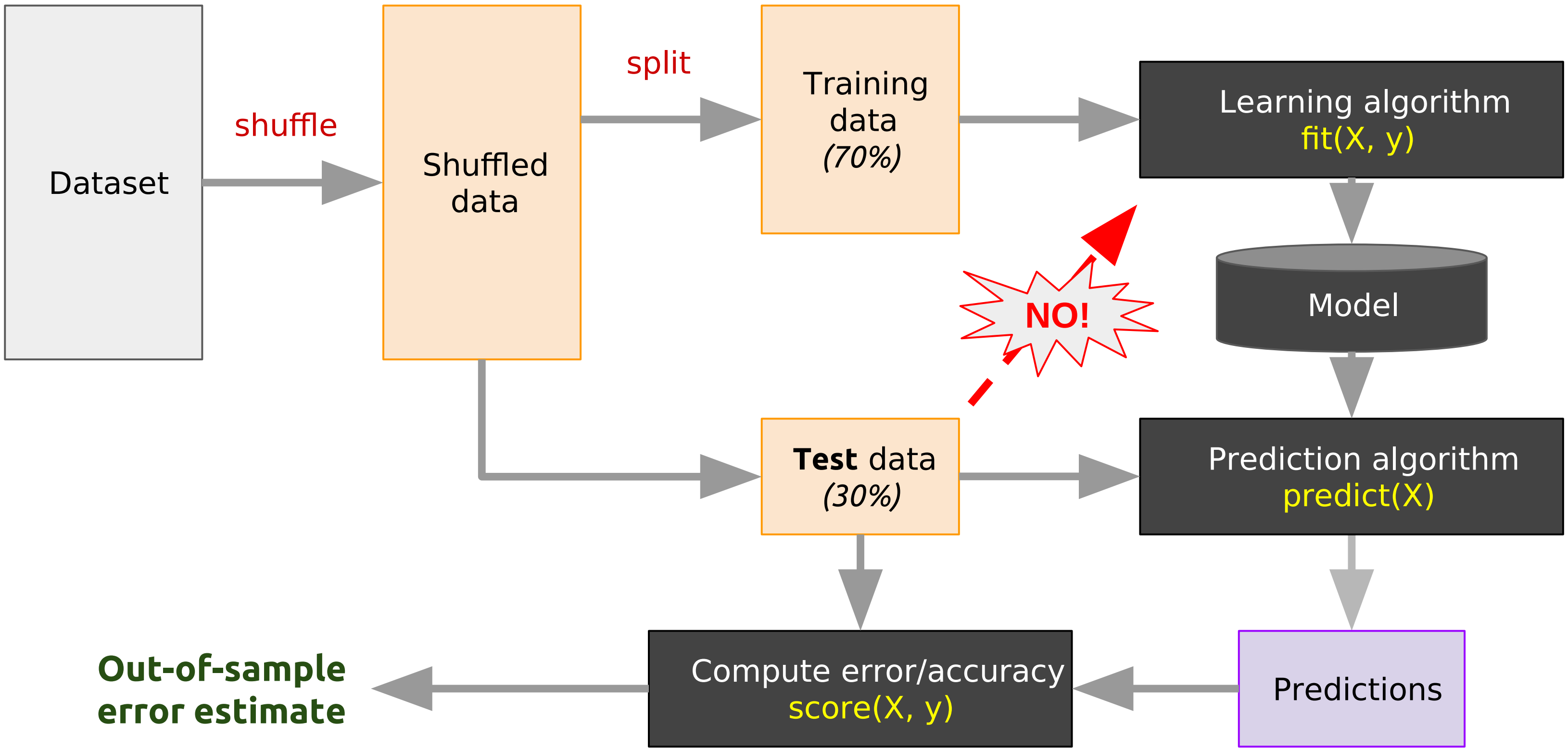
Deep learning and Artificial Neural Network for signal applications – **The learning step**

Dr. Anthony Fleury - HDR
IMT Nord Europe

Dr. Sebastien Ambellouis
Ingénieur de recherche Université Gustave Eiffel
Associé à IMT Nord Europe

sebastien.ambellouis@univ-eiffel.fr
anthony.fleury@imt-nord-europe.fr

Training set: Train/Test Splits



Slides credit: Eva Mohedano/ Kevin McGuinness

Training: Remember Metrics/Loss function

Classification Metrics:

$$Accuracy = \frac{\boxed{TP + TN}}{\boxed{TP + TN + FP + FN}} \longrightarrow$$

Not differenciable!

