Deep learning for NLP

Eneko Agirre, Gorka Azkune
Ander Barrena, Oier Lopez de Lacalle

@eagirre @gazkune @4nderB @oierldl #dl4nlp

http://ixa2.si.ehu.eus/eneko/dl4nlp

Session 2: Multilayer Perceptron

Plan for the course

- Introduction: machine learning and NLP
- Multilayer perceptron
- Word representation and
  Recurrent neural networks (RNN)
- Sequence-to-Sequence (seq2seq) and
  Machine Translation
- Attention, transformers and
  Natural language inference
- Pre-trained transformers, BERT, GPT
- Bridging the gap between natural languages
  and the visual world

Quiz

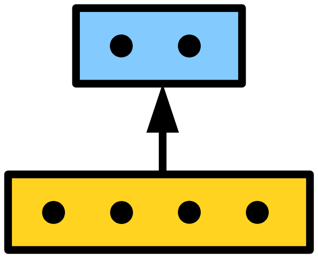
Find definition and slide for the following:

- Supervised machine learning
- Document classification
- Document regression
- Linear regression
- Logistic regression
- Train, development, test
- Softmax classification
- Loss function J
- Gradients ∇
- Stochastic gradient descent
- Learning rate η
- Mini-batch
- Optimizer
- Overfitting
- Regularization, L2, early stopping

Plan for this session

- Multiple layers ~ Deep: MLP
- Learning rate
- More regularization
- Hyperparameters
- Backpropagation and gradients

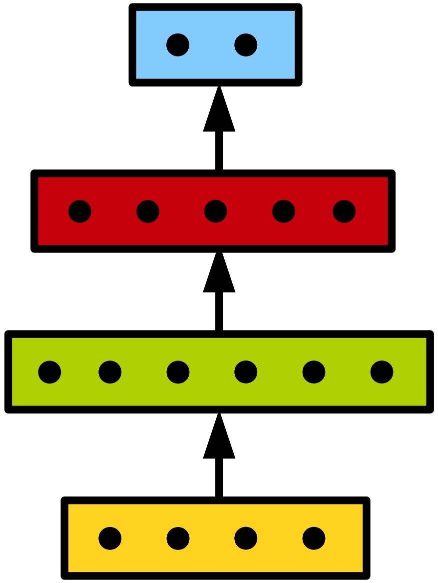
Deep: Multilayer perceptron



Logistic Regression

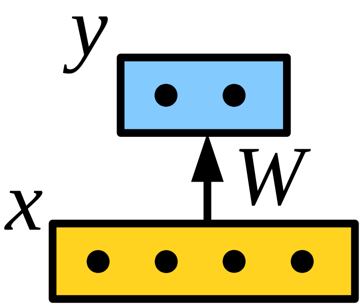
- An input layer – just a feature vector
- An output layer – class probabilities

Deep: Multilayer perceptron



- An input layer
  – just a feature vector
- One or more hidden layers, each computed on the layer below
  – latent features.
- An output layer, based on the top hidden layer
  – class probabilities
- Also known as Feed Forward or Dense

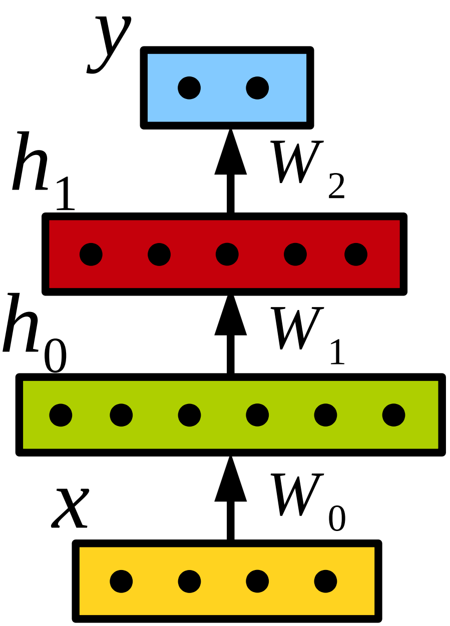
Deep: Multilayer perceptron



Logistic Regression

$$y = softmax(xW + b)$$

Deep: Multilayer perceptron

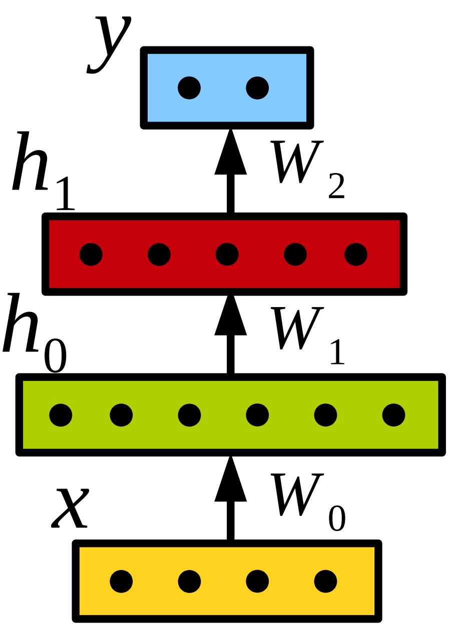


$$y = softmax(h_1 W_2 + b_2)$$

$$h_1 = f(h_0 W_1 + b_1)$$

$$h_0 = f(x W_0 + b_0)$$

Deep: Multilayer perceptron



$$y = softmax(h_1 W_2 + b_2)$$

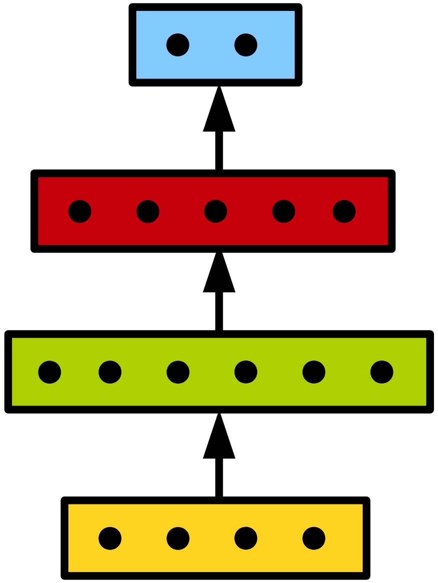
$$h_1 = f(h_0 W_1 + b_1)$$

$$h_0 = f(x W_0 + b_0)$$

softmax

$$J_x = -\log P(y = c | x) = -\log\left(\frac{\exp(h_1 W_2[c])}{\sum_{c' \in C} \exp(h_1 W_2[c'])}\right)$$

Deep: Multilayer perceptron



Layers have the same structure

$$h_i = f(h_{i-1} W_i + b_i)$$

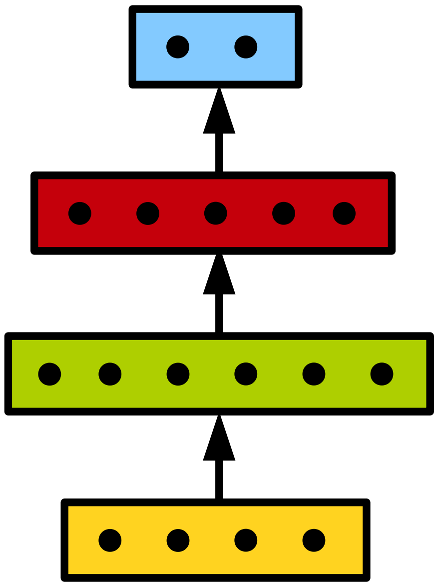
Non-linear functions!

$$Sigmoid: \quad \sigma(x) = \frac{1}{1+\exp(-x)}$$

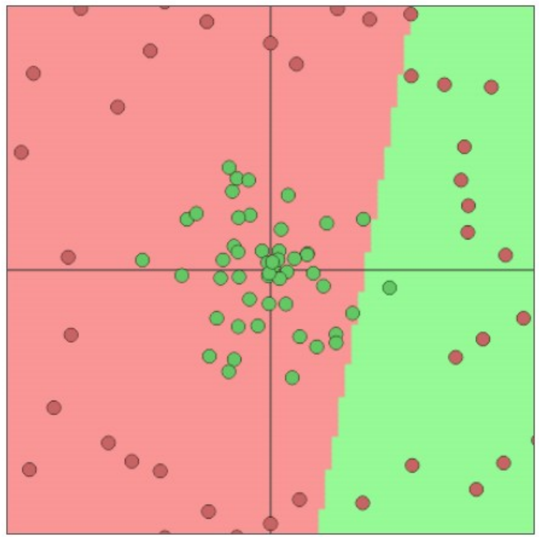
$$Hyperbolic: \quad \tanh(x) = \frac{1-\exp(-2x)}{1+\exp(-2x)}$$

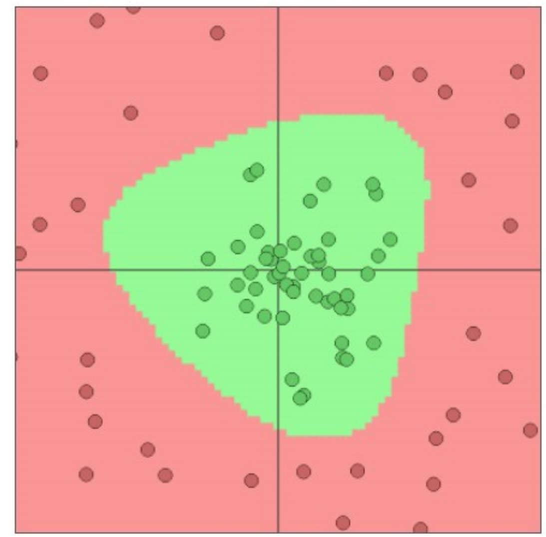
$$Rectified\ linear\ unit: \quad rect(x) = max(0, x)$$

Deep: Multilayer perceptron



Motivations?

