Backpropagation

Today

- The method for computing *gradients* in neural nets
  - Backpropagation
- A simple *strategy* for optimization
  - (Mini)batch gradient descent
- Next time
  - Fancy strategies for optimization
  - Stochastic gradient descent and variants

Training in general

- Remember from last time
  - We are given a limited sample of training data
  - These examples can be seen as an approximation of the *data-generating distribution* p_{data}
  - Thus, we only have access to $\hat{p}_{\text{data}}$
- We have also seen that in ML we must define a *loss*
  - In general, we minimize an *expected loss* $E_{\mathbf{x},y\sim\hat{p}_{\text{data}}} L(\hat{y}, y)$
  - The expectation just means the average over the *m* training examples
  - For e.g. regression with MSE, we would like to minimize this:

    $\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}(\hat{y}_i - y_i)^2$

From maximum likelihood to loss

- We set up a model *f(**x**)* for the output
  - Linear regression: $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$
  - Logistic regression: $f(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{w}^\top \mathbf{x}}}$
- This model is really an attempt to predict *y* given **x**: $p(y|\mathbf{x})$
- If we know the distribution of our model, we can use ML to optimize for **w**

$$\mathbf{w}_{\mathrm{ML}} = \arg\max \mathcal{L}(\mathbf{w}) = \arg\max \sum_{i=1}^{m} \log p(y_i|\mathbf{x}_i)$$

  - Least squares: *p(y|x)* is Gaussian
  - Classification: *p(y|x)* is multinomial (multinoulli)

From maximum likelihood to loss

- If we now take the negative of the LL, $\mathbf{w}_{\text{ML}} = \arg\max \mathcal{L}(\mathbf{w}) = \arg\max \sum_{i=1}^{m} \log p(y_i|\mathbf{x}_i)$
  - we get something called the *cross-entropy* (CE): $-\sum_{i=1}^{m} \log p(y_i|\mathbf{x}_i)$
- This should be understood as:
  - The cross-entropy quantifies the statistical divergence between the outputs of the model and the examples in the training set
- Therefore, maximizing LL is equivalent to minimizing the CE:

$$\mathbf{w}_{\text{ML}} = \arg\min - \sum_{i=1}^{m} \log p(y_i|\mathbf{x}_i)$$

From maximum likelihood to loss

- In statistical machine learning the CE is our *loss* or *cost* function, denoted *L*
- In practice we usually take the average over the presented examples

  $L = -\frac{1}{m}\sum_{i=1}^{m} \log p(y_i|\mathbf{x}_i)$
- Remember that *we* specify the model *f* that tries to predict $p(y|\mathbf{x})$
  - The output we can call: $\hat{y} = f(\mathbf{x})$
- Therefore, we usually see *L* as a function of our function output and the true *y*
  - MSE loss: $L(\hat{y}, y) = \frac{1}{m}\sum_{i=1}^{m} \|y_i - \hat{y}_i)\|^2$
  - Log loss: $L(\hat{y}, y) = -\frac{1}{m}\sum_{i=1}^{m} y_i \log \hat{y}_i$

From loss to gradient to learning

$$L(\hat{y}, y) = \frac{1}{m} \sum_{i=1}^{m} \|y_i - \hat{y}_i)\|^2$$

$$L(\hat{y}, y) = -\frac{1}{m} \sum_{i=1}^{m} y_i \log \hat{y}_i$$

- Based on our loss *L*, we can derive a gradient w.r.t. our prediction :
- But how do we find the gradient w.r.t. the parameters (weights/biases)?

$\hat{y}$

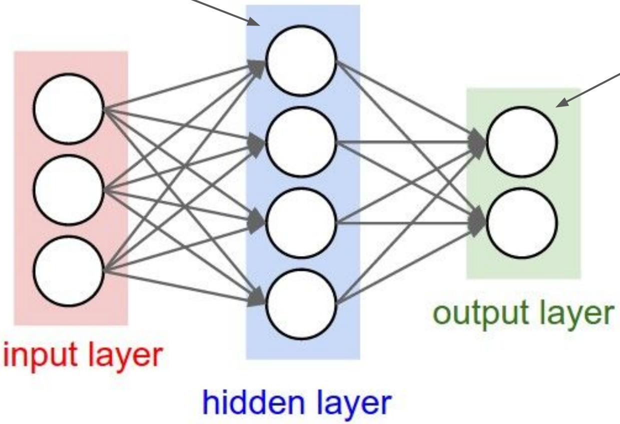
Computational graph

- $\hat{\mathbf{y}}$ = model($\mathbf{x}$)
- L = lossfunction($\hat{\mathbf{y}}$, $\mathbf{y}$)
- All part of the computational graph

$$\boxed{\mathbf{h} = f_1(\mathbf{a}_1) = f_1(\mathbf{W_1}\mathbf{x} + \mathbf{b_1})}$$