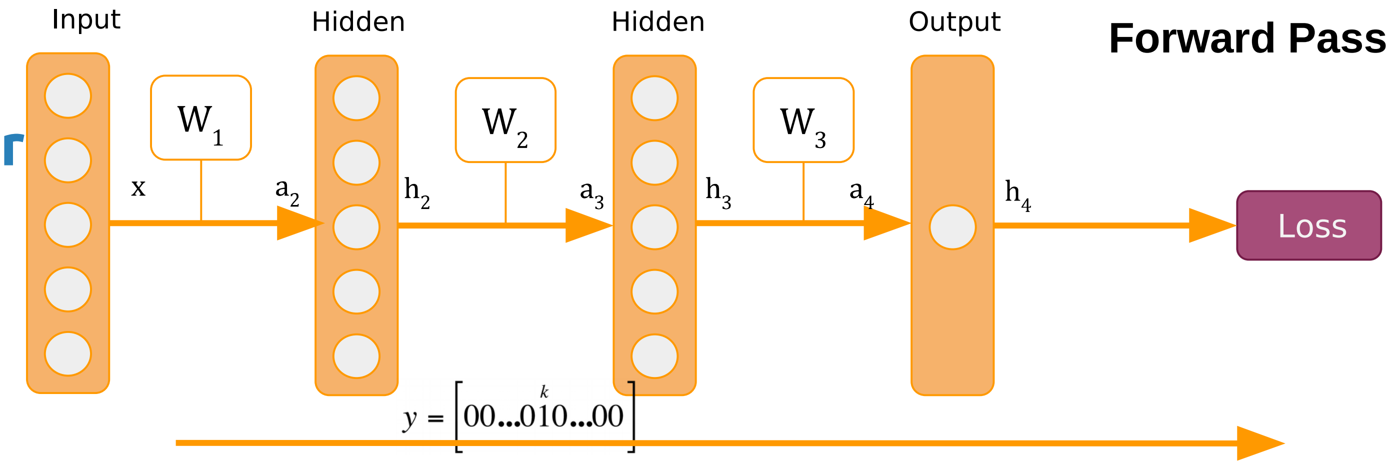
Example: Binary cross entropy:

$$L = -\frac{1}{N}\sum_{i=1}^{N} y_i \log f(\mathbf{x}_i) + (1 - y_i)\log(1 - f(\mathbf{x}_i)) \longrightarrow$$

Differenciable!

Training: Remember Backprop

Forward Pass



Probability Class given an input (softmax)

$$p(c_k = 1|\mathbf{x}) = \frac{\exp(a_k)}{\sum_c \exp(a_c)}$$

Loss function; e.g., negative log-likelihood (good for classification)

$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x}))$$

Regularization term (L2 Norm) aka as weight decay

$$L(\mathbf{x}, y; \mathbf{W}) = -\sum_j y_j \log(p(c_j|\mathbf{x})) + \frac{\lambda}{2}\|\mathbf{W}\|_2^2$$