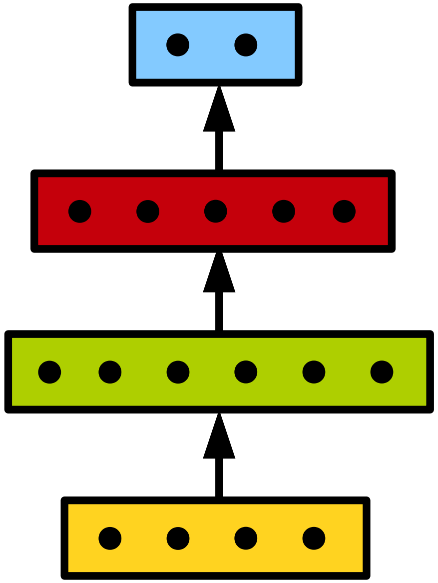




Source: http://cs.stanford.edu/people/karpathy/convnetjs/demo/classify2d.html

Without non-linearities, there is no extra expresivity: a sequence of linear transformations is a linear transformation

Deep: Multilayer perceptron



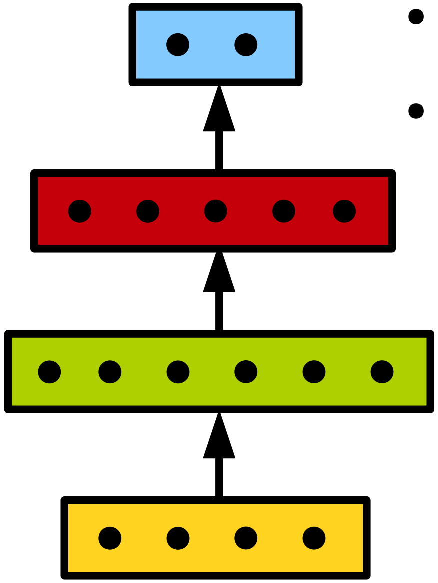
A MLP with one layer can learn to reproduce any function

- Non-linear functions!

Why should we need anything else?

Deep: Multilayer perceptron

Training: SGD (again!)



- Start with random parameters: W (includes bias)
- Each epoch
  - Shuffle training data
  - For each mini-batch (set of K examples)
    - **Compute the loss function (forward)**
    - **Compute the gradient of the loss function (backward)**
    - Update parameters: (learning rate η)

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

  - Measure train and dev. accuracy
- Continue until loss function converges / time is up / dev. accuracy stops increasing

Supervised doc. Classification
Softmax classification

Overfitting and regularization

- W can be very good for training, with enough layers and capacity the model can memorize the training data!
  - Generalize very poorly to test data (= the real world)
- First solution: add a regularizer to the loss function that avoids the model to fit the training data

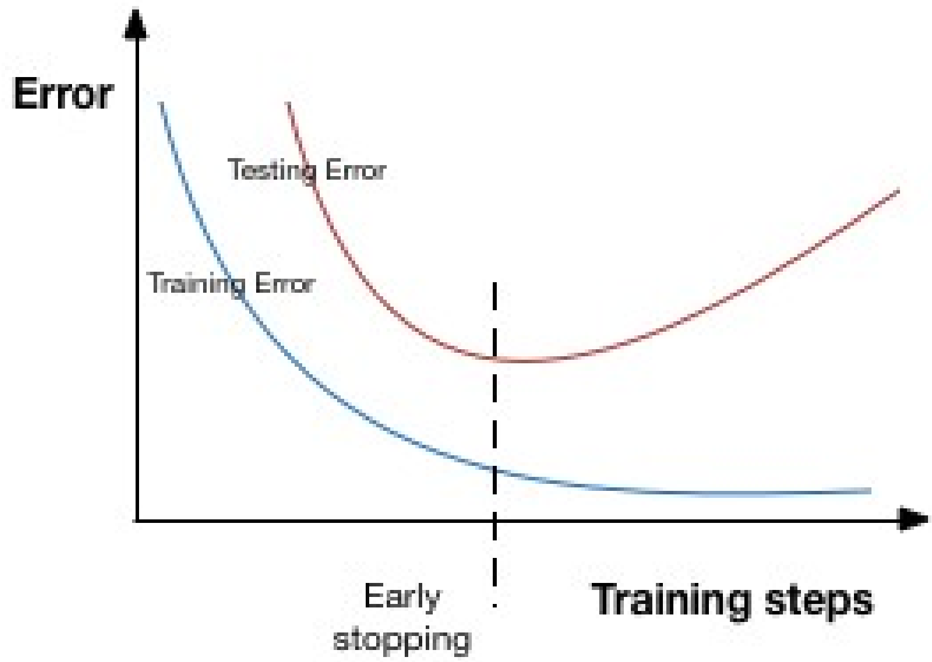
$$J_i(W) = -\log\left(\frac{\exp\left(W_{c_i}^T x\right)}{\sum_{c' \in C} \exp\left(W_{c'}^T x\right)}\right) + \lambda \sum_k W_k^2$$

squared L2 norm

Supervised doc. Classification Softmax classification

Overfitting and regularization

- Overfitting can be seen in this graph
- **Early stopping** finishes training as soon as development error starts to increase



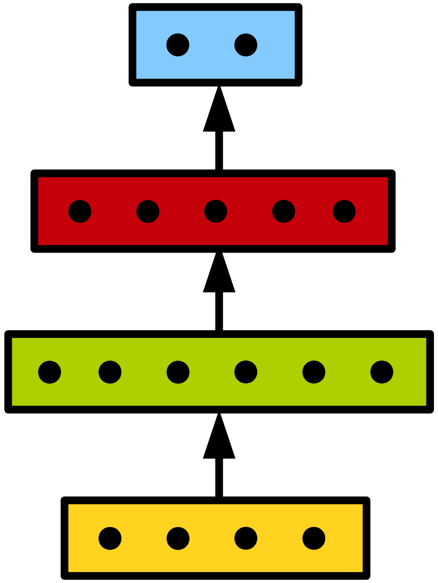
Source: chatbotslife.com

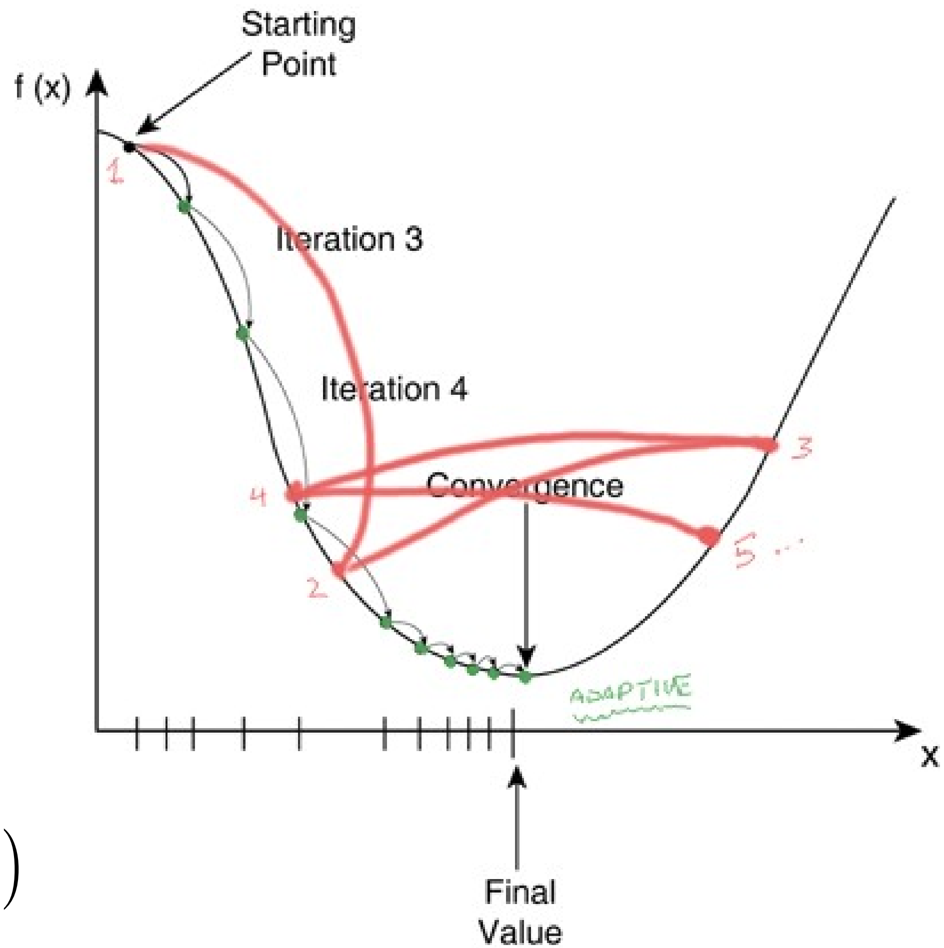
- **Experimental setup**:
  %80 train, %10 development, %10 test (blind!!)
- **Model selection**:
  best accuracy (lowest error) at development

Plan for this session

- Multiple layers ~ Deep: MLP
- Learning rate
- More regularization
- Hyperparameters
- Backpropagation and gradients

