INTRODUCTION TO MACHINE LEARNING

NEURAL NETWORKS II

Mingon Kang, Ph.D.
Department of Computer Science @ UNLV

Learning Neural Networks

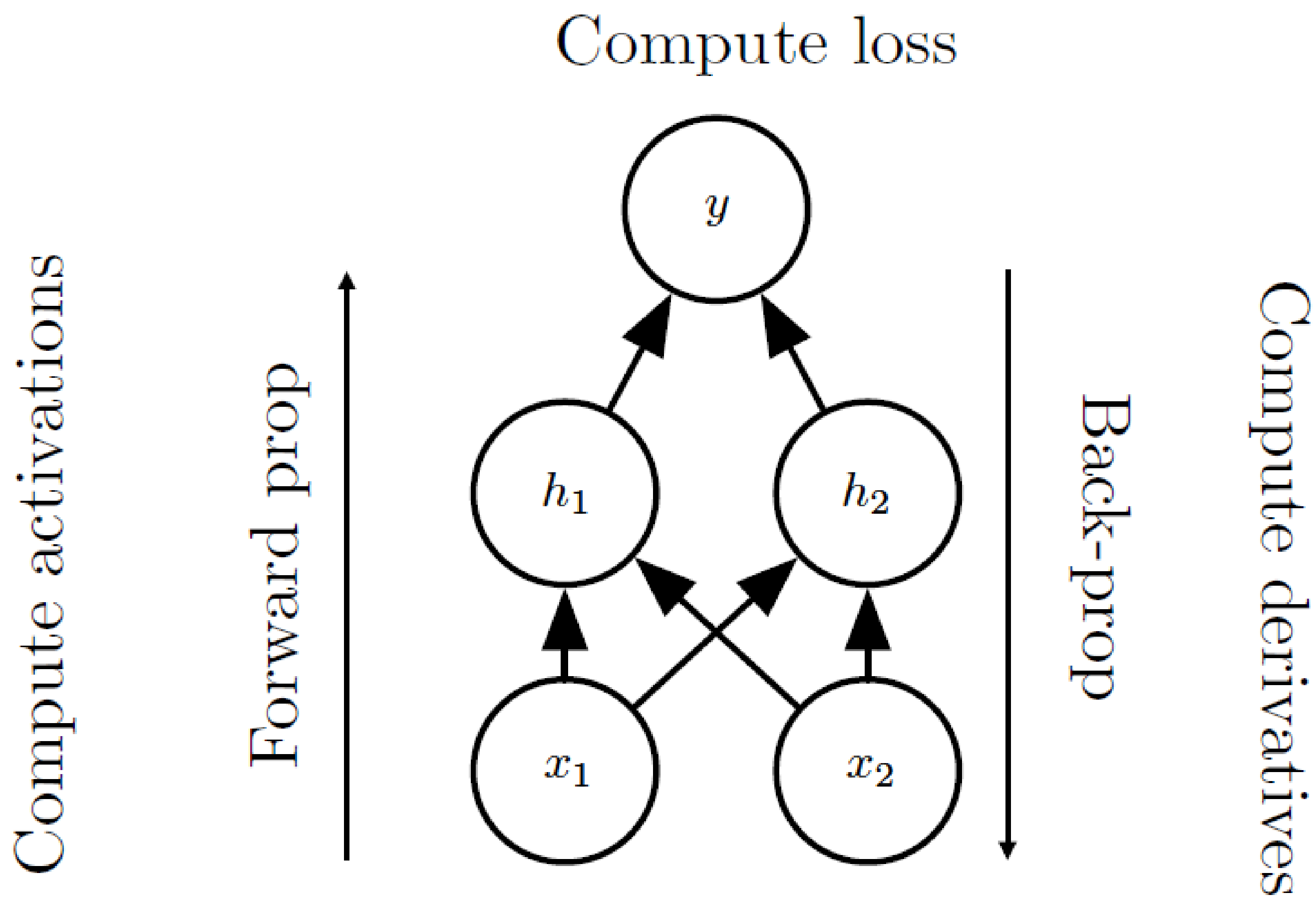
- NN have a number of parameters to train
  - Weights and biases