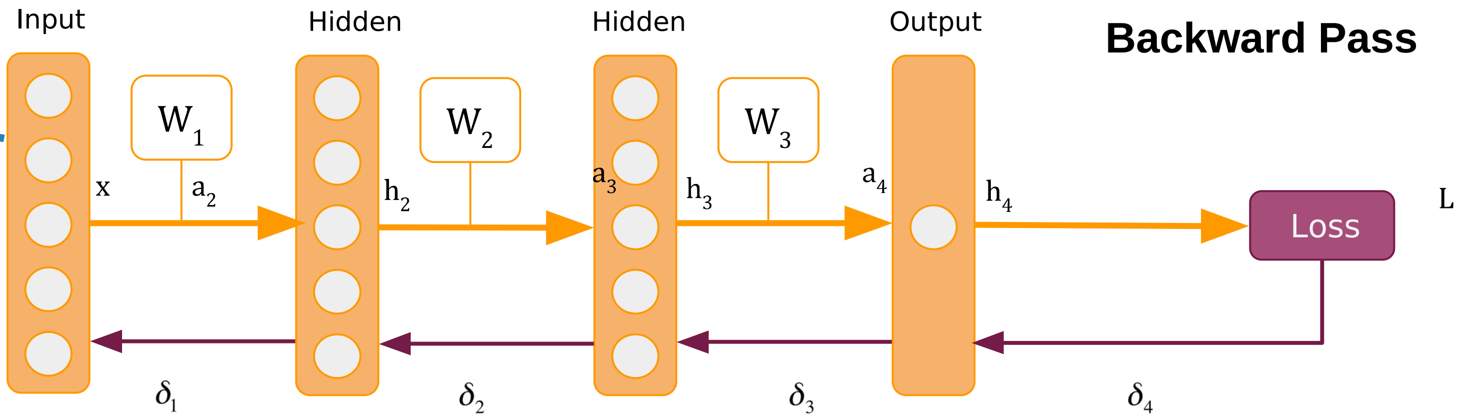
Minimize the loss (plus some regularization term) w.r.t. Parameters over the whole training set.

$$\mathbf{W}^* = argmin_\theta \sum_j L(\mathbf{x}^n, y^n; \mathbf{W})$$

Figure Credit: Kevin McGuiness

Training: Remember Backprop

Backward Pass



1. Find the error in the top layer:

$$\delta_K = \frac{\partial L}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K}\frac{\partial h_K}{\partial a_K}$$

$$\delta_K = \frac{\partial L}{\partial h_K} \bullet g'(a_K)$$

2. Compute weight updates

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_{k+1}}\frac{\partial a_{k+1}}{\partial W_k}$$

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_{k+1}} \bullet h_k$$

$$\frac{\partial L}{\partial W_k} = \delta_{k+1} \bullet h_k$$

To simplify we don't consider the bias

3. Backpropagate error to layer below

$$\delta_k = \frac{\partial L}{\partial a_k}$$

$$\delta_k = \frac{\partial L}{\partial a_{k+1}}\frac{\partial a_{k+1}}{\partial h_k}\frac{\partial h_k}{\partial a_k}$$

$$\delta_k = W_k^T \frac{\partial L}{\partial a_{k+1}} \bullet g'(a_k)$$

$$\delta_k = W_k^T \delta_{k+1} \bullet g'(a_k)$$

Figure Credit: Kevin McGuiness

Training: Monitoring progress

1. Split data into train, validation, and test sets

    Keep 10-30% of data for validation

2. Fit model parameters on train set using SGD

3. After each epoch:

    Test model on validation set and compute loss

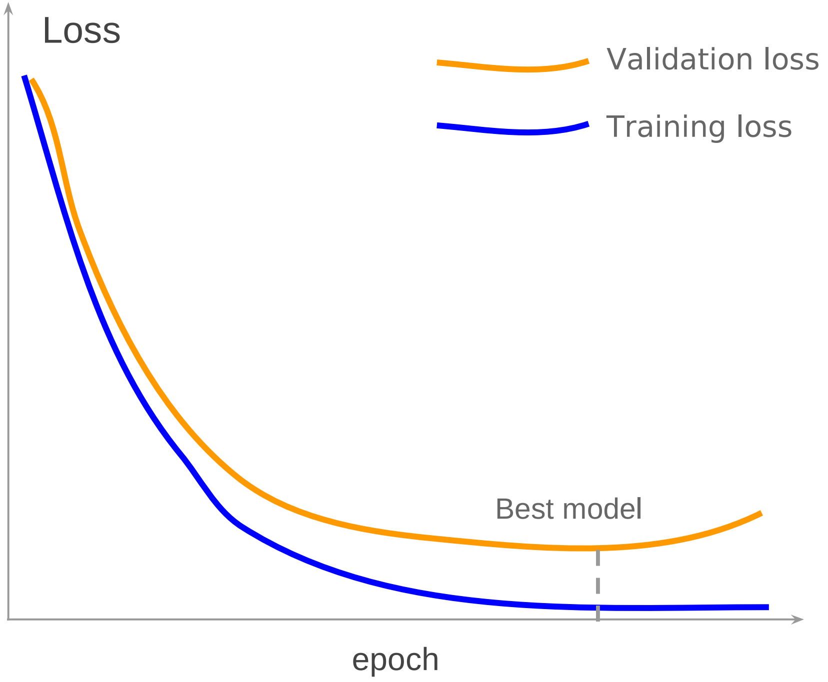
        Also compute whatever other metrics you are interested in, e.g. top-5 accuracy