SGD convergence: learning rate

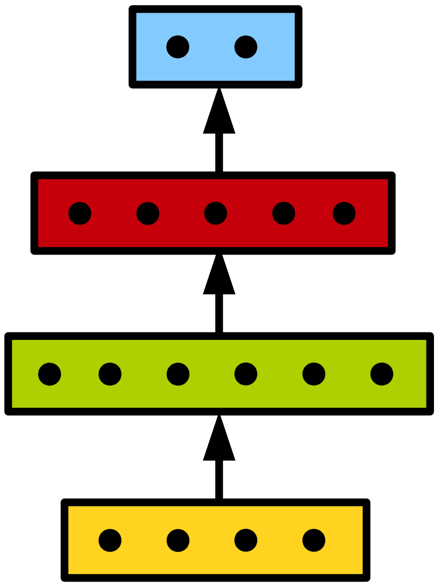


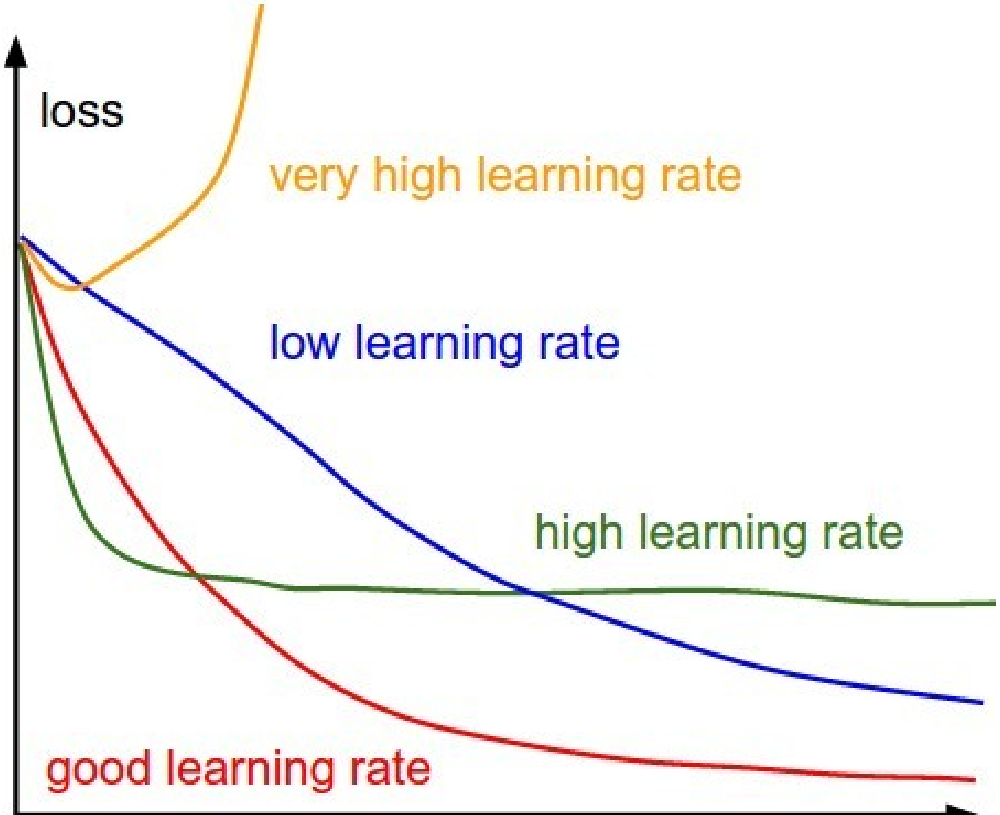


$$W = W - \eta \frac{1}{K} \sum_{i}^{K-1} \nabla J_i(W)$$

Source: cs231n.github.io

SGD convergence: learning rate

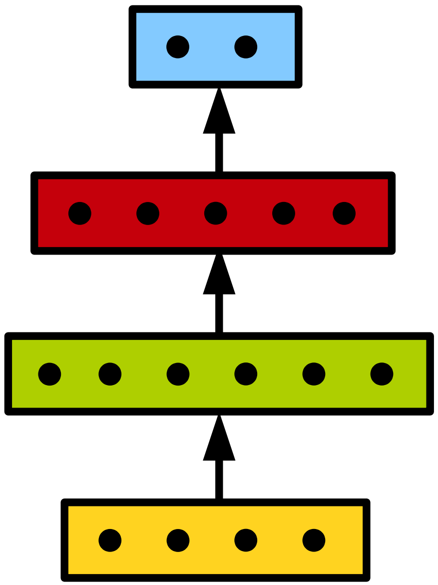




Source: *cs231n.github.io*

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

SGD convergence



Optimizers user diff. learning rates

- Fixed, adam, ...

Don’t expect convergence: use early stopping

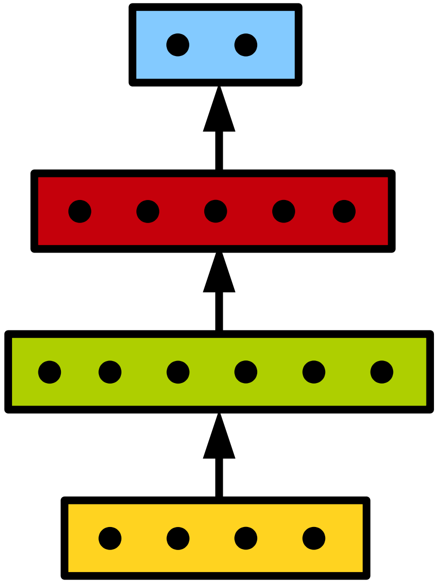
Don’t expect a unique solution: ensembling helps

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

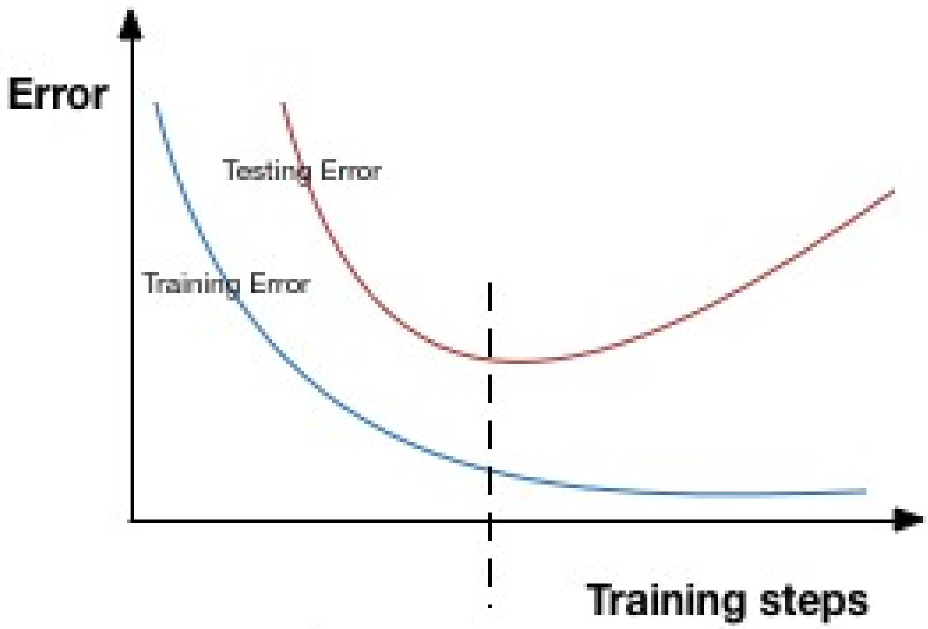
Plan for this session

- Multiple layers ~ Deep: MLP
- Learning rate
- More regularization
- Hyperparameters
- Backpropagation and gradients

Overcapacity and regularization

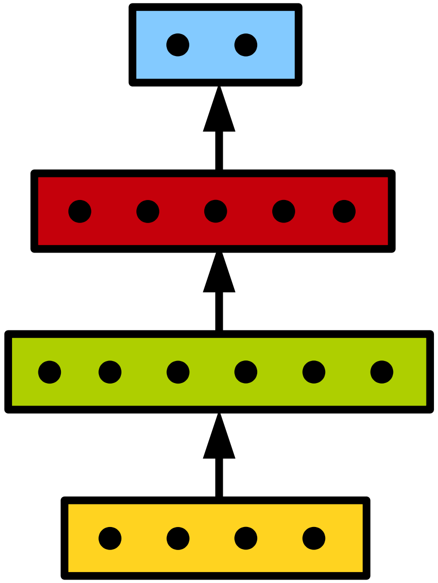


The more layers / hidden units, the more capacity and risk of overfitting (memorize whole training data)!