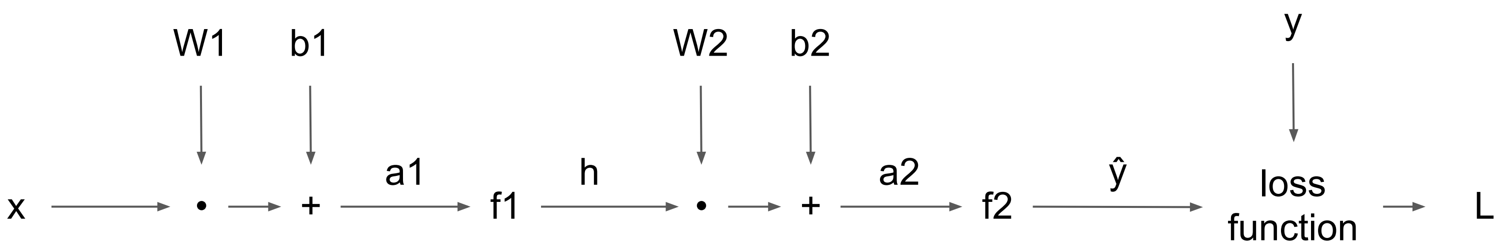
$$\boxed{\hat{\mathbf{y}} = f_2(\mathbf{a}_2) = f_2(\mathbf{W_2}\mathbf{h} + \mathbf{b_2})}$$

http://cs231n.github.io/neural-networks-1



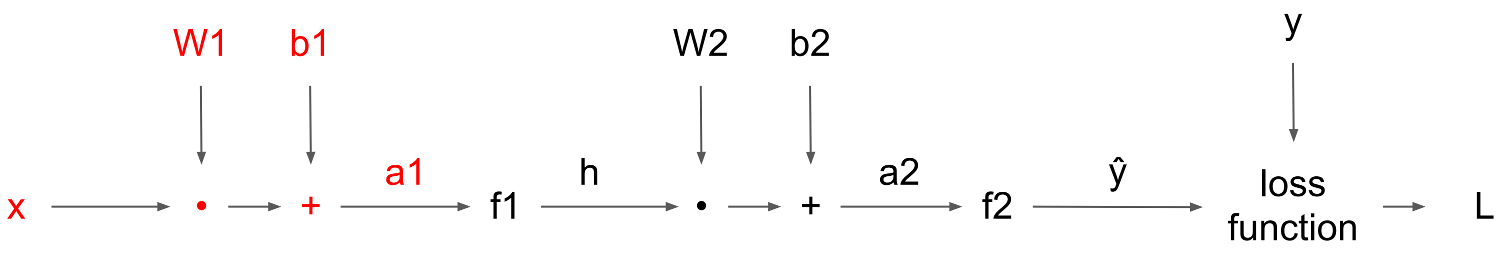
Computational graph

- **ŷ** = model(**x**)
- *L* = lossfunction(**ŷ**, **y**)
- All part of the computational graph



