IN5550: Neural Methods in Natural Language Processing
Sub-lecture 3.4
Computation graphs

Andrey Kutuzov, David Samuel

University of Oslo



Contents



1. Computation graphs
2. Neural network toolkits
3. Next week trailer

Computation graphs



Neural networks are just differentiable parameterized functions:

Computation graphs



Neural networks are just differentiable parameterized functions:

- so we can use the already familiar gradient-based optimization (SGD, etc)

Computation graphs



Neural networks are just differentiable parameterized functions:

- so we can use the already familiar gradient-based optimization (SGD, etc)
- But we have to calculate the gradient
  - potentially error prone and needs to be repeated for every change in model configuration/number of hidden layers/activation function/loss function, etc

Computation graphs



Neural networks are just differentiable parameterized functions:

- so we can use the already familiar gradient-based optimization (SGD, etc)
- But we have to calculate the gradient
  - potentially error prone and needs to be repeated for every change in model configuration/number of hidden layers/activation function/loss function, etc
- The good news is that we can use reverse-mode automatic differentiation (i.e. back-propagation) to automatically calculate the gradients

Computation graphs



Neural networks are just differentiable parameterized functions:

- so we can use the already familiar gradient-based optimization (SGD, etc)
- But we have to calculate the gradient
  - potentially error prone and needs to be repeated for every change in model configuration/number of hidden layers/activation function/loss function, etc
- The good news is that we can use reverse-mode automatic differentiation (i.e. back-propagation) to automatically calculate the gradients
- This means that instead of spending lots of your time deriving gradients by hand, you can use all that time to explore model options.

Computation graphs



- These days, deep neural networks are usually trained using the computation graph abstraction:

Computation graphs



- These days, deep neural networks are usually trained using the computation graph abstraction:
- representation of the process of computing a mathematical expression;

Computation graphs



- These days, deep neural networks are usually trained using the computation graph abstraction:
- representation of the process of computing a mathematical expression;
- allows to construct arbitrary deep and complex network architectures;

Computation graphs



- These days, deep neural networks are usually trained using the computation graph abstraction:
- representation of the process of computing a mathematical expression;
- allows to construct arbitrary deep and complex network architectures;
- computation is broken down into separate operations;

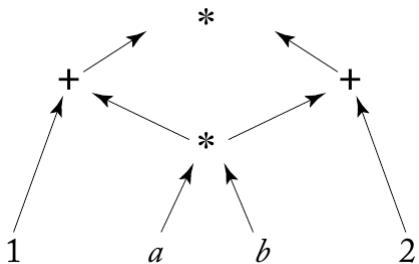
Computation graphs



- These days, deep neural networks are usually trained using the computation graph abstraction:
- representation of the process of computing a mathematical expression;
- allows to construct arbitrary deep and complex network architectures;
- computation is broken down into separate operations;
- each operation and variable is a node in a graph:

Computation graphs

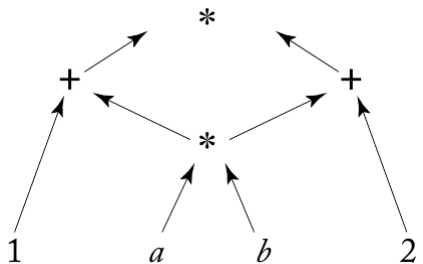
- These days, deep neural networks are usually trained using the computation graph abstraction:
- representation of the process of computing a mathematical expression;
- allows to construct arbitrary deep and complex network architectures;
- computation is broken down into separate operations;
- each operation and variable is a node in a graph:



directed acyclic graph (DAG)

Computation graphs

- These days, deep neural networks are usually trained using the computation graph abstraction:
- representation of the process of computing a mathematical expression;
- allows to construct arbitrary deep and complex network architectures;
- computation is broken down into separate operations;
- each operation and variable is a node in a graph:



directed acyclic graph (DAG)

Question: **what does this graph produce when** $a = 2, b = 1$**?**

Computation graphs



Training basics

- Similar to linear classifiers, deep NNs are trained by gradient calculation (SGD, etc)



Training basics

- Similar to linear classifiers, deep NNs are trained by gradient calculation (SGD, etc)
- Two main stages:

Computation graphs



Training basics

- Similar to linear classifiers, deep NNs are trained by gradient calculation (SGD, etc)
- Two main stages:
  1. Forward pass (compute predictions for given inputs);
     - move forward along the graph from input to output and compute the loss.