Source: chatbotslife.com

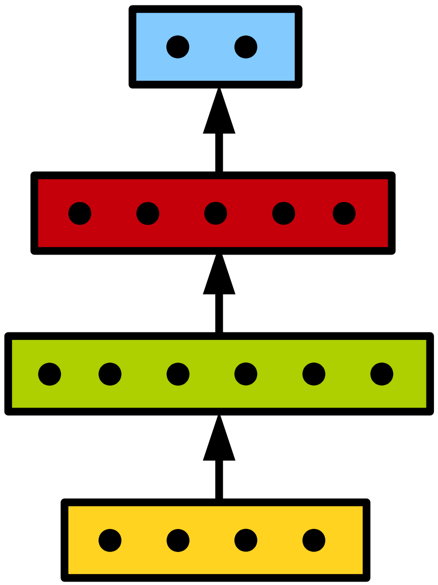
Overcapacity and regularization

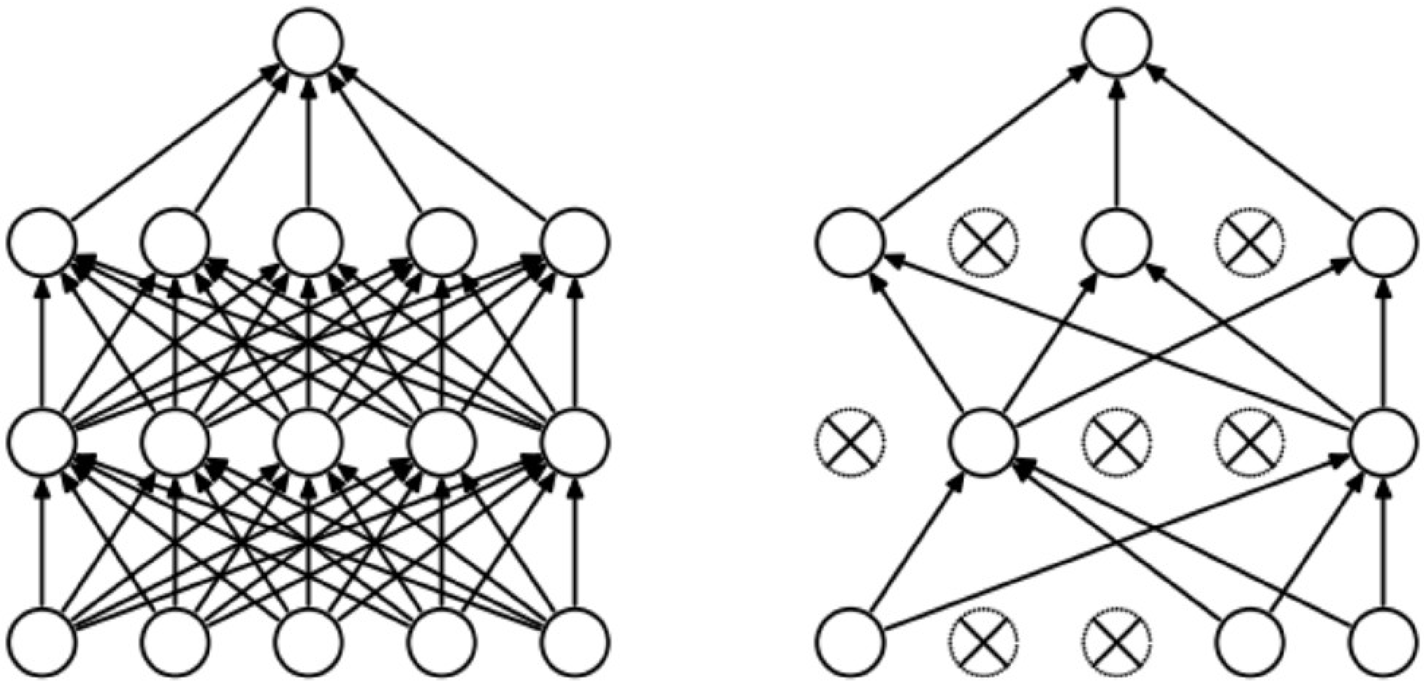


The more layers / hidden units, the more capacity and risk of overfitting (memorize whole training data)!

- L2 on parameters $J_i(W) = \ldots + \lambda \sum_k W_k^2$
- Early stopping
- Dropout
  - At training time deactivate 50% of the activations at random
  - At test time use all activations

Overcapacity and regularization

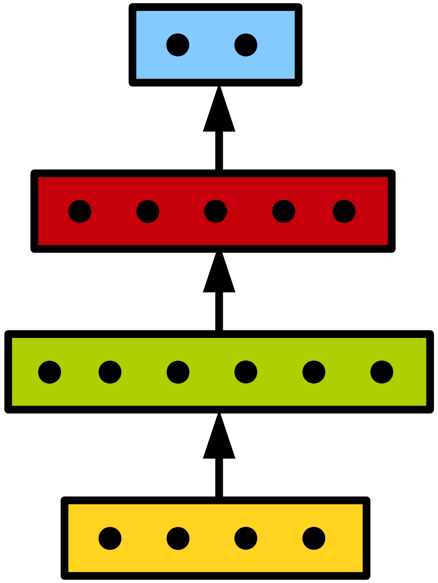




(a) Standard Neural Net (b) After applying dropout.

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

MLP: hyperparameters



Topology: number and size of layers

Non-linearity

Optimizer

Learning-rate

Size of mini-batch

Weight of L2 regularization

Dropout rate

Quiz

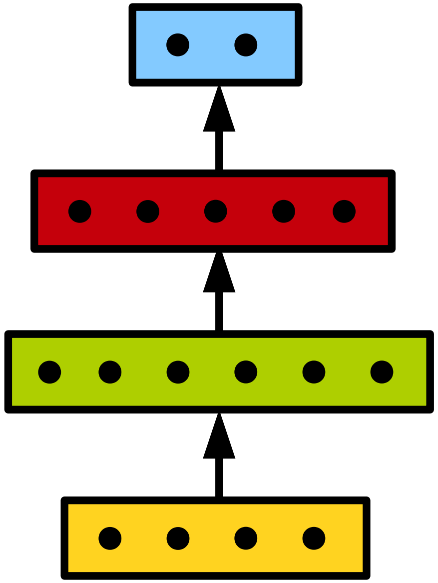
Find definition and slide for the following:

- Supervised machine learning
- Document classification
- Document regression
- Linear regression
- Logistic regression
- Train, development, test
- Softmax classification
- Loss function J
- Gradients ∇
- Stochastic gradient descent
- Learning rate η
- Mini-batch
- Optimizer
- Overfitting
- Regularization, L2, early stopping

Plan for this session

- Multiple layers ~ Deep: MLP
- Learning rate
- More regularization
- Hyperparameters
- Backpropagation and gradients

Computing gradients over graph



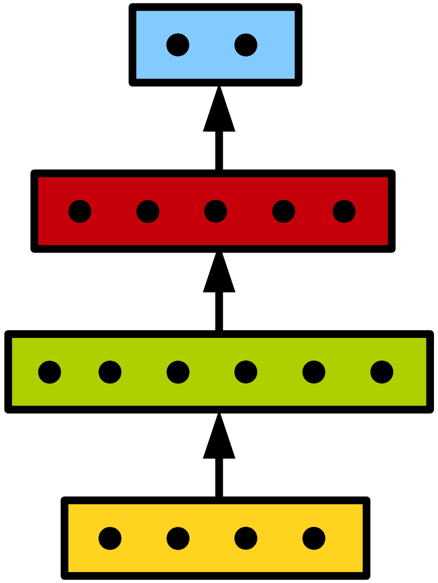
SGD requires gradient of J $\qquad \nabla J$

- For each example x_i $\qquad \dfrac{\delta J_{x_i}}{\delta w}$
- For each parameter $w \in W$

Backpropagation!

- Forward pass to compute J
- Backward pass to compute ∇J

Computing gradients over graph



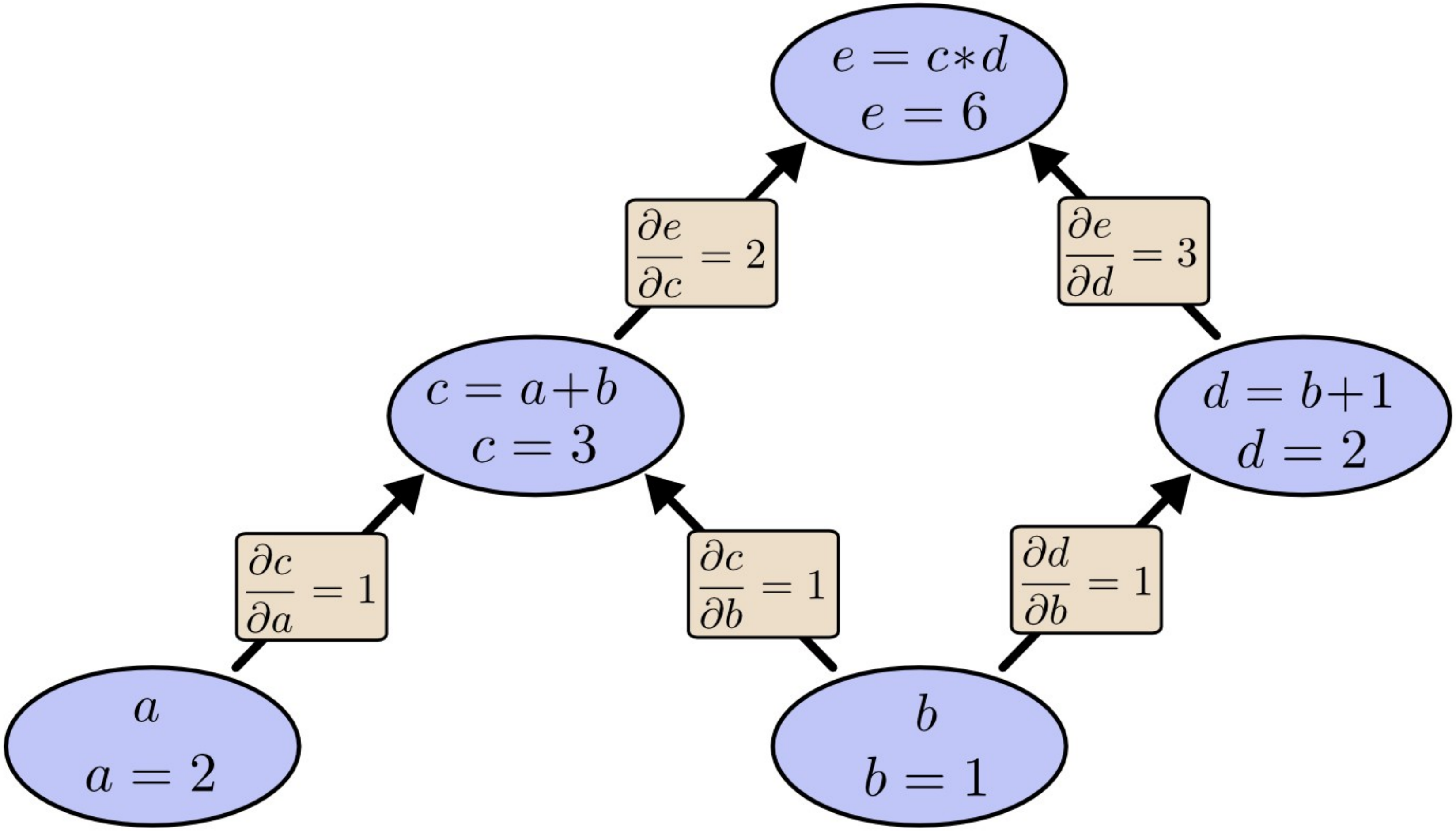
In the backward pass, we apply the chain rule to compute gradient

