Hyperparameters tuning

Plan for the lecture

- Hyperparameter tuning in general
  - General pipeline
  - Manual and automatic tuning
  - What should we understand about hyperparameters?
- Models, libraries and hyperparameter optimization
  - Tree-based models
  - Neural networks
  - Linear models

Plan for the lecture: models

- Tree-based models
  - GBDT: `XGBoost, LightGBM, CatBoost`
  - RandomForest/ExtraTrees
- Neural nets
  - `Pytorch, Tensorflow, Keras...`
- Linear models
  - SVM, logistic regression
  - `Vowpal Wabbit, FTRL`
- Factorization Machines (out of scope)
  - `libFM, libFFM`

How do we tune hyperparameters

1. **Select the most influential parameters**
   a. There are tons of parameters and we can’t tune all of them
2. **Understand, how exactly they influence the training**
3. **Tune them!**
   a. Manually (change and examine)
   b. Automatically (hyperopt, etc.)

Hyperparameter optimization software

- A lot of libraries to try:
  - *Hyperopt*
  - Scikit-optimize
  - Spearmint
  - GPyOpt
  - RoBO
  - SMAC3

Automatic hyperparameter optimization

```
def xgb_score(param):
    # run XGBoost with parameters `param`

def xgb_hyperopt():
    space = {
         'eta' : 0.01,
         'max_depth' :        hp.quniform('max_depth', 10, 30,1),
         'min_child_weight' : hp.quniform('min_child_weight', 0, 100, 1),
         'subsample' :        hp.quniform('subsample', 0.1, 1.0, 0.1),
         'gamma' :            hp.quniform('gamma', 0.0, 30, 0.5),
         'colsample_bytree' : hp.quniform('colsample_bytree', 0.1, 1.0, 0.1),

         'objective':'reg:linear',

         'nthread' : 28,
         'silent' : 1,
         'num_round' : 2500,
         'seed' : 2441,
         'early_stopping_rounds':100
         }


    best = fmin(xgb_score, space, algo=tpe.suggest, max_evals=1000)
```

Color-coding legend

1. Underfitting (bad)
2. **Good fit and generalization (good)**
3. Overfitting (bad)

Color-coding legend

- A parameter in red
  - *Increasing it impedes fitting*
  - Increase it to reduce overfitting
  - Decrease to allow model fit easier
- A parameter in green
  - *Increasing it leads to a better fit (overfit) on train set*
  - Increase it, if model underfits
  - Decrease if overfits

Conclusion

- Hyperparameter tuning in general
  - General pipeline
  - Manual and automatic tuning
  - What should we understand about hyperparameters?
- Models, libraries and hyperparameter optimization
  - Tree-based models
  - Neural networks
  - Linear models

Plan for the video

- Tree-based models
  - GBDT: `XGBoost, LightGBM, CatBoost`
  - RandomForest/ExtraTrees
- Neural nets
  - `Pytorch, Tensorflow, Keras...`
- Linear models
  - SVM, logistic regression
  - `Vowpal Wabbit, FTRL`
- Factorization Machines (out of scope)
  - `libFM, libFFM`

Tree-based models

Model	Where
GBDT	*XGBoost* (dmlc/xgboost) *LightGBM* (Mictrosoft/LighGBM) *CatBoost* (catboost/catboost)
RandomForest, ExtraTrees	*scikit-learn*
Others	*RGF* (baidu/fast_rgf)

GBDT

XGBoost

- max_depth

the deeper a tree can be grown the better it can fit a dataset.
So increasing this parameter will lead to faster fitting to the train set.
Depending on the task, the optimal depth can vary a lot, sometimes it is 2, sometimes it is 27. If you increase the depth and can not get the model to overfit, that is, the model is becoming better and better on the validation set as you increase the depth.
It can be a sign that there are a lot of important interactions to extract from the data. So it's better to stop tuning and try to generate some features.
I would recommend to start with a max_depth of about seven.
Also remember that as you increase the depth, the learning will take a longer time. So do not set depth to a very higher values unless you are 100% sure you need it.

LightGBM

- max_depth/num_leaves

In LightGBM, it is possible to control the number of leaves in the tree rather than the maximum depth.

It is nice since a resulting tree can be very deep, but have small number of leaves and not over fit.