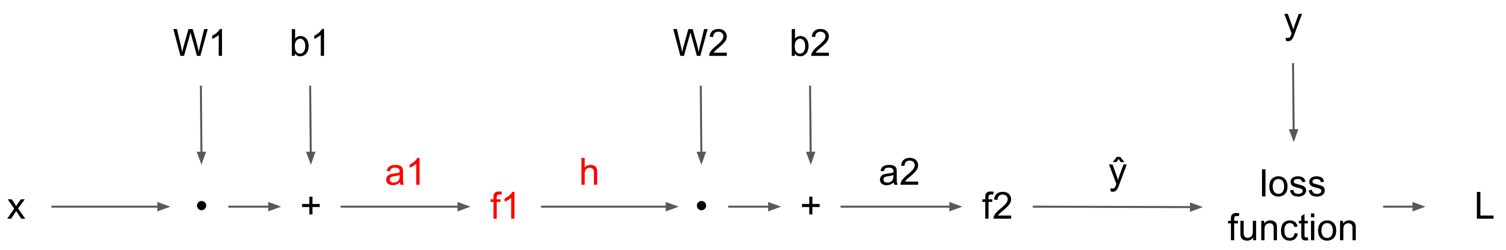
Finding gradients for our weights

- From the chain rule, we get: $$\frac{\partial L}{\partial \mathbf{W_1}} = \frac{\partial L}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial \mathbf{W_1}} = \frac{\partial L}{\partial \mathbf{a_1}} \mathbf{x}^T$$ $$\frac{\partial L}{\partial \mathbf{b_1}} = \frac{\partial L}{\partial \mathbf{a_1}} \frac{\partial \mathbf{a_1}}{\partial \mathbf{b_1}} = \frac{\partial L}{\partial \mathbf{a_1}}$$
- They depend on $\frac{\partial L}{\partial \mathbf{a_1}}$



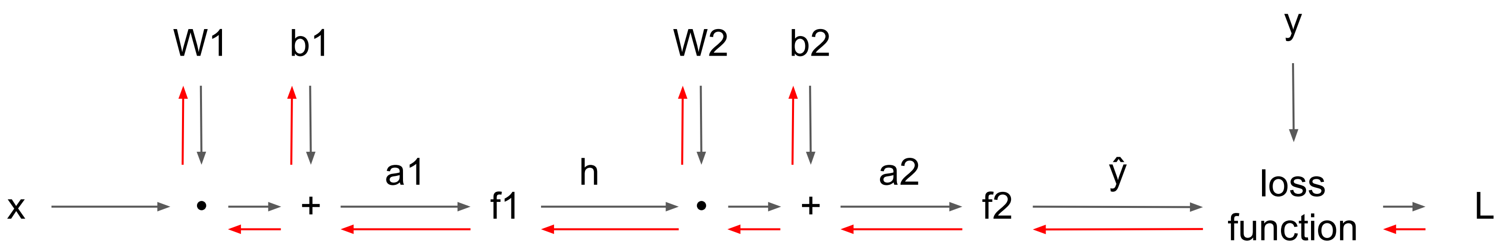
Finding gradients for our weights

- Again, chain rule: $\frac{\partial L}{\partial \mathbf{a_1}} = \frac{\partial L}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{a_1}} = \frac{\partial L}{\partial \mathbf{h}} J_{f_1}[\mathbf{a_1}]$
- Notice that the dependencies for the gradients go forward in the computational graph
- The gradients thus need to be **back propagated** from the loss



Backprop

- Back propagate the gradients starting from the loss



$$\frac{\partial L}{\partial \mathbf{a_2}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{a_2}}$$

$$\frac{\partial L}{\partial L} = 1$$

$$\frac{\partial L}{\partial \hat{\mathbf{y}}} = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y})$$

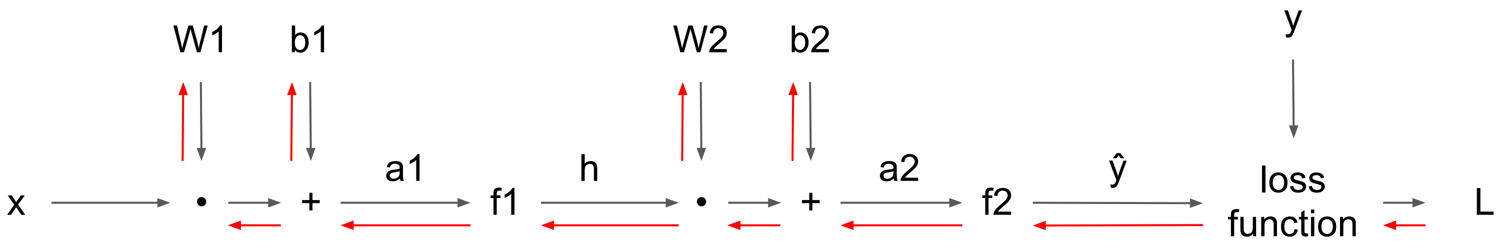
Deltas

- Usually we can go directly to the derivative at the output layer
- This is achieved by using an appropriate output activation + loss pair
- **Linear outputs + MSE loss** (one output vector)