- Learning is to estimate the optimal parameters that perform well with training data, as well as test data potentially.
  - Minimize total residual sum of squares (MSE)
  - Maximize maximum likelihood function (MLE)

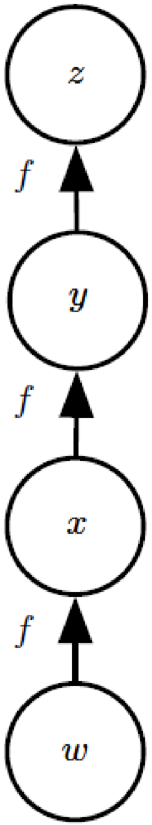
Backpropagation

- Backpropagation
  - the most popular solution in NN
  - "Chain Rule" of calculus
    - $\frac{d_z}{d_x} = \frac{d_z}{d_y}\frac{d_y}{d_x}$
  - Particular implementation of the chain rule
    - Uses dynamic programming (table filling)
    - Avoids re-computing repeated subexpressions
    - Speed vs memory tradeoff

Simple Back-Prop Example



Repeated Subexpressions



$$\frac{\partial z}{\partial w} \tag{6.50}$$

$$=\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\frac{\partial x}{\partial w} \tag{6.51}$$

$$=f'(y)f'(x)f'(w) \tag{6.52}$$

$$=f'(f(f(w)))f'(f(w))f'(w) \tag{6.53}$$

Back-prop avoids computing this twice

Figure 6.9

Symbol-to-Symbol Differentiation

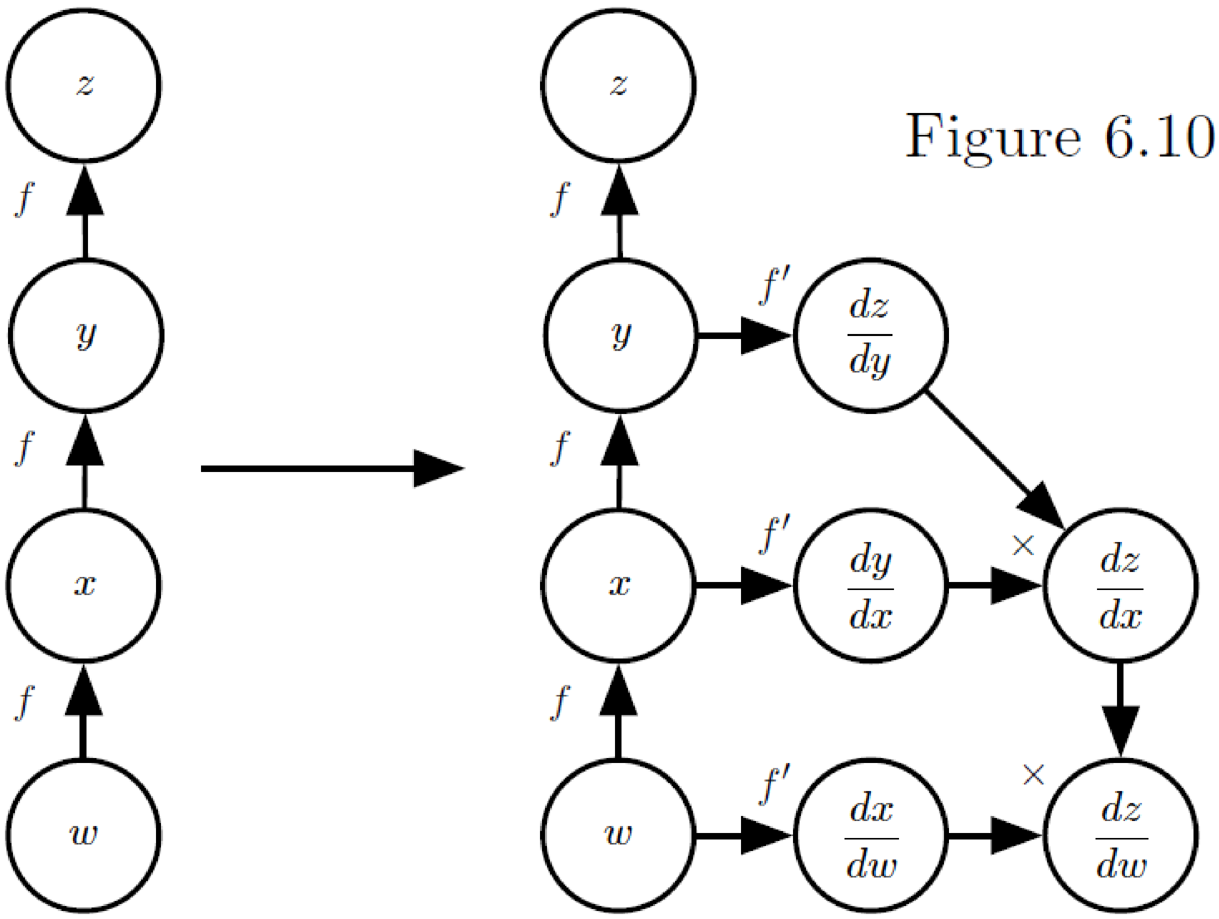
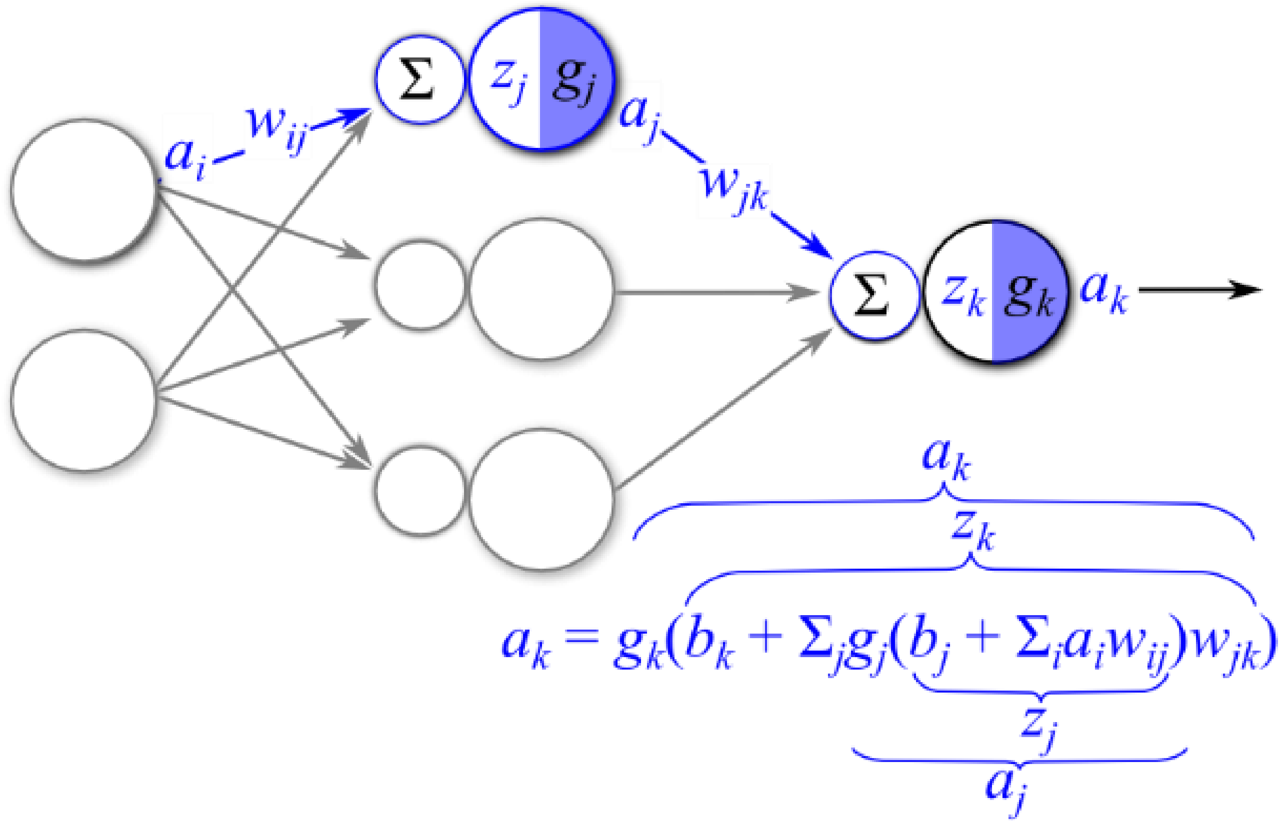