- $(f \circ g)' = (f' \circ g)\, g'$
- $y=f(u)$ $u=g(x)$

$$\frac{dy}{dx} = \frac{dy}{du}\frac{du}{dx}$$

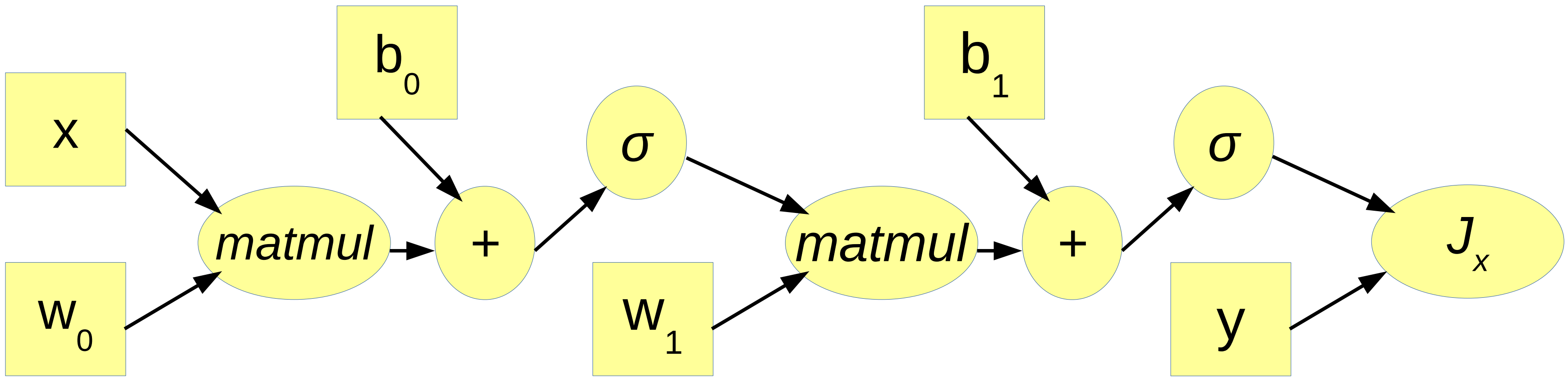
Very efficient!

Computing gradients over graph



Source: colah.github.io/posts/2015-08-Backprop/

Computing gradients over graph



$$J_x = -\log P(y=c|x)$$

Cross-entropy loss
for binary case
y in {0,1}

$$h_1 = \sigma(\boxed{h_0 w_1 + b_1}) \quad \underline{h_1}$$

$$h_0 = \sigma(\boxed{x w_0 + b_0}) \quad \underline{h_0}$$

Simplification: one feature, scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph

$$J_x = -\log P(y=c|x)$$
$$= -\log\left(\exp(\underline{h_1}[c]) / \sum_{c' \in C} \exp(\underline{h_1}[c'])\right)$$
$$= -\log\left(\exp(\underline{h_1}[c]) / \exp(\underline{h_1}[1]) + \exp(\underline{h_1}[0])\right)$$
$$= -\log\left(\exp(\underline{h_1}[c]) / Z\right)$$
$$= -y\log\left(\exp(\underline{h_1}[1]) / Z\right) - (1-y)\log\left(\exp(\underline{h_1}[0]) / Z\right)$$

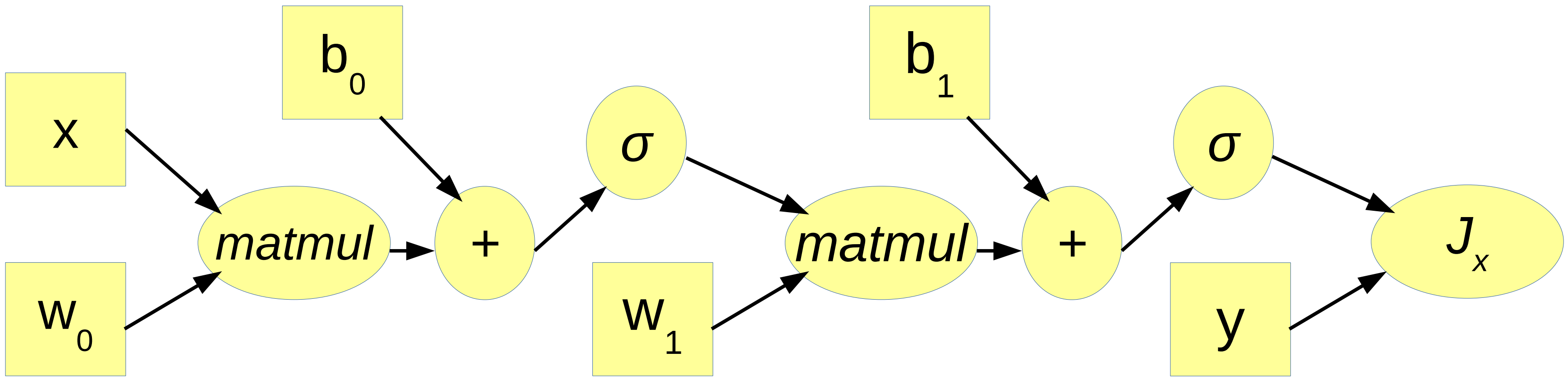
Cross-entropy loss
multiclass
y in {0,1}

$$J_x = -y\log\sigma(\underline{h_1}) - (1-y)\log(1-\sigma(\underline{h_1}))$$

Cross-entropy
binary
y in {0,1}

Chris Yeh: https://chrisyeh96.github.io/2018/06/11/logistic-regression

Computing gradients over graph



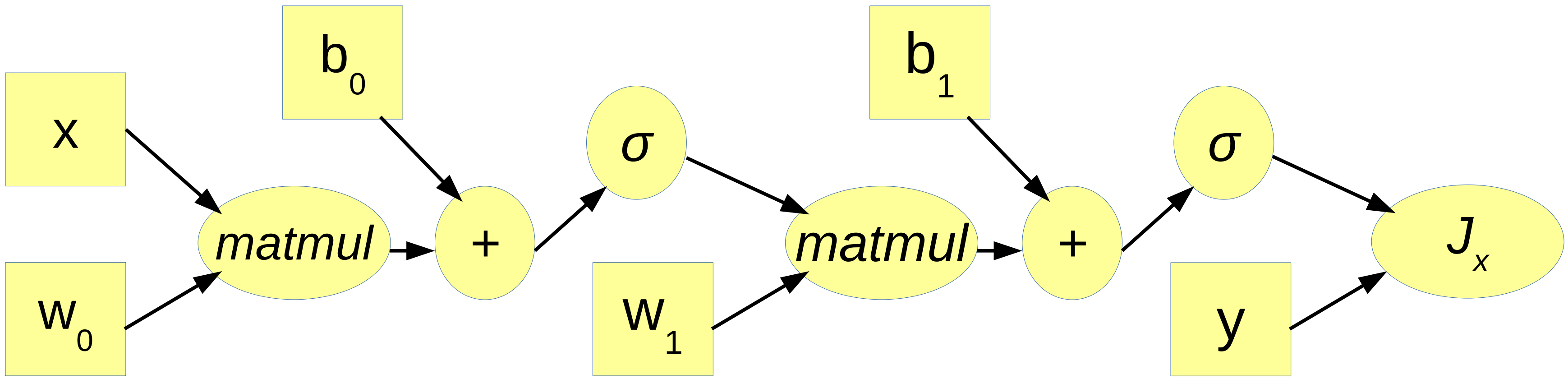
$$J_x = -\log P(y=c|x)$$
$$= -y \log h_1 - (1-y) \log (1-h_1)$$
$$h_1 = \sigma(h_0 w_1 + b_1)$$
$$h_0 = \sigma(x w_0 + b_0)$$

Cross-entropy loss
for binary case
y in {0,1}

Simplification: one feature,
scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