Computation graphs

Training basics

- Similar to linear classifiers, deep NNs are trained by gradient calculation (SGD, etc)
- Two main stages:
  1. Forward pass (compute predictions for given inputs);
     - move forward along the graph from input to output and compute the loss.
  2. Backward pass (compute gradients with respect to the loss)
     - move backwards along the graph from the resulting loss through all the layers to the input.

Computation graphs

Training basics

- Similar to linear classifiers, deep NNs are trained by gradient calculation (SGD, etc)
- Two main stages:
  1. Forward pass (compute predictions for given inputs);
     - move forward along the graph from input to output and compute the loss.
  2. Backward pass (compute gradients with respect to the loss)
     - move backwards along the graph from the resulting loss through all the layers to the input.
- How exactly they are performed is dependent on which nodes are there in the graph.

Computation graphs

Training basics

- Similar to linear classifiers, deep NNs are trained by gradient calculation (SGD, etc)
- Two main stages:
  1. Forward pass (compute predictions for given inputs);
     - move forward along the graph from input to output and compute the loss.
  2. Backward pass (compute gradients with respect to the loss)
     - move backwards along the graph from the resulting loss through all the layers to the input.
- How exactly they are performed is dependent on which nodes are there in the graph.
- Operations, parameters and variables are nodes in a DAG.

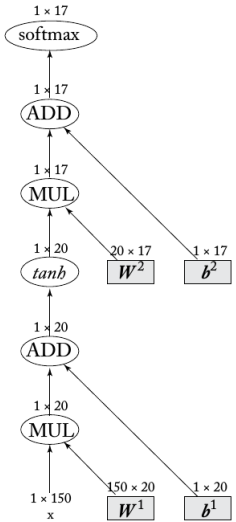
Computation graphs



Training basics

- Similar to linear classifiers, deep NNs are trained by gradient calculation (SGD, etc)
- Two main stages:
  1. Forward pass (compute predictions for given inputs);
     - move forward along the graph from input to output and compute the loss.
  2. Backward pass (compute gradients with respect to the loss)
     - move backwards along the graph from the resulting loss through all the layers to the input.
- How exactly they are performed is dependent on which nodes are there in the graph.
- Operations, parameters and variables are nodes in a DAG.
- The whole graph should be differentiable.

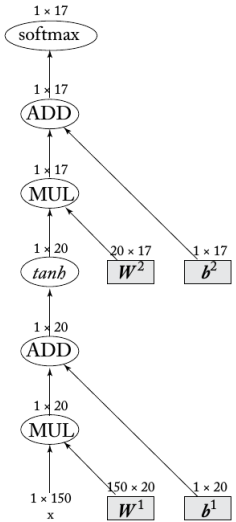
Computation graphs



Think over before the Q&A session / lab session:

- How many total layers are there in this computation graph?

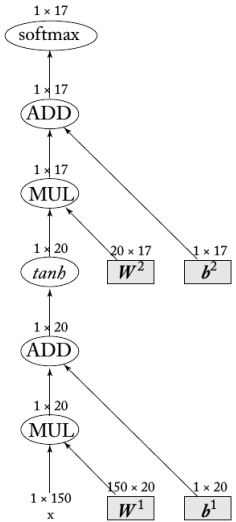
Computation graphs



Think over before the Q&A session / lab session:

- How many total layers are there in this computation graph?
- How many hidden layers?

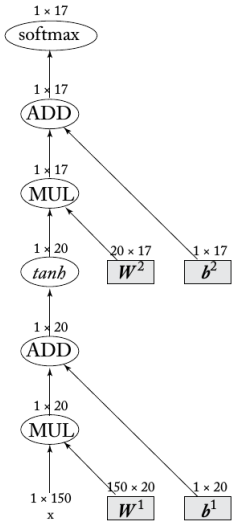
Computation graphs



Think over before the Q&A session / lab session:

- How many total layers are there in this computation graph?
- How many hidden layers?
- What is the shape of its output?

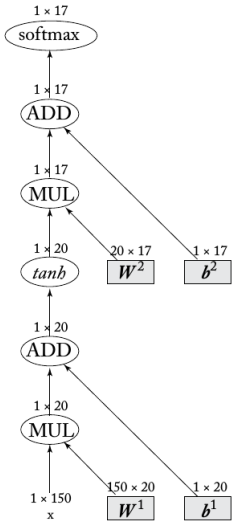
Computation graphs



Think over before the Q&A session / lab session:

- How many total layers are there in this computation graph?
- How many hidden layers?
- What is the shape of its output?
- **How can you describe what this architecture does?**

Computation graphs



Think over before the Q&A session / lab session:

- How many total layers are there in this computation graph?
- How many hidden layers?
- What is the shape of its output?
- **How can you describe what this architecture does?**
- Any NLP task for which it can be used?