    Save a snapshot of the model

4. Plot **learning curves** as training progresses

5. Stop when validation loss starts to increase

6. Use model with minimum validation loss



Overfitting

Symptoms:

Validation loss decreases at first, then starts increasing

Training loss continues to go down

Try:

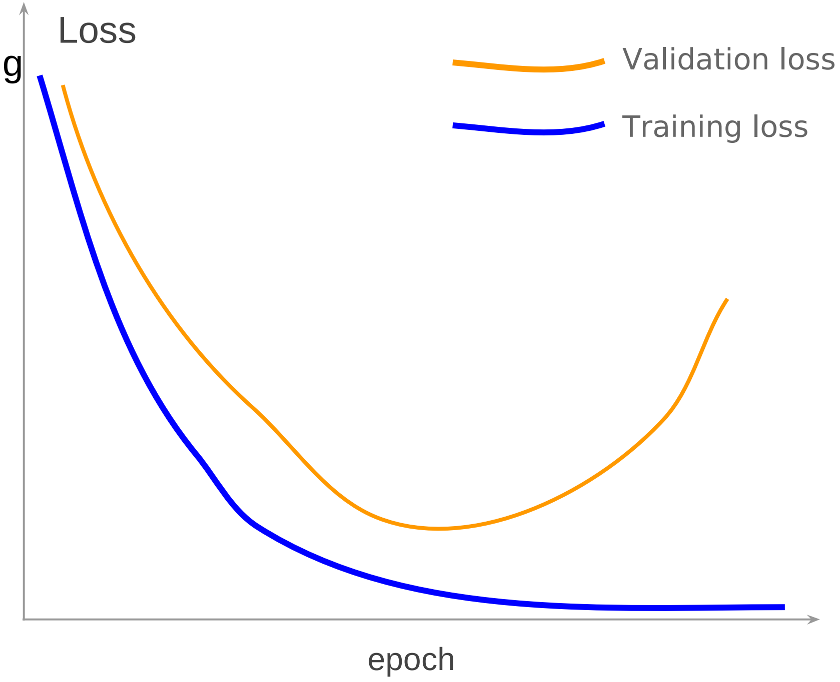
Find more training data

Add stronger regularization

dropout, drop-connect, L2

Data augmentation (flips, rotations, noise)

Reduce complexity of your model



Underfitting

Symptoms:

Training loss decreases at first but then stops

Training loss still high

Training loss tracks validation loss

Try:

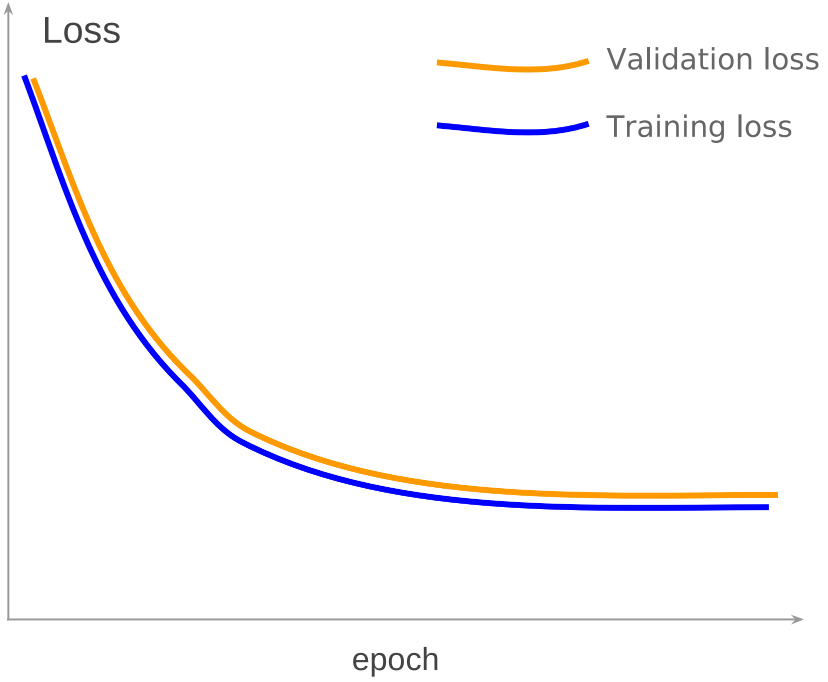
Increase model capacity

Add more layers, increase layer size

Use more suitable network architecture

E.g. multi-scale architecture

Decrease regularization strength



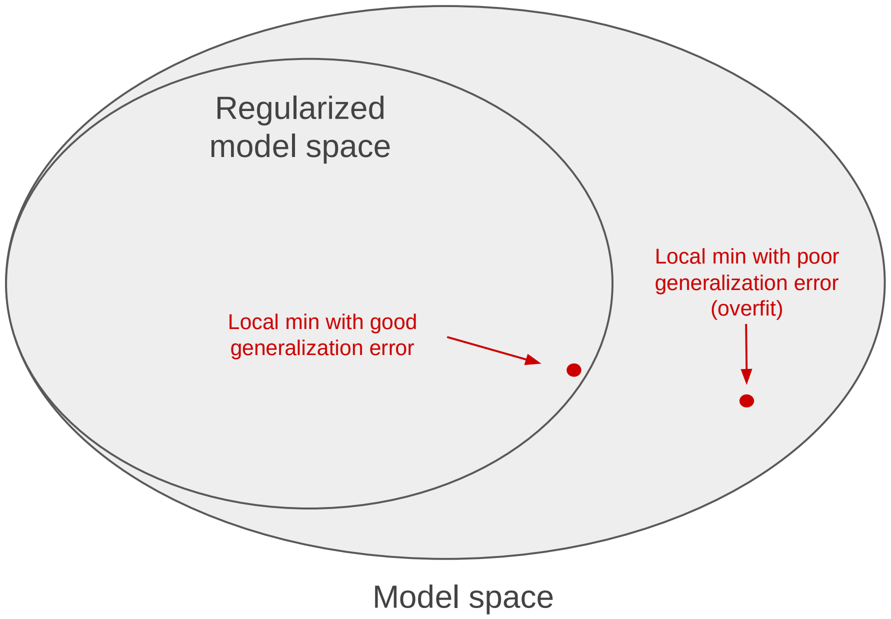
Regularization

Early stopping is a form of structural risk minimization

- Limits the space of models we explore to only those we expect to have good generalization error
- Helps prevent **overfitting**
- A type of **regularization**

Other regularization techniques:

- Weight constraints: e.g. **L2 regularization**
  - Aka. weight decay
- Dropout
- Transfer learning, pretraining