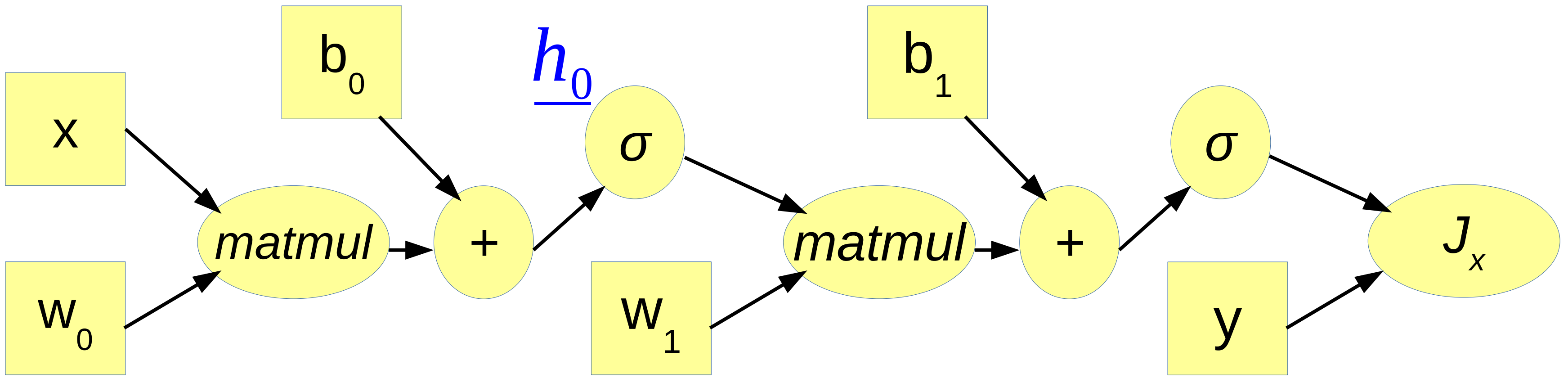
$$J_x = -\log P(y=c|x)$$
$$= -y \log h_1 - (1-y)\log(1-h_1)$$
$$h_1 = \sigma\left(\boxed{h_0 w_1 + b_1}\right) \qquad h_1$$
$$h_0 = \sigma\left(\boxed{x w_0 + b_0}\right) \qquad h_0$$

Cross-entropy loss
for binary case
y in {0,1}

Simplification: one feature,
scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



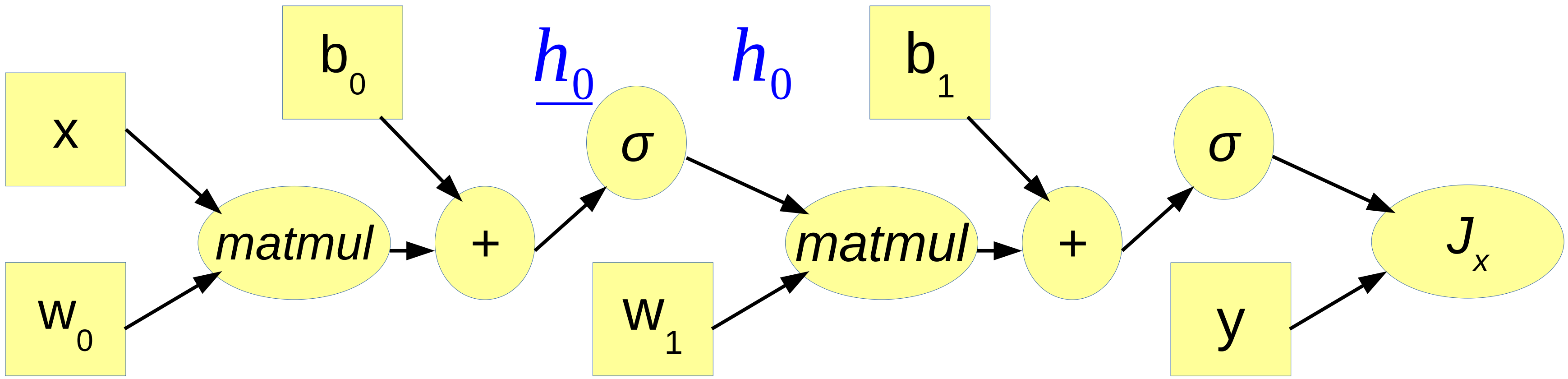
$$J_x = -\log P(y = c \mid x)$$
$$= -y \log h_1 - (1-y)\log(1-h_1)$$
$$h_1 = \sigma\left(\boxed{h_0 w_1 + b_1}\right) \quad h_1$$
$$h_0 = \sigma\left(\boxed{x w_0 + b_0}\right) \quad h_0$$

Cross-entropy loss
for binary case
y in {0,1}

Simplification: one feature, scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



$$J_x = -\log P(y = c | x)$$
$$= -y \log h_1 - (1-y) \log(1-h_1)$$

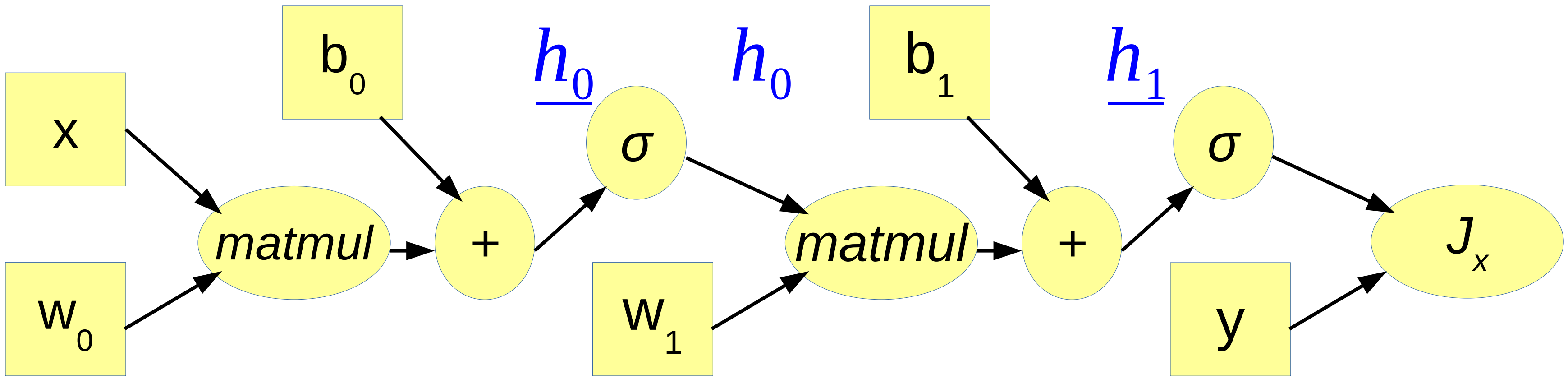
Cross-entropy loss
for binary case
y in {0,1}

$$h_1 = \sigma\left(\boxed{h_0 w_1 + b_1}\right) \quad h_1$$
$$h_0 = \sigma\left(\boxed{x w_0 + b_0}\right) \quad h_0$$

Simplification: one feature, scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



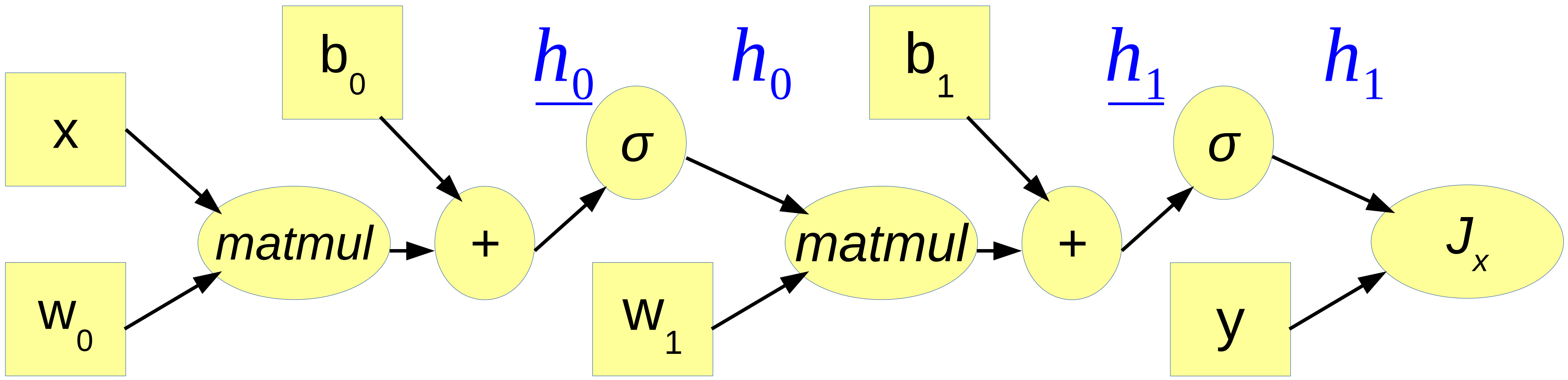
$$J_x = -\log P(y = c \mid x)$$
$$= -y \log h_1 - (1-y)\log(1-h_1)$$
$$h_1 = \sigma\left(\boxed{h_0 w_1 + b_1}\right) \quad h_1$$
$$h_0 = \sigma\left(\boxed{x w_0 + b_0}\right) \quad h_0$$