$$L(\hat{\mathbf{y}}, \mathbf{y}) = \frac{1}{2}\|\hat{\mathbf{y}} - \mathbf{y}\|^2$$

$$\hat{\mathbf{y}} = f(\mathbf{a}_2) = \mathbf{a}_2$$

$$\boxed{\frac{\partial L}{\partial \mathbf{a}_2} = \hat{\mathbf{y}} - \mathbf{y}}$$



$$\frac{\partial L}{\partial \mathbf{a_2}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{a_2}}$$

$$\frac{\partial L}{\partial L} = 1$$

$$\frac{\partial L}{\partial \hat{\mathbf{y}}} = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y})$$

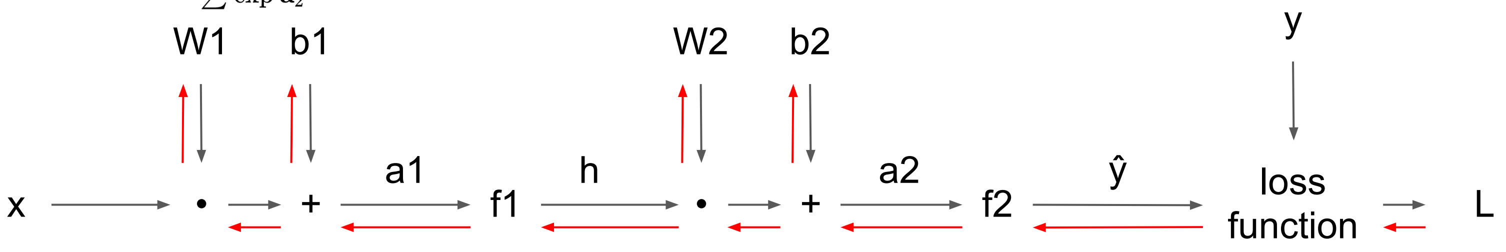
Deltas

- Usually we can go directly to the derivative at the output layer
- This is achieved by using an appropriate output activation + loss pair
- **Softmax outputs + log loss** (again only one output vector)

$$L(\hat{\mathbf{y}}, \mathbf{y}) = -\mathbf{y}^\top \log \hat{\mathbf{y}}$$

$$\hat{\mathbf{y}} = f(\mathbf{a}_2) = \frac{\exp \mathbf{a}_2}{\sum \exp \mathbf{a}_2}$$

$$\boxed{\frac{\partial L}{\partial \mathbf{a}_2} = \hat{\mathbf{y}} - \mathbf{y}}$$



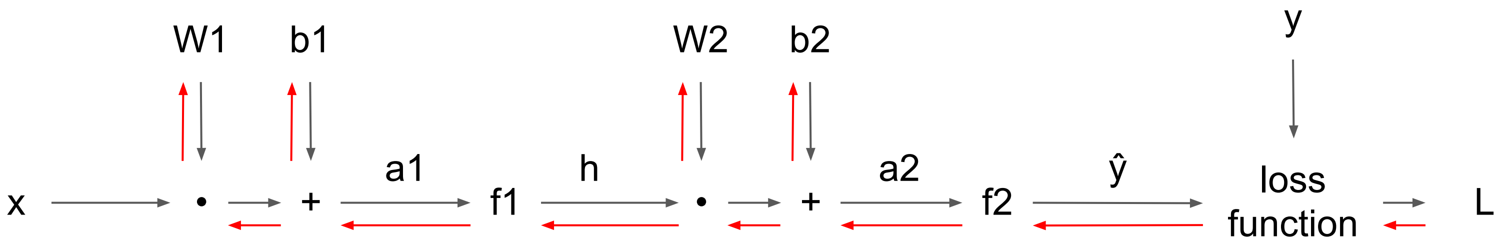
$$\frac{\partial L}{\partial \mathbf{a_2}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{a_2}}$$

$$\frac{\partial L}{\partial L} = 1$$

$$\frac{\partial L}{\partial \hat{\mathbf{y}}} = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y})$$