Figure 6.10

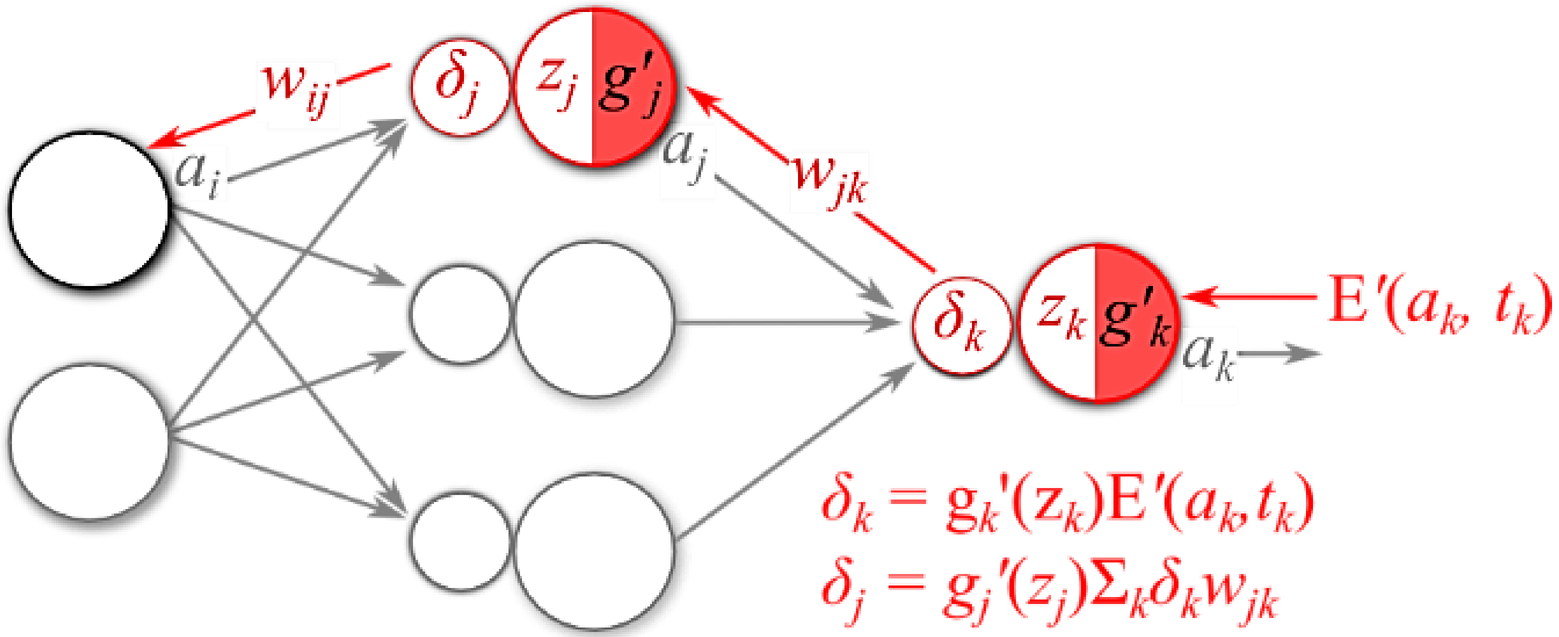
Forward-propagate

I. Forward-propagate Input Signal



Back-propagate

II. Back-propagate Error Signals

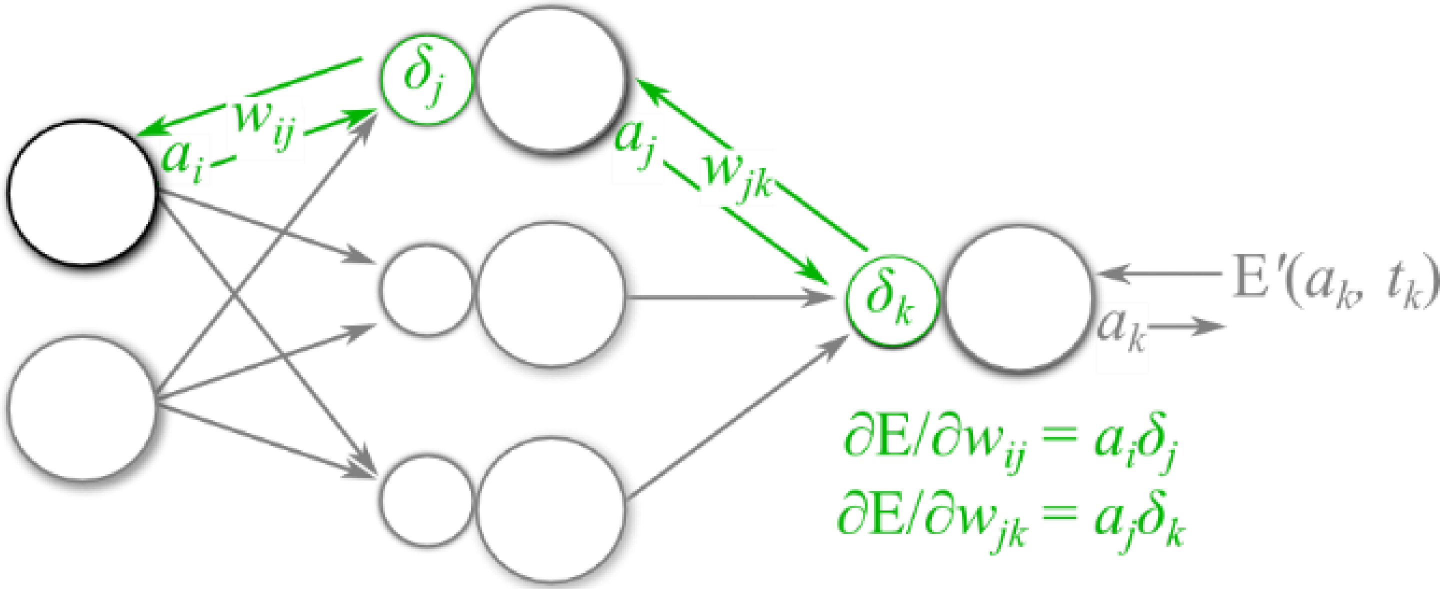


$\delta_k = g_k'(z_k)E'(a_k, t_k)$

$\delta_j = g_j'(z_j)\Sigma_k\delta_k w_{jk}$

Update Parameters

III. Calculate Parameter Gradients



$$\partial E/\partial w_{ij} = a_i\delta_j$$
$$\partial E/\partial w_{jk} = a_j\delta_k$$

IV. Update Parameters

$$w_{ij} = w_{ij} - \eta(\partial E/\partial\, w_{ij})$$
$$w_{jk} = w_{jk} - \eta(\partial E/\partial w_{jk})$$

for learning rate η

Backpropagation

- Backpropagation shows how parameters in a network changes the cost function.
  - $\partial C / \partial w_{jk}^{l}$ and $\partial C / \partial b_{j}^{l}$, where C is a cost function
- Introduce an intermediate quantity δ_j^l, which is an error in the j-th neuron in the l-th layer.
  - Relate δ_j^l to $\partial C / \partial w_{jk}^{l}$ and $\partial C / \partial b_{j}^{l}$

Backpropagation

- Define the error δ_j^l of neuron j in layer l by
  - $\delta_j^l \equiv \frac{\partial C}{\partial z_j^l}$, where $z_j^l = \sum_k w_{jk}^l a_k^{l-1} + b_j^l$
- Equation for the error in the output layer
  - $\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L} = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L)$
  - If considering quadratic cost function and sigmoid activation
    - $C = \frac{1}{2} \sum_j (y_i - a_j^L)^2, \sigma = 1/(1 + e^{-z_j^L})$
    - Then, $\frac{\partial C}{\partial a_j^L} = a_j^L - y_i, \sigma'(z_j^L) = \sigma(z_j^L)(1 - \sigma(z_j^L))$

Ref: http://neuralnetworksanddeeplearning.com/chap2.html

Backpropagation

- Derivative w.r.t. biases in a network
  - $\frac{\partial C}{\partial b_j^l} = \delta_j^l$
- Derivative w.r.t. weights in a network
  - $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$