Computation graphs



Forward pass

- Go over all nodes starting from the input;

Computation graphs



Forward pass

- Go over all nodes starting from the input;
- apply the operations sequentially;

Computation graphs



Forward pass

- Go over all nodes starting from the input;
- apply the operations sequentially;
- produce the output up to the final scalar loss-node.

Computation graphs



Forward pass

- Go over all nodes starting from the input;
- apply the operations sequentially;
- produce the output up to the final scalar loss-node.

Loss calculation

- Loss functions are as a rule the same as with linear models;
- cross-entropy is arguably the dominant one: $L(\hat{y}, y) = -y\ log(\hat{y}) - (1 - y) log(1 - \hat{y})$
- NB: the output should be squashed by sigmoid or like.

Computation graphs

Forward pass

- Go over all nodes starting from the input;
- apply the operations sequentially;
- produce the output up to the final scalar loss-node.

Loss calculation

- Loss functions are as a rule the same as with linear models;
- cross-entropy is arguably the dominant one: $L(\hat{y}, y) = -y\ log(\hat{y}) - (1 - y) log(1 - \hat{y})$
- NB: the output should be squashed by sigmoid or like.
- ...however, new loss functions are also being introduced:
  - e.g., Von Mises-Fisher loss [Kumar and Tsvetkov, 2018], etc.



Backward pass

- In simple linear classifiers, we compute the partial derivatives of model parameters with respect to the loss...

Computation graphs



Backward pass

- In simple linear classifiers, we compute the partial derivatives of model parameters with respect to the loss...
- ...in multi-layer networks it is the same...



Backward pass

- In simple linear classifiers, we compute the partial derivatives of model parameters with respect to the loss...
- ...in multi-layer networks it is the same...
- ...but the number of parameters can be crazy big.



Backward pass

- In simple linear classifiers, we compute the partial derivatives of model parameters with respect to the loss...
- ...in multi-layer networks it is the same...
- ...but the number of parameters can be crazy big.
- Gradient is computed using a special algorithm:
- **back-propagation a.k.a reverse differentiation** [Linnainmaa, 1970, Rumelhart et al., 1986];



Backward pass

- In simple linear classifiers, we compute the partial derivatives of model parameters with respect to the loss...
- ...in multi-layer networks it is the same...
- ...but the number of parameters can be crazy big.
- Gradient is computed using a special algorithm:
- **back-propagation a.k.a reverse differentiation** [Linnainmaa, 1970, Rumelhart et al., 1986];
- propagating loss values backwards through all layers...
- ...religiously computing derivatives at each.

Computation graphs



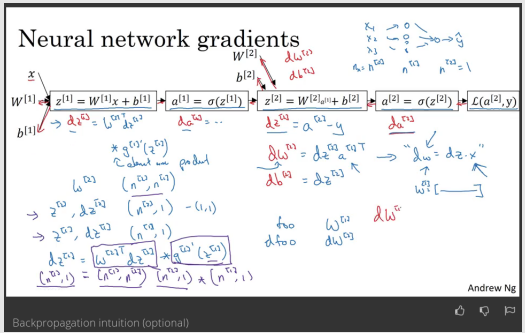
Backward pass

- *Back-propagation* is very efficient in terms of computation speed, but very error-prone if done manually...

Computation graphs

Backward pass

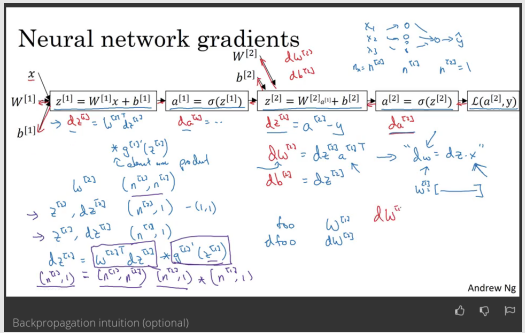
- *Back-propagation* is very efficient in terms of computation speed, but very error-prone if done manually...



