model = Sequential()
    ...

    model.compile(...)
    return model
```

```python
from keras.wrappers.scikit_learn import KerasClassifier

model = KerasClassifier(build_fn=create_model)

first_neurons = [8, 9]
activation =  ['relu', 'elu']
...

param_grid = dict(first_neurons=first_neurons,
                  activation=activation,
                  ...)
```

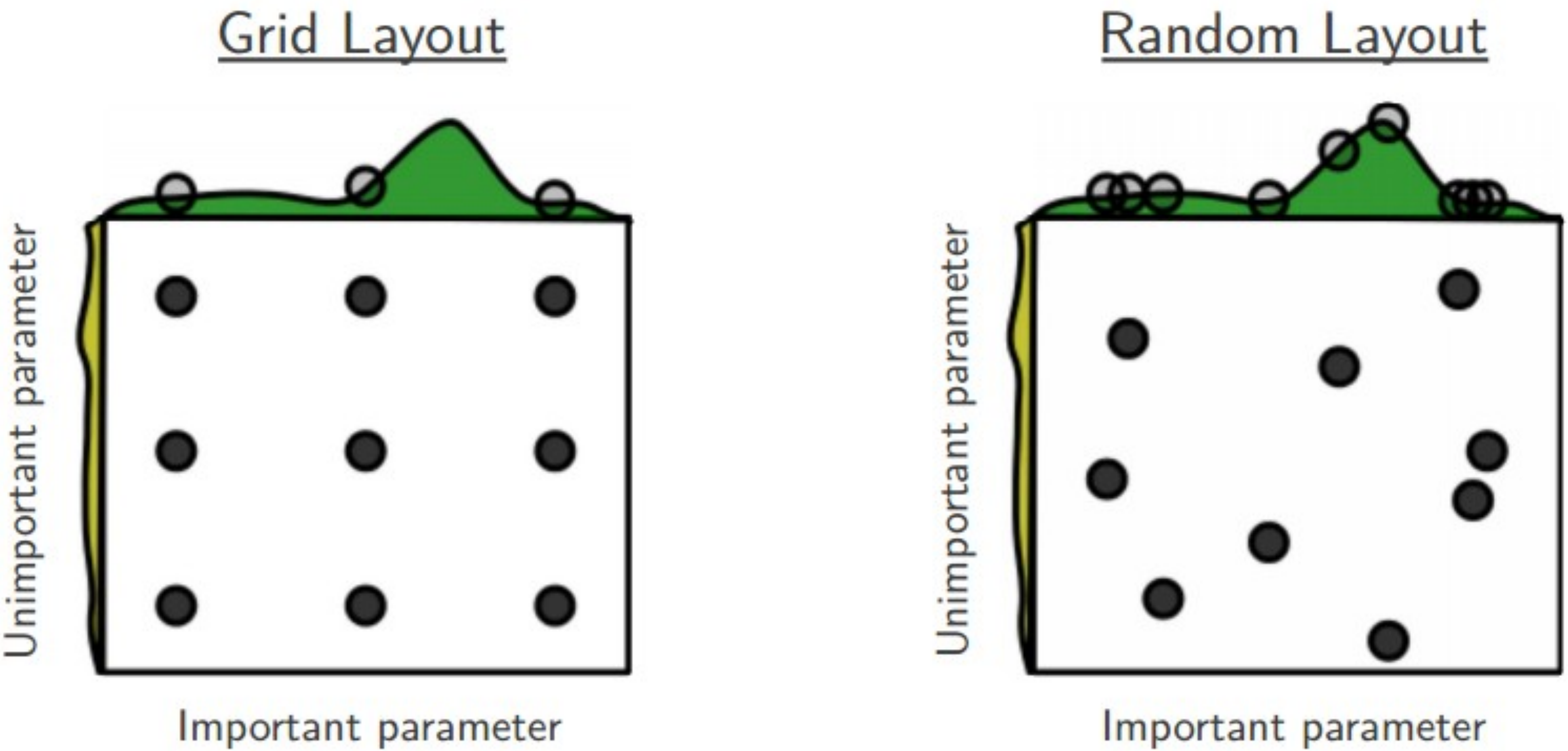
```python
from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(estimator=model,
                    param_grid=param_grid
                    n_jobs=1,
                    cv=3,
                    verbose=2)
grid_result = grid.fit(x, y)
...

print("Best: %f using %s" % (grid_result.best_score_,
    grid_result.best_params_))
```

<u>Random Search for Hyper-Param Optim</u> (2012)

- Randomly select the point from configuration space
- Not guaranteed to find the best parameters
- Better in higher dimensions
- Better results in less iterations


Grid Layout
Unimportant parameter
Important parameter
Random Layout
Unimportant parameter
Important parameter

```python
from sklearn.model_selection import RandomizedSearchCV

random_search = RandomizedSearchCV(
    estimator=model,
    param_distributions=param_dist,
    n_iter=n_iter_search,
    n_jobs=1,
    cv=3,
    verbose=2)

random_search.fit(x, y)
```

Each new guess independent from the previous run!

Bayesian optimization

- Build a model that predicts the metrics from hyperparam configs
- At each new try this model will become more and more confident
- Use Gaussian Process that will not only predict the metric but also its uncertainty (mean & var.)

Hyperas (Hyperopt **with** Keras)

- A Hyperopt wrapper for Keras models
- Hyperas lets you just use Hyperopt without learning its syntax
- Simply wrap the params you want in double curly brackets & choose a distribution over which to optimize

```
def create_model(x_train, y_train, x_test, y_test):
    ...
    model.add(Dropout({{uniform(0, 1)}}))
    ...
    model.compile(...)
    model.fit(...)

    score = model.evaluate(x_test, y_test, verbose=0)
    return {'loss': -score[1], 'status': STATUS_OK,
            'model': model}

def data():
    ...
    return x_train, y_train, x_test, y_test
```

```python
best_run, best_model = optim.minimize(
    model=create_model,
    data=data,
    algo=tpe.suggest,
    max_evals=10,
    trials=Trials())

_, _, X_test, Y_test = data()
print("Evaluation of best performing model:")
print(best_model.evaluate(X_test, Y_test))
```