Backpropagation Algorithm

1) Input x: set the corresponding activation a^1 for the input layer
2) Feedforward: for each $l = 2,3, \ldots, L$ compute $z^l = w^l a^{l-1} + b^l$ and $a^l = \sigma(z^l)$
3) Output error δ^L for cost function
4) Backpropagate the error: For each $l = L-1, L-2, \ldots, 2$ compute δ^l
5) Update weights and biases: $\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1} \delta_j^l$, $\frac{\partial C}{\partial b_j^l} = \delta_j^l$

Vanishing Gradient Problem

- Why?
  - With deep layers on networks, the gradients of the loss function approaches zero, which make the network to fail to train.
  - In backpropagation, the gradient decreases exponentially as propagate down to the initial layers.
- Solutions
  - Use other activation functions (e.g., ReLU) than sigmoid
  - Batch normalization

Stochastic Gradient Descent

- Stochastic Gradient Descent (SGD)
  - Calculates the error and updates the model for each example in the training dataset.

Ref: https://machinelearningmastery.com/gentle-introduction-mini-batch-gradient-descent-configure-batch-size/

Batch Gradient Descent

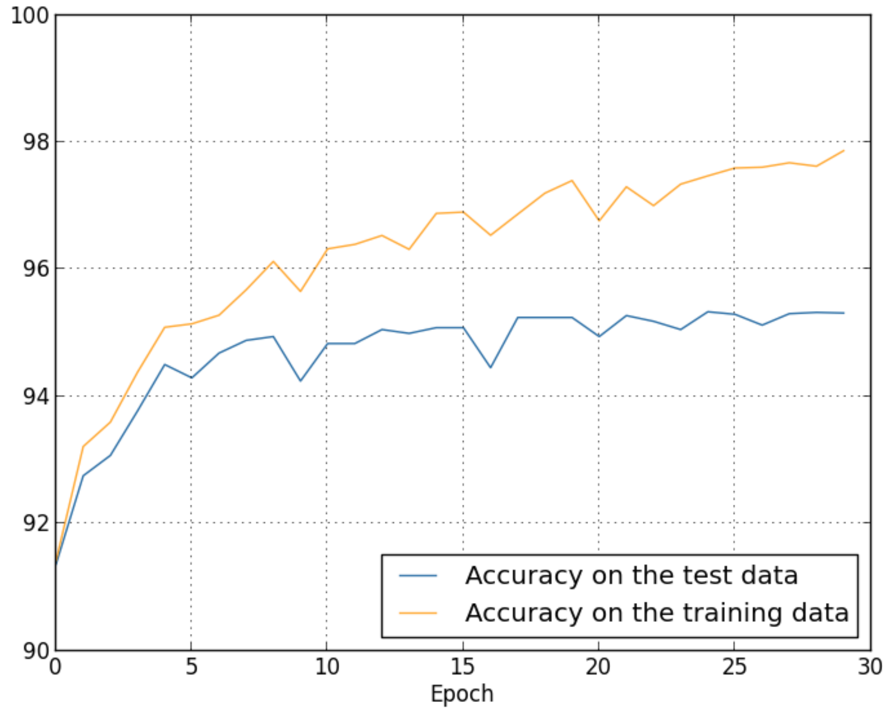
- Batch Gradient Descent
  - Calculates the error for each example in the training dataset, but only updates the model after **all training examples** have been evaluated.
  - One cycle through the entire training dataset is called a training epoch. Therefore, it is often said that batch gradient descent performs model updates at the end of each training epoch.

Mini-Batch Gradient Descent

- Mini-batch gradient descent
  - Splits the training dataset into small batches that are used to calculate model error and update model coefficients.
  - Implementations may choose to sum the gradient over the mini-batch or take the average of the gradient which further reduces the variance of the gradient.
  - Mini-batch gradient descent seeks to find a balance between the robustness of stochastic gradient descent and the efficiency of batch gradient descent.
  - It is the most common implementation of gradient descent used in the field of deep learning.

Overfitting for Deep learning

- Solutions for overfitting on Neural Network
  - Regularization
  - Dropout/DropConnect



Ref: http://neuralnetworksanddeeplearning.com/chap3.html#overfitting_and_regularization