GBDT

XGBoost	LightGBM
• max_depth • subsample	• max_depth/num_leaves • bagging_fraction

Some simple parameter controls a fraction of objects to use when feeding a tree.
It's a value between 0 and 1.
One might think that it's better always use all the objects, right? But in practice, it turns out that it's not.
Actually, if only a fraction of objects is used at every duration, then the model is less prone to overfitting.
So using a fraction of objects, the model will fit slower on the train set, but at the same time it will probably generalize better than this over-fitted model. So, it works kind of as a regularization.

GBDT

XGBoost	LightGBM
• max_depth	• max_depth/num_leaves
• subsample	• bagging_fraction
• colsample_bytree, colsample_bylevel	• feature_fraction

Similarly, if we can consider only a fraction of features split, this is controlled by parameters colsample_bytree and colsample_bylevel. 如果over fitting就lower down

GBDT

XGBoost	LightGBM
• max_depth	• max_depth/num_leaves
• subsample	• bagging_fraction
• colsample_bytree, colsample_bylevel	• feature_fraction
• min_child_weight, lambda, alpha	• min_data_in_leaf, lambda_l1, lambda_l2

min_child_weight increse it, model become more conservative

In my experience, it's one of the most important parameters to tune in XGBoost and LightGBM. Depending on the task, I find optimal values to be 0, 5, 15, 300, so do not hesitate to try a wide range of values, it depends on the data.

GBDT

XGBoost	LightGBM
• max_depth	• max_depth/num_leaves
• subsample	• bagging_fraction
• colsample_bytree, colsample_bylevel	• feature_fraction
• min_child_weight, lambda, alpha	• min_data_in_leaf, lambda_l1, lambda_l2
• eta num_round	• learning_rate num_iterations

Eta is essentially a learning weight, like in gradient descent. And the num_round is the how many learning steps we want to perform or in other words how many tree's we want to build.

, the higher the learning rate, the faster the model fits to the train set and probably it can lead to over fitting. And more steps model does, the better the model fits.

GBDT

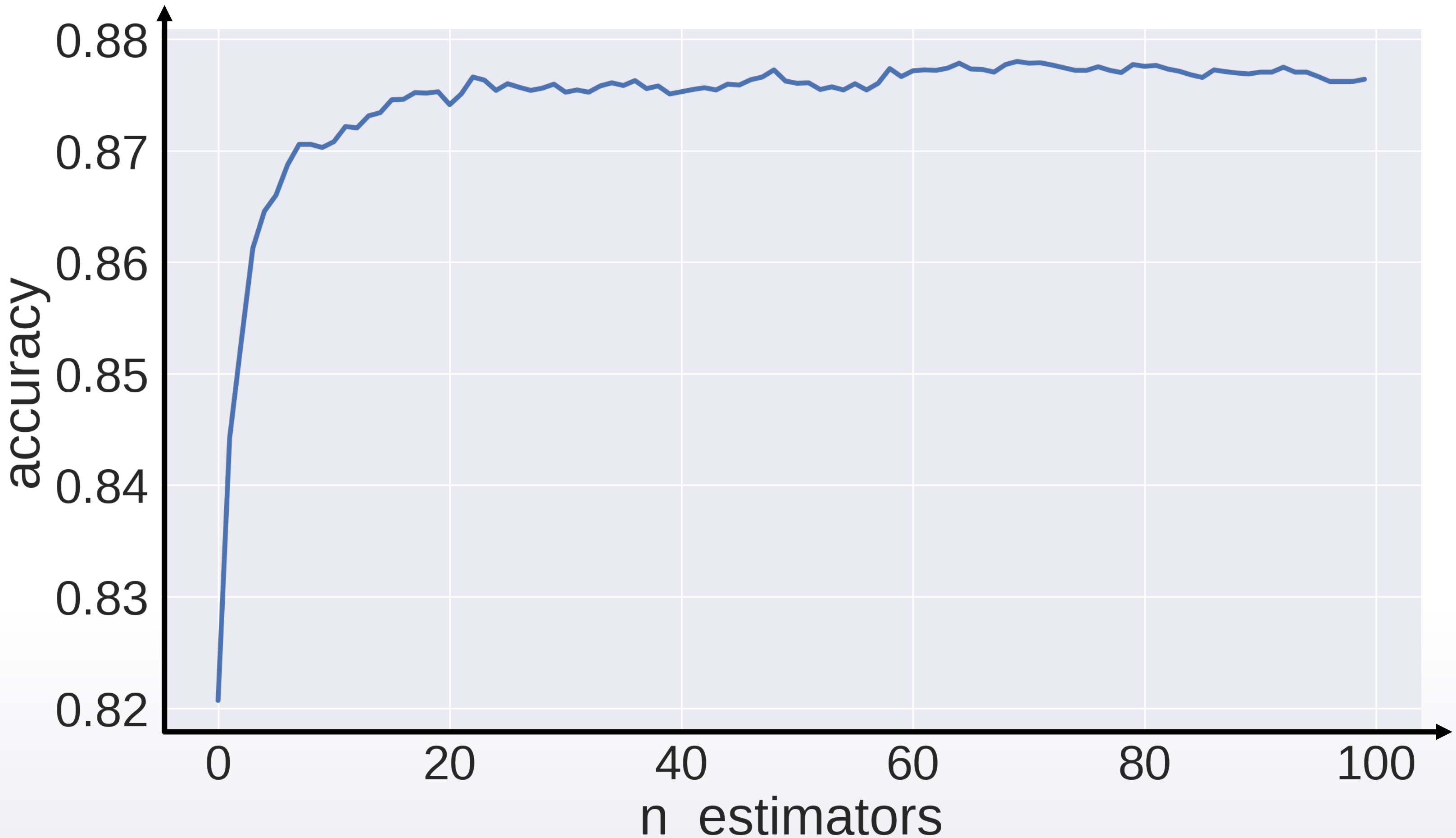
one could jump 1,000 places up or down on the leaderboard just by training a model with different random seeds