Back to backprop

- This is where we currently are



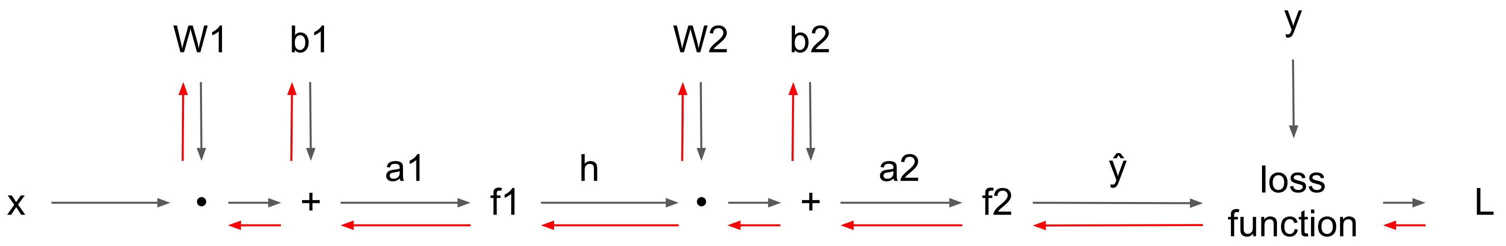
$$\frac{\partial L}{\partial \mathbf{a_2}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{a_2}}$$

$$\frac{\partial L}{\partial L} = 1$$

$$\frac{\partial L}{\partial \hat{\mathbf{y}}} = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y})$$

Backprop

- Now we go further back



$$\frac{\partial L}{\partial \mathbf{a_1}} = \frac{\partial L}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{a_1}} \qquad \frac{\partial L}{\partial \mathbf{a_2}} = \frac{\partial L}{\partial \hat{\mathbf{y}}} \frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{a_2}} \qquad \frac{\partial L}{\partial L} = 1$$

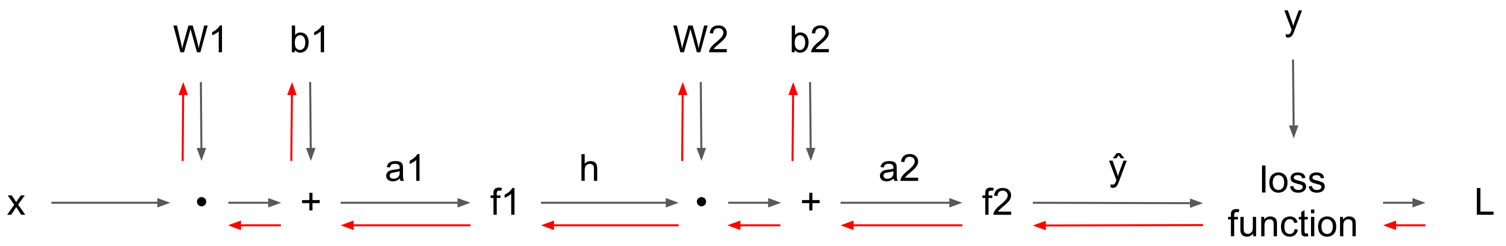
$$\frac{\partial L}{\partial \mathbf{h}} = \frac{\partial L}{\partial \mathbf{a_2}} \frac{\partial \mathbf{a_2}}{\partial \mathbf{h}} \qquad \frac{\partial L}{\partial \hat{\mathbf{y}}} = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y})$$

Backprop

- What about the parameters?

$$\frac{\partial L}{\partial \mathbf{b_i}} = \frac{\partial L}{\partial \mathbf{a_i}}\frac{\partial \mathbf{a_i}}{\partial \mathbf{b_i}} = \frac{\partial L}{\partial \mathbf{a_i}}$$

$$\frac{\partial L}{\partial \mathbf{W_i}} = \frac{\partial L}{\partial \mathbf{a_i}}\frac{\partial \mathbf{a_i}}{\partial \mathbf{W_i}}, \quad \frac{\partial \mathbf{a_1}}{\partial \mathbf{W_1}} = \mathbf{x}^T, \quad \frac{\partial \mathbf{a_2}}{\partial \mathbf{W_2}} = \mathbf{h}^T$$