Cross-entropy loss
for binary case
y in {0,1}

Simplification: one feature, scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



$$J_x = -\log P(y=c|x)$$
$$= -y \log h_1 - (1-y) \log(1-h_1)$$

Cross-entropy loss
for binary case
y in {0,1}

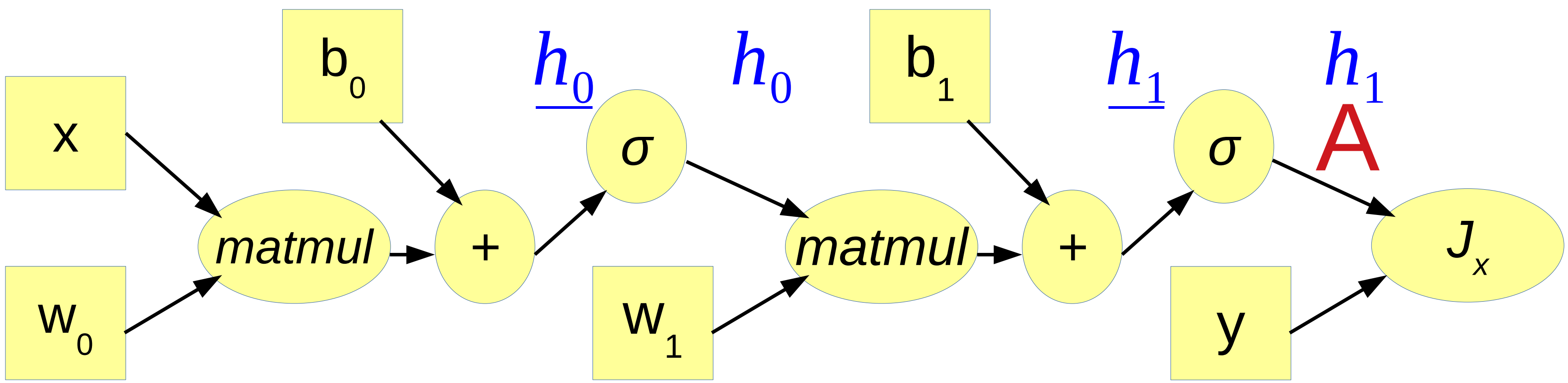
$$h_1 = \sigma(\boxed{h_0 w_1 + b_1}) \quad \underline{h_1}$$

$$h_0 = \sigma(\boxed{x w_0 + b_0}) \quad \underline{h_0}$$

Simplification: one feature, scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



$$J_x = -\log P(y=c|x)$$
$$= -y \log h_1 - (1-y)\log(1-h_1)$$

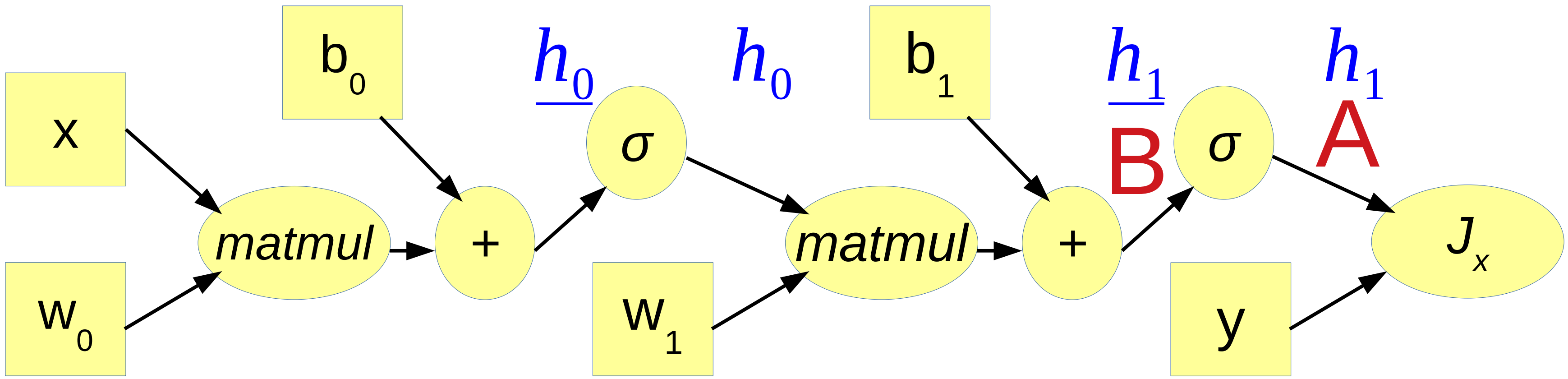
$$h_1 = \sigma\left(\boxed{h_0 w_1 + b_1}\right) \quad h_1$$
$$h_0 = \sigma\left(\boxed{x w_0 + b_0}\right) \quad h_0$$

Cross-entropy loss
for binary case
y in {0,1}

Simplification: one feature,
scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



$$J_x = -\log P(y=c|x)$$
$$= -y \log h_1 - (1-y) \log(1-h_1)$$

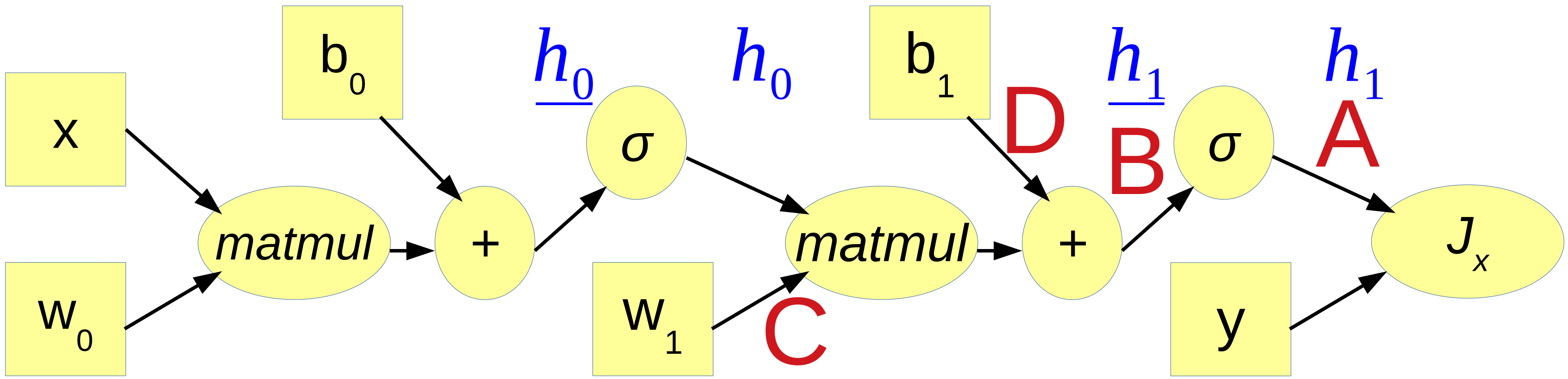
$$h_1 = \sigma\left(\boxed{h_0 w_1 + b_1}\right) \quad h_1$$
$$h_0 = \sigma\left(\boxed{x w_0 + b_0}\right) \quad h_0$$

Cross-entropy loss
for binary case
y in {0,1}

Simplification: one feature,
scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



$$J_x = -\log P(y=c|x)$$
$$= -y \log h_1 - (1-y)\log(1-h_1)$$

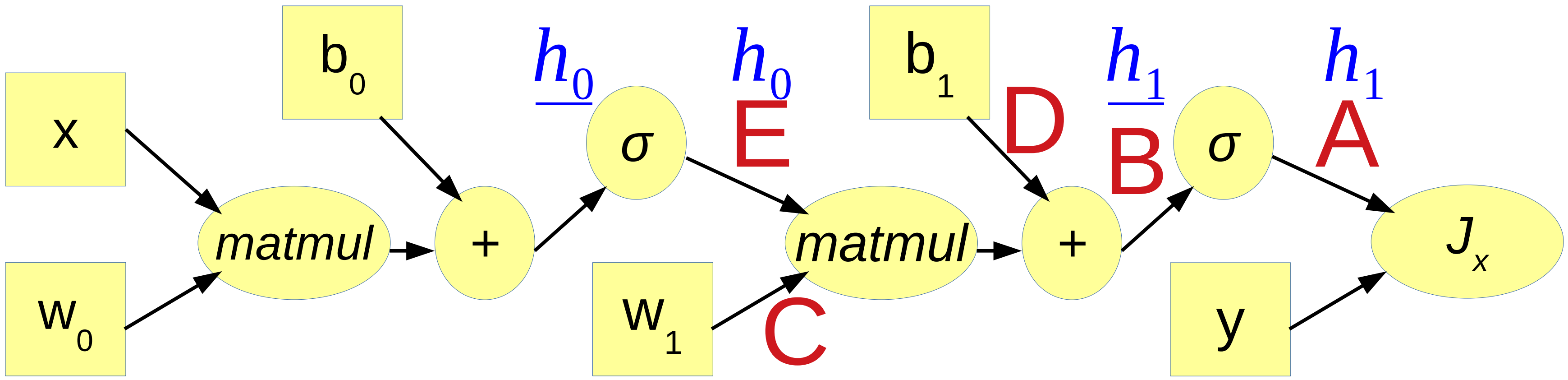
Cross-entropy loss
for binary case
y in {0,1}

$$h_1 = \sigma\left(\boxed{h_0 w_1 + b_1}\right) \quad h_1$$
$$h_0 = \sigma\left(\boxed{x w_0 + b_0}\right) \quad h_0$$