XGBoost	LightGBM
• max_depth	• max_depth/num_leaves
• subsample	• bagging_fraction
• colsample_bytree, colsample_bylevel	• feature_fraction
• min_child_weight, lambda, alpha	• min_data_in_leaf, lambda_l1, lambda_l2
• eta num_round	• learning_rate num_iterations
Others:	Others:
• seed	• *_seed

I think it doesn't make too much sense to fix seed in XGBoost, as anyway every changed parameter will lead to completely different model. But I would use this parameter to verify that different random seeds do not change training results much

sklearn.RandomForest/ExtraTrees

- N_estimators (the higher the better)



sklearn.RandomForest/ExtraTrees

- N_estimators (the higher the better)
- max_depth
- max_features
- min_samples_leaf

Others:

- *criterion*

sklearn.RandomForest/ExtraTrees

- N_estimators (the higher the better)
- max_depth
- max_features
- min_samples_leaf

Others:

- *criterion* In my experience Gini is better more often, but sometimes Entropy wins.
- **random_state**
- **n_jobs**

Conclusion

- Tree-based models
  - GBDT: `XGBoost, LightGBM, CatBoost`
  - RandomForest/ExtraTrees
- Neural nets
  - `Pytorch, Tensorflow, Keras...`
- Linear models
  - SVM, logistic regression
  - `Vowpal Wabbit, FTRL`
- Factorization Machines (out of scope)
  - `libFM, libFFM`

Hyperparameter tuning
part III

Plan for the lecture: models

- Tree-based models
  - GBDT: `XGBoost, LightGBM, CatBoost`
  - RandomForest/ExtraTrees
- Neural nets
  - `Pytorch, Tensorflow, Keras...`
- Linear models
  - SVM, logistic regression
  - `Vowpal Wabbit, FTRL`
- Factorization Machines (out of scope)
  - `libFM, libFFM`

Plan for the lecture: models

- **What framework to use?**
  - Keras, Lasagne
  - TensorFlow
  - MxNet
  - PyTorch
  - sklearn's MLP
  - …