$$\frac{\partial L}{\partial \mathbf{a_1}} = \frac{\partial L}{\partial \mathbf{h}}\frac{\partial \mathbf{h}}{\partial \mathbf{a_1}} \qquad \frac{\partial L}{\partial \mathbf{a_2}} = \frac{\partial L}{\partial \hat{\mathbf{y}}}\frac{\partial \hat{\mathbf{y}}}{\partial \mathbf{a_2}} \qquad \frac{\partial L}{\partial L} = 1$$

$$\frac{\partial L}{\partial \mathbf{h}} = \frac{\partial L}{\partial \mathbf{a_2}}\frac{\partial \mathbf{a_2}}{\partial \mathbf{h}} \qquad \frac{\partial L}{\partial \hat{\mathbf{y}}} = \nabla_{\hat{\mathbf{y}}} L(\hat{\mathbf{y}}, \mathbf{y})$$

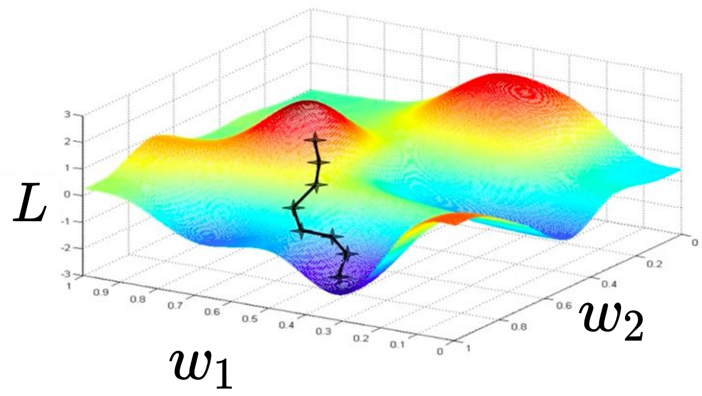
Gradient descent

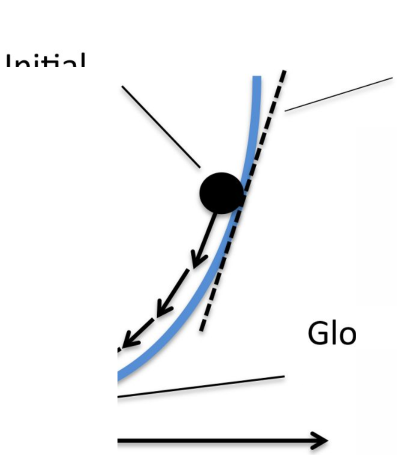
- We have
  - A “function” that maps input data to outputs using parameters (**W**,**b**)
  - A loss *L* that quantifies the difference between our predictions and the desired outputs
  - A gradient **g** of *L* w.r.t. the function parameters

Gradient descent

- Remember that *f* is highly non-linear and *L* is generally non-convex
- In GD, we follow the gradient of *L* with *small steps*
- We need to use *random initialization*

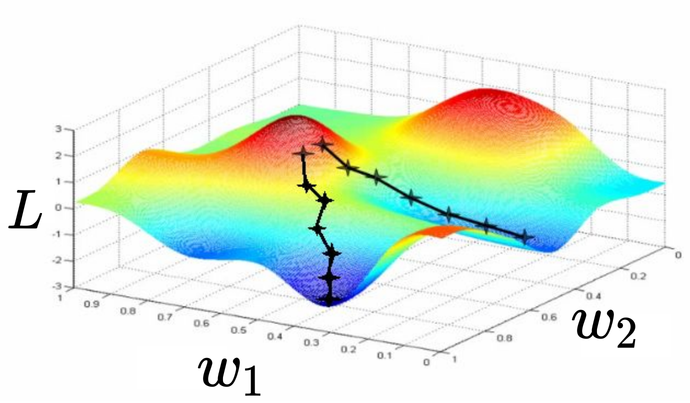
http://onlineresize.club/pictures-club.html





gradient-descent-aynk-7cbe95a77...

http://blog.datumbox.com/wp-content/uploads/2013/10/gradient-descent.png



Batch gradient descent

- The basic idea of GD is to just follow the negative of **g**:

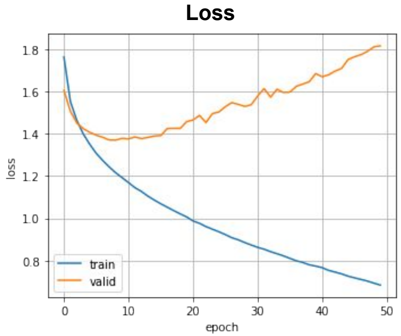
  $$\mathbf{W} \leftarrow \mathbf{W} - \mathbf{g}$$