Computation graphs

Backward pass

- *Back-propagation* is very efficient in terms of computation speed, but very error-prone if done manually...



- ... fortunately, automatic tools for gradient computation with *backprop* do exist and are well-developed!

Contents



1 Computation graphs

2 Neural network toolkits

3 Next week trailer

Neural network toolkits



Software libraries

No lack of open-source deep learning toolkits:

Neural network toolkits



Software libraries

No lack of open-source deep learning toolkits:

- TensorFlow by Google [Abadi et al., 2015] (`https://tensorflow.org`)
- PyTorch by Meta [Paszke et al., 2017] (`https://pytorch.org/`)

Neural network toolkits



Software libraries

No lack of open-source deep learning toolkits:

- TensorFlow by Google [Abadi et al., 2015] (`https://tensorflow.org`)
- PyTorch by Meta [Paszke et al., 2017] (`https://pytorch.org/`)
- Jax by Google [Bradbury et al., 2018] (`https://jax.readthedocs.io//`)

Neural network toolkits



Software libraries

No lack of open-source deep learning toolkits:

- TensorFlow by Google [Abadi et al., 2015] (`https://tensorflow.org`)
- PyTorch by Meta [Paszke et al., 2017] (`https://pytorch.org/`)
- Jax by Google [Bradbury et al., 2018] (`https://jax.readthedocs.io//`)

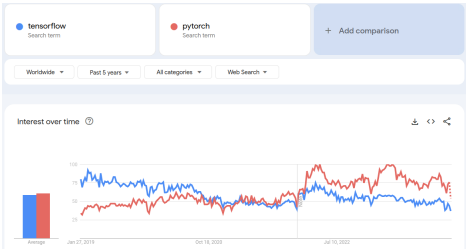
Neural network toolkits

Software libraries

No lack of open-source deep learning toolkits:

- TensorFlow by Google [Abadi et al., 2015] (`https://tensorflow.org`)
- PyTorch by Meta [Paszke et al., 2017] (`https://pytorch.org/`)
- Jax by Google [Bradbury et al., 2018] (`https://jax.readthedocs.io//`)

Every tech giant wants to introduce its own 'definitive deep learning library'.



`https://trends.google.com`

Neural network toolkits



This course

- IN5550 is currently based on PyTorch library...

Neural network toolkits



This course

- IN5550 is currently based on PyTorch library...
- ...since it is more flexible and arguably easier to learn.

This course

- IN5550 is currently based on PyTorch library...
- ...since it is more flexible and arguably easier to learn.



- We are going to use well-established architectures...

Neural network toolkits

This course

- IN5550 is currently based on PyTorch library...
- ...since it is more flexible and arguably easier to learn.



- We are going to use well-established architectures...
- ...so everything can be done via PyTorch.

This course

- IN5550 is currently based on PyTorch library...
- ...since it is more flexible and arguably easier to learn.



- We are going to use well-established architectures...
- ...so everything can be done via PyTorch.
- Makes it easy to use deep learning elements in your code.
- `https://pytorch.org/`

Contents



1. Computation graphs
2. Neural network toolkits
3. Next week trailer

Next week trailer



Dense representations of linguistic features

- Making the discrete world continuous.

Next week trailer



Dense representations of linguistic features

- Making the discrete world continuous.
- Vector semantics.
- Word embeddings.

  Word2vec!

Next week trailer



Dense representations of linguistic features

- Making the discrete world continuous.
- Vector semantics.
- Word embeddings.
  Word2vec!
- Stay tuned!

References I


Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015).
TensorFlow: Large-scale machine learning on heterogeneous systems.
Software available from tensorflow.org.


Bradbury, J., Frostig, R., Hawkins, P., Johnson, M. J., Leary, C., Maclaurin, D., Necula, G., Paszke, A., VanderPlas, J., Wanderman-Milne, S., and Zhang, Q. (2018).
JAX: composable transformations of Python+NumPy programs.


Kumar, S. and Tsvetkov, Y. (2018).
Von Mises-Fisher loss for training sequence to sequence models with continuous outputs.
In International Conference on Learning Representations.

References II

Linnainmaa, S. (1970).
The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors.
PhD thesis, Master's Thesis (in Finnish), Univ. Helsinki.

Paszke, A., Gross, S., Chintala, S., and Chanan, G. (2017).
Pytorch: Tensors and dynamic neural networks in Python with strong GPU acceleration.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986).
Learning representations by back-propagating errors.
Nature, 323(6088):533.