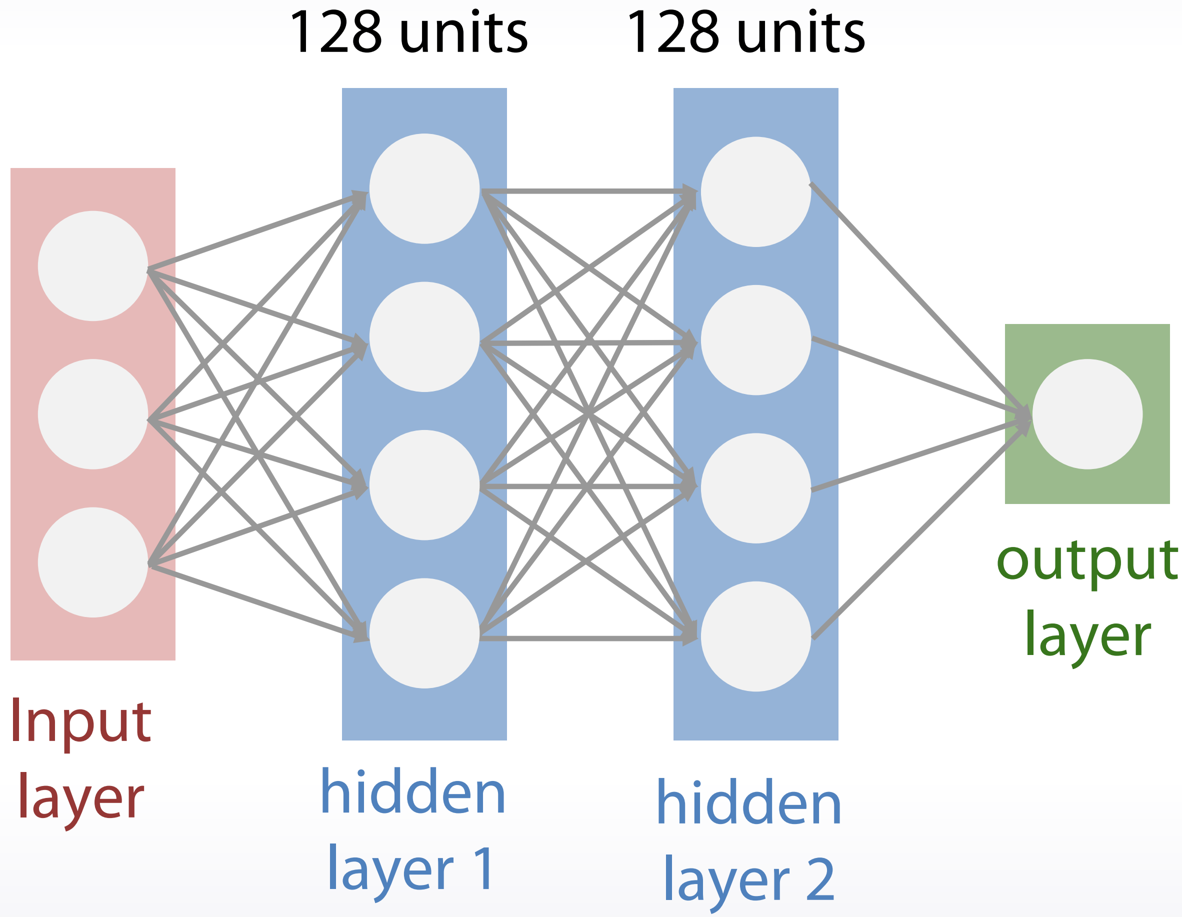
They implement the same functionality! (except sklearn)