Regularization: Weight decay

Add a penalty to the loss function for large weights

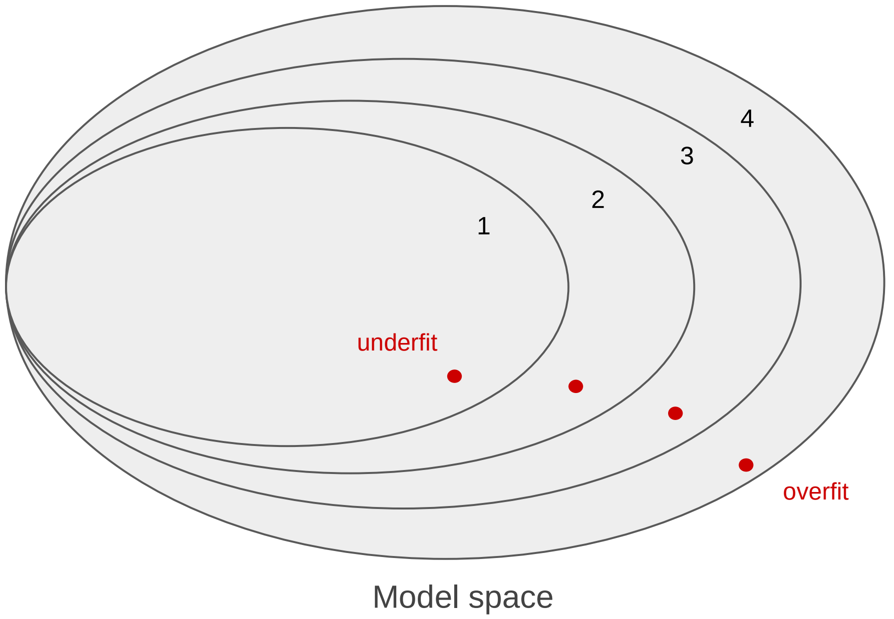
L2 regularization on weights

$$L = L_{\text{data}} + \frac{\lambda}{2}||W||_2^2$$

Differentiating, this translates to decaying the weights with each gradient descent step

$$w_{t+1} = w_t - \alpha \Delta_w L_{\text{data}} \boxed{- \lambda w}$$

1 > 2 > 3 > 4



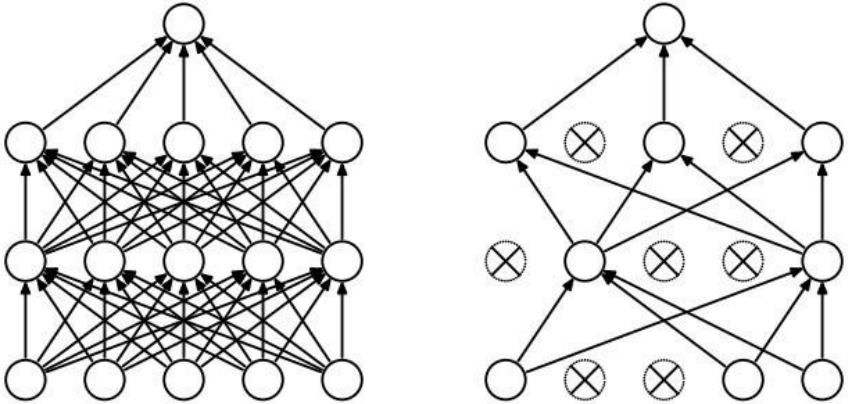
Regularization: Dropout

Modern regularization technique for deep nets

Used by many deepnets

Method:

- During training, outputs of a layer to zero randomly with probability p
  - Prevents units from co-adapting too much
  - Forces network to learn more robust features
- At test time, dropout is disabled and unit output is multiplied by p



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

Srivastava et al. **Dropout: A simple way to prevent neural networks from overfitting.** JRML 15(1), 2014, pp 1929-1958.

Hyperparameters

Can already see we have lots of **hyperparameters** to choose:

1. Learning rate
2. Regularization constant
3. Number of epochs
4. Number of hidden layers
5. Nodes in each hidden layer
6. Weight initialization strategy
7. Loss function
8. Activation functions
9. …   :(

Choosing these is difficult, and a bit of an art.

There are some reasonable **heuristics**:

1. Try 0.1 for the learning rate. If this doesn't work, divide by 3. Repeat.
2. Multiply LR by 0.1 every 1-10 epochs.
3. Try ~ 0.00001 as regularization constant
4. Try an existing network architecture and adapt it for your problem
5. Start smallish, keep adding layers and nodes until you overfit too much

You can also do a **hyperparameter search** if you have enough compute:

Randomized search tends to work well

Other Important definitions / Reference

Batch size: A CNN doesn't process its inputs one-at-a-time: to increase throughput, it will process the data in batches, or sometimes called mini-batches.

Batch Normalization: Normalization done at each layer for a batch before the activation function (paper in the following link too).
https://gist.github.com/shagunsodhani/4441216a298df0fe6ab0
Helpful for the vanishing/exploding gradient

Deep Learning
An MIT Press book, 2016
Ian Goodfellow and Yoshua Bengio and Aaron Courville
http://www.deeplearningbook.org/contents/guidelines.html