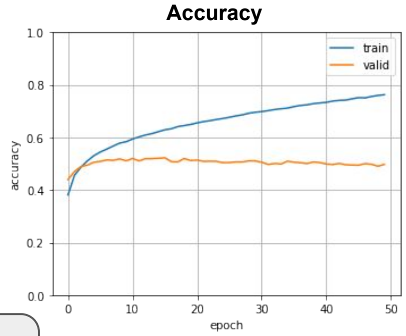
- This will in general *not work* - why?
- We need to use a small *learning rate*:

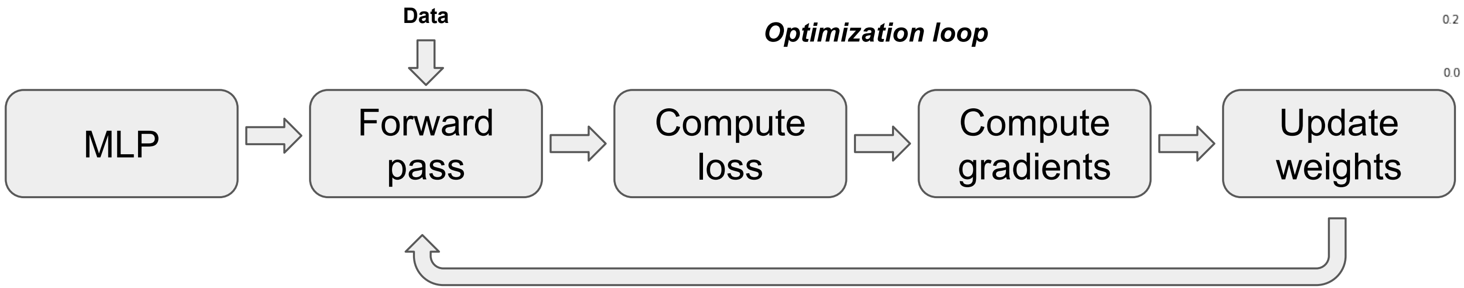
  $$\mathbf{W} \leftarrow \mathbf{W} - \epsilon\mathbf{g}$$

Last week - MLP using PyTorch

- Introduction to PyTorch
- MLP classifier on CIFAR-10 dataset
- **Goal** - To understand overfitting & generalization







torch.nn.module
(network model)

torch.nn.module
(predict output)

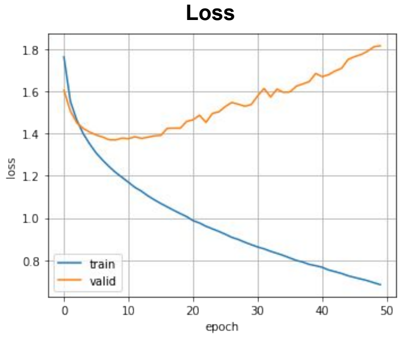
torch.nn.functional.cross_entropy
(loss function)

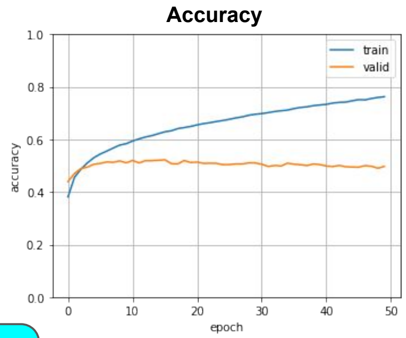
torch.backward
(back-propagation)

torch.optim.SGD
(stochastic gradient descent optimization)

Challenge

- **Goal** - To understand backpropagation and gradient descent
- Implement forward and backward pass for MLP and optimize using mini-batch gradient descent
  (without using torch modules!)





Data

Optimization loop

MLP → Forward pass → Compute loss → Compute gradients → Update weights

~~torch.nn.module~~
(network model)

~~torch.nn.module~~
(predict output)

~~torch.nn.functional.cross_entropy~~
(loss function)

~~torch.backward~~
(back-propagation)

~~torch.optim.SGD~~
(mini-batch gradient descent optimization)