- **I recommend:**
  - PyTorch
  - Keras

Neural Nets

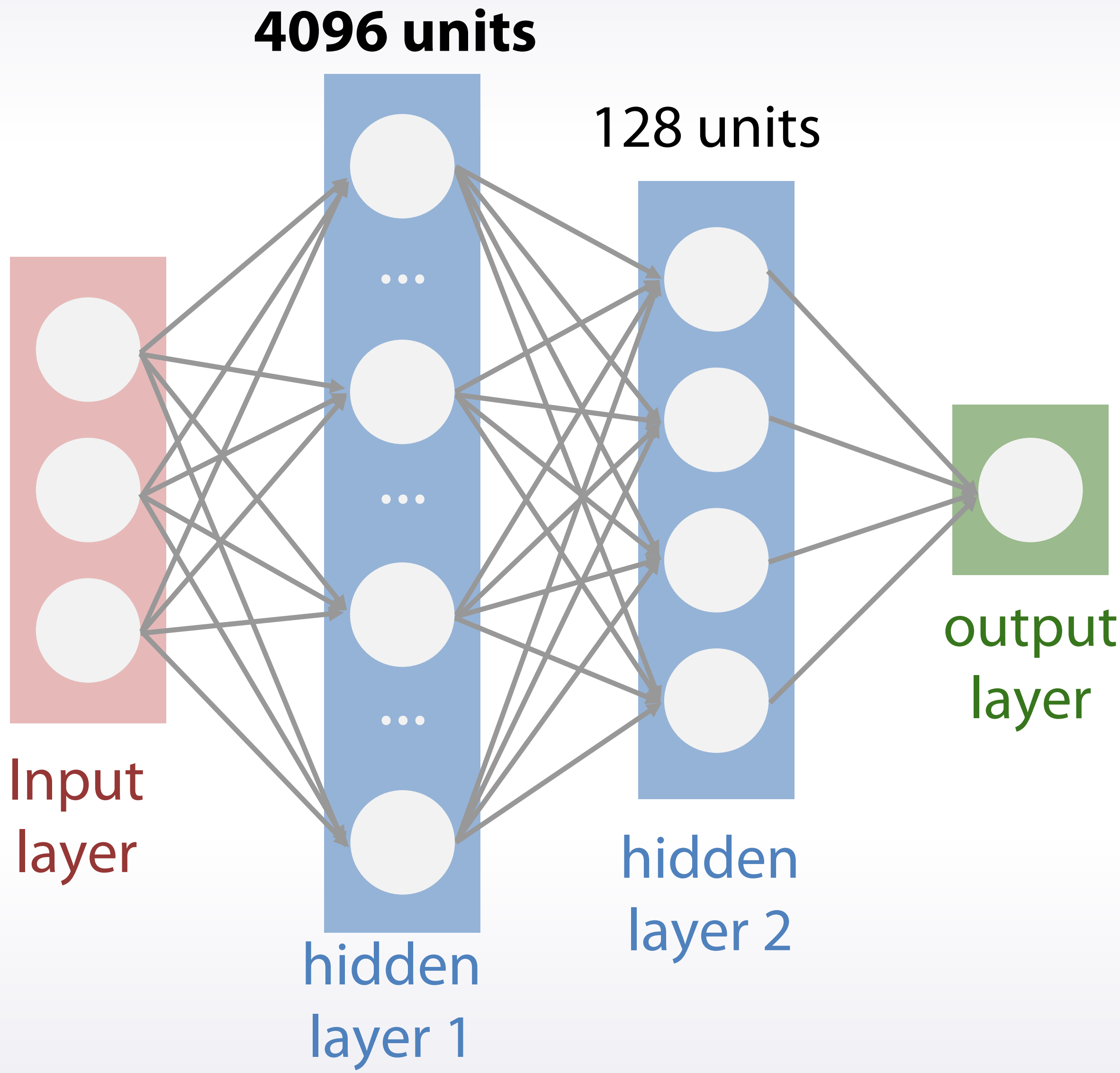
- Number of neurons per layer
- Number of layers
- Optimizers
  - SGD + momentum
  - Adam/Adadelta/Adagrad/...
    - In practice lead to more overfitting
- Batch size
- Learning rate
- Regularization
  - L2/L1 for weights
  - Dropout/Dropconnect
  - Static dropconnect