Simplification: one feature, scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



$$J_x = -\log P(y=c|x)$$
$$= -y \log h_1 - (1-y) \log(1-h_1)$$

$$h_1 = \sigma\left(\boxed{h_0 w_1 + b_1}\right) \quad h_1$$

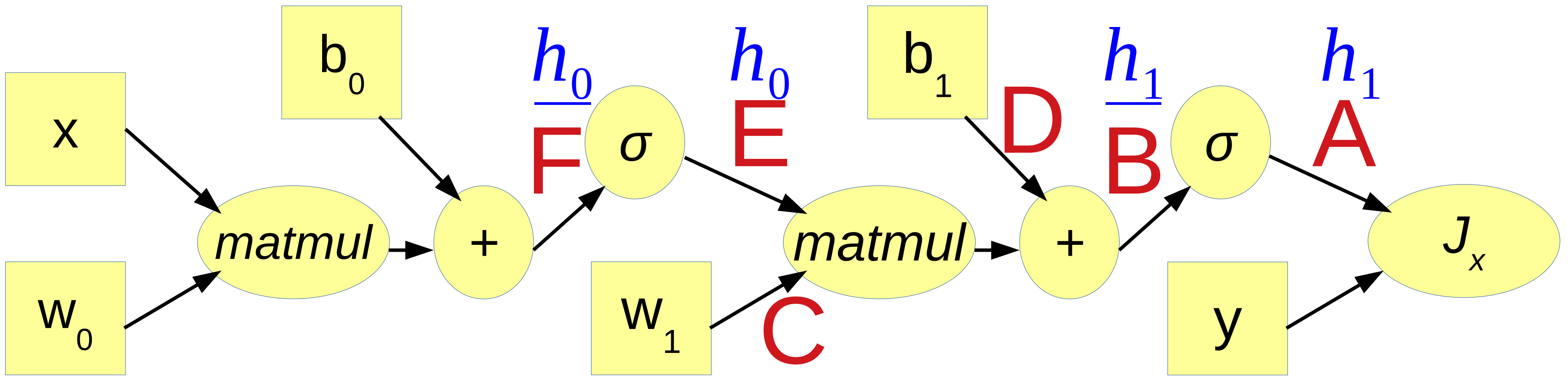
$$h_0 = \sigma\left(\boxed{x w_0 + b_0}\right) \quad h_0$$

Cross-entropy loss
for binary case
y in {0,1}

Simplification: one feature,
scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



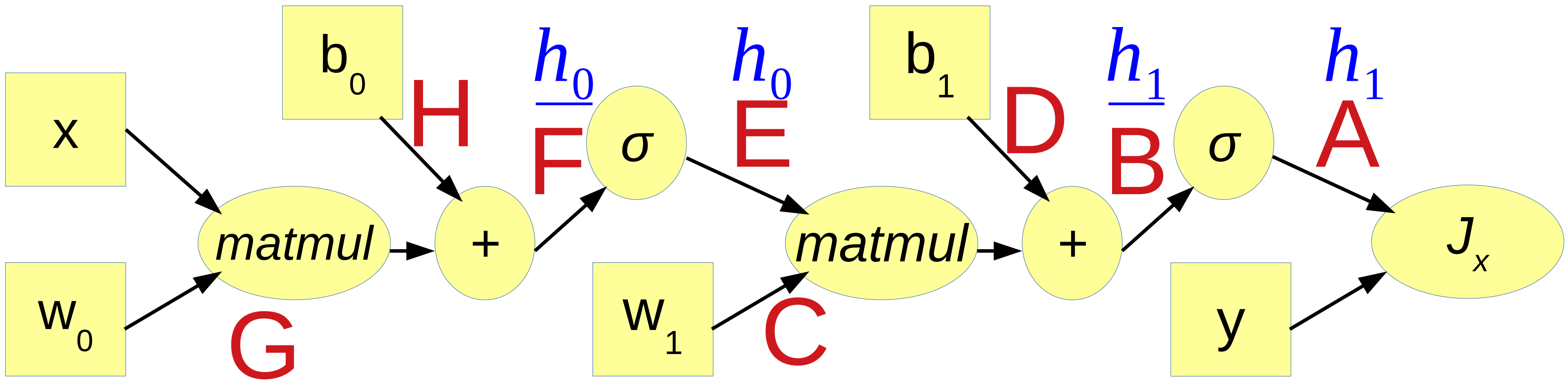
$$J_x = -\log P(y=c|x)$$
$$= -y \log h_1 - (1-y)\log(1-h_1)$$
$$h_1 = \sigma(\boxed{h_0 w_1 + b_1}) \quad \underline{h_1}$$
$$h_0 = \sigma(\boxed{x w_0 + b_0}) \quad \underline{h_0}$$

Cross-entropy loss
for binary case
y in {0,1}

Simplification: one feature, scalar variables

Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



$$J_x = -\log P(y=c|x)$$
$$= -y\log h_1 - (1-y)\log(1-h_1)$$

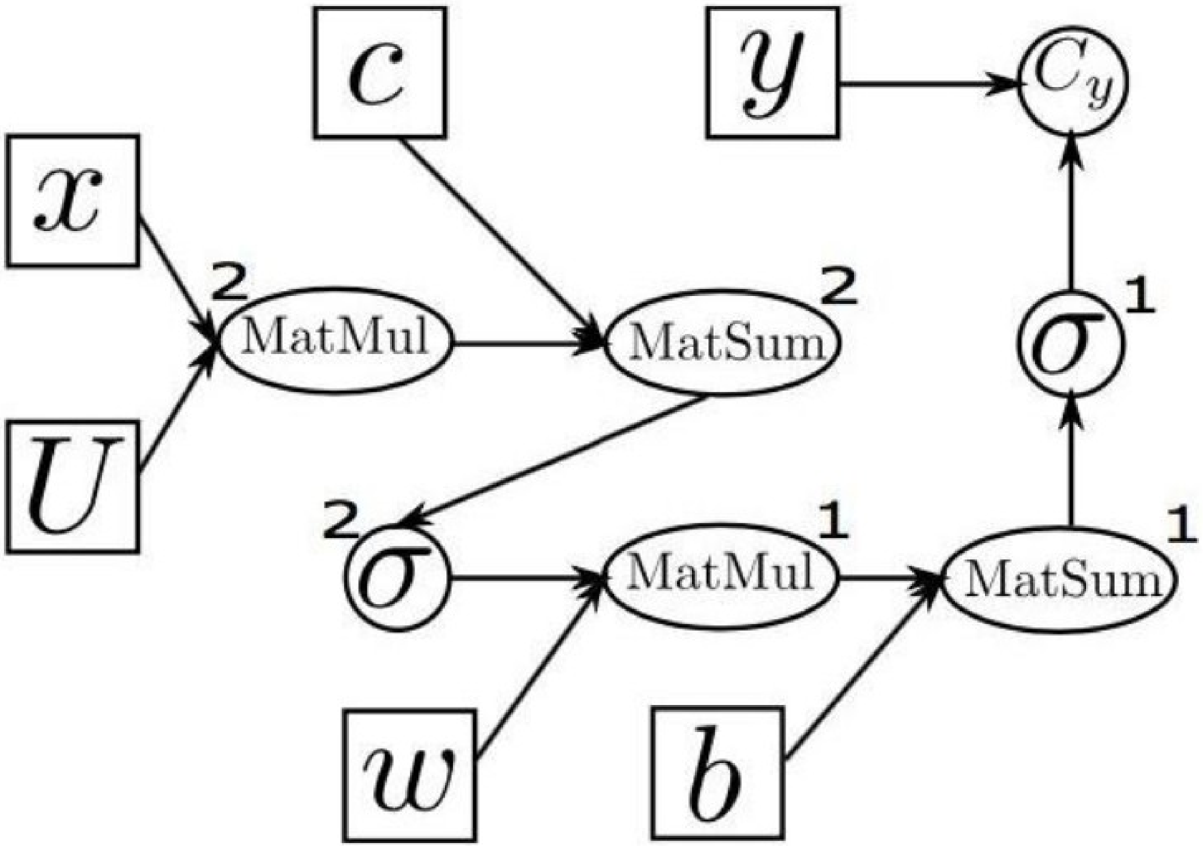
$$h_1 = \sigma\left(\boxed{h_0 w_1 + b_1}\right) \quad \underline{h_1}$$
$$h_0 = \sigma\left(\boxed{x w_0 + b_0}\right) \quad \underline{h_0}$$

Cross-entropy loss
for binary case
y in {0,1}

Simplification: one feature, scalar variables

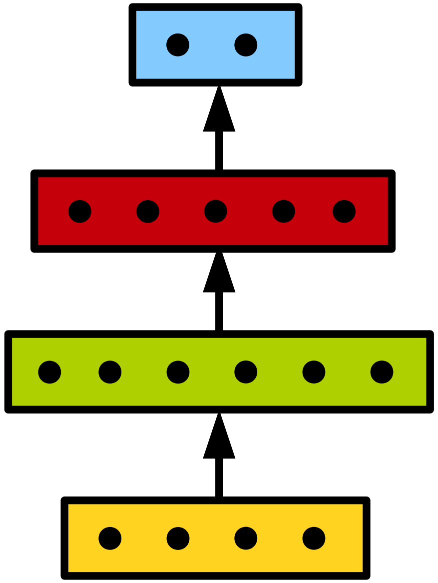
Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



Kyunghyun Cho: https://github.com/nyu-dl/NLP_DL_Lecture_Note/raw/master/lecture_note.pdf (p. 19)

Computing gradients over graph



All operations in the graph need to be *differentiable*!

- matrix multiplication/addition
- log, exp
- sigmoid, tanh, ReLU (really!)

Many functions are not

- IF, accuracy, BLEU, etc.

=> Neural Turing Machine (differentiable)

THANKS!

Acknowledgements:

- Overall slides: Sam Bowman and Kyunghyun cho (NYU), Chris Manning and Richard Socher (Stanford)
- All source url's listed in the slides.