Static dropconnect

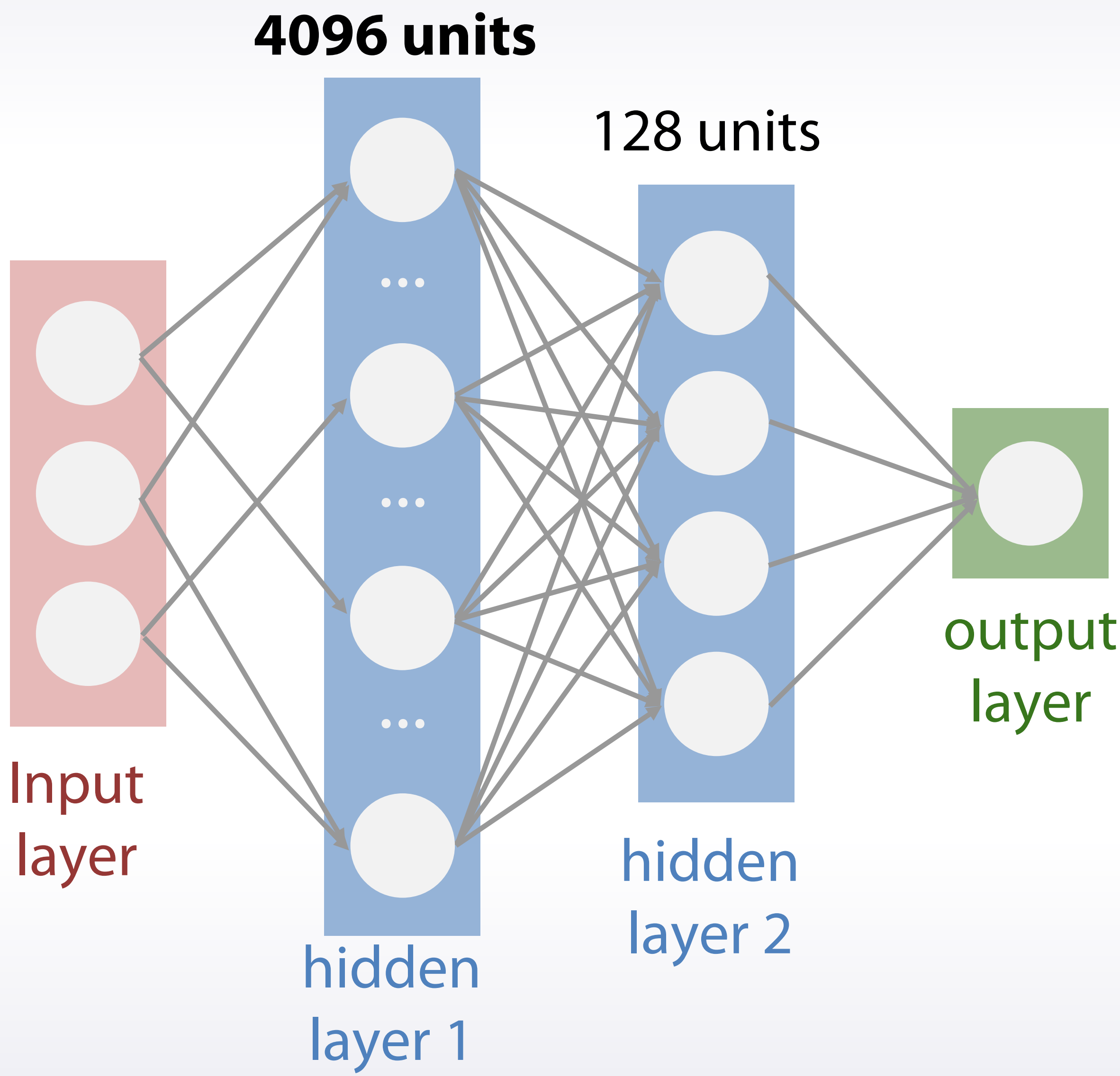


Based on image from *Stanford University CS231n*

Static dropconnect



Static dropconnect



Linear models

- **Scikit-learn**
  - `SVC/SVR`
    - Sklearn wraps `libLinear` and `libSVM`
    - Compile yourself for multicore support

Linear models

- **Scikit-learn**
  - `SVC/SVR`
    - Sklearn wraps `libLinear` and `libSVM`
    - Compile yourself for multicore support
  - `LogisticRegression/LinearRegression` + regularizers
  - `SGDClassifier/SGDRegressor`

Linear models

- **Scikit-learn**
  - `SVC/SVR`
    - Sklearn wraps `libLinear` and `libSVM`
    - Compile yourself for multicore support
  - `LogisticRegression/LinearRegression` + regularizers
  - `SGDClassifier/SGDRegressor`
- **`Vowpal Wabbit`**
  - `FTRL`

Linear models

- Regularization parameter (C, alpha, lambda, …)
  - Start with very small value and increase it.
  - SVC starts to work slower as C increases

- Regularization type
  - *L1/L2/L1+L2* -- try each
  - *L1* can be used for feature selection

Tips

- **Don't spend too much time tuning hyperparameters**
  - Only if you don't have any more ideas or you have spare computational resources
- **Be patient**
  - It can take thousands of rounds for GBDT or neural nets to fit
- **Average everything**
  - Over random seed
  - Or over small deviations from optimal parameters
    - e.g. average *max_depth=4,5,6* for an optimal *5*

Conclusion

- Hyperparameter tuning in general
  - General pipeline
  - Manual and automatic tuning
  - What should we understand about hyperparameters?
- Models, libraries and hyperparameter optimization
  - Tree-based models
  - Neural networks
